CF GRADO SUPERIOR

CICLOS FORMATIVOS

R.D. 1538/2006

Administración de Sistemas Gestores de Bases de Datos

Ra-Ma®

LUIS HUESO IBÁÑEZ

www.ra-ma.es/cf

Administración de Sistemas Gestores de Bases de Datos

LUIS HUESO IBÁÑEZ

Ra-Ma®

Índice

INTRODUCCIÓN 7

CAPÍTULO 1. REVISIÓN DE CONCEPTOS DE BASES DE DATOS 9

1.1 INTRODUCCIÓN. DEFINICIÓN DE BASES DE DATOS Y SGBD 10
1.2 ARQUITECTURA DE SISTEMASDE BASES DE DATOS 10
1.3 FUNCIONES DEL SISTEMA GESTOR DE BASE DE DATOS (SGBD) 12
1.4 COMPONENTES 14
1.5 USUARIOS DE LOS SGBD 16
1.6 TIPOS DE SGBD 16
1.7 SISTEMAS GESTORES DE BASE DE DATOS COMERCIALES Y LIBRES 18

RESUMEN DEL CAPÍTULO 19
EJERCICIOS PROPUESTOS 19
TEST DE CONOCIMIENTOS 20

CAPÍTULO 2. INSTALACIÓN Y CONFIGURACIÓN DE UN SGBD 21

2.1 REVISIÓN MYSQL 22
2.2 ARQUITECTURA MYSQL 22
2.3 INSTALACIÓN DE MYSQL 23
2.3.1 Cuestiones generales sobre la instalación 23
2.3.2 Instalación de MySQL en Windows 23
2.4 CONFIGURACIÓN SERVIDOR 26
2.4.1 Configuración con el asistente 26
2.4.2 Variables y opciones en MySQL 35
2.4.3 Variables de sistema del servidor 37
2.4.4 Variables de estado del servidor 41
2.4.5 Comandos para gestión de variables 43
2.5 ESTRUCTURA DEL DICCIONARIODE DATOS 47
2.5.1 Las tablas de *INFORMATION_SCHEMA* 48
2.6 FICHEROS LOG 53
2.6.1 El registro de errores (*Error Log*) 53
2.6.2 El registro general de consultas 54
2.6.3 El registro binario (*Binary Log*) 54
2.6.4 El registro de consultas lentas (*Slow Query Log*) 56
2.6.5 Mantenimiento de ficheros de registro (*log*) 57
2.6.6 Registro en *InnoDB* 57
2.7 CASO BASE 58

RESUMEN DEL CAPÍTULO 59
EJERCICIOS PROPUESTOS 59
TEST DE CONOCIMIENTOS 60

CAPÍTULO 3. GESTIÓN DE CUENTAS DE USUARIO Y PERMISOS 61
3.1 EL SISTEMA DE PERMISOS MYSQL 62
3.1.1 Tablas de permisos 62
3.1.2 Privilegios en MySQL 65
3.1.3 Control de acceso detallado 67
3.1.4 Cuándo tienen efecto los cambios de privilegios 70
3.2 GESTIÓN DE RECURSOS 71
3.2.1 Limitar recursos de cuentas 74
3.2.2 Asignar contraseñas a cuentas 76
3.3 CONEXIONES SEGURAS 78
3.3.1 Conceptos básicos de SSL 78
3.3.2 Requisitos y variables SSL 79
3.3.3 Opciones SSL de GRANT 80
3.3.4 Conexiones seguras a MySQL 82
3.4 CASO BASE 86
RESUMEN DEL CAPÍTULO 87
EJERCICIOS PROPUESTOS 87
TEST DE CONOCIMIENTOS 88
CAPÍTULO 4. AUTOMATIZACIÓN DE TAREAS: CONSTRUCCIÓN DE GUIONES DE ADMINISTRACIÓN 89
4.1 HERRAMIENTAS PARA AUTOMATIZAR TAREAS 90
4.2 PROCEDIMIENTOS Y FUNCIONESALMACENADOS 90
4.2.1 Sintaxis y ejemplos de rutinas almacenadas 91
4.2.2 Parametros y variables 96
4.2.3 Instrucciones condicionales 99
4.2.4 Instrucciones repetitivas o *loops* 101
4.2.5 SQL en rutinas: Cursores 103
4.2.6 Gestión rutinas almacenadas 110
4.2.7 Manejo de errores 110
4.3 TRIGGERS 114
4.3.1 Gestión de disparadores 114
4.3.2 Usos de disparadores 116
4.3.3 Administración de disparadores 119
4.4 VISTAS 119
4.4.1 Gestión de vistas 120
4.5 EVENTOS 122
4.5.1 Gestión Eventos 123
4.6 CASO BASE 126
RESUMEN DEL CAPÍTULO 127
EJERCICIOS PROPUESTOS 127
TEST DE CONOCIMIENTOS 128
CAPÍTULO 5. OPTIMIZACIÓN Y MONITORIZACIÓN 129
5.1 ÍNDICES 130
5.1.1 Tipos de Índices 131

5.1.2 Estructura de un índice 132
5.1.3 Índices en MySQL 132
5.1.4 Gestión de índices 134
5.2 OPTIMIZACIÓN EN MYSQL 137
5.2.1 Optimización del diseño de bases de datos 137
5.2.2 Procesamiento de consultas 138
5.2.3 Optimización de consultas e índices 139
5.2.4 Otros aspectos de optimizacion 156
5.3 OPTIMIZACIÓN DEL SERVIDOR 159
5.3.1 Almacenamiento 159
5.3.2 Optimización de motores de almacenamiento 159
5.3.3 Memoria 161
5.3.4 Rendimiento 168
5.4 HERRAMIENTAS DE MONITORIZACIÓN DE MYSQL 169
5.4.1 Comandos show 169
5.4.3 Otras herramientas 178
5.5 CASO BASE 179

RESUMEN DEL CAPÍTULO 180
EJERCICIOS PROPUESTOS 181
TEST DE CONOCIMIENTOS 182

CAPÍTULO 6. BASES DE DATOS DISTRIBUIDAS Y ALTA DISPONIBILIDAD 183
6.1 CONCEPTOS DE BASES DE DATOS DISTRIBUIDAS 184
6.1.1 Introducción 184
6.1.2 Arquitectura de un DDBMS 185
6.1.3 Técnicas de fragmentación, replicación y distribución 186
6.1.4 Tipos de sistemas de bases distribuidas 187
6.2 REPLICACIÓN EN MYSQL 188
6.2.1 Introducción 188
6.2.2 Arquitectura y Configuración 189
6.2.3 Implementación 190
6.2.4 Administración y Mantenimiento 193
6.3 BALANCEO DE CARGA Y ALTA DISPONIBILIDAD 195
6.3.1 Fundamentos 196
6.3.2 Mysql cluster 197
6.3.3 Organización de los datos 198
6.3.4 Instalación y Configuración del Cluster 199
6.3.5 Gestión de MySQL cluster 206
6.3.6 Programas del cluster 208
6.4 CASO BASE 210

RESUMEN DEL CAPÍTULO 210
EJERCICIOS PROPUESTOS 211
TEST DE CONOCIMIENTOS 211

APÉNDICE A. CONECTORES Y APIS DE MYSQL 213
APÉNDICE B. COPIAS DE SEGURIDAD Y RECUPERACIÓN DE BASES DE DATOS 219
APÉNDICE C. INSTALACIÓN DE UNA MÁQUINA VIRTUAL DE UBUNTU EN VIRTUALBOX 227
APÉNDICE D. REVISIÓN DE HERRAMIENTAS DE MYSQL EN ENTORNOS LINUX 235
APÉNDICE E. INTRODUCCIÓN A LA ADMINISTRACIÓN DE SISTEMAS GESTORES: ORACLE 239
APÉNDICE F. INTRODUCCIÓN A LA ADMINISTRACIÓN DE SISTEMAS GESTORES: SQL SERVER 271
MATERIAL ADICIONAL 305
ÍNDICE ALFABÉTICO 307

Introducción

La información es un recurso cada vez más crítico e importante para las empresas y organizaciones. Por ello es crucial disponer de herramientas que permitan y faciliten su administración.

El administrador de bases de datos debe asegurar y controlar la información de manera que siempre esté accesible al usuario.

Para ello los administradores tienen como tareas principales el diseño, instalación y configuración de sistemas gestores de bases de datos, la realización copias de seguridad, la gestión usuarios y permisos sobre los datos y la monitorización y optimización del rendimiento de los SGBD

De todo ello nos ocupamos en este libro que está concebido para introducir al lector en todos los aspectos importantes de la administración.

Por motivos de sencillez la mayoría de ejercicios y ejemplos del libro se basan en el uso de MySQL sobre máquinas *Windows*, sin embargo hay numerosas referencias e incluso ejercicios en entornos *Linux* de forma que el alumno pueda tener una visión más completa y heterogénea.

Administrar sistemas no es una tarea sencilla ni mucho menos sistemática, este y otros libros pueden ayudar pero sólo la experiencia y el trabajo diario con sistemas reales en producción permitirán la formación completa de un administrador.

Además, es imposible detallar en un solo libro todos los contenidos y todas las posibilidades que pueden darse en el trabajo administrativo. Es por ello crucial para un administrador saber hacer uso de las herramientas de ayuda que proporcionan tanto las herramientas de su gestor de bases de datos cómo de la documentación oficial, en este caso de MySQL, disponible en la web.

A partir del segundo capítulo se ha incluirlo un caso base hipotético para trabajar sobre él lo explicado en cada capítulo. Éste se basa en un caso o ejemplo concreto. En nuestro caso hemos elegido una aplicación de plena actualidad como son las dedicadas a ofrecer música y vídeo bajo demanda.

Conviene señalar que la mayoría de ejemplos del libro se realizan usando comandos en lugar de interfaces gráficas ya que éstos proporcionan mayor flexibilidad y potencia al administrador. Además permiten conocer mejor las interioridades de nuestros sistemas.

Este libro surge con el propósito de acercar al lector a los aspectos más importantes que encierran los sistemas gestores de bases de datos, ante la creciente demanda de personal cualificado para su administración. Con tal propósito, puede servir de apoyo también para estudiantes del Ciclo Formativo Administración de Sistemas Informáticos en Red de Grado Superior y de Ingenierías Técnicas.

Para todo aquél que use este libro en el entorno de la enseñanza (Ciclos Formativos o Universidad), se ofrecen varias posibilidades: utilizar los conocimientos aquí expuestos para inculcar aspectos genéricos de los sistemas gestores de bases de datos o simplemente centrarse en preparar a fondo alguno de ellos. La extensión de los contenidos aquí incluidos hace imposible su desarrollo completo en la mayoría de los casos.

Ra-Ma pone a disposición de los profesores una guía didáctica para el desarrollo del tema que incluye las soluciones a los ejercicios expuestos en el texto. Puede solicitarlo a *editorial@ra-ma.com*, acreditándose como docente y siempre que el libro sea utilizado como texto base para impartir las clases.

1 Revisión de conceptos de bases de datos

OBJETIVOS DEL CAPÍTULO

- ✓ Instalar el servidor MySQL.
- ✓ Conocer las opciones de configuración de MySQL.
- ✓ Optimizar el funcionamiento de MySQL.
- ✓ Monitorizar MySQL.
- ✓ Aprender a gestionar ficheros de registro.
- ✓ Conocer la estructura del diccionario de datos de MySQL.

1.1 INTRODUCCIÓN. DEFINICIÓN DE BASES DE DATOS Y SGBD

Conviene, antes de comenzar con el tema principal del libro, recordar los conceptos más relevantes relacionados con sistemas gestores de bases de datos, así como las herramientas relacionadas.

En primer lugar, y aunque probablemente ya se ha visto en otros módulos, es importante diferenciar entre el concepto de base de datos y el de sistema gestor, ya que es habitual confundirlos y sin embargo son cosas muy distintas.

Definición 1:

Una base de datos es un conjunto de datos relacionados y organizados con cierta estructura. Según dicha organización distinguimos entre diferentes modelos de bases de datos como el relacional, jerárquico o en red.

El modelo de bases de datos más extendido es el **relacional** y es el que trabajaremos en este libro.

Para su manipulación y gestión surgieron los sistemas gestores de bases de datos (**SGBD** en lo sucesivo).

Definición 2:

El sistema de gestión de la base de datos (SGBD) es una aplicación que permite a los usuarios definir, crear y mantener bases de datos, proporcionando acceso controlado a las mismas. Es una herramienta que sirve de interfaz entre el usuario y las bases de datos.

Es decir, por un lado tenemos los datos organizados según ciertos criterios y, por otro, un software que nos permite o facilita su gestión con distintas herramientas y funcionalidades que describimos a continuación.

1.2 ARQUITECTURA DE SISTEMAS DE BASES DE DATOS

Hay tres características importantes inherentes a los sistemas de bases de datos: la separación entre los programas de aplicación y los datos, el manejo de múltiples vistas por parte de los usuarios y el uso de un catálogo para almacenar el esquema de la base de datos. En 1975, el comité ANSI-SPARC (*American National Standard Institute - Standards Planning and Requirements Committee*) propuso una arquitectura de tres niveles para los sistemas de bases de datos, que resulta muy útil a la hora de conseguir estas tres características.

El objetivo de la arquitectura de tres niveles es el de separar los programas de aplicación de la base de datos física. En esta arquitectura, el esquema de una base de datos se define en tres niveles de abstracción distintos como se aprecia en la siguiente imagen:

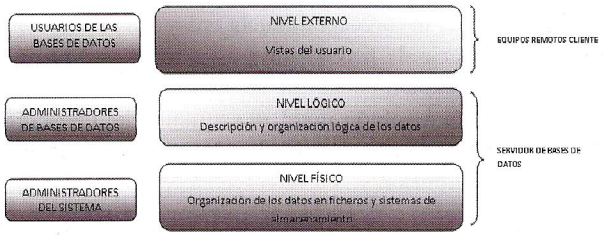


Figura 1.1. *Arquitectura de Sistemas de Bases de Datos*

En el nivel interno se describe la estructura física de la base de datos mediante un esquema interno. Este esquema se especifica mediante un modelo físico y describe todos los detalles para el almacenamiento de la base de datos, así como los métodos de acceso.

En el nivel conceptual se describe la estructura de toda la base de datos para una comunidad de usuarios (todos los de una empresa u organización), mediante un esquema conceptual. Este esquema oculta los detalles de las estructuras de almacenamiento y se concentra en describir entidades, atributos, relaciones, operaciones de los usuarios y restricciones. En este nivel se puede utilizar un modelo conceptual o un modelo lógico para especificar el esquema.

En el nivel externo se describen varios esquemas externos o vistas de usuario. Cada esquema externo describe la parte de la base de datos que interesa a un grupo de usuarios determinado y oculta a ese grupo el resto de la base de datos. En este nivel se puede utilizar un modelo conceptual o un modelo lógico para especificar los esquemas.

Hay que destacar que los tres esquemas no son más que descripciones de los mismos datos pero con distintos niveles de abstracción. Los únicos datos que existen realmente están a nivel físico, almacenados en un dispositivo, como puede ser un disco. En un SGBD basado en la arquitectura de tres niveles, cada grupo de usuarios hace referencia exclusivamente a su propio esquema externo. Por lo tanto, el SGBD debe transformar cualquier petición expresada en términos de un esquema externo a una petición expresada en términos del esquema conceptual, y luego, a una petición en el esquema interno, que se procesará sobre la base de datos almacenada. Si la petición es de una obtención (consulta) de datos, será preciso modificar el formato de la información extraída de la base de datos almacenada para que coincida con la vista externa del usuario. En definitiva, tenemos la vista del usuario, la del sistema gestor y la vista física o de almacenamiento.

La arquitectura de tres niveles es útil para explicar el concepto de **independencia de datos,** que podemos definir como *la capacidad para modificar el esquema en un nivel del sistema sin tener que modificar el esquema del nivel inmediato superior*. Se pueden definir dos tipos de independencia de datos:

La **independencia lógica** es la capacidad de modificar el esquema conceptual sin tener que alterar los esquemas externos ni los programas de aplicación. Se puede modificar el esquema conceptual para ampliar la base de datos o para reducirla. Si, por ejemplo, se reduce la base de datos eliminando una entidad, los esquemas externos que no se refieran a ella no deberán verse afectados.

La **independencia física** es la capacidad de modificar el esquema interno sin tener que alterar el esquema conceptual (o los externos). Por ejemplo, puede ser necesario reorganizar ciertos ficheros físicos con el fin de mejorar el rendimiento de las operaciones de consulta o de actualización de datos. Dado que la independencia física se refiere solo a la separación entre las aplicaciones y las estructuras físicas de almacenamiento, es más fácil de conseguir que la independencia lógica.

En los SGBD que tienen la arquitectura de varios niveles es necesario ampliar el catálogo o diccionario, de modo que incluya información sobre cómo establecer la correspondencia entre las peticiones de los usuarios y los datos entre los diversos niveles. El SGBD utiliza una serie de procedimientos adicionales para realizar estas correspondencias haciendo referencia a la información de correspondencia que se encuentra en el catálogo. La independencia de datos se consigue porque al modificarse el esquema en algún nivel, el esquema del nivel inmediato superior permanece sin cambios, solo se modifica la correspondencia entre los dos niveles. No es preciso modificar los programas de aplicación que hacen referencia al esquema del nivel superior.

Por lo tanto, la arquitectura de tres niveles puede facilitar la obtención de la verdadera independencia de datos, tanto física como lógica. Sin embargo, los dos niveles de correspondencia implican un gasto extra durante la ejecución de una consulta o de un programa, lo cual reduce la eficiencia del SGBD. Es por esto que muy pocos SGBD han implementado esta arquitectura completa.

1.3 FUNCIONES DEL SISTEMA GESTOR DE BASE DE DATOS (SGBD)

La función principal de un **SGBD** es permitir a los usuarios realizar las cuatro operaciones fundamentales posibles, tanto sobre las estructuras de datos como sobre los datos que albergan, es decir operaciones de inserción o creación, consulta, actualización y borrado, de una manera eficiente y coherente.

Para tal fin, la mayoría de SGBD incorporan las siguientes características y funciones:

Un catálogo

Donde se almacenen las descripciones de los datos y sea accesible por los usuarios. Este **catálogo** es lo que se denomina *diccionario de datos* y contiene información que describe los datos de la base de datos (metadatos). Normalmente, un diccionario de datos describe entre otras cosas:

- Nombre, tipo y tamaño de los datos.
- Relaciones entre los datos.
- Restricciones de integridad sobre los datos.
- Usuarios autorizados a acceder a los objetos de base de datos.
- Estadísticas de utilización, tales como la frecuencia de las transacciones y el número de accesos realizados a los objetos de la base de datos.

Garantizar la integridad

Disponer de un mecanismo que garantice que todas las actualizaciones correspondientes a una determinada transacción se realicen, o que no se realice ninguna. Una transacción es un conjunto de acciones que cambian el contenido de la base de datos. Una **transacción** en el sistema informático de la empresa inmobiliaria sería dar de alta a un empleado o eliminar un inmueble. Una transacción un poco más complicada sería eliminar un empleado y reasignar sus inmuebles a otro empleado. En este caso hay que realizar varios cambios sobre la base de datos. Si la transacción falla durante su realización, por ejemplo porque falla el hardware, la base de datos quedará en un estado inconsistente. Algunos de los cambios se habrán hecho y otros no, por lo tanto, los cambios realizados deberán ser deshechos para devolver la base de datos a un estado consistente.

Permitir actualizaciones

Asegurar que la base de datos se actualice correctamente cuando varios usuarios la están actualizando concurrentemente. Uno de los principales objetivos de los SGBD es el permitir que varios usuarios tengan acceso concurrente a los datos que comparten. El acceso concurrente es relativamente fácil de gestionar si todos los usuarios se dedican a leer datos, ya que no pueden interferir unos con otros. Sin embargo, cuando dos o más usuarios están accediendo a la base de datos y al menos uno de ellos está actualizando datos, pueden interferir de modo que se produzcan inconsistencias en la base de datos. El SGBD se debe encargar de que estas interferencias no se produzcan en el acceso simultáneo.

Recuperación de datos

Permitir recuperar las bases de datos en caso de que ocurra algún suceso que la dañe. Como se ha comentado antes, cuando el sistema falla en medio de una transacción, la base de datos se debe devolver a un estado consistente. Esta falta puede ser a causa de un fallo en algún dispositivo hardware o un error del software, que hagan que el SGBD aborte, o puede ser a causa de que el usuario detecte un error durante la transacción y la aborte antes de que finalice. En todos estos casos, el SGBD debe proporcionar un mecanismo capaz de recuperar la base de datos llevándola a un estado consistente.

FAT Server - El peso en el servidor
FAT client ↘ el peso en los clientes.

Integración

ODBC, JDBC

Ser capaz de integrarse con algún software de comunicación. Muchos usuarios acceden a la base de datos desde terminales. En ocasiones estos terminales se encuentran conectados directamente a la máquina sobre la que funciona el SGBD. En otras ocasiones los terminales están en lugares remotos, por lo que la comunicación con la máquina que alberga al SGBD se debe hacer a través de una red. En cualquiera de los dos casos, el SGBD recibe peticiones en forma de mensajes y responde de modo similar. Todas estas transmisiones de mensajes las maneja el gestor de comunicaciones de datos. Aunque este gestor no forma parte del SGBD, es necesario que el SGBD se pueda integrar con él para que el sistema sea comercialmente viable.

Cumplir restricciones

Proporcionar los medios necesarios para garantizar que tanto los datos de la base de datos, como los cambios que se realizan sobre estos datos, sigan ciertas reglas. La integridad de la base de datos requiere la validez y consistencia de los datos almacenados. Se puede considerar como otro modo de proteger la base de datos, pero además de tener que ver con la seguridad, tiene otras implicaciones. La integridad se ocupa de la calidad de los datos. Normalmente se expresa mediante restricciones, que son una serie de reglas que la base de datos no puede violar. Por ejemplo, se puede establecer la restricción de que cada empleado no puede tener asignados más de diez inmuebles. En este caso sería deseable que el SGBD controlara que no se sobrepase este límite cada vez que se asigne un inmueble a un empleado.

Herramientas de administración

Proporcionar herramientas que permitan administrar la base de datos de modo efectivo, lo que implica un diseño óptimo de las mismas, garantizar la disponibilidad e integridad de los datos, controlar el acceso al servidor y a los datos, monitorizar el funcionamiento del servidor y optimizar su funcionamiento. Muchas de ellas van integradas en el sistema gestor, otras son creadas por terceros o por el propio administrador según sus requerimientos.

ACTIVIDADES 1.1

- Averigua y explica el significado del término ACID en el contexto de los sistemas gestores de bases de datos.
- Busca al menos tres diferencias importantes entre Access y MySQL.
- ¿Qué diferencias principales piensas que hay entre administrar bases de datos y administrar un SGBD?
- Averigua en qué consiste y para qué sirve la minería de datos. → Recogida de datos para su utilizados.

ACID → (Atomicity, consintency, Isolation, Durability)
- Atomic → transaccion indivisible, se hace toda o no se hace
- Consistencia → Que quede en un atado consistente
- Aislamiento → Transacciones simultaneas no pueden mezclase una detrás de otra.
- Permanencia, que los datos esten siempre ahí.

1.4 COMPONENTES

Son los elementos que deben proporcionar los servicios comentados en la sección anterior. No se puede generalizar ya que varían mucho según la tecnología. Sin embargo, es muy útil conocer sus componentes y cómo se relacionan cuando se trata de comprender lo que es un sistema de bases de datos.

El **SGBD** es la aplicación que interacciona con los usuarios de los programas de aplicación y la base de datos. En general, un SGBD suele incluir los siguientes componentes:

- **Lenguaje de definición de datos** (**DDL**: *Data Definition Language*)

  Sencillo lenguaje artificial para definir y describir los objetos de la base de datos, su estructura, relaciones y restricciones.

- **Lenguaje de control de datos** (**DCL**: *Data Control Language*)

  Encargado del control y seguridad de los datos (privilegios y modos de acceso, etc.). Este lenguaje permite especificar la estructura y el tipo de los datos, así como las restricciones sobre los datos. Todo esto se almacenará en la base de datos.

- **Lenguaje de manipulación de datos** (**DML**: *Data Manipulation Language*)

  Para la inserción, actualización, eliminación y consulta de datos. Para tal fin el lenguaje por excelencia es el conocido SQL (*Structured Query Language*). Incluye instrucciones para los tres tipos de lenguajes comentados y por su sencillez y potencia se ha convertido en el lenguaje estándar de los **SGBD relacionales**.

- **Diccionario de datos**

  Esquemas que describen el contenido del **SGBD** incluyendo los distintos objetos con sus propiedades.

- **Objetos**: **Tablas base y vistas (tablas derivadas)**
  - Consultas.
  - Dominios y tipos definidos de datos.
  - Restricciones de tabla y dominio y aserciones.
  - Funciones y procedimientos almacenados.
  - Disparadores o *triggers*.
  - Usuarios
  - Perfiles

- Hibridos
- Guardan como objetos
- Relacional, almacenar y recuperar información

- **Distintas herramientas para**:
  - Seguridad: de modo que los usuarios no autorizados no puedan acceder a la base de datos.
  - Integridad: que mantiene la integridad y la consistencia de los datos.
  - El control de concurrencia: que permite el acceso compartido a la base de datos.
  - El control de recuperación: que restablece la base de datos después de que se produzca un fallo del hardware o del software.
  - Gestión del diccionario de datos (o catálogo): accesible por el usuario que contiene la descripción de los datos de la base de datos.
  - Programación de aplicaciones.
  - Importación/exportación de datos (migraciones).
  - Distribución de datos.
  - Replicación (arquitectura maestro-esclavo).
  - Sincronización (de equipos replicados).
- **Optimizador de consultas**

  Para determinar la estrategia óptima para la ejecución de las consultas.
- **Gestion de transacciones**

  Este módulo realiza el procesamiento de las transacciones.
- **Planificador** (*scheduler*)

  Para programar y automatizar la realización de ciertas operaciones y procesos.
- **Copias de seguridad**

  Para garantizar que la base de datos se puede devolver a un estado consistente en caso de que se produzca algún fallo.

Todos los SGBD no presentan la misma funcionalidad, depende de cada producto. En general, los grandes SGBD multiusuario ofrecen todas las funciones que se acaban de citar y muchas más. Los sistemas modernos son conjuntos de programas extremadamente complejos y sofisticados, con millones de líneas de código y con una documentación consistente en varios volúmenes. Lo que se pretende es proporcionar un sistema que permita gestionar cualquier tipo de requisitos y que tenga un 100% de fiabilidad ante cualquier fallo hardware o software. Los SGBD están en continua evolución, tratando de satisfacer los requerimientos de todo tipo de usuarios. Por ejemplo, muchas aplicaciones de hoy en día necesitan almacenar imágenes, vídeo, sonido, etc. Para satisfacer a este mercado, los SGBD deben cambiar. Conforme vaya pasando el tiempo irán surgiendo nuevos requisitos que serán incorporados paulatinamente.

ACTIVIDADES 1.2

- Busca, resume y comente opiniones en distintos foros o páginas sobre los sistemas MySQL, SQL Server y Oracle.
- ¿Qué limitaciones tiene la versión Oracle Express Edition respecto a la extendida?
- ¿Qué lenguaje específico usa SQL Server para implementar el lenguaje SQL?

1.5 USUARIOS DE LOS SGBD

Generalmente distinguimos cuatro grupos de usuarios de sistemas gestores de bases de datos: los usuarios administradores, los diseñadores de la base de datos, los programadores y los usuarios de aplicaciones que interactúan con las bases de datos.

Administrador de la base de datos

DBA + operadores

Se encarga del diseño físico de la base de datos y de su implementación, realiza el control de la seguridad y de la concurrencia, mantiene el sistema para que siempre se encuentre operativo y se encarga de que los usuarios y las aplicaciones obtengan buenas prestaciones. El administrador debe conocer muy bien el SGBD que se esté utilizando, así como el equipo informático sobre el que esté funcionando.

Diseñadores de la base de datos

→ Administrador de datos, DA

Realizan el diseño lógico de la base de datos, debiendo identificar los datos, las relaciones entre datos y las restricciones sobre los datos y sus relaciones. El diseñador de la base de datos debe tener un profundo conocimiento de los datos de la empresa y también debe conocer sus reglas de negocio. Las reglas de negocio describen las características principales de los datos tal y como los ve la empresa. Para obtener un buen resultado, el diseñador de la base de datos debe implicar en el desarrollo del modelo de datos a todos los usuarios de la base de datos, tan pronto como sea posible. El diseño lógico de la base de datos es independiente del SGBD concreto que se vaya a utilizar, es independiente de los programas de aplicación, de los lenguajes de programación y de cualquier otra consideración física.

Programadores de aplicaciones

Se encargan de implementar los programas de aplicación que servirán a los usuarios finales. Estos programas de aplicación son los que permiten consultar datos, insertarlos, actualizarlos y eliminarlos. Estos programas se escriben mediante lenguajes de tercera generación o de cuarta generación.

Usuarios finales

Clientes de la base de datos que hacen uso de ella sin conocer en absoluto su funcionamiento y organización. Son personas con pocos o nulos conocimientos de informática.

1.6 TIPOS DE SGBD

Existen numerosos SGBD en el mercado que podemos clasificar según lo siguiente:

- **Modelo lógico en el que se basan**
  - Modelo Jerárquico.
  - Modelo de Red.
  - Modelo Relacional.
  - Modelo Orientado a Objetos.
- **Número de usuarios**
  - Monousuario.
  - Multiusuario.
- **Número de sitios**
  - Centralizados.
  - Distribuidos: homogéneos y heterogéneos.
- **Ámbito de aplicación**
  - Propósito General.
  - Propósito Específico.

Basados en el modelo **relacional**, los datos se describen como relaciones que se suelen representar como tablas bidimensionales consistentes en filas y columnas. Cada fila (*tupla*, en terminología relacional) representa una ocurrencia. Las columnas (atributos) representan propiedades de las filas. Cada *tupla* o fila se identifica por una clave primaria o identificador.

Esta organización de la información permite recuperar de forma flexible los datos de una o varias tablas, así como combinar registros de diferentes tablas para formar otras nuevas. No todas las definiciones posibles de tablas son válidas según el modelo relacional. En él, deben emplearse diseños normalizados que garantizan que no se producirán anomalías en la actualización de la base de datos.

Los SGBD relacionales se han impuesto hasta llegar a dominar casi totalmente el mercado actual. Ello se ha debido principalmente a su flexibilidad y sencillez de manejo. Igualmente conviene destacar la amplia implantación del lenguaje SQL, que se ha convertido en un estándar para el manejo de datos en el modelo relacional, lo que ha supuesto una ventaja adicional para su desarrollo.

ACTIVIDADES 1.3

- Averigua el significado de DNS y LDAP. ¿Con qué tipos de bases de datos está relacionado (relacional, jerárquica o en red)?

1.7 SISTEMAS GESTORES DE BASE DE DATOS COMERCIALES Y LIBRES

Con el advenimiento de Internet, el software libre se ha consolidado como alternativa, técnicamente viable y económicamente sostenible al software comercial, contrariamente a lo que a menudo se piensa, convirtiéndose el software libre como otra alternativa para ofrecer los mismos servicios a un coste cada vez más reducido.

Estas alternativas se encuentran tanto para herramientas de ofimática como Openoffice o Microsoft Office. También disponemos de herramientas mucho más avanzadas a un nivel de propósito general como MySQL, SQL Server y si hablamos de software con más potencia y funcionalidad vale la pena señalar a Postgresql u Oracle, entre otros.

No conviene decantarse por uno en concreto, debemos usar en primer lugar lo que mejor nos funciona dadas nuestras restricciones particulares. Así, si solamente quiero una aplicación de agenda puede ser perfectamente válido un Access o incluso un Openoffice calc (y en según qué casos incluso un *Bloc de notas*).

Si por el contrario, mi base de datos requiere cierta cantidad de accesos de usuarios diversos, control de integridad y otras funcionalidades, debería plantearme algo como MySQL.

Finalmente, si hablamos de una gran corporación que requiere herramientas avanzadas para grandes bases de datos quizás debamos plantearnos sistemas más potentes como Oracle o Postgresql.

En cuanto a si debe ser libre o no la decisión dependerá de si disponemos de personal cualificado en cuyo caso un sistema libre es más barato y potente. En caso contrario el sistema de pago es la elección más adecuada. No obstante todas las tecnologías disponen de servicios de soporte de gran calidad.

Por otro lado, los sistemas libres cada vez proveen una mejor y más eficiente documentación tanto a nivel oficial como a través de múltiples foros y *blogs*, lo que hace que cada vez lo use más gente y mejore continuamente. Sin embargo siempre existe el riesgo de que sea comercializado y deje de estar disponible de forma abierta.

En resumen, para la toma de esta decisión se tendrán en cuenta factores como:

- Documentación.
- Seguridad, control de acceso a los recursos.
- Volúmenes de información que soportará y número de accesos esperable.
- Complejidad en la migración de los datos.
- Soporte ofrecido.

Como siempre, al final la solución estará determinada por las características de la organización en cuanto a requerimientos de su sistema de información y al personal de que dispone, recursos económicos, etc.

En todo caso una solución general de compromiso podría ser aquella que involucre software libre y un contrato de soporte.

Postgreer – libre
Firebird → Libre

ACTIVIDADES 1.4

- Averigua el significado de los siguientes acrónimos en el contexto de los SGBD (SAP, ERP y DSS).
- Haz un listado de al menos tres sistemas gestores de bases de datos libres (*open source*) y tres comerciales indicando tres de sus características principales. Discute los motivos por los que consideras que algunas empresas ofrecen productos de software gratuitos.

↳ Plan.

ERP – Planificación de recursos de la empresa
Gran software que controla todo los programas utilizados pa la empresa
DSS → Sistema de soporte a la decisión.

RESUMEN DEL CAPÍTULO

En este capítulo introductorio hemos repasado someramente las principales características de los sistemas gestores de bases de datos actuales.

De los mismos, hemos visto sus componentes, funcionalidades principales y usuarios que trabajan con los denominados SGBD. Las posibilidades han crecido enormemente de un tiempo a esta parte tanto en lo que respecta a sistemas de pago como de código libre y hoy en día disponemos de un repertorio bastante amplio y potente de herramientas que hacen más fácil nuestra tarea, tanto como administración de bases de datos como de diseño y uso de las mismas.

EJERCICIOS PROPUESTOS

- **1.** Comenta qué se entiende por software libre considerando aspectos como:
  - Gratuidad.
  - Código fuente.
  - Uso comercial.
- **2.** Lista al menos 3 ventajas e inconvenientes de los productos de pago respecto a los libres.
- **3.** ¿Qué tiene que ver la administración SGBD con el diseño de bases de datos?
- **4.** Cita al menos 3 ventajas de usar bases de datos frente a los tradicionales sistemas de ficheros.
- **5.** Enumera al menos tres objetos típicos de una base de datos indicando su función.
- **6.** ¿Qué es una base de datos distribuida?
- **7.** Indica resumidamente las fases involucradas en el desarrollo de una base de datos desde su concepción hasta su puesta en marcha.
- **8.** ¿Para qué sirve un disparador en un SGBD?
- **9.** Explica con tus palabras qué es el diccionario de datos en un SGBD.
- **10.** Eres administrador de la base de datos. Indica un problema y su posible solución que te pueda surgir considerando dos casos: una base de datos con miles de usuarios y centrada en consultas, como un buscador, y otra de venta *online* con miles de usuarios y operaciones por segundo.

TEST DE CONOCIMIENTOS

1 ¿Qué es una base de datos?
- a) Un programa para organizar datos.
- b) Un software que facilita la gestión de datos.
- c) Un conjunto de datos organizados.
- d) Todo lo anterior.

2 ¿Cuál es el significado de GPL en el contexto informático?
- a) *General Public Library*.
- b) *Great Politic Licence*.
- c) *General Public Licence*.

3 ¿Cuáles de los siguientes objetos no son equivalentes a una tabla?
- a) Vista.
- b) Consulta.
- c) *Trigger*.
- d) Procedimiento.

4 ¿Qué se quiere decir cuando se habla de nivel conceptual?
- a) Lo que percibe el usuario.
- b) La imagen de la base de datos vista por el ordenador.
- c) El código para crear la base de datos.
- d) Una imagen de la base de datos independiente de la implementación física.

5 Las bases de datos son:
- a) Relacionales.
- b) Relacionales o jerárquicas.
- c) Primero eran en red y ahora son relacionales.
- d) La mayoría son relacionales.

6 Un modelo es:
- a) Una forma de representar información.
- b) Un programa para dibujar cajas y flechas.
- c) Una forma de representar un sistema.
- d) Una representación de un conjunto de datos.

7 Un sistema de información:
- a) Describe los datos de un sistema.
- b) Permite controlar la información de una empresa.
- c) Es el conjunto de elementos para gestionar la información de un sistema.
- d) Todo lo anterior.

8 ¿Qué es cierto respecto a los SGBD y bases de datos?
- a) No hay diferencia.
- b) Uno hace referencia a un software y una base es conceptual.
- c) Las bases de datos se crean necesariamente con un SGBD.
- d) Un SGBD es una herramienta CASE.

9 Los sistemas libres:
- a) Son más potentes y mejores que los comerciales.
- b) Son más baratos.
- c) Son más difíciles.
- d) Ninguno de los anteriores necesariamente.

10 La independencia física:
- a) Hace que podamos acceder a los datos desde cualquier equipo.
- b) Permite modificar los modelos independientemente de su almacenamiento.
- c) Evita problemas de redundancia.
- d) Hace que podamos usar las bases de datos independientemente del sistema operativo.

2 Instalación y configuración de un SGBD

OBJETIVOS DEL CAPÍTULO

- ✓ Instalar un servidor MySQL.
- ✓ Conocer las principales opciones de configuración de MySQL.
- ✓ Optimizar el funcionamiento de MySQL.
- ✓ Monitorizar MySQL.
- ✓ Aprender a gestionar ficheros de registro.
- ✓ Conocer la estructura del diccionario de datos de MySQL.

2.1 REVISIÓN MYSQL

MySQL es un sistema de gestión de bases de datos relacionales.

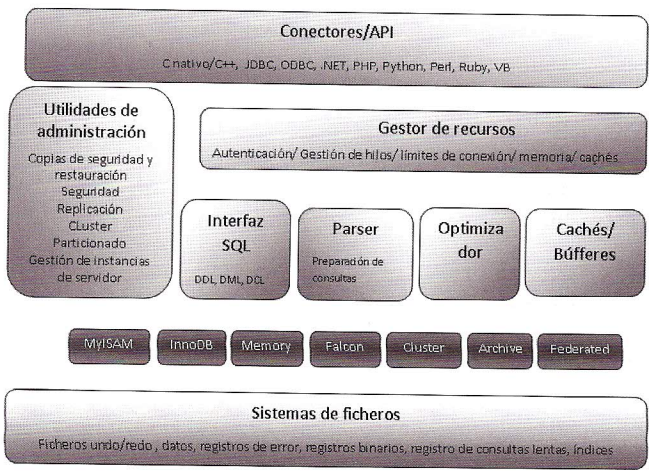
Es un SGBD Open Source, lo que significa que es posible para cualquiera usar y modificar el software. Cualquiera puede bajar el software MySQL desde Internet y usarlo libremente. Si lo deseas, puedes estudiar el código fuente y cambiarlo para adaptarlo a tus necesidades. El software MySQL usa la licencia GPL (GNU *General Public License*), descrita en el enlace: *http://www.fsf.org/licenses/*.

Es un sistema cliente/servidor que consiste en un servidor SQL *multi-threaded* (multihilo), que trabaja con diferentes programas y bibliotecas cliente, herramientas administrativas y un amplio abanico de interfaces de programación para aplicaciones (APIs).

A continuación, describiremos el proceso de instalación y configuración de un SGBD como MySQL. Es una de las tareas más importantes de un administrador. Aunque la instalación es un proceso relativamente sencillo es la configuración posterior lo que entraña mayor dificultad, siendo además un proceso constante muy dependiente de las circunstancias concretas de nuestras aplicaciones.

2.2 ARQUITECTURA MYSQL

El esquema general de organización de MySQL responde a la siguiente figura donde se representan los distintos subsistemas del sistema gestor de MySQL que permiten llevar a cabo todas las funciones del administrador de la base de datos.



2.3 INSTALACIÓN DE MYSQL

2.3.1 CUESTIONES GENERALES SOBRE LA INSTALACIÓN

Antes de instalar MySQL, debes tener en cuenta lo siguiente:

Plataforma

Que usarás en la instalación. Típicamente entre sistemas Windows o Linux. En la actualidad existen versiones disponibles para ambos.

Distribución

El desarrollo de MySQL se divide en entregas (*releases*) sucesivas, y el usuario puede decidir cuál es la que mejor satisface sus necesidades. Después de haber elegido la versión a instalar, se debe optar por un formato de distribución. Las entregas están disponibles en formato binario o código fuente.

Integridad

Descargar la distribución que se desea instalar y por seguridad verificar su integridad.

Versión

Escoger la versión de MySQL a instalar. La primera decisión a tomar es si se desea emplear una entrega "en producción" (estable) o una entrega de desarrollo. En el proceso de desarrollo de MySQL coexisten múltiples entregas, cada una con un diferente estado de madurez. En el momento de escribir este libro la versión más estable es la 5.1 aunque está en desarrollo y pruebas la 6.0. Es conveniente siempre trabajar con la estable, especialmente si nuestra intención es usarlo en entornos productivos. Para más información sobre versiones, consultar la bibliografía detallada al final del libro.

Formato

Escoger un formato de distribución. Después de haber decidido qué versión de MySQL instalar, debes elegir entre una distribución binaria o una de código fuente. Probablemente la elección más frecuente sea la distribución binaria por su facilidad de instalación, salvo que queramos configurar MySQL incluyendo o excluyendo algunas características de las distribuciones binarias estándar.

2.3.2 INSTALACIÓN DE MYSQL EN WINDOWS

Antes de instalar MySQL en Windows debes elegir un paquete de instalación

En la versión 5.0 de MySQL hay tres paquetes de instalación para elegir con distintas funcionalidades. Nosotros elegimos el completo para disponer de todas ellas.

Una vez descargado el paquete (generalmente con extensión *msi*) pulsamos dos veces con el ratón sobre el mismo y se inicia el proceso de instalación mediante un asistente. A partir de ese momento seremos consultados por una serie de parámetros para ajustar el servidor a nuestras necesidades.

Hay disponibles tres tipos de instalación: típica, completa y personalizada.

La instalación típica instala el servidor MySQL, el cliente de línea de comandos *mysql*, y las utilidades de línea de comandos. Los clientes y utilidades incluyen *mysqldump*, *myisamchk* y otras herramientas que facilitan la administración del servidor.

La instalación completa instala todos los componentes incluidos en el paquete. El paquete completo incluye componentes como el servidor incrustado (*embedded*), el conjunto de pruebas de rendimiento (*benchmarks*), *scripts* de mantenimiento y documentación.

La instalación personalizada otorga un control completo sobre los paquetes que se desean instalar y el directorio de instalación que se utilizará.

Dado que somos administradores usaremos la instalación personalizada de manera que podamos adaptar mejor el software a nuestras necesidades.

Una vez descargado el software de la página, comenzamos la instalación pulsando dos veces con el ratón en el ejecutable y, a partir de ahí, se nos muestra un cuadro de diálogo preguntándonos por el tipo de instalación.

Una vez que se pulsa con el ratón en el botón **Instalar**, el asistente de instalación de MySQL comienza el proceso de instalación y realiza ciertos cambios en el sistema, que se describen a continuación:

Cambios en el Registro

Se crea una clave de registro durante una situación típica de instalación, localizada en *HKEY_LOCAL_MACHINE\SOFTWARE\MySQL AB*. También crea una clave cuyo nombre es el número de versión principal (número de la serie) del servidor que se encuentra instalado, como MySQL *Server* 5.0. Contiene dos valores de cadena, *Location* y *Version*. La cadena *Location* contiene el directorio de instalación. La cadena *Version* contiene el número de entrega (*release*).

Estas claves del registro son útiles para ayudar a herramientas externas a identificar la ubicación en la que se instaló el servidor MySQL, evitando un rastreo completo del disco para descubrirla.

Cambios en el menú Inicio

El asistente de instalación crea una nueva entrada en el menú *Inicio* de Windows, bajo una opción cuyo nombre es el número de versión principal (número de la serie) del servidor que se encuentra instalado.

Se crean las siguientes entradas dentro de la nueva sección del menú *Inicio*:

- MySQL *Command Line Client*: cliente de línea de comandos.
- MySQL *Server Instance Config Wizard*: asistente de configuración.
- MySQL *Documentation*: es un vínculo a la documentación del servidor.

Cambios en el sistema de ficheros

El asistente de instalación de MySQL, por defecto instala el servidor MySQL en *C:\Program Files\MySQL\MySQL Server 5.0*, donde *Program Files* es el directorio de aplicaciones por defecto del sistema, y 5.0 es el número de versión principal del MySQL instalado. Ésta es la nueva ubicación donde se recomienda instalar MySQL, en sustitución de la antigua ubicación por defecto, *C:\mysql*.

Por defecto, todas las aplicaciones MySQL se almacenan en un directorio común localizado en *C:\Archivos de Programa\MySQL*, donde *Program Files* es el directorio de aplicaciones por defecto del sistema. Una instalación típica de MySQL en el ordenador de un desarrollador podría verse así:

```
C:\Archivos de Programa\MySQL\MySQL Server 5.0
C:\Archivos de Programa\MySQL\MySQL Administrator 1.0
C:\Archivos de Programa\MySQL\MySQL Query Browser 1.0
```

Donde los dos últimos directorios indican la ubicación de las herramientas gráficas (GUI) de administración y generación de consultas.

(En versiones superiores a la 5.0 estos dos últimos directorios corresponden al actual MySQL *Workbench* que integra el *administrator*, el *browser*, *migration toolkit* y añade una herramienta de diseño de bases de datos).

Esta proximidad entre los distintos directorios facilita la administración y el mantenimiento de todas las aplicaciones MySQL instaladas en un sistema en particular.

Es muy importante para un administrador tener clara la ubicación de directorios y archivos tras una instalación de cualquier programa, especialmente de servicios que serán accesibles por distintos clientes.

El directorio de instalación contiene los subdirectorios indicados en la siguiente tabla:

Tabla 2.1 Conformación directorios instalación MySQL

Directorio	Contenido
bin	Programas cliente y el servidor mysqld
data	Ficheros de registro y de bases de datos
Docs	Documentación
Examples	Programas y *scripts* de ejemplo
include	Ficheros de inclusión
lib	Bibliotecas
scripts	*Scripts* de utilidades
share	Ficheros con mensajes de error

ACTIVIDADES 2.1

- Realiza una instalación de MySQL tal como se detalla en el apartado anterior y comprueba los cambios comentados en tu sistema.
- Si dispones de un equipo con Linux (o en su defecto una máquina virtual, ver apéndice, por ejemplo con Virtualbox) intenta instalar MySQL siguiendo los pasos explicados en el manual de la web de MySQL. Observa las diferencias en cuanto a los cambios en el sistema de archivos usando los comandos de Linux para localizar ficheros.
- Haz una comprobación de integridad usando el algoritmo md5 (con la herramienta md5sums) del paquete MySQL server en su última versión estable. Indica los pasos seguidos.

2.4 CONFIGURACIÓN SERVIDOR

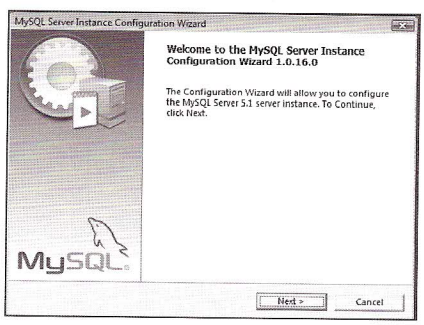
El asistente de configuración MySQL automatiza el proceso de configurar el servidor bajo Windows. Crea un fichero *my.ini* personalizado, realizando una serie de preguntas y aplicando las respuestas a una plantilla para generar un fichero *my.ini* apropiado para la instalación en curso.

2.4.1 CONFIGURACIÓN CON EL ASISTENTE

Arrancar el asistente de configuración de MySQL

El asistente de configuración generalmente se puede ejecutar de tres formas distintas: a continuación del asistente de instalación, pulsando con el ratón en la entrada MySQL *Server Instance Config Wizard* de la sección MySQL del menú Inicio o dirigiéndose al directorio *bin* de la instalación de MySQL y ejecutando directamente el fichero *MySQLInstanceConfig.exe*.

Una vez que pulsemos dos veces con el ratón aparecerá una ventana similar a la siguiente:



A partir de ahí y pulsando sobre *next* seguimos las siguientes indicaciones.

Escoger una opción de mantenimiento

Si el asistente de configuración detecta un fichero *my.ini* preexistente, se tiene la opción de reconfigurar el servidor o quitar la instancia borrando el fichero *my.ini* y deteniendo y quitando el servicio MySQL.

Para reconfigurar un servidor existente, debe escogerse la opción *Re-configure Instance* y pulsar con el ratón en el botón *next*. El fichero *my.ini* actual será renombrado como *mytimestamp.ini.bak*, donde *timestamp* es la fecha y hora en que el fichero *my.ini* existente se creó. Para quitar la instancia del servidor actual, debe seleccionarse la opción *Remove Instance* y pulsar con el ratón en el botón *Next*.

Si se selecciona la opción *Remove Instance*, se continúa con una ventana de confirmación. Al pulsar con el ratón en el botón *Execute*, el asistente de configuración detendrá y quitará el servicio MySQL, tras lo cual borrará el fichero *my.ini*. Los ficheros del servidor, incluyendo el directorio *data*, no se eliminarán.

Si se opta por *Re-configure Instance*, se continúa hacia el cuadro de diálogo *Configuration Type*, donde puede elegirse el tipo de instalación a configurar.

Escoger un tipo de configuración

Cuando se inicia el asistente de configuración MySQL para una instalación nueva o se escoge la opción *Re-configure Instance* para una configuración existente, se avanza hacia el cuadro de diálogo *Configuration Type*.

Hay disponibles dos tipos de configuración: **configuración detallada** (*Detailed configuration*) y **configuración estándar** (*Standard configuration*). La configuración estándar está orientada a usuarios nuevos que deseen comenzar rápidamente con MySQL sin tener que tomar varias decisiones relativas a la configuración del servidor. La detallada está dirigida a usuarios avanzados que deseen un control más preciso sobre la configuración del servidor.

Si se trata de un usuario nuevo de MySQL y necesita un servidor configurado para un ordenador de desarrollo con un único usuario, la configuración estándar debería cubrir sus necesidades. De este modo el asistente de configuración establece todas las opciones de configuración automáticamente, a excepción de las Opciones de servicio (*Service options*) y las de seguridad (*Security options*).

La configuración estándar establece opciones que pueden ser incompatibles con sistemas donde existen instalaciones de MySQL previas. Si se posee una instalación anterior además de la que se está configurando, se recomienda optar por la configuración detallada (*Detailed configuration*).

Tipo de servidor

Hay tres tipos de servidor distintos para elegir, y el tipo que se escoja afectará a las decisiones que el asistente de configuración MySQL tomará en relación al uso de memoria, disco y procesador.

- *Developer machine* (Ordenador de desarrollo): esta opción se aplica a ordenadores de escritorio orientados a un uso personal solamente. Se asume que se estarán ejecutando otras aplicaciones, por lo que el servidor MySQL se configura para utilizar una cantidad mínima de recursos del sistema.
- *Server machine* (Servidor): esta opción se aplica a servidores donde MySQL se ejecuta junto con otras aplicaciones de servidor como son FTP, correo electrónico y servidores web. MySQL se configura para utilizar una cantidad moderada de recursos del sistema.
- *Dedicated MySQL Server Machine* (Servidor MySQL dedicado): esta opción se aplica a ordenadores donde solamente se ejecuta el servidor MySQL. Se asume que no hay otras aplicaciones ejecutándose. El servidor MySQL se configura para utilizar todos los recursos disponibles en el sistema.

Base de datos

El cuadro de diálogo *Uso de la base de datos* (*Database usage*) permite indicar los gestores de tablas que se planea utilizar al crear tablas de MySQL. La opción que se escoja determinará si el motor de almacenamiento *InnoDB* estará disponible y qué porcentaje de los recursos de servidor estarán disponibles para *InnoDB*.

- **Base de datos multifuncional** (*Multifunctional database*): esta opción habilita tanto el motor de almacenamiento *InnoDB* como *MyISAM* y reparte los recursos uniformemente entre ambos. Se recomienda para usuarios que emplearán los dos motores de almacenamiento en forma habitual.
- **Base de datos transaccional exclusiva** (*Transactional database only*): esta opción habilita tanto el motor de almacenamiento *InnoDB* como *MyISAM*, pero destina más recursos del servidor al motor *InnoDB*. Se recomienda para usuarios que emplearán *InnoDB* casi exclusivamente, y harán un uso mínimo de *MyISAM*.
- **Base de datos no-transaccional exclusiva** (*Non-Transactional database only*): esta opción deshabilita completamente el motor de almacenamiento *InnoDB* y destina todos los recursos del servidor al motor *MyISAM*. Recomendado para usuarios que no utilizarán *InnoDB*.

Espacio de tablas InnoDB

Algunos usuarios pueden querer ubicar los ficheros *InnoDB* en una ubicación diferente al directorio de datos del servidor MySQL. Esto puede ser deseable si el sistema tiene disponible un dispositivo de almacenamiento con mayor capacidad o mayor rendimiento, como un sistema RAID.

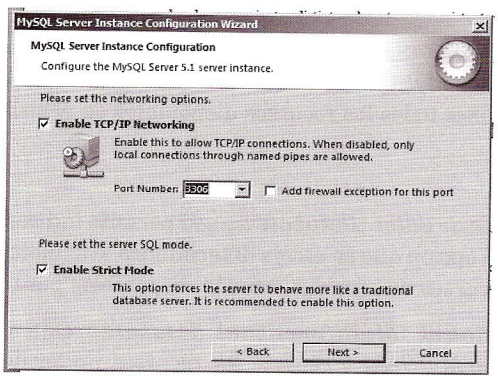
Conexiones concurrentes

Es importante establecer un límite para las conexiones simultáneas que se podrán establecer con el servidor MySQL para evitar que éste se quede sin recursos. El cuadro de diálogo conexiones simultáneas (*Concurrent connections*) permite indicar el uso que se planea darle al servidor y establecer en consecuencia el límite de conexiones simultáneas. También es posible introducir manualmente el límite.

- **Soporte de decisiones** (*Decision support* (DSS)/OLAP): debe escogerse esta opción si el servidor no necesitará una gran cantidad de conexiones simultáneas. El número máximo de conexiones se establece en 100, asumiéndose un promedio de 20 conexiones simultáneas.
- **Proceso de transacciones en línea** (*Online Transaction processing* (OLTP)): debe escogerse esta opción si el servidor necesitará un gran número de conexiones simultáneas. El número máximo de conexiones se establece en 500.
- **Configuración manual** (*Manual setting*): debe escogerse esta opción para establecer manualmente el número máximo de conexiones simultáneas que admitirá el servidor. El número deseado puede elegirse de una lista desplegable o teclearse si no figura en ella.

Redes

El cuadro de diálogo *Opciones de red* (*Networking options*) permite activar o desactivar el protocolo TCP/IP y modificar el número de puerto por el que se accederá al servidor MySQL.



El protocolo TCP/IP está activado por defecto. Para desactivarlo debe quitarse la marca de la casilla al lado de la opción Activar TCP/IP (*Enable* TCP/IP *networking*).

Por defecto se utiliza el puerto 3306 para acceder a MySQL. Para modificar este valor, el número deseado puede elegirse de una lista desplegable o teclearse si no figura en la lista. Si el puerto indicado ya se encuentra en uso, se solicitará la confirmación de la elección.

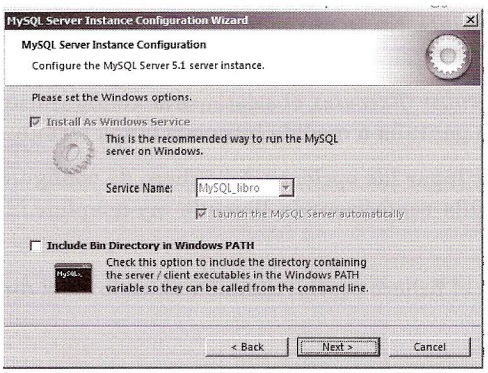
Conjunto de caracteres

El servidor MySQL soporta múltiples conjuntos de caracteres y es posible establecer uno por defecto, que se aplicará a todas las tablas, columnas y bases de datos, a menos que se sustituya. Debe emplearse el cuadro de diálogo *Character set* para cambiar en el servidor el conjunto de caracteres por defecto.

- **Juego de caracteres estándar** (*Standard character set*): esta opción establecerá a *Latin1* el juego de caracteres por defecto en el servidor. *Latin1* se usa para el inglés y la mayoría de idiomas de Europa Occidental.
- **Soporte multilingüe mejorado** (*Best support for multilingualism*): esta opción establece a UTF8 como el conjunto de caracteres por defecto en el servidor. UTF8 puede almacenar caracteres de muchos idiomas diferentes en un único juego.
- **Selección manual del conjunto de caracteres por defecto** (*Manual selected default character set*): esta opción se emplea cuando se desea elegir manualmente el juego de caracteres por defecto del servidor a través de una lista desplegable.

Opciones de servicio

En plataformas basadas en Windows NT, el servidor MySQL puede instalarse como un servicio. De ese modo, se iniciará automáticamente durante el inicio del sistema, e incluso será reiniciado automáticamente por Windows en caso de producirse un fallo en el servicio.

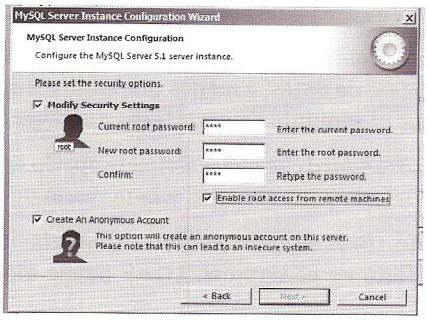


El asistente de configuración instala por defecto el servidor MySQL como un servicio, utilizando el nombre de servicio MySQL. Si se desea evitar la instalación del servicio, debe vaciarse la casilla al lado de la opción *Instalar* como servicio Windows (*Install as Windows service*). Se puede modificar el nombre del servicio eligiendo un nuevo nombre o tecleándolo en la lista desplegable provista.

Para instalar el servidor MySQL como un servicio pero que no se ejecute al iniciarse Windows, debe vaciarse la casilla al lado de la opción *Ejecutar el servidor MySQL automáticamente* (*Launch the MySQL server automatically*).

La ventana de diálogo de las opciones de seguridad

Se recomienda fuertemente que se establezca una contraseña para el usuario *root* del servidor MySQL. El asistente de configuración MySQL la solicita por defecto. Si no se desea establecer una contraseña debe vaciarse la casilla al lado de la opción *Modificar configuración de seguridad (Modify security settings*).



Para establecer la contraseña del usuario *root*, se debe introducir tanto en el cuadro de texto *Nueva contraseña de root* (*New root password*) como en *Confirmar* (*Confirm*). Si se está reconfigurando un servidor existente, también será necesario introducir la contraseña en vigencia dentro del cuadro de texto *Contraseña de root actual* (*Current root password*).

Para evitar que el usuario *root* inicie sesión desde cualquier punto de la red debe marcarse la casilla al lado de la opción *Root solo puede conectarse en modo local* (*Root may only connect from localhost*). Esto fortalece la seguridad de la cuenta de *root*.

Para crear una cuenta de usuario anónimo debe marcarse la casilla al lado de la opción *Crear una cuenta de anónimo* (*Create An Anonymous Account*). No se recomienda crear un usuario anónimo porque puede disminuir la seguridad del servidor y ocasionar dificultades de inicio de sesión y de permisos.

Después de pulsar en el botón *Ejecutar* (*Execute*), el asistente de configuración MySQL llevará a cabo una serie de tareas cuyo avance se mostrará en la pantalla a medida que cada etapa termine.

El asistente determina en primer lugar las opciones del fichero de configuración, basándose en las preferencias del usuario, y empleando una plantilla confeccionada llamada *my-template.ini* que se localiza en el directorio de instalación del servidor.

Luego, guarda dichas opciones en el fichero *my.ini*, también ubicado en el directorio de instalación.

No solo el servidor, sino también los programas auxiliares de MySQL permiten usar opciones, bien en ficheros o como argumentos en su invocación. Para una lista de las mismas podemos usar la ayuda volcándola por pantalla (o redireccionándolo a un fichero) ejecutando el programa con la opción *--help*.

Si se optó por crear un servicio de Windows para el servidor MySQL, el asistente de configuración creará e iniciará el servicio. Si se está reconfigurando un servicio existente, el asistente de configuración reiniciará el servicio para que tomen efecto los cambios realizados.

Después de que el asistente haya completado sus tareas, se mostrará un resumen. Pulsando el botón *Terminar* (*Finish*) se abandonará el asistente.

Arrancar y detener el servidor la primera vez

Para iniciar el servidor, se emplea el programa *mysqld.exe* desde la consola *msdos* de Windows:

```
C:\> C:\Program Files\MySQL\MySQL Server 5.0\bin\mysqld -console
```

En servidores que incluyen soporte para *InnoDB*, se debería mostrar algo similar a esto a medida que el servidor se inicia:

```
InnoDB: The first specified datafile c:\ibdata\ibdata1 did not exist:
InnoDB: a new database to be created!
InnoDB: Setting file c:\ibdata\ibdata1 size to 209715200
InnoDB: Database physically writes the file full: wait...
InnoDB: Log file c:\iblogs\ib_logfile0 did not exist: new to be created
InnoDB: Setting log file c:\iblogs\ib_logfile0 size to 31457280
InnoDB: Log file c:\iblogs\ib_logfile1 did not exist: new to be created
InnoDB: Setting log file c:\iblogs\ib_logfile1 size to 31457280
InnoDB: Log file c:\iblogs\ib_logfile2 did not exist: new to be created
InnoDB: Setting log file c:\iblogs\ib_logfile2 size to 31457280
InnoDB: Doublewrite buffer not found: creating new
InnoDB: Doublewrite buffer created
InnoDB: creating foreign key constraint system tables
InnoDB: foreign key constraint system tables created
011024 10:58:25  InnoDB: Started
```

Cuando el servidor finaliza su secuencia de inicio, después de este mensaje se debería ver un texto similar al siguiente, que indica que el servidor está listo para dar servicio a conexiones de clientes:

```
mysqld: ready for connections
Version: '5.0.9-beta'  socket: ''  port: 3306
```

El servidor continúa con la emisión por pantalla de cualquier otro mensaje de diagnóstico que se genere. Puede abrirse una nueva ventana de consola en la cual ejecutar programas cliente (si cerramos la consola en que hemos arrancado el servidor, éste se cerrará también).

Si se omite la opción *--console*, el servidor dirige la información de diagnóstico hacia el registro de errores en el directorio de datos (por defecto, *C:\Program Files\MySQL\MySQL Server 5.0\data*). El registro de errores es el fichero con extensión *.err*.

El siguiente comando detendrá al servidor MySQL:

```
C:\> C:\Program Files\MySQL\MySQL Server 5.0\bin\mysqladmin -u root shutdown
```

Esto invoca la utilidad administrativa de MySQL (*mysqladmin*) para conectarse al servidor y transmitirle la orden de finalización. El comando se conecta como el usuario *root* de MySQL, el cual es la cuenta administrativa por defecto en el sistema de permisos de MySQL. Debe advertirse que los usuarios en este sistema son enteramente independientes de cualquier usuario de inicio de sesión perteneciente a Windows.

El comando *mysqld --verbose --help* sirve para mostrar todas las opciones que *mysqld* es capaz de reconocer en el arranque.

Arrancar MySQL como un servicio de Windows

Antes de instalar MySQL como un servicio Windows, se debería detener primero el servidor (si está en ejecución) mediante el siguiente comando indicado en la anterior sección:

```
C:\> C:\Program Files\MySQL\MySQL Server 5.0\bin\mysqladmin -u root shutdown
```

Este comando instalará el servidor como un servicio:

```
C:\> mysqld -install
```

Si se producen problemas al instalar *mysqld* como un servicio usando solo el nombre del servidor, debe intentarse indicando la ruta completa. Por ejemplo:

```
C:\>Program Files\MySQL\MySQL Server 5.0\bin\mysqld -install
```

La ruta al directorio *bin* de MySQL puede agregarse a la variable de entorno de Windows PATH accesible desde el *Escritorio* de Windows al pulsar con el botón derecho del ratón en el ícono *Mi PC* y seleccionar *Propiedades Opciones Avanzadas → Variables de Entorno*.

MySQL soporta los siguientes argumentos adicionales cuando se instala como servicio:

- Puede indicarse un nombre para el servicio inmediatamente a continuación de la opción *--install*. El nombre por defecto es MySQL.
- Si se indica un nombre de servicio solamente puede especificarse una opción a continuación. Por convención, ésta debería ser *–defaults-file=file_name* para indicar el nombre de un fichero de opciones (típicamente *my.ini*), en cuyo contenido el servidor leerá las opciones de configuración cuando se inicie.

A modo de un ejemplo, considérese el siguiente comando:

```
C:\>Program Files\MySQL\MySQL Server 5.0\bin\mysqld --install MySQL--defaults-file=C:\
my-opts.cnf
```

Aquí, el nombre de servicio por defecto (MySQL) se suministró a continuación de la opción *--install*. Si no se hubiera indicado la opción *--defaults-file*, este comando hubiese tenido como efecto que el servidor leyera el grupo [*mysqld*] de los ficheros de opciones estándar. No obstante, debido a que la opción *--defaults-file* se encuentra presente, el servidor leerá las opciones del grupo [*mysqld*], pero solo del fichero indicado.

También es posible especificar opciones como *Parámetros de Inicio* (*Start parameters*) en la utilidad *Services* de Windows antes de iniciar el servicio MySQL.

Una vez que el servidor MySQL ha sido instalado como servicio, será iniciado automáticamente después del arranque de Windows. El servicio también puede iniciarse desde la utilidad *Services*, o empleando el comando *net start mysql*.

Cuando se ejecuta como servicio, *mysqld* no tiene acceso a una ventana de consola, por lo que no puede mostrar mensajes. Si *mysqld* no se inicia, debe consultarse el registro de errores para ver si el servidor ha dejado allí mensajes que indiquen la causa del problema. El registro de errores se encuentra en el directorio de datos de MySQL (por ejemplo, *C:\Program Files\MySQL\MySQL Server 5.0\data*). Es el fichero con extensión *.err*.

Cuando un servidor MySQL se instala como servicio, se detendrá automáticamente si estaba en ejecución al momento de cerrar Windows. También puede detenerse manualmente, ya sea a través de la utilidad *Services*, del comando *NET STOP MYSQL* o del comando *mysqladmin shutdown*.

También existe la opción de instalar el servidor como un servicio de inicio manual si no se desea que el servicio se inicie en cada arranque de Windows. Para esto, debe emplearse la opción *--install-manual* en lugar de *--install*:

```
C:\> C:\Program Files\MySQL\MySQL Server 5.0\bin\mysqld -install-manual
```

Para cancelar un servidor que fue instalado como servicio, primero se lo debe detener, si está en ejecución, por medio del comando *NET STOP MYSQL*. Después de esto se usará la opción *--remove* para cancelarlo:

```
C:\> C:\Program Files\MySQL\MySQL Server 5.0\bin\mysqld -remove
```

Comprobar la instalación

Cualquiera de los siguientes comandos permitirá comprobar si el servidor MySQL está en funcionamiento:

Todos ellos (y otros) son programas incluidos en el directorio *bin* de MySQL por lo cual es conveniente incluir dicho directorio en el *PATH* del sistema para evitar así tener que indicar toda la ruta para ejecutarlos.

Para visualizar las bases de datos:

EJEMPLO 2.1

```
C:\>Program Files\MySQL\MySQL Server 5.0\bin\mysqlshow

Salida:
+-----------+
| Databases |
+-----------+
| mysql     |
| test      |
+-----------+
```

Para visualizar la base de datos MySQL para la gestión de cuentas y permisos:

EJEMPLO 2.2

```
C:\> C:\mysql\bin\mysqlshow mysql
+--------------+
|    Tables    |
+--------------+
| columns_priv |
| db           |
| func         |
| host         |
| tables_priv  |
| user         |
+--------------+
```

Para comprobar las cuentas iniciales:

EJEMPLO 2.3

```
C:\> C:\mysql\bin\mysql -e "SELECT Host,Db,User FROM mysql.user"
C:\> C:\Program Files\MySQL\MySQL Server 5.0\bin\mysqlshow -u root
```

Para comprobar el estado actual del servidor:

EJEMPLO 2.4

```
C:\> C:\Program Files\MySQL\MySQL Server 5.0\bin\mysqladmin version status proc
```

Para conectarme a una base de datos directamente:

EJEMPLO 2.5

```
C:\> C:\Program Files\MySQL\MySQL Server 5.0\bin\mysql test
```

Lo anterior son programas incluidos en el directorio *bin* de la instalación. Para un uso más detallado de estos programas podemos volcar su ayuda bien por pantalla con la opción *--help* como ya es habitual o bien enviándola a un fichero que luego podemos consultar. Es lo que hacemos con el siguiente comando para el programa *mysqladmin*:

EJEMPLO 2.6

```
C:\> C:\Program Files\MySQL\MySQL Server 5.0\bin\mysqladmin -help>mysqladmin_
ayuda
```

ACTIVIDADES 2.2

- Ejecuta de nuevo el programa asistente de configuración de MySQL (en la carpeta *bin*) y configura el servidor para escuchar en el puerto 4000 con el nombre de servicio *servmysql2* y otras opciones que consideres apropiadas para un entorno normal de trabajo. Guarda previamente el fichero *my.ini* y compáralo con el nuevo generado por el asistente.
- Piensa en las razones por las que puede ser interesante arrancar dos o más servicios MySQL en un mismo equipo. ¿Qué opciones como mínimo deben ser distintas en ambos ficheros para que puedan arrancar ambos servicios?
- Añade dos *servicios MySQL a Windows* con los nombres *mysql1* y *mysql2*, de manera que se inicien automáticamente con el sistema. Cada uno debe usar un fichero de configuración distinto (por ejemplo: *my1.ini*, *my2.ini*).

- Carga en el servidor que consideres el *script* de las bases de datos para usar en el resto del libro. Hazlo de dos formas:
  a. Con el programa cliente de MySQL.
  b. Con el comando *source* de MySQL.
- Ejecuta el comando *SHOW DATABASES* para ver las bases creadas en el servidor. Hazlo de dos modos, desde el cliente y desde la consola MSDOS.

2.4.2 VARIABLES Y OPCIONES EN MYSQL

El servidor dispone de un conjunto de variables que determinan su funcionamiento. Una de las tareas más importantes del administrador implica el conocimiento y ajuste óptimo de los valores de las mismas según los requerimientos de las aplicaciones.

Debemos diferenciar entre variables del servidor y las opciones que permiten modificar el valor de las variables.

Podemos ajustar los valores de las diferentes variables usando ficheros de opciones, incluyendo dichas opciones cuando arrancamos el servidor o modificándolas con el comando *SET*.

La mejor forma de conocer las variables es buscarlas cuando se necesitan. En tal caso debemos consultar el manual donde disponemos de una referencia detallada en donde para cada variable se detallan normalmente su nombre largo y corto para usar en línea de comandos, su nombre para ficheros de opciones (no siempre coincide), si son modificables con *SET*, el nombre de la variable, su alcance (global o de sesión) y si es o no dinámica (modificable en tiempo de ejecución), su dominio o valores permitidos, su tipo y su valor por defecto.

Cuando instalamos el servidor la mayor parte adquiere su valor por defecto, pero podemos modificarlas de tres modos distintos:

Opciones en línea de comandos

Cuando iniciamos el servidor (o cualquier otro programa de MySQL) podemos incluir valores para las variables directamente en la línea de comandos. La mayoría admiten una forma larga precedida por dos guiones, o corta, precedida por un guión. Algunas de ellas implican cierto valor, en cuyo caso serán seguidas por un igual y el valor correspondiente para el caso largo, o directamente por el valor (sin espacio en blanco) para el caso corto. Si los valores asignados incluyen espacios deben ir entre comillas.

El siguiente ejemplo ilustra lo dicho:

EJEMPLO 2.7

```
C:\>mysql -uroot --password=root -e "show databases"
```

En este ejemplo usamos el programa cliente *mysql* para conectarnos con un usuario y contraseña especificados en la línea de comandos y ejecutar un comando con la opción *–e* (*execute*) para ejecutar un comando que nos mostrará las bases de datos en nuestro servidor.

EJEMPLO 2.8

```
C:\>mysqld -skip-grant-tables --console
```

En este caso iniciamos el servidor sin tablas de permisos, de forma que cuando accede cualquier cliente lo podrá hacer sin necesidad de autenticarse. Además, los mensajes serán enviados a la consola.

Cuando se cambia una variable usando las opciones de arranque, los valores de la variable pueden darse con un sufijo K, M o G para indicar *kilobytes*, *megabytes* o *gigabytes*, respectivamente. Por ejemplo, el siguiente comando arranca el servidor con un tamaño de *key buffer* de 16 *megabytes*:

```
C:\>mysqld -key_buffer_size=16M
```

No importa que los sufijos se escriban en mayúscula o minúscula; 16 M y 16 m son equivalentes.

Ficheros de opciones

Cuando queremos que las opciones sean permanentes lo normal es hacer que los programas (*mysqld* entre ellos) de MySQL puedan leer opciones de inicio desde ficheros de opciones (también llamados ficheros de configuración). Estos proporcionan una forma conveniente de especificar opciones comúnmente usadas.

Los siguientes programas soportan ficheros de opciones: *myisamchk*, *myisampack*, *mysql*, *mysql.server*, *mysqladmin*, *mysqlbinlog*, *mysqlcc*, *mysqlcheck*, *mysqld_safe*, *mysqldump*, *mysqld*, *mysqlhotcopy*, *mysqlimport* y *mysqlshow*.

Sin embargo podemos unificar las opciones para todos ellos en un único fichero llamado *my.ini* para Windows o *my.cnf* en Linux.

Estos ficheros permiten incluir los siguientes elementos:

- Comentarios precedidos con #.
- Opciones para programas o grupos usando corchetes.
- Opciones tipo opción = valor.
- Opciones sin valor asociado.

Veamos su uso en el siguiente ejemplo:

EJEMPLO 2.9

```
[client]
port=3306
password="guara"

[mysqld]
port=3306
key_buffer_size=16M
max_allowed_packet=8M

[mysqldump]
quick

[mysqladmin]
force
```

En este caso, cuando iniciemos el servidor se pondrá a escuchar en el puerto 3306 los valores de memoria de claves y un máximo tamaño de paquete de 8 M.

Así mismo, todos los clientes deberán conectarse al puerto 3306 y su *password* será *guara*. Esta opción se aplicará a todos los clientes que se conecten al servidor.

Por último cuando hagamos una copia de seguridad usando *mysqldump* ésta se hará en modo rápido o *quick* y *mysqladmin* se ejecutará con la opción *force*.

Comando SET

En MySQL, podemos asignar un valor a una variable de sistema dinámica del servidor usando con el comando *SET* cuyos detalles estudiaremos en la siguiente sección.

Como resumen incluimos una tabla con los tipos de variables del servidor *mysql*:

Tabla 2.2 Tipos de variables en MySQL

Tipos	Descripción
Dinámicas-Estáticas	Según si son o no modificables en tiempo de ejecución.
Globales-De sesión	Según si afectan al comportamiento de todas las conexiones o solo a un cliente en particular.
De estado-De sistema	Según si se refieren al estado del servidor en un momento dado o a su comportamiento en general.

2.4.3 VARIABLES DE SISTEMA DEL SERVIDOR

Una vez instalado debemos revisar la conformación de directorios y muy especialmente el fichero *my.ini*. En él se recogen los parámetros que determinan el funcionamiento de nuestro servidor. Existen múltiples opciones para configurarlo, todas ellas perfectamente documentadas en el manual o la web de MySQL. En esta sección nos centraremos en las más relevantes y típicamente usadas.

Para modificar el fichero *my.ini*, se debe abrir con un editor de texto (recomiendo *notepad++* de descarga y uso libre: *http://notepad-plus-plus.org/*) y realizar cualquier cambio necesario. También se puede modificar con la utilidad *mysqladmin* comentada en próximos capítulos.

Para una lista de las variables del servidor podemos ejecutar *mysqld --verbose –help*, también podemos enviar la salida a un fichero si queremos guardarla y verlas con detenimiento, *mysqld --help > fichero_ayuda*.

A continuación veremos algunas de las variables más relevantes agrupadas según su funcionalidad:

Variables generales

- *-help*, -?

  Muestra un mensaje de ayuda corto y sale. Con las opciones *--verbose* y *--help* simultáneamente podemos ver el mensaje detallado.

- *-basedir=path*, -b *path*

  El *path* al directorio de instalación de MySQL. Todas las rutas se resuelven normalmente relativas a ésta.

- *-bind-address*=IP

  La dirección IP asociada al servidor para cuando el equipo dispone de más de una IP.

- *-console*

  Escribe los mensajes de error por *stderr* y *stdout* incluso si *--log-error* está especificado. En Windows, *mysqld* no cierra la pantalla de consola si se usa esta opción.

- *-datadir=path, -h path*

  La ruta al directorio de datos.

- *-default-storage-engine=type*

  Esta opción es un sinónimo para *--default-table-type*. Cambia el valor por defecto de tipo de tablas (normalmente *Innodb* o *MyIsam*).

- *-default-time-zone=type*

  Cambia la zona horaria del servidor. Esta opción cambia la variable global de sistema *time_zone*. Si no se da esta opción, la zona horaria por defecto es la misma que la del sistema.

- *-max_binlog_size*

  Es el tamaño del fichero de registro binario que registra los comandos SQL que modifican objetos de las bases de datos.

- *-ndbcluster*

  Si el binario incluye soporte para el motor de almacenamiento *NDBCluster*, la opción por defecto de desactivar el soporte para MySQL *Cluster* puede ignorarse usando esta opción.

- *-skip-grant-tables*

  Esta opción hace que el servidor no use el sistema de privilegios en absoluto. Da a cualquiera que tenga acceso al servidor acceso ilimitado a todas las bases de datos. Puede hacer que un servidor en ejecución empiece a usar las tablas de privilegios de nuevo ejecutando *mysqladmin flush-privileges* o el comando *mysqladmin reload* desde una consola de sistema, o utilizando el comando *FLUSH PRIVILEGES* desde la consola de *MySQL*.

- *-skip-ndbcluster*

  Deshabilita el motor de almacenamiento *NDBCluster*. Este es el comportamiento por defecto para los binarios compilados con el soporte para el motor de almacenamiento de *NDBCluster*, lo que significa que el sistema reserva memoria y otros recursos para este motor de almacenamiento solo si *--skip-ndbCluster* está habilitado explícitamente por la opción *--ndbCluster*.

- *-standalone*

  Solo para sistemas basados en Windows-*NT*; le dice al servidor MySQL que no se ejecute como servicio.

- *-version, -V*

  Muestra información de versión y termina.

  Variables de registro o *log*.

- *-general-log*

  Permite activar o desactivar el registro general.

- *-general-log-file, -l [file]*

  Log de conexiones y consultas en este fichero. Si no especifica un nombre de fichero, MySQL usará *host_name.log* como nombre de fichero.

- *-log-bin=[file]*

  El fichero de *logs* binario. Registra todas las consultas que cambian datos en este fichero. Se usa para copias de seguridad y replicación. Se recomienda especificar un nombre de fichero; en caso contrario MySQL usará *host_name-bin*.

- *- log-bin-index[=file]*

  El fichero *índice* para *log* binario. Si no especifica un nombre de fichero y si no especifica uno en *--log*-bin, MySQL, usará *host_name-bin.index* como el nombre de fichero.

- *-log-error[=file]*

  Mensajes de error y de arranque en este fichero. Si no especifica un nombre de fichero, MySQL usa *host_name.err* como nombre de fichero. Si el nombre de fichero no tiene extensión, una extensión *.err* se añade al nombre.

- *-log-queries-not-using-indexes*

  Si usa esta opción con *--log-slow-queries*, las consultas que no usan índices también se *loguean* en el *log* de consultas lentas.

- *-log-slow-queries[=file]*

  Registra todas las consultas que han tardado más de *long_query_time* segundos en ejecutarse.

- *-log-warnings[=level], -W [level]*

  Muestra advertencias tales como *Aborted connection* en el *log* de errores. Se recomienda activar esta opción, por ejemplo, si usamos replicación (nos da más información acerca de lo que está ocurriendo, como mensajes acerca de fallos de red y *reconexiones*). Esta opción está activada por defecto, para desactivarla, debemos usar *--skip-log-warnings*. O ponerla a *level = 0*. Las conexiones abortadas no se registran en el *log* de errores a no ser que el valor *level* sea mayor que 1.

Variables relacionadas con la memoria y la caché

- *-join_buffer_size*

  Es la mínima cantidad de memoria usada para consultas que implican combinaciones (*join*) de tablas sin usar índices.

- *-read_buffer_size*

  Cada hilo (*thread*) que realiza un recorrido secuencial sobre los valores de una tabla crea un *buffer* del tamaño indicado por esta variable.

- *-read_rnd_buffer_size*

  Indica el valor de memoria requerida cada vez que se leen registros de una tabla según el orden de sus índices. Es especialmente útil en consultas de ordenación.

- *-sort_buffer_size*

  Cada vez que se necesita hacer una ordenación se crea un *buffer* del tamaño indicado por esta variable.

- *-sql_buffer_result*

  Con valor uno hace que se creen tablas temporales para almacenar los resultados de consultas. Muy útil cuando para no tener que bloquear tablas durante mucho tiempo en consultas de gran tamaño que pueden tardar tiempo en ser enviadas al cliente.

- *-binlog_cache_size*

  Tamaño de la caché para guardar cambios del registro binario durante una transacción.

- *-query_cache_size*

  La cantidad de memoria usada para cachear resultados de consultas.

- *-thread_cache_size*

  Numero de hilos que pueden ser reutilizados por nuevos clientes

  Variables relacionadas con tablas *MyISAM*.

- *-key_buffer_size*

  Es el tamaño de memoria usado para bloques de índices.

- *-bulk_insert_buffer_size*

  Tamaño de memoria usada para acelerar operaciones de inserción de datos mediante comandos *INSERT* o *LOAD DATA*.

Variables tipo have

Indican la tenencia o no de cierta característica, como por ejemplo *have_crypt* cuyo valor (ON u OFF) indica si se puede usar la función de encriptación *ENCRYPT()*.

Hemos comentado variables generales que afectan al uso de cualquier tipo de motor de almacenamiento. Existen, sin embargo, grupos de variables que afectan en particular a los motores *MyISAM* e *InnoDB*. La mayoría de ellas comienzan con el prefijo *myisam* e *innodb*, respectivamente.

Variables relacionadas con la red

Determinan parámetros relacionados con conexiones, resolución de nombres, etc.

- *-skip-networking*

  No escucha conexiones TCP/IP en absoluto. Toda interacción con *mysqld* debe hacerse vía *named pipes* o memoria compartida (en Windows) o ficheros *socket* en Unix. Esta opción se recomienda encarecidamente en sistemas donde solo se permitan clientes locales.

- *-skip-name-resolve*

  Evita que el servidor compruebe el nombre del equipo que se conecta. Es útil si no disponemos de una red con nombres.

- *-net_buffer_length*

  Es el tamaño del *buffer* asociado a cada cliente para la conexión y para los resultados de sus consultas.

2.4.4 VARIABLES DE ESTADO DEL SERVIDOR

El servidor mantiene muchas variables de estado que proveen de información sobre su estado. Pueden consultares con el comando *SHOW STATUS LIKE 'patron'*, siendo patrón una cadena dentro de la variable.

A continuación detallamos las más importantes:

Variables generales

- *-Aborted_clients*

  El número de conexiones que han sido abortadas debido a que el cliente murió sin cerrar la conexión apropiadamente.

- *-Aborted_connects*

  El número de intentos de conexión al servidor MySQL que han fallado.

- *-Bytes_received*

  El número de *bytes* recibidos desde todos los clientes.

- *-Bytes_sent*

  El número de *bytes* enviados hacia todos los clientes.

- *-Connections*

  El número de intentos de conexión (con éxito o no) al servidor MySQL.

- *-Last_query_cost*

  El coste total de la última consulta compilada tal como ha sido computada por el optimizador de consultas. Es útil para comparar el coste de diferentes planes de ejecución para la misma consulta. El valor por defecto de 0 significa que no se ha compilado ninguna consulta todavía.

- *-Max_used_connections*

  El número máximo de conexiones que han sido utilizadas simultáneamente desde que el servidor ha sido iniciado.

- *-Open_tables*

  El número de tablas que están actualmente abiertas.

- *-Opened_tables*

El número de tablas que han sido abiertas. Si *Opened_tables* es grande, probablemente el valor de *table_cache* es demasiado pequeño.

- *-Threads_created*

  El número de subprocesos creados para gestionar conexiones. Si Threads_created es grande, debería incrementar el valor de *thread_cache_size*. La tasa de éxitos de la caché puede ser calculada como *Threads_created / Connections*.

- *-Threads_running*

  El número de subprocesos que no están durmiendo.

- *-Uptime*

  El número de segundos que el servidor ha estado funcionando ininterrumpidamente.

Variables tipo Com

Son todas ellas variables de estado que contabilizan el número de veces que cierto comando se ha ejecutado en la sesión actual. Su formato siempre es *Com_xxx* siendo *xxx* el comando *sql* computado, por ejemplo *Com_insert* contabiliza el número de inserciones realizadas. Son todas ellas variables de estado y, por tanto, no pueden modificarse ni incluirse en ficheros de opciones.

Variables tipo Handler

Variables relacionadas con operaciones de lectura y escritura sobre las tablas, su formato es *Handler_xxx* siendo *xxx* la cadena que indica la operación realizada. Por ejemplo *Handler_read_key* indica el número de veces que se ha leído una fila de una tabla basándose en el índice.

Variables tipo InnoDB

Los motores *InndoDB* dada su mayor complejidad disponen de un conjunto específico de variables de estado que facilitan su optimización. La mayoría tienen el formato *Innodb_data_read* indica la cantidad de datos leídos desde que el servidor fue iniciado.

Variables tipo Key

Para contaje del número de operaciones relacionadas con índices. Su formato es *Key_xxx*, donde *xxx* representa la operación correspondiente. Por ejemplo *key_reads* computa el número de lecturas de disco de bloques de índices.

Variables tipo Qcache

Variables relacionadas con número de operaciones sobre la *query cache*. Su formato es *Qcache_xxx* siendo *xxx* la operación correspondiente. Por ejemplo, *Qcache_inserts* muestra el número de consultas añadidas a la caché.

Variables tipo ssl

Indican aspectos relacionados con la criptografía de clave asimétrica o SSL. Su formato es *ssl_xxx*, siendo *xxx* el nombre de la característica SSL que queremos mostrar. Por ejemplo, *ssl_cipher_list* contiene los métodos de cifrado que soporta nuestro servidor.

Variables relacionadas con Threads

Son variables relacionadas con los hilos o conexiones creadas en el servidor, por ejemplo, *threads_created* indica el número de conexiones existentes en el servidor.

2.4.5 COMANDOS PARA GESTIÓN DE VARIABLES

A continuación indicamos los comandos útiles para la gestión de variables de servidor además de para la obtención de información relevante de forma rápida y eficiente.

Comandos de consulta: *SHOW*

Para obtener los valores de las variables podemos usar distintos comandos SHOW que describimos a continuación:

- *SHOW VARIABLES*

```
mysql> SHOW VARIABLES;
```

También podemos filtrar la salida con *LIKE*.

EJEMPLO 2.10

```
mysql> SHOW VARIABLES LIKE '%cadena_busqueda%';
```

- *SHOW STATUS*

Para variables de estado podemos ver sus valores utilizando la sentencia *SHOW STATUS*.

```
mysql> SHOW STATUS;
+-----------------------------------+------------+
| Variable_name                     | Value      |
+-----------------------------------+------------+
| Aborted_clients                   | 0          |
| Aborted_connects                  | 0          |
| Bytes_received                    | 155372598  |
| Bytes_sent                        | 1176560426 |

...

| Connections                       | 30023      |
| Created_tmp_disk_tables           | 0          |
| Created_tmp_files                 | 3          |
| Created_tmp_tables                | 2          |

...

| Threads_created                   | 217        |
```

```
| Threads_running                    | 88         |
| Uptime                             | 1389872    |
+------------------------------------+------------+
```

Muchas variables de estado son inicializadas a 0 por la sentencia *FLUSH STATUS*.

También podemos filtrar la salida de *SHOW STATUS* con *LIKE*.

```
mysql> SHOW STATUS LIKE '%cadena_busqueda%';
```

- *SHOW BINARY LOGS*

Sirve para mostrar un listado de los ficheros de *log* binarios en el servidor, así como su tamaño.

```
mysqL> SHOW BINARY LOGS;
+---------------+-----------+
| Log_name      | File_size |
+---------------+-----------+
| binlog.000015 |    724935 |
| binlog.000016 |    733481 |
+---------------+-----------+
```

- *SHOW BINLOG EVENTS*

```
SHOW BINLOG EVENTS
[IN 'log_name'] [FROM pos] [LIMIT [offset,] row_count]
```

Muestra el contenido del fichero binario indicado (cuidado ya que estos ficheros pueden ser muy grandes y saturar recursos del sistema. Conviene usar límites).

- *SHOW ENGINE*

Muestra información sobre *logs* o estado del motor de almacenamiento en cuestión (*BDB, INNODB, NDB y NDBCLUSTER*).

```
SHOW ENGINE engine_name {LOGS | STATUS }
```

- *SHOW ENGINES*

Da información sobre el tipo de tablas o motores soportados por el servidor.

```
mysql> SHOW ENGINES\G
*************************** 1. row ***************************
 Engine: MyISAM
Support: DEFAULT
Comment: Default engine as of MySQL3.23 with great performance
*************************** 2. row ***************************
 Engine: MEMORY
Support: YES
Comment: Hash based, stored in memory, useful for temporary tables
```

- *SHOW ERRORS*

Muestra los mensajes de error generados en el servidor y cuyo número depende de *LIMIT*. También muestra el número de errores usando *COUNT*.

```
SHOW ERRORS [LIMIT [offset,] row_count]
```

```
SHOW COUNT(*) ERRORS
```

- *SHOW WARNINGS*

Igual que con *show error* este comando nos permite ver cierto número de avisos del servidor.

```
SHOW WARNINGS [LIMIT [offset,] row_count]
SHOW COUNT(*) WARNINGS
```

- *SHOW OPEN TABLES*

Muestra tablas abiertas de la base de datos indicada y cumpliendo con el patrón *'pattern'* o condición *WHERE*.

```
SHOW OPEN TABLES [{FROM | IN} db_name]
[LIKE 'pattern' | WHERE expr]
```

- *SHOW PROCESSLIST*

Muestra los procesos activos en el servidor. Esta información es también accesible usando el programa *mysqladmin* con la opción *processlist*.

```
C:>mysqladmin processlist
```

- *SHOW [FULL] PROCESSLIST*

```
mysql> SHOW FULL PROCESSLIST\G
   *************************** 1. row ***************************
Id: 1
User: system user
Host:
db: NULL
Command: Connect
Time: 1030455
State: Waiting for master to send event
Info: NULL
```

Si omitimos la opción *FULL* se muestran los 100 primeros caracteres del campo *Info*.

- *SHOW STATUS*

Muestra el estado de las variables de estado del servidor según sean globales o de sesión. Este comando admite filtros por patrón o condiciones WHERE.

```
SHOW [GLOBAL | SESSION] STATUS
[LIKE 'pattern' | WHERE expr]
```

- *SHOW VARIABLES*

Muestra variables (globales o de sesión) del servidor que cumplan con un patron o condición.

```
SHOW [GLOBAL | SESSION] VARIABLES [LIKE 'pattern' |
WHERE expr]
```

Comandos modificación de variables: SET

El comando *SET* nos permite modificar o crear nuevas variables en tiempo de ejecución. Solo es válido para variables dinámicas o para crear nuevas variables de usuario. Su sintaxis es:

```
SET variable_assignment [, variable_assignment] ...

variable_assignment:

user_var_name = expr | [GLOBAL | SESSION] system_var_name = expr | [@@global. | @@
session. | @@]system_var_name = expr
```

Aquí, una asignación de variables se asocia con un valor o expresión numérica o de tipo cadena especificando si es de tipo global o de sesión. En ese sentido, *GLOBAL* es equivalente a *@@global* y *SESSION* a *@@session* o *@@*.

Así, los siguientes comandos son equivalentes:

```
SET sort_buffer_size=10000;
SET @@session.sort_buffer_size=10000;
```

Es importante notar que los cambios que haga un usuario sobre una variable que sea global y de sesión no afectarán a las correspondientes variables para usuarios que ya estén conectados. Solamente a las variables de sesión de usuarios que se conecten después del cambio.

Los cambios en cualquier tipo de variable permanecen hasta que la sesión termina o se reinicia el servidor. De modo que si queremos un cambio permanente debemos incluir los valores en ficheros de opciones o como opciones de inicio del servidor.

SET sirve para variables tanto del sistema como de usuario, precedidas por @.

Por ejemplo, *SET GLOBAL key_buffer_size=32* establece el valor de la variable *key_buffer_size* con un valor de 32 *bytes*.

Cuando usamos el comando *SET* para cambiar las variables de sistema los sufijos no pueden usarse, pero el valor puede tomar la forma de una expresión. En el siguiente ejemplo asignamos 10 MB al valor de la variable *sort_buffer_size*.

EJEMPLO 2.11

```
mysql> SET sort_buffer_size = 10 * 1024 * 1024;
```

Para especificar explícitamente si queremos cambiar la variable global o de sesión para este mismo caso usamos la opción *GLOBAL* o *SESSION*:

EJEMPLO 2.12

```
mysql> SET GLOBAL sort_buffer_size = 10 * 1024 * 1024;
mysql> SET SESSION sort_buffer_size = 10 * 1024 * 1024;
```

Sin dicha opción, el comando actualiza la variable de sesión por defecto.

ACTIVIDADES 2.2

- Usa los comandos *SHOW VARIABLES* para conocer el valor de todas las variables y enviar el resultado a un fichero. Repite lo anterior para variables relacionadas con el motor *InnoDB*.
- Haz que uno o más de tus compañeros se conecten a tu servidor. Comprueba quién está conectado usando el comando correspondiente. Intenta desconectarlo con el comando *kill*.
- ¿Cómo sabemos si una variable es o no dinámica?
- Explica los posibles atributos de una variable de servidor. Usa como ejemplo la variable de sistema *port*.
- ¿Qué hace la variable uptime? Indica su valor en tu servidor. ¿Es posible modificar su valor con comandos *SET*?

2.5 ESTRUCTURA DEL DICCIONARIO DE DATOS

El diccionario de datos es un componente esencial en cualquier SGBD ya que contiene información (metadatos) sobre los objetos de bases de datos alojadas en nuestro servidor. *Metadatos* son datos acerca de los datos, tales como el nombre de las bases de datos o tabla, el tipo de datos de una columna o permisos de acceso. Otros términos que a veces se usan para esta información son el diccionario de datos o el catálogo del sistema.

En la mayoría de SGBD esta información se almacena en una base de datos. Para el caso de MySQL, dicha base de datos que se crea por defecto en la instalación, que se llama *information_schema*. Por ejemplo, podemos consultar la información sobre tablas de la base de datos db5 con el siguiente *select*:

EJEMPLO 2.13

```
mysql> SELECT table_name, table_type, engine
    -> FROM information_schema.tables
    -> WHERE table_schema = 'db5'
    -> ORDER BY table_name DESC;
+------------+------------+--------+
| table_name | table_type | engine |
+------------+------------+--------+
| v56        | VIEW       | NULL   |
| v3         | VIEW       | NULL   |
| v2         | VIEW       | NULL   |
| v          | VIEW       | NULL   |
| tables     | BASE TABLE | MyISAM |
|.....|
| t2         | BASE TABLE | MyISAM |
| t          | BASE TABLE | MyISAM |
| pk         | BASE TABLE | InnoDB |
|.....|
+------------+------------+--------+
17 rows in set (0.01 sec)
```

El comando pide una lista de todas las tablas en la base de datos *db5*, en orden alfabético inverso, mostrando tres informaciones: el nombre de la tabla, su tipo y su motor.

INFORMATION_SCHEMA es la base de datos de información que almacena información acerca de todas las otras bases de datos que mantiene el servidor MySQL. Dentro de *INFORMATION_SCHEMA* hay varias tablas de solo lectura. En realidad son vistas, no tablas, así que no podrás ver ningún fichero asociado con ellas.

Cada usuario MySQL tiene derecho a acceder a estas tablas, pero solo a los registros que se corresponden a los objetos a los que tiene permiso de acceso.

La única forma de acceder al contenido de sus tablas es con *SELECT*. No puede insertar, actualizar o borrar su contenido.

No hay diferencia entre el requerimiento de permisos para *SHOW* y para *SELECT*. En cada caso, debe tener algún permiso de un objeto para consultar información acerca del mismo.

2.5.1 LAS TABLAS DE *INFORMATION_SCHEMA*

A continuación, se describen algunas de las tablas más relevantes del diccionario de datos. Para más detalles se recomienda consultar la documentación oficial.

- *SCHEMADATA*

  Proporciona información acerca de las bases de datos o esquemas en nuestro servidor.

Tabla 2.3 Estructura tabla schemadata

Standard Name	SHOW name
CATALOG_NAME	
SCHEMA_NAME	
DEFAULT_CHARACTER_SET_NAME	
DEFAULT_COLLATION_NAME	
SQL_PATH	

- *TABLES*

  Proporciona información acerca de las tablas en las bases de datos.

Tabla 2.4 Estructura tabla tables

Standard Name	SHOW name
TABLE_CATALOG	
TABLE_SCHEMA	Table_...
TABLE_NAME	Table_...
TABLE_TYPE	
ENGINE	Engine
VERSION	Version
ROW_FORMAT	Row_format
TABLE_ROWS	Rows
AVG_ROW_LENGTH	Avg_row_length
DATA_LENGTH	Data_length
MAX_DATA_LENGTH	Max_data_length
INDEX_LENGTH	Index_length
DATA_FREE	Data_free
AUTO_INCREMENT	Auto_increment
CREATE_TIME	Create_time
UPDATE_TIME	Update_time
CHECK_TIME	Check_time
TABLE_COLLATION	Collation
CHECKSUM	Checksum
CREATE_OPTIONS	Create_options
TABLE_COMMENT	Comment

- *COLUMNS*

  Proporciona información acerca de columnas en tablas.

- *USER_PRIVILEGES*

  Proporciona información acerca de permisos globales. Esta información procede de la tabla de permisos *mysql.user*.

  Esta tabla no es estándar. Toma sus valores de la tabla *mysql.user*.

- *SCHEMA_PRIVILEGES*

  Proporciona información acerca del esquema de permisos (base de datos). Esta información procede de la tabla de permisos *mysql.db*.

- *TABLE_PRIVILEGES*

  Proporciona información de permisos de tablas. Esta información procede de la tabla de permisos *mysql.tables_priv*.

El campo *PRIVILEGE_TYPE* puede contener uno (y solo uno) de estos valores: *SELECT*, *INSERT*, *UPDATE*, *REFERENCES*, *ALTER*, *INDEX*, *DROP*, *CREATE VIEW*.

- *COLUMN_PRIVILEGES*

  Proporciona información acerca de permisos de columnas. Esta información procede de la tabla de permisos *mysql.columns_priv*.

- *CHARACTER_SETS*

  Proporciona información acerca de los conjuntos de caracteres disponibles.

- *TABLE_CONSTRAINTS*

  Describe qué tablas tienen restricciones. El valor *CONSTRAINT_TYPE* puede ser *UNIQUE*, *PRIMARY KEY o FOREIGN KEY*. La columna *CONSTRAINT_TYPE* puede contener uno de estos valores: *UNIQUE, PRIMARY KEY*, *FOREIGN KEY*, *CHECK*. Esta es una columna *CHAR* (no *ENUM*).

- *KEY_COLUMN_USAGE*

  Describe qué columnas clave tienen restricciones.

  Si la restricción es una clave ajena, entonces esta es la columna de la clave foránea, no la columna a la que la clave foránea hace referencia.

  Por ejemplo, suponiendo dos tablas llamadas *t1* y *t3* con las siguientes definiciones:

EJEMPLO 2.14

```
CREATE TABLE t1(s1 INT,s2 INT,s3 INT,PRIMARY KEY(s3))ENGINE=InnoDB;

CREATE TABLE t3(s1 INT,s2 INT,s3 INT,KEY(s1),CONSTRAINT CO FOREIGN KEY (s2)
REFERENCES t1(s3)) ENGINE=InnoDB;
```

Para estas dos tablas, la tabla *KEY_COLUMN_USAGE* tendría dos registros:

Un registro con *CONSTRAINT_NAME='PRIMARY'*, *TABLE_NAME='t1'*, *COLUMN_NAME='s3'*, *ORDINAL_POSITION=1*, *POSITION_IN_UNIQUE_CONSTRAINT=NULL*.

Un registro con *CONSTRAINT_NAME='CO'*, *TABLE_NAME='t3'*, *COLUMN_NAME='s2'*, *ORDINAL_POSITION=1*, *POSITION_IN_UNIQUE_CONSTRAINT=1*.

- *ROUTINES*

  Proporciona información acerca de rutinas almacenadas (procedimientos y funciones). No incluye funciones definidas por el usuario (UDFs) de momento.

- *VIEWS*

  Proporciona información acerca de las vistas en las bases de datos.

  La columna *VIEW_DEFINITION* tiene la mayoría de lo que ve en el campo *CREATE TABLE* que produce *SHOW CREATE VIEW*. Ignora las palabras antes de *SELECT* y tras *WITH CHECK OPTION*. Por ejemplo, si el comando original era:

EJEMPLO 2.15

```
CREATE VIEW v AS
SELECT s2,s1 FROM t
WHERE s1 > 5
ORDER BY s1
WITH CHECK OPTION;
```

Entonces la definición de la vista es:

```
SELECT s2,s1 FROM t WHERE s1 > 5 ORDER BY s1
```

La columna *CHECK_OPTION* siempre tiene un valor de *NONE*.

La columna *IS_UPDATABLE* es *YES* si la vista es actualizable, NO si la vista no es actualizable.

- *TRIGGERS*

Proporciona información acerca de disparadores.

Debes tener el permiso *SUPER* para ver esta tabla.

Las columnas *TRIGGER_SCHEMA y TRIGGER_NAME* contienen el nombre de la base de datos en que se produce el disparador y el nombre del disparador, respectivamente.

La columna *EVENT_MANIPULATION* contiene uno de los valores *'INSERT'*, *'DELETE'* o *'UPDATE'*.

Cada disparador se asocia exactamente con una tabla. Las columnas *EVENT_OBJECT_SCHEMA* y *EVENT_OBJECT_TABLE* contienen la base de datos en que ocurre esta tabla, y el nombre de la tabla.

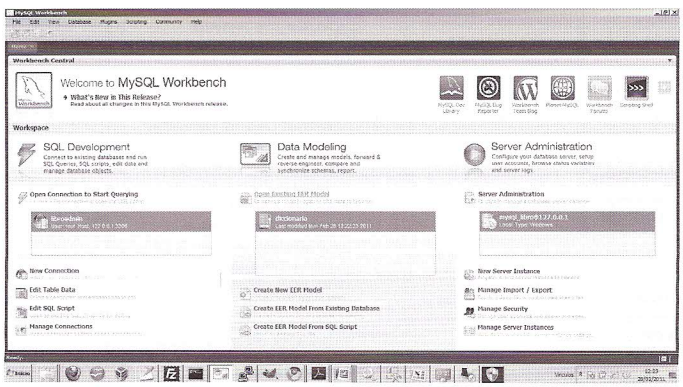
La columna *ACTION_STATEMENT* contiene el comando a ejecutarse cuando el disparador se invoca. Esto es lo mismo que el texto mostrado en la columna *Statement* de la salida de *SHOW TRIGGERS*.

La columna *ACTION_ORIENTATION* siempre contiene el valor *'ROW'*.

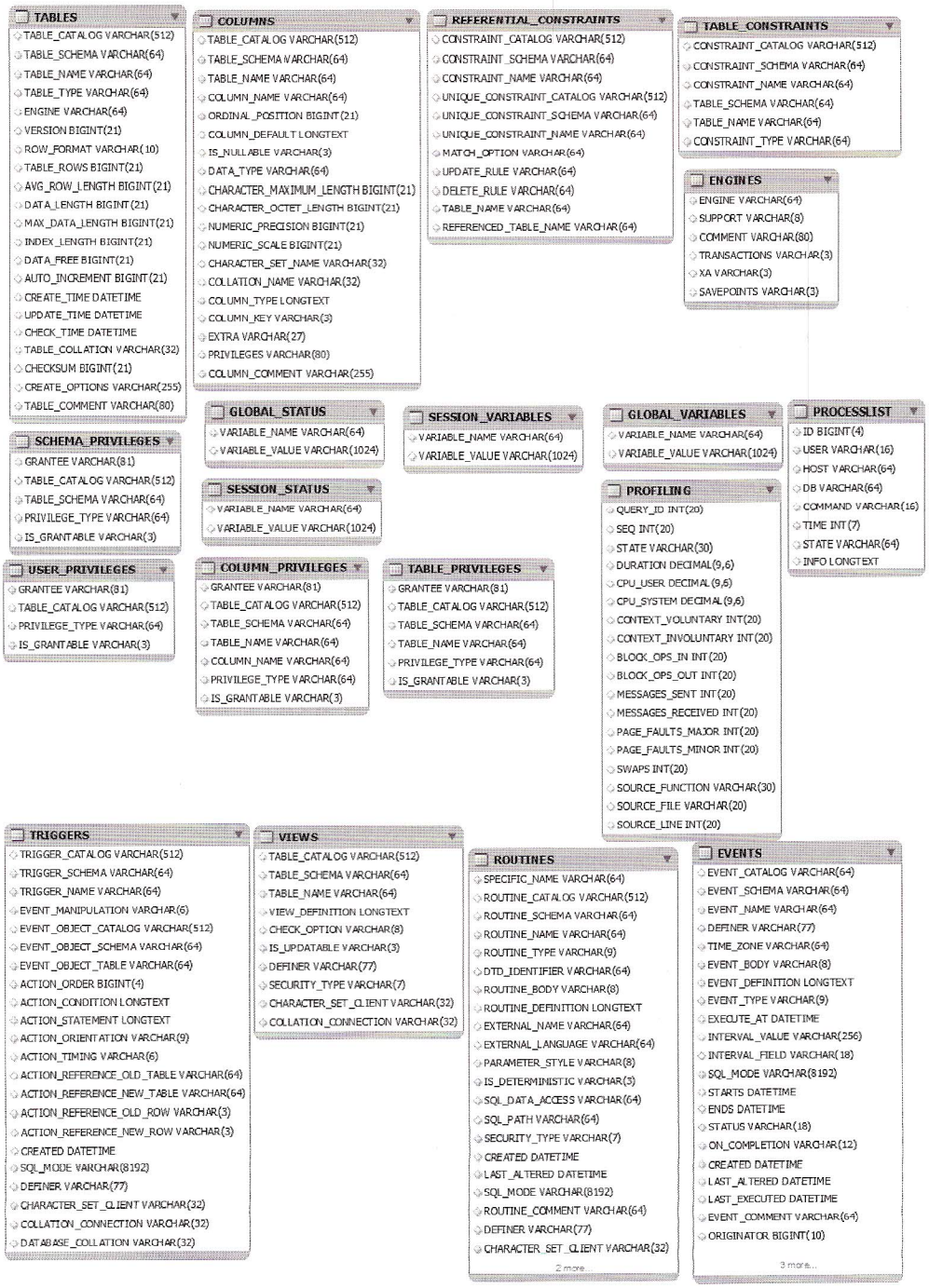
La columna *ACTION_TIMING* contiene uno de los dos valores *'BEFORE'* o *'AFTER'*.

Las columnas *action_reference_old_row* y *action_reference_new_row* contienen el antiguo y nuevo identificador de columna respectivamente lo que significa que *ACTION_REFERENCE_OLD_ROW* siempre contiene el valor *'OLD'* y *ACTION_REFERENCE_NEW_ROW* siempre contiene el valor *'NEW'*.

Podemos ver y obtener el modelo correspondiente a la base de datos comentada usando la opción "*Create EER model from existing database*" de la herramienta MySQL *Workbench*, como se ve en la siguiente imagen:



Estas son las tablas principales de la base *information_schema obtenidas* usando *Workbench*:



2.6 FICHEROS LOG

Los ficheros de registro (*log*) de MySQL:

MySQL tiene varios archivos de registro diferentes que pueden ayudarnos a encontrar lo que está ocurriendo en servidor:

Tabla 2.5 Archivos de registro en MySQL

Archivo de registro	Tipo de información registrado en el archivo
El registro de error	Registra problemas encontrados iniciando, ejecutando o parando mysqld.
El registro de consultas	Registra las conexiones de clientes establecidas y las sentencias ejecutadas.
El registro binario	Registra todas las sentencias que cambian datos. También utilizado para replicación.
El registro de lentitud	Registra todas las sentencias que tardarón más de *long_query_time* segundos en ejecutarse,o no utilizaron índices.

Por defecto, todos los registros son creados en el directorio de datos de *mysqld*. Se puede forzar a *mysqld* a que cierre y reabra los archivos de registro (o en algunos casos, cambiar a un nuevo registro) mediante el volcado de registros. Este volcado de registros ocurre cuando ejecuta la sentencia *FLUSH LOG*S o el comando *mysqladmin flush-logs*.

2.6.1 EL REGISTRO DE ERRORES (*ERROR LOG*)

El archivo de registro de errores contiene información que indica cuando se ha iniciado y parado *mysqld* y también si ha ocurrido algún error crítico mientras el servidor se estaba ejecutando.

En MySQL podemos especificar dónde queremos almacenar el registro de errores con la opción *--log-error*[=*file_name*]. Si no se proporciona ningún valor para *file_name*, *mysqld* utiliza el nombre *host_name.err* y escribe el archivo en el directorio de datos. Si ejecuta *FLUSH LOGS*, el registro de errores es renombrado con el sufijo *-old* y *mysqld* crea un nuevo archivo de registro.

Si no especificas *--log-error*, o (en Windows) no utiliza la opción *--Console*, los errores se escriben en *stderr*, la salida estándar de errores.

En Windows, la salida de errores es siempre escrita al archivo *.err* si no se especifica la opción *--console*.

ACTIVIDADES 2.3

- Indica en *my.ini* al servidor que registre los errores en el fichero de error *server_error*. Reinicia el servidor manualmente sin la opción *–console* y comprueba los mensajes editando dicho fichero.
- Detén el servidor abruptamente (con *ctrl.-z*) y comprueba cómo se ha modificado el fichero anterior.
- Prueba la función *perror* incluida en directorio *bin*. ¿Cuál es su objeto?

2.6.2 EL REGISTRO GENERAL DE CONSULTAS

Si queremos registrar los sucesos en nuestro servidor, debemos iniciarlo con la opcion *--general_log* con valor 1. Si no se da un nombre de fichero con la opción *--general_log_file*, por defecto se usa *hostname.log* siendo hostname el nombre de nuestro equipo. Esto registra todas las conexiones y sentencias a un archivo. Este registro puede ser muy útil cuando sospechemos que hay un error en un cliente y queramos saber exactamente qué hemos enviado al servidor.

El servidor *mysqld* escribe las sentencias al registro de consultas en el orden en que las recibe. Este orden puede ser diferente del de ejecución. Esto es aquí diferente que en el registro de actualizaciones o el registro binario, que son escritos tras la ejecución de la sentencia, pero antes de que se libere cualquier bloqueo (El registro de consultas también contiene todas las sentencias, mientras que el registro binario no contiene sentencias que solo seleccionen datos).

Los reinicios del servidor y volcado de registros no provocan que se genere un nuevo archivo de registro de consultas general (aunque el volcado lo cierra y lo reabre).

ACTIVIDADES 2.4

- Configura MySQL para registrar consultas generales en el fichero *glog*. Comprueba el funcionamiento haciendo que un compañero se conecte a tu servidor y ejecute varias consultas.
- Averigua viendo el fichero *general log* la hora en que se conectó tu compañero y ejecutó la/s consulta/s anterior/es.
- Accede al servidor a través del Workbench. ¿Qué se registra en el *log* general? ¿Qué diferencia a *Workbench* de *mysql.exe*?

2.6.3 EL REGISTRO BINARIO (*BINARY LOG*)

El registro binario contiene toda la información que está disponible en el registro de actualizaciones, en un formato más eficiente y de una manera que es segura para las transacciones. Registra todas las sentencias que han actualizado datos o podrían haberlo hecho (por ejemplo, un *DELETE* que no encontró filas concordantes). Las sentencias se almacenan en la forma de "eventos" que describen las modificaciones.

El registro binario también contiene información sobre cuánto ha tardado cada sentencia que actualizó la base de datos. No contiene sentencias que no hayan modificado datos. Si queremos registrar todas las sentencias (por ejemplo, para identificar una sentencia problemática) deberíamos utilizar el registro general de consultas.

El propósito principal del registro binario es el de actualizar la base de datos durante una operación de recuperación tan completamente como sea posible, porque el registro binario contiene todas las actualizaciones hechas tras la copia de seguridad.

Ejecutar el servidor con el registro binario activado hace que el rendimiento baje en torno a un 1%. Aún así, los beneficios del registro binario para las operaciones de restauración y el hecho de permitirnos poder establecer replicación generalmente son superiores a este decremento de rendimiento.

Cuando se ha iniciado con la opción *--log-bin[=file_name] mysqld* escribe un archivo de registro que contiene todos los comandos SQL que actualizan datos. Si no se da ningún valor para *file_name*, el valor por defecto es el nombre de la máquina *host* seguido por *-bin*. Si se da el nombre del archivo, pero ninguna ruta, el archivo se escribe en el directorio de datos.

Si se proporciona una extensión en el nombre del registro (por ejemplo, *--log-bin=file_name.extension*), la extensión se ignora y elimina sin aviso.

mysqld agrega una extensión numérica a el nombre del registro binario. Este número se incrementa cada vez que se inicia el servidor o se vuelcan los registros. También se crea un nuevo registro binario cuando el actual llega al tamaño especificado en *max_binlog_size*.

Para poder averiguar qué archivos de registro binario diferentes han sido utilizados, *mysqld* también crea un archivo de índice de los registros binarios que contiene los nombres de todos los archivos de registro binario utilizados. Por defecto, éste tiene el mismo nombre que el archivo de registro binario, con la extensión *'.index'*. Podemos cambiar el nombre del archivo del índice con la opción *--log-bin-index[=file_name]*. No deberíamos editar este archivo manualmente mientras *mysqld* se está ejecutando; hacerlo podría confundir a *mysqld*.

Podemos borrar todos los archivos de registro binario con la sentencia *RESET MASTER*, o solo algunos de ellos, hasta uno dado con *PURGE MASTER LOGS TO 'fichero de logs'*. O hasta una fecha dada con *PURGE MASTER LOGS BEFORE 'fecha-hora'*.

Un cliente con el privilegio *SUPER* puede desactivar el registro binario de sus propias sentencias utilizando una sentencia *SET SQL_LOG_BIN=0*.

Se puede examinar el archivo de registro binario con la utilidad *mysqlbinlog*. Esto podría ser útil cuando queramos reprocesar sentencias almacenadas en el registro. Por ejemplo, podemos actualizar un servidor MySQL desde el registro binario de la siguiente manera:

```
C:\> mysqlbinlog log-file | MySQL-h nombre_servidor
```

El registro binario se hace inmediatamente después de que se complete una consulta, pero antes de que se libere cualquier bloqueo o se haga ningún *commit* (es el comando *sql* que hace que se ejecute una transacción o conjunto de operaciones). Esto asegura que el registro está almacenado en el orden de ejecución.

Las actualizaciones a las tablas no-transaccionales se almacenan en el registro binario inmediatamente después de su ejecución. Para las tablas transaccionales como las tablas *BDB* o *InnoDB*, todas las actualizaciones (*UPDATE*, *DELETE* o *INSERT*) que cambian alguna tabla son almacenadas en caché hasta que se recibe una sentencia *COMMIT* en el servidor (siempre que la variable *autocommit* esté puesta a 0 o *FALSE*. En caso contrario cada sentencia se considera una transacción que se confirma automáticamente). En ese momento *mysqld* escribe la transacción completa al registro binario antes de que se ejecute confirme (*COMMIT*). Cuando el flujo de ejecución que gestiona la transacción comienza, reserva un *buffer* de tamaño *binlog_cache_size* para almacenar consultas. Si se requiere más espacio, el flujo abre un archivo temporal para almacenar la transacción. El archivo temporal se borra cuando acaba el flujo.

ACTIVIDADES 2.5

- Activa el registro binario en tu servidor y comprueba su funcionamiento haciendo varias inserciones o modificaciones en cualquiera de las tablas de las bases de datos.
- Usa el programa *mysqlbinlog* para visualizar los cambios efectuados en *ebanca* y para volcar todos los cambios realizados a partir de las 12:01 del 10 del 10 de 2011 en un fichero.
- Indica el efecto de reiniciar el servidor y del comando *FLUSH LOGS* sobre los *logs* de registro binario. Elimina todos los registros binarios menos el último con el comando *PURGE BINARY LOGS*.

2.6.4 EL REGISTRO DE CONSULTAS LENTAS (*SLOW QUERY LOG*)

Cuando se inicia con la opción *--slow-query-log=1*, *mysqld* escribe un archivo de registro que contiene todos las sentencias SQL que llevaron más de *long_query_time* segundos para ejecutarse completamente. Para ello, el registro de consultas lentas debe activarse con la opción *--slow-query-log* al valor ON ó 1. El tiempo para adquirir los bloqueos de tabla iniciales no se cuenta como tiempo de ejecución.

Si no se da ningún valor a *--slow_query_log_file[=file_name]*, el nombre por defecto es el nombre de la máquina *host* con el sufijo *-slow.log*. Si se da un nombre de archivo, pero no como ruta absoluta, el archivo se escribe en el directorio de datos.

Una sentencia se registra en el registro de consultas lentas después de que haya sido ejecutada y todos los bloqueos liberados. El orden de registro puede diferir del de ejecución.

El registro de consultas lentas se puede utilizar para encontrar consultas que tomen excesivo tiempo y sean por tanto candidatos a optimización. De cualquier modo, examinar un registro de consultas lentas puede convertirse en una tarea difícil. Para hacerlo más simple, puede procesar el registro de consultas lentas utilizando el comando *mysqldumpslow* que le ofrecerá un resumen de las sentencias que aparecen en el registro.

ACTIVIDADES 2.6

- Indica las variables de tu servidor relacionadas con los *logs* de consultas lentas.
- Activa el registro de consultas lentas con la opción *slow-query-log*. Compruébalo con el comando *SHOW* correspondiente. Pon el tiempo mínimo a 1 segundo.
- ¿Cómo podrías diseñar una consulta lenta en tu servidor?

2.6.5 MANTENIMIENTO DE FICHEROS DE REGISTRO (*LOG*)

Se deben limpiar estos archivos regularmente para asegurarse de que no ocupan demasiado espacio.

Cuando se utiliza MySQL con el registro activado, deberíamos hacer copias de seguridad de los registros viejos de vez en cuando y eliminarlos y decirle a MySQL que comience a registrar en archivos nuevos.

Podemos forzar al servidor para que comience a utilizar archivos de registro nuevos usando *mysqladmin flush-logs* o con la sentencia *SQL FLUSH LOGS*.

Existen soluciones basadas en *Shell script* (para sistemas Linux) o lenguajes como *Perl* que combinados con el programador de tareas (*cron* para Linux) permiten automatizar este tipo de tareas, algo muy conveniente sobre todo para los registros, dada la gran cantidad de información que generan y lo rápido que pueden llegar a crecer.

No obstante no debe descartarse almacenar los *logs* de varios meses e incluso años dado que pueden ser una gran fuente de información diversa acerca del funcionamiento de nuestro servidor, el tipo de consultas, el volumen, etc., lo que es de gran ayuda para su optimización.

2.6.6 REGISTRO EN *INNODB*

Lo explicado en las anteriores secciones se aplica a todos los motores de almacenamiento sin embargo el tipo *InnoDB* incorpora además un tipo especial de registro donde almacena las operaciones sobre tablas *InnoDB*.

Para configurarlo se usan principalmente tres variables:

-innodb_log_files_in_group: número de ficheros a rotar. Útil para el mantenimiento. Por defecto su valor es 2.

-innodb_log_file_size: tamaño de los ficheros.

-innodb_log_buffer_size: *buffer* usado para el registro antes del volcado a disco de los mismos.

ACTIVIDADES 2.7

- Explica métodos que se te ocurran para gestionar los *logs* generados por el servidor. Indica programas relacionados para Windows y Linux. ¿Qué variable de servidor está directamente relacionada con el tiempo de vida de los ficheros de registro?
- Indica ventajas e inconvenientes del uso de cada tipo de *log*. Piensa en el tipo de aplicación (orientada a escrituras, a lecturas, etc.).

2.7 CASO BASE

Se pretende usar un servidor dedicado para la base de datos *mediaserver* que proporciona servicios de *streaming* de audio y vídeo para clientes remotos los cuales pueden ser registrados o no.

En la web encontrarás el código para recrear la base de datos además de la descripción de la aplicación y sus requisitos en cuanto a datos. Este es un ejemplo real y cercano al alumno. No obstante es perfectamente aplicable a otras aplicaciones conocidas como Youtube o Twiter, siempre desde luego usando versiones simplificadas.

Diseño y cálculos iniciales

En esta sección debemos determinar lo siguiente:

- -Hardware necesario (detalles de procesador, memoria RAM y almacenamiento).
- -Diseño particionado físico.
- -Sistemas operativos que instalaremos. Ubicación y espacio necesario.
- -Versión de MySQL a instalar.
- -Estimación inicial del tamaño de la base de datos y el ritmo previsto de crecimiento.
- -Estima la frecuencia de consultas, inserciones y modificaciones.
- -Según las operaciones de datos que preveas, calcula el espacio necesario para los ficheros *log* de cada tipo (general, errores, binario, consultas lentas y de *innodb*, en caso de haberlo).

Configuración

- Configura las variables más importantes comentadas en el capítulo acorde con la previsión realizada.

RESUMEN DEL CAPÍTULO

En este capítulo se ha tratado de introducir al alumno en los conceptos principales de la administración de un SGBD, desde su instalación y configuración básica inicial hasta su mantenimiento y monitorización con el uso de variables de configuración y ficheros de registro pasando por la descripción del diccionario de datos propio de MySQL.

En los siguientes capítulos profundizaremos en otras tareas de administración más complejas, las cuales hacen uso de lo explicado aquí, especialmente en la parte de variables del servidor.

EJERCICIOS PROPUESTOS

- **1.** Vuelca en un fichero la ayuda del comando *mysqld*.
- **2.** Dentro del fichero *my.ini*, ¿para qué sirven las secciones *mysqld* y *mysql*?
- **3.** ¿De qué manera piensas que podemos hacer que el servidor funcione sin que nadie pueda acceder a él de manera remota? ¿Y sin usar resolución de nombres de dominio?
- **4.** Configura los registros de errores binarios y comprueba qué ocurre en el registro de errores apagando e iniciando el servidor de manera incorrecta, por ejemplo, usando *taskkill* o apagándolo desde el administrador de tareas de Windows.
- **5.** Crea una tabla *t1* en la base test con un campo numérico y de tipo *InnoDB*. Modifícala para que sea *MyISAM*. ¿Qué cambios observas en el sistema de ficheros en el directorio de datos de MySQL?
- **6.** Averigua el tamaño máximo de los archivos de registro binario y en qué variable se configura. Configura dicha variable para un tamaño máximo de 5 KB y comprueba su funcionamiento después de algunas inserciones en la tabla *test.t1*.
- **7.** ¿Qué dos aspectos debemos tener en cuenta en el servidor para permitir el acceso remoto? Haz que uno o más compañeros se conecten a tu servidor. Indica el comando para visualizarlas usando el programa *mysqladmin*.
- **8.** Usa tu máquina virtual con Linux para comprobar el contenido del directorio *bin* de MySQL. Indica al menos tres programas adicionales que incluya, con respecto a la instalación de Windows y coméntalos brevemente usando la ayuda de *man*.
- **9.** Averigua el significado del concepto de replicación en el contexto de bases de datos. ¿Qué relación crees que tiene la replicación con los registros binarios?
- **10.** Has perdido todos tus datos aunque posees una copia de seguridad completa (un fichero *sql* generado con el programa *mysqldump*) de hace 48 horas. Dispones también de un registro binario de esas 48 horas. ¿Cómo retornas el servidor a su estado exacto en el momento de la pérdida de datos?

TEST DE CONOCIMIENTOS

1 ¿Qué se entiende por SQL?
- a) Un lenguaje para la gestión de bases de datos.
- b) Un estándar de SGBD.
- c) Un gestor de bases de datos.
- d) Un lenguaje de programación.

2 El diccionario de datos en un SGBD es:
- a) Una base de datos donde se guardan los datos de las bases de datos del sistema.
- b) La información sobre los datos almacenados en una base de datos.
- c) Lo que hay en la base de datos *informnation_schema*.
- d) Lo que hay en la base de datos *mysql*.

3 ¿En qué directorio se almacenen los programas de MySQL?
- a) *data*.
- b) *scripts*.
- c) *bin*.
- d) *tables*.

4 ¿Qué hace la opción *console* cuando arrancamos el servidor?
- a) Mostrar la versión.
- b) Volcar por pantalla la salida del comando del servidor.
- c) Reiniciar el servidor.
- d) Activar los permisos en las bases de datos.

5 Si tenemos varias tarjetas de red en nuestro servidor, ¿cómo lo asociamos a una de ellas?
- a) Poniendo en el fichero my.*ini ip=ip-interfaz*.
- b) Usando la opción *bind*.
- c) Usando la opción *bind-address*.
- d) Usando la opción *ipconfig*.

6 La palabra *engine* en MySQL designa:
- a) Un tipo de tabla.
- b) Un motor de almacenamiento.
- c) Un tipo de estructura en la que almacenar datos.
- d) Todas las anteriores.

7 ¿Cuál de los siguientes no es un campo de la tabla *columns* del diccionario de datos?
- a) Type.
- b) *Numeric_scale*.
- c) Extra.
- d) *Table_schema*.

8 El comando *purge*:
- a) Es como *flush* pero más rápido.
- b) Permite reparar tablas.
- c) Elimina datos espurios.
- d) Vacía el registro de actualización de consultas.

3 Gestión de cuentas de usuario y permisos

OBJETIVOS DEL CAPÍTULO

- ✓ Entender lo que es una cuenta de usuario.
- ✓ Entender el sistema de acceso a los recursos del servidor y el sistema de permisos.
- ✓ Aprender los comandos SQL para la gestión de cuentas y permisos.

3.1 EL SISTEMA DE PERMISOS MYSQL

El sistema de privilegios de MySQL se encarga de establecer quién puede conectar al servidor y de asegurar que cada usuario legalmente conectado ejecute solamente las operaciones que tenga permitidas. Como usuario, cuando conectamos a un servidor MySQL, nuestra identidad se determina mediante el equipo desde el que nos conectamos y el nombre de usuario que especifiquemos. Después, cuando efectuamos peticiones, el sistema nos otorga privilegios acorde a nuestra identidad y las operaciones que queramos ejecutar.

Es decir, el servidor trabaja con cuentas (equipo más usuario y contraseña) que tienen ciertos permisos o privilegios. Dichas **cuentas y permisos** se encuentran organizados en la base de datos llamada *mysql* que describiremos en este capítulo.

El acceso al servidor tiene lugar en dos etapas, en la primera se comprueba nuestra identidad y en la segunda nuestros permisos para hacer operaciones sobre objetos del SGBD.

3.1.1 TABLAS DE PERMISOS

El servidor usa las tablas *user*, *db*, y *host* en la base de datos MySQL en ambas etapas de control de acceso. Las columnas más importantes en estas tablas de permisos se muestran a continuación:

Tabla 3.1. Tabla de permisos en MySQLI

Nombre tabla	user	db	host
Alcance columnas	Host	Host	Host
	User	Db	Db
	Password	User	
Columnas privilegios	Select_priv	Select_priv	Select_priv
	Insert_priv	Insert_priv	Insert_priv
	Update_priv	Update_priv	Update_priv
	Delete_priv	Delete_priv	Delete_priv
	Index_priv	Index_priv	Index_priv
	Alter_priv	Alter_priv	Alter_priv
	Create_priv	Create_priv	Create_priv

	Drop_priv	Drop_priv	Drop_priv
	Grant_priv	Grant_priv	Grant_priv
	Create_view_priv	Create_view_priv	Create_view_priv
	Show_view_priv	Show_view_priv	Show_view_priv
	Create_routine_priv	Create_routine_priv	
	Alter_routine_priv	Alter_routine_priv	
	References_priv	References_priv	References_priv
	Reload_priv		
	Shutdown_priv		
	Process_priv		
	File_priv		
	Show_*db*_priv		
	Super_priv		
	Create_tmp_table_priv	Create_tmp_table_priv	Create_tmp_table_priv
	Lock_tables_priv	Lock_tables_priv	Lock_tables_priv
	Execute_priv		
	Repl_slave_priv		
	Repl_client_priv		
Columnas seguridad	ssl_type		
	ssl_cipher		
	x509_issuer		
	x509_subject		
Columnas recursos control	max_questions		
	max_updates		
	max_connections		
	max_*user*_connections		

Durante la segunda etapa de control de acceso, el servidor efectúa una verificación de petición para asegurar que cada cliente tiene suficientes privilegios para cada petición que recibe. Para ello consulta las tablas de permisos *user*, *db*, *host* , *tables_priv* y *columns_priv*. Las tablas *user*, y *host* determinan los permisos en la primera fase de la conexión en que se comprueba si la cuenta puede acceder al servidor además de los permisos globales. Las tablas *db*, *tables_priv* y *columns_priv* proporcionan permisos a nivel de base de datos, tabla y columna respectivamente.

Los permisos de la tabla *user* son globales, es decir se aplican a todas las bases de datos. Si tenemos un permiso a 'Y' en un permiso de dicha tabla podremos hacer la operación, por ejemplo *DROP*, sobre cualquier tabla de cualquier base de datos de nuestro sistema. Si por el contrario está a 'N' pasaremos al siguiente nivel, es decir a la tabla *db* y así sucesivamente con el resto de tablas *tables_priv* y *columns_priv*.

Tabla 3.2 Tablas de permisos MySQL II

Nombre tabla	tables_priv	columns_priv
Alcance de columnas	Host	Host
	Db	Db
	User	User
	Table_name	Table_name
		Column_name
Columnas privilegios	Table_priv	Column_priv
	Column_priv	
Otras columnas	Timestamp	Timestamp
	Grantor	

Cada tabla de permisos contiene columnas de alcance y columnas de privilegios:

Las columnas de alcance determinan el alcance de cada entrada (registro) en las tablas; esto es, el contexto en que el registro se aplica. Por ejemplo, un registro de la tabla *user* con los valores *Host* y *User* de *guara.org* y antonio se usaría para autenticar conexiones hechas al servidor desde el equipo *guara.org* por el cliente antonio. De forma similar, un registro de la tabla *db* con las columnas *Host*, *User*, y *Db* con valores guara.org, antonio y liga se usaría cuando antonio conectase desde el equipo *guara.org* para acceder a la base de datos liga . Las tablas *tables_priv* y *columns_priv* contienen columnas de alcance indicando tablas o combinaciones de tabla/columna sobre las que tiene permiso cierta cuenta.

Las columnas de privilegios indican qué privilegios se otorgan a una cuenta sobre ciertas bases, tablas y/o columnas y otros objetos del sistema gestor. El servidor combina la información de diversas tablas de permisos para tener una descripción completa de los permisos de un usuario. Las reglas usadas para ello se describen más adelante en el capítulo.

En las tablas *user*, *db*, y *host*, cada privilegio se lista en una columna separada que se declara como *ENUM('N','Y') DEFAULT 'N'*. En otras palabras, cada privilegio puede estar desactivado o activado, estando desactivados por defecto.

En las tablas *tables_priv, columns_priv,* y *procs_priv* , las columnas de privilegios se declaran como columnas de tipo *SET*. Los valores en estas columnas pueden contener cualquier combinación de los privilegios que afectan a tablas, columnas y procesos respectivamente, como se ve en la siguiente tabla:

Tabla 3.3 Dominio campos de privilegios

Nombre de tabla	Nombre de columna	Posible conjunto de elementos
tables_priv	Table_priv	'Select', 'Insert', 'Update', 'Delete', 'Create', 'Drop', 'Grant', 'References', 'Index', 'Alter'
tables_priv	Column_priv	'Select', 'Insert', 'Update', 'References'
columns_priv	Column_priv	'Select', 'Insert', 'Update', 'References'
procs_priv	Proc_priv	'Execute', 'Alter Routine', 'Grant'

El servidor usa las tablas de permisos como sigue:

Las columnas de alcance (*user* y *host*) de la tabla *user* determinan si se rechazan o permiten conexiones entrantes. Para conexiones permitidas, cualquier privilegio otorgado en la tabla *user* indica los privilegios globales del usuario (*superusuario*). Estos privilegios se aplican a todas las bases de datos en el servidor.

Las columnas de alcance de la tabla *db* determinan qué usuarios pueden acceder a qué bases de datos desde qué equipo. La columna de privilegios determina qué operaciones se permiten. Un privilegio otorgado a nivel de base de datos se aplica a la base de datos y a todas sus tablas.

La tabla *host* se usa en conjunción con la tabla *db* cuando queremos que un registro de la tabla *db* se aplique a varios equipos. Por ejemplo, si queremos que un usuario sea capaz de usar una base de datos desde varios equipos en nuestra red, podemos dejar el valor *Host* vacío en el registro de usuario de la tabla *db* , luego rellenamos la tabla *host* con un registro para cada uno de estos equipos. Las tablas tables_priv y columns_priv son similares a la tabla *db* , pero son más detalladas: se aplican a nivel de tabla y de columna en lugar de a nivel de base de datos. Un privilegio otorgado a nivel de tabla se aplica a la tabla y a todas sus columnas. Un privilegio otorgado a nivel de columna se aplica solo a la columna especificada.

El servidor *mysqld* lee los contenidos de las tablas de permisos en memoria cuando arranca. Podemos decirle que las vuelva a leer mediante el comando *FLUSH PRIVILEGES* o ejecutando los comandos *mysqladmin flush-privileges* o *mysqladmin reload*.

Para consultar los permisos de una cuenta dada, podemos usar el comando *SHOW GRANTS* . Por ejemplo, para determinar los permisos que se otorgan a una cuenta con valores *Host* y *User* de *pc3.guara.com* y antonio, usamos este comando:

```
mysql> SHOW GRANTS FOR 'antonio'@'pc3.guara.com';
```

3.1.2 PRIVILEGIOS EN MYSQL

La información sobre los privilegios de las cuentas está almacenada en las tablas *user, db, host, tables_priv, columns_priv*, y *procs_priv* de la base de datos mysql.

Los posibles privilegios se muestran en la siguiente tabla, junto al nombre de columna asociado con cada privilegio en las tablas *grant* y el contexto en que el privilegio se aplica.

Tabla 3.4 Privilegios en MySQL

Privilegio	Columna	Contexto
CREATE	Create_priv	bases de datos, tablas o índices
DROP	Drop_priv	bases de datos o tablas
GRANT OPTION	Grant_priv	bases de datos, tablas, o procedimientos almacenados
REFERENCES	References_priv	bases de datos o tables
ALTER	Alter_priv	tablas
DELETE	Delete_priv	tablas
INDEX	Index_priv	tablas
INSERT	Insert_priv	tablas
SELECT	Select_priv	tablas
UPDATE	Update_priv	tablas
CREATE VIEW	Create_view_priv	vistas
SHOW VIEW	Show_view_priv	vistas
ALTER ROUTINE	Alter_routine_priv	procedimientos almacenados
CREATE ROUTINE	Create_routine_priv	procedimientos almacenados
EXECUTE	Execute_priv	procedimientos almacenados
FILE	File_priv	acceso a archivos en la máquina del servidor
CREATE TEMPORARY TABLES	Create_tmp_table_priv	administración del servidor
LOCK TABLES	Lock_tables_priv	administración del servidor
CREATE USER	Create_*user*_priv	administración del servidor
PROCESS	Process_priv	administración del servidor
RELOAD	Reload_priv	administración del servidor
REPLICATION CLIENT	Repl_client_priv	administración del servidor
REPLICATION SLAVE	Repl_slave_priv	administración del servidor
SHOW DATABASES	Show_*db*_priv	administración del servidor
SHUTDOWN	Shutdown_priv	administración del servidor
SUPER	Super_priv	administración del servidor

3.1.3 CONTROL DE ACCESO DETALLADO

A continuación explicamos el proceso de acceso detalladamente desde que accedemos al sistema con las credenciales correspondientes hasta que intentamos ejecutar cualquier tipo de comando *sql*.

3.1.3.1 Control de acceso, nivel 1: Comprobación de la conexión

Cuando intentamos conectar a un servidor MySQL, éste aceptará o rechazará la conexión basándose en nuestra identidad y clave. En caso de que nuestras credenciales sean válidas, el servidor acepta la conexión, y entra en el estado 2 en espera de peticiones.

Nuestra identidad se basa en dos elementos de información:

- ✓ El nombre de máquina cliente (o IP) desde donde nos conectamos
- ✓ Nuestro nombre de usuario MySQL

La comprobación de la identidad se realiza utilizando las tres columnas de la tabla *user* (*Host, User*, y *Password*). El servidor solo acepta la conexión si las columnas *Host* y *User* de alguna de las tablas *user* es coincidente con el nombre de máquina y usuario del cliente, y además el cliente proporciona la clave especificada en ese registro.

Los valores de *Host* en la tabla *user* pueden ser especificados como una IP incluyendo *'localhost'*, como redes con su máscara y pudiendo incluir caracteres comodín como '%' o '_'. Por ejemplo:

```
mysql> GRANT ALL PRIVILEGES ON db.* TO david@'192.58.197.0/255.255.255.0';
```

Permite a David conectarse desde cualquier cliente que tenga un número IP de la red 192.58.197.0

La máscara de red solo puede ser utilizada para decirle al servidor que use 8, 16, 24 o 32 bits para la dirección.

Un valor vacío de *Host* en un registro de la tabla *db* significa que los privilegios de dicho registro deben ser combinados con aquellos que se encuentren en el registro de la tabla *host* que concuerde con el nombre del cliente. Los privilegios se combinan utilizando operaciones *AND* (intersección).

En la columna *User*, los caracteres comodín no están permitidos, pero podemos especificar un valor en blanco, que será válido para cualquier nombre, en tal caso el usuario es considerado anónimo, sin nombre de usuario, no un usuario con el nombre que el cliente especificó realmente.

La columna *Password* puede estar vacía. Esto no es un comodín que permite que cualquier clave sea permitida. Significa que el usuario debe conectarse sin especificar una clave.

Los valores que no están vacíos de *Password* en la tabla *user* representan claves cifradas. MySQL no almacena las claves en forma de texto llano para que cualquiera pueda verlo. En vez de esto, la clave suministrada por un usuario que se está intentando conectar es cifrada (utilizando la función *PASSWORD()*). La clave cifrada se utiliza entonces durante el proceso de conexión en el momento de comprobar si es correcta. (Esto se realiza sin que la clave cifrada viaje nunca sobre la conexión.) Desde el punto de vista de MySQL, la clave cifrada es la clave REAL, así que no debería darse acceso a ella a nadie. En concreto, no de acceso de lectura a las tablas de la base de datos MySQL a usuarios no-administrativos.

Es posible que el nombre del cliente y del usuario de una conexión entrante concuerde con más de un registro en la tabla *user*.

Cuando hay la posibilidad de múltiples concordancias, el servidor debe determinar cuál de ellas utilizar. El problema se resuelve de la siguiente manera:

Siempre que el servidor lee la tabla *user* a memoria, ordena los registros. Cuando un cliente intenta conectar, el servidor mira a través de los registros en el orden establecido.

El servidor utiliza el primer registro que concuerda con el nombre y usuario del cliente.

Para ver como esto ocurre, supongamos que la tabla *user* es como esta:

```
+-----------+----------+-
| Host      | User       |
+-----------+----------+
| %         | admin      |
| %         | antonio    |
| localhost | admin      |
| localhost |            |
+-----------+----------+-
```

Cuando el servidor lee la tabla, ordena las entradas con los valores de *Host* más específicos primero. Los nombres de cliente y números de IP son los más específicos. El comodín '%' significa ``cualquier cliente" y es menos específico. Registros con el mismo valor de *Host* se ordenan con el valor de *User* más específico (un valor de *User* en blanco significa ``cualquier usuario" y es menos específico). En la tabla *user* recién mostrada, el resultado después de ordenar sería el siguiente:

```
+-----------+----------+-
| Host      | User       |
+-----------+----------+-
| localhost | admin      |
| localhost |            |
| %         | antonio    |
| %         | admin      |
+-----------+----------+-
```

Cuando un cliente intenta conectar, el servidor mira los registros ordenados y utiliza la primera concordancia.

Si puede conectar al servidor, pero sus privilegios no son los que espera, probablemente está siendo identificado como algún otro usuario. Para averiguar qué cuenta de usuario utilizó el servidor para identificarle, usamos la función *CURRENT_USER()* que devuelve un valor en formato *usuario@nombre_equipo* que indica los valores de *User* y *Host* del registro concordante en la tabla *user*. Supongamos que antonio conecta y ejecuta la siguiente sentencia:

```
mysql> SELECT CURRENT_USER();

+----------------+
| CURRENT_USER() |
+----------------+
| @localhost     |
+----------------+
```

El resultado mostrado indica que el registro que concuerda en la tabla *user* tiene un valor vacío en la columna *User*. En otras palabras, el servidor trata a Antonio como a un usuario anónimo lo que puede dar lugar a errores.

3.1.3.2 Nivel 2: Verificación de permisos

Como ya hemos señalado, una vez establecida una conexión, el servidor entra en el estado 2 del control de acceso. Por cada petición que viene en la conexión, el servidor determina que operación realizar, y entonces comprueba si la cuenta tiene suficientes privilegios para hacerlo. Aquí es donde las columnas de privilegios de las tablas *grant* entran en juego. Estos privilegios puede venir de cualquiera de las tablas *user*, *db*, *host*, *tables_priv*, o *columns_priv*

La tabla *user* otorga privilegios que se asignan de manera global, y que se aplican sin importar sobre qué base de datos trabajamos. Por ejemplo, si la tabla *user* le otorga el privilegio *DELETE,* usted podrá borrar registros de cualquier tabla en cualquier base de datos en todo el servidor. En otras palabras, los privilegios de la tabla *user* son privilegios de superusuario. Es aconsejable otorgar privilegios en la tabla *user* solo a *superusuarios* tales como administradores de base de datos. Para otros usuarios, debería dejar los privilegios de la tabla *user* con el valor 'N' y otorgar los privilegios únicamente a niveles más específicos. Puede otorgar privilegios para bases de datos, tablas o columnas particulares.

Las tablas *db* y *host* otorgan privilegios específicos para una base de datos.

El servidor lee y ordena las tablas *db* y *host* al mismo tiempo que lee la tabla *user*. El servidor ordena la tabla *db* basándose en el rango de las columnas *Host*, *Db* y *User*, y ordena la tabla *host* basándose en el rango de las columnas *Host* y *Db*. Igual que con la tabla *user*, la ordenación coloca los valores menos específicos en última posición, y cuando el servidor busca correspondencias, utiliza la primera que encuentra. Las columnas *Db*, *Table_name* y *Column_name* no pueden contener caracteres comodín ni estar en blanco en ninguna de las tablas. Los valores en el resto de columnas de estas tablas pueden incluir los comodines habituales '%' y '_'.

El servidor ordena las tablas *tables_priv* y *columns_priv* basándose en las columnas *Host*, *Db*, y *User*. Esto es similar a la ordenación de la tabla *db*, pero más simple, porque únicamente la columna *Host* puede contener comodines.

El proceso de verificación de peticiones se describe aquí.

Para peticiones que requieran privilegios de administrador, como *SHUTDOWN* o *RELOAD*, el servidor comprueba únicamente el registro de la tabla *user* porque es la única tabla que especifica los privilegios administrativos. El acceso se otorga si el registro permita la operación demandada, y es denegado en caso contrario.

Para peticiones sobre bases de datos (*INSERT, UPDATE*, etc.), el servidor primero comprueba los privilegios globales del usuario (*superuser*) mirando el registro de la tabla *user*. Si el registro permite la operación demandada, se otorga el acceso. Si los privilegios globales de la tabla *user* son insuficientes, el servidor determina los privilegios específicos sobre la base de datos comprobando las tablas *db* y *host*:

El servidor busca en la tabla *db* una concordancia en las columnas *Host*, *Db* y *User*. Las columnas *Host* y *User* se hacen concordar con el nombre de *host* y de usuario MySQL. La columna *Db* se hace concordar con la base de datos a la que el usuario quiere acceder. Si no hay ningún registro para *Host* y *User*, se deniega el acceso.

Si hay un registro que concuerda en el registro de la tabla *db* y su columna *Host* no está vacía, ese registro define los privilegios específicos del usuario en la base de datos.

Si la columna *Host* del registro concordante de la tabla *db* se encuentra vacía, significa que la tabla hosts enumera qué hosts pueden tener acceso a la base de datos. En este caso, una comprobación más se realiza en la tabla *host* para encontrar una concordancia en las columnas *Host* y *Db*. Si ningún registro de la tabla *host* concuerda, se deniega el acceso. Si hay una concordancia, los privilegios específicos sobre la base de datos del usuario son calculados como la intersección (¡no unión!) de los privilegios en los registros de las tablas *db* y *host*; es decir, los privilegios que tienen valor 'Y' en ambos registros.

Tras determinar los privilegios específicos de la base de datos otorgados por los registros de las tablas *db* y *host*, el servidor los añade a los privilegios globales otorgados por la tabla *user*. Si el resultado permite la operación demandada, se otorga el acceso. En caso contrario, el servidor comprueba sucesivamente la tabla del usuario y los privilegios de las columnas en las tablas *tables_priv* y *columns_priv*, los añade a los privilegios del usuario, y permite o deniega el acceso basándose en el resultado.

Podemos utilizar la tabla *host* para indicar hosts que no son seguros. Supongamos que tenemos una máquina public.sierra.com que está situada en un lugar público que no consideramos seguro. Podemos permitir el acceso a todos los hosts de nuestra red excepto a esa máquina utilizando registros de la tabla *host* como este:

EJEMPLO 3.1

```
+----------------------+--+-------------------------
| Host                 |Db|                         |
+----------------------+--+-------------------------
| public.sierra.com    |%|(todos privilegios a 'N')    |
| %.sierra.com         |%|(todos privilegios a 'Y')    |
+----------------------+---+------------------------
```

Naturalmente, conviene siempre comprobar nuestras entradas en las tablas grant (por ejemplo, utilizando *SHOW GRANTS*) para estar seguro de que sus privilegios de acceso son realmente los que pensamos que son.

Para consultar los privilegios posibles usamos *SHOW PRIVILEGES*.

3.1.4 CUÁNDO TIENEN EFECTO LOS CAMBIOS DE PRIVILEGIOS

Cuando *mysqld* se inicia, todos los contenidos de las tablas *grant* se cargan en memoria y se hacen efectivas para el control de acceso en ese punto.

Si modificamos las tablas *grant* utilizando *GRANT, REVOKE*, o *SET PASSWORD*,el servidor se da cuenta de estos cambios y recarga las tablas en memoria inmediatamente.

Si modificamos las tablas *grant* directamente utilizando sentencias como *INSERT*, *UPDATE*, o *DELETE*, los cambios no tendrán efecto en la comprobación de privilegios hasta que se reinicie el servidor, o bien se le comunique a éste que debe recargar las tablas. Para recargar las tablas manualmente, ejecutamos la sentencia *FLUSH PRIVILEGES* o los comandos *mysqladmin flush-privileges* o *mysqladmin reload*.

Si modificamos las tablas de permisos directamente (sin usar comandos *GRANT* o *REVOKE*) pero olvidamos recargarlas, sus cambios no tienen efecto hasta que reinicie el servidor.

ACTIVIDADES 3.1

- Revisa la definición de la tabla *user* en tu servidor y explica los permisos *shutdown*, *process* y *file*.
- ¿Cuál es la diferencia entre los tipos *enum* y *set* en *MySQL?*.
- Explica los cuatro tipos de columnas en las tablas *user* y *db*.

- ¿Qué permisos globales se dan por defecto cuando creamos un usuario nuevo con el comando *CREATE USER*?, ¿qué tablas se modifican cuando hacemos esta operación?
- Indica en cada caso el significado de los valores para los campos *user* y *host* de la tabla *user*.

USER	HOST
'%'	''
'%.guara.com'	''
'localhost'	'admin_blog'
'122.233.155.0/24'	'user'

- Supón que tenemos los siguientes valores sobre la tabla *user* para los campos *host* y *user*:

USER	HOST
'u_ebanca'	guara.com
guara.com	''
''	'u_ebanca'

Cuando se conecte el usuario *u_banca*, ¿con qué cuenta lo hará?

- Crea un usuario local llamado *u1* sin contraseña. ¿Qué puede hacer sobre la base de datos test? Crea ahora con el usuario *root* la tabla *test.prueba1* con un campo de tipo *int*. Haz lo necesario para que *u1* pueda insertar registros en dicha tabla. Cambia los permisos dándole a *u1* solo permisos de actualización sobre dicha tabla. Intenta modificar una fila. Haz lo necesario para conseguirlo. Explica.
- ¿Podemos hacer inserciones en la base de *ebanca* si tenemos permisos globales de escritura y sin embargo para dicha base de datos tenemos el permiso *insert* a 'N'?
- Averigua para qué sirve el permiso *SUPER*.

3.2 GESTIÓN DE RECURSOS

Esta sección describe cómo gestionar cuentas y permisos para clientes en tu servidor MySQL. En particular el significado de los nombres de cuenta y contraseñas usados en MySQL, cómo preparar una nueva **cuenta** y borrar una existente y cómo cambiar contraseñas.

En resumen se trata de estudiar las operaciones básicas habituales de consulta, inserción o creación, eliminación y actualización sobre cuentas y permisos.

Podemos crear cuentas MySQL de dos formas:

- Usando comandos ***GRANT***.
- Manipulando las tablas de permisos MySQL directamente.

El método recomendado es usar comandos *GRANT* , ya que son más concisos y menos propensos a errores. Otra opción para crear cuentas es usar uno de los diversos programas proporcionados por terceras partes que ofrecen capacidades para administradores de MySQL. *phpMyAdmin* es uno de ellos.

Los siguientes ejemplos muestran cómo usar el programa cliente *mysql* para añadir nuevos usuarios. Esto significa que para realizar cambios, debemos conectar al servidor como el usuario *root*, y la cuenta *root* debe tener el privilegio *INSERT* para la base de datos *mysql* además del permiso administrativo global *RELOAD*.

En primer lugar, usamos el programa MySQL para conectar al servidor como el usuario *root*:

```
C:\>mysql --user=root
```

Si hemos asignado una contraseña a la cuenta *root,* necesitaremos la opción *--password* o *-p* para este comando SQL y también para los mostrados a continuación en esta sección.

Tras la conexión al servidor como *root*, podemos añadir nuevas cuentas. El siguiente comando usa *GRANT* para inicializar nuevas cuentas:

EJEMPLO 3.2

```
mysql> GRANT ALL PRIVILEGES ON *.* TO 'antonio'@'localhost'
-> IDENTIFIED BY 'sierra' WITH GRANT OPTION;
mysql> GRANT ALL PRIVILEGES ON *.* TO 'antonio'@'%'
-> IDENTIFIED BY 'sierra' WITH GRANT OPTION;
mysql> GRANT RELOAD,PROCESS ON *.* TO 'admin'@'localhost';
```

De este modo se crean las cuentas y usuarios en caso de no existir. (para crear usuarios disponemos de los comandos *CREATE USER y SET PASSWORD* para darle contraseña).

Las dos primeras cuentas son cuentas de *superusuario* con plenos permisos para hacer cualquier cosa. Una cuenta (*'antonio'@'localhost'*) puede usarse solo cuando se conecte desde el equipo local. La otra (*'antonio'@'%'*) puede usarse para conectarse desde cualquier otro equipo. Debe notarse que es necesario tener ambas cuentas para que antonio sea capaz de conectarse desde cualquier sitio como antonio.

Una cuenta tiene un nombre de usuario de *admin* y no tiene contraseña. Esta cuenta puede usarse solo desde el equipo local. Tiene los privilegios administrativos *RELOAD* y *PROCESS* . Estos permiten al usuario *admin* ejecutar los comandos *mysqladmin reload, mysqladmin refresh,* y *mysqladmin flush-xxx* , así como *mysqladmin processlist*. No se dan permisos para acceder a ninguna base de datos. Podemos añadir tal privilegio posteriormente mediante un comando *GRANT* adicional.

Como alternativa a *GRANT*, podemos crear la misma cuenta directamente mediante comandos *INSERT* sobre las tablas de permisos. En este caso debemos recargar las tablas de permisos usando *FLUSH PRIVILEGES* para que los permisos creados (o modificados con *UPDATE*) tengan efecto inmediato. A continuación vemos un ejemplo:

EJEMPLO 3.3

```
C:\>mySQL--user=root mysql
mysql> INSERT INTO user
-> VALUES('localhost','antonio',PASSWORD('sierra'),
-> 'Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y');
mysql> INSERT INTO user
-> VALUES('%','antonio',PASSWORD('sierra'),
-> 'Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y');
mysql> INSERT INTO user SET Host='localhost',User='admin',
-> Reload_priv='Y', Process_priv='Y';
mysql> FLUSH PRIVILEGES;
```

La razón de usar *FLUSH PRIVILEGES* al crear cuantas con *INSERT* es decir al servidor que vuelva a leer las tablas de permisos. De otro modo, los cambios no se tienen en cuenta hasta que se reinicie el servidor. Con *GRANT, FLUSH PRIVILEGES* no es necesario.

La razón para usar la función *PASSWORD()* con *INSERT* es *encriptar* las contraseñas. El comando *GRANT encripta* la contraseña, así que *PASSWORD()* no es necesario.

El valor 'Y' activa permisos para las cuentas. Para la cuenta *admin* , podemos emplear la sintaxis extendida *INSERT* usando *SET*.

En los siguientes ejemplos creamos tres cuentas y les damos acceso a bases de datos específicas. Cada una de ellas tiene su nombre de usuario y contraseña.

En todos ellos los usuarios creados tienen permisos de consulta o lectura, inserción, modificación y borrado para datos, creación y eliminación de bases de datos y tablas para todas las tablas de la base *motorblog*.

EJEMPLO 3.4

```
mysql> GRANT SELECT,INSERT,UPDATE,DELETE,CREATE,DROP
-> ON motorblog.*
-> TO 'user_mblog'@'localhost'
-> IDENTIFIED BY 'guara';

mysql> GRANT SELECT,INSERT,UPDATE,DELETE,CREATE,DROP
-> ON motorblog.*
-> TO 'user_mblog'@'guara.com';
```

La primera cuenta puede acceder a la base de datos notas, pero solo desde el equipo local mientras que la segunda (sin contraseña) puede acceder la base de datos notas, pero solo desde el equipo *guara.com*.

Si queremos inicializar las cuentas de *user_mblog* sin usar *GRANT*, usamos los comandos *INSERT* modificando las tablas de permisos directamente:

EJEMPLO 3.5

```
mysql> INSERT INTO user (Host,User,Password)
-> VALUES('localhost','user_mblog',PASSWORD('guara'));

mysql> INSERT INTO user (Host,User)
-> VALUES('guara.com',' user_mblog ');

mysql> INSERT INTO db
-> (Host,Db,User,Select_priv,Insert_priv,
-> Update_priv,Delete_priv,Create_priv,Drop_priv)
-> VALUES('localhost','notas','user_mblog',
-> 'Y','Y','Y','Y','Y','Y');

mysql> INSERT INTO db
-> (Host,Db,User,Select_priv,Insert_priv,
-> Update_priv,Delete_priv,Create_priv,Drop_priv)
-> VALUES('guara.com','notas','user_mblog',
-> 'Y','Y','Y','Y','Y','Y');
mysql> FLUSH PRIVILEGES;
```

Los primeros dos comandos *INSERT* añaden registros en la tabla *user* que permiten al usuario *user_mblog* conectar desde los equipos local y *guara.com*, pero no otorga privilegios globales (todos los privilegios se inicializan al valor por defecto 'N'). Los siguientes dos comandos *INSERT* añaden registros en la tabla *db* que otorgan privilegios a *user_mblog* para las bases de datos *motorblog*, pero solo cuando se accede desde los equipos apropiados. Como siempre, cuando modificamos las tablas de permisos directamente, debemos decirle al servidor que las recargue con *FLUSH PRIVILEGES* para que los cambios en los permisos tengan efecto.

Para revocar permisos usamos el comando *REVOKE* con la siguiente sintaxis:

```
REVOKE priv_type [(column_list)] [, priv_type [(column_list)]] ...
    ON [object_type] {tbl_name | * | *.* | db_name.*}
    FROM user [, user] ...
```

Es muy similar a *GRANT* ya que debemos especificar el tipo de privilegio y las columnas o grupos de columnas a las que afecta la revocación, después indicar con la cláusula *ON* la tabla y finalmente con *FROM* la cuenta o cuentas afectadas.

Por ejemplo si queremos eliminar la cuenta del usuario local *admin_mblog*:

EJEMPLO 3.6

```
REVOKE ALL PRIVILEGES ON notas.* FROM amin_mblog@localhost
```

Lo que elimina todos los permisos de *admin_mblog* sobre todas las tablas de la base de datos notas.

Para eliminar una cuenta, usamos el comando *DROP USER nombre_usuario*.

3.2.1 LIMITAR RECURSOS DE CUENTAS

Una forma de limitar los recursos de los servidores MySQL es asignar a la variable de sistema *max_user_connections* un valor distinto de cero. Sin embargo, éste método es estrictamente global, y no está permitido para la administración de cuentas individuales. Además, limita solo el número de conexiones simultáneas hechas usando una sóla cuenta, y no lo que un cliente puede hacer una vez conectado. Ambos tipos de control son interesantes para muchos administradores de MySQL, particularmente aquéllos que trabajan en ISPs.

En MySQL podemos limitar los siguientes recursos de servidor para cuentas individuales:

- El número de consultas que una cuenta puede realizar por hora.
- El número de actualizaciones que una cuenta puede hacer por hora.
- El número de veces que una cuenta puede conectar con el servidor por hora.

Cualquier comando que un cliente puede realizar cuenta en el límite de consultas. Solo los comandos que modifiquen la base de datos o las tablas cuentan en el límite de actualizaciones.

Una cuenta en este contexto es un registro en la tabla *user*. Cada cuenta se identifica unívocamente por los valores de las columnas *User* y *Host*.

Como prerrequisito para usar esta característica, la tabla *user* en la base de datos MySQL debe contener las columnas relacionadas con el recurso. Los límites de recursos se guardan en las columnas *max_questions*, *max_updates*, max_*connections*, y *max_user_connections*.

Para cambiar el límite de recursos con un comando *GRANT* usamos la cláusula *WITH* que nombra cada recurso a ser limitado y un contador por hora indicando el valor límite. Por ejemplo, para crear una nueva cuenta que pueda acceder con todos los permisos (menos *GRANT*) la base de datos *motorblog*, pero solo de forma limitada, utilizaríamos el siguiente comando:

EJEMPLO 3.7

```
mysql> GRANT ALL ON motorblog.* TO 'admin2_mblog'@'localhost'
-> IDENTIFIED BY 'sierra'
-> WITH MAX_QUERIES_PER_HOUR 20
-> MAX_UPDATES_PER_HOUR 10
-> MAX_CONNECTIONS_PER_HOUR 5
-> MAX_USER_CONNECTIONS 2;
```

No todos los tipos de límites necesitan nombrarse en la cláusula *WITH*, pero los nombrados pueden presentarse en cualquier orden. El valor para cada límite por hora debe ser un entero representando el contador por hora. Si el comando *GRANT* no tiene cláusula *WITH*, los límites se inicializan con el valor por defecto de cero (o sea, sin límite). Para *MAX_USER_CONNECTIONS*, el límite es un entero indicando el máximo número de conexiones simultáneas que la cuenta puede hacer en cualquier momento. Si el límite asignado es el valor por defecto de cero, la variable de sistema *max_user_connections* determina el número de conexiones simultáneas para la cuenta.

Para inicializar o cambiar los límites de una cuenta existente, debemos usar el comando *GRANT USAGE* a nivel global (*ON* *.*). El siguiente comando cambia el límite de consultas para *user_mblog* a 100:

EJEMPLO 3.8

```
mysql> GRANT USAGE ON *.* TO ' user_mblog '@'localhost'
-> WITH MAX_QUERIES_PER_HOUR 100;
```

Este comando deja los permisos existentes de la cuenta inalterados y modifica solo los valores cuyo límite se especifica.

Para eliminar un límite existente, ponemos su valor a cero. Por ejemplo, para eliminar el límite de cuántas veces por hora se puede conectar *user_mblog* en local usamos este comando:

EJEMPLO 3.9

```
mysql> GRANT USAGE ON *.* TO 'user_mblog'@'localhost'
-> WITH MAX_CONNECTIONS_PER_HOUR 0;
```

El conteo de recursos se hace por cuenta, no por cliente. Además, si una cuenta tiene un límite de 50 consultas, no puede incrementar el límite a 100 haciendo dos conexiones simultáneas al servidor. Las consultas de ambas conexiones se computan juntas.

El contador actual por hora de uso de recursos puede reiniciarse globalmente para todas las cuentas, o individualmente para una cuenta dada: Para reiniciar los contadores actuales a cero para todas las cuentas disponemos del comando *FLUSH USER_RESOURCES*. Los contadores también pueden reiniciarse recargando las tablas de permisos (por ejemplo, con un comando *FLUSH PRIVILEGES* o *mysqladmin reload*). Los contadores para una cuenta individual pueden ponerse a cero cambiando cualquiera de sus límites. Para hacerlo, use *GRANT USAGE* como se ha descrito anteriormente y especifique un valor límite igual al valor que tiene la cuenta en ese momento.

Los reinicios de contadores no afectan el límite *MAX_USER_CONNECTIONS*.

Todos los contadores empiezan a cero cuando el servidor arranca y sus valores no se guardan al reiniciar.

3.2.2 ASIGNAR CONTRASEÑAS A CUENTAS

Se pueden asignar contraseñas desde la línea de comandos usando el comando *mysqladmin*:

```
C:\> mysqladmin -u nombres_usuario -h equipo password "nuevacontr"
```

La cuenta para la que este comando cambia la contraseña es la que tiene un registro en la tabla *user* que coincida el *user_name* con la columna *User* y un equipo cliente desde el que se conecta en la columna *Host*.

Otra forma de asignar una contraseña en una cuenta es con el comando *SET PASSWORD*, cuya sintaxis es:

```
SET PASSWORD [FOR user] =
    {
        PASSWORD('some password')
      | 'encrypted password'
    }
```

Es decir, indicamos la cuenta y el *password* en texto claro usando la función *password* que lo *encriptara* o directamente *encriptado*.

EJEMPLO 3.10

```
mysql> SET PASSWORD FOR 'antonio'@'%' = PASSWORD('guara');
```

Solo los usuarios tales como *root* con acceso de modificación para la base de datos *mysql*, puede cambiar la contraseña de otro usuario. Si no estamos conectados como usuarios anónimos, podemos cambiar nuestra propia contraseña omitiendo la cláusula *FOR*:

EJEMPLO 3.11

```
mysql> SET PASSWORD = PASSWORD('guara');
```

Podemos usar el comando *GRANT USAGE* globalmente *(ON* *.*) para asignar una contraseña a una cuenta sin afectar los permisos actuales de la cuenta:

EJEMPLO 3.12

```
mysql> GRANT USAGE ON *.* TO 'usuario_liga'@'%' IDENTIFIED BY 'guara';
```

Aunque generalmente es preferible asignar contraseñas usando uno de los métodos precedentes, podemos hacerlo modificando la tabla *user* directamente como en los siguientes ejemplos:

Para establecer una contraseña al crear una nueva cuenta, especificamos un valor para la columna *Password*

EJEMPLO 3.13

```
mysql> INSERT INTO user (Host,User,Password)
-> VALUES('%','antonio',PASSWORD('guara'));
mysql> FLUSH PRIVILEGES;
```

Para cambiar la contraseña en una cuenta existente, usamos *UPDATE* para especificar el valor de la columna *Password*:

EJEMPLO 3.14

```
mysql> UPDATE user SET Password = PASSWORD('falcon')
-> WHERE Host = '%' AND User = 'usuario_liga';
mysql> FLUSH PRIVILEGES;
```

Cuando especifiquemos una contraseña en una cuenta mediante *SET PASSWORD, INSERT, o UPDATE*, debemos usar la función *PASSWORD()* para *encriptarlo*.

Si inicializa la contraseña usando el comando *GRANT . IDENTIFIED BY* o *mysqladmin password* , ambos *encriptan* la contraseña. En estos casos, el uso de la función *PASSWORD()* no es necesario.

ACTIVIDADES 3.2

- Crea un usuario local nuevo llamado *admin* con *CREATE USER*, asígnale la contraseña *admin* encriptándola con la función *PASSWORD* y dale permisos globales sobre todas las bases de datos con *GRANT*. Haz cada operación con un comando distinto.
- Elimina el permiso *SUPER* y *GRANT* del usuario *admin* anterior. Usa las dos formas vistas: con *REVOKE* y de manera manual.
- Haz lo necesario para que el usuario *admin ebanca* pueda conectarse a la base de datos *ebanca* desde una única IP *www.guara.com*.

- ¿Qué ocurre si intentamos crear un usuario con contraseña con el comando *CREATE USER* sin usar la función *PASSWORD*?, ¿podemos conocer la contraseña original de un usuario?
- ¿Con qué cinco permisos deberíamos ser más cautelosos a la hora de concederlos y por qué?
- Averigua para qué sirven las funciones *encode*, *md5* y *password* de MySQL.

3.3 CONEXIONES SEGURAS

El protocolo TLS (*Transport Layer Security*), sucesor de SSL (*Security Socket Layer*) permite incorporar los cuatro elementos de seguridad completa en una conexión, es decir garantiza la confidencialidad (mediante encriptación de los datos que evita que nadie pueda entender los datos) integridad (mediante firma digital que evita que los datos puedan ser modificados sin que el usuario se entere), autenticación (el emisor demuestra sus credenciales) y no repudio de forma el que recibe un mensaje sabe con seguridad quién es el autor, el cual por tanto no puede negarlo.

MySQL incluye soporte para conexiones seguras entre los clientes y el servidor, utilizando el protocolo TLS. Esta sección explica cómo utilizar conexiones TLS entre clientes y servidores MySQL. También explicamos una manera de configurar **SSH** en Windows que permite *entunelar* nuestras conexiones a MySQL para que queden ocultas a través del programa *Putty* que implementa el protocolo SSH.

La configuración de MySQL tiene la misión de ser tan rápida como sea posible, así que no se usan las conexiones cifradas por defecto. Hacerlo, haría que el protocolo cliente/servidor fuese mucho más lento. Cifrar datos es una operación que requiere un uso intensivo de CPU, y por tanto obliga a la máquina a realizar trabajo adicional que retrasa otras tareas del servidor. Para aplicaciones que requieran la seguridad que proveen las conexiones cifradas, el trabajo de computación extra está justificado.

MySQL permite que el cifrado sea activado para conexiones individuales. Podemos escoger entre una conexión normal sin cifrar, o una segura cifrada mediante SSL dependiendo de los requerimientos de las aplicaciones individuales.

3.3.1 CONCEPTOS BÁSICOS DE SSL

Para entender como MySQL utiliza TLS, es necesario explicar algunos conceptos básicos sobre TLS y **X509**. Aquellos ya familiarizados con ellos, pueden saltarse esta parte.

Por defecto, MySQL utiliza conexiones sin cifrar entre el cliente y el servidor. Esto significa que cualquiera con acceso a la red podría ver el tráfico y mirar los datos que están siendo enviados o recibidos. Incluso podría cambiar los datos mientras están aún en tránsito entre el cliente y el servidor. Para mejor la seguridad un poco, podemos comprimir el tráfico entre el cliente y el servidor utilizando la opción *--compress* cuando ejecutemos programas cliente.

Cuando necesitemos mover información sobre una red de una manera segura, una conexión sin cifrar es inaceptable. El cifrado es la manera de hacer que cualquier dato sea ilegible. De hecho, hoy en día la práctica requiere muchos elementos adicionales de seguridad en los algoritmos de cifrado. Deben resistir muchos tipos de ataques conocidos.

El más utilizado es el llamado PKI o infraestructura de clave pública. En este sistema la comunicación se basa en el uso de algoritmos de cifrado asimétricos que tienen dos claves de cifrado para cada parte de la comunicación (una pública, y otra privada). Cualquier dato cifrado con la clave privada puede ser solo descifrado utilizando la clave

privada correspondiente, solo en poder del propietario del certificado. También funciona a la inversa, datos cifrados con la clave privada solo pueden descifrarse con la clave pública. También incorpora sistemas de verificación de identidad, utilizando el estándar X509.

X509 hace posible identificar a alguien en Internet. Para ello algunas empresas o entidades llamadas Autoridades de Certificación o CA asignan certificados electrónicos a cualquiera que los necesita. Un certificado consiste en la clave pública de su propietario junto con otros datos personales todo ello firmado con la clave privada de la autoridad correspondiente.

El propietario de un certificado puede enseñárselo a otra entidad como prueba de su identidad. Si la entidad confía en dicha autoridad y dispone de su clave pública podrá ver la información del certificado y así autenticar al emisor del mensaje además de conocer su clave pública.

3.3.2 REQUISITOS Y VARIABLES SSL

Para utilizar conexiones SSL entre el servidor MySQL y los programas cliente, debemos disponer de claves privadas y públicas para servidores y clientes así como de certificados o de una autoridad que los emita (podemos usar la nuestra propia). Para ello podemos usar el programa OpenSSL para Linux o Windows. Su uso queda fuera del alcance de este libro sin embargo dispone de una web con documentación suficiente para la gestión de claves y certificados.

Para conseguir que las conexiones seguras funcionen con MySQL, debemos hacer lo siguiente:

Para comprobar si un servidor *mysqld* que se está ejecutando tiene soporte para *OpenSSL*, examinamos el valor de la variable de sistema *have_openssl*:

EJEMPLO 3.15

```
mysql> SHOW VARIABLES LIKE 'have_openssl';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| have_openssl  | YES   |
+---------------+-------+
```

Si el valor es *YES*, es que nuestro servidor soporta conexiones *OpenSSL*. En caso contrario debemos usar una versión que sí lo soporte.

Las variables más importantes relacionadas con conexiones seguras son las siguientes (todas ellas son de carácter global):

- *-ssl*

  Permite que el servidor acepte conexiones ssl de un cliente. Requiere de las opciones *ssl-ca*, *ssl-cert* y *ssl-key*.

  Funcionará solo si ambos, servidor y cliente tienen soporte para SSL. En otro caso se establecerá una comunicación normal sin *encriptar*.

  Para asegurarnos de que cierta cuenta se conecta por SSL podemos crearla con *GRANT* usando la opción *REQUIRE SSL*.

- *-ssl-ca=fichero*

  Indica el fichero que contiene autoridades certificadoras de confianza.

- *-ssl-cert=fichero*

  Indica el fichero que contiene el certificado de la autoridad de confianza correspondiente incluyendo la clave pública y datos de autenticación.

- *-ssl-key=fichero*

  El fichero con la clave privada del servidor.

3.3.3 OPCIONES SSL DE GRANT

MySQL puede comprobar los atributos de un certificado *X509* además de la autentificación usual que se basa en nombre de usuario y clave. Para especificar las opciones relacionadas con SSL para una cuenta MySQL, utiliza la cláusula *REQUIRE* de la sentencia *GRANT*.

Hay diferentes maneras de limitar los tipos de conexión para una cuenta:

- Si una cuenta no tiene requerimientos de SSL o *X509*, las conexiones sin cifrar se permiten siempre que el nombre de usuario y la clave sean válidas. De cualquier manera, se pueden también utilizar conexiones cifradas, si el cliente tiene los certificados y archivos de claves apropiados.
- La opción *REQUIRE* SSL limita al servidor para que acepte únicamente conexiones cifradas SSL para la cuenta. Ten en cuenta que esta opción puede pasarse por alto si hay algún registro ACL que permite conexiones no-SSL.

EJEMPLO 3.16

```
mysql> GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost' IDENTIFIED BY
'pelavaras' REQUIRE SSL
```

REQUIRE X509 significa que el cliente debe tener un certificado pero que el certificado exacto, entidad certificadora y sujeto no importan. El único requerimiento es que debería ser posible verificar su firma con uno de los certificados CA.

EJEMPLO 3.17

```
mysql> GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost' IDENTIFIED BY 'pelavaras'
REQUIRE X509;
```

REQUIRE ISSUER 'issuer' coloca una restricción en la conexión mediante la cual el cliente debe presentar un certificado *X509* válido, emitido por la autoridad CA cuyo nombre es *'issuer'*. Si el cliente presenta un certificado que es válido pero tiene un emisor diferente, el servidor rechaza la conexión. La utilización de certificados *X509* siempre implica cifrado, así que la opción SSL no es necesaria.

EJEMPLO 3.18

```
mysql> GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost' IDENTIFIED BY 'pelavaras'
REQUIRE ISSUER '/C=ES/ST=aragon/L=zaragoza/O=MySQL españa AB/CN=barrabes/
Email=barrabes@guara.com';
```

Debe notarse que el valor de *ISSUER* debe introducirse como una única cadena de caracteres.

REQUIRE SUBJECT 'subject' establece la restricción a los intentos de conexión de que el cliente debe presentar un certificado *X509* válido con sujeto 'subject'. Si el cliente presenta un certificado que, aunque válido, tiene un sujeto diferente, el servidor rechaza la conexión.

EJEMPLO 3.19

```
mysql> GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost' IDENTIFIED BY 'pelavaras'
REQUIRE SUBJECT '/C=ES/ST=Aragon/L=Huesca/O=MySQL demo clientcertificate/
CN=Sorinas/Email=sorinas@guara.com';
```

Nótese que el valor de *SUBJECT* debe ser introducido como una única cadena de caracteres.

REQUIRE CIPHER 'cipher' es necesario para asegurar que se utilizan longitudes de cifra y claves suficientemente fuertes. El protocolo SSL por sí mismo puede ser débil si se utilizan viejos algoritmos con claves de cifrado cortas. Utilizando esta opción, podemos pedir un método exacto de cifrado para permitir una conexión.

EJEMPLO 3.20

```
mysql> GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost' IDENTIFIED BY 'pelavaras'
REQUIRE CIPHER 'EDH-RSA-DES-CBC3-SHA';
```

Las opciones *SUBJECT, ISSUER*, y *CIPHER* pueden combinarse en la sentencia *REQUIRE* de la siguiente manera:

EJEMPLO 3.21

```
mysql> GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'
-> IDENTIFIED BY 'pelavaras'
-> REQUIRE SUBJECT '/C=ES/ST=Aragon/L=Huesca/
O=MySQL demo client certificate/
CN=Alberto Car/Email=acar@guara.com'
-> AND ISSUER '/C=ES/ST=Aragon/L=Huesca/
O=MySQL España AB/CN=L Hues/Email=lhues@guara.com'
-> AND CIPHER 'EDH-RSA-DES-CBC3-SHA';
```

Observamos que los valores de *SUBJECT* e *ISSUER* deben ser introducidos cada uno como una única cadena de caracteres.

El orden de las opciones no importa, pero ninguna opción puede ser especificada dos veces.

3.3.4 CONEXIONES SEGURAS A MYSQL

En este apartado vernos dos formas de conexión segura entre cliente y servidor MySQL. En la primera usamos las variables de configuración ya vistas y en la segunda usaremos un método indirecto usando una conexión ssh. Éste último sistema es válido incluso si no disponemos de nuestra infraestructura de clave pública o si nuestra cliente no soporta SSL (aunque desde luego no serviría para una autenticación en los dos sentidos).

3.3.4.1 Conexiones con SSL

Para comprobar las conexiones SSL, iniciamos el servidor de la siguiente manera indicando los ficheros de la autoridad (su clave pública o certificado) así como los correspondientes a las claves privada y pública del servidor:

```
C:\> mysqld --ssl-ca=ca-cert.pem --ssl-cert=server-cert.pem  --ssl-key=server-key.pem
```

En la parte del cliente tenemos varias opciones para conectarnos dependiendo de cómo fue creada con el comando GRANT. Como vernos en la siguiente sección éste incluye la posibilidad de añadir la opción REQUIRE con los valores NONE o SSL *option*. En este ejemplo se conceden todos los permisos sobre la base *test* a *root* con la contraseña *root* y la obligación de conectarse usando SSL.

```
GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'
  IDENTIFIED BY 'root' REQUIRE SSL;
```

REQUIRE NONE indica que la cuenta no requiere opciones SSL o X509, de este modo la comunicación será totalmente insegura.

REQUIRE SSL indica que solo se permiten conexiones seguras para esa cuenta. Para ello el cliente deberá especificar como mínimo el fichero que contiene los certificados de autoridades en que confía con la opción –ssl-ca.

```
C:\> mysql --ssl-ca=ca-cert.pem
```

REQUIRE X509 en este caso el cliente también debe poseer su propio certificado y clave privada que indicará en la ejecución del comando.

```
C:\> mysql --ssl-ca=ca-cert.pem -ssl-key=client-key.pem -ssl-cert=client-cert.pem
```

3.3.4.2 conexión segura por Entunelamiento (Port Forwarding)

En esta sección tratamos una forma alternativa de crear conexiones seguras entre cliente y servidor que sirve no solo para MySQL sino para otros servicios.

Se trata de usar el servicio ssh que permite establecer comunicaciones seguras entre clientes y servidores. La explicación detallada de éste servicio de red queda fuera del alcance de este libro, no obstante su uso está muy extendido y la documentación al respecto es muy abundante.

Conexiones con SSH

SSH (*Secure SHell*) es un protocolo similar a *Telnet* que permite abrir un *shell* en una máquina remota, con la diferencia de que SSH encapsula toda la información que viaja por la red utilizando criptografía de clave pública. Este protocolo se implementa con arquitectura cliente-servidor, por lo que se necesita un servidor SSH en la máquina remota (por defecto, en el puerto 22 TCP) y un cliente SSH que nos permita conectarnos a ese servidor.

Veamos resumidamente cómo funciona una conexión SSH:

- El cliente inicia la conexión con el comando:

```
C:\>ssh usuario@guara
```

- El servidor guara comprueba si el host del cliente tiene permiso para conectar, tras lo cual cliente y servidor intercambian en texto plano sus identificadores de versión, para comprobar si los protocolos soportados coinciden y la comunicación es posible.
- El servidor envía al cliente su clave pública DSA (algoritmo asimétrico) en texto sin cifrar, que el cliente usará para cifrar los mensajes hacia el servidor, haciéndolos solo legibles para el propio servidor.
- Si no es la primera vez que se conecta a ese servidor, el cliente compara la clave pública recibida del servidor con la que tiene almacenada para verificar su autenticidad. Si es la primera vez que se conecta a ese servidor y no la tiene almacenada, el cliente no sabe si la clave pública recibida es realmente la del servidor o la de un impostor que ha interceptado la conexión, lo que se conoce con el nombre de ataque *man in the middle*, por lo que pide confirmación al usuario para aceptarla como válida:

```
1. The authenticity of host guara (192.168.1.3) can't be established.
2. DSA key fingerprint is e9:df:72:2c:eb:1d:bf:b2:3a:38:96:2a:3b:6b
Are you sure you want to continue connecting (yes/no)?
```

Para confirmarlo el usuario debería obtener la huella digital o *fingerprint* por otro medio alternativo y seguro: correo electrónico cifrado, por teléfono, correo postal, en persona, etc. Confirmar la clave sin verificarla adecuadamente implica asumir el riesgo de una posible suplantación. Si confiamos en la clave, le decimos al cliente que continúe con la conexión:

Como hemos confiado en la clave pública recibida, el cliente nos informa de que ésta se ha añadido a un archivo, normalmente llamado *known_hosts*, donde se almacenan los *hosts* conocidos. En adelante cada vez que el cliente se conecte a ese servidor comparará la clave pública recibida al iniciar sesión con la almacenada:

- El cliente genera una clave de sesión (válida solo para la sesión en curso), selecciona un algoritmo de cifrado simétrico y envía un mensaje conteniendo la clave de sesión y el algoritmo simétrico seleccionado, cifrado con la clave pública DSA del servidor. A partir de este momento todo el tráfico entre cliente y servidor viaja cifrado utilizando el algoritmo de cifrado simétrico seleccionado y la clave de sesión.
- Ahora el cliente se debe identificar ante el servidor, cosa que haremos mediante contraseña (también puede hacerse mediante claves DSA, sin necesidad de password). El servidor solicita al usuario la contraseña correspondiente a *usuario*, si el *login* es válido y no hay restricciones adicionales obtendremos un *shell* o terminal remota en el servidor SSH, de modo que podremos usar el sistema como si estuviéramos sentados allí, con los privilegios que tenga ese usuario.

Creacion de un tunel con MySQL a través de ssh

Con *OpenSSH* se puede crear un túnel encriptado entre dos máquinas y enviar a través de él los datos de otros protocolos (es un tipo de VPN, *Virtual Private Network*, Red Privada Virtual). Esto es útil Para *entunelar* protocolos que son inseguros por enviar la información sin cifrar, como Telnet, FTP, HTTP, MySQL, las X, etc.

El único requisito para configurar un túnel SSH es que sea un protocolo basado en TCP.

Vamos a verlo con un ejemplo. Desde el cliente queremos hacer consultas al servidor de base de datos MySQL instalado en el servidor. Pero MySQL no soporta conexiones seguras entre los clientes y el servidor, de manera que la identificación de los usuarios remotos y los datos de las consultas serán visibles a cualquier máquina de la red. Por ello, utilizaremos SSH para establecer una conexión segura, de la siguiente manera:

En el cliente, el cliente MySQL (*mysql*) hace una petición al puerto 3306 TCP de localhost, donde escucha el cliente SSH (*ssh*).

El cliente SSH (*ssh*) escucha el puerto 3306 TCP y redirecciona las peticiones al servidor SSH (*sshd*) a través del túnel, de manera que todo el tráfico de datos entre el cliente y el servidor MySQL van cifrados por la red.

En el servidor, el servidor SSH (*sshd*) redirecciona el tráfico procedente del túnel al puerto 5000 TCP de localhost, donde escucha el servidor MySQL (*mysqld*), de manera que éste ve las conexiones como procedentes de la propia máquina.

Podemos visualizar mejor el proceso con el siguiente esquema:

Veamos cuál es el procedimiento a seguir:

- Configuramos el servidor MySQL para que escuche en el puerto 5000 TCP de localhost. Para ello, añadimos lo siguiente en el archivo de configuración:

EJEMPLO 3.22

```
[mysqld]
#skip-networking
bind-address = 127.0.0.1
port = 5000
```

- Una vez reiniciado el servidor MySQL, solo podremos conectar con la base de datos desde *localhost*, en la máquina local.
- Levantamos el túnel desde el cliente con el siguiente comando (el túnel se establecerá una vez introducido el *password*):

EJEMPLO 3.23

```
C:\> ssh -N -L 3306:127.0.0.1:5000 usuario@guara
```

3306:127.0.0.1:5000: abre un túnel que conecta el puerto 3306 del cliente con el 5000 del servidor, en cuyos extremos están el cliente y el servidor SSH.

La opción N establece el túnel, ya que indica no ejecutar comandos remotos mientras que L indica que este extremo es el cliente del túnel.

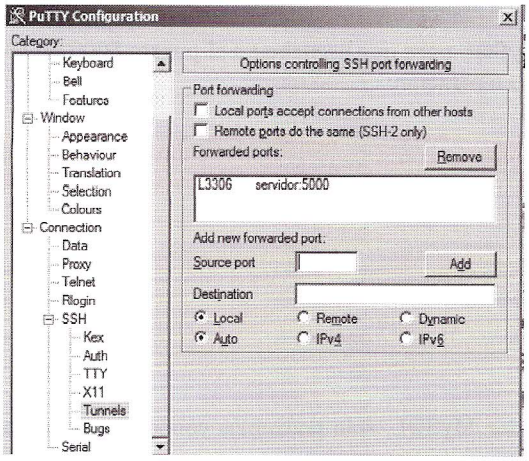
Para conectarnos al servidor MySQL desde el cliente tenemos que conectarnos con el extremo local del túnel ejecutando:

EJEMPLO 3.24

```
C:\>mysql -u root -h 127.0.0.1 -p test
Enter password:
mysql>
```

Si preferimos usar métodos gráficos disponemos de clientes ssh como *Putty* o *SecureCRT* de *http://www.vandyke.com/*. Otra opción es *f-secure* de *http://www.f-secure.com/*.

Para el caso de *Putty* es muy sencillo, solamente debemos configurar la sección de *Tunnel* tal como se muestra en la siguiente figura:



ACTIVIDADES 3.3

- ¿Qué servicio proporciona el protocolo TLS respecto a la seguridad?, ¿qué diferencia hay con SSL?
- Define con tus palabras los siguientes conceptos:
  a. Certificado digital.
  b. Firma digital.
  c. Clave pública.
  d. Infraestructura de clave pública (PKI).
  e. PGP.
  f. CRL.
- Usa algún programa como *Win32 OpenSSL* o *openssh* para generar un par de claves públicas y privadas en tu equipo.
- Crea un túnel para el servicio *mysql* entre tu equipo *Ubuntu* y tu servidor local usando el programa *Putty*. Haz alguna consulta para comprobar cómo el tráfico es encriptado para lo cual puedes usar algún *sniffer* como *wireshark* o *tcpdump*.
- Busca e indica las variables de sistema (de sesión y globales) relacionadas con la seguridad en MySQL (ayúdate del comando SHOW VARIABLES).

3.4 CASO BASE

Diseña la política de usuarios (tipos de usuarios y permisos). Ten en cuenta que algunos usuarios accederán al servidor mediante la aplicación, es decir, es posible que el usuario que accede a la aplicación no sea el que ejecute los comandos SQL sobre la base de datos.

Indica los comandos necesarios para la creación de los usuarios en cada caso.

Determina en qué circunstancias (qué usuarios, desde qué ubicaciones, en qué momentos, etc.) es conveniente el uso de comunicaciones seguras con el servidor.

RESUMEN DEL CAPÍTULO

En este capítulo se han visto las características de MySQL relacionadas con la seguridad.

En primer lugar hemos estudiado el esquema de tablas que usa el servidor para organizar las cuentas de usuario y permisos asi como los distitntos objetos sobre los que se asignan permisos.

Después hemos descrito el funcionamiento del sistema de acceso al servidor por parte de una cuenta y cómo el servidor establece dichos permisos. También hemos visto cómo gestionar usuarios y permisos a través del instrucciones SQL (DCL).

Por último hemos incluido una sección para describir distintas formas de crear y configurar conexiones seguras usando el protocolo TLS, sucesor de SSL.

EJERCICIOS PROPUESTOS

- **1.** Discute la conveniencia de permitir accesos de administrador remoto desde el punto de vista de la seguridad.
- **2.** Crea usuarios y permisos para la base de datos de *ebanca* considerando las restricciones de seguridad siguientes:

  Hay tres usuarios:

  - *Administrador*: todos los permisos.
  - *Operador nivel1*: tiene acceso de lectura en todas las tablas. Puede operar en todas tablas menos en el campo saldo de cuenta y en la tabla movimiento donde solo puede consultar. No puede modificar la estructura de ninguna tabla.
  - *Operador nivel2*: puede consultar todas las tablas y modificar la tabla cliente y cuenta menos los campos *cod_cuenta*, *fecha_creacion* y *saldo*:
  - ¿Puedes recuperar tu contraseña de *root* si la has olvidado?, ¿cómo entrarías a tu servidor en ese caso?
- **3.** Discute la conveniencia de permitir accesos de administrador remoto desde el punto de vista de la seguridad.
- **4.** Busca nombres y webs de autoridades de certificación oficiales en la web tanto en España como en otros países. Usa alguna de ellas para crearte un certificado.
- **5.** ¿Cómo se crean las tablas de permisos en MySQL?, ¿qué cuentas se crean por defecto en Windows y Linux?, ¿te parece seguro? Explícalo.
- **6.** Supón que se están haciendo muchas consultas desde una cuenta a tu base de datos *motorblog* y no tienes medios de aumentar tus recursos físicos (memoria y CPU). ¿De qué manera podrías limitarlos? Responde suponiendo ahora que las consultas se hacen desde distintos equipos.
- **7.** Investiga en qué consiste el llamado *sql injection*. Pon un ejemplo.
- **8.** Crea un túnel directo en tu servidor MySQL usando *ssh* de forma que las conexiones al puerto 4000 en tu equipo local se redirijan al puerto 3306 del servidor MySQL. Úsalo para conectarte usando MySQL Workbench.

TEST DE CONOCIMIENTOS

1 ¿Qué es una cuenta?
- a) Un usuario y una ip.
- b) Lo que te permite acceder a un sistema *Linux*.
- c) Un usuario de una base de datos.
- d) Un permiso especial.

2 SSL es:
- a) Un protocolo de la capa 3 en el modelo *tcp/ip*.
- b) Un sistema de seguridad en red.
- c) Un protocolo de la capa de aplicación.
- d) Un sistema de claves públicas y privadas.

3 *Verisign* se conoce como:
- a) Una empresa de certificados digitales.
- b) Un software que usa SSL.
- c) Una autoridad de de certificación.
- d) Todo lo anterior.

4 El permiso *process* permite:
- a) Ejecutar rutinas almacenadas.
- b) Mostrar los procesos activos del servidor.
- c) Ejecutar *mysqladmin showprocesslist*.
- d) Crear usuarios.

5 ¿Qué conseguimos con el comando *mysqladmin --flush threads*?
- a) Cancelar las conexiones de lo usuarios remotos.
- b) Poner a cero las variables de sesión de los usuarios.
- c) Registrar la sesión de un usuario.
- d) Reiniciar las conexiones de los usuarios remotos.

6 ¿Qué permisos se requieren para otorgar permisos en una base de datos?
- a) *process*.
- b) *reload*.
- c) *gran*.
- d) *revoque*.

7 La tabla de permisos *host* contiene:
- a) Direcciones desde donde no se puede acceder al servidor.
- b) Direcciones desde donde se puede acceder al servidor.
- c) Equipos que pueden acceder a cierta base de datos.
- d) Equipos comentados al servidor.

8 ¿Cómo creo un usuario en mi servidor?
- a) Con *GRANT*.
- b) *CREATE USER*.
- c) Con *INSERT*.
- d) Con todos ellos.

9 ¿De qué forma podemos activar los permisos?
- a) Con *FLUSH* y *mysqladmin*.
- b) No hace falta, se activan siempre nada más crearlos.
- c) Con *mysqladmin* solamente.
- d) Usando *REVOKE*.

10 ¿Podemos modificar manualmente la tabla *user*?
- a) Solo si tenemos permisos.
- b) No, solo se modifica con *GRANT* y *REVOKE*.
- c) Siempre que después reiniciemos el servidor.
- d) Solo si después hacemos *flush*.

4 Automatización de tareas: construcción de guiones de administración

OBJETIVOS DEL CAPÍTULO

- ✓ Instalar el servidor MySQL.
- ✓ Conocer las opciones de configuración de MySQL.
- ✓ Optimizar el funcionamiento de MySQL.
- ✓ Monitorizar MySQL.
- ✓ Aprender a gestionar ficheros de registro.
- ✓ Conocer la estructura del diccionario de datos de MySQL.

4.1 HERRAMIENTAS PARA AUTOMATIZAR TAREAS

Para optimizar su trabajo el administrador de bases de datos debe contar con herramientas para automatizar sus tareas dada la gran y diversa cantidad de trabajos que debe realizar.

De hecho automatizar es la clave de una buena administración. Sin esa posibilidad el administrador quedaría limitado enormemente en su trabajo.

Hoy en día casi todos los sistemas gestores, comerciales y libres, cuentan con herramientas para la automatización que en la mayoría de los casos consisten en el uso de rutinas (procedimientos y funciones) almacenados, disparadores o *triggers* y eventos además de la posibilidad de usar APIs de distintos lenguajes de programación como *perl*, *php* o *python*. Existen en cada SGBD herramientas avanzadas como en el caso de MySQL *Enterprise monitor* que facilitan la automatización y control de los servidores sin necesidad de recurrir a la programación. No obstante y dado que nunca una herramienta satisfará las necesidades específicas de todo administrador es importante conocer y ser capaz de manejar las herramientas básicas que otorgan mayor flexibilidad y potencia a costa de una mayor exigencia de conocimientos para el administrador.

A continuación veremos las **herramientas** básicas existentes en casi todos los SGBD comerciales y libres.

Dado que la sintaxis y uso de cada una de ellas llevaría otro libro hemos preferido centrarnos en ofrecer ejemplos variados y minimizar la parte teórica que por otra parte, al menos en el caso de MySQL, se encuentra disponible gratuitamente en la web oficial.

4.2 PROCEDIMIENTOS Y FUNCIONES ALMACENADOS

Las rutinas (procedimientos y funciones) almacenados son un conjunto de comandos SQL que pueden almacenarse en el servidor. Una vez que se hace, los clientes no necesitan lanzar cada comando individual sino que pueden en su lugar llamar al procedimiento almacenado como un único comando.

Las rutinas almacenadas pueden mejorar el rendimiento ya que se necesita enviar menos información entre el servidor y el cliente. El intercambio que hay aumenta la carga del servidor de la base de datos ya que la mayoría del trabajo se realiza en la parte del servidor y no en el cliente.

Además al ir integrados en la base de datos permiten la portabilidad a otros sistemas sin necesidad de adaptaciones.

Una función almacenada es un programa almacenado que devuelve un valor. Si bien los procedimientos almacenados pueden devolver valores a través de parámetros *OUT o INOUT* en las funciones solo se devuelven a través de cero o un único valor de retorno. A diferencia de los procedimientos almacenados, las funciones almacenadas se pueden utilizar en expresiones y pueden incluirse en otras funciones o procedimientos así como en el interior de sentencias SQL como *SELECT, UPDATE, DELETE* e *INSERT*.

El esquema general de una rutina almacenada se resume en la siguiente imagen:

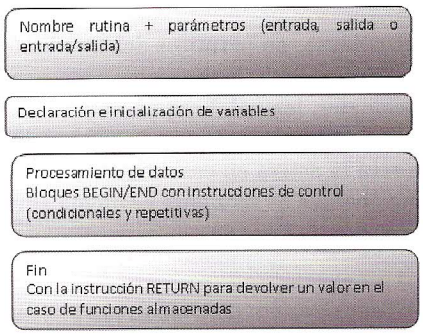


Figura 4.1. *Esquema general de una rutina*

4.2.1 SINTAXIS Y EJEMPLOS DE RUTINAS ALMACENADAS

Para construir nuestros propios **procedimientos** y **funciones** (así como otros objetos como *triggers* y vistas) necesitaremos un editor como *notepad++* u otro de nuestra preferencia y nuestros programas servidor y cliente de MySQL. También podemos crearlos directamente desde la consola cliente *mysql* o usando la GUI (*Graphical User Interface) de MySQL Workbench.*

La sintaxis general para la creación de un procedimiento o función es:

```
CREATE PROCEDURE sp_name ([parameter[,...]])
[characteristic ...] routine_body
CREATE FUNCTION sp_name ([parameter[,...]])
RETURNS type
[characteristic ...] routine_body
parameter:
[ IN | OUT | INOUT ] param_name type
type:
Any valid MySQL data type
characteristic:
LANGUAGE SQL
| [NOT] DETERMINISTIC
| { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
| SQL SECURITY { DEFINER | INVOKER }
```

- *sp_name*: es el nombre de la rutina almacenada.
- *parameter*: son los parámetros que en general se caracterizan por un tipo y un nombre. El tipo puede ser de entrada (*IN*), salida (*OUT*) o entrada/salida (*INOUT*).
- *routine_body*: es el cuerpo de la rutina formado genralemnte por sentencias SQL. En caso de haber más de una deben ir dentro de un bloque delimitado por sentencias *BEGIN* y *END* como veremos en los siguientes ejemplos.
- *Deterministic*: indica si es determinista o no es decir si siempre produce el mismo resultado.
- *Contains* SQL/*no* SQL: especifica si contiene sentencias SQL o no.

- *Modifies* SQL *data* / *Reads* SQL *data*: indica si las sentencias modifican o no los datos.
- SQL *security*: determina si debe ejecutarse con permisos del creador (*definer*) o del que lo invoca (*invoker*).

Los corchetes indican parámetros opcionales, las palabras en mayúsculas son reservadas del lenguaje SQL y el resto son opciones que se explican detalladamente en el manual de referencia. Todos los procedimientos o funciones se crean asociados a una base de datos que será la activa en ese momento o la que pongamos como prefijo en el nombre del mismo. De forma resumida una rutina obedece al siguiente esquema simplificado:

```
nombre(parametros)+modificadores
begin
declaración (DECLARE) y establecimiento de variables (SET)
proceso de datos (instrucciones sql/ instrucciones de control)
end
```

En lo sucesivo iremos explicando las cláusulas más típicas con distintos ejemplos. Dado su caráter ilustrativo los crearemos en la base de datos *test* creada por defecto en la instalación de MySQL.

Veamos un primer ejemplo de un procedimiento almacenado:

EJEMPLO 4.1

```
1. DELIMITER $$

2. DROP PROCEDURE IF EXISTS hola_mundo$$
3. CREATE PROCEDURE test.hola_mundo()
4. BEGIN
5. SELECT 'hola mundo';
6. END$$
```

Es un procedimiento muy simple cuyo único objeto es imprimir por pantalla la cadena 'hola mundo'. Este y otros ejemplos más complejos servirán de base para explicar su sintaxis. A continuación lo explicamos detalladamente por líneas:

Línea 1. La palabra clave *DELIMITER* indica el carácter de comienzo y fin del procedimiento. Típicamente sería un ; pero dado que necesitamos un ; para cada sentencia SQL dentro del procedimiento es conveniente usar otro carácter (normalmente $$ o //).

Línea 2. Eliminamos el procedimiento si es que existe. Esto evita errores cuando queremos modificar un procedimiento existente.

Línea 3. Indica el comienzo de la definición de un procedimiento donde debe aparecer el nombre seguido por paréntesis entre los que pondremos los prámetros en caso de haberlos. En este caso precedemos al nombre con la base de datos *test* a la que pertenecerá el procedimiento.

Línea 4. *Begin* indica el comienzo de una serie de bloques de sentencias *sql* que componen el cuerpo del procedimiento cuando hay mas de una.

Línea 5. Conjunto de sentencias SQL, en este caso un *select* que imprime la cadena por pantalla.

Línea 6. Fin de la definición del procedimiento seguido de un doble $ indicando que ya hemos terminado.

En caso de encontrarnos en un cliente (tanto desde nuestra consola como desde el *browser* de *mysql workbench)* podemos ejecutar el código del procedimiento directamente. Si hemos usado un editor guardaremos el código en un fichero con un nombre y extensión apropiados, en este caso *hola_mundo.sql* que ejecutaremos desde el cliente con el comando *source:*

En nuestro caso crearemos el procedimiento con el comando *source*:

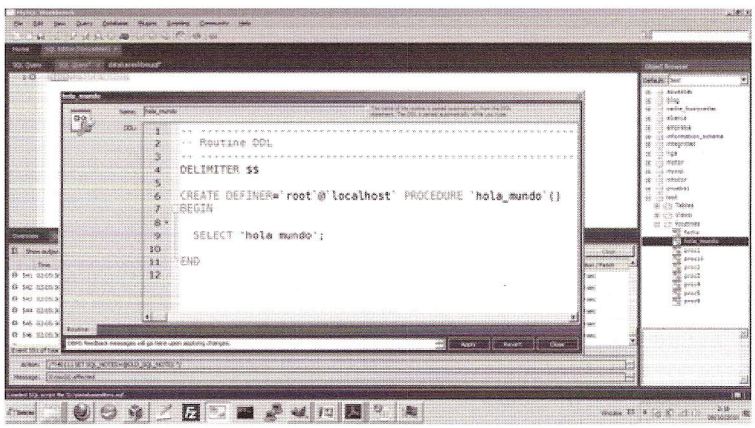
```
mysql> source hola_mundo.sql
Query OK, 0 rows affected, 1 warning (0.01 sec)

Query OK, 0 rows affected (0.00 sec)
```

Una vez creado estamos en condiciones de ejecutarlo llamándolo con el commando *CALL*:

```
mysql> call hola_mundo() $$
+-------------+
| Hola mundo |
+-------------+
| Hola mundo|
+-------------+
1 row in set (0.01 sec)
Query OK, 0 rows affected (0.01 sec)
```

Si queremos comprobar que el procedimiento existe en la base de datos *test* usaremos el comando *SHOW CREATE PROCEDURE hola_mundo*.

Para realizar lo mismo usando el *Workbench* usaríamos un *script tab* en el que incluiríamos el código del procedimiento, quedaría algo asi:



También podemos crear procedimientos directamente desde la consola:

EJEMPLO 4.2

```
CREATE PROCEDURE version SELECT version(); $$
```

En este trivial ejemplo mostramos directamente la versión de MySQL usando la función versión.

Como hemos observado las sentencias *BEGIN* y *END* solo son necesarias en caso de tener más de una sentencia.

EJEMPLO 4.3

```
DELIMITER $$
CREATE PROCEDURE fecha()
LANGUAGE SQL
NOT DETERMINISTIC
COMMENT 'A Procedure'
SELECT CURRENT_DATE, RAND() FROM t $$
```

Observamos algunas líneas nuevas como *LANGUAGE* para indicar el lenguaje, *NOT DETERMINISTIC* que indica que el algoritmo no siempre produce el mismo resultado cada vez que se llama y *COMMENT* para documentar el procedimiento con comentarios.

Estas cláusulas permiten configurar el comportamiento del procedimiento o funciones. Veremos y comentaremos otras a lo largo del capítulo.

En este ejemplo obtenemos la fecha actual (motivo por el cual es no *deterministic*) así como un número aleatorio por pantalla.

Veamos unos ejemplos de dos funciones almacenadas creadas sobre la base de datos *test*:

EJEMPLO 4.4

```
DELIMITER $$
CREATE FUNCTION estado(in_estado CHAR(1))
    RETURNS VARCHAR(20)
BEGIN
    DECLARE estado VARCHAR(20);

    IF in_estado = 'P' THEN
        SET estado='caducado';
    ELSEIF in_estado = 'O' THEN
        SET estado='activo';
    ELSEIF in_status = 'N' THEN
        SET estado='nuevo';
    END IF;
    RETURN(estado);
END;$$
```

En este ejemplo trivial la función recibe un valor de estado como entrada y comprueba su valor. Según cual sea se asignará con el comando SET el valor abreviado a la variable estado que es devuelta.

EJEMPLO 4.5

```
DELIMITER $$
CREATE FUNCTION esimpar(numero int)
       RETURNS int
BEGIN
DECLARE impar INT;
        IF MOD(numero,2)=0 THEN SET impar=FALSE;
        ELSE SET impar=TRUE;
        END IF;
        RETURN(impar);
END ;$$
```

En este caso recibimos un número como entrada devolviendo *TRUE* si es par y *FALSE* en caso de que sea impar.

Una función puede ser llamada con su nombre y una lista de parámetros con el tipo de dato apropiado. Veamos el siguiente ejemplo en el que llamamos a una función desde la línea de comandos:

EJEMPLO 4.6

```
mysql> SET @x=impar(42);
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT impar(42);
```

En el anterior ejemplo la función *impar* devuelve un 0 o un 1 si la variable pasada como parámetro es o no par. Dicho valor se asigna a una variable de sesión *@x* que podemos mostrar con un *SELECT*. Otra opción es usar directamente la función como una expresión en la cláusula *SELECT* tal como se ve en el ejemplo.

Sin embargo es más usual llamar a las funciones desde otras funciones o procedimientos como en el siguiente ejemplo:

EJEMPLO 4.7

```
DELIMITER $$
DROP PROCEDURE IF EXISTS muestra_estado$$
CREATE PROCEDURE muestra_estado(in numero int)
BEGIN
IF (esimpar(numero)) THEN
    SELECT CONCAT(numero," es impar");
ELSE
    SELECT CONCAT(numero," es par");
END IF;
END;$$
```

Ahora la nueva función nos muestra un mensaje indicando la paridad del parámetro.

De este modo las funciones permiten reducir la complejidad aparente del código encapsulando el código y simplificando por tanto su mantenimiento y legibilidad.

En estos ejemplos ya hemos incluido algunos ejemplos con instrucciones de control como *IF*. A continuación explicaremos los detalles de sintaxis más importantes.

4.2.2 PARAMETROS Y VARIABLES

Igual que en otros lenguajes de programación los procedimientos y funciones usan variables y parámetros que determinan la salida del algoritmo.

Veámoslo en el siguiente ejemplo más completo que el anterior:

EJEMPLO 4.8

```
DELIMITER $$
DROP PROCEDURE IF EXISTS proc1 $$
CREATE PROCEDURE proc1 /*nombre */
(IN parametro1 INTEGER)        /*parametros */
BEGIN                           /*comienzo de bloque */
DECLARE variable1 INTEGER;     /*variables */
DECLARE variable2 INTEGER;     /*variables */
IF parametro1 = 17 THEN  /*instrucción condicional */
SET variable1 = parametro1;    /*asignación */
ELSE
SET variable2 = 30;            /*asignación */
END IF;                   /*fin de condicional*/
INSERT INTO t VALUES
(variable1),(variable2); /*instruccion sql */
END $$                    /*final de bloque*/
DELIMITER ;$$
```

Encontramos dos nuevas cláusulas para el manejo de variables:

DECLARE: crea una nueva variable con su nombre y tipo. Los tipos son los usuales de MySQL como *char*, *varchar*, *int*, *float*, etc...Esta cláusula puede incluir una opción para indicar valores por defecto. Si no se indica, dichos valores serán *NULL*. Por ejemplo:

```
DECLARE a, b INT DEFAULT 5;
```

Crea dos variables enteras con valor 5 por defecto.

SET: permite asignar valores a las variables usando el operador de igualdad.

En el ejemplo se recibe una variable entera de entrada llamada *parámetro1*. A continuación se declaran sendas variables *variable1* y *variable2* de tipo entero y se testea el valor del parámetro, en caso de que sea 17 se asigna su valor al a variable *v1* y sino la variable *v2* se le asigna el valor 30.

Obviamente el ejemplo es absurdo y solo tiene propósitos didácticos.

Profundizaremos en los detalles en las siguientes secciones.

Tipos de parámetros

También observamos la posibilidad de incluir un parámetro. Existen tres tipos:

- ***IN***: Es el tipo por defecto y sirve para incluir parámetros de entrada que usara el procedimiento. En este caso no se mantienen las modificaciones.

EJEMPLO 4.9

```
DELIMITER $$
CREATE PROCEDURE proc2(IN p INT) SET @x = p
$$

mysql> CALL proc2(12345)$$
Query OK, 0 rows affected (0.00 sec)
mysql> SELECT @x$$
+-------+
| @x |
+-------+
| 12345 |
+-------+
1 row in set (0.00 sec)
```

En este ejemplo de procedimiento se establece el valor de una variable de sesión (precedida por @) al valor de entrada *p*.

- ***OUT***: Parámetros de salida. El procedimiento puede asignar valores a dichos parámetros que son devueltos en la llamada.

EJEMPLO 4.10

```
CREATE PROCEDURE proc3(OUT p INT)  SET p = -5 $$
mysql> CALL proc3(@y)$$
mysql> SELECT @y$$
+------+
| @y |
+------+
| -5 |
+------+
```

En este caso hemos creado una nueva variable *@y* al llamar al procedimiento cuyo valor se actualiza dentro del mismo por ser ésta de tipo *OUT*.

- ***INOUT***: Permite pasar valores al proa que serán modificados y devueltos en la llamada.

EJEMPLO 4.11

```
CREATE PROCEDURE proc4(INOUT p INT)  SET p = p-5 $$
mysql> SET @y=0$$
mysql> CALL proc4(@y)$$
mysql> SELECT @y$$
+------+
| @y |
+------+
| -5 |
+------+
```

Esta vez usamos el mismo valor del parámetro para incrementar su valor previamente asignado con *SET*.

Alcance de las variables

Las variables tienen un alcance que está determinado por el bloque *BEGIN/END* en el que se encuentran. Es decir, no podemos ver una variable que se encuentra fuera de un procedimiento salvo que la asignemos a un parámetro *out* o a una variable de sesión (usando la @). El siguiente ejemplo ilustra lo dicho:

EJEMPLO 4.12

```
DELIMITER $$
CREATE PROCEDURE proc5()
BEGIN
DECLARE x1 CHAR(5) DEFAULT 'fuera';
BEGIN
DECLARE x1 CHAR(5) DEFAULT 'dentro';
SELECT x1;
END;
SELECT x1;
END; $$
```

Las variables *x1* del primer y segundo bloque *Begin/End* son distintas, solo tienen validez dentro del bloque como se demuestra en la llamada al procedimiento.

```
mysql> CALL proc5()$$
+--------+
| x1 |
+--------+
| dentro |
+--------+
+--------+
| x1 |
+--------+
| fuera |
+--------+
```

ACTIVIDADES 4.1

- Sobre la base de pruebas test crea un procedimiento para mostrar el año actual.
- Crea y muestra una variable de usuario con SET. ¿Debe ser de sesión o puede ser global?
- Usa un procedimiento que sume uno a la variable anterior cada vez que se ejecute. En este caso la variable es de entrada salida ya que necesitamos su valor para incrementarlo y además necesitamos usarlo después de la función para comprobarlo.
- Crea un procedimiento que muestre las tres primeras letras de una cadena pasada como parámetro en mayúsculas.
- Crea un procedimiento que muestre dos cadenas pasadas como parámetros concatenadas y en mayúscula.
- Crea una función que devuelva el valor de la hipotenusa de un triángulo a partir de los valores de sus lados.
- Crea una función que calcule el total de puntos en un partido tomando como entrada el resultado en formato 'xxx-xxx'.

4.2.3 INSTRUCCIONES CONDICIONALES

En muchas ocasiones el valor de una o más variables o parámetros determinará el proceso de las mismas. Cuando esto ocurre debemos usar instrucciones condicionales de tipo simple o *IF* cuando solamente hay una condición, alternativas o *IF THEN ELSE* cuando hay dos posiblidades o múltiple, o *CASE* cuando tenemos un conjunto de condiciones distintas.

IF-THEN-ELSE

Como hemos visto en el ejemplo 4.2 podemos incluir instrucciones **condicionales** usando *IF* o de manera mas completa *IF-THEN-ELSE* para varias condiciones.

La sintaxis general para esta construcción es:

```
IF expr1 THEN
...
ELSEIF expr2 THEN
...
ELSE
...
END IF
```

En el siguiente ejemplo insertamos o actualizamos la tabla de prueba *t* en la base de datos test según el valor de entrada:

EJEMPLO 4.13

```
DELIMITER $$
CREATE PROCEDURE proc7 (IN par1 INT)
BEGIN
DECLARE var1 INT;
SET var1 = par1 + 1;
IF var1 = 0 THEN
INSERT INTO t VALUES (17);
END IF;
IF par1 = 0 THEN
UPDATE t SET s1 = s1 + 1;
ELSE
UPDATE t SET s1 = s1 + 2;
END IF;
END; $$
```

Cuando el valor de la variable 1 es 0 entonces hacemos una inserción, en caso de que sea 0 el parámetro de entrada actualizamos sumando 1 al valor actual y sino sumamos 2.

CASE

Cuando hay muchas condiciones el uso de la anterior estructura genera confusión en el código. Para estos casos es más apropiado el uso de la instrucción *CASE*.

Su sintaxis general es:

```
CASE expression
        WHEN value THEN
            statements
        [WHEN value THEN
            statements ...]
        [ELSE
            statements]
    END CASE;
```

Donde *expr* es una expression cuyo valor puede coincidir con uno de los posibles *val1*, *val2*, etc.

En otro caso se ejecutan las instrucciones seguidas por *ELSE*.

En el siguiente ejemplo podemos ver su funcionamiento:

EJEMPLO 4.14

```
CREATE PROCEDURE proc8(IN parameter1 INT)
BEGIN
DECLARE variable1 INT;
SET variable1 = parameter1 + 1;
CASE variable1
WHEN 0 THEN INSERT INTO t VALUES (17);
WHEN 1 THEN INSERT INTO t VALUES (18);
ELSE INSERT INTO t VALUES (19);
END CASE;
END; $$
```

ACTIVIDADES 4.2

- Crea una función que devuelva 1 ó 0 si un número es o no divisible por otro
- Usa las estructuras condicionales para mostrar el día de la semana según un valor de entrada numérico, 1 para domingo, 2 lunes, etc.
- Crea una función que devuelva el mayor de tres números pasados como parámetros.
- Sobre la base de datos *liga* crea una función que devuelva 1 si ganó el visitante y 0 en caso contrario. El parámetro de entrada es el resultado con el formato 'xxx-xxx'.

- Crea un procedimiento que diga si una palabra, pasada como parámetro, es palíndroma.
- Crea una función en la base *liga* que compruebe si los partidos ganados por un equipo coinciden con el campo *pg* en la tabla equipo.
- Usa una función para insertar registros de movimientos en una cuenta de un cliente comprobando previamente que la fecha es menor que la actual y que la operación no deja la cuenta en negativo. La función devolverá un 0 en caso de error de entrada y 1 en cualquier otro caso.

4.2.4 INSTRUCCIONES REPETITIVAS O *LOOPS*

Los *loops* permiten iterar un conjunto de instrucciones un número determinado de veces. Para ello MySQL provee tres tipos de instrucciones: *Simple loop, Repeat until* y *while loop*.

Simple loop

Su sintaxis básica es:

```
[etiqueta:] LOOP
instrucciones
END LOOP
[etiqueta];
```

Donde la palabra opcional *label* permite etiquetar el *loop* para podernos referir a el dentro del bloque.

El siguiente ejemplo muestra un bucle infinito que no se recomienda probar:

```
Infinite_loop: LOOP
    SELECT 'Esto no acaba nunca!!!';
END LOOP inifinite_loop;
```

En el siguiente ejemplo etiquetamos el *loop* con el nombre *loop_label*. El *loop* o bucle se ejecuta mientras no lleguemos a la condición de la línea 8. En caso de que se cumpla la orden *LEAVE* termina el *loop* etiquetado como *loop_label*.

EJEMPLO 4.15

```
0. DELIMITER $$
1. CREATE PROCEDURE proc9()
2. BEGIN
3.        DECLARE cont INT;
4.        SET cont = 0;
5.        loop_label: LOOP
6.             INSERT INTO t VALUES (cont);
7.             SET cont = cont + 1;
8.             IF cont >= 5 THEN
9.                  LEAVE loop_label;
10.            END IF;
11.       END LOOP;
12. END; $$
```

Como vemos todo el proceso queda delimitado en un bloque *BEGIN/END* el cual incluye un bucle que comienza en la línea 5 y termina en la línea 11. A su vez éste bucle realiza la inserción de una fila con el valor del contador *cont* en la tabla t (si estamos en la base de datos *test* deberíamos crearla) en la línea 6, incrementa el valor del contador *cont* en una unidad con *SET* en la línea 7 e incluye una instrucción condicional simple que comprueba el valor del contador *cont* de manera que cuando éste supere el valor 5 se producirá las salida del bucle con la instrucción LEAVE de la línea 9. Finalmente termina el *IF*.

Repeat until loop

Sintaxis general:

```
[etiqueta:] REPEAT
instrucciones
UNTIL expresion
END REPEAT [etiqueta]
```

En el siguiente ejemplo se muestran los números impares desde 0 a 10:

EJEMPLO 4.16

```
DELIMITER $$
CREATE PROCEDURE proc10()
BEGIN
DECLARE i int;
SET i=0;
loop1: REPEAT
          SET i=i+1;
IF MOD(i,2)<>0 THEN /*número impar*/
select concat(i," es impar");
END IF;
UNTIL i >= 10
END REPEAT;
END; $$
```

While loop

Sintaxis general:

```
[etiqueta:] WHILE expression DO
instrucciones
END WHILE [etiqueta]
```

El siguiente ejemplo es igual que el anterior usando *while:*

EJEMPLO 4.17

```
DELIMITER $$
CREATE PROCEDURE proc10()
DECLARE i int;
SET i=1;
loop1: WHILE i<=10 DO
          IF MOD(i,2)<>0 THEN
               SELECT CONCAT(i," es impar");
          END IF;
          SET i=i+1;
END WHILE loop1;
```

ACTIVIDADES 4.3

- Sobre la base *test* crea un procedimiento que muestre la suma de los primeros *n* números enteros, siendo *n* un parámetro de entrada.
- Haz un procedimiento que muestre la suma de los términos 1/*n* con n entre 1 y *m*. es decir 1/2+1/3+...1/*m* siendo m el parámetro de entrada. Ten en cuenta que *m* no puede ser cero.
- Crea una función que determine si un número es primo devolviendo 0 ó 1.
- Usando la función anterior crea otra que calcule la suma de los primeros *m* números primos empezando en el 1.
- Crea un procedimiento para generar y almacenar en la tabla *primos* (primos(id, numero)) de la base test los primeros números primos comprendidos entre 1 y *m* (parámetro de entrada). Modifica el procedimiento para almacenar en la variable de salida @*np* el número de primos almacenado.
- Crea un procedimiento que genere n registros aleatorios en la tabla movimientos de la base *ebanca*. Cada registro deberá contener datos de clientes y cuentas existentes. La cantidad deberá estar entre 1 y 100000 y la fecha será la actual.

4.2.5 SQL EN RUTINAS: CURSORES

Hasta ahora todos los ejemplos contenían instrucciones o expresiones referidas a cálculos matemáticos o de de cadenas sencillos sin implicar el uso de datos de una base de datos. Normalmente sin embargo el uso de procedimientos implica manipular datos de tablas de bases de datos lo que implica usar instrucciones *sql*. En esta sección veremos ejemplos diversos de procedimientos que acceden a bases de datos haciendo uso de las **instrucciones** explicadas en apartados anteriores.

En general podemos usar cualquier instrucción de *sql*, tanto perteneciente al *DDL*, *DML* o *DCL*.

Como siempre veamos un ejemplo para empezar en el que usamos sentencias *sql* de definición (*drop* y *create*) y sentencias *sql* de manipulación (*insert*, *update* y *delete*):

EJEMPLO 4.18

```
DELIMITER $$
CREATE PROCEDURE simple_sql(  )
BEGIN
DECLARE i INT DEFAULT 1;
/* instrucción DDL */
DROP TABLE IF EXISTS test_table ;
CREATE TABLE test_table(id          INT PRIMARY KEY,
          some_data VARCHAR(30))
      ENGINE=innodb;
/* INSERT usando una variable de procedimiento */
WHILE (i<=10) DO
INSERT INTO TEST_TABLE VALUES(i,CONCAT("record ",i));
SET i=i+1;
END WHILE;
/* Ejemplo de acualización usando variables de procedimiento*/
SET i=5;
   UPDATE test_table
SET some_data=CONCAT("I updated row ",i)
WHERE id=i;

/* DELETE with a procedure variable */
DELETE FROM test_table WHERE id>i;

END;$$
```

En el siguiente ejemplo usamos la propiedad de las sentencias *select* de enviar valores a variables usando *INTO*:

EJEMPLO 4.19

```
SELECT expresion1 [, expresion2 ....]
      INTO variable1 [, variable2 ...]
      otras instrucciones SELECT

DELIMITER $$
CREATE PROCEDURE motorblog.obtener_datos_noticia(id_noticia INT)
BEGIN
DECLARE vtitulo     VARCHAR(200);
DECLARE vcontenido   TEXT;
DECLARE vfecha DATE;

SELECT titulo, contenido, fecha INTO vtitulo, vcontenido, vfecha FROM noticias
WHERE id=id_noticia;

/* Procesamos los datos obtenidos, por ejemplo mostrándolos*/
SELECT vtitulo, vcontenido, vfecha;

END;$$
```

En el anterior ejemplo observamos como la sentencia *SELECT* asigna los valores de la fila seleccionada para asignarlos a su vez a nuevas variables internas del procedimiento. No obstante muchas veces querremos recuperar más de una fila para manipular sus datos, en estos casos no sirve la sentencia anterior y requerimos el uso de cursores. Conceptualmente un cursor se asocia con un conjunto de filas o una consulta sobre una tabla de una base de datos.

Un cursor se define del siguiente modo:

```
DECLARE cursor_name CURSOR FOR SELECT_statement;
```

EJEMPLO 4.20

```
DECLARE cursor1 CURSOR FOR
SELECT titulo, contenido, fecha FROM noticias;
```

Y debe hacerse después de declarar todas las variables necesarias para el procedimiento.

Un ejemplo con variables sería:

EJEMPLO 4.21

```
DELIMITER $$
CREATE PROCEDURE cursor_demo(id_noticia INT)
BEGIN
DECLARE vid INT;
DECLARE vtitulo VARCHAR(30);
DECLARE c1 CURSOR FOR
SELECT id, titulo FROM noticias WHERE id=id_noticia;
END; $$
```

Comandos relacionados con cursores

Para manipular los cursores disponemos de una serie de comandos:

- *OPEN*: inicializa el conjunto de resultados asociados con el cursor.

```
OPEN cursor_name
```

- *FETCH*: extrae la siguiente fila de valores del conjunto de resultados del cursor moviendo su puntero interno una posición.

```
FETCH cursor_name INTO variable list;
```

- *CLOSE*: cierra el cursor liberando la memoria que ocupa y haciendo imposible el acceso a cualquiera de sus datos.

```
CLOSE cursor_name ;
```

En el siguiente ejemplo vemos cómo extraer una sola fila de una tabla:

EJEMPLO 4.22

```
OPEN cursor1;
FETCH cursor1 INTO vtitulo, vcontenido, vfecha;
CLOSE cursor1;
```

Para el caso de más de una fila necesitamos un bucle:

EJEMPLO 4.23

```
DELIMITER $$
CREATE PROCEDURE cursor_demo2(id_noticia INT)
BEGIN
DECLARE tmp VARCHAR(30);
DECLARE cursor2 CURSOR FOR SELECT titulo FROM noticias;
   OPEN cursor2;
l_cursor: LOOP
FETCH cursor2 INTO tmp;
END LOOP l_cursor;
CLOSE cursor2;
END; $$
```

En el anterior ejemplo se produce un error similar al siguiente:

```
mysql> call simple_cursor_loop();
    ERROR 1329 (02000): No data to FETCH
```

Dado que cuando llegamos a la última fila no hay más datos que obtener asi que necesitamos de algun modo detectar ese momento. Para ello usaremos un manejador de errores o *handler* (explicado en la siguiente sección). Para ello necesitamos la siguiente instrucción:

```
DECLARE CONTINUE HANDLER FOR NOT FOUND SET l_last_row_fetched=1;
```

Que hace dos cosas:

- Establecer la variable ultima_fila = 1.
- Permitir al programa continuar su ejecución.

Así nuestro procedimiento quedaría del siguiente modo:

EJEMPLO 4.24

```
DELIMITER $$
CREATE PROCEDURE cursor_demo3()
BEGIN
DECLARE tmp VARCHAR(200);
DECLARE lrf BOOL;
DECLARE nn INT;
DECLARE cursor2 CURSOR FOR SELECT titulo FROM noticias;
DECLARE CONTINUE HANDLER FOR NOT FOUND SET lrf=1;
SET lrf=0,nn=0;
OPEN cursor2;
l_cursor: LOOP
FETCH cursor2 INTO tmp;
SET nn=nn+1;
IF lrf=1 THEN
LEAVE l_cursor;
END IF;
END LOOP l_cursor;
CLOSE cursor2;
SELECT nn;
END; $$
```


Casi todos los cursores necesitan al menos un manejador para el caso de *not found*.

En este caso hemos declarado dos nuevas variables, *lrf* (*last row fetched* o última fila extraida) que es una variable *booleana* con posibles valores 0 y 1 indicando si hemos llegado o no a la última fila del cursor, por su parte *nn* almacena el número de noticias o registros contenidos en el cursor. Gracias a la sentencia *LEAVE*. podemos terminar el bucle cuando *lrf* adquiere el valor 1 o lo que es lo mismo, se alcanza el final del cursor.

Finalmente cuando ejecutemos el procedimiento veremos el número de noticias gracias a la sentencia final *SELECT nn*.

Veremos ahora el mismo procedimiento con las distintas instrucciones ya comentadas:

Cursor con *repeat until*

EJEMPLO 4.25

```
DELIMITER $$
CREATE PROCEDURE cursor_demo4()
BEGIN
DECLARE tmp VARCHAR(200);
DECLARE lrf bool;
DECLARE nn int;
DECLARE cursor2 CURSOR FOR SELECT titulo FROM noticias;
DECLARE CONTINUE HANDLER FOR NOT FOUND SET lrf=1;

SET lrf=0,nn=0;
OPEN cursor2;
l_cursor: REPEAT
FETCH cursor2 INTO tmp;
set nn=nn+1;
IF lrf=1 THEN LEAVE l_cursor;
END IF;
UNTIL lrf
END REPEAT l_cursor;
CLOSE cursor2;
SELECT nn;
END; $$
```

Cursor con *while*

EJEMPLO 4.26

```
DELIMITER $$
CREATE PROCEDURE cursor_demo5()
BEGIN
DECLARE tmp VARCHAR(200);
DECLARE lrf bool;
DECLARE nn int;
DECLARE cursor2 CURSOR FOR SELECT titulo FROM noticias;
DECLARE CONTINUE HANDLER FOR NOT FOUND SET lrf=1;

SET lrf=0,nn=0;
OPEN cursor2;
l_cursor: WHILE(lrf=0) DO
FETCH cursor2 INTO tmp;
SET nn=nn+1;
IF lrf=1 THEN LEAVE l_cursor;
END IF;

END WHILE l_cursor;
CLOSE cursor2;
SELECT nn;
END; $$
```

Posiblemente ésta última es la construcción mas usada ya que, a diferencia de las anteriores, se evalua la condición antes de leer un registro del cursor.

Para ilustrar lo visto hasta ahora veremos un ejemplo más elaborado en el que se obtienen y muestran el número de noticias publicadas de cada autor para lo cual se precisan dos cursores:

(Obviamente pude calcularse con una consulta pero el propósito ahora solo es didáctico).

EJEMPLO 4.27

```
DELIMITER $$
CREATE PROCEDURE noticias_autor(  )
   READS SQL DATA
BEGIN
DECLARE vautor,na_count INT;
DECLARE fin BOOL;
DECLARE autor_cursor cursor FOR SELECT id_autor FROM autor;
DECLARE noticia_cursor cursor FOR SELECT autor FROM noticias WHERE autor=vautor;
DECLARE CONTINUE HANDLER FOR NOT FOUND SET fin=1;
SET na_count=0;
OPEN autor_cursos;
autor_loop: LOOP
FETCH ac into vautor;
IF fin=1 THEN LEAVE autor_loop;
END IF;
OPEN noticia_cursor;
SET na_count=0;
noticias_loop: LOOP
FETCH nc INTO vautor;
IF fin=1 THEN LEAVE autor_loop;
END IF;
SET na_count=na_count+1;
END LOOP noticias_loop;
CLOSE noticia_cursor;
SET fin=0;
SELECT CONCAT('El autor',vautor,'tiene', na_count,' noticias');
END LOOP autor_loop;
CLOSE autor_cursor;
END;$$
```

Como vemos para cada autor de la tabla de autores se obtiene un cursor con sus noticias. Éste se usa para contar el número de noticias y devolver el resultado usando un select que muestra la información correspondiente a cada autor.

4.2.6 GESTIÓN RUTINAS ALMACENADAS

Las rutinas se manipulan con los comandos *CREATE* (ya visto), *DROP* y *SHOW*

Eliminación rutinas

Para eliminar procedimientos o funciones usamos el comando SQL *DROP* con la siguiente sintaxis:

```
DROP {PROCEDURE | FUNCTION} [IF EXISTS] sp_name
```

Consulta rutinas

Podemos ver información más o menos detalladas de nuestras rutinas usando los comandos:

```
SHOW CREATE {PROCEDURE | FUNCTION} sp_name
SHOW {PROCEDURE | FUNCTION} STATUS [LIKE 'pattern']
```

Donde en el segundo caso podemos hacer un filtro por patrones.

Estos comandos, y en general todos los de tipo *SHOW*, se nutren del diccionario de datos gracias a la tabla *INFORMATION_SCHEMA.ROUTINES* que también podemos consultar con instrucciones *SELECT*.

ACTIVIDADES 4.4

- Haz un procedimiento que muestre el nombre del autor que más noticias ha publicado en el último mes.
- Desarrolla usando cursores un procedimiento que muestre los datos del cliente, la cuenta y el saldo de los clientes con saldo negativo en alguna de sus cuentas.
- Calcula con un procedimiento la suma de todos los ingresos y cargos (por separado) en todas las cuentas de *ebanca*.
- Crea un procedimiento que muestre el número máximo de partidos seguidos ganados por un equipo en casa.

4.2.7 MANEJO DE ERRORES

Cuando un programa almacenado encuentra una condición de error, la ejecución se detiene y se devuelve un error a la aplicación que llama. Ese es el comportamiento predeterminado. ¿Qué pasa si necesitamos otro tipo de comportamiento? ¿Qué pasa si, por ejemplo, queremos que la trampa de error, *log*, o informar al respecto y luego continuar con la ejecución de nuestra aplicación? Para ese tipo de control tenemos que definir controladores de excepciones en nuestros programas (iguales que los vistos en la parte de cursores).

Veamos un ejemplo de procedimiento sin manejo de errores:

EJEMPLO 4.28

```
CREATE PROCEDURE insertar_noticia
(titulo   VARCHAR(200), contenido TEXT, fecha DATE)
MODIFIES SQL DATA
BEGIN
INSERT INTO noticias(titulo, contenido, fecha) VALUES(in_location,in_address1,in_
address2,zipcode);
END$$
```

Funciona bien cuando no existe el registro:

```
mysql> CALL insertar_noticia('titulo1','noticia de prueba','11-10-2011');
```

Sin embargo, si intentamos insertar una noticia ya existente MySQL genera un error similar al siguiente:

```
ERROR 1062 (23000): Duplicate entry 'titulo1' for key 1
```

que indica la existencia de una clave repetida en el campo título. En general los errores deben ser prevenidos y tratados o manejados. El mismo procedimiento con manejo de errores sería:

EJEMPLO 4.29

```
DELIMITER $$
CREATE PROCEDURE insertar_autor
(pautor INT,plogin VARCHAR(45),OUT estado VARCHAR(45))
MODIFIES SQL DATA
BEGIN
DECLARE CONTINUE HANDLER FOR 1062 SET estado='Duplicate Entry';
SET estado='OK';
INSERT INTO autores(id_autor, login) VALUES(pautor,plogin);
END;$$
```

Pero si queremos hacer algo con el error debemos usar la variable *out_status*. En el siguiente ejemplo llamamos al procedimiento dentro de otro procedimiento condicionando la salida al valor de la variable estado de tipo *out*:

EJEMPLO 4.30

```
CREATE PROCEDURE insertar_comentario
(pautor INT,pfecha DATE, pcontenido VARCHAR(30))
MODIFIES SQL DATA
BEGIN
DECLARE estado VARCHAR(20);

CALL insertar_autor(pautor, plogin, estado);
IF estado='Duplicate Entry' THEN
SELECT CONCAT('Warning: autor repetido ',pautor,'login',plogin) AS warning;
END IF;
INSERT INTO comentarios(autor, contenido, fecha) VALUES(pautor,pcontenido,pfecha
);

END;$$
```

Cuando llamamos al procedimiento *insertar_autor*, éste devuelve en la variable estado el valor almacenado en dicho procedimiento de forma que podemos controlar lo que ocurrió en el mismo y actuar en consecuencia. En nuestro ejemplo simplemente avisamos con un *warning*.

4.2.7.1 Sintaxis de manejador

```
DECLARE {CONTINUE | EXIT} HANDLER FOR
        {SQLSTATE sqlstate_code| MySQL error code| condition_name}
        handler_actions
```

- **Tipo de manejador**: *EXIT* o *CONTINUE*.
- **Condición del manejador**: estado SQL (*SQLSTATE*), error propio de MySQL o código de error definido por el usuario.
- **Acciones del manejador**: acciones a tomar cuando se active el manejador.

Tipos de manejador

- *EXIT*

  Cuando se encuentra un error el bloque que se está ejecutando actualmente se termina. Si este bloque es el bloque principal el procedimiento termina, y el control se devuelve al procedimiento o programa externo que invocó el procedimiento. Si el bloque está encerrado en un bloque externo dentro del mismo programa almacenado, el control se devuelve a ese bloque exterior.

- *CONTINUE*

  Para el caso de *CONTINUE*, la ejecución continúa en la declaración siguiente a la que ocasionó el error.

En cualquier caso, las declaraciones se define dentro de la curva (el controlador de las acciones) se ejecutan bien antes de la *EXIT* o *CONTINUE* se lleva a cabo.

Veremos ahora ejemplos de ambos tipos de controladores. En el siguiente ejemplo el procedimiento crea un registro de autor. Para manejar la posibilidad de que el autor ya exista se crea el manejador de tipo *EXIT* que en caso de

activarse establecerá el valor de la variable duplicate_key a 1 y devolverá el control al bloque *BEGIN/END* exterior (de ahi el uso de dos bloques).

EJEMPLO 4.31

```
DELIMITER $$
CREATE PROCEDURE insertar_autor(pautor INT,plogin VARCHAR(45))
MODIFIES SQL DATA
BEGIN
DECLARE duplicate_key INT DEFAULT 0;
BEGIN
DECLARE EXIT HANDLER FOR 1062 /* clave repetida*/
SET duplicate_key=1;
INSERT INTO autores(id_autor,login) VALUES(pautor,plogin);
END;

IF duplicate_key=1 THEN
SELECT CONCAT('error en la inserción clave duplicada') as "Resultado";

ELSE SELECT CONCAT('Autor ',plogin,' creado') as "Resultado";
END IF;

END;$$
call insertar_autor(8,'autor1');
```

Cuando llamemos a este procedimiento dos veces consecutivas observaremos el mensaje generado por el manejador informandonos del uso de una clave (*id_autor*) repetida.

Un ejemplo de la misma funcionalidad implementada con un controlador de *CONTINUE* sería:

EJEMPLO 4.32

```
CREATE PROCEDURE insertar_autor(pautor INT,plogin VARCHAR(45))
MODIFIES SQL DATA
BEGIN
DECLARE duplicate_key INT DEFAULT 0;

DECLARE CONTINUE HANDLER FOR 1062 /* Duplicate key*/
SET duplicate_key=1;
INSERT INTO autores(id_autor,login) VALUES(pautor,plogin);

IF duplicate_key=1 THEN
SELECT CONCAT('Error en la inserción de ',plogin,'clave duplicada') as
"Resultado";
ELSE
SELECT CONCAT('Autor',plogin,' creado') as "Resultado";
END IF;
END$$
```

Un controlador de *EXIT* es más adecuado para los errores catastróficos ya que no permite ninguna forma de continuación de la tramitación.

Un controlador de *CONTINUE* es más adecuado cuando se tiene algún procesamiento alternativo que se ejecutará si la excepción se produce.

Un desencadenador de manejador define las circunstancias que activan un manejador. Pueden ser por un error de código, un error de SQL (*SQLSTATE*) o por una circunstancia definida por el usuario. Por defecto al indicar un error numérico nos referiremos a un error de SQL, por ejemplo:

```
DECLARE CONTINUE HANDLER FOR 1062 SET duplicate_key=1;
```

Significa que cuando se produzca el error 1062 de MySQL la variable *duplicate_key* se pondrá a 1 (los códigos de error como ya indicamos se encuentran definidos en el manual de referencia así como se pueden obtener usando la función *perror()* incluida por defecto en la distribución del servidor).

El mismo aejemplo anterior usando un error estándar o *SQLSTATE* sería:

```
DECLARE CONTINUE HANDLER FOR SQLSTATE '23000' SET duplicate_key=1;
```

4.3 TRIGGERS

Un *trigger* o **disparador** es un tipo especial de rutina almacenada que se activa o ejecuta cuando en una tabla ocurre un evento de tipo *INSERT, DELETE* o *UPDATE*. Es decir, los disparadores implementan una funcionalidad asociada a cualquier cambio en una tabla. Por ejemplo, en la base de datos *ebanca* lo siguiente crea un disparador para sentencias *INSERT* dentro de la tabla. El disparador suma las cantidades insertadas cada vez que se introduce un nuevo movimiento en la variable de usuario *@sum*:

EJEMPLO 4.33

```
mysql> CREATE TRIGGER insertar_movimiento BEFORE INSERT ON movimiento
FOR EACH ROW SET @sum = @sum + NEW.cantidad;
```

4.4.1 GESTIÓN DE DISPARADORES

Las instrucciones para gestionar disparadores son *CREATE TRIGGER*, *SHOW TRIGGER* y *DROP TRIGGER*.

Crear trigger

```
CREATE TRIGGER nombre_disp momento_disp evento_disp
ON nombre_tabla FOR EACH ROW sentencia_disp
```

- *momento_disp*: es el momento en que el disparador entra en acción. Puede ser ***BEFORE*** (antes) o ***AFTER*** (despues), para indicar que el disparador se ejecute antes o después que la sentencia que lo activa.

- *evento_disp*: indica la clase de sentencia que activa al disparador. Puede ser *INSERT, UPDATE* o *DELETE*. Por ejemplo, un disparador *BEFORE* para sentencias *INSERT* podría utilizarse para validar los valores a insertar.

No puede haber dos disparadores en una misma tabla que correspondan al mismo momento y sentencia. Por ejemplo, no se pueden tener dos disparadores *BEFORE UPDATE*. Pero sí es posible tener los disparadores *BEFORE UPDATE* y *BEFORE INSERT* o *BEFORE UPDATE* y *AFTER UPDATE*.

- *FOR EACH ROW*: hace referencia a las acciones a llevar a cabo sobre cada fila de la tabla indicada.
- *Sentencia_disp*.

Es la sentencia que se ejecuta cuando se activa el disparador. Si se desean ejecutar múltiples sentencias, deben colocarse entre *BEGIN ... END*, el constructor de sentencias compuestas. Esto además posibilita emplear las mismas sentencias permitidas en rutinas almacenadas.

Las columnas de la tabla asociada con el disparador pueden referenciarse empleando los alias *OLD* y *NEW*. *OLD.nombre_col* hace referencia a una columna de una fila existente, antes de ser actualizada o borrada. *NEW.nombre_col* hace referencia a una columna en una nueva fila a punto de ser insertada o en una fila existente después de ser actualizada.

Las palabras clave *OLD* y *NEW* permiten acceder a columnas en los registros afectados por un disparador. En un disparador para *INSERT*, solamente puede utilizarse *NEW.nom_col* ya que no hay una versión anterior del registro. En un disparador para *DELETE* solo puede emplearse *OLD.nom_col*, porque no hay un nuevo registro. En un disparador para *UPDATE* se puede emplear *OLD.nom_col* para referirse a las columnas de un registro antes de que sea actualizado, y *NEW.nom_col* para referirse a las columnas del registro luego de actualizarlo.

El uso de *SET NEW.nombre_col = valor* necesita que se tenga el privilegio *UPDATE* sobre la columna. El uso de *SET nombre_var = NEW.nombre_col* necesita el privilegio *SELECT* sobre la columna.

Eliminación de triggers

Para eliminar el disparador, se emplea una sentencia *DROP TRIGGER*. El nombre del disparador debe incluir el nombre de la tabla:

```
DROP TRIGGER [IF EXISTS] [schema_name.]trigger_name
```

Consulta de triggers

Podemos obtener informacion de los *trigger* creados con *SHOW TRIGGERS*.

```
SHOW TRIGGERS [{FROM | IN} db_name]
    [LIKE 'pattern' | WHERE expr]
```

Este comando nos permite mostrar *trigger* de una base de datos filtrándolo con un patrón o cláusula WHERE.

Cuando creamos *triggers* se crea un nuevo registro en la tabla *INFORMATION_SCHEMA* llamada *INFORMATION_SCHEMA.TRIGGERS* que podemos visualizar con el siguiente comando:

```
mysql> SELECT trigger_name, action_statement FROM information_schema.triggers
```

4.4.2 USOS DE DISPARADORES

Aunque su uso es muy variado y depende mucho del tipo de aplicación o base de datos con que trabajemos podemos hacer una clasificación más o menos general.

Control de sesiones

En ocasiones puede ser interesante recoger ciertos valores en variables de sesión creadas por el usuario que al final nos permitan ver un resumen de lo realizado en dicha sesión. Este es el caso del ejemplo anterior.

En dicho ejemplo vemos como antes de insertar uno o varios movimientos se acumula la cantidad de todos ellos en la variable de usuario *@sum*. Para utilizarlo se debe establecer el valor de la variable acumulador a cero, ejecutar una o varias sentencia *INSERT* y ver qué valor presenta luego la variable:

```
mysql> SET @sum = 0;
mysql>INSERT INTO movimiento VALUES(137,14.98),(141,1937.50),(97,-100.00);
mysql> SELECT @sum AS 'Total insertado';
+----------------------+
| Total amount inserted |
+----------------------+
| 1852.48 |
+----------------------+
```

En este caso, el valor de *@sum* luego de haber ejecutado la sentencia *INSERT* es 14.98 + 1937.50 - 100, o 1852.48.

Control de valores de entrada

Un uso posible de los disparadores es el control de valores insertados o actualizados en tablas.

En el ejemplo siguiente se crea un disparador en la tabla movimiento para *UPDATE* que verifica los valores utilizados para actualizar cada columna, y modifica el valor para que se encuentre en un rango de 0 a 100. Esto debe hacerse en un disparador *BEFORE* porque los valores deben verificarse antes de emplearse para actualizar el registro:

EJEMPLO 4.34

```
delimiter $$
CREATE TRIGGER comprobacion_saldo BEFORE UPDATE ON movimiento
FOR EACH ROW
BEGIN
IF NEW.cantidad < 0 THEN
SET NEW.cantidad = 0;
ELSEIF NEW.cantidad > 100 THEN
SET NEW.cantidad = 100;
END IF;
END;$$
```

En este caso cada vez que se actualize la tabla cuenta se controlará el valor del saldo para que sea siempre positivo.

Mantenimiento de campos derivados

Otro uso típico de los *triggers* es para manetnimineto de campos derivados o redundantes, o sea campos que pueden calcularse a partir de otros como por ejemplo el campo saldo en la tabla cuenta de *ebanca*. El siguiente *trigger* actualiza ese valor cada nuevo ingreso:

EJEMPLO 4.35

```
DELIMITER $$
CREATE TRIGGER actualizar_cuenta BEFORE INSERT ON movimiento
FOR EACH ROW
BEGIN
   UPDATE cuenta SET saldo= saldo+NEW.cantidad WHERE cod_cuenta=OLD.cod_cuenta;
END;$$
```

Estadísticas

Podemos registrar estadísticas de operaciones o valores de nuestras bases de datos en tiempo real usando *triggers*. Por ejemplo podemos registrar los ingresos que se hacen cada mes en una tabla aparte con el siguiente *trigger*:

EJEMPLO 4.36

```
DELIMITER $$
CREATE TRIGGER ingresos_dia AFTER INSERT ON movimiento
FOR EACH ROW
BEGIN
IF existe(MONTH(NEW.fecha),idia)=0 THEN
INSERT INTO idia(cantidad,fecha) VALUES(NEW.cantidad, NEW.fecha);
ELSE UPDATE idia SET cantidad=NEW.cantidad+cantidad WHERE mes=MONTH(NEW.fecha);
END IF;
END;$$
```

Para lo cual debemos crear la fucnión existe que nos devuelve 1 ó 0 si existe o no el registro para cada mes.

Registro y auditoría

Cuando muchos usarios acceden a las bases de datos puede ser que el registro de *log* no sea suficiente o simplemente dificulte la revisión de la actividad en el servidor en el sentido de saber quién ha hecho que operación y a qué hora. Para ello existen soluciones (por ejemplo *scripts* en *perl*) que permiten filtrar los ficheros de registro para obtener la información que necesito. Sin mebargo podemos también usar *triggers* que me faicliten dicha tarea. Podemos asignar

un *trigger* a una tabla que se dispare después (*AFTER*) de una sentencia *DELETE* o *UPDATE*, que guarde los valores del registro, así como alguna otra información de utilidad en una tabla de *log*.

Vamos a examinar un caso práctico. Queremos saber quién y a qué hora modificó la tabla movimientos en la base *ebanca*. Para ello creamos un *trigger* que registre dichas actualizaciones incluyendo los datos antiguos y los nuevos para cada registro modificado:

EJEMPLO 4.37

Lo primero es crear una tabla simple de *log/auditoría*:

```
CREATE TABLE auditoria_movimientos
(
id_mov int not null auto_increment,
cod_cuenta_ant varchar(100),
fecha_ant datetime,
cantidad_ant int,
cod_cuenta_n varchar(100),
fecha_n datetime,
cantidad_n int,
usuario varchar(40),
fecha_mod datetime,
primary key(id)
) ENGINE = InnoDB;
```

Y ahora crearemos un *trigger* para que vaya llenando los registros de esta tabla cada vez que alguien ejecute una actualización sobre la tabla:

```
CREATE TRIGGER trigger_auditoria_movimientos AFTER UPDATE ON movimientos
FOR EACH ROW
BEGIN
INSERT INTO auditoria_movimientos(cod_cuenta_ant,fecha_ant,cantidad_ant, cod_cuenta_n,
fecha_n, cantidad_n, usuario,fecha_mod)
VALUES (OLD.cod_cuenta, OLD.fecha,OLD.cantidad, NEW.cod_cuenta, NEW.fecha_n, NEW.
cantidad_n, CURRENT_USER(), NOW() );
END;
```

Como se puede observar, el *trigger* creado anteriormente se activará con la ejecución de la actualización *(UPDATE)* y agregará un nuevo registro a la tabla de auditoría cada vez que se actualize la tabla movimientos. De una forma sencilla, usando las funciones *CURRENT_USER()* y *NOW()* sabemos quién realizó una actualización y cuándo lo hizo.

Podríamos agregar una columna "accion" a esta tabla de auditoría, y registrar también sentencias *INSERT* y *DELETE* en esta misma tabla.

4.4.3 ADMINISTRACIÓN DE DISPARADORES

ACTIVIDADES 4.5

- Haz un disparador que cree un registro en la tabla *nrojos* de la base *ebanca* con los campos cliente, cuenta, fecha y saldo cada vez que algún cliente se quede en números rojos en alguna de sus cuentas.
- Crea un disparador para que cada vez que se registre un partido se actualicen los campos *pg* y *pp* según el caso en la tabla equipo.
- Crea un disparador que cada vez que se borre una noticia de la base de datos *motorblog*, registre en la tabla *log_borrados* el título de la noticia, el usuario y la fecha y hora.
- Haz lo necesario para que cada vez que un cliente de *ebanca* ingrese más de 1000 euros se le bonifique con 100, solo para clientes con cuentas que superen tres años de antigüedad y entre el 1 de enero de 2011 y el 31 de marzo de 2011.
- Haz lo necesario para que cada vez que se actualice el campo *pg* o *pp* en la tabla equipo de la base liga se actualice el campo puesto.

4.4 VISTAS

Son objetos de la base de datos que, mediante una consulta incluyen un subconjunto de datos de la base. Es como una consulta guardada.

Mediante el uso de vistas podemos filtrar mucha información que de otro modo puede resultar costosa de obtener especialmente en bases de datos de mucha complejidad con un alto grado de normalización y muchas claves ajenas.

El uso de vistas es algo cotidiano sobre todo en los entornos de producción empresarial, ya que permite por un lado que el Administrador de Bases de Datos pueda, de alguna manera, "proteger" los accesos directos a las tablas e implementar una medida de seguridad adicional, ya que una vista, al no ser una tabla en si misma, solamente estaría exponiendo el contenido, pero de un modo "protegido", y por otro lado, el Administrador de este modo, permite así mismo que los usuarios puedan "Ver", ya que de esto se trata, los datos en sí mismos, pero de un modo mas "seguro".

Para el usuario final, ver los datos en modo de "vista" o de tabla, es exactamente igual, pero no para el DBA, quien debe velar por la integridad de los mismos. Puede, y de hecho sucede, que un usuario pueda consultar una tabla a través de una "vista", pero no de un modo directo con una instrucción "SELECT".

Otro motivo para crear vistas afecta a la seguridad en el sentido de que podemos usarlas como interfaz de nuestra base de datos para distintos usuarios añadiendo asi una nueva capa de seguridad sobre nuestros datos en lo que se conoce como el nivel externo. Además en nigún caso podemos acceder a datos e las tablas en que se basan las vistas si estos no figuran en la definición de las vistas.

Podemos además dar permisos sobre **vistas** como si fuesen tablas lo que nos permite proteger las tablas originales restringiendo su acceso directo tal como vimos en el capítulo anterior.

Como en el caso de rutinas almacenadas podemos usar la cláusula SQL *security* que permite establecer los permisos con que se ejcuta la vista.

Admiten el uso de más de una tabla, otras vistas, *subconsultas* y *joins*.

Podemos también, en ciertos casos borrar y actualizar los datos de una vista de forma que queden reflejados en los correspondientes campos de las tablas subyacentes. Existen ciertas restricciones:

- Que no se usen tablas temporales.
- No usar cláusulas *GROUP BY* ni *HAVING*.
- No usar uniones ni reuniones externas.
- No usar consultas correlacionadas.
- Para el caso de reuniones *INNER* podemos actualizar o insertar siempre y cuando los campos afectados sean únicamente los de una de las tablas implicadas en el *join*.

Además las vistas reflejan instantáneamente los cambios producidos en las tablas de manera que aunque estos cambien se verán reflejados en la consulta que se haga sobre la vista.

Las vistas pueden crearse a partir de varios tipos de sentencias *SELECT*. Pueden hacer referencia a tablas o a otras vistas. Pueden usar combinaciones *UNION* y *subconsultas*. El *SELECT* inclusive no necesita hacer referencia a otras tablas. En el siguiente ejemplo se define una vista que selecciona dos columnas de otra tabla, así como una expresión calculada a partir de ellas:

Algunas vistas son actualizables. Esto significa que se las puede emplear en sentencias como *UPDATE, DELETE* o *INSERT* para actualizar el contenido de la tabla subyacente. Para que una vista sea actualizable, debe haber una relación uno-a-uno entre los registros de la vista y los registros de la tabla subyacente.

Con respecto a la posibilidad de agregar registros mediante sentencias *INSERT*, es necesario que las columnas de la vista actualizable también cumplan los siguientes requisitos adicionales:

- No debe haber nombres duplicados entre las columnas de la vista.
- La vista debe contemplar todas las columnas de la tabla en la base de datos que no tengan indicado un valor por defecto.
- Las columnas de la vista deben ser referencias a columnas simples y no columnas derivadas. Una columna derivada es la que deriva de una expresión.

No puede insertar registros en una vista conteniendo una combinación de columnas simples y derivadas, pero puede actualizarla si actualiza únicamente las columnas no derivadas.

La cláusula *WITH CHECK OPTION* puede utilizarse en una vista actualizable para evitar inserciones o actualizaciones excepto en los registros determinados por la cláusula *WHERE* incluida en la definición de la vista.

4.4.1 GESTIÓN DE VISTAS

Los comandos que nos permiten trabajar con vistas en MySQL son *CREATE VIEW*, *ALTER VIEW* y *DROP VIEW* y *SHOW CREATE VIEW*.

Creación, *CREATE VIEW*

```
CREATE
    [OR REPLACE]
```

```
[ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]
[DEFINER = { user | CURRENT_USER }]
[SQL SECURITY { DEFINER | INVOKER }]
VIEW view_name [(column_list)]
AS select_statement
[WITH [CASCADED | LOCAL] CHECK OPTION]
```

Esta sentencia crea una vista nueva o reemplaza una existente si se incluye la cláusula *OR REPLACE*. *sentencia_select* es una sentencia *SELECT* que proporciona la definición de la vista. Puede estar dirigida a tablas de la base o a otras vistas.

ALGORITHM indica el *algoritmo* que queremos que use. Existen *MERGE*, *TEMPTABLE* en cuyo caso se crea una tabla temporal y-*UNDEFINED*, en cuyo caso dejamos que sea MySQL quien decida el algoritmo por sí mismo.

Por ejemplo, para poder dar acceso a sus datos a un jugador de la base de datos liga podemos crear la siguiente vista con el algoritmo *MERGE*:

EJEMPLO 4.38

```
DELIMITER $$
mysql> create ALGORITHM=MERGE view vdatos_jugador as select * from jugador where
id_jugador=identificador_jugador$$
```

Así, mediante la vista cada jugador solo tendrá acceso a sus datos.

El caso de *TEMPTABLE* tiene una gran ventaja y una gran desventaja:

- **Desventaja**: la vista no es actualizable, por lo que cualquier cambio se deberá hacer en la tabla original.
- **Ventaja**: los bloqueos se liberan antes, ya que la consulta de la vista se hace a partir de la tabla temporal. Esto permite que otros *threads* accedan antes a la tabla que ejecutando una consulta mucho más pesada usando el algoritmo *MERGE*.

DEFINER permite determinar quién la cuenta asociada a la vista

SQL_SECURITY establece con que permisos se ejecutará la vista cuando se haga referencia a la misma pudiéndose elegir entre el que la definió (*DEFINER*) y el que la invoca (*INVOKER*).

WITH CHECK OPTION sirve para evitar modificaciones de filas que no están afectadas por la consulta que define la vista. Con *LOCAL* y *CASCADED* valor por defecto, se establece el alcance de esta comprobación, si es solo en la vista actual (*LOCAL*) o también se extiende a otras vistas incluidas en la actual (*CASCADED*).

Modificación, *ALTER VIEW*

```
ALTER [ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]
    VIEW nombre_vista [(columnas)]
    AS sentencia_select
    [WITH [CASCADED | LOCAL] CHECK OPTION]
```

Tanto con la cláusula *REPLACE* en el comando *CREATE VIEW* como con *ALTER* podemos modificar una vista existente.

Eliminación, *DROP VIEW*

El comando que elimina vistas es el siguiente:

```
DROP VIEW [IF EXISTS]
    nombre_vista [, nombre_vista] ...
    [RESTRICT | CASCADE]
```

Consulta de vistas

Para la obtención de información de definición de una vista usamos *SHOW CREATE VIEW*.

También podemos acceder directamente a la tabla *VIEWS* de *IINFORMATION_SCHEMA*. Por ejemplo para ver información de una vista en la base de datos test usamos lo siguiente:

```
SELECT VIEW_DEFINITION FROM INFORMATION_SCHEMA.VIEWS
WHERE TABLE_SCHEMA ='test';
```

ACTIVIDADES 4.6

- Crea una vista con el saldo total de las cuentas del banco.
- Usa una vista para mostrar los datos de cada jugador incluidos los de su equipo.
- Intenta modificar la vista anterior con otro usuario, haz lo necesario para permitirlo.
- Crea una vista que muestre para cada cuenta, el saldo y los datos del titular.
- Crea una vista no actualizable que muestre el título de la noticia y fecha de las noticias en la base de datos *motorblog*. Intenta sumar 1 día a las fechas de publicación para comprobar que es no actualizable.
- Crea una vista con las noticias del año actual. Intenta modificar el año en una de ellas. Repítelo añadiendo la cláusula *WITH CHECK OPTION* y explica el resultado.

4.5 EVENTOS

En MySQL los eventos son tareas que se ejecutan de acuerdo a un horario. Por lo tanto, a veces nos referiremos a ellos como los eventos programados.

Conceptualmente, esto es similar a la idea del programa *cron* de *Linux* o el Programador de tareas (comando AT) de Windows.

También se conocen como *triggers* temporales ya que conceptualmente son similares diferenciándose en que el *trigger* se activa por un evento sobre la abse de datos mientras que el evento según una marca de tiempo.

Un evento se identifica por su nombre y el esquema o base de datos al que se le asigna. Lleva a cabo una acción específica de acuerdo a un horario. Esta acción consiste en una o varias instrucciones SQL dentro de un bloque *BEGIN/END*.

Distinguiremos dos tipos de eventos, los que se programan para una única ocasión y los que ocurren periódicamente cada cierto tiempo.

La variable *global event_scheduler* determina si el programador de eventos está habilitado y en ejecución en el servidor. Esta variable puede tomar los valores ON para activarlo, OFF para desactivarlo y DISABLED si queremos imposibilitar la activación (ponerla a ON) en tiempo de ejecución.

Cuando el Programador de eventos (*Scheduler*) se detiene (variable *global_event_scheduler* está en *OFF*), puede ser iniciado por establecer el valor de *global_event_scheduler* en *ON* (véase el punto siguiente).

Observando la salida del siguiente comando podemos comprobar que el programador de eventos está activo ya que se ejecuta como un hilo más del servidor:

```
mysql> SHOW PROCESSLIST \G
```

Si *global_event_scheduler* no se ha establecido en *DISABLED* podemos activar el programador con siguiente comando:

```
SET GLOBAL event_scheduler = ON;
```

4.5.1 GESTIÓN EVENTOS

Los comandos para la gestión de eventos son *CREATE EVENT*, *ALTER EVENT*, *SHOW EVENT* y *DROP EVENT*.

Creación eventos

Un evento se define mediante la instrucción *CREATE EVENT*.

```
CREATE
[DEFINER = { user | CURRENT_USER }]
EVENT
[IF NOT EXISTS]
event_name
ON SCHEDULE schedule
[ON COMPLETION [NOT] PRESERVE]
[ENABLE | DISABLE | DISABLE ON SLAVE]
[COMMENT 'comment']
DO event_body;
schedule:
AT timestamp [+ INTERVAL interval] ...
| EVERY interval
[STARTS timestamp [+ INTERVAL interval] ...]
[ENDS timestamp [+ INTERVAL interval] ...]
interval:
quantity {YEAR | QUARTER | MONTH | DAY | HOUR | MINUTE |
WEEK | SECOND | YEAR_MONTH | DAY_HOUR | DAY_MINUTE |
DAY_SECOND | HOUR_MINUTE | HOUR_SECOND | MINUTE_SECOND}
```

Donde se crea un evento con un nombre asociado a una base de datos o esquema determinado:

- La cláusula *ON SCHEDULE* permite establecer cómo y cuándo se ejecutará el evento. Una sola vez, durante un intervalo, cada cierto tiempo o en una fecha hora de inicio y fin determinadas.
- *DEFINER* especifica el usuario cuyos permisos se tendrán en cuenta en la ejecución del evento.
- *event_body* es el contenido del evento que se va a ejecutar.
- Las cláusulas *COMPLETION* permiten mantener el evento aunque haya expirado mientras que *DISABLE* permite crear el evento en estado inactivo.
- *DISABLE ON SLAVE* sirve para indicar que el evento se creó en el *master* de una replicación y que, por tanto, no se ejecutará en el esclavo.

En el siguiente ejemplo de la base *ebanca* se bonifica con 100 euros a las cuentas dadas de alta en el intervalo de un mes:.

EJEMPLO 4.39

```
DELIMITER $$
CREATE EVENT bonificacion
ON SCHEDULE AT CURRENT_TIMESTAMP + INTERVAL 1 MONTH
DO
UPDATE ebanca.cliente SET saldo = saldo + 100 where fecha_creacion between now()
and date_add(now(),interval -1 month);$$
```

En este otro ejemplo cada mes se eliminan de la tabla noticias en la base *motorblog* las noticias fechadas hace más de 30 días. Previamente se almacenan en una tabla de de histórico. El evento comienza el 01-01-2012:

EJEMPLO 4.40

```
DELIMITER $$
CREATE EVENT archivo_noticias
ON SCHEDULE EVERY 1 MONTH
STARTS '2012-01-01 00:00:00' ENABLE
DO
BEGIN
INSERT INTO historico_noticias
SELECT * FROM noticias
WHERE fecha <=
DATE_SUB(CURRENT_DATE(), INTERVAL 30 DAY);
DELETE FROM noticias
WHERE fecha <=
DATE_SUB(CURRENT_DATE(), INTERVAL 30 DAY);
END;$$
```

Modificación eventos

Para modificar un evento usamos la orden *ALTER EVENT* con la siguiente sintaxis:

```
ALTER
[DEFINER = { user | CURRENT_USER }]
EVENT event_name
[ON SCHEDULE schedule]
[ON COMPLETION [NOT] PRESERVE]
[RENAME TO new_event_name]
[ENABLE | DISABLE | DISABLE ON SLAVE]
[COMMENT 'comment']
[DO event_body]
```

Consulta de eventos

Para la información asociada a un evento usamos *SHOW EVENT*

```
SHOW EVENTS [{FROM | IN} schema_name]
    [LIKE 'pattern' | WHERE expr]
```

Que muestra información de eventos asociados a un esquema o base de datos y filtrado según el patrón que determinemos o una cláusula *WHERE*.

Como siempre podemos recurrir al diccionario de datos consultando la tabla *INFORMATION_SCHEMA.EVENTS*.

Para finalizar esta sección incluimos una tabla resumen de los objetos estudiados en este tema.

Tabla 4.1 Resumen de objetos para automatización de tareas

PROCEDIMIENTOS	Pequeños programas que hacen tareas sencillas y bien definidas. Se llama con un comando CALL.
FUNCIONES	Similares a los procedimientos salvo que devuelven cero o un valor y todos los parámetros son de tipo IN.
VISTAS	Son partes determinadas de la base de datos en forma de tablas procedentes de consultas sobre las tablas originales.
TRIGGER	De tipo AFTER y BEFORE permiten desencadenar acciones ante modificaciones de las bases de datos.
EVENTOS	Sirven para realizar ciertas acciones en ciertos momentos temporales.
SCRIPTS	Son pequeños programas escritos en lenguajes como *perl*, *php*, *python* o *C* que permiten ampliar la funcionalidad de nuestro servidor.
Comandos para programación de tareas	Programas propios del sistema operativo. CRON para Unix, AT para Windows.

ACTIVIDADES 4.7

Crea eventos para lo siguiente:

- Crea un evento que cargue una comisión del 2% sobre las cuentas en números rojos cada primero de mes comenzando el 1 de enero de 2012.
- Crea un evento que registre diariamente los movimientos superiores a 1000 euros en una tabla *temp*. Créalo deshabilitado.
- Programa un evento que cuatro veces al año elimine los usuarios del blog que no publican hace más de tres meses (puedes crear un procedimiento que devuelva el número de noticias de un autor a partir de una fecha dada).
- Programa un análisis (*ANALYZE*) de las tablas de la base *liga* para el 1 de febrero de 2012.

4.6 CASO BASE

Los requisitos de la aplicación *mediaserver* hacen necesaria la creación de los siguientes objetos en la base de datos:

- Función: calcula el número de archivos reproducidos por un usuario.
- Función: calcula el tiempo de uso de un archivo.
- Función: calcula el porcentaje de uso de un archivo
- *Trigger*: al eliminar un usuario darlo de alta en la tabla de histórico con los campos *id_historico, fecha_alta, fecha_baja e id_usuario*.
- *Trigger*: cada vez que un usuario reproduzca más de la mitad un archivo actualizar el campo *num_archivos_vistos*
- *Trigger*: cada vez que se modifique el contenido de un archivo (según la función md5) guardar el anterior en la tabla *archivos_modificados(id_archivo, fecha_hora)*
- Procedimiento: muestra los datos de los archivos con mayor tiempo de reproducción. El número de archivos es introducido como parámetro.
- Procedimiento: elimina los archivos modificados hace más de un mes respecto de la fecha introducida como parámetro.
- Procedimiento: borra reproducciones que hayan durado menos de cierto porcentaje de su tiempo total.
- Evento: cada semana vuelca las estadísticas de uso de los usuarios en una tabla de histórico. Estos datos incluyen número de reproducciones de cada archivo según el mes, tiempo medio de reproducción en cada mes.
- Evento: cada año se eliminan los archivos con porcentaje de uso menor de 0.001%.
- Evento: cada mes elimina archivos modificados en el mes anterior.
- Vistas: con los datos de autor, *nombre_archivo* y tipo de la tabla archivos.

RESUMEN DEL CAPÍTULO

Hemos estudiado formas de automatizar tareas. Herramientas que para el administrador son vitales en tanto en cuanto le confieren toda la potencia necesaria para adecuar sus tareas a sus necesidades de su entorno productivo u organizativo.

Desde las rutinas que amplían la ya extensa funcionalidad del sistema gestor hasta los eventos que permiten programar tareas en periodos de tiempo predeterminados pasando por los *trigger* que también se ejecutan ante ciertos eventos relacionados con operaciones sobre las bases de datos. Por último, las vistas permiten filtrar contenido a los usuarios del tercer nivel aumentando así la seguridad y evitando errores.

EJERCICIOS PROPUESTOS

- **1.** Haz lo necesario para poder saber el mes y año con mejor saldo total en la base *ebanca*.
- **2.** En la base *motorblog* haz un procedimiento que ponga en negrita cada ocurrencia de una palabra en el contenido de una noticia. Los parámetros son 0 para aplicarlo a todas las noticias o el *id* de la noticia.
- **3.** Crea un procedimiento que encripte una cadena de caracteres cambiando cada letra por la siguiente, por ejemplo la *a* sería la *b* y así sucesivamente (usa las funciones ASCII y CHAR)
- **4.** Haz lo necesario sobre la base de datos *ebanca* para alimentar la tabla *num_rojos* con datos de las cuentas que han tenido números rojos en algún momento del año.
- **5.** Crea una función que devuelva 1 ó 0 si una frase es o no palíndroma.
- **6.** Crea una vista para la cuenta limitada que permita ver el número de noticias por día. Asigna los permisos necesarios a dicha cuenta. ¿Es actualizable la vista?
- **7.** Haz lo necesario en la base de datos liga para registrar los puntos metidos por cada equipo cada mes.
- **8.** Crea un evento que recoja cada 3 horas el estado del servidor en cuanto al número de consultas realizadas y conexiones creadas. Los valores deben almacenarse en la tabla *test.estado_servidor*(*idestado*, *numero_consultas*, *numero_conexiones*).

TEST DE CONOCIMIENTOS

1 Un *trigger*:
- a) Se dispara cada vez que un usario accede al sistema.
- b) Es una función para copias de seguridad.
- c) Es una función que se ejecuta cuando hay un cambio en una tabla.
- d) Ninguna de las anteriores.

2 Diferencia entre procedimiento y función:
- a) Las funciones siempre devuelven un valor.
- b) Las funciones solo devuelven un valor como máximo.
- c) Los procedimientos no permiten variables *out*.
- d) Las funciones no pueden acceder a tablas.

3 Las variables de sesión:
- a) Van precedidas por una arroba.
- b) Desparecen al cerrar la sesión.
- c) Se reinician al cerrar la sesión.
- d) Permanecen asociadas a cada usuario siempre.

4 Una variable de entrada/salida:
- a) Es actualizable.
- b) Es de solo lectura.
- c) Entra y sale.
- d) No debe modificarse.

5 Las vistas:
- a) Facilitan la visualización de las tablas.
- b) Son trozos de tabla.
- c) Permiten acceder a partes de una tabla.
- d) Es lo que usa la caché de MySQL.

6 Un evento:
- a) Es lo que ocurre cada vez que un usario accede al sevidor.
- b) Se registra en cada consulta.
- c) Ocurre a una hora dada.
- d) Ocurre con cierta frecuencia.

7 Un *trigger* de tipo *before*:
- a) Actua solo antes de una actualización.
- b) No tiene sentido en borrados.
- c) No tiene sentido actualizaciones.
- d) Solo se usa en consultas.

8 La palabra reservada OLD:
- a) Permite guardar datos borrados.
- b) Se usa en *triggers*, especialmente para inserciones.
- c) Permite referisnos a campos que van a modificarse.
- d) Es muy útil en consultas.

9 Una transacción en el contexto de MySQL:
- a) Debe incluir un *commit* y un *rollback*.
- b) Describe una operación crítica.
- c) Sirve para controlar errores.
- d) Permite asegurar una serie de operaciones críticas sobre la base de datos.

10 ¿Cómo accedemos al código de una rutina?
- a) Consultando la base de datos MySQL.
- b) Consultado la base de datos *information_schema*.
- c) Es imposible.
- d) Se guardan compiladas.

5 Optimización y monitorización

OBJETIVOS DEL CAPÍTULO

- ✓ Conocer los conceptos relacionados con la indexación.
- ✓ Conocer los tipos de tablas en MySQL.
- ✓ Aprender a gestionar índices.
- ✓ Optimizar consultas.
- ✓ Saber parametrizar y optimizar el funcionamiento del servidor para adaptarlo a cada contexto.
- ✓ Aprender a monitorizar el funcionamiento del servidor con distintas herramientas.

Cuando detectamos un funcionamiento incorrecto del servidor debemos tomar una serie de medidas que empiezan por determinar el origen del problema para lo cual disponemos de distintas herramientas. Lo primero es identificar el cuello de botella. Aunque en este capítulo nos centraremos en la optimización del servidor debemos señalar que los problemas no necesariamente provienen del nuestro servidor sino que pueden estar relacionados con aspectos ajenos al mismo como el estado de la red respecto al ancho de banda disponible o el propio sistema operativo.

Veremos en esta sección algunas de las medidas más recomendables para averiguar y solucionar problemas de rendimiento que afectan a nuestro servidor tanto desde el punto de vista de los comandos SQL que usamos como desde la configuración del servidor tomando en cuenta las variables de que dispone MySQL.

5.1 ÍNDICES

Un **índice** de una base de datos es una estructura de datos que mejora la velocidad de las operaciones, permitiendo un rápido acceso a los registros de una tabla. Al aumentar drásticamente la velocidad de acceso, se suelen crear sobre aquellos campos sobre los cuales se hagan frecuentes búsquedas.

Para comprender mejor lo que son y cómo funcionan podemos pensar en un ejemplo con una consulta sobre nuestra base de datos '*ebanca*'. Imaginemos que queremos obtener el detalle de un movimiento concreto en una cuenta de un cliente, algo como:

```
SELECT * FROM movimientos WHERE codigo_cuenta=2324
```

Para hacer la consulta MySQL debe ir leyendo uno a uno todos los registros de la tabla movimientos e ir seleccionando los que coincidan con el número de cuenta indicado. Considerando que puede haber millones de registros esto puede ser un trabajo muy poco eficiente.

Sin embargo, si tuviésemos ordenados los registros por el código de cuenta todo sería tan facil como localizar el número y capturar los datos. Para lograr esto se usan índices o campos indizados. Son lo mismo que un índice cualquiera de un libro. Permiten localizar rápidamente lo que me interesa y saber donde está para ir a buscarlo.

Los índices se crean sobre campos que suelen ser muy utilizados en consultas, para crearlo en este ejemplo usaríamos el comando:

```
ALTER TABLE movimientos ADD INDEX (codigo_cuenta)
```

De este modo le decimos al servidor que cree una lista ordenada con todos los números de cuenta en la tabla movimientos, es decir crea una nueva tabla donde se reflejan los números de cuenta y su posición dentro de la tabla tal como se hace en un índice normal de un libro.

Esto hace que sacrifiquemos algo de espacio de almacenamiento a costa de ahorrar CPU en el procesamiento de consultas.

Normalmente los índices se crean sobre campos que son también campos clave o parte de una clave primaria o secundaria ya que son campos sobre los que se suelen hacer consultas. Podemos decir que prácticamente toda clave va a ser un índice aunque lo contrario obviamente no es cierto.

Antes de profundizar en el uso de índices veremos conceptos importantes relacionados con los mismos.

5.1.1 TIPOS DE ÍNDICES

Parciales

Como hemos visto los índices intercambian espacio por optimización de consultas. Esto no siempre es lo mejor ya que depende del campo sobre el cual indicemos. Supongamos que hacemos un índice sobre el campo pcontenido en la base '*blog*'. Dicho campo tiene un tamaño de 255 caracteres o *bytes*, suponiendo que tenmos tres millones de noticias nos queda un espacio aproximado de 600 MB de espacio en disco extra. Si queremos ahorrar espacio a costa de disminuir un poco el rendimiento podemos hacer índices parciales con una sentencia como la siguiente:

```
ALTER TABLE posts ADDD INDEX (pcontenido(120))
```

Que reduciría el espacio ocupado a la mitad aproximadamente.

Multicolumna

Existe también la posibilidad de crear índices sobre más de una columna del siguiente modo:

```
ALTER TABLE posts ADDD INDEX (pcontenido(120), pfecha)
```

Lo que resulta especialmente útil cuando las consultas se suelen hacer sobre dichas columnas, es decir si se usan mucho en cláusulas *WHERE*.

Podría pensarse en usar un par de índices pero MySQL solo permite un índice por tabla y consulta. En ese caso debería elegir uno de los dos así que no serviría de mucho tener ambos.

Secundarios y cluster

Cuando se trata de tablas *MyISAM*, MySQL guarda los índices en archivos separados que contienen una lista de valores de índice y un valor que representa el *offset* del registro en la tabla. De este modo cualquier búsqueda requiere dos accesos a disco, uno para buscar en el índice y otro para localizar el registro.

Sin embargo, las tablas *InnoDB* usan los índices denominados *cluster*. En ellos se guardan las claves primarias junto con los propios registros ordenados con respecto a la clave primaria, así que las búsquedas por clave primaria son tremendamente rápidas. No ocurre igual para índices secundarios o no *clusterizados* que contienen los valores de clave primaria. Para estos la situación es la misma que en tablas *MyISAM*.

Hay situaciones en que conviene tener cuidado al usar índices *cluster*:

- Al usar varios índices secundarios, si la clave primaria es grande habrá muchas copias, una para cada índice secundario, suponiendo un derroche de espacio en disco.
- Al usar consultas de modificación de claves primarias:
  - Debe reorganizarse el *tablespace* o lugar en que guardamos los datos para reubicar el nuevo registro.
  - Actualizar los índices secundarios sobre ese registro.

Conviene pués elegir las claves primarias moderadamente pequeñas y sobre campos lo más estáticos posibles, es mejor usar el DNI a un número de teléfono, por ejemplo, para identificar a un cliente.

5.1.2 ESTRUCTURA DE UN ÍNDICE

Como ya hemos indicado, los índices son estructuras de datos. Describirlos llevaría otro libro así que simplemente damos un breve repaso de cómo son y donde se usan. A continuación veremos las diferentes estructuras de índices que soporta MySQL.

B-tree

Una de las estructuras más habituales en SGBD es la de **árbol** b o *b-tree*. Un árbol esta formado por un conjunto de nodos cada uno de los cuales contiene valores de índice ordenados que apuntan a registros de disco. Tienen la ventaja de la rapidez a la hora de recorrerlos buscando valores.

La idea tras los **árboles B** es que los nodos internos deben tener un número variable de nodos hijo dentro de un rango predefinido. Cuando se inserta o se elimina un dato de la estructura, la cantidad de nodos hijo varía dentro de un nodo. Para que siga manteniéndose el número de nodos dentro del rango predefinido, los nodos internos se juntan o se parten. Dado que se permite un rango variable de nodos hijo, los árboles B no necesitan *rebalancearse* tan frecuentemente como los árobles binarios de búsqueda *auto-balanceables*, pero por otro lado pueden desperdiciar memoria, porque los nodos no permanecen totalmente ocupados.

Son especialmente recomendables para consultas basadas en rangos de datos. Especialmente aquellos que usan cláusulas como *BETWEEN* y operadores de comparación.

Hash

Se basan en una función que mapea o hace corresponder valores de clave con valores numéricos de manera que se pueda asociar a cada valor de clave un número (normalmente único) que permita localizar al registro correspondiente. Éste número debe limitarse al máximo tamaño de almacenamiento para que el sistema sea viable.

Aunque tiene algunas contraindicaciones es un método muy rápido, incluso más que los *b-tree*, sin embargo es menos predecible y flexible ya que surgen otros problemas como la posibilidad (aunque remota) de colisiones (un mismo hash para distintas claves) o su lentitud para consultas basadas en rangos de valores.

R-tree

Son índices usados para datos de tipo espacial o n-dimanesionales como corordenadas x,y, z de un objeto. En el cada nodo contiene un número variable de entradas no superior a un máximo cada una de las cuales contiene un apuntador al nodo hijo y el llamado MBR (*Minimum Bounding Rectangle*) o superficie, volumen mínimo que alberga todos los puntos existentes en los nodos hijos.

Es decir MySQL indexa las diferentes formas que pueden ser representadas que son puntos, líneas y polígonos usando el rectángulo mínimo representable *mbr*. Para ello calcula el rectángulo más pequeño que puede realizar para que la forma quede contenida completamente. Se almacena las coordenadas del rectángulo usándolas para buscar las formas en un área dada.

Existen una serie de tipos de dato en MySQL que permiten almacenar valores espaciales como *geometry*, *point*, *linestring* o *polygon*. Para saber más puedes consultar el manual.

5.1.3 ÍNDICES EN MYSQL

Cuando creamos índices en MySQL podemos indicar no solo la estructura del índice (*hash* o *b-tree*) que queremos sino además el tipo. Entre ellos se encuentran los siguientes:

- *UNIQUE*: formados por campos cuyo valor no se repite en la tabla.
- *PRIMARY*: son índices sobre los campos que forman parte de la clave primaria de una tabla. Para este caso es mejor que sean numéricos y en caso de no saber cual elegir como clave conviene crear un identificador *autonumérico*.
- *full-text*: formados por uno o varios campos de texto y utilizados para la búsqueda de palabras dentro de un campo (típicamente para campos de gran tamaño) en funciones de búsqueda de cadenas (*full-text search*), solo para tablas de tipo *MyISAM* y en campos de tipo *CHAR*, *VARCHAR* y *TEXT*. Éste tipo de búsquedas se realizan con el comando SQL *MATCH...AGAINST* cuya sintaxis es:

  ```
  MATCH (col1,col2,...) AGAINST (expr [search_modifier])
  ```

  Dónde se buscan coincidencias de ciertas columnas con una cadena (*expr*). El modificador *search_modifier* permite establecer el tipo de búsqueda.

  Se usa sobre todo en consultas para buscar cadenas dentro de columnas de gran tamaño.
- *SPATIAL*: son índices usados para datos de tipo espacial como *LINE* o *CURVE*.

  Para el caso de columnas de tipo *CHAR*, *VARCHAR* y *TEXT* pueden crearse índices parciales que incluyan un número de caracteres para crear el índice (para el caso *TEXT* es obligatorio especificar dicho número). Lo mismo ocurre para los tipos *BINARY*, *VARBINARY* y *BLOB* (igual que con *TEXT* aquí es obligatorio indciar el número) cambiando caráteres por *bytes*.

El siguiente comando crea un índice con prefijo de 100 caracteres sobre la tabla noticias de *motorblog*:

EJEMPLO 5.1

```
CREATE INDEX icontenido on noticias(contenido(100));
```

Además de los índices es conveniente conocer los tipos de tablas o motores de almacenamiento en MySQL. Cada uno de ellos tiene sus propias características en cuanto al uso de índices que conviene conocer para un uso óptimo de los mismos.

MySQL permite principalmente dos motores de almacenamiento, *MyISAM* e *InnoDB*. Entre ellos la principal diferencia es si son transaccionales, es decir permiten el uso de las cláusulas *COMMIT* para confirmar un conjunto de comandos SQL y *ROLLBACK* para deshacerlos o invalidarlos de modo que las transacciones sean íntegras. Son motores ideales para sistemas con muchas modificaciones y lecturas concurrentes. Es el caso de *InnoDB*, que es actualmente el motor por defecto en el servidor MySQL.

Por otro lado el motor *MyISAM* es no transaccional lo que permite más rapidez y menos requisitos de memoria y almacenamiento a costa de una menor seguridad en las operaciones sobre los datos.

A continuación veremos las características de los motores más importantes en lo que respecta a los índices que usan.

Índices en tablas MyISAM

Cuando creamos una tabla de este tipo se crean tres ficheros con el mismo nombre y extensión *.frm* para la estructura de la tabla, *.MYD* para los datos y *.MYI* para sus índices.

Usan índices de tipo *b-tree* como estructura de índice permitiendo la compresión por prefijo para claves de tipo cadena como por ejemplo una *url* dónde el prefijo *http* es común a todos los valores clave.

También permite claves empaquetadas. Esta carácterística se configura con la opción *PACK_KEY=1* en el comando *CREATE TABLE* que hace compresión sobre números y cadenas. Si su valro es 0 evita cualquier compresión y si es *DEFAULT* comprime solo cadenas.

Otra característica es la escritura retardada controlada por la opción *delay_key_write* puesto a *ON* por defecto como variablede servidor en cuyo caso permite especificar si queremos diferir la escritura o modificación de índices hasta que se cierre la tabla lo que es muy útil para modificaciones masivas de datos.

Índices en tablas heap

Estas tablas se usan para trabajos masivos ya que se almacenan completamente en memoria. Pueden usar índices *b-tree* y *hash* (por defecto). Combinar la rapidez de este tipo de tablas y la flexibilidad de lso índices btree permite un gran rendimiento.

Índices en tablas BDB

Son tablas en las que los índices se guardan junto con los datos de las tablas y estos solo pueden ser de tipo *b-tree*.

Proporcionan compresión por prefijo, índices *cluster* y requieren una clave primaria como en *InnoDB*.

Índices en tablas InnoDB

Este tipo de tablas usan las denominadas *tablespaces* para el almacenamiento de datos e índices. Por defecto, si no se especifica otra cosa en las variables de configuración (ver tema 5 sobre optimización) cuando creamos tablas *InnoDB* se crean en el directorio de datos dos ficheros de datos *autoextensibles* de 10MB llamado *ibdata1* y dos ficheros de 5MB llamados *ib_logfile0* e *ib_logfile1* para los registros.

La definición de las tablas y columnas *InnoDB* se guardan igualmente en fichero con extensión *.frm*.

Los índices son almacenados junto con los datos dentro de estos ficheros de datos. Utiliza **índices cluster** explicados anteriormente.

En cuanto a la estructura de índice usan el tipo *b-tree* y no permiten empaquetado ni compresión.

No requieren claves primarias ya que es el servidor el que, si no las define el administrador, las crea internamente.

5.1.4 GESTIÓN DE ÍNDICES

Si surgen problemas de lentitud en ciertas consultas o si un índice da problemas en una tabla necesitamos saberlo. En particular datos como que columnas están indexadas, cuantos valores tienen o cuanto de grande es el índice son informaciones relevantes a la hora de optimizar el funcionamiento de nuestra base de datos.

Los comandos para la gestión de índices son:

Creación de índices

```
CREATE [UNIQUE|FULLTEXT|SPATIAL] INDEX index_name
    [index_type]
    ON tbl_name (index_col_name,...)
    [index_type]

index_col_name:
    col_name [(length)] [ASC | DESC]

index_type:
    USING {BTREE | HASH}
```

Donde indicamos:

- El tipo de índice: *UNIQUE, FULLTEXT* o *SPATIAL.*
- *index_name*: el nombre del índice.
- *index_type*: si es *btree* o *hash*.
- *index_col_name*: nombre de la columna sobre la que se crea el índice, tamaño y orden en que se almacena.

Mostrar los índices

```
SHOW {INDEX | INDEXES | KEYS}
    {FROM | IN} tbl_name
    [{FROM | IN} db_name]
    [WHERE expr]
```

Donde podemos usar indistintamente *INDEX*, *INDEXES* o *KEYS* para ver los índices de cierta tabla especificada por *FROM* o *IN*. Podemos incluir un filtro según los campos que devuelve el commando.

La salida contiene los siguientes campos:

- *Table*: nombre de la tabla que contiene el índice.
- *Non_unique*: si es (1) o no (0) único (*UNIQUE* o *PRIMARY*).
- *Key_name*: nombre del índice.
- *Seq_in_index*: si es parte de un índice multicolumna, ¿cuál es su número de orden dentro del índice? Esto es importante ya que los valores de índice se almacenan concatenados y ordenados según los valores de las columnas más a la izquierda.
- *Column_name*: el nombre del camp o columna usado para crear el índice.
- *Collation*: cómo está ordenado el índice. El valor A indica ascendente y *null* sin ordenar.
- *Cardinality*: estimación del número de valores distintos en el índice. Valor actualizado por el comando *analyze table* o el programa *myisamchk –a*.
- *Sub_part*: número de caracteres indexados si l acolumna no está totalemtne indexada. El valor es *null* si el índice es sobre la columna complete.
- *Packed*: cómo está empaquetada la columna. El valor es *null* si no hay empaquetamiento.
- *Null*: si la columna admite valores nulos.
- *Index_type*: estructura usad para el índice (*btree*, *hash*, *fulltext* o *rtree*).

Aplicando el comando sobre la base *ebanca*:

```
mysql>SHOW INDEXES FROM movimiento
*************************** 1. row ***************************
       Table: movimiento
  Non_unique: 0
    Key_name: PRIMARY
Seq_in_index: 1
 Column_name: fecha
   Collation: A
 Cardinality: 0
    Sub_part: NULL
      Packed: NULL
        Null:
  Index_type: BTREE
     Comment:
```

Este índice es único (clave principal, y tiene 0 elementos, no está empaquetado y es de tipo *b-tree*.

Eliminar índices

```
DROP INDEX index_name ON tbl_name
```


También se pueden gestionar los índices usando el comando *ALTER TABLE ADD INDEX o ALTER TABLE DROP INDEX* como se indica en el manual de MySQL. Por ejemplo para añadir un índice de tipo *b-tree* a nuestra tabla cliente usaríamos algo como en el ejemplo 5.2.

EJEMPLO 5.2

```
ALTER TABLE cliente ADD INDEX nombre USING BTREE
```

Y para eliminarlo:

```
ALTER TABLE cliente DROP INDEX nomreindice
```

Los índices no pueden actualizarse directamente, sino que deben ser eliminados y después añadidos.

Es conveniente obtener ciertas estadísticas de índices sobre todo porque a lo largo del tiempo las tablas que cambian mucho generan ineficiencias en sus índices como son la fragmentación de bloques y las estadísticas incorrectas.

Para ello disponemos del comando *OPTIMIZE TABLE* que permite *defragmentar* una tabla así como actualizar y reordenar índices. Es importante sobre todo cuando manejamos tablas que sufren muchos cambions tanto en sus datos como en uss índices. Funciona solo sobre tablas tipo *MyIsam* y *Archive,* para tablas *InnoDB* el comando está mapeado, es decir, funciona pero usando los comandos *ALTER TABLE* y *ANALYZE TABLE*. Éste último se encarga

de actualizar los índices de una tabla con el fin de optimizar las consultas. Para tablas *InnoDB, MyISAM* y *archive* actualiza los índices añadiendo cierto nivel de compresión o empaquetamiento en los mismos.

Veremos más sobre esto en la sección de optimización, más adelante en este capítulo.

ACTIVIDADES 5.1

- Usa los todos los comandos que conozcas para ver información de los índices en la tabla cliente de la base de datos *ebanca* y justifica su existencia.
- Usa el comando *OPTIMIZE TABLE* dos veces seguidas sobre tablas *MyISAM* e *InnoDB* y explica el resultado. ¿Para qué se usa dicho comando?
- ¿Qué tablas de la base *ebanca* consideras importante optimizar?, ¿por qué deberías hacer esto y cuál es el tiempo que consideras óptimo?
- Crea un índice de tipo *fulltext* sobre los campos titulo y contenido de la tabla noticias. Haz un ejemplo de búsqueda de noticias relacionadas con motor.

5.2 OPTIMIZACIÓN EN MYSQL

Tarde o temprano en la vida de un administrador surge el problema de optimizar o mejorar consultas lentas o ineficientes, al menos cuando se trabaja con bases de datos de tamaño medio/grande o cuando el volumen de accesos empieza a ser considerable. La velocidad de respuesta del servidor ante consultas de sus usuarios es un aspecto crítico en nuestras aplicaciones.

En esta sección aprenderemos a optimizar nuestras aplicaciones desde el punto de vista de diseño físico (motores de creación y definición de bases de datos, tablas, índices, etc.), como de los comandos SQL de manipulación y control (consultas, inserciones, creación de usuarios etc.).

5.2.1 OPTIMIZACIÓN DEL DISEÑO DE BASES DE DATOS

Ya hemos visto qué podemos hacer para mejorar el rendimiento en la parte lógica de nuestras bases, es decir, en el uso de SQL. En esta sección veremos lo que debemos tener en cuenta en nuestros diseños para mejorar el desempeño de nuestro servidor.

Es importante hacer un diseño óptimo de nuestras bases de modo que sea más fácil construir aplicaciones óptimas así como aumentar su tamaño sin perjudicar el rendimiento.

Para ello se deben crear con la idea general de minimizar el espacio que ocupan en disco de modo que se reduzca el flujo de entrada salida en disco.

En este sentido debemos tomar en cuenta lo siguiente:

- Usar los tipos de datos menores posible siempre que se ajusten a nuestros requisitos. Por ejemplo un *MEDIUMINT* ocupa un 25% menos que un *INT*.
- Siempre que sea posible usaremos *NOT NULL* en la definición de nuestros campos ya que facilita el uso de índices y evita la comprobación en consultas en que se comprueba si cierto valor es nulo.

- Si usamos *COMPACT* o *COMPRESSED* para la opción *ROW_FORMAT* cuando creamos tablas *InnoDB* reduciremos espacio a costa de incrementar el uso de CPU en algunas operaciones. Para tablas MyISAM ya existentes podemos usar el programa *myisampack* para comprimirlas siempre que sean de solo lectura
- Para tablas *MyISAM* podemos indicar un *ROW_FORMAT FIXED* en la creación de tablas de forma que los datos se almacenen ocupando un tamaño fijo. Esto puede desperdiciar espacio pero aumenta la velocidad.
- El índice correspondiente a la clave primaria debe ser lo más pequeño posible para acelerar la búsqueda de datos en consultas especialmente en *InnoDB* cuyos índices secundarios contienen una copia del índice primario. Para el caso de índices texto es mejor crear el índice sobre los primeros caracteres de la misma.
- Debemos crear solo los índices adecuados según nuestras necesidades, ya que estos mejoran la búsqueda de datos pero ralentizan las operaciones de inserción y actualización.
- Si hacemos operaciones de búsqueda sobre varios campos es mejor crear un índice conjunto que uno por cada campo. En este caso es mejor que el primer campo del índice sea el más usado. Si en todas las consultas aparecen varios campos es mejor que el primero en el índice sea el que tenga más duplicados.
- Es conveniente que los campos iguales se declaren exactamente igual sobre todo si intervienen en combinaciones o *joins* de varias tablas.
- Podemos minimizar la redundancia usando identificadores numéricos en nuestras tablas evitando así repetir datos y facilitando la ocombinación de las mismas.
- Si la velocidad es importante, por ejemplo en aplicaciones de *data mining,* podemos añadir redundancia duplicando datos y agregando tablas con datos estadísticos.
- Para tipos no binarios podemos usar la opción *BINARY* para acelerar consultas que impliquen comparaciones ya que de ese modo la comparación se hace por *bytes* y no por caracteres.
- Cuando solo algunos de los campos de una tabla se consultan frecuentemente y si son campos de tipo cadena puede ser conveniente dividir la tabla en dos o más partes y usar un campo común para combinarlas cuando sea necesario.
- Como hemos visto en algunas consultas se usan tablas temporales que, dependiendo de su tamaño pueden almacenarse en memoria o disco. Para las primeras el tamaño se determina por la menor de las variables de sistema *tmp_table_size* y *max_heap_table_size*. Sus correspondientes variables de estado son *created_tmp_tables* y *created_tmp_disk_tables* para las tablas creadas en disco. Es conveniente consultar estos valores cuando realizamos consultas pesadas.
- Particionar tablas cuando sea necesario, tanto verticalmente dividiendo tablas con muchos accesos parciales (solo a ciertas columnas) como horizontalmente cuando las tablas contienen gran cantidad de datos fácilmente separables.

5.2.2 PROCESAMIENTO DE CONSULTAS

En esencia el proceso es muy simple, un usuario o aplicación acceden al servidor a traves de una interfaz (navegador, función de un lenguaje de programación, etc.) buscando cierta información. El servidor devuelve los datos siempre que se cumplan los requisitos de acceso y los permisos sean correctos.

El proceso normal de una consulta se divide en tres etapas, a saber: *parseo*, análisis y optimización, pero antes el sevidor consulta la caché para evitar repetir el proceso para cada *select*.

Caché de consultas

Cualquier *SELECT* que se quiera ejecutar en el servidor es buscado previamente en la caché. MySQL almacena cada consulta (salvo las que incluyan la opción *SQL_NO_CACHE*) en la caché usando un *hash* y siempre y cuando dicha opción esté activada con la variable de servidor *query_cache_type=ON* o 1. Otros posibles valores son *OFF* o 0 y 2 o *DEMAND*. En el caso *ON* se cachean todas las consultas salvo las que incluyen la opción *SQL_NO_CACHE*, en el caso *OFF* no se usa la caché (aunque se mantiene) para repetir consultas y en el caso *DEMAND* solo se cachean consultas con la opcinon *SQL_CACHE*.

Solo las consultas que se repiten muchas veces interesa que sean cacheadas. No obstante debemos ser conscientes de que no todas las consultas son cacheables, en particular no se cachean aquellas que forman parte de una subconsulta y las que figuran en el cuerpo de un procedimiento, función, *trigger* o evento.

Parseo, análisis y optimización

Una vez que no se encuentra una consulta en la caché debe ser procesada por el servidor. En primer lugar se *parsea* y analiza teniendo en cuenta aspectos como:

- El tipo de consulta: si es de tipo *INSERT*, *UPDATE*, *SELECT* o *DELETE* o un comando administrativo como *GRANT* o *SET*.
- Tablas involucradas.
- Contenido de la cláusula *WHERE* (si existe).
- Otros modificadores como *GROUP BY*, *ORDER BY*, etc.

De este modo la consulta se trocea en piezas más básicas que son tomadas por el optimizador que a su vez intenta obtener la información necesaria de la manera más óptima posible.

No es el objeto de este libro entrar en detalles sobre este tema asi que es suficiente que sepamos que detrás de una consulta hay un proceso complejo cuyo objeto es lograr que su procesamiento sea lo más rápido posible.

Para ello MySQL toma decisiones en función de la consulta las cuales están relacionadas con varios aspectos especialmente los relacionados con los índices creados sobre columnas de las tablas implicadas en las consultas.

5.2.3 OPTIMIZACIÓN DE CONSULTAS E ÍNDICES

Para optimizar consultas es imprescindible tener en cuenta dos aspectos fundamentales:

1. Cómo se ejecutan: cada fila de la primera tabla de una consulta se procesa comparándola con cada fila de las tablas subsiguientes en una subconsulta u siguiente tabla en una combinación.
2. Qué índices hay: si se ha creado más de un índice sobre un mismo campo el servidor debe elegir el más óptimo para lo cual en general se regirá por su cardinalidad o número de registros.
3. Como se almacenan los índices: como hemos visto en general los índices son archivos ordenados (por la columna o columnas) que contienen registros de la/s columna/s indizada/s junto con la dirección física del registro con los datos de la tabla correspondiente. Si el índice contiene varias columnas el orden es según las columnas más a la izquierda en la definición del índice.

La mejor forma de mejorar el rendimiento de consultas es crear índices sobre los campos más consultados o usados en consultas. Sin embargo crear demasiados índices también puede suponer un derroche de espacio y perjudicar al rendimiento al tener que buscar los índices adecuados en cada consulta y mantenerlos sincronizados con los datos. Como siempre debemos buscar una situación de compromiso.

Si una consulta afecta solamente a campos indexados, por ejemplo los que forman parte de una clave primaria, no es necesario acceder a los datos de la tabla sino que podemos usar directamnte el archivo de índices para obtener el resultado. Por ejemplo la consulta que devuelve los títulos de todas las noticias en la base de atos motorblog es mucho más rápida si hay un índice sobre el campo título ya que de este modo el servidor no necesita acceder a los datos directamente en el fichero de datos sino simplemente usando el fichero de índices. En este concepto reside la clave (nunca mejor dicho) de la optimización de consultas.

Los índices son especialmente importantes en las siguientes operaciones:

- ✓ Consultas con cláusulas *WHERE* que contienen columnas indexadas. De este modo se puede hacer un filtro previo usando únicamente el archivo de índice.
- ✓ Para descartar filas en consultas ya que si hay más de un índice sobre una misma columna el servidor elegirá el más restrictivo, o sea el que menos filas contenga.
- ✓ En combinaciones de tablas cuando existe un índice sobre los campos comunes.
- ✓ Para encontrar el valor de una función de agregado sobre campos indexados sin necesidad de acceder a los registros en disco.
- ✓ Para ordenar o agrupar campos indizados de tablas siempre que se haga sobre la parte más a la izquierda del índice (o los primeros campos del mismo).
- ✓ Para casos en que solo se requieren columnas indizadas no se precisará acceder a los datos de la tabla. Sin embargo para tablas pequeñas o tablas sobre las que se accede a gran parte de sus campos el uso de índices puede ser ineficiente.

MySQL soporta índices multicolumna de hasta 16 campos. Son índices que incluye más de un campo. Estos se almacenan internamente como una conjunto ordenado de valores resultado de concatenar los valores de los campos implicados.

De este modo cuando hagamos una consulta que use estos campos MySQL solo usará los índices si el conjunto de valores a buscar pertenece a un rango conocido para las columnas de la izquierda y opcionalmente para las de la derecha (según su orden en la definición del índice).

Por ejemplo si tenemos un índice sobre los campos título y fecha de la tabla noticias en la base *motorblog*, en las siguientes consultas sí se usará:

EJEMPLO 5.3

```
SELECT * FROM noticias WHERE titulo='hidrógeno';

SELECT * FROM noticias
  WHERE titulo='hidrógeno' AND fecha='2011-10-10';

SELECT * FROM noticias
  WHERE titulo='hidrógeno'
  AND (fecha='2012-10-10' OR fecha='2011-10-10');

SELECT * FROM noticias
  WHERE titulo='hidrógeno' AND fecha >='2011-10-10';
```

Esto es asi porque el optimizador sabe exactamente que filas de índice seleccionar (recordemos que los índices se almacenan ordenados).

Sin embargo en los siguientes ejemplos no se usan índices ya que en estos casos se necesita recorrer todo el archivo de índices para comprobar la condición *WHERE*:

EJEMPLO 5.4

```
SELECT * FROM noticias WHERE fecha='2011-10-10';

SELECT * FROM noticias
  WHERE titulo='hidrógeno' OR fecha='2011-10-10';
```

Aunque estos consejos pueden ser útiles para tenerlos en cuanta a la hora de diseñar nuestra base de datos, lo mejor para ajusta rlos índices adecuados a nuestras necesidades es hacer uso de la sentencia *EXPLAIN* que nos permite conocer cual es el plan que MySQL seguirá cuando realice ciertas consultas.

5.2.3.1 Comando EXPLAIN

Para la optimización de consultas la herramienta más típicamente utilizadas es *EXPLAIN*. Es un comando que permite obtener información sobre cómo se llevarán a cabo las consultas dependiendo de nuestras tablas, columnas y de las condiciones en nuestras cláusulas *WHERE*.

En su uso más simple devuelve la descripción de una tabla: *EXPLAIN* tabla igual que lo haría el comando *DESCRIBE* tabla o *SHOW COLUMNS FROM* tabla. Sin embargo en conjunción con *SELECT* lo que devuelve es la estrategia de ejecución de la consulta. Esta es su sintaxis:

```
EXPLAIN [EXTENDED | PARTITIONS] SELECT select_options
```

- *EXTENDED*: nos permite obtener información adicional mientras que *PARTITIONS* nos da información para tablas particionadas.
- *select_options*: es la consulta propiamente dicha.

El uso de *EXPLAIN* permite detectar cuando un índice se usa o no, si se usa incorrectamente o ver si las consultas se ejecutan de forma óptima. De ese modo el administrador puede corregirlas y optimizarlas.

Por ejemplo, si queremos resultado de *EXPLAIN* en la consulta que obtiene los datos de un enlace de la tabla enlaces en *motorblog*:

EJEMPLO 5.5

```
EXPLAIN EXTENDED SELECT * FROM wp_links WHERE link_id=24\G
```

Es una consulta sencilla en la que el campo de búsqueda es un índice (*PRIMARY KEY*), así que MySQL solo tiene que encontrar la primera coincidencia en el índice, acceder a la fila correspondiente y devolver el resultado.

El resultado sería el siguiente:

```
id: 1
select_type: SIMPLE
table: wp_links
type: const
possible_keys: PRIMARY
key: PRIMARY
key_len: 8
ref: const
rows: 1
Extra:
```

- *id*: es el número de tabla en la consulta. En este caso solo hay una.
- *select_type*: es el rol de la tabla en la consulta pudiendo adquirir, entre otros, los siguientes valores:
  - *SIMPLE*: para consultas que no sean uniones o incluyan subconsultas.
  - *UNION*: haciendo referencia a la segunda o última consulta en una consulta de unión.
  - *DEPENDENT UNION*: lo mismo que la anterior pero cuando la última consulta depende de otras.
  - *UNION RESULT*: es el resultado de una unión.
  - *SUBQUERY*: hace referencia a la primer SELECT en una subconsultas.
  - *DEPENDENT SUBQUERY*: igual que la anterior pero cuando la consulta es dependiente de otra externa a ella. La palabra *DEPENDENT* nos informa sobre consultas correlacionadas.
  - *PRIMARY*: haciendo referencia a la primera consulta cuando hay subconsultas.
  - *DERIVED*: hace refencia a subconsultas dentro de cláusulas *FROM*.
- *Table*: es el nombre de la tabla de la que se extraen los registros.
- *Type*: indica qué tipo de valores se usan en la consulta. Valores posibles son *const* para constante, *ref*, *range*, *index* y *ALL*.
- *Possible_keys*: posibles índices que pueden usarse para buscar registros.
- *Key*: el nombre del índice que MySQL ha decidido usar de entre todos los posibles.
- *Ref*: son las columnas o valores que se usan para hacer coincidir con la clave. Se utiliza cuando usamos parte del índice de una tabla.
- *Eq_ref*: se usa cuando se utiliza todo el índice primario o único de una tabla.
- *Rows*: indica el número de filas que MySQL piensa que debe examinar antes de devolver los resultados. Es solo una estimación que no necesariamente coincide con el número de filas devueltas.
- *Extra*: es para información adicional, como el uso de ciertas cláusulas o el uso de índices para obtener el resultado de la consulta (sin necesidad de acceder a los registros de la tabla y, por tanto, mucho más rápido).

Veremos a continuación cómo podemos optimizar distintos tipos de consultas usando el comando *EXPLAIN*:

Forzar índices

Cuando las tablas son pequeñas o simplemente si MySQL no lo considera apropiado puede ocurrir que no se usen índices. Es algo que podemos evitar con la siguiente cláusula en consultas *SELECT* después de la cláusula *FROM*.

```
USE|FORCE|IGNORE {INDEX|KEY}
      [{FOR {JOIN|ORDER BY|GROUP BY}] ([index_list])
```

Con *USE* indicamos a MySQL que use únicamente el índice especificado, si usamos *IGNORE* evitamos el uso del índice mientras que *FORCE* indica que se use el índice siempre que se pueda para resolver la consulta ya que acceder a la tabla en disco es con toda seguridad muy costoso en tiempo. El uso de FOR indica al servidor que use el índice solo para consultas específicas de combinación, de ordenación o agrupación.

En el siguiente ejemplo forzamos el índice *fecha* para la obtención de los datos de noticias:

EJEMPLO 5.6

```
select * from wp_posts force index (fecha) order by fecha;
```

Optimizando consultas simples con WHERE

Para consultas con *WHERE* el servidor usa el tipo range indicando que la consulta require un cierto número de registros resultado de una serie de condiciones sobre algunos campos. Si esos campos están indexados veremos el valor range para el campo type en la salida de *EXPLAIN*. En el siguiente ejemplo obtenemos los datos de las fuentes cuyo identificador está entre 10 y 20. Primero lo hacemos con la tabla sin indexar:

EJEMPLO 5.7

```
EXPLAIN SELECT * FROM fuentes WHERE fuente_id between 10 AND 20;

id: 1
select_type: SIMPLE
table: wp_links
type: ALL
possible_keys: NULL
key: NULL
key_len: NULL
ref: NULL
rows: 71
Extra: Using where
```

Como vemos l a consulta es simple, debe recorrer todas las filas de la table Fuentes y usa un filtro WHERE. Obviamente no hay índices implicados, así que es muy ineficiente.

Si ahora indexamos con una clave primaria:

```
ALTER TABLE Fuentes ADD PRIMARY KEY cp(link_id)
Obtenemos el siguiente resultado:

id: 1
select_type: SIMPLE
table: wp_links
type: range
possible_keys: PRIMARY
```

```
key: PRIMARY
key_len: 4
ref: NULL
rows: 1
Extra: Using where
```

Vemos que ahora se indica *range* como *type* (es decir, va a usar el archivo de índice para buscar las filas necesarioas) y *rows* = 1, es decir el servidor estima que necesitará acceder aproximadamente a 1 filas para poder realizar la consulta. También se añade información extra indicando que hay un filtro *WHERE*. Si en lugar de mostrar todos los campos buscásemos solo uno como por ejemplo *fuente_id* el resultado sería similar pero en el campo *Extra* tendríamos también '*Using index*' indicando que puede resolver la consulta usando solamente el índice sin necesidad de leer la tabla en disco lo cual es deseable.

Puede ocurrir que el índice sea compuesto de varias partes o columnas. En este caso el modo de ejecución dependerá del orden que usemos en la creación del índice y las condiciones *WHERE*.

En el siguiente ejemplo obtenemos los datos de fuentes con un identificador mayor que 10 y fecha superior a enero de 2011. Previamente creamos un índice sobre los campos *fuente_id* y *fuente_fecha*:

```
CREATE UNIQUE INDEX inff(fuente_id, fuente_fecha) ON fuentes;
EXPLAIN SELECT * FROM fuentes WHERE fuente_id>10 AND fuente_fecha> '2011-01-01';
```

La salida de *EXPLAIN* muestra ahora una consulta de tipo *range* que requerirá aproximadamente acceder a unas 27 filas en disco a través del uso de los índices correspondientes.

Sin embargo, si recordamos cómo se organizan los índices vemos que la misma consulta usando un *OR* no tendrá el mismo resultado ya que entonces el servidor no tiene modo de eliminar posibles filas usando el índice. Es decir, prevé que *a priori* cualquier fila del índice puede cumplir la condición. En el caso anterior al estar el índice ordenado por el campo *fuente_id* y haber una condición *AND* es capaz de calcular usando el índice el número de filas a leer en disco.

En general si observamos que una consulta se realiza muchas veces y no es sobre un índice podemos pensar en indexarla para mejorar el rendimiento. Sin embargo no debemos olvidar que indexar no es necesariamente lo mejor y que cada caso merece un estudio previo.

Optimizando consultas con *ORDER BY* y *GOUP BY*

En muchas consultas se incluyen cláusulas *ORDER BY* lo que obliga a hacer una ordenación sobre uno o varios campos. El proceso de ordenación es costoso y conviene evitarlo en la medida de lo posible. Su uso se reflehja en la salida del comando *EXPLAIN* en el campo Extra cuando toma el valor *Using filesort*. Si los campos de ordenación están indizados la consulta se hará mucho más rápidamente. Pero no siempre será asi ya que debe cumplirse que la consulta se haga usando el mismo orden de los campos (o al menos los campos definidos en los primeros lugares) y que la ordenación no incluya campos de más de una clave. Por ejemplo, con el siguiente comando obtenemos el plan de ejecución de la consulta que obtiene los datos de los equipos ordenados por ciudad:

EJEMPLO 5.8

```
mysql>explain select * from equipo order by ciudad;
```

En el campo *Extra* de *EXPLAIN* veremos el valor:

```
Extra:Using filesort
```

Lo que indica la necesidad de ordenación. Como hemos señalado para evitarlo los campos de ordenación deben ser todos ellos parte de un índice y la ordenación debe ser en el mismo orden. En el siguiente ejemplo sobre la base del *motorblog* podemos verlo:

```
mysql>explain select * from wp_posts order by id;
```

El resultado no incluirá el valor *using filesort* anterior para el campo *Extra* lo que demuestra el uso de índices para la consulta.

Cuando usamos índices múltiples formados por más de un campo debemos tener en cuenta que en MySQL se almacenan en orden ascendente o descendente. Por ejemplo el siguiente comando:

EJEMPLO 5.9

```
mysql>explain select local from partido order by local,visitante asc;
```

Devolverá en su campo *Extra* el valor '*Using index*' dado que solo necesita recurrir al archivo de índice para recuperar el resultado. Sin embargo si cambiamos el *asc* a *desc* (en MySQL todos los índices se crean ordenados ascendentemente) obtenemos además la opción '*Using filesort*' indicando la necesidad de crear una tabla temporal para ordenar.

Los índices formados por más de un campo también requieren un *filesort* sí se consultan en orden distinto, por ejemplo:

EJEMPLO 5.10

```
mysql>EXPLAIN SELECT local FROM partido ORDER BY visitante, local asc;
```

Requerirá indefectiblemente un *filesort*, ya que el índice formado por los campos local y visitante se almacenan en ese orden de forma que si se acceden en orden distinto debe usarse una tabla temporal para hacer la ordenación. Esto se generaliza para índices de más de dos campos. Siempre lo ideal es usar la parte más a la izquierda del

Cuando tenemos además cláusulas *GROUP BY* puede ser que se requiera la creación de tablas temporales lo que se especifica con el valor '*Using temporary*' en el campo Extra de la salida de *EXPLAIN*. Esto ocurre sobre todo si usamos campos distintos en *GROUP BY* y *ORDER BY*.

En el siguiente ejemplo obtenemos el número de partidos de cada equipo como local:

EJEMPLO 5.11

```
mysql>EXPLAIN SELECT count(*) FROM partido GROUP BY local;
```

En este caso obtenemos en el campo *Extra* el valor *Using index* indicando que la consulta se puede realizar solamente usando el archivo de índices.

Sin embargo, si lo que queremos es el número de partidos de cada equipo como visitante, es decir:

```
mysql>EXPLAIN SELECT count(*) FROM partido GROUP BY visitante;
```

En este caso el campo *Extra* contendrá el valor *Using index*, *Using temporary*, *Using filesort* indicando la necesidad del índice, de una tabla temporal y de una ordenación para realizar la consulta. Esto es así por la forma en que se almacena el índice ya que esta ordenado por local y no por visitante.

Otro ejemplo de índices multicolumna puede ser el siguiente. Sobre la base de datos *motorblog*, en la table noticias creamos un índice sobre los campos *id* y autor en este orden. Si quisieramos hacer la siguiente consulta:

```
SELECT autor from noticias WHERE id in (1,2,4,5,10);
```

Veríamos en el campo Extra el valor *Using where*; *Using index* indicando que la consulta se resuelve usando únicamente el índice. Sin embargo todavía deben revisarse todas las filas del índice. Esto es así porque el índice está ordenado por autor, id y entonces se deben comprobar todas las filas porque los valores de búsqueda no son los de ordenación. Para mejorar esta consulta una solución puede ser crear el índice al revés, es decir, sobre los campos id y autor. Al hacerlo el resultado es que en el campo *rows* de *EXPLAIN* indica el valor 5 lo que obviamente mejora considerablemente la consulta.

Sin embargo el resultado sería el mismo si hubiésemos indizado solamente el campo *id*. Aunque en este caso hubiésemos necesitado accede a disco por no ser el campo autor parte del índice.

Para casos en que usamos agrupaciones como por ejemplo la siguiente consulta que obtiene el número de noticias de cada autor:

```
SELECT autor, count(id) FROM noticias GROUP BY autor;
```

Si usamos EXPLAIN vemos como se require una tabal temporarl (*Using temporary*) además de una nueva lectura de la table ordenada (*Using filesort*). Podemos evitar esto último agregando la cláusula *ORDER BY null* al final de la consulta.

Optimizando consultas de más de un índice

Si una consulta puede usar dos índices distintos MySQL intentará ejecutar la que requiera el uso de menos filas. Para ello podemos ejecutar la sentencia *SHOW INDEX FROM nombre_tabla* y ver la cardinalidad. MySQL elegirá, salvo que le indiquemos lo contrario, aquella que tenga la cardinalidad menor. Por ejemplo:

EJEMPLO 5.12

```
mysql> SELECT titulo FROM noticias WHERE post_date>'2010-02-02' AND id>2000';
```

Al haber índices sobre ambos campos MySQL debe elegir el más óptimo que será el que menos filas requiera para realizar la consulta. Si usamos *EXPLAIN* obtenemos que en este caso usará el índice *post_date* mientras que si subimos de 2000 a 20000 en la condición de *id* se decantará por el índice sobre *id*.

Optimizando consultas con operaciones

La consulta más sencilla que incluye la cláusula *WHERE* tiene que ver con cálculos matemáticos o de cadenas. Así la siguiente consulta:

EJEMPLO 5.13

```
mysql> SELECT * FROM ebanca.cuenta WHERE saldo/2< 20 ;
```

Debemos evitar operaciones con los campos ya que en este caso se debe hacer la división con cada valor mientras que en este otro:

EJEMPLO 5.14

```
mysql> SELECT * FROM ebanca.cuenta WHERE saldo< 20/2 ;
```

La operación se hace una sola vez y el resultado se usa para la comparación con los valores de la columna.

Optimizando consultas de más de una tabla

Cuando hay varias tablas implicadas en una consulta, ya sea porque se trata de subconsultas o de combinación de tablas, es importante considerar, entre otros, el campo *type* en la salida de *EXPLAIN*.

Los valores posibles son:

- *eq_ref*: se da cuando la clave ajena de la tabla secundaria es única en cuyo caso cada fila de la tabla principal se combina con una única fila de la secundaria.
- *ref*: Es el caso contrario, la clave ajena no tiene por que ser única de forma que cada fila de la tabla principal se combina con cada coincidencia en la secundaria.
- *unique_subquery*: se usa cuando la subconsulta se basa solo en campos clave de una tabla de forma que es una función la que hace al cálculo sin necesidad de acceder a la tabla original.
- *index_subquery*: similar a la anterior pero permitiendo índices no únicos.

También es necesario entender el funcionamiento de este tipo de consultas desde el punto de vista del servidor. En general para una consulta de combinación de dos o más tablas el proceso es el siguiente:

1. El servidor lee la primera fila de la primera tabla.
2. A continuación compara dicha fila con cada fila de la segunda tabla según las condiciones *WHERE* especificadas.
3. Para cada coincidencia repite el proceso con la tercera tabla. El proceso se repite para cada fila de la segunda tabla con la tercera y así sucesivamente hasta recorrer todas las tablas.

Según *EXPLAIN* cada fila (*row*) de una tabla es usada para encontrar coincidencias en la segunda, y así sucesivamente.

Veamos algunos ejemplos de consulta que combina la tabla noticias y autor suponiendo que hemos suprimido los índices en ambas. En ese caso al hacer la siguiente consulta de las noticias del autor con *id = 1*:

```
mysql>SELECT * FROM autor STRAIGHT_JOIN noticias ON noticias.autor=autor.nombre AND
autor.id=1\G
```

(Hemos usado un *STRAIGHT_JOIN* para asegurar que la combinación se realice en el orden que queremos para ilustrar major el uso de índices).

Obtenemos dos filas o *row*:

```
1. row ***************************
id: 1
select_type: SIMPLE
table: autor
type: ALL
possible_keys: NULL
key: NULL
key_len: NULL
ref: NULL
rows: 6
Extra: Using where

2. row ***************************
id: 1
select_type: SIMPLE
table: noticias
type: ALL
possible_keys: NULL
key: NULL
key_len: NULL
ref: NULL
rows: 3843
Extra: Using where
```

Es decir, debemos recorrer todas las tablas (debido al *type ALL*) para obtener la consulta lo que es absolutamente lógico al no haber índices creados.

Esto es ineficiente puesto que requiere recorrer 6 × 3843 filas en disco. En tablas de mayor tamaño o combinaciones de más tablas la cifra puede llegar a ser astronómica.

Para optimizar la consulta podemos crear índices en la columna autor de la tabla noticias. De ese modo el servidor podrá recurrir al índice para hacer las comprobaciones de coincidencias. Dado que puede haber varias (*type: ref*) debe hacer una estimación según sus estadísticas internas. En nuestro caso obtenemos lo siguiente:

```
id: 1
select_type: SIMPLE
table: p
type: ref
possible_keys: in1
```

```
key: in1
key_len: 8
ref: const
rows: 5
Extra:

2. row ***************************
id: 1
select_type: SIMPLE
table: u
type: ALL
possible_keys: NULL
key: NULL
key_len: NULL
ref: NULL
rows: 6
Extra: Using where
```

Quedando claro el ahorro en lecturas a disco ya que pasamos de leer 6 × 3843 filas a 5 × 6. (En este caso el servidor estima que habrá unas 5 noticias para el autor con id 1).

Pero aún podemos optimizar más ya que si nos fijamos la condición *WHERE* que filtra al autor con un *id* determinado puede indexarse también como clave primaria:

```
ALTER TABLE autor ADD PRIMARY KEY(id);
```

Al hacer *EXPLAIN* de nuevo obtenemos:

```
1. row ***************************
id: 1
select_type: SIMPLE
table: p
type: ref
possible_keys: in1
key: in1
key_len: 8
ref: const
rows: 5
Extra:
2. row ***************************
id: 1
select_type: SIMPLE
table: u
type: const
possible_keys: PRIMARY
key: PRIMARY
key_len: 4
ref: const
rows: 1
Extra:
```

De este modo la salida indica que necesitamos leer únicamente 8 × 1 filas en total. Al ser el índice único o de clave primaria el servidor solo necesitará leer, como máximo, una fila de la tabla de autores por cada fila de noticias.

El campo *type* anterior indica *const* ya que hemos usado un valor fijo cómo *id* de *autor*

En el caso anterior vimos el tipo *ref* que se establece por haber varias posibles coincidencias de noticias con cada fila de autor. Es decir cada autor puede tener varias noticias. Esto ocurre por que el índice sobre autor en la tabla de noticias no es único ni clave primaria. Si hubiésemos hecho la consulta de este modo:

```
mysql>SELECT * FROM noticias STRAGIHT_JOIN autor WHERE noticias.autor=autor.nombre\G
1. row ***************************
id: 1
select_type: SIMPLE
table: p
type: ALL
possible_keys: NULL
key: NULL
key_len: NULL
ref: NULL
rows: 3843
Extra:
2. row ***************************
id: 1
select_type: SIMPLE
table: u
type: eq_ref
possible_keys: PRIMARY
key: PRIMARY
key_len: 4
ref: motorblog.p.autor
rows: 1
Extra: Using where
```

Ahora encontramos *eq_ref* en el campo *type* de la salida de *EXPLAIN*. Esto es así porque para cada fila de la tabla noticias hay un solo posible autor.

Al final el administrador, según los requisitos de sus usuarios y aplicaciones debe decidir el orden correcto así como los índices necesarios para cada consulta.

En el siguiente ejemplo estudiaremos el uso de una subconsulta para saber los datos de la noticia con mayor *id*, de nuevo usaremos la tabla noticias sin índices:

EJEMPLO 5.15

```
mysql> SELECT * FROM noticias n1 WHERE id=(SELECT max(id) FROM noticias n2)';
```

El resultado de *EXPLAIN* sobre esta consulta sería el siguiente:

```
1. row ***************************
id: 1
select_type: PRIMARY
table: n1
type: ALL
possible_keys: NULL
key: NULL
key_len: NULL
ref: NULL
rows: 3843
Extra: Using where
2. row ***************************
id: 2
select_type: SUBQUERY
table: n2
type: ALL
possible_keys: NULL
key: NULL
key_len: NULL
ref: NULL
rows: 3843
Extra:
```

Solamente destacamos el hecho de usar un *WHERE* en la primera tabla mientras que la segunda se trata como de tipo *SUBQUERY*.

Sin embargo si creamos un índice único:

```
CREATE UNIQUE INDEX in1 ON noticias(id);
```

El resultado ahora de EXPLAIN:

```
1. row ***************************
id: 1
select_type: PRIMARY
table: p1
type: const
possible_keys: in1
key: in1
key_len: 4
ref: const
rows: 1
Extra:
2. row ***************************
Id: 2
select_type: SUBQUERY
table: NULL
type: NULL
possible_keys: NULL
key: NULL
key_len: NULL
```

```
ref: NULL
rows: NULL
Extra: Select tables optimized away
```

Se observa como en la segunda tabla no se tiene nada en cuenta al ser una consulta trivial, solo necesita usar la primera, de ahí el contenido de *Extra*. Es decir solamente usando el índice *id* puede realizar la consulta.

Ahora vemos qué ocurre para el caso de una consulta correlacionada en la que buscamos los datos de noticias cuyos autores se hayan registrado antes del año 2010:

```
EXPLAIN SELECT * FROM noticias n WHERE autor in(SELECT id FROM autor a WHERE a.id=n.
autor);

1. row ***************************
id: 1
select_type: PRIMARY
table: p
type: ALL
possible_keys: NULL
key: NULL
key_len: NULL
ref: NULL
rows: 3843
Extra: Using where

2. row ***************************
id: 2
select_type: DEPENDENT SUBQUERY
table: a
type: ALL
possible_keys: NULL
key: NULL
key_len: NULL
ref: NULL
rows: 6
Extra: Using where
```

Ahora la salida muestra como el campo *select_type* tiene el valor de *DEPENDENT SUBQUERY* indicando que la consulta es correlacionada.

Sin embargo debemos leer todas las filas de ambas tablas para poder realizarla. Su optimización pasa por crear los índices sobre *autor* e *id*. Entonces obtenemos:

```
1. row ***************************
id: 1
select_type: PRIMARY
table: p
type: ALL
possible_keys: NULL
key: NULL
key_len: NULL
```

```
ref: NULL
rows: 3843
Extra: Using where

2. row ***************************
id: 2
select_type: DEPENDENT SUBQUERY
table: a
type: eq_ref
possible_keys: in1
key: in1
key_len: 4
ref: nmotor.p.post_author
rows: 1
Extra: Using where
```

Como en el caso anterior, al ser el índice sobre *id* único solo necesita hacer una lectura a disco por cada fila de noticias, de ahí el tipo *eq_ref*.

Sin embargo si en la subconsulta usáramos uno o más campos que no están indexados el campo type sería *ALL* indicando que se deben revisar todas las filas de la tabla en la subconsulta para cada fila examinada de la consulta externa o principal.

Otras optimizaciones: *INSERT*, *UPDATE*, *DELETE*

La inserción de una fila requiere los siguientes pasos:

1. Conectar a la base de datos.
2. Enviar el comando *INSERT* al servidor.
3. Parsear el comando.
4. Insertar el registro.
5. Insertar los índices.
6. Cerrar la conexión.

Es un proceso tanto más costoso cuanto más llena está la tabla. Para optimizarlo podemos hacer lo siguiente:

- Usar comandos *INSERT* con valores múltiples.
- Ajustar la variable *bulk_insert_buffer_size* y *key_buffer_size* para tablas *MyISAM* y comandos *INSERT* y *LOAD DATA INFILE*. En todo caso para inserciones masivas es mejor usar éste último.
- Si hay muchos clientes insertando simultáneamente podemos usar la opción *DELAYED* del comando *INSERT*.
- Para tablas *MyISAM* en las que hay inserciones concurrentes así como lecturas de manera simultánea podemos poner la variable *concurrent_insert* a 1 de forma que se permiten inserciones concurrentes siempre que no haya huecos en la tabla producto de la eliminación de filas. Para evitar esto podemos darle el valor 2. Si esta variable está a 0 deshabilitamos esta opción.

- Para inserciones masivas en tabla *MyISAM* que usan *LOAD DATA INFILE* es posible desactivar los índices con el programa *mysisamchk –keys-used=0 –rq /rutaficherodetabla*. Para luego volver a activarlos recreándolos con *myisam –rq /rutaficherodetabla*. Antes y después de este proceso debemos ejecutar *mysqladmin flush-tables*. Alternativamente podemos desactivar los índices con la opción *DISABLE KEYS* de *ALTER TABLE* sin necesidad en este caso de usar *FLUSH TABLES*.
- Para tablas no transaccionales es conveniente bloquearlas para escritura antes de realizar inserciones. Para ello usamos *LOCK TABLES* nombre tabla *WRITE* y *UNLOCK TABLES* para desbloquear. En el caso de tablas transaccionales los comandos son *START TRANSACTION* y *COMMIT*.

En cuanto a la actualización o *UPDATE* su optimización tiene que ver con la necesidad de actualizar los índices así como a las condiciones ompuestas por la cláusula *WHERE*, en ese sentido se ve afectada del mismo modo que una consulta *SELECT*.

Si necesitamos hacer gran cantidad de actualizaciones lo mejor es acumularlas y ejecutarlas en un momento dado en la medida de lo posible.

Para el caso de tablas *MyISAM* que usan tamaño dinámico de fila (*dynamic row format*) es importante usar el comando *OPTIMIZE TABLE* comentado en la siguiente sección.

Por último para sentencias de eliminación o *DELETE* el tiempo que tardan es exactamente proporcional al número de índices. Para acelerar el proceso se debe incrementar el valor de la variable de sistema *key_buffer_size*.

Para borrados de todos los datos de una tabla es mejor el uso de *TRUNCATE TABLE* teniendo en cuenta que es no segura para tablas transaccionales como *InnDB* en el sentido de que puede generar errores si hay una transacción en curso.

5.2.3.2 Comandos de mantenimiento de tablas

MySQL dispone de una serie de comandos que facilitan la optimización de sentencias SQL. Veremos a continuación el uso de las sentencias *ANALYZE*, *OPTIMIZE*, *REPAIR*, *CHECK* y *CHEKSUM*.

ANALYZE TABLE

Este comando analiza y almacena la distribución de claves de una tabla. Funciona con tablas *MyISAM*, *BDB* e *InnoDB*. Para tablas *MyISAM* este enunciado es equivalente a usar el programa *myisamchk --analyze*. Es especialmente útil cuando hemos cargado una gran cantidad de datos para actualizar el índice. Su sintaxis es la siguiente:

```
ANALYZE [NO_WRITE_TO_BINLOG | LOCAL] TABLE tbl_name [, tbl_name] ...
```

MySQL usa la distribución de claves almacenada para decidir el orden en que se harán las consultas que usen combinaciones (*join*) de tablas. También las usa para decidir qué índices tiene que utilizar para una tabla específica dentro de una consulta.

El comando retorna el nombre de tabla el tipo de mensaje (error, info, nota, o warning) y el *Msg_text* o mensaje informativo.

Podemos comprobar la distribución de claves almacenada con el comando *SHOW INDEX*. Si la tabla no ha cambiado desde el último comando *ANALYZE TABLE*, la tabla no se vuelve a analizar.

De forma predeterminada, *ANALYZE TABLE* se escribe en el *log* binario para que se repliquen en los esclavos de replicación. Esta acción se puede suprimir con la palabra clave *NO_WRITE_TO_BINLOG* opcional o su alias *LOCAL*.

REPAIR TABLE

Solo para tablas MyISAM y ARCHIVE. Su función es reparar tablas corruptas (con datos y/o índices erróneos). Normalmente no debe usarse salvo que ocurra algún error grave en el sistema. Su sintaxis es:

```
REPAIR [NO WRITE TO BINLOG | LOCAL] TABLE
    tbl_name [, tbl_name] ...
    [QUICK] [EXTENDED] [USE_FRM]
```

Donde las cláusulas son las mismas que *ANALYZE* salvo:

- *QUICK*, que se usa para reparar únicamente los archivos de índices.
- *EXTENDED*, para crear el índice fila a fila en lugar de crearlo de golpe, lo que es menos seguro.
- *USE_FRM*, cuando el fichero *.MYI* de índices está ausente o corrupto, para lo cual se basa en el fichero *.frm* correspondiente.

Su ejecución devuelve el nombre de la table procesada, la operación realizada (siempre *'repair'*), el *Msg_type* o tipo de mensaje (*status*, *error*, *info*, *note* o *warning*) y un *Msg_text* o mensage descriptivo del estado de la tabla (*ok* si todo ha ido bien).

OPTIMIZE TABLE

Solo para tablas *MyISAM*, *InnoDB* y *ARCHIVE* se usa si hemos hecho una gran cantidad de borrados o modificaciones sobre tablas con tipos variables (*VARCHAR*, *VARBINARY*, *BLOB* o *TEXT*). De este modo se reutiliza el espacio y desfragmenta la tabla optimizando el espacio ocupado por los datos.

Su sintaxis es:

```
OPTIMIZE [NO_WRITE_TO_BINLOG | LOCAL] TABLE
    tbl_name [, tbl_name] ...
```

Proporciona la misma salida que *REPAIR*.

CHECK TABLE

Revisa tablas y vistas de tipo *MyISAM*, *InnoDB* y *ARCHIVE* buscando errores.

Su sintaxis es:

```
CHECK TABLE tbl_name [, tbl_name] ... [option] ...

option = {FOR UPGRADE | QUICK | FAST | MEDIUM | EXTENDED | CHANGED}
```

Donde el valor *option* determina incompatibilidades de versión (*FOR UPGRADE*) o el grado (creciente) de profundidad en la revisión de la tabla.

La salida es similar a *REPAIR* salvo que la operación es check. Si el mensaje obtenido no es *ok* o *Table up to date* es conveniente usar *REPAIR* para corregir lo sposibles errores.

CHECKSUM TABLE

Genera una suma de verificación de la tabla. Cualquier cambio que hagamos a la misma producirá una suma completamente distinta. De este modo podremos saber si ha sido alterada de manera fortuita o sin que nos demos cuenta.

Su sintaxis es:

```
CHECKSUM TABLE tbl_name [, tbl_name] ... [ QUICK | EXTENDED ]
```

La cláusula *QUICK* permite hacer está suma mucho más rápida aunque debe estar activada con la opción *CHECKSUM*=1 cuando creamos la tabla.

5.2.4 OTROS ASPECTOS DE OPTIMIZACION

Sistema de permisos

Como norma general deberemos evitar un sistema de permisos muy sobrecargado para evitar accesos a las tablas de permisos en cada consulta.

Benchmark

Otro aspecto a contemplar (en general para todas las instrucciones *sql*) es el uso de la función *benchmark* que permite hacernos una idea de la eficiencia de las funciones que utilizamos así como su tiempo de proceso.

Identificar consultas lentas

Como vimos en el Capítulo 2 existe un log para consultas lentas. En el se registran aquellas consultas que han tardado mas de un cierto tiempo en ejecutarse. Ello no significa que la consulta sea errónea o este mal hecha. Ni siquiera que siempre vaya a tardar lo mismo. La lentitud pude tener que ver con que la tabla ha estado bloqueada, el servidor estaba recién iniciado (y la caché de índices vacía) o con que en ese momento había otros miles de solicitudes de consultas simultáneas. La versión de MySQL en Linux incorpora el *script mysqldumpslow* que resume el *log* de consultas lentas.

Modificadores de MySQL

Si observas la sintaxis del comando *SELECT* verás que se permiten ciertas cláusulas reservadas que determinar el modo de hacer la consulta en cuanto a su optimización.

```
SELECT
[ALL | DISTINCT | DISTINCTROW ]
[HIGH_PRIORITY]
[STRAIGHT_JOIN]
[SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT]
[SQL_CACHE | SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS]
```

- *SQL_CACHE*: indica que no se almacenen los resultados de una consulta en la caché salvo que la variable *query_cache_type* esté a 1 en cuayo caso no tiene efecto y todas las consultas son cacheadas. Para el caso contrario, *SQL _NO_CACHE*, tiene efecto para cualquier valor de *query_cache_type*.
- *STRAIGHT JOIN*: MySQL determina el orden en que hacel os joins en una consulta con independencia del orden en que se encutren las tablas en la misma. Sin embargo podemos forzar la el orden que queramos usando *STRAIGHT JOIN*, véase el ejemplo 5.16.

EJEMPLO 5.16

```
select * from tabla1 STRAIGHT JOIN tabla2 WHERE...
```

- **Uso de índices**:

  Podemos obligar a MySQL a considerar ciertos índices (y omitir el resto) en una consulta con *USE INDEX*, a usarlos con *FORCE* y a ignorar algunos de ellos con *IGNORE INDEX*.

- **Resultados**:

  Usando *SQL_BIG_RESULT* indicamos a MySQL que el resultado contendrá muchos registros. De menara que el servidor usará tablas temporales. *SQL_BUFFER_RESULT* obliga al servidor a incluir el resultado en una tabla temporal.

Estos modificadores permiten liberar bloqueos de tabla antes de lo normal aunque hay saber bien lo que hace para no abusar de ellos y provocar problemas de memoria.

Medir la eficiencia en consultas

En la mayoría de los casos, se puede calcular el rendimiento de consulta mediante el conteo de búsquedas en disco. Para tablas pequeñas, generalmente se puede encontrar una fila en una búsqueda (ya que el índice esta probablemente cacheado). Para tablas mayores, se puede recurrir a la siguiente formuña estimativa:

Numero_busquedas=log (row_count) / log (index_block_length / 3 * 2 / (index_length data_pointer_length +)) + 1.

Un bloque de índices por lo general equivale a 1.024 *bytes* y el puntero de datos suele ser de cuatro *bytes*. Para una tabla de 500.000 filas con un valor de clave de longitud de tres *bytes* (el tamaño de *MEDIUMINT*), la fórmula sería: *nbusquedas* = (500.000)/log(* 1024/3 2/(3 + 4)) + 1 = 4 busquedas.

Este índice se requieren almacenamiento de alrededor de 500.000 * 7 * 3/2 = 5,2 MB (suponiendo un índice de ocupación de unos 2/3), así que probablemente la mayor parte del índice se encuentre en memoria y solo se necesite una o dos lecturas a disco para encontrar la fila.

Tambíen se pueden estimar velocidades de comandos *insert*, *delete* y *update*. Para ello puedes consultar la apartado 7.3.2

Controlar el optimizador

Podemos usar dos variables del servidor para tal fin:

- *optimizer_prune_level*: para restringir el número de posibilidades a estudiar para optimizar la consulta. Conviene que sea 1 ya que rara vez MySQL "olvida" planes de optimización importantes.
- *optimizer_search_depth*: para restringir el nivel de detalle a efectos de optimizar la consulta. Conviene tener un valor pequeño o cero para que MySQL lo calcule automáticamente.

Para acabar esta sección incluimos un resumen de los comandos de MySQL relacionados con la optimización y en general la mejora del funcionamiento de nuestro servidor.

Tabla 5.1 Resumen comandos SQL de optimización y seguridad

EXPLAIN	Permite saber como se va a ejecutar cierta consulta así como un sinónimo de DESCRIBE.
REPAIR	Repara tablas MyISAM corruptas es similar al programa myisamchk.
CHECK TABLE	Chequea una tabla para encontrar errores. Sirve para tablas MyISAM e InnoDB.
ANALYZE	Analiza la distribución de claves para una tabla. Sirve para tablas MyISAM e InnoDB.
OPTIMIZE	Cuando se han hecho muchas modificaciones a una tabla actualizando estadísticas y reordenando índices.
CHECKSUM	Devuelve la suma de verificación de una tabla.
SHOW	Conjunto de comandos para mostrar información diversa sobre toda clase de objetos del servidor.

ACTIVIDADES 5.2

(Supón en todos los ejercicios que en las bases de trabajo no hay ningún índice creado).

- Indica el comando necesario para eliminar los índices de la tabla movimientos de la base *ebanca* y realiza una consulta sobre todos los movimientos de una cuenta dada. Observa y explica el resultado de aplicar *EXPLAIN* a dicha consulta.
- Crea un índice sobre el campo *cuenta* de la tabla movimiento y comprueba las diferencias en el rendimiento al usar *explain*.
- Explica el resultado de *EXPLAIN* sobre una consulta que obtenga todos los registros de DNI de cliente y números de cuenta. Observa la diferencia con otra en la que necesitas todos los datos de clientes y de todas sus cuentas. ¿Cómo optimizarías aún más la primera consulta?
- Diseña una consulta para obtener el número de cuentas por cliente. Usa un índice para mejorar el resultado de la anterior consulta.
- Calcula el número de filas estimadas necesarias para realizar la consulta que devuelva los datos (DNI y datos de cuenta) del cliente con el mayor saldo. ¿Cómo optimizarías esta consulta?

5.3 OPTIMIZACIÓN DEL SERVIDOR

En lo que concierne al funcionamiento óptimo del servidor son numerosos los parámetros. En esta sección nos ocuparemos fundamentalmente de aquellos que tienen que ver con MySQL sin olvidar que la optimización empieza desde el hardware e incluye aspectos diversos como el sistema operativo o el tipo de sistema de ficheros.

5.3.1 ALMACENAMIENTO

El compromiso entre CPU y operaciones E/S en disco suele ser crítico para el funcionamiento óptimo de un servidor. Podemos disponer de una buena CPU y estar limitados por operaciones de lectura y escritura de datos a disco que limitan su funcionamiento debido al tiempo que requieren los discos en moverse por las pistas en busca de los datos. Esto se agrava por el hecho de que los accesos suelen ser aleatorios.

El parámetro más determinante es el denominado tiempo de latencia que depende de dos factores, a saber, el tiempo necesario para mover las cabezas del lector y la velocidad de rotación del disco parámetro que debemos considerar a la hora de adquirir nuestros discos, ya sean SCSI o IDE.

Así que el primer cuello de botella se sitúa en las I/O de disco, claramente la parte más lenta del sistema. Si disponemos de una buena RAM y configuramos las caché de MySQL (como veremos a continuación) de manera óptima podemos aliviar bastante esta limitación dejando la carga de trabajo a la CPU principalmente.

Los problemas de disco se hacen más evidentes cuando la cantidad de datos comienza a crecer tanto que la caché comienza a ser inefectiva. Para grandes bases de datos donde los datos de acceso más o menos al azar, puede estar seguro de que necesitarás al menos un acceso a disco para leer y un par de búsquedas de disco para escribir registros. Para minimizar este problema, los discos de su uso con tiempos búsqueda bajos.

Otra forma es reducir las búsquedas usando enlaces simbólicos de manera que podamos tener diferentes bases de datos en diferentes discos.

Para mejorar la confiabilidad podemos usar sistemas RAID como por ejemplo RAID 0 + 1 (dividir y replicar) o hacerlo por software usando un gestor de volúmenes. En general podemos aumentar el sistema RAID a medida que aumenta la criticidad de los datos aunque para aplicaciones de muchas escrituras esto puede suponer disminuir el rendimiento.

5.3.2 OPTIMIZACIÓN DE MOTORES DE ALMACENAMIENTO

En esta sección veremos aspectos específicos de los dos tipos de motores más utilizados en MySQL, *MyISAM* e *InnoDB*.

InnoDB

Es el tipo de motor por defecto en las nuevas versiones de MySQL. Es el adecuado si buscamos confiabilidad y tenemos un elevado índice de concurrencia de transacciones. A continuación veremos algunos de los aspectos a tener en cuenta para la optimización y mejor uso de este tipo de motores.

Se pueden configurar usando las variables *innodb_data_home_dir* (directorio de almacenamiento de *InnoDB*) e *innodb_data_file_path* con la sintaxis:

```
innodb_data_file_path=datafile_spec1[;datafile_spec2]...

file_name:file_size[:autoextend[:max:max_file_size]]
```

Siendo *datafile_specx* cada una de las rutas al espacio de almacenamiento. Éstas pueden consisten en un nombre de fichero con el tamaño pudiendo asignar atributos como *autoextend* para permitir que sea autoextensible y el máximo tamaño (solo permitidos para el último *datafile* en la lista).

Por ejemplo:

```
[mysqld]
innodb_data_home_dir = /ibdata
innodb_data_file_path=ibdata1:50M;ibdata2:50M:autoextend
```

- Así mismo las variables *innodb_log_file_size* e *innodb_log_buffer_size* permiten establecer el tamaño de los ficheros *log* (su valor recomendado es un 25% del valor de la variable *innodb_buffer_pool_size*) y memoria de *log* (para almacenar los *logs* sin necesidad de escribir a disco).
- El uso del comando *OPTIMIZE TABLE* permite reorganizar y compactar los datos así como reconstruir el archivo de índices reduciendo el número de accesos a disco necesarios para leer las tablas y optimizando su funcionamiento.
- Es mejor usar claves primarias de tamaño pequeño puesto que cada índice secundario incluye una copia de la misma. En caso de claves principales largas es mejor usar un campo autonumérico.
- Es mejor usar tipos variables que fijos, por ejemplo *VARCHAR* en lugar de *CHAR* puesto que se ocupa menos espacio. Un *CHAR(10)* ocupa siempre 10 *bytes* con independencia del tamaño del dato guardado mientras que en un *VARCHAR(10)* ocupa el tamaño del dato.
- Para servidores con muchas transacciones es mejor poner la variable de sistema *autocommit* a *off* o 0 e incluir todas nuestras transacciones (formadas por operaciones DML) entre sentencias *START TRANSACTION* y *COMMIT*
- Para acelerar importaciones de grandes cantidades de datos podemos desactivar la comprobación de claves secundarias (*UNIQUE*) así como de claves ajenas (*FOREIGN KEY*) con el comando poniendo las variables de sistema unique_checks y foreign_key_checks a 0. Así mismo podemos desactivar el *auto_commit* (*set auto_commit*=0) y añadir un *COMMIT* al final de la importación.
- Es mejor definir las claves primarias en la creación de las tablas InnoDB debido a la forma en que se almacenan. Modificarlas después de crearlas es computacionalmente más costoso.
- Para evitar accesos a disco es conveniente aumentar el valor de *innodb_buffer_pool_size* a valores en el orden de hasta el 80% de nuestra memoria física.

MyISAM

Este tipo de motor es más adecuado para aplicaciones con poca concurrencia y muchos accesos de lectura. Los siguientes son algunos aspectos para su optimización.

- Usar el comando *ANALYZE TABLE* o el programa *myisamchk* con la opción *analyze* cadavez que hagamnos inserciones de datos.

- Si hacemos usualmente consultas de ordenación sobre un mismo campo podemos usar el programa *myisamchk* para ordenar los datos con respecto a cierto índice, por ejemplo *myisamchk –sort-index –sort-records=1 noticias* ordenaría la tabla noticias (en realidad el fichero con los datos de la tabla noticias) de la base de datos *motorblog* con respecto a su primer índice.
- Conviene ejecutar el comando *OPTIMIZE TABLE* o el programa *myisamchk* con la opción *r* después de hacer borrados de datos, ya que así evitamos la fragmentación y facilitamos las inserciones concurrentes.
- Si nuestras tablas cambian con frecuencia es mejor evitar tipos de datos variables (*VARCHAR*, *VARBINARY*, *BLOB* y *TEXT*) ya que en tal caso el valor *ROW_FORMAT* sería *DYNAMIC* lo que la hace más lenta.
- Si frecuentemente consultamos datos ordenados por varios campos o expresiones conviene ejecutar el comando *ALTER TABLE...ORDER BY exp1*, *exp2*, etc., después de realizar operaciones de modificación de datos en la tabla. De este modo ésta se almacena en el orden indicado.
- Para acelerar el proceso de inserción de datos podemos usar la opción *INSERT DELAYED* cuando no es importante saber el momento en que se insertan los datos. De este modo las inserciones se acumulan para ejecutarse en una sola escritura.
- Usando la opción *LOW_PRIORITY* en el comando *INSERT* priorizamos los comandos *SELECT*. Del mismo modo la opción *HIGH PRIORITY* en el comando *SELECT* prioriza los comandos *INSERT* incluso en el caso de que haya alguno esperando.
- Para acelerar operaciones de inserción podemos bloquear las tablas para escritura, por ejemplo *LOCK TABLES a WRITE* bloquearía la tabla a que después de las inserciones desbloquearíamos con *UNLOCK TABLES*. No obstante para grandes cantidades de datos es conveniente alternar bloqueos y desbloqueos para permitir a otros procesos acceder a las tablas.
- En general es mucho más rápido usar LOAD DATA INFILE que comandos INSERT. Para mejorar el funcionamiento de ambas opciones podemos usar la variable *key_buffer_size* como veremos más adelante.

5.3.3 MEMORIA

Son el intermediario entre el procesador y el disco, aun asi siendo mucho mas rapidas que los discos son todavía significativamente mas lentas que el procesador. El sistema operativo almacena en la memoria libre disponible como caché los datos de lectura/escritura a disco, lo que significa que si se repiten consultas es fácil que no se toque el disco.

La gestión de la memoria es quizás el aspecto más relevante en cuanto a la optimización del servidor puesto que una gran cantidad de variables de sistema están relacionadas con su uso. Veremos a continuación aquellas que afectan al funcionamiento general para después especificar las relacionadas con los dos principales motores *MyISAM* e *InnoDB*.

5.3.3.1 Aspectos generales de la memoria

La forma de minimizar esta diferencia de velocidades es usar cachés a nivel de procesador, memorias pequeñas mucho más rápidas que la RAM.

Asi que mejorar el rendimiento supone incrementar memoria suponiendo un hardware con *cache-cpu*, un sistema operativo que haga un uso eficiente de dicha caché.

Otro hecho a considerar en este sentido es si nuestro servidor es dedicado, es decir, si estamos ejecutando otras aplicaciones, que obviamente también requieren memoria, en él. Algunos sistemas como Linux son capaces de hacer *swap* sobre el disco cuando dispone de poca memoria para sus aplicaciones.

En cuanto a MySQL en particular es conveniente ser consciente de cómo trabaja en cuanto a uso de memoria. Para ello, veremos a continuación las variables relacionadas con la memoria de cada tipo de motor de almacenamiento.

MySQL dispone de *buffers* y cachés para su funcionamiento interno. Los *buffers* se dividen en dos grandes grupos: *buffer* de claves y de conexiones. Los primeros se usan a nivel global y los segundos para cada sesión. Los más importantes son los de claves *MyISAM (key_buffer_size)* e *InnoDB* (*innodb_buffer_pool_size*). Estos son utilizados por MySQL para almacenar índices que son muy accedidos de manera que se puedan evitar accesos a disco. Es por ello que es conveniente aumentar sus valores par mejorar el rendimiento, siempre y cuando eso no afecte al funcionamiento del resto del sistema operativo. Podemos consultar su valor con el comando *SHOW* o viendo el tamaño de los archivos de índices, para el caso *MyISAM* con extension *.MYI*. Debemos recordar que *InnoDB* usa índices *cluster* de manera que guarda todos los índices y datos en el mismo área o página de memoria.

Ya vimos en el Capítulo 2 como visualizar y modificar las variables de servidor. Como ya comentamos es imposible detallar todas y no es el objeto del libro. En lo que respecta a la optimización del funcionamiento del servidor y *table_open_cache*. Deberíamos comprobar sus valores antes de hacer cualquier otra modificación.

- *table_open_cache*: indica el número de tablas abiertas por todas las conexiones. Dicho número puede consultarse usando la variable de estado *opened_tables*. Por ejemplo:

```
mysql> show status like 'opened_tables'
```

- *sort_buffer_size*: tamaño de memoria usado para la ordenación. Se crea para cada nueva sesión.
- *read_buffer_size*: *buffer* usado por cada hilo o *thread* para guardar los datos de cada tabla que lee.
- *read_rnd_buffer_size*: cantidad de memoria usada cada vez que se realiza una lectura de datos ordenados según un índice. Permite optimizar consultas *ORDER BY*.
- *join_buffer_size*: establece el tamaño mínimo de memoria requerido para combinaciones de tablas cuando no hay o no es posible usar índices.

Hay dos comandos *sql* que nos pueden ayudar a optimizar memoria, *FLUSH TABLES* y *FLUSH PRIVILEGES*.

El comando *FLUSH TABLES* o el comando *mysqladmin flush-tables* cierra todas las tablas que no están en uso a la vez y las marcas de todas las tablas en uso a ser cerrado cuando el se está ejecutando actualmente termina hilo. Esto efectivamente libera más memoria en uso.

El servidor almacena en caché información de comandos de gestión de usuarios y permisos como *GRANT* y *CREATE USER*. Esta memoria no se libera por el correspondiente *REVOKE*, *DROP USER* por lo que para un servidor que ejecuta muchos casos de los estados que hacen que el almacenamiento en caché, habrá un aumento en el uso de memoria. Esta memoria caché puede ser liberada con *FLUSH PRIVILEGES*.

5.3.3.2 Memoria y motores de almacenamiento

InnoDB

Para el caso de tablas *InnoDB* hay una zona de memoria llamada *buffer pool* para cachear datos e índices en memoria.

Esta zona es controlada por las siguientes variables:

- *innodb_buffer_pool_size*: es el tamaño. En general, si disponemos de memoria, aumentarlo mejorará el rendimiento.
- *inndob_buffer_pool_instances*: divide el tamaño del *buffer pool* entre varias instancias para evitar la competencia debida al acceso compartido al *buffer* por parte de los procesos o hilos.
- *innodb_adaptive_hash_index*: permite activar (con valor *on*) el uso de índices tipo *hash*. Estos índices se obtienen a partir de los propios índices de InnoDB de tipo *b-tree*. Esto ocurre de manera interna transparente al usuario. Es un sistema de las tablas InnoDB que permite acelerar ciertas consultas cuando se ejecutan frecuentemente.

MyISAM

Este tipo de motor usa la llamada *MyISAM key cache* para optimizar las consultas de forma que las consultas realizadas con mayor frecuencia se almacenen en caché y no sea necesario recuperarlas de disco.

Esta caché es accedida de forma compartida por todas sesiones. Esto puede causar problemas de competencia por la caché. Para evitarlos MySQL dispone de la posibilidad de crear múltiples cachés con el comando *CACHE INDEX*, así podemos asignar todos o algunos de los índices de una tabla a una caché determinada.

Por ejemplo, el siguiente comando:

```
mysql>SET GLOBAL keycache1.key_buffer_size=128*1024;
```

Crea una caché llamada *keycache1* de 128 KB. Para destruirla pondríamos su valor a cero.

El comando:

```
mysql>CACHE INDEX noticias IN cachenoticias;
```

Asignaría los índices de la tabla noticias a la caché anterior.

Una regla general para crear cachés puede ser usar una para tablas muy consultadas, otra para tablas que se actualizan con frecuencia y otra por defecto (con un tamaño aproximadamente igual a la mitad del total).

El comando *LOAD INDEX INTO CACHE* permite precargar bloques de índices en la caché de manera que se minimice el acceso a disco en las consultas subsiguientes. Esto es útil si vamos a realizar un gran número de consultas que usen los índices. Por ejemplo, para la tabla noticias:

```
mysql> LOAD INDEX INTO CACHE noticias;
```

Las variables más importantes que afectan al funcionamiento de la caché y memoria en este tipo de motores son:

- *key_buffer_size*: esta variable determina el tamaño del también llamado *key cache*, es decir la caché de claves.

Para ver su valor podemos usar el comando *SHOW*, como en el siguiente ejemplo:

EJEMPLO 5.17

```
mysql>show variables like 'key_buffer_size';
*************************** 1. row ***************************
Variable_name: key_buffer_size   Value: 26214400
```

- *myisam_sort_buffer_size*: el tamaño de memoria o *buffer* reservado para ordenaciones de índices durante un comando *REPAIR TABLE, ALTER TABLE* o *CREATE INDEX*.

Sus valores deben ser ajustados considerando aspectos como el número de clientes, el tipo habitual de consultas (si hay muchas ordenaciones, *joins*, etc.), y la calidad de servicio que se pretende dar. Por ejemplo, si tenemos suficiente RAM y un número moderado de clientes, un *key_buffer_size* de 64 MB y un *table_open_cache* de 256 MB, puede ser suficiente. Sin embargo, con muchas conexiones y poca memoria convendría bajar *key_buffer_size* a valores en torno a 512 K y *read_buffer_size* a 100 K como en este ejemplo:

EJEMPLO 5.18

```
C:\>mysqld  --key_buffer_size=512K --sort_buffer_size=100K \
--read_buffer_size=100K
```

Cuando surgen problemas de memoria podemos recurrir a lo siguiente:

- Añadir más memoria.
- Reducir el número máximo de conexiones.
- Reducir el tamaño de los *buffers*.

En general hay que monitorizar y dejar un margen de memoria disponible garantizando que, en el peor escenario, no habrá problema de memoria.

Tablas temporales (MEMORY o HEAP)

Este tipo de tablas que ya mencionamos anteriormente están muy relacionadas con la optimización ya que se almacenan en memoria y usan índices hash por defecto asi que pueden ser muy útiles para hacer determinadas consultas. Podemos crear tablas temporales manualmente añadiendo la cláusula ENGINE = MEMORY al final del comando CREATE TABLE las variables relacionadas son:

- *max_heap_table_size*: determina el tamaño máximo permitido a una tabla creada manulamente.
- *max_tmp_tables*: determina el máximo número de tablas abiertas permitidas a un cliente.

  MySQL también usa tablas temporales para resolver más fácilmente las consultas. Dichas tablas se pueden crear en disco o en memoria dependiendo del tamaño. Este comportamiento es controlado por las variables.

- *tmp_table_size*: es el tamaño máximo de las tablas temporales creadas por el servidor (en caso de existir la variable *max_heap_table_size* será el mínimo valor de ambos). En caso de requerir una tabla de mayor tamaño MySQL la crea en disco (*in-disk table*). Aumentar su valor es especialmente útil si usamos muchas consultas de tipo *GROUP BY*.

Para saber si el servidor usa tablas temporales en una de nuestras consultas usaremos *EXPLAIN*. En caso de ser así veremos en el campo extra el valor *Using temporary*. Normalmente esto ocurrirá en tres casos:

- Cuando usamos *GROUP BY* y *ORDER BY* sobre campos distintos.
- Cuando hay *ORDER BY* y *GROUP BY* sobre campos diferentes a los pertenecientes a la primera tabla de una combinación o *join*.
- Cuando usamos *DISTINCT* con *ODER BY*.

Si usamos el modificador *SQL_SMALL_RESULT*.

En los siguientes casos MySQL usará probablemente una tabla en disco:

- Si hay columnas tipo *BLOB* o *TEXT* en la tabla.
- Si hay columnas mayores de 512 *bytes* en un *ORDER BY* o *DISTINCT*.
- Si hay columnas mayores de 512 *bytes* en el *SELECT* cuando éste incluye uniones (cláusulas *UNION* o *UNION ALL*).

Mediante el siguiente comando podemos conocer el número de tablas y ficheros temporales creados:

```
mysql> show status like 'created_tmp%';
```

Es decir *Created_tmp_disk_tables*, *Created_tmp_files*, *Created_tmp_tables*.

Si consideramos que se van a necesitar muchas tablas temporales deberemos incrementar el valor de la variable *tmp_table_size* teniendo cuidado de no ocupar demasiada memoria.

5.3.3.3 Caché de consultas de MySQL

MySQL usa la llamada *query cache* almacenando el texto de una consulta SELECT junto con el resultado que se le envió al cliente. Si se recibe una consulta idéntica posteriormente, el servidor devuelve el resultado de la caché de consultas en lugar de *parsear* y ejecutar la consulta de nuevo.

La caché de consultas es muy útil en aplicaciones con tablas que no cambian frecuentemente y donde el servidor recibe muchas consultas idénticas. Esta es la típica situación de muchos servidores Web que generan páginas dinámicas basadas en contenido de base de datos. Sin embargo en aplicaciones dónde hay frecuentes modificaciones de datos o que usan tablas pequeñas y consultas sencillas activar la caché puede incluso disminuir la eficiencia.

No devuelve datos antiguos. Cuando las tablas se modifican, cualquier entrada relevante en la caché de consultas se elimina.

Funcionamiento

Las consultas deben ser exactamente las mismas (*byte* a *byte*) para se consideradas idénticas.

Antes que una consulta se guarde en la caché de consultas, MySQL comprueba que el usuario tenga permisos de *SELECT* para todas las bases de datos y tablas involucradas. Si no es el caso, el resultado cacheado no se usa.

Si un resultado de consulta se retorna desde la caché de consultas, el servidor incrementa la variable de estado *Qcache_hits* (sin embargo no se incremente el contador de consultas *Com_select*). Podemos comprobarlo con los comandos:

```
mysql>show variables like '%Qcache%';
mysql>show status like '%cache%';
```

Si una tabla cambia, entonces todas las consultas cacheadas que usen esa tabla pasan a ser inválidas y se eliminan de la caché. Una tabla puede cambiarse por varios tipos de comandos, tales como *INSERT*, *UPDATE*, *DELETE*, *TRUNCATE*, *ALTER TABLE*, *DROP TABLE* o *DROP DATABASE*.

También funciona dentro de transacciones cuando se usa tablas *InnoDB*. Pero no lo hace en sentencias dentro de procedimientos, funciones o *triggers*.

Una consulta tampoco se cachea bajo las siguientes condiciones:

- ✓ Se refiere a funciones definidas por el usuario (UDFs).
- ✓ Se refiere a variables de usuario.
- ✓ Se refiere a las tablas en la base de datos del sistema MySQL.
- ✓ Es cualquiera de las siguientes formas:

```
SELECT ... IN SHARE MODE
SELECT ... FOR UPDATE
SELECT ... INTO OUTFILE ...
SELECT ... INTO DUMPFILE ...
SELECT * FROM ... WHERE autoincrement_col IS NULL
```

- ✓ Usa tablas *TEMPORARY*.
- ✓ No usa ninguna tabla.
- ✓ El usuario tiene permisos a nivel de columan para cualquiera de las tablas involucradas.

Coniguración

La variable de sistema *have_query_cache* indica si la caché de consultas está disponible:

```
mysql> SHOW VARIABLES LIKE 'have_query_cache';

+------------------+-------+
| Variable_name    | Value |
+------------------+-------+
| have_query_cache | YES   |
+------------------+-------+
```

Muchas otras variables de sistema controlan las operaciones de la caché de consultas. Todas ellas tienen nombres que empiezan con *query_cache.*

Para especificar el tamaño de la caché de consulta debemos inicializar la variable de sistema *query_cache_size* con un valor distinto de 0 (valor por defecto). Ponerla a 0 desactiva la caché.

Si el tamaño de la caché de consultas es mayor que 0, la variable *query_cache_type* determina su funcionamiento. Esta variable puede tener los siguientes valores:

- Un valor de 1 u *on* permite el cacheo excepto para aquellos comandos que empiecen con *SELECT SQL_NO_CACHE*.
- Un valor de 2 o *DEMAND* provoca el cacheo de solo los comandos que empiecen con *SELECT SQL_CACHE.*

Inicializar con el valor *GLOBAL* la *variable query_cache_type* determina el comportamiento de la caché de consultas para todos los clientes que se conecten tras el cambio. Cada cliente puede controlar el comportamiento de la caché para su propia conexión mediante el valor *SESSION* de *query_cache_type*. Por ejemplo, un cliente puede desactivar el uso de la caché de consultas para sus propias consultas así:

```
mysql> SET SESSION query_cache_type = OFF;
```

Para controlar el tamaño máximo de resultados de consultas individuales que pueden cachearse, inicialice la variable *query_cache_limit* variable. El valor por defecto es 1 MB.

Cuando se cachea una consulta, sus resultados (los datos que se envían al cliente) se guardan en la caché a medida que se reciben. Por lo tanto los datos normalmente no se guardan en un gran paquete. La caché reserva bloques para

guardar estos datos bajo demanda, así que cuando se llena un bloque, se prepara un nuevo bloque. Puesto que la reserva de memoria es una operación costosa (lenta), la caché de consultas reserva bloques con un tamaño mínimo dado por la variable de sistema *query_cache_min_res_unit*. Cuando la consulta queda ejecutada, el último bloque de resultados se ajusta a su tamaño real para liberar la memoria no usada. En función del tipo de consulta que ejecute el servidor, puede resultar útil cambiar el valor de *query_cache_min_res_unit*:

El valor por defecto de *query_cache_min_res_unit* es 4 KB. Debería ser un valor adecuado para la mayoría de los casos.

Si ejecutamos muchas consultas con resultados pequeños, el tamaño por defecto del bloque puede llevar a fragmentación de memoria, que se evidencia con un alto número de bloques libres. La fragmentación puede obligar a la caché a borrar consultas por falta de memoria. En este caso, debemos decrementar el valor de *query_cache_min_res_unit*. El número de bloques libres y de consultas borradas debido a falta de espacio se puede consultar en las variables *Qcache_free_blocks* y *Qcache_lowmem_prunes*.

Si la mayoría de las consultas tienen resultados grandes (compruebe las variables de estado *Qcache_total_blocks* y *Qcache_queries_in_cache*), podemos incrementar el rendimiento incrementando el valor *query_cache_min_res_unit*.

Mantenimiento

Podemos defragmentar la caché de consultas para mejor uso de esta memoria con el comando *FLUSH QUERY CACHE*. El comando no elimina ninguna consulta de la caché. Su ejecución solo deja un bloque de memoria libre.

Los comandos *RESET QUERY CACHE* y *FLUSH TABLES* eliminan todos los resultados de consultas de la caché de consultas.

Como ya hemos visto el comando SHOW STATUS nos permite ver para ver las variables de estado de la caché como en el siguiente ejemplo:

EJEMPLO 5.19

```
mysql> SHOW STATUS LIKE 'Qcache%';
+-------------------------+--------+
| Variable_name           | Value  |
+-------------------------+--------+
| Qcache_free_blocks      | 36     |
| Qcache_free_memory      | 138488 |
| Qcache_hits             | 79570  |
| Qcache_inserts          | 27087  |
| Qcache_lowmem_prunes    | 3114   |
| Qcache_not_cached       | 22989  |
| Qcache_queries_in_cache | 415    |
| Qcache_total_blocks     | 912    |
+-------------------------+--------+
```

El número total de consultas *SELECT* es igual a:

Com_select + *Qcache_hits* + consultas con errores detectados por el *parser*.

El valor *Com_select* es igual a:

Qcache_inserts + *Qcache_not_cached* + consultas con errores encontrados durante la comprobación de permisos sobre columnas.

Cada consulta cacheada requiere como mínimo dos bloques (uno para el texto de la consulta y otro o más para el resultado de la misma). Además, cada tabla utilizada por una consulta requiere un bloque. Sin embargo, si dos o más consultas usan la misma tabla, solo se reserva un bloque.

La información proporcionada por la variable de estado *Qcache_lowmem_prunes* puede ayudarnos a ajustar el tamaño de la caché de consultas. Cuenta el número de consultas que se han eliminado de la caché para liberar memoria con el fin de cachear nuevas consultas. La caché de consultas usa una estrategia de resultado usado menos recientemente (LRU, siglas en inglés de *Least Recently Used*) para decidir qué consultas eliminar de la caché.

5.3.4 RENDIMIENTO

Medir el rendimiento es una tarea compleja y diversa ya que todos los parámetros del servidor se ven involucrados. Además ajustar por ejemplo una variable de sistema en una consulta puede ser que afecte negativamente a otra, así que es un proceso que requiere de experiencia, paciencia y cautela.

El objeto de medir es estimar los requisitos de nuestro servidor cuando se ve sometido a una alta carga de trabajo.

Existen diferentes posibilidades que podemos utilizar, aunque su uso queda fuera del alcance de este libro, las nombraremos brevemente:

- Función *BENCHMARK()*: para pruebas de ejecución de ciertos cálculos una cierta cantidad de veces.
- MySQL *benchmark Suite*: conjunto de *scripts* en Perl incorporados en la distribución fuente de MysQL (directorio *sql-bench*) y que realizan operaciones decomprobación tales como determinar las funciones que soporta el servidor o el tamaño máximo de una consulta. De momento se ejecutan como un solo hilo aunque se espera que en breve haya versiones multihilo.
- *Scripts* propios o de terceros: creados usando distintos lenguajes de programación para adaptarlos a los requiermientos de nuestras aplicaciones.

ACTIVIDADES 5.3

- Activa la *query* caché y comprueba cómo la consulta que devuelve todas las noticias se ejecuta en menor tiempo al ejecutarla por segunda vez.
- Usa los modificadores *SQL_BIG_RESULT* para comprobar los distintos tiempos de la consulta que obtiene las noticias.
- Indica la consulta que obtiene los movimientos ordenado por cuenta sin usar índices ni caché. Observa y explica la diferencia en rendimiento al usar índices y caché.
- Diseña una consulta para obtener el título y contenido de las noticias ordenadas por fecha. Indica que variables relacionadas con la memoria pueden mejorar el rendimiento de la misma. Intenta, así mismo, dar un valor a las mismas acorde con tu sistema.
- Indica los comandos necesarios para ver las variables de sistemas globales y de sesión, así como las variables de estado relacionadas con la caché.

5.4 HERRAMIENTAS DE MONITORIZACIÓN DE MYSQL

Monitorizar es una tarea imprescindible del administrador. Por muy bien que hagamos las cosas siempre debemos estar atentos a cualquier cambio de configuración, posibles errores de usuarios y/o programadores, caídas del sistema o del software, escalabilidad, etc. Sin duda un buen diseño del sistema y de las tareas de mantenimiento son aspectos clave pero sin duda monitorizar es algo que no debemos olvidar.

Obviamente no podemos **monitorizar** en todo momento y debemos hacerlo con cierta regularidad y de la manera más automatizada posible.

Existen multitud de programas que permiten monitorizar, no solo nuestro servidor de bases de datos, sino múltiples servicios en redes de toda clase.

En la presente sección trataremos dos clases de programas:

- Programas propios de MySQL: comandos *show* y *mysqladmin*.
- Programas de terceros: *hyperic* y *phpmytop*.

Por último haremos una revisión del *software* de monitorización disponible para MySQL.

5.4.1 COMANDOS SHOW

SHOW tiene varias formas que proporcionan información acerca de los objetos de la base del sistema gestor (bases de datos, tablas, etc.) o acerca del estado del servidor. Esta sección describe estos puntos:

```
SHOW [FULL] COLUMNS FROM tbl_name [FROM db_name] [LIKE 'pattern']
SHOW CREATE DATABASE db_name
SHOW CREATE TABLE tbl_name
SHOW DATABASES [LIKE 'pattern']
SHOW ENGINE engine_name {LOGS | STATUS }
SHOW [STORAGE] ENGINES
SHOW ERRORS [LIMIT [offset,] row_count]
SHOW GRANTS FOR user
SHOW INDEX FROM tbl_name [FROM db_name]
SHOW INNODB STATUS
SHOW [BDB] LOGS
SHOW PRIVILEGES
SHOW [FULL] PROCESSLIST
SHOW [GLOBAL | SESSION] STATUS [LIKE 'pattern']
SHOW TABLE STATUS [FROM db_name] [LIKE 'pattern']
SHOW [OPEN] TABLES [FROM db_name] [LIKE 'pattern']
SHOW TRIGGERS
SHOW [GLOBAL | SESSION] VARIABLES [LIKE 'pattern']
SHOW WARNINGS [LIMIT [offset,] row_count]
```

El comando *SHOW* también tiene formas que proporcionan información acerca de servidores de replicación maestros y esclavos como:

```
SHOW BINLOG EVENTS
SHOW MASTER LOGS
SHOW MASTER STATUS
SHOW SLAVE HOSTS
SHOW SLAVE STATUS
```

En la sintaxis para un comando *SHOW* dado incluye una parte *LIKE 'pattern'* , *'pattern'* es una cadena que puede contener los caracteres de SQL '%' y '_' . El patrón es útil para restringir la salida del comando para valores coincidentes.

Veremos a continuación ejemplos de los más importantes dado que el resto son muy intuitivos e incluso los hemos tratado en capítulos anteriores.

Motores

```
SHOW ENGINE engine_name {LOGS | STATUS }
```

SHOW ENGINE muestra información de log o estado acerca de motores de almacenamiento. Destacamos por su uso creciente el siguiente:

```
SHOW ENGINE INNODB STATUS
```

La salida de dicho comando es extensa asi que lo mejor si queremos estudiarlo con detalle es enviarlo a un fichero con una redirección:

EJEMPLO 5.20

```
C:\>mysql -e "show engine innodb status" >c:\mysql\innodb_status
```

Para tablas *innodb* con este comando obtenemos datos de estado acerca de las tablas de este tipo presentes en nuestro sistema. Esto nos permite ver la siguiente información:

- Bloqueos de tabla y registro de cada transacción activa.
- Semáforos de tipo *wait* para hilos.
- Procesos de E/S pendientes o en marcha.
- Estadísitcas de uso de *buffer*.
- Datos de *log*.
- Información de operaciones sobre filas.

Errores

```
SHOW ERRORS [LIMIT [offset,] row_count]
SHOW COUNT(*) ERRORS
```

Este comando es similar a *SHOW WARNINGS*, excepto que en lugar de mostrar errores, advertencias y notas solo muestra errores.

La cláusula *LIMIT* tiene la misma sintaxis que para el comando *SELECT*.

El comando *SHOW COUNT(*) ERRORS* muestra el número de errores. Puede recibir este número de la variable error_count:

```
SHOW COUNT(*) ERRORS;
SELECT @@error_count;
```

Logs binarios

```
SHOW BINARY LOGS
SHOW MASTER LOGS
```

Lista los ficheros de *log* binario en el servidor.

```
SHOW BINLOG EVENTS
   [IN 'log_name'] [FROM pos] [LIMIT [offset,] row_count]
```

Muestra los eventos en ficheros de *log* binario.

```
mysql> SHOW BINARY LOGS;
+---------------+-----------+
| Log_name      | File_size |
+---------------+-----------+
| binlog.000015 |    724935 |
| binlog.000016 |    733481 |
+---------------+-----------+
```

Índices

```
SHOW INDEX FROM tbl_name [FROM db_name]
```

SHOW INDEX retorna información de índice de tabla.

SHOW KEYS es sinónimo para *SHOW INDEX*. También puedes listar los índices de una tabla con el comando *mysqlshow -k db_name, tbl_name*.

Tablas abiertas

```
SHOW OPEN TABLES [{FROM | IN} db_name]
    [LIKE 'pattern' | WHERE expr]
```

SHOW OPEN TABLES lista la tablas abiertas en ese momento en la caché.

Es muy útil para ver el estado de bloqueo de tablas con el campo *In_ use* cuyo valor será un entero mayor o igual a cero según el estado de bloqueo.

Lista de procesos

```
SHOW [FULL] PROCESSLIST
```

Muestra qué flujos están en ejecución. Puedes obtener esta información usando el comando *mysqladmin processlist*. Con el permiso *SUPER*, puedes ver todos los flujos. De otro modo, podrás ver solo los propios (esto es, flujos asociados con la cuenta MySQL que está usando). Este comando es útil si obtiene el mensaje de error "demasiadas conexiones" para encontrar qué ocurre. MySQL reserva una conexión extra para usar por cuentas con el permiso *SUPER*, para asegurar que el administrador siempre es capaz de conectar y comprobar el sistema (asumiendo que no da este permiso a todos los usuarios).

Algunos estados vistos comúnmente en la salida de *SHOW PROCESSLIST*:

- *Checking table*

  El flujo está realizando un chequeo (automático) de la tabla.

- *Closing tables*

  Significa que el flujo está volcando los datos que han cambiado de la tabla a disco y cerrando las tablas usadas. Esto debe ser una operación rápida. Si no lo es, debe verificar que no tiene el disco lleno y que el disco no tiene un uso muy pesado.

- *Connect Out*

  Esclavo conectando con el maestro.

- *Copying to tmp table on disk*

  El conjunto de resultados temporal era mayor que *tmp_table_size* y el flujo está cambiando la tabla temporal de memoria a disco para ahorrar memoria.

- *Creating tmp table*

  El flujo está creando una tabla temporal para guardar parte del resultado de una consulta.

- *Deleting from main table*

  El servidor está ejecutando la primera parte de un borrado de tablas múltiple y borrando solo la primera tabla.

- *Deleting from reference tables*

  El servidor está ejecutando la segunda parte de un borrado de tablas múltiples y borrando los registros coincidentes de las otras tablas.

- *Flushing tables*

  El flujo está ejecutando *FLUSH TABLES* y espera a que todos los flujos cierren sus tablas.

- *Killed*

  Alguien ha enviado un comando *KILL* al flujo y debería abortar en cuanto chequee el *flag kill*. El *flag* se chequea en cada vuelta al bucle principal de MySQL, pero en algunos casos puede tardar algo de tiempo en que muera el flujo. Si el flujo está bloqueado por algún otro flujo, el *kill* tiene efecto en cuanto el otro flujo libera el bloqueo.

- *Locked*

  La consulta está bloqueada por otra consulta.

- *Sending data*

  El flujo está procesando registros para un comando *SELECT* y también enviando datos al cliente.

- *Sorting for group*

  El flujo está ordenando para un *GROUP BY*.

- *Sorting for order*

  El flujo está ordenando para un *ORDER BY*.

- *Opening tables*

  El flujo está intentando abrir una tabla. Esto debería ser un proceso muy rápido, a no ser que algo importante evite la abertura. Por ejemplo, un comando *ALTER TABLE* o *LOCK TABLE* puede evitar abrir una tabla hasta que el comando acabe.

- *Removing duplicates*

  La consulta usaba *SELECT DISTINCT* de forma que MySQL no podía optimizar las distintas operaciones en una fase temprana. Debido a ello, MySQL necesita una fase extra para borrar todos los registros duplicados antes de enviar el resultado al cliente.

- *Reopen table*

  El flujo obtivo un bloqueo para la tabla, pero se dio cuenta tras obtenerlo que la estructura de la tabla cambió. Se libera el bloqueo, cierra la tabla y trata de reabrirla.

- *Repair by sorting*

  El código de reparación está usando una ordenación para crear índices.

- *Repair with keycache*

  El código de reparación está usando creación de claves una a una en la caché de claves. Esto es mucho más lento que *Repair by sorting*.

- *Searching rows for update*

  El flujo hace una primera fase para encontrar todos los registros coincidentes antes de actualizarlos. Esto debe hacerse si *UPDATE* está cambiando el índice que se usa para encontrar los registros implicados.

- *Sleeping*

  El flujo espera que el cliente envíe un nuevo comando.

- *System lock*

  El flujo espera obtener un bloqueo de sistema externo para la tabla. Si no está usando múltiples servidors *mysqld* accediendo a las mismas tablas, puede deshabilitar los bloqueos de sistema con la *opción --skip-external-locking*.

- *Upgrading lock*

  El handler *INSERT DELAYED* trata de obtener un bloqueo para la tabla para insertar registros.

- *Updating*

  El flujo está buscando registros para actualizar.

- *User Lock*

  El flujo espera un *GET_LOCK()*.

- *Waiting for tables*

  El flujo obtuvo una notificación que la estructura de la tabla cambió y necesita reabrirla. Sin embargo, para ello, debe esperar a que el resto de flujos cierren la tabla en cuestión.

  Esta notificación tiene lugar si otro flujo ha usado *FLUSH TABLES* o uno de los siguientes comandos en la tabla en cuestión: *FLUSH TABLES tbl_name*, *ALTER TABLE*, *RENAME TABLE*, *REPAIR TABLE*, *ANALYZE TABLE* o *OPTIMIZE TABLE*.

- *Waiting for handler insert*

  El *handler INSERT DELAYED* ha procesado las inserciones pendientes y espera nuevas.

La mayoría de estados se corresponden a operaciones rápidas. Si un flujo está en alguno de ellos varios segundos, puede existir un problema que necesite investigar.

Hay algunos estados que no se mencionan en la lista precedente, pero varios de ellos son útiles solo para encontrar fallos en el servidor.

Perfiles

```
SHOW PROFILE [type [, type] ... ]
    [FOR QUERY n]
    [LIMIT row_count [OFFSET offset]]

type:
    ALL
  | BLOCK IO
  | CONTEXT SWITCHES
  | CPU
  | IPC
  | MEMORY
  | PAGE FAULTS
  | SOURCE
  | SWAPS
```

Muestra información indicando el uso de recursos por parte de las sentencias en la sesión actual.

Se controla por la variable *profiling* normalmente puesta a *off*. Para activarlo usamos el comando:

```
mysql> SET profiling = 1;
```

SHOW PROFILE muestra informacion total sobre las operaciones (según el *type* especificado) y el tiempo de duración de las mismas por parte del servidor.

SHOW PROFILES muestra la duración de los commandos ejecutados en la sesión actual.

Según el valor de *type* podemos obtener la siguiente información:

- *ALL*: toda información posible.
- *BLOCK IO*: operaciones de bloqueo por entrada/salida.
- CPU: tiempos de uso de CPU del usuario y el sistema.

Algunos ejemplos de uso pueden ser:

EJEMPLO 5.21

```
mysql> SHOW PROFILES;
mysql> SHOW PROFILE;

mysql> SHOW PROFILE FOR QUERY 1;
+--------------------+----------+
| Status             | Duration |
+--------------------+----------+
| query end          | 0.000107 |
| freeing items      | 0.000008 |
| logging slow query | 0.000015 |
| cleaning up        | 0.000006 |
+--------------------+----------+
4 rows in set (0.00 sec)

mysql> SHOW PROFILE CPU FOR QUERY 2;
+----------------------+----------+----------+------------+
| Status               | Duration | CPU_user | CPU_system |
+----------------------+----------+----------+------------+
| checking permissions | 0.000040 | 0.000038 |   0.000002 |
| creating table       | 0.000056 | 0.000028 |   0.000028 |
| After create         | 0.011363 | 0.000217 |   0.001571 |
| query end            | 0.000375 | 0.000013 |   0.000028 |
| freeing items        | 0.000089 | 0.000010 |   0.000014 |
| logging slow query   | 0.000019 | 0.000009 |   0.000010 |
| cleaning up          | 0.000005 | 0.000003 |   0.000002 |
+----------------------+----------+----------+------------+
7 rows in set (0.00 sec)
```

Estado del servidor

```
SHOW [GLOBAL | SESSION] STATUS [LIKE 'pattern']
```

SHOW STATUS proporciona información de estado del servidor. Esta información puede obtenerse usando el comando *mysqladmin extended-status*.

Aquí se muestra una salida parcial:

```
mysql> SHOW STATUS;
+--------------------------+------------+
| Variable_name            | Value      |
+--------------------------+------------+
| Aborted_clients          | 0          |
| Aborted_connects         | 0          |
| Bytes_received           | 155372598  |
| Bytes_sent               | 1176560426 |
| Connections              | 30023      |
| Created_tmp_disk_tables  | 0          |
| Created_tmp_tables       | 8340       |
| Created_tmp_files        | 60         |
...                        ...          ...
| Open_tables              | 1          |
| Open_files               | 2          |
| Open_streams             | 0          |
| Opened_tables            | 44600      |
| Questions                | 2026873    |
...                        ...          ...
| Table_locks_immediate    | 1920382    |
| Table_locks_waited       | 0          |
| Threads_cached           | 0          |
| Threads_created          | 30022      |
| Threads_connected        | 1          |
| Threads_running          | 1          |
| Uptime                   | 80380      |
+--------------------------+------------+
```

Con *GLOBAL*, se obtienen los valores de estado para todas las conexiones a MySQL. Con *SESSION*, se obtienen los valores de estado para la conexión actual. Si no usamos estas opciones, por defecto es *SESSION*. *LOCAL* es sinónimo de *SESSION*.

Ten en cuenta que algunas variables de estado solo tienen un valor global. Para ellas obtiene el mismo valor para *GLOBAL* y *SESSION*.

Tablas

```
SHOW TABLE STATUS [FROM db_name] [LIKE 'pattern']
```

SHOW TABLE STATUS funciona como *SHOW TABLE*, pero proporciona mucha más información acerca de cada tabla. Puede obtener esta lista usando el comando *mysqlshow --status db_name*. Este comando también muestra información sobre vistas.

En el comentario de tabla, las tablas *InnoDB* reportan el espacio libre del espacio de tabla al que pertenece la tabla. Para una tabla localizada en el espacio compartido, este es el espacio libre del espacio de tabla compartido. Si usa múltiples espacios y la tabla tiene el suyo, el espacio libre es solo para esa tabla.

Para tablas *MEMORY* (*HEAP*) los valores *Data_length*, *Max_data_length*, e *Index_length* aproximan la cantidad real de memoria reservada. El algoritmo de reserva reserva grandes cantidades de memoria para reducir el número de operaciones de reserva.

Para vistas, todos los campos mostrados por *SHOW TABLE STATUS* son *NULL*, excepto que *Name* indicara el nombre de vista y *Comment* indicara *view*.

Variables del servidor

Para consultar el valor de las variables de servidor disponemos del siguiente comando:

```
SHOW [GLOBAL | SESSION] VARIABLES [LIKE 'pattern']
```

Este comando ya se vió en el Capítulo 2. Muestra los valores de las variables de sistema de MySQL. Esta información puede obtenerse también usando el comando *mysqladmin variables*.

Con la opción *GLOBAL*, obtenemos los valores que se usan para nuevas conexiones de MySQL. Con *SESSION* obtenemos los valores que hay en efecto para la conexión actual.

Para monitorizar el estado usaremos el comando *SHOW STATUS* que nos mostrará todas las variables de estado. Podemos filtrar esta salida usando *LIKE*, por ejemplo:

EJEMPLO 5.22

```
mysql>SHOW STATUS LIKE '%Com%'
```

Nos mostrará el estado de las variables de estado que computan el número de veces que ciertos comandos han sido ejecutados tal como vimos en el Capítulo 2.

Programa *mysqladmin*

Es un cliente que permite realizar operaciones de administración sobre el servidore mysql. En esta sección describiremos algunas de estas tareas básicas relacionadas con la monitorización.

La sintaxis general del programa es:

```
#mysqladmin [options] command [command-arg] [command [command-arg]] ...
```

Esencialmente mostraremos las opciones del programa seguido de un ejemplo:

- ✓ Saber si el servidor está activo:

EJEMPLO 5.23

```
# mysqladmin -u root -p ping
Enter password:
mysqld is alive
```

- ✓ Conocer la version:

EJEMPLO 5.24

```
# mysqladmin -u root -pxxx version
mysqladmin  Ver 8.42 Distrib 5.0.67 on i686
Copyright (C) 2000-2006 MySQLAB
Server version          5.0.67
Protocol version        10
Connection              Localhost via UNIX socket
UNIX socket             /var/lib/mysql/mysql.sock
Uptime:                 100 days 0 hours 0 min 0 sec
Threads: 1  Questions: 241986  Slow queries: 0  Opens: 11097
Flush tables: 2  Open tables: 32  Queries per second avg: 1.23
```

- Estado actual del servidor:

EJEMPLO 5.25

```
# mysqladmin -u root -pxxx status
Uptime: 8640000
Threads: 1  Questions: 241986  Slow queries: 0  Opens: 11097
Flush tables: 2  Open tables: 32 Queries per second avg: 1.23
```

Donde se muestran los siguientes campos:

- **Uptime**: segundos desde los que se inició el servidor.
- **Threads**: numero total de clientes conectados.
- **Questions**: número total de consultas que el servidor ha ejecutado desde desde su inicio.
- **Slow queries**: número total de consulta que el servidor ha ejecutado y que tienen un tiempo mayor de ejecución a la variable *long_query_time*.
- **Opens**: número total de tablas abiertas por el servidor.
- **Flush tables**: cuantas tablas han sido "volcadas".
- **Open tables**: número total de tablas abiertas en la base de datos.

- Variables de estado:

EJEMPLO 5.26

```
# mysqladmin -u root -pxxx extended-status
-Procesos en ejecución
# mysqladmin -u root -pxxx processlist
Para saberlo cada dos segundos usamos la opción -i
#mysqladmin -u root -pxxx -i 1 processlist
```

(Para detener o matar un proceso consultar la opción *kill*).

5.4.3 OTRAS HERRAMIENTAS

Existen infinidad de proyectos, programas y *scripts* dedicados con mayor o menor detalle a facilitar la monitorización. En esta sección nos limitaremos a enumerar algunos de ellos indicando la web de referencia para que podáis consultarla y probarlos vosotros mismos.

- *Mysql Enterprise monitor*

  Es la herramienta oficial de MySQL. Monitoriza continuamente nuestro sistema enviando alertas ante peligros potenciales. Forma parte de *MySQL Enterprise Edition*, edición de pago de MySQL para la administración.

  http://www.mysql.com/products/enterprise/monitor.html

- *MONyog*

  Es una herramienta comercial con características similares a la anterior en cuanto a su capacidad para monitorizar y avisar al administrador sobre fallos del servidor antes de que estos puedan producirse. Es parte del proyecto *Webyog*.

  http://www.webyog.com

- *mysqlreport*

  Solo para sistemas Unix, realiza informes de estado del servidor:

- *mysqlsniffer*

  Escucha en la red buscando paquetes con el protocolo de MySQL:

- *matatkit*

  Conjunto de *scripts* para la monitorización y mejora del rendimiento de MySQL:

  http://www.maatkit.org/

- *mycheckpoint*

  Utilidad de monitorización para sistemas Unix:

  http://code.openark.org/forge/mycheckpoint

- *cacti*

  Utilidad de monitorización de sistemas en general muy potente y *open source*:

  http://www.cacti.net/

- *mtop*

  Utilidad para sistemas Unix similar al popular *top* para monitorizar procesos en MySQL.

ACTIVIDADES 5.4

- Averigua las principales características de MySQL monitor Enterprise.
- Indica a qué variables debemos prestar mayor atención dependiendo de si tenemos una aplicación de muchas lecturas, mucho proceso o muchas escrituras.
- Comenta y prueba al menos cuatro programas que permitan monitorizar procesos en sistemas operativos Linux y Windows (en la página *Sysinternals* de Microsoft).

5.5 CASO BASE

Haz una estimación del uso del servidor en cuanto a necesidad de almacenamiento y memoria según las operaciones SQL y número medio de usuarios y conexiones. Divídelo en los siguientes apartados:

- ÍNDICES

  Crea los índices necesarios según el diseño y consultas estudiadas en el apartado anterior.

- SQL

  Ajusta el diseño de la base de datos siguiendo los consejos del capítulo.

  Determina las consultas más frecuentes y/o exigentes de los usuarios y de la aplicación. Usa el comando *Explain* para ver el plan de ejecución en cada caso.

- SERVIDOR

  Optimiza las variables relacionadas con el motor *MyISAM* según el análisis del apartado anterior.

- COMANDOS OPTIMIZACIÓN

  Indica la necesidad y la frecuencia de uso de los comandos de optimización: *ANALYZE*, *OPTIMIZE TABLE*, *REPAIR TABLE*.

- COMANDOS SEGURIDAD

  Determina la conveniencia, frecuencia y modo de usar los comandos *CHECK*, *CHECKSUM* para asegurar la estabilidad e integridad de las tablas.

- ALMACENAMIENTO

  Determina en que casos convendría usar RAID en cualquiera de sus versiones y teniendo en cuenta la necesidades de disponibilidad y criticidad de los datos.

- MOTORES

  Considera la necesidad de usar tablas *MyISAM* o *InnpoDB* para las tablas de esta base de datos.

- MEMORIA

  Ajusta las variables correspondientes para optimizar el uso de memoria según el tipo de tabla.

- MONITORIZACIÓN

  ¿Que aspectos del servidor consideras que sería conveniente monitorizar y con qué frecuencia para evitar un decremento en el rendimiento?

RESUMEN DEL CAPÍTULO

En este capítulo se ha tratado de exponer lo esencial en lo que se refiere al la optimización tanto de consultas como del funcionamiento del servidor en general. Es un tema complejo puesto que cada sistema es distinto y requiere un "tuneado" propio.

Usar un servidor requiere estar al tanto de cada detalle de configuración para que el usuario no perciba errores en el desempeño. Hemos estudiado las variables más relevantes para el mejor uso de la memoria y del disco. También hemos cubierto con cierto detalle el tema de optimización de consultas, sin duda el principal caballo de batalla de cualquier administrador.

Se han propuesto además a través de las distintas actividades ejercicios para la optimización de toro tipo de consultas.

Hemos hecho un repaso de los comandos más relevantes para la monitorización del servidor y su estado así como de las principales herramienta en disponibles para tal fin.

Por último, los ejemplos expuesto y propuestos serán sin duda una fuente de adquisición de la experiencia mínima necesaria para iniciarse en el mundo de la optimización.

EJERCICIOS PROPUESTOS

- **1.** Usa el comando *explain* para explicar razonadamente las diferencias de rendimiento entre usar % en consultas *like*, usando el carácter comodín (%) antes y después de la palabra a buscar y usándolo solo después. Hazlo para la consulta que busca los datos de las noticias cuyo título contiene la palabra motor y para aquellas cuyo título empieza por *Motor*.
- **2.** La mayoría de tus usuarios buscan noticias de tu blog por el contenido, que índice crees conveniente y de que tipo para facilitárselo? Indica el comando necesario para crearlo.
- **3.** Crea la tabla *t2* en la base de datos test con los campos *c1*, *c2* y *c3*. Agrega un índice sobre dicha tabla llamado *k1* con los campos *c1* y *c2*. Observa el resultado de *explain* sobre las siguientes consultas en la tabla *t2* de la base test teniendo en cuenta sus índices y explícalo.
  - SELECT * FROM t1 FORCE INDEX(k1) WHERE c1=1 ORDER BY c2;
  - SELECT * FROM t1 ORDER BY c1 DESC, c2 ASC;
  - SELECT c1 FROM t1 ORDER BY pow(c1,2);
- **4.** Optimiza las siguientes consultas suponiendo que se hacen con frecuencia y que no hay índices creados sobre las tablas afectadas salvo que se diga explícitamente:
  - SELECT titulo FROM noticias WHERE id BETWEEN 10 and 1000;
  - SELECT c2, c1 FROM test.t1 ORDER BY c2, c1; (supón que c1 y c2 forman una clave en ese orden)
  - SELECT c.cod_cuenta,cl.nombre,cl.dni,cl.nombre FROM cuenta c STRAIGHT_JOIN cliente cl ON c.cod_cliente=cl.codigo_cliente ORDER BY c.fecha_creacion, c.cod_cliente desc;
  - SELECT e.ciudad, count(j.id_jugador) AS njugadores FROM equipo e
  - STRAIGHT_JOIN jugador j ON e.nombre=j.equipo
  - GROUP BY e.ciudad, e.puesto, e.nombre;
- **5.** Indica cómo optimizar el uso de *load data infile* en tablas *MyISAM* con muchos índices usando el comandos *mysqladmin* y *myisamchk* y la sección del manual de MySQL donde se optimizan comandos INSERT. Aplícalo a la tabla noticias en la base *motorblog* (vacíala para después recargarla. Crea previamente una copia de seguridad de sus datos en un fichero) y comprueba la diferencia de tiempos.
- **6.** En las siguientes situaciones, indica los comandos *SHOW* (con la cláusula *LIKE* correspondiente) que usarías para determinar las variables del sistema (tanto de servidor como de estado) a considerar para mejorar el rendimiento:
  - Aplicación con muchas actualizaciones (*delete*, *insert*, *update*).
  - Aplicación con muchas conexiones y consultas.
  - Aplicación con mucho volumen de datos.
- **7.** Usa una máquina virtual con Linux y comprueba la salida del comando *mysqlreport*. Observa el resultado e interpreta la sección *'Questions'*.
- **8.** Usa *mysqlslap* para simular una consulta de todas las noticias de la base de datos blog con una concurrencia de 10000 y 20000 iteraciones.

TEST DE CONOCIMIENTOS

1 ¿Cómo se almacenan los índices en tablas *InnoDB*?
- a) Solo los valores con su dirección de memoria.
- b) Las almacena directamente en el disco duro.
- c) Se guardan en un fichero junto con los de las tablas *MyISAM*.
- d) Se almacenan en ficheros separados.

2 ¿Cuál de los siguientes no es un tipo de índice?
- a) *Btree*.
- b) *Hash*.
- c) *Spatial*.
- d) *Fulltext*.

3 ¿Cuántos accesos a disco requiere la consulta: *select * from equipo, jugador, partido*?
- a) Muchos.
- b) Pocos.
- c) 3.
- d) 2.

4 ¿Qué significa *using temporary* en el contexto del comando *explain*?
- a) Que hay que esperar un poco.
- b) Necesita usar el directorio temporal.
- c) Se requiere una tabla temporal en memoria.
- d) Se requiere una tabla temporal en disco.

5 ¿Por qué podría interesar no usar la caché en una consulta?
- a) Para obtener datos fiables.
- b) Poruqe cambian mucho los datos.
- c) Para tardar más tiempo.
- d) Todo lo anterior.

6 Los índices en MySQL:
- a) Siempre están ordenados.
- b) Están ordenados ascendentemente.
- c) Están ordenados descendentemente.
- d) Se puede elegir el tipo de ordenación.

7 Las tablas tipo *memory*:
- a) Guardan índices.
- b) Existen solo en memoria.
- c) Son copias de trozos de tablas.
- d) Están en desuso.

8 ¿Qué programas que sirvan para la monitorización reconoces?
- a) *Mtop*.
- b) *Mytop*.
- c) *Mysql sandbox*.
- d) *Mysqltest*.

9 Tu memoria está saturada. ¿Qué variables de entorno deberías contemplar?
- a) *key_buffer_size*.
- b) *read_buffer_size*.
- c) *net_buffer_length*.
- d) *thread_cache_size*.

10 Tu aplicación va a tener muchas consultas, ¿qué variables debes monitorizar?
- a) *thread_cache_size*.
- b) *threads_cached*.
- c) *threads_created*.
- d) *max_connections*.

6 Bases de datos distribuidas y alta disponibilidad

OBJETIVOS DEL CAPÍTULO

- ✓ Conocer los conceptos de los sistemas distribuidos.
- ✓ Entender la replicación, alta disponibilidad y balanceo de carga.
- ✓ Implementar un sistema de replicación en MySQL.
- ✓ Diseñar e implementar un cluster usando servidores MySQL.

6.1 CONCEPTOS DE BASES DE DATOS DISTRIBUIDAS

En un sistema de bases de datos centralizado todos los componentes (software, datos y soportes físicos) residen en un único lugar físico. Los clientes (aplicaciones, funciones, programas cliente, usuarios) acceden al sistema a través de distintas interfaces que se conectan al servidor. Sin embargo existe otra arquitectura cada vez más extendida en la que se opta por un esquema distribuido en el que los componentes se distribuyen en distintos computadores comunicados a través de una red de cómputo. Dichos sistemas se conocen con el nombre de Sistemas Gestores de Bases de Datos Distribuidas o DDMGS.

6.1.1 INTRODUCCIÓN

Una base de datos distribuida es una colección de datos que pertenece lógicamente al mismo sistema pero que se almacenan en distintas máquinas conectadas por una red.

El surgimiento de las mismas se debe a varias razones entre las que destacan las siguientes:

- Mejora de rendimiento. Cuando grandes bases de datos se distribuyen en varios sitios las consultas y transacciones que afectan a un solo sitio son más rápidas al ser más pequeña la base de datos local. Los sitios no se ven congestionados por muchas transacciones ya que estas se dispersan entre varios. Por último cuando una transacción requiera acceso a más de un sitio, estos pueden hacerse en paralelo de forma que se reduce el tiempo de respuesta.
- Fiabilidad. Definida como la probabilidad de que un sistema esté activo en un tiempo dado. Al distribuirse los datos es más fácil asegurar la fiabilidad del sistema ya que si falla algún sitio es poco probable que afecte a la mayoría de transacciones.
- Disponibilidad. Probabilidad de que un sistema este activo de manera continuada durante un tiempo. Se ve mejorada por las mismas razones expuestas anteriormente. Además, replicar datos puede mejorar ambos aspectos.
- Tipos de aplicaciones. Muchas aplicaciones tienen un marcado carácter distribuido al estar sus componentes dispersos en distintos sitios. Por ejemplo una cadena de supermercados puede tener distintas sedes en distintas ciudades de forma que los clientes puedan acceder solamente a sitios y datos de su ciudad
- Por contra distribuir implica una mayor complejidad en el diseño, implementación y gestión de los datos para lo cual un buen SGBDD debe incorporar además de los componentes habituales en un SGBD centralizado lo siguientes:
- Capacidad. Para tener acceso a sitios remotos e intercambiar información con los mismos para la ejecución de consultas y transacciones.
- Diccionario. Disponer de un catálogo o *metainformación* sobre cómo están distribuidos y replicados los datos del sistema distribuido.
- La capacidad de optimizar consultas y transacciones sobre datos que estén en más de un sitio.
- Mantener la integridad en los permisos sobre datos replicados.
- Mantener la consistencia de las copias de un elemento replicado.
- Garantizar la recuperación del sistema en una caída.

Todo ello eleva enormemente la complejidad de tal SGBDD. Debemos añadir además la dificultad en el diseño óptimo de bases de datos distribuidas especialmente en cuanto a las dos decisiones importantes, a saber, qué datos distribuir y qué datos replicar.

6.1.2 ARQUITECTURA DE UN DDBMS

En general en un sistema distribuido disponemos de varias formas o modelos para organizar el software. Se habla de la arquitectura cliente-servidor consistente en dividir el software (la funcionalidad) entre equipos que hacen de clientes y otros que hacen de servidores de manera que podemos tener desde una máquina ligera son disco que se conecta a un servidor hasta un equipo que ejecuta un servidor y cliente. A partir de aquí surgen numerosas posibilidades de organización que se caracterizan por tres parámetros:

Autonomía

Tiene que ver con quién tiene el control sobre que datos del SGBDD, es decir determina el nivel de independencia de los SGBD. Distinguimos entre:

- **Integración fuerte:** un equipo hace de coordinador enviando las solicitudes de información a los equipos donde resida la información
- **Sistema semiautónomo:** los SGBD son independientes pero participan en el conjunto pudiendo compartir parte de sus datos
- **Sistema aislado:** el SGBD no tienen constancia de que existan otros gestores

Distribución

Hace referencia a cómo se distribuyen los datos a lo largo del sistema

- Distribución cero: no hay distribución de datos.
- Cliente-servidor: los datos residen en los servidores mientras los clientes proveen una interfaz de acceso a los mismos además de otras posibles funcionalidades como caché de consultas.
- Servidores colaborativos: no se distingue entre servidores y clientes, cada máquina tiene toda la funcionalidad del SGBDD.

Heterogeneidad

Se refiere a la heterogeneidad de los componentes del sistema en distintos niveles:

- Hardware.
- Comunicaciones.
- Sistema operativo.

Cualquier combinación de estos parámetros determina un modelo arquitectónico distinto para nuestro SGBDD. Mostramos un resumen en la siguiente figura:

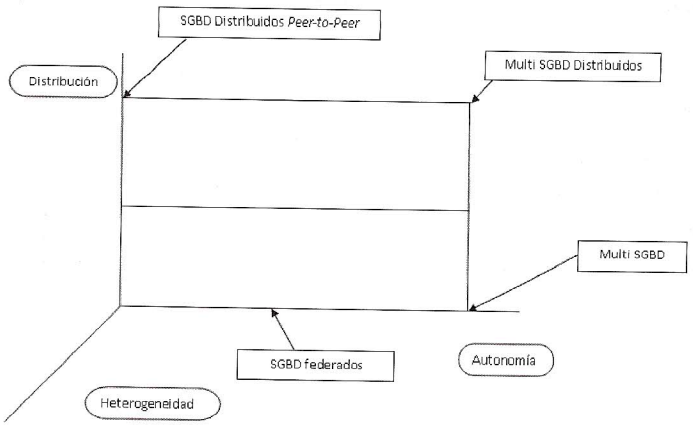


Figura 6.1. *Arquitecturas de SGBDD*

Por otra parte un SGBDD debe proporcionar lo que se conoce como transparencia en la distribución lo que implica que cuando un usuario accede a la base todo lo que ocurre es transparente y queda al margen de cualquier detalle de cómo se ejecuta la transacción, es decir al usuario se le presenta un esquema que no incluye información sobre la distribución de los datos. Es el software cliente quien consulta al catálogo del SGBDD para conocer la ubicación de los datos y poder así descomponer la consulta y distribuirla entre los distintos servidores. En particular la transparencia debe serlo en cuanto dónde están los datos (transparencia de red), que datos están replicados (transparencia de replicación) y que datos están fragmentados (transparencia de fragmentación).

6.1.3 TÉCNICAS DE FRAGMENTACIÓN, REPLICACIÓN Y DISTRIBUCIÓN

Diseñar bases de datos distribuidas implica decidir sobre cómo fragmentarlas, cómo replicarlas y cómo distribuirlas además del diseño previo de la base de datos en sí.

La información sobre cómo se hace este proceso se almacena en el catálogo global del sistema al que tiene acceso el cliente cuando quiere acceder a la información.

Veremos ahora distintas técnicas para hacer lo anterior.

Fragmentación

Consideraremos a nuestra base de datos formada por unidades lógicas que son las relaciones o tablas. Supongamos que en nuestro ejemplo de banca electrónica queremos separar a nuestros clientes por regiones de España para facilitar el acceso. Entonces necesitaríamos dividir la tabla de clientes en tres partes almacenando cada una en un computador, es lo que denominamos "fragmentación horizontal" que divide las *tuplas* según una o varias condiciones sobre distintos atributos, en este caso sobre el campo "región" en empleado.

En otra situación podríamos querer dividir la información de un empleado en sus campos más accedidos y ligeros como nombre, apellidos, dirección y almacenarla en un servidor más potente mientras que otros campos más pasados como currículum, foto, historial etc. Podrían almacenarse en otro servidor con menos capacidad. Es lo que denominamos *fragmentación vertical* y debe hacerse con cuidado ya que tenemos que incluir un atributo común, usualmente una clave, en ambos fragmentos para poder recuperar la información completa cuando sea necesario.

Por supuesto existe también la posibilidad de una fragmentación mixta que incluya los dos casos comentados.

Sin embargo debemos ser conscientes de que fragmentar es un proceso complejo que solo debe hacerse cuando se considere necesario por motivos de eficiencia ya que al hacerlo hacemos también más complejo (más costoso en términos de CPU y consumo de recursos en general) el acceso a los datos por parte de los clientes.

La información sobre la fragmentación se guarda en lo que se denomina un esquema de fragmentación que es una definición de todos los atributos de la base de datos. Un esquema de reparto permite definir donde repartiremos los fragmentos. Si se da el caso de repartirlos en más de un sitio diremos que está replicado.

Fragmentar permite acercar los datos al consumidor, reducir el tráfico de red y mejorar la disponibilidad de los datos.

Replicación

Es fundamentalmente útil para mejorar la disponibilidad de los datos. El caso más extremo es la replicación en todos los sitios de la misma copia de la base de datos que también es el caso que ofrece mayor disponibilidad aunque puede reducir considerablemente la eficiencia en operaciones de actualización dado que cada operación debe realizarse en cada base de datos y esto puede traer problemas de integridad y consistencia de datos. En el extremo opuesto tenemos la no replicación en la que cada fragmento está en un sitio en cuyo caso son disjuntos salvo las claves en fragmentos verticales, es lo que se denomina reparto no redundante. Entre ambos extremos se da una amplia gama de posibilidades que queda descrita lo que se denomina esquema de replicación. Todo ello se describe en el catálogo de replicación dónde se indica que objetos son replicados, dónde está la réplica y cómo se propagan las actualizaciones. El catálogo es un conjunto de tablas guardadas en cada sitio en el que hay réplicas. Las configuraciones posibles son variadas y veremos algún ejemplo en secciones posteriores en este capítulo.

La elección de un sitio para un fragmento depende de los objetivos de rendimiento y disponibilidad para el sistema y de los tipos y frecuencia de transacciones. También depende de si la base es más orientada a consultas o por el contrario hay un alto grado de operaciones de escritura.

La replicación como hemos señalado distribuye la carga, acelera consultas y mejora la disponibilidad pero sobre todo es óptima para sistemas con muchas lecturas sin embargo requiere N veces más actualización y consume N veces mas almacenamiento.

6.1.4 TIPOS DE SISTEMAS DE BASES DISTRIBUIDAS

El término SGBDD engloba muchos tipos de sistemas que pueden ser muy distintos entre sí. Lo que los conecta a todos es el hecho de que en todos ellos los datos y el software gestor están repartidos en diferentes computadoras conectadas por una red de comunicaciones.

Para su clasificación tenemos en cuanta en primer lugar el grado de homogeneidad software que indica hasta que punto todos los equipos, servidores y clientes, comparten el mismo software con la misma funcionalidad diferenciando entre sistemas homogéneos y heterogéneos. Otro factor relacionado es el grado de autonomía local que determina si tenemos que acceder a los datos a través de un único punto (autonomía cero) en cuyo caso se da la impresión al cliente de un sistema de base de datos centralizado, o por el contrario se permite que coexistan varios servidores independientes en el aspecto administrativo pero que a la vez comparten datos entre sí, es lo que se denomina sistema federado o sistema de *multibases* de datos. Este tipo de sistema es centralizado para usuarios locales y distribuido para usuarios globales.

Un tercer tipo de diferencia es el grado de transparencia o de integración de esquemas como ya se señalo en el apartado de arquitectura. Un alto grado de transparencia requiere que el usuario no sea consciente de la organización

de los datos mientras que en sistemas poco transparentes el usuario debe conocer la información de cómo y dónde están los datos para poder acceder a los mismos.

En este último caso es vital tener un buen sistema de nombres para evitar ambigüedades e inconsistencias de datos. El problema es aún mayor en sistemas de multibases de datos donde cada servidor es administrado independientemente y pude contener hasta modelos de datos distintos.

En general es deseable un alto grado de transparencia para minimizar la complejidad desde el punto de vista del usuario y administrador.

6.2 REPLICACIÓN EN MYSQL

El servidor MySQL hace replicación basada en un maestro y uno o más esclavos que no son sino servidores que mantienen en todo momento una copia exacta de los datos del servidor maestro. Este sistema permite distribuir copias de las bases de datos en distintos sitios, balancear la carga entre varios servidores (esclavos) especialmente en aplicaciones de consulta, facilitar las copias de seguridad al evitar tener que hacerlas en el maestro y permitir alta disponibilidad y tolerancia a fallos haciendo que si falla un maestro uno de los esclavos asuma su papel.

6.2.1 INTRODUCCIÓN

MySQL soporta replicación asíncrona unidireccional, es decir un servidor actúa como maestro y uno o más actúan como esclavos. El servidor maestro escribe actualizaciones en el fichero de *log* binario, y mantiene un índice de los ficheros para rastrear las rotaciones de *logs*. Estos *logs* sirven como registros de actualizaciones para enviar a los servidores esclavos. Cuando un esclavo se conecta al maestro, informa al maestro de la posición hasta la que el esclavo ha leído los *logs* en la última actualización satisfactoria. El esclavo recibe cualquier actualización que ha tenido lugar desde entonces, y se bloquea y espera para que el master le envíe nuevas actualizaciones.

La replicación unidireccional tiene beneficios para la robustez, velocidad, y administración del sistema:

-La robustez se incrementa con un escenario maestro/esclavo. En caso de problemas con el maestro, puede sustituirlo con un esclavo.

-Puede conseguirse un mejor tiempo de respuesta dividiendo la carga de consultas de clientes a procesar entre los servidores maestro y esclavo. Se puede enviar consultas SELECT al esclavo para reducir la carga de proceso de consultas del maestro. Sin embargo, las sentencias que modifican datos deben enviarse siempre al maestro, de forma que el maestro y el esclavo no se desincronicen.

Esta estrategia de balanceo de carga es efectiva si dominan consultas que no actualizan datos, aunque este es el caso más habitual.

-Se pueden realizar copias de seguridad usando un servidor esclavo sin molestar al maestro. El maestro continúa procesando actualizaciones mientras se realiza la copia de seguridad.

6.2.2 ARQUITECTURA Y CONFIGURACIÓN

Antes de entrar en detalles de implementación veremos algunas posibilidades que ofrece la replicación en cuanto a configuraciones y disposición del maestro y los esclavos. El esquema habitual es el consistente en un maestro y varios esclavos como en la siguiente figura:

Figura 6.2. *Esquema básico de replicación*

Aunque hay otras variantes como las que indicamos a continuación.

Un maestro, un esclavo

Es el modo más básico, un maestro con uno o varios esclavos. Es la más útil cuando hay pocas escrituras respecto a lecturas. Además el modelo es fácilmente escalable sin más que añadir esclavos. Hay que tener enguanta, eso sí, el ancho de banda ya que si aumenta el ritmo de actualizaciones podemos llegar a saturarlo.

Figura 6.3. *Esquema maestro esclavo*

Maestro dual

No hay esclavos, solamente dos maestros sincronizados. Esto se da en situaciones en que ambos servidores necesitan acceso de escritura a la misma base de datos compartida. En este caso la latencia o tiempo entre operaciones es un factor crítico. Además si la red mediadora es una WAN es mucho más probable que haya pequeños errores e interrupciones que dificulten las tareas de sincronización.

Una variante es la replicación con maestro dual y esclavos propios de cada maestro.

Anillo de replicación

La configuración anterior es en realidad un caso particular de anillo donde hay un conjunto de servidores de tal modo que cada uno es esclavo de un vecino y maestro del otro.

Las ventajas son sobre todo geográficas, cada sitio dispone de un maestro de forma que puede hacer sus propias actualizaciones sin incurrir en latencias de red. Sin embargo es un sistema muy vulnerable ya que un fallo en un elemento cierra el sistema en el punto de ruptura. Esto puede aliviarse añadiendo esclavos a cada maestro.

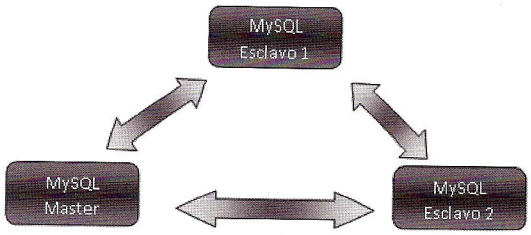


Figura 6.4. *Esquema tipo anillo*

Pirámide

Para organizaciones de gran tamaño y dispersas organizadas de forma piramidal puede haber un único maestro que replique a esclavos dispersos de forma similar a la organización. A su vez los esclavos pueden hacer de maestros para otros esclavos situados en regiones más pequeñas. En la figura representamos el caso más simple.

Aunque hemos visto algunos ejemplos, existen muchas posibilidades y prácticamente no hay límite en la flexibilidad de MySQL al respecto de la replicación aunque normalmente salvo para organizaciones muy grandes suele ser suficiente un esquema simple y escalable.

6.2.3 IMPLEMENTACIÓN

A continuación damos una descripción, en forma de pasos numerados, de cómo inicializar una replicación completa del servidor MySQL. Supondremos que queremos replicar todas las bases de datos en el maestro y no hay una replicación previamente configurada. Necesita parar el servidor maestro brevemente para completar los pasos descritos aquí.

Este procedimiento está escrito en términos de inicializar un esclavo único, pero puede usarlo para inicializar múltiples esclavos.

Antes de comenzar ten en cuenta que este procedimiento y algunos de los comandos de replicación SQL mostrados en secciones posteriores requieren el privilegio *SUPER*.

1. Preparamos una cuenta en el maestro que pueda usar el esclavo para conectar. Esta cuenta debe tener el privilegio *REPLICATION SLAVE*. Si la cuenta se usa solo para replicación (lo que se recomienda), no necesitas dar ningún privilegio adicional.

Supón que tu dominio es *sierra.com* y que quieres crear una cuenta con un nombre de usuario de replicación que puedan usar los esclavos para acceder al maestro desde cualquier equipo en su dominio usando una contraseña. Para crear la cuenta, usamos el comando GRANT:

```
mysql> GRANT REPLICATION SLAVE ON *.*
-> TO 'replicador'@'%.sierrra.com' IDENTIFIED BY 'slavepass';
```

Si quieres usar los comandos *LOAD TABLE FROM MASTER o LOAD DATA FROM MASTER* desde el servidor esclavo, necesitarás dar a esta cuenta privilegios adicionales:

- El privilegio global *SUPER y RELOAD*.
- El privilegio *SELECT* para todas las tablas que quiere cargar. Cualquiera de las tablas maestras desde las que las cuentas no pueden hacer un *SELECT* son ignoradas por *LOAD DATA FROM MASTER*.

Si usas solo tablas MyISAM, vuelque todas las tablas y bloquee los comandos de escritura ejecutando un comando *FLUSH TABLES WITH READ LOCK*:

```
mysql> FLUSH TABLES WITH READ LOCK;
```

Dejamos el cliente en ejecución desde el que lanza el comando *FLUSH TABLES* para que pueda leer los efectos del bloqueo (si sale del cliente, el bloqueo se libera). Luego hacemos una copia de los datos de su servidor maestro.

La forma más fácil de hacerlo es usar un programa de archivo para crear una copia de seguridad binaria de las bases de datos en nuestro directorio de datos del maestro. Por ejemplo, use *tar* en Unix, o *WinRAR*, *WinZip* o cualquier software similar en Windows. Para usar *tar* para crear un archivo que incluya todas las bases de datos, cambiamos la localización al directorio de datos del maestro y ejecutamos el comando:

```
shell> tar -cvf /tmp/mysql-snapshot.tar .
```

Luego copiamos el archivo en un directorio (por ejemplo *tmp*) del servidor esclavo. En esa máquina, desempaquetamos el fichero.

No se necesita incluir ningún fichero de *log* en el archivo, o los ficheros *master.info* o *relay-log.info files*.

Mientras el bloqueo de *FLUSH TABLES WITH READ LOCK* está en efecto, leemos el valor del nombre y el desplazamiento del *log* binario actual en el maestro:

```
mysql > SHOW MASTER STATUS;
+---------------+----------+--------------+------------------+
| File | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+---------------+----------+--------------+------------------+
| mysql-bin.003 | 73 | test | manual,mysql |
+---------------+----------+--------------+------------------+
```

La columna *File* muestra el nombre del *log*, mientras que *Position* muestra el desplazamiento. En este ejemplo, el valor del *log* binario es *mysql-bin.003* y el desplazamiento es 73. Guarde los valores. Los necesitará más tarde cuando inicialice el servidor. Estos representan las coordenadas de la replicación en que el esclavo debe comenzar a procesar nuevas actualizaciones del maestro.

Una vez que tiene los datos y ha guardado el nombre y desplazamiento del *log*, puede reanudar la actividad de escritura en el maestro:

```
mysql> UNLOCK TABLES;
```

Si está usando tablas InnoDB , la forma más rápida de hacer una copia binaria de los datos de las tablas InnoDB es parar el maestro y copiar los ficheros de datos InnoDB, ficheros de *log* y ficheros de definición de tablas (ficheros *.frm*). Para guardar los nombres de ficheros actuales y desplazamientos, debe ejecutar el siguiente comando antes de parar el servidor:

```
mysql> FLUSH TABLES WITH READ LOCK;
mysql> SHOW MASTER STATUS;
```

Luego guarde el nombre del *log* y el desplazamiento desde la salida de *SHOW MASTER STATUS* como se mostró antes. Tras guardar el nombre del *log* y el desplazamiento, pare el servidor sin bloquear las tablas para asegurarse que el servidor para con el conjunto de datos correspondiente al fichero de *log* correspondiente y desplazamiento:

```
C:\> mysqladmin -u root shutdown
```

Debemos asegurarnos que la sección [mysqld] del fichero *my.cnf* en el maestro incluye una opción *log-bin*. Esta sección debe también tener la opción *server-id=master_id*, donde *master_id* debe ser un entero positivo de 1 a 2^32 - 1. Por ejemplo:

```
[mysqld]
log-bin=mysql-bin
server-id=1
```

Si estas opciones no están presentes, añádelas y reinicia el servidor.

Detenemos el servidor que se vaya a usar como esclavo y añadimos lo siguiente al fichero *my.cnf* :

```
[mysqld]
server-id=slave_id
```

El valor *slave_id*, como el valor *master_id*, debe ser un entero positivo de 1 a 2^32 - 1. Además, es muy importante que el *ID* del esclavo sea diferente del *ID* del maestro. Por ejemplo:

```
[mysqld]
server-id=2
```

Si está preparando varios esclavos, cada uno debe tener un valor de *server-id* único que difiera del maestro y de cada uno de los otros esclavos. Piensa en los valores de *server-id* como algo similar a las direcciones IP: estos *IDs* identifican unívocamente cada instancia de servidor en la comunidad de replicación.

Si no especifica un *server-id*, se usa 1 si no ha definido un *master-host*, de otro modo se usa 2. Tenga en cuenta que en caso de omisión de *server-id*, un maestro rechaza conexiones de todos los esclavos, y un esclavo rechaza conectar a un maestro. Por lo tanto, omitir el *server-id* es bueno solo para copias de seguridad con un *log* binario.

Si hemos hecho una copia de seguridad binara de los datos del maestro, lo copiamos en el directorio de datos del esclavo antes de arrancar el esclavo. Hay que asegurarse de que los privilegios en los ficheros y directorios son correctos. El usuario que ejecuta el servidor MySQL debe ser capaz de leer y escribir los ficheros, como en el maestro.

Arrancamos el esclavo. Si hemos estado replicando previamente, arrancamos el esclavo con la opción *--skip-slave-start* para que no intente conectar inmediatamente al maestro. También podemos arrancar el esclavo con la opción *--log-warnings* para obtener más mensajes en el *log* de errores acerca de problemas (por ejemplo, problemas de red o conexiones).

Si hicimos una copia de seguridad de los datos del maestro usando *mysqldump*, cargamos el fichero de volcado en el esclavo:

```
C:\> mysql -u root -p < dump_file.sql
```

Ejecutamos los siguientes comandos en el esclavo, reemplazando los valores de opciones con los valores relevantes para su sistema:

```
mysql> CHANGE MASTER TO
MASTER_HOST='master_host_name',
MASTER_USER='replication_user_name',
MASTER_PASSWORD='replication_password',
MASTER_LOG_FILE='recorded_log_file_name',
MASTER_LOG_POS=recorded_log_position;
```

La siguiente tabla muestra la longitud máxima para las opciones de cadenas de caracteres:

```
MASTER_HOST 60
```

```
MASTER_USER 16
MASTER_PASSWORD 32
MASTER_LOG_FILE 255
```

Arrancamos el servidor esclavo:

```
mysql> START SLAVE;
```

Una vez realizado este procedimiento, el esclavo debe conectar con el maestro y atrapar cualquier actualización que haya ocurrido desde que se obtuvieron los datos.

Cualquier posible mensaje de error en el *log* de errores del esclavo si no es capaz de replicar por ninguna otra razón. Una vez que un esclavo está replicando, puede encontrar en su directorio de datos un fichero llamado *master.info* y otro llamado *relay-log.info*. El esclavo usa estos dos ficheros para saber hasta que punto ha procesado el *log* binario del maestro. No debemos borrar o editar estos ficheros, a no ser que realmente sepamos lo que hacemos y entendamos las implicaciones. Incluso en tal caso, es mejor usar el comando *CHANGE MASTER TO*.

Una vez que tenemos una copia de los datos, podemos usarla para actualizar otros esclavos siguiendo el procedimiento descrito.

Para la mayor durabilidad y consistencia posible en una inicialización de replicación usando InnoDB con transacciones debemos usar *innodb_flush_logs_at_trx_commit*=1, *sync-binlog*=1, y *innodb_safe_binlog* en el fichero *my.cnf* del maestro.

6.2.4 ADMINISTRACIÓN Y MANTENIMIENTO

Una vez configurada la replicación lo más probable es que no haya que hacer mucho más. Sin embargo, como administradores es imprescindible conocer las herramientas que nos van a permitir controlar el estado de la replicación entre otras tareas administrativas. A continuación veremos algunas de ellas.

Monitorización

Una vez en funcionamiento podemos preguntarnos cosas como si están todos los esclavos replicando, si ha habido a algún error de sincronización o si alguno de los esclavos se está retrasando demasiado. Para conocer responder a estas cuestiones disponemos de varios comandos

- Estado del maestro: el comando *show master status* nos informa del estado de replicación del maestro.

  ```
  mysql> SHOW MASTER STATUS \G
  ```

  Cuya salida nos incluye el nombre del archivo y posición del registro binario actual en donde se escribirá la siguiente consulta.

  Con el comando *show master logs* (equivalente a *show binary logs*) conoceremos que registros binarios existen en el disco.

  ```
  mysql> SHOW MASTER LOGS \G
  ```

  Aunque si queremos información mas detallada tendremos que inspeccionar los archivos a mano.

- Estado del esclavo: En este caso disponemos de mas posibilidades en cuanto a la información que nos producen los comandos, en concreto el comando *show slave status.*

```
mysql> SHOW SLAVE STATUS \G
```

Nos muestra un resumen detallado del contenido de los archivos *master.info* y *log.info* además de otros metadatos como los campos *Last_errno* y *Last_error* que dan información de los errores en la replicación más reciente. Los campos más importantes son *Slave_IO_Running* y *Slave_SQL_Running* acerca de si los hilos de E/S y de SQL están funcionando.

```
mysql> SHOW SLAVE HOSTS \G
```

Muestra los *hosts* registrados con el maestro incluyendo su identificador, nombre de *host*, puerto e *id* del maestro.

Rotación del registro

Ya hemos visto como mostrar ficheros binarios. Sin embargo estos se van acumulando así que si queremos eliminarlos usaremos purgue master *logs to* "numero de registro".

```
mysql> PURGUE MASTER LOGS TO 'BIN003' \G
```

Elimina todos los registros anteriores al 3.

Sin embargo es necesario tener algún sistema de autoamatizar estas tareas haciendo cierta rotación de archivos de forma que siempre haya un máximo determinado y se vayan borrando automáticamente.

- Cambiar de maestro: Es posible cambiar de maestro usando el comando *change master to* opciones. Para ello debemos seguir los siguientes pasos:

1. Desconectar clientes.
2. Asegurarnos de que maestro y esclavos están bien sincronizados.
3. Ejecutar *reset master* en el nuevo maestro.
4. Asegurarnos de que el nuevo maestro está sincronizado con los esclavos.
5. Apagar el antiguo maestro.
6. Ejecutar *change master to* con las opciones correspondientes tal y como vimos en la sección anterior.

```
mysql> CHANGE MASTER TO option [, option] ...

option:
    MASTER_HOST = 'host_name'
  | MASTER_USER = 'user_name'
  | MASTER_PASSWORD = 'password'
  | MASTER_PORT = port_num
  | MASTER_CONNECT_RETRY = interval
  | MASTER_LOG_FILE = 'master_log_name'
  | MASTER_LOG_POS = master_log_pos
```

Si tenemos que cambiar de maestro por que se ha venido abajo deberemos usar como nuevo maestro el esclavo más actualizado (con *show slave status*) para saber el nombre del registro y la posición y después ejecutar *change master to* en el y el resto de esclavos.

✓ mysqlbinlog

Además de comandos para monitorizar el estado de la replicación disponemos de la herramienta *mysqlbinlog* que es un ejecutable incluido en la carpeta *bin* de la instalación de MySQL que permite visualizar registros binarios. Su uso es muy simple, solo hay que añadir al comando el nombre de archivo de registro binario que queremos ver.

Si el fichero es grande y queremos inspeccionarlo con comodidad podemos volcarlo a un fichero. También es conveniente usar expresiones regulares si queremos hacer búsquedas detalladas.

```
C:\> mysqlbinlog logbin001
```

Muestra entre otros datos, para cada consulta la posición dentro del registro, la fecha, el servidor que primero la ejecutó y el hilo encargado.

También se puede ejecutar en servidores remotos incluyendo la información de *host usuario* y *password*.

```
C:\> mysqlbinlog logbin001 -h esclavo -u usuario -p password
```

ACTIVIDADES 6.1

- Realiza un esquema de replicación usando tres equipos, un maestro y dos esclavos
- (puedes usar un solo equipo con varias instalaciones de MySQL, varios equipos del aula o varios equipos virtuales).
- Explica lo que debe hacerse para agregar un esclavo al sistema anterior.
- Supón que cae el maestro, indica los pasos que consideras necesarios para sustituirlo por un esclavo.
- ¿Qué limitaciones y ventajas observas en la replicación de bases de datos?
- Modifica el esquema anterior para organizar un esquema de dos niveles, haciendo que uno de los esclavos a su vez sea maestro de otro esclavo.
- ¿En qué te basarías para determinar el número óptimo de esclavos?

6.3 BALANCEO DE CARGA Y ALTA DISPONIBILIDAD

Una vez conocido el proceso de replicación pasamos al proceso de balanceo de carga y alta disponibilidad.

El primer concepto hace referencia a la necesidad de repartir la carga entre varios equipos o servidores cuando hay un gran volumen de accesos. En función del tipo de acceso puede variar el tipo de balanceo.

En cuanto a la alta disponibilidad se refiere al hecho de disponer en todo momento de todos los datos y de forma transparente desde el punto de vista del cliente. Es decir, el funcionamiento del *cluster* debe permanecer transparente al cliente con independencia del estado de los servidores internos.

6.3.1 FUNDAMENTOS

Hay diversas soluciones para proveer balanceo de carga y alta disponibilidad. Para el caso de MySQL una de ellas es la replicación. Este sistema se adapta muy bien para aplicaciones que requieren un gran número de leturas tal como se observa en la siguiente figura:

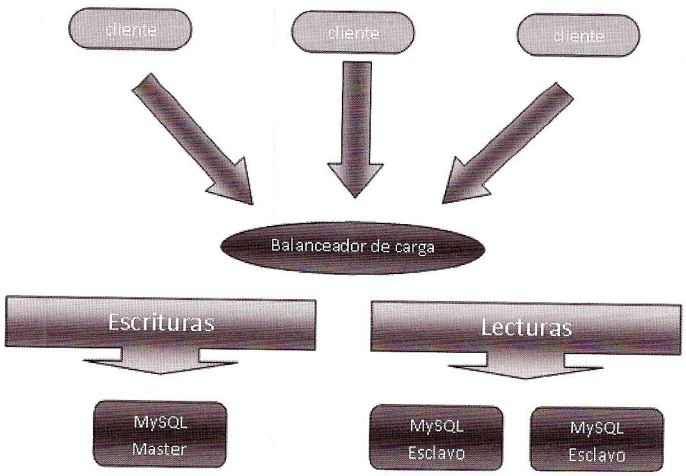


Figura 6.5. *Balanceo de carga con replicación*

En este caso los accesos por parte de los servidores web (que a su vez requieren acceder a bases de datos) se reparten la carga de datos usando un balanceador de carga de MySQL en la parte de lectura y un master en la parte de escritura.

Los principales objetivos de un balanceador en general son:

- **Escalabilidad**.

  El aumento de potencia se consigue simplemente añadiendo más elementos al *cluster* en un proceso lineal aunque en ocasiones dependerá de la aplicación.

- **Eficiencia**.

  El *cluster* se encarga de gestionar las peticiones de forma que se compense la carga de cada máquina y estén equilibradas. Esto es especialmente importante cuando usamos máquinas de distinta potencia y características.

- **Disponibilidad**.

  En éste tipo de *cluster* todos los datos están replicados entre los esclavos de forma que la pérdida de cualquiera de ellos es transparente a los clientes.

- **Transparencia**.

  El hecho de que haya un *cluster* o un solo servidor es inapreciable desde el punto de vista de los clientes.

Otra solución más potente es el uso del cluster de MySQL que además permite particionar los datos en distintos nodos o equipos evitando tener que replicar toda la información. Su arquitectura lo hace especialmente útil en aplicaciones con muchas escrituras.

6.3.2 MYSQL CLUSTER

MySQL *Cluster* es una versión de MySQL que ofrece alta disponibilidad y balanceo de carga en un entorno de computación distribuida. Usa el motor de almacenamiento NDB (*Network Database*) *Cluster* para permitir la ejecución de varios servidores MySQL en un *cluster*. Integra un servidor MySQL estándar con el motor *ndb* que se carga completamente en memoria.

Un *cluster* es una serie de computadoras o *hosts* que ejecutan uno o más procesos llamados *nodos*, que pueden ser servidores MySQL (para acceder a datos), nodos de datos (con los datos) y servidores de coordinación. Adicionalmente otro software nos puede facilitar el acceso a los datos.

A continuación mostramos un esquema ilustrativo:

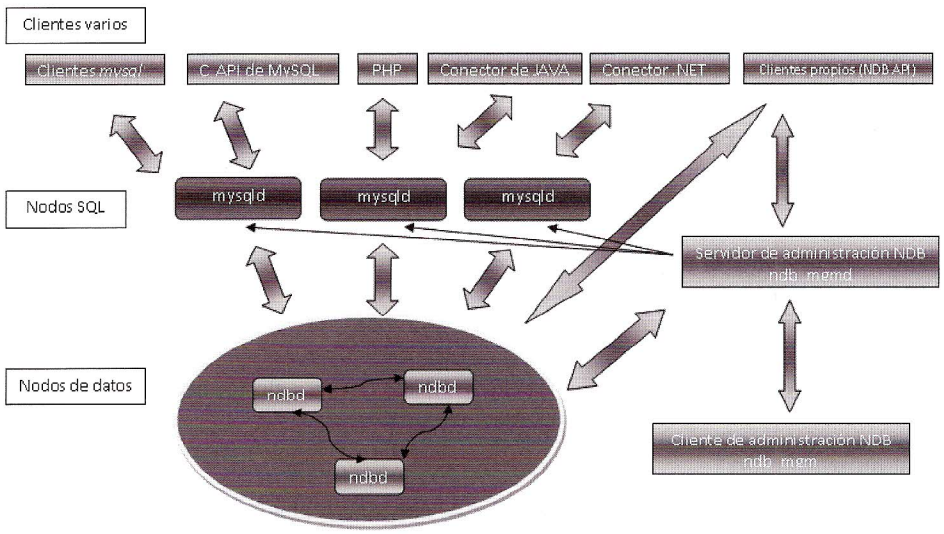


Figura 6.6. *Esquema general de un cluster con MySQL*

Todos los programas, que no tienen porque estar en distintos equipos, trabajan juntos y coordinados formando un *cluster* de bases de datos.

Los datos se guardan en los nodos de datos y son accedidos mediante los nodos de aplicación (nodos API) o servidores *mysqld* (distinto del habitual *mysqld*). Todos los cambios en los datos son percibidos inmediatamente por todos los nodos de acceso a datos.

Antes de pasar a la práctica describiremos brevemente estos y otros elementos importantes manejados por el *cluster*.

Nodos del cluster

- **Nodo de administración** (*Management node*).

  Es el nodo principal que permite controlar y gestionar al resto. Se lanza con el comando *ndb_mgmd*.

  Contiene la configuración global del *cluster* y cualquier evento ocurrido en el *cluster* se registra en él. Así mismo el resto de nodos obtienen su configuración particular del nodo gestor.

- **Nodo de datos** (*Data node*).

  Se encarga de almacenar los datos. Su número es igual al número de réplicas multiplicado por el número de fragmentos. Hablaremos sobre este tema en la siguiente sección. Una réplica es suficiente para funcionar aunque se recomienda usar al menos dos si queremos redundancia y alta disponibilidad real.

  Se lanza con el comando *ndbd*.

- **Nodos SQL** (SQL *node*).

  Son los nodos que acceden a los datos del *cluster*, normalmente servidores MySQL que usan el motor *ndb*. Es decir, son servidores MySQL lanzados con las opciones *–ndbcluster* y *–ndb-connectstring* entre otras. Es una caso especial de API *node*. Otros válidos pueden ser *ndb-restore* para restaurar una copia de seguridad del *cluster* o aplicaciones escritas usando la API del *cluster*.

Aplicaciones cliente

- **Clientes estándar**.

  El *cluster* puede interactuar con aplicaciones escritas en lenguajes como *php* o *Java* igual que en el caso del servidor MySQL. Dicha interacción tiene lugar usando nodos *sql* como intermediarios.

- **Clientes del gestor**.

  Son programas que acceden al nodo gestor para operaciones de mantenimiento y administración. Se escriben usando la MGM API o interfaz de gestión hecha para el lenguaje C.

- **Registro de eventos**.

  El *cluster* almacena los eventos clasificados por categoría. Se dan dos clases, los *log* del *cluster* para eventos que afectan a todo el *cluster* y los *log* de nodo para eventos particulares.

- **Puntos de comprobación**.

  Son registros de tiempo en que todas las transacciones han sido grabadas a disco por lo que no hay posibilidad de fallo. Se dividen en los globales que afectan al *cluster* y los locales que afectan a cada nodo.

6.3.3 ORGANIZACIÓN DE LOS DATOS

- **Partición**.

  Es cada porción de datos en que se dividen los datos del *cluster*.

- **Réplica**.

  Cada réplica es una copia de una partición de datos. Su número es el número de nodos por el número de nodos de grupo.

- **Nodo de datos**.

  Almacena réplicas de datos. Hay tantos nodos de datos como particiones aunque pueden albergar réplicas de otras particiones.

- **Nodo**.

  Almacena un conjunto de réplicas asignadas por su grupo de nodo.

✓ **Grupo de nodo**.

Almacena una copia completa de los datos divididos en particiones replicadas. Su número es igual al número de nodos del *cluster* dividido por el número de réplicas.

El esquema de fragmentación de datos lo hace el *cluster* automáticamente aunque se puede definir por el usuario con ciertas limitaciones.

Se puede ver el esquema general en el siguiente ejemplo:

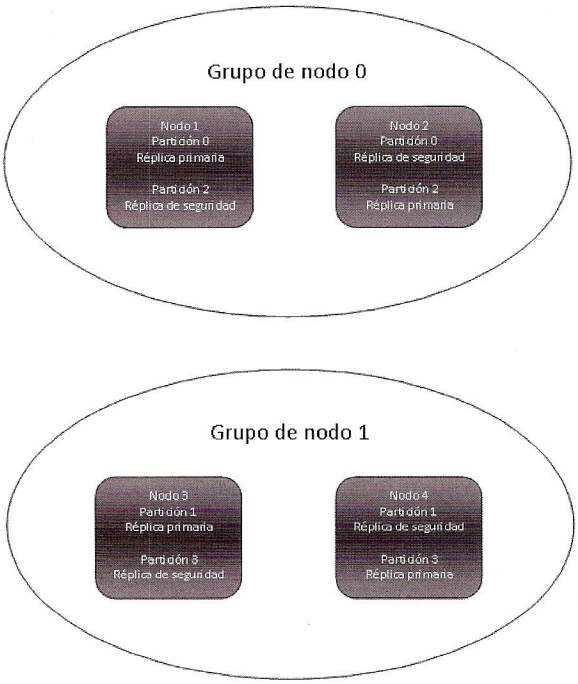


Figura 6.7. *Particionado de datos del cluster*

Todos los datos se dividen en 4 particiones, la 0 y la 2 en el grupo de nodo 0 y la 1 y 3 en el grupo 1. El nodo 1 alberga la réplica primaria de la partición 0 y el 2 la de la partición 2 y una copia de la 0. Ocurre de manera similar en el grupo 1.

Con este esquema se consigue que siempre que al menos un nodo de cada grupo esté funcionando dispondremos de una copia completa de todos los datos.

6.3.4 INSTALACIÓN Y CONFIGURACIÓN DEL CLUSTER

En este apartado veremos como poner en marcha un *cluster* en varios equipos abarcando desde la obtención del programa hasta su instalación y configuración.

Usaremos como guía un ejemplo disponible también en el manual de MySQL. Daremos por supuesto que usamos equipos estándar sin limitaciones especiales de memoria, CPU o hardware en general.

Para poder hacer una prueba sin necesidad de equipos adicionales se recomienda usar un software de virtualización tal como *virtualbox* o *vmware* de forma que podamos instalar varios equipos en nuestro equipo físico. Para ello se requiere un mínimo de memoria RAM. Con los equipos disponibles hoy en día no es de esperar que haya ningun problema a este respecto.

Supondremos un *cluster* básico de cuatro nodos en distintos equipos:

Nodo mgmd	192.168.0.10
Nodo sql:	192.168.0.20
Nodo de datos A	192.168.0.30
Nodo de datos A	192.168.0.40

Visto esquemáticamente quedaría así:

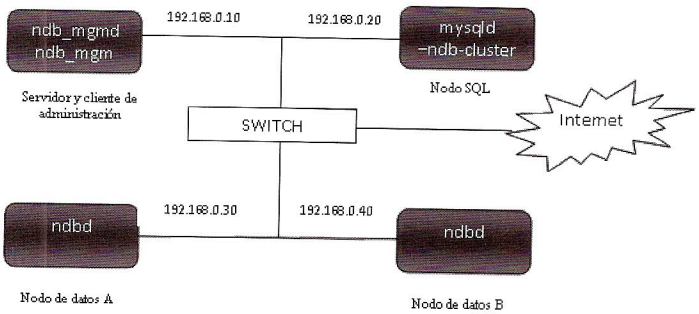


Figura 6.8. *Red de equipos del cluster*

Instalación

Lo primero es descargar el software de la página oficial de MySQL. El nombre será algo similar a *mysql-cluster-gpl-noinstall-ver-winarch.zip* donde ver hace referencia a la versión.

Una vez descargado lo descomprimimos en el directorio de instalación, por ejemplo *C:\mysql*.

Configuración

Salvo el nodo administrador cada nodo participante en un *cluster* tiene su propia configuración más una sección correspondiente en el fichero global de configuración (*config.ini*) del nodo administrador.

Igual que ocurría con el servidor mysql normal disponemos de variables de entorno que nos permiten establecer el comportamiento del servidor y asi adaptarlo y optimizarlo según nuestras necesidades.

En general veremos las variables más importantes que afectan al funcionamiento del *cluster*. Para el resto de variables y otra información adicional debemos consultar el manual de referencia.

Para que las modificaciones de las variables tengan efecto tenemos tres opciones que indicaremos con una letra:

- N- *Rolling restart*. Debemos reiniciar cada nodo uno a uno de forma que el *cluster* esté siempre activo.
- S *system restart*. Todo el *cluster* debe apagarse y reiniciarse tras hacer los cambios correspondientes.
- I *initial restart*. En este caso los nodos de datos deben iniciarse con la opción *–initial*.

Una de las opciones de configuración común a todos los nodos salvo el administrador es *connectstring*. Ésta tiene la siguiente sintaxis:

```
[nodeid=node_id, ]
    [bind-address=host-definition, ]
    host-definition[; bind-address=host-definition]
    host-definition[; bind-address=host-definition]
    [, ...]]

host-definition:
    host_name[:port_number]
```

Que debe incluirse en los respectivos archivos *ini* de configuración.

nodeid hace referencia al identificador del nodo en el *cluster*, *bind-address* es la *ip* asignada que se quiere usar para la conexión al nodo de administración (para el caso de un *host* con varias *interfaces*) y *host-definition* es el nombre del *host* y opcionalmente su puerto.

Configuración para los nodos SQL

Estos nodos admiten una serie de opciones en las secciones *[SQL]* o *[api]* del fichero *config.ini* del equipo que contiene el nodo administrador.

Las más importantes son *HostName* para especificar la IP o nombre del equipo que hace de nodo (es equivalente a *ExecuteOnHostComputer* auque esta solo permite nombres) y el *id* o identificador del nodo dentro del *cluster*. Las primeras son de tipo S mientras que *id* es de tipo N.

En el otro lado, fichero my.ini, disponemos de numerosas opciones. Indicamos las más relevantes:

- *ndb_config_from_host* / *ndb_config_from_port*: especifica el *hostname* o ip/puerto del nodo administrador de donde obtiene su configuración.
- *ndb_log_bin:* registra actualizaciones en las bases de datos siempre y cuando el *cluster* tenga está opción activada. Por defecto está a *ON* o 1.

En general para conocer las variables que afectan a servidores MySQL del *cluster* podemos ejecutar los comandos siguientes:

Para variables de estado:

```
mysql>SHOW STATUS VARIABLES LIKE 'ndb%'
```

Para variables de sistema:

```
mysql>SHOW VARIABLES LIKE 'ndb%'
```

En cuanto al fichero *my.ini* las opciones imprescindibles son *ndbcluster* y *ndb-connecstring*:

```
[mysqld]
#Opciones para mysqld:
ndbcluster
ndb-connectstring=192.168.0.10
```

Que hacen que el servidor se ejecute dentro del *cluster*.

Para los nodos de datos

En el fichero *config.ini* del nodo administrador configuramos los nodos de datos en la sección *[ndbd]* para valores específicos o *[ndbd default]* para valores compartidos por todos los nodos.

Igual que los nodos SQL requiere el uso de las variables *ExecuteOnComputer* o *HostName*. Para el caso particular de nodos de datos también es obligatorio especificar el número de réplicas con *NoOfReplicas*.

- *Id*: es el identificador del nodo en el *cluster*.
- *ServerPort*: es el puerto de escucha aunque por defecto el *cluster* asigna puertos dinámicamente.
- *NodeGroup*: especifica el grupo al que pertenece un nodo. El *cluster* no obstante crea grupos automáticamente así que este parámetro es solo necesario cuando queremos añadir un nuevo nodo a un *cluster* en funcionamiento. Es de tipo I y S.
- *NoOfReplicas*: solo válido en la sección *ndb_default* establece el número de réplicas que debe ser un valor no superior a 4 y divisible por el número de nodos de datos. Este valor determina el número de nodos en cada grupo de nodos. Es de tipo I y S.
- *DataDir*: especifica el directorio de datos donde se almacenan ficheros de *log* y de error.
- *DataMemory*: establece la cantidad de espacio en *bytes* disponible para cada registro de datos y de índices. Es importante tener en cuenta que todos los datos se cargan en memoria cuando el *cluster* está activo. Su valor por defecto es 80 MB y el valor mínimo 1 MB. El máximo está condicionado por la RAM disponible. Es de tipo N.
- *IndexMemory*: determina el espacio que ocupan los índices tipo *hash* usados típicamente para claves primarias y secundarias. Su valor por defecto es 18MB y el mínimo 1MB. Es de tipo N.

Los nodos de datos requieren únicamente el programa *ndbd.exe* o *ndbmtd.exe*. usaremos el primero aunque lo mismo se aplica en el segundo.

En cada nodo debemos tener los directorios *C:\mysql*, *C:\mysql\bin* y *C:\mysql\cluster-data*. Copiaremos entonces el ficheros *ndbd.exe* del equipo donde descargamos el *zip* en cada directorio *C:\mysql\bin* de los nodos de datos. Y añadiremos lo siguiente en sus respectivos *C:\mysql\my.ini*.

```
[mysql_cluster]
# Opciones para nodos de datos:
ndb-connectstring=192.168.0.10
```

Para el nodo de administración

Las variables más relevantes son los ya vistos *DataDir*, *ExecuteOnComputer*, *Hostname* e *Id*.

Además podemos indicar el tipo de *logging* con *LogDestination* cuyos valores pueden ser:

- *Console*: para consola.
- *Syslog*: para sistemas Unix.
- *File*: opción por defecto mediante la cual podemos especificar un fichero, su tamaño máximo (*maxsize*) y el número máximo de ficheros de *log* (*maxfiles*) como en el siguiente ejemplo:

```
FILE:filename=cluster.log,maxsize=1000000,maxfiles=6
```

Todas las variables salvo las *ExecuteOnComputer* y *HostName* son de tipo N.

Existen otras variables dispuestas en diferentes secciones (como [*tcp*] para variables relativas a la red o [*shm*] para memoria compartida) del fichero *config.ini*, sin embargo para una configuración media lo visto es más que suficiente. En todo caso en el manual oficial puedes consultar cualquier detalle adicional.

Para instalar el programa encargado de administrar el *cluster*, como en el caso anterior creamos los mismos directorios y copiamos los programas *ndb_mgmd.exe* (servidor de administración) y *ndb_mgm.exe* (cliente de administración) en el directorio *C:\mysql\bin* del equipo con la IP 192.168.0.10.

Necesitamos crear dos ficheros de configuración. El habitual *my.ini*:

```
[mysql_cluster]
# Opciones para el servidor de administración
config-file=C:/mysql/bin/config.ini
```

Y el fichero de configuración global *config.ini* con la información de la composición del *cluster* y sus nodos

Se recomienda además incluir las IP de cada nodo, el valor de los parámetros *Datamemory* e *Indexmemory*, el número de réplicas y los directorios donde cada nodo de datos almacena datos y *logs* (*DataDir*). Veamos un ejemplo:

```
[ndbd default]

# Opciones que afectan a los nodos de datos:
NoOfReplicas=2 # Numero de réplicas
DataDir=C:/mysql/bin/cluster-data # Directorio de datos de cada nodo de datos
DataMemory=80M # Memoria requerida para almacenamiento de datos
IndexMemory=18M # Memoria requerida para almacenmaiento de índices

[ndb_mgmd]
# Opciones de nodos de administración:
HostName=192.168.0.10 # Hostname o dirección IP del nodo administrador
DataDir=C:/mysql/bin/cluster-logs # Directorio para ficheros log del administrador

[ndbd]
# Opciones de nodos de datos, nodo A:
HostName=192.168.0.30 # Hostname o dirección IP

[ndbd]
# nodo B:
HostName=192.168.0.40
```

```
[mysqld]
# Opciones de nodo SQL:
HostName=192.168.0.20
```

El arranque inicial del *cluster* se reduce al arranque de todos los ejecutables mencionados en su máquina correspondiente. El primero en arrancar debe ser el nodo de administración seguido por los nodos de datos y finalmente los nodos SQL.

Para ello en el nodo de administración ejecutamos la siguiente orden:

```
C:\mysql\bin> ndb_mgmd
2010-06-23 07:53:34 [MgmtSrvr] INFO     -- NDB Cluster Management Server. mysql-5.1.51-
ndb-7.1.10
2010-06-23 07:53:34 [MgmtSrvr] INFO     -- Reading cluster configuration from 'config.
ini'
```

Con algunas líneas más que se imprimen por pantalla. La opción *–f* o *–config-file* indica dónde se encuentra el fichero de configuración.

En los nodos de datos:

```
C:\mysql\bin> ndbd
2010-06-23 07:53:46 [ndbd] INFO       -- Configuration fetched from 'localhost:1186',
generation: 1
```

No debemos arrancar el nodo SQL mientras no terminen los nodos de datos así que mientras podemos ir arrancando el nodo cliente de administración:

```
C:\mysql\bin> ndb_mgm
-- NDB Cluster -- Management Client --
ndb_mgm>
```

Como primera comprobación podemos ejecutar el comando *ALL STATUS*:

```
ndb_mgm> ALL STATUS
```

Que nos devolverá algo como lo siguiente:

```
Connected to Management Server at: localhost:1186
Node 2: starting (Last completed phase 3) (mysql-5.1.51-ndb-7.1.10)
Node 3: starting (Last completed phase 3) (mysql-5.1.51-ndb-7.1.10)

Node 2: starting (Last completed phase 4) (mysql-5.1.51-ndb-7.1.10)
Node 3: starting (Last completed phase 4) (mysql-5.1.51-ndb-7.1.10)

Node 2: Started (version 7.1.10)
Node 3: Started (version 7.1.10)

ndb_mgm>
```

Por último arrancamos el nodo SQL:

```
C:\mysql\bin> mysqld --console
```

Con la opción adicional *–console* por si hubiese errores iniciales.

Ahora ejecutamos en el cliente de administración el comando *SHOW*.

```
ndb_mgm> SHOW
Connected to Management Server at: localhost:1186
Cluster Configuration
---------------------
[ndbd(NDB)] 2 node(s)
id=2 @192.168.0.30 (Version: 5.1.51-ndb-7.1.10, Nodegroup: 0, Master)
id=3 @192.168.0.40 (Version: 5.1.51-ndb-7.1.10, Nodegroup: 0)

[ndb_mgmd(MGM)] 1 node(s)
id=1 @192.168.0.10 (Version: 5.1.51-ndb-7.1.10)

[mysqld(API)] 1 node(s)
id=4 @192.168.0.20 (Version: 5.1.51-ndb-7.1.10)
```

Y para comprobar el que el nodo SQL está conectado podemos usar un cliente de MySQL con la orden *SHOW ENGINE NDB STATUS*.

En todos los casos hemos operado manualmente. Sin embargo si pensamos usar el *cluster* de forma continuada lo mejor es instalarlo como servicio de Windows con los comandos correspondientes. Por ejemplo para instalar el nodo SQL ejecutaríamos *C:/mysql/bin/mysqld –install*. Para después arrancar y detener los servicios usaríamos los comandos NET START Y NET STOP seguidos del nombre del servicio.

Si queremos empezar a usar bases de datos en el *cluster* procederemos de la manera habitual siempre usando como interfaz el nodo SQL tal como si fuese un simple servidor normal y con la única salvedad de que debemos incluir la opción *ENGINE* cuando creamos las tablas.

```
CREATE TABLE tbl_name (col_name column_definitions) ENGINE=NDBCLUSTER;
```

Para finalizar, para cerrar el *cluster* debemos acceder al cliente de administración y ejecutar:

```
C:\mysql\bin> ndb_mgm -e shutdown
```

Lo que finalizará el nodo servidor de administración y todos los nodos de datos. En el nodo SQL se detiene de la forma habitual con el comando *mysqladmin shutdown*. En el caso de que sean servicios de Windows usaremos *NET STOP nombreservicio*.

ACTIVIDADES 6.2

Pon en marcha un *cluster* como el descrito en la sección anterior. Elabora un pequeño manual en *pdf* indicando todos los pasos seguidos.

- Esboza un *seudocódigo* (si conoces algún lenguaje de programación como *perl* o PHP puedes desarrollar el código y probarlo) para proporcionar balanceo de carga a tu sistema de manera manual sin usar MySQL *cluster*.
- ¿Cómo logra MySQL *cluster* la alta disponibilidad?
- Explica lo que entiendes por los conceptos de disponibilidad, rendimiento, balanceo de carga y *clustering*.

6.3.5 GESTIÓN DE MYSQL CLUSTER

Inicio y comandos básicos del cluster

En primer lugar describiremos el proceso de arranque en el que se dan varias posibilidades:

- *Arranque inicial*: es el arranque cuando los nodos de datos están vacíos lo que ocurre la primera vez que iniciamos el *cluster* o bien cuando arrancamos los nodos de datos con la opción *–initial*.
- *System restart*: es el reinicio normal tras haber apagado el *cluster*.
- *Node restart*: es el reinicio de uno de los nodos mientras el *cluster* sigue activo.

Fase 1. Inicialización y establecimiento de variables

En esta fase cada nodo de datos (*ndbd*) obtiene sus parámetros de configuración del nodo administrador. Estas variables incluyen su *id*, puertos para la comunicación internodo y memoria.

El siguiente paso es leer los datos y los ficheros de log por si se requiere restaurar el sistema a uno de los *checkpoints* usando los ficheros de *log*.

Finalmente se establecen los grupos de nodos y el *cluster* empieza a estar activo.

A partir de ahí podemos empezar a usar el cliente administrador del *cluster* (*ndb_mgm*) para gestionarlo. En la siguiente tabla mostramos los comandos más importantes:

Tabla 6.1 Comandos administración *cluster*

help	Muestra la ayuda.
Show	Muestra información del estado del *cluster*.
Node_id start/stop	Inicia/detiene un nodo de datos.
Node_id restart opción	Reinicia el nodo correspondiente con las opciones. *-i* equivalente a iniciar el nodo con *–initial* de forma que se recrea el sistema de ficheros del nodo de datos. *–n* detiene el nodo para iniciarlo solo si se usa el comando start. *-a* hace que se abosten las operaciones pendientes.
Node_id status	Informa sobre el estado del nodo.
Node_id report tipo	Hace un informe de dos tipos: *-backupstatus*: si hay una copia de seguridad en proceso muestra su estado. *-memoryusage*: porcentaje d emoemoira en uso de datos e índices.
Quit exit	Finaliza el cliente de administración.
Shutdown	Termina los nodos de datos y de administración, no los SQL.
Create nodegroup node_id, node_id...	Crea un nuevo grupo de nodos que debe tener los mismos nodos que los grupos existentes.
Drop nodegroup nodegroup_id	Elimina un grupo de nodos del custer aunque estos siguen existiendo como porcesos independientes. Solo funcionará si los nodos no contienen datos.

Copia de seguridad

Una copia de seguridad nos permite tener una copia del estado del *cluster* y sus datos en un momento dado.

Esta información podemos dividirla en tres partes principales:

- *Metadatos*: nombre y definiciones de todas las tablas.
- *Datos de tablas*: los datos de cada tabla almacenada por el nodo (según los fragmentos que contenga).
- *Registro*: de cómo y cuando los datos se guardaron en la base de datos en los fragmentos que almacena el nodo.

Toda la información se encuentra en todos los nodos que participan en el *backup* en los ficheros *BACKUP-backup_id.node_id.ctl*, *BACKUP-backup_id.node_id.data* y *BACKUP-backup_id.node_id.log*, respectivamente.

Para iniciar el *backup* debemos ejecutar el comando *START BACKUP*

```
START BACKUP [backup_id] [wait_option] [snapshot_option]
wait_option:
WAIT {STARTED | COMPLETED} | NOWAIT
snapshot_option:
SNAPSHOTSTART | SNAPSHOTEND
```

Donde *wait_option* determina que hacer una vez lanzado el comando.

- *nowait*: hace que se devuelva el control inmediatamente al cliente administrador.
- *wait started*: el cliente administrador espera a que se inicie el *backup* antes de recibir el control.
- *wait completed*: es la opción por defecto que hace que se devuelva el control cuanodo finalice el *backup*.

La opción *snapshot* permite especificar si el *backup* debe reflejar el estado de las bases de datos en el momento de empezar (*SNAPSHOTSTART*) o en el momento de acabar (*SNAPSHOTSEND*) que es el valor por defecto.

Algunos comandos válidos para iniciar un *backup* desde el cliente *ndb_mgm* o la consola *mysql*:

```
ndb_mgm> START BACKUP
c:\>ndb_mgm -e "START BACKUP 6 WAIT COMPLETED SNAPSHOTSTART"
```

Para detener un *backup* activo:

```
ndb_mgm> ABORT BACKUP backup_id
```

Registro de eventos en el cluster

El *cluster* refleja dos tipos de eventos, los del *cluster* o globales relativos a todos los eventos en el *cluster* y almacenados en el fichero *ndb_node_id_cluster.log* en el directorio donde reside *ndb_mgm*. Las variables que controlan esto son *DataDir* y *LogDestination*.

Por otro lado los registros locales de cada nodo se almacenan en el fichero *ndb_node_id_out.log* en el directorio de datos del nodo particular.

Cada evento registrado se clasifica según tres criterios:

- Categoría del evento calificada por uno de los siguientes valores: *STARTUP*, *SHUTDOWN*, *STATISTICS*, *CHECKPOINT*, *NODERESTART*, *CONNECTION*, *ERROR* o *INFO*.
- Prioridad determinada por un número según su gravedad desde la más grave (1) hasta la más leve (15).

- Nivel de seguridad uno de los siguientes valores *ALERT* (un error que debe corregirse de forma inmediata), *CRITICAL* (condiciones extremas tal como límite de recursos), *ERROR* (fallos en la configuración), *WARNING* (todo correcto pero debemos tener cuidado en cierto aspectos), *INFO* (información normal), o *DEBUG* (mensajes generados en el desarrollo del software).

Cada evento registrado contiene una marca de tiempo, el tipo de nodo, la gravedad del mismo, la *id* del nodo y una descripción como en el siguiente ejemplo:

```
2007-01-26 19:35:55 [MgmSrvr] INFO-- Node 1: Data usage is 2%(60 32K pages of total
2560)
```

Para gestionar eventos disponemos de los siguientes comandos desde el cliente de administración:

- ✓ *CLUSTERLOG ON/OFF*: activa/desactiva el registro de eventos.
- ✓ *CLUSTERLOG INFO*: muestra información sobre los parámetros del *cluster log*.
- ✓ *node_id CLUSTERLOG category*=umbral: registra eventos con prioridad menor o igual a *threshold* (el umbral es por defecto 7 en todos los eventos salvo en el de error que es 15). Los eventos no se registran si la prioridad de su categoría es igual o superior a la prioridad del evento.
- ✓ *CLUSTERLOG FILTER severity_level*: registra eventos del nivel de seguridad especificado.

6.3.6 PROGRAMAS DEL CLUSTER

Existen varios programas que nos permiten realizar distintas tareas relacionadas con la gestión del *cluster*. Están contenidos en el directorio *bin* del mismo y en esta sección haremos una breve descripción de cada uno de ellos y su uso básico exceptuando los ya vistos *mysql*, *ndbd*, *ndb_mgm* y *ndbd_mgm*.

Cualquiera de ellos admite la opción *–help* que permite ampliar el conocimiento de su uso.

mysqld

Es el tradicional proceso del servidor MySQL, para usarse en el *cluster* debe incorporar soporte para el motor *ndbcluster*. En todo caso debemos incluir la opción *ndbcluster* en el fichero de configuración para activar esta característica (en la sección [*mysqld*]). El comando *SHOW ENGINES* nos informará del estado de los motores soportados por el servidor.

Como vimos en secciones anteriores este programa requiere tres piezas de información del *cluster* que son el *id*, el nombre de *host* o *ip* y el puerto donde conectarse al nodo administrador del *cluster* (típicamente 1186). Podemos usar el cliente MySQL para comprobar la conexión al *cluster* con el comando *SHOW PROCESSLIST*.

ndbmtd

Es el usado para gestionar datos en tablas *ndbcluster*. El objetivo es usarlo en ordenadores *multicore*.

Su diferencia es que debe configurarse la opción *MaxNoOfExecutionThreads* en el config.ini del *cluster*. Normalmente se usa el mismo número de *cores*.

ndb_config

Extrae la configuración de para los todos del *cluster* desde un nodo administrador. Admite varias opciones como *–configinfo* para volcar toda la información o *–connections* para volcar solo información de conexiones.

ndb_delete_all

Elimina todas las filas de una tabla *ndb* tal como hace el comando TRUNCATE TABLE. Con la opción *–t* la operación tiene carácter transaccional lo que hace que se ejecute como una sola transacción.

```
ndb_delete_all -c connect_string tbl_name -d db_name
```

ndb_desc

Proporciona información detallada sobre tablas *ndbcluster*. Admite la opción –c para indicar los parámetros de conexión y la opción –c para añadir información de particionamiento.

```
C:>ndb_desc -c connect_string tbl_name -d db_name [-p]
```

ndb_drop_table

Elimina una tabla *ndbcluster*.

```
C:> ndb_drop_index -c connect_string table_name -d db_name
```

ndb_restore

Lee los datos originados en un *backup* del *cluster* y los restaura en la base de datos correspondiente.

Admite entre otras las opciones *--backup_path=path* que indica la ruta al directorio de *backup*, *--connect* para los parámetros de conexión, *--exclude/include-databases=db-list* para excluir/incluir ciertas bases de datos o *–prnt_data* para volcar los datos en *stdout* (o redireccionarlos a un fichero).

ndb_select_all

Vuelca todas las filas de una tabla *ndbcluster* incluyendo detalles referidos a la forma de almacenamiento de los datos.

```
ndb_select_all -c connect_string tbl_name -d db_name [> file_name]
```

ndb_select_count

Muestra el número de filas de una o más tablas del *cluster*.

```
C:>ndb_select_count [-c connect_string] -ddb_name tbl_name[, tbl_name2[, …]]
```

ndb_show-tables

Muestra una lista de todos los objetos de las bases de datos del *cluster* incluyendo índices, *triggers* internos y *cluster* de disco.

```
C:>ndb_show_tables [-c connect_string]
```

ndb_waiter

Muestra repetidamente el estado de los nodos del *cluster* pudiendo dar los valores *no contact* cuando no hay conexión, *unknown* cuando hay contacto pero se desconoce el estado del nodo, *not_started* cuando se ha detenido pero aún no se ha iniciado, *starting* cuando se ha iniciado pero aún no se ha unido al *cluster*, *started* cuando ya es operacional y *shutting_down* cuando está apagándose.

```
C:>ndb_waiter [-c connect_string]
```

ACTIVIDADES 6.3

- Investiga cómo harías una copia de seguridad del los datos del *cluster*.
- ¿Qué pasos debemos seguir para agregar un nuevo nodo al *cluster*?

6.4 CASO BASE

Estudia las posibilidades de crecimiento en cuanto a necesidad de memoria, espacio y procesador en la aplicación *mediaserver* y determina cuando sería conveniente implementar una arquitectura maestro-esclavo.

Determina la conveniencia de usar uno o más esclavos para tu servidor en función del crecimiento previsto. En tal caso investiga cómo implementar un sistema de *fail-over* de manera que cuando caiga un servidor maestro uno de los esclavos le suplante.

Investiga la herramienta MMM (*Multi Master Replication*) de *mysql-mmm.org* y explica cómo puede mejorar tu aplicación y en qué condiciones.

Si tuvieses que hacer un particionado (horizontal) manual de los datos de la aplicación, ¿cuál crees más conveniente y sobre qué tablas? Explica.

A partir de que momento consideras necesario plantearse la necesidad de un *cluster* de MySQL.

RESUMEN DEL CAPÍTULO

Los sistemas de información son cada vez más amplios y crecen más deprisa. Es por ello necesario recurrir a soluciones escalables y eficientes como son la replicación y *clusterización* de aplicaciones para garantizar la eficiencia y la alta disponibilidad en nuestros servicios. Las soluciones en el contexto de MySQL pasa por el uso de la replicación como solución para sistemas de tamaño medio y con relativamente pocos accesos de escritura. En un nivel superior se sitúa el uso de MySQL *cluster*, software que nos proporciona mucha más disponibilidad y flexibilidad a la hora de gestionar nuestras bases de datos ya que proporciona particionado de datos, balanceo de carga y alta disponibilidad de manera automática o mediante configuración manual.

EJERCICIOS PROPUESTOS

- **1.** Explica los conceptos de partición, réplica, grupo y nodo.
- **2.** ¿Cuántos nodos de datos necesitamos en un clúster con tres réplicas y tres grupos de nodos?
- **3.** Forma en clase un *cluster* con cuatro nodos de datos y dos grupos. Comprueba la alta disponibilidad primero apagando un nodo de datos y, después, los nodos de uno de los grupos.
- **4.** Indica las variables del servidor relacionadas con el *cluster*.
- **5.** ¿Qué tipo de particionado se hace en MySQL Cluster?
- **6.** Revisa el artículo *Improving Database Performance with Partitioning* en la url: *http://dev.mysql.com/tech-resources/articles/performance-partitioning.html*. Haz un resumen y comenta las posibilidades de particionado de la tabla noticias en la base *motorblog*. ¿Cómo y en qué condiciones sería conveniente particionarla?
- **7.** ¿Soporta MySQL el particionado vertical?, ¿cómo harías un particionado vertical de la tabla noticias separando el campo contenido del resto?, ¿por qué sería bueno hacerlo?

TEST DE CONOCIMIENTOS

1 ¿Qué es el balanceo de carga?
- a) La gestión de clientes de una base de datos.
- b) Repartir la carga entre varios servidores.
- c) Distribuir bien los datos entre distintos servidores esclavos.
- d) Usar el mismo software en todos los servidores de bases de datos.

2 Cuando fragmentamos datos:
- a) Los dividimos en trozos.
- b) Dividimos una tabla en varios grupos de columnas.
- c) Dividimos una tabla en grupos de filas.
- d) Las dos anteriores sirven.

3 La replicación de datos:
- a) Favorece la disponibilidad.
- b) No tiene sentido pues genera redundancia.
- c) Puede ser inconveniente para sistemas de muchas modificaciones.
- d) Es incompatible con la fragmentación.

4 La replicación en MySQL:
- a) Requiere al menos dos servidores MySQL.
- b) Requiere al menos un maestro y un esclavo.
- c) Sirve para hacer copias de seguridad.
- d) Permite optimizar las consultas.

5 ¿Qué privilegios se requieren para hacer una replicación?
a) *Super* y *process*.
b) *Process* y *load*.
c) *Super* y *load*.
d) *Super* y *load* y *select* sobre las tablas implicadas.

6 ¿Por qué debemos bloquear las tablas MyISAM para lectura en una replicación?
a) Para que no se corrompan.
b) Porque no son transaccionales.
c) Porque podría haber una escritura en el proceso.
d) Para que nadie acceda a las tablas mientras se configura.

7 ¿Qué entendemos por coordenadas de una replicación?
a) El desplazamiento del registro binario.
b) El desplazamiento y el último fichero de *log* binario.
c) La ubicación de los ficheros binarios.
d) La IP y directorio de datos de replicación.

8 Un *cluster* en el contexto de sistemas gestores de bases de datos es:
a) Un conjunto de servidores replicados trabajando coordinadamente.
b) Un conjunto de servidores que trabaja con una sola copia de datos.
c) Un método para balancear la carga del gestor de bases de datos.
d) Un conjunto de programas coordinados para proporcionar balanceo y alta disponibilidad.

9 Un *cluster* mínimo en MySQL incluye:
a) Un servidor de bases de datos y un nodo de datos.
b) Un servidor un nodo y un programa administrador.
c) Al menos dos nodos de datos y un administrador.
d) Un servidor replicado.

10 La opción *IndexMemory* de MySQL:
a) Establece la memoria minima del *cluster*.
b) La memoria mínima del *cluster* para gestionar índices.
c) La memoria máxima para almacenar índices de las tablas *ndb-cluster*.
d) La caché de índices del *cluster*.

APÉNDICE

A

Conectores y APIs de MySQL

Con el tiempo la potencialidad de los gestores va creciendo permitiendo realizar tareas cada vez más complejas tanto en la parte de administración como en otros aspectos como análisis de datos o monitorización. Sin embargo siempre es posible encontrar limitaciones a las posibilidades que ofrecen los SGBD. Para superarlas siempre disponemos de la posibilidad de crear nuestros propios programas o *scripts* de administración. Para tal fin MySQL provee varios posibles conectores y API (*Application Program Interface*) que a continuación describimos brevemente aunque antes aclararemos algunos conceptos importantes sobre el significado de algunos términos:

Esta sección proporciona una introducción a las opciones disponibles a la hora de desarrollar una aplicación PHP que necesite interactuar con una base de datos MySQL.

API

Una Interfaz de Programación de Aplicaciones, o API, declara las clases, métodos, funciones y variables a las que necesitará llamar una aplicación para llevar a cabo una tarea determinada. En el caso de las aplicaciones PHP que necesiten comunicarse con un servidor de bases de datos, las API que se necesitarán se ofrecen generalmente en forma de extensiones de PHP.

Las API pueden ser procedurales u orientadas a objetos. Con una API procedural se realizan llamadas a funciones para llevar a cabo las tareas, mientras con una API orientada a objetos se instancian clases, y entonces se realizan llamadas a métodos de los objetos creados. Entre ambas opciones, la segunda es generalmente la vía recomendada, puesto que está más actualizada y conlleva una mejor organización de código.

Conector

En el contexto de MySQL el término conector hace referencia al software que permite a una aplicación conectarse a un servidor de bases de datos MySQL. MySQL proporciona conectores para ciertos lenguajes, entre ellos PHP.

Si nuestra aplicación PHP necesita comunicarse con un servidor de bases de datos, tendrá que usar el código PHP que realice tareas tales como conectar al servidor de bases de datos, realizar consultas y otras funciones relacionadas con bases de datos. Es necesario tener un software instalado en el sistema que proporcione a la aplicación en PHP la API, que manejará la comunicación entre el servidor de bases de datos y la aplicación. A este software generalmente se le conoce como conector, dado que permite a una aplicación conectar con un servidor de bases de datos.

Driver

Un *driver* es un software diseñado para comunicarse con un tipo específico de servidor de bases de datos. Puede también utilizar una biblioteca externa, como por ejemplo la Biblioteca Cliente de MySQL o el *Driver* nativo de MySQL. Estas bibliotecas generan el protocolo de bajo nivel que se utiliza para comunicarse con el servidor de bases de datos.

A modo de ejemplo, la capa de abstracción de bases de datos Objetos de Datos de PHP (PDO) utilizará alguno de los *drivers* para bases de datos disponibles. Uno de ellos es el *driver* PDO MYSQL, que permite comunicarse con un servidor MySQL.

Extensión

En la documentación de PHP aparece éste término. El código fuente de PHP consiste por un lado de un núcleo, y por otro de extensiones opcionales para el núcleo tales como *mysql* y su versión mejorada *mysqli*.

No deben confundirse los términos API y extensión, dado que una extensión no debe necesariamente proporcionar una API al programador.

PDO

Los Objetos de Datos de PHP o PDO, son una capa de abstracción de bases de datos específicas para aplicaciones PHP. PDO ofrece una API homogénea para las aplicaciones PHP, independientemente del tipo de servidor de bases de datos con el que se vaya a conectar la aplicación. En teoría, si se utiliza la API PDO, se podría cambiar el servidor de bases de datos en uso, por ejemplo de *Firebird* a MySQL, y solo se necesitarían algunos cambios menores en el código PHP.

Otros ejemplos de capas de abstracción de bases de datos son JDBC para aplicaciones Java o DBI para *Perl*.

A continuación listamos los conectores, API y *drivers* que soporta MySQL en el momento de escribir el libro (versión 5.5):

- **Connector/ODBC**: es el conjunto de *drivers* que permiten acceder a bases de datos MySQL desde plataformas Windows, Unix o Mac. Es necesario usarlo cuando queremos acceder a MySQL desde aplicaciones que requieran ODBC (*Open Database Connectivity*) como *Microsoft Officce* o *ColdFusion*. ODBC es un conjunto de métodos para acceder a servidores de bases de datos de programas cliente.
- **Connector/NET**: permite a desarrolladores crear fácilmente aplicaciones *.NET* que requieran conexiones seguras y de alta disponibilidad a MySQL. Es muy utlizado en distintas versiones de *Visual Studio*.
- **Connector/J**: es un *driver* JDBC hecho completamente en Java para el acceso a datos de servidores con independencia del sistema operativo o del servidor al que se accede.
- **Connector/C++**: es un conector para C++. Incluye numerosas clases similares en funcionalidad a los JDBC.
- **Connector/C**: es una librería de C con numerosas funciones para el acceso al servidor.
- **Connector/OpenOffice.org**: es un conector para el software *OpenOffice.org*. es un proyecto *open source* con licencia GPL
- **php/API**: el conjunto de clases, funciones y métodos y variables que permiten acceder a un servidor mediante PHP. Existen dos formas de hacerlo mediante PHP.
  - Extensiones MySQL: conjunto de funciones para acceder al servidor. Dado que es la forma más frecuente de acceso (en aplicaciones pequeñas) mostramos un ejemplo:

```
<?php
// Conectando y seleccionando una base de datos
$conexion = mysql_connect('mysql_host', 'usuario', 'password')
    or die('Imposible conectar: ' . mysql_error());
echo 'Conectado correctamente';
mysql_select_db('ebanca') or die('imposible activar la base de datos seleccionada');

// ejecutando una consulta
$query = 'SELECT * FROM cuentas';
$result = mysql_query($query) or die('error en la consulta' . mysql_error());

// mostrando resultados en HTML
echo "<table>\n";
while ($line = mysql_fetch_array($result, MYSQL_ASSOC)) {
    echo "\t<tr>\n";
    foreach ($line as $col_value) {
        echo "\t\t<td>$col_value</td>\n";
    }
    echo "\t</tr>\n";
```

```
}
echo "</table>\n";

// liberando el resultado
mysql_free_result($result);

// Cerrando la conexión
mysql_close($conexion);
?>
```

En este ejemplo se han usado varias funciones de PHP para acceder a una base de datos del servidor. La primera es *mysql_connect* que recibe los argumentos del servidor el usuario y el *password* para la conexión, a continuación se activa una base de datos *mysql_select_db* y se crea y ejecuta una consulta (*mysql_query*), despues se imprimen los resultados en formato *html* usando la función *mysql_fetch_assoc* para extraer cada fila devuelta por la consulta y finalmente se cierra la conexión con el servidor.

Del mismo modo que ejecutamos consultas podemos hacer cualquier otra operación o comando SQL lo que a su vez nos permite tener una nueva forma de realizar todo tipo de tareas. Además la especial orientación web de PHP permite hacer todo vía web con las ventajas que ello implica.

(Un caso especial es el programa *phpMyadmin* disponible de forma gratuita y desarrollado totalmente en PHP con el objeto de permitir la administración de nuestros servidores y bases de datos vía web).

- Extension mysqli: solo disponible en las últimas versiones de PHP a partir de la 5. Es una versión mejorada (*improved*) de la anterior que incluye mejoras de seguridad y una interfaz de acceso orientada a objetos.

- **Perl API**: es el conjunto de funciones de *perl* para el acceso a MySQL para lo cual se requiere el módulo DBI así como el *database driver DBD::mysql*.

Perl es un lenguaje más potente que PHP para manipulación de cadenas y texto en general mediante las llamadas expresiones regulares, aunque su uso con MySQL está menos extendido que PHP es muy recomendable si se requiere un gran procesamiento de texto.

Podemos ver un ejemplo de conexión a MySQL desde *perl*:

```
#!/usr/bin/perl  -w
use DBI;
#definition of variables
$db="MYTEST";
$host="localhost";
$user="root";
$password="rootpass";  # password de root

#conectar a MySQL
my $dbh   = DBI->connect ("DBI:mysql:database=$db:host=$host",
                          $user,
                          $password)
                          or die "Can't connect to database: $DBI::errstr\n";

#desconectar del servidor
$dbh->disconnect or warn "Disconnection error: $DBI::errstr\n";

exit;
```

Para acabar recordar que existen otros conectores o *drivers* para el acceso a MySQL en lenguajes de gran potencia como *Python*, *Ruby*, *Tcl* y *Eiffel*.

Es importante notar que para ser capaz de abarcar todas las tareas del administrador y por muy completo que sea nuestro servidor de bases de datos es importante el conocimiento de uno o más lenguajes de programación ya que ello permitirá siempre poder cubrir todas las tareas administrativas por complejas que sean.

Normalmente cada sistema gestor suele 'llevarse bien' con un lenguaje, en el caso de MySQL es PHP, para SQL Server es Visual Studio etc., aunque por otra parte cada vez más los sistemas gestores de bases de datos se están convirtiendo en auténticos sistemas operativos abarcando la mayor parte de funcionalidad de los mismos e incluyendo en algunos casos sus propios lenguajes de programación.

ACTIVIDADES A.1

- Instala el programa *appserv* para disponer de PHP y un servidor web *Apache*.
- Implementa el ejemplo de conexión a MySQL desde un programa PHP mostrando los datos de las cuentas de la base de datos *ebanca*.
- Instala *perl* en Windows desde la página de *activestate.com*.
- Instala el paquete DBI usando la herramienta *ppm*. Siguiendo las instrucciones del manual oficial de MySQL instala el *database driver* o DBD de MySQL *DBD:mysqlPP* o *DBD:mysql*.
- Conéctate al servidor MySQL usando esta vez el ejemplo de *perl* y compruébalo creando una base de datos.

APÉNDICE

B

Copias de seguridad y recuperación de bases de datos

Hay muchas formas de hacer copias de seguridad, en primer lugar podemos usar copias directas usando programas como el comando *cp* de Windows o Linux para copiar los ficheros que contienen nuestras tablas en una ubicación segura. Hay herramientas complementarias como *rsync* (*cwrsync* para Windows) para sincronizar archivos de forma remota o *winzip*, *tar* y *gzip* para empaquetar y comprimir las copias. Aún así estos métodos solo funcionan para tablas MyISAM.

No obstante lo mejor es recurrir a las herramientas que proporciona el propio servidor.

Para hacer una copia de una tabla a un nivel SQL, podemos utilizar las instrucciones *SELECT INTO ... OUTFILE* o *BACKUP TABLE*.

Otra técnica para hacer copias de seguridad de una base de datos es utilizar el programa *mysqldump* o, en el caso de linux el *script mysqlhotcopy*.

mysqldump

El cliente *mysqldump* puede utilizarse para volcar una base de datos o colección de bases de datos para copia de seguridad o para transferir datos a otro servidor SQL (no necesariamente un servidor MySQL).

Hay tres formas de invocar *mysqldump*:

```
#> mysqldump [opciones] nombre_de_base_de_datos [tablas]
#> mysqldump [opciones] --databases DB1 [DB2 DB3...]
#> mysqldump [opciones] --all-databases
```

Si no se nombra ninguna tabla o se utiliza la opción *--databases* o *--all-databases*, se vuelcan las bases de datos enteras.

Para obtener una lista de las opciones que soporta nuestra versión de *mysqldump*, ejecutamos *mysqldump --help*.

mysqldump soporta, entre otras, las siguientes opciones:

- *--help*, -?

  Muestra un mensaje de error y sale.

- --add-locks

  Rodea cada volcado de tabla con los comandos *LOCK TABLES* y *UNLOCK TABLES*. Esto provoca inserciones más rápidas cuando el fichero volcado se recarga.

- *--compress*, -C

  Comprime toda la información enviada entre el cliente y el servidor si ambos admiten compresión.

- *--databases*, -B

  Vuelca varias bases de datos. Normalmente, *mysqldump* trata el primer argumento de la línea de comandos como un nombre de base de datos y los siguientes argumentos como nombres de tablas. Con esta opción, trata todos los argumentos como nombres de bases de datos.

- --delete-master-logs

  En servidores de replicación maestros, borra los *logs* binarios tras realizar la operación de volcado.

- *--host*=nombre_de_equipo, -h nombre_de_equipo

  Vuelca datos de un servidor MySQL en el equipo dado. Por defecto el equipo es *localhost*.

- --lock-all-tables, -x

  Bloquea todas las tablas de todas las bases de datos. Esto se consigue estableciendo un bloqueo de lectura global que dura durante todo el volcado. Esta opción desactiva automáticamente *--single-Transaction* y *--lock-tables*.

- --lock-tables, -l

  Bloquea todas las tablas antes de comenzar el volcado. Las tablas se bloquean con *READ LOCAL* para permitir inserciones concurrentes en caso de tablas MyISAM. Para tablas transaccionales como *InnoDB* y BDB, *--single-Transaction* es una opción mucho mejor, ya que no necesita bloquear las tablas.

  Debemos tener en cuenta que al volcar múltiples bases de datos, *--lock-tables* bloquea tablas para cada base de datos separadamente. Así, esta opción no garantiza que las tablas en el fichero volcado sean lógicamente consistentes entre bases de datos. Tablas en bases de datos distintas pueden volcarse en estados completamente distintos.

- *--master-data*[=valor]

  Esta opción causa que se escriba en la salida la posición y el nombre de fichero del *log* binario. Esta opción necesita el permiso *RELOAD* y el *log* binario debe estar activado. Si el valor de la opción es igual a 1, la posición y nombre de fichero se escriben en la salida del volcado en forma de comando *CHANGE MASTER* que hace que un servidor esclavo empiece desde la posición correcta en el log binario del maestro si usa este volcado SQL del maestro para preparar un esclavo. Si el valor de la opción es igual a 2, el comando *CHANGE MASTER* se escribe como un comentario SQL. Ésta es la acción por defecto si se omite valor.

- *--no-create-db,* -n

  Esta opción suprime el *comando CREATE DATABASE /*!32312 IF NOT EXISTS*/ db_name* que se incluye de otro modo en la salida si se especifica las opciones *--databases* o *--all-databases*.

- *--no-data*, -d

  No escribe ningún registro de la tabla. Esto es muy útil si solo quiere obtener un volcado de la estructura de una tabla.

- *--opt*

  Esta opción es una abreviatura; es lo mismo que especificar *--add-drop-table --add-locks --create-options --disable-keys --extended-insert --lock-tables --quick --set-charset*. Causa una operación de volcado rápida y produce un fichero de volcado que puede recargarse en un servidor MySQL rápidamente.

- *--password*[=contraseña], -p[contraseña]

  La contraseña a usar al conectar con el servidor. Si usamos la opción en forma corta (-p), no puede haber un espacio entre la opción y la contraseña. Si omite el valor de contraseña a continuación de la opción *--password* o -p en la línea de comandos, aparece un *prompt* pidiéndola.

- --*port*=número_de_puerto, -P número_de_puerto

  El puerto TCP/IP a usar en la conexión.

- --protocol={TCP | SOCKET | PIPE | MEMORY}

  Protocolo de conexión a usar.

- *--quick*, -q

  Esta opción es útil para volcar tablas grandes. Fuerza *mysqldump* a recibir los registros de una tabla del servidor uno a uno en lugar de recibir el conjunto completo de registros y guardarlos en memoria antes de escribirlos.

- *--result-file*=fichero, -r fichero

  Redirige la salida a un fichero dado. Esta opción debe usarse en Windows, ya que previene que los caracteres de nueva línea '*\n*' se conviertan en secuencias de retorno/nueva línea '*\r\n*'.

- --single-Transaction

  Esta opción realiza un comando *SQL BEGIN* antes de volcar los datos del servidor. Es útil solo con tablas transaccionales tales como las *InnoDB* y *BDB*, ya que vuelca el estado consistente de la base de datos cuando se ejecuta *BEGIN* sin bloquear ninguna aplicación.

  La opción *--single-Transaction* y la opción *--lock-tables* son mutuamente exclusivas, ya que *LOCK TABLES* provoca que cualquier transacción pendiente se confirme implícitamente.

  Para volcar tablas grandes, debemos combinar esta opción con *--quick*.

- *--tab*=ruta, -T ruta

  Produce ficheros con datos separados por tabuladores. Para cada tabla volcada *mysqldump* crea un fichero *nombre_de_tabla.sql* que contiene el comando *CREATE TABLE* que crea la tabla, y un *fichero nombre_de_tabla.txt* que contiene los datos. El valor de esta opción es el directorio en el que escribir los ficheros.

  Por defecto, los ficheros de datos *.txt* se formatean usando tabuladores entre los valores de las columnas y una nueva línea tras cada registro. El formato puede especificarse explícitamente usando las opciones *--fields-xxx y --lines--xxx*.

- *--user*=nombre_de_usuario, -u nombre_de_usuario

  Nombre de usuario MySQL a usar al conectar con el servidor.

- *--verbose*, -v

  Modo explícito. Muestra más información sobre lo que hace el programa.

- --version, -V

  Muestra información de versión y termina.

- --where='condición_where', -w 'condición_where'

  Vuelca solo registros seleccionados por la condición *WHERE* dada.

El uso más común de *mysqldump* es para hacer una copia de seguridad de toda la base de datos, como en el siguiente ejemplo:

```
#>mysqldump --opt nombre_de_base_de_datos > fichero_de_seguridad.sql
```

El siguiente ejemplo muestra cómo volcar el fichero de seguridad de nuevo en el servidor:

```
#> mysql nombre_de_base_de_datos < fichero_de_seguridad.sql
```

El siguiente ejemplo obtiene el mismo resultado que el anterior:

```
#> mysql -e "source /ruta/fichero_de_seguridad.sql" nombre_de_base_de_datos
```

mysqldump es muy útil para poblar bases de datos copiando los datos de un servidor MySQL a otro usando por ejemplo una tubería:

```
#> mysqldump --opt nombre_de_base_de_datos | mysql --host=nombre_de_equipo_remoto -C
nombre_de_base_de_datos
```

También es posible volcar varias bases de datos con un solo comando:

```
#> mysqldump --databases nombre_de_base_de_datos_1 [nombre_de_base_de_datos_2 ...] >
mis_bases_de_datos.sql
```

Si queremos volcar todas las bases de datos, podemos usar la opción *--all-databases*.

```
#> mysqldump --all-databases > todas_las_bases_de_datos.sql
```

Si las tablas se guardan con el motor de almacenamiento *InnoDB*, *mysqldump* proporciona una forma de realizar una copia de seguridad de las mismas (consulte los comandos a continuación). Esta copia de seguridad solo necesita un bloqueo local de todas las tablas (usando *FLUSH TABLES WITH READ LOCK*) al principio del volcado. En cuanto obtiene el bloqueo, se lee el log binario y se libera el bloqueo. Si y solo si un comando de actualización largo está en ejecución cuando se ejecuta *FLUSH...*, el servidor MySQL puede quedar bloqueado hasta que acabe este comando largo, y luego el volcado queda sin ningún bloqueo. Si el servidor MySQL recibe solo comandos de actualización cortos (en el sentido de "poco tiempo de ejecución"), incluso si son muchos, el período inicial de bloqueo no debe ser un problema.

```
#> mysqldump --all-databases --single-Transaction > todas_las_bases_de_datos.sql
```

Para una recuperación en un momento dado (también conocido como *roll-forward*, cuando necesita restaurar una copia de seguridad antigua y recrear los cambios que han ocurrido desde tal copia de seguridad), es útil rotar el *log* binario o al menos conozca las coordinadas del *log* binario que se corresponden con el volcado:

```
#> mysqldump --all-databases --master-data=2 > todas_las_bases_de_datos.sql

#> mysqldump --all-databases --flush-logs --master-data=2 > todas_las_bases_de_datos.
sql
```

El uso simultáneo de *--master-data* y *--single-Transaction* proporciona una forma de hacer copias de seguridad en línea apropiadas para recuperaciones en un momento dado, si las tablas se guardan con el motor de almacenamiento *InnoDB*.

```
#1> mysqldump --single-Transaction --all-databases > backup_domingo_1_PM.sql
```

Esto es una copia de seguridad en línea, sin bloqueos, que no molesta a las lecturas y escrituras de las tablas. Hemos supuesto antes que nuestras tablas son *InnoDB*, así que *--single-Transaction* hace una lectura consistente y garantiza que los datos vistos por *mysqldump* no cambian. (Los cambios hechos por otros clientes a las tablas *InnoDB* no son vistas por el proceso *mysqldump*.) Si también tenemos otros tipos de tablas, debemos suponer que no han sido cambiadas durante la copia de seguridad. Por ejemplo, para las tablas MyISAM en la base de datos *mysql*, debemos suponer que no se estaban haciendo cambios administrativos a las cuentas MySQL durante la copia de seguridad.

El archivo *.sql* resultante producido por el comando *mysqldump* contiene una serie de sentencias *SQL INSERT* que se pueden utilizar para recargar las tablas volcadas más tarde.

Las copias de seguridad completas son necesarias, pero no son siempre convenientes. Producen ficheros muy grandes y llevan tiempo generarse. No son óptimos en el sentido de que cada copia completa sucesiva incluye todos los datos, incluidas las partes que no han cambiado desde el último. Después de realizar una copia de seguridad completa inicial, es más eficiente hacer copias incrementales. Son más pequeñas, y llevan menos tiempo de realización. A cambio, en el momento de la recuperación, no podrá restaurarlo únicamente recargando la copia completa. También deberá procesar las copias incrementales para recuperar los cambios incrementales.

Para hacer copias de seguridad incrementales, necesitamos guardar los cambios incrementales. El servidor MySQL debería ser iniciado siempre con la opción *--log-bin* para que almacene estos cambios en un archivo mientras actualiza los datos. Esta opción activa el registro binario, así que el servidor escribe cada sentencia SQL que actualiza datos en un archivo llamado registro binario de MySQL. En el siguiente ejemplo se crea una copia incremental:

```
#> mysqldump --single-Transaction --flush-logs --master-data=2 --all-databases > backup_
domingo_1_PM.sql
```

Tras ejecutar este comando, el directorio de datos contiene un nuevo archivo de registro binario, *sierra-bin.000007*. El archivo *.sql* resultante contiene estas líneas:

```
-- Position to start replication or point-in-time recovery from
-- CHANGE MASTER TO MASTER_LOG_FILE='sierra-bin.000007',MASTER_LOG_POS=4;
```

Como el comando *mysqldump* ha hecho una copia de seguridad completa, estas líneas significan dos cosas:

El archivo *.sql* contiene todos los cambios hechos antes de cualquier cambio escrito al registro binario *sierra-bin.000007* o posterior.

Todos los cambios registrados tras la copia de seguridad no están presentes en el archivo *.sql*, pero lo están en el registro binario *sierra-bin.000007* o posterior.

El lunes, a las 1 PM, podemos crear una copia de seguridad incremental volcando los registros para comenzar un nuevo registro binario. Por ejemplo, ejecutando un comando *mysqladmin flush-logs* creamos *sierra-bin.000008*. Todos los cambios producidos entre el domingo a la 1 PM cuando se hizo la copia completa, y el lunes a la 1 PM están en el archivo *sierra-bin.000007*. Esta copia incremental es importante, así que es una buena idea copiarla a un lugar seguro. (Por ejemplo, en cinta o DVD, o copiándolo a otra máquina.). Si el martes a la 1 PM, ejecutamos otro comando *mysqladmin flush-logs*, todos los cambios desde el lunes a la 1 PM hasta el martes a la 1 PM están en el archivo *sierra-bin.000008* (que también debería ser copiado a un lugar seguro).

Los registros binarios de MySQL ocupan espacio de disco. Para aligerar espacio, púrguelos de vez en cuando. Una manera de hacerlo es borrar los registros binarios que no se necesiten, como cuando hacemos una copia de seguridad completa:

```
#> mysqldump --single-transaction --flush-logs --master-data=2 --all-databases --delete-
master-logs > backup_domingo_1_PM.sql
```

Debemos tener en cuenta que: Borrar los registros binarios de MySQL con *mysqldump --delete-master-logs* puede ser peligroso si nuestro servidor es un servidor maestro de replicación, porque los servidores esclavos pueden no haber procesado completamente los contenidos del registro binario.

La descripción de la sentencia *PURGE MASTER LOGS* explica lo que debe ser verificado antes de borrar los registros binarios de MySQL.

Usar copias de seguridad para una recuperación

Ahora supongamos que tenemos un fallo catastrófico el miércoles a las 8 AM que requiere restauración de las copias de seguridad. Para recuperarnos, primero restauramos la última copia de seguridad completa que tenemos (la del domingo a la 1 PM). El archivo de copia completo es tan solo una serie de sentencias SQL, así que restaurarlo es muy fácil:

```
#> mysql < backup_domingo_1_PM.sql
```

En este punto, el estado de los datos ha sido restaurado al del domingo a la 1 PM. Para restaurar los datos hechos desde entonces, debemos usar las copias incrementales, es decir los archivos de registro binario *sierra-bin.000007* y *sierra-bin.000008*. Los extraeremos, si es necesario, de allá donde estuviesen guardados, y luego procesaremos sus contenidos de la siguiente manera:

```
#> mysqlbinlog sierra-bin.000007 sierra-bin.000008 | mysql
```

Ahora hemos recuperado los datos hasta su estado del martes a la 1 PM, pero todavía hay cambios que faltan desde esa fecha hasta el momento del fallo. Para no perderlos, deberíamos haber hecho que el servidor MySQL almacenase sus registros MySQL en un lugar seguro (discos RAID, ...) diferente del lugar donde almacena sus archivos de datos, para que estos registros no estuvieran únicamente en el disco destruido. (Es decir, iniciar el servidor con la opción *--log-bin* que especifique una localización en un dispositivo físico diferente del que contiene el directorio de datos. De esta manera, los registros no se pierden aún si el dispositivo que contiene el directorio sí.) Si hubiésemos hecho esto, podríamos tener el archivo *sierra-bin.000009* a mano, y podríamos aplicarlo para restaurar hasta los cambios más recientes sin ninguna pérdida a como estaban en el momento del fallo.

ACTIVIDADES B.1

- Realiza una copia de todas tus bases de datos con *mysqldump*. Detén tu servidor MySQL y elimina el contenido de la carpeta de datos (comprueba la opción *datadir* para asegurarte de donde se encuentra). Restaura la copia.
- Activa el registro binario en tu servidor MySQL en una ubicación distinta al directorio de datos. Realiza una copia de seguridad completa y haz varias operaciones de modificación de datos. Elimina ahora todos los datos como en el ejercicio 1 y restaura por completo todos los datos.
- Repite el ejercicio anterior pero en lugar de eliminar directamente la carpeta, elimina los datos con comandos *DROP*.
- Configura en tus equipos Ubuntu (con la herramienta *cron*) y Windows (puedes usar el comando *at*) para realizar copias de seguridad periódicas cada mes e incrementales mediante el registro binario.
- Añade a lo anterior para hacer copias remotas usando *sshfs* y *scp* después de *mysqldump* y guardando los registros binarios en el equipo remoto.
- Usa la utilidad *mysqlhotcopy* de MySQL en Ubuntu para hacer una copia de las tablas MyISAM (debes convertir primero las tablas InnoDB).

APÉNDICE

C

Instalación de una máquina virtual de Ubuntu en Virtualbox

Algunos ejercicios del libro hacen referencia o requieren el uso de Linux. Dado que Ubuntu es un sistema robusto, de amplio uso y muy bien documentado es muy recomendable incluir este pequeño manual para explicar su instalación. Además hemos incluido la instalación de un sistema Debian ligero basado en la version 5. De este modo podemos fácilmente crear una red heterogénea de varios equipos y hacer pruebas tanto de acceso al servidor MySQL como de creación de escenarios de replicación y *clusterización* de servidores.

INSTALACIÓN DE UBUNTU EN VIRTUALBOX

En primer lugar debemos obtener la distribución de Ubuntu correspondiente en la web oficial

http://www.ubuntu.com/desktop/get-ubuntu/download

Podemos elegir entre varias distribuciones según nuestros requerimientos, sin embargo para el propósito de este libro es suficiente con usar una *desktop edition* normal. Una vez descargado, el archivo iso correspondiente será el que usemos como fuente de instalación de Ubuntu en Virtualbox.

Por otro lado descargamos e instalamos el programa Virtualbox en Windows 7 aunque es lo mismo si tenemos Windows Vista o Windows XP.

La instalación es muy sencilla, solamente debemos prestar atención a las pantallas de aviso de Windows (que, al menos en Windows 7 son unas cuantas).

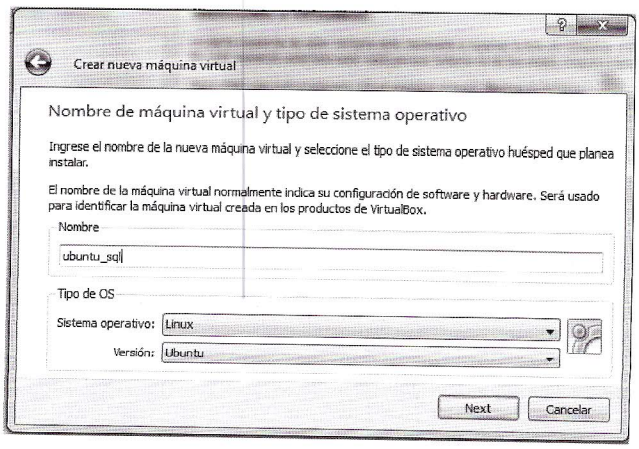
Pulsamos aceptar en todas ellas sin ningún miedo y finalmente tendremos instalado Virtualbox en el directorio de programas por defecto en nuestro equipo.

Creación de una máquina virtual

Ahora ya podemos empezar a crear máquinas virtuales. Una máquina virtual tiene los mismos componentes que una máquina real pero siendo todo falso (virtualizado), por lo que tendremos que asignarle memoria RAM, disco duro, etc.

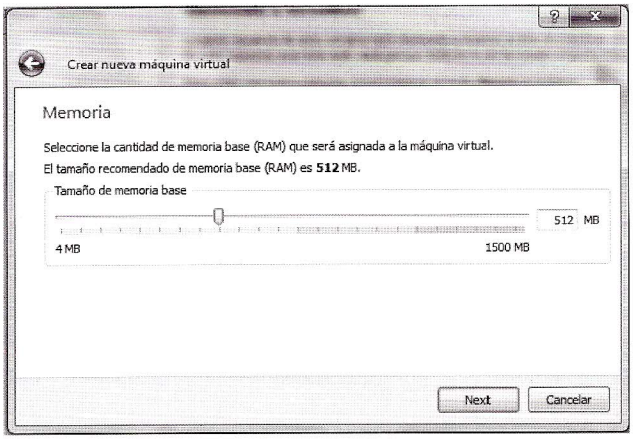
Aunque en esta sección crearemos solamente una máquina Ubuntu es interesante la posibilidad de clonar estas u otras para formar una red y hacer pruebas, podemos por ejemplo instalar varias máquinas Windows y *Linux* para formar una red heterogénea. Sin duda también será un buen ejercicio de aprendizaje de conceptos de redes.

Para iniciar la creación de una máquina iniciamos primero la aplicación *Virtualbox* y pulsaremos en nueva. Se inicia entonces un asistente que nos preguntará por el nombre del sistema operativo y el tipo:

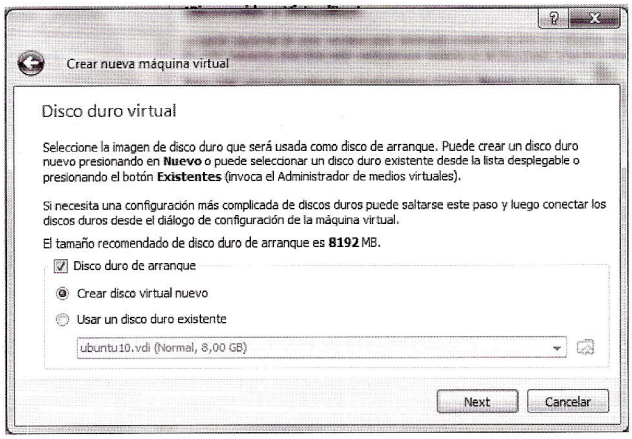


Ahora debemos escribir el nombre de la máquina virtual y seleccionar el sistema operativo y la versión. Como lo que vamos a hacer es instalar Ubuntu 10.04 le ponemos ese nombre a la máquina virtual, seleccionamos Linux como sistema operativo y Ubuntu como versión. Cuando terminemos, pulsamos en *Siguiente*.

A continuación, debemos asignarle la memoria RAM a la máquina virtual. Esta cantidad dependerá mucho de la que dispongamos. En general será suficiente con asignar 512 MB. De todos podremos variarla en cualquier momento siempre y cuando la máquina virtual esté apagada.



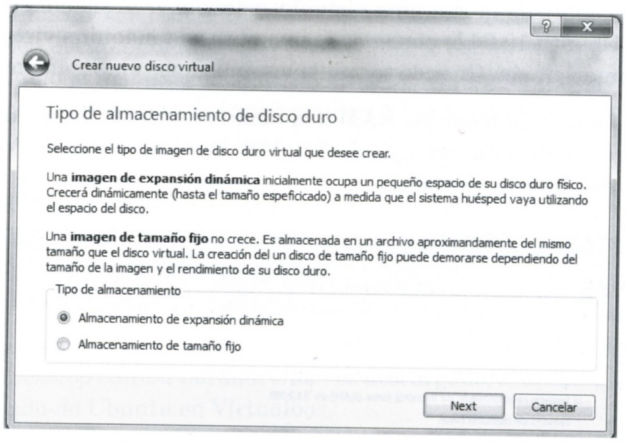
En este paso debemos Crear un disco duro virtual nuevo y pulsar en Siguiente. Comienza otro asistente. Éste nos guiará en la creación de un nuevo disco duro virtual. Pulsamos *Siguiente*:



Ahora debemos elegir entre dos tipos de almacenamiento:

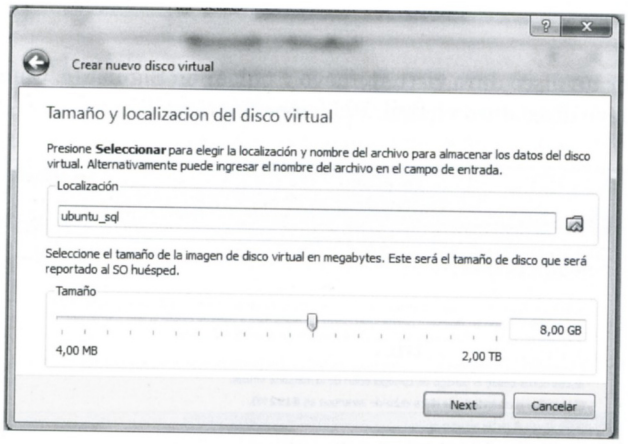
Almacenamiento de expansión dinámica: el disco duro virtual va creciendo en tamaño conforme lo necesitamos hasta el máximo que tengamos asignado. Es muy cómodo y útil si no nos sobra espacio en disco.

Almacenamiento de tamaño fijo: tiene un mejor rendimiento pero con la desventaja que se creará un archivo con el tamaño asignado.



Una vez elegido el tipo de almacenamiento, pulsamos en *Siguiente*.

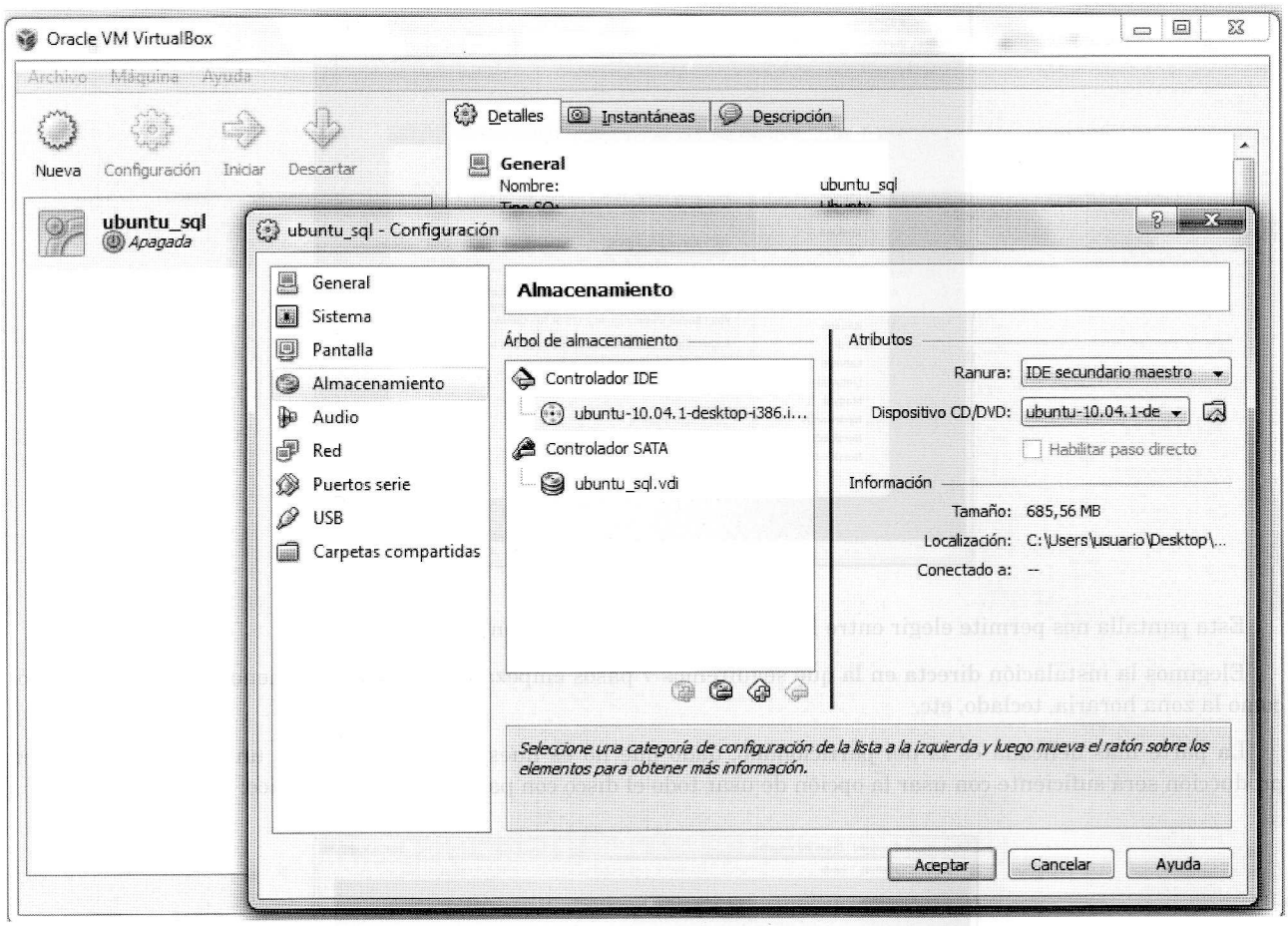
En este momento debemos asignarle la ubicación (se podría incluso poner en un disco duro externo o memoria USB) y el tamaño máximo que tendrá nuestro disco duro virtual. En principio con 8 GB será suficiente para empezar trabajar con Ubuntu pero podemos asignarle todo lo que queramos siempre con un mínimo, de 4 GB:



Finalizado el asistente para la creación del disco duro, hacemos pulsamos en *Terminar*.

Configurar la máquina virtual

Ya tenemos nuestra máquina virtual creada. Sin embargo, lo que tenemos es completamente equivalente a un equipo nuevo que no tiene sistema operativo, es decir, si lo intentamos arrancar, nos aparecerá un mensaje de error. Por eso lo he vamos a hacer a continuación consiste en asignarle un CD para que arranque desde el mismo. Con la máquina virtual seleccionada, hacemos clic sobre Configuración. En el menú de la izquierda pulsamos sobre *Almacenamiento* y después seleccionamos el icono del *CD-ROM* como controlador IDE. En la parte derecha veremos un atributo llamado CD/DVD, seleccionamos el dispositivo físico (si tuviésemos un CD-ROM con Ubuntu) o la imagen *iso* que hemos descargado de Internet.



Instalar Ubuntu 10.04 en la máquina virtual

Una vez que tenemos configurada la máquina virtual para arrancar desde el CD, ya podemos iniciarla.

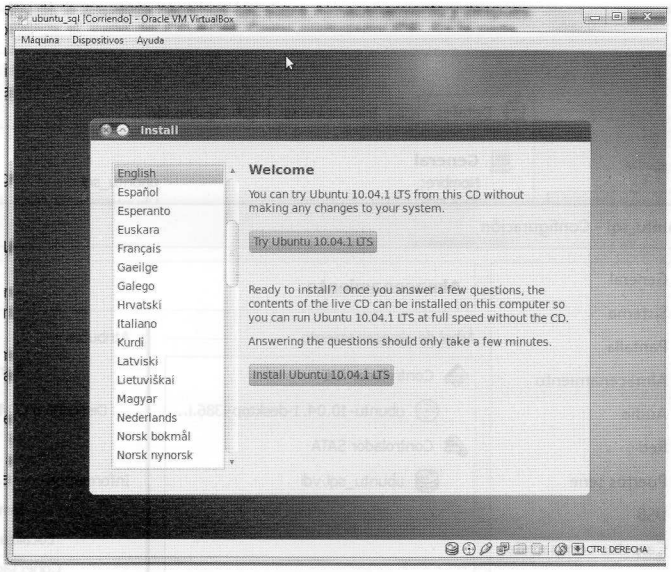
Con la máquina virtual seleccionada, hacemos clic en Iniciar.

Ahora ya comienza la instalación de Ubuntu que pasamos a detallar.

En primer lugar dependiendo del tipo de distribución podremos elegir entre un sistema *live* o una instalación directa. El primer caso es mejor ya que nos permite asegurarnos de que el sistema reconoce correctamente el hardware antes de iniciar la instalación real.

En todo caso ahora nos centraremos en una instalación directa.

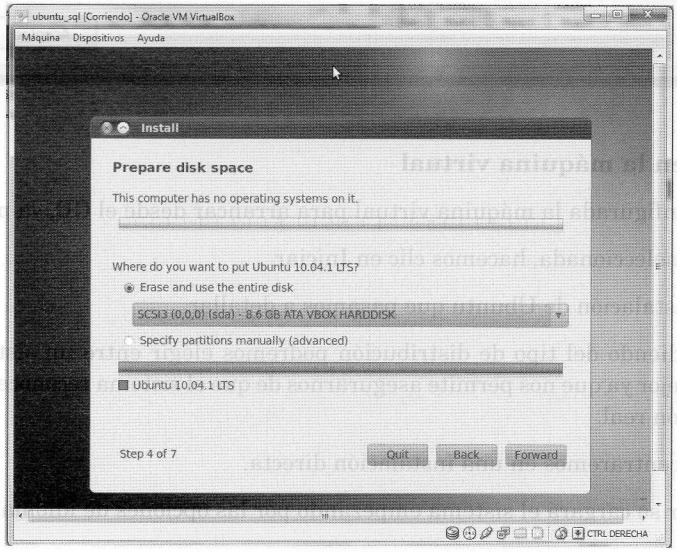
Cuando pulsemos iniciar se cargará el sistema empezando por las opciones de idioma:



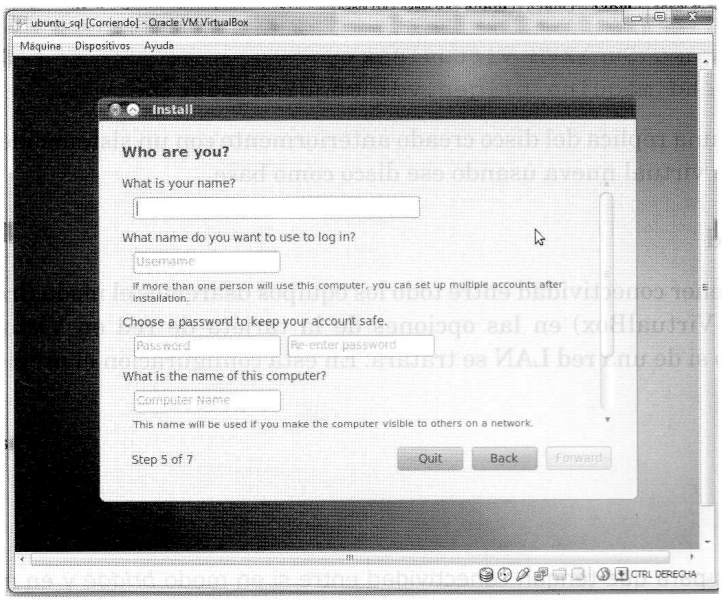
Esta pantalla nos permite elegir entre los dos tipos de arranque mencionados.

Elegimos la instalación directa en la que seguiremos 7 pasos empezando por aspectos menores y muy intuitivos como la zona horaria, teclado, etc.

La parte más delicada es la del particionado, dado que no es nuestra intención tener un sistema Ubuntu en producción será suficiente con usar la opción de usar todo el disco con particionado automático:



Después tenemos que completar el formulario con los datos de un usuario normal así como el nombre del equipo:



Y finalmente se nos informa sobre la configuración de la instalación y se procede a ejecutar pulsando *Install* (si usamos la opción *Advanced* podremos cambiar la ubicación del cargador de arranque para cuando tenemos varios sistemas en un mismo disco duro):

Una vez que se ha terminado la instalación debemos pulsar el botón *Reiniciar ahora* para empezar a disfrutar de nuestro nuevo Ubuntu 10.04.

INSTALACIÓN DE UN SISTEMA DEBIAN LIGERO

En primer lugar creamos una nueva máquina virtual en VirtualBox, con 512 MB de RAM.

Usaremos un disco duro de 1 GB, que particionaremos durante la instalación de Debian.

Usaremos un disco duro virtual de expansión dinámica. Una vez creada la máquina virtual, cargaremos el ISO del S.O. *Debian* en la unidad de CD/DVD virtual de la máquina que hemos creado.

Ahora ya estamos preparados para iniciar la instalación de Debian.

El proceso es muy similar a la instalación de Ubuntu y solo debemos tener en cuenta que cuando nos pregunte por el tipo de sistema a instalar marcar la opción mínima de forma que se nos instale un sistema sin entorno gráfico más ligero y manejable.

Después de comenzar la instalación y tras el corto proceso de carga, seleccionamos el idioma y teclado de manera similar a lo que hicimos en Ubuntu. Una vez seleccionado todo, comenzará la carga del programa de instalación,

Le asignamos un nombre a la máquina y configuramos reloj. Ahora llegamos a la parte más importante del proceso, el particionado del disco aunque en nuestro caso, dado que se usará solo para pruebas usaremos una única partición.

Clonación del sistema

Con el propósito mencionado de poder simular una red de varios equipos podemos hacer uso de la herramienta de clonación de Virtualbox *VBoxManage.exe* disponible en el directorio de instalación del programa.

Así, lo ejecutamos desde la consola de Windows en:

```
C:\Archivos de programa\Sun\XVM VirtualBox\
VBoxManage.exe clonehd maquina_origen.vdi maquina_destino.vdi
```

Ahora ya disponemos de una réplica del disco creado anteriormente con un sistema idéntico al anterior. Solo queda entonces crear una máquina virtual nueva usando ese disco como base.

Configuracion de la Red

Finalmente para poder tener conectividad entre todo los equipos usaremos el modo *Bridge* (hay otras posibilidades descritas en el manual de VirtualBox) en las opciones de la tarjeta de red en VirtualBox. Después deberemos configurar cada equipo como si de una red LAN se tratara. En esta configuración el propio equipo anfitrión es un *host* más dentro de la red.

ACTIVIDADES C.1

- Configura dos equipos para que tengan conectividad entre sí en modo *bridge* y en modo solo anfitrión.
- Instala la última versión de MySQL en tu equipo Ubuntu usando el programa *apt-get*.
- Accede al servidor de Ubuntu desde tu equipo anfitrión Windows.

APÉNDICE

D

Revisión de herramientas de MySQL en entornos Linux

Dado que en Windows es más complejo el uso de herramientas de terceros, como *scripts* o programas, se trata de que el alumno sea consciente de su existencia en otras plataformas además de su gran utilidad y posibilidad de modificación para ajustarlos a necesidades propias.

Como hemos visto MySQL incluye el directorio *bin* con programas necesarios para diferentes tareas, en concreto *mysqld* y *mysql* son los utilizados para establecer una conexión entre un cliente y un usuario.

Como el libro está orientado a usuarios Windows hemos obviado muchas utilidades solo compatibles con programas Linux. A continuación listaremos las más importantes:

Utilidades de servidor

- *mysql_safe*: permite arrancar el servidor de forma más segura haciendo que se reinicie si hay errores.
- *mysql.server:* es un *script* que permite iniciar MySQL en diferentes niveles de ejecución para sistemas Unix y Solaris.
- *mysqld_multi:* permite iniciar/detener multiples servidores especificados en *my.ini* por un número como por ejemplo *[mysqld23]*.
- *mysql_secure_installation:* permite realizar la instalación de manera segura con contraseñas para el administrador y eliminando cuentas públicas o anónimas.
- *mysql_upgrade:* se ejecuta tras una actualización del servidor y sirve para actualizar y/o reparar tablas incompatibles así como actualizar las tablas de permisos.

Utilidades de cliente

- *mysqlimport*: permite prácticamente hacer lo mismo que el comando *LOAD DATA INFILE* usando opciones como equivalentes a las cláusulas del citado comando.
- *mysqlslap*: es un simulador que emula cierta carga en un servidor como si muchos clientes accediesen simultáneamente al mismo.
- *Innochecksum*: para crear sumas de verificación en tablas *InnoDB offline*.
- *myisam_ftdump*: vuelca información sobre índices *full-text* en tablas MyISAM.
- *mmyisamchk*: sirve para comprobar, optimizar y reparar tablas MyISAM.
- *myisamlog*: procesa el contenido de ficheros *log* en tablas MyISAM.
- *myisampack*: comprime tablas MyISAM de solo lectura.
- *mysqldumpslow*: hace resúmenes de los ficheros de registro de consultas lentas.
- *mysqlhotcopy*: hace copias de seguridad de talbas MyISAM en caliente, mientras el servidor está activo.
- *mysql_find_rows:* busca mediante expresiones regulares sentencias SQL en ficheros.
- *mysql_setpermission*: establece permisos de forma interactiva.
- *mysql_zap*: elimina procesos que cumplen cierto patrón.
- *my_print_defaults*: imprime valores de opciones en los ficheros de configuración.
- *perror*: muestra el significado de los códigos de error.
- *replace*: hace sustituciones de texto en ficheros.
- *innnotop*: utilidad de monitorización de servidores, dispone de modos para monitorizar replicación, transacciones y consultas.

Además existen las herramientas gráficas que integran algunas de las ya nombradas, la más importante es el *workbench* que permite administrar, manipular y diseñar bases de datos de manera gráfica.

- *mysql migration toolkit:* permite hacer migraciones de sistemas de bases de datos a MySQL.

Utilidades comerciales

- MySQL Enterprise Monitor: monitoriza servidores alentando de posibles problemas antes de que surjan.
- SQLyog/MONyog: herramientas avanzadas para facilitar la administración, monitorización y optimización de servidores MySQL.

Utilidades de terceros

Por último queremos mencionar algunas herramientas de terceros que pueden resulta un complemente de gran valor para el administrador:

- *phpMyadmin*: escrito en PHP permite realizar todo tipo de tareas admnistrativas desde una interfaz gráfica vía web.

  http://www.phpmyadmin.net/home_page/index.php

- *Maatkit*: conjunto de herramientas de línea de comandos para acceso a bases de datos *open source* como MySQL o *Postgresql*.

  http://code.google.com/p/maatkit/

- *Mmm*: conjunto de *scripts* para la gestión de la replicación en configuraciones *master-master* (en todo momento uno de los nodos es de escritura).

  http://mysql-mmm.org/

- *Openark kit*: conjunto de utilidades para la administración, auditoría y diagnóstico de bases de datos MySQL.

  http://code.google.com/p/openarkkit/

- *Mycheckpoint*: utilidad para la monitorización de bases de datos MySQL.

  http://code.google.com/p/mycheckpoint/

- *Mysqlsandbox*: programa para instalar múltiples servidores de manera aislada del sistema en que se alojan e independientemente de los servidores activos. Permite probar servidores y su funcionamiento de una manera segura.

  http://mysqlsandbox.net/

- *cacti*: es un programa para graficar datos en general de toda clase de datos de procesos de red. Para el caso de MySQL incluye plantillas que pueden obtenerse de:

  http://code.google.com/p/mysql-cacti-templates/
  http://www.cacti.net/

- *mysqlreport*

  http://hackmysql.com/mysqlreport

- *mtop*: muestra los hilos (*threads*) iniciados por el servidor MySQL para cada una de sus conexiones.

ACTIVIDADES D.1

- Instala el programa *phpMyAdmin* en tu máquina Windows y comprueba su funcionamiento vía web.
- Instala el programa *Maatkit* en tu equipo Ubuntu. Describe la utilidad de cada uno de los *scripts* que incorpora.
- Usando el programa *innodbtop* vuelca el estado de tus tablas *innodb* a un fichero.
- Haz un *script* en *perl* para que cada hora haga la operación anterior enviando un correo con el contenido del fichero.
- Usa *mysqltest* para generar consultas a tu servidor y comenta las diferencias entre los programas *mtop* e *inndbtop.*

APÉNDICE

E

Introducción a la administración de sistemas gestores: Oracle

ARQUITECTURA DEL SERVIDOR

Oracle es un software de bases de datos basado en la arquitectura tipo *grid* que se basa en integrar componentes de software y hardware para conseguir satisfacer los distintos requerimientos. Este tipo de arquitectura resulta ser mucho más flexible y adaptable a distintos entornos aunque también tiene ciertos inconvenientes.

Dispone de dos clases de arquitecturas típicas, cliente/servidor y el tipo *multitier*. Éste último es el que está siendo más utilizado.

- Arquitectura *Client/Server*.

  En este esquema una base de datos se divide en dos partes: un *front-end* o cliente y un *back-end* o servidor.

  El cliente es normalmente otra máquina que ejecuta un software específico que permite consultar toda clase de información al servidor. Sus requisitos son mínimos y puede haber muchos clientes para un mismo servidor.

  El servidor ejecuta el software *Oracle* y gestiona las peticiones recibidas de las aplicaciones o equipos clientes

- Arquitectura *Multitier*: servidores de aplicación.

  Esta arquitectura dispone típicamente de los siguientes componentes:

  - Un cliente que inicia una operación.
  - Uno o más servidores de aplicación que ejecutan partes de dicha operación liberando, de este modo al servidor de parte de su carga. Además pueden agregar un nivel intermedio de seguridad al situarse entre los clientes y servidores de múltiples bases de datos.
  - Un servidor de bases de datos que almacena la mayor parte de los datos.

De este modo se añade una capa más de *software* que refuerza la seguridad y descarga al servidor de gran parte de su trabajo.

ESTRUCTURAS DE LAS BASES DE DATOS FÍSICAS

Oracle implementa diversos tipos de estructuras para la gestión de sus datos. Éstas incluyen ficheros de datos, ficheros de control, ficheros de registro, ficheros de variables y ficheros de copia de seguridad.

Datafiles

Cada base de datos dispone de uno o más ficheros de datos o *datafiles* que contienen todos los datos de la base de datos. Los datos referentes a las estructuras lógicas como tablas e índices se guardan también en estos ficheros.

Estos ficheros agrupados forman una unidad lógica de almacenamiento para la base de datos llamada *tablespace* de forma que un *datafile* e puede ser asignado a un solo *tablespace*. Además tienen la capacidad de ampliarse automáticamente cuando están llenos.

Control Files

Cada base de datos de Oracle contiene un fichero de control que con información sobre la estructura física de la base incluyendo el nombre, nombres y direcciones de los ficheros de datos y de registro así como fecha/hora de la creación de la base.

Son ficheros que permiten iniciar a *Oracle* las distintas bases de datos así como facilitar las copias de seguridad.

Ficheros de registro offline/online

Toda base de datos Oracle tiene un conjunto de dos o mas ficheros de registro *online* (llamados también *online redo log files*). Estos archivos junto con los ficheros de registro originales forman el llamado redo *log* de la base de datos. Estos ficheros son equivalentes a los ficheros de registro binario de MySQL ya que almacenan información sobre cambios producidos en la base de datos.

Cada cierto tiempo se almacenan en los llamados ficheros de registro *offline*.

Dada la importancia de estos archivos Oracle permite crear varias copias del mismo en distintos discos o equipos.

Parameter Files

Son ficheros que contienen información sobre valores de parámetros de una base de datos. Hay dos tipos: los llamados *pfiles* o ficheros de parámetros y los llamfos *spfiles* o ficheros de parámetros de servidor. Permiten almacenar y gestionar los parámetros de cada base de datos.

Alert and Trace Log Files

Son ficheros donde se registran los errores de servidor o de alguno de sus procesos. Es un archivo clave para la optimización tanto del servidor como de cada base de datos.

Estos archivos pueden ser grandes y complejos así que existen varias utilidades para su gestión cómoda de los mismos como son:

Automatic Diagnostic Repository (ADR)

Sirven para visualizar información y resúmenes de datos.

Incident Packaging Service (IPS)

Extraen información de los ficheros de traza para enviarlos a Oracle.

Backup Files

Son ficheros de copia de otros ficheros que pueden gestionarse de forma manual o a través del servidor de manera programada.

REVISIÓN DE LAS ESTRUCTURAS DE BASE DE DATOS

Oracle divide sus estructuras de datos en: bloques de datos que representan la unidad mínima de almacenamiento, extensiones que agrupan varios bloques, segmentos que agrupan a varias extensiones y se van creando dinámicamente a mediada que se van llenando. Finalmente están los *tablespaces* que agrupan a varios segmentos. Cada base de datos está lógicamente dividida en dos o más *tablespaces*, las llamadas *SYSTEM* y *SYSAUX*. Además los *tablespaces* pueden ser *offline* o fuera de línea para tareas administrativas u *online* que es su estado normal. También los hay de solo lectura para facilitar tareas de *backup*.

Objetos y esquemas

En el contexto de Oracle un esquema representa una colección de objetos de bases de datos, típicamente tablas, vistas e índices. También pueden incluir *clusters*, grupos de una o más tablas almacenadas físicamente próximas de forma que se facilita su acceso cuando se usan con frecuencia y sinónimos o alias para tablas, vistas, vistas

materializadas, secuencias, operador, procedimiento, función, paquete, clase de Java, objetos definidos por el usuario o incluso otro sinónimo.

Diccionario de datos

Cada base de datos tiene asociado un diccionario de datos o un conjunto de tablas y vistas con información sobre la base, como las estructuras físicas y lógicas o los usuarios con permisos sobre la base. Este diccionario se crea a la vez que la base de datos y su información se actualiza de forma automática por parte del servidor.

Instancia de bases de datos

Un servidor Oracle de bases de datos consiste en una base de datos de Oracle y uno o más instancias de bases de datos. Cada vez que se inicia una base de datos se habilita una zona de memoria compartida llamada *System Global Area* (SGA) y se inician los procesos correspondientes. Estos procesos y la zona de memoria es lo que se llama una instancia de Oracle.

Cuando los usuarios se conectan a un servidor Oracle lo hacen a una instancia

Procesos en segundo plano

Oracle usa tres tipos de procesos, procesos en segundo plano, dedicados a operaciones asíncronas de E/S y otras tareas, procesos de servidor para gestionar peticiones de clientes y los procesos de usuario. Estos últimos suelen estar en diferentes equipos. Son también llamados procesos cliente y se encargan de permitir el acceso al servidor.

Estructuras de memoria

En *Oracle* se usan estructuras de memoria de distinto tipo con el propósito de compartir datos o permitir a cada usuario tener una zona privada. Existen esencialmente dos zonas de memoria asociadas con una base de datos: la *System Global Area* (SGA) que contiene datos e información de control para cada instancia de la base de datos. Y la *Program Global Areas* (PGA), que son zonas de memoria creadas por el servidor cuando se inicia un proceso de *Oracle*, ya sea de *background* o de servidor.

Acceso a la base de datos

La conexión a la base de datos se usa el componente *Oracle net services* que incluye los protocolos TCP/IP, HTTP, FTP y WebDAV. En particular *Oracle Net* se encarga de crear y mantener las conexiones.

Cada conexión requiere iniciar una instancia, montar una base de datos y abrirla, procesos que se pueden realizar usando el componente *Oracle Enterprise Manager*, la herramienta de línea de comandos *SQL*Plus*, el comando *srvctl* de *Express Edition.* Cuando la instancia se activa el servidor lee los ficheros de parámetros *spfile* o *pfile* para establecer los parámetros de inicio, después ubica un SGA y crea los procesos de segundo plano entre ellos los de escritura con el *database writer* o DBW y los de registro con el LGWR.

UTILIDADES DE ORACLE

El gestor *Oracle* dispone de un conjunto de utilidades para la gestión y administración de sus bases de datos.

Oracle Real Application Testing

Es un sistema de pruebas tanto a nivel de hardware como de software para permitir testear nuestras bases de datos ante posibles cambios. Se compone de dos partes, el reproductor de bases de datos para probar cambios de configuración, actualizaciones de bases de datos, almacenamiento y actualizaciones de sistema operativo y *hardware*.

Por otro lado el analizador de rendimiento SQL (SQL *Performance Analyzer*) testea cambios relacionados con la ejecución de comandos SQL facilitando su optimización.

Concurrency Features

En sistemas de bases de datos multiusuario se deben considerar especialmente la concurrencia a los datos, su consistencia y su funcionamiento óptimo con máximo desempeño.

Oracle dispone de varias herramientas para tal fin.

- La concurrencia, es decir, el acceso simultáneo de varios usuarios a la misma información puede generar problemas de consistencia de datos y de rendimiento. Debe controlarse para minimizar la interferencia entre distintas operaciones de usuarios sobre los mismos datos. Para ello *Oracle* hace uso de diferentes tipos de bloqueos y el uso el concepto de transacción consistente en definir puntos de inicio y fin de operaciones de modificación de datos controlando cuando los datos pueden o no ser alterados o consultados.
- La consistencia de lecturas tiene que ver con garantizar que los datos involucrados en una operación y en un tiempo dado son consistentes y no están cambiando y con el tiempo de espera para el acceso a datos manipulados por otros usuarios.
- Mecanismos de caché.

  Oracle optimiza su funcionamiento causando sistemas de caché con datos de usuario en memoria, datos de registro, datos del diccionario, de resultados de consultas frecuentes, entre otros.
- Mecanismos de bloqueo.

  Los llamados *locks*, controlan el acceso concurrente a los datos, así cuando se están modificando datos estos mecanismos los bloquean hasta que la operación es confirmada (*comitted*) para asegurar la integridad de los mismos.

UTILIDADES DE GESTIÓN

Autogestión de la base de datos

Incluye gestión automática de operaciones deshacer, gestión automática de memoria y ficheros, gestión de espacio libre y de recuperación de datos (RMAN).

Tareas de mantenimiento automáticas

Permiten automatizar tareas de mantenimiento programándolas en distintos periodos de tiempo. Esta característica se puede usar con las llamadas *maintenance* Windows. Que permiten establecer los puntos de comienzo y fin así como los recursos de CPU y de E/S permitidos.

Oracle Enterprise Manager

Es una herramienta que facilita la gestión centralizada del entorno de las bases de datos para un conjunto de productos de Oracle. Incluye una consola gráfica, servidores de administración, agentes inteligentes de Oracle, servicios comunes y varias utilidades administrativas.

SQL Developer y SQL*Plus

Es una herramienta gráfica que permite navegar, editar y eliminar objetos de las bases de datos así como ejecutar código SQL y PL/SQL, exportar/importar datos y hacer informes.

Gestión automática de memoria

Desde la versión 11g está característica controla automáticamente las llamadas *System Global Area* (SGA) o zonas compartidas de memoria y *Program Global Area* (PGA) o zonas de memoria de los procesos.

Gestión automática de almacenamiento

Gracias a esta característica los ficheros de cada base de datos se distribuyen automáticamente sobre todos los discos disponibles optimizando en cada momento el espacio disponible y proporcionado redundancia.

Diagnóstico automático

El *Automatic Database Diagnostic Monitor* (ADDM) permite realizar análisis de rendimiento en periodos de tiempo determinados por dos instantáneas de la base de datos llamadas *Automatic Workload Repository* (AWR) *snapshots*. Con el objeto de informar y proponer soluciones a problemas de rendimiento generados en dicho período.

SQL Tuning Advisor

Es una herramienta que optimiza comandos SQL ofreciendo recomendaciones sobre la mejor forma de ejecución incluyendo el ahorro en tiempo y recursos que se obtiene.

SQL Access Advisor

Realiza recomendaciones de modificación del esquema como por ejemplo la creación/modificación de índices o vistas para optimizar comandos SQL. Se basa en usar fuentes de SQL como comandos recientes o comandos indicados por el usuario.

El programador de tareas

Oracle dispone del paquete *DBMS_SCHEDULER* que agrupa una serie de procedimientos y tareas que pueden ser llamados desde cualquier programa PL/SQL. Este paquete permite establecer cuándo y qué tareas serán ejecutadas.

Gestor de recursos

Habitualmente son los sistemas operativos los que se encargan de controlar y administrar los recursos. En *Oracle* existe una posibilidad de hacerlo a través del *Database Resource Manager* que controla el reparto de recursos entre varias sesiones incluyendo qué sesiones y recursos pueden activarse y por cuenta.

Diagnosability Features

Es una utilidad para prevenir, detectar, diagnosticar y corregir errores críticos causados por errores de código en las bases de datos, corrupción de metadatos y de datos del usuario.

Copias de seguridad y restauración Oracle

Para ello dispone de *Recovery Manager* (RMAN). Es una utilidad que permite hacer copias de seguridad y restauración de las mismas además de incluir un repositorio de históricos. Es accesible por línea de comandos y

desde el *Enterprise Manager*. Permite entre otras posibilidades hacer copias incrementales, recuperación de bloques, compresión y copias encriptadas.

User-managed backup and recovery: permite realizar operaciones de copia y restauración con ayuda de comandos propios del sistema operativo y SQL*Plus.

Podemos hacer también copias de seguridad lógicas de esquemas con las utilidades *Data Pump Export* y *Data Pump Import*.

Alta disponibilidad

La alta disponibilidad se logra con las utilidades:

- *Oracle Real Application Clusters* (Oracle RAC): permite ejecutar cualquier aplicación en varios servidores *clusterizados* lo que proporciona la mayor nivel de disponibilidad y escalabilidad. Además se puede hacer en caliente sin perturbar el funcionamiento del sistema ni la integridad de los datos.
- *Oracle Data Guard*: conjunto de servicios para crear, mantener y monitorizar bases de datos y prevenir fallos de toda clase. Si una base cae está utilidad es capaz de sustituirla por otra copia consistente con la caída.
- *Oracle Streams* permite la gestión de toda clase de información intercambiada en una base de datos o entre bases de datos.
- *Oracle Flashback Technology*: es un conjunto de caraterísticas de Oracle para ver el estado de los datos en distintos momentos en el pasado. También se usa en recuperaciones de datos.
- Gestión de almacenamiento automático (*Automatic Storage Management* -ASM): permite distribuir los ficheros de datos de manera transparente en todo el sistema de almacenamiento. Incorpora el concepto de SAME (*strip and mirror everything,* divide y replica todo).
- *Recovery Manager*: gestión de copias de seguridad y restauración. El proceso RMAN determina de forma dinámica la forma más eficiente de realizar las operaciones relacionadas con copias de seguridad determinado automáticamente los cambios producidos en las estructuras de las bases de datos.

Business Intelligence

Una característica especial de *Oracle* que la hace especialmente útil en grandes aplicaciones es la de permitir obtener información útil a partir de los datos ara lo cual incluye las siguientes características:

- *Data Warehousing*: un *data warehouse* es una base de datos relacional diseñada para el análisis más que parea transacciones. Normalmente contiene datos históricos derivados de transacciones y de otras fuentes diversas. Es capaz de integrar os mismos y proponer conclusiones útiles para el negocio a partir de los mismos. Para ello incluye una herramienta para la extracción, transformación y carga de datos o ETL y un motor de análisis online (OLAP) entre otras.
- *Materialized Views*: para mejorar el rendimiento *Oracle* incorpora vistas materializadas con copias de datos en lugar de solo la definición de las mismas, de este modo ue redirige las consultas SQL a dichas vistas en lugar de hacerlas sobre las tablas gracias al mecanismo llamado *query rewrite*.
- Compresión de tablas: Oracle es capaz de hacer compresión de manera transparente en todo tipo de operaciones, especialmente las relacionadas con copias de seguridad, estructuras de datos y datos y tráfico de red.
- Pararelismo: *Oracle* es capaz de ejecutar comandos SQL en paralelo simultáneamente en múltiples procesadores reduciendo enormemente el tiempo de proceso sobre todo en grandes bases de datos.

- *Analytic* SQL: Oracle tiene gran cantidad de posibilidades a la hora de realizar operaciones analíticas en la base de datos incluyendo medias, sumas acumuladas, clasificaciones, informes diversos y comparaciones de datos en distintos periodos. Destaca la herramienta OLAP (*Oracle online analytical processing*) para el análisis de grandes cantidades de datos en distintos escenarios.
- Minería de datos: *Oracle* proporciona infraestructura para proporcionar la posibilidad de *data mining* en aplicaciones de bases de datos como *call centers*, ATM's o *e-business*. Para ello dispone de soporte para PL/SQL, *Java* API y diversas funciones de SQL. También hay una herramienta gráfica llamada *Oracle Data Miner*.
- *Very Large Databases* (VLDB): para grandes bases de datos existe l aposibilidad de particionar permitiendo bases escalables y flexibles además de mejorar la disponibilidad con un coste mínimo en recursos y desempeño.

Aspectos de seguridad

Mediante esta característica de *Oracle* podemos controlar cómo se accede y se usa a la base de datos. De este modo se consigue prevenir el acceso no autorizado a bases de datos y objetos del servidor además de poder auditar los servidores.

La seguridad en las bases de datos puede clasificarse en dos tipos, seguridad de sistema que controla quién puede acceder y que puede hacer al servidor y seguridad de datos para las acciones permitidas en las distintas bases de datos y sus objetos.

Integridad de datos y triggers

Para aplicar las reglas de negocio, que establecen restricciones y condiciones sobre los datos (por ejemplo que el saldo de una cuenta nuca sea negativo o no supere cierto valor) se usan restricciones de integridad y *triggers* o disparadores (escritos en PL/SQL, Java o C). Las primeras son definiciones sobre las condiciones que debe cumplir una columna en una tabla de forma que si no se cumplen los datos no podrán ser modificados. Los ejemplos son conocidos como la cláusula *not null*, y las de creación de claves.

Integración de la información

En un entorno distribuido Oracle permite las aplicaciones y componentes compartir e intercambiar información mediante tres medios:

SQL distribuido: en un entorno de servidores Oracle homogéneos es la capacidad de usar comandos SQL contra varios servidores que alojan una misma base de datos todo ello de forma transparente al usuario, tanto en lo que respecta a la ubicación de los datos (transparencia de ubicación) como en lo que respecta a la realización de transacciones (transparencia de transacción) y optimización de consultas distribuidas minimizando el tráfico de red.

- *Oracle Streams*: es una característica que permite la propagación y gestión de datos, transacciones y eventos en los intercambios de datos en una base de datos o entre varias bases. Es el sistema que provee replicación de bases de datos.
- *Oracle Database Gateways and Generic Connectivity*: con esta característica se permite extender el concepto de base de datos distribuida a otras bases de datos e incluso a otros procesos y fuentes de datos.

Desarrollo de aplicaciones

Oracle dispone de API de varios lenguajes de programación así como de la posibilidad de usar SQL para instrucciones de definición de datos, manipulación y control de datos, transacciones y sesiones, y también de PL/SQL como lenguaje procedural propio base para la creación de rutinas almacenadas, paquetes (agrupaciones de rutinas) y disparadores.

Así mismo dispone de conectores o controladores para Java, cobol, C/C++ PHP y Visual Basic, entre otros.

INSTALACIÓN Y CONFIGURACIÓN BÁSICA

Oracle dispone de una versión gratuita dirigida a estudiantes, pequeñas empresas y desarrolladores que quieran embeberla junto con sus aplicaciones. Esta versión limitada de *Oracle* 10g solo podrá correr en servidores con 1 solo procesador y con hasta 1 Gb de RAM, y podrá manejar un tamaño máximo de 4 Gb de almacenamiento en el disco.

A continuación os mostramos, paso a paso, cómo instalar Oracle 10g Express Edition en un PC con Windows XP:

1 Descargaremos el fichero *OracleXE.exe* de la web de Oracle:

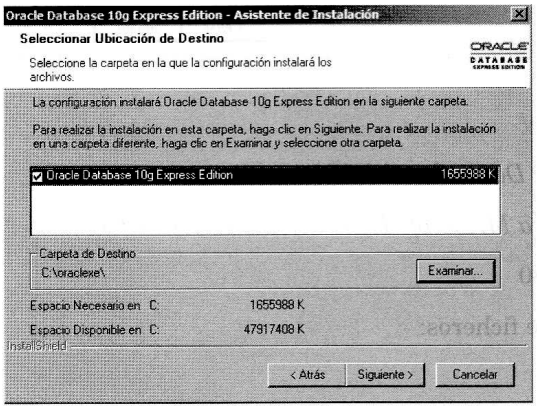
http://www.Oracle.com/technology/software/products/database/xe/index.html

(Necesitaremos ser usuarios registrados de Oracle, el registro es gratuito).

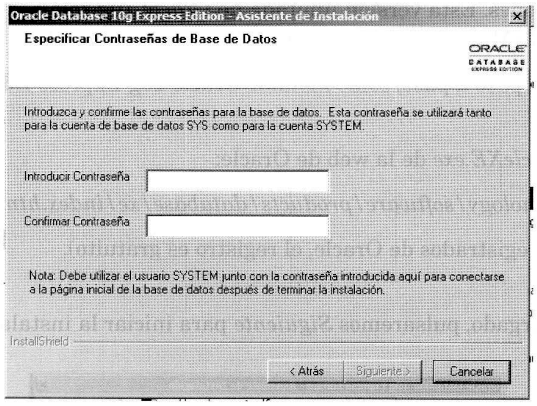
2 Ejecutaremos el fichero descargado, pulsaremos *Siguiente* para iniciar la instalación:

Aceptaremos el contrato de licencia y pulsaremos Siguiente

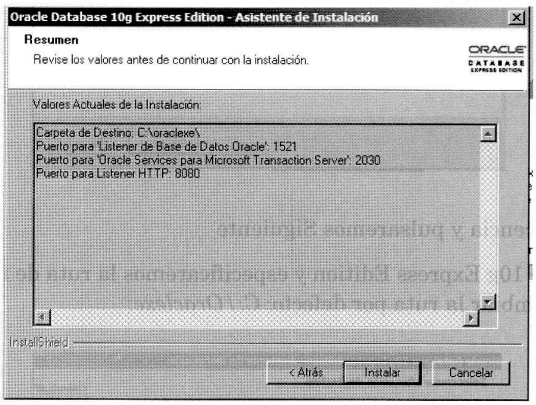
3 Marcaremos Oracle Database 10g Express Edition y especificaremos la ruta de instalación de *Oracle*, pulsando el botón *Examinar* podremos cambiar la ruta por defecto: *C:/Oraclexe*:



4 Introduciremos la contraseña para el usuario SYS y para el usuario *SYSTEM* y pulsaremos *Siguiente*:



5 A continuación aparecerá una ventana con las opciones de instalación elegidas, pulsaremos *Instalar* para iniciar el proceso:



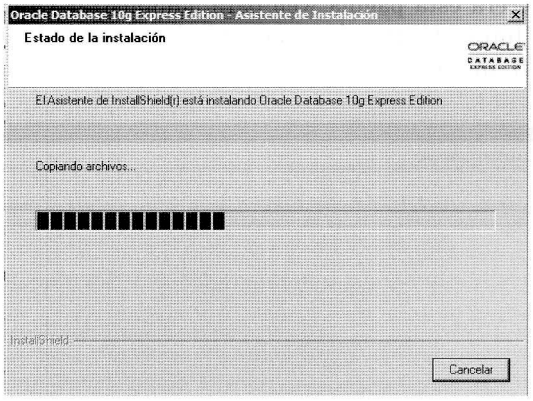
Carpeta de Destino: *C:/Oraclexe/*

Puerto para *'Listener de Base de Datos Oracle'*: 1521

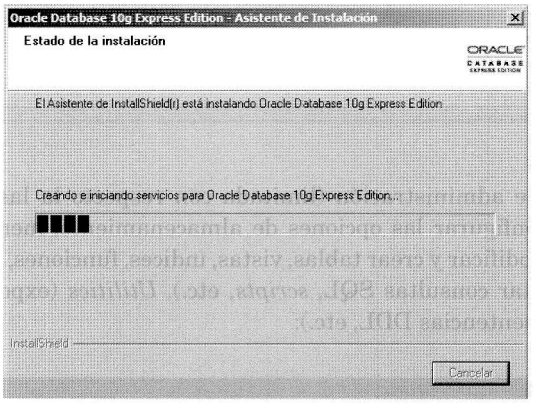
Puerto para *'Oracle Services para Microsoft Transaction Server'*: 2030

Puerto para *Listener HTTP*: 8080

Se iniciará el proceso de copia de ficheros:



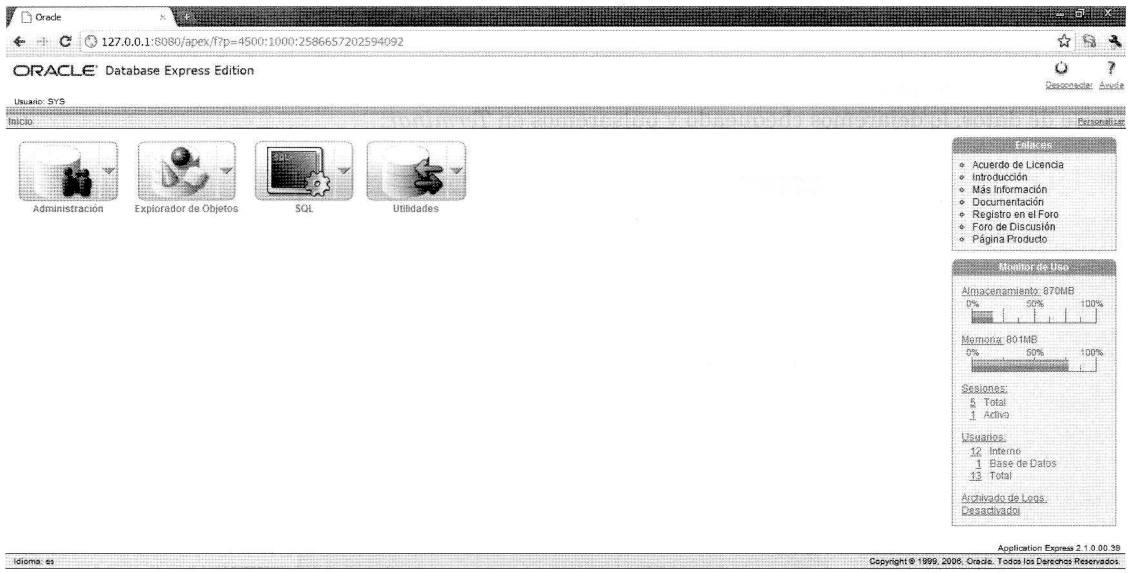
Y el proceso de configuración automática de la base de datos. Por defecto, el instalador de Oracle 10g Express Edition, crea y configura una base de datos:



6 Tras la finalización del proceso de creación de la base de datos, el asistente permite iniciar la página de configuración de la base de datos, lo dejaremos chequeado y pulsaremos en *Terminar*:

Tras unos segundos nos aparecerá esta página web para administrar Oracle 10g Express Edition, accesible roduciendo en el explorador de Internet: *http://127.0.0.1:8080/ape*. En *Username* introduciremos el nombre del uario (*system* ó *sys*) y en *Password* introduciremos la contraseña especificada en el paso 4:

Nos aparecerá una ventana de administración (limitada con respecto la las versiones completas) con varias ciones: *Administration* (para configurar las opciones de almacenamiento, memoria, usuarios y monitorización), *ject Browser* (para visualizar, modificar y crear tablas, vistas, índices, funciones, *triggers*, procedimientos, paquetes, uencias, etc), *SQL* (para ejecutar consultas SQL, *scripts*, etc.), *Utilities* (exportación, importación, papelera de iclaje, informes, generación de sentencias DDL, etc.):



El programa de instalación de Oracle 10g Express Edition habrá creado los siguientes servicios:

Tabla E.1 Servicios principales incluidos en Oracle

Nombre	Ubicación
*Oracle***JobSchedulerXE**	C:/Oraclexe/app/Oracle/product/10.2.0/server/Bin/extjob.exe XE
*Oracle***MTSRecoveryService**	C:/Oraclexe/app/Oracle/product/10.2.0/server/BIN/omtsreco.exe "OracleMTSRecoveryService"
*Oracle***ServiceXE**	C:/Oraclexe/app/Oracle/product/10.2.0/server/bin/ORACLE.EXE XE
*Oracle***XEClrAgent**	C:/Oraclexe/app/Oracle/product/10.2.0/server/bin/OraClrAgnt.exe
*Oracle***XETNSListener**	C:/Oraclexe/app/Oracle/product/10.2.0/server/BIN/tnslsnr.exe

CUENTAS DE USUARIO Y PERMISOS

En Oracle los usuarios y roles tienen privilegios sobre los objetos del sistema gestor además de ciertos recursos disponibles.

A continuación veremos los comandos esenciales para la gestión de dichas cuentas, privilegios y recursos.

GESTION DE CUENTAS

Sintaxis para crear una cuenta

```
CREATE USER user
   IDENTIFIED BY password
   [ DEFAULT TABLESPACE tablespace
   | TEMPORARY TABLESPACE tablespace
   | { QUOTA { size_clause | UNLIMITED } ON tablespace }...
   | PROFILE profile
   | PASSWORD EXPIRE
```

- *DEFAULT TABLESPACE*: zona del usuario por defecto donde se almacenarán sus datos.
- *TEMPORARY TABLESPACE*: para el almacenaje de segmentos de datos temporales.
- *QUOTA*: espacio asignado para el usuario en cierto *tablespace*.
- *PROFILE*: el perfil asignado a ese usuario.
- *PASSWORD EXPIRE*: obliga al usuario a modificar su contraseña antes de entrar por primera vez.

Modificar cuentas

Los usuarios, además de administradores, pueden modificar sus cuentas con el comando *ALTER USER* y sus contraseñas con *PASSWORD*.

```
ALTER USER
{ user
    { IDENTIFIED
      { BY password [ REPLACE old_password ]}
    | DEFAULT TABLESPACE tablespace
    | TEMPORARY TABLESPACE tablespace
    | { QUOTA { size_clause|UNLIMITED} ON tablespace
      }
    | PROFILE profile
    | DEFAULT ROLE { role [, role ]...
                   | ALL [ EXCEPT role [, role ] ... ]
                   | NONE
                   }
    | PASSWORD EXPIRE;
```

Donde se ha agregado la cláusula especial *default role* para asignar roles al usuario, aunque solamente roles que se hayan asignado previamente con e comando *grant*, como luego veremos.

Para Modificar la contraseña de usuario usamos *PASSWORD* usuario o:

```
ALTER USER user IDENTIFIED BY password
```

Eliminar cuentas

Para borrar una cuenta usaremos el comando:

```
DROP usuario [cascade]
```

CASCADE: esta opción hace que se eliminen todos los objetos del usuario antes de su eliminación.

GESTIÓN DE PERFILES

En Oracle aparece un objeto nuevo que es el *profile* o perfil y que permite agrupar una serie de recursos bajo un mismo nombre de forma que dichos recursos se puedan asignar a usuarios o grupos de usuarios en función de las necesidades de la aplicación.

Crear un perfil

```
CREATE PROFILE profile
   LIMIT { resource_parameters
         | password_parameters
         };
resource_parameters::=
{ { SESSIONS_PER_USER
  | CPU_PER_SESSION
  | CPU_PER_CALL
  | CONNECT_TIME
  | IDLE_TIME
  | LOGICAL_READS_PER_SESSION
}
password_parameters::=
{ { FAILED_LOGIN_ATTEMPTS
```

```
    | PASSWORD_LIFE_TIME
    | PASSWORD_REUSE_TIME
    | PASSWORD_REUSE_MAX
    | PASSWORD_LOCK_TIME
    | PASSWORD_GRACE_TIME
    }
    { expr | UNLIMITED | DEFAULT }
}
```

Recursos

- *Profile*: nombre del perfil.
- *DEFAULT*: implica que el perfil hereda el limite asignado al recurso en el perfil definido por defecto en el sistema.

Tipos de recursos

- *SESSIONS_PER_USER*: número de sesiones concurrentes del mismo usuario.
- *CPU_PER_SESSION*: centésimas de Segundo de CPU por sesión.
- *CPU_PER_CALL*: centésimas de Segundo de CPU por llamada.
- *CONNECT_TIME*: tiempo de sesión en minutos.
- *IDLE_TIME*: tiempo de inactividad en minutos.
- *LOGICAL_READS_PER_SESSION*: número de lecturas de bloques de datos permitidos en una sesión.

Parámetros de contraseñas

- *FAILED_LOGIN_ATTEMPTS*: número máximo de intentos de acceso fallidos antes de bloquear la cuenta.
- *PASSWORD_LIFE_TIME*: número de días en que puede usarse le mismo *password*, si además se incluye un *PASSWORD_GRACE_TIME* (por defecto es 180 días), entonces el *password* expira en ese tiempo si no se ha modificado.
- *PASSWORD_REUSE_TIME*: días en que un *password* no puede reutilizarse.
- *PASSWORD_REUSE_MAX*: número de cambios necesarios para poder reutilizar un *password*.
- *PASSWORD_LOCK_TIME*: número de días que una cuenta quedará bloqueada despues una serie de fallos de acceso (por defecto 1).
- *PASSWORD_GRACE_TIME*: días tras los cuales comienza el período de gracia en que el usuario es advertido del inminente bloqueo de su cuenta aunque en esos días se le sigue permitiendo el acceso.

Borrar un perfil

```
DROP PROFILE profile [ CASCADE ] ;
```

CASCADE: esta opción elimina el perfile de todos los usuarios a los que estaba asignado.

Mostrar usuarios y perfiles

Oracle define una serie de vistas para permitir acceder a información acerca de los objetos de la base de datos. Para el caso de usuarios y perfiles debemos consultar la vista *DBA_USERS*, para cuotas *DBA_TS_QUOTAS*, para información sobre perfiles *DBA_PROFILE* y para información sobre sesiones *V$SESSION*.

SISTEMA DE PRIVILEGIOS Y ROLES

En Oracle aparece también el sistema de roles mediante el cual se permite solo a ciertos usuarios gestionar datos así como imponer restricciones de acceso o acciones concretas sobre objetos como bases de datos, tablas, filas e incluso recursos como tiempo de CPU.

Los roles agrupan privilegios u otros roles bajo un mismo nombre para facilitar su asignación a muchos usuarios. Oracle distingue dos tipos de privilegios en este sentido:

- Privilegios de sistema para permitir realizar acciones administrativas de carácter general en la base de datos.

  Están registrados en la tabla *DBA_SYS_PRIVS* del diccionario de datos.

- Privilegios asociados con objetos del servidor.

Gestión de privilegios

Podemos dar y quitar privilegios a usuarios y roles con *GRANT* y *REVOKE*.

```
GRANT { grant_system_privileges
      | grant_object_privileges
      } ;
{ system_privilege
| role
| ALL PRIVILEGES
}
grant_system_privileges::=
  [, { system_privilege
     | role
     | ALL PRIVILEGES
     }
  ]...
TO grantee_clause
  [ WITH ADMIN OPTION ]

grant_object_privileges::=
{ object_privilege | ALL [ PRIVILEGES ] }
  [ (column [, column ]...) ]
    [, { object_privilege | ALL [ PRIVILEGES ] }
       [ (column [, column ]...) ]
    ]...
on_object_clause
TO grantee_clause
  [ WITH HIERARCHY OPTION ]
```

```
  [ WITH GRANT OPTION ]

on_object_clause ::=
{ [ schema. ] object
| { DIRECTORY directory_name
  | JAVA { SOURCE | RESOURCE } [ schema. ] object
  }
}
```

- *system_privilege*: nombre del privilegio concedido.
- *role*: nombre del rol concedido.
- *IDENTIFIED BY*: cuando se concede un privilegio de sistema debe incluirse el usuario (si no existe) y *password*.
- *WITH ADMIN OPTION*: para permitir al usuario conceder el mismo privilegio a otros usuarios.
- *grantee_clause*: indica los usuarios o roles a los que se concede el privilegio.
- *grant_object_privileges*: cláusulas para privilegios de objetos.
- *object_privilege*: nombre del privilegio de objeto.
- *ALL [PRIVILEGES]*: todos los privilegios sobre el objeto.
- *Column*: columna de la tabla o vista sobre la que se concede el privilegio.
- *on_object_clause*: identifica el objeto sobre el que se conceden los privilegios.
- *WITH GRANT OPTION*: permite al usuario conceder y revocar el privilegio a otros.
- *WITH HIERARCHY OPTION*: Permite conceder el privilegio en todos los subobjetos, como subvistas.
- *Object*: nombre del objeto sobre el que se conceden los privilegios. Pueden ser tablas y vistas, secuencias, procedimientos, funciones o paquetes (funciones agrupadas), tipos de datos definidos por el usuario, alias de los objetos anteriores, directorios, librerías, operadores o índices, código fuente de Java y clases de Java.

Gestión de roles de usuarios

Normalmente los roles sirven para facilitar la gestión de aplicaciones de bases de datos utilizando los llamados roles de aplicación y de privilegios para grupos de usuarios con similares requerimientos.

Crear un rol

Para ello usamos el comando *CREATE ROLE*:

```
create_role::=
CREATE ROLE role
   [ NOT IDENTIFIED
   | IDENTIFIED { BY password
                | USING [ schema. ] package
                }
   ] ;
```

- *role*: nombre del rol a crear.
- *NOT IDENTIFIED*: no se requiere una contraseña para activar el rol.
- *IDENTIFIED*: para indicar que un usuario debe ser autorizado por el método especificado antes de que se active el rol con el comando *SET ROLE*.
- *BY password*: indica el *password* necesario para poder activar el rol por parte del usuario.

Por ejemplo, para crear un rol director con un *password*:

```
CREATE ROLE ebanca_director IDENTIFIED BY warehouse;
```

Así los usuarios a los que se conceda el rol deben especificar el *password* para activar el rol.

Activar un rol

Con el comando *SET ROLE* podemos activar o desactivar un rol (cuando iniciamos sesión todos salvo los activos por defecto están inactivos).

```
SET ROLE
( role_name [ IDENTIFIED BY password ]
| ALL [EXCEPT role1, role2, ... ]
| NONE );
```

- *role_name*: nombre del rol a activar.
- *IDENTIFIED BY password*: contraseña para activarlo si es que se creó con contraseña.
- *ALL*: indica que todos los roles deben activarse para dicho usuario y sesión salvo los iniciados por *EXCEPT*.
- *NONE*: desactiva todos los roles incluidos los activos por defecto.

Modificar un rol

El comando es muy similar así como el significado de las distintas cláusulas.

```
ALTER ROLE clerk IDENTIFIED EXTERNALLY;
ALTER ROLE role
  { NOT IDENTIFIED
  | IDENTIFIED
      { BY password
      | USING [ schema. ] package
      }
  } ;
```

Eliminar roles

```
DROP ROLE role ;
```

CREAR LISTAS DE CONTROL DE ACCESO

Oracle permite crear listas de control de acceso para restringir los equipos a los que ciertos usuarios tienen acceso. Para tal fin dispone de un paquete (conjunto de funciones) llamado *DBMS_NETWORK_ACL_ADMIN* con las funciones necesarias.

El proceso requiere primero crear la lista de acceso y los privilegios, a continuación se asigna dicha lista a uno o más *hosts* de una red.

Creando la lista de control

Con el procedimiento *DBMS_NETWORK_ACL_ADMIN.CREATE_ACL* pasándole los parámetros correspondientes según la sintaxis siguiente:

```
BEGIN
 DBMS_NETWORK_ACL_ADMIN.CREATE_ACL (
  acl          => 'file_name.xml',
  description  => 'file description',
  principal    => 'user_or_role',
  is_grant     => TRUE|FALSE,
  privilege    => 'connect|resolve',
  start_date   => null|timestamp_with_time_zone,
  end_date     => null|timestamp_with_time_zone);
END;
```

- *acl*: nombre del fichero *xml* generado con los datos de la *acl*.
- *description*: breve descripción del objeto de esta *acl*.
- *principal*: la cuenta principal de usuario al que se concede o deniega el permiso.
- *is_grant*: para especificar si se concede o no el permiso.
- *privilege*: el permiso es *connect* para permitir al usuario conecta ra servicios en equipos externos (cuando el usuario hace uso de los paquetes *UTL_TCP*, *UTL_HTTP*, *UTL_SMTP* y *UTL_MAIL*) o *resolve* para permitir resolver direcciones IP (cuando se hace uso del paquete *UTIL_INADDR*).
- *start_date*/*end_date*: (opcional) fecha de comienzo/fin de la lista.

Asignar los hosts

Tras crear la lista se requiere incluir los *hosts* para los que se creó la lista, para lo cual usamos el procedimiento almacenado:

DBMS_NETWORK_ACL_ADMIN.ASSIGN_ACL

```
BEGIN
 DBMS_NETWORK_ACL_ADMIN.ASSIGN_ACL (
  acl         => 'file_name.xml',
  host        => 'network_host',
  lower_port  => null|port_number,
  upper_port  => null|port_number);
END;
/
```

- *acl*: nombre de la *acl*.
- *host*: nombre del equipo.
- *lower_port*/*upper_port*: (opcional) el rango de puertos permitido.

El paquete dispone de varios procedimientos más para la gestión de *acls*. Para su uso recomendamos consultar el manual oficial.

AUTOMATIZACIÓN TAREAS PL/SQL

Al igual que en MySQL en *Oracle* existe la posibilidad de ampliar la funcionalidad usando un lenguaje específico de *Oracle* llamado PL/SQL y que incluye los elementos característicos de los lenguajes de programación es decir variables, sentencias de control de flujo, bucles, etc.

PL/SQL es un lenguaje estructurado. Su unidad básica es el bloque. Un bloque PL/SQL tiene 3 partes: zona de declaraciones, zona ejecutable y zona de tratamiento de excepciones. La sintaxis de un bloque es la siguiente:

```
[ DECLARE
  -- declaraciones ]
BEGIN
  -- sentencias
[ EXCEPTION
  -- tratamiento de excepciones ]
END;
```

Mediante bloques pueden construirse procedimientos, funciones y disparadores. Es posible el anidamiento de bloques.

En esta sección resumimos brevemente la sintaxis del lenguaje, especialmente la referida a aspectos de administración, incluyendo algunos ejemplos.

ELEMENTOS BÁSICOS DEL LENGUAJE

Una línea en PL/SQL contiene grupos de caracteres conocidos como unidades léxicas, que pueden ser clasificadas como:

Delimitadores

Son símbolos que tiene un significado especial en PL/SQL. Entre ellos están los operadores aritméticos, lógicos o relacionales.

Identificadores

Son empleados para nombrar objetos de programas en PL/SQL así como a unidades dentro del mismo, estas unidades y objetos incluyen constantes, cursores, variables, subprogramas, excepciones y paquetes.

Literales

Son un valor de tipo numérico, carácter, cadena o lógico no representado por un identificador (es un valor explícito).

Comentarios

Son una aclaración que el programador incluye en el código. Son soportados 2 estilos de comentarios, el de línea simple y de multilínea, para lo cual son empleados ciertos caracteres especiales como son:

```
-- Linea simple
/*
Conjunto de Lineas
*/
```

Declaraciones

Todo bloque, subprograma o paquete PL/SQL consta de una parte declarativa en la que incluimos la declaración de variables y constantes con sus valores respectivos.

TIPOS DE DATOS

Además de los habituales tipos numéricos, de carácter, etc. PL dispone de tipos más complejos como son los tipos registro, tabla y *array*.

Registros PL/SQL

Declaración de un registro.

Un registro es una estructura de datos en PL/SQL, almacenados en campos, cada uno de los cuales tiene su propio nombre y tipo y que se tratan como una sola unidad lógica.

La sintaxis general es la siguiente:

```
TYPE <nombre> IS RECORD
(
campo <tipo_datos> [NULL | NOT NULL]
[,<tipo_datos>...]
);
```

El siguiente ejemplo crea un tipo PAIS, que tiene como campos el código, el nombre y el continente.

```
TYPE PAIS IS RECORD
(
CO_PAIS      NUMBER  ,
DESCRIPCION VARCHAR2(50),
CONTINENTE   VARCHAR2(20));
```

Los registros son un tipo de datos, por lo que podremos declarar variables de dicho tipo de datos.

Pueden asignarse todos los campos de un registro utilizando una sentencia *SELECT*. En este caso hay que tener cuidado en especificar las columnas en el orden conveniente según la declaración de los campos del registro.

Se puede declarar un registro basándose en una colección de columnas de una tabla, vista o cursor de la base de datos mediante el atributo *%ROWTYPE*.

Por ejemplo, puedo declarar una variable de tipo registro como *PAISES%ROWTYPE;*

```
DECLARE
  miPAIS PAISES%ROWTYPE;
BEGIN
  /* Sentencias ... */
END;
```

Lo cual significa que el registro *miPAIS* tendrá la siguiente estructura: *CO_PAIS NUMBER*, *DESCRIPCION VARCHAR2(50)*, *CONTINENTE VARCHAR2(20)*.

Tablas PL/SQL

Las tablas de PL/SQL son tipos de datos que nos permiten almacenar varios valores del mismo tipo de datos. Es similar a un *array*, tiene dos componentes: Un índice que permite acceder a los elementos en la tabla PL/SQL y una columna de escalares o registros que contiene los valores de la tabla PL/SQL. Puede incrementar su tamaño dinámicamente.

La sintaxis general para declarar una tabla de PL es la siguiente:

```
TYPE <nombre_tipo_tabla> IS TABLE OF
<tipo_datos> [NOT NULL]
INDEX BY BINARY_INTEGER ;
```

Una vez que hemos definido el tipo, podemos declarar variables y asignarle valores.

```
DECLARE
   /* Definimos el tipo operacion como tabla PL/SQL */
   TYPE operacion IS TABLE OF NUMBER INDEX BY BINARY_INTEGER ;
   /* Declaramos una variable del tipo operacion */
   op operacion;
BEGIN
   op(1) := 1;
   op(2) := 2;
   op(3) := 3;
END;
```

Tablas PL/SQL de registros

Es posible declarar elementos de una tabla PL/SQL como de tipo registro.

```
DECLARE

  TYPE PAIS IS RECORD
  (
    CO_PAIS      NUMBER NOT NULL ,
    DESCRIPCION VARCHAR2(50),
    CONTINENTE  VARCHAR2(20)
  );
  TYPE PAISES IS TABLE OF PAIS INDEX BY BINARY_INTEGER ;
  tPAISES PAISES;
BEGIN

  tPAISES(1).CO_PAIS := 27;
  tPAISES(1).DESCRIPCION := 'ITALIA';
  tPAISES(1).CONTINENTE  := 'EUROPA';

END;
```

ESTRUCTURAS DE CONTROL DE FLUJO EN PL/SQL

En PL/SQL solo disponemos de la estructura condicional IF. Su sintaxis se muestra a continuación:

Sentencia IF-THEN

Similar al caso de MySQL permite asociar una condición con una secuencia de instrucciones tal como se observa en la sintaxis:

```
IF condition THEN
   secuencia de instrucciones
END IF;
```

Sentencia IF-THEN-ELSE

Es el caso anterior con dos posibles alternativas para la condición:

```
IF condition THEN
   secuencia de instrucciones
ELSE
   secuencia de instrucciones
END IF;
```

Sentencia IF-THEN-ELSIF

Para casos en que debemos seleccionar una acción de entre varias alternativas posibles que a su vez son excluyentes.

```
IF condicion1 THEN
   secuencia de instrucciones
ELSIF condition2 THEN
   secuencia de instrucciones
ELSE
   secuencia de instrucciones
END IF;
```

Sentencia CASE

De manera similar a la anterior caso podemos usar CASE mejorando la legibilidad del programa.

```
CASE selector
   WHEN expresion1 THEN   secuencia de instrucciones 1;
   WHEN expresion2 THEN   secuencia de instrucciones 2;
   ...
   WHEN expresionN THEN   secuencia de instrucciones N;
  [ELSE   secuencia de instrucciones N+1;]
END CASE;
```

Donde la palabra *selector* indica una expresión que suele ser una variable usada en las expresiones asociadas a *WHEN*.

ESTRUCTURAS DE CONTROL ITERATIVAS EN PL/SQL

En PL/SQL tenemos a nuestra disposición los iteradores *LOOP WHILE* y *FOR*.

Todos ellos admiten el uso de etiquetas o *labels* que permiten identificarlos a ellos y a sus variables, algo especialmente útil cuando usamos varios de ellos anidados. Para etiquetarlos usamos la sintaxis:

```
<<nombre loop>>
Definición del bucle
END nombre loop
```

Iterador LOOP

El bucle *LOOP*, se repite tantas veces como sea necesario hasta que se fuerza su salida con la instrucción *EXIT*. Su sintaxis es la siguiente:

```
LOOP
   instrucciones
      IF (expresion) THEN
         instrucciones
        EXIT;
      END IF;
END LOOP;
```

En este caso hemos añadido un IF para ilustrar el uso de *EXIT* que fuerza al bucle a terminar de forma inmediata e incondicional. Esta instrucción solo puede incluirse dentro de bucles *LOOP*. También podemos condicionar la salida usando *EXIT-WHEN* condición.

Podemos usar etiquetas del siguiente modo:

```
<<etiqueta_externa>>
LOOP
   ...
   LOOP
      ...
      EXIT etiqueta_externa WHEN condicion  --salimos de ambos bucles
   END LOOP;
   ...
END LOOP etiqueta_externa;
```

En este caso hemos incluido dos bucles anidados para ilustrar el hecho de que un EXIT nos sacaría del bucle indicado, en este caso el más externo.

Iterador WHILE-LOOP

El bucle *WHILE*, se repite mientras que se cumpla expresión.

```
WHILE (expresion) LOOP
        -- instrucciones
END LOOP;
```

En este caso se repiten las sentencias mientras se cumpla la condición. Otros lenguajes disponen de iteradores que se repiten al menos una vez como ya vimos en el propio MySQL. En *Oracle* no existe esta posibilidad pero podmeos implementarla usando el iterador anterior *LOOP* con *EXIT*.

```
LOOP
   secuencia de instrucciones
   EXIT WHEN expression booleana;
END LOOP;
```

Otra forma de hacer lo mismo con *WHILE* y una expresión *booleana* (que devuelve verdadero, *TRUE* o falso, *FALSE*).

```
done := FALSE;
WHILE NOT done LOOP
   secuencia de instrucciones
   done := expression booleana;
END LOOP;
```

Iterador FOR

El bucle *FOR*, se repite tanta veces como le indiquemos en los identificadores numéricos inicio y final.

```
FOR contador IN [REVERSE] inicio..final LOOP
    -- Instrucciones

END LOOP;
```

En el caso de especificar *REVERSE* el bucle se recorre en sentido inverso.

Los contadores son considerados variables locales al bloque al que pertenecen (son declaradas implícitamente) de manera que para hacer referencia a variables con el mismo nombre dentro de otros bloques debemos anteponer la etiqueta identificadora del bloque.

CURSORES EN PL/SQL

PL/SQL utiliza cursores para gestionar las instrucciones *SELECT*. Un cursor es un conjunto de registros devuelto por una instrucción SQL. Técnicamente los cursores son fragmentos de memoria que reservados para procesar los resultados de una consulta *SELECT*.

```
CURSOR cursor_name [(parameter[, parameter]...)]
   [RETURN return_type] IS select_statement;
```

PL/SQL distingue entre cursos implícitos que devuelven cero o una sola fila y los explícitos que pueden devolver varias.

Para trabajar con un cursor necesitamos realizar las siguientes tareas:

Declarar el cursor

```
DECLARE CURSOR cursor_name [(parameter[, parameter]...)]
   [RETURN return_type] IS select_statement;
```

Donde el tipo devuelto representa una fila de una tabla

```
cursor_parameter_name [IN] datatype [{:= | DEFAULT} expression]
```

Abrir el cursor

Usamos la instrucción *OPEN*.

```
OPEN nombre_cursor;
```

o bien, en el caso de un cursor con parámetros:

```
OPEN nombre_cursor(valor1, valor2, ..., valorN);
```

Leer los datos del cursor

Con la instrucción *FETCH*.

```
FETCH nombre_cursor INTO lista_variables;
```

o bien si usamos el tipo registro:

```
FETCH nombre_cursor INTO registro_PL/SQL;
```

Y lo procesamos con un *loop*.

```
LOOP
   FETCH nombre_cursos INTO registro;
   EXIT WHEN nombre_cursor%NOTFOUND;
   -- procesamos datos
END LOOP;
```

Cerrar el cursor

Para liberar los recursos y cerrar el cursor usamos la instrucción *CLOSE*.

```
CLOSE nombre_cursor;
```

PL/SQL dispone de un tipo especial de cursores llamado cursores de actualización que permiten actualizar datos de las filas devueltas por el cursor. Para declararlos la sintaxis es similar.

Declaración y utilización de cursores de actualización

```
CURSOR nombre_cursor IS
     instrucción_SELECT
    FOR UPDATE
```

Para actualizar los datos del cursor hay que ejecutar una sentencia *UPDATE* especificando la cláusula *WHERE CURRENT OF* <nombre_cursor>.

```
UPDATE <nombre_tabla> SET
<campo_1> = <valor_1>
[,<campo_2> = <valor_2>]
WHERE CURRENT OF <nombre_cursor>
```

MANEJO DE ERRORES EN PL/SQL

En PL/SQL una advertencia o error es considera una excepción. Pueden ser propias del lenguaje (como divisiones por cero o ausencia de datos en un cursor) o definidas por el programador en la parte declarativa de cualquier bloque, subprograma o paquete.

Las excepciones se controlan dentro de su propio bloque. La estructura de bloque de una excepción se muestra a continuación.

```
DECLARE
-- Declaraciones
BEGIN
-- Ejecucion
EXCEPTION
-- Excepcion
END;
```

Cuando ocurre un error, se ejecuta la porción del programa marcada por el bloque *EXCEPTION*, transfiriéndose el control a ese bloque de sentencias.

El siguiente ejemplo muestra un bloque de excepciones que captura las excepciones *NO_DATA_FOUND* y *ZERO_DIVIDE*. Cualquier otra excepción será capturada en el bloque *WHEN OTHERS THEN*.

```
DECLARE
-- Declaraciones
BEGIN
 -- Ejecucion
EXCEPTION
WHEN NO_DATA_FOUND THEN
-- Se ejecuta cuando ocurre una excepcion de tipo NO_DATA_FOUND
WHEN ZERO_DIVIDE THEN
-- Se ejecuta cuando ocurre una excepcion de tipo ZERO_DIVIDE

WHEN OTHERS THEN
-- Se ejecuta cuando ocurre una excepcion de un tipo no tratado
  -- en los bloques anteriores

END;
```

Como ya hemos dicho cuando ocurre un error, se ejecuta el bloque *EXCEPTION*, transfiriéndose el control a las sentencias del bloque. Una vez finalizada la ejecución del bloque de *EXCEPTION* no se continúa ejecutando el bloque anterior.

Si existe un bloque de excepción apropiado para el tipo de excepción se ejecuta dicho bloque. Si no existe un bloque de control de excepciones adecuado al tipo de excepción se ejecutará el bloque de excepción *WHEN OTHERS THEN* (si existe!). *WHEN OTHERS* debe ser el último manejador de excepciones.

PL/SQL proporciona un gran número de excepciones predefinidas que permiten controlar las condiciones de error más habituales.

Las excepciones predefinidas no necesitan ser declaradas. Simplemente se utilizan cuando estas son lanzadas por algún error determinado.

PL/SQL permite al usuario definir sus propias excepciones, las que deberán ser declaradas y lanzadas explícitamente utilizando la sentencia *RAISE*.

La sentencia *RAISE* permite lanzar una excepción en forma explícita. Es posible utilizar esta sentencia en cualquier lugar que se encuentre dentro del alcance de la excepción.

Las excepciones deben ser declaradas en el segmento *DECLARE* de un bloque, subprograma o paquete. Se declara una excepción como cualquier otra variable, asignándole el tipo *EXCEPTION*. Las mismas reglas de alcance aplican tanto sobre variables como sobre las excepciones. Veamos un ejemplo completo y comentado de todo lo explicado:

```
DECLARE
-- Declaraciones
    MyExcepcion EXCEPTION;
BEGIN
-- Ejecucion
EXCEPTION
-- Excepcion
END;
DECLARE
-- Declaramos una excepcion identificada por VALOR_NEGATIVO
    VALOR_NEGATIVO EXCEPTION;
    valor NUMBER;
BEGIN
-- Ejecucion
valor := -1;
    IF valor < 0 THEN
          RAISE VALOR_NEGATIVO;
    END IF;

EXCEPTION
-- Excepcion
WHEN VALOR_NEGATIVO THEN
    dbms_output.put_line('El valor no puede ser negativo');
END;
DECLARE
err_num NUMBER;
err_msg VARCHAR2(255);
  result  NUMBER;
BEGIN
SELECT 1/0 INTO result
  FROM DUAL;
EXCEPTION
WHEN OTHERS THEN
err_num := SQLCODE;
err_msg := SQLERRM;
DBMS_OUTPUT.put_line('Error:'||TO_CHAR(err_num));
DBMS_OUTPUT.put_line(err_msg);
END;
```

En ocasiones queremos enviar un mensaje de error personalizado al producirse una excepción PL/SQL. Para ello es necesario utilizar la instrucción *RAISE_APPLICATION_ERROR*.

La sintaxis general es la siguiente:

```
RAISE_APPLICATION_ERROR(<error_num>,<mensaje>);
```

Siendo *error_num* un entero negativo comprendido entre -20001 y -20999 y mensaje la descripcion del error.

```
DECLARE
   v_div NUMBER;
BEGIN
```

```
    SELECT 1/0 INTO v_div FROM DUAL;
EXCEPTION
    WHEN OTHERS THEN
    RAISE_APPLICATION_ERROR(-20001,'No se puede dividir por cero');
END;
```

BLOQUES PL/SQL

Como ya mencionamos un programa de PL/SQL está compuesto por bloques que pueden ser anónimos (sin nombre) o subprogramas (procedimientos, funciones o disparadores).

Los bloques PL/SQL presentan una estructura específica compuesta de tres partes bien diferenciadas:

La sección declarativa en donde se declaran todas las constantes y variables que se van a utilizar en la ejecución del bloque.

La sección de ejecución que incluye las instrucciones a ejecutar en el bloque PL/SQL.

La sección de excepciones en donde se definen los manejadores de errores que soportará el bloque PL/SQL.

Cada una de las partes anteriores se delimita por una palabra reservada, de modo que un bloque PL/SQL se puede representar como sigue:

```
     [ declare | is | as ]
     /*Parte declarativa*/
begin
     /*Parte de ejecucion*/
[ exception ]
     /*Parte de excepciones*/
end;
```

De las anteriores partes, únicamente la sección de ejecución es obligatoria, que quedaría delimitada entre las cláusulas *BEGIN* y *END*. Los bloques anónimos identifican su parte declarativa con la palabra reservada *DECLARE*.

Sección de declaración de variables

En esta parte se declaran las variables que va a necesitar nuestro programa. Una variable se declara asignandole un nombre o "identificador" seguido del tipo de valor que puede contener. También se declaran cursores, de gran utilidad para la consulta de datos, y excepciones definidas por el usuario. También podemos especificar si se trata de una constante, si puede contener valor nulo y asignar un valor inicial.

La sintaxis generica para la declaracion de constantes y variables es:

```
nombre_variable [CONSTANT] <tipo_dato> [NOT NULL][:=valor_inicial]
```

Donde tipo_dato: es el tipo de dato que va a poder almacenar la variable, la cláusula CONSTANT indica la definición de una constant.

La cláusula *NOT NULL* impide que a una variable se le asigne el valor nulo, y por tanto debe inicializarse a un valor diferente de *NULL*.

Como hemos visto anteriormente los bloques de PL/SQL pueden ser bloques anónimos (*scripts*) y subprogramas.

Los subprogramas son bloques de PL/SQL a los que asignamos un nombre identificativo y que normalmente almacenamos en la propia base de datos para su posterior ejecución. Entre ellos distinguimos procedimientos, funciones y disparadores o *triggers* y subprogramas sin nombre o anónimos.

Procedimientos

Como en el caso de MySQL son bloques PL/SQL que no pueden devolaver ningún valor. Un procedimiento tiene un nombre, un conjunto de parámetros (opcional) y un bloque de código.

Su sintaxis es:

```
CREATE [OR REPLACE]
PROCEDURE <procedure_name> [(<param1> [IN|OUT|IN OUT] <type>,
                             <param2> [IN|OUT|IN OUT] <type>, ...)]
IS
  -- Declaracion de variables locales
BEGIN
  -- Sentencias
[EXCEPTION]
  -- Sentencias control de excepcion
END [<procedure_name>];
```

Los parámetros pueden ser de entrada (*IN*), de salida (*OUT*) o de entrada salida (*IN OUT*). El valor por defecto es *IN* y se toma ese valor en caso de que no especifiquemos nada.

En el siguiente ejemplo se actualiza el saldo de una cuenta.

```
CREATE OR REPLACE
PROCEDURE Actualiza_Saldo(cuenta NUMBER,
                          new_saldo NUMBER DEFAULT 10 )
IS
  -- Declaracion de variables locales
BEGIN
  -- Sentencias
  UPDATE SALDOS_CUENTAS
    SET SALDO = new_saldo,
        FX_ACTUALIZACION = SYSDATE
  WHERE CO_CUENTA = cuenta;

END Actualiza_Saldo;
```

Funciones

Una función es un subprograma que devuelve un valor.

La sintaxis para construir funciones es la siguiente:

```
CREATE [OR REPLACE]
FUNCTION <nombre funcion>[(<param1> IN <tipo>, <param2> IN <tipo>, ...)]
RETURN <return_type>
IS
result <return_type>;
BEGIN
```

```
return(result);
[EXCEPTION]
-- Sentencias control de excepcion
END [<fn_name>];

En el siguiente ejemplo obtenemos el precio de un producto:
CREATE OR REPLACE
FUNCTION fn_Obtener_Precio(p_producto VARCHAR2)
RETURN NUMBER
IS
result NUMBER;
BEGIN
SELECT PRECIO INTO result
  FROM PRECIOS_PRODUCTOS
  WHERE CO_PRODUCTO = p_producto;
return(result);
EXCEPTION
WHEN NO_DATA_FOUND THEN
  return 0;
END ;
```

Las funciones pueden utilizarse en bloques PL/SQL así como en sentencias SQL de manipulación de datos (*SELECT*, *UPDATE*, *INSERT* y *DELETE*):

Triggers

Un *trigger* es un bloque PL/SQL asociado a una tabla, que se ejecuta como consecuencia de una determinada instrucción SQL (una operación DML: *INSERT*, *UPDATE* o *DELETE*) sobre dicha tabla.

La sintaxis para crear un *trigger* es la siguiente:

```
CREATE [OR REPLACE] TRIGGER <nombre_trigger>
{BEFORE|AFTER}
                {DELETE|INSERT|UPDATE [OF col1, col2, ..., colN]
            [OR {DELETE|INSERT|UPDATE [OF col1, col2, ..., colN]...]}
ON <nombre_tabla>
[FOR EACH ROW [WHEN (<condicion>)]]
DECLARE
-- variables locales
BEGIN
--Sentencias
[EXCEPTION]
-- Sentencias control de excepcion
END <nombre_trigger>;
```

Los *triggers* pueden definirse para las operaciones *INSERT*, *UPDATE* o *DELETE*, y pueden ejecutarse antes o después de la operación. El modificador *BEFORE AFTER* indica que el *trigger* se ejecutará antes o despues de ejecutarse la sentencia SQL definida por *DELETE INSERT UPDATE*. Si incluimos el modificador OF el *trigger* solo se ejecutará cuando la sentencia SQL afecte a los campos incluidos en la lista.

El alcance de los disparadores puede ser la fila o de orden. El modificador *FOR EACH ROW* indica que el *trigger* se disparará cada vez que se realizan operaciones sobre una fila de la tabla. Si se acompaña del modificador *WHEN*, se establece una restricción; el *trigger* solo actuará, sobre las filas que satisfagan la restricción.

Dentro del ambito de un *trigger* disponemos de las variables *OLD* y *NEW*. Estas variables se utilizan del mismo modo que cualquier otra variable PL/SQL, con la salvedad de que no es necesario declararlas, son de tipo *%ROWTYPE* y contienen una copia del registro antes (*OLD*) y después (*NEW*) de la acción SQL *(INSERT, UPDATE, DELETE)* que ha ejecutado el *trigger*. Utilizando esta variable podemos acceder a los datos que se están insertando, actualizando o borrando.

ACTIVIDADES E.1

- Instala e inicia Oracle Express mediante tu navegador web y crea una nueva cuenta administrativa y otra para un usuario de bases de datos vía web.
- Carga el *script ebanca.sql* de la base de datos *ebanca* (con las correcciones necesarias para adaptarlo a Oracle) en el servidor Oracle usando la interfaz web.
- Crea un *trigger* que impida a los usuarios hacer operaciones de modificación de datos en horas de trabajo. En caso de que se den dichas operaciones se levantará un error informando al usuario (*RAISE ERROR*).
- Crea un *trigger* que registre en una nueva tabla el usuario, la fecha y hora en que se ha modificado el saldo anterior y nuevo de una cuenta.
- Crea un procedimiento que incremente el saldo de una cuenta de inversión un porcentaje dado para aquellas cuentas que superen un cierto saldo.
- Crea una función que devuelva el saldo neto de un cliente en todas sus cuentas.
- Añade al ejercicio anterior la posibilidad de que el cliente no tenga ninguna de modo que se genere un error que sea manejado informando directamente al usuario.
- Comprueba, desde el panel de administración las opciones relativas a las bases de datos, memoria, *tablespaces* y usuarios. Relaciónalo con lo visto en teoría.
- Crea un usuario con privilegios de lectura sobre las tablas de ebanca y de modificación sobre el campo saldo de la tabla cuentas.
- Crea un rol con permisos de lectura sobre las tablas de ebanca y asígnalo al usuario invitado.
- Observa las estadísitcas del servidor Oracle usando los botones de sesiones, estadísitcas de sistema, *top sql* y operaciones largas.
- Repite los ejercicios anteriores usando el programa *SQL*Plus* para conectarte al servidor Oracle.

APÉNDICE

F

Introducción a la administración de sistemas gestores: SQL Server

ARQUITECTURA DE SQL SERVER

Los componentes de SQL se dividen en dos amplios grupos: el motor relacional y las utilidades externas.

SQL Server Database Engine

Es el núcleo del servidor, se encarga de la gestión del trabajo de las bases de datos.

Incluye los siguientes componentes:

- **Algebrizer**: comprueba los comandos SQL y los transforma para ser procesados internamente.
- **Query optimizer**: determina la forma óptima de procesar los comandos SQL.
- **Query Engine**: ejecuta finalmente los comandos de acuerdo con el plan generado por el optimizador.
- **Motor de almacenamiento**: procesa las operaciones de E/S a disco.
- **Buffer Manager**: optimiza el uso dde memoria reucieno la dependencia de acceso a disco.
- **Checkpoint**: realiza operaciones de escritura de memoria a los ficheros.
- **Resource monitor**: optimiza la caché de consultas en relación a la carga actual de memoria.
- **Lock manager**: gestiona los bloqueos sobre distintos objetos de la base de datos.
- **SQLOS**: es una capa añadida que funciona como un pequeño sistema operativo capaz de gestionar recursos propios del servidor tales como memoria, hilos, peticiones de E/S, etc.

Transact-SQL

Del mismo modo que la mayoría de sistemas gestores, SQL Server implementa su propia verisón de SQL que incorpora el estándar además de extensiones y mejoras incluidas por SQL Server. Todos los comandos enviados al servidor deben ser comandos T-SQL válidos disntinguiendo entre los DML (comandos de modificación), DDL (comandos de definición) y DCL (comandos de control).

Policy based manamement

O PBM es el conjunto de declaraciones administrativas que afectan al servidor, a las bases de datos y a cualquier objeto de las mismas.

Es un componente que centraliza las tareas de administración.

NET Common Language Runtime

Es un componente que permite la integración de Visual Studio de Microsoft con SQL Server de forma que se pueda ampliar la funcionalidad de T-SQL tanto para crear prutinas almacenadas como tablas o bases de datos desde el propio Visual Studio.

Service Broker

Es un componente que permite balancear la carga de proceso de forma dinámica mediante el uso de *buffering*, colas e instancias para cada petición.

Replication services

Estos servicios sirven para poder hacer réplicas de datos en diferentes ubicaciones de manera dinámica así como poder realizar transacciones que impliquen datos distribuidos.

Búsquedas Full-text

Estas búsquedas, realizadas sobre todas las palabras de un texto se optimizan enormemente con índices *full-text* ya que en estos se index cada palabra del campo correpondiente.

Server Management Object (SMO)

Son un conjunto de objetos que porporcionan la configuración con el objeto de facilitar la programación y el desarrollode *scripts* en el servidor.

Servicios de SQL Server

Estos componentes son de tipo cliente y sirven para controlar o comunicarse con el servidor.

- **Agente SQL Server**

  Realiza trabajos automáticos del servidor.

- **Base de datos de correo**

  Permite envío de correo vía *smtp*. Este correo puede provenir del propio dservidor, código T-SQL, tareas programadas, alertas, etc.

- **Distributed Transaction Coordinator (DTC)**

  Proceso que gestiona transacciones que afectan a varios servidores.

Business Intelligence (BI)

Es el conjunto de herramientas que permiten la gestión de datos con el objeto de realizar análisis, informes, *data mining* y visualización. En este sentido incluye tres herramientas que pueden ser utilizadas con el *-Business inteligence Development Studio* (BIDS) una versión de Visual Studio para el soporte de BI compatible con Visual Studio.

- **Integration Services (IS)** permite la obtención y transformación de datos de toda clase de fuentes para ser integrados con la herramienta *extract-transform-load* (ETL) del servidor.
- **Analysis Services** para el análisis *online* de bases de datos multidimensionales mediante OLAP (*Online Analytical Processing*) así como métodos de búsqueda de patrones y tendencias en datos mediante *Data-Mining*.
- **Reporting Services (RS)** para la visualziación vía web y obtención de informes de datos y su exportación a distintos formatos como *pdf* o *Excel*.

Interfaces de usuario (UI) y otras herramientas

- **SQL Server Management Studio**

  Es la herramienta más potente y útil para el administrador y desaroolador de bases de datos. Porporciona una interfaz gráfica para la gestión de la mayoría de características de SQL Server tanto de administración como de desarrollo incluyendo el Solution Explorer para la gestión de proyectos.

- **SQL Server Configuration Manager**

  Herramienta utilizada para iniciar y detener el servidor así como para establecer parámetros de configuración y conectividad

- **SQL Profiler/Trace**

  Permite depurar aplicaciones y hacer seguimiento de ciertas trazas de eventos u operaciones sobre datos en tiempo real para facilitar la optimización del servidor.

- **Performance Monitor**

  Es una herramienta visual para monitorizar el estado del servidor y todos sus procesos y operaciones en un momento dado.

Diccionario de datos en SQL Server

Como en la mayoría de gestores de datos, SQL Server incorpora un diccionario de datos en el que se registran toda la información relativa a los distintos objetos de las bases de datos y del servidor.

Esta puede ser consultado con los comandos T-SQL o a través del el *Object Explorer* de *Management Studio.*

El diccionario se divide básicamente en dos partes.

- **Bases de datos del sistema**
  - Master: con información de las bases de datos del servidor.
  - Msdb: mantiene un listado de actividades de *backup* y tareas o *jobs* para cada usuario.
  - Tempdb: espacio temporal para creación de tablas, bases, rutinas almacenadas o para espacio temporal en el procesamiento de consultas.

- **Vistas**

  Es donde se almacena la información realmente ´tuil del diccionario se compone de las siguientes vistas:

  - *Catalogo*: contiene información de datos estáticos acerca de tablas, seguridad y configuración del servidor.
  - *Dynamic management view* (DMW): almacena información de estado del servidor en cuanto a recursos en uso y onexiones activas.
  - *Funciones de sistema y variables globales*: información de estado del servidor, de las bases de datos y variables de configuración.
  - *Information schema*: con información sobre objetos de cada base de datos siguiendo el estándar ISO de SQL.

Ediciones de SQL Server

Existen diversas ediciones del Servidor SQL a continuación veremos un listado de las mismas con una brve descripción. Finalmente mostraremos el proceso de instalación de la edición libre *express*.

- *Enterprise* (*Developer*) *edition*: es la más avanzada e incluye todas las posibilidades del servidor. Estaá oritentada a grandes sistemas de muchas bases de datos. La versión developer es igual pero solo liceniada para pruebas y desarrollo.

- *Standard edition*: es la edición para sistemas medios y recomendable para probar antes de pasar a la *Enterprise Edition*.

- *Workgroup edition*: edición más ligera ya que no incluye el componente Integration Services. Está orientada para pequeñois negocios.

- *Edición Web*: sirve para alojar aplicaciones de sitios web. En partículas la SQL Server Express Edition es una versión limitada a 4GB de espacio para bases de datos y no más de 25 usuarios. Recomendable para pequeñas aplicaciones *.Net* que utilicen SQL server como base de datos.
- *Edición compacta* o *Compact Edition* (CE): es un servidor compacto más limitado que el anterior y cuyo uso está orientado para aplicaciones de móviles.
- SQL *Data Services* (SDS): proporciona características de alta disponibilidad y es la base de datos de *Microsoft Azure*.

Instalación y configuración básica

Igual que Oracle, SQL Server dispone de una versión 2008 R2 express gratuita limitada para pruebas, no obstante la última versión SQL Server 2008 R2 de pago también está disponible en versión de prueba normalmente por un mes.

En esta sección describiremos el proceso de instlación de la versión *express*.

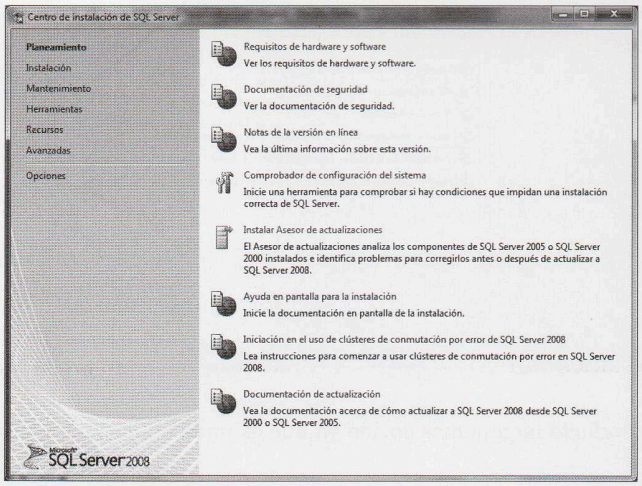
Es importante tener en consideración que, previamente a la instalación es recomendbale instalar las últimas actualizaciones del sistema operativo así como disponer de una cuenta de usuario para SQL Server.

Podemos encontrar el programa en la página oficial de Microsoft:

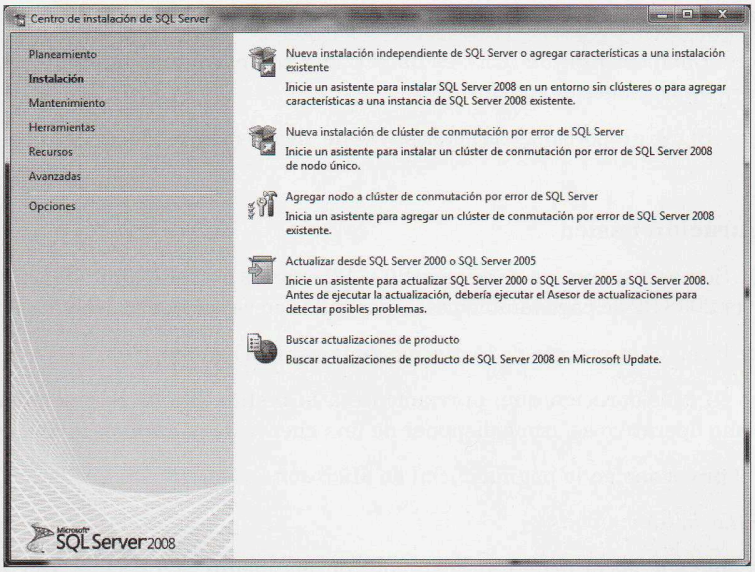
http://www.microsoft.com/downloads/

Una vez pulsemos doble clic en el fichero descargado (normalmente SQLEXP32_x86_ENU.exe) comenzara la extracción de ficheros que puede llevar unos minutos:

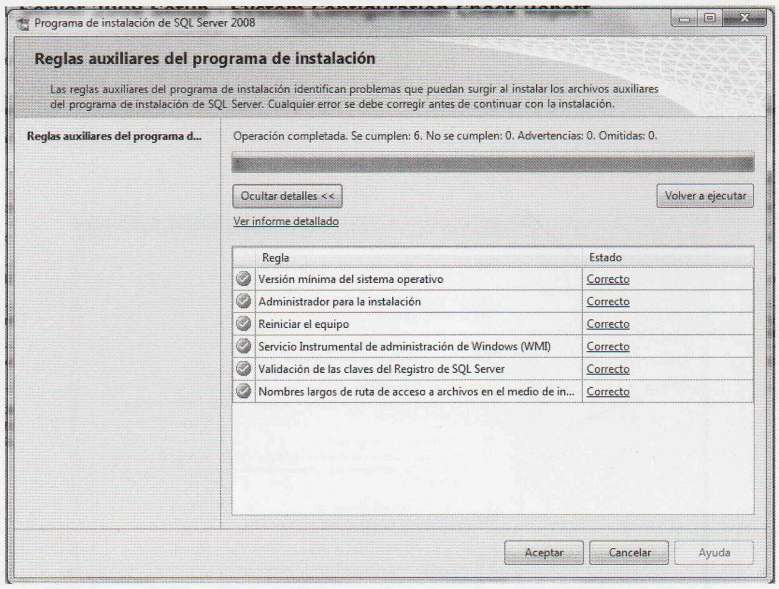
A continuación aparecerá la ventana de SQL *Server Installation Center*. En cuyo lado izquierdo veremos las diferentes opciones. La primera de ellas, *Planning*, nos permitirá estudiar los requerisitos de hardware y software entre otros componentes previos a la instalación.



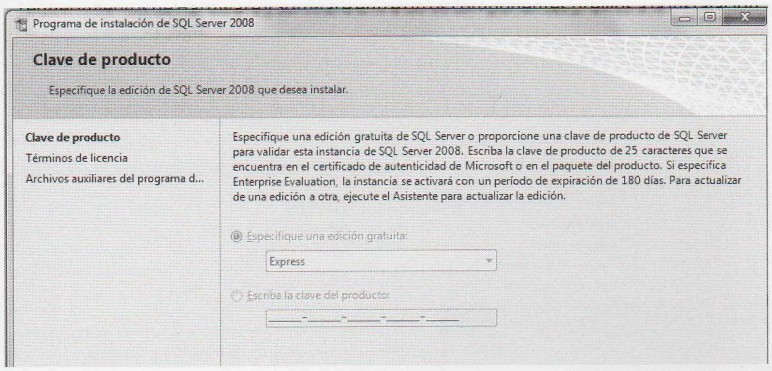
Iniciamos la instalación con nuestro escenario particular, en este caso pulsando en instalación nueva:



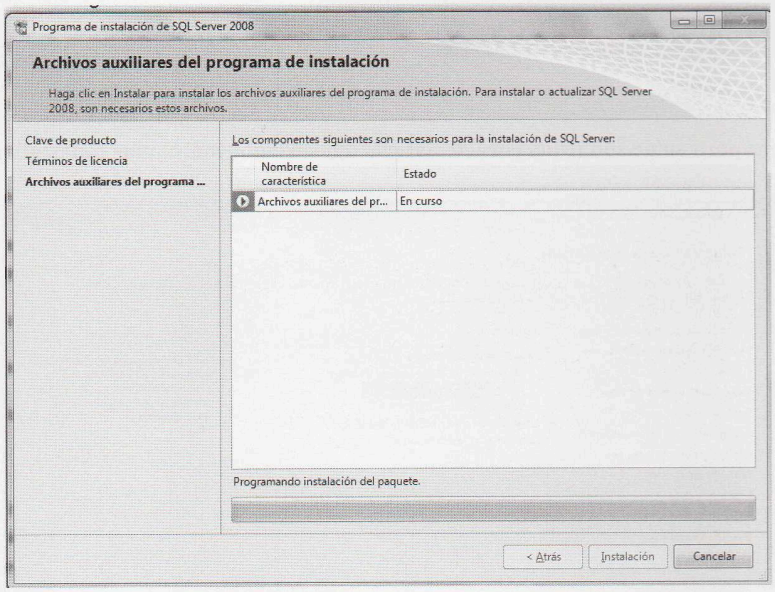
En la ventana *Reglas auxiliares* del programa de *Instalación* se validan todas las reglas de soporte para la instalación y hacemos clic en *Aceptar*.



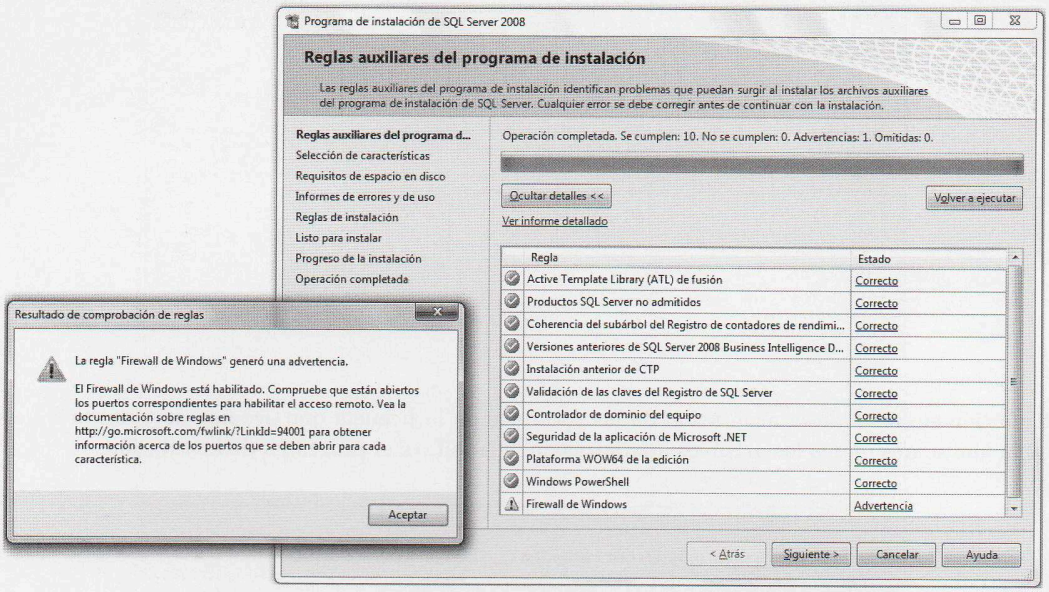
En la ventana *Clave del producto* no tenemos opción ya que es una versión *Express* libre:



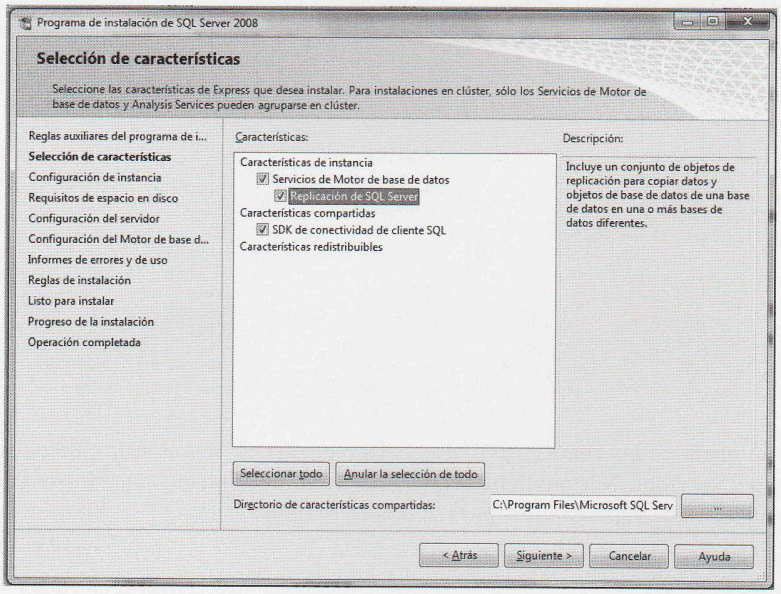
A continuación se nos pide la aceptación de los términos de la licencia que aceptamos y después veremos una pantalla en la que se nos indica los archivos auxiliares que necesitará el programa de instalación:



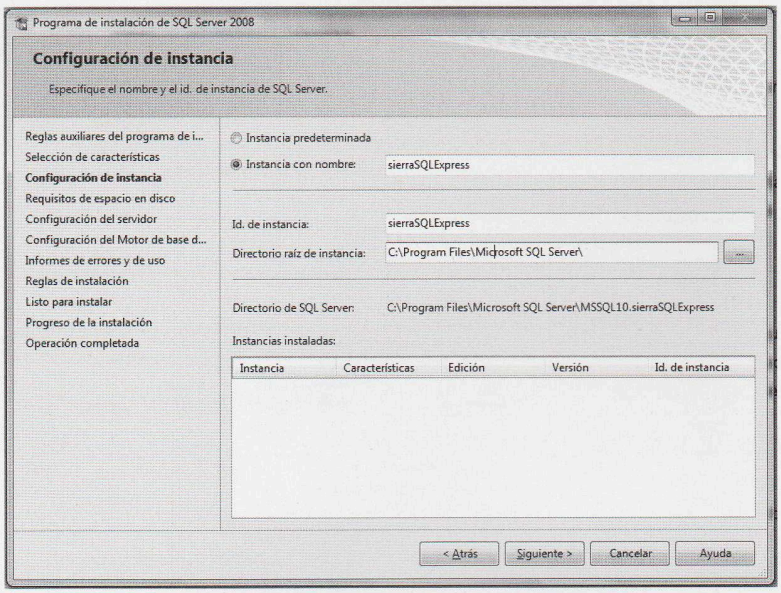
Finalmente se nos informa de las reglas auxiliares del programa de instalación que deben ser cumplidas para la correcta instalación del servidor. Normalmente vemos un aviso del *firewall* de Windows para advertirnos de la necesidad de abrir los puertos correspondientes para el acceso remoto:



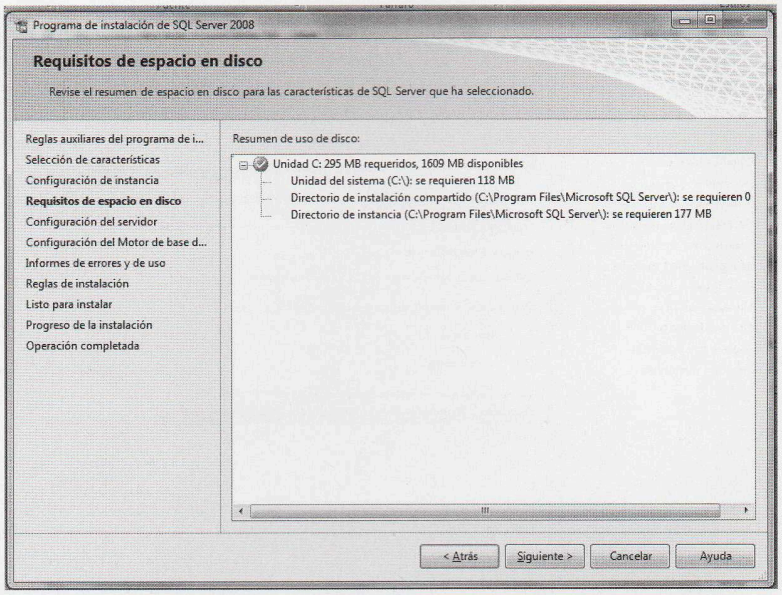
Pulsmos en siguiente para seleccionar las características del servidor marcamos todas ellas:



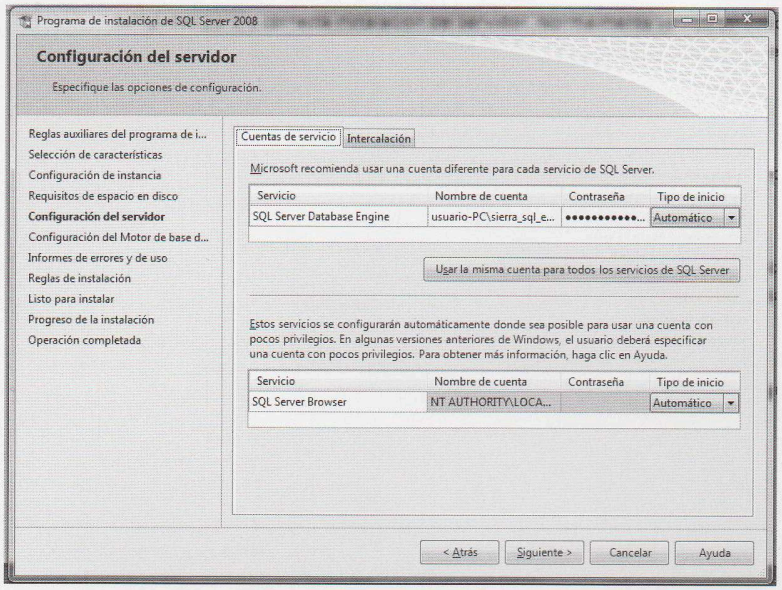
Continuamos pulsando *Siguiente* hasta que aparece el configurador de la instancia donde indicaremos el nombre de la instancia (*sierraSQLExpress*) y el directorio de la misma:



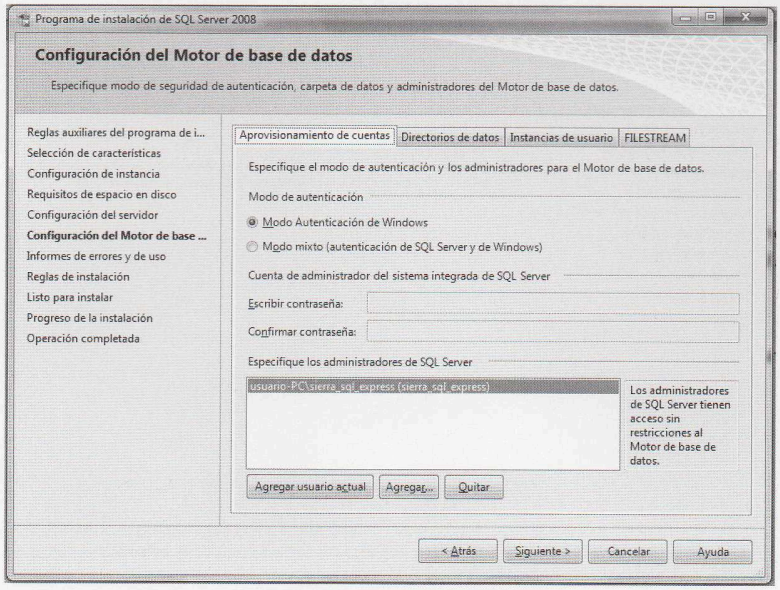
Nos informa del espcacio en disco requerido y disponible, si todo es correcto seguimos pulsando *Siguiente*:



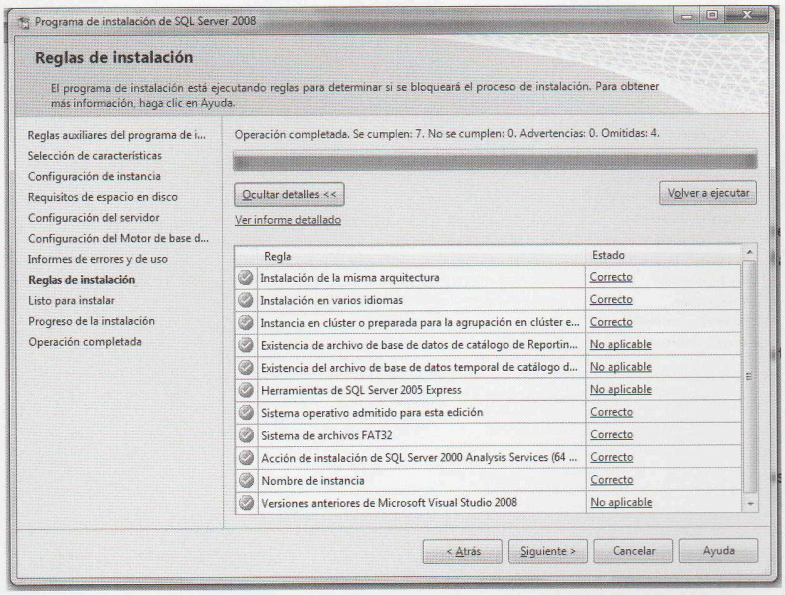
Se nos pregunta por las cuentas con las que se ejecutará el servidor. Podemos crear una ad hoc para hacer pruebas, en este caso *sierra_sql_express* con la misma contraseña:



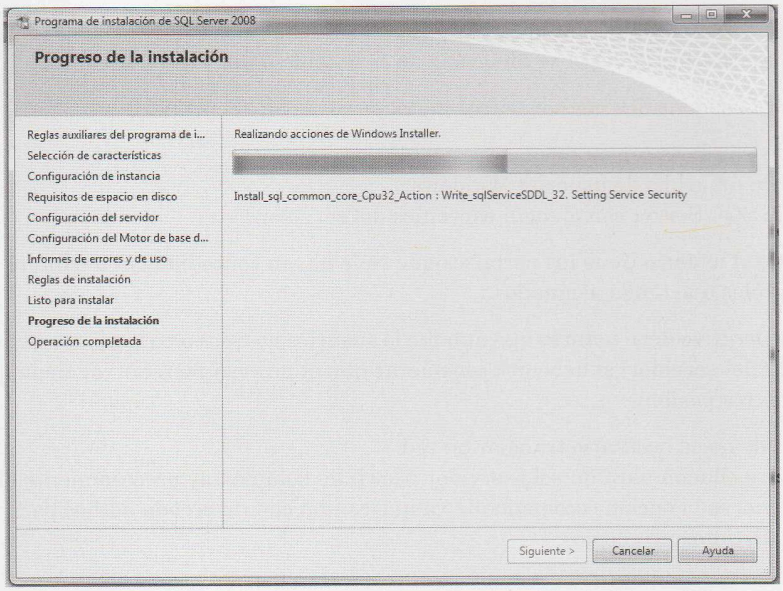
Configuramos el motor de bases de datos, entre otras cosas los directorios de datos y el tipo de acceso y usuarios administradores, en este caso el mismo de antes:



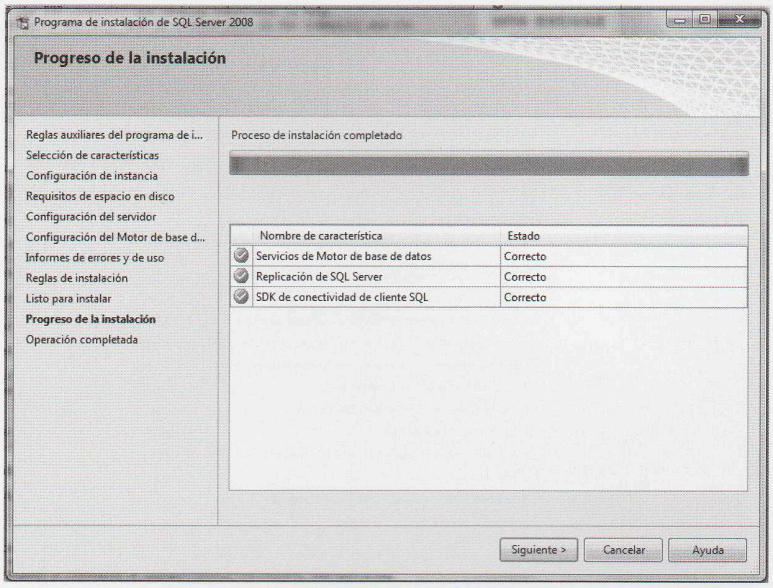
Completamos el formulario relativo a los informes de errores de uso según nuestro interés y pulsando *Siguiente* para completar la preparación de la instalación:



Si todo es correcto tras pulsar *Siguiente* se nos mostrarán las características a instalar del servidor y pulsando en *Instalar* comenzará la instalación:



Finalmente se nos informa de la finalización del proceso de instalación indicando los componentes instalados, y ya estamos listos para comenzar a usar el servidor:



Cuentas de usuario y permisos

La arquitectura de seguridad de SQL server pude representarse mediante el siguiente esquema:

La autenticación en SQL server es similar a MySQL, una vez autenticado el usuario se le otorgan los permisos correspondientes. La información de *login* se almacena en la base de datos *master*.

Un usuario en genral se identifica mediante tres métodos posibles:

- Como usuario de Windows con su *login* correspondiente.
- Como miembro de un grupo de usuarios Windows.
- Como usuario de SQL Server con el *login* específico del servidor.

Una vez autenticado el usuario tiene los permisos que se le hayan concedido via mediante grant l incluyendo los correspondientes a los roles que tenga asignados.

A nivel de sistema operativo debe notarse que aunque lo susurios no tienen porque interaccionar con el sistema de archivos (salvo a través del servidor) si debemos considerar que el proceso SQL Server tiene que hacerlo por lo cual requiere una cuenta de tres posibles:

- *Local*: para cuando no es necesario trabajar en red.
- *Local de sistema*: es buena para un solo servidor peor falla cuando hay procesamiento distribuido en red.
- *Cuenta de dominio*: es la opción recomendada ya que es una cuenta creada ad-hoc para el servidor a la que se pueden otorgar distintos privilegios.

Vamos a ver ahora las opciones más interesantes desde el punto de vista de T-SQL:

Gestion de cuentas

Para crear un usuario usamos el comando *CREATE LOGIN* cuya sintaxis básica es la siguiente:

```
CREATE LOGIN loginName { WITH <option_list1> | FROM <sources> }

<option_list1> ::=
    PASSWORD = { 'password' | hashed_password HASHED } [ MUST_CHANGE ]
    [ , <option_list2> [ ,... ] ]

<option_list2> ::=
    SID = sid
    | DEFAULT_DATABASE = database
    | DEFAULT_LANGUAGE = language
    | CHECK_EXPIRATION = { ON | OFF}
    | CHECK_POLICY = { ON | OFF}
    | CREDENTIAL = credential_name

<sources> ::=
    WINDOWS
```

loginName: es el nombre.

Password: permite introducir la contraseña con o sin *hash*, solo para cuentas sql.

Must_change: indica la necesidad de cambiar la contraseña la primera vez que la cuenta es usada.

Credential: es el nombre de la credencial a la que se muestra la cuenta.

Check expiration: obliga a que se apliquen las políticas de caducidad de cuentas en el servidor.

Check_policy: obliga a que se apliquen las políticas de caducidad de cuentas Windows.

- **Eliminar una cuenta**

  Con el comando:

```
DROP LOGIN login_name
```

- **Denegar una cuenta de Windows**

  Una nueva posibilidad es denegar a usuarios o grupos de Windows el acceso al servidor SQL incluso aunque no existan como tales en el servidor.

```
DENY CONNECT usuario/grupo
```

- **Modificar cuentas**

  De manera similar a la creación de cuentas podemos modificarlas con *alter login*:

```
ALTER LOGIN login_name
    {
    <status_option>
    | WITH <set_option> [ ,... ]
    | <cryptographic_credential_option>
    }
```

```
<status_option> ::=ENABLE | DISABLE

<set_option> ::=
    PASSWORD ='password' | hashed_password HASHED
    [
      OLD_PASSWORD ='oldpassword'
      | <password_option> [<password_option> ]
    ]
    | DEFAULT_DATABASE =database
    | DEFAULT_LANGUAGE =language
   | NAME = login_name
    | CHECK_POLICY = { ON | OFF }
    | CHECK_EXPIRATION = { ON | OFF }
    | CREDENTIAL =credential_name
    | NO CREDENTIAL

<password_option> ::=
    MUST_CHANGE | UNLOCK
<cryptographic_credentials_option> ::=
         ADD CREDENTIAL credential_name
          | DROP CREDENTIAL credential_name
```

A nivel de base de datos, los usuarios son independientes de las cuentas, así podemos crear y eliminar usarios a partir de cuentas o *logins* para cada base de datos.

- **Crear usuarios**

```
CREATE USER user_name
    [ { FOR | FROM }
      LOGIN login_name
      | WITHOUT LOGIN
    ]
    [ WITH DEFAULT_SCHEMA =schema_name ]
```

Donde *for* o *from* especifica el *login* asociado a ese usuario.

- **Borrar usuarios**

```
DROP USER user_name
```

- **Modificar usuarios**

```
ALTER USER userName
     WITH <set_item> [ ,...n ]

<set_item> ::=
     NAME =newUserName
     | DEFAULT_SCHEMA =schemaName
     | LOGIN =loginName
```

SQL Server utiliza el paradigma de roles para englobar un conjunto de permisos bajo un mismo nombre. Distinguimos roles de sistema y de bases de datos. Del primero existen algunos predefinidos, por ejemplo *Dbcreator* que permite crear, modificar, eliminar y restaurar cualquier base de datos o el rol *Processadmin* para

crear y eliminar procesos del servidor. Aunque sin duda es el *Sysadmin* el usuario que tiene todo el control sobre todos los objetos del sistema.

En el caso de roles de bases de datos existe el *db_owner* con todos los permisos sobre la base de datos o *db_ddladmin* que puede ejecutar comandos *ddl* como *create*, *drop* o *alter*.

Para añadir un usuario a un rol se usa un procedimiento alamcenado llamado *sp_addsrvtolemember*.

```
sp_addsrvrolemember [ @loginame= ] 'login'      , [ @rolename = ] 'role'
```

Para eliminarlo *sp_dropsrvrolemember*.

```
sp_dropsrvrolemember [ @loginame = ] 'login' , [ @rolename = ] 'role'
```

Para crear/eliminar un rol usamos *create role*.

```
CREATE ROLE role_name [ AUTHORIZATION owner_name ]
DROP ROLE role_name
```

Donde *AUTHORIZATION* indica el usuario o rol de la base de datos que va a ser propietario del rol.

Para asignar un miembro al rol:

```
sp_addrolemember [ @rolename = ] 'role' ,
    [ @membername = ] 'security_account'
```

Para eliminar un miembro de un rol:

```
sp_droprolemember [ @rolename = ] 'role' ,
     [ @membername = ] 'security_account'
```

Gestion de permisos y privilegios

En SQL Server los permisos se otorgan sobre los distintos objetos usando los comandos DCL (*GRANT*, *REVOKE* y *DENY*) y procedimientos almacenados predefinidos.

Se distinguen permisos sobre objetos y en forma de sentencia.

Son los *statement permissions* que incluyen *alter table*, *alter database* o *create function* y se otorgan con la siguiente sintaxis:

```
GRANT { ALL | statement [ ,...n ] }
TO security_account [ ,...n ]
```

Donde *security account* puede ser un rol, un usuario de SQL Server o un usuario o grupo de Windows.

Los permisos posibles sobre objetos incluyen *select*, *insert*, *update*, *delete*, *dri* (para claves ajenas) y *execute* para ejecución de procedimientos almacenados.

Para conceder permisos se usa el comando *grant*:

```
GRANT
    { ALL [ PRIVILEGES ] | permission [ ,...n ] }
    {
        [ ( column [ ,...n ] ) ] ON { table | view }
        | ON { table | view } [ ( column [ ,...n ] ) ]
        | ON { stored_procedure | extended_procedure }
        | ON { user_defined_function }
```

```
    }
TO security_account [ ,...n ]
[ WITH GRANT OPTION ]
[ AS { group | role } ]
```

Del mismo modo para revocar permisos usamos:

```
REVOKE { ALL | statement [ ,...n ] }
FROM security_account [ ,...n ]
Para el caso de sentencias T-sql
```

Y

```
REVOKE [ GRANT OPTION FOR ]
    { ALL | permission [ ( column [ ,...n ] ) ] [ ,...n ] }
    {
        [ ( column [ ,...n ] ) ] ON { table | view }
        | ON { table | view } [ ( column [ ,...n ] ) ]
        | ON { stored_procedure | extended_procedure }
        | ON { user_defined_function }
    }
{ TO | FROM }
    security_account [ ,...n ]
[ CASCADE ]
[ AS { group | role } ]
```

Encriptación

SQL server incropora la funcionalidad de encriptación de datos usando cuatro métodos:

- ***Frase de paso***

  Es como usar una contraseña pero con más restricciones.

  En el ejemplo se usa la función *EncryptByPassPhrase* con la frase de paso seguida de los datos bancarios a encriptar para lo cual el campo número de cuenta debe ser de tipo *varbinary*:

  Para desencriptar la información puede usarse la función *DecryptByPassPhrase*.

```
CREATE TABLE CCard (
CCardID INT IDENTITY PRIMARY KEY NOT NULL,
 CustomerID INT NOT NULL,
 CreditCardNumber VARBINARY(128),
 Expires CHAR(4)
 );
 INSERT CCard(CustomerID, CreditCardNumber, Expires)
 VALUES(1,EncryptByPassPhrase('Passphrase', '12345678901234567890'),
 '0808');
```

- **Clave simétrica**

  Las frases de paso son texto claro y poco seguras así que es mejor usar encriptación simétrica con el comando *CREATE SIMETRIC KEY*, como en el ejemplo:

```
CREATE SYMMETRIC KEY CCardKey
WITH ALGORITHM = TRIPLE_DES
ENCRYPTION BY PASSWORD = 'P@s$wOrD';
```

Donde el algoritmo usado es típicamente *triple_des* o *aes_250*, aunque hay algunos más.

Para usar la clave debe ser abierta en primer lugar con el comando *OPEN SIMETRIC KEY ccardkey DECRIPTION BY PASSWORD='P@s$wOrD'.*

Después hacemos las operaciones normales como inserción de datos:

```
INSERT CCard(CustomerID, CreditCardNumber, Expires)
VALUES(7,EncryptByKey(Key_GUID('CCardKey'),'11112222333344445555'),
'0808');
```

Después para desencriptar usamos *decriptbykey* y finalmente la cerramos con:

```
Close simatric key ccardkey
```

Clave asimétrica

Funcionan igual que las anteriores pero con el comando *create asimetric key*, como en el siguiente ejemplo:

```
CREATE ASYMMETRIC KEY AsyKey
WITH ALGORITHM = RSA_512
ENCRYPTION BY PASSWORD = 'P@s$w0rD';
```

Donde *rsa_512* es el algoritnmo de clave pública típicamente usada:
También puede usarse claves a partir de un fichero:

```
CREATE ASYMMETRIC KEY AsyKey
FROM FILE = ' C:\SQLServerBible\AsyKey.key'
ENCRYPTION BY PASSWORD = 'P@s$w0rD';
```

En el caso de claves asimétricas no hay necesidad de abrirlas.

Encriptación transparente de datos

Existe la posibilidad de encriptar todas la base de datos en lo que se conoce como encriptación de datos tranparente que permite, de una forma transparente al usuario que los datos estén permanentes, entre encriptadas.

Para ello se requiere una *master key* con los siguientes comandos:

```
USE master;
go
CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'Pa@sW0rD';
go --
CREATE CERTIFICATE SQLBibleCert WITH SUBJECT = 'SQLBibleCert'
```

Ahora se crean las claves de la base de datos:

```
USE ebanca
go --
CREATE DATABASE ENCRYPTION KEY
WITH ALGORITHM = AES_128
ENCRYPTION BY SERVER CERTIFICATE sierracert
```

Ahora solo es cuestión de usar el comando alter para encriptar la base completa:

```
ALTER DATABASE AdventureWorks2008
SET ENCRYPTION ON
```

AUTOMATIZACIÓN TAREAS T-SQL

Sintaxis básica de T-SQL para rutinas, eventos, *jobs* y *triggers*.

Transact SQL es el lenguaje de programación que proporciona SQL Server para ampliar SQL con los elementos caracteristicos de los lenguajes de programación: variables, sentencias de control de flujo y bucles.

Con Transact SQL vamos a poder programar procedimientos almacenados, funciones, *triggers* y *scripts*.

En general, un programa en T-SQL está compuesto por uno o varios bloques. Un bloque delimita el alcance de las variables y sentencias del *script*. Dentro de un mismo *script* se diferencian los diferentes blouqes a través de la instrucción *GO*. Ya que T-SQL no permite la ejecución de ciertas sentencias dentro del mismo bloque.

El siguiente ejemplo nos muestra un programa con dos bloques en que obtenemos el resultado de una consulta y la fecha actual.

```
-- Este es el primer lote del script
SELECT * FROM COMENTARIOS
GO -- GO es el separador de lotes
-- Este es el segundo lote del script
SELECT getdate()  -- getdate() función integrada que devuelve
                  -- la fecha
```

Variables en Transact SQL

Una variable es un valor identificado por un nombre (identificador) sobre el que podemos realizar modificaciones. En Transact SQL los identificadores de variables deben comenzar por el caracter @, es decir, el nombre de una variable debe comenzar por @. Para declarar variables en Transact SQL debemos utilizar la palabra clave declare, seguido del identificador y tipo de datos de la variable. Podemos asignar valores a una variable de varias formas:

- A través de la instrucción *SET*.
- Utilizando una sentencia *SELECT*.
- Realizando un *FETCH* de un cursor.

El siguiente ejemplo muestra como asignar una variable utilizando la instrucción *SET*.

```
DECLARE @nombre VARCHAR(100)
-- La consulta debe devolver un único registro
SET @nombre =  (SELECT nombre
           FROM CLIENTES
           WHERE ID = 1)
PRINT @nombre
```

El siguiente ejemplo muestra como asignar variables utilizando una sentencia *SELECT*.

```
WHERE ID = 1

PRINT @nombre
PRINT @apellido1
PRINT @apellido2
```

Un punto a tener en cuenta cuando asignamos variables de este modo, es que si la consulta *SELECT* devuelve más de un registro, las variables quedarán asignadas con los valores de la última fila devuelta.

Por úlitmo veamos como asignar variables a través de un cursor.

```
DECLARE @nombre VARCHAR(100),
    @apellido1 VARCHAR(100),
    @apellido2 VARCHAR(100)

DECLARE CDATOS CURSOR
FOR
SELECT nombre , Apellido1, Apellido2
FROM CLIENTES

OPEN CDATOS
FETCH CDATOS INTO @nombre, @apellido1, @apellido2

WHILE (@@FETCH_STATUS = 0)
BEGIN
    PRINT @nombre
    PRINT @apellido1
    PRINT @apellido2
    FETCH CDATOS INTO @nombre, @apellido1, @apellido2
END

CLOSE CDATOS
DEALLOCATE CDATOS
```

Estructuras de control en Transact SQL

Estructura condicional IF

```
IF (<expresion>)
  BEGIN
    ...
  END
ELSE IF (<expresion>)
  BEGIN
    ...
  END
ELSE
  BEGIN
    ...
  END
```

En el siguiente ejemplo se inserta un nuevo registro en la tabla noticias en caso de que no exista:

```
DECLARE @id_noticia int,
    @titulo varchar(255)
set @id_noticia = 5
set @titulo = 'Introduccion a T-SQL '
IF EXISTS(SELECT * FROM noticias
          WHERE id_noticia = @id_noticia)
  BEGIN
    UPDATE noticias
    SET titulo = @titulo
    WHERE id_noticia = @id_noticia
  END

ELSE
  BEGIN
    INSERT INTO noticias
    (id_noticia, titulo) VALUES
    (@id_noticia, @titulo)
  END
```

Estructura condicional CASE

La sintaxis general es:

```
CASE <expresion>
    WHEN <valor_expresion> THEN <valor_devuelto>
    WHEN <valor_expresion> THEN <valor_devuelto>
    ELSE <valor_devuelto>  -- Valor por defecto
END
```

Ejemplo de *CASE*:

```
DECLARE @Web varchar(100),
        @diminutivo varchar(3)
SET @diminutivo = 'DJK'
SET @Web = (CASE @diminutivo
                WHEN 'DJK' THEN 'www.devjoker.com'
                WHEN 'ALM' THEN 'www.aleamedia.com'
                ELSE 'www.devjoker.com'
            END)
PRINT @Web
```

Bucle WHILE

Es el único tipo de bucle del que dispone Transact SQL.

```
WHILE <expresion>
BEGIN
   ...
END
```

Un ejemplo del bucle *WHILE*.

```
DECLARE @coRecibo int
WHILE  EXISTS (SELECT * FROM RECIBOS WHERE PENDIENTE = 'S')-- La subconsulta se ejecuta
una vez por cada iteracion del bucle!
BEGIN
SET @coRecibo = (SELECT TOP 1 CO_RECIBO
FROM RECIBOS WHERE PENDIENTE = 'S')
UPDATE RECIBOS SET PENDIENTE = 'N'WHERE CO_RECIBO = @coRecibo
END
```

Los bucles admiten también las instrucciones *break* y *continue* para salir del mismo o proseguir con más iteraciones.

Cursores en Transact SQL

Para trabajar con cursores debemos seguir los siguientes pasos:

- Declarar el cursor, utilizando *DECLARE*.
- Abrir el cursor, utilizando *OPEN*.
- Leer los datos del cursor, utilizando *FETCH...INTO*.
- Cerrar el cursor, utilizando *CLOSE*.
- Liberar el cursor, utilizando *DEALLOCATE*.

La sintaxis general es la siguiente:

```
-- Declaración del cursor
DECLARE <nombre_cursor> CURSOR
FOR
<sentencia_sql>

-- apertura del cursor
OPEN <nombre_cursor>

-- Lectura de la primera fila del cursor
FETCH <nombre_cursor> INTO <lista_variables>

WHILE (@@FETCH_STATUS = 0)
BEGIN
-- Lectura de la siguiente fila de un cursor
FETCH <nombre_cursor> INTO <lista_variables>
...
END -- Fin del bucle WHILE

-- Cierra el cursor
CLOSE <nombre_cursor>
-- Libera los recursos del cursor
DEALLOCATE <nombre_cursor>
```

El siguente ejemplo muestra el uso de un cursor:

```
-- Declaracion de variables para el cursor
DECLARE @Id int,
```

```
    @Nombre varchar(255),
    @Apellido1 varchar(255),
    @Apellido2 varchar(255),
    @NifCif varchar(20),
    @FxNacimiento datetime

-- Declaración del cursor
DECLARE cClientes CURSOR FOR
SELECT  Id, Nombre, Apellido1,
    Apellido2, NifCif, FxNacimiento
FROM CLIENTES
-- Apertura del cursor
OPEN cClientes
-- Lectura de la primera fila del cursor
FETCH cClientes INTO    @id, @Nombre, @Apellido1,
                  @Apellido2, @NifCif, @FxNacimiento

WHILE (@@FETCH_STATUS = 0 )
BEGIN
PRINT @Nombre + ' ' + @Apellido1 + ' ' + @Apellido2
-- Lectura de la siguiente fila del cursor
FETCH cClientes INTO    @id, @Nombre, @Apellido1,
                  @Apellido2, @NifCif, @FxNacimiento
END

-- Cierre del cursor
CLOSE cClientes
-- Liberar los recursos
DEALLOCATE cClientes
```

Cuando trabajamos con cursores, la funcion *@@FETCH_STATUS* nos indica el estado de la última instrucción *TCH* emitida, los valores posibles son 0 en caso de éxito, -1 en caso de error o -2, en caso de no faltar la fila uperada.

En la apertura del cursor, podemos especificar los siguientes parámetros:

```
DECLARE <nombre_cursor> CURSOR
[ LOCAL | GLOBAL ]
[ FORWARD_ONLY | SCROLL ]
[ STATIC | KEYSET | DYNAMIC | FAST_FORWARD ]
[ READ_ONLY | SCROLL_LOCKS | OPTIMISTIC ]
[ TYPE_WARNING ]
FOR <sentencia_sql>
```

El primer conjunto de parámetros que podemos especificar es [*LOCAL*|*GLOBAL*]. A continuación mostramos el nificado de cada una de estas opciones.

- *LOCAL/GLOBAL*: especifica que el ámbito del cursor es local global de forma que pueda hacerse referencia al nombre del cursor en cualquier procedimiento almacenado o proceso por lotes que se ejecute en la conexión.
- *FORWARD_ONLY*: especifica que el cursor solo se puede desplazar de la primera a la última fila. *FETCH NEXT* es la única opción de recuperación admitida.

- *SCROLL*: especifica que están disponibles todas las opciones de recuperación (*FIRST*, *LAST*, *PRIOR*, *NEXT*, *RELATIVE*, *ABSOLUTE*).
- *STATIC*: define un cursor que hace una copia temporal de los datos que va a utilizar. Todas las solicitudes que se realizan al cursor se responden desde esta tabla temporal de *tempdb*; por tanto, las modificaciones realizadas en las tablas base no se reflejan en los datos devueltos por las operaciones de recuperación realizadas en el cursor y, además, este cursor no admite modificaciones.
- *KEYSET*: especifica que la pertenencia y el orden de las filas del cursor se fijan cuando se abre el cursor. El conjunto de claves que identifica las filas de forma única está integrado en la tabla denominada *keyset* de *tempdb*.
- *DYNAMIC*: define un cursor que, al desplazarse por él, refleja en su conjunto de resultados todos los cambios realizados en los datos de las filas. Los valores de los datos, el orden y la pertenencia de las filas pueden cambiar en cada operación de recuperación. La opción de recuperación *ABSOLUTE* no se puede utilizar en los cursores dinámicos.
- *FAST_FORWARD*: especifica un cursor *FORWARD_ONLY*, *READ_ONLY* con las optimizaciones de rendimiento habilitadas. No se puede especificar FAST_*FORWARD* si se especifica también *SCROLL* o *FOR_UPDATE*.
- *READ_ONLY*: evita que se efectúen actualizaciones a través de este cursor. No es posible hacer referencia al cursor en una cláusula *WHERE CURRENT OF* de una instrucción *UPDATE* o *DELETE*. Esta opción reemplaza la capacidad de actualizar el cursor.
- *SCROLL_LOCKS*: especifica que se garantiza que las actualizaciones o eliminaciones posicionadas realizadas a través del cursor serán correctas. Microsoft SQL Server bloquea las filas cuando se leen en el cursor para garantizar que estarán disponibles para futuras modificaciones. No es posible especificar *SCROLL_LOCKS* si se especifica también *FAST_FORWARD* o *STATIC*.
- *OPTIMISTIC*: especifica que las actualizaciones o eliminaciones posicionadas realizadas a través del cursor no se realizarán correctamente si la fila se ha actualizado después de ser leída en el cursor. SQL Server no bloquea las filas al leerlas en el cursor. En su lugar, utiliza comparaciones de valores de columna *timestamp* o un valor de suma de comprobación si la tabla no tiene columnas *timestamp*, para determinar si la fila se ha modificado después de leerla en el cursor. Si la fila se ha modificado, el intento de actualización o eliminación posicionada genera un error. No es posible especificar *OPTIMISTIC* si se especifica también *FAST_FORWARD*.
- *TYPE_WARNING*: especifica que se envía un mensaje de advertencia al cliente si el cursor se convierte implícitamente del tipo solicitado a otro.

Para actualizar los datos de un cursor debemos especificar *FOR UPDATE* despues de la sentencia *SELECT* en la declaración del cursor, y *WHERE CURRENT OF* <nombre_cursor> en la sentencia *UPDATE* tal y como muestra el siguiente ejemplo:

```
-- Declaracion de variables para el cursor
DECLARE @Id int,
    @Nombre varchar(255),
    @Apellido1 varchar(255),
    @Apellido2 varchar(255),
    @NifCif varchar(20),
    @FxNacimiento datetime
-- Declaración del cursor
DECLARE cClientes CURSOR FOR
SELECT  Id, Nombre, Apellido1,
    Apellido2, NifCif, FxNacimiento
FROM CLIENTES
FOR UPDATE
```

```
-- Apertura del cursor
OPEN cClientes
-- Lectura de la primera fila del cursor
FETCH cClientes
INTO @id, @Nombre, @Apellido1, @Apellido2, @NifCif, @FxNacimiento

WHILE (@@FETCH_STATUS = 0 )
BEGIN
UPDATE Clientes
SET APELLIDO2 = isnull(@Apellido2,'') + ' - Modificado'
WHERE CURRENT OF cClientes
-- Lectura de la siguiente fila del cursor
FETCH cClientes
INTO @id, @Nombre, @Apellido1, @Apellido2,
     @NifCif, @FxNacimiento
END
-- Cierre del cursor
CLOSE cClientes
-- Liberar los recursos
DEALLOCATE cClientes
```

Control de errores en *Transact* SQL

SQL Server proporciona el control de errores a través de las instrucciones *TRY* y *CATCH*.

La sintaxis de *TRY CATCH* es la siguiente:

```
BEGIN TRY
    ...
END TRY
BEGIN CATCH
    ...
END CATCH
```

El siguiente ejemplo ilustra su uso:

```
    PRINT 'No hay error'
END TRY
BEGIN CATCH
    PRINT 'Se ha producido un error'
END CATCH
```

Funciones especiales de Error

Las funciones especiales de error, están disponibles únicamente en el bloque *CATCH* para la obtención de información detallada del error.

Son:

- *ERROR_NUMBER*(), devuelve el número de error.
- *ERROR_SEVERITY*(), devuelve la severidad del error.
- *ERROR_STATE*(), devuelve el estado del error.
- *ERROR_PROCEDURE*(), devuelve el nombre del procedimiento almacenado que ha provocado el error.
- *ERROR_LINE*(), devuelve el número de línea en el que se ha producido el error.
- *ERROR_MESSAGE*(), devuelve el mensaje de error.

Son extremadamente útiles para realizar una auditoría de errores.

Lógicamente, podemos utilizar estas funciones para almacenar esta información en una tabla de la base de datos y registrar todos los errores que se produzcan.

Generar un error con RAISERROR

En ocasiones es necesario provocar voluntariamente un error, por ejemplo nos puede interesas que se genere un error cuando los datos incumplen una regla de negocio.

Podemos provocar un error en tiempo de ejecución a través de la función *RAISERROR*.

```
DECLARE @tipo int,
        @clasificacion int

 SET @tipo = 1
 SET @clasificacion = 3
 IF (@tipo = 1 AND @clasificacion = 3)
 BEGIN
    RAISERROR ('El tipo no puede valer uno y la clasificacion 3',
        16, -- Severidad
        1   -- Estado
        )
END
```

La función *RAISERROR* recibe tres parámetros, el mensaje del error (o código de error predefinido), la severidad y el estado.

La severidad indica el grado de criticidad del error. Admite valores de 0 al 25, pero solo podemos asignar valores del 0 al 18. Los errores el 20 al 25 son considerados fatales por el sistema, y cerraran la conexion que ejecuta el comando *RAISERROR*. Para asignar valores del 19 al 25 necesitarán ser miembros de la función de SQL Server *sysadmin*.

El estado es un valor para permitir que el programador identifique el mismo error desde diferentes partes del código. Admite valores entre 1 y 127.

Procedimientos almacenados

Un procedimiento tiene un nombre, un conjunto de parámetros (opcional) y un bloque de código.

En Transact SQL los procedimientos almacenados pueden devolver valores (numerico entero) o conjuntos de resultados.

Para crear un procedimiento almacenado debemos emplear la sentencia *CREATE PROCEDURE*.

```
CREATE PROCEDURE <nombre_procedure> [@param1 <tipo>, ...]
AS
-- Sentencias del procedure
```

Para modificar un procedimiento almacenado debemos emplear la sentencia *ALTER PROCEDURE*.

```
ALTER PROCEDURE <nombre_procedure> [@param1 <tipo>, ...]
AS
-- Sentencias del procedure
```

El siguiente ejemplo muestra un procedimiento almacenado, denominado *spu_addCliente* que inserta un registro en la tabla "CLIENTES".

```
CREATE PROCEDURE spu_addCliente @nombre varchar(100),
                        @apellido1 varchar(100),
                        @apellido2 varchar(100),
                        @nifCif varchar(20),
                        @fxNaciento datetime
AS
INSERT INTO CLIENTES
(nombre, apellido1, apellido2, nifcif, fxnacimiento) VALUES
(@nombre, @apellido1, @apellido2, @nifCif, @fxNaciento)
```

Para la ejecutar un procedimiento almacenado debemos utilizar la sentencia *EXEC*. Cuando la ejecución del procedimiento almacenado es la primera instrucción del lote, podemos omitir el uso de *EXEC*.

El siguiente ejemplo muestra la ejecución del procedimiento almacenado anterior.

```
DECLARE @fecha_nacimiento datetime
set @fecha_nacimiento = convert(datetime, '13/05/1975', 103)
EXEC spu_addCliente 'Pedro', 'Herrarte', 'Sanchez',
               '00000002323', @fecha_nacimiento
```

Si queremos que los parámetros de un procedimiento almacenado sean de entrada/salida debemos especificarlo a través de la palabra clave OUTPUT, tanto en la definición del procedure como en la ejecución.

El siguiente ejemplo muestra la definición de un procedimiento con parámetros de salida.

```
CREATE PROCEDURE spu_ObtenerSaldoCuenta @numCuenta varchar(20),
                                @saldo decimal(10,2) output
AS
BEGIN
SELECT @saldo = SALDO
FROM CUENTAS
WHERE NUMCUENTA = @numCuenta
END
```

Un procedimiento almacenado puede devolver valores numericos enteros a través de la instrucción *RETURN*. Normalmente debemos utilizar los valores de retorno para determinar si la ejecución del procedimiento ha sido correcta o no. Si queremos obtener valores se recomienda utilizar parámetros de salida o funciones escalares (se verán mas adelante en este tutorial).

En *Transact* SQL es que pueden devolver uno o varios conjuntos de resultados.

El siguiente ejemplo muestra un procedimiento almacenado que devuelve un conjunto de resultados.

```
CREATE PROCEDURE spu_MovimientosCuenta @numCuenta varchar(20)
AS
BEGIN
SELECT  @numCuenta,
    SALDO_ANTERIOR,
    SALDO_POSTERIOR,
    IMPORTE,
    FXMOVIMIENTO
FROM MOVIMIENTOS
INNER JOIN CUENTAS ON MOVIMIENTOS.IDCUENTA = CUENTAS.IDCUENTA
WHERE NUMCUENTA = @numCuenta
ORDER BY FXMOVIMIENTO DESC
END
```

Funciones almacenadas

SQL Server proporciona al usuario la posibilidad de definir sus propias funciones, conocidad como UDF (user defined functions). Existen tres tipos de funciones. Éstas son: escalares, en línea y en línea de múltiples sentencias.

Funciones escalares

Las funciones escalares devuelven un único valor de cualquier tipo de los datos tal y como *int*, *money*, *varchar*, real, etc.

La sintaxis para una función escalar es la siguiente:

```
CREATE FUNCTION <Scalar_Function_Name, sysname, FunctionName>
(
-- Lista de parámetros
<@Param1, sysname, @p1> <Data_Type_For_Param1, , int>, ...
)
-- Tipo de datos que devuelve la función.
RETURNS <Function_Data_Type, ,int>
AS
BEGIN
...
END
```

El siguiente ejemplo muestra cómo utilizar una función escalar en un *script* Transact SQL.

```
DECLARE @NumCuenta VARCHAR(20),
  @Resultado DECIMAL(10,2)

SET @NumCuenta = '200700000001'
```

```
SET @Resultado = dbo.fn_MultiplicaSaldo(@NumCuenta, 30.5)

PRINT @Resultado
```

Las funciones escalares son muy similares a con parámetros de salida, pero éstas pueden ser utilizadas en consultas de selección y en la cláusula *where* de las mismas.

Las funciones no pueden ejecutar sentencias *INSERT* o *UPDATE*.

- **Funciones en línea**

Las funciones en línea son las funciones que devuelven un conjunto de resultados correspondientes a la ejecución de una sentencia *SELECT*.

La sintaxis para una función de tabla en línea es la siguiente:

```
CREATE FUNCTION <Inline_Function_Name, sysname, FunctionName>
(
-- Lista de parámetros
<@param1, sysname, @p1> <Data_Type_For_Param1, , int>,...
)
RETURNS TABLE
AS
RETURN
(

-- Sentencia Transact SQL

)
```

El siguiente ejemplo muestra cómo crear una función en línea.

```
CREATE FUNCTION fn_MovimientosCuenta
(
@NumCuenta VARCHAR(20)
)
RETURNS TABLE
AS
RETURN
(
SELECT MOVIMIENTOS.*
FROM MOVIMIENTOS
INNER JOIN CUENTAS ON MOVIMIENTOS.IDCUENTA = CUENTAS.IDCUENTA
WHERE CUENTAS.NUMCUENTA = @NumCuenta
)
```

No podemos utilizar la cláusula *ORDER BY* en la sentencia de una función el línea.

Las funciones en línea pueden utilizarse dentro de *joins* o *querys* como si fueran una tabla normal.

- **Funciones en línea de múltiples sentencias**

Las funciones en línea de multiples sentencias son similares a las funciones en línea excepto que el conjunto de resultados que devuelven puede estar compuesto por la ejecución de varias consultas *SELECT*.

Este tipo de función se usa en situaciones donde se requiere una mayor lógica de proceso.

La sintaxis para unas funciones de tabla de multisentencias es la siguiente:

```
CREATE FUNCTION <Table_Function_Name, sysname, FunctionName>
(
-- Lista de parámetros
<@param1, sysname, @p1> <data_type_for_param1, , int>, ...
)
RETURNS
-- variable de tipo tabla y su estructura
<@Table_Variable_Name, sysname, @Table_Var> TABLE
(
<Column_1, sysname, c1> <Data_Type_For_Column1, , int>,
<Column_2, sysname, c2> <Data_Type_For_Column2, , int>
)
AS
BEGIN
  -- Sentencias que cargan de datos la tabla declarada
  RETURN
END
```

El siguiente ejemplo muestra el uso de una función de tabla de *multisentencias*.

```
/* Esta funcion busca la tres cuentas con mayor saldo
   * y obtiene los tres últimos movimientos de cada una
   * de estas cuentas
   */

 CREATE FUNCTION fn_CuentaMovimietos()
 RETURNS @datos TABLE
 ( -- Estructura de la tabla que devuelve la funcion.
NumCuenta varchar(20),
Saldo decimal(10,2),
Saldo_anterior decimal(10,2),
Saldo_posterior decimal(10,2),
Importe_Movimiento decimal(10,2),
FxMovimiento datetime
 )
 AS
 BEGIN
-- Variables necesarias para la lógica de la funcion.
DECLARE @idcuenta int,
  @numcuenta varchar(20),
  @saldo decimal(10,2)

-- Cursor con las 3 cuentas de mayor saldo
DECLARE CDATOS CURSOR FOR
SELECT TOP 3 IDCUENTA, NUMCUENTA, SALDO
FROM CUENTAS
ORDER BY SALDO DESC
```

```
OPEN CDATOS
FETCH CDATOS INTO @idcuenta, @numcuenta, @saldo

-- Recorremos el cursor
WHILE (@@FETCH_STATUS = 0)
BEGIN
-- Insertamos la cuenta en la variable de salida
INSERT INTO @datos
(NumCuenta, Saldo)
VALUES
(@numcuenta, @saldo)
-- Insertamos los tres últimos movimientos de la cuenta
INSERT INTO @datos
(Saldo_anterior, Saldo_posterior,
 Importe_Movimiento, FxMovimiento )
SELECT TOP 3
 SALDO_ANTERIOR, SALDO_POSTERIOR,
 IMPORTE, FXMOVIMIENTO
FROM MOVIMIENTOS
WHERE IDCUENTA = @idcuenta
ORDER BY FXMOVIMIENTO DESC
-- Vamos a la siguiente cuenta
FETCH CDATOS INTO @idcuenta, @numcuenta, @saldo
END

CLOSE CDATOS;
DEALLOCATE CDATOS;

RETURN
 END
```

Triggers en Transact SQL

SQL Server proporciona dos tipos de *triggers*:

- *Trigger DML*: se ejecutan cuando un usuario intenta modificar datos mediante un evento de lenguaje de manipulación de datos (DML). Los eventos DML son instrucciones *INSERT*, *UPDATE* o *DELETE* de una tabla o vista.
- *Trigger DDL*: se ejecutan en respuesta a una variedad de eventos de lenguaje de definición de datos (DDL). Estos eventos corresponden principalmente a instrucciones *CREATE*, *ALTER* y *DROP* de *Transact*-SQL, y a determinados procedimientos almacenados del sistema que ejecutan operaciones de tipo DDL.

Trigger DML

Los *trigger* DML se ejecutan cuando un usuario intenta modificar datos mediante un evento de lenguaje de manipulación de datos (DML). Los eventos DML son instrucciones *INSERT*, *UPDATE* o *DELETE* de una tabla o vista.

La sintaxis general de un *trigger* es la siguiente:

```
CREATE TRIGGER <Trigger_Name, sysname, Trigger_Name>
```

```
ON <Table_Name, sysname, Table_Name>
AFTER <Data_Modification_Statements, , INSERT,DELETE,UPDATE>
AS
BEGIN
-- SET NOCOUNT ON added to prevent extra result sets from
-- interfering with SELECT statements.
SET NOCOUNT ON;
-- Insert statements for trigger here
END
```

Antes de ver un ejemplo es necesario conocer las tablas *inserted* y *deleted*.

Las instrucciones de *triggers* DML utilizan dos tablas especiales denominadas *inserted* y *deleted*. SQL Server crea y administra automáticamente ambas tablas. La estructura de las tablas *inserted* y *deleted* es la misma que tiene la tabla que ha desencadenado la ejecución del *trigger*.

La primera tabla (*inserted*) solo está disponible en las operaciones *INSERT* y *UPDATE* y en ella están los valores resultantes despues de la inserción o actualización. Es decir, los datos insertados. *Inserted* estará vacia en una operación *DELETE*.

En la segunda (*deleted*), disponible en las operaciones *UPDATE* y *DELETE*, están los valores anteriores a la ejecución de la actualización o borrado. Es decir, los datos que serán borrados. *Deleted* estará vacia en una operacion *INSERT*.

¿No existe una tabla *UPDATED*? No, hacer una actualización es lo mismo que borrar (*deleted*) e insertar los nuevos (*inserted*). La sentencia *UPDATE* es la única en la que *inserted* y *deleted* tienen datos simultaneamente.

No se puede modificar directamente los datos de estas tablas.

El siguiente ejemplo graba un historico de saldos cada vez que se modifica un saldo de la tabla cuentas.

```
CREATE TRIGGER TR_CUENTAS
ON CUENTAS
AFTER UPDATE
AS
BEGIN
-- SET NOCOUNT ON impide que se generen mensajes de texto
-- con cada instrucción
SET NOCOUNT ON;
INSERT INTO HCO_SALDOS
(IDCUENTA, SALDO, FXSALDO)
SELECT IDCUENTA, SALDO, getdate()
FROM INSERTED
END
```

Una consideración a tener en cuenta es que el *trigger* se ejecutará aunque la instruccion DML (*UPDATE*, *INSERT* o *DELETE*) no haya afectado a ninguna fila. En este caso *inserted* y *deleted* devolverán un conjunto de datos vacío.

Podemos activar y desactivar *triggers* a través de las siguientes instrucciones.

```
-- Desactiva el trigger TR_CUENTAS
DISABLE TRIGGER TR_CUENTAS ON CUENTAS
GO
```

```
-- activa el trigger TR_CUENTAS
ENABLE TRIGGER TR_CUENTAS ON CUENTAS
GO
-- Desactiva todos los trigger de la tabla CUENTAS
ALTER TABLE CUENTAS DISABLE TRIGGER ALL
GO
-- Activa todos los trigger de la tabla CUENTAS
ALTER TABLE CUENTAS ENABLE TRIGGER ALL
```

Trigger DDL

Los *trigger* DDL se ejecutan en respuesta a una variedad de eventos de lenguaje de definición de datos (DDL). Estos eventos corresponden principalmente a instrucciones *CREATE*, *ALTER* y *DROP* de T-SQL, y a determinados procedimientos almacenados del sistema que ejecutan operaciones de tipo DDL.

La sintaxis general de un *trigger* es la siguiente:

```
CREATE TRIGGER <trigger_name, sysname, table_alter_drop_safety>
ON DATABASE
FOR <data_definition_statements, , DROP_TABLE, ALTER_TABLE>
AS
BEGIN
  ...
END
```

La siguiente instrucción impide que se ejecuten sentencias DROP TABLE y ALTER TABLE en la base de datos.

```
CREATE TRIGGER TR_SEGURIDAD
ON DATABASE FOR DROP_TABLE, ALTER_TABLE
```

ACTIVIDADES F.1

- Descarga e instala *SQL Server With Advanced Services* en tu equipo.
- Comenta la estructura de directorios de tu instalación de SQL Server y la función de cada uno.
- Accede a la herramienta de configuración del servidor e indica cada servicio y su función dentro de SQL *Server*.
- Comprueba como reiniciar el servidor actual desde el icono de SQL *Server* Services.
- Investiga cómo habilitar el protocolo TCP/IP para conexiones remotas e indica el puerto que usa por defecto.
- Usa SQL *Server Management Studio* y el comando *sqlcmd* (usa *sqlcmd –h* o el manual *online* de *microsoft*) para cargar el *script* de bases de datos que incluye el libro en tu servidor.
- Investiga y comenta brevemente el objeto de la herramienta *powershell* de SQL *Server* (es accesible desde *Management Studio*).
- Crea una función que devuelva el *id* del máximo encestador de un determinado equipo pasado como parámetro.
- Codifica un procedimiento almacenado que devuelva los datos del mayor encestador de un determinado equipo pasado como parámetro.
- Crea un *trigger* DML para que cuando se modifiquen los puntos de un jugador se actualice el campo correspondiente en la tabla equipo.
- Crea un *trigger* DDL para impedir el borrado de tablas de la base liga mostrando un error al usuario.
- Usand *Management Studio* crea un rol con permisos de lectura sobre todas las bases del servidor.
- Crea un nuevo usuario invitado con el rol anterior y permisos de escritura sobre liga.
- Programa una copia de seguridad semanal usando *sql agent.*

Material adicional

El material adicional de este libro puede descargarlo en nuestro portal Web: *http://www.ra-ma.es*.

Debe dirigirse a la ficha correspondiente a esta obra, dentro de la ficha encontrará el enlace para poder realizar la descarga. Dicha descarga consiste en un fichero ZIP con una contraseña de este tipo: XXX-XX-XXXX-XXX-X la cual se corresponde con el ISBN de este libro.

Podrá localizar el número de ISBN en la página 2 (página de créditos). Para su correcta descompresión deberá introducir los dígitos y los guiones.

Cuando descomprima el fichero obtendrá los archivos que complementan al libro para que pueda continuar con su aprendizaje.

INFORMACIÓN ADICIONAL Y GARANTÍA

- RA-MA EDITORIAL garantiza que estos contenidos han sido sometidos a un riguroso control de calidad.
- Los archivos están libres de virus, para comprobarlo se han utilizado las últimas versiones de los antivirus líderes en el mercado.
- RA-MA EDITORIAL no se hace responsable de cualquier pérdida, daño o costes provocados por el uso incorrecto del contenido descargable.
- Este material es gratuito y se distribuye como contenido complementario al libro que ha adquirido, por lo que queda terminantemente prohibida su venta o distribución.

ny.ini, 26, 30, 32, 37, 59
MyISAM, 27, 44, 47, 133

N

NEW, 115
NEW. OLD, 115

O

OLD, 115
Optimización, 139
OUT, 97

P

Parámetro, 97
Parseo, 139
Perfil, 174
Permiso, 62
Privilegio, 62, 65
Procedimiento, 91
procs_priv, 65
Puerto, 28

R

RAID, 28
Registro binario, 54
Relacional, 10, 17
Replicación, 186, 187
Restricción, 13, 14, 16, 50
utinas, 90

S

Servicio, 26, 27, 29, 30, 31, 32, 38
Sesión, 30, 31, 46
SET, 96
SGBD, 10, 12, 13, 14, 15, 16, 17, 18, 19, 20, 22, 47
Show, 43
SQL, 14, 15, 17, 22, 48, 55, 56, 57
SSH, 78. V
SSL, 78.

T

tables_priv, 64
Transacción, 12
trigger, 114
Tupla, 17

U

User, 62
UTF8, 29

V

Variable, 37, 41, 43, 44, 45, 46, 59, 87
Vista, 119

W

While, 102

X

X509, 78

Índice Alfabético

A

AFTER, 114
ANALYZE, 136, 141
Árbol, 132
Árboles B, 132

B

BEFORE, 114, 115
Begin, 92
btree, 134
B-tree, 132

C

Caché, 139
Caracteres, 29, 50
Catálogo, 12
Cluster, 197
columns_priv, 64
Comando, 90
Condicional, 99
Configuración, 26
Consultas, 14
Cuenta, 62, 71
Cursores, 103

D

db, 64
DCL, 14
DDBMS, 185
DDL, 14
DECLARE, 96
Diccionario de datos, 14, 47
Diseño lógico, 16
Disparador, 14, 114
Distribución, 186
DML, 14
Dominios, 14

E

Eficiencia, 157
Error, 53, 111 170
Evento, 207
explain, 141

F

Fragmentacion, 186
full-text, 133
Función, 91

G

GLOBAL, 45, 46
GRANT, 71

H

Handler, 106
Heap, 134
Herramienta, 90
Host, 62

I

Índice, 130
Índice cluster, 134
INFORMATION_SCHEMA, 48
InnoDB, 27, 28, 31, 47, 50, 55, 134
INOUT, 97
Instrucciones, 103

L

Lenta, 156
Log, 31, 38, 39, 44, 53, 54, 55, 56, 57
Loops, 101

M

Monitorización, 193
Monitorizar, 169