Recursosinformáticos

SQL

Los fundamentos del lenguaje

Eric GODOC

eni ediciones

SQL

Los fundamentos del lenguaje

Este libro sobre los fudamentos del lenguaje SQL se dirige a **desarrolladores** e**informáticos principiantes** que deban trabajar con un Sistema Gestor de Bases de Datos Relacionales (SGBDR) para almacenar y manipular datos. Su objetivo es describir los **principales comandos más utilizados del lenguaje SQL** (independientemente de las variaciones realizadas por los editores de los diferentes SGBDR) para permitir al lector hacerse cargo rápidamente de una base de datos relacional y ser capaz de **crear tablas**, de **consultarlas**, de**modificarlas**, **de insertar y suprimir registros**.

El libro comienza con una breve historia sobre la creación de la norma SQL y algunas **nociones sobre el modelo relacional**. A continuación, cada capítulo aborda una subdivisión de SQL; la **creación y** la **manipulación de tablas, y** a continuación la **gestión de los datos** de estas tablas. El autor continúa con **las funciones SQL,** la **seguridad de los datos** y las **transacciones** y acaba abordando temas un poco más complejos como las **cargas masivas**, las**importaciones** y **exportaciones de tablas**, los **trigger**, el **PL/SQL** y los**errores que se encuentran con más frecuencia**.

Los ejemplos que se utilizan en este libro se han realizado con la versión Oracle 10g Express Release 10.2.0.1.0 y la versión MySql 5.1.54 y se pueden**descargar** en esta página.

Los capítulos del libro:
Preámbulo - Introducción - La definición de los datos (LDD) - La manipulación de los datos (LMD) - Las funciones - La seguridad de los datos (DCL) - El control de transacciones (TCL) - Para ir más lejos - Presentación de PL/SQL - Los errores más comunes - Anexos

---

Eric GODOC

Eric Godoc es Director de proyectos informáticos en un departamento de sistemas de información. Sus proyectos entorno al desarrollo y migraciones de aplicaciones en grandes empresas le han reportado una gran experiencia en la manipulación de bases de datos relacionales. Con este libro ofrece a los lectores su experiencia, y sobre todo, proporciona los medios para aprender las bases del lenguaje SQL. Su objetivo es puedan responder a la mayoría de necesidades que puedan tener en la utilización de una base de datos relacional, sea cual sea.

Introducción

Esta obra se dirige a los desarrolladores y a aquellos que no sean informáticos que deban trabajar con un sistema gestor de bases de datos relacionales (SGBDR).

El objetivo es describir el funcionamiento del lenguaje SQL independientemente de las variaciones realizadas por los editores de SGBDR.

El libro presenta los principales comandos más utilizados del lenguaje SQL. Ofrece por tanto una visión general y no tiene como objetivo mostrar todas las opciones del lenguaje, aunque hace hincapié en aspectos prácticos y concretos.

Empezaremos con una breve historia sobre la creación y la norma SQL y algunos conceptos sobre el modelo relacional. En otros libros de nuestra editorial se puede encontrar una descripción más detallada del modelo relacional.

A continuación, cada capítulo aborda una subdivisión de SQL, en primer lugar la creación y modificación de tablas y a continuación la gestión de los datos de estas tablas.

Se encadena con la seguridad de los datos y algunas ideas sobre transacciones, y se finaliza con temas un poco más complejos como las cargas masivas, la importación y exportación de tablas, así como PL/SQL y los triggers.

El objetivo de este libro es proporcionar las bases que permitan ser autónomo en la manipulación de una base de datos relacional.

Cada SGBDR ha desarrollado sus propias extensiones de la norma SQL, o ha añadido funciones específicas. De todos modos, aprendiendo las bases del lenguaje SQL se puede responder a la mayoría de las necesidades en la utilización de una base de datos relacional.

Los ejemplos indicados en este libro se han realizado con la versión Oracle 10g Express Release 10.2.0.1.0 y la versión MySQL 5.1.54.

Un poco de historia

Las bases de datos son indispensables en cualquier desarrollo informático. En la mayoría de casos los datos se almacenan en una estructura de datos.

Se habla de BBDD para designar el almacenamiento de los datos y de SGBD para designar los elementos que se ponen a disposición del desarrollador para manipular estos datos.

Existen diferentes tipos de bases de datos.

De tipo jerárquico, como IMS/DL1, que se encuentran mayoritariamente en los Mainframes. Estos elementos se organizan como un árbol con un nivel de jerarquía y de punteros entre registros.

De datos en red (o Codasyl) como IDS2 o SOCRATE que prácticamente no se utilizan en la actualidad, y que utilizan un poco el modelo jerárquico, pero permiten navegar entre los elementos y no solo de forma descendente.

Las bases de datos de tipo relacional aparecieron en los años 80. Se basan en los trabajos desarrollados por un investigador, Edgard Codd, que trabajaba para IBM en el modelo relacional a principios de los años 70. Los datos se organizan en tablas diferentes sin nivel de jerarquía. No hay punteros pero los datos de las tablas permiten realizar vínculos entre las mismas.

El lenguaje SQL - *Structured Query Language* - significa lenguaje de consulta estructurada. Lo creó IBM a principios de los años 70. Una start-up llamada Relational Software produjo la primera versión comercial en 1979. Esta start-up se convirtió posteriormente en Oracle Corp.

El lenguaje SQL se divide en varios subconjuntos:

- El **DDL** (*Data Definition Language*), que agrupa todos los comandos utilizados para crear, modificar o eliminar las estructuras de la base de datos (tablas, índices, vistas, etc.). Se trata principalmente de los comandos CREATE, ALTER y DROP.
- El **DML** (*Data Manipulation Language*), que agrupa los comandos utilizados para manipular los datos contenidos en la base de datos. Se trata principalmente de los comandos SELECT, INSERT, DELETE y UPDATE.
- El **DCL** (*Data Control Language*), que agrupa los comandos utilizados para administrar la seguridad de acceso a los datos. Se trata principalmente de los comandos GRANT y REVOKE.
- El **TCL** por (*Transaction Control Language*), que agrupa los comandos utilizados para administrar la confirmación o no de actualizaciones realizadas sobre la base de datos. Se trata principalmente de los comandos COMMIT y ROLLBACK.

Las normas SQL

La primera versión de SQL normalizada por ANSI data de 1986.

Más tarde se sucedieron versiones más o menos importantes.

La norma SQL2 o SQL92 es la más importante. La mayoría de los SGBDR existentes implementan esta versión.

A continuación siguieron otras evoluciones; SQL-3, SQL:2003 y SQL:2008 que aportan algunas funciones complementarias pero que no todos los SGBDR tienen por qué haber implementado, por ejemplo las funciones orientadas a objetos en SQL-3 o funciones para manipular tipos XML en SQL-2003.

Cada proveedor de SGBDR ha implementado a su manera el lenguaje SQL y ha agregado sus propias extensiones. Los ejemplos que se ofrecen en este libro para ilustrar los conceptos pueden no ser compatibles con todos los SGBDR.

Los ejemplos mostrados en este libro se crearon principalmente con Oracle, que es el más utilizado (alrededor del 50% del mercado) y MySQL que es el más utilizado por las aplicaciones Web.

Norma	Nombre	Comentarios
ISO/CEI 9075:1986	SQL-86 o SQL-87	Editada por ANSI y adoptada por ISO en 1987.
ISO/CEI 9075:1989	SQL-89 o SQL-1	Revisión menor.
ISO/CEI 9075:1992	SQL-92 o SQL2	Revisión mayor.
ISO/CEI 9075:1999	SQL-99 o SQL3	Expresiones racionales, consultas recursivas, disparadores, tipos no escalares y algunas funciones orientadas a objetos.
ISO/CEI 9075:2003	SQL:2003	Introducción de funciones para la manipulación de XML, « window functions », comandos estandarizados y columnas con valores automáticos (incluyendo columnas de identidad).
ISO/CEI 9075:2008	SQL:2008	Se añaden algunas funciones de ventanas (ntile, lead, lag, first value, last value, nth value), limitación del número de registros (OFFSET / FETCH), pequeñas mejoras en los distintos tipos, cursores y mecanismos de autoincrementos.

Descripción rápida del modelo relacional

El modelo relacional fue creado, como decíamos antes, por un investigador, Edgard Codd, que trabajaba en IBM a principios de los años 70. Trabajó a partir de principios matemáticos simples, la teoría de conjuntos y la lógica de predicados.

El modelo relacional se basa en el concepto de conjunto. Esquemáticamente el modelo relacional se puede representar a través de una tabla; igualmente se puede llamar una tabla, una relación.

Este conjunto tiene atributos (las columnas) y líneas que contienen los valores (los registros). La forma más comúnmente utilizada para representar una tabla es esta:

NUMERO	TIPO	MARCA	FECHA_COMPRA	PRECIO	NUM_PROPIETARIO	COLOR
1	SP	1	15/01/10	159	190120	ROJO
2	SP	1	14/03/10	99	190215	
3	CT	3	02/05/10	49	190001	NEGRO
4	DE	4	25/07/10	89	190222	BLANCO
5	IP	5	30/09/10	359	190561	

El modelo relacional presenta los datos de forma lógica, es totalmente independiente del modelo físico. Es el proveedor el que decide el modo de almacenamiento físico de las tablas. Esta es la mayor ventaja de las bases de datos relacionales, la independencia entre lo lógico y lo físico.

Una vez se han definido las tablas, hay que disponer de un lenguaje para manipularlas, se trata del álgebra relacional. También lo inventó Edgard Codd. Con la ayuda de estos comandos, se pueden consultar las relaciones existentes y crear nuevas relaciones. Hablamos de operadores de unión, intersección, diferencia, producto cartesiano, división y composición.

SQL implementa el álgebra relacional y los sistemas de gestión de bases de datos relacionales (SGBDR) implementan el modelo relacional.

1. Principales conceptos del modelo relacional

Los tres principales conceptos del modelo relacional son el dominio, el producto cartesiano y las relaciones.

Dominio

Es un conjunto de valores representado por un nombre.

Por ejemplo:

El tipo de teléfono es un dominio que comprende los valores CT, DE, SP y IP (Con Tapa, Deslizante, Smartphone y iPhone).

El color es otro dominio (ROJO, NEGRO, VERDE...).

El número de ocurrencias de cada uno de los dominios da la CARDINALIDAD.

Para los tipos de teléfono, la cardinalidad es 4.

Producto cartesiano

Representa la unión entre dos dominios. Si, por ejemplo, se realiza el producto cartesiano entre los tipos de teléfono y los colores, se obtienen registros R1, R2.

En nuestro ejemplo, el producto cartesiano del dominio 1: tipo de teléfono y del dominio 2: color del teléfono da:

(CT,VERDE), (CT,NEGRO), (CT,ROJO), (DE,VERDE), (DE,NEGRO), etc.

Relación

El concepto de relación es el fundamento del modelo relacional. La relación permite relacionar los dominios según ciertos criterios.

Por ejemplo, si se quiere crear una relación llamada TELROJO, se indicará que se quieren asociar todos los teléfonos del dominio « Tipo Teléfono » solo con el elemento « ROJO » del dominio « Color ».

La representación de esta relación se hace en forma de tabla de dos dimensiones.

RELACIÓN: TELROJO

TIPO	COLOR
SP	ROJO
CT	TOJO
DE	ROJO
IP	ROJO

Tipo y color son **atributos**.

Cada línea es única y representa un objeto de la relación.

El **grado** es el número de atributos de una relación (en este caso = 2).

2. Principales reglas

El modelo relacional gestiona, pues, un objeto principal, la relación, asociada a los conceptos de dominio y atributo.

A esta relación se aplican reglas para respetar las restricciones vinculadas al análisis. A continuación puede ver algunas de estas reglas:

Coherencia

Cualquier valor que toma un atributo debe pertenecer al dominio sobre el que se ha definido.

Unicidad

Todos los elementos de una relación deben ser diferentes.

Identificador

Atributo o conjunto de atributos que permiten dotar de una característica única a cada elemento de la relación.

Clave primaria

Identificador mínimo de una relación.

Claves secundarias

Otros identificadores de la relación.

Integridad referencial

Esta regla impone que un atributo o un conjunto de atributos de una relación aparezcan como clave primaria en otra relación.

Clave foránea

Atributo o conjunto de atributos que cumplen la regla de integridad referencial.

Ejemplos:

RELACIÓN TELÉFONOS

NÚMERO	TIPO	MARCA	FECHA_COMPRA	PRECIO	NUM_PROPIETARIO	COLOR
1	SP	1	15/01/10	159	190120	ROJO
2	SP	2	14/03/10	99	190215	
3	CT	3	02/05/10	49	190001	NEGRO
4	DE	4	25/07/10	89	190222	BLANCO
5	IP	5	30/09/10	359	190561	

NÚMERO es el identificador primario.

FECHA_COMPRA, PRECIO, NUM_PROPIETARIO y COLOR son claves secundarias.

TIPO y MARCA son claves foráneas que hacen referencia a las claves primarias de las relaciones MARCA_TEL y TIPO_TEL.

RELACIÓN MARCA_TEL

MARCA	DESC_MARCA	PAIS
1	SAMSUNG	COREA
2	SONY	JAPON
3	PHILIPS	HOLANDA
4	MOTOROLA	USA
5	APPLE	USA

MARCA es el identificador primario.

PAIS es el identificador secundario.

Valor nulo

En el modelo relacional, se admite el concepto nulo. Es un valor que representa una información desconocida o inaplicable en una columna. Se representa como _ , ˆ o NULL.

Restricción de entidad

Cualquier valor que forme parte de una clave primaria no puede ser NULL.

Ejemplo:

RELACIÓN TELÉFONO

NÚMERO	TIPO	MARCA	FECHA_COMPRA	PRECIO	NUM_PROPIETARIO	COLOR
1	SP	1	15/01/10	159	190120	ROJO
2	SP	2	14/03/10	99	190215	
3	CT	3	02/05/10	49	190001	NEGRO
4	DE	4	25/07/10	89	190222	BLANCO
5	IP	5	30/09/10	359	190561	

Se puede no indicar el color, mientras que el número del teléfono, que es la clave primaria, es obligatorio.

Los operadores en álgebra relacional

El álgebra relacional ha llevado al desarrollo de SQL que se ha convertido en el estándar en lo que respecta a la gestión de datos.

Es un método de extracción que permite la manipulación de tablas y columnas. Se basa en la creación de nuevas tablas (tablas resultantes) a partir de tablas existentes. Estas nuevas tablas se convierten en objetos que se pueden utilizar inmediatamente.

Los operadores del álgebra relacional que permiten crear las tablas resultantes se basan en la teoría de conjuntos.

La sintaxis y la notación utilizadas aquí son las que más se usan.

1. Unión

La unión entre dos relaciones de igual estructura (grado y dominios) da una tabla resultante de la misma estructura que tiene como elementos el conjunto de los diferentes elementos de las dos relaciones iniciales.

Notación: Rx = R1 ∪ R2

Ejemplos:

Consideremos las relaciones TELEFONO_SMARTPHONE y TELEFONO_DESLIZANTE

RELACIÓN TELEFONO_SMARTPHONE

NÚMERO	TIPO	MARCA	FECHA_COMPRA	PRECIO	NUM_PROPIETARIO	COLOR
1	SP	1	15/01/10	159	190120	ROJO
2	SP	1	14/03/10	99	190215	

RELACIÓN TELEFONO_DESLIZANTE

NÚMERO	TIPO	MARCA	FECHA_COMPRA	PRECIO	NUM_PROPIETARIO	COLOR
4	DE	4	25/07/10	89	190222	BLANCO

UNIÓN DE LAS DOS RELACIONES

NÚMERO	TIPO	MARCA	FECHA_COMPRA	PRECIO	NUM_PROPIETARIO	COLOR
1	SP	1	15/01/10	159	190120	ROJO
2	SP	1	14/03/10	99	190215	
4	DE	4	25/07/10	89	190222	BLANCO

2. Intersección

La intersección entre dos relaciones de igual estructura (grado y dominios) da una tabla resultante de igual estructura que tiene como elementos el conjunto de elementos comunes a las dos relaciones iniciales.

Notación: Rx = R1 ∩ R2

Ejemplo:

RELACIÓN MARCA_TEL

CLAVE	DESC_MARCA	PAÍS

1	SAMSUNG	COREA
2	SONY	JAPÓN
3	PHILIPS	HOLANDA
4	MOTOROLA	USA
5	APPLE	USA

RELACIÓN MARCA_ORDENADOR

CLAVE	DESC_MARCA	PAÍS
1	SAMSUNG	COREA
2	SONY	JAPÓN
3	DELL	USA
4	HP	USA
5	APPLE	USA

MARCAS comunes a las dos relaciones:

CLAVE	DESC_MARCA	PAÍS
1	SAMSUNG	COREA
2	SONY	JAPÓN
5	APPLE	USA

3. Diferencia

La diferencia entre dos relaciones de igual estructura (grado y dominio) da una tabla resultante de igual estructura que tiene como elementos el conjunto de elementos de la primera relación que no están en la segunda.

Notación: Rx = R1 - R2

Ejemplo:

MARCAS presente en la relación 1 y no en la relación 2:

CLAVE	DESC_MARCA	PAÍS
3	PHILIPS	HOLANDA
4	MOTOROLA	USA

4. División

La división entre dos relaciones es posible siempre que la relación divisor esté incluida totalmente en la relación dividendo. El cociente de la división corresponde a la información que, presente en el dividendo, no está en el divisor.

Igualmente es posible definir la división de la siguiente manera: consideremos R1 y R2 como relaciones, de manera que R2 está totalmente incluida en R1. El cociente R1÷R2 se forma de registros r tales que para todos los registros r' definidos en R2, exista el registro r.r' definido en R1.

Notación: Rx = R1÷R2

Ejemplo:

RELACIÓN MARCA_TEL

CLAVE	DESC_MARCA	PAÍS	VENTAS
1	SAMSUNG	COREA	123456
2	SONY	JAPÓN	789012
3	PHILIPS	HOLANDA	3456789
4	MOTOROLA	USA	12356
5	APPLE	USA	1111356

RELACIÓN MARCA_ORDENADOR

CLAVE	DESC_MARCA	PAÍS
1	SAMSUNG	COREA
2	SONY	JAPÓN
3	DELL	USA
4	HP	USA
5	APPLE	USA

La división entre las dos relaciones permite aislar la información complementaria a la relación MARCA_ORDENADOR y muestra la relación MARCA_TEL:

VENTAS
123456
789012
1111356

5. Restricción

La restricción se basa en una condición. A partir de una relación, se crea otra con la misma estructura que solo tenga los elementos de la relación inicial que respondan a la condición.

Notación: Rx = σ (condición) R1

La condición se expresa de la siguiente forma:

```
[NO] [(] atributo operador valor [)] [{Y/O}condición]
```

operador

Un operador de comparación: =, <>, >, <, >=, <=

valor

Una constante u otro atributo.

Ejemplo:

Marcas de ordenador de los USA

```
MARCAUSA=σ(PAIS="USA")MARCA_ORDENADOR
```

CLAVE	DESC_MARCA	PAÍS
3	DELL	USA
4	HP	USA

5	APPLE	USA

6. Proyección

La proyección de una relación sobre un grupo de atributos da una relación que tiene como estructura solo estos atributos, y como elementos n-tupla diferentes compuestos por los valores asociados de estos atributos.

Notación: Rx = π R (A1, A2,..., An)

Ejemplo: sobre Desc_marca y País de la tabla MARCA_ORDENADOR:

```
LISTMARCA=π MARCA_ORDENADOR(DESC_MARCA,PAÍS)
```

DESC_MARCA	PAÍS
SAMSUNG	COREA
SONY	JAPÓN
DELL	USA
HP	USA
APPLE	USA

Si hubiera duplicados, se habrían eliminado del resultado.

7. Producto cartesiano

El producto cartesiano entre dos relaciones crea una relación que tiene como estructura todos los atributos de las dos relaciones existentes y como elementos la asociación de cada fila de la primera tabla con cada fila de la segunda.

Notación: Rx = S1 X S2

8. Join

La unión entre dos relaciones se produce por la restricción sobre el producto cartesiano.

Notación: Rx = S1 JOIN (condición) S2.

Ejemplo:

RELACIÓN MARCA_TEL

CLAVE	DESC_MARCA	PAÍS
1	SAMSUNG	COREA
2	SONY	JAPÓN
3	PHILIPS	HOLANDA
4	MOTOROLA	USA
5	APPLE	USA

RELACIÓN MARCA_ORDENADOR

CLAVE	DESC_MARCA	PAÍS
1	SAMSUNG	COREA

2	SONY	JAPÓN
3	DELL	USA
4	HP	USA
5	APPLE	USA

JOIN entre estas dos relaciones:

```
TELMARCA = MARCA_TEL JOIN (MARCA_TEL.CLAVE=MARCA_ORDENADOR.CLAVE)
MARCA_ORDENADOR.
```

CLAVE	DESC_MARCA	PAÍS
1	SAMSUNG	COREA
2	SONY	JAPÓN
3	DELL	USA
4	HP	USA
5	APPLE	USA
1	SAMSUNG	COREA
2	SONY	JAPÓN
3	DELL	USA
4	HP	USA
5	APPLE	USA

Los diferentes tipos de join son:

Theta-join

La condición es una comparación entre dos atributos.

Equi-join

La condición es la igualdad entre dos atributos.

Natural join

Equi-join entre los atributos que tienen el mismo nombre.

9. Cálculos elementales

Proyección sobre una relación asociada a un cálculo en cada fila para crear uno o varios atributos nuevos.

Notación: Rx = π S (A1,...,N1 = expresión calculada,...)

La expresión calculada puede ser:

- una operación aritmética,
- una función matemática,
- una función sobre una cadena.

10. Cálculo de agregados

Proyección sobre una relación asociada a uno o más cálculos estadísticos sobre un atributo para todos los elementos de la relación o del agrupamiento vinculado a la proyección para crear uno o más atributos nuevos.

Notación: Rx = π S (A1,...,N1= función estadística (Ax),...)

Las principales funciones estadísticas son:

- **COUNT (*)**: número de filas.
- **COUNT (atributo)**: número de valores no nulos.
- **SUM (atributo)**: suma de los valores no nulos.
- **AVG (atributo)**: promedio de los valores no nulos.
- **MAX (atributo)**: valor máximo (no nulo).
- **MIN (atributo)**: valor mínimo (no nulo).

Los sistemas de gestión de bases de datos que utilizan SQL

Los SGBDR en el mercado son bastante numerosos. Los más utilizados son Oracle y DB2 (IBM) especialmente en empresas que tienen volúmenes de datos importantes y grandes restricciones de rendimiento transaccionales.

Algunos están más orientados a aplicaciones Web como MySQL (Oracle) y PostgreSQL (libre).

También podemos citar SQL Server (Microsoft), Sybase (Sysbase), Informix (Informix), Access (Microsoft), etc.

Aquí no vamos a poder comparar todas las características, ventajas e inconvenientes de todos estos SGBDR. La elección de una base de datos es una decisión que se debe reflexionar bien, en función de criterios como el volumen de datos, las restricciones de rendimiento, los costes de compra y mantenimiento, los conocimientos de los programadores, etc.

Los tipos de datos

En esta sección nos ocuparemos de los tipos de datos más utilizados para la descripción de las columnas de una tabla. Existen tres grandes familias de datos: numéricos, caracteres (o alfanumérico) y cronológico (fechas y horas).

Cada SGBDR ha creado tipos específicos para necesidades precisas o por problemas de almacenamiento. Al final del capítulo se indica una lista de los tipos de datos específicos para Oracle y MySQL.

1. Numéricos

Los principales tipos numéricos de la norma SQL2 son **NUMERIC** (o **DECIMAL**) para datos con coma decimal, **INTEGER** para los enteros largos y **SMALLINT** para los enteros cortos.

En la manipulación de datos financieros o contables, hay que utilizar estos tipos de datos y evitar tipos de datos binarios que no garantizan la exactitud de los cálculos con redondeo.

Oracle ha « simplificado » fusionando los dos tipos NUMERIC e INTEGER en un solo tipo llamado NUMBER (o DECIMAL).

Ejemplo Oracle:

```
PRECIO  NUMBER (8,2) o PRECIO  DECIMAL (8,2) son idénticos
NBTELEF NUMBER (5)   o NBTELEF NUMBER  (5,0) son idénticos
```

En el primer ejemplo, se describe un tipo decimal de 8 dígitos de longitud de los que dos son decimales, como 123456,12.

En el segundo ejemplo, no se indica la longitud después de la coma ya que se trata de un entero.

MySQL, por su parte, también ha fusionado los dos tipos numéricos en el tipo DECIMAL.

Ejemplo MySQL:

```
PRECIO  DECIMAL (8,2)
NBTELEF DECIMAL (5)
CODITEL SMALLINT
```

En la gran mayoría de los casos, el tipo NUMBER (o DECIMAL para MySQL) es suficiente para describir un dato numérico. Además, garantiza la exactitud del redondeo en los cálculos.

2. Caracteres

El tipo alfanumérico clásico se define con CHAR para los datos de longitud fija y VARCHAR para las cadenas de caracteres de longitud variable.

La mayor parte de las veces, se aconseja utilizar VARCHAR. De hecho un dato definido de 50 caracteres que solo contiene dos se guardará con dos caracteres con VARCHAR mientras que con CHAR se almacenarán 50 caracteres. CHAR completa con espacios hasta la longitud máxima.

En el caso de VARCHAR, la longitud real de la cadena se almacena en un byte que permite así al SGBDR devolver correctamente el dato.

En Oracle, el tamaño máximo de un dato CHAR es de 2000 caracteres y de 4000 caracteres en el caso de VARCHAR. Normalmente se habla de bytes y no caracteres pero la mayoría de los SGBDR se configuran para que 1 byte = 1 carácter.

En MySQL, el tamaño máximo de una cadena CHAR o VARCHAR normalmente es de 255 caracteres. No obstante, si la longitud introducida es mayor, MySQL transforma automáticamente el tipo en TEXT, MEDIUMTEXT o LONGTEXT. Tenga en cuenta que la transformación afecta al almacenamiento del dato, y se pierde el concepto de tamaño variable; los tipos TEXT, MEDIUMTEXT y LONGTEXT son de longitud fija. Consulte la documentación antes de crear las columnas de una tabla para utilizar el tipo correcto en función de sus necesidades.

Una última observación: Oracle recomienda usar el tipo VARCHAR2 en lugar de VARCHAR. A priori estos dos tipos son idénticos pero Oracle podría utilizarlos de forma diferente en versiones futuras.

Esto puede crear un problema de compatibilidad. En el caso en que las líneas de código se utilicen en SGBDR diferentes, es preferible utilizar VARCHAR que es el estándar SQL, ya que VARCHAR2 sólo es válido en Oracle.

Ejemplo:

```
        TIPO        CHAR(2),
        COLOR       VARCHAR(25));
```

3. Fechas y horas

Los tipos de formato cronológico son principalmente DATE, TIME y TIMESTAMP.

Los formatos por defecto de fechas o de horas varían de una base de datos a otra. Para conocer el formato por defecto hay que utilizar:

Para Oracle:

```
SELECT SYSDATE FROM DUAL
```

Para MySQL :

```
SELECT NOW()
```

que devuelven la fecha del día en el formato utilizado por la base de datos.

Es preferible no confiar en un formato definido por una base de datos, ya que puede evolucionar o la secuencia de código se podría utilizar en otra base de datos que no tendría por qué tener el mismo formato de fecha por defecto.

En la sección Las funciones de gestión de fechas y horas del capítulo Las funciones veremos cómo seleccionar o insertar datos de este tipo.

En la creación de sus tablas, no es necesario conocer el formato de almacenamiento, hay que declarar la columna DATE, TIME o TIMESTAMP.

DATE: formato fecha, norma ISO AAAA-MM-DD

TIME: formato hora, norma ISO HH:MM:ss.xxx

TIMESTAMP : formato fecha y hora, norma ISO AAAA-MM-DD HH:MM:ss.xxx

Atención: Oracle no tiene el tipo TIME, hay que utilizar el tipo TIMESTAMP, o almacenar la hora en un formato CHAR(10) y después utilizar los formatos de conversión o almacenar las horas convirtiéndolas en segundos en un campo de tipo NUMBER.

MySQL ofrece además los siguientes tipos:

- DATETIME que es equivalente a TIMESTAMP.
- YEAR que devuelve el año en cuatro caracteres.

Ejemplo

```
        FECHA_COMPRA            DATE,
        FECHA_ACTUALIZACION     TIMESTAMP
```

Más tarde veremos que la manipulación de las fechas y las horas es muy diferente en los diferentes SGBDR; las funciones de conversión son claramente diferentes. Existen multitud de funciones de fecha y hora en MySQL y Oracle (véase en el capítulo Las funciones, la sección Los diferentes formatos de visualización de fechas).

4. Los otros tipos de datos

a. Los otros tipos numéricos

MySQL utiliza los siguientes tipos:

Tipos numéricos	Valor mínimo Con signo o sin signo	Valor máximo Con signo o sin signo	Almacenamiento (bytes)
TINYINT	-128 0	127 255	1
SMALLINT	-32768 0	32767 65535	2
MEDIUMINT	-8388608 0	8388607 16777215	3
BIGINT	-9223372036854775808 0	9223372036854775807 18446744073709551615	8

Existen igualmente los tipos FLOAT, REAL y DOUBLE PRECISION que se utilizan principalmente para disminuir el espacio en disco utilizado, ya que se almacenan en binario.

Se pueden utilizar en caso de cálculos científicos para acelerar un poco los procesos respecto a un tipo DECIMAL clásico.

b. Los otros tipos carácter

Oracle utiliza los siguientes tipos:

Tipos carácter	Valor mínimo	Valor máximo
LONG	1	2 Gb
CLOB o NCLOB	1	4 Gb

MySQL utiliza los siguientes tipos:

Tipos carácter	Valor mínimo	Valor máximo
TINYBLOB	1	255 bytes
TINYTEXT	1	255 bytes
BLOB	1	65535 bytes
TEXT	1	65535 bytes
MEDIUMBLOB	1	16777215 bytes

MEDIUMTEXT	1	16777215 bytes
LONGBLOB	1	4 294 967 295 bytes
LONGTEXT	1	4 294 967 295 bytes

c. Los tipos binarios

Oracle utiliza los siguientes tipos:

Tipos numéricos	Valor mínimo	Valor máximo	Observaciones
RAW	1	2000 bytes	Introducido en hexadecimal
LONG RAW	1	2 Gb	Introducido en hexadecimal
BLOB	1	4 Gb	Objeto binario grande
BFILE	1	4 Gb	Puntero a un fichero binario externo

Esta lista no es muy exhaustiva, consulte la documentación de la base de datos para obtener el detalle de todos los tipos posibles.

La creación de las tablas

En esta sección, vamos a ver cómo crear una tabla, agregar o eliminar columnas, poner comentarios en las columnas y las tablas, e igualmente, el método para copiar una tabla a otra y cómo asignar un sinónimo a un nombre de tabla.

1. El comando CREATE

CREATE es el comando base del lenguaje SQL para crear un elemento. Se utiliza para crear una TABLA, un ÍNDICE, una VISTA o un SINÓNIMO. En función del elemento que demos de alta, la sintaxis es diferente.

En esta sección, vamos a tratar la creación de TABLAS. Una tabla se define principalmente por las columnas que la componen y las reglas que se aplican a estas columnas.

No abordaremos los aspectos del almacenamiento físico de la tabla. De hecho, cada SGBDR tiene su propia sintaxis en este tema. En la mayoría de los casos son los DBA (*Database Administrator* - administrador de la base de datos) los que especifican las normas de almacenamiento y las opciones a aplicar en las tablas. El almacenamiento de las tablas es un elemento determinante en términos de rendimiento y de seguridad de la base de datos. Por tanto es muy recomendable preguntar a un administrador antes de lanzarse a la creación de una tabla.

Las tablas creadas para realizar una prueba o un juego de pruebas se pueden eliminar con el comando DROP TABLE un vez se ha terminado, para no «contaminar» la base con tablas temporales.

Antes de crear una tabla, es preferible hacerse algunas preguntas y respetar algunas reglas.

Trate de dar nombres que identifiquen bien las tablas y las columnas para luego encontrar fácilmente estos datos. Los DBA definen casi siempre las reglas de nomenclatura de las tablas y de las columnas. A veces proporcionan scripts estándar para crear una tabla. Es pues una obligación, consultar las normas en vigor en la empresa antes de cualquier creación.

Las reglas mínimas a respetar en la nomenclatura de una tabla no son muy numerosas y se resumen en: un nombre de tabla debe comenzar por una letra, es único y no puede pasar de 256 caracteres.

Cuando una columna tiene el mismo significado en diferentes tablas, se aconseja conservar el mismo nombre. De hecho, si la columna se encuentra en varias tablas y se ha respetado el modelo relacional, esto significa que la columna es la clave de una tabla. Conservando el mismo nombre se simplifican las uniones entre las dos tablas, ya que las columnas clave se identificarán fácilmente.

En el capítulo La manipulación de los datos (LMD) - sección La utilización de los alias, veremos cómo distinguir dos columnas del mismo nombre en una misma selección.

Sintaxis:

```
CREATE TABLE nombre_de_tabla  (Nombre_columna Tipo_columna,
                               Nombre_columna Tipo_columna,
                               Nombre_columna Tipo_columna,
                            ... );
```

Veamos tres estructuras de tablas:

Tabla TELEFONO

TELEFONO
NUMERO TIPO MARCA FECHA_COMPRA PRECIO NUM_PROPIETARIO COLOR

Tabla MARCA

MARCA
MARCA DESC_MARCA PAIS

Tabla TIPO_TEL

TIPO_TEL
TIPO DESC_TIPO

Los scripts necesarios para crear las tablas son:

```
CREATE TABLE TELEFONO (NUMERO       INTEGER,
                       TIPO         CHAR(2),
                       MARCA        INTEGER,
                       FECHA_COMPRA DATE,
                       PRECIO       NUMBER(9,2),
                       NUM_PROPIETARIO INTEGER,
                       COLOR        VARCHAR(25));
```

Sintaxis MySQL (se reemplaza « NUMBER » por « DECIMAL »):

```
CREATE TABLE TELEFONO (NUMERO       INTEGER,
                       TIPO         CHAR(2),
                       MARCA        INTEGER,
                       FECHA_COMPRA DATE,
                       PRECIO       DECIMAL (9,2),
                       NUM_PROPIETARIO INTEGER,
                       COLOR        VARCHAR(25));
```

Sintaxis Oracle y MySQL:

```
CREATE TABLE TIPO_TEL   (TIPO         CHAR(2),
                         DESC_TIPO    VARCHAR(25));
```

```
CREATE TABLE MARCA_TEL (MARCA       INTEGER,
                        DESC_MARCA  VARCHAR(25),
                        PAIS        VARCHAR(30));
```

Se trata de ejemplos simples. Ninguna columna tiene restricciones o valor por defecto. No se ha definido ninguna clave.

Podemos constatar que las columnas TIPO y MARCA son las claves de las tablas TIPO_TEL y MARCA_TEL y que encontramos estas columnas con el mismo nombre en la tabla TELEFONO. De este modo, si realizamos uniones entre estas tablas, utilizaremos estas columnas.

Por necesidades de normalización o simplemente para conocer en todo momento a qué tabla pertenecen las columnas, es posible poner un prefijo al nombre de las columnas con los primeros caracteres del nombre de la tabla.

Ejemplo:

```
CREATE TABLE TELEFONO (TELEF_NUMERO       INTEGER ,
                       TELEF_TIPO         VARCHAR(2),
                       TELEF_MARCA        INTEGER,
                       TELEF_FECHA_COMPRA DATE,
                       TELEF_PRECIO       NUMBER(9,2),
                       TELEF_PROPIETARIO  VARCHAR(25),
                       TELEF_NUM_PROVEEDOR INTEGER);
```

De este modo, al utilizar una columna, el programador ya sabe a qué tabla pertenece.

2. Los comentarios (COMMENT)

La cláusula COMMENT permite añadir comentarios a las columnas y las tablas. Esta opción es muy útil para los futuros programadores, que podrán conocer la función de una tabla y el contenido de cada columna visualizando su descripción.

Esto no dispensa de tener una documentación en papel que describa cada tabla de la base de datos.

Sin embargo, si toma el hábito de documentar todas las tablas y las columnas directamente en la base de datos cuando se crean, el mantenimiento y los posibles evolutivos de la base de datos se simplifican.

Una base de datos tiene un ciclo de vida importante y será utilizada por muchos y diversos usuarios.

Los comentarios serán de gran ayuda sobre todo cuando la base de datos es antigua y la documentación en papel ya no exista. Especialmente en caso de migración de base de datos o de migración de un sistema a otro.

La sintaxis de esta cláusula difiere de un SGBDR a otro.

En Oracle, la sintaxis es la siguiente:

```
COMMENT ON <'COLUMN' o 'TABLE'> <Nombre columna o nombre de tabla> IS
'Comentario';
```

Agregar un comentario en una tabla:

```
COMMENT ON TABLE TELEFONO IS 'Lista de teléfonos de la empresa';
```

Agregar un comentario en una columna:

```
COMMENT ON COLUMN TELEFONO.TIPO IS 'Indica si es un teléfono de tipo
SMARTPHONE, CON TAPA u OTRO';
```

Los comentarios se almacenan en dos tablas Oracle:

```
user_tab_comments            Los comentarios de las tablas
user_col_comments            Los comentarios de las columnas
```

Ejemplo de script SQL Oracle que muestra las columnas de una tabla así como los comentarios asociados:

```
 SET FLU OFF
 SET PAGESIZE 255
 SET FEEDBACK OFF
 SET WRAP OFF
 SET VER OFF
 SET ARRAYSIZE 1
 COL type FORMAT A15
 COL not_null FORMAT A8
```

```
 PROMPT .
 PROMPT Estructura de la tabla TELEFONO:
 PROMPT .
 SELECT UPPER(a.column_name) nombre,
        a.data_type || '(' ||
        DECODE(a.data_type,'NUMBER',a.data_precision,a.data_length)
 || ')' tipo,DECODE(a.nullable,'N','   N','') not_null,
b.comments comentario
 FROM user_tab_columns a, user_col_comments b
 WHERE a.table_name = 'TELEFONO'
 AND   b.table_name = TELEFONO'
 AND   a.table_name = b.table_name
 AND   a.column_name = b.column_name
 ORDER BY a.column_id;
```

En MySQL, los comentarios se insertan en el momento de la creación de la tabla añadiendo el comando COMMENT de la siguiente forma:

```
CREATE TABLE TELEFONO (
NUMERO       INTEGER    COMMENT 'Número teléfono',
TIPO         VARCHAR(2) COMMENT 'Tipo'     ,
MARCA        INTEGER    COMMENT 'Nombre de la Marca',
FECHA_COMPRA DATE,
PRECIO       DECIMAL(9,2) COMMENT 'Precio sin contrato',
PROPIETARIO  VARCHAR(25),
NUM_PROVEEDOR INTEGER);
```

Para mostrar las columnas y los comentarios, se debe utilizar este comando:

```
SHOW FULL COLUMNS FROM TELEFONO;
```

3. Crear una tabla a partir de otra

Otro método para crear una tabla consiste en duplicar o basarse en tablas existentes.

Este método se utiliza a menudo por los desarrolladores para crear tablas de pruebas y juegos de prueba. No se recomienda utilizar este método para la definición inicial de la base de datos.

La creación de una tabla asociada a un comando de selección de columna de otra tabla o de varias tablas permite generar rápidamente la estructura de una tabla específica y en el mismo comando llenar esta tabla.

Con esta sintaxis, las posibilidades se amplían, dado que se pueden utilizar todas las opciones del comando SELECT.

Es posible recuperar algunas columnas, filtrar los datos y copiar solo la estructura sin los datos añadiendo una condición que no se cumpla nunca.

Sintaxis

```
CREATE TABLE <Nueva tabla> AS SELECT <nombre columna1>,<nombre
columna2>... o <*> FROM <Tabla a copiar> WHERE ... ... ;
```

Ejemplo: copia de estructura y de datos

Con Oracle, hay que utilizar la siguiente sintaxis:

```
CREATE TABLE COPIA_TELEFONO AS SELECT * FROM TELEFONO;
```

Con este comando, se copia la estructura de la tabla así como todo el contenido de la tabla TELEFONO en la tabla COPIA_TELEFONO.

Atención: con tablas muy grandes, puede haber problemas de espacio en disco o de tiempos de respuesta importantes. Se recomienda probar la consulta SELECT antes de ejecutar la creación

propiamente dicha.

Por ejemplo contando el número de filas de la tabla de teléfonos con SELECT COUNT(*) FROM TELEFONO que permite conocer el volumen a transferir.

Con MySQL, la sintaxis es la siguiente (el « AS » desaparece):

```
CREATE TABLE COPIA_TELEFONO SELECT * FROM TELEFONO;
```

Para recuperar solo la estructura de la tabla origen, hay que añadir una condición que no se cumpla nunca. En la sección La selección de datos del capítulo La manipulación de los datos (LMD), veremos la descripción detallada de la cláusula SELECT.

Ejemplo: copia de estructura

```
CREATE TABLE COPIA_TELEFONO AS SELECT * FROM TELEFONO WHERE 1=2;
```

Como la cláusula WHERE no se cumple nunca, el SELECT no devuelve ninguna fila.

También es posible seleccionar sólo algunas columnas de la tabla origen.

Ejemplo: copia de una parte de la estructura

```
CREATE TABLE COPIA_TELEFONO AS SELECT NUMERO, NUM_PROPIETARIO FROM
TELEFONO WHERE 1=2;
```

La tabla COPIA_TELEFONO contiene sólo dos columnas y estará vacía.

COPIA_TELEFONO
NUMERO NUM_PROPIETARIO

Con MySQL, hay que copiar la estructura de una tabla en otra utilizando «LIKE».

```
CREATE TABLE COPIA_TELEFONO LIKE TELEFONO;
```

También es posible seleccionar sólo algunas filas de la tabla origen.

Ejemplo: copia de una parte de la estructura y selección de filas (sintaxis Oracle)

```
CREATE TABLE COPIA_TELEFONO AS SELECT NUMERO, NUM_PROPIETARIO,
COLOR FROM TELEFONO WHERE TIPO = 'SP';
```

De este modo la tabla TELEFONO contiene los siguientes datos:

NUMERO	TIPO	MARCA	FECHA_COMPRA	PRECIO	NUM_PROPIETARIO	COLOR
1	SP	1	15/01/10	159	190120	ROJO
2	SP	2	14/03/10	99	190215	NEGRO
3	CL	3	02/05/10	49	190001	NEGRO
4	CO	4	25/07/10	89	190222	BLANCO
5	IP	5	30/09/10	359	190561	NEGRO

La tabla COPIA_TELEFONO contiene los siguientes elementos:

NUMERO	NUM_PROPIETARIO	COLOR
1	190120	ROJO
2	190215	NEGRO

Solo se han seleccionado las dos primeras líneas de la tabla TELEFONO. Corresponden al criterio Tipo= 'SP' y la tabla COPIA_TELEFONO tiene sólo tres columnas.

Con este tipo de selección, es posible crear muy fácilmente un juego de pruebas para su uso o para un usuario. En la sección Las vistas de este capítulo veremos que el funcionamiento de las vistas es muy parecido a este método.

4. Utilización de sinónimos

Para dar un nombre diferente a una tabla, hay que utilizar el comando CREATE SYNONYM.

En una base de datos, las tablas están asignadas a la persona (usuario) que las ha creado. Se dice que pertenecen a un usuario. Posteriormente, este usuario que normalmente es el DBA da privilegios (comando GRANT) al resto de personas que tengan que utilizar sus tablas.

Por defecto, una tabla tiene como prefijo el usuario que la ha creado. Ejemplo: ALEX.TELEFONO.

Los desarrolladores no tienen por qué saber el usuario que ha creado las tablas.

« ALEX » creará SINÓNIMOS de las tablas para que el usuario no tenga que poner el prefijo.

Esta función no está implementada en MySQL.

Sintaxis:

```
CREATE SYNONYM <Nombre sinónimo > FOR <Nombre tabla>;
```

Ejemplo:

```
CREATE SYNONYM TELEFONO FOR ALEX.TELEFONO;
```

De este modo, cuando un programador utilice la tabla, solo indicará TELEFONO:

```
SELECT * FROM TELEFONO;
```

También se puede utilizar un sinónimo para simplificar un nombre de tabla que esté normalizado, o para apuntar a un archivo cuyo nombre no conocen los usuarios.

Ejemplos:

```
CREATE SYNONYM TELEFONO FOR ALEX.TE_ANYO_2009_BARCELONA;
CREATE SYNONYM TELEFONO FOR ALEX.TEL_ESPANYA_ESTE_010402;
```

5. Las SECUENCIAS

A menudo, en una tabla hay una columna que hace de identificador único y que casi siempre es de tipo numérico.

Para dejar al SGBDR gestionar el incremento de esta columna, se puede utilizar un objeto de tipo SEQUENCE y asociarlo a una columna.

Cada vez que se inserta una fila, no es necesario asignar un valor a esta columna, sino que el sistema se

ocupa de ello. De este modo, estaremos seguros de no tener duplicados en esta columna.

Se puede crear una secuencia independiente de una columna de tabla. En el funcionamiento de una aplicación, un contador puede servir para muchas funciones.

Sintaxis básica:

```
CREATE SEQUENCE <nombre secuencia>;
```

El sistema crea un objeto que toma por defecto el valor 1 y se incrementa de 1 en 1.

Ejemplo:

```
CREATE SEQUENCE S_NUMERO;
```

Si ahora queremos asignar un valor de inicio, un incremento o un valor máximo, hay que utilizar las siguientes opciones.

Ejemplo:

```
CREATE SEQUENCE S_NUMERO START WITH 5 INCREMENT BY 1
MINVALUE 2 MAXVALUE 999999 CYCLE;
```

En este ejemplo la secuencia se iniciará en el valor 5, se incrementará de 1 en 1, será siempre superior a 2 e inferior a 999999. Cuando la secuencia llegue al valor máximo 999999, volverá a comenzar en 2.

Si no queremos que la secuencia vuelva a empezar después de llegar al valor máximo, hay que eliminar la opción « CYCLE ». El SGBDR devolverá un error cuando se le pida el valor siguiente a 999999. Es bastante arriesgado autorizar una secuencia con un bucle, ya que puede generar duplicados en la base de datos o provocar un error si la columna se ha declarado como UNIQUE.

Para saber el valor actual o incrementar una secuencia, hay que utilizar las siguientes funciones:

- CURRVAL: recupera el valor actual.
- NEXTVAL: incrementa la secuencia prevista (1 por defecto).

Ejemplo:

```
SELECT S_NUMERO.NEXTVAL FROM DUAL;
SELECT S_NUMERO.CURRVAL FROM DUAL;
```

La tabla « DUAL » se utiliza para trabajar con funciones no vinculadas a una tabla real. Es una tabla virtual.

Para vincular una secuencia a una columna de una tabla, en Oracle hay que crear un trigger. En otro capítulo veremos la utilización de los triggers pero ahora veremos cómo crear uno en la columna NUMERO de la tabla TELEFONO que utiliza la secuencia S_NUMERO.

Ejemplo de columna auto-incrementada con Oracle:

```
CREATE OR REPLACE TRIGGER TR_NUMERO
BEFORE INSERT ON TELEFONO
FOR EACH ROW
DECLARE

BEGIN

   SELECT S_NUMERO.NEXTVAL
   INTO :NEW.NUMERO FROM DUAL;
END;
/
```

De este modo, en cada inserción en la tabla TELEFONO, no es necesario indicar ningún valor para la

columna NUMERO.

Existe otra forma de utilizar la secuencia en la inserción: codificándola en el comando INSERT:

```
INSERT INTO TELEFONO VALUES
(S_NUMERO.NEXTVAL,'CT',5,to_date('01/01/2009','DD/MM/YYYY'),
 99,122120,'BLANCO');
```

Atención: puede haber « agujeros » en una serie de cifras que provengan de una secuencia. De hecho, si después de un INSERT el usuario no confirma la transacción con un COMMIT o se ejecuta un ROLLBACK, el último valor de la secuencia no se modifica y se continúa incrementando normalmente.

Oracle implementa el objeto secuencia como se indicaba antes.

MySQL no implementa las secuencias, pero existe la función AUTO_INCREMENT sobre una columna en la creación de la tabla. Esta columna debe ser la PRIMARY KEY.

Ejemplo de columna auto-incrementada con MySQL:

```
CREATE TABLE TELEFONO  (NUMERO       INTEGER AUTO_INCREMENT,
                        TIPO         VARCHAR(2),
                        MARCA        INTEGER,
                        FECHA_COMPRA DATE,
                        PRECIO      DECIMAL(9,2),
                        NUM_PROPIETARIO INTEGER,
                        COLOR        VARCHAR(25),
CONSTRAINT PK_TELEFONO PRIMARY KEY (NUMERO,TIPO,MARCA));
```

Para eliminar una secuencia, hay que utilizar DROP SEQUENCE.

Ejemplo:

```
DROP SEQUENCE S_NUMERO;
```

Para modificar una secuencia, hay que utilizar ALTER SEQUENCE.

Ejemplos:

Cambiar el valor máximo de la secuencia:

```
ALTER SEQUENCE S_NUMERO MAXVALUE 888888;
```

Modificar el incremento de 1 a 5:

```
ALTER SEQUENCE S_NUMERO INCREMENT BY 5;
```

La eliminación de tablas

La eliminación de tablas es una operación simple pero hay que hacerlo con prudencia. La eliminación es definitiva y no habrá ninguna posibilidad de recuperar los datos de la tabla una vez se ha ejecutado la orden.

1. El comando DROP

El comando DROP permite eliminar definitivamente una tabla. Se elimina la tabla y su contenido.

A menudo se utiliza el comando DROP justo antes de la creación de una tabla. Así se evitan los errores con una tabla existente.

El comando destruye automáticamente los índices y restricciones de la tabla así como los comentarios. Por el contrario, el comando no elimina los sinónimos.

Si es una tabla sensible, es preferible guardarla previamente con un CREATE ... AS SELECT … por ejemplo.


Atención: la tabla no puede estar siendo utilizada por otra persona.

En el caso de un error no se podrá recuperar la tabla (ROLLBACK). Algunas versiones de SGBDR permiten la recuperación después de un DROP (Oracle a partir de la versión 10g por ejemplo).

En MySQL, los comandos de manipulación de tablas comportan una confirmación automática (COMMIT). Así pues, es imposible recuperar una tabla después de un DROP.

Sintaxis:

```
DROP TABLE nombre_de_tabla;
```

Ejemplo:

```
DROP TABLE TELEFONO;
```

La modificación de tablas

Una vez se han creado las tablas, pueden evolucionar en el tiempo y podrá ser necesario añadir, eliminar columnas o modificar restricciones de columnas.

En algunos casos, se podrá renombrar una tabla. Todo esto es lo que vamos a detallar en las siguientes secciones.

1. El comando ALTER

El comando ALTER se utiliza para realizar diferentes acciones. Se puede utilizar para eliminar o añadir una columna de una tabla. También para añadir o eliminar una restricción o añadir un valor por defecto a una columna.

Atención: No se permite cambiar el nombre de una columna ni tampoco su tipo o atributos NULL o NOT NULL.

Algunos SGBDR aceptan el comando MODIFY y permiten modificar el tipo de una columna.

Atención no obstante al contenido de esta columna; en el paso de VARCHAR a INTEGER la conversión automática realizada por el SGBDR modificará el contenido de los datos.

Existe el riesgo de perder información o de tornar incompatibles algunos datos con su utilización. De manera general, se desaconseja modificar el tipo de dato de una columna. Para evitar problemas, habría que vaciar la tabla antes de cambiar el tipo de dato.

Sintaxis:

```
ALTER TABLE nombre_de_tabla [ADD nombre_de_columna Tipo_columna]
                            [,DROP COLUMN nombre_de_columna]
                            [,ADD CONSTRAINT nombre_restriccion]
                            [,DROP CONSTRAINT nombre_restriccion];
```

Ejemplos:

Añadir una columna:

```
ALTER TABLE TELEFONO ADD TELEF_NUM_PIN INTEGER;
```

Tabla TELEFONO

TELEFONO
NUMERO TIPO MARCA FECHA_COMPRA PRECIO NUM_PROPIETARIO COLOR TELEF_NUM_PIN

La columna añadida se sitúa al final de la definición de la tabla.

Eliminar una columna:

```
ALTER TABLE TELEFONO DROP COLUMN TELEF_NUM_PIN;
```

2. Renombrar una tabla (RENAME)

El comando RENAME permite cambiar el nombre de una tabla. Este comando se puede utilizar en caso de que la tabla se deba volver a crear conservando los datos actuales. Renombraremos la tabla actual y a continuación volveremos a hacer el CREATE inicial.

Es más sencillo y más rápido que hacer un CREATE … AS SELECT … de la tabla en cuestión.

Sintaxis:

```
RENAME nombre_de_tabla_antigua TO nombre_de_tabla_nueva;
```

Ejemplo:

```
RENAME TELEFONO TO COPIA_TELEFONO;
```

Atención: algunos SGBDR no implementan este comando. En este caso, hay que crear otra tabla con la misma estructura como en el siguiente ejemplo:

```
CREATE TABLE COPIA_TELEFONO AS SELECT * FROM TELEFONO;
```

y a continuación eliminar la tabla inicial:

```
DROP TABLE TELEFONO;
```

Las vistas

En esta sección vamos a ver cómo crear o eliminar vistas. Las vistas son elementos muy útiles en la programación SQL. Permiten principalmente crear tablas « virtuales » específicas para un dominio o para un tipo de usuarios.

1. Por qué utilizar vistas

En una base de datos, hay tablas permanentes que se han definido después de un análisis de las necesidades y un modelo en forma de tabla.

Si se respeta el modelo relacional, no hay datos redundantes a excepción de las claves que se utilizan para las uniones. Por contra, los usuarios o los desarrolladores tienen necesidad de extracciones específicas de datos. Estas extracciones se materializan en forma de consultas ejecutadas manualmente o incluso en los programas.

Si estas peticiones son repetitivas o comunes a muchos usuarios, puede ser necesario crear una vista. La vista es una representación lógica de la base de datos resultante de una consulta para una necesidad concreta y repetitiva. A diferencia de una tabla, no está almacenada en disco (salvo que se especifique) sino en memoria.

La vista también permite simplificar la base de datos para el usuario, quien no tiene que conocer la totalidad del esquema sino simplemente algunos elementos específicos útiles para su trabajo.

Si sus tablas tienen información confidencial, la vista permite ocultar ciertas columnas. Así el usuario sólo ve lo que le queramos mostrar.

La creación de una vista sigue el mismo mecanismo que el CREATE TABLE … AS SELECT … explicado en las secciones anteriores. De hecho, la vista es una suma de columnas que provienen de una o más tablas.

La principal ventaja de una vista es que está permanentemente actualizada. De hecho, una vista se actualiza automáticamente cuando se modifica alguna de las tablas a las que hace referencia. Por contra, una vista no es un objeto propiamente dicho, sino un resultado de una consulta, por lo que no podemos actualizar datos sobre una vista. Los datos pertenecen a las tablas de SELECT.

Una vista representa en un instante dado la imagen de las tablas que utiliza.

2. La creación de vistas

La creación se realiza con el comando CREATE VIEW y a continuación un comando SELECT recupera las columnas que se quieren extraer de las tablas.

Igual que en el CREATE TABLE ... AS SELECT ... se pueden utilizar todas las posibilidades que ofrece el comando SELECT. Se pueden tener consultas bastante complejas en la creación de una vista.

De todos modos se debe prestar atención al tiempo de ejecución de la consulta y a las filas que se devuelven. Una vista no tiene índices ya que un SELECT de una vista recorre el conjunto de la vista. Una vista no debe sustituir a una tabla. Si los datos contenidos en una vista son muy numerosos y existe la necesidad de tener los datos archivados o se utiliza de manera recurrente, quizá habría que revisar el esquema de datos y crear una nueva tabla.

Para un usuario, la vista se comporta como una tabla. Puede realizar sus consultas sobre la vista igual que sobre una tabla. También es posible crear un conjunto de vistas por tipo de usuario o cargo en la empresa. Así cada uno tendrá los datos que necesita para usos concretos.

Sintaxis:

```
CREATE VIEW <Nombre_Vista> AS SELECT ...
```

Si se quieren todas las fechas de compra de los teléfonos de la empresa con la descripción del tipo y de la

marca, se hará un SELECT de las tres tablas y se recuperarán las columnas necesarias con este orden:

Ejemplo:

```
CREATE VIEW TEL_FECHA AS
SELECT TELEFONO.FECHA_COMPRA,TELEFONO.TIPO    ,TIPO_TEL.DESC_TIPO,
       TELEFONO.MARCA    ,MARCA_TEL.DESC_MARCA
FROM TELEFONO, TIPO_TEL, MARCA_TEL
WHERE TELEFONO.TIPO   = TIPO_TEL.TIPO AND
      TELEFONO.MARCA = MARCA_TEL.MARCA;
```

Este comando creará la vista TEL_FECHA que tendrá esta estructura.

Tabla TEL_FECHA

TEL_FECHA
FECHA_COMPRA TIPO DESC_TIPO MARCA DESC_MARCA

Cada vez que los datos de las tablas TELEFONO, TIPO_TEL y MARCA_TEL se modifiquen, la vista TEL_FECHA se actualizará automáticamente.

El acceso a la vista se hará igual que para una tabla clásica, con el comando SELECT.

Ejemplo:

Tabla TELEFONO

NUMERO	TIPO	MARCA	FECHA_COMPRA	PRECIO	NUM_PROPIETARIO	COLOR
1	SP	1	15/01/10	159	190120	ROJO
2	SP	2	14/03/10	99	190215	NEGRO
3	CT	3	02/05/10	49	190001	NEGRO
4	DE	4	25/07/10	89	190222	BLANCO
5	IP	5	30/09/10	359	190561	NEGRO

Tabla TIPO_TEL

TIPO	DESC_TIPO
SP	SMARTPHONE
CT	CON TAPA
DE	DESLIZANTE
IP	IPHONE
OT	OTRO

Tabla MARCA_TEL

MARCA	DESC_MARCA	PAIS
1	SAMSUNG	COREA
2	SONY	JAPÓN

3	PHILIPS	HOLANDA
4	MOTOROLA	USA
5	APPLE	USA

Teniendo en cuenta el contenido de las tablas, a continuación podemos ver que la vista TEL_FECHA contiene:

```
SELECT * FROM TEL_FECHA;
```

FECHA_COMPRA	TIPO	MARCA	TIPO	MARCA
15/01/10	SP	SMARTPHONE	1	SAMSUNG
14/03/10	SP	SMARTPHONE	2	SONY
02/05/10	CT	CON TAPA	3	PHILIPS
25/07/10	DE	DESLIZANTE	4	MOTOROLA
30/09/10	IP	IPHONE	5	APPLE

Almacenamiento de las vistas

La vista no se almacena físicamente sobre el disco, sino que se carga en memoria en su primera utilización. La recuperación de los datos de una vista es muy rápida a partir de la segunda ejecución. En términos de rendimiento, las vistas son una solución a tener en cuenta siempre que se utilice varias veces el mismo SELECT en una sesión.

Si se accede a menudo a estos datos, el acceso a una vista es más rápido que la ejecución de un SELECT sobre varias tablas.

No obstante, se debe consultar con los DBA que darán las normas de la empresa y buenas prácticas respecto a la utilización de las vistas.

Atención: algunos SGBDR ofrecen almacenar las vistas en disco. De este modo se comportan exactamente como las tablas y necesitan entonces otro análisis de rendimiento.

3. La eliminación de vistas

El comando DROP VIEW permite eliminar definitivamente una vista. La vista y su contenido se eliminan.

El comando DROP es definitivo, es imposible recuperar la vista si se ha eliminado por error.

Este comando no tiene ninguna influencia en las tablas que se utilizan en el SELECT de creación de la vista.

Sintaxis:

```
DROP VIEW nombre_vista;
```

Ejemplo:

```
DROP VIEW TEL_FECHA;
```

Los índices

En este capítulo abordaremos un concepto importante: los índices. Todas las bases de datos utilizan índices. La implementación física de estos difiere de un SGBDR a otro.

Existen varios tipos de índices y varios métodos para tratarlos. Veremos cómo crear y eliminar estos índices y porqué utilizar un tipo de índice u otro en función de las necesidades existentes.

1. Los índices y la norma SQL

En primer lugar hay que indicar que los índices no forman parte de la norma SQL. De hecho, el índice se utiliza para acelerar una búsqueda en una tabla y se basa en los ficheros físicos que se crean cuando se crea el índice.

Se trata pues, de una implementación física y en la norma SQL igual que ocurre con las tablas, no se trata la parte física. Cada SGBDR lo implementa a su manera.

Por contra, los índices son prácticamente indispensables en una base de datos relacional. El tiempo de acceso a los datos es un parámetro muy importante para todos los usuarios y desarrolladores, la utilización o no de un índice puede aumentar el tiempo de respuesta de forma exponencial.

En el caso de tablas con millones de filas, el acceso concreto a un dato puede durar varias horas sin índice, o algunos segundos con índice.

Sin índice, se recorrerá toda la tabla hasta encontrar el registro que se quiere consultar.

Es el SGBDR el que genera los ficheros de índice, el usuario no puede intervenir en cómo se almacenan.

No obstante, tenga en cuenta no crear índices para todas las columnas. Se debe crear un índice según el uso que los programas y usuarios hagan de la tabla. Los índices ralentizan los procesos de actualización, ya que el SGBDR debe recalcular las claves después de cada inserción, eliminación o modificación de filas.

Hay que fijarse en las columnas, analizar las actualizaciones de una tabla y basarse en un análisis funcional de los datos, y a continuación consultar con los DBA las normas de la empresa y el método que utiliza por defecto el SGBD.

2. Los diferentes métodos de organización de los índices

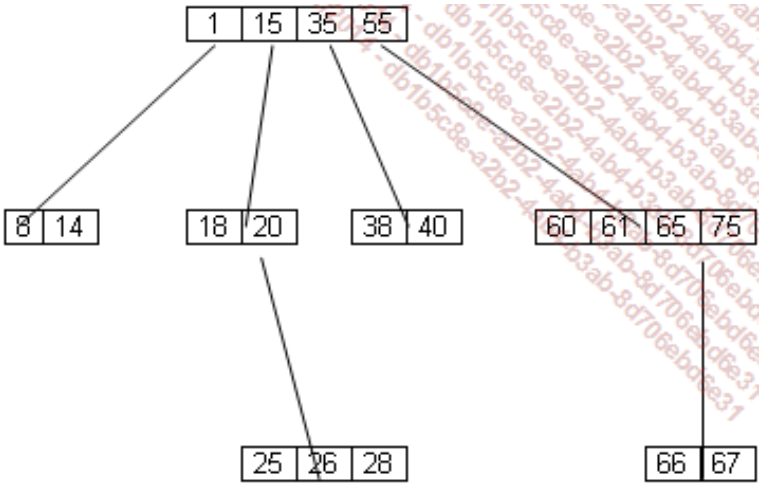
Existen cinco métodos principales de gestión de los índices: Hash, Secuencial, Bitmap, Árbol y Cluster.

Como desarrollador, es poco probable que tenga que elegir el método de indexación de las tablas. Es un tema que normalmente lo gestiona el DBA. No obstante, es bueno conocer su existencia para responder a problemáticas de rendimiento.

No vamos a entrar en el detalle de cada uno de estos métodos, los más utilizados son los índices en árbol (B*Tree) y los índices bitmap.

Por defecto, si no se especifica nada Oracle crea índices de tipo B*Tree. Este tipo de índice es conveniente para tablas voluminosas con claves únicas o con muy pocos duplicados (< 5 %).

*Ejemplo índice B*Tree*



*Representación Índice B*Tree*

Si se hace la siguiente consulta: SELECT * FROM <Tabla> WHERE Identificador = 18;

pasará directamente por el 15 y a continuación al 18.

Los índices de tipo bitmap se utilizan en tablas voluminosas que tienen muchas claves en común con una tasa de actualización baja. La base es crear para cada valor una cadena de bits y trabajar sólo con los bits.

Ejemplo índice bitmap

NUMERO	CLIENTE	MARCA	MOTOROLA	APPLE	SAMSUNG
1	Juan	MOTOROLA	1	0	0
2	Enrique	APPLE	0	1	0
3	Alex	APPLE	0	1	0
4	Valentín	SAMSUNG	0	0	1
5	Valeria	APPLE	0	1	0
6	Pedro	MOTOROLA	1	0	0

En la columna APPLE se encuentra la cadena 011010. A continuación el SGBD accede a los registros de forma muy rápida.

No olvide consultar a su DBA antes de crear cualquier índice.

3. La creación de un índice

El índice es el principal elemento que permite mejorar los tiempos de acceso a los datos. Por el contrario, se penaliza la actualización de los datos ya que el SGBDR deberá mantener los ficheros de índice en cada actualización.

En el caso de una base de datos que evoluciona poco, puede ser interesante multiplicar los índices para optimizar los tiempos de respuesta. Los índices se deben crear en columnas que tengan valores que distingan a los registros. El índice óptimo será, por ejemplo, el de una clave única.

No hay que confundir PRIMARY KEY y CREATE INDEX UNIQUE; la primera es una restricción de integridad que comprueba la unicidad de una columna (vea la sección La integridad de los datos); la segunda es una creación de fichero de índice utilizado para un mejor rendimiento.

La confusión puede venir del hecho que algunos SGBDR como Oracle crean automáticamente un índice cuando se crea una clave primaria.

La creación del índice se puede realizar sobre una o más columnas. Hay que elegir bien las columnas que lo

necesitan. Hay que analizar las consultas más utilizadas, las más costosas en tiempo, las claves primarias y foráneas…

No hay reglas establecidas para la creación de los índices. Se aconseja crear índices en columnas que se declaran como PRIMARY KEY (si el SGBDR no la crea por defecto), sobre las columnas FOREIGN KEY, sobre las columnas a las que más se acceda, las que actúen como unión entre tablas, las columnas más excluyentes, etc.

Si aparecen problemas de rendimiento después de la creación de un índice, hay que eliminarlo y comprobar la utilización que se hace de la tabla, sobre todo en las actualizaciones.

Sintaxis:

```
CREATE [UNIQUE] INDEX <nombre índice> ON <nombre tabla>
<nombre columna 1> [ASC|DESC], <nombre columna 2> [ASC|DESC], ... ...
```

Las opciones posibles son:

- UNIQUE que indica al SGBDR que esta columna será única y tendrá que controlar cuando haya una inserción que no haya duplicados. Es posible crear tantos UNIQUE INDEX como sean necesarios pero la PRIMARY KEY es única.
- ASC o DESC indica al SGBDR cómo se debe ordenar el índice, por defecto de modo ascendente. Esta opción puede ser útil si las consultas piden que se devuelva esa columna en un orden concreto (con ORDER BY). Si el índice se crea con la opción DESC y luego se utiliza un ORDER BY <columna> ASC, el índice no será eficaz.

Ejemplos:

```
CREATE INDEX I2_TIPO ON TELEFONO (TIPO);
CREATE INDEX I3_TIPMAR ON TELEFONO (TIPO, MARCA);

CREATE INDEX I4_COLOR ON TELEFONO (COLOR DESC);
```

Una columna se puede utilizar en varios índices. Es interesante dar un número a los índices, porque así se puede saber automáticamente cuántos hay en una tabla.

Cuando el índice está compuesto por varias columnas, es preferible poner la columna que más diferencie los registros la primera. Así por ejemplo, si el índice está compuesto por tres columnas y una consulta sólo comprueba la primera o las dos primeras columnas, el SGBDR utilizará parcialmente el índice.

Por regla general no es necesario indexar las tablas pequeñas. El análisis y el mantenimiento del índice serán más costosos que la lectura de la tabla completa.

Si la creación del índice se hace en una tabla grande, puede ser muy larga en términos de tiempos de respuesta. También hay que vigilar el espacio en disco, ya que el sistema creará ficheros para almacenar estos índices.

Ejemplo de consulta que permite visualizar los índices de una tabla con Oracle:

```
PROMPT .
PROMPT Índices de la tabla TELEFONO
PROMPT .
BREAK ON "INDEX" ON "SCRIPT"
SELECT index_name "INDEX",
       LOWER(column_name) "COLUMNA(S)"
FROM user_ind_columns
WHERE table_name = UPPER('TELEFONO')
ORDER BY index_name, column_position;
```

4. La eliminación de un índice

El comando DROP INDEX permite eliminar un índice.

El comando DROP es definitivo, será imposible recuperar el índice en caso de error. Un usuario sólo puede suprimir los índices que haya creado él mismo y a los que le haya autorizado el administrador.

Este comando no se puede utilizar para eliminar un índice creado por el sistema después de la creación de una clave primaria.

Sintaxis:

```
DROP INDEX <nombre_índice>
```

Ejemplo:

```
DROP INDEX I2_TIPO;
DROP INDEX I4_COLOR;
```

La integridad de los datos

Las restricciones de integridad permiten mantener la coherencia de la base de datos. Se confía al SGBDR las tareas de control de la validez de los datos que se insertan.

Las restricciones sustituyen a los controles realizados por los programas.

Existen varios tipos de controles. Es posible indicar al SGBDR:

- qué valor por defecto se va a asignar a una columna (DEFAULT),
- que una columna no pueda ser null (NOT NULL),
- que una columna deba ser única (UNIQUE),
- o codificar un control en una columna (CHECK).

Existen igualmente dos restricciones particulares que son la PRIMARY KEY y la FOREIGN KEY, vamos a detallar sus funciones.

1. La PRIMARY KEY

Por definición, la PRIMARY KEY es la clave primaria, es decir, la clave principal de una tabla. El SGBDR controlará en cada inserción o modificación que la clave sea única en la tabla. En caso contrario, rechaza la petición de modificación con un mensaje de error de tipo: «Violation constraint …».

La PRIMARY KEY siempre es una clave UNIQUE, compuesta de una o varias columnas en función del método de creación; lo más importante es que no puede haber dos filas de la tabla con la misma clave.

A menudo se trata de un número que se incrementa de uno en uno en cada inserción de una fila en la tabla.

También se pueden utilizar datos de empresa, como los números de la seguridad social o del permiso de conducir, pero es necesario que TODAS las filas de la tabla tengan un valor para esta o estas columnas. De hecho, una clave primaria no puede tomar el valor NULL.

La creación de una clave primaria genera en la mayor parte de los SGBDR la creación automática de un índice en esta columna.

Hay dos métodos para declarar una clave primaria. Si la clave corresponde a una sola columna, hay que utilizar esta sintaxis:

Ejemplo: declaración de PRIMARY KEY con una columna

```
CREATE TABLE TELEFONO (NUMERO       INTEGER PRIMARY KEY,
                       TIPO        VARCHAR(2),
                       MARCA        INTEGER,
                       FECHA_COMPRA DATE,
                       PRECIO       DECIMAL(9,2),
                       NUM_PROPIETARIO INTEGER,
                       COLOR        VARCHAR(25));
```

Automáticamente la columna número será NOT NULL y UNIQUE.

Si hay más columnas que componen la clave, habrá que utilizar la cláusula CONSTRAINT que permite declarar una restricción de integridad.

Ejemplo: declaración de PRIMARY KEY con varias columnas

Las

```
CREATE TABLE TELEFONO (NUMERO       INTEGER ,
                       TIPO         VARCHAR(2),
                       MARCA        INTEGER,
                       FECHA_COMPRA DATE,
                       PRECIO       DECIMAL(9,2),
                       NUM_PROPIETARIO INTEGER,
                       COLOR        VARCHAR(25),
CONSTRAINT PK_TELEFONO PRIMARY KEY (NUMERO,TIPO,MARCA));
```

columnas número, tipo y marca componen la clave. Serán NOT NULL y la asociación de las tres es UNIQUE.

Igualmente podemos remarcar que se da un nombre a la restricción, en este caso PK_TELEFONO. De esta forma,

cuando el SGBDR nos indique una inserción incorrecta en la tabla, hará referencia a la restricción explícitamente.

Las restricciones de clave primaria normalmente toman la notación PK_<nombre tabla> (PK por Primary Key); así el desarrollador o el usuario puede saber sólo leyendo el nombre de la restricción que el problema encontrado radica en la clave primaria de esa tabla.

En el caso en que la tabla ya esté creada y que haya que añadir una clave primaria, hay que utilizar ALTER TABLE.

Ejemplo: creación de una PRIMARY KEY en una tabla existente

```
ALTER TABLE TELEFONO ADD
CONSTRAINT PK_TELEFONO PRIMARY KEY (NUMERO,TIPO,MARCA);
```

2. La FOREIGN KEY

La FOREIGN KEY es la clave foránea, es decir, que se basa en otra tabla para indicar cómo controlar las columnas de nuestra tabla principal.

La tabla foránea debe tener una clave primaria para que el SGBDR pueda realizar el vínculo entre las dos tablas.

El principio es simple, en cada actualización de la tabla principal, el SGBDR comprueba que el valor de la columna en cuestión exista en la otra tabla.

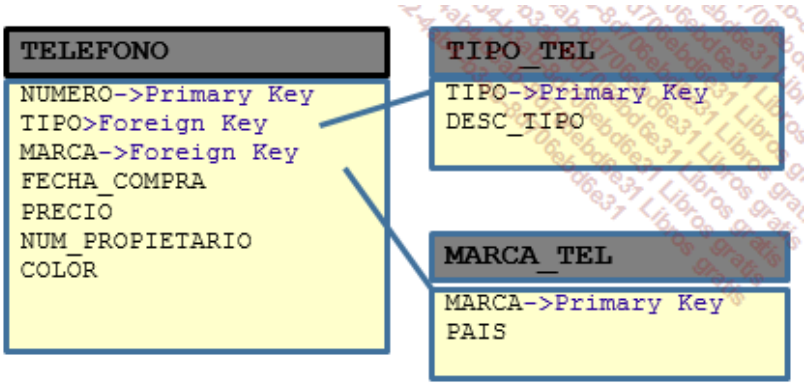
Es preferible que el nombre de la columna sea igual en las dos tablas. En caso contrario, el SGBDR cogerá la clave primaria de la tabla secundaria.

Es posible tener varias restricciones de claves foráneas en una misma tabla.

La utilización o no de FOREIGN KEY depende de las normas de desarrollo de la empresa. Se puede preferir controlar la integridad por programa y no dejar esta tarea a la base de datos.

En el caso de utilización de las FOREIGN KEY, sus programas deben controlar los códigos de retorno en cada actualización de la base de datos, para comprobar que no se viola ninguna restricción.

Ejemplo:



Sintaxis:

```
CONSTRAINT <nombre restricción> FOREIGN KEY (columna 1, columna 2 ...)
REFERENCES <tabla secundaria> );
```

En el ejemplo, en primer lugar hay que crear una clave primaria en la tabla TIPO_TEL y a continuación crear la clave foránea en la tabla TELEFONO.

Ejemplo de creación de una clave foránea:

```
ALTER TABLE TIPO_TEL ADD
CONSTRAINT PK_TELEFONO PRIMARY KEY (TIPO);

CREATE TABLE TELEFONO (NUMERO       INTEGER ,
                       TIPO         VARCHAR(2),
                       MARCA        INTEGER,
```

```
                        FECHA_COMPRA DATE,
                        PRECIO       DECIMAL(9,2),
                        NUM_PROPIETARIO INTEGER,
                        COLOR        VARCHAR(25),
CONSTRAINT FK_TIPO FOREIGN KEY (TIPO) REFERENCES TIPO_TEL);
```

3. Los valores por defecto (DEFAULT)

El usuario no tiene porqué llenar forzosamente todas las columnas al insertar la fila. El desarrollador puede entonces decidir asignar un valor por defecto a esta columna para no tener valores NULL en la base de datos.

En la siguiente sección de este mismo capítulo abordaremos la gestión de los valores NULL y veremos que es bastante particular.

Es preferible tener valores por defecto bien definidos para evitar obtener resultados de consultas erróneos.

La asignación de un valor por defecto se realiza durante la creación de la tabla o con un ALTER TABLE.

Es posible indicar diferentes valores por defecto:

- un valor numérico o carácter,
- resultados de una función como la fecha del día (CURRENT_DATE o SYSDATE) o la fecha y la hora del día (CURRENT_TIMESTAMP) útil sobre todo para las columnas que hacen el seguimiento de las actualizaciones realizadas en una tabla,
- para asignar el nombre del User que ha realizado la actualización con la función USER.

Para que el SGBDR tenga en cuenta un valor por defecto no se tiene que asignar ningún valor a la columna en el comando INSERT. La columna no debe aparecer en el comando. Si se indica un valor blanco o null, estos sustituyen al valor por defecto.

Ejemplo de valores por defecto:

En la creación de la tabla:

```
CREATE TABLE TELEFONO  (NUMERO       INTEGER PRIMARY KEY,
                        TIPO         VARCHAR(2) DEFAULT 'OT',
                        MARCA        INTEGER,
                        FECHA_COMPRA DATE DEFAULT CURRENT_DATE,
                        PRECIO       DECIMAL(9,2) DEFAULT 0,
                        NUM_PROPIETARIO INTEGER,
                        COLOR        VARCHAR(25));
```

Una vez creada la tabla, se puede modificar un valor por defecto de la siguiente forma:

```
ALTER TABLE TELEFONO MODIFY TIPO DEFAULT 'SP';
```

Añadir una columna y asignarle un valor por defecto:

```
ALTER TABLE TELEFONO ADD (USUARIO VARCHAR2(25) DEFAULT USER);
```

4. El valor NULL

En una inserción o modificación, si no se especifica el valor de una columna, esta está vacía y toma el valor NULL.

NULL no corresponde a espacios ni al valor cero, es un valor en sí mismo. Para utilizar una columna que contenga el valor NULL, hay que especificar IS NULL en el SELECT para probar el contenido.

Por ejemplo, si las filas de la tabla TELEFONO tienen el contenido:

NUMERO	TIPO	MARCA	FECHA_COMPRA	PRECIO	NUM_PROPIETARIO	COLOR
1	SP	1	15/01/10	159	190120	ROJO
2	SP	2	14/03/10	99	190215	
3	CT	3	02/05/10	49	190001	NEGRO
4	DE	4	25/07/10	89	190222	BLANCO

Las líneas 2 y 5 se han

5	IP	5	30/09/10	359	190561	

introducido sin especificar el valor para el COLOR, la columna se considera como NULL.

Si se realiza una selección de la columna como:

```
SELECT COUNT(*) FROM TELEFONO WHERE COLOR NOT IN
(ROJO',’NEGRO,’BLANCO’);
```

El resultado es 0. De hecho, las dos líneas que tienen los valores NULL no se tienen en cuenta, incluso si el color es diferente de los valores buscados.

Por el contrario, con la siguiente selección:

```
SELECT COUNT(*) FROM TELEFONO WHERE COLOR NOT IN
(’ROJO’,’NEGRO’,’BLANCO’) OR COLOR IS NULL;
```

El resultado será 2.

Hay que manejar con precaución las columnas que aceptan el valor NULL. Por precaución, es preferible poner todas las columnas que se utilizan en una cláusula WHERE en NOT NULL y asignar valores por defecto cuando se inserte o modifique una fila.

Todas las columnas que formen parte de una clave deben ser NOT NULL para evitar cualquier problema de indexación.

5. La cláusula UNIQUE

Como se indicó en la sección donde se explicaba el funcionamiento de las PRIMARY KEY, la cláusula UNIQUE permite indicar al SGBDR que el valor de esta columna no puede estar duplicado en la tabla.

Atención, una columna declarada UNIQUE puede contener NULL si no se especifica la cláusula NOT NULL al crearse. En este caso puede haber varias filas con la columna a NULL, ya que la unicidad no se aplica en este caso.

Ejemplo: añadir la unicidad en la creación de la tabla

```
CREATE TABLE TELEFONO (NUMERO       INTEGER PRIMARY KEY,
                       TIPO         VARCHAR(2) DEFAULT ’OT’,
                       MARCA        INTEGER,
                       FECHA_COMPRA DATE DEFAULT CURRENT_DATE,
                       PRECIO       DECIMAL(9,2) DEFAULT 0,
                       NUM_PROPIETARIO INTEGER UNIQUE,
                       COLOR        VARCHAR(25));
```

Modificar una columna:

```
ALTER TABLE TELEFONO MODIFY (NUM_PROPIETARIO UNIQUE)
```

6. CHECK

Esta cláusula permite realizar toda clase de controles en las columnas de una tabla. Hay que utilizarla con precaución ya que puede ser origen de problemas significativos de rendimiento. De hecho, en cada modificación de la columna, el comando se ejecuta y puede ralentizar sensiblemente las actualizaciones.

Se pueden controlar columnas con valores alfanuméricos o numéricos, llamando a una función o incluso especificando un SELECT concreto.

Atención: **la cláusula CHECK está implementada en MySQL pero no tiene ningún efecto**. No se tienen en cuenta estas restricciones. Hay que poner controles por programa o utilizar un TRIGGER. MySQL implementa la cláusula sólo por razones de compatibilidad con la norma. Oracle la implementa y la gestiona, pero no completamente, **no se puede llamar a otras tablas en forma sub-SELECT, por ejemplo**. Sólo gestiona controles con constantes, con columnas de la tabla y con funciones simples.

Ejemplo de controles posibles según la norma SQL92:

```
CREATE TABLE TELEFONO
```

El

```
(NUMERO       INTEGER PRIMARY KEY,
 TIPO         VARCHAR(2) CHECK (VALUE IN (SELECT TIPO FROM TIPO_TEL)),
 MARCA        INTEGER CHECK (VALUE BETWEEN 1 AND 99),
 FECHA_COMPRA DATE,
 PRECIO       DECIMAL(9,2) CHECK (VALUE > 0),
 NUM_PROPIETARIO INTEGER,
 COLOR        VARCHAR(25));
```

contenido de la columna TIPO debe estar en la tabla TIPO_TEL.

La columna MARCA debe tener un valor entre 1 y 99.

La columna PRECIO debe ser mayor que 0.

Con Oracle se puede crear la tabla con los controles en Marca y Precio pero no en Tipo con la siguiente sintaxis:

```
CREATE TABLE TELEFONO
(NUMERO       INTEGER PRIMARY KEY,
 TIPO         VARCHAR(2),
 MARCA        INTEGER CONSTRAINT MARCA_CTRL
                      CHECK (MARCA BETWEEN 1 AND 99),
 FECHA_COMPRA DATE,
 PRECIO       DECIMAL(9,2) CONSTRAINT PRECIO_CTRL  CHECK (PRECIO > 0),
 NUM_PROPIETARIO INTEGER,
 COLOR        VARCHAR(25));
```

7. Algunos consejos

Realizar un análisis detallado del contenido de cada tabla, de los vínculos entre las tablas de las claves únicas, de las columnas que no deben estar vacías, etc.

Es preferible dar nombres significativos a las tablas y dar nombres idénticos a las columnas que representen la misma cosa.

También se debe tener en cuenta los valores por defecto a asignar a las columnas, o las secuencias a implementar, etc.

Una reflexión más importante debe ser sobre qué tipos numéricos hay que utilizar. ¿El espacio en disco es un tema primordial? ¿Se manipulan datos financieros? ¿Cuál es el nivel de redondeo esperado? ¿Cuál es el valor máximo de la columna? ¿Necesito decimales? etc.

A continuación hay que analizar cómo los programas y/o los usuarios acceden a los datos. ¿Cuáles son los campos principales en las tablas? ¿Con qué columnas el usuario realiza sus consultas? ¿Cuántas filas de media recupera cada consulta?

En función del uso que se haga de la aplicación, debe decidir crear tablas con muchas columnas o bien, muchas tablas con pocas columnas. Esto depende también de la cantidad de memoria asignada a la base de datos, de los accesos que se realicen a las tablas, de la cantidad de columnas devueltas por las consultas, etc.

En cualquier caso siempre se debe consultar con los administradores para conocer las reglas comunes a respetar y las restricciones específicas en cada sitio.

También se debe consultar al director del proyecto para conocer posibles evoluciones en el esquema de datos que puedan estar previstas.

8. Ejercicios de aplicación

Primer ejercicio

A partir del contenido de esta tabla, escriba la sintaxis de creación de la tabla PELICULAS.

Cree una clave primaria con las columnas IDENT_PELICULA y un índice no único GENERO 1 vinculado a PAIS.

IDENT_ PELICULA	TITULO	GENERO1	RECAUDACION	FECHA_ ESTRENO	PAIS	NUM_ ENTRADAS	FECHA_ALTA
1	SUBWAY	POLICIACA	390.659,52	10/04/85	1	2.917.562	25/05/11 11:31
SINOPSIS	Cuenta las aventuras de la población subterránea en los túneles del metro de París.						

2	NIKITA	DRAMA	5.017.971,00	21/02/90	1	3.787.845	15/04/11 09:30
SINOPSIS	Nikita, condenada a cadena perpetua, es obligada a trabajar en secreto para el gobierno como agente de los servicios secretos.						
3	STAR WARS 6 - EL RETORNO DEL JEDI	ACCIÓN	191.648.000,00	19/10/83	2	4.263.000	01/01/10 08:00
SINOPSIS	El imperio galáctico es más poderoso que nunca: la construcción de la nueva arma, la Estrella de la Muerte, amenaza todo el universo.						

Segundo ejercicio

Añadir una columna llamada NUM_REAL.

Esta columna es una clave externa en la tabla REALIZADOR.

Añadir un valor por defecto en la columna RECAUDACION con el valor 0.

Poner las columnas TITULO y PAIS como NOT NULL.

Eliminar la restricción.

Tercer ejercicio

Crear una vista PELICULAS2 a partir de la tabla PELICULAS que contenga las cuatro primeras columnas de la tabla PELICULAS y la columna SINOPSIS.

Elimina esta vista.

Cuarto ejercicio

Crear una secuencia en la columna IDENT_PELICULA que comience en el 12 y con un valor máximo de 9999.

Renombrar la tabla PELICULAS a PELICULASOLD.

Crear una nueva tabla PELICULAS a partir de tabla PELICULASOLD.

Eliminar la tabla PELICULAS y la tabla PELICULASOLD.

9. Corrección de los ejercicios de aplicación

Primer ejercicio

Consulta en formato estándar:

```
DROP TABLE PELICULAS;
CREATE TABLE PELICULAS (IDENT_PELICULA    INTEGER PRIMARY KEY,
                        TITULO            VARCHAR(50),
                        GENERO1           VARCHAR(20),
                        RECAUDACION       DECIMAL(15,2),
                        FECHA_ESTRENO     DATE,
                        PAIS              SMALLINT,
                        NUM_ENTRADAS      INTEGER,
                        SINOPSIS          VARCHAR(2000),
                        FECHA_ALTA        TIMESTAMP
                        );
```

Descripción Oracle de tabla:

```
DESC PELICULAS
 Nombre                                    NULL ?   TIPO
 ----------------------------------------- -------- ---------------
```

```
 IDENT_PELICULA                            NOT NULL NUMBER(38)
 TITULO                                             VARCHAR2(50)
 GENERO1                                            VARCHAR2(20)
 RECAUDACION                                        NUMBER(15,2)
 FECHA_ESTRENO                                      DATE
 PAIS                                               NUMBER(38)
 NUM_ENTRADAS                                       NUMBER(38)
 SINOPSIS                                           VARCHAR2(2000)
 FECHA_ALTA                                         TIMESTAMP(6)
```

Se puede observar que la columna IDENT_PELICULA se transforma automáticamente en NOT NULL aunque no se haya especificado en el CREATE. Sólo con declararla PRIMARY KEY la transforma en NOT NULL.

Consulta de creación de índice:

```
CREATE INDEX I2_GENEROPAIS ON PELICULAS (GENERO1, PAIS);
```

Segundo ejercicio

Consulta para agregar una columna:

```
ALTER TABLE PELICULAS ADD (NUM_REAL INTEGER);
```

Añadir una clave foránea:

```
-- Creación de la tabla REALIZADOR
DROP TABLE REALIZADOR;

CREATE TABLE REALIZADOR
                 (NUM_REAL    INTEGER PRIMARY KEY,
                  NOMBRE            VARCHAR(50));

-- Añadir la restricción de integridad
ALTER TABLE PELICULAS ADD CONSTRAINT FK_REALIZADOR FOREIGN KEY
(NUM_REAL) REFERENCES REALIZADOR;
```

Añadir un valor por defecto en la columna RECAUDACION con el valor 0:

```
ALTER TABLE PELICULAS MODIFY RECAUDACION DEFAULT 0;
```

Transformar las columnas TITULO y PAIS en NOT NULL:

```
ALTER TABLE FILM MODIFY TITULO NOT NULL;
ALTER TABLE FILM MODIFY PAIS NOT NULL;
```

Eliminar la restricción:

```
ALTER TABLE PELICULAS DROP CONSTRAINT FK_REALIZADOR;
```

Descripción de la tabla PELICULAS en este momento:

```
DESC PELICULAS

 Nombre                                    NULL ?   Tipo
 ----------------------------------------- -------- --------------

 IDENT_PELICULA                            NOT NULL NUMBER(38)
 TITULO                                    NOT NULL VARCHAR2(50)
 GENERO1                                            VARCHAR2(20)
 RECAUDACION                                        NUMBER(15,2)
 FECHA_ESTRENO                                      DATE
 PAIS                                      NOT NULL NUMBER(38)
 NUM_ENTRADAS                                       NUMBER(38)
 SINOPSIS                                           VARCHAR2(2000)
```

```
FECHA_ALTA                                TIMESTAMP(6)
NUM_REAL                                  NUMBER(38)
```

Tercer ejercicio

Crear una vista PELICULAS2 a partir de la tabla PELICULAS que contenga las cuatro primeras columnas de la tabla PELICULAS así como la columna SINOPSIS.

```
CREATE VIEW PELICULAS2 AS
  SELECT IDENT_PELICULA, TITULO, GENERO1, SINOPSIS
  FROM PELICULAS;
```

Eliminar esta vista:

```
DROP VIEW PELICULAS2;
```

Cuarto ejercicio

Crear una secuencia en la columna IDENT_PELICULA que comience en 12 y con un valor máximo de 9999.

```
CREATE SEQUENCE S_PELICULAS START WITH 12 INCREMENT BY 1
MINVALUE 12 MAXVALUE 9999 CYCLE;
```

Renombrar la tabla PELICULAS como PELICULASOLD:

```
RENAME PELICULAS TO PELICULASOLD;
```

Crear una nueva tabla PELICULAS a partir de la tabla PELICULASOLD:

```
CREATE TABLE PELICULAS AS SELECT * FROM PELICULASOLD;
```

Eliminar la tabla PELICULAS y la tabla PELICULASOLD:

```
DROP TABLE PELICULAS;
DROP TABLE PELICULASOLD;
```

Introducción

El lenguaje de manipulación de datos permite a los usuarios y a los desarrolladores acceder a los datos de la base, modificar su contenido, insertar o eliminar filas.

Se basa en cuatro comandos básicos que son SELECT, INSERT, DELETE y UPDATE.

El administrador de la base de datos, que es el único que puede asignar o no los derechos, no siempre autoriza estos cuatro comandos.

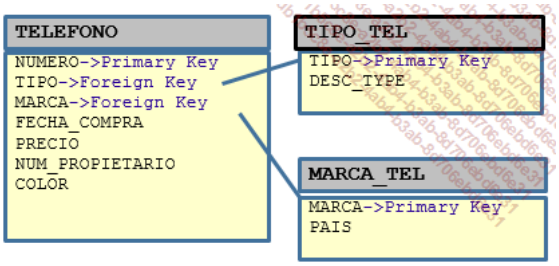
Para un usuario x, se podrá indicar que solo puede utilizar el comando SELECT. Por razones evidentes de seguridad, no todos los usuarios tendrán acceso a los comandos de modificación.

La selección de datos

El comando SELECT permite realizar consultas simples de forma rápida sin un conocimiento profundo del lenguaje de programación.

De todos modos, puede ser muy potente si se conocen todas las funciones y todas las posibilidades del lenguaje. Se pueden realizar consultas complejas, con numerosas tablas pero siempre hay que poner atención al rendimiento que puede disminuir rápidamente en un comando SQL mal construido o que no utilice los índices correctos en las tablas. Hay que vigilar y utilizar las herramientas de análisis de consultas (vea el capítulo Para ir más lejos - Algunos conceptos de rendimiento) antes de ejecutar una consulta sobre una base de datos real con tablas importantes.

Las tablas de base que se utilizan en los siguientes párrafos son:



Modelo de datos utilizado para ilustrar la selección de datos

1. El comando de selección de datos SELECT

El SELECT es el comando más importante y el más utilizado en SQL. Con este comando podemos recuperar las filas de una o más tablas y transformar los datos para su utilización o incluso realizar cálculos.

Vamos a describir poco a poco las posibilidades de este comando en los siguientes párrafos.

La utilización más normal consiste en seleccionar filas de una tabla:

```
SELECT FECHA_COMPRA, TIPO, MARCA FROM TELEFONO;
```

En este ejemplo, hemos seleccionado tres columnas de la tabla TELEFONO.

El comando nos va a devolver todas las filas de la tabla para estas tres columnas.

Si se hubieran querido todas las columnas y todas las filas de la tabla, el comando habría sido este:

```
SELECT * FROM TELEFONO;
```

El asterisco es práctico cuando no se conocen los nombres de las columnas, pero el resultado es raramente legible con tablas que contengan un gran número de columnas. Para saber el número de columnas, haga antes un DESC de la tabla (DESC <nombre tabla>).

NUMERO	TIPO	MARCA	FECHA_COMPRA	PRECIO	NUM_PROPIETARIO	COLOR
1	SP	1	15/01/10	159	190120	ROJO
2	SP	1	14/03/10	99	190215	
3	CT	3	02/05/10	49	190001	NEGRO
4	DE	4	25/07/10	89	190222	BLANCO
5	IP	5	30/09/10	359	190561	

La sintaxis es simple:

```
SELECT <columna 1>, <columna 2> ... | * FROM <tabla1>, <tabla2> ...
```

Si algunas columnas tienen el mismo nombre perteneciendo a tablas distintas, habrá que añadir el nombre de la tabla delante de la columna para que el sistema sepa qué columna debe utilizar.

No es obligatorio poner el nombre de las tablas delante de cada columna, pero por una cuestión de legibilidad y de mantenimiento es preferible ponerlas cuando se trata de selecciones complejas.

En la sección La utilización de los alias de este capítulo, veremos que para hacer la lectura más fácil podemos dar un alias a cada tabla. Este alias suele ser simple y permite encontrar fácilmente la tabla. Por ejemplo, TEL para TELEFONO o TIP para TIPO_TEL.

```
SELECT TELEFONO.FECHA_COMPRA,
       TELEFONO.TIPO,
       TELEFONO.MARCA,
       TIPO_TEL.DESC_TIPO,
       MARCA_TEL.DESC_MARCA
FROM TELEFONO,TIPO_TEL, MARCA_TEL;
```

2. Las opciones DISTINCT y ALL

Por defecto, en la ejecución de un SELECT se recuperan todas las filas (la opción ALL es automática). Si se quiere suprimir los duplicados hay que añadir la cláusula DISTINCT.

La cláusula DISTINCT se aplica a todas las columnas presentes en el comando.

Ejemplo:

```
SELECT TIPO, MARCA, PRECIO FROM TELEFONO;
```

y

```
SELECT DISTINCT TIPO, MARCA, PRECIO FROM TELEFONO;
```

Los dos SELECT anteriores tienen el mismo resultado aunque haya un duplicado en las dos primeras líneas. Las dos primeras columnas son iguales pero no la tercera.

TIPO	MARCA	PRECIO
SP	**1**	**159**
SP	**1**	**99**
DT	3	49
DE	4	89
IP	5	359

Por el contrario, si reducimos la selección a dos columnas:

Si se añade la cláusula DISTINCT, una de las filas que contienen 'SP' y '1' se eliminará.

```
SELECT TIPO, MARCA FROM TELEFONO;
```

TIPO	MARCA
SP	1

SP	1
CT	3
DE	4
IP	5

```
SELECT DISTINCT TIPO, MARCA FROM TELEFONO;
```

TIPO	MARCA
SP	1
CT	3
DE	4
IP	5

La cláusula DISTINCT no se puede utilizar con operadores de agrupamiento (ver GROUP BY). De hecho los operadores de tipo COUNT o SUM eliminan automáticamente los duplicados.

3. La utilización de los alias

En una consulta SQL que contenga varias tablas, es preferible asignar un diminutivo a cada nombre de tabla que se denomina alias. Siempre bajo la óptica de hacer que las consultas sean más legibles para todos los programadores.

Este alias se puede utilizar para un nombre de columna y también para un resultado de una función o un SELECT anidado.

Un alias se sitúa a continuación del elemento al que sustituirá.

Ejemplo de alias en una tabla:

```
SELECT TIPO, MARCA, PRECIO FROM TELEFONO TEL;
```

Cuando existen varias tablas, hay que dar un alias diferente para cada una de las tablas:

```
SELECT TEL.FECHA_COMPRA,
       TEL.MARCA,
       MAR.DESC_MARCA
FROM TELEFONO TEL, MARCA_TEL MAR;
```

El alias en una columna sustituye en la visualización el nombre inicial de la columna. Esto puede permitir dar un resultado más explícito para un usuario u ocultar el nombre real de la columna.

Ejemplo de alias en una columna:

```
SELECT TIPO C1, MARCA C2, PRECIO FROM TELEFONO TEL;
```

El alias también es interesante para poner nombre al resultado de un cálculo. Así el usuario comprenderá inmediatamente el contenido de esta columna.

Ejemplo de alias en un resultado de un cálculo:

```
SELECT TIPO C1, MARCA C2, PRECIO*166.386 PRECIO_PESETAS FROM TELEFONO
TEL;
```

El alias también se utiliza cuando dos columnas tienen el mismo nombre en tablas diferentes. Hay que indicar al sistema el origen de la columna.

Por ejemplo, la columna TIPO existe en la tabla TELEFONO y en la tabla TIPO_TEL. Es bastante lógico, ya que se trata de la FOREIGN KEY de la tabla TELEFONO que apunta a la tabla TIPO_TEL.

Si se quiere escribir una consulta que utilice estas dos tablas, la unión se realizará con esta clave, entonces es necesario diferenciar las dos columnas.

Ejemplo de alias para diferenciar dos columnas con el mismo nombre:

```
SELECT TEL.FECHA_COMPRA,
       TEL.TIPO,
       TIP.DESC_TIPO,
       TEL.MARCA,
       MAR.DESC_MARCA
FROM  TELEFONO TEL,
      TIPO_TEL TIP,
      MARCA_TEL MAR
WHERE TEL.TIPO   = TIP.TIPO
  AND TEL.MARCA = MAR.MARCA;
```

FECHA_COMPRA	TIPO	DESC_TIPO	MARCA	DESC_MARCA
15/01/10	SP	SMARTPHONE	1	SAMSUNG
14/03/10	SP	SMARTPHONE	2	SONY
02/05/10	CT	CON TAPA	3	PHILIPS
25/07/10	DE	DESLIZANTE	4	MOTOROLA
30/09/10	IP	IPHONE	5	APPLE

4. La cláusula WHERE

En los párrafos anteriores hemos visto cómo recuperar todas las filas de las tablas. Para poder seleccionar parte de las filas de las tablas hay que añadir una restricción con la cláusula WHERE.

Todo lo que se escribe después de esta cláusula afecta al resultado de la consulta.

También se puede utilizar con los comandos de actualización DELETE y UPDATE.

La condición puede ser muy sencilla como en este ejemplo:

```
SELECT TIPO, MARCA, PRECIO FROM TELEFONO WHERE TIPO = 'SP';
```

o mucho más compleja utilizando funciones o uniones.

El ejemplo siguiente utiliza la función EXISTS y la función IN que detallaremos en el capítulo Las funciones, en la sección Las funciones de comparación y de comprobación.

Ejemplo en Oracle:

En este caso, se comprueban las tres columnas, el TIPO debe ser

```
/* Selección de los teléfonos de tipo Smartphone o Con Tapa */
/* con un precio inferior a 200 euros y con una marca que exista */

SELECT TEL.TIPO, TEL.MARCA, TEL.PRECIO FROM TELEFONO
TEL
WHERE TEL.TIPO IN ('SP', 'CT') /* SP=Smartphone o CT=Con Tapa */
  AND TEL.PRECIO < 200
  AND EXISTS
     (SELECT MARCA FROM MARCA_TEL MAR WHERE MAR.MARCA =
TEL.MARCA);
```

'SP' o 'CT', el PRECIO debe ser inferior a 200 y la MARCA debe existir en la tabla MARCA_TEL.

Las restricciones detrás de una cláusula WHERE son prácticamente ilimitadas, tan sólo nos frenaría el rendimiento, que podría disminuir si las restricciones fueran con columnas no indexadas en tablas voluminosas.

Algunas veces es preferible hacer varios comandos SELECT simples y utilizar tablas temporales o vistas para almacenar los resultados intermedios en lugar de realizar una consulta compleja. Siempre hay que tener en cuenta que la persona que mantendrá los programas normalmente no es la que los escribe.

Como en el ejemplo anterior, hay que intentar escribir comandos legibles, bien identados y si es posible, comentados.

Con Oracle, para escribir comentarios hay que añadir /* al principio y */ al final del comentario, en una línea a parte o en una línea que contenga código SQL.

Con MySQL, hay que añadir -- o # en una línea vacía o en una línea con comandos SQL.

Ejemplo en MySQL:

```
-- Selección de teléfonos de tipo Smartphone o Con Tapa
-- con un precio inferior a 200 euros y una marca que exista

SELECT TEL.TIPO, TEL.MARCA, TEL.PRECIO
FROM TELEFONO TEL
WHERE TEL.TIPO IN ('SP', 'CT') # SP=Smartphone o CT=Con Tapa
  AND TEL.PRECIO < 200
  AND EXISTS
     (SELECT MARCA FROM MARCA_TEL MAR WHERE MAR.MARCA =
TEL.MARCA);
```

5. Las uniones

En una base de datos relacional, la unión es uno de los elementos esenciales que permite extraer datos de varias tablas aplicando condiciones en las columnas.

Si retomamos uno de los ejemplos anteriores y no se indica qué columnas permiten realizar la unión, el sistema realizará el producto cartesiano de las tablas y devolverá para cada fila de la primera tabla todas las filas del resto de tablas.

Si tenemos cinco filas en cada una de las tablas TELEFONO y TIPO_TEL, el sistema devolverá 5 x 5 = 25 filas.

Ejemplo de unión sin restricciones que devuelve el producto cartesiano del contenido de las dos tablas:

```
SELECT TEL.FECHA_COMPRA,
       TEL.TIPO,
       TIP.DESC_TIPO,
FROM TELEFONO TEL,TIPO_TEL TIP;
```

Hay que vincular las tablas entre ellas para obtener un resultado conforme a lo que esperamos.

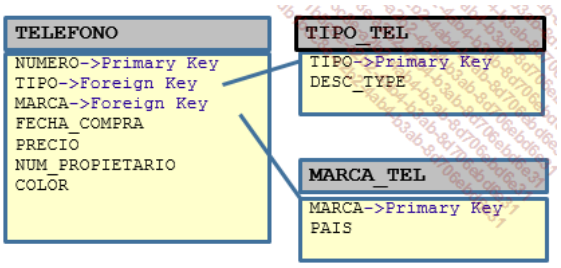
Ya hemos indicado en el capítulo La definición de los datos (LDD) que la tabla TELEFONO tenía dos FOREIGN KEY a TIPO_TEL y MARCA_TEL.

Vamos a utilizar estas claves para vincular las tres tablas. El objetivo es obtener por cada fila de la tabla TELEFONO la descripción del tipo de teléfono y la descripción de la marca del teléfono.

FECHA_COMPRA	TIPO	DESC_TIPO
15/01/10	SP	SMARTPHONE
15/01/10	SP	CON TAPA
15/01/10	SP	DESLIZANTE
15/01/10	SP	IPHONE
15/01/10	SP	OTRO
14/03/10	SP	SMARTPHONE
14/03/10	SP	CON TAPA
14/03/10	SP	DESLIZANTE
14/03/10	SP	IPHONE
14/03/10	SP	OTRO
02/05/10	CT	SMARTPHONE
02/05/10	CT	CON TAPA
02/05/10	CT	DESLIZANTE
02/05/10	CT	IPHONE
02/05/10	CT	OTRO
25/07/10	CO	SMARTPHONE
25/07/10	CO	CON TAPA
25/07/10	CO	DESLIZANTE
25/07/10	CO	IPHONE
25/07/10	CO	OTRO
...	...	...
...	...	...
...	...	...

a. La unión interna

Recordatorio de la descripción de nuestras tres tablas de muestra:



Modelo de datos utilizado para ilustrar la selección de datos

Hay que hacer la unión en la tabla TIPO_TEL con la columna TIPO y la unión en la tabla MARCA_TEL con la columna MARCA.

Ejemplo de una unión entre tres tablas:

```
SELECT TEL.FECHA_COMPRA,
       TEL.TIPO,
       TIP.DESC_TIPO,
       TEL.MARCA,
       MAR.DESC_MARCA
FROM TELEFONO TEL,
     TIPO_TEL TIP,
     MARCA_TEL MAR
WHERE TEL.TIPO  = TIP.TIPO AND
      TEL.MARCA = MAR.MARCA;
```

FECHA_COMPRA	TIPO	DESC_TIPO	MARCA	DESC_MARCA
15/01/10	SP	SMARTPHONE	1	SAMSUNG
14/03/10	SP	SMARTPHONE	2	SONY
02/05/10	CT	CON TAPA	3	PHILIPS
30/09/10	IP	IPHONE	5	APPLE

Resultado:

Existe otra notación para obtener el mismo resultado utilizando el comando JOIN que está definido en las normas SQL. La norma recomienda trabajar con esta orden para distinguir bien las condiciones de las uniones. De hecho, en un WHERE, a veces es difícil diferenciar las restricciones de las uniones.

Ejemplo de una unión con tres tablas con el comando JOIN:

```
SELECT TEL.FECHA_COMPRA,
       TEL.TIPO,
       TIP.DESC_TIPO,
       TEL.MARCA,
       MAR.DESC_MARCA
FROM TELEFONO TEL
     INNER JOIN TIPO_TEL TIP  ON TEL.TIPO  = TIP.TIPO
     INNER JOIN MARCA_TEL MAR ON TEL.MARCA = MAR.MARCA;
```

La palabra INNER es opcional, indica que se trata de una unión interna, la unión por defecto cuando no se especifica otra cosa.

El operador ON precede al criterio de unión que se utilizará.

En este caso, tenemos dos uniones con la tabla TELEFONO, después de cada cláusula JOIN se indica la tabla con la que se realiza la unión.

Otra dificultad que podríamos encontrar en la utilización de uniones sería el hecho de suponer que todas las filas de la tabla principal (TELEFONO) tienen correspondencia en las dos tablas secundarias (TIPO_TEL y MARCA_TEL).

Si, por ejemplo, el tipo 'DE' no existe en la tabla TIPO_TEL la fila de la tabla TELEFONO que tiene el tipo 'DE' (en el ejemplo la fila 4), no será devuelta por la consulta.

Resultado:

```
SELECT TEL.FECHA_COMPRA,
       TEL.TIPO,
       TIP.DESC_TIPO,
       TEL.MARCA,
       MAR.DESC_MARCA
FROM TELEFONO TEL
JOIN TIPO_TEL TIP  ON TEL.TIPO  = TIP.TIPO
JOIN MARCA_TEL MAR ON TEL.MARCA = MAR.MARCA;
```

FECHA_COMPRA	TIPO	DESC_TIPO	MARCA	DESC_MARCA
15/01/10	SP	SMARTPHONE	1	SAMSUNG
14/03/10	SP	SMARTPHONE	2	SONY
02/05/10	CT	CON TAPA	3	PHILIPS
30/09/10	IP	IPHONE	5	APPLE

La fila 4 de la tabla TELEFONO no aparece aunque está en la tabla. Por defecto, si la unión no se verifica, el sistema no devuelve la fila.

Existen varios tipos de unión: interna, externa o natural.

b. La unión externa

Para devolver todas las filas de la tabla principal, hay que indicarlo al SGBDR aunque no haya correspondencia con las tablas secundarias. En este caso, el sistema devuelve las columnas sin correspondencia a NULL.

Ejemplo con JOIN en el que hay que añadir la palabra **LEFT** (o RIGHT) para indicar que se quieren todas las filas de la tabla a la izquierda de la palabra JOIN, en este caso TELEFONO.

Si se quisieran las filas de la tabla TIPO_TEL o MARCA_TEL se indicaría con la palabra **RIGHT**. Las tablas están a la derecha de la palabra JOIN.

*Si se quieren a la vez las filas de las tablas de la derecha y de la izquierda, hay que utilizar la palabra**FULL** en lugar de RIGHT o LEFT.*

```
SELECT TEL.FECHA_COMPRA,
       TEL.TIPO,
       TIP.DESC_TIPO,
       TEL.MARCA,
       MAR.DESC_MARCA
FROM TELEFONO TEL
   LEFT OUTER JOIN TIPO_TEL TIP  ON TEL.TIPO  = TIP.TIPO
   LEFT OUTER JOIN MARCA_TEL MAR ON TEL.MARCA = MAR.MARCA;
```

FECHA_COMPRA	TIPO	DESC_TIPO	MARCA	DESC_MARCA
15/01/10	SP	SMARTPHONE	1	SAMSUNG
14/03/10	SP	SMARTPHONE	2	SONY
02/05/10	CT	CON TAPA	3	PHILIPS
30/09/10	IP	IPHONE	5	APPLE

Resultado:

La palabra OUTER es opcional e indica que se realiza una unión externa, lo contrario que la palabra INNER vista anteriormente.

La misma consulta con FULL en lugar de LEFT devolverá además las filas de las tablas TIPO_TEL y MARCA_TEL que no estén en TELEFONO. Por ejemplo la fila 'OTRO' en la tabla TIPO_TEL no se utiliza en TELEFONO.

Ejemplo con la opción FULL:

Resultado con la opción FULL:

En la cuarta fila, falta la descripción del tipo ya que el tipo 'DE' no existe en la tabla TIPO_TEL.

La última fila viene de TIPO_TEL y no tiene correspondencia en las otras tablas pero se muestra sólo la descripción indicada. No hay correspondencia en TELEFONO y la consulta no puede mostrar FECHA_COMPRA o MARCA.

```
SELECT TEL.FECHA_COMPRA,
       TEL.TIPO,
```

```
       TIP.DESC_TIPO,
       TEL.MARCA,
       MAR.DESC_MARCA
FROM TELEFONO TEL
   FULL OUTER JOIN TIPO_TEL TIP  ON TEL.TIPO  = TIP.TIPO
   FULL OUTER JOIN MARCA_TEL MAR ON TEL.MARCA = MAR.MARCA;
```

Otra posibilidad sin utilizar la palabra JOIN sólo en Oracle, es añadir el carácter «(+)» después de la columna de la tabla secundaria en cuestión.

Ejemplo:

FECHA_COMPRA	TIPO	DESC_TIPO	MARCA	DESC_MARCA
15/01/10	SP	SMARTPHONE	1	SAMSUNG
14/03/10	SP	SMARTPHONE	2	SONY
02/05/10	CT	CON TAPA	3	PHILIPS
25/07/10	DE		4	MOTOROLA
30/09/10	IP	IPHONE	5	APPLE
		OTRO		

En este caso la descripción del tipo de teléfono DESC_TIPO es NULL.

Funcionalmente, el ejemplo no tiene mucho sentido, pero es importante definir en la construcción de la base de datos las reglas, las claves y las relaciones entre las tablas para evitar este tipo de problemas.

En este caso concreto, sería preferible prohibir la creación de una fila en la tabla TELEFONO si el TIPO de teléfono no existe en la tabla TIPO_TEL. Se puede hacer por programa o con una restricción.

```
SELECT TEL.FECHA_COMPRA,
       TEL.TIPO,
       TIP.DESC_TIPO,
       TEL.MARCA,
       MAR.DESC_MARCA
FROM TELEFONO TEL,TIPO_TEL TIP,MARCA_TEL MAR
WHERE TEL.TIPO     = TIP.TIPO(+) AND
      TEL.MARCA    = MAR.MARCA;
```

FECHA_COMPRA	TIPO	DESC_TIPO	MARCA	DESC_MARCA
15/01/10	SP	SMARTPHONE	1	SAMSUNG
14/03/10	SP	SMARTPHONE	2	SONY
02/05/10	CT	CON TAPA	3	PHILIPS
25/07/10	DE		4	MOTOROLA
30/09/10	IP	IPHONE	5	APPLE

c. La unión natural

La unión natural deja al sistema asociar las columnas que tienen el mismo nombre. En este ejemplo se puede no indicar que las relaciones se hacen con TIPO y MARCA. La sintaxis difiere entre los diferentes SGBDR.

No se aconseja utilizar este tipo de unión, sobre todo por razones de legibilidad. También hay riesgo de que dos columnas tengan el mismo nombre sin pertenecer a una clave, y así el resultado no será el que queremos.

Además, el programador debe conocer perfectamente la estructura de estas tablas y no deben añadirse columnas a las tablas en el futuro.

Ejemplo de unión natural en Oracle:

```
SELECT FECHA_COMPRA,
       TIPO,
       DESC_TIPO,
       MARCA,
       DESC_MARCA
FROM TELEFONO NATURAL JOIN TIPO_TEL
              NATURAL JOIN MARCA_TEL;
```

FECHA_COMPRA	TIPO	DESC_TIPO	MARCA	DESC_MARCA
15/01/10	SP	SMARTPHONE	1	SAMSUNG
14/03/10	SP	SMARTPHONE	2	SONY
02/05/10	CT	CON TAPA	3	PHILIPS
30/09/10	IP	IPHONE	5	APPLE

Atención, con Oracle, no se pueden utilizar alias con este tipo de unión.

Ejemplo de unión natural en MySQL:

```
SELECT TEL.FECHA_COMPRA,
       TEL.TIPO,
       TIP.DESC_TIPO,
       TEL.MARCA,
       MAR.DESC_MARCA
FROM TELEFONO TEL NATURAL JOIN TIPO_TEL TIP
                  NATURAL JOIN MARCA_TEL MAR;
```

Es preferible utilizar el operador ON para indicar claramente las columnas que se utilizan para la unión a fin de evitar cualquier riesgo de unión inadecuada.

d. La unión cruzada

Es la unión más sencilla que existe. No se indica ningún criterio de vinculación y el sistema devuelve el producto cartesiano de las dos tablas.

Se puede utilizar un simple FROM o añadir la palabra 'CROSS'.

```
SELECT TEL.FECHA_COMPRA,
       TEL.MARCA,
       MAR.DESC_MARCA
FROM TELEFONO TEL, MARCA_TEL MAR;
```

o

Resultados:

```
SELECT TEL.FECHA_COMPRA,
       TEL.MARCA,
       MAR.DESC_MARCA
FROM TELEFONO TEL CROSS JOIN MARCA_TEL MAR;
```

FECHA_COMPRA	MARCA	DESC_MARCA

15/01/10	1	SAMSUNG
15/01/10	1	SONY
15/01/10	1	PHILIPS
15/01/10	1	MOTOROLA
15/01/10	1	APPLE
14/03/10	2	SAMSUNG
14/03/10	2	SONY
14/03/10	2	PHILIPS
14/03/10	2	MOTOROLA
14/03/10	2	APPLE
02/05/10	3	SAMSUNG
02/05/10	3	SONY
02/05/10	3	PHILIPS
02/05/10	3	MOTOROLA
02/05/10	3	APPLE
25/07/10	4	SAMSUNG
25/07/10	4	SONY
25/07/10	4	PHILIPS
25/07/10	4	MOTOROLA
25/07/10	4	APPLE
30/09/10	5	SAMSUNG
30/09/10	5	SONY
30/09/10	5	PHILIPS
30/09/10	5	MOTOROLA
30/09/10	5	APPLE

e. Sintaxis de las diferentes formas de unión

Unión interna

```
SELECT <columna 1>, <columna 2>, etc ...
FROM <tabla izquierda> [INNER] JOIN <tabla derecha 1> ON <criterio 1>,
                       [INNER] JOIN <tabla derecha 2> ON <criterio 2>,
                        etc ...
```

o

Unión externa

```
SELECT <columna 1>, <columna 2>, etc ...
FROM <tabla izquierda> [LEFT|RIGHT|FULL] [OUTER] JOIN <tabla derecha
1> ON <criterios>,
                    [LEFT|RIGHT|FULL] [OUTER] JOIN <tabla derecha
2> ON <criterios>,
                     etc ...
```

Unión natural

```
SELECT <columna 1>, <columna 2>, etc ...
FROM <tabla izquierda> NATURAL JOIN <tabla derecha 1>,
                       NATURAL JOIN <tabla derecha 2>,
                       etc ...
```

Unión cruzada

```
SELECT <columna 1>, <columna 2>, etc ...
FROM <tabla izquierda> CROSS JOIN <tabla derecha 1>,
                       CROSS JOIN <tabla derecha 2>,
                       etc ...
```

o

```
SELECT <columna 1>, <columna 2>, etc ...
FROM <tabla izquierda>, <tabla derecha 1>, <tabla derecha 2> etc ...
WHERE <criterio 1> AND <criterio 2> etc ...
```

6. La ordenación (ORDER BY)

Cuando se devuelven columnas de una o más tablas con un SELECT, a menudo es interesante obtener el resultado ordenado por ciertas columnas.

Para ello, se utilizará la cláusula ORDER BY al final de la consulta. Se puede ordenar por cualquier columna de una tabla, pero deben formar parte de la selección.

La cláusula sólo se puede utilizar una vez en una consulta y siempre debe ser la última de la consulta.

La ordenación por defecto es ascendente, se indica con la palabra ASC (del más pequeño al más grande); es posible indicar que se quiere realizar la ordenación de modo descendente con la palabra DESC.

Sintaxis del comando ORDER BY:

```
ORDER BY <columna 1> [ASC|DESC], <columna 2> [ASC|DESC]...
```

Ejemplo:

Ordenación por TIPO de teléfono de la tabla TELEFONO ascendentemente y ordenación por la descripción de la marca de la tabla MARCA_TEL descendentemente.

```
SELECT TEL.FECHA_COMPRA,
       TEL.TIPO,
```

Se puede comprobar que para las dos últimas filas que tienen el tipo 'SP' la ordenación se hace por la columna DESC_MARCA del más grande al más pequeño (DESC).

```
       TIP.DESC_TIPO,
       TEL.MARCA,
       MAR.DESC_MARCA
FROM TELEFONO TEL JOIN TIPO_TEL TIP  ON TEL.TIPO =
TIP.TIPO
                  JOIN MARCA_TEL MAR ON TEL.MARCA =
MAR.MARCA
ORDER BY TEL.TIPO ASC, MAR.DESC_MARCA DESC;
```

Igualmente existe la posibilidad de indicar el orden de la columna en el SELECT en lugar de su nombre.

Ejemplo sin el nombre de las columnas:

FECHA_COMPRA	TIPO	DESC_TIPO	MARCA	DESC_MARCA
02/05/10	CT	CON TAPA	3	PHILIPS
25/07/10	DE	DESLIZANTE	4	MOTOROLA
30/09/10	IP	IPHONE	5	APPLE
14/03/10	SP	SMARTPHONE	2	SONY
15/01/10	SP	SMARTPHONE	1	SAMSUNG

```
SELECT TEL.FECHA_COMPRA,
       TEL.TIPO,
       TIP.DESC_TIPO,
       TEL.MARCA,
       MAR.DESC_MARCA
FROM TELEFONO TEL JOIN TIPO_TEL TIP  ON TEL.TIPO =
TIP.TIPO
                  JOIN MARCA_TEL MAR ON TEL.MARCA =
MAR.MARCA
ORDER BY 2 ASC, 5 DESC;
```

Aunque esta notación funciona y permite no tener que volver a escribir los nombres de las columnas, no ayuda a la legibilidad de la consulta. Con selecciones múltiples, el orden se vuelve ilegible para el que no haya escrito el comando.

Atención:

La ordenación puede ser causa del bajo rendimiento de la consulta. De hecho, si la consulta devuelve millones de filas, el sistema en primer lugar recupera todas las filas, luego las ordena todas y por último las retorna en el orden establecido. El sistema utilizará espacio en disco para almacenar los elementos a ordenar además de la memoria para hacer la ordenación.

Se desaconseja ordenar en el caso de que no haya cláusula WHERE o que esta sea poco restrictiva y que la tabla contenga millones de registros.

Tenga también en cuenta que el rendimiento será mejor si ordena por columnas indexadas.

7. Los agrupamientos (GROUP BY)

Esta cláusula permite eliminar los duplicados en las filas devueltas y realizar cálculos con las columnas seleccionadas.

La utilización de funciones matemáticas no es obligatoria con un GROUP BY, por contra es posible un GROUP BY en una columna sin hacer cálculos. En este caso, el GROUP BY sustituye al DISTINCT. De hecho, sólo se devolvería un valor por columna.

Ejemplo:

```
SELECT DISTINCT TIPO FROM TELEFONO;
```

es equivalente a:

```
SELECT TIPO FROM TELEFONO GROUP BY TIPO;
```

En el comando SELECT habrá, por una parte, las columnas sobre las que se quiere hacer el agrupamiento y las columnas sobre las que se hará el cálculo.

La cláusula GROUP BY permite agrupar las filas por valor y realizar cálculos sobre las columnas.

Esta cláusula se puede utilizar con las funciones SUM, COUNT, AVG, MAX o MIN que son las más conocidas. También existen VARIANCE y STDDEV para calcular la desviación estándar, pero son menos utilizadas.

Por ejemplo, para saber el número de teléfonos presentes en la tabla por TIPO, la columna de agrupamiento será TIPO y el sistema contará el número de filas presentes en la tabla para cada tipo diferente.

Otro ejemplo, para saber el precio acumulado por TIPO de teléfono de la base de datos. El sistema acumulará en una variable los diferentes precios por tipo de teléfono.

Los siguientes ejemplos se basan en una tabla TELEFONO que contiene los siguientes elementos:

NUMERO	TIPO	MARCA	FECHA_COMPRA	PRECIO	NUM_PROPIETARIO	COLOR
1	SP	1	15/01/10	159	190120	ROJO
2	SP	2	14/03/10	99	190215	
3	CT	3	02/05/10	49	190001	NEGRO
4	DE	4	25/07/10	89	190222	BLANCO
5	IP	5	30/09/10	359	190561	
6	DE	6	01/01/09	99	122120	BLANCO

Ejemplo 1: número de teléfonos por tipo, ordenado por tipo

En este ejemplo los teléfonos 'DESLIZANTE' y 'SMARTPHONE' tienen cada uno dos elementos en la tabla, y sólo hay un elemento para los otros dos tipos ('CON TAPA' e 'IPHONE').

```
SELECT TEL.TIPO, TIP.DESC_TIPO, COUNT(*) AS NUM_TEL
FROM TELEFONO TEL JOIN TIPO_TEL TIP   ON TEL.TIPO =
TIP.TIPO
GROUP BY  TEL.TIPO, TIP.DESC_TIPO
ORDER BY TEL.TIPO ASC;
```

TIPO	DESC_TIPO	NUM_TEL
CT	CON TAPA	1
DE	DESLIZANTE	2
IP	IPHONE	1
SP	SMARTPHONE	2

Observe que el resultado del COUNT se asocia al nombre de columna NUM_TEL con la cláusula « AS ».

Ejemplo 2: precio acumulado por tipo, ordenado por tipo

```
SELECT TEL.TIPO, TIP.DESC_TIPO, SUM(PRECIO) AS COSTE
FROM TELEFONO TEL JOIN TIPO_TEL TIP   ON TEL.TIPO =
TIP.TIPO
GROUP BY TEL.TIPO, TIP.DESC_TIPO
ORDER BY TEL.TIPO ASC;
```

En este ejemplo, teníamos los precios de 89 y 99 para los 'DESLIZANTE', 89+99 = 188. Lo mismo para los 'SMARTPHONE', 159+99=258.

Los otros dos son únicos en la tabla, así que se muestra su precio, 49 para 'CON TAPA' y 359 para 'IPHONE'.

TIPO	DESC_TIPO	COSTE
CT	CON TAPA	49
DE	DESLIZANTE	188
IP	IPHONE	359
SP	SMARTPHONE	258

8. Las funciones utilizadas en un agrupamiento

Acabamos de ver el comando GROUP BY que se utiliza con funciones específicas. A continuación detallaremos las diferentes funciones vinculadas al comando GROUP BY.

Tenga en cuenta que estas funciones se pueden utilizar sin el GROUP BY; en este caso se aplican a todas las filas devueltas por el SELECT. No se pueden tener columnas simples y funciones de cálculo en una consulta sin tener una función GROUP BY.

Ejemplo de función de cálculo sin agrupamiento:

```
SELECT COUNT(*) FROM TELEFONO;
```

El resultado es 6, que corresponde al número de filas de la tabla TELEFONO.

```
SELECT SUM(PRECIO) FROM TELEFONO;
```

El resultado es 854, que corresponde al total de los precios de la tabla TELEFONO.

a. COUNT (contar filas)

Esta función permite contabilizar un número de filas respecto a los criterios solicitados.

Se escribe COUNT (*) o COUNT(<nombre de columna>). El resultado es un valor numérico. La notación « * » indica que se tienen en cuenta todas las filas, sin tener en cuenta el contenido. Por el contrario, si se indica un nombre de columna en el COUNT, solo se contabilizarán las filas cuyo valor no sea NULL en la columna.

Por ejemplo, para saber el número de filas de la tabla TELEFONO que tienen el mismo precio:

```
SELECT TEL.PRECIO, COUNT(*) AS NUM_PRECIO
FROM TELEFONO TEL
GROUP BY TEL.PRECIO
ORDER BY TEL.PRECIO ASC;
```

Se puede comprobar que el precio 99 está dos veces en la tabla y que el resto de precios sólo están una vez.

También se pueden poner restricciones, por ejemplo:

PRECIO	NUM_PRECIO
49	1
89	1
99	2
159	1
359	1

```
SELECT TEL.PRECIO, COUNT(*) AS NUM_PRECIO
FROM TELEFONO TEL
WHERE TEL.PRECIO < 300 AND TEL.TIPO IN ('CT','SP')
GROUP BY TEL.PRECIO
ORDER BY TEL.PRECIO ASC;
```

Se puede comprobar que el precio 359 no está presente (es superior a 300) y que la fila que tiene el precio 49 tampoco está ya que es de tipo 'CT'.

PRECIO	NUM_PRECIO
89	1
99	2
159	1

Caso particular de COUNT DISTINCT: permite contar el número de valores diferentes en la base de datos. Si se quiere saber el número de colores de teléfonos diferentes en la tabla, escribiremos:

```
SELECT COUNT(DISTINCT TEL.COLOR) AS NUM_TEL
FROM TELEFONO TEL;
```

El resultado es 3. Las filas que no tienen valor en la columna color no se tienen en cuenta. Se ignora el valor NULL.

```
SELECT COUNT(TEL.COLOR) AS NUM_TEL
FROM TELEFONO TEL;
```

Si quitamos la cláusula DISTINCT, el resultado es 4. Hay cuatro filas que tienen valor en la columna COLOR.

```
SELECT COUNT(*) AS NUM_TEL
FROM TELEFONO TEL;
```

Si se sustituye TEL.COLOR por * el resultado es 6. De hecho, en este caso se cuentan el número de filas de la tabla, independientemente de la columna COLOR.

b. SUM (suma de valores)

Esta función permite acumular los valores de una columna. Se aplica sólo a columnas de tipo numérico.

Para calcular la suma de los precios por tipo, escribiremos:

```
SELECT TEL.COLOR, SUM(TEL.PRECIO) AS PRECIO_ACUMULADO
FROM TELEFONO TEL JOIN TIPO_TEL TIP   ON TEL.TIPO =
TIP.TIPO
GROUP BY TEL.COLOR
ORDER BY TEL.COLOR ASC;
```

La última fila representa las filas de la tabla TELEFONO que no tienen valor en la columna COLOR.

Para saber la suma total hay que escribir:

COLOR	PRECIO_ACUMULADO
BLANCO	188
NEGRO	49
ROJO	159
	458

```
SELECT SUM(TEL.PRECIO) AS PRECIO_ACUMULADO
FROM TELEFONO TEL;
```

c. MAX y MIN (valores máximo y mínimo)

Estas dos funciones permiten recuperar el valor más grande y el valor más pequeño de una columna.

Por ejemplo, encontrar el teléfono más caro:

```
SELECT MAX(TEL.PRECIO) AS PRECIO_MAX
FROM TELEFONO TEL;
```

o encontrar el teléfono más caro y el menos caro por TIPO:

```
SELECT TEL.TIPO, TIP.DESC_TIPO, MIN(TEL.PRECIO) AS PRECIO_MIN,
                            MAX(TEL.PRECIO) AS PRECIO_MAX,
                            COUNT(*) AS NUM_TEL
FROM TELEFONO TEL JOIN TIPO_TEL TIP ON TEL.TIPO =
TIP.TIPO
GROUP BY TEL.TIPO, TIP.DESC_TIPO
ORDER BY TEL.TIPO ASC;
```

TIPO	DESC_TIPO	PRECIO_MIN	PRECIO_MAX	NUM_TEL
CT	CON TAPA	49	49	1
DE	DESLIZANTE	89	99	2
IP	IPHONE	359	359	1
SP	SMARTPHONE	99	159	2

En este resultado, vemos que dos tipos de teléfono tienen un precio mínimo diferente al precio máximo, el 'DE' y el 'SP', y que están dos veces cada uno en la tabla.

Los tipos 'CT' e 'IP' sólo tienen una fila en la tabla y por tanto, sólo un precio.

Combinando las funciones de cálculo, las posibilidades son muy grandes.

Igualmente se puede considerar la creación de vistas que contengan resultados de cálculo con fines estadísticos, por ejemplo. De este modo, la información está disponible inmediatamente.

Cálculo de la media del precio de un teléfono por tipo: en este ejemplo, se suman todos los precios y se divide por el número filas por tipo.

Así, se suman todos los precios (SUM) y a continuación se divide por el número de filas (COUNT).

Existe una función que calcula directamente el promedio: se trata de AVG.

```
SELECT TEL.TIPO, TIP.DESC_TIPO, MIN(TEL.PRECIO) AS PRECIO_MIN,
                                MAX(TEL.PRECIO) AS PRECIO_MAX,
                                SUM(TEL.PRECIO)/COUNT(*) AS
PRECIO_MEDIO
FROM TELEFONO TEL JOIN TIPO_TEL TIP   ON TEL.TIPO =
TIP.TIPO
GROUP BY TEL.TIPO, TIP.DESC_TIPO
ORDER BY TEL.TIPO ASC;
```

TIPO	DESC_TIPO	PRECIO_MIN	PRECIO_MAX	PRECIO_MEDIO
CT	CON TAPA	49	49	49
DE	DESLIZANTE	89	99	94
IP	IPHONE	359	359	359
SP	SMARTPHONE	99	159	129

d. AVG (promedio)

Esta función permite calcular el promedio de una serie de cifras.

Por supuesto, se aplica a una columna de tipo numérico.

Por ejemplo, para saber el precio medio de los teléfonos por tipo como en el ejemplo anterior, basta con sustituir el SUM/COUNT por un AVG para obtener el mismo resultado.

```
SELECT TEL.TIPO, TIP.DESC_TIPO, MIN(TEL.PRECIO) AS PRECIO_MIN,
                                MAX(TEL.PRECIO) AS PRECIO_MAX,
                                AVG(TEL.PRECIO) AS PRECIO_MEDIO
FROM TELEFONO TEL JOIN TIPO_TEL TIP   ON TEL.TIPO =
TIP.TIPO
GROUP BY  TEL.TIPO, TIP.DESC_TIPO
ORDER BY TEL.TIPO ASC;
```

e. HAVING

Más que como una función, se puede considerar la cláusula HAVING como un filtro adicional que actúa sobre el resultado de un GROUP BY.

Igual que el WHERE filtra las filas devueltas por el SELECT, el HAVING filtra el resultado de las funciones utilizadas.

Selección de los precios de la base de datos que están al menos dos veces:

```
SELECT TEL.PRECIO, COUNT(*) AS NUM_TEL
FROM TELEFONO TEL
GROUP BY TEL.PRECIO
HAVING COUNT(*) > 1;
```

Otro ejemplo, si se quiere calcular el promedio del precio por TIPO sólo en los teléfonos que estén varias veces en la tabla (un promedio de un solo valor no tiene mucho interés…):

PRECIO	NUM_TEL
99	2

```
SELECT TEL.TIPO, TIP.DESC_TIPO, MIN(TEL.PRECIO) AS PRECIO_MIN,
                                MAX(TEL.PRECIO) AS PRECIO_MAX,
                                AVG(TEL.PRECIO) AS PRECIO_MEDIO
FROM TELEFONO TEL JOIN TIPO_TEL TIP ON TEL.TIPO = TIP.TIPO
GROUP BY  TEL.TIPO, TIP.DESC_TIPO
HAVING COUNT(*) > 1
ORDER BY TEL.TIPO ASC;
```

Los tipos 'CT' e 'IP' solo tienen un valor en la tabla TELEFONO, por lo que no aparecen en el resultado.

TIPO	DESC_TIPO	PRECIO_MIN	PRECIO_MAX	PRECIO_MEDIO
DE	DESLIZANTE	89	99	94
SP	SMARTPHONE	99	159	129

9. La instrucción CASE

La instrucción CASE se puede utilizar en la selección de datos para dar formato a un resultado según el contenido de una columna.

Es posible comprobar los valores de una columna y tratarlos en función de su contenido.

Existen dos maneras de escribir el CASE: comprobando constantes respecto a una columna, o añadiendo condiciones (=, <, >, …) a esta columna.

Sintaxis con una constante:

```
SELECT <columna 1>, <columna 2>,
   CASE <columna 3>
      WHEN <constante 1> THEN <valor mostrado>
      WHEN <constante 2> THEN <valor mostrado>
      WHEN <constante 3> THEN <valor mostrado>
      ... ...
      ELSE <valor por defecto>
   END AS <encabezado de columna>
FROM <tabla1>, <tabla2> ... ...
WHERE ... ...
```

Sintaxis con una condición:

```
SELECT <columna 1>, <columna 2>,
   CASE
      WHEN <columna 3> <condición> <constante 1> THEN <valor
mostrado>
      WHEN <columna 3> <condición> <constante 2> THEN <valor
mostrado>
      WHEN <columna 3> <condición> <constante 3> THEN <valor
mostrado>
      ... ...
      ELSE <valor por defecto>
   END AS <encabezado de columna>
FROM <tabla1>, <tabla2> ... ...
WHERE ... ...
```

Ejemplo con el control del precio con valores fijos:

```
SELECT TEL.FECHA_COMPRA AS COMPRA,
       TEL.TIPO AS TIPO,
       TIPO.DESC_TIPO AS DESCRIPCION,
       TEL.MARCA AS MARCA,
       MAR.DESC_MARCA AS DESCRIPCION,
       TEL.PRECIO AS PRECIO,
       CASE TEL.PRECIO
            WHEN 49 THEN 'PRECIO MINIMO'
            WHEN 359 THEN 'PRECIO MAXIMO'
            WHEN 159 THEN 'PRECIO MEDIO'
            ELSE 'OTRO PRECIO'
        END AS TIPO_PRECIO
FROM TELEFONO TEL,
     TIPO_TEL TIPO ,
     MARCA_TEL MAR
WHERE TEL.TIPO  = TIPO.TIPO AND
      TEL.MARCA = MAR.MARCA;
```

Resultado:

COMPRA	TIPO	DESCRIPCION	MARCA	DESCRIPCION	PRECIO	TIPO_PRECIO
15/01/10	SP	SMARTPHONE	1	SAMSUNG	159	**PRECIO MEDIO**
14/03/10	SP	SMARTPHONE	2	SONY	99	**OTRO PRECIO**
02/05/10	CT	CON TAPA	3	PHILIPS	49	**PRECIO MINIMO**
25/07/10	DE	DESLIZANTE	4	MOTOROLA	89	**OTRO PRECIO**
30/09/10	IP	IPHONE	5	APPLE	359	**PRECIO MAXIMO**

Ahora si se quieren poner límites a este mismo control de precio se escribirá:

```
SELECT TEL.FECHA_COMPRA AS COMPRA,
       TEL.TIPO AS TIPO,
       TIPO.DESC_TIPO AS DESCRIPCION,
       TEL.MARCA AS MARCA,
       MAR.DESC_MARCA AS DESCRIPCION,
       TEL.PRECIO AS PRECIO,
       CASE
           WHEN TEL.PRECIO <= 49 THEN 'PRECIO MINIMO'
           WHEN TEL.PRECIO >= 359 THEN 'PRECIO MAXIMO'
           WHEN TEL.PRECIO > 49 AND TEL.PRECIO < 359 THEN 'PRECIO MEDIO'
       END AS TIPO_PRECIO
FROM TELEFONO TEL,
     TIPO_TEL TIPO ,
     MARCA_TEL MAR
WHERE TEL.TIPO  = TIPO.TIPO AND
      TEL.MARCA = MAR.MARCA;
```

Resultado:

COMPRA	TIPO	DESCRIPCION	MARCA	DESCRIPCION	PRECIO	TIPO_PRECIO
15/01/10	SP	SMARTPHONE	1	SAMSUNG	159	PRECIO MEDIO
14/03/10	SP	SMARTPHONE	2	SONY	99	PRECIO MEDIO
02/05/10	CT	CON TAPA	3	PHILIPS	49	PRECIO MINIMO

Las dos filas que estaban indicadas con 'OTRO PRECIO' se convierten en 'PRECIO MEDIO'.

25/07/10	DE	DESLIZANTE	4	MOTOROLA	89	PRECIO MEDIO
30/09/10	IP	IPHONE	5	APPLE	359	PRECIO MAXIMO

Las sintaxis Oracle y MySQL son equivalentes.

El CASE se debe utilizar con cuidado ya que el rendimiento tenderá a disminuir. Este comando es bastante exigente en términos de recursos, las condiciones se ejecutarán fila por fila.

10. Resumen de las posibles sintaxis del SELECT

A continuación puede ver las principales sintaxis que ha podido ver en esta sección. No están todas las posibilidades del SELECT pero sí las esenciales.

```
SELECT <nombre columna1>, <nombre columna 2> ... ... ,
    [SUM/COUNT/AVG/MIN/MAX](<nombre columna 3>),
    [SUM/COUNT/AVG/MIN/MAX](<nombre columna 4>),

    [CASE <nombre columna 5>
     WHEN ...
     END AS <encabezado de columna>],
     <nombre columna 5>,
     ... ...
FROM <tabla 1>
   JOIN <tabla 2> ON <tabla1.columna1> = <tabla2.columna1>
   JOIN <tabla 3> ON <tabla1.columna2> = <tabla3.columna2>
WHERE ... ... ...
   GROUP BY <nombre columna 1>, <nombre columna 2> ... ...
   HAVING [SUM/COUNT/AVG/MIN/MAX]<nombre columna 4>
         <operador aritmético> <valor> ... ...
ORDER BY <nombre columna 1>, <nombre columna 2> ... ...
```

11. Los operadores de conjuntos

Los operadores de conjuntos trabajan, como su nombre indica, sobre los conjuntos.

Se distinguen tres operadores principales que son: UNION que fusiona dos o más tablas, INTERSECT que buscará filas comunes a dos o más tablas, y EXCEPT que devolverá las filas de una tabla no presentes en otra tabla.

a. El operador UNION

UNION permite fusionar los datos de varias tablas.

Por ejemplo, tenemos una tabla MARCA_TEL y una tabla MARCA_PC que contienen los siguientes datos:

Tabla MARCA_TEL

MARCA	DESC_MARCA	PAIS
1	SAMSUNG	COREA
2	SONY	JAPON
3	PHILIPS	HOLANDA
4	MOTOROLA	USA
5	APPLE	USA

Tabla MARCA_PC

IDENT	DESC_MARCA	PAIS
1	SAMSUNG	COREA
2	SONY	JAPON
3	DELL	USA
4	HP	USA
5	APPLE	USA

Si se quieren recuperar todos los países que hay en estas dos tablas, escribiremos:

```
SELECT PAIS FROM MARCA_TEL
UNION
SELECT PAIS FROM MARCA_PC;
```

PAIS
COREA
JAPON
HOLANDA
USA

Automáticamente el sistema elimina los duplicados y muestra los valores una sola vez.

Para obtener todos los valores, hay que añadir la cláusula ALL como a continuación:

```
SELECT PAIS FROM MARCA_TEL
UNION ALL
SELECT PAIS FROM MARCA_PC;
```

PAIS
COREA
JAPON
HOLANDA
USA
USA
COREA

Esto es lo que hay que recordar del operador UNION:

JAPON
USA
USA
USA

Las columnas seleccionadas de las diferentes tablas deben ser del mismo tipo y del mismo tamaño.

- Se pueden unir tantas tablas como se quiera.
- El sistema elimina automáticamente los duplicados.

Otro ejemplo, si se seleccionan dos columnas como en la siguiente consulta, el sistema eliminará las filas duplicadas en las dos columnas (parejas Samsung/Corea, Sony/Japón y Apple/USA).

La unión se puede comparar con un SELECT DISTINCT sobre las dos tablas.

```
SELECT DESC_MARCA, PAIS FROM MARCA_TEL
UNION
SELECT DESC_MARCA, PAIS FROM MARCA_PC;
```

DESC_MARCA	PAIS
APPLE	USA
DELL	USA
HP	USA
MOTOROLA	USA
PHILIPS	HOLANDA
SAMSUNG	COREA
SONY	JAPON

La cláusula WHERE se puede utilizar para restringir el rango de filas seleccionables. En este caso los dos SELECT tienen un comportamiento independiente, habrá que añadir un WHERE en el primer o segundo SELECT o en los dos en función del resultado que queramos.

Si, por ejemplo, no se quiere ninguna marca americana en la primera tabla MARCA_TEL escribiremos:

De este modo 'MOTOROLA' que sólo está en MARCA_TEL no se seleccionará. Por el contrario, 'APPLE' se seleccionará dado que está en la otra tabla.

Si ahora no queremos ninguna marca americana de ninguna de las dos tablas, escribiremos:

También se puede utilizar la cláusula ORDER BY; se deberá poner al final y se aplica al conjunto del resultado obtenido.

Ejemplo:

Sintaxis del operador UNION:

```
SELECT DESC_MARCA, PAIS FROM MARCA_TEL
WHERE PAIS <> 'USA'
UNION
SELECT DESC_MARCA, PAIS FROM MARCA_PC;
```

DESC_MARCA	PAIS
APPLE	USA
DELL	USA
HP	USA
PHILIPS	HOLANDA
SAMSUNG	COREA
SONY	JAPON

```
SELECT DESC_MARCA, PAIS FROM MARCA_TEL
WHERE PAIS <> 'USA'
UNION
SELECT DESC_MARCA, PAIS FROM MARCA_PC
WHERE PAIS <> 'USA';
```

DESC_MARCA	PAIS
PHILIPS	HOLANDA
SAMSUNG	COREA
SONY	JAPON

```
SELECT DESC_MARCA, PAIS FROM MARCA_TEL
WHERE PAIS <> 'USA'
UNION
SELECT DESC_MARCA, PAIS FROM MARCA_PC
WHERE PAIS <> 'USA'
ORDER BY PAIS;
```

```
SELECT <columna 1>, <columna 2>, ... FROM <tabla1>
WHERE ...
UNION
SELECT <columna 1>, <columna 2>, ... FROM <tabla2>
WHERE ...
ORDER BY <columna 1> ...
```

b. El operador INTERSECT

La diferencia con UNION es simple. UNION selecciona los datos de una tabla y a continuación los datos de la otra tabla. Se recuperan TODAS las filas de las dos tablas y se eliminan los duplicados.

La intersección selecciona los datos que están en la tabla 1 y en la tabla 2 y a continuación también elimina los duplicados.

También es posible agregar las cláusulas WHERE y ORDER BY como con UNION.

Ejemplo:

```
SELECT DESC_MARCA, PAIS FROM MARCA_TEL
INTERSECT
SELECT DESC_MARCA, PAIS FROM MARCA_PC;
```

Las filas presentes en las dos tablas son las siguientes:

Si retomamos el último ejemplo de UNION, sólo quedan dos filas en común en las dos tablas.

DESC_MARCA	PAIS

APPLE	USA
SAMSUNG	COREA
SONY	JAPON

```
SELECT DESC_MARCA, PAIS FROM MARCA_TEL
WHERE PAIS <> 'USA'
INTERSECT
SELECT DESC_MARCA, PAIS FROM MARCA_PC
WHERE PAIS <> 'USA'
ORDER BY PAIS;
```

DESC_MARCA	PAIS
SAMSUNG	COREA
SONY	JAPON

Tenga en cuenta que el operador INTERSECT no está implementado en todos los SGBDR.

En ese caso se puede sustituir por este tipo de comando:

Sintaxis del operador INTERSECT:

```
SELECT DESC_MARCA, PAIS FROM MARCA_TEL TAB1
WHERE EXISTS (SELECT DESC_MARCA, PAIS FROM MARCA_PC TAB2
              WHERE TAB1.DESC_MARCA = TAB2.DESC_MARCA
                AND TAB1.PAIS = TAB2.PAIS)
```

```
SELECT <columna 1>, <columna 2>, ... FROM <tabla1>
WHERE ...
INTERSECT
SELECT <columna 1>, <columna 2>, ... FROM <tabla2>
WHERE ...
ORDER BY <columna 1> ...
```

c. El operador EXCEPT

EXCEPT es el contrario de INTERSECT; solo se seleccionan las filas de la primera tabla que no estén en la segunda tabla.

En este ejemplo solo las filas MOTOROLA y PHILIPS están en la tabla MARCA_TEL y no están en la tabla MARCA_PC.

También se pueden utilizar las cláusulas WHERE y ORDER BY como en UNION.

Ejemplo:

```
SELECT DESC_MARCA, PAIS FROM MARCA_TEL
EXCEPT
SELECT DESC_MARCA, PAIS FROM MARCA_PC;
```

DESC_MARCA	PAIS
MOTOROLA	USA
PHILIPS	HOLANDA

Tenga en cuenta que el operador EXCEPT no está implementado en todos los SGBDR.

En ese caso, se puede sustituir por este tipo de comando:

Sintaxis del operador EXCEPT:

```
SELECT DESC_MARCA, PAIS FROM MARCA_TEL TAB1
WHERE NOT EXISTS (SELECT DESC_MARCA, PAIS FROM MARCA_PC TAB2
              WHERE TAB1.DESC_MARCA = TAB2.DESC_MARCA
                AND TAB1.PAIS = TAB2.PAIS)
```

```
SELECT <columna 1>, <columna 2>, ... FROM <tabla1>
WHERE ...
EXCEPT
SELECT <columna 1>, <columna 2>, ... FROM <tabla2>
WHERE ...
ORDER BY <columna 1> ...
```

12. Cómo hacer una consulta: algunos consejos

Antes de hacer una consulta, debería poder consultar el modelo de datos, preferiblemente en formato gráfico. En primer lugar, hay que identificar la tabla principal que se desea consultar.

Si es posible, también deberíamos consultar la descripción de los índices de cada tabla, buscar las columnas que necesitamos y anotar las que nos faltan.

Para saber el número de filas afectadas por la consulta, podemos hacer un COUNT(*) en un primer momento y en función del resultado, afinar la cláusula WHERE. Una vez que el número de filas seleccionado es correcto, podemos sustituir el COUNT(*) con las columnas de la tabla principal.

A continuación, hay que buscar en aquella(s) tabla(s) las columnas que faltan, encontrar el vínculo que permita realizar la unión entre estas dos tablas. Atención, el vínculo puede estar formado por varias columnas. Para cada tabla, hay que añadir en la consulta las columnas que se quieran recuperar y las columnas de las claves que sean iguales. Podemos utilizar el COUNT(*) para probar la consulta.

En Oracle, también existe la posibilidad de usar la pseudo-columna « ROWNUM » para devolver sólo algunas filas cuando probamos la consulta. El rownum es el número de fila asignado por la base de datos en una selección. El equivalente MySQL es la cláusula LIMIT.

Ejemplo Oracle:

```
SELECT DESC_MARCA, PAIS FROM MARCA_TEL
WHERE PAIS <> 'USA' AND ROWNUM < 3;
```

Ejemplo MySQL:

En este ejemplo, la selección devuelve las dos primeras filas aunque el resultado final tenga muchas más.

```
SELECT DESC_MARCA, PAIS FROM MARCA_TEL
WHERE PAIS <> 'USA' LIMIT 2;
```

Si sólo queremos una fila, escribiremos:

o:

DESC_MARCA	PAIS
SAMSUNG	COREA
SONY	JAPON

Tenga presente el número de filas de cada tabla, ya que puede influir enormemente en el rendimiento.

Hay que utilizar lo máximo posible las columnas declaradas en los índices. Una unión entre tres o cuatro tablas grandes puede ser muy rápida si se utilizan los índices de cada una de las tablas. Basta que una tabla no tenga índice para colapsar el rendimiento de la consulta.

Existen herramientas que nos indican cómo el sistema se mueve entre las tablas para acceder a la información (ver el capítulo Para ir más lejos - Algunos conceptos de rendimiento).

```
SELECT DESC_MARCA, PAIS FROM MARCA_TEL
WHERE PAIS <> 'USA' AND ROWNUM = 1;
```

```
SELECT DESC_MARCA, PAIS FROM MARCA_TEL
WHERE PAIS <> 'USA' LIMIT 1;
```

A veces el número de filas devueltas después de varias uniones es inferior al número de filas seleccionables en la tabla principal.

Es normal si todas las ocurrencias de la tabla principal no están en las tablas secundarias. La unión interna solo devuelve las filas comunes. Para comprobar la consulta, es posible utilizar una unión externa que seleccionará todas las filas de la tabla principal.

En Oracle, se puede utilizar EXPLAIN PLAN para analizar la eficacia de una consulta o EXPLAIN en MySQL. Lo veremos en el capítulo Para ir más lejos - Algunos conceptos de rendimiento.

¿Cómo estar seguro de que el resultado de la consulta es correcto? No es fácil responder a esta pregunta, sobre todo cuando una consulta se puede escribir de varias formas.

Lo más sencillo es hacer el ejercicio a mano mirando el contenido tabla por tabla, contando el número de filas de la tabla principal y buscando en las tablas secundarias los elementos correspondientes con consultas individuales en estas tablas. Cruzando los resultados, podremos comprobar la consulta.

Utilizando funciones (count, substr, to_char…) en las cláusulas WHERE sobretodo, aumenta el riesgo de no devolver el número de filas deseado si no aprendemos perfectamente la sintaxis. Para probar una función, lo más sencillo es ponerla en las columnas a seleccionar para visualizar su contenido.

Ejemplo:

```
SELECT ACTOR.APELLIDO||' '||ACTOR.NOMBRE ACTOR,
ACTOR.FECHA_NACIMIENTO "NACIDO EL",
      CAST.ROL, PAIS.DESCRIPCION, PELICULAS.TITULO, PELICULAS.FEHA_ESTRENO
FROM PELICULAS PELICULAS, CASTING CAST, ACTOR ACTOR, PAIS PAIS
WHERE
   PELICULAS.IDENT_PELICULA = CAST.IDENT_PELICULA AND
   CAST.IDENT_ACTOR         = ACTOR.IDENT_ACTOR AND
   PAIS.IDENT_PAIS          = ACTOR.NACIONALIDAD AND
   SUBSTR(PELICULAS.TITULO,1,1) = 'S' AND SUBSTR(PELICULAS.
TITULO,6,3) = 'WAR'
ORDER BY ACTOR.APELLIDO||' '||ACTOR.NOMBRE, PELICULA.TITULO
DESC;
```

La doble pipe || indica una concatenación entre dos columnas.

Subiendo los SUBSTR al SELECT y simplificando la cláusula WHERE, podemos visualizar el valor de los SUBSTR así.

```
SELECT  PELICULAS.TITULO, SUBSTR(PELICULAS.TITULO,1,1) SUB1,
SUBSTR(PELICULAS.TITULO,6,3) SUB2
FROM PELICULAS PELICULAS ORDER BY PELICULAS.TITULO;
```

TITULO	SUB1	SUB2
AVATAR	A	R
BIENVENIDOS AL NORTE	B	ENI
NIKITA	N	A
STAR WARS 6: EL RETORNO DEL JEDI	**S**	**WAR**
SUBWAY	S	Y

Aún más sencillo, utilizando la pseudo-tabla DUAL.

El principio básico en la construcción de una consulta es cortar en trozos sencillos de probar y ejecutar cada trozo individualmente antes de probar la consulta completa.

No dude en probar diferentes sintaxis para cruzar los resultados.

```
SELECT  SUBSTR('STAR WARS 6: EL RETORNO EL JEDI',1,1) SUB1,
        SUBSTR('STAR WARS 6: EL RETORNO EL JEDI',6,3) SUB2 FROM
DUAL;
```

SUB1	SUB2
S	WAR

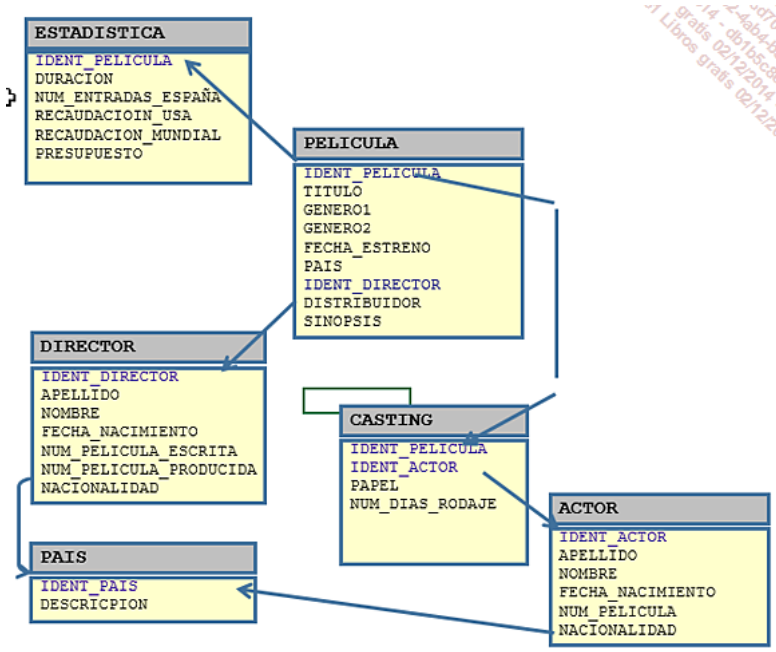
13. Ejercicios sobre la selección de datos

a. Preguntas generales

1. ¿Cuáles son los cuatro tipos de unión descritos en este libro?
2. ¿Cuál es la función que permite contar el número de ocurrencias?
3. ¿Cuál es el comando para ordenar y por defecto la ordenación es descendente o ascendente?
4. ¿Cuáles son las funciones de agrupamiento que se pueden utilizar con un GROUP BY?
5. ¿Cuáles son los tres operadores de conjuntos?

b. Ejercicios de aplicación

Descripción y contenido de las tablas utilizadas en los ejercicios siguientes.



Modelo de datos utilizado para ilustrar los ejercicios de manipulación de tablas

TABLA PELICULA

Consulta de creación de la tabla (sintaxis estándar):

```
CREATE TABLE PELICULA (IDENT_PELICULA  INTEGER,
                       TITULO          VARCHAR(50),
                       GENERO1         VARCHAR(20),
                       GENERO2         VARCHAR(20),
                       FECHA_ESTRENO   DATE,
                       PAIS            SMALLINT,
                       IDENT_DIRECTOR  INTEGER,
                       DISTRIBUIDOR    VARCHAR(50),
                       SINOPSIS        VARCHAR(2000));
```

Consulta de inserción de filas (sintaxis Oracle):

```
INSERT INTO PELICULA VALUES
(1,'SUBWAY','POLICIACA','DRAMA',TO_DATE('10/04/1985','DD/MM/YYYY'),
1,1,'FILMAX','Explica las aventuras de la población subterránea en los
túneles del metro de París');

INSERT INTO PELICULA VALUES
(2,'NIKITA','DRAMA','ROMANTICA',TO_DATE('21/02/1990','DD/MM/YYYY'
),1,1,'FILMAX','Nikita, condenada a cadena perpetua, es obligada a trabajar
para el gobierno como agente de los servicios secretos.');

INSERT INTO PELICULA VALUES (3,'STAR WARS 6: EL RETORNO DEL JEDI',
'ACCION','SF',TO_DATE('19/10/1983','DD/MM/YYYY'),2,2,'20th
Century Fox ','El imperio galáctico es más poderoso que nunca:
la construcción de la nueva arma, la Estrella de la Muerte, amenaza todo
el universo.');

INSERT INTO PELICULA VALUES
(4,'AVATAR','ACCION','SF',TO_DATE('16/10/2009','DD/MM/YYYY'),2,3,
'20th Century Fox ','A pesar de su parálisis, Jake Sully, un viejo marine
inmovilizado en una silla de ruedas, en el fondo sigue siendo un soldado');

INSERT INTO PELICULA VALUES (5,'BIENVENIDOS AL NORTE','COMEDIA',
'',TO_DATE('27/02/2008','DD/MM/YYYY'),1,4,'Lauren Films',
'Philippe Abrams, director de correos de Salon-de-Provence, es trasladado
al Norte.');
```

Para la sintaxis MySQL, basta con cambiar el TO_DATE('10/04/1985','DD/MM/YYYY') por STR_TO_DATE('10/04/1985','%d/%m/%Y').

Contenido de la tabla PELICULA:

IDENT_ PELICULA	TITULO	GENERO1	GENERO2	FECHA_ ESTRENO	PAIS	IDENT_ DIRECTOR	DISTRIBUIDOR
1	SUBWAY	POLICIACA	DRAMA	10/04/85	1	1	FILMAX
SINOPSIS	Cuenta las aventuras de la población subterránea en los túneles del metro de París						
2	NIKITA	DRAMA	ROMANTICA	21/02/90	1	1	FILMAX
SINOPSIS	Nikita, condenada a cadena perpetua, es obligada a trabajar en secreto para el gobierno como agente de los servicios secretos.						
3	STAR WARS 6 - EL RETORNO DEL JEDI	ACCION	SF	19/10/83	2	2	20th Century Fox
SINOPSIS	El imperio galáctico es más poderoso que nunca: la construcción de la nueva arma, la Estrella de la Muerte, amenaza todo el universo.						
4	AVATAR	ACCION	SF	16/10/09	2	3	20th Century

TABLA ACTOR

Consulta de creación de la tabla (sintaxis estándar):

							Fox
SINOPSIS	A pesar de su parálisis, Jake Sully, un viejo marino inmovilizado en una silla de ruedas, en el fondo sigue siendo un soldado						
5	BIENVENIDO AL NORTE	COMEDIA		27/02/08	1	4	Lauren Films
SINOPSIS	Philippe Abrams, director de correos de Salon-de-Provence, es trasladado al Norte.						

```
CREATE TABLE ACTOR (IDENT_ACTOR      INTEGER,
                    APELLIDO         VARCHAR(50),
                    NOMBRE           VARCHAR(20),
                    FECHA_NACIMIENTO DATE,
                    NUM_PELICULA     INTEGER,
                    NACIONALIDAD     SMALLINT);
```

Para la sintaxis MySQL basta con cambiar los 'DD/MM/YYYY' por '%d/%m/%Y' y los TO_DATE por STR_TO_DATE.

Consulta de inserción de filas (sintaxis Oracle):

```
INSERT INTO ACTOR VALUES
(1,'ADJANI','ISABELLE',TO_DATE('27/06/1955','DD/MM/YYYY'),42,1);
INSERT INTO ACTOR VALUES
(2,'LAMBERT','CHRISTOPHE',TO_DATE('29/03/1957','DD/MM/YYYY'),64,1);
INSERT INTO ACTOR VALUES
(3,'BOHRINGER','RICHARD',TO_DATE('16/06/1942','DD/MM/YYYY'),132,1);
INSERT INTO ACTOR VALUES
(4,'GALABRU','MICHEL',TO_DATE('27/10/1922','DD/MM/YYYY'),277,1);
INSERT INTO ACTOR VALUES
(5,'PARILLAUD','ANNE',TO_DATE('06/05/1960','DD/MM/YYYY'),35,1);
INSERT INTO ACTOR VALUES
(6,'FORD','HARRISON',TO_DATE('13/06/1942','DD/MM/YYYY'),64,2);
INSERT INTO ACTOR VALUES
(7,'FISHER','CARRIE',TO_DATE('21/10/1956','DD/MM/YYYY'),74,2);
INSERT INTO ACTOR VALUES
(8,'SALDANA','ZOE',TO_DATE('19/06/1978','DD/MM/YYYY'),31,2);
INSERT INTO ACTOR VALUES
(9,'WEAVER','SIGOURNEY',TO_DATE('08/10/1949','DD/MM/YYYY'),66,2);
INSERT INTO ACTOR VALUES
(10,'RENO','JEAN',TO_DATE('30/06/1948','DD/MM/YYYY'),75,1);
INSERT INTO ACTOR VALUES
(11,'BOON','DANY',TO_DATE('26/06/1966','DD/MM/YYYY'),23,1);
INSERT INTO ACTOR VALUES
(12,'MERAD','KAD',TO_DATE('27/03/1964','DD/MM/YYYY'),55,3);
```

Contenido de la tabla ACTOR:

IDENT_ ACTOR	APELLIDO	NOMBRE	FECHA_ NACIMIENTO	NUM_ PELICULA	NACIONALIDAD
1	ADJANI	ISABELLE	27/06/55	42	1
2	LAMBERT	CHRISTOPHE	29/03/57	64	1
3	BOHRINGER	RICHARD	16/06/42	132	1
4	GALABRU	MICHEL	27/10/22	277	1
5	PARILLAUD	ANNE	06/05/60	35	1
6	FORD	HARRISON	13/06/42	64	2
7	FISHER	CARRIE	21/10/56	74	2
8	SALDANA	ZOE	19/06/78	31	2
9	WEAVER	SIGOURNEY	08/10/49	66	2
10	RENO	JEAN	30/06/48	75	1
11	BOON	DANY	26/06/66	23	1
12	MERAD	KAD	27/03/64	55	3

TABLA ESTADÍSTICA

Consulta de creación de la tabla (sintaxis estándar):

```
CREATE TABLE ESTADISTICA
     (IDENT_PELICULA      INTEGER,
      DURACION            INTEGER,
      NUM_ENTRADAS_ESPANA DECIMAL(15,0),
      RECAUDACION_USA     DECIMAL(15,2),
      RECAUDACION_MUNDIAL DECIMAL(15,2),
      PRESUPUESTO         DECIMAL(12,2));
```

Consulta de inserción de filas (sintaxis estándar):

```
INSERT INTO ESTADISTICA VALUES
(1,104,2917562,390659,1272637.45,2.6);
INSERT INTO ESTADISTICA VALUES (2,118,3787845,5017971,0,7.6);
INSERT INTO ESTADISTICA VALUES
(3,133,4263000,191648000,472000000,32);
INSERT INTO ESTADISTICA VALUES
(4,170,12018251,760505847,2946271769,237);
INSERT INTO ESTADISTICA VALUES (5,100,21000000,0,245000000,11);
```

Contenido de la tabla ESTADISTICA:

IDENT_ PELICULA	DURACION	NUM_ ENTRADAS_ ESPANA	RECAUDACION_USA	RECAUDACION_ MUNDIAL	PRESUPUESTO
1	104	2 917 562	390 659	1 272 637	2,6
2	118	3 787 845	5 017 971	0	7,6
3	133	4 263 000	191 648 000	472 000 000	32
4	170	12 018 251	760 505 847	2 946 271 769	237
5	100	21 000 000	0	245 000 000	11

TABLA DIRECTOR

Consulta de creación de la tabla (sintaxis estándar):

```
CREATE TABLE DIRECTOR (IDENT_DIRECTOR INTEGER,
                       APELLIDO       VARCHAR(50),
                       NOMBRE         VARCHAR(20),
                       NACIONALIDAD   SMALLINT);
```

Consulta de inserción de filas (sintaxis Oracle):

```
INSERT INTO DIRECTOR VALUES
(1,'ADJANI','ISABELLE',42);
INSERT INTO DIRECTOR VALUES
(2,'LAMBERT','CHRISTOPHE',64);
INSERT INTO DIRECTOR VALUES
(3,'BOHRINGER','RICHARD',132);
INSERT INTO DIRECTOR VALUES
(4,'GALABRU','MICHEL',277);
INSERT INTO DIRECTOR VALUES
(5,'PARILLAUD','ANNE',35);
INSERT INTO DIRECTOR VALUES
(6,'FORD','HARRISON',64);
INSERT INTO DIRECTOR VALUES
(7,'FISHER','CARRIE',74);
INSERT INTO DIRECTOR VALUES
(8,'SALDANA','ZOE',31);
INSERT INTO DIRECTOR VALUES
(9,'WEAVER','SIGOURNEY',66);
INSERT INTO DIRECTOR VALUES
(10,'RENO','JEAN',75);
INSERT INTO DIRECTOR VALUES
(11,'BOON','DANY',23);
INSERT INTO DIRECTOR VALUES
(12,'MERAD','KAD',55);
```

Contenido de la tabla DIRECTOR:

IDENT_ DIRECTOR	APELLIDO	NOMBRE	NACIONALIDAD
1	ADJANI	ISABELLE	42
2	LAMBERT	CHRISTOPHE	64
3	BOHRINGER	RICHARD	132
4	GALABRU	MICHEL	277
5	PARILLAUD	ANNE	35
6	FORD	HARRISON	64
7	FISHER	CARRIE	74
8	SALDANA	ZOE	31
9	WEAVER	SIGOURNEY	66
10	RENO	JEAN	75
11	BOON	DANY	23
12	MERAD	KAD	55

TABLA CASTING

Consulta de creación de la tabla (sintaxis estándar):

```
CREATE TABLE CASTING (IDENT_PELICULA INTEGER,
                      IDENT_ACTOR    INTEGER,
                      PAPEL          VARCHAR(20));
```

Consulta de inserción de filas (sintaxis estándar):

```
INSERT INTO CASTING VALUES
(1,1,'Principal');
INSERT INTO CASTING VALUES
(1,2,'Secundario');
INSERT INTO CASTING VALUES
(2,3,'Principal');
INSERT INTO CASTING VALUES
(2,4,'Secundario');
INSERT INTO CASTING VALUES
(3,5,'Principal');
INSERT INTO CASTING VALUES
(3,6,'Secundario');
INSERT INTO CASTING VALUES
(4,7,'Principal');
INSERT INTO CASTING VALUES
(4,8,'Secundario');
INSERT INTO CASTING VALUES
(5,9,'Principal');
INSERT INTO CASTING VALUES
(5,10,'Secundario');
INSERT INTO CASTING VALUES
(6,11,'Principal');
INSERT INTO CASTING VALUES
(6,12,'Secundario');
```

Contenido de la tabla CASTING:

IDENT_ PELICULA	IDENT_ ACTOR	PAPEL
1	1	Principal
1	2	Secundario
2	3	Principal
2	4	Secundario
3	5	Principal
3	6	Secundario
4	7	Principal
4	8	Secundario
5	9	Principal
5	10	Secundario
6	11	Principal
6	12	Secundario

TABLA PAIS

Consulta de creación (sintaxis estándar):

```
CREATE TABLE PAIS (IDENT_PAIS INTEGER,
                   NOMBRE     VARCHAR(20));
```

Consulta de inserción de filas (sintaxis estándar):

```
INSERT INTO PAIS VALUES
(1,'FRANCIA');
INSERT INTO PAIS VALUES
(2,'USA');
INSERT INTO PAIS VALUES
(3,'ESPAÑA');
```

Contenido de la tabla PAIS:

IDENT_ PAIS	NOMBRE
1	FRANCIA
2	USA
3	ESPAÑA

Primer ejercicio

Seleccionar toda la información de las películas realizadas por un director francés ordenadas por el nombre de la película.

Segundo ejercicio

Seleccionar el nombre de la película, la fecha de estreno, el nombre del director, el nombre de los actores, su fecha de nacimiento, así como el presupuesto de la película. Ordenarlo todo descendientemente por título de la película y nombre de los actores.

Tercer ejercicio

Encontrar el número de actores por película en la base de datos. Mostrar el título, la fecha de estreno, el nombre del director y el distribuidor.

Cuarto ejercicio

Seleccionar el título de la película, la fecha de estreno, el nombre y apellidos del director, el nombre y apellidos del actor, su fecha de nacimiento, el presupuesto de la película y número de entradas en España de las películas que tengan un actor argelino.

Quinto ejercicio

Seleccionar la película que haya recaudado más en todo el mundo.

Sexto ejercicio

Seleccionar el actor que ha participado en dos películas distintas.

Séptimo ejercicio

Seleccionar la persona que es director y actor a la vez.

Octavo ejercicio

```
CREATE TABLE DIRECTOR
                    (IDENT_DIRECTOR         INTEGER,
                     APELLIDO               VARCHAR(50),
                     NOMBRE                 VARCHAR(20),
                     FECHA_NACIMIENTO       DATE,
                     NUM_PELICULA_ESCRITA   INTEGER,
                     NUM_PELICULA_PRODUCIDA INTEGER,
                     NACIONALIDAD           SMALLINT);
```

```
INSERT INTO DIRECTOR VALUES
('1','BESSON','LUC',TO_DATE('18/03/1959','DD/MM/YYYY'),40,99,1);
INSERT INTO DIRECTOR VALUES
('2','LUCAS','GEORGES',TO_DATE('14/05/1944','DD/MM/YYYY'),79,64,2);
INSERT INTO DIRECTOR VALUES
('3','CAMERON','JAMES',TO_DATE('16/08/1954','DD/MM/YYYY'),22,23,2);
INSERT INTO DIRECTOR VALUES
('4','BOON','DANY',TO_DATE('26/06/1966','DD/MM/YYYY'),5,1,1);
```

IDENT_ DIRECTOR	APELLIDO	NOMBRE	FECHA_ NACIMIENTO	NUM_ PELICULA_ ESCRITA	NUM_ PELICULA_ PRODUCIDA	NACIONALIDAD
1	BESSON	LUC	18/03/59	40	99	1
2	LUCAS	GEORGES	14/05/44	79	64	2
3	CAMERON	JAMES	16/08/54	22	23	2
4	BOON	DANY	26/06/66	5	1	1

```
CREATE TABLE CASTING
     (IDENT_PELICULA   INTEGER,
      IDENT_ACTOR      INTEGER,
      PAPEL            VARCHAR(100),
      NUM_DIAS_RODAJE  INTEGER);
```

```
INSERT INTO CASTING VALUES (1,1,'HELENA',100);
INSERT INTO CASTING VALUES (1,2,'FRED',100);
INSERT INTO CASTING VALUES (1,3,'INSPECTOR GESBERG',NULL);
INSERT INTO CASTING VALUES (1,4,'LA FLORISTA',35);
INSERT INTO CASTING VALUES (1,10,'EL BATERÍA',20);
INSERT INTO CASTING VALUES (2,5,'NIKITA',68);
INSERT INTO CASTING VALUES (2,10,'VICTOR EL LIMPIADOR',9);
INSERT INTO CASTING VALUES (3,6,'HAN SOLO',201);
INSERT INTO CASTING VALUES (3,7,'PRINCESA LEIA',203);
INSERT INTO CASTING VALUES (4,8,'NEYTIRI',50);
INSERT INTO CASTING VALUES (4,9,'Dr. Grace Augustine',45);
INSERT INTO CASTING VALUES (5,11,'ANTOINE BAILLEUL',125);
INSERT INTO CASTING VALUES (5,12,'PHILIPPE ABRAMS',126);
```

IDENT_PELICULA	IDENT_ACTOR	PAPEL	NUM_DIAS_RODAJE
1	1	HELENA	100
1	2	FRED	100
1	3	INSPECTOR GESBERG	NULL
1	4	LA FLORISTA	35
1	10	EL BATERÍA	20
2	5	NIKITA	68
2	10	VICTOR EL LIMPIADOR	9
3	6	HAN SOLO	201
3	7	PRINCESA LEIA	203
4	8	NEYTIRI	50
4	9	Dr. Grace Augustine	45
5	11	ANTOINE BAILLEUL	125
5	12	PHILIPPE ABRAMS	126

```
CREATE TABLE PAIS
     (IDENT_PAIS  SMALLINT,
      DESCRIPCION VARCHAR(100));
```

```
INSERT INTO PAIS VALUES (1,'FRANCIA');
INSERT INTO PAIS VALUES (2,'USA');
INSERT INTO PAIS VALUES (3,'ARGELIA');
```

IDENT_PAIS	DESCRIPCION
1	FRANCIA
2	USA
3	ARGELIA

Seleccionar los actores que han participado en películas cuyo nombre comience por la letra S. Indicar su papel y su nacionalidad.

Noveno ejercicio

Seleccionar los actores que han nacido entre 1948 y 1978 así como el número total de días de rodaje que han hecho.

Primer ejercicio

Seleccionar toda la información de las películas realizadas por un director francés ordenadas por el nombre de la película.

Segundo ejercicio

Seleccionar el nombre de la película, la fecha de estreno, el nombre del director, el nombre de los actores, su fecha de nacimiento, así como el presupuesto de la película. Ordenarlo todo descendientemente por título de la película y nombre de los actores.

Tercer ejercicio

Encontrar el número de actores por película en la base de datos. Mostrar el título, la fecha de estreno, el nombre del director y el distribuidor.

Cuarto ejercicio

Seleccionar el título de la película, la fecha de estreno, el nombre y apellidos del director, el nombre y apellidos del actor, su fecha de nacimiento, el presupuesto de la película y número de entradas en España de las películas que tengan un actor argelino.

Quinto ejercicio

Seleccionar la película que haya recaudado más en todo el mundo.

Sexto ejercicio

Seleccionar el actor que ha participado en dos películas distintas.

Séptimo ejercicio

Seleccionar la persona que es director y actor a la vez.

Octavo ejercicio

Seleccionar los actores que han participado en películas cuyo nombre comience por la letra S. Indicar su papel y su nacionalidad.

Noveno ejercicio

Seleccionar los actores que han nacido entre 1948 y 1978 así como el número total de días de rodaje que han hecho.

La inserción de datos

Después de haber creado la estructura de las tablas y antes de poder seleccionar datos de la base, hay que insertar valores.

Hemos visto en el capítulo sobre el LDD, que es posible rellenar una tabla a partir de otra al crear la tabla con CREATE TABLE, pero en la mayoría de los casos la tabla se crea vacía y a continuación se insertan los datos.

1. El comando INSERT

El comando INSERT permite insertar valores en una tabla.

El método más simple consiste en insertar de una vez en una fila todas las columnas de la tabla.

En este caso, basta con indicar al sistema todos los valores que se quieren insertar en el orden de las columnas de la tabla.

Ejemplo de inserción en la tabla TELEFONO:

```
INSERT INTO TELEFONO
VALUES
(7,'SP' ,1,TO_DATE('2009-01-15','YYYY-MM-DD'),189,190622,'ROJO');
```

En este caso, es imprescindible que los valores estén en el orden de las columnas. Es bastante arriesgado trabajar así, ya que si la tabla evoluciona en el tiempo con nuevas columnas, este orden no funcionará. El sistema indicará que faltan valores.

Es preferible indicar las columnas de este modo:

```
INSERT INTO TELEFONO
(NUMERO, TIPO, MARCA, FECHA_COMPRA,PRECIO,NUM_PROPIETARIO,COLOR)
VALUES
(1,'SP' ,1,to_date('15/01/2010','DD/MM/YYYY'),159,190120,'ROJO');

INSERT INTO TELEFONO
(NUMERO, TIPO, MARCA, FECHA_COMPRA,PRECIO,NUM_PROPIETARIO)
VALUES
(2,'SP' ,2,to_date('14/03/2010','DD/MM/YYYY'), 99,190215);

INSERT INTO TELEFONO
(NUMERO, TIPO, MARCA, FECHA_COMPRA,PRECIO,NUM_PROPIETARIO,COLOR)
VALUES
(3,'CT' ,3,to_date('02/05/2010','DD/MM/YYYY'),  49,190001,'NEGRO');

INSERT INTO TELEFONO
(NUMERO, TIPO, MARCA, FECHA_COMPRA,PRECIO,NUM_PROPIETARIO,COLOR)
VALUES
(4,'DE' ,4,to_date('25/07/2010','DD/MM/YYYY'), 89,190222,'BLANCO');

INSERT INTO TELEFONO
(NUMERO, TIPO, MARCA, FECHA_COMPRA,PRECIO,NUM_PROPIETARIO)
VALUES
(5,'IP' ,5,to_date('30/09/2010','DD/MM/YYYY'),359,190561);

INSERT INTO TELEFONO
(NUMERO, TIPO, MARCA, FECHA_COMPRA,PRECIO,NUM_PROPIETARIO)
VALUES
(6,'DE' ,6,to_date('01/01/2009','DD/MM/YYYY'),99,122120,'BLANCO');
```

Para la sintaxis MySQL, basta con cambiar el TO_DATE('10/04/1985','DD/MM/YYYY') por STR_TO_DATE('10/04/1985','%d/%m/%Y').

La mayor parte de las columnas de la tabla no se habían declarado como NOT NULL, es posible no rellenar algunas columnas. No obstante, ponga atención a las posibles restricciones de integridad.

El siguiente comando sólo rellena cuatro columnas de la tabla:

```
INSERT INTO TELEFONO
  (NUMERO, TIPO, MARCA, NUM_PROPIETARIO)
VALUES
  (8,'SP' ,2, 190001);
```

Después de añadir dos filas a la tabla TELEFONO, ahora contiene los siguientes datos:

NUMERO	TIPO	MARCA	FECHA_COMPRA	PRECIO	NUM_PROPIETARIO	COLOR
1	SP	1	15/01/10	159	190120	ROJO
2	SP	2	14/03/10	99	190215	
3	CT	3	02/05/10	49	190001	NEGRO
4	DE	4	25/07/10	89	190222	BLANCO
5	IP	5	30/09/10	359	190561	
6	DE	6	01/01/09	99	122120	BLANCO
7	**SP**	**1**	**15/01/09**	**189**	**190622**	**ROJO**
8	**SP**	**2**	**10/03/14**	**0**	**190001**	

Se puede observar que la última fila no tiene precio ni color dado que no se ha indicado nada en el INSERT. Por el contrario, la fecha de compra ha tomado por defecto la fecha actual dado que en el CREATE TABLE habíamos indicado « DEFAULT CURRENT_DATE ».

Sintaxis general:

```
INSERT INTO <nombre tabla>
   (<columna 1>, <columna 2>, ... ... )
VALUES
   (<val1>, <val2>, ... ... );
```

Inserción en las tablas TIPO_TEL y MARCA_TEL

```
INSERT INTO TIPO_TEL VALUES ('SP','SMARTPHONE');
INSERT INTO TIPO_TEL VALUES ('CT','CON TAPA');
INSERT INTO TIPO_TEL VALUES ('DE','DESLIZANTE');
INSERT INTO TIPO_TEL VALUES ('IP','IPHONE');
INSERT INTO TIPO_TEL VALUES ('OT','OTRO');


INSERT INTO MARCA_TEL VALUES (1, 'SAMSUNG','COREA');
INSERT INTO MARCA_TEL VALUES (2, 'SONY','JAPON');
INSERT INTO MARCA_TEL VALUES (3, 'PHILIPS','HOLANDA');
INSERT INTO MARCA_TEL VALUES (4, 'MOTOROLA','USA');
INSERT INTO MARCA_TEL VALUES (5, 'APPLE','USA');
```

2. Inserción a partir de otra tabla

En los ejemplos anteriores, se insertan constantes en la tabla y las filas una a una.

Puede ser interesante recuperar datos de otras tablas o de la misma para aumentar la productividad.

Si existe otra tabla que contenga los datos que se quieren insertar en la tabla TELEFONO, hay que utilizar un SELECT dentro del INSERT e indicar las columnas que se quiere recuperar.

Ejemplo con una tabla llamada COMPRA_TEL que contiene los teléfonos comprados recientemente pero sin asignar a nadie:

Tabla COMPRA_TEL

IDENTIFICADOR	TIPO	MARCA	NUM_ PROVEEDOR	PRECIO_ COMPRA	COLOR	FECHA_ SALIDA
100101	CT	2	111235	59	ROJO	01/01/09
100102	SP	3	54658	49	ROSA	15/01/09
100103	DE	5	666697	19	NEGRO	15/03/09
100104	DE	5	6464654	29	BLANCO	01/02/10
100105	IP	5	121313	119	ROJO	15/07/09
100106	SP	4	646548	39	BLANCO	30/06/08
100107	SP	1	111235	89	ROJO	15/02/09
100108	SP	2	54658	14	VERDE	30/09/08

Hay que recuperar las columnas TIPO, MARCA, COLOR. El PRECIO de la tabla TELEFONO se calculará a partir del PRECIO_COMPRA de la tabla COMPRA_TEL con una fórmula.

El IDENTIFICADOR no se recupera ya que se trata de la clave de la tabla COMPRA_TEL; tendremos que asignar un NUMERO en la tabla TELEFONO utilizando la secuencia S_NUMERO que asignará un número único y que se irá incrementando.

Consultas de creación de la tabla COMPRA_TEL y de la secuencia S_NUMERO:

```
CREATE TABLE COMPRA_TEL (IDENTIFICADOR  INTEGER
                         TIPO           VARCHAR(2),
                         MARCA          INTEGER,
                         NUM_PROVEEDOR   INTEGER,
                         PRECIO_COMPRA   DECIMAL(9,2),
                         COLOR           VARCHAR(25),
                         FECHA_SALIDA    DATE,
CONSTRAINT PK_COMPRA_TEL PRIMARY KEY (IDENTIFICADOR));

CREATE SEQUENCE S_NUMERO START WITH 9 INCREMENT BY 1
MINVALUE 9 MAXVALUE 999999 NOCYCLE;
```

Comandos de inserción en la tabla COMPRA_TEL:

```
INSERT INTO COMPRA_TEL VALUES (100101,'CT',2,111235 ,
59  ,'ROJO',TO_DATE('01/01/2009','DD/MM/YYYY'));
INSERT INTO COMPRA_TEL VALUES (100102,'SP',3,54658  ,49  ,'ROSA'
,TO_DATE('15/01/2009','DD/MM/YYYY'));
INSERT INTO COMPRA_TEL VALUES (100103,'DE',5,666697 ,19  ,'NEGRO'
,TO_DATE('15/03/2009','DD/MM/YYYY'));
INSERT INTO COMPRA_TEL VALUES (100104,'DE',5,6464654,29  ,'BLANCO'
TO_DATE('01/02/2010','DD/MM/YYYY'));
INSERT INTO COMPRA_TEL VALUES (100105,'IP',5,121313 ,
119 ,'ROJO',TO_DATE('15/07/2009','DD/MM/YYYY'));
INSERT INTO COMPRA_TEL VALUES (100106,'SP',4,646548 ,
39  ,'BLANCO',TO_DATE('30/06/2008','DD/MM/YYYY'));
INSERT INTO COMPRA_TEL VALUES (100107,'SP',1,111235 ,
89  ,'ROJO',TO_DATE('15/02/2009','DD/MM/YYYY'));
INSERT INTO COMPRA_TEL VALUES (100108,'SP',2,54658  ,14  ,'VERDE'
,TO_DATE('30/09/2008','DD/MM/YYYY'));
```

Ejemplo de inserción a partir de otra tabla con la sintaxis Oracle:

Esta consulta inserta las siete filas de la tabla COMPRA_TEL en la tabla TELEFONO.

```
INSERT INTO TELEFONO
    (NUMERO,TIPO, MARCA, COLOR, PRECIO, FECHA_COMPRA)
 SELECT S_NUMERO.NEXTVAL,TIPO,MARCA,COLOR,
        PRECIO_COMPRA*2.5,CURRENT_DATE FROM COMPRA_TEL;
```

La columna NUMERO se actualiza con la secuencia S_NUMERO.

La columna precio es un cálculo sobre el PRECIO_COMPRA multiplicado por 2,5.

La FECHA_COMPRA se inicializa con la fecha del día CURRENT_DATE.

Resultado en la tabla TELEFONO, las siete líneas están en negrita:

NUMERO	TIPO	MARCA	FECHA_COMPRA	PRECIO	NUM_PROPIETARIO	COLOR
1	SP	1	15/01/10	159	190120	ROJO
2	SP	2	14/03/10	99	190215	
3	CT	3	02/05/10	49	190001	NEGRO
4	DE	4	25/07/10	89	190222	BLANCO
5	IP	5	30/09/10	359	190561	
6	DE	6	01/01/09	99	122120	BLANCO
7	SP	1	15/01/09	189	190622	ROJO
8	SP	2	10/03/11	0	190001	
9	**CT**	**2**	**11/03/11**	**147,5**		**ROJO**
10	**SP**	**3**	**11/03/11**	**122,5**		**ROSA**
11	**DE**	**5**	**11/03/11**	**47,5**		**NEGRO**
12	**IP**	**5**	**11/03/11**	**297,5**		**ROJO**
13	**SP**	**4**	**11/03/11**	**97,5**		**BLANCO**
14	**SP**	**1**	**11/03/11**	**222,5**		**ROJO**
15	**SP**	**2**	**11/03/11**	**35**		**VERDE**

El mismo ejemplo en MySQL:

```
INSERT INTO TELEFONO (TIPO, MARCA, COLOR, PRECIO, FECHA_COMPRA)
(SELECT TIPO,MARCA,COLOR,PRECIO_COMPRA*2.5,NOW() FROM COMPRA_TEL);
```

En este caso no se indica secuencia para el número de teléfono, MySQL no tiene el concepto de secuencia.

Por el contrario, dado que se ha declarado la columna como AUTO_INCREMENT, es el sistema el que asigna un número en cada fila insertada.

```
CREATE TABLE TELEFONO (NUMERO       INTEGER AUTO_INCREMENT, ...
```

La CURRENT_DATE se ha sustituido por la función NOW().

En este ejemplo hemos cogido todas las filas de la tabla COMPRA_TEL. También habríamos podido condicionar las filas con una cláusula WHERE.

Por ejemplo, recuperar las filas que sean del tipo « SP »:

```
INSERT INTO TELEFONO
```

```
   (NUMERO,TIPO, MARCA, COLOR, PRECIO, FECHA_COMPRA)
SELECT S_NUMERO.NEXTVAL,TIPO,MARCA,COLOR,
       PRECIO_COMPRA*2.5,CURRENT_DATE FROM COMPRA_TEL
WHERE TIPO = 'SP';
```

En este caso se pueden utilizar todas las posibilidades del SELECT.

3. Resumen de la sintaxis del INSERT

```
INSERT INTO <nombre tabla> (<nombre columna 1>, <nombre columna 2> ... ...)
VALUE (<valor 1>, <valor 2> ... ... )
```

o

```
INSERT INTO <nombre tabla> (<nombre columna 1>, <nombre columna 2> ... ...)
SELECT <nombre columna 1>, <nombre columna 2> ... ...
FROM <tabla 2>
[JOIN <tabla 3> ON <tabla2.columna1> = <tabla3.columna1>]
[JOIN <tabla 4> ON <tabla2.columna2> = <tabla4.columna2>]
[WHERE ... ... ... ]
[GROUP BY <nombre columna 1>, <nombre columna 2> ... ... ]
[HAVING [SUM/COUNT/AVG/MIN/MAX]<nombre columna 4> <operador
aritmético> <valor> ... ... ]
[ORDER BY <nombre columna 1>, <nombre columna 2> ...]
```

4. Ejercicios de inserción de datos

Estos ejercicios se basan en las tablas indicadas en la sección Ejercicios sobre la selección de datos de este capítulo.

Primer ejercicio

Añadir una nueva película en la tabla PELICULA:

Número 6, Jungla de Cristal 4, película de acción estrenada el 4 de julio de 2007. El director es Len Wiseman y la distribuye la Twentieth Century Fox Film Corporation.

Sinopsis: « En su cuarta aventura, el inspector John McClane se enfrenta a un nuevo tipo de terrorismo. La red informática nacional que controla todas las comunicaciones, los transportes y la Energía de los Estados Unidos está siendo saboteada de manera sistemática, sumiendo al país en el caos. »

Segundo ejercicio

Creación de una nueva tabla ACTOR_FRANCIA a partir de la tabla ACTOR seleccionando sólo los actores franceses.

Tercer ejercicio

Inserción de « Bruce Willis » nacido el 19 de marzo de 1955 con 89 películas en su haber y 152 días de rodaje en la tabla de actores e inserción de un registro en la tabla CASTING vinculándolo a la película número 6.

La eliminación de datos

La eliminación no es el comando más utilizado pero es importante conocer su sintaxis. Hay que poner mucha atención al utilizar el DELETE ya que frecuentemente se eliminan más filas de las previstas.

Para evitar esto, dos consejos: probar previamente con un SELECT la cláusula WHERE para verificar el número de filas afectadas, y utilizar con buen criterio los comandos COMMIT y ROLLBACK que detallaremos en el capítulo El control de transacciones (TCL).

1. El comando DELETE

El comando DELETE permitirá eliminar una o varias filas de una tabla.

La sintaxis del comando DELETE es simple si no limitamos las filas. Si no se indica la cláusula WHERE, se borrarán todas las filas de la tabla.

Si son tablas grandes, se desaconseja utilizar un DELETE sin WHERE, ya que existe riesgo de colapsar el sistema, que no podrá almacenar todas las filas borradas para poderlas restituir si se pide un ROLLBACK (ver capítulo Para ir más lejos).

Una consulta DELETE puede no realizarse si no se cumple una restricción de integridad al eliminar una fila.

Ejemplo de eliminación de todas las filas de una tabla:

```
DELETE FROM TELEFONO;
```

Para eliminar sólo algunas filas hay que añadir la cláusula WHERE.

Se pueden utilizar todas las posibilidades de la sintaxis utilizada para el SELECT.

Para eliminar las filas de la tabla TELEFONO que tienen la columna COLOR a NULL y cuya FECHA_COMPRA sea inferior a 31/12/2009.

Sintaxis Oracle:

```
DELETE FROM TELEFONO WHERE COLOR IS NULL AND FECHA_COMPRA <
TO_DATE('31/12/2009','DD/MM/YYYY');
```

Sintaxis MySQL:

```
DELETE FROM TELEFONO WHERE COLOR IS NULL AND
FECHA_COMPRA < STR_TO_DATE('31/12/2010','%d/%m/%Y');
```

También es posible condicionar la eliminación con un comando SELECT. Por ejemplo, para eliminar las filas de la tabla TELEFONO cuya MARCA no exista en la tabla MARCA_TEL.

```
DELETE FROM TELEFONO WHERE MARCA NOT IN (SELECT MARCA FROM
MARCA_TEL);
```

2. El comando TRUNCATE

El comando TRUNCATE permite eliminar TODAS las filas de una tabla sin posibilidad de condicionarlo.

No es un comando LMD, sino un comando LDD lo que significa que desencadena un COMMIT implícito. No se pueden recuperar datos después de un TRUNCATE, por lo que hay que utilizarlo con precaución.

En términos de rendimiento, el TRUNCATE es más rápido que el DELETE. El sistema hace menos operaciones internas con un TRUNCATE (no lo pasa por la caché, menos escritura en el log…).

Por el contrario, si existen claves foráneas, el TRUNCATE no funcionará. Igualmente hay que tener

privilegios especiales para utilizar el TRUNCATE. Normalmente los DBA no permiten utilizar este comando.

Tenga en cuenta también que los triggers no se desencadenan al ejecutar un TRUNCATE.

Este comando es para eliminar el contenido completo de una tabla que no está vinculada a ninguna otra y sobre la que estamos seguros de no querer hacer un ROLLBACK.

Sintaxis:

```
TRUNCATE TABLE <nombre tabla>;
```

Ejemplo:

```
TRUNCATE TABLE TIPO_TEL;
```

3. Ejercicios sobre la eliminación de datos

Estos ejercicios se basan en las tablas indicadas en la sección Ejercicios sobre la selección de datos de este capítulo.

Primer ejercicio

Eliminación de 'CARRIE FISHER' de la tabla ACTOR.

Segundo ejercicio

Eliminación de la tabla ESTADISTICA de las filas cuya recaudación mundial sea 0.

La modificación de datos

Hemos visto la inserción y la eliminación de datos; solo queda la modificación de una o más filas de una tabla con el comando UPDATE.

1. El comando UPDATE

Este comando es, sintácticamente, sencillo de implementar. Permite actualizar de 1 a n columnas de una tabla con el mismo comando.

Igual que el DELETE es posible añadir condiciones con la cláusula WHERE.

Ejemplo de modificación en la tabla TELEFONO:

```
UPDATE TELEFONO SET PRECIO = 99;
```

Si no se pone ninguna condición, **TODOS los precios** de la tabla TELEFONO serán 99.

NUMERO	TIPO	MARCA	FECHA_COMPRA	PRECIO	NUM_PROPIETARIO	COLOR
1	SP	1	15/01/10	**99**	190120	ROJO
2	SP	2	14/03/10	**99**	190215	
3	CT	3	02/05/10	**99**	190001	NEGRO
4	DE	4	25/07/10	**99**	190222	BLANCO
5	IP	5	30/09/10	**99**	190561	
6	DE	6	01/01/09	**99**	122120	BLANCO
7	SP	1	15/01/09	**99**	190622	ROJO
8	SP	2	10/03/11	**99**	190001	

Antes de ejecutar un comando UPDATE es importante comprobar el número de filas afectadas por esta modificación. Se aconseja hacer un SELECT para visualizar las filas seleccionadas.

En caso de error, siempre es posible ejecutar un ROLLBACK justo después del UPDATE para restablecer los datos de origen.

Ejemplo de modificación de una sola fila en la tabla TELEFONO:

```
UPDATE TELEFONO SET PRECIO = 99 WHERE PRECIO = 49;
```

Sólo la fila con el número 3 se actualiza.

NUMERO	TIPO	MARCA	FECHA_COMPRA	PRECIO	NUM_PROPIETARIO	COLOR
1	SP	1	15/01/10	159	190120	ROJO
2	SP	2	14/03/10	99	190215	
3	**CT**	**3**	**02/05/10**	**99**	**190001**	**NEGRO**
4	DE	4	25/07/10	89	190222	BLANCO
5	IP	5	30/09/10	359	190561	
6	DE	6	01/01/09	99	122120	BLANCO
7	SP	1	15/01/09	189	190622	ROJO
8	SP	2	10/03/11	0	190001	

```
UPDATE TELEFONO SET PRECIO = 99 WHERE PRECIO > 19 AND PRECIO < 100;
```

En este caso se actualizan cuatro filas: la 2, 3, 4 y 6. Observe que el sistema actualiza aunque el valor ya sea 99.

En el UPDATE se pueden utilizar todas las opciones del WHERE.

También se pueden realizar cálculos en la actualización. Si, por ejemplo, queremos aumentar el precio de los teléfonos un 20 %, hay que escribir:

```
UPDATE TELEFONO SET PRECIO = PRECIO * 1.20;
```

Para poder modificar varias columnas, hay que poner las columnas a continuación.

Por ejemplo, para modificar el precio, el color y la fecha de compra, la sintaxis es la siguiente:

```
UPDATE TELEFONO SET PRECIO = 99, COLOR = 'ROJO', FECHA_COMPRA =
TO_DATE('15/01/2011','DD/MM/YYYY')
WHERE PRECIO > 19 AND PRECIO < 100;
```

Es posible asignar un valor NULL a una columna introduciendo:

```
UPDATE TELEFONO SET COLOR = NULL WHERE PRECIO >= 99;
```

Sintaxis del comando UPDATE:

```
UPDATE <nombre tabla> SET <nombre columna 1> = <valor 1>,
 <nombre columna 2> = <valor 2>,
 <nombre columna 3> = <valor 3>,
 ... ...
WHERE ... ... ;
```

2. Ejercicios sobre la modificación de datos

Estos ejercicios se basan en las tablas indicadas en la sección Ejercicios sobre la selección de datos de este capítulo.

Primer ejercicio

Modificar el GENERO2 de la tabla PELICULA para completar el campo cuando no esté informado y sustituir 'SF' por 'CIENCIA FICCION'.

Segundo ejercicio

División de RECAUDACION_USA y RECAUDACION_MUNDIAL por 1.000.000 para expresar la cifra en millones en la tabla ESTADISTICA.

Soluciones de los ejercicios

1. Solución de los ejercicios sobre la selección de datos

a. Preguntas generales

1. ¿Cuáles son los cuatro tipos de unión descritos en este libro?

Interna, externa, natural y cruzada

2. ¿Cuál es la función que permite contar el número de ocurrencias?

COUNT

3. ¿Cuál es el comando para ordenar y, por defecto, la ordenación es descendente o ascendente?

ORDER BY, por defecto ascendente (ASC)

4. ¿Cuáles son las funciones de agrupamiento que se pueden utilizar con un GROUP BY?

Las más utilizadas son COUNT, SUM, AVG, MIN, MAX. También existen VARIANCE y STDDEV

5. ¿Cuáles son los tres operadores de conjuntos?

UNION, INTERSECT y EXCEPT

b. Ejercicios de aplicación

Primer ejercicio

Seleccionar toda la información sobre las películas realizadas por un director francés ordenado por el nombre de la película.

Primera posibilidad, se conoce el valor de columna NACIONALIDAD que corresponde a FRANCIA: 1.

```
SELECT * FROM PELICULA T1, DIRECTOR T2 WHERE
       T1.IDENT_DIRECTOR = T2.IDENT_DIRECTOR AND
       T2.NACIONALIDAD = 1
ORDER BY T1.TITULO;
```

Segunda posibilidad, se agrega una unión con la tabla PAIS y se iguala con la descripción « FRANCIA ».

otra posible sintaxis:

```
SELECT * FROM PELICULA T1, DIRECTOR T2, PAIS T3 WHERE
       T1.IDENT_DIRECTOR = T2.IDENT_DIRECTOR AND
       T2.NACIONALIDAD = T3.IDENT_PAIS AND
       T3.DESCRIPCION = 'FRANCIA'
ORDER BY T1.TITULO;
```

Resultados:

Se comprueba que TODAS las columnas de las tres tablas se tienen en cuenta. De hecho, hemos puesto * en la consulta y el sistema coge las columnas de la primera tabla, a continuación de la segunda y así sucesivamente.

```
SELECT * FROM PELICULA T1
       INNER JOIN DIRECTOR T2 ON T1.IDENT_DIRECTOR =
                                 T2.IDENT_DIRECTOR
       INNER JOIN PAIS T3        ON T2.NACIONALIDAD = T3.IDENT_PAIS
WHERE T3.DESCRIPCION = 'FRANCIA'
ORDER BY T1.TITULO;
```

Para tener un resultado más legible, hay que seleccionar algunas columnas del siguiente modo:

Resultados:

```
IDENT_PELICULA TITULO                                              GENERO1
GENERO2          FECHA_ESTRENO      PAIS IDENT_DIRECTOR DISTRIBUIDOR  SINOPSIS
IDENT_DIRECTOR APELLIDO                                            NOMBRE
FECHA_NAC NUM_PELICULA_ESCRITA NUM_PELICULA_PRODUCIDA NACIONALIDAD
IDENT_PAIS DESCRIPCION
```

```
         5 BIENVENIDOS AL NORTE                                    COMEDIA
27/02/08         1             4 PATHE        Philippe Abrams, director
de correos de Salon-de-Provence, es trasladado al Norte.
4 BOON                                                  DANY  26/06/66
5               1           1          1 FRANCIA
```

```
         2 NIKITA                                                  DRAMA
ROMANTICA           21/02/90          1                1 GAUMONT Nikita,
condenada a cadena perpetua es obligada a trabajar en secreto para el
gobierno como agente secreto.
1 BESSON                                                LUC   18/03/59
40              99           1          1 FRANCIA
```

Segundo ejercicio

Seleccionar el nombre de la película, la fecha de estreno, el nombre del director, el nombre de los actores, su fecha de nacimiento, así como el presupuesto de la película. Todo ello ordenado por el título de la película y el nombre de los actores descendentemente.

o

```
          1 SUBWAY                                                POLICIACA
  DRAMA                10/04/85          1                     1 GAUMONT
  Cuenta las aventuras de la población subterránea en los túneles
  del metro de París
  1 BESSON                                                LUC   18/03/59
  40              99           1          1 FRANCIA
```

```
SELECT T1.TITULO, T1.GENERO1, T1.FECHA_ESTRENO, T2.APELLIDO, T2.NOMBRE
FROM PELICULA T1, DIRECTOR T2, PAIS T3 WHERE
       T1.IDENT_DIRECTOR = T2.IDENT_DIRECTOR AND
       T2.NACIONALIDAD = T3.IDENT_PAIS AND
       T3.DESCRIPCION = 'FRANCIA'
ORDER BY T1.TITULO;
```

TITULO	GENERO1	FECHA_ESTRENO	APELLIDO	NOMBRE
BIENVENIDOS AL NORTE	COMEDIA	27/02/08	BOON	DANY
NIKITA	DRAMA	21/02/90	BESSON	LUC
SUBWAY	POLICIACA	10/04/85	BESSON	LUC

```
SELECT T1.TITULO, T1.FECHA_ESTRENO, T2.APELLIDO, T4.APELLIDO,
T4.FECHA_NACIMIENTO, T5.PRESUPUESTO
FROM PELICULA T1, DIRECTOR T2, CASTING T3, ACTOR T4, ESTADISTICA
T5 WHERE
   T1.IDENT_DIRECTOR = T2.IDENT_DIRECTOR AND
   T1.IDENT_PELICULA = T3.IDENT_PELICULA AND
   T1.IDENT_PELICULA = T5.IDENT_PELICULA AND
   T3.IDENT_ACTOR    = T4.IDENT_ACTOR
ORDER BY T1.TITULO, T4.APELLIDO DESC;
```

```
SELECT PELICULA.TITULO, PELICULA.FECHA_ESTRENO, DIRECTOR.APELLIDO,
ACTOR.APELLIDO, ACTOR.FECHA_NACIMIENTO, ESTA.PRESUPUESTO
FROM PELICULA PELICULA, DIRECTOR DIRECTOR, CASTING CAST, ACTOR ACTOR,
ESTADISTICA ESTA WHERE
   PELICULA.IDENT_DIRECTOR = DIRECTOR.IDENT_DIRECTOR AND
   PELICULA.IDENT_PELICULA = CAST.IDENT_PELICULA AND
   PELICULA.IDENT_PELICULA = ESTA.IDENT_PELICULA AND
   CAST.IDENT_ACTOR        = ACTOR.IDENT_ACTOR
ORDER BY PELICULA.TITULO, ACTOR.APELLIDO DESC;
```

En este ejemplo hemos asignado CAST como alias de la tabla CASTING. Ponga atención a las palabras reservadas por los SGBDR. En este caso en Oracle la sintaxis es válida, pero en MySQL no, ya que CAST es una palabra reservada.

Resultados:

TITULO	FECHA_ ESTRENO	APELLIDO	APELLIDO	FECHA_ NACIMIENTO	PRESUPUESTO
AVATAR	16/10/09	CAMERON	WEAVER	08/10/49	237
AVATAR	16/10/09	CAMERON	SALDANA	19/10/78	237
BIENVENIDOS AL NORTE	27/02/08	BOON	MERAD	27/03/64	11
BIENVENIDOS AL NORTE	27/02/08	BOON	BOON	26/06/66	11
NIKITA	21/02/90	BESSON	RENO	30/04/48	7,6
NIKITA	21/02/90	BESSON	PARILLAUD	06/05/60	7,6
STAR WARS 6: EL RETORNO DEL JEDI	19/10/83	LUCAS	FORD	13/06/42	32
STAR WARS 6: EL RETORNO DEL JEDI	19/10/83	LUCAS	FISHER	21/10/56	32
SUBWAY	10/04/85	BESSON	RENO	30/06/48	2,6
SUBWAY	10/04/85	BESSON	LAMBERT	29/03/57	2,6
SUBWAY	10/04/85	BESSON	GALABRU	27/10/22	2,6

Se puede ver que hay dos columnas con el mismo nombre ya que no hemos puesto ningún alias a los nombres de columna. Se habría podido escribir:

Tercer ejercicio

Encontrar el número de actores por película en la base de datos. Mostrar el título, la fecha de estreno, el apellido del director y el distribuidor.

Todas las columnas con las que no se opera deben aparecer en el GROUP BY. Así la película sólo se muestra una vez.

Resultados:

Cuarto ejercicio

Seleccionar el título de la película, la fecha de estreno,

SUBWAY	10/04/85	BESSON	BOHRINGER	16/06/42	2,6
SUBWAY	10/04/85	BESSON	ADJANI	27/06/55	2,6

```
SELECT PELICULA.TITULO, PELICULA.FECHA_ESTRENO, DIRECTOR.APELLIDO
DIRECTOR, ACTOR.APELLIDO ACTOR, ACTOR.FECHA_NACIMIENTO,
ESTA.PRESUPUESTO
FROM PELICULA PELICULA, DIRECTOR DIRECTOR, CASTING CAST, ACTOR ACTOR,
ESTADISTICA ESTA WHERE
   PELICULA.IDENT_DIRECTOR = DIRECTOR.IDENT_DIRECTOR AND
   PELICULA.IDENT_PELICULA = CAST.IDENT_PELICULA AND
   PELICULA.IDENT_PELICULA = ESTA.IDENT_PELICULA AND
   CAST.IDENT_ACTOR        = ACTOR.IDENT_ACTOR
ORDER BY PELICULA.TITULO, ACTOR.APELLIDO DESC;
```

```
SELECT T1.TITULO, T1.FECHA_ESTRENO, T2.APELLIDO,
T1.DISTRIBUIDOR,COUNT(T3.IDENT_PELICULA) AS NUM_ACTOR
FROM PELICULA T1, DIRECTOR T2, CASTING T3, ACTOR T4, ESTADISTICA
T5 WHERE
   T1.IDENT_DIRECTOR = T2.IDENT_DIRECTOR AND
   T1.IDENT_PELICULA = T3.IDENT_PELICULA AND
   T1.IDENT_PELICULA = T5.IDENT_PELICULA AND
   T3.IDENT_ACTOR    = T4.IDENT_ACTOR
GROUP BY T1.TITULO, T1.FECHA_ESTRENO, T2.APELLIDO, T1.DISTRIBUIDOR
ORDER BY T1.TITULO;
```

TITULO	FECHA_ ESTRENO	APELLIDO	DISTRIBUIDOR	NUM_ ACTOR
AVATAR	16/10/09	CAMERON	20th Century Fox	2
BIENVENIDOS AL NORTE	27/02/08	BOON	LAUREN FILMS	2
NIKITA	21/02/90	BESSON	FILMAX	2
STAR WARS 6: EL RETORNO DEL JEDI	19/10/83	LUCAS	20th Century Fox	2
SUBWAY	10/04/85	BESSON	FILMAX	5

```
SELECT PELICULA.TITULO, PELICULA.FECHA_ESTRENO,
       DIRECTOR.APELLIDO||' '||DIRECTOR.NOMBRE DIRECTOR, ACTOR.APELLIDO
APELLIDO, ACTOR.NOMBRE NOMBRE, ACTOR.FECHA_NACIMIENTO,
       ACTOR.NUM_PELICULA,ESTA.PRESUPUESTO, ESTA.NUM_ENTRADAS_ESPANA
ENTRADAS
FROM PELICULA PELICULA, DIRECTOR DIRECTOR, CASTING CAST, ACTOR ACTOR,
     ESTADISTICA ESTA, PAIS PAIS WHERE
   PELICULA.IDENT_DIRECTOR  = DIRECTOR.IDENT_DIRECTOR AND
   PELICULA.IDENT_PELICULA  = CAST.IDENT_PELICULA AND
   PELICULA.IDENT_PELICULA  = ESTA.IDENT_PELICULA AND
   CAST.IDENT_ACTOR         = ACTOR.IDENT_ACTOR AND
   PAIS.IDENT_PAIS          = ACTOR.NACIONALIDAD AND
   PAIS.DESCRIPCION         = 'ARGELIA'
ORDER BY PELICULA.TITULO;
```

TITULO	FECHA_ ESTRENO	DIRECTOR	APELLIDO	NOMBRE	FECHA_ NACIMIENTO	NUM_ PELICULA	PRESU- PUESTO	ENTRADAS
BIENVENIDOS AL NORTE	27/02/08	BOON DANY	MERAD	KAD	27/03/64	55	11	21000000

```
SELECT PELICULA.TITULO, PELICULA.FECHA_ESTRENO,
       DIRECTOR.APELLIDO||' '||DIRECTOR.NOMBRE DIRECTOR, ACTOR.APELLIDO
APELLIDO, ACTOR.NOMBRE NOMBRE, ACTOR.FECHA_NACIMIENTO,
       ACTOR.NUM_PELICULA,ESTA.PRESUPUESTO, ESTA.NUM_ENTRADAS_ESPANA
ENTRADAS
FROM PELICULA PELICULA, DIRECTOR DIRECTOR, CASTING CAST, ACTOR ACTOR,
     ESTADISTICA ESTA WHERE
   PELICULA.IDENT_DIRECTOR = DIRECTOR.IDENT_DIRECTOR AND
   PELICULA.IDENT_PELICULA = CAST.IDENT_PELICULA AND
   PELICULA.IDENT_PELICULA = ESTA.IDENT_PELICULA AND
   CAST.IDENT_ACTOR        = ACTOR.IDENT_ACTOR AND
   ACTOR.NACIONALIDAD IN (SELECT PAIS.IDENT_PAIS FROM PAIS WHERE
                          PAIS.DESCRIPCION = 'ARGELIA' )
ORDER BY PELICULA.TITULO;
```

```
SELECT PELICULA.TITULO, PELICULA.FECHA_ESTRENO,
       DIRECTOR.APELLIDO||' '||DIRECTOR.NOMBRE DIRECTOR, ACTOR.APELLIDO
```

el nombre y apellido del director, el nombre y apellido del actor, su fecha de nacimiento, el presupuesto de la película y el número de entradas en España de las películas que tienen un actor argelino.

Resultados:

En la consulta se puede observar otra forma de recuperar el apellido y el nombre en una sola columna utilizando los dobles pipes que sirven para concatenar dos columnas. En este ejemplo se unen el apellido y el nombre y se asigna al alias 'DIRECTOR'.

También podemos escribir la consulta de la siguiente forma:

o incluso:

Las tres sintaxis obtienen el mismo resultado. En términos de rendimiento, la primera sintaxis será sin duda la que lo tenga mejor, si la tabla país no tiene demasiadas filas. Las otras dos sintaxis, en términos de rendimiento, son equivalentes.

Quinto ejercicio

Seleccionar la película que ha tenido la recaudación más elevada en todo el mundo.

Resultado:

Sexto ejercicio

Seleccionar el actor que ha participado en dos películas diferentes.

```
APELLIDO, ACTOR.NOMBRE NOMBRE, ACTOR.FECHA_NACIMIENTO,
       ACTOR.NUM_PELICULA,ESTA.PRESUPUESTO, ESTA.NUM_ENTRADAS_ESPANA
ENTRADAS
FROM PELICULA PELICULA, DIRECTOR DIRECTOR, CASTING CAST, ACTOR ACTOR,
     ESTADISTICA ESTA WHERE
   PELICULA.IDENT_DIRECTOR = DIRECTOR.IDENT_DIRECTOR AND
   PELICULA.IDENT_PELICULA = CAST.IDENT_PELICULA AND
   PELICULA.IDENT_PELICULA = ESTA.IDENT_PELICULA AND
   CAST.IDENT_ACTOR        = ACTOR.IDENT_ACTOR AND
   EXISTS (SELECT PAIS.IDENT_PAIS FROM PAIS WHERE PAIS.IDENT_PAIS
         = ACTOR.NACIONALIDAD AND PAIS.DESCRIPCION = 'ARGELIA' )
ORDER BY PELICULA.TITULO;
```

```
SELECT PELICULA.TITULO, PELICULA.FECHA_ESTRENO,
       DIRECTOR.APELLIDO||' '||DIRECTOR.NOMBRE DIRECTOR,
ESTA.PRESUPUESTO, ESTA.NUM_ENTRADAS_ESPANA ENTRADAS,
ESTA.RECAUDACION_USA REC_USA,
ESTA.RECAUDACION_MUNDIAL REC_MUNDO
FROM PELICULA PELICULA, DIRECTOR DIRECTOR, ESTADISTICA ESTA
WHERE
   PELICULA.IDENT_DIRECTOR  = DIRECTOR.IDENT_DIRECTOR AND
   PELICULA.IDENT_PELICULA  = ESTA.IDENT_PELICULA AND
   ESTA.RECAUDACION_MUNDIAL = (SELECT MAX(RECAUDACION_MUNDIAL) FROM
ESTADISTICA)
ORDER BY PELICULA.TITULO;
```

TITULO	FECHA_ ESTRENO	DIRECTOR	PRESUPUESTO	ENTRADAS	REC_USA	REC_MUNDO
AVATAR	16/10/09	CAMERON JAMES	237	12018251	760505847	2946271769

```
SELECT  ACTOR.APELLIDO||' '||ACTOR.NOMBRE ACTOR,
ACTOR.FECHA_NACIMIENTO "NACIDO EL", PELICULA.TITULO TITULO,
PELICULA.FECHA_ESTRENO
"ESTRENADA EL", DIRECTOR.APELLIDO||' '||DIRECTOR.NOMBRE DIRECTOR
FROM PELICULA PELICULA, DIRECTOR DIRECTOR, CASTING CAST, ACTOR ACTOR
WHERE
   PELICULA.IDENT_DIRECTOR = DIRECTOR.IDENT_DIRECTOR AND
   PELICULA.IDENT_PELICULA = CAST.IDENT_PELICULA AND
   CAST.IDENT_ACTOR        = ACTOR.IDENT_ACTOR AND
   ACTOR.IDENT_ACTOR IN
       (SELECT IDENT_ACTOR FROM CASTING GROUP BY IDENT_ACTOR
                    HAVING COUNT(*) = 2)
ORDER BY PELICULA.TITULO, ACTOR.APELLIDO DESC;
```

Observe la utilización de comillas dobles en los alias para utilizar encabezados de columna significativos.

Resultados:

ACTOR	NACIDO EL	TITULO	ESTRENADA EL	DIRECTOR
RENO JEAN	30/06/48	NIKITA	21/02/90	BESSON LUC
RENO JEAN	30/06/48	SUBWAY	10/04/85	BESSON LUC

En este ejemplo, se recupera al mismo tiempo los títulos de las películas y los directores. Se utiliza una subconsulta para calcular el número de películas por actor y se comprueba el identificador del actor con una cláusula 'IN' la presencia o no del actor.

Otra posibilidad, si sólo se quiere recuperar el nombre del actor y no el nombre de las películas.

```
SELECT ACTOR.APELLIDO||' '||ACTOR.NOMBRE ACTOR,
ACTOR.FECHA_NACIMIENTO "NACIDO EL", COUNT(*) NUM_PELICULAS
FROM CASTING CAST, ACTOR ACTOR
WHERE
     CAST.IDENT_ACTOR = ACTOR.IDENT_ACTOR
GROUP BY ACTOR.APELLIDO||' '||ACTOR.NOMBRE, ACTOR.FECHA_NACIMIENTO
HAVING COUNT(*) = 2
ORDER BY ACTOR.APELLIDO||' '||ACTOR.NOMBRE;
```

Resultados:

ACTOR	NACIDO EL	NUM_PELICULAS
RENO JEAN	30/06/48	2

En este caso, no se cogen las películas asociadas, el HAVING COUNT sólo devuelve una fila. El título de la película estaba incluido en el SELECT y en el GROUP BY. Encuentra una fila con Jean Reno en Subway y una fila con Jean Reno en Nikita.

Séptimo ejercicio

Seleccionar la persona que es director y actor a la vez.

```
SELECT  ACTOR.APELLIDO||' '||ACTOR.NOMBRE ACTOR,
```

No sabiendo si para una persona el IDENT_ACTOR

```
ACTOR.FECHA_NACIMIENTO "NACIDO EL"
FROM PELICULA PELICULA, DIRECTOR DIRECTOR, CASTING CAST, ACTOR ACTOR
WHERE
   PELICULA.IDENT_DIRECTOR = DIRECTOR.IDENT_DIRECTOR AND
   PELICULA.IDENT_PELICULA = CAST.IDENT_PELICULA AND
   CAST.IDENT_ACTOR        = ACTOR.IDENT_ACTOR AND
   ACTOR.APELLIDO          = DIRECTOR.APELLIDO AND
   ACTOR.NOMBRE            = DIRECTOR.NOMBRE AND
   ACTOR.FECHA_NACIMIENTO  = DIRECTOR.FECHA_NACIMIENTO
ORDER BY ACTOR.APELLIDO||' '||ACTOR.NOMBRE;
```

es igual al IDENT_DIRECTOR, debemos comprobar por APELLIDO, NOMBRE y FECHA_NACIMIENTO para comparar las personas. Hay riesgo (mínimo) de que personas con el mismo nombre y apellido tengan la misma fecha de nacimiento.

Habría que evolucionar el modelo de datos para dar un identificador único a las personas y luego asignarles uno o más roles.

Resultados:

ACTOR	NACIDO EL
BOON DANY	20/06/66

Otra posibilidad sería comprobar los nombres, apellidos y fechas de nacimiento sin comprobar su presencia en la tabla PELICULA.

Octavo ejercicio

Seleccionar los actores que han participado en películas cuyo nombre comience por la letra 'S'. Indicar su papel y su nacionalidad.

```
SELECT  ACTOR.APELLIDO||' '||ACTOR.NOMBRE ACTOR,
ACTOR.FECHA_NACIMIENTO "NACIDO EL"
FROM DIRECTOR DIRECTOR, ACTOR ACTOR
WHERE
   ACTOR.APELLIDO          = DIRECTOR.APELLIDO AND
   ACTOR.NOMBRE            = DIRECTOR.NOMBRE AND
   ACTOR.FECHA_NACIMIENTO = DIRECTOR.FECHA_NACIMIENTO
ORDER BY ACTOR.APELLIDO||' '||ACTOR.NOMBRE;
```

o

```
SELECT ACTOR.APELLIDO||' '||ACTOR.NOMBRE ACTOR,
ACTOR.FECHA_NACIMIENTO "NACIDO EL",
CAST.ROLE, PAIS.DESCRIPCION, PELICULA.TITULO, PELICULA.FECHA_ESTRENO
FROM PELICULA PELICULA, CASTING CAST, ACTOR ACTOR, PAIS PAIS
WHERE
   PELICULA.IDENT_PELICULA = CAST.IDENT_PELICULA AND
   CAST.IDENT_ACTOR        = ACTOR.IDENT_ACTOR AND
   PAIS.IDENT_PAIS         = ACTOR.NACIONALIDAD AND
   SUBSTR(PELICULA.TITULO,1,1) = 'S'
ORDER BY ACTOR.APELLIDO||' '||ACTOR.NOMBRE, PELICULA.TITULO DESC;
```

En el primer script, se buscan todas las películas que tengan una 'S' en la primera posición y en el segundo se buscan todas las películas que comiencen por 'S' con 'LIKE S%'.

Resultados:

Noveno ejercicio

Seleccionar los actores que han nacido entre enero de 1948 y mayo de 1978 así como los días de rodaje total que han realizado.

```
SELECT ACTOR.APELLIDO||' '||ACTOR.NOMBRE ACTOR,
ACTOR.FECHA_NACIMIENTO "NACIDO EL",
CAST.ROLE, PAIS.DESCRIPCION, PELICULA.TITULO, PELICULA.FECHA_ESTRENO
FROM PELICULA PELICULA, CASTING CAST, ACTOR ACTOR, PAIS PAIS
WHERE
   PELICULA.IDENT_PELICULA = CAST.IDENT_PELICULA AND
   CAST.IDENT_ACTOR        = ACTOR.IDENT_ACTOR AND
   PAIS.IDENT_PAIS         = ACTOR.NACIONALIDAD AND
   PELICULA.TITULO     LIKE 'S%'
ORDER BY ACTOR.APELLIDO||' '||ACTOR.NOMBRE, PELICULA.TITULO DESC;
```

o

Resultados:

ACTOR	NACIDO EL	PAPEL	PAIS	TITULO	ESTRENADA EL
ADJANI ISABELLE	27/06/55	HELENA	FRANCIA	SUBWAY	10/04/85
BOHRINGER RICHARD	16/06/42	INSPECTOR GESBERG	FRANCIA	SUBWAY	10/04/85
FISHER CARRIE	21/10/56	PRINCESA LEIA	USA	STAR WARS 6: EL RETORNO DEL JEDI	19/10/83
FORD HARRISON	13/06/42	HAN SOLO	USA	STAR WARS 6: EL RETORNO DEL JEDI	19/10/83
GALABRU MICHEL	27/10/22	LA FLORISTA	FRANCIA	SUBWAY	10/04/85
LAMBERT CHRISTOPHE	29/03/57	FRED	FRANCIA	SUBWAY	10/04/85
RENO JEAN	30/06/48	EL BATERÍA	FRANCIA	SUBWAY	10/04/85

2. Solución de los eje

```
SELECT ACTOR.APELLIDO||' '||ACTOR.NOMBRE ACTOR,
ACTOR.FECHA_NACIMIENTO "NACIDO EL",
      SUM(CAST.NUM_DIAS_RODAJE) NUM_DIAS
FROM CASTING CAST, ACTOR ACTOR
WHERE
   CAST.IDENT_ACTOR        = ACTOR.IDENT_ACTOR AND
  ACTOR.FECHA_NACIMIENTO >= TO_DATE('01/01/1948','DD/MM/YYYY') AND
  ACTOR.FECHA_NACIMIENTO <= TO_DATE('31/05/1978','DD/MM/YYYY')
GROUP BY ACTOR.APELLIDO||' '||ACTOR.NOMBRE,ACTOR.FECHA_NACIMIENTO
ORDER BY ACTOR.APELLIDO||' '||ACTOR.NOMBRE ASC;
```

```
SELECT ACTOR.APELLIDO||' '||ACTOR.NOMBRE ACTOR,
ACTOR.FECHA_NACIMIENTO "NACIDO EL",
       SUM(CAST.NUM_DIAS_RODAJE) NUM_DIAS
FROM CASTING CAST, ACTOR ACTOR
WHERE
   CAST.IDENT_ACTOR      = ACTOR.IDENT_ACTOR AND
   ACTOR.FECHA_NACIMIENTO BETWEEN
       TO_DATE('01/01/1948','DD/MM/YYYY') AND
       TO_DATE('31/05/1978','DD/MM/YYYY')
GROUP BY ACTOR.APELLIDO||' '||ACTOR.NOMBRE,ACTOR.FECHA_NACIMIENTO
ORDER BY ACTOR.APELLIDO||' '||ACTOR.NOMBRE ASC;
```

ACTOR	NACIDO EL	NUM_DIAS
ADJANI ISABELLE	27/06/55	100
BOON DANY	26/06/66	125
FISHER CARRIE	21/10/56	203
LAMBERT CHRISTOPHE	29/03/57	100
MERAD KAD	27/03/64	126
PARILLAUD ANNE	06/05/60	68
RENO JEAN	30/06/48	29
WEAVER SIGOURNEY	08/10/49	45

Primer ejercicio

Añadir una nueva película en la tabla PELICULA:

Número 6, Jungla de Cristal 4, película de acción estrenada el 4 de julio de 2007. El director es Len Wiseman y la distribuye la Twentieth Century Fox Film Corporation.

Sinopsis: « En su cuarta aventura, el inspector John McClane se enfrenta a nuevo tipo de terrorismo. La red informática nacional que controla todas las comunicaciones, los transportes y la energía de los Estados Unidos es saboteada de manera sistemática, sumiendo al país en el caos. »

Recuperar el número de película más alto.

Crear la nueva fila:

```
SELECT MAX(IDENT_PELICULA) FROM PELICULA;

MAX(IDENT_PELICULA)
---------------
              5
```

- En Oracle:

```
INSERT INTO PELICULA VALUES (6,'JUNGLA DE CRISTA 4','ACCION','',
TO_DATE('04/07/2007','DD/MM/YYYY'),2,5,'Twentieth
Century Fox Film Corporation',' En su cuarta aventura, el inspector
John McClane se enfrenta a nuevo tipo de terrorismo. La red informática
nacional que controla todas las comunicaciones, los transportes y la
energía de los Estados Unidos es saboteada de manera sistemática,
sumiendo al país en el caos.');
```

- En MySQL:

```
INSERT INTO PELICULA VALUES (6,'JUNGLA DE CRISTAL 4','ACCION','',
STR_TO_DATE('04/07/2007','%d/%m/%Y'),2,5,'Twentieth
Century Fox Film Corporation',' En su cuarta aventura, el inspector
John McClane se enfrenta a nuevo tipo de terrorismo. La red informática
nacional que controla todas las comunicaciones, los transportes y la
energía de los Estados Unidos es saboteada de manera sistemática,
sumiendo al país en el caos.');
```

Segundo ejercicio

Creación de una nueva tabla ACTOR_FRANCIA a partir de la tabla ACTOR seleccionando sólo actores franceses.

Contenido de la tabla ACTOR_FRANCIA:

```
CREATE TABLE ACTOR_FRANCIA AS SELECT * FROM ACTOR WHERE
NACIONALIDAD = (SELECT IDENT_PAIS FROM PAIS WHERE DESCRIPCION =
'FRANCIA');
```

IDENT_ ACTOR	APELLIDO	NOMBRE	FECHA_ NACIMIENTO	NUM_ PELICULA	NACIONALIDAD
1	ADJANI	ISABELLE	27/06/55	42	1
2	LAMBERT	CHRISTOPHE	29/03/57	64	1
3	BOHRINGER	RICHARD	16/06/42	132	1
4	GALABRU	MICHEL	27/10/22	277	1

Tercer ejercicio

Inserción de « Bruce Willis » nacido el 19 de marzo de 1955 con 89 películas en su haber y 152 días de rodaje en la tabla de actores e inserción de un registro en la tabla CASTING vinculándolo a la película número 6.

5	PARILLAUD	ANNE	06/05/60	35	1
10	RENO	JEAN	30/06/48	75	1
11	BOON	DANY	26/06/66	23	1

Recuperar el número de actor más alto:

```
SELECT MAX(IDENT_ACTOR) FROM ACTOR;
```

o utilización de una subconsulta para recuperar el número más alto +1:

Para insertar en la tabla CASTING:

```
INSERT INTO ACTOR VALUES ((SELECT MAX(IDENT_ACTOR)+1 FROM
ACTOR),'WILLIS','BRUCE',TO_DATE('19/03/1955','DD/MM/YYYY'),152,2);
```

```
INSERT INTO CASTING VALUES (6,12,'John McClane',152);
```

3. Soluciones de los ejercicios sobre la eliminación de datos

Primer ejercicio

Eliminación de 'CARRIE FISHER' de la tabla ACTOR.

Comprobación antes de la eliminación de que la consulta sólo elimina una fila:

```
SELECT COUNT(*) FROM ACTOR WHERE APELLIDO = 'FISHER' AND NOMBRE =
'CARRIE';


  COUNT(*)
----------
         1
```

Y ejecución de la eliminación.

Segundo ejercicio

Eliminación de la tabla ESTADISTICA de las filas cuya recaudación mundial sea 0.

```
DELETE FROM ACTOR WHERE APELLIDO = 'FISHER' AND NOMBRE = 'CARRIE';
```

Comprobación del número de filas afectadas.

y ejecución de la eliminación:

```
SELECT COUNT(*) FROM ESTADISTICA WHERE RECAUDACION_MUNDIAL = 0;


  COUNT(*)
----------
         1
```

```
DELETE FROM ESTADISTICA WHERE RECAUDACION_MUNDIAL = 0;
```

4. Solución de los ejercicios sobre la modificación de datos

Primer ejercicio

Modificar el GENERO2 de la tabla PELICULA para completar los campos vacíos y sustituir SF' por 'CIENCIA FICCION'.

Comprobación de las filas afectadas:

```
SELECT IDENT_PELICULA,TITULO,GENERO1,GENERO2 FROM PELICULA WHERE
GENERO2 IS NULL OR GENERO2 = ' ';
```

Comprobación:

IDENT_PELICULA	TITULO	GENERO1	GENERO2
5	BIENVENIDOS AL NORTE	COMEDIA	
6	JUNGLA DE CRISTAL 4	ACCION	

Segundo ejercicio

División de RECAUDACION_USA y RECAUDACION_MUNDIAL por 1.000.000 para expresar la cifra en millones en la tabla ESTADISTICA.

```
UPDATE PELICULA SET GENERO2 = 'AVENTURAS' WHERE GENERO2 IS NULL OR
GENERO2 = ' ';
```

Contenido de la tabla ESTADISTICA:

```
SELECT IDENT_PELICULA,TITULO,GENERO1,GENERO2 FROM PELICULA WHERE
GENERO2 = 'AVENTURAS';
```

IDENT_PELICULA	TITULO	GENERO1	GENERO2
5	BIENVENIDOS AL NORTE	COMEDIA	AVENTURAS
6	JUNGLA DE CRISTAL 4	ACCION	AVENTURAS

```
UPDATE ESTADISTICA SET RECAUDACION_MUNDIAL = RECAUDACION_MUNDIAL /
```

```
1000000;
```

IDENT_ PELICULA	DURACION	NUM_ ENTRADAS_ ESPAÑA	RECAUDACION_ USA	RECAUDACION_ MUNDIAL	PRESUPUESTO
1	104	2 917 562	390 659	**1,27**	2,6
2	118	3 787 845	5 017 971	**0,00**	7,6
3	133	4 263 000	191 648 000	**472,00**	32
4	170	12 018 251	760 505 847	**2946,27**	237
5	100	21 000 000	0	**245,00**	11

Introducción

Las funciones son muy variadas y a menudo se implementan de manera diferente en cada SGBDR. No vamos a detallar todas las existentes. Abordaremos las más comunes y utilizadas.

Las funciones numéricas

Se pueden utilizar todos los operadores: +, -, *, / y también funciones como el valor absoluto, el coseno, los logaritmos, el módulo, el redondeo, etc.

Consulte la documentación del SGBDR para conocer las funciones que se implementan en la versión de la base de datos utilizada.

No vamos a ser muy exhaustivos en todas las funciones existentes, pero vamos a describir algunas funciones implementadas en los SGBDR.

Para ilustrar los ejemplos en los siguientes capítulos, vamos a añadir una columna a la tabla TELEFONO, se trata de la columna VARIACION (variación del precio) utilizando la sintaxis ALTER TABLE.

```
ALTER TABLE TELEFONO ADD (VARIACION NUMBER);
```

El contenido de la tabla TELEFONO ahora es:

NUMERO	TIPO	MARCA	FECHA_ COMPRA	PRECIO	NUM_ PROPIETARIO	COLOR	VARIA- CION
1	SP	1	15/01/10	159	190120	ROJO	2,3
2	SP	2	14/03/10	99	190215		4,2
3	CT	3	02/05/10	49	190001	NEGRO	-1
4	DE	4	25/07/10	89	190222	BLANCO	3
5	IP	5	30/09/10	359	190561		-1,2
6	DE	5	01/01/09	99	122120	BLANCO	0
7	SP	1	15/01/09	189	190622	ROJO	-1,15
8	SP	2	10/03/11	0	190001		3,2

1. ABS: valor absoluto

```
SELECT NUMERO, FECHA_COMPRA, PRECIO, ABS(VARIACION) FROM TELEFONO;
```

NUMERO	FECHA_COMPRA	PRECIO	VARIACION
1	15/01/10	159	2,3
2	14/03/10	99	4,2
3	02/05/10	49	1
4	25/07/10	89	3
5	30/09/10	359	1,2
6	01/01/09	99	0
7	15/01/09	189	1,15
8	10/03/11	0	3,2

2. ASCII: valor ASCII de un carácter

En este ejemplo vamos a mostrar el código ASCII del primer carácter de la columna COLOR para algunas filas:

```
SELECT NUMERO, COLOR, ASCII(SUBSTR(COLOR,1,1))AS CODIGO FROM
TELEFONO
WHERE PRECIO IN (49,99,189);
```

NUMERO	COLOR	CODIGO
2		
3	NEGRO	78
6	BLANCO	66
7	ROJO	82

3. COS: coseno - SIN: seno

En este ejemplo se va a mostrar el coseno del precio para los teléfonos de color BLANCO o NEGRO.

```
SELECT NUMERO, FECHA_COMPRA, PRECIO, COS(PRECIO) AS COSENO FROM
TELEFONO WHERE COLOR IN ('BLANCO','NEGRO');
```

NUMERO	FECHA_COMPRA	PRECIO	COSENO
3	02/05/10	49	0,3005925
4	25/07/10	89	0,5101770
6	01/01/09	99	0,0398209

Para obtener el seno, utilice la misma sintaxis:

```
SELECT NUMERO, FECHA_COMPRA, PRECIO, SIN(PRECIO) AS SENO FROM TELEFONO
WHERE COLOR IN ('BLANCO','NEGRO');
```

4. LOG (<número base>,<columna>): logaritmo de la columna seleccionada en la base indicada

En este ejemplo se va a mostrar el logaritmo en base 2 del precio para los teléfonos de color BLANCO.

```
SELECT NUMERO, FECHA_COMPRA, PRECIO, LOG(2,PRECIO) AS LOG FROM TELEFONO
WHERE COLOR = 'BLANCO';
```

NUMERO	FECHA_COMPRA	PRECIO	LOG
4	25/07/10	89	6,4757334
6	01/01/09	99	6,6293566

5. MOD(<columna>,<valor>): módulo

El módulo devuelve el resto de la división de una columna por un valor. En este ejemplo se va a mostrar el módulo del precio dividido por 4.

```
SELECT NUMERO, FECHA_COMPRA, PRECIO, MOD(PRECIO,4) AS MODULO FROM
TELEFONO
WHERE COLOR = 'BLANCO';
```

NUMERO	FECHA_COMPRA	PRECIO	MODULO
4	25/07/10	89	1
6	01/01/09	99	3

6. ROUND(<columna>,[<precisión>]): redondeo

El redondeo retorna el entero más cercano de un valor decimal. En este ejemplo, se va a mostrar el redondeo de la columna variación para los teléfonos de color ROJO.

Tenga en cuenta que el redondeo 2,5 será 2 y el redondeo -2,5 será -3.

```
SELECT NUMERO, FECHA_COMPRA, VARIACION, ROUND(VARIACION) AS REDONDEO
FROM TELEFONO
WHERE COLOR = 'ROJO';
```

NUMERO	FECHA_COMPRA	VARIACION	REDONDEO
1	15/01/10	2,3	2
7	15/01/09	-1,15	-1

También es posible precisar el número de cifras tras la coma indicando un valor en el campo <precisión>.

```
SELECT NUMERO, FECHA_COMPRA, VARIACION, ROUND(VARIACION,1) AS REDONDEO
FROM TELEFONO
WHERE COLOR = 'ROJO';
```

NUMERO	FECHA_COMPRA	VARIACION	REDONDEO
1	15/01/10	2,3	2
7	15/01/09	-1,15	-1

7. SQRT: raíz cuadrada

Este ejemplo muestra la raíz cuadrada del precio para los teléfonos rojos.

```
SELECT NUMERO, FECHA_COMPRA, PRECIO, SQRT(PRECIO) AS RAIZ FROM
TELEFONO
WHERE COLOR = 'ROJO';
```

NUMERO	FECHA_COMPRA	PRECIO	RAIZ
1	15/01/10	159	12,6095
7	15/01/09	189	13,7477

Las funciones de comparación y de comprobación

La necesidad de comprobar la validez de un valor interviene a menudo en las cláusulas WHERE y HAVING. Puede ser necesario comprobar la presencia de tal o cual valor en una lista (operador IN) o comprobar los límites inferiores y superiores (operador BETWEEN) de valores, o incluso buscar un valor aproximado (operador LIKE).

1. IN - NOT IN

Este operador permite comprobar la existencia de un valor en una lista. Esta lista se puede construir con valores fijos numéricos o alfanuméricos, o con la ayuda de una subconsulta que devuelva el contenido de una tabla.

Ejemplo de consulta que busca los teléfonos que son de color rojo o blanco:

```
SELECT TEL.FECHA_COMPRA,
       TEL.TIPO,
       TIP.DESC_TIPO,
       TEL.MARCA,
       MAR.DESC_MARCA,
       TEL.COLOR
FROM  TELEFONO TEL,
      TIPO_TEL TIP,
      MARCA_TEL MAR
WHERE TEL.TIPO  = TIP.TIPO
  AND TEL.MARCA = MAR.MARCA
  AND TEL.COLOR IN ('ROJO', 'BLANCO');
```

FECHA_COMPRA	TIPO	DESC_TIPO	MARCA	DESC_MARCA	COLOR
15/01/09	SP	SMARTPHONE	1	SAMSUNG	ROJO
15/01/10	SP	SMARTPHONE	1	SAMSUNG	ROJO
27/07/10	DE	CON TAPA	4	MOTOROLA	BLANCO
01/01/09	DE	DESLIZANTE	5	APPLE	BLANCO

Otro ejemplo con el operador « NOT IN », para buscar los teléfonos que no son ni ROJO ni BLANCO:

```
SELECT TEL.FECHA_COMPRA,
       TEL.TIPO,
       TIP.DESC_TIPO,
       TEL.MARCA,
       MAR.DESC_MARCA,
       TEL.COLOR
FROM  TELEFONO TEL,
      TIPO_TEL TIP,
      MARCA_TEL MAR
WHERE TEL.TIPO  = TIP.TIPO
  AND TEL.MARCA = MAR.MARCA
  AND TEL.COLOR NOT IN ('ROJO', 'BLANCO');
```

FECHA_COMPRA	TIPO	DESC_TIPO	MARCA	DESC_MARCA	COLOR
02/05/10	CL	CON TAPA	3	PHILIPS	NEGRO

Se pueden utilizar las dos opciones a la vez, si por ejemplo se quieren los teléfonos rojos o blancos pero que no sean de la marca SAMSUNG ni APPLE:

```
SELECT TEL.FECHA_COMPRA,
       TEL.TIPO,
```

```
       TIP.DESC_TIPO,
       TEL.MARCA,
       MAR.DESC_MARCA,
       TEL.COLOR
FROM  TELEFONO TEL,
      TIPO_TEL TIP,
      MARCA_TEL MAR
WHERE TEL.TIPO  = TIP.TIPO
  AND TEL.MARCA = MAR.MARCA
  AND TEL.COLOR IN ('ROJO', 'BLANCO')
  AND MAR.DESC_MARCA NOT IN ('SAMSUNG','APPLE');
```

Sólo queda el MOTOROLA de color BLANCO que corresponde a la selección:

FECHA_COMPRA	TIPO	DESC_TIPO	MARCA	DESC_MARCA	COLOR
27/07/10	CO	CON TAPA	4	MOTOROLA	BLANCO

El IN (y el NOT IN) también se pueden utilizar creando la lista de valores con una selección de una tabla. Si, por ejemplo, se quieren recuperar todos los teléfonos que son de un fabricante americano. Hay que hacer la siguiente consulta:

```
SELECT TEL.NUMERO,
       TEL.TIPO,
       TEL.MARCA,
       TIP.DESC_TIPO,
       TEL.COLOR
FROM  TELEFONO TEL,
      TIPO_TEL TIP
WHERE TEL.TIPO   = TIP.TIPO
  AND TEL.MARCA IN (SELECT MAR.MARCA FROM MARCA_TEL MAR
                    WHERE MAR.PAIS = 'USA');
```

Los teléfonos 4,5 y 6 han sido fabricados por las marcas 4 y 5 (Motorola y Apple) que son americanas. El resultado de la subconsulta tiene como resultado (4,5).

```
SELECT MAR.MARCA FROM MARCA_TEL MAR
WHERE MAR.PAIS = 'USA';
```

Así pues, el sistema traducirá internamente el resultado de la consulta principal por «AND TEL.MARCA IN (4,5);».

NUMERO	TIPO	MARCA	DESC_TIPO	COLOR
4	DE	4	DESLIZANTE	BLANCO
5	IP	5	IPHONE	
6	DE	5	DESLIZANTE	BLANCO

Con un **HAVING** la utilización de un IN o de un NOT IN se aplica sobre el resultado de la función utilizada por el Having. Por ejemplo, si se quiere recuperar los teléfonos por tipo y cuyo redondeo de la raíz cuadrada sea múltiplo de 7 (7,14,21).

El contenido de la base es el siguiente:

TIPO	DESC_TIPO	PRECIO
CT	CON TAPA	49
DE	DESLIZANTE	99
DE	DESLIZANTE	89
IP	IPHONE	359

SP	DESLIZANTE	99
SP	SMARTPHONE	159
SP	SMARTPHONE	189
SP	SMARTPHONE	0

```
Total CT = 49, raíz: 7
Total DE = 188, raíz: 14
Total IP = 359, raíz: 19
Total SP = 447, raíz: 21
```

Ejecutamos la siguiente consulta:

```
SELECT TEL.TIPO,
       TIP.DESC_TIPO,
       SUM(PRECIO) P_TOTAL,
       ROUND(SQRT(SUM(PRECIO))) AS RAIZ
FROM  TELEFONO TEL,
      TIPO_TEL TIP
WHERE TEL.TIPO   = TIP.TIPO
GROUP BY TEL.TIPO,TIP.DESC_TIPO
HAVING ROUND(SQRT(SUM(PRECIO))) IN (7,14,21);
```

Aquí se puede ver que se pueden encadenar funciones en la misma consulta.

TIPO	DESC_TIPO	P_TOTAL	RAIZ
SP	SMARTPHONE	447	21
DE	DESLIZANTE	188	14
CT	CON TAPA	49	7

Observe que el IN casi siempre se puede sustituir por una unión entre dos tablas. Si la tabla que es la base para los valores del IN tiene muchas filas, es preferible hacer una unión para limitar los posibles problemas de rendimiento. De hecho, el SGBDR compara el valor buscado con cada uno de los valores contenidos en el IN, por lo que el tiempo de respuesta puede ser largo.

Por ejemplo, la siguiente consulta:

```
SELECT TEL.NUMERO,
       TEL.TIPO,
       TEL.MARCA,
       TIP.DESC_TIPO,
       TEL.COLOR
FROM  TELEFONO TEL,
      TIPO_TEL TIP
WHERE TEL.TIPO   = TIP.TIPO
  AND TEL.MARCA IN (SELECT MAR.MARCA FROM MARCA_TEL MAR
                    WHERE MAR.PAIS = 'USA');
```

se puede sustituir por esta:

```
SELECT TEL.NUMERO,
       TEL.TIPO,
       TEL.MARCA,
       TIP.DESC_TIPO,
       TEL.COLOR
FROM  TELEFONO TEL,
      TIPO_TEL TIP,
      MARCA_TEL MAR
WHERE TEL.TIPO   = TIP.TIPO
  AND TEL.MARCA = MAR.MARCA
```

```
  AND MAR.PAIS = 'USA';
```

o incluso por esta:

```
SELECT TEL.NUMERO,
       TEL.TIPO,
       TEL.MARCA,
       TIP.DESC_TIPO,
       TEL.COLOR
FROM TELEFONO TEL INNER JOIN TIPO_TEL TIP  ON TEL.TIPO =
TIP.TIPO
                  INNER JOIN MARCA_TEL MAR ON TEL.MARCA =
MAR.MARCA
 WHERE MAR.PAIS = 'USA';
```

Las tres consultas devuelven exactamente el mismo resultado:

NUMERO	TIPO	MARCA	DESC_TIPO	COLOR
4	DE	4	DESLIZANTE	BLANCO
5	IP	5	IPHONE	
6	DE	5	DESLIZANTE	BLANCO

Como conclusión, podemos decir que el IN (NOT IN) abre muchas posibilidades, aunque hay que poner atención a los problemas de rendimiento y no dude en probar la unión como posible alternativa.

2. EXISTS - NOT EXISTS (existencia o no)

Este operador permite comprobar la existencia o no de un valor en otra tabla. A menudo se trata de realizar una subconsulta dentro de una consulta.

Ejemplo de consulta que busca todos los teléfonos que tienen una marca que existe en la tabla MARCA:

```
SELECT TEL.NUMERO,
       TEL.TIPO,
       TEL.MARCA,
       TIP.DESC_TIPO,
       TEL.COLOR,
       TEL.PRECIO
FROM  TELEFONO TEL,
      TIPO_TEL TIP
WHERE TEL.TIPO   = TIP.TIPO AND
      EXISTS (SELECT MAR.MARCA FROM MARCA_TEL MAR WHERE
MAR.MARCA = TEL.MARCA);
```

NUMERO	TIPO	MARCA	DESC_TIPO	COLOR	PRECIO
1	SP	1	SMARTPHONE	ROJO	159
2	SP	2	SMARTPHONE		99
3	CT	3	CON TAPA	NEGRO	49
4	DE	4	DESLIZANTE	BLANCO	89
5	IP	5	IPHONE		359
6	DE	5	DESLIZANTE	BLANCO	99
7	SP	1	SMARTPHONE	ROJO	189
8	SP	2	SMARTPHONE		

Del mismo modo, esta vez buscaremos los teléfonos cuya marca no exista en la tabla MARCA.

```
SELECT TEL.NUMERO,
       TEL.TIPO,
       TEL.MARCA,
       TIP.DESC_TIPO,
       TEL.COLOR,
       TEL.PRECIO
FROM  TELEFONO TEL,
      TIPO_TEL TIP
WHERE TEL.TIPO   = TIP.TIPO AND
      NOT EXISTS (SELECT MAR.MARCA FROM MARCA_TEL MAR WHERE
MAR.MARCA = TEL.MARCA);
```

3. BETWEEN (entre dos valores)

El comando BETWEEN permitirá limitar una comprobación con valores mínimo y máximo.

Por ejemplo, si se quieren seleccionar los teléfonos que tienen un precio entre 100 y 400 euros, escribiremos:

```
SELECT TEL.NUMERO,
       TEL.TIPO,
       TEL.MARCA,
       TIP.DESC_TIPO,
       TEL.COLOR,
       TEL.PRECIO
FROM  TELEFONO TEL,
      TIPO_TEL TIP
WHERE TEL.TIPO   = TIP.TIPO
  AND TEL.PRECIO BETWEEN 100 AND 400;
```

NUMERO	TIPO	MARCA	DESC_TIPO	COLOR	PRECIO
1	SP	1	SMARTPHONE	ROJO	159
5	IP	5	IPHONE		359
7	SP	1	SMARTPHONE	ROJO	189

También se pueden utilizar sub-SELECT para sustituir los límites. Por ejemplo para buscar los teléfonos cuyo precio mínimo sea el mínimo de una tabla de referencia y el precio máximo el precio máximo de otra tabla de referencia.

```
CREATE TABLE TAB_REF1 (PRECIO_REF NUMBER);
CREATE TABLE TAB_REF2 (PRECIO_MAX NUMBER);
INSERT INTO TAB_REF1 VALUES (59);
INSERT INTO TAB_REF1 VALUES (79);
INSERT INTO TAB_REF1 VALUES (99);
INSERT INTO TAB_REF1 VALUES (109);
INSERT INTO TAB_REF2 VALUES (899);
INSERT INTO TAB_REF2 VALUES (999);
INSERT INTO TAB_REF2 VALUES (699);

SELECT TEL.NUMERO,
       TEL.TIPO,
       TEL.MARCA,
       TIP.DESC_TIPO,
       TEL.COLOR,
       TEL.PRECIO
FROM  TELEFONO TEL,
      TIPO_TEL TIP
WHERE TEL.TIPO   = TIP.TIPO
  AND TEL.PRECIO BETWEEN
      (SELECT MIN(PRECIO_REF) FROM TAB_REF1) AND
      (SELECT MAX(PRECIO_MAX) FROM TAB_REF2);
```

El MIN(PRECIO_REF) = 59 y el MAX(PRECIO_MAX) = 999, con lo que la consulta devolverá todas las filas de teléfonos que tengan un precio entre 59 y 999.

NUMERO	TIPO	MARCA	DESC_TIPO	COLOR
1	SP	1	SMARTPHONE	ROJO
2	SP	2	SMARTPHONE	
4	DE	4	DESLIZANTE	BLANCO
5	IP	5	IPHONE	
6	DE	5	DESLIZANTE	BLANCO
7	SP	1	SMARTPHONE	ROJO

Se pueden utilizar todas las combinaciones con los sub-SELECT pero piense siempre en el rendimiento, que puede disminuir con tablas grandes.

4. LIKE (que contiene parte del valor)

El LIKE permite buscar valores que contienen una parte de una cadena de caracteres. Puede que sea necesario buscar todos los teléfonos fabricados por una empresa cuyo país comience por la letra U.

Después de la U situaremos el carácter % que significa que se cogen todas las descripciones de la tabla PAIS que comiencen por U.

```
SELECT TEL.FECHA_COMPRA,
       TEL.TIPO,
       TIP.DESC_TIPO,
       TEL.MARCA,
       MAR.DESC_MARCA,
       TEL.COLOR,
       MAR.PAIS
FROM  TELEFONO TEL,
      TIPO_TEL TIP,
      MARCA_TEL MAR
WHERE TEL.TIPO  = TIP.TIPO
  AND TEL.MARCA = MAR.MARCA
  AND MAR.PAIS LIKE 'U%';
```

Sólo quedan el MOTOROLA y los APPLE de color BLANCO que corresponden a la selección:

FECHA_COMPRA	TIPO	DESC_TIPO	MARCA	DESC_MARCA	COLOR	PAIS
27/07/10	CO	CON TAPA	4	MOTOROLA	BLANCO	USA
30/09/10	IP	IPHONE	5	APPLE		USA
01/01/89	CO	DESLIZANTE	5	APPLE	BLANCO	USA

Si ahora se quieren los países que comiencen por U o que tengan una A en su nombre:

```
SELECT TEL.FECHA_COMPRA,
       TEL.TIPO,
       TIP.DESC_TIPO,
       TEL.MARCA,
       MAR.DESC_MARCA,
       TEL.COLOR,
       MAR.PAIS
FROM  TELEFONO TEL,
      TIPO_TEL TIP,
      MARCA_TEL MAR
```

```
WHERE TEL.TIPO  = TIP.TIPO
  AND TEL.MARCA = MAR.MARCA
  AND (MAR.PAIS LIKE 'U%'OR
       MAR.PAIS LIKE '%A%');
```

FECHA_COMPRA	TIPO	DESC_TIPO	MARCA	DESC_MARCA	COLOR	PAIS
14/03/10	SP	SMARTPHONE	2	SONY		JAPON
10/03/11	SP	SMARTPHONE	2	SONY		JAPON
02/05/10	CT	CON TAPA	3	PHILIPS	NEGRO	HOLANDA
27/07/10	DE	CON TAPA	4	MOTOROLA	BLANCO	USA
30/09/10	IP	IPHONE	5	APPLE		USA
01/01/89	DE	DESLIZANTE	5	APPLE	BLANCO	USA

E

'ABCDE%', que indica que se buscan todas las ocurrencias que empiezan por 'ABCDE'.

Se puede utilizar también delante de una cadena de caracteres: '%ABCDE' lo que indica que se quieren recuperar las ocurrencias que acaben por ABCDE.

También en el medio de una cadena de caracteres: 'AB%CDE' que indica que se buscan las ocurrencias que comiencen por 'AB' y terminen por 'CDE'.

En cuanto al rendimiento, tenga en cuenta que la utilización del % al principio de la cadena impide al sistema utilizar un índice en dicha columna. Por el contrario, en el medio o al final de la cadena el carácter % no impide la posible utilización de un índice.

Las funciones de gestión de fechas y horas

Existen multitud de formatos de visualización, de utilización y de funciones de cada base de datos, no podemos describirlas todas aquí. En la sección Los diferentes formatos de visualización de fechas veremos las que se utilizan más comúnmente y el método para manipularlas.

1. Fecha del día: CURRENT_DATE

En Oracle existe CURRENT_DATE y SYSDATE. En MySQL, se puede utilizar CURRENT_DATE, CURDATE y NOW.

Estas funciones permiten devolver la fecha del día. En función del SGBDR, existen diferentes funciones que realizan lo mismo y son equivalentes.

Una pequeña particularidad en Oracle: SYSDATE devuelve la fecha del servidor en el que está instalada la base de datos y CURRENT_DATE retorna la fecha de la sesión del usuario.

Por ejemplo, si la base de datos está instalada en un servidor americano y el usuario está en España, habrá una diferencia de mínimo 6 horas entre las dos fechas.

Oracle:

```
SELECT CURRENT_DATE, SYSDATE FROM DUAL;
```

MySQL:

```
SELECT CURRENT_DATE,CURRENT_DATE(), CURDATE(), NOW() FROM
DUAL;
```

En MySQL el « () » significa 'Función'; si no hay paréntesis es una variable.

Existen algunas diferencias de visualización entre funciones y variables. NOW, por ejemplo, muestra la fecha y la hora actual y el resto sólo la fecha para MySQL. La visualización de estas funciones de tipo de fecha también depende de las opciones y parámetros utilizados por la BBDD (En Oracle NLS_DATE_FORMAT).

Ya hemos visto que es preferible proporcionar el formato de fecha que se desea obtener en la consulta con la función TO_CHAR en Oracle o la función DATE_FORMAT en MySQL.

Ejemplo en Oracle:

```
SELECT TO_CHAR(CURRENT_DATE,'DD/MM/YYYY') FROM DUAL;
```

Ejemplo en MySQL:

```
SELECT DATE_FORMAT(NOW(), '%d/%m/%Y')
```

2. Hora actual

En Oracle, no existe ninguna función que devuelva la hora del día. Hay que utilizar SYSDATE aplicando una máscara para mostrar sólo la hora.

Por el contrario MySQL utiliza las funciones CURRENT_TIME o CURTIME() que permiten recuperar la hora en el momento de la consulta.

Ejemplo en Oracle:

```
SELECT TO_CHAR(SYSDATE,'HH24:MI:SS') FROM DUAL;
```

Ejemplo en MySQL:

```
SELECT CURRENT_TIME
```

Existen diferentes formatos de visualización que son específicos de cada base de datos.

En Oracle los principales son:

- HH, HH12, HH24: formato de visualización de la hora en 12 o 24
- MI: minutos
- SS: segundos

En MySQL los principales son:

- %h, %H: formato de visualización de la hora en 12 o 24
- %i: minutos
- %s: segundos

Otro ejemplo: para transformar un número de segundos en formato HH:MM:SS hay que aplicar una función de este tipo:

```
SELECT HORA_EN_SEGUNDOS,
      TRIM(TO_CHAR(TRUNC(TRUNC(HORA_EN_SEGUNDOS/60)/60),'999900')
|| ':' || MOD(TRUNC(HORA_EN_SEGUNDOS/60),60)
|| ':' ||  MOD(HORA_EN_SEGUNDOS,60)) AS HMS
FROM « TABLA »;
```

3. Fecha y hora del día: CURRENT_TIMESTAMP

El timestamp es, de hecho, una concatenación de la fecha y la hora. También se pueden aplicar formatos de visualización diferentes. En MySQL es equivalente a NOW().

Ejemplo en Oracle con la visualización estándar:

```
SELECT CURRENT_TIMESTAMP FROM DUAL;

CURRENT_TIMESTAMP
---------------------------------
20/05/11 12:21:55,562000 +02:00
```

Ejemplo en Oracle aplicando un formato:

```
SELECT TO_CHAR(current_timestamp, 'DD/MM/YYYY son las HH24:MI:SS') AS
Fecha_y_Hora FROM DUAL;

FECHA_Y_HORA
---------------------------------
17/01/2014 son las 10:44:18
```

Ejemplo en MySQL con la visualización estándar:

```
SELECT CURRENT_TIMESTAMP

+---------------------+
| current_timestamp   |
+---------------------+
| 2011-05-20 12:24:31 |
+---------------------+
```

Ejemplo en MySQL aplicando un formato:

```
SELECT DATE_FORMAT(current_timestamp, «%d/%m/%Y à %H:%i:%S’) AS
Fecha_y_Hora;

+----------------------------+
| Fecha_y_Hora               |
+----------------------------+
| 17/01/2014 son las 10:42:19 |
+----------------------------+
```

4. Los diferentes formatos de visualización de fechas

Según la base de datos utilizada, los formatos de visualización y de manipulación de los datos son muy diferentes. Vamos a describir aquí algunos formatos y métodos simples para poder realizar las operaciones más comunes con fechas y horas. No podemos describir todas las opciones; no obstante, si quiere ir más allá consulte la documentación asociada para poder descubrir todas las posibilidades para gestionar las fechas y las horas.

Vamos a describir en los siguientes ejemplos cómo extraer el día, el mes o el año.

Ya hemos visto la sintaxis para mostrar una fecha en formato estándar.

Ejemplo en Oracle:

```
SELECT TO_CHAR(FECHA_COMPRA,’DD/MM/YY’) COMPRA
FROM TELEFONO;
```

COMPRA
15/01/10
14/03/10
02/05/10
25/07/10
30/09/10

Ejemplo en MySQL:

```
SELECT DATE_FORMAT(FECHA_COMPRA,’%d/%c/%Y’) COMPRA
FROM TELEFONO
```

Si ahora se quiere recuperar en la misma consulta, además de la fecha, el mes, habrá que modificar el formato de la siguiente manera:

Ejemplo en Oracle:

```
SELECT TO_CHAR(FECHA_COMPRA,’DD/MM/YY’) COMPRA,
TO_CHAR(FECHA_COMPRA,’MM’) MES
FROM TELEFONO;
```

COMPRA	MES
15/01/10	01
14/03/10	03
02/05/10	05
25/07/10	07
30/09/10	09

Ejemplo en MySQL:

```
SELECT DATE_FORMAT(FECHA_COMPRA,'%d/%c/%Y') COMPRA,
DATE_FORMAT(FECHA_COMPRA,'%c') MES
FROM TELEFONO;
```

Si se quiere obtener sólo el año en cuatro cifras, escribiremos:

Ejemplos en Oracle con la fecha del sistema:

```
SELECT TO_CHAR(SYSDATE,'YYYY') AÑO
FROM DUAL ;
```

AÑO
2014

Ejemplos en Oracle con la fecha de la tabla TELEFONO:

```
SELECT TO_CHAR(FECHA_COMPRA,'YYYY') AÑO
FROM TELEFONO;
```

AÑO
2010
2010
2010
2010
2010

Ejemplo en MySQL con la función DATE_FORMAT:

```
SELECT DATE_FORMAT(FECHA_COMPRA,'%Y') AÑO
FROM TELEFONO
```

Si se quiere mostrar la fecha en el formato "Número_del_día "de" Nombre_del_mes "de" Año_en_cuatro_cifras " (por ejemplo, 14 de ENERO de 2014), se deberá introducir:

Ejemplos en Oracle:

```
SELECT TO_CHAR(SYSDATE,'DD MONTH YYYY') FECHA_LARGA
FROM DUAL;
```

FECHA LARGA
14 de ENERO de 2014

```
SELECT TO_CHAR(FECHA_COMPRA,'DD MONTH YYYY') FECHA_LARGA
FROM TELEFONO;
```

FECHA_LARGA
15 de ENERO de 2010
14 de MARZO de 2010
02 de MAYO de 2010

25 de JULIO de2010
30 de SEPTIEMBRE de 2010

Ejemplos en MySQL:

```
SELECT DATE_FORMAT(FECHA_COMPRA,'%d %b %Y') FECHA_LARGA
FROM TELEFONO;
```

A continuación puede ver una lista de formatos de visualización de fechas utilizados por Oracle:

FORMATO	SIGNIFICADO	EJEMPLO
D	Número de día de la semana (1 a 7)	2
DAY	Nombre del día en letras	VIERNES
DD	Número de día	15
DDD	Número de día del año (1 a 366)	10
DY	Abreviatura del nombre de día en letras	VIE.
HH o HH12	Horas en cifras (1 a 12)	11
HH24	Horas en cifras (1 a 24)	18
J	Número de días desde el 1 de enero de 4712 antes de Cristo. (calendario juliano)	2456668
MI	Minutos en cifras	12
MM	Número del mes	01
MON	Abreviación del nombre del mes	ENE
MONTH	Nombre del mes en letras	ENERO
Q	Número del trimestre	1
RM	Número de mes en números romanos	I
SS	Segundos en cifras	11
SSSSS	Milisegundos en cifras	98451
W	Número del fin de semana en el mes (1 a 5)	4
WW	Número del fin de semana en el año (1 a 53)	45
Y	Última cifra del año	4
YEAR	Año en letras	DOS MIL CATORCE
YY	Dos últimas cifras del año	14
YYY	Tres últimas cifras del año	014
YYYY	Cuatro últimas cifras del año	2014

A continuación puede ver una lista de los formatos de visualización de fechas en MySQL:

FORMATO	SIGNIFICADO	EJEMPLO
%a	Nombre del día en letras en inglés abreviado	Fri
%b	Nombre del mes en letras en inglés abreviado	Jan
%c	Número del mes (1 a 12)	10

%d	Número del día	15
%D	Día del mes, con un sufijo inglés (1st, 2nd, 3rd, etc.)	17th
%e	Número del día	11
%f	Milisegundos en cifras	98451
%H	Horas en cifras (00 a 23)	23
%h	Horas en cifras (01 a 12)	12
%I	Horas en cifras (01 a 12)	01
%i	Minutos en cifras	38
%j	Número del día del año (1 a 366)	356
%k	Horas en cifras (1 a 23)	22
%l	Horas en cifras (1 a 12)	04
%m	Número del mes (01 a 12)	11
%M	Nombre del mes en letras en inglés	January
%p	AM o PM	PM
%r	Hora, en formato 12 horas (hh:mm:ss [AP/PM])	10:23:26 AM
%s	Segundos en cifras	4
%S	Segundos en cifras	5
%T	Hora, en formato 24 horas (hh:mm:ss)	10:23:26
%U	Número de la semana (00..53) (siendo el domingo el primer día de la semana)	02
%u	Número de la semana (00..53) (siendo el lunes el primer día de la semana)	03
%V	Número de la semana (01..53) (siendo el domingo el primer día de la semana)	02
%v	Número de la semana (01..53) (siendo el lunes el primer día de la semana)	03
%W	Nombre del día en letras (en inglés)	Friday
%w	Número del día de la semana (0=domingo)	5
%X	Año en cuatro cifras (siendo el domingo el primer día de la semana)	2014
%x	Año en cuatro cifras (siendo el lunes el primer día de la semana)	2014
%y	Dos últimas cifras del año	14
%Y	Año en cuatro cifras	2014

5. La manipulación de las fechas y las horas

A menudo es necesario manipular las fechas y/o las horas. En la mayoría de desarrollos hay que calcular la diferencia entre dos fechas o calcular una fecha final sumando días a una fecha.

La manipulación de las fechas debe respetar los formatos utilizados para las bases de datos. Ya hemos visto cómo mostrar una fecha indicando un formato específico de visualización.

Si se quiere, por ejemplo, calcular el número de días transcurridos desde la compra de un teléfono, escribiremos:

Ejemplo en Oracle:

```
SELECT SYSDATE,FECHA_COMPRA,SYSDATE - FECHA_COMPRA AS NUM_DIAS FROM
TELEFONO;
```

SYSDATE	FECHA_COMPRA	NUM_DIAS
17/01/14	15/01/10	1463,48374
17/01/14	14/03/10	1405,48374
17/01/14	02/05/10	1356,48374
17/01/14	25/07/10	1272,48374
17/01/14	30/09/10	1205,48374

Para mostrar el número de meses entre dos fechas, en Oracle se utilizará la función**MONTHS_BETWEEN**.

Ejemplo en Oracle:

```
SELECT SYSDATE,FECHA_COMPRA,TO_CHAR(MONTHS_BETWEEN(SYSDATE,
FECHA_COMPRA),'99.99') AS NUM_MESES FROM TELEFONO;
```

SYSDATE	FECHA_COMPRA	NUM_MESES
17/01/14	15/01/10	48.08
17/01/14	14/03/10	46.11
17/01/14	02/05/10	44.50
17/01/14	25/07/10	41.76
17/01/14	30/09/10	39.60

Para mostrar el número de años, de meses y días entre dos fechas, en MySQL se utilizará la función**TIMESTAMPDIFF** o **DATEDIFF** (no tiene en cuenta la hora).

Ejemplo en MySQL utilizando la función TIMESTAMPDIFF:

```
SELECT FECHA_COMPRA, CURRENT_DATE,
TIMESTAMPDIFF(YEAR  , FECHA_COMPRA, CURRENT_DATE ) NB_ANYOS,
TIMESTAMPDIFF(MONTH , FECHA_COMPRA, CURRENT_DATE ) NUM_MESES,
TIMESTAMPDIFF(DAY   , FECHA_COMPRA, CURRENT_DATE ) NUM_DIAS
FROM TELEFONO;
```

FECHA_COMPRA	CURRENT_DATE	NUM_ANYOS	NUM_MESES	NUM_DIAS
2010-01-15	2014-01-17	4	48	1463
2010-03-14	2014-01-17	3	46	1405
2010-05-02	2014-01-17	3	44	1356
2010-07-25	2014-01-17	3	41	1272
2010-09-30	2014-01-17	3	39	1205

Existen otras funciones que se pueden utilizar para manipular fechas. A continuación veremos algunas.

En Oracle:

Recuperación del último día del mes: ***LAST_DAY***

Recuperación de la próxima fecha del día pedido: ***NEXT_DAY***

```
SELECT LAST_DAY(SYSDATE) FROM DUAL ;
LAST_DAY
--------
31/01/14
```

```
SELECT SYSDATE,NEXT_DAY(SYSDATE,'LUNES') FROM DUAL ;
SYSDATE  NEXT_DAY
-------- --------
17/01/14 20/01/14
```

Añadir un número de meses a una fecha: ***ADD_MONTHS***

```
SELECT SYSDATE,ADD_MONTHS(SYSDATE,4) FROM DUAL ;
SYSDATE  ADD_MONTHS
-------- --------
17/01/14 17/05/14
```

Redondear una fecha al día siguiente si pasan las 12H00: ***ROUND***

```
SELECT TO_CHAR(SYSDATE,'DD/MM/YYYY HH24:MI'),ROUND(SYSDATE) FROM DUAL;
TO_CHAR(SYSDATE,' ROUND(SY
----------------- --------
17/01/2014 15:43  18/01/14

TO_CHAR(SYSDATE,' ROUND(SY
----------------- --------
17/01/2014 09:43  17/01/14
```

Redondear una fecha a la mañana del día ***TRUNC****:*

```
SELECT TO_CHAR(SYSDATE,'DD/MM/YYYY HH24:MI'),TRUNC(SYSDATE) FROM DUAL;
TO_CHAR(SYSDATE,' TRUNC(SY
----------------- --------
17/01/2014 15:43  17/01/14
```

En MySQL:

Realizar cálculos sobre una fecha: sumar ***DATE_ADD****, restar:* ***DATE_SUB***

```
SELECT CURRENT_DATE FECHADIA,DATE_ADD(CURRENT_DATE,INTERVAL 3 DAY)
SUMAR3DIAS, DATE_SUB(CURRENT_DATE,INTERVAL 3 DAY) RESTAR3DIAS;
+------------+-------------+-----------------+
| FECHADIA   | SUMAR3DIAS  | RESTAR3DIAS     |
+------------+-------------+-----------------+
| 2014-01-17 | 2014-01-20  | 2014-01-14      |
+------------+-------------+-----------------+
```

El intervalo se debe expresar en números. A continuación veremos el formato utilizado. En el ejemplo se pide un decalaje en días. Existen multitud de posibilidades para expresar el intervalo.

Valores posibles para el tipo de intervalo
DAY
DAY_HOUR
DAY_MICROSECOND
DAY_MINUTE
DAY_SECOND

HOUR
HOUR_MICROSECOND
HOUR_MINUTE
HOUR_SECOND
MICROSECOND
MINUTE
MINUTE_MICROSECOND
MINUTE_SECOND
MONTH
QUARTER
SECOND
SECOND_MICROSECOND
WEEK
YEAR
YEAR_MONTH

Obtener el nombre del día: ***DAYNAME***

```
SELECT CURRENT_DATE FECHADIA,DAYNAME(CURRENT_DATE) NOMBREDIA;
+------------+-----------+
| FECHADIA   | NOMBREDIA |
+------------+-----------+
| 2014-01-17 | Friday    |
+------------+-----------+
```

Obtener el número del día de la semana (0=domingo): ***DAYOFWEEK***

```
SELECT CURRENT_DATE FECHADIA,DAYOFWEEK(CURRENT_DATE) NUMDIA;
+------------+-----------+
| FECHADIA   | NUMDIA    |
+------------+-----------+
| 2014-01-17 | 6         |
+------------+-----------+
```

Obtener el número del día del año: ***DAYOFYEAR***

```
SELECT CURRENT_DATE FECHADIA,DAYOFYEAR(CURRENT_DATE) NUMDIA;
+------------+-----------+
| FECHADIA   | NUMDIA    |
+------------+-----------+
| 2014-01-17 | 17        |
+------------+-----------+
```

Recuperación del último día del mes: ***LAST_DAY***

```
SELECT LAST_DAY(CURRENT_DATE) ULTDIA;
ULTDIA
----------
2014-01-31
```

Las funciones con cadenas de caracteres

1. LOWER / UPPER / UCASE / LCASE (minúsculas y mayúsculas)

Existen dos funciones para convertir las cadenas de caracteres en minúsculas o en mayúsculas. Los nombres difieren entre cada SGBD.

En Oracle, se utilizará LOWER y UPPER, en MySQL se permiten LOWER, UPPER, UCASE y LCASE.

LCASE y LOWER se utilizan para convertir a minúsculas y UCASE y UPPER para convertir a mayúsculas.

Ejemplo en Oracle:

```
SELECT LOWER('Esto es una PRUEBA') AS MINUSCULA FROM DUAL;

MINUSCULA
---------------
esto es una prueba

SELECT UPPER('Esto es una PRUEBA) AS MAYUSCULA FROM DUAL;

MAYUSCULA
---------------
ESTO ES UNA PRUEBA
```

Ejemplo en MySQL:

```
SELECT LCASE('Esto es una PRUEBA) AS MINUSCULA

+--------------------+
| MINUSCULA          |
+--------------------+
| esto es una prueba |
+--------------------+

SELECT UCASE('Esto es una PRUEBA) AS MAYUSCULA

+--------------------+
| MAYUSCULA          |
+--------------------+
| ESTO ES UNA PRUEBA |
+--------------------+
```

La sintaxis es:

```
SELECT LOWER o UPPER (<columna o variable>) ... FROM <tabla1>,
<tabla2> ...
```

con la posibilidad de utilizar LCASE y UCASE en MySQL.

2. Eliminar los espacios a la derecha o izquierda de una cadena de caracteres: TRIM / LTRIM / RTRIM

Para eliminar espacios en una cadena de caracteres, hay que utilizar LTRIM si están a la izquierda (Left) o RTRIM si están a la derecha (Right). Para eliminar espacios a izquierda y derecha, hay que utilizar TRIM.

Ejemplo en Oracle:

```
SELECT '*'||LTRIM('   ELIMINACIÓN DE LOS ESPACIOS A LA IZQUIERDA   ')||'*'
```

```
AS SUPIZQ FROM DUAL;

SUPIZQ
----------------------------------------------
*ELIMINACIÓN DE LOS ESPACIOS A LA IZQUIERDA   *

SELECT '*'||RTRIM('   ELIMINACIÓN DE LOS ESPACIOS A LA DERECHA   ')||'*'
AS SUPDER FROM DUAL;

SUPDER
----------------------------------------------
*   ELIMINACIÓN DE LOS ESPACIOS A LA DERECHA*

SELECT '*'||TRIM('   ELIMINACIÓN DE LOS ESPACIOS A IZQUIERDA Y DERECHA
')||'*' AS SUPIZQDER FROM DUAL;

SUPIZQDER
-----------------------------------------------------
*ELIMINACIÓN DE LOS ESPACIOS A IZQUIERDA Y DERECHA*
```

Ejemplo en MySQL:

```
SELECT CONCAT('*',LTRIM('   ELIMINACIÓN DE LOS ESPACIOS A LA IZQUIERDA
'),'*') AS SUPIZQ FROM DUAL;

+-------------------------------------------------+
| SUPIZQ                                          |
+-------------------------------------------------+
| * ELIMINACIÓN DE LOS ESPACIOS A LA IZQUIERDA   * |
+-------------------------------------------------+


SELECT CONCAT('*',RTRIM('   ELIMINACIÓN DE LOS ESPACIOS A LA DERECHA
'),'*') AS SUPDER FROM DUAL;


+-----------------------------------------------+
| SUPDER                                        |
+-----------------------------------------------+
| *   ELIMINACIÓN DE LOS ESPACIOS A LA DERECHA * |
+-----------------------------------------------+


SELECT CONCAT('*',TRIM('   ELIMINACIÓN DE LOS ESPACIOS A IZQUIERDA Y
DERECHA   '),'*') AS SUPIZQDER FROM DUAL;

+-------------------------------------------------------+
| SUPIZQDER                                             |
+-------------------------------------------------------+
| * ELIMINACIÓN DE LOS ESPACIOS A IZQUIERDA Y DERECHA * |
+-------------------------------------------------------+
```

La sintaxis es:

```
SELECT RTRIM o LTRIM o TRIM (<columna o variable>) ...  FROM
<tabla1>, <tabla2> ...
```

3. Transformar un dato numérico o una fecha en caracteres: TO_CHAR

Las funciones TO_CHAR en Oracle y CAST en MySQL permiten convertir un dato numérico o una fecha en caracteres.

Ejemplo en Oracle:

```
SELECT TO_CHAR(PRECIO) PRECIO, TO_CHAR(FECHA_COMPRA,'DD/MM/YY') COMPRA
FROM TELEFONO;
```

PRECIO	COMPRA
159	15/01/10
99	14/03/10
49	02/05/10
89	25/07/10
359	30/09/10

Ejemplo en MySQL:

```
SELECT CAST(PRECIO AS CHAR) PRECIO, CAST(FECHA_COMPRA AS DATE) COMPRA
FROM TELEFONO
```

La sintaxis en Oracle:

```
SELECT TO_CHAR(<columna o variable>,[<FORMAT>],<cadena
buscada> ...  FROM <tabla1>, <tabla2> ...
```

La sintaxis en MySQL:

```
SELECT CAST(<columna o variable> as <TIPO>) ...  FROM <tabla1>,
<tabla2> ...
```

4. Encontrar la posición de una cadena de caracteres en otra cadena: INSTR

Puede ser interesante encontrar en una cadena la posición de uno o varios caracteres en concreto. Para ello, hay que utilizar la función INSTR (cadena,cadena buscada).

Ejemplo en Oracle:

```
SELECT INSTR('ESTO ES UN EJEMPLO PARA ENCONTRAR LA POSICIÓN DE
UNA CADENA','EJEMPLO') AS POSICION FROM DUAL;

  POSICION
----------
       13
```

La cadena « EJEMPLO » empieza en la posición 13 en la cadena.

Esta instrucción puede ser muy útil para extraer una parte de la cadena sin saber su posición inicial.

Por ejemplo, si se quiere recuperar las dos siguientes palabras a la palabra « agente » en la sinopsis de una película « Nikita, condenada a cadena perpetua es obligada a trabajar en secreto para el gobierno como agente de los servicios secretos. »

Ejemplo en Oracle y MySQL:

En este caso se recuperan los 32 caracteres siguientes a la palabra buscada.

```
SELECT SUBSTR(RESUME,INSTR(SINOPSIS,'agente'),32) AS EXTRACCION FROM
PELICULAS WHERE TITULO = 'NIKITA';

EXTRACCION
--------------------------------
agente de los servicios secretos
```

La sintaxis es:

```
SELECT INSTR(<columna o variable>,<cadena buscada>) ...  FROM
<tabla1>, <tabla2> ...
```

5. Agregar caracteres antes o después de una cadena: LPAD / RPAD

Si se quiere completar una cadena con espacios o por cualquier otro carácter, hay que utilizar las funciones LPAD para agregar por la izquierda y RPAD para agregar por la derecha.

Hay que indicar la cadena, el carácter que se quiere añadir y el número de veces que se quiere repetir el carácter.

Ejemplo para Oracle y MySQL:

En este caso se completa la cadena con puntos hasta que la cadena completa tenga 50 caracteres.

```
SELECT LPAD('AÑADO PUNTOS A LA IZQUIERDA',50,'.') AS AIZQUIERDA
FROM DUAL;

AIZQUIERDA
--------------------------------------------------
......................AÑADO PUNTOS A LA IZQUIERDA



SELECT RPAD('AÑADO PUNTOS A LA DERECHA',50,'.') AS ADERECHA
FROM DUAL;

ADERECHA
--------------------------------------------------
AÑADO PUNTOS A LA DERECHA.......................
```

La sintaxis es:

```
SELECT RPAD(<columna o variable>,<tamaño máximo al final>,
<carácter(es) de relleno>) ...  FROM <tabla1>,
<tabla2> ...
```

```
SELECT LPAD(<columna o variable>,<tamaño máximo al final>,
<carácter(es) de relleno>) ...  FROM <tabla1>,
<tabla2> ...
```

6. Extraer parte de una cadena de caracteres: SUBSTR

Es posible extraer 1 o n caracteres de una cadena indicando la posición de inicio y la longitud deseada. En el siguiente ejemplo, se recuperan los cuatro primeros caracteres de la descripción de la marca.

Ejemplo en Oracle y MySQL:

```
SELECT SUBSTR(DESC_MARCA,1,4) FROM MARCA_TEL;

SUBS
----
SAMS
SONY
PHIL
MOTO
APPL
```

La sintaxis es:

```
SELECT SUBSTR(<columna o variable>,<posición inicial>,<Longitud>) ...
FROM <tabla1>, <tabla2> ...
```

Otras funciones

1. NVL: comprobar si una columna es null

NVL por « Null Value » permite saber si una columna tiene datos o no y asignarle un valor en caso de que esté a nulos.

Sintaxis:

```
SELECT NVL(<nombre columna>,<valor asignado>), ...
```

El valor asignado debe ser del mismo tipo que la columna.

Ejemplo:

```
SELECT NUMERO,TIPO,MARCA,FECHA_COMPRA,PRECIO,NUM_PROPIETARIO,
NVL(COLOR,'SIN') FROM TELEFONO;
```

Resultado:

NUMERO	TIPO	MARCA	FECHA_COMPRA	PRECIO	NUM_PROPIETARIO	COLOR
1	SP	1	15/01/10	159	190120	ROJO
2	SP	1	14/03/10	99	190215	**SIN**
3	CT	3	02/05/10	49	190001	NEGRO
4	DE	4	25/07/10	89	190222	BLANCO
5	IP	5	30/09/10	359	190561	**SIN**

Se puede comprobar que los teléfonos que tienen « COLOR » a NULL son sustituidos por el valor « SIN ».

2. Comprobar varios valores: COALESCE

Esta función permite comprobar varios valores NULL de columnas en una misma función evitando así funciones « IF » « THEN », etc.

Comprueba cada columna y asigna el resultado de izquierda a derecha. Se asigna la primera columna no nula. Si todas las columnas son NULL, la función tomará el valor por defecto que debe ser el último parámetro.

Sintaxis:

```
COALESCE(<columna1>, <columna2>,... <valor por defecto>);
```

Ejemplo:

Si se tienen los siguientes datos en la tabla TELEFONO:

NUMERO	TIPO	MARCA	FECHA_COMPRA	PRECIO	NUM_PROPIETARIO	COLOR
1	SP	1	15/01/10	159	190120	ROJO
2	***<NULL>***	1	14/03/10	99	190215	***<NULL>***
3	***<NULL>***	3	02/05/10	49	190001	NEGRO

4	DE	4	25/07/10	89	190222	BLANCO
5	IP	5	30/09/10	359	190561	***<NULL>***

Se quiere clasificar los teléfonos en función del contenido de estas dos columnas. Si TIPO es NULL se coge el COLOR, si COLOR es NULL se coge el valor 'NINGUNO DE LOS DOS'.

```
SELECT
NUMERO,TIPO,MARCA,FECHA_COMPRA,PRECIO,COLOR,COALESCE(TIPO,COLOR,
'NINGUNO DE LOS DOS') RESULTADO FROM TELEFONO;
```

Resultados:

NUMERO	TIPO	MARCA	FECHA_ COMPRA	PRECIO	COLOR	RESULTADO
1	SP	1	15/01/10	159	ROJO	**SP**
2	<*NULL*>	1	14/03/10	99	<*NULL*>	***NINGUNO DE LOS DOS***
3	<*NULL*>	3	02/05/10	49	NEGRO	**NEGRO**
4	DE	4	25/07/10	89	BLANCO	**DE**
5	IP	5	30/09/10	359	<*NULL*>	**IP**

3. Comparar dos columnas: NULLIF

Esta función compara dos columnas. Si las dos son idénticas devolverá el valor NULL; si no devolverá la primera columna.

Sintaxis:

```
NULLIF(<columna1>, <columna2>)
```

Ejemplo:

```
SELECT NUMERO,TIPO,MARCA,FECHA_COMPRA,PRECIO,COLOR,
NULLIF(TIPO,COLOR) RESULTADO FROM TELEFONO;
```

Resultados:

NUMERO	TIPO	MARCA	FECHA_COMPRA	PRECIO	COLOR	RESULTADO
1	SP	1	15/01/10	159	ROJO	**SP**
2	<*NULL*>	1	14/03/10	99	<*NULL*>	***<NULL>***
3	<*NULL*>	3	02/05/10	49	NEGRO	***<NULL>***
4	DE	4	25/07/10	89	BLANCO	**DE**
5	IP	5	30/09/10	359	<*NULL*>	**IP**

4. Cambiar el tipo de una columna: CAST

Esta función permite cambiar el tipo de una columna durante la ejecución del comando SQL. Por ejemplo, cambiar una columna que inicialmente es VARCHAR en INTEGER para hacer un cálculo o comprobar el valor numérico. Atención, el nuevo tipo debe ser compatible con el contenido real de la columna.

Sintaxis:

```
CAST(<columna> AS <tipo>)
```

La columna MARCA es VARCHAR2, se pasa a INTEGER para comprobar:

```
SELECT NUMERO,TIPO,MARCA,FECHA_COMPRA,PRECIO,NUM_PROPIETARIO,
COLOR
FROM TELEFONO
WHERE CAST(MARCA AS INTEGER) < 4;
```

Resultados:

NUMERO	TIPO	MARCA	FECHA_COMPRA	PRECIO	NUM_PROPIETARIO	COLOR
1	SP	1	15/01/10	159	190120	ROJO
2	SP	1	14/03/10	99	190215	
3	CL	3	02/05/10	49	190001	NEGRO

5. Comprobar el contenido de un dato con DECODE

DECODE es una función relativamente potente que permite comprobar el contenido de los datos directamente en el SELECT o en el WHERE.

Funciona de forma booleana, es decir que se comprueba un valor y si el resultado es cierto realiza una acción y si es falso realiza otra acción.

De forma parecida a CASE, DECODE permite no utilizar lenguajes como PL/SQL y hacer algo de desarrollo directamente en la consulta SQL.

En un DECODE se pueden utilizar otras funciones SQL para dar formato a un dato antes de utilizarlo.

DECODE no existe en MySQL. Hay que utilizar el « IF » que prácticamente sustituye a esta función.

La sintaxis simple es la siguiente:

```
SELECT <columna 1>, <columna 2>,
      DECODE (<columna 3>, <1er valor comprobado>, <valor si cierto>,
                           <2o  valor comprobado>, <valor si cierto>,
                           <3er valor comprobado>, <valor si cierto>,
                             ... <valor si falso>),
      <columna 4>
      ... ...
FROM <tabla1>, <tabla2> ... ...
WHERE ... ...
```

Ejemplo con control del precio (sintaxis Oracle):

```
SELECT TEL.FECHA_COMPRA AS COMPRA,
       TEL.TIPO AS TIPO,
       TIPO.DESC_TIPO AS DESCRIPCION,
       TEL.MARCA AS MARCA,
       MAR.DESC_MARCA AS DESCRIPCION,
       TEL.PRECIO AS PRECIO,
       DECODE(TEL.PRECIO,49,'PRECIO MINIMO','OTRO PRECIO') AS TIPO_PRECIO
FROM TELEFONO TEL,
     TIPO_TEL TIPO ,
     MARCA_TEL MAR
WHERE TEL.TIPO  = TIPO.TIPO AND
      TEL.MARCA = MAR.MARCA;
```

Ejemplo con control del precio (sintaxis MySQL):

```
SELECT TEL.FECHA_COMPRA AS COMPRA,
       TEL.TIPO AS TIPO,
       TIPO.DESC_TIPO AS DESCRIPCION,
       TEL.MARCA AS MARCA,
```

```
       MAR.DESC_MARCA AS DESCRIPCION,
       TEL.PRECIO AS PRECIO,
       IF(TEL.PRECIO=49,'PRECIO MINIMO','OTRO PRECIO') AS TIPO_PRECIO
FROM TELEFONO TEL,
     TIPO_TEL TIPO ,
     MARCA_TEL MAR
WHERE TEL.TIPO   = TIPO.TIPO AND
      TEL.MARCA = MAR.MARCA;
```

Resultado:

COMPRA	TIPO	DESCRIPCION	MARCA	DESCRIPCION	PRECIO	TIPO_PRECIO
15/01/10	SP	SMARTPHONE	1	SAMSUNG	159	**OTRO PRECIO**
14/03/10	SP	SMARTPHONE	2	SONY	99	**OTRO PRECIO**
02/05/10	CT	CON TAPA	3	PHILIPS	49	**PRECIO MINIMO**
25/07/10	DE	DESLIZANTE	4	MOTOROLA	89	**OTRO PRECIO**
30/09/10	IP	IPHONE	5	APPLE	359	**OTRO PRECIO**

Solo se ha comprobado un valor; si ahora en lugar de poner 'OTRO PRECIO' en el resto de filas, se quiere afinar la comprobación, hay que añadir otros valores a continuación. El último valor del DECODE es el valor que se coge si no se cumple ninguna de las condiciones anteriores.

Ejemplo con varios controles (sintaxis Oracle):

```
SELECT TEL.FECHA_COMPRA AS COMPRA,
       TEL.TIPO AS TIPO,
       TIPO.DESC_TIPO AS DESCRIPCION,
       TEL.MARCA AS MARCA,
       MAR.DESC_MARCA AS DESCRIPCION,
       TEL.PRECIO AS PRECIO,
       DECODE(TEL.PRECIO,49,'PRECIO MINIMO',
                         359,'PRECIO MAXIMO',
                         159,'PRECIO MEDIO','OTRO PRECIO')
       AS TIPO_PRECIO
FROM TELEFONO TEL,
     TIPO_TEL TIPO ,
     MARCA_TEL MAR
WHERE TEL.TIPO   = TIPO.TIPO AND
      TEL.MARCA = MAR.MARCA;
```

Ejemplo con varios controles (sintaxis MySQL):

```
SELECT TEL.FECHA_COMPRA AS COMPRA,
       TEL.TIPO AS TIPO,
       TIPO.DESC_TIPO AS DESCRIPCION,
       TEL.MARCA AS MARCA,
       MAR.DESC_MARCA AS DESCRIPCION,
       TEL.PRECIO AS PRECIO,
       IF(TEL.PRECIO=49,'PRECIO MINIMO',
          IF(TEL.PRECIO=359,'PRECIO MAXIMO',
             IF(TEL.PRECIO=159,'PRECIO MEDIO,'OTRO PRECIO')
             )
          )  AS TIPO_PRECIO
FROM TELEFONO TEL,
     TIPO_TEL TIPO ,
     MARCA_TEL MAR
WHERE TEL.TIPO   = TIPO.TIPO AND
      TEL.MARCA = MAR.MARCA;
```

Resultado:

COMPRA	TIPO	DESCRIPCION	MARCA	DESCRIPCION	PRECIO	TIPO_ PRECIO
15/01/10	SP	SMARTPHONE	1	SAMSUNG	159	**PRECIO MEDIO**
14/03/10	SP	SMARTPHONE	2	SONY	99	**OTRO PRECIO**
02/05/10	CT	CON TAPA	3	PHILIPS	49	**PRECIO MINIMO**
25/07/10	DE	DESLIZANTE	4	MOTOROLA	89	**OTRO PRECIO**
30/09/10	IP	IPHONE	5	APPLE	359	**PRECIO MAXIMO**

El DECODE también se puede posicionar en la cláusula WHERE.

Si se quieren mostrar las filas cuyos teléfonos son de color « Oscuro ». Este concepto de « Oscuro » agruparía a los colores 'NEGRO' y 'ROJO'.

Ejemplo con un DECODE en la cláusula WHERE (sintaxis Oracle):

```
SELECT TEL.FECHA_COMPRA AS COMPRA,
       TEL.TIPO AS TIPO,
       TIPO.DESC_TIPO AS DESCRIPCION,
       TEL.MARCA AS MARCA,
       MAR.DESC_MARCA AS DESCRIPCION,
       TEL.PRECIO AS PRECIO,
       DECODE(TEL.PRECIO,49,'PRECIO MINIMO',359,'PRECIO
MAXIMO',159,'PRECIO MEDIO','OTRO PRECIO') AS TIPO_PRECIO,
       TEL.COLOR AS COLOR
FROM TELEFONO TEL,
     TIPO_TEL TIPO ,
     MARCA_TEL MAR
WHERE TEL.TIPO   = TIPO.TIPO AND
      TEL.MARCA = MAR.MARCA AND
      DECODE(TEL.COLOR,'NEGRO','Oscuro',
                       'BLANCO','Claro',
                       'ROJO','Oscuro','Otro')='Oscuro';
```

Ejemplo con un DECODE en la cláusula WHERE (sintaxis MySQL):

```
SELECT TEL.FECHA_COMPRA AS COMPRA,
       TEL.TIPO AS TIPO,
       TIPO.DESC_TIPO AS DESCRIPCION,
       TEL.MARCA AS MARCA,
       MAR.DESC_MARCA AS DESCRIPCION,
       TEL.PRECIO AS PRECIO,
       IF(TEL.PRECIO=49,'PRECIO MINIMO',
       IF(TEL.PRECIO=359,'PRECIO MAXIMO',
       IF(TEL.PRECIO=159,'PRECIO MEDIO','OTRO PRECIO')))
AS TIPO_PRECIO,
       TEL.COLOR AS COLOR
FROM TELEFONO TEL,
     TIPO_TEL TIPO ,
     MARCA_TEL MAR
WHERE TEL.TIPO   = TIPO.TIPO AND
      TEL.MARCA = MAR.MARCA AND
      IF(COLOR='NEGRO','Oscuro',
        IF(COLOR='BLANCO','Claro',
           IF(COLOR='ROJO','Oscuro','Otro')))='Oscuro';
```

Resultado:

						TIPO_	

COMPRA	TIPO	DESCRIPCION	MARCA	DESCRIPCION	PRECIO	PRECIO	COLOR
15/01/10	SP	SMARTPHONE	1	SAMSUNG	159	PRECIO MEDIO	**ROJO**
02/05/10	CT	CON TAPA	3	PHILIPS	49	PRECIO MINIMO	**NEGRO**

Un último ejemplo para mostrar que se pueden incluir funciones en un DECODE.

Si se quiere mostrar la MARCA cuya descripción empiece por una 'S' o 'M', se utilizará la función SUBSTR.

Ejemplo con una función en un DECODE (sintaxis Oracle):

```
SELECT TEL.FECHA_COMPRA AS COMPRA,
       TEL.TIPO AS TIPO,
       TIPO.DESC_TIPO AS DESCRIPCION,
       TEL.MARCA AS MARCA,
       MAR.DESC_MARCA AS DESCRIPCION,
       TEL.PRECIO AS PRECIO,
       DECODE(TEL.PRECIO,49,'PRECIO MINIMO',359,'PRECIO
MAXIMO',159,'PRECIO MEDIO','OTRO PRECIO') AS TIPO_PRECIO,
       TEL.COLOR AS COLOR
FROM TELEFONO TEL,
     TIPO_TEL TIPO ,
     MARCA_TEL MAR
WHERE TEL.TIPO   = TIPO.TIPO AND
      TEL.MARCA = MAR.MARCA AND
      DECODE(SUBSTR(MAR.DESC_MARCA,1,1),'S','OK','M','OK','KO')
= 'OK';
```

Ejemplo con una función en un DECODE (sintaxis MySQL):

```
SELECT TEL.FECHA_COMPRA AS COMPRA,
       TEL.TIPO AS TIPO,
       TIPO.DESC_TIPO AS DESCRIPCION,
       TEL.MARCA AS MARCA,
       MAR.DESC_MARCA AS DESCRIPCION,
       TEL.PRECIO AS PRECIO,
       IF(TEL.PRECIO=49,'PRECIO MINIMO',
       IF(TEL.PRECIO=359,'PRECIO MAXIMO',
                    IF(TEL.PRECIO=159,'PRECIO MEDIO','OTRO
PRECIO')))  AS TIPO_PRECIO,
       TEL.COLOR AS COLOR
FROM TELEFONO TEL,
     TIPO_TEL TIPO ,
     MARCA_TEL MAR
WHERE TEL.TIPO   = TIPO.TIPO AND
      TEL.MARCA = MAR.MARCA AND
      IF(SUBSTR(MAR.DESC_MARCA,1,1)='S','OK',
         IF(SUBSTR(MAR.DESC_MARCA,1,1)='M','OK','KO')) = 'OK';
```

Resultado:

COMPRA	TIPO	DESCRIPCION	MARCA	DESCRIPCION	PRECIO	TIPO_ PRECIO	COLOR
15/01/10	SP	SMARTPHONE	1	**S**AMSUNG	159	PRECIO MEDIO	ROJO
14/03/10	SP	SMARTPHONE	2	**S**ONY	99	OTRO PRECIO	
25/07/10	DE	DESLIZANTE	4	**M**OTOROLA	89	OTRO PRECIO	BLANCO

Como conclusión, el DECODE puede ser muy útil en muchas situaciones, pero hay que pensar que puede no

ser fácil de leer para una persona que no haya escrito la consulta (hay que documentarlo al máximo). En términos de rendimiento, el DECODE penaliza bastante, así que se puede ralentizar la consulta.

Ejercicios sobre las funciones

Estos ejercicios se basan en las tablas indicadas en la lección Ejercicios sobre la selección de datos del capítulo anterior.

Primer ejercicio

Recuperar la fecha del día en formato DD-MM-YYYY.

Segundo ejercicio

Convertir la siguiente cadena en mayúsculas y a continuación buscar la posición de la cadena 'GOL'.

'En el partido Málaga - Barcelona se marcó el primer gol en el minuto 69'

Tercer ejercicio

Eliminar los espacios a la izquierda y agregar '-' a la siguiente cadena hasta obtener una cadena de 50 caracteres.

' El tiempo no permite realizar el trabajo'

Cuarto ejercicio

Seleccionar las películas cuyo director se llame 'LUC' y que se hayan estrenado entre el '01/01/85' y el '30/05/1995'.

Quinto ejercicio

Mostrar la fecha y la hora del día en el formato:

Hoy es Viernes 24 de enero de 2014 y son las 16 horas 26 minutos

Sexto ejercicio

Calcular el número de días que han pasado desde el estreno de todas las películas de la tabla PELICULAS.

A continuación mostrar estas cifras expresadas en meses.

Solución de los ejercicios sobre las funciones

Primer ejercicio

Recuperar la fecha del día en formato DD-MM-YYYY.

```
SELECT TO_CHAR(CURRENT_DATE,'DD/MM/YYYY') FROM DUAL;
```

Segundo ejercicio

Convertir la siguiente cadena en mayúsculas y a continuación buscar la posición de la cadena 'GOL'.

'En el partido Málaga - Barcelona se marcó el primer gol en el minuto 69'

```
SELECT INSTR(UPPER('En el partido Málaga - Barcelona se marcó el primer
gol en el minuto 69'),'GOL') AS POSICION FROM DUAL;

  POSICION
----------
        52
```

Tercer ejercicio

Eliminar los espacios a la izquierda y agregar '-' a la siguiente cadena hasta obtener una cadena de 50 caracteres.

' El tiempo no permite realizar el trabajo'

```
SELECT RPAD(LTRIM('           El tiempo no permite realizar
el trabajo'),50,'-') AS MEF FROM DUAL;

MEF
---

El tiempo no permite realizar el trabajo---
```

Cuarto ejercicio

Seleccionar las películas cuyo director se llame 'LUC' y que se hayan estrenado entre el '01/01/85' y el '30/05/1995'.

```
SELECT * FROM PELICULA T1 WHERE
   EXISTS (SELECT IDENT_DIRECTOR FROM DIRECTOR WHERE NOMBRE
= 'LUC' AND IDENT_DIRECTOR = T1.IDENT_DIRECTOR)
   AND FECHA_ESTRENO BETWEEN ('01/01/85') AND ('30/05/1995')
ORDER BY TITULO;
```

o

```
SELECT * FROM PELICULA T1, DIRECTOR T2 WHERE
       T1.IDENT_DIRECTOR = T2.IDENT_DIRECTOR AND
       T2.NOMBRE = 'LUC' AND
       FECHA_ESTRENO BETWEEN ('01/01/85') AND ('30/05/1995')
ORDER BY TITULO;
```

o

```
SELECT * FROM PELICULA T1 WHERE
   T1.IDENT_DIRECTOR IN (SELECT IDENT_DIRECTOR FROM
```

```
DIRECTOR WHERE NOMBRE = 'LUC')
   AND FECHA_ESTRENO BETWEEN ('01/01/85') AND ('30/05/1995')
ORDER BY TITULO;
```

Este ejercicio muestra claramente que existen diferentes posibilidades al crear una consulta obteniendo el mismo resultado.

Si hay índice en la columna IDENT_DIRECTOR en las dos tablas, la segunda consulta será la que tiene el mejor rendimiento.

Resultado del SELECT:

IDENT_ PELICULA	TITULO	GENERO1	GENERO2	FECHAS_ ESTRENO	PAIS	IDENT_ DIRECTOR	DISTRI-BUIDOR
2	NIKITA	DRAMA	ROMANTICO	21/02/90	1	1	FILMAX
SINOPSIS	Nikita, condenada a cadena perpetua es obligada a trabajar en secreto para el gobierno como agente de los servicios secretos.						
1	SUBWAY	POLICIACA	DRAMA	10/04/85	1	1	FILMAX
SINOPSIS	Cuenta las aventuras de la población subterránea en los túneles del metro de París						

Quinto ejercicio

Mostrar la fecha y la hora del día en el formato:

Hoy es Viernes 24 de enero de 2014 y son las 16 horas 26 minutos

En Oracle la sintaxis es la siguiente:

```
SELECT 'HOY ES '||TO_CHAR(CURRENT_TIMESTAMP, 'DAY DD MONTH
YYYY'))||' Y SON LAS '||TO_CHAR(CURRENT_TIMESTAMP, 'HH24')||' HORAS Y
'||TO_CHAR(CURRENT_TIMESTAMP, 'MI')||' MINUTOS' AS EJERCICIO_5 FROM DUAL;

EJERCICIO_5
-----------------------------------------------------------------
HOY ES VIERNES 24 DE ENERO DE 2014 Y SON LAS 16 HORAS Y 26 MINUTOS
```

En lugar de CURRENT_TIMESTAMP se puede utilizar CURRENT_DATE; Oracle también guarda la hora en la variable CURRENT_DATE.

```
SELECT 'HOY ES '||TO_CHAR(CURRENT_DATE, 'DAY DD MONTH YYYY')||'
Y SON LAS '||TO_CHAR(CURRENT_DATE,'HH24')||' HORAS Y
'||TO_CHAR(CURRENT_DATE,'MI')||' MINUTOS' AS EJERCICIO_5 FROM DUAL;
```

Con MySQL la sintaxis es la siguiente:

```
SELECT DATE_FORMAT(CURRENT_TIMESTAMP, 'HOY ES %A %D %M %Y A %H
HORAS Y %I MINUTOS') AS FECHA_Y_HORA;
+-----------------------------------------------------------------+
|  FECHA_Y_HORA                                                   |
+-----------------------------------------------------------------+
|  HOY ES VIERNES 24 DE ENERO DE 2014 Y SON LAS 16 HORAS Y 26 MINUTOS  |
+-----------------------------------------------------------------+
```

Sexto ejercicio

Calcular el número de días que han pasado desde el estreno de todas las películas de la tabla PELICULAS.

En Oracle la sintaxis es la siguiente:

```
SELECT TITULO,FECHA_ESTRENO,CURRENT_DATE - FECHA_ESTRENO AS
NUM_DIAS_DESDE_ESTRENO FROM
PELICULA;

TITULO                              FECHA_ES  NUM_DIAS_DESDE_ESTRENO
----------------------------------- --------  ----------------------
SUBWAY                              10/04/85             10516,7427
NIKITA                              21/02/90             8738,74267
STAR WARS 6: EL RETORNO DEL JEDI    19/10/83             11055,7427
AVATAR                              16/10/09             1561,74267
BIENVENIDO AL NORTE                 27/02/08             2158,74267
```

Con MySQL la sintaxis es la siguiente:

```
SELECT TITULO,FECHA_ESTRENO,TIMESTAMPDIFF(DAY  , FECHA_ESTRENO, CURRENT_DATE )
NUM_DIAS_DESDE_ESTRENO FROM PELICULAS;
+---------------------------------+---------------+------------------------+
| TITULO                          | FECHA_ESTRENO | NUM_DIAS_DESDE_ESTRENO |
| SUBWAY                          | 1985-04-10    |                  10516 |
+---------------------------------+---------------+------------------------+
| NIKITA                          | 1990-02-21    |                   8738 |
| STAR WARS 6: EL RETORNO DEL JEDI| 1983-10-19    |                  11055 |
| AVATAR                          | 2009-10-16    |                   1561 |
| BIENVENIDOS AL NORTE            | 2008-02-27    |                   2158 |
+---------------------------------+---------------+------------------------+
```

Para mostrar las cifras en meses.

Con Oracle la sintaxis es la siguiente:

```
SELECT TITULO,FECHA_ESTRENO,TO_CHAR(MONTHS_BETWEEN
(CURRENT_DATE,FECHA_ESTRENO),'9999') AS
NUM_MESES_DESDE_ESTRENO FROM PELICULAS;

TITULO                                            FECHA_ES NUMMES
------------------------------------------------- -------- ------
SUBWAY                                            10/04/85    345
NIKITA                                            21/02/90    287
STAR WARS 6: EL RETORNO DEL JEDI                  19/10/83    363
AVATAR                                            16/10/09     51
BIENVENIDO AL NORTE                               27/02/08     71
```

En MySQL la sintaxis es la siguiente:

```
SELECT TITULO,FECHA_ESTRENO,TIMESTAMPDIFF(MONTH , FECHA_ESTRENO, CURRENT_DATE )
NUM_MESES_DESDE_ESTRENO FROM PELICULAS;

+---------------------------------+---------------+-------------------------+
| TITULO                          | FECHA_ESTRENO | NUM_MESES_DESDE_ESTRENO |
+---------------------------------+---------------+-------------------------+
| SUBWAY                          | 1985-04-10    |                     345 |
| NIKITA                          | 1990-02-21    |                     287 |
| STAR WARS 6: EL RETORNO DEL JEDI| 1983-10-19    |                     363 |
| AVATAR                          | 2009-10-16    |                      51 |
| BIENVENIDO AL NORTE             | 2008-02-27    |                      70 |
+---------------------------------+---------------+-------------------------+
```

Introducción

El lenguaje SQL permite asignar derechos a los usuarios a través de comandos DCL (*Data Control Language*o en castellano lenguaje de control de los datos (LCD)).

En general, los comandos del DCL normalmente los utiliza el DBA (*DataBase Administrator*). La seguridad de los datos es muy importante en una empresa, por tanto estas funciones se tienen que manejar con precaución.

Por qué definir derechos

En una empresa existen tipos de usuarios muy diferentes. Algunos simplemente quieren consultar algunas tablas, otros tienen que insertar y modificar datos, las personas del departamento de informática tendrán necesidades más amplias como crear o eliminar tablas.

Se tiene que poder controlar los accesos a la base de datos y permitir a cada uno realizar sus funciones sin interferir en el trabajo de otra persona.

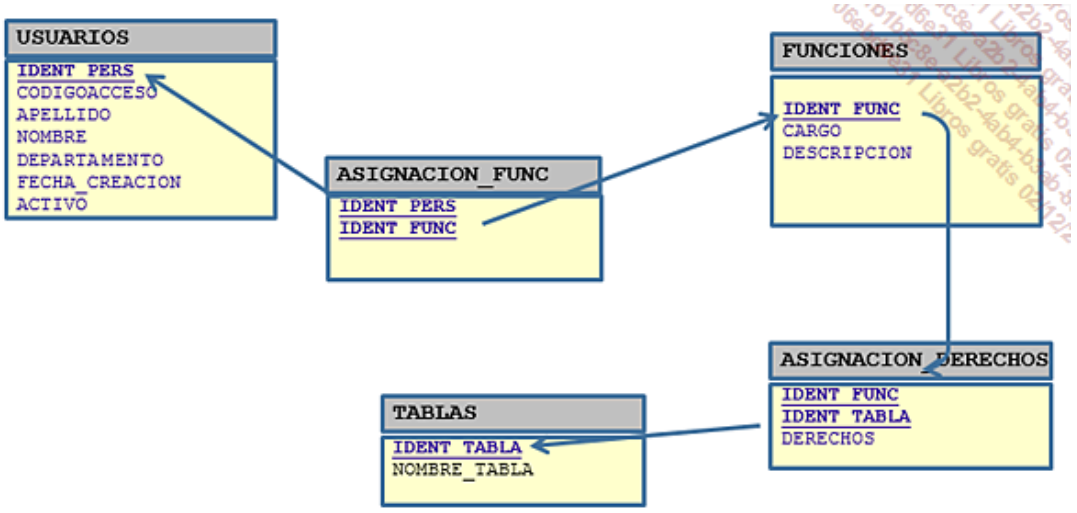
Antes de asignar derechos a una tabla, hay que clasificar a todos los usuarios de la base de datos en función de su trabajo y determinar para cada uno qué tablas utiliza y para cada una de las tablas indicar si puede actualizar o sólo tener acceso de lectura.

Este trabajo lo deben hacer personas del departamento de informática y los responsables de las aplicaciones (a menudo llamados Gestores de proyecto).

Una vez se ha recabado toda esta información, podemos clasificarlas en una tabla para poder hacer evolucionar esta gestión de los derechos y que pueda mantenerla el administrador de la base de datos u otra persona del departamento de informática dedicada a este trabajo.

Por ejemplo:

Descripción de un modelo de datos que permite gestionar los derechos de los usuarios.



Modelo de datos utilizado para ilustrar el DCL

TABLA: USUARIOS

Consulta de creación de la tabla:

```
CREATE TABLE USUARIOS (IDENT_PERS INTEGER,CODIGOACCESO CHAR(8),
APELLIDO VARCHAR(50), NOMBRE VARCHAR(50), DEPARTAMENTO VARCHAR(40),
FECHA_CREACION DATE, ACTIVO CHAR(1));
```

Consultas de inserción de datos:

Contenido de la tabla USUARIOS:

```
INSERT INTO USUARIOS VALUES
(1,'ASMITH','SMITH','ALAN','CORREO',TO_DATE('03/04/2002','DD/MM/
YYYY'),'S');
INSERT INTO USUARIOS VALUES
(2,'GPEROC','PEROC','GEORGES','COMERCIAL',TO_DATE('04/05/2006',
'DD/MM/YYYY'),'S');
INSERT INTO USUARIOS VALUES
(3,'MBALTOP','BALTOP','MICHELE','DIRECCION',TO_DATE('01/02/1998',
'DD/MM/YYYY'),'S');
INSERT INTO USUARIOS VALUES
(4,'SVERSACE','VERSACE','SANDRINE','INFORMATICA
DEV',TO_DATE('05/02/2007','DD/MM/YYYY'),'S');
INSERT INTO USUARIOS VALUES
(5,'XMARTIN','MARTIN','XAVIER','CONTROLE DE
GESTION',TO_DATE('06/11/1987','DD/MM/YYYY'),'S');
INSERT INTO USUARIOS VALUES
(6,'RSANCHEZ','SANCHEZ','ROBERTO','FACTURACION',TO_DATE('08/09/2001',
'DD/MM/YYYY'),'S');
INSERT INTO USUARIOS VALUES
(7,'MSEGAFRE','SEGAFREDI','MANUELLA','ADMINISTRADOR',TO_DATE('15/
```

TABLA: FUNCIONES

Consulta de creación de la tabla:

Consultas de inserción de datos:

Contenido de la tabla FUNCIONES:

TABLA:

```
05/2005','DD/MM/YYYY'),'S');
```

IDENT_PERS	CODIGOACCESO	APELLIDO	NOMBRE	DEPARTAMENTO	FECHA_CREACION	ACTIVO
1	ASMITH	SMITH	ALAN	CORREO	03/04/02	S
2	GPEROC	PEROC	GEORGES	COMERCIAL	04/05/06	S
3	MBALTOP	BALTOP	MICHELE	DIRECCION	01/02/98	S
4	SVERSACE	VERSACE	SANDRINE	INFORMATICA	05/02/07	S
5	XMARTIN	MARTIN	XAVIER	CONTROL DE GESTION	06/11/87	S
6	RSANCHEZ	SANCHEZ	ROBERTO	FACTURACION	08/09/01	S
7	MSEGAFRE	SEGAFREDI	MANUELLA	ADMINISTRADOR	15/05/05	S

```
CREATE TABLE FUNCIONES (IDENT_FUNC INTEGER, CARGO VARCHAR(50),
DESCRIPCION VARCHAR(80));
```

```
INSERT INTO FUNCIONES VALUES (1,'GESTION COMERCIAL','INTRODUCIR
PEDIDOS');
INSERT INTO FUNCIONES VALUES (2,'GESTION COMERCIAL','INTRODUCIR
CLIENTES');
INSERT INTO FUNCIONES VALUES (3,'FACTURACION','GESTION
FACTURAS');
INSERT INTO FUNCIONES VALUES (4,'FACTURACION','CONTROL
FACTURAS');
INSERT INTO FUNCIONES VALUES (5,'STOCK','CONSULTAS');
INSERT INTO FUNCIONES VALUES (6,'STOCK','ASIGNACION
UBICACION');
INSERT INTO FUNCIONES VALUES (7,'GESTION COMERCIAL','CONSULTA
CLIENTES');
```

IDENT_FUNC	CARGO	DESCRIPCION
1	GESTION COMERCIAL	INTRODUCIR PEDIDOS
2	GESTION COMERCIAL	INTRODUCIR CLIENTES
3	FACTURACION	GESTION FACTURAS
4	FACTURACION	CONTROL FACTURAS
5	STOCK	CONSULTAS
6	STOCK	ASIGNACION UBICACION
7	GESTION COMERCIAL	CONSULTA CLIENTES

ASIGNACION_FUNC

Consulta de creación de la tabla:

```
CREATE TABLE ASIGNACION_FUNC (IDENT_PERS INTEGER, IDENT_FUNC
INTEGER);
```

Consultas de inserción de datos:

Contenido de la tabla

```
INSERT INTO ASIGNACION_FUNC VALUES (1,1);
INSERT INTO ASIGNACION_FUNC VALUES (1,4);
INSERT INTO ASIGNACION_FUNC VALUES (1,7);
INSERT INTO ASIGNACION_FUNC VALUES (2,2);
INSERT INTO ASIGNACION_FUNC VALUES (2,3);
INSERT INTO ASIGNACION_FUNC VALUES (3,4);
INSERT INTO ASIGNACION_FUNC VALUES (4,1);
INSERT INTO ASIGNACION_FUNC VALUES (4,7);
INSERT INTO ASIGNACION_FUNC VALUES (5,2);
INSERT INTO ASIGNACION_FUNC VALUES (6,3);
INSERT INTO ASIGNACION_FUNC VALUES (7,1);
INSERT INTO ASIGNACION_FUNC VALUES (7,2);
INSERT INTO ASIGNACION_FUNC VALUES (7,3);
INSERT INTO ASIGNACION_FUNC VALUES (7,4);
INSERT INTO ASIGNACION_FUNC VALUES (7,5);
INSERT INTO ASIGNACION_FUNC VALUES (7,6);
```

ASIGNACION_FUNC:

IDENT_PERS	IDENT_FUNC
1	1
1	4
1	7
2	2
2	3
3	4
4	1
4	7
5	2
6	3
7	1
7	2
7	3
7	4
7	5
7	6

TABLA:
TABLES

Consulta de creación de la tabla:

Consultas de inserción de datos:

```
INSERT INTO TABLE
INSERT INTO TABLE
INSERT INTO TABLE
INSERT INTO TABLE
INSERT INTO TABLE
INSERT INTO TABLE
INSERT INTO TABLE
INSERT INTO TABLE
INSERT INTO TABLE
```

Contenido de la tabla TABLAS:

TABLA:

```
CREATE TABLE TABLAS (IDENT_TABLA INTEGER, NOMBRE_TABLA VARCHAR(40));
```

IDENT_TABLA	NOMBRE_TABLA
1	CLIENTES
2	PROVEEDORES
3	FACTURAS
4	PEDIDOS
5	STOCK
6	PED_FACT
7	COMPRADOR
8	ARTICULOS
999	TODAS

ASIGNACION_DERECHOS

Consulta de creación de la tabla:

```
CREATE TABLE ASIGNACION_DERECHOS (IDENT_FUNC INTEGER, IDENT_TABLA
INTEGER, DERECHOS VARCHAR(30));
```

Consultas de inserción de los datos:

Contenido de la tabla

```
INSERT INTO ASIGNACION_DERECHOS VALUES (1,1,'SELECCION');
INSERT INTO ASIGNACION_DERECHOS VALUES (1,1,'ELIMINACION');
INSERT INTO ASIGNACION_DERECHOS VALUES (1,1,'MODIFICACION');
INSERT INTO ASIGNACION_DERECHOS VALUES (1,2,'TODO');
INSERT INTO ASIGNACION_DERECHOS VALUES (1,3,'TODO');
INSERT INTO ASIGNACION_DERECHOS VALUES (2,4,'SELECCION');
INSERT INTO ASIGNACION_DERECHOS VALUES (2,5,'SELECCION');
INSERT INTO ASIGNACION_DERECHOS VALUES (2,4,'INSERCION');
INSERT INTO ASIGNACION_DERECHOS VALUES (3,1,'TODO');
INSERT INTO ASIGNACION_DERECHOS VALUES (3,2,'TODO');
INSERT INTO ASIGNACION_DERECHOS VALUES (3,3,'TODO');
INSERT INTO ASIGNACION_DERECHOS VALUES (4,999,'TODO');
INSERT INTO ASIGNACION_DERECHOS VALUES (5,999,'SELECCION');
```

```
INSERT INTO ASIGNACION_DERECHOS VALUES (6,999,'TODO');
INSERT INTO ASIGNACION_DERECHOS VALUES (7,1,'SELECCION');
```

ASIGNACION_DERECHOS:

IDENT_FUNC	IDENT_TABLA	DERECHOS
1	1	SELECCION
1	1	ELIMINACION
1	1	MODIFICACION
1	2	TODO
1	3	TODO
2	4	SELECCION
2	5	SELECCION
2	4	INSERCION
3	1	TODO
3	2	TODO
3	3	TODO
4	999	TODO
5	999	SELECCION
6	999	TODO
7	1	SELECCION

El valor 999 en la segunda columna significa que no se indica número de tabla, ya que se toman todas las tablas por defecto. Se asigna el derecho indicado en la columna tres a todas las tablas.

El « TODO » en la columna 3 significa que se dan todos los derechos en la tabla indicada en la columna 2.

Crear un usuario

Los usuarios suelen ser creados por el administrador de la base de datos según las reglas de seguridad propias de cada empresa. El comando de creación de un usuario es muy simple.

Sintaxis Oracle:

```
CREATE <usuario> IDENTIFIED BY <Contraseña>;
```

Sintaxis MySQL:

```
CREATE ’<usuario>’ IDENTIFIED BY ’<Contraseña>’;
```

Ejemplo en Oracle:

```
CREATE USER ASMITH IDENTIFIED BY ASMITH;
```

Ejemplo en MySQL:

```
CREATE USER ’ASMITH’ IDENTIFIED BY ’ASMITH’;
```

Otro método para declarar un usuario en MySQL:

```
GRANT ALL PRIVILEGES ON *.* TO ASMITH@localhost
 IDENTIFIED BY ’ASMITH’ WITH GRANT OPTION;
```

El Sr. Smith podrá conectarse a la base de datos utilizando ASMITH/ASMITH@<Nombre de la base>.

La creación de un usuario no le permite acceder a las tablas y hacer selecciones. Hay que asignarle derechos específicos en función de su perfil.

Cambiar la contraseña de un usuario

Por razones de seguridad, puede ser necesario modificar la contraseña de un usuario. En la mayor parte de sistemas, es el DBA el que puede utilizar este comando.

Sintaxis Oracle:

```
ALTER USER <usuario> IDENTIFIED BY <Nueva contraseña>;
```

Sintaxis MySQL:

```
SET PASSWORD FOR ’<usuario>’@’<host>’ = PASSWORD(’contraseña’);
```

Ejemplo Oracle:

```
ALTER USER ASMITH IDENTIFIED BY ABCD12E;
```

Ejemplo MySQL:

```
SET PASSWORD FOR ASMITH@localhost=PASSWORD(’ABCD12E’);
```

Asignar derechos (GRANT)

1. Asignar derechos de manipulación de una tabla

A partir de algunas de estas tablas de ejemplos, vamos a poder asignar derechos a los usuarios con los comandos GRANT y REVOKE.

La asignación de los derechos está reservada a los creadores de la tabla; no obstante el administrador de la base de datos puede dar autorización a otro usuario para administrar los derechos sobre las tablas.

El comando GRANT permite asignar accesos por usuario sobre una o más tablas.

Los derechos más utilizados son:

- SELECT: autoriza la selección de datos.
- UPDATE: autoriza la modificación de datos.
- DELETE: autoriza la eliminación de datos.
- INSERT: autoriza la inserción de datos.

La sintaxis es la siguiente:

```
GRANT <derecho1>, <derecho2>, ...
ON TABLE <nombre tabla>
TO <usuario1>, <usuario2> ...
[ WITH GRANT OPTION]
```

Para asignar todos los derechos, hay que utilizar la siguiente sintaxis:

Sintaxis Oracle:

```
GRANT ALL PRIVILEGES
ON <tabla1>, <tabla2>,  ...
[ WITH GRANT OPTION] TO <usuario1>, <usuario2> ...
```

Sintaxis MySQL:

```
GRANT ALL PRIVILEGES
ON TABLE <tabla1>, <tabla2>,  ...
[ WITH GRANT OPTION] TO <usuario1>, <usuario2> ...
```

El ALL PRIVILEGES otorga TODOS los derechos por lo que se tiene que utilizar con precaución y preferentemente asignarlos a personas de perfil administrador.

La cláusula WITH GRANT OPTION da autorización a los usuarios designados para asignar estos derechos sobre estas tablas a otros usuarios.

El análisis de las tablas de ejemplos nos permite sacar los elementos que nos indicarán cómo asignar o no derechos a los usuarios con el comando GRANT.

Selección de los derechos por usuario:

```
SELECT CODIGOACCESO,ASFUNC.IDENT_FUNC,NOM_TABLA,DERECHOS FROM
USUARIOS USU, FUNCIONES FUNC, ASIGNACION_FUNC ASFUNC,
      TABLAS TAB , ASIGNACION_DERECHOS ASDERECHOS
WHERE USU.IDENT_PERS         = ASFUNC.IDENT_PERS AND
      ASFUNC.IDENT_FUNC      = FUNC.IDENT_FUNC AND
      FUNC.IDENT_FUNC        = ASDERECHOS.IDENT_FUNC AND
```

```
      ASDERECHOS.IDENT_TABLA = TAB.IDENT_TABLA
ORDER BY CODIGOACCESO,ASFUNC.IDENT_FUNC;
```

CODIGOACCESO	IDENT_ FUNC	NOM_TABLA	DERECHOS
ASMITH	1	FACTURAS	TODO
ASMITH	1	PROVEEDORES	TODO
ASMITH	1	CLIENTES	SELECCION
ASMITH	1	CLIENTES	ELIMINACION
ASMITH	1	CLIENTES	MODIFICACION
ASMITH	4	TODO	TODO
ASMITH	7	CLIENTES	SELECCION
GPEROC	2	STOCK	SELECCION
GPEROC	2	PEDIDOS	SELECCION
GPEROC	2	PEDIDOS	INSERCION
GPEROC	3	FACTURAS	TODO
GPEROC	3	CLIENTES	TODO
GPEROC	3	PROVEEDORES	TODO
MBALTOP	4	TODO	TODO
MSEGAFRE	1	PROVEEDORES	TODO
MSEGAFRE	1	FACTURAS	TODO
MSEGAFRE	1	CLIENTES	SELECCION
MSEGAFRE	1	CLIENTES	ELIMINACION
MSEGAFRE	1	CLIENTES	MODIFICACION
MSEGAFRE	2	PEDIDOS	INSERCION
MSEGAFRE	2	STOCK	SELECCION
MSEGAFRE	2	PEDIDOS	SELECCION
MSEGAFRE	3	CLIENTES	TODO
MSEGAFRE	3	FACTURAS	TODO
MSEGAFRE	3	PROVEEDORES	TODO
MSEGAFRE	4	TODO	TODO
MSEGAFRE	5	TODO	SELECCION
MSEGAFRE	6	TODO	TODO
RSANCHEZ	3	FACTURAS	TODO
RSANCHEZ	3	CLIENTES	TODO
RSANCHEZ	3	PROVEEDORES	TODO
SVERSACE	1	FACTURAS	TODO
SVERSACE	1	CLIENTES	MODIFICACION
SVERSACE	1	CLIENTES	ELIMINACION

SVERSACE	1	CLIENTES	SELECCION
SVERSACE	1	PROVEEDORES	TODO
SVERSACE	7	CLIENTES	SELECCION
XMARTIN	2	PEDIDOS	INSERCION
XMARTIN	2	PEDIDOS	SELECCION
XMARTIN	2	STOCK	SELECCION

Si cogemos la tercera fila:

ASMITH	CLIENTES	SELECCION

Se puede traducir de la siguiente forma:

```
GRANT SELECT ON TABLE CLIENTES TO ASMITH;
```

Ahora, para gestionar todas las autorizaciones del usuario ASMITH, hay que escribir varios comandos. También se puede ver que Smith tiene varias funciones (1, 4 y 7), y los accesos a las tablas se acumulan.

CODIGOACCESO	**IDENT_ FUNC**	**NOM_TABLA**	**DERECHOS**
ASMITH	1	FACTURAS	TODO
ASMITH	1	PROVEEDORES	TODO
ASMITH	1	CLIENTES	SELECCION
ASMITH	1	CLIENTES	ELIMINACION
ASMITH	1	CLIENTES	MODIFICACION
ASMITH	4	TODO	TODO
ASMITH	7	CLIENTES	SELECCION

Smith tiene en una fila NOM_TABLA a « TODO » y DERECHOS a « TODO », lo que significa que tiene todos los derechos de acceso sobre todas las tablas.

Lo que se traduce como:

```
GRANT ALL PRIVILEGES ON CLIENTES TO ASMITH;
GRANT ALL PRIVILEGES ON PROVEEDORES TO ASMITH;
GRANT ALL PRIVILEGES ON FACTURAS TO ASMITH;
GRANT ALL PRIVILEGES ON PEDIDOS TO ASMITH;
GRANT ALL PRIVILEGES ON STOCK TO ASMITH;
GRANT ALL PRIVILEGES ON PED_FACT TO ASMITH;
GRANT ALL PRIVILEGES ON COMPRADORES TO ASMITH;
GRANT ALL PRIVILEGES ON ARTICULOS TO ASMITH;
```

Para asignar derechos a VERSACE sobre la tabla CLIENTES:

SVERSACE	1	CLIENTES	MODIFICACION
SVERSACE	1	CLIENTES	ELIMINACION
SVERSACE	1	CLIENTES	SELECCION

hay que escribir:

```
GRANT SELECT, DELETE, UPDATE ON CLIENTES TO AVERSACE;
```

También se puede ir más allá en el control de accesos especificando las columnas que se autorizan o no para cada usuario.

Por ejemplo, para autorizar a Smith a modificar solo las columnas IDENT_PERS, APELLIDO y NOMBRE de la tabla USUARIOS habrá que escribir:

```
GRANT UPDATE (IDENT_PERS,APELLIDO,NOMBRE) ON USUARIOS TO ASMITH;
```

Para asignar un privilegio a todos los usuarios de la base de datos, es posible utilizar la palabra clave PUBLIC:

```
GRANT UPDATE (IDENT_PERS,APELLIDO,NOMBRE) ON USUARIOS TO PUBLIC;
```

2. Asignar derechos sobre los objetos de la base

En la sección anterior hemos visto cómo asignar derechos sobre la manipulación de tablas. El comando GRANT también permite otorgar autorización para crear tablas, vistas o incluso sesiones.

Normalmente se les llama privilegios de sistema. Se tienen que manipular con precaución ya que se trata de la seguridad de la base de datos. Casi siempre son los DBA los que conceden estos derechos.

Los comandos GRANT se utilizan para dar o no derechos sobre comandos del LDD como CREATE, ALTER y DROP.

Se aplican sobre todos los objetos de la base de datos como:

- TABLAS
- INDICES
- SECUENCIAS
- SESIONES
- VISTAS
- PROCEDIMIENTOS
- SINONIMOS
- TRIGGERS
- Etc.

Por ejemplo, si se quiere autorizar la creación de tablas para el usuario ASMITH, se escribirá:

Sintaxis Oracle:

```
GRANT CREATE TABLE TO ASMITH;
```

Sintaxis MySQL:

```
GRANT CREATE ON * TO 'ASMITH';
```

De la misma manera, para dar derechos para modificar una tabla:

```
GRANT ALTER ANY TABLE TO ASMITH;
```

Al contrario que en el CREATE, los comandos DROP y ALTER llevan a continuación la palabra 'ANY'.

La palabra « ANY » significa que la autorización se da sólo sobre el esquema del usuario. Si no se pone la palabra « ANY », los derechos se aplican para todos los esquemas.

Para dar varios privilegios con un solo comando, la sintaxis es la siguiente:

```
GRANT CREATE TABLE,
DROP ANY TABLE,
ALTER ANY TABLE
TO ASMITH;
```

Con este comando, el usuario ASMITH podrá crear, modificar y eliminar cualquier tabla.

La sintaxis general en Oracle es:

```
GRANT <privilegio> [,<privilegio 2>,<privilegio 3>] <objeto> TO
<nombre usuario>;
```

La sintaxis general en MySQL es:

```
GRANT <privilegio> [,<privilegio 2>,< privilegio 3>] ON <objeto>,[*] TO
'<nombre usuario>';
```

En función de cada objeto, los derechos pueden ser diferentes. A continuación puede ver una lista con los derechos más frecuentes por objeto.

OBJETO	ACCIONES POSIBLES
TABLE	CREATE TABLE CREATE ANY TABLE ALTER ANY TABLE DROP ANY TABLE COMMENT ANY TABLE SELECT ANY TABLE INSERT ANY TABLE UPDATE ANY TABLE DELETE ANY TABLE LOCK ANY TABLE FLASHBACK ANY TABLE
INDEX	CREATE ANY INDEX ALTER ANY INDEX DROP ANY INDEX
VIEW (VISTA)	CREATE VIEW CREATE ANY VIEW DROP ANY VIEW COMMENT ANY TABLE FLASHBACK ANY TABLE
PROCEDURE	CREATE PROCEDURE CREATE ANY PROCEDURE ALTER ANY PROCEDURE DROP ANY PROCEDURE EXECUTE ANY PROCEDURE
SEQUENCE	CREATE SEQUENCE CREATE ANY SEQUENCE DROP ANY SEQUENCE SELECT ANY SEQUENCE
	CREATE SESSION ALTER SESSION

SESSION	ALTER RESOURCE COST RESTRICT SESSION
SYNONYM	CREATE SYNONYM CREATE ANY SYNONYM CREATE PUBLIC SYNONYM DROP ANY SYNONYM DROP PUBLIC SYNONYM
TRIGGER	CREATE TRIGGER CREATE ANY TRIGGER ALTER ANY TRIGGER DROP ANY TRIGGER ADMINISTER DATABASE TRIGGER

3. Otros derechos

En esta sección, vamos a ver algunos comandos GRANT que se aplican a objetos menos utilizados o menos conocidos.

Es posible asignar derechos sobre los objetos físicos de la base de datos como los TABLESPACES.

Por ejemplo, si se quiere autorizar al usuario ASMITH a crear TABLESPACES, se escribirá:

Ejemplo Oracle:

```
GRANT CREATE TABLESPACE TO ASMITH;
```

Del mismo modo, para que ASMITH pueda crear otros usuarios, se le asignará el derecho CREATE USER.

Ejemplo Oracle:

```
GRANT CREATE USER TO ASMITH;
```

Otro ejemplo: para autorizar al usuario ASMITH a crear y eliminar ROLES, se escribirá:

Ejemplo Oracle:

```
GRANT CREATE ROLE, DROP ANY ROLE TO ASMITH;
```

En función de cada objeto, los derechos pueden ser diferentes. A continuación puede ver una lista de los derechos por objeto en Oracle.

OBJETO	ACCIONES POSIBLES
DATABASE	ALTER SYSTEM AUDIT SYSTEM AUDIT ANY
PROFILE	CREATE PROFILE ALTER PROFILE DROP PROFILE
ROLE	CREATE ROLE DROP ANY ROLE ALTER ANY ROLE
TABLESPACE	CREATE TABLESPACE ALTER TABLESPACE DROP TABLESPACE

	MANAGE TABLESPACE UNLIMITED TABLESPACE
USER	CREATE USER ALTER USER DROP USER

También se pueden asignar derechos sobre acciones concretas como las siguientes (para Oracle):

ACCIÓN	SINTAXIS	EXPLICACIÓN
ANALYZE ANY	GRANT ANALYZE ANY TO ASMITH;	Autoriza a ASMITH a ejecutar análisis de objetos de la base.
GRANT ANY	GRANT GRANT ANY OBJECT PRIVILEGE TO ASMITH;	Autoriza a ASMITH a dar privilegios a otros usuarios sobre objetos que no sean del sistema.
	GRANT GRANT ANY PRIVILEGE TO ASMITH;	Autoriza a ASMITH a dar privilegios a otros usuarios sobre todos los objetos del sistema.
SYSDBA SYSOPER	GRANT SYSDBA TO ASMITH; o GRANT SYSOPER TO ASMITH;	Autoriza a ASMITH a ejecutar operaciones del sistema sobre la base de datos como STARTUP, SHUTDOWN o CREATE DATABASE por ejemplo.Observe que este GRANT lo debe realizar un usuario que sea SYSDBA o SYSOPER.
SELECT ANY DICTIONARY	GRANT SELECT ANY DICTIONARY TO ASMITH;	Autoriza a ASMITH a consultar todas las tablas de la cuenta SYS.


Puede visualizar los derechos asignados consultando la tabla: USER_TAB_PRIVS (SELECT * FROM USER_TAB_PRIVS;)

Eliminar derechos (REVOKE)

1. Eliminar derechos sobre la manipulación de una tabla

Una vez asignados los derechos, hay que poder quitarlos si el usuario abandona la empresa o cambia de departamento. Para ello, utilizaremos el comando REVOKE.

La sintaxis Oracle es la siguiente:

```
REVOKE <derecho1>, <derecho2>, ...
ON TABLE <nombre tabla>
FROM <usuario1>, <usuario2> ...;
```

La sintaxis MySQL es la siguiente:

```
REVOKE <derecho1>, <derecho2>, ...
ON [TABLE <nombre tabla>],[*]
FROM <usuario1>, <usuario2> ...;
```

Por ejemplo, para quitar los derechos de lectura de la tabla USUARIOS a SMITH:

```
REVOKE SELECT ON USUARIOS FROM ASMITH;
```

Para quitar todos los derechos a Smith:

```
REVOKE ALL PRIVILEGES ON USUARIOS FROM ASMITH;
```

Del mismo modo que el GRANT, si añadimos la palabra PUBLIC, se eliminan los derechos sobre esta tabla para todos los usuarios de la base de datos.

```
REVOKE ALL PRIVILEGES ON USUARIOS FROM PUBLIC;
```

2. Eliminar derechos sobre los objetos de la base

Como para la manipulación de una tabla, se pueden eliminar derechos que se han asignado para la creación de tablas o de índices.

Ejemplo Oracle:

```
REVOKE CREATE TABLE FROM ASMITH;
```

Ejemplo MySQL:

```
REVOKE CREATE ON * ON 'ASMITH';
```

Para eliminar los derechos de modificación, de eliminación y de creación de tablas en un solo comando, la sintaxis es:

Ejemplo Oracle:

```
REVOKE CREATE TABLE, ALTER ANY TABLE, DROP ANY TABLE FROM ASMITH;
```

Ejemplo MySQL:

```
REVOKE CREATE, ALTER, DROP ON * TO 'ASMITH';
```

Utilización de los roles

Un rol es un grupo al que se le van a asignar los mismos derechos. En lugar de asignar los derechos individualmente a cada usuario, se pueden crear grupos que tendrán los derechos y a continuación asignar los usuarios a un grupo.


Este concepto de rol no existe actualmente en MySQL.

Sintaxis Oracle:

```
CREATE ROLE <nombre rol>;
```

Ejemplo Oracle:

```
CREATE ROLE CONTROL_GESTION;
```

A continuación, hay que asignar derechos al rol que acabamos de crear de la misma manera que para un usuario x.

```
GRANT ALL PRIVILEGES ON CLIENTES TO CONTROL_GESTION;
GRANT ALL PRIVILEGES ON PROVEEDORES TO CONTROL_GESTION;
GRANT ALL PRIVILEGES ON FACTURAS TO CONTROL_GESTION;
```

A continuación asignamos este rol a los usuarios:

```
GRANT CONTROL_GESTION TO ASMITH;
GRANT CONTROL_GESTION TO BMARTIN;
etc ...
```

El concepto de rol es bastante interesante en sitios con muchos usuarios, ya que permite simplificar la gestión de los diferentes perfiles.

Se deben definir los diferentes roles en la utilización de la base de datos y en las aplicaciones existentes y a continuación clasificar los usuarios por rol. A cada persona nueva que llegue, tan solo hay que crear el usuario y asignarle un rol.

También existen roles definidos en Oracle que son CONNECT, DBA o RESOURCE pero están reservados a los administradores de la base de datos. Es preferible crear sus propios roles para gestionar la seguridad.

Eliminar un rol

Sintaxis Oracle:

```
DROP ROLE <nombre rol>;
```

Ejemplo Oracle:

```
DROP ROLE CONTROL_GESTION;
```

Ejercicios sobre la seguridad

Primer ejercicio

Crear un usuario ALFREDO y asignarle los derechos de crear una sesión y de seleccionar datos de la tabla FACTURAS.

Segundo ejercicio

Asignar a todos los usuarios el derecho de seleccionar datos de la tabla STOCK.

Tercer ejercicio

Asignar derechos al usuario ALFREDO para que pueda modificar en la tabla STOCK solo las columnas ARTICULO y CANTIDAD.

Cuarto ejercicio

Eliminar para el usuario ALFREDO los derechos sobre la modificación de la columna CANTIDAD.

Solución de los ejercicios

Primer ejercicio

Crear el usuario:

```
CREATE USER ALFREDO IDENTIFIED BY ALFREDO;
```

Autorizarlo a conectarse:

```
GRANT CREATE SESSION TO ALFREDO;
```

Permitirle seleccionar:

```
GRANT SELECT ON FACTURAS TO ALFREDO;
```

Segundo ejercicio

Utilización de la palabra PUBLIC:

```
GRANT SELECT ON STOCK TO PUBLIC;
```

Tercer ejercicio

Creación de la tabla STOCK:

```
CREATE TABLE STOCK (IDENT NUMBER(9), ARTICULO CHAR(10), CANTIDAD
NUMBER(10), FECHACOMPRA DATE, ULTSALIDA DATE, CANTIDADMIN NUMBER(10),
CANTIDADMAXNUMBER(10));
```

Inserción de valores:

```
INSERT INTO STOCK VALUES (1,'IPHONE',12,'10/10/2013','05/01/2014',2,40);
INSERT INTO STOCK VALUES (2,'GALAXY S',2,'01/05/2013','28/02/2014',1,25);
```

Asignación de derechos:

```
GRANT UPDATE (ARTICULO,CANTIDAD) ON STOCK TO ALFREDO;
```

Si ahora el usuario ALFREDO prueba a modificar otra columna, habrá un mensaje de error indicándole que no tiene derechos:

```
UPDATE STOCK SET CANTIDADMIN=9 WHERE IDENT =1
              *
ERROR en la fila 1:
ORA-01031: privilegios insuficientes
```

Por el contrario, en la columna CANTIDAD, sí puede:

```
UPDATE STOCK SET CANTIDAD=15 WHERE IDENT =1 ;

1 fila(s) se ha actualizado.
```

Cuarto ejercicio

Eliminación de los derechos UPDATE sobre la columna CANTIDAD.

No se pueden eliminar los derechos solo sobre una columna, el REVOKE se aplica sobre todas las columnas.

```
REVOKE UPDATE ON STOCK FROM ALFREDO;
```

A continuación deberá volver a asignar el derecho solo sobre la columna ARTICULO.

```
GRANT UPDATE (ARTICULO) ON STOCK TO ALFREDO;
```

La problemática de los accesos concurrentes

En la mayoría de desarrollos informáticos se plantea el problema de los accesos simultáneos a ciertos datos por parte de usuarios diferentes.

De hecho, un programador de aplicaciones debe prever la gestión de los accesos concurrentes utilizando las herramientas que proporciona la base de datos.

La mayoría de los SGBDR permiten bloquear datos antes de la actualización para impedir a otros usuarios modificar el mismo dato antes que el primero haya confirmado la modificación.

1. Ilustración de accesos concurrentes

a. Ejemplo 1: actualizaciones simultáneas

Volvamos a la tabla TELEFONO:

```
SELECT * FROM TELEFONO;
```

NUMERO	TIPO	MARCA	FECHA_COMPRA	PRECIO	NUM_PROPIETARIO	COLOR
1	SP	1	15/01/10	159	190120	ROJO
2	SP	1	14/03/10	99	190215	
3	CT	3	02/05/10	49	190001	NEGRO
4	DE	4	25/07/10	89	190222	BLANCO
5	IP	5	30/09/10	359	190561	

Ahora un usuario lee el registro número 2 y realiza una modificación de la tabla sumando 10 € al precio del teléfono.

En el mismo momento otro usuario también suma 15 € a este registro número 2.

USUARIO 1	Valor PRECIO	USUARIO 2	Valor PRECIO
LECTURA TELEFONO NUMERO 2	99		
ACTUALIZACIÓN PRECIO PRECIO = PRECIO + 10	109	LECTURA TELEFONO NUMERO 2	99
		ACTUALIZACIÓN PRECIO= PRECIO + 15	114
GRABACIÓN REGISTRO NUMERO 2	109		
		GRABACIÓN REGISTRO NUMERO 2	114

Como resultado de este pequeño escenario, el valor final del precio del teléfono será de 114 €. Tendría que ser de 124 €. De hecho, si las dos transacciones se hubieran realizado una después de la otra, el resultado hubiera sido:

99 + 10 = 109 y 109 + 15 = 124 €

Por tanto, el usuario 1 piensa que ha sumado 10 € al precio y el segundo usuario realmente ha sumado los 15 € al precio.

El usuario 2 no ha tenido en cuenta la modificación del primer usuario que pone el precio a 109. La suma

de los 15 € se debería aplicar a 109 y no a 99.

b. Ejemplo 2: incoherencia de los datos después de una modificación de otro usuario

Un usuario lee la tabla TELEFONO al inicio de su programa, y a continuación realiza acciones respecto a lo que ha recuperado.

Al final del proceso vuelve a leer la misma tabla para recuperar la misma información. En ese lapso de tiempo otro usuario ha realizado una modificación o una inserción en la tabla.

USUARIO 1	Valor COLOR	USUARIO 2	Valor COLOR
LECTURA TELEFONO NUMERO 3	NEGRO		
SI COLOR = NEGRO LLAMADA PROG 1		LECTURA TELEFONO NUMERO 3	NEGRO
SI COLOR = NEGRO ACTUALIZACION TABLA Y		ACTUALIZACION COLOR a «ROJO»	ROJO
		GRABACIÓN REGISTRO NÚMERO 3	ROJO
LECTURA TELEFONO NUMERO 3	ROJO		
SI COLOR = NEGRO LLAMADA PROG 2	ERROR		

El usuario 1 debe bloquear el registro para que nadie pueda modificarlo mientras se ejecuta su programa.

Este fenómeno se llama: **lectura no repetitiva**.

Otro ejemplo con la aparición de nuevas filas entre dos consultas.

USUARIO 1	Registros encontrados	USUARIO 2	Registros en la BD
LECTURA DEL NÚMERO DE TELÉFONOS DE TIPO « SP »	2		
PREPARACIÓN DE LA TABLA PARA ALMACENAR LOS TELÉFONOS CON DOS FILAS		INSERCIÓN DE DOS NUEVAS FILAS EN LA TABLA TELÉFONO DE TIPO « SP »	4
		GRABACIÓN DE LOS NUEVOS REGISTROS	4
LECTURA DEL NÚMERO DE TELÉFONOS DE TIPO « SP »	4		
INSERCIÓN DE LOS TELÉFONOS DE TIPO « SP » EN LA TABLA	ERROR EN NÚMERO FILAS > TAMAÑO DE LA TABLA		

Este fenómeno se llama: **filas fantasmas**.

2. El mecanismo de bloqueo

Los mecanismos de bloqueo existen en los SGBDR para permitir a los programadores que sus actualizaciones sean seguras y garantizar la coherencia de la base de datos.

Sin embargo, los bloqueos pueden provocar el bloqueo completo de los usuarios. Se trata de un problema de bloqueo ilustrado en el siguiente ejemplo.

USUARIO 1	**USUARIO 2**
BLOQUEO DE LA FILA 3 DE TELEFONO	
	BLOQUEO DE LA FILA 4 DE TELEFONO
BLOQUEO DE LA FILA 4 DE TELEFONO	
¡ ATENCIÓN !	BLOQUEO DE LA FILA 3 DE TELEFONO
	¡ ATENCIÓN !

Los dos usuarios se bloquean esperando que se libere la fila deseada.

Hay que « matar » uno de los dos usuarios para desbloquearlos.

Explicamos todo esto para mostrar que se debe integrar el bloqueo en las normas de desarrollo de la empresa. El ejemplo muestra que hay que liberar los recursos bloqueados lo más rápidamente posible.

Si hay que modificar muchas filas, es preferible bloquear la tabla entera para evitar los inter-bloqueos o deadlock en inglés.

Concepto de transacción

Para limitar los problemas indicados en los párrafos anteriores, el programador se debe preguntar: ¿cuáles son los objetos (fila, columna , tabla…) que manipulo en mi proceso y cómo evitar que otro usuario no pueda bloquearlos antes que yo haya terminado mis actualizaciones?

En este momento, comenzamos a hablar de « TRANSACCIÓN ».

1. Definición de una transacción

La transacción permite limitar el inicio y el final de una acción en la base de datos, sobre una o más tablas, que deben quedar coherentes.

La transacción es un concepto que proviene de las aplicaciones llamadas « transaccionales ». Cada vez que hay intercambio de datos entre la aplicación y el usuario, se habla de aplicación transaccional.

Todas las aplicaciones utilizan mecanismos de bloqueo.

2. Cómo evitar las incoherencias de datos

Existen varios métodos para garantizar la coherencia de la base de datos:

- Ejecutar las transacciones unas después de las otras (en serie). El inconveniente principal es el tiempo de espera para los usuarios. Esto equivale a tener una aplicación monousuario.
- Posibilidad de bloquear al principio del programa principal todos los objetos implícitamente y liberarlos al final. Este método corre el riesgo de bloquear al resto de usuarios mucho tiempo si el usuario no valida rápidamente la pantalla y, por ejemplo, se va a una reunión.
- Sin bloqueo de registros al principio, pero con lectura de los datos y guardado en memoria del contenido para volver a leerlos justo antes de la actualización, para asegurar que los datos no se han modificado en ese lapso de tiempo. Si los valores guardados son diferentes de los valores que se han vuelto a leer, habría que cancelar la transacción y mostrar un mensaje al usuario para indicar el problema y decirle que debe volver a introducir los datos.

3. Implementación de un bloqueo

Según la norma SQL-92, se puede indicar a la base de datos qué tipo de método se quiere utilizar.

Para activar cualquier modo de bloqueo, hay que utilizar el comando: SET TRANSACTION ISOLATION LEVEL.

Existen cuatro modos de bloqueo.

El modo por defecto en MySQL es el modo REPEATABLE-READ; en Oracle es el modo READ COMMITTED.

Generalmente la acción se aplica a una sesión, la del usuario activo, aunque algunos SGBDR permiten aplicar el nivel de bloqueo a todas las sesiones en curso, utilizando la palabra GLOBAL.

Vamos a describir los diferentes modos.

a. READ UNCOMMITTED

Cuando se activa este modo, el usuario tiene la posibilidad de visualizar las filas que están siendo modificadas por otra persona, incluso si no se ha confirmado la modificación. Desde el punto de vista funcional, no se puede garantizar la integridad de los datos; de hecho, como el otro usuario no ha confirmado su modificación, los datos son volátiles.

b. READ COMMITTED

El usuario solo puede visualizar los datos confirmados. En ningún caso se pueden ver filas que se estén modificando y no se hayan confirmado. Este es el modo de funcionamiento que utiliza Oracle. Este modo garantiza la coherencia de datos entre usuarios.

c. REPEATABLE-READ

Todos los datos utilizados por una consulta se bloquean, lo que impide cualquier modificación de datos por parte de otros usuarios. Por el contrario, el resto de usuarios tienen la posibilidad de insertar nuevas filas en la tabla. Este nivel es bastante restrictivo y puede provocar tiempos de espera largos a los usuarios.

d. SERIALIZABLE

Es el modo más restrictivo para los usuarios. Como contrapartida, es el modo más seguro para el programador, pero se desaconseja utilizarlo excepto en aplicaciones con muy pocos usuarios. Este modo bloquea toda la tabla a cada petición del usuario. El resto de usuarios no pueden realizar ninguna acción hasta que el primero no haya confirmado su transacción.

e. Sintaxis

Sintaxis MySQL:

```
SET [SESSION | GLOBAL] TRANSACTION ISOLATION LEVEL
{ READ UNCOMMITTED
| READ COMMITTED
| REPEATABLE READ
| SERIALIZABLE
};
```

Sintaxis Oracle:

```
SET TRANSACTION ISOLATION LEVEL
{ READ COMMITTED
| SERIALIZABLE
};
```

Oracle ofrece otras posibilidades, especialmente el READ ONLY que permite indicar que sólo se hace una lectura o el READ WRITE para indicar que se hará una actualización.

4. Implementación de un bloqueo aplicativo

Las reglas de bloqueo que se aplican para las transacciones las implementan casi siempre los administradores de la base de datos para el conjunto de los programadores.

Independientemente de las reglas que se gestionan a nivel de transacción, como se indica en la sección anterior, es posible bloquear filas de una tabla para evitar que otro usuario las modifique antes de que acabemos nuestro proceso.

Existen dos métodos principales para gestionar este caso. El bloqueo exclusivo (FOR UPDATE) que bloquea todas las filas devueltas por el SELECT, impidiendo cualquier modificación por parte de otro usuario y el bloqueo compartido (SHARE MODE) que deja la posibilidad al resto de usuarios de visualizar las filas bloqueadas en el estado anterior al bloqueo.

Sintaxis Oracle:

```
SELECT ... FROM .... WHERE ....
FOR UPDATE;
```

o

```
LOCK TABLE <nombre tabla> IN SHARE MODE;
```

Sintaxis MySQL:

```
SELECT ... FROM .... WHERE ....
FOR UPDATE;
```

o

```
SELECT ... FROM .... WHERE ....
LOCK IN SHARE MODE;
```

Ejemplo Oracle:

```
SELECT NUMERO, TIPO, NUM_PROPIETARIO, FECHA_COMPRA, MARCA FROM
TELEFONO
WHERE COLOR = 'ROJO'
FOR UPDATE;
```

o

```
LOCK TABLE TELEFONO IN SHARE MODE;
```

Ejemplo MySQL:

```
SELECT TEL.FECHA_COMPRA,
       TEL.TIPO,
       TIP.DESC_TIPO,
       TEL.MARCA,
       MAR.DESC_MARCA,
       TEL.COLOR
FROM  TELEFONO TEL ,
      TIPO_TEL TIP,
      MARCA_TEL MAR
WHERE TEL.TIPO  = TIP.TIPO
  AND TEL.MARCA = MAR.MARCA
  AND TEL.COLOR IN ('ROJO', 'BLANCO')
LOCK IN SHARE MODE;
```

En Oracle no se pueden hacer los bloqueos compartidos en un comando SELECT. Para ello se debe utilizar el comando LOCK TABLE que permite poner varios tipos de bloqueo sobre una tabla (ROW SHARE, ROW EXCLUSIVE, SHARE UPDATE, SHARE, SHARE ROW EXCLUSIVE, EXCLUSIVES). La descripción de cada opción se encuentra en la documentación oficial.

a. Cómo saber los bloqueos que tiene una tabla

Cuando se accede a una tabla y el SGBDR nos indica que este dato no está disponible devolviendo el código ORA-00054 u ORA-00060, es posible saber los bloqueos que tiene una tabla.

Para ello, se debe acceder a las tablas de « sistema » que contienen todas las acciones realizadas en un instante t por los usuarios de la base de datos.

Si un usuario ha bloqueado la tabla TELEFONO, habrá ejecutado una consulta de este tipo:

```
SELECT * FROM TELEFONO FOR UPDATE;
```

Si otra persona intenta actualizar esta tabla, se le devolverá un error indicando que la tabla está « bloqueada ».

Para conocer la lista de las tablas bloqueadas, hay que consultar la tabla V$LOCKED_OBJECT que tiene un registro por cada objeto y la sesión que lo utiliza. Si se hace una unión con la tabla ALL_OBJECTS, se pueden listar las tablas en cuestión.

Ejemplo Oracle:

```
SELECT T1.OBJECT_ID, T2.OBJECT_NAME, T1.SESSION_ID FROM V$LOCKED_OBJECT
T1, ALL_OBJECTS T2 WHERE T1.OBJECT_ID = T2.OBJECT_ID;
```

El resultado es:

```
 OBJECT_ID OBJECT_NAME                    SESSION_ID
---------- ------------------------------ ----------
     13637 TELEFONO                               28
```

Para saber quién es el usuario de la sesión 28, en nuestro ejemplo, se debe consultar la tabla V$SESSION que contiene otra información

sobre el usuario.

Ejemplo Oracle:

El

```
SELECT T3.SERIAL# ,T3.SID,T1.OBJECT_ID,T2.OBJECT_NAME,
T1.SESSION_ID,T3.USERNAME,T3.STATUS,T3.OSUSER,T3.PROCESS
FROM V$LOCKED_OBJECT T1, ALL_OBJECTS T2, V$SESSION T3
WHERE T1.OBJECT_ID = T2.OBJECT_ID
  AND T3.SID       = T1.SESSION_ID;
```

resultado es:

En el

```
   SERIAL#        SID  OBJECT_ID OBJECT_NAME
---------- ---------- ---------- ------------------------------
       997         28      13637 TELEFONO

SESSION_ID USERNAME                       STATUS
---------- ------------------------------ ------
        28 SYSTEM                         ACTIVE
OSUSER                       PROCESS
-- ----------------------- ---------
xxxxxxxxxxxx               7328:9188
```

resultado, se observa el OSUSER que es el nombre del usuario, el nombre de conexión en USERNAME y la tabla en OBJECT_NAME.

b. Cómo eliminar bloqueos que tenga una tabla

Ahora que hemos visto cómo encontrar los bloqueos que tiene una tabla, se puede, en la medida de lo derechos otorgados por el DBA, eliminar este bloqueo.

Existen dos métodos para eliminar un bloqueo: eliminar la sesión Oracle del usuario, o eliminar la sesión del sistema (UNIX/Windows).

El primer método consiste en desconectar el usuario de la base de datos, lo que libera la sesión y la/s tabla/s bloqueada/s.

Si retomamos el ejemplo anterior, teníamos:

Los

```
   SERIAL#        SID  OBJECT_ID OBJECT_NAME
---------- ---------- ---------- ------------------------------
       997         28      13637 TELEFONO

SESSION_ID USERNAME                       STATUS
---------- ------------------------------ ------
        28 SYSTEM                         ACTIVE
OSUSER                       PROCESS
-- ----------------------- ---------
xxxxxxxxxxxx               7328:9188
```

dos elementos más importantes son el SID y el SERIAL#. Con estos dos números, puede cancelar la sesión del usuario con el comando ALTER SYSTEM KILL SESSION 'SID,SERIAL#';.

Ejemplo Oracle:

La

```
ALTER SYSTEM KILL SESSION '28,997';
```

sesión del usuario queda abierta (pero inactiva), pero en el próximo comando se le devolverá el siguiente mensaje, que significa que ya no está conectado a la base de datos:

El

```
ORA-03113: fin de archivo en canal de comunicación
```

segundo método consiste en cancelar la sesión de sistema del usuario, la sesión UNIX o Windows.

Retomemos el ejemplo de la sección anterior.

Ejemplo Oracle:

El

```
SELECT T3.SERIAL# ,T3.SID,T1.OBJECT_ID,T2.OBJECT_NAME,
T1.SESSION_ID,T3.USERNAME,T3.STATUS,T3.OSUSER,T3.PROCESS
FROM V$LOCKED_OBJECT T1, ALL_OBJECTS T2, V$SESSION T3
WHERE T1.OBJECT_ID = T2.OBJECT_ID
  AND T3.SID       = T1.SESSION_ID;
```

resultado es:

El

```
   SERIAL#        SID  OBJECT_ID OBJECT_NAME
---------- ---------- ---------- ------------------------------
       997         28      13637 TELEFONO

SESSION_ID USERNAME                       STATUS
---------- ------------------------------ ------
        28 SYSTEM                         ACTIVE
OSUSER                       PROCESS
-- ------------------------ ---------
xxxxxxxxxxxx                 7328:9188
```

contenido de la columna PROCESS corresponde en UNIX al PID que permite cancelar la sesión en caso de bloqueo. Para ello, deberá hacer un kill -9 7328.

En Windows, para encontrar el número de sesión, hay que ir al Administrador de tareas, seleccionar la pestaña **Procesos**, hacer clic en la fila cuyo número de la columna **PID** corresponda al buscado, y a continuación hacer clic en el botón **Finalizar proceso** o clic derecho y **Terminar el proceso**.

5. Validación de las modificaciones (COMMIT)

El principio básico en todas los SGBDR es dejar al usuario la tarea de confirmar sus modificaciones al final de la transacción y hacer así visibles los cambios realizados en los datos al resto de usuarios.

El programador puede programar esto en el momento que quiera para garantizar la integridad referencial de la base de datos. Por tanto, está vinculado al propio funcionamiento de cada programa.

El COMMIT valida TODAS las modificaciones, inserciones o eliminaciones que se han realizado desde el último COMMIT o desde el inicio de la transacción.

En un programa, se pueden poner varios COMMIT que permitan separar el tratamiento en secuencias coherentes funcionalmente.

El COMMIT permite igualmente liberar todos los bloqueos que se hubieran puesto anteriormente.

Sintaxis:

```
COMMIT;
```

También existe el concepto de AUTOCOMMIT especialmente en MySQL y Oracle que indica que cada comando SQL es una transacción y por tanto cada comando se confirma automáticamente después de su ejecución.

Esta opción se debe manejar con precaución ya que impide cualquier vuelta atrás en caso de error.

Sintaxis:

```
SET AUTOCOMMIT ON;
SET AUTOCOMMIT OFF;
```

6. Cancelación de las modificaciones (ROLLBACK)

El ROLLBACK es el contrario del COMMIT. Este invalida todas las modificaciones, inserciones y eliminaciones que se hubieran realizado desde el último COMMIT, desde el inicio de la transacción o desde el último punto de recuperación (SAVEPOINT).

Es muy útil cuando se realizan pruebas de comandos SQL sobre una base de datos y se comenten errores. En este caso, antes de salir, hay que ejecutar un ROLLBACK y la base de datos vuelve a su estado inicial.

Sintaxis:

```
ROLLBACK [TO SAVEPOINT <nombre del punto de recuperación>];
```

7. Los puntos de sincronización (SAVEPOINT)

Cuando se coloca un conjunto de sentencias SQL para validar un proceso funcional, existe la posibilidad de poner puntos de control en varios momentos, lo que permite, en caso de algún problema, volver a un punto en el que los datos sean coherentes.

Tenga en cuenta que un COMMIT anula todos los SAVEPOINT. Un ROLLBACK sin SAVEPOINT también anula todos los SAVEPOINT y devuelve a la base de datos a su estado inicial.

Sintaxis:

```
SAVEPOINT <nombre del punto de control>;
```

Ejemplo:

```
SAVEPOINT ANTES_INSERT;
```

8. Ejemplo de utilización de los puntos de sincronización

Vaciado de una tabla y a continuación creación de filas con activación de puntos de sincronización antes de cada inserción:

```
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

/* Vaciado de la tabla PELICULAS */
DELETE FROM PELICULAS;

SELECT COUNT(*) FROM ACTOR WHERE APELLIDO = 'FISHER' AND NOMBRE =
'CARRIE';
```

Primera inserción en la tabla PELICULAS: SUBWAY

```
INSERT INTO PELICULAS VALUES (1,'SUBWAY','POLICIACA','DRAMA',
TO_DATE('10/04/1985','DD/MM/YYYY'),1,FILMAX',
'Cuenta las aventuras de la población subterránea en los túneles
del metro de París');
```

Primera sincronización

```
/* Posicionamiento del primer punto */
SAVEPOINT FASE_NUMERO1;
```

Segunda inserción en la tabla PELICULAS: NIKITA

```
INSERT INTO PELICULAS VALUES (2,'NIKITA','DRAMA','ROMANTICA',
TO_DATE('21/02/1990','DD/MM/YYYY'),1,1,'FILMAX',
'Nikita, condenada a cadena perpetua, es obligada a trabajar en secreto
para el gobierno como agente de los servicios secretos.');

DELETE FROM ACTOR WHERE APELLIDO = 'FISHER' AND NOMBRE = 'CARRIE';

SELECT * FROM PELICULAS T1 WHERE
   EXISTS (SELECT IDENT_DIRECTOR FROM DIRECTOR
   WHERE NOMBRE = 'LUC' AND IDENT_DIRECTOR =
   T1.IDENT_DIRECTOR) AND FECHA_ESTRENO
   BETWEEN ('01/01/85') AND ('30/05/1995')
ORDER BY TITULO;
```

Segunda sincronización

```
/* Posicionamiento del segundo punto */
SAVEPOINT FASE_NUMERO2;

INSERT INTO PELICULAS VALUES (3,'SUBWAY','POLICIACA','DRAMA',
TO_DATE('10/04/1985','DD/MM/YYYY'),1,1,'FILMAX',
'Cuenta las aventuras de la población subterránea en los túneles del
metro de París');

/* Retorno al punto número 2 para no insertar dos veces la película
'SUBWAY' */
ROLLBACK TO FASE_NUMERO2;

COMMIT;
```

resultado es el siguiente:

Y si

```
SQL> SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

Transacción definida.

SQL>
SQL> /* Vaciado de la tabla PELICULAS */
SQL> DELETE FROM FILM;

2 fila(s) eliminadas(s).

SQL>
SQL> SELECT COUNT(*) FROM ACTOR WHERE APELLIDO = 'FISHER' AND NOMBRE =
'CARRIE';


  COUNT(*)
----------
         1

SQL>
SQL> INSERT INTO PELICULAS VALUES (1,'SUBWAY','POLICIACA','DRAMA',
TO_DATE('10/04/1985','DD/MM/YYYY'),1,1,'FILMAX',
'Cuenta las aventuras de la población subterránea en los túneles del
metro de París');
1 fila insertada.
```

```
SQL>
SQL> /* Posicionamiento del primer punto */
SQL> SAVEPOINT FASE_NUMERO1;
Savepoint creado.
```

```
SQL>
SQL> INSERT INTO PELICULAS VALUES (2,'NIKITA','DRAMA','ROMANTICA',
TO_DATE('21/02/1990','DD/MM/YYYY'),1,1, 'FILMAX',
'Nikita, condenada a cadena perpetua, es obligada a trabajar en
secreto para el gobierno como agente de los servicios secretos.');

1 fila insertada.
```

```
SQL>
SQL> DELETE FROM ACTOR WHERE APELLIDO = 'FISHER' AND NOMBRE = 'CARRIE';
1 fila(s) eliminada(s).

SQL>
SQL> SELECT * FROM FILM T1 WHERE
  2   EXISTS (SELECT IDENT_DIRECTOR FROM DIRECTOR WHERE
       NOMBRE = 'LUC' AND IDENT_DIRECTOR = T1.IDENT_DIRECTOR)
  3   AND FECHA_ESTRENO BETWEEN ('01/01/85') AND ('30/05/1995')
  4   ORDER BY TITULO;

IDENT_PELI TITULO
---------- --------------------------------------------------
GENERO1              GENERO2              FECHA_ES      PAIS
```

```
IDENT_DIRECTOR
------------------- -------------------- -------- ----------
-----------------
DISTRIBUIDOR
--------------------------------------------------
SINOPSIS
----------------------------------------------------------------
--------------

        2 NIKITA
DRAMA                ROMANTICA            21/02/90          1
1
FILMAX
Nikita, condenada a cadena perpetua, es obligada a trabajar en secreto
para el gobierno como agente de los servicios secretos.


IDENT_PELI TITULO
---------- --------------------------------------------------
GENERO1              GENERO2              FECHA_ES      PAIS
IDENT_DIRECTOR
------------------- -------------------- -------- ----------
-----------------
DISTRIBUIDOR
--------------------------------------------------
SINOPSIS
----------------------------------------------------------------
--------------

s.

         1 SUBWAY
POLICIACA            DRAMA                10/04/85          1
1
FILMAX

IDENT_PELI TITULO
---------- --------------------------------------------------
GENERO1              GENERO2              FECHA_ES      PAIS
IDENT_DIRECTOR
------------------- -------------------- -------- ----------
-----------------
DISTRIBUIDOR
--------------------------------------------------
SINOPSIS
----------------------------------------------------------------
--------------

Cuenta las aventuras de la población subterránea en los túneles del
metro de París


SQL>
SQL> /* Posicionamiento del segundo punto */
SQL> SAVEPOINT FASE_NUMERO2;
Savepoint creado.
```

```
SQL>
SQL> INSERT INTO PELICULAS VALUES (3,'SUBWAY','POLICIACA','DRAMA',
TO_DATE('10/04/1985','DD/MM/YYYY'),1,1,'FILMAX',
'Cuenta las aventuras de la población subterránea en los túneles del
metro de París');

1 fila insertada.

SQL>
SQL> /* Retorno al punto número 2 para no insertar dos veces
la película 'SUBWAY'
*/
SQL> ROLLBACK TO FASE_NUMERO2;

Cancelación (rollback) realizada.

SQL>
```

```
SQL> COMMIT;
Validación realizada
```

miramos en la tabla PELICULAS, sólo hay una fila 'SUBWAY'; la segunda inserción no se ha tenido en cuenta ya que se ha anulado con el ROLLBACK TO FASE_NUMERO2 conservando las actualizaciones anteriores al punto número 2.

Contenido de la tabla PELICULAS:

IDENT_ PELICULA	**TITULO**	**GENERO1**	**GENERO2**	**FECHA_ ESTRENO**	**PAIS**	**IDENT_ DIRECTOR**	**DISTRIBUIDOR**
1	SUBWAY	POLICIACA	DRAMA	10/04/85	1	1	FILMAX
SINOPSIS	Cuenta las aventuras de la población subterránea en los túneles del metro de París.						
2	NIKITA	DRAMA	ROMÁNTICA	21/02/90	1	1	FILMAX
SINOPSIS	Nikita, condenada a cadena perpetua, es obligada a trabajar en secreto para el gobierno como agente de los servicios secretos.						

Carga de datos masiva con SQL*Loader

Después de haber creado los esqueletos de las diferentes tablas, ahora hay que alimentarlas.

Cuando existe un histórico, puede ser interesante cargar rápidamente y de forma masiva todo este histórico.

Antes de nada, habrá que poner este histórico en el formato esperado para que la carga funcione con la herramienta elegida.

Retomemos por ejemplo la tabla PELICULAS que llenamos con múltiples INSERT en el capítulo La manipulación de los datos (LMD) - Ejercicios de aplicación.

TABLA PELICULAS

Consulta de creación de la tabla (sintaxis estándar):

```
CREATE TABLE PELICULAS (IDENT_PELICULAS INTEGER,
                        TITULO          VARCHAR(50),
                        GENERO1         VARCHAR(20),
                        GENERO2         VARCHAR(20),
                        FECHA_ESTRENO   DATE,
                        PAIS            SMALLINT,
                        IDENT_DIRECTOR  INTEGER,
                        DISTRIBUIDOR    VARCHAR(50),
                        SINOPSIS        VARCHAR(2000));
```

Consulta de inserción de filas (sintaxis Oracle):

```
INSERT INTO PELICULAS VALUES
(1,'SUBWAY','POLICIACA','DRAMA',TO_DATE('10/04/1985','DD/MM/YYYY'),
1,1,'FILMAX','Cuenta las aventuras de la población subterránea en los
túneles del metro de París');

INSERT INTO PELICULAS VALUES
(2,'NIKITA','DRAMA','ROMANTICA',TO_DATE('21/02/1990','DD/MM/YYYY'
),1,1,'FILMAX','Nikita, condenada a cadena perpetua, es obligada a
trabajar en secreto para el gobierno como agente de los servicios
secretos.');

INSERT INTO PELICULAS VALUES (3,'STAR WARS 6: EL RETORNO DEL JEDI',
'ACCIÓN','SF',TO_DATE('19/10/1983','DD/MM/YYYY'),2,2,'20th
Century Fox ','El imperio galáctico es más poderoso que nunca: la
construcción de la nueva arma, la Estrella de la Muerte, amenaza todo
el universo.');

INSERT INTO PELICULAS VALUES
(4,'AVATAR','ACCIÓN','SF',TO_DATE('16/10/2009','DD/MM/YYYY'),2,3,'
20th Century Fox ','A pesar de su parálisis, Jake Sully, un viejo marine
inmovilizado en una silla de ruedas, en el fondo sigue siendo un soldado');

INSERT INTO PELICULAS VALUES (5,'BIENVENIDOS AL
NORTE','COMEDIA','',TO_DATE('27/02/2008','DD/MM/YYYY'),1,4,'Lauren
Films',' Philippe Abrams, director de correos de Salon-de-Provence, es
trasladado al Norte.');
```

Para la sintaxis MySQL, basta con cambiar el TO_DATE('10/04/1985','DD/MM/YYYY') por STR_TO_DATE('10/04/1985','%d/%m/%Y').

Podemos imaginar que las películas estaban almacenadas anteriormente en un fichero Excel o simplemente en un fichero de texto.

Antes de poder cargar de una sola vez todas las filas, hay que dar formato a un fichero para que contenga una fila por registro y cada columna separada por una coma o un punto y coma, por ejemplo.

Los campos de texto deben limitarse con una comilla. Las fechas deben tener el mismo formato.

Este sería el resultado:

```
1,'SUBWAY','POLICIACA','DRAMA',10/04/85,1,1,'FILMAX',' Cuenta las
aventuras de la población subterránea en los túneles del metro
de París'
2,'NIKITA','DRAMA','ROMANTICA',21/02/90,1,1,'FILMAX',' Nikita,
condenada a cadena perpetua, es obligada a trabajar en secreto
para el gobierno como agente de los servicios secretos.'
3,'STAR WARS 6: EL RETORNO DEL JEDI','ACCIÓN','SF',19/10/83,2,2,'20th
Century Fox','El imperio galáctico es más poderoso que nunca: la
construcción de la nueva arma, la Estrella de la Muerte, amenaza todo
el universo.'
4,'AVATAR','ACCIÓN','SF',16/10/09,2,3,'20th Century Fox','A pesar de
su parálisis, Jake Sully, un viejo marine inmovilizado en una silla
de ruedas, en el fondo sigue siendo un soldado'
5,'BIENVENIDOS AL NORTE','COMEDIA','',27/02/08,1,4,'Lauren Films',
'Philippe Abrams, director de correos de Salon-de-Provence, es
trasladado al Norte.'
```

Con Oracle, las fechas deben tener el mismo formato en el fichero que en la variable NLS_DATE_FORMAT.

Una vez tenemos el fichero, hay que utilizar la herramienta de carga SQL*Loader que ofrece Oracle.

Para que funcione, hay que suministrar a la herramienta un fichero de control que indique qué formato tienen los datos de entrada. Normalmente a este fichero se le llama <Tabla>.ctl.

Ejemplo de un fichero de control para un fichero con comas como separadores: carga_peliculas.csv

```
    LOAD DATA
    INFILE  'Carga_peliculas.csv'
    BADFILE 'Carga_peliculas.csv.bad'
    DISCARDFILE 'Carga_peliculas.dsc'
    DISCARDMAX 999
    TRUNCATE
    INTO TABLE PELICULAS
    FIELDS TERMINATED BY ','
    (IDENT_PELICULAS,TITULO,GENERO1,GENERO2,
     FECHA_ESTRENO,PAIS,IDENT_DIRECTOR ,DISTRIBUIDOR,SINOPSIS )
```

INFILE: fichero que contiene los datos a cargar.

BADFILE: fichero que contendrá las filas rechazadas.

DISCARDFILE: fichero que contendrá los errores encontrados.

TRUNCATE: vacía la tabla antes de la carga.

DISCARDMAX: número máximo de errores permitidos.

FIELDS TERMINATED BY ',': indica que el separador de campos es la coma.

*Sintaxis de ejecución de SQL*Loader:*

```
sqlldr data=Carga_peliculas.csv control=Carga_peliculas.ctl
                                log=Carga_peliculas.log
                                bad=Carga_peliculas.bad
                                discard=Carga_peliculas.dsc
userid=user/passwd
```

Este comando se ejecuta en la línea de comandos del sistema operativo.

Resultado en el log:

```
SQL*Loader: Release 10.2.0.1.0 - Production on Vie. Junio 24
16:29:01 2014

Copyright (c) 1982, 2005, Oracle.  All rights reserved.

Archivo de control:   Carga_peliculas.ctl
Archivo de datos:     Carga_peliculas.csv
  Archivo BAD:     Carga_peliculas.bad
  Archivo DISCARD: Carga_peliculas.dsc
 (Permitir 999 desechados)

Número a cargar: ALL
Número a ignorar: 0
Errores permitidos: 50
Matriz de enlace:     64 filas, máximo de 256000 bytes
Continuación:    ninguno especificado
Ruta de acceso utilizada:      Convencional

Tabla PELICULAS, cargada desde cada registro físico.
Opción INSERT activa para esta tabla: TRUNCATE

   Nombre columna               Posición   Long Term
Entorno Tipo de dato
------------------------------ ---------- ----- ---- ----
--------------------
IDENT_PELICULAS                     FIRST     *   ,
CARACTER
TITULO                               NEXT     *   ,
CARACTER
GENERO1                              NEXT     *   ,
CARACTER
GENERO2                              NEXT     *   ,
CARACTER
FECHA_ESTRENO                        NEXT     *   ,
CARACTER
PAIS                                 NEXT     *   ,
CARACTER
IDENT_DIRECTOR                       NEXT     *   ,
CARACTER
DISTBIUTEUR                          NEXT     *   ,
CARACTER
SINOPSIS                             NEXT     *   ,
CARACTER


Tabla PELICULAS:
  5 filas se han cargado correctamente.
  0 filas no cargadas debido a errores de datos.
  0 filas no cargadas porque todas las cláusulas WHEN han fallado.
  0 filas no cargadas porque todos los campos eran nulos.
```

SQL*Loader indica « 5 filas se han cargado correctamente. » y no hay ningún registro en los ficheros .log y .bad ya que la carga ha sido correcta.

También es posible cargar un fichero que no tenga separadores, indicando a SQL*Loader las posiciones de inicio y fin de cada columna.

Es necesario que todas las columnas en todos los registros tengan el mismo tamaño; de hecho, estamos indicando la posición de inicio y de fin de cada campo. Los campos CHAR se completarán con espacios en blanco si fuera necesario.

A continuación puede ver algunas filas del fichero de entrada sin separadores:

```
1SUBWAY                          POLICIACADRAMA
10/04/8511FILMAX
```

```
Cuenta las aventuras de la población subterránea en los túneles del
metro de París
2NIKITA                          DRAMA    ROMANTICA
21/02/9011FILMAX
Nikita, condenada a cadena perpetua, es obligada a trabajar en secreto
para el gobierno como agente de los servicios secretos.
3STAR WARS 6: EL RETORNO DEL JEDI ACCIÓN SF
19/10/832220th Century Fox
El imperio galáctico es más poderoso que nunca: la construcción de
la nueva arma, la Estrella de la Muerte, amenaza todo el universo.
4AVATAR                          ACCIÓN    SF
16/10/092320th Century Fox
A pesar de su parálisis, Jake Sully, un viejo marine inmovilizado en
una silla de ruedas, en el fondo sigue siendo un soldado
5BIENVENIDOS AL NORTE            COMEDIA
27/02/0814LAUREN FILMS
Philippe Abrams, director de correos de Salon-de-Provence, es
trasladado al Norte.
```

El fichero de control será diferente de la versión con separadores.

Ejemplo de un fichero de control con posición de inicio y fin de cada columna de la tabla en el fichero de entrada: carga_peliculas_fija.csv

```
     LOAD DATA
     INFILE  'Carga_peliculas_fija.csv'
     BADFILE 'Carga_peliculas_fija.csv.bad'
     DISCARDFILE 'Carga_peliculas_fija.dsc'
     DISCARDMAX 999
     TRUNCATE
     INTO TABLE PELICULAS
     (IDENT_PELICULAS    POSITION (  1 :  1 ) INTEGER EXTERNAL,
      TITULO             POSITION (  2 : 32 ) CHAR,
      GENERO1            POSITION ( 33 : 40 ) CHAR,
      GENERO2            POSITION ( 41 : 50 ) CHAR,
      FECHA_ESTRENO      POSITION ( 51 : 58 )
"TO_DATE(:FECHA_ESTRENO,'DD/MM/YY')",
      PAIS               POSITION ( 59 : 59 ) INTEGER EXTERNAL,
      IDENT_DIRECTOR     POSITION ( 60 : 60 ) INTEGER EXTERNAL,
      DISTRIBUIDOR       POSITION ( 61 : 77 ) CHAR,
      SINOPSIS           POSITION ( 78 : 239) CHAR)

sqlldr data= Carga_peliculas_fija.csv control=Carga_peliculas_fija.ctl
log=Carga_peliculas_fija.log bad=Carga_peliculas_fija.bad
discard=Carga_peliculas_fija.dsc userid=user/passwd
```

Para la fechas es necesario indicar el formato.

Resultado:

```
SQL*Loader: Release 10.2.0.1.0 -

Copyright (c) 1982, 2005, Oracle. All rights reserved.

Archivo de control:   Carga_peliculas.ctl
Archivo de datos:     Carga_peliculas.csv
  Archivo BAD:     Carga_peliculas.bad
  Archivo DISCARD: Carga_peliculas.dsc
 (Permitir 999 desechados)

Número a cargar: ALL
Número a ignorar: 0
Errores permitidos: 50
Matriz de enlace:     64 filas, máximo de 256000 bytes
Continuación:    ninguno especificado
Ruta de acceso utilizada:      Convencional
```

```
Tabla PELICULAS, cargada desde cada registro físico.
Opción INSERT activa para esta tabla: TRUNCATE

  Nombre columna               Posición   Long Term
Entorno Tipo de dato
------------------------------ ---------- ----- ---- ----
--------------------
IDENT_PELICULAS                       1:1     1          CARACTER
TITULO                               2:32    31          CARACTER
GENERO1                             33:40     8          CARACTER
GENERO2                             41:50    10          CARACTER
FECHA_ESTRENO                       51:58     8          CARACTER
  cadena SQL para la columna: "TO_DATE(:FECHA_ESTRENO,'DD/MM/YY')"
PAIS                                59:59     1          CARACTER
IDENT_DIRECTOR                      60:60     1          CARACTER
DISTRIBUIDOR                        61:77    17          CARACTER
SINOPSIS                           78:239   162          CARACTER


Tabla PELICULAS :
  5 filas se han cargado correctamente.
  0 filas no cargadas debido a errores de datos.
  0 filas no cargadas porque todas las cláusulas WHEN han fallado.
  0 filas no cargadas porque todos los campos eran nulos.


Espacio asignado a la matriz de enlace:        16768 bytes(64
filas)
Bytes de buffer de lectura: 1048576

Nombre total de registros lógicos ignorados:          0
Nombre total de registros lógicos leídos:             5
Nombre total de registros lógicos rechazados:         0
Nombre total de registros lógicos desechados:         0


Tiempo transcurrido:     00:00:06.00
Tiempo de CPU:           00:00:00.14
```

La importación y exportación de tablas con Oracle

Los SGBDR ofrecen en general un mecanismo de exportación e importación de tablas o de toda la base de datos. Estos procedimientos se utilizan principalmente para hacer copias de seguridad o transferir las bases de datos de un servidor a otro.

El formato de los ficheros extraídos normalmente no lo puede utilizar ninguna otra aplicación, sólo Oracle. Es un formato propietario de la base de datos. Hay que evitar comprimir estos archivos ya que se pueden corromper.

1. La exportación de tablas

En Oracle, se utilizará el ejecutable « exp » y en MySQL « mysqldump ». Estas herramientas se proporcionan al instalar el SGBDR.

Sintaxis Oracle para exportar la tabla PELICULAS:

```
exp <usuario/contraseña> file=PELICULAS.dmp TABLES=PELICULAS LOG=PELICULAS.log
```

Este comando se ejecuta en la línea de comandos del sistema operativo.

Informe de ejecución que indica que se han exportado cinco filas de la tabla PELICULAS:

```
Export: Release 10.2.0.1.0 - Production on Vie. Jul. 1 15:26:15
2014

Copyright (c) 1982, 2005, Oracle.  All rights reserved.


Conectado a: Oracle Database 10g Express Edition Release
10.2.0.1.0 - Production

Exportación hecha con el juego de caracteres WE8MSWIN1252 y el juego de
caracteres NCHARAL16UTF16

Preparado para exportar las tablas especificadas ...  a través de la
ruta convencional...
. . exportación de la tabla PELICULAS         5 filas exportadas      <
Proceso de exportación terminado con éxito.
```

También se pueden añadir condiciones a la exportación. Por ejemplo, si sólo queremos las películas cuyo identificador sea superior a 3, añadiremos la cláusula QUERY:

```
exp <usuario/contraseña> file=PELICULAS.dmp TABLES=PELICULAS
LOG=PELICULAS.log
query="'where IDENT_PELICULAS > 3'
```

La exportación sólo ha extraído dos filas:

```
Preparado para exportar las tablas especificadas ... a través de
la ruta directa ...
. . exportación de la tabla PELICULAS         2 filas exportadas <
```

También es posible exportar toda la base de datos añadiendo la opción FULL=Y. Esta opción está reservada al administrador de la base de datos.

```
exp <usuario/contraseña> file=TODALABASE.dmp LOG=TODALABASE.log full=y
```

Por último, es posible poner todos los parámetros de la exportación en un fichero de parámetros de esta forma:

```
exp parfile=param_carga_peliculas.prm
```

Consulte la documentación de la base de datos para ver todas las posibilidades que ofrece el export.

Sintaxis MySQL para exportar la tabla PELICULAS:

```
mysqldump -h<nombre o dirección de máquina, por defecto localhost>
-u<usuario> -p<contraseña> -rPELICULAS.dmp <nombre base> PELICULAS
```

-r: nombre del fichero resultado seguido del nombre de la base y del nombre de la(s) tabla(s) exportada(s).

Al contrario que en Oracle, el fichero de la exportación MySQL es legible. Encontraremos los comandos de creación de las tablas así como los Insert de las filas.

Fichero de exportación MySQL:

```
-- MySQL dump 10.13  Distrib 5.1.54, for Win32 (ia32)
--
-- Host: localhost    Database: TEST
-- ------------------------------------------------------
-- Server version       5.1.54-community

/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8 */;
/*!40103 SET @OLD_TIME_ZONE=@@TIME_ZONE */;
/*!40103 SET TIME_ZONE='+00:00' */;
/*!40014 SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0 */;
/*!40014 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS,
FOREIGN_KEY_CHECKS=0 */;
/*!40101 SET @OLD_SQL_MODE=@@SQL_MODE,
SQL_MODE='NO_AUTO_VALUE_ON_ZERO' */;
/*!40111 SET @OLD_SQL_NOTES=@@SQL_NOTES, SQL_NOTES=0 */;

--
-- Table structure for table `peliculas`
--

DROP TABLE IF EXISTS `peliculas`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `peliculas` (
  `IDENT_PELICULAS` int(11) DEFAULT NULL,
  `TITULO` varchar(50) DEFAULT NULL,
  `GENERO1` varchar(20) DEFAULT NULL,
  `GENERO2` varchar(20) DEFAULT NULL,
  `FECHA_ESTRENO` date DEFAULT NULL,
  `PAIS` smallint(6) DEFAULT NULL,
  `IDENT_DIRECTOR` int(11) DEFAULT NULL,
  `DISTRIBUIDOR` varchar(50) DEFAULT NULL,
  `SINOPSIS` varchar(2000) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `peliculas`
--

LOCK TABLES `peliculas` WRITE;
/*!40000 ALTER TABLE `film` DISABLE KEYS */;
INSERT INTO `peliculas` VALUES (1,'SUBWAY','POLICIACA','DRAMA',
'1985-04- 10',1,1,'FILMAX','Cuenta las aventuras de la población
```

```
subterránea en los túneles del metro de París'),
(2,'NIKITA','DRAMA','ROMANTICA','1990-02-
21',1,1,'FILMAX','Nikita, condenada a cadena perpetua, es obligada a
trabajar en secreto para el gobierno como agente de los servicios
secretos.'),(3,'STAR WARS 6: EL RETORNO DEL JEDI','ACCIÓN','SF',
'1983-10-19',2,2,'20th Century Fox','El imperio galáctico es más
poderoso que nunca: la construcción de la nueva arma, la Estrella
de la Muerte, amenaza todo el universo.'),(4,'AVATAR','ACCIÓN','SF',
'2009-10-16',2,3,'20th Century Fox ','A pesar de su parálisis,
Jake Sully, un viejo marine inmovilizado en una silla de ruedas, en el
fondo sigue siendo un soldado'),(5,'BIENVENIDO AL NORTE','COMEDIA','',
'2008-02-27',1,4,'LAUREN FILMS', 'Philippe Abrams, director de correos
de Salon-de-Provence, es trasladado al Norte.'),(6,'JUNGLA DE CRISTAL
4','ACCIÓN','','2007-07-04',2,5,'Twentieth Century Fox Film
Corporation','En su cuarta aventura, el inspector John McClane se
enfrenta a un nuevo tipo de terrorismo.
La red informática nacional que controla todas las comunicaciones,
los transportes y la energía de los Estados Unidos, es saboteada
de manera sistemática, sumiendo al país en el caos.');
/*!40000 ALTER TABLE `peliculas` ENABLE KEYS */;
UNLOCK TABLES;
/*!40103 SET TIME_ZONE=@OLD_TIME_ZONE */;

/*!40101 SET SQL_MODE=@OLD_SQL_MODE */;
/*!40014 SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS */;
/*!40014 SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS */;
/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
/*!40111 SET SQL_NOTES=@OLD_SQL_NOTES */;

-- Dump completed on 2011-07-01 16:32:10
```

En MySQL también es posible extraer toda la base de datos sin indicar ningún nombre de tabla.

```
mysqldump -h<nombre o dirección de máquina, por defecto localhost>
-u<usuario> -p<contraseña> <nombre base de datos> >TODALABASE.dmp
```

2. La importación de tablas

Una vez se ha exportado la base de datos, hay que poder importar los resultados en otra base de datos o en la misma. Para ello, se utilizarán los mecanismos de importación.

En Oracle, se utilizará el ejecutable « imp» y en MySQL el comando « mysql ».

Ejemplo de una importación en una base de datos Oracle:

```
imp <usuario/contraseña > file=PELICULAS.dmp TABLES=PELICULAS
LOG=IMPORTPELICULAS.log FROMUSER=<usuario> TOUSER=<usuario>
```

Este comando se ejecuta sobre la línea de comandos del sistema operativo.

```
Import: Release 10.2.0.1.0 - Production on Vie. Jul. 1 17:01:05
2014

Copyright (c) 1982, 2005, Oracle. All rights reserved.


Conectado a: Oracle Database 10g Express Edition Release
10.2.0.1.0 - Production


Fichero de exportación creado por EXPORT:V10.02.01 a través de
la ruta convencional
importación realizada con el juego de caracteres WE8MSWIN1252 y
```

```
el juego
NCHAR AL16UTF16

. Importación de objetos <USUARIO> en <USUARIO>
. . Importación de la tabla                          "PELICULAS"
5 filas importadas       <
```

Ejemplo de importación para una base de datos MySQL:

```
mysql -hlocalhost -u <usuario> -p<contraseña> <Nombre base de datos>
< PELICULAS.dmp
```

Al contrario que en Oracle, no es necesario borrar primero las tablas. En la exportación hemos visto que el comando DROP está incluido en el script de carga.

```
DROP TABLE IF EXISTS `peliculas`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `peliculas` (
  `IDENT_PELICULAS` int(11) DEFAULT NULL,
...
```

Los sub-SELECT

Podemos anidar una consulta dentro de otra, incluyéndola después de la cláusula WHERE o sustituyendo una constante del comando IN o EXISTS, por ejemplo.

En función de lo que devuelva el sub-SELECT, no se podrá posicionar en cualquier sitio.

Si el resultado de la consulta del sub-SELECT sólo devuelve una fila, podremos utilizar la subconsulta en lugar de cualquier constante.

Por ejemplo, si se quieren recuperar todas las películas que tengan en su casting al menos un actor francés, hay que hacer un sub-SELECT que recupere el identificador de la tabla PAIS que corresponda al PAIS 'FRANCIA'. A continuación comprobaremos que la columna NACIONALIDAD de la tabla ACTOR corresponde al valor del sub-SELECT.

Antes de probar la consulta completa, es preferible probar la subconsulta para comprobar que es correcta y que devuelve una sola fila.

Por ejemplo:

```
SELECT PAIS.IDENT_PAIS FROM PAIS WHERE
PAIS.DESCRIPCION = 'FRANCIA';
```

muestra:

```
IDENT_PAIS
----------
         1
```

A continuación se puede incluir la subconsulta en la consulta principal del siguiente modo:

```
SELECT SUBSTR(PELICULAS.TITULO,1,20) TITULO, PELICULAS.FECHA_ESTRENO,
       SUBSTR((DIRECTOR.APELLIDO||' '||DIRECTOR.NOMBRE),1,25)
 DIRECTOR,
       SUBSTR(RTRIM(ACTOR.APELLIDO||' '||ACTOR.NOMBRE),1,25) ACTOR,
       ACTOR.FECHA_NACIMIENTO NACIDO,ACTOR.NUM_PELICULAS NUM_PELICULAS,
       ESTA.PRESUPUESTO,ESTA.NUM_ENTRADAS_ESPANA NUM_ENTRADAS
FROM   PELICULAS PELICULAS, DIRECTOR DIRECTOR, CASTING CAST,
       ACTOR ACTOR, ESTADISTICA ESTA
WHERE
   PELICULAS.IDENT_DIRECTOR  = DIRECTOR.IDENT_DIRECTOR AND
   PELICULAS.IDENT_PELICULAS = CAST.IDENT_PELICULAS AND
   PELICULAS.IDENT_PELICULAS = ESTA.IDENT_PELICULAS AND
   CAST.IDENT_ACTOR          = ACTOR.IDENT_ACTOR AND
   ACTOR.NACIONALIDAD        = (SELECT PAIS.IDENT_PAIS FROM PAIS WHERE
                           PAIS.DESCRIPCION = 'FRANCIA')
ORDER BY PELICULAS.TITULO, ACTOR.APELLIDO;
```

Resultados:

```
TITULO               FECHA_ES  DIRECTOR               ACTOR
  NACIDO       NUM_PELICULAS       PRESUPUESTO  NUM_ENTRADAS
-------------------- -------- -------------------------
------------------------ -------- ---------- ----------
BIENVENIDOS AL NORTE 27/02/08 BOON DANY              BOON DANY
  26/06/66         23         11   21000000
NIKITA                 21/02/90 BESSON LUC             PARILLAUD ANNE
  06/05/60         35        7,6    3787845
NIKITA                 21/02/90 BESSON LUC             RENO JEAN
  30/06/48         75        7,6    3787845
SUBWAY                 10/04/85 BESSON LUC             ADJANI ISABELLE
  27/06/55         42        2,6    2917562
SUBWAY                 10/04/85 BESSON LUC             BOHRINGER RICHARD
```

```
  16/06/42        132        2,6    2917562
SUBWAY               10/04/85 BESSON LUC             GALABRU MICHEL
  27/10/22        277        2,6    2917562
SUBWAY               10/04/85 BESSON LUC              LAMBERT CHRISTOPHE
  29/03/57         64        2,6    2917562
SUBWAY               10/04/85 BESSON LUC             RENO JEAN
  30/06/48         75        2,6    2917562

8 fila(s) seleccionadas(s).
```

También es posible incluir una subconsulta directamente en el SELECT principal siempre que devuelva una sola fila.

Para recuperar la descripción del PAIS sin hacer una unión, podemos escribir la consulta del siguiente modo.

```
SELECT SUBSTR(PELICULAS.TITULO,1,20) TITULO, PELICULAS.FECHA_ESTRENO,
       SUBSTR((DIRECTOR.APELLIDO||' '||DIRECTOR.NOMBRE),1,25) DIRECTOR,
       SUBSTR(RTRIM(ACTOR.APELLIDO||' '||ACTOR.NOMBRE),1,25) ACTOR,
       ACTOR.FECHA_NACIMIENTO NACIDO,ACTOR.NUM_PELICULAS NUM_PELICULAS,
       ESTA.PRESUPUESTO,ESTA.NUM_ENTRADAS_ESPANA NUM_ENTRADAS,
       (SELECT PAIS.DESCRIPCION FROM PAIS WHERE PAIS.IDENT_PAIS =
ACTOR.NACIONALIDAD) PAIS
FROM   PELICULAS PELICULAS, DIRECTOR DIRECTOR, CASTING CAST,
       ACTOR ACTOR, ESTADISTICA ESTA
WHERE
   PELICULAS.IDENT_DIRECTOR  = DIRECTOR.IDENT_DIRECTOR AND
   PELICULAS.IDENT_PELICULAS = CAST.IDENT_PELICULAS AND
   PELICULAS.IDENT_PELICULAS = ESTA.IDENT_PELICULAS AND
   CAST.IDENT_ACTOR          = ACTOR.IDENT_ACTOR
ORDER BY PELICULAS.TITULO, ACTOR.APELLIDO;
```

El vínculo entre la subconsulta y la consulta principal se hace con la cláusula WHERE que utiliza una columna de la consulta principal: ACTOR.NACIONALIDAD.

Resultados:

```
TITULO               FECHA_ES DIRECTOR             ACTOR
  NACIDO     NUM_PELICULAS     PRESUPUESTO   NUM_ENTRADAS PAIS
-------------------- -------- ------------------------
------------------------ -------- ---------- ----------
AVATAR               16/10/09 CAMERON JAMES          SALDANA ZOE
  19/06/78         31        237   12018251 USA
AVATAR               16/10/09 CAMERON JAMES          WEAVER SIGOURNEY
  08/10/49         66        237   12018251 USA
BIENVENIDOS AL NORTE 27/02/08 BOON DANY              BOON DANY
  26/06/66         23         11   21000000 FRANCIA
BIENVENIDOS AL NORTE 27/02/08 BOON DANY              MERAD KAD
  27/03/64         55         11   21000000 ARGELIA
NIKITA               21/02/90 BESSON LUC             PARILLAUD ANNE
  06/05/60         35        7,6    3787845 FRANCIA
NIKITA               21/02/90 BESSON LUC             RENO JEAN
  30/06/48         75        7,6    3787845 FRANCIA
STAR WARS 6: EL RETO 19/10/83 LUCAS GEORGES          FISHER CARRIE
  21/10/56         74         32    4263000 USA
STAR WARS 6: EL RETO 19/10/83 LUCAS GEORGES          FORD HARRISON
  13/06/42         64         32    4263000 USA
SUBWAY               10/04/85 BESSON LUC             ADJANI ISABELLE
  27/06/55         42        2,6    2917562 FRANCIA
SUBWAY               10/04/85 BESSON LUC             BOHRINGER RICHARD
  16/06/42        132        2,6    2917562 FRANCIA
SUBWAY               10/04/85 BESSON LUC             GALABRU MICHEL
  27/10/22        277        2,6    2917562 FRANCIA
SUBWAY               10/04/85 BESSON LUC             LAMBERT CHRISTOPHE
  29/03/57         64        2,6    2917562 FRANCIA
SUBWAY               10/04/85 BESSON LUC             RENO JEAN
  30/06/48         75        2,6    2917562 FRANCIA
```

```
13 fila(s) seleccionada(s).
```

Si ahora la subconsulta devuelve varias ocurrencias, se puede utilizar en una cláusula IN. Si se buscan los actores americanos o argelinos, escribiremos la siguiente consulta.

Ejemplo:

```
SELECT SUBSTR(PELICULAS.TITULO,1,20) TITULO, PELICULAS.FECHA_ESTRENO,
       SUBSTR((DIRECTOR.APELLIDO||' '||DIRECTOR.NOMBRE),1,25) DIRECTOR,
       SUBSTR(RTRIM(ACTOR.APELLIDO||' '||ACTOR.NOMBRE),1,25) ACTOR,
       ACTOR.FECHA_NACIMIENTO NACIDO,ACTOR.NUM_PELICULAS NUM_PELICULAS,
       ESTA.PRESUPUESTO,ESTA.NUM_ENTRADAS_ESPANA NUM_ENTRADAS
FROM   PELICULAS PELICULAS, DIRECTOR DIRECTOR, CASTING CAST,
       ACTOR ACTOR, ESTADISTICA ESTA
WHERE
   PELICULAS.IDENT_DIRECTOR  = DIRECTOR.IDENT_DIRECTOR AND
   PELICULAS.IDENT_PELICULAS = CAST.IDENT_PELICULAS AND
   PELICULAS.IDENT_PELICULAS = ESTA.IDENT_PELICULAS AND
   CAST.IDENT_ACTOR          = ACTOR.IDENT_ACTOR AND
   ACTOR.NACIONALIDAD IN (SELECT PAIS.IDENT_PAIS FROM PAIS WHERE
                          PAIS.DESCRIPCION IN ('USA', 'ARGELIA'))
ORDER BY PELICULAS.TITULO, ACTOR.APELLIDO;
```

Resultados:

```
TITULO                FECHA_ES DIRECTOR              ACTOR
  NACIDO       NUM_PELICULAS     PRESUPUESTO   NUM_ENTRADAS
-------------------- -------- -------------------------
------------------------- -------- ---------- ----------
AVATAR               16/10/09 CAMERON JAMES           SALDANA ZOE
  19/06/78         31        237   12018251
AVATAR               16/10/09 CAMERON JAMES             WEAVER SIGOURNEY
  08/10/49         66        237   12018251
BIENVENIDOS AL NORTE 27/02/08 BOON DANY               MERAD KAD
  27/03/64         55         11   21000000
STAR WARS 6: EL RETO 19/10/83 LUCAS GEORGES           FISHER CARRIE
  21/10/56         74         32    4263000
STAR WARS 6: EL RETO 19/10/83 LUCAS GEORGES           FORD HARRISON
  13/06/42         64         32    4263000
```

Las subconsultas también se pueden utilizar para comprobar la existencia de un valor en una tabla. Es el mismo tipo de búsqueda que el IN, pero a veces un poco más rápido en su ejecución.

La misma consulta anterior pero con la cláusula EXISTS en lugar de la cláusula IN devuelve el mismo resultado.

```
SELECT SUBSTR(PELICULAS.TITULO,1,20) TITULO, PELICULAS.FECHA_ESTRENO,
       SUBSTR((DIRECTOR.APELLIDO||' '||DIRECTOR.NOMBRE),1,25) DIRECTOR,
       SUBSTR(RTRIM(ACTOR.APELLIDO||' '||ACTOR.NOMBRE),1,25) ACTOR,
       ACTOR.FECHA_NACIMIENTO NACIDO,ACTOR.NUM_PELICULAS NUM_PELICULAS,
       ESTA.PRESUPUESTO,ESTA.NUM_ENTRADAS_ESPANA NUM_ENTRADAS
FROM   PELICULAS PELICULAS, DIRECTOR DIRECTOR, CASTING CAST,
       ACTOR ACTOR, ESTADISTICA ESTA
WHERE
   PELICULAS.IDENT_DIRECTOR  = DIRECTOR.IDENT_DIRECTOR AND
   PELICULAS.IDENT_PELICULAS = CAST.IDENT_PELICULAS AND
   PELICULAS.IDENT_PELICULAS = ESTA.IDENT_PELICULAS AND
   CAST.IDENT_ACTOR          = ACTOR.IDENT_ACTOR AND
   EXISTS (SELECT PAIS.IDENT_PAIS FROM PAIS WHERE
                  PAIS.DESCRIPCION IN ('USA', 'ARGELIA') AND
                  PAIS.IDENT_PAIS = ACTOR.NACIONALIDAD )
ORDER BY PELICULAS.TITULO, ACTOR.APELLIDO;
```

Otra posibilidad, las subconsultas con la misma tabla que la tabla principal.

Por ejemplo, si se quieren mostrar los actores que tienen el mismo nombre que otro actor.

```
SELECT SUBSTR(PELICULAS.TITULO,1,20) TITULO, PELICULAS.FECHA_ESTRENO,
       SUBSTR((DIRECTOR.APELLIDO||' '||DIRECTOR.NOMBRE),1,25) DIRECTOR,
       SUBSTR(RTRIM(ACT1.APELLIDO||' '||ACT1.NOMBRE),1,25) ACTOR,
       ACT1.FECHA_NACIMIENTO NACIDO,ACT1.NUM_PELICULAS NUM_PELICULAS,
       ESTA.PRESUPUESTO,ESTA.NUM_ENTRADAS_ESPANA NUM_ENTRADAS
FROM   PELICULAS PELICULAS, DIRECTOR DIRECTOR, CASTING CAST,
       ACTOR ACT1, ESTADISTICA ESTA
WHERE
   PELICULAS.IDENT_DIRECTOR  = DIRECTOR.IDENT_DIRECTOR AND
   PELICULAS.IDENT_PELICULAS = CAST.IDENT_PELICULAS AND
   PELICULAS.IDENT_PELICULAS = ESTA.IDENT_PELICULAS AND
   CAST.IDENT_ACTOR          = ACT1.IDENT_ACTOR AND
   EXISTS (SELECT * FROM ACTOR ACT2 WHERE ACT2.NOMBRE =
ACT1.NOMBRE
AND ACT2.APELLIDO <> ACT1.APELLIDO) ORDER BY PELICULAS.TITULO, ACT1.APELLIDO;
```

El resultado es « No se ha seleccionado ninguna fila » ya que en la tabla ACTOR no hay dos actores que tengan el mismo nombre.

Si ahora añadimos una fila en la tabla ACTOR con « Isabelle Huppert »:

```
INSERT INTO ACTOR VALUES
(13,'HUPPERT','ISABELLE',TO_DATE('16/03/1953','DD/MM/YYYY'),102,1);
```

Ahora hay dos « Isabelle » en la tabla, y el resultado de la consulta anterior es:

```
TITULO               FECHA_ES DIRECTOR              ACTOR
  NACIDO      NUM_PELICULAS     PRESUPUESTO   NUM_ENTRADAS
-------------------- -------- -------------------------
------------------------
- -------- ---------- ---------- ----------
SUBWAY               10/04/85 BESSON LUC               ADJANI
ISABELLE
  27/06/55         42        2,6    2917562
```

Isabelle Huppert no aparece ya que no está en la tabla CASTING.

Algunos conceptos de rendimiento

En la utilización de una base de datos, a menudo se encuentran problemas de tiempos de respuesta grandes en una consulta u otra.

Las razones son múltiples, se puede tratar de una consulta que no utiliza ningún índice, de una tabla muy grande, de uniones múltiples, de problemas de acceso a disco o de capacidad de memoria, etc.

Lo que se denomina el « tuning » de una base de datos es muy complejo y necesita mucha experiencia y conocimientos de bases de datos y sistemas operativos.

Las reglas básicas cuando se escribe una consulta son:

- Comprobar que los criterios de búsqueda (WHERE) utilizan los índices.
- Comprobar que las uniones entre tablas se hacen sobre las claves de las tablas y que los índices son correctos en estas tablas.
- Comprobar que la selección no devuelve millones de filas.
- Comprobar que las estadísticas de la base de datos están activadas y actualizadas regularmente (sobre todo en Oracle).
- No utilizar muchas funciones en un mismo SELECT.


Las estadísticas son datos que sirven a la base de datos para saber qué camino es el más adecuado para obtener un dato.

1. Utilización de EXPLAIN PLAN

Existe un medio para saber el camino que utiliza el SGBDR para acceder a un elemento. Hay que utilizar el comando EXPLAIN PLAN que analiza el comando e indica el camino elegido. Para ello, se almacenan los elementos en una tabla: PLAN_table en Oracle.

La sintaxis a utilizar es esta:

```
EXPLAIN PLAN SET STATEMENT_ID='<identificador>' INTO PLAN_TABLE FOR
SELECT ... ... ;
```

Se indica al SGBDR que almacene en una tabla llamada « PLAN_TABLE » bajo el identificador elegido (STATEMENT_ID) los análisis realizados sobre la consulta que se indica después del SELECT.

Ejemplo con un SELECT sobre tres tablas, el identificador es 'PRUEBA-REND'

```
DELETE FROM PLAN_TABLE WHERE STATEMENT_ID='PRUEBA-REND';

EXPLAIN PLAN SET STATEMENT_ID='PRUEBA-REND' INTO PLAN_TABLE FOR
SELECT TL.FECHA_COMPRA,
       TL.TIPO,
       TT.DESC_TIPO,
       TL.MARCA,
       MT.DESC_MARCA,
       TL.COLOR
FROM  TELEFONO TL,
      TIPO_TEL TT,
      MARCA_TEL MT
WHERE TL.TIPO  = TT.TIPO
  AND TL.MARCA = MT.MARCA
  AND TL.COLOR IN ('ROJO', 'BLANCO');
```

Se comienza eliminando de la tabla las filas que puedan existir con el identificador 'PRUEBA-REND'.

Para mostrar a continuación el resultado del análisis realizado por el SGBDR, se debe realizar un SELECT de la tabla PLAN_TABLE.

Ejemplo de selección en la tabla PLAN_TABLE:

```
SET LINES 132
SET PAGES 50
SET LONG 500
COL PLAN FOR A45
COL OTRO FOR A85 WRAP
SELECT OBJECT_NAME,OPERATION||' '||OPTIONS||' '||OBJECT_NAME
        ||' '||DECODE(ID,0,'COST = '||POSITION) PLAN
       FROM PLAN_TABLE WHERE STATEMENT_ID = 'PRUEBA-REND';
```

El resultado es como este:

```
OBJECT_NAME                    PLAN
---------------------------------------------------------------
                               SELECT STATEMENT   Cost = 7
                               HASH JOIN
                               MERGE JOIN CARTESIAN
TIPO_TEL                       TABLE ACCESS FULL TIPO_TEL
                               BUFFER SORT
MARCA_TEL                      TABLE ACCESS FULL MARCA_TEL
TELEFONO                       TABLE ACCESS FULL TELEFONO
```

En este caso, lo que es importante controlar, es sobre todo las tablas con ACCESS FULL; ello significa que el SGBDR va a recorrer toda la tabla en la consulta.

En el ejemplo se puede ver que las tres tablas tienen ACCESS FULL, lo que puede provocar que la consulta sea más lenta en las tablas que tengan muchas filas. ACCESS FULL indica que el SGBDR no utiliza los índices para acceder a los datos.

En el caso de tablas pequeñas, esto no es un problema, pero cuando una de las tablas tiene un gran número de filas, el SGBDR debe utilizar un índice.

También podemos ver el dato llamado « Cost », que nos da una indicación del « coste » de la consulta. Este « coste » lo calcula Oracle mediante varios parámetros, sobretodo el número de lecturas E/S, el tiempo necesario para la lectura de un bloque, el número de ciclos de CPU por segundo, etc.

La unidad no es importante, lo importante es comparar el coste a medida que optimizamos una consulta o que añadimos índices. Esto permite comprobar que se está en el buen camino en términos de optimización del rendimiento.

Si retomamos el ejemplo anterior y añadimos índices a las tablas, y particularmente en la columna COLOR de la tabla TELEFONO.

Para mostrar la estructura, los índices y las restricciones de una tabla con SQL*Plus, se puede utilizar este tipo de script:

```
  SET FLU OFF
  SET PAGESIZE 255
  SET WRAP OFF
  SET VER OFF
  SET ARRAYSIZE 1
  COL TIPO FORMAT A15
  COL NOT_NULL FORMAT A8
  PROMPT .
  PROMPT .-------------------------.
  PROMPT ESTRUCTURA DE LA TABLA TELEFONO :
  PROMPT .---------------------------.
  PROMPT .
  SELECT UPPER(A.COLUMN_NAME) NOM,A.DATA_TIPO || '(' ||
         DECODE(A.DATA_TIPO,'NUMBER',A.DATA_PRECISION,
         A.DATA_LENGTH) || ')' TIPO,
```

```
        DECODE(A.NULLABLE,'N','   N','') NOT_NULL,
        B.COMMENTS COMENTARIO
  FROM USER_TAB_COLUMNS A, USER_COL_COMMENTS B
  WHERE A.TABLE_NAME = 'TELEFONO'
  AND   B.TABLE_NAME = 'TELEFONO'
  AND   A.TABLE_NAME = B.TABLE_NAME
  AND   A.COLUMN_NAME = B.COLUMN_NAME
  ORDER BY A.COLUMN_ID;
  PROMPT .
  PROMPT .------------------------.
  PROMPT INDICES DE LA TABLA TELEFONO:
  PROMPT .---------------------------.
  PROMPT .
  SELECT INDEX_NAME "INDEX",
        LOWER(COLUMN_NAME) "COLUMNA(S)"
  FROM USER_IND_COLUMNS
  WHERE TABLE_NAME = UPPER('TELEFONO')
  ORDER BY INDEX_NAME, COLUMN_POSITION;
  PROMPT .
  PROMPT .------------------------.
  PROMPT CLAVES FORANEAS QUE REFERENCIA LA TABLA 'TELEFONO':
  PROMPT .------------------------.
  PROMPT .
  SELECT B.CONSTRAINT_NAME "CLAVES FORANEAS",
        LOWER(B.TABLE_NAME) "TABLE"
  FROM USER_CONSTRAINTS A, USER_CONSTRAINTS B
  WHERE A.TABLE_NAME = 'TELEFONO'
  AND A.CONSTRAINT_NAME = B.R_CONSTRAINT_NAME;
```

Creación de un índice en la tabla TELEFONO

```
CREATE INDEX I_COLOR ON TELEFONO (COLOR);
```

Volvemos a ejecutar el análisis del comando. Este es el nuevo resultado:

```
OBJECT_NAME                    PLAN
---------------------------------------------------------------
                               SELECT STATEMENT   COST = 5
                               NESTED LOOPS
                               NESTED LOOPS
MARCA_TEL                      TABLE ACCESS FULL MARCA_TEL
                               INLIST ITERATOR
TELEFONO                       TABLE ACCESS BY INDEX ROWID
TELEFONO
I_COLOR                        INDEX RANGE SCAN I_COLOR
TIPO_TEL                       TABLE ACCESS BY INDEX ROWID
TIPO_TEL
PK_TELEFONO                    INDEX UNIQUE SCAN PK_TELEFONO
```

Se puede comprobar rápidamente que en la tabla TELEFONO, se utiliza el nuevo INDICE (I_COLOR) donde se indica « TABLE ACCESS BY INDEX ROWID ».

También accede a la tabla TIPO_TEL con el índice ROWID y recorre la PRIMARY KEY de la tabla TELEFONO. (INDEX UNIQUE SCAN PK_TELEFONO).

Respecto al primer análisis, se ve rápidamente una mejora notable de los accesos a los datos, que se traduce en una respuesta más rápida de la consulta.

El « coste » ha bajado a 5 indicando que la optimización ha sido correcta. Como el número de filas por tabla es pequeño, es normal que la diferencia entre antes y después de la optimización sea pequeña. Si hubiéramos tenido tablas con miles de filas, la diferencia hubiera sido mucho más grande.

La optimización de una consulta debe llevarse a cabo paso a paso observando después de cada modificación el impacto en los caminos elegidos y el coste. De hecho, algunas acciones pueden ser contradictorias en términos de rendimiento y hacer que aumente el tiempo de respuesta.

2. Utilización del paquete DBMS_XPLAN.DISPLAY

También se puede mostrar el resultado de un análisis utilizando un paquete proporcionado por Oracle.

Este paquete ofrece la misma información que EXPLAIN PLAN en otro formato.

```
SELECT * FROM TABLA(DBMS_XPLAN.DISPLAY);
```

Esto devuelve un resultado con el siguiente formato:

```
PLAN_TABLE_OUTPUT
------------------------------------------------------------------
------------------------------------------------
Plan hash value: 257614804
------------------------------------------------------------------------
---------------
| Id  | Operation        Name        | Rows | Bytes | Cost (%CPU)| Time |
------------------------------------------------------------------------
|   0 | SELECT STATEMENT|             |    2 |    80 |    5(0)| 00:00:01 |
|   1 |  NESTED LOOPS   |             |    2 |    80 |    5(0)| 00:00:01 |
|   2 |   NESTED LOOPS  |             |    2 |    58 |    4(0)| 00:00:01 |
|   3 |   TABLE ACCESS  | MARCA_TEL   |    5 |    55 |    2(0)| 00:00:01 |
            FULL
|   4 |    INLIST       |             |      |       |        |          |
             ITERATOR
|*  5 |     TABLE ACCESS| TELEFONO    |    1 |    18 |    2(0)| 00:00:01 |
             BY INDEX ROWID
|*  6 |      INDEX RANGE| I_COLOR     |    2 |       |    1(0)| 00:00:01 |
                 SCAN
|   7 |  TABLE ACCESS   | TIPO_TEL    |    1 |    11 |    1(0)| 00:00:01 |
          BY INDEX ROWID
|*  8 |   INDEX UNIQUE  | PK_TELEFONO |    1 |       |    0(0)| 00:00:01 |
                  SCAN
------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   5 - filter("TL"."MARCA"="MT"."MARCA")
   6 - access("TL"."COLOR"='BLANCO' OR "TL"."COLOR"='ROJO')
   8 - access("TL"."TIPO"="TT"."TIPO")
```

En este formato se puede comprobar el « coste » por acceso. De hecho, la columna « cost » indica el coste por cada acción.

3. Optimización de las consultas mediante la utilización de HINTS

En la mayoría de los casos, Oracle utiliza el índice correcto en el acceso a los datos.

De todos modos, en algunos casos, si se quiere « forzar » la utilización de un índice en concreto, es posible indicarlo en la consulta.

Para ello, hay que utilizar comandos llamados HINTS que se insertan en la consulta y que indican a Oracle que utilice tal o cual índice o un método de acceso particular.

En este ejemplo, buscamos los teléfonos de tipo « SP »:

```
SELECT FECHA_COMPRA,
       TIPO,
       DESC_TIPO,
       MARCA,
       COLOR
FROM  TELEFONO
WHERE TIPO = 'SP';
```

Se crea un índice para el TIPO.

```
CREATE INDEX I_TIPO ON TELEFONO (TIPO);
```

El plan de acceso inicial es:

```
OBJECT_NAME   PLAN
------------- -------------------------------------------
              SELECT STATEMENT   COST = 2
TELEFONO      TABLE ACCESS FULL TELEFONO
```

Existe un índice sobre la columna TIPO pero Oracle no lo utiliza. Incluso con una restricción sobre TIPO, Oracle puede realizar un ACCESS FULL si estima que el número de filas es insuficiente, con lo que el acceso completo de la tabla será más rápido.

Ahora, si se quiere utilizar el índice I_TIPO, se puede escribir la consulta de este modo:

```
SELECT /*+ INDEX(TELEFONO I_TIPO) */
       FECHA_COMPRA,
       TIPO,
       MARCA,
       COLOR
FROM  TELEFONO WHERE TIPO =’SP’;
```

El plan es ahora:

```
OBJECT_NAME   PLAN
------------- -------------------------------------------
              SELECT STATEMENT COST = 2
TELEFONO      TABLE ACCESS BY INDEX ROWID TELEFONO
I_TIPO        INDEX RANGE SCAN I_TIPO
```

En algunos casos, puede ser conveniente realizar un ACCESS FULL aunque Oracle utilice un índice.

En este caso escribiremos:

```
SELECT /*+ FULL(TELEFONO) */
       FECHA_COMPRA,
       TIPO,
       MARCA,
       COLOR
FROM  TELEFONO WHERE TIPO =’SP’;
```

La sintaxis es:

```
SELECT /*+ <COMANDO>(<NOMBRE TABLA> <NOMBRE INDICE> */ <columna 1>,
<columna 2>
FROM TABLA;
```

La utilización de HINTS se debe hacer con precaución; si las estadísticas están actualizadas, Oracle utilizará la ruta más óptima para realizar la consulta.

4. Conclusión

Como conclusión, hay que indicar que la optimización de una consulta SQL puede ser compleja y no hay que dudar en pedir ayuda a los DBA, que pueden analizar los resultados de las consultas y darnos buenos consejos.

También hay que acercarse a los expertos (Gestores del proyecto) para saber los tipos de acceso que realizan los usuarios sobre una u otra tabla.

Presentación de SQL*Plus

1. Utilización estándar

SQL*Plus es una herramienta que proporciona Oracle que permite ejecutar comandos SQL directamente en la línea de comandos.

Esta herramienta es muy útil para probar sus comandos y optimizar una consulta paso a paso.

Antes de poder ejecutar SQL*Plus, hay que modificar algunas variables:

- ORACLE_HOME: ubicación de la instalación de Oracle (ejemplo en UNIX: /opt/oracle/product/11.2.0).
- ORACLE_SID: nombre de la base de datos Oracle instalada.

Sintaxis de conexión:

```
sqlplus <usuario>/<contraseña>[@<dirección IP>:<número puerto >/
<nombre de la base de datos>]
```

Si hay sólo una base de datos instalada, no es necesario indicar el nombre de la base de datos, Oracle coge el ORACLE_SID activo.

Si la base de datos Oracle está en otra máquina, hay que indicar la dirección IP de la máquina.

Ejemplo

```
sqlplus SMITH/SMITH@162.12.34.56:1521/BASETEST
```

Se puede empezar a introducir comandos SQL desde que aparezca el prompt SQL>.

Para ejecutar un comando:

```
/
```

Para salir de SQL*Plus:

```
QUIT
```

Para ejecutar el editor por defecto:

```
ed <nombre fichero>
```

Por defecto, en Windows el editor es Notepad y en UNIX es vi.

Para ejecutar un fichero que esté en el directorio y que contenga comandos SQL, hay que teclear:

```
@<nom prog>.sql
```

Una vez se ha ejecutado el script, se puede visualizar el contenido tecleando L.

Para visualizar una fila en particular:

```
L <número de fila>
```

Cambiar un elemento:

```
C/<cadena antigua >/<cadena nueva>/
```

Eliminar una fila:

```
DEL <número de fila>
```

Salir temporalmente de SQL*Plus para ir al sistema operativo (Windows, UNIX...):

```
HOST
```

Volver a SQL*Plus desde el host:

```
exit
```

Ejecutar un comando del sistema operativo:

```
-- DOS
HOST DIR /P
-- UNIX
HOST ls -l
```

Están permitidos todos los comandos SQL. Esto permite visualizar directamente en pantalla el resultado de la consulta.

2. El comando SPOOL

Es posible redirigir el resultado de todo lo que se ejecuta en pantalla. Para ello, hay que utilizar el comando SPOOL.

Ejemplo:

```
SPOOL Fichero_Lista_Peliculas;
SELECT PELICULAS.TITULO, PELICULAS.FECHA_ESTRENO, DIRECTOR.APELLIDO,
ACTOR.APELLIDO, ACTOR.FECHA_NACIMIENTO, ESTA.PRESUPUESTO
FROM PELICULAS PELICULAS, DIRECTOR DIRECTOR, CASTING CAST, ACTOR
ACTOR,
ESTADISTICA ESTA WHERE
   PELICULAS.IDENT_DIRECTOR  = DIRECTOR.IDENT_DIRECTOR AND
   PELICULAS.IDENT_PELICULAS = CAST.IDENT_PELICULAS AND
   PELICULAS.IDENT_PELICULAS = ESTA.IDENT_PELICULAS AND
   CAST.IDENT_ACTOR          = ACTOR.IDENT_ACTOR
ORDER BY PELICULAS.TITULO, ACTOR.NOM DESC;
SPOOL OFF;
```

Se crea un fichero **Fichero_Lista_Peliculas.lst** en el directorio actual con el siguiente contenido:

```
SQL> SELECT PELICULAS.TITULO, PELICULAS.FECHA_ESTRENO, DIRECTOR.APELLIDO,
ACTOR.APELLIDO, ACTOR.FECHA_NACIMIENTO, ESTA.PRESUPUESTO
  2  FROM PELICULAS PELICULAS, DIRECTOR DIRECTOR, CASTING CAST, ACTOR
ACTOR, ESTADISTICA ESTA WHERE
  3     PELICULAS.IDENT_DIRECTOR  = DIRECTOR.IDENT_DIRECTOR AND
  4     PELICULAS.IDENT_PELICULAS = CAST.IDENT_PELICULAS AND
  5     PELICULAS.IDENT_PELICULAS = ESTA.IDENT_PELICULAS AND
  6     CAST.IDENT_ACTOR          = ACTOR.IDENT_ACTOR
  7  ORDER BY PELICULAS.TITULO, ACTOR.NOM DESC;

TITULO                                             FECHA_ES
-------------------------------------------------- --------
               APELLIDO
```

```
------------------------------------------------
APELLIDO                                          FECHA_NA
PRESUPUESTO
------------------------------------------------ -------- ----------
AVATAR                                            16/10/09
CAMERON
WEAVER                                            08/10/49
237


AVATAR                                            16/10/09
CAMERON
SALDANA                                           19/06/78
237
```

El comando Spool primero muestra la consulta y a continuación su resultado.

El comando SPOOL graba la información igual que se muestra en pantalla.

3. Los comandos SET

Existen bastantes comandos que permiten mejorar la visualización e intervenir en el entorno.

La sintaxis estándar es:

```
SET <Opción> <valor>
```

a. Los comandos SET que afectan a la visualización de los datos

Aumentar el tamaño de una línea:

```
SET LINE 300;
```

Poner como separador de campos el punto y coma:

```
SET COLSEP ';';
```

Mostrar o no los encabezados de columnas:

```
SET HEAD [OFF|ON]
```

Dar una longitud por defecto a las columnas numéricas:

```
SET NUM <valor>
```

Definir la longitud de una página en número de líneas:

```
SET PAGES <número>
```

Mostrar o no los espacios al final de la línea:

```
SET TRIMSPOOL [OFF|ON]
```

Definir un formato de visualización de números por defecto:

```
SET NUMFORMAT 999.999.999.999D99
```

muestra por defecto todos los números con el signo $ delante y un punto entre los miles de millones, los millones y los miles.

 Atención, este formato por defecto se aplica a todas las columnas numéricas que se seleccionen.

Ejemplo de utilización:

```
SELECT PELICULAS.TITULO, PELICULAS.FECHA_ESTRENO,
       SUBSTR(DIRECTOR.APELLIDO||' '||DIRECTOR.NOMBRE,1,30)
DIRECTOR,ESTA.RECAUDACION_USA,ESTA.RECAUDACION_MUNDIAL
FROM PELICULAS PELICULAS, DIRECTOR DIRECTOR, ESTADISTICA ESTA WHERE
   PELICULAS.IDENT_DIRECTOR  = DIRECTOR.IDENT_DIRECTOR AND
   PELICULAS.IDENT_PELICULAS = ESTA.IDENT_PELICULAS
ORDER BY PELICULAS.TITULO;

TITULO                              RECAUDACION_USA  RECAUDACION MUNDIAL
 ------------------------------ ------------------- ------------------
AVATAR                             $760.505.847,00    $2.946.271.769,00
BIENVENIDOS AL NORTE                         $,00      $245.000.000,00
NIKITA                                 $5.017.971,00                $,00
STAR WARS 6: EL RETORNO DEL JE     $191.648.000,00      $472.000.000,00
SUBWAY                                   $390.659,00        $1.272.637,45
```

Determinar el número de espacios entre columnas:

```
SET SPACE <nn>
```

Listar el orden de ejecución en el resultado de un script (no tiene ningún efecto en ejecuciones interactivas):

```
SET ECHO [OFF|ON]
```

Determinar el número de caracteres máximo visualizables por los tipos CLOB, LONG, NCLOB y XML:

```
SET LONG <n>;
```

Determinar el número de líneas en blanco al inicio de una nueva página:

```
SET NEWPAGE <n>
```

Si se aplica alguno de estos SET en el ejemplo anterior, podemos llegar a formatear un resultado que se puede utilizar en Excel.

```
SELECT PELICULAS.TITULO, PELICULAS.FECHA_ESTRENO, DIRECTOR.APELLIDO,
ACTOR.APELLIDO,ACTOR.FECHA_NACIMIENTO, ESTA.PRESUPUESTO
FROM PELICULAS PELICULAS, DIRECTOR DIRECTOR, CASTING CAST, ACTOR ACTOR,
ESTADISTICA ESTA WHERE
   PELICULAS.IDENT_DIRECTOR  = DIRECTOR.IDENT_DIRECTOR AND
   PELICULAS.IDENT_PELICULAS = CAST.IDENT_PELICULAS AND
   PELICULAS.IDENT_PELICULAS = ESTA.IDENT_PELICULAS AND
   CAST.IDENT_ACTOR          = ACTOR.IDENT_ACTOR
ORDER BY PELICULAS.TITULO, ACTOR.NOM DESC;
SET LINE 200
SET COLSEP ';'
SET HEAD OFF
SET NUM 9
SET TRIMSPOOL ON
SET PAGE 3000
SPOOL Fichero_Lista_Peliculas;
/
```

```
SPOOL OFF;
```

En el fichero resultante, sólo queda eliminar la primera y las dos últimas líneas para obtener un fichero en formato CSV (legible por Excel).

```
SQL> /

AVATAR                          ;16/10/09;CAMERON;WEAVER   ;08/10/49;  237
AVATAR                            ;16/10/09;CAMERON;SALDANA  ;19/06/78;  237
BIENVENIDOS AL NORTE            ;27/02/08;BOON;MERAD       ;27/03/64;   11
BIENVENIDOS AL NORTE             ;27/02/08;BOON;BOON        ;26/06/66;   11
NIKITA                            ;21/02/90;BESSON;RENO      ;30/06/48;  7,6
NIKITA                            ;21/02/90;BESSON;PARILLAUD ;06/05/60;  7,6
STAR WARS 6: EL RETORNO DEL JEDI ;19/10/83;LUCAS;FORD       ;13/06/42;  32
STAR WARS 6: EL RETORNO DEL JEDI ;19/10/83;LUCAS;FISHER     ;21/10/56; 32
SUBWAY                            ;10/04/85;BESSON;RENO      ;30/06/48;  2,6
SUBWAY                            ;10/04/85;BESSON;LAMBERT   ;29/03/57;  2,6
SUBWAY                            ;10/04/85;BESSON;GALABRU   ;27/10/22;  2,6
SUBWAY                            ;10/04/85;BESSON;BOHRINGER ;16/06/42;  2,6
SUBWAY                            ;10/04/85;BESSON;ADJANI    ;27/06/55;  2,6
13 fila(s) seleccionada(s).

SQL> SPOOL OFF;
```

b. Los comandos SET de entorno

Mostrar (ON) o eliminar (OFF) la visualización del resultado cuando el comando es ejecutado por un script (no tiene ningún efecto en ejecuciones interactivas de comandos):

```
SET TERM [OFF|ON]
```

Mostrar (ON) o no mostrar (OFF) el número de filas encontradas por el comando SQL:

```
SET FEED [OFF|ON]
```

Cambiar el carácter de sustitución por defecto (&) para las variables:

```
SET DEFINE $
```

Eliminar (OFF) la sustitución de variables:

```
SET SCAN [OFF|ON]
```

Mostrar el comando SQL con las sustituciones de variables antes de su ejecución:

```
SET VERIFY [OFF|ON]
```

Obligar a teclear <return> entre cada cambio de página (ON):

```
SET PAUSE [OFF|ON]
```

Definir el carácter que indica la ejecución de un comando (por defecto ;):

```
SET SQLT <x>
```

Indicar la extensión por defecto de los scripts SQL (por defecto .sql):

```
SET SUFFIX <xxx>
```

Mostrar la hora en el prompt:

```
SET TIME [ON|OFF]
```

Mostrar las estadísticas después de cada comando SQL:

```
SET TIMING [ON|OFF]
```

Commit después de cada comando SQL (por defecto OFF):

```
SET AUTOCOMMIT [ON|OFF]
```

4. La utilización de variables en SQL*Plus

También es posible **hacer que el usuario introduzca valores**. Para ello hay que utilizar el símbolo « & », que indica a Oracle que este valor es externo.

Aquí puede ver un ejemplo en el que se pide desde qué fecha se deben seleccionar las películas.

```
SELECT PELICULAS.TITULO, PELICULAS.FECHA_ESTRENO, DIRECTOR.APELLIDO,
ACTOR.APELLIDO,ACTOR.FECHA_NACIMIENTO, ESTA.PRESUPUESTO
FROM PELICULAS PELICULAS, DIRECTOR DIRECTOR, CASTING CAST, ACTOR
ACTOR, ESTADISTICA ESTA
WHERE
   PELICULAS.IDENT_DIRECTOR  = DIRECTOR.IDENT_DIRECTOR AND
   PELICULAS.IDENT_PELICULAS = CAST.IDENT_PELICULAS AND
   PELICULAS.IDENT_PELICULAS = ESTA.IDENT_PELICULAS AND
   CAST.IDENT_ACTOR          = ACTOR.IDENT_ACTOR AND
   PELICULAS.FECHA_ESTRENO > TO_DATE('&FECHA_MIN','DD/MM/YYYY')
ORDER BY PELICULAS.TITULO, ACTOR.APELLIDO DESC;
```

Hay que limitar el &<nombre valor> con comillas para los campos no numéricos. Durante la ejecución, SQL*Plus muestra « Introduzca un valor para FECHA_MIN: »

Ejemplo:

```
/
Introduzca un valor para fecha_min: 15/01/2014
antiguo   8 :    PELICULAS.FECHA_ESTRENO >
TO_DATE('&FECHA_MIN','DD/MM/YYYY')
nuevo     8 :    PELICULAS.FECHA_ESTRENO >
TO_DATE('15/01/2001','DD/MM/YYYY')

AVATAR                    ;16/10/09;CAMERON ;WEAVER  ;08/10/49; 237
AVATAR                    ;16/10/09;CAMERON ;SALDANA ;19/06/78; 237
BIENVENIDOS AL NORTE      ;27/02/08;BOON    ;MERAD   ;27/03/64;  11
BIENVENIDOS AL NORTE      ;27/02/08;BOON    ;BOON    ;26/06/66;  11
SQL> /
Introduzca un valor para fecha_min: 15/01/2014
antiguo   8 :    PELICULAS.FECHA_ESTRENO >
TO_DATE('&FECHA_MIN','DD/MM/YYYY')
nuevo     8 :    PELICULAS.FECHA_ESTRENO >
TO_DATE('15/01/2014','DD/MM/YYYY')

no se ha seleccionado ninguna fila
```

Podemos comprobar en el último ejemplo que siendo la fecha muy reciente, SQL*Plus nos indica que no ha seleccionado ninguna fila.

5. La llamada a procedimientos almacenados

El comando para llamar a un procedimiento almacenado es el comando EXECUTE en Oracle y el comando CALL en MySQL.

Sintaxis Oracle:

```
EXECUTE <Nombre procedimiento almacenado> (<param 1>, <param 2>, etc
...)
```

Sintaxis MySQL:

```
CALL <Nombre procedimiento almacenado> (<param 1>, <param 2>, etc ...)
```

Presentación de los TRIGGER

Un TRIGGER permite ejecutar comandos que se ejecutarán cada vez que se produzca un evento en una tabla. También se les denomina « desencadenadores ».

El contenido del código ejecutado por un trigger suele ser PL/SQL, C o Java.

Los TRIGGER se utilizan a menudo para gestionar la integridad funcional de una aplicación. Permiten realizar controles sobre el contenido de las tablas automáticamente. Los TRIGGER también pueden servir para recuperar información a lo largo de un día sobre la actividad de la base de datos. Estos datos podrán ser tratados por otra aplicación.

En general, los controles se codifican en los programas de las aplicaciones, los TRIGGER permiten añadir otro tipo de controles.

La primera ventaja del TRIGGER es que está vinculado a una acción sobre la base de datos (INSERT, UPDATE, DELETE) así que no hay riesgo de olvidar de modificar un programa. De hecho, suele ser complicado modificar todos los programas de un aplicativo para añadir un control sobre un INSERT por ejemplo; habría que encontrar todos los programas afectados, modificarlos, y probar cada uno de los programas.

El TRIGGER se desencadena sistemáticamente, no se puede olvidar una actualización y la modificación se hace independientemente de los programas de la aplicación.

Un TRIGGER se puede ejecutar antes o después del comando SQL. Se indica con AFTER o BEFORE.

En un TRIGGER BEFORE se puede controlar antes de cualquier modificación de la base algunos elementos e impedir así la actualización.

En un TRIGGER AFTER, tiene lugar la actualización y posteriormente se desencadena el trigger.

Sintaxis general de un TRIGGER:

```
CREATE OR REPLACE TRIGGER <nombre del trigger>
[BEFORE o AFTER] [INSERT o DELETE o UPDATE]
ON <nombre de tabla>
[FOR EACH ROW]
[WHEN]

DECLARE
... ...
BEGIN
... ...

END;
```

Encontramos la misma sintaxis que en los procedimientos almacenados desde el DECLARE (ver capítulo Presentación de PL/SQL - Creación de un procedimiento almacenado).

La cláusula FOR EACH ROW significa que el trigger trata todas las filas afectadas por el comando SQL.

Por ejemplo, un DELETE para borrar varias filas; en este caso, el TRIGGER se aplicará a todas las filas borradas. En caso contrario, el TRIGGER se activa antes o después del borrado de las filas.

La cláusula WHEN permite añadir un criterio suplementario que se aplica a las filas seleccionadas, por ejemplo si se quieren prohibir las eliminaciones solo sobre los registros que tengan un identificador superior a cierto valor.

Si se quiere impedir que los usuarios borren una fila de la tabla PELICULAS, se escribirá:

```
SET SERVEROUTPUT ON
CREATE OR REPLACE TRIGGER T_BOR_PELICULAS
BEFORE DELETE
ON PELICULAS
FOR EACH ROW
DECLARE
```

```
BEGIN

   DBMS_OUTPUT.PUT_LINE( 'No puede eliminar PELICULAS') ;
   DBMS_OUTPUT.PUT_LINE( 'Contacte con el Administrador');
   RAISE_APPLICATION_ERROR(-20502, 'Comando no autorizado');
END;
/
```

Si ahora intenta eliminar una fila de la tabla PELICULAS:

```
Trigger creado.

SQL> DELETE from PELICULAS where ident_film = 1;

No puede eliminar PELICULAS
Contacte con el Administrador

DELETE from PELICULAS film where ident_film = 1
            *
ERROR en la línea 1:
ORA-20502: Comando no autorizado
ORA-06512: en "xxxxx.T_BOR_PELICULAS", línea 7
ORA-04088: error en la ejecución del trigger
'xxxx.T_BOR_PELICULAS'
```

Se muestran los mensajes previstos (No puede...) y a continuación la llamada al módulo de errores (RAISE_APPLICATION_ERROR) para el proceso indicando el nombre del trigger que ha provocado el error (xxxx.T_BOR_PELICULAS).

RAISE_APPLICATION_ERROR es un procedimiento almacenado proporcionado por Oracle que permite crear sus propios mensajes de error. Hay que utilizar códigos de error comprendidos entre -20000 y -20999 que Oracle no utiliza.

1. Creación de un TRIGGER de control y actualización en una tabla

Si queremos ir un poco más lejos, es posible, por ejemplo, comprobar un valor en una tabla cuando insertemos un registro en otra tabla y en función del resultado aceptar o rechazar la inserción.

En la creación de una fila en la tabla ACTOR, comprobamos el código de nacionalidad, verificando que exista en la tabla PAIS.

Si la nacionalidad es desconocida, hay que crearla en la tabla PAIS, con la descripción: « A COMPLETAR ».

Ejemplo de script:

```
CREATE OR REPLACE TRIGGER T_INS_ACTOR
BEFORE INSERT
ON ACTOR
FOR EACH ROW
DECLARE
   numvalor NUMBER(5);
BEGIN
   numvalor := 0;
   SELECT COUNT(*) INTO numvalor FROM PAIS WHERE PAIS.IDENT_PAIS =
:new.NACIONALIDAD;

   IF (numvalor = 0) THEN
       DBMS_OUTPUT.PUT_LINE('La nacionalidad es desconocida:
'||:new.NACIONALIDAD) ;
   DBMS_OUTPUT.PUT_LINE('Se realizará un insert en la tabla PAIS');
   DBMS_OUTPUT.PUT_LINE( 'Complete la descripción del PAIS
```

```
posteriormente');
      INSERT INTO PAIS VALUES (:new.NACIONALIDAD,'A COMPLETAR');
   END IF;
END;
/
```

Aquí puede ver el resultado cuando intentamos insertar un nuevo actor:

```
INSERT INTO ACTOR VALUES
(14,'POELVOORDE','BENOIT',TO_DATE('22/09/1964',
'D D/MM/YYYY'),43,4);
La nacionalidad es desconocida: 4
Se va a realizar un insert en la tabla PAIS
Complete la descripción del PAIS posteriormente
```

Se muestran los mensajes previstos en el TRIGGER. Comprobemos en la tabla PAIS si se ha añadido una fila.

```
SELECT * FROM PAIS;
         1 FRANCIA
         2 USA
         3 ARGELIA
         4 A COMPLETAR
```

Última comprobación, para controlar que también se ha creado el actor:

```
SELECT * FROM ACTOR ORDER BY IDENT_ACTOR;

      1 ADJANI          ISABELLE      27/06/55         42   1
      2 LAMBERT         CHRISTOPHE    29/03/57         64   1
      3 BOHRINGER       RICHARD       16/06/42        132   1
      4 GALABRU         MICHEL        27/10/22        277   1
      5 PARILLAUD       ANNE          06/05/60         35   1
      6 FORD            HARRISON      13/06/42         64   2
      7 FISHER          CARRIE        21/10/56         74   2
      8 SALDANA         ZOE           19/06/78         31   2
      9 WEAVER          SIGOURNEY     08/10/49         66   2
     10 RENO            JEAN          30/06/48         75   1
     11 BOON            DANY          26/06/66         23   1
     12 MERAD           KAD           27/03/64         55   3
     13 HUPPERT         ISABELLE      16/03/53        102   1
     14 POELVOORDE      BENOIT        22/09/64         43   4
```

Si se quieren utilizar los valores de la tabla ACTOR que se van a modificar, hay que utilizar las variables especiales llamadas **new** y **old**.

New.<columna> contiene los nuevos valores que se tendrán en cuenta y old.<columna> contiene los valores anteriores de la tabla ACTOR antes de la modificación.

Si ahora sólo queremos aplicar el trigger a los actores nacidos antes del 31/12/1950 hay que añadir la siguiente cláusula WHEN:

```
CREATE OR REPLACE TRIGGER T_INS_ACTOR
BEFORE INSERT
ON ACTOR
FOR EACH ROW
WHEN (old.FECHA_NACIMIENTO <= TO_DATE('31/12/1950','DD/MM/YYYY'))

DECLARE
   numvalor NUMBER(5);
BEGIN
   numvalor := 0;
   SELECT COUNT(*) INTO numvalor FROM PAIS WHERE PAIS.IDENT_PAIS =
:new.NACIONALIDAD;
```

```
   IF (numvalor = 0) THEN
       DBMS_OUTPUT.PUT_LINE('La nacionalidad es desconocida:
'||:new.NACIONALIDAD) ;
   DBMS_OUTPUT.PUT_LINE('Se va a realizar un insert en la tabla PAIS ');
   DBMS_OUTPUT.PUT_LINE('Complete la descripción PAIS
posteriormente');
      INSERT INTO PAIS VALUES (:new.NACIONALIDAD,'A COMPLETAR');
   END IF;
END;
/
```

Observe que en el caso en que se añade un control sobre una columna de la tabla en la cláusula WHEN no hay que poner los ':'.

2. Creación de un TRIGGER después de las actualizaciones

Supongamos que se quiere grabar en una tabla anexa el número de actualizaciones realizadas en el día en las tablas PELICULAS, ACTOR y CASTING.

Descripción de la tabla de seguimiento de actualizaciones:

Tabla SEGUI_ACT

SEGUI_ACT
FECHA_DIA
NUM_PELICULAS_AÑADIDAS
NUM_PELICULAS_MODIFICADAS
NUM_PELICULAS_ELIMINADAS
NUM_ACTORES_AÑADIDOS
NUM_ACTORES_MODIFICADOS
NUM_ACTORES_ELIMINADOS
NUM_CASTING_AÑADIDOS
NUM_CASTING_MODIFICADOS
NUM_CASTING_ELIMINADOS

Script de creación:

```
CREATE TABLE SEGUI_ACT (FECHA_DIA DATE,
NUM_PELICULAS_AÑADIDAS       NUMBER(6,0),
NUM_PELICULAS_MODIFICADAS    NUMBER(6,0),
NUM_PELICULAS_ELIMINADAS     NUMBER(6,0),
NUM_ACTORES_AÑADIDOS         NUMBER(6,0),
NUM_ACTORES_MODIFICADOS      NUMBER(6,0),
NUM_ACTORES_ELIMINADOS       NUMBER(6,0),
NUM_CASTING_AÑADIDOS         NUMBER(6,0),
NUM_CASTING_MODIFICADOS      NUMBER(6,0),
NUM_CASTING_ELIMINADOS       NUMBER(6,0));
```

Inicialización de la tabla:

```
INSERT INTO SEGUI_ACT VALUES (SYSDATE,0,0,0,0,0,0,0,0,0);
```

Creación del TRIGGER en la tabla PELICULAS:

```
CREATE OR REPLACE TRIGGER ACT_PELICULAS
AFTER INSERT OR DELETE OR UPDATE
ON PELICULAS
FOR EACH ROW
```

```
DECLARE
BEGIN
   IF INSERTING THEN
     UPDATE SEGUI_ACT SET
             NUM_PELICULAS_AÑADIDAS = NUM_PELICULAS_AÑADIDAS + 1
     WHERE TO_CHAR(FECHA_DIA,'DD/MM/YYYY') =
           TO_CHAR(SYSDATE,'DD/MM/YYYY');
   END IF;
   IF UPDATING THEN
     UPDATE SEGUI_ACT SET
             NUM_PELICULAS_MODIFICADAS = NUM_PELICULAS_MODIFICADAS + 1
     WHERE TO_CHAR(FECHA_DIA,'DD/MM/YYYY') =
           TO_CHAR(SYSDATE,'DD/MM/YYYY');
   END IF;
   IF DELETING THEN
     UPDATE SEGUI_ACT SET
             NUM_PELICULAS_ELIMINADAS = NUM_PELICULAS_ELIMINADAS + 1
     WHERE TO_CHAR(FECHA_DIA,'DD/MM/YYYY') =
           TO_CHAR(SYSDATE,'DD/MM/YYYY');
   END IF;
END;
/
```

Creación del TRIGGER en la tabla ACTOR:

```
CREATE OR REPLACE TRIGGER ACT_ACTOR
AFTER INSERT OR DELETE OR UPDATE
ON ACTOR
FOR EACH ROW
DECLARE
BEGIN
   IF INSERTING THEN
     UPDATE SEGUI_ACT SET
             NUM_ACTORES_AÑADIDOS = NUM_ACTORES_AÑADIDOS + 1
     WHERE TO_CHAR(FECHA_DIA,'DD/MM/YYYY') =
           TO_CHAR(SYSDATE,'DD/MM/YYYY');
   END IF;
   IF UPDATING THEN
     UPDATE SEGUI_ACT SET
             NUM_ACTORES_MODIFICADOS = NUM_ACTORES_MODIFICADOS + 1
     WHERE TO_CHAR(FECHA_DIA,'DD/MM/YYYY') =
           TO_CHAR(SYSDATE,'DD/MM/YYYY');
   END IF;
   IF DELETING THEN
     UPDATE SEGUI_ACT SET
             NUM_ACTORES_ELIMINADOS = NUM_ACTORES_ELIMINADOS + 1
     WHERE TO_CHAR(FECHA_DIA,'DD/MM/YYYY') =
           TO_CHAR(SYSDATE,'DD/MM/YYYY');
   END IF;
END;
/
```

Creación del TRIGGER en la tabla CASTING:

```
CREATE OR REPLACE TRIGGER ACT_CASTING
AFTER INSERT OR DELETE OR UPDATE
ON CASTING
FOR EACH ROW
DECLARE
BEGIN
   IF INSERTING THEN
     UPDATE SEGUI_ACT SET
             NUM_CASTING_AÑADIDOS = NUM_CASTING_AÑADIDOS + 1
     WHERE TO_CHAR(FECHA_DIA,'DD/MM/YYYY') =
           TO_CHAR(SYSDATE,'DD/MM/YYYY');
   END IF;
   IF UPDATING THEN
     UPDATE SEGUI_ACT SET
```

```
            NUM_CASTING_MODIFICADOS = NUM_CASTING_MODIFICADOS + 1
    WHERE TO_CHAR(FECHA_DIA,'DD/MM/YYYY') =
          TO_CHAR(SYSDATE,'DD/MM/YYYY');
  END IF;
  IF DELETING THEN
    UPDATE SEGUI_ACT SET
            NUM_CASTING_ELIMINADOS = NUM_CASTING_ELIMINADOS + 1
    WHERE TO_CHAR(FECHA_DIA,'DD/MM/YYYY') =
          TO_CHAR(SYSDATE,'DD/MM/YYYY');
  END IF;
END;
/
```

Para cada TRIGGER las acciones se aplican sobre todos los comandos gracias a la orden: AFTER **INSERT OR DELETE OR UPDATE**

Los TRIGGER se desencadenan después de las actualizaciones para tener sólo en cuentan las que se han hecho realmente.

Ahora ejecutamos algunos comandos sobre estas tablas:

```
DELETE FROM ACTOR;
DELETE FROM PELICULAS WHERE IDENT_PELICULAS = 3;
INSERT INTO CASTING VALUES (1,1,'HELENA',100);
INSERT INTO CASTING VALUES (1,2,'FRED',100);
INSERT INTO CASTING VALUES (1,3,'INSPECTOR GESBERG',NULL);
INSERT INTO CASTING VALUES (1,4,'LA FLORISTA',35);
INSERT INTO CASTING VALUES (1,10,'EL BATERÍA',20);
INSERT INTO CASTING VALUES (2,5,'NIKITA',68);
INSERT INTO CASTING VALUES (2,10,'VICTOR EL LIMPIADOR',9);
INSERT INTO CASTING VALUES (3,6,'HAN SOLO',201);
UPDATE PELICULAS SET GENERO2 = 'ACCIÓN' WHERE IDENT_PELICULAS IN (3,5);
```

Lo que provoca los siguientes cambios en la tabla SEGUI_ACT:

```
SELECT * FROM SEGUI_ACT;
```

FECHA_DIA	06/07/14
NUM_PELICULAS_AÑADIDAS	0
NUM_PELICULAS_MODIFICADAS	2
NUM_PELICULAS_ELIMINADAS	1
NUM_ACTORES_AÑADIDOS	0
NUM_ACTORES_MODIFICADOS	0
NUM_ACTORES_ELIMINADOS	14
NUM_CASTING_AÑADIDOS	8
NUM_CASTING_MODIFICADOS	0
NUM_CASTING_ELIMINADOS	0

Tenga en cuenta que TRUNCATE no se considera como un DELETE y no tendrá ningún efecto en un TRIGGER ON DELETE.

Las tablas del sistema (Oracle)

Oracle y MySQL utilizan para sus necesidades un conjunto de tablas para almacenar todos los elementos creados por un usuario. Todos los objetos se almacenan en las llamadas tablas del sistema.

Podemos acceder a ellas simplemente con el comando: `SELECT * FROM <Nombre tabla>;`

1. Tablas del sistema para tablas y columnas

a. Oracle

NOMBRE TABLA	CONTENIDO
ALL_COL_COMMENTS	Lista todos los comentarios de las columnas de todas las tablas.
ALL_TABLES	Lista todas las tablas.
ALL_TAB_COLUMNS	Lista todas las columnas de todas las tablas.

b. MySQL

NOMBRE TABLA	CONTENIDO
INFORMATION_SCHEMA.TABLES	Lista todas las tablas.
INFORMATION_SCHEMA.COLUMNS	Lista todas las columnas de todas las tablas.

2. Tablas del sistema para índices y vistas

a. Oracle

NOMBRE TABLA	CONTENIDO
ALL_INDEXES	Lista todos los índices.
ALL_IND_COLUMNS	Lista todas las columnas de todos los índices.
ALL_VIEWS	Lista todas las vistas.

b. MySQL

NOMBRE TABLA	CONTENIDO
INFORMATION_SCHEMA. STATISTICS	Lista toda la información sobre los índices.
INFORMATION_SCHEMA. VIEWS	Lista todas las vistas.

3. El resto de tablas del sistema

a. Oracle

NOMBRE TABLA	CONTENIDO
ALL_CATALOG	Lista todas las tablas, vistas, secuencias y sinónimos.

ALL_CONSTRAINTS	Lista las restricciones.
ALL_OBJECTS	Lista todos los objetos a los que puede acceder el usuario.
ALL_SEQUENCES	Lista las secuencias.
ALL_SYNONYMS	Lista los sinónimos.
ALL_TRIGGERS	Lista todos los triggers.
ALL_TRIGGERS_COLS	Lista todas las columnas de los triggers.
ALL_USERS	Lista los usuarios declarados.

b. MySQL

NOMBRE TABLA	CONTENIDO
INFORMATION_SCHEMA.SCHEMATA	Lista todas las tablas, vistas, secuencias y sinónimos.
INFORMATION_SCHEMA.CONSTRAINTS	Lista todas las restricciones.
INFORMATION_SCHEMA.COLUMN_PRIVILEGES	Lista todos los objetos a los que puede acceder el usuario.
INFORMATION_SCHEMA.USER_PRIVILEGES	Lista los usuarios declarados.
INFORMATION_SCHEMA.COLUMN_PRIVILEGES	Lista los privilegios sobre las columnas.
INFORMATION_SCHEMA.TABLE_PRIVILEGES	Lista los privilegios sobre las tablas.
INFORMATION_SCHEMA.ROUTINES	Lista las funciones y los procedimientos almacenados.
INFORMATION_SCHEMA.TRIGGERS	Lista toda la información de los triggers.

Algunos scripts útiles

1. Saber el tamaño real de una columna

En una columna declarada como VARCHAR, puede ser interesante saber el tamaño real de cada valor.

Esta consulta permite además ordenar el resultado.

Sintaxis:

```
SELECT <nombre de columna>, LENGTH (RTRIM(<nombre de columna>))
FROM <nombre tabla> WHERE    ..
ORDER BY LENGTH (RTRIM(<nombre de columna>)),<nombre de columna>
```

Ejemplo:

```
SELECT TITULO,LENGTH (RTRIM(TITULO))AS LONGITUD_TITULO FROM PELICULAS
ORDER BY LENGTH (RTRIM(TITULO)), TITULO;
```

TITULO	LONGITUD_TITULO
AVATAR	6
NIKITA	6
SUBWAY	6
BIENVENIDOS AL NORTE	20
STAR WARS 6 - EL RETORNO DEL JEDI	33

2. Buscar y eliminar duplicados en una tabla

A menudo nos encontramos con filas duplicadas en una tabla después de una mala manipulación o de un error en la aplicación que no controle los duplicados.

Si retomamos la tabla ACTOR y añadimos la fila n°13 con 'HARRISON FORD' que existe ya en la fila 6.

Contenido de la tabla ACTOR:

IDENT_ACTOR	APELLIDO	NOMBRE	FECHA_NACIMIENTO	NUM_PELICULAS	NACIONALIDAD
1	ADJANI	ISABELLE	27/06/55	42	1
2	LAMBERT	CHRISTOPHE	29/03/57	64	1
3	BOHRINGER	RICHARD	16/06/42	132	1
4	GALABRU	MICHEL	27/10/22	277	1
5	PARILLAUD	ANNE	06/05/60	35	1
6	**FORD**	**HARRISON**	**13/06/42**	**64**	**2**
7	FISHER	CARRIE	21/10/56	74	2
8	SALDANA	ZOE	19/06/78	31	2
9	WEAVER	SIGOURNEY	08/10/49	66	2
10	RENO	JEAN	30/06/48	75	1
11	BOON	DANY	26/06/66	23	1
12	MERAD	KAD	27/03/64	55	3
13	**FORD**	**HARRISON**	**01/03/44**	**3**	**2**

En primer lugar, hay que buscar las filas que puedan ser duplicados mediante la columna ROWID que es el identificador único generado por Oracle para todas las filas de la base de datos.

Sintaxis:

```
SELECT * FROM <nombre tabla> A WHERE A.ROWID > ANY
(SELECT B.ROWID FROM <nombre tabla> B WHERE A.<1a clave común> =
B.<1a clave común>) AND  A.<2a clave común> = B.<2a clave común>);
```

Esta sintaxis funciona sólo en Oracle, MySQL no tiene el concepto de ROWID.

Ejemplo en la tabla ACTOR:

```
SELECT * FROM ACTOR A WHERE A.ROWID > ANY
```

Ahora, para borrar la fila:

```
(SELECT B.ROWID FROM ACTOR B WHERE A.NOMBRE = B.NOMBRE AND A.APELLIDO
= B.APELLIDO);
```

IDENT_ACTOR	APELLIDO	NOMBRE	FECHA_NACIMIENTO	NUM_PELICULAS	NACIONALIDAD
13	FORD	HARRISON	01/03/44	3	2

```
DELETE FROM ACTOR A WHERE A.ROWID > ANY
(SELECT B.ROWID FROM ACTOR B WHERE A.NOMBRE = B.NOMBRE AND  A.APELLIDO
= B.APELLIDO);
```

3. Extraer los datos de una tabla en un fichero plano

Para extraer el contenido de una tabla en un fichero de texto hay que utilizar el comando SPOOL en Oracle, o el SELECT … INTO OUTFILE en MySQL.

Estas dos funciones tienen la misma finalidad: disponer los datos en un fichero para poder utilizarlos en un procesador de textos o una hoja de cálculo, por ejemplo.

Ejemplo Oracle - comando SPOOL

```
SET HEADING OFF FEEDBACK OFF ECHO OFF VERIFY OFF SPACE 0 PAGESIZE
0 TERMOUT OFF WRAP OFF
SET LINESIZE  300
SET COLSEP ";"

SPOOL Extraccion_Actores.TXT
SELECT * FROM ACTOR;

SPOOL OFF
SET TERMOUT ON
```

Los comandos SET permiten formatear el resultado, las opciones son numerosas y las más conocidas son:

- LINESIZE: longitud de la línea.
- PAGESIZE: número de líneas por página.
- HEADING: con (ON) o sin (OFF) encabezados de columnas.
- ECHO: muestra (ON) o no (OFF) el resultado en la pantalla.
- COLSEP: carácter de separación entre columnas.
- etc.

Resultado obtenido:

```
SQL> SELECT * FROM ACTOR;
1;ADJANI                                ;ISABELLE     ;
27/06/55;        42;          1
2;LAMBERT                               ;CHRISTOPHE   ;
29/03/57;        64;          1
3;BOHRINGER                             ;RICHARD      ;
16/06/42;       132;          1
4;GALABRU                               ;MICHEL       ;
27/10/22;       277;          1
5;PARILLAUD                             ;ANNE         ;
06/05/60;        35;          1
6;FORD                                  ;HARRISON     ;
13/06/42;        64;          2
7;FISHER                                ;CARRIE       ;
21/10/56;        74;          2
8;SALDANA                               ;ZOE          ;
19/06/78;        31;          2
9;WEAVER                                          ;SIGOURNEY
;08/10/49;        66;          2
10;RENO                                           ;JEAN
;30/06/48;        75;          1
11;BOON                                           ;DANY
;26/06/66;        23;          1
12;MERAD                                          ;KAD
;27/03/64;        55;          3
SQL>
SQL> SPOOL OFF
```

Tenga en cuenta que encontramos también todos los comandos que se han escrito en la línea de comandos.

Ejemplo MySQL - comando INTO OUTFILE

```
SELECT * FROM ACTOR INTO OUTFILE 'C:\\Extraccion_Actores.TXT';
```

También existen opciones para dar formato al resultado, como por ejemplo:

```
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
  LINES TERMINATED BY '\n' etc ...
```

4. Mostrar el contenido de una tabla sin conocer su estructura

Podemos encontrar toda la información de una tabla sin conocer la estructura utilizando las tablas internas de Oracle.

Si, por ejemplo, aplicamos el siguiente script sobre la tabla PELICULAS, se generará un comando SELECTde la tabla que se puede utilizar posteriormente.

```
COLUMN VARIAB1 NOPRINT
COLUMN VARIAB2 NOPRINT
SELECT 'A' VARIAB1, 0 VARIAB2, 'SELECT '
  FROM DUAL
UNION
SELECT 'B', COLUMN_ID, DECODE(COLUMN_ID, 1, '    ', '  , ')
         || DECODE(DATA_TYPE,'DATE','TO_CHAR('||'"'||
         COLUMN_NAME||'"'||','YYYYMMDDHH24MISS'') '||'"'||
         COLUMN_NAME||'"'  ,'"'||COLUMN_NAME||'"')
  FROM USER_TAB_COLUMNS
 WHERE TABLE_NAME=UPPER('PELICULAS')
UNION
SELECT 'C', 0, 'FROM PELICULAS'
  FROM DUAL
UNION
SELECT 'D', 0, ';'
  FROM DUAL
 ORDER BY 1,2
/
```

Este es el resultado de la selección anterior:

La ejecución del comando muestra los elementos de la tabla PELICULAS.

Sustituyendo el nombre de la tabla por una variable (&1), podemos visualizar todas las estructuras de tablas conociendo sólo el nombre de las tablas.

```
SELECT
    "IDENT_PELICULAS"
  , "TITULO"
  , "GENERO1"
  , "GENERO2"
  , TO_CHAR("FECHA_ESTRENO",'YYYYMMDDHH24MISS') "FECHA_ESTRENO"
  , "PAIS"
  , "IDENT_DIRECTOR"
  , "DISTRIBUIDOR"
  , "SINOPSIS"
FROM PELICULAS
;
```

```
 000001. 'SUBWAY'
 'POLICIACA'
 'DRAMA'              19850410000000  000001.  000001. 'FILMAX'
'Cuenta las aventuras de la población subterránea en los túneles
del metro de París'

 000003. 'STAR WARS 6: EL RETORNO DEL JEDI'
                'ACCIÓN'
 'SF'                 19831019000000  000002.  000002. '20th
Century Fox '
'El imperio galáctico es más poderoso que nunca: la construcción de
la nueva arma, la Estrella de la Muerte, amenaza todo el universo'

 000004. 'AVATAR'
'ACCIÓN'
 'SF'                 20091016000000  000002.  000003. '20th
Century Fox '
'A pesar de su parálisis, Jake Sully, un viejo marine inmovilizado
en una silla de ruedas, en el fondo sigue siendo un soldado'

000005. 'BIENVENIDOS AL NORTE'
'COMEDIA'
 ''                   20080227000000  000001.  000004. 'LAUREN FILMS'
' Philippe Abrams, director de correos de Salon-de-Provence, es
trasladado al Norte.'
```

5. Generar comandos de inserción en una tabla a partir de un fichero Excel

Puede ser muy molesto crear comandos INSERT a partir de un fichero Excel que contenga los datos a crear.

Organizando el fichero Excel en el mismo orden de las columnas de la tabla se pueden generar fácilmente los comandos necesarios.

Ejemplo de fichero Excel:

	A	B	C	D	E	F	G	H	I
1	**Número**	**Título**	**Género 1**	**Género 2**	**Fecha de estreno**	**País**	**Director**	**Distribuidor**	**Sinopsis**
2	1	SUBWAY	POLICIACA	DRAMA	10/04/85	1	1	FILMAX	Cuenta las aventuras de la población subterránea en los túneles del metro de París
3	2	NIKITA	DRAMA	ROMANTICA	21/02/90	1	1	FILMAX	Nikita, condenada a cadena perpetua, es obligada a trabajar en secreto para el gobierno como agente de los servicios secretos
4	3	STAR WARS 6: EL RETORNO DEL JEDI	ACCIÓN	SF	19/10/83	2	2	20th Century Fox	El imperio galáctico es más poderoso que nunca: la construcción de la nueva arma, la Estrella de la Muerte, amenaza todo el universo
5	4	AVATAR	ACCIÓN	SF	16/10/09	2	3	20th Century Fox	A pesar de su parálisis, Jake Sully, un viejo marine inmovilizado en una silla de ruedas, en el fondo sigue siendo un soldado
6	5	BIENVENIDOS AL NORTE	COMEDIA		27/02/08	1	4	LAUREN FILMS	Philippe Abrams, director de correos de Salon-de-Provence, es trasladado al Norte

Al final de cada fila (en la columna J), hay que escribir una fórmula de este tipo:

Celda J2

Celda J3

Celda J4

Celda J5

Celda J6

Cada valor de las celdas da el siguiente resultado:

A continuación basta con copiar/pegar las celdas J2 a J6 en un editor de texto cualquiera (Notepad, Textpad, etc.) y crear un fichero con la extensión «.sql» y ejecutarlo en SQL*Plus con @<nombre script>.

Esta alternativa es más adecuada a la utilización de SQL*Loader cuando no haya muchas filas a insertar. También se puede utilizar este método para realizar Updates.

```
="INSERT INTO PELICULAS VALUES
("&A2&",'"&B2&"','"&C2&"','"&D2&"',TO_DATE('"&E2&"','DD/MM/YYYY'),
"&F2&","&G2&",'"&H2&"','"&I2&"');"
```

```
="INSERT INTO PELICULAS VALUES
("&A3&",'"&B3&"','"&C3&"','"&D3&"',TO_DATE('"&E3&"','DD/MM/YYYY'),
```

```
"&F3&","&G3&",'"&H3&"','"&I3&"');"
```

```
="INSERT INTO PELICULAS VALUES
("&A4&",'"&B4&"','"&C4&"','"&D4&"',TO_DATE('"&E4&"','DD/MM/YYYY'),
"&F4&","&G4&",'"&H4&"','"&I4&"');"
```

```
="INSERT INTO PELICULAS VALUES
("&A5&",'"&B5&"','"&C5&"','"&D5&"',TO_DATE('"&E5&"','DD/MM/YYYY'),
"&F5&","&G5&",'"&H5&"','"&I5&"');"
```

```
="INSERT INTO PELICULAS VALUES
("&A6&",'"&B6&"','"&C6&"','"&D6&"',TO_DATE('"&E6&"','DD/MM/YYYY'),
"&F6&","&G6&",'"&H6&"','"&I6&"');"
```

```
INSERT INTO PELICULAS VALUES
(1,'SUBWAY','POLICIACA','DRAMA',TO_DATE('10/04/1985','DD/MM/YYYY'),
1,1,'FILMAX','Cuenta las aventuras de la población subterránea en
los túneles del metro de París');
INSERT INTO PELICULAS VALUES
(2,'NIKITA','DRAMA','ROMANTICA',TO_DATE('21/02/1990','DD/MM/YYYY'
),1,1,'FILMAX','Nikita, condenada a cadena perpetua, es obligada a
trabajar en secreto para el gobierno como agente de los servicios
secretos');
INSERT INTO PELICULAS VALUES (3,'STAR WARS 6: EL RETORNO DEL
JEDI','ACCIÓN','SF',TO_DATE('19/10/1983','DD/MM/YYYY'),2,2,'20th
Century Fox ','El imperio galáctico es más poderoso que nunca: la
construcción de la nueva arma, la Estrella de la Muerte, amenaza todo
el universo');
INSERT INTO PELICULAS VALUES
(4,'AVATAR','ACCIÓN','SF',TO_DATE('16/10/2009','DD/MM/YYYY'),2,3,'
20th Century Fox ','A pesar de su parálisis, Jake Sully, un viejo
marine inmovilizado en una silla de ruedas, en el fondo sigue siendo
un soldado');
INSERT INTO PELICULAS VALUES (5,'BIENVENIDOS AL NORTE','COMEDIA',
'',TO_DATE('27/02/2008','DD/MM/YYYY'),1,4,'LAUREN FILMS',
'Philippe Abrams, director de correos de Salon-de-Provence, es
trasladado al Norte.');
```

6. Procedimiento almacenado para eliminar filas en una tabla

Existe el comando TRUNCATE que vacía por completo una tabla, pero que no permite volver atrás en caso de error. De hecho, el ROLLBACK no funciona con el TRUNCATE, la ejecución es inmediata e irreversible.

El interés principal del TRUNCATE es su rapidez respecto al DELETE.

El comando DELETE es más seguro ya que permite la vuelta atrás en caso de error. Sin embargo, en tablas grandes, no siempre se puede eliminar tantas filas como queramos sin cometer errores, sobre todo en los segmentos rollback que sean demasiado pequeños.

Los segmentos rollback son espacios de almacenamiento en los que Oracle almacena la información de actualización sobre las tablas. Esto es lo que permite volver atrás en el caso de utilización de un ROLLBACK.

Es preferible cortar los DELETE en paquetes y hacer commit regularmente para descargar el motor Oracle y los segmentos de rollback.

En el siguiente ejemplo, se eliminan las filas de la tabla PELICULAS utilizando IDENT_PELICULAS como criterio de corte por paquete.

Hay que pasar un parámetro de entrada para indicarle a partir de qué número se empieza la eliminación.

Se eliminan 10000 filas por paquete: variable « paso ».

También se muestran después de cada DELETE las posiciones de las variables (irows e indic).

Al inicio del procedimiento se calcula el número de filas de la tabla PELICULAS para poder parar el bucle utilizando select max.

```
CREATE OR REPLACE PROCEDURE Elim_Filas_Peliculas
(iinicio        IN number)
AS
paso number;
indic number;
irows number;
maxIDENT_PELICULAS number;
done boolean;

BEGIN
done := FALSE;
```

```
paso :=100000;
indic := iinicio;
irows := iinicio + paso;

maxIDENT_PELICULAS :=0;
select max(IDENT_PELICULAS) into maxIDENT_PELICULAS  from clie;

while not done loop
   DELETE from PELICULAS where IDENT_PELICULAS between indic
and irows ;
   commit;

   DBMS_OUTPUT.PUT_LINE('indic '|| indic||' irows ' ||irows );
   irows := irows+paso;
   indic := indic+paso;
   if indic > maxIDENT_PELICULAS  then
      done := TRUE;
   end if;

END loop;
commit;
END;
/
```

Introducción

El PL/SQL es un lenguaje estructurado creado por Oracle que permite asociar comandos SQL con comandos de un lenguaje procedural.

Este lenguaje permite crear funciones específicas o incluso realizar aplicaciones más o menos complejas.

Los elementos creados en PL/SQL se deben compilar antes de ejecutarse.

Todas las instrucciones SQL se pueden utilizar en un bloque PL/SQL. Un « bloque » es un trozo de código PL/SQL, equivalente a una función o un procedimiento en otro lenguaje.

Sintaxis general

En un programa PL/SQL, se pueden utilizar los tipos Oracle para las variables así como crear sus propios tipos.

El bloque se divide en tres partes:

- Una parte declarativa
- Una parte de proceso
- Una parte de gestión de errores

La parte declarativa permite declarar e inicializar todas las variables que se utilizarán en la parte de proceso.

La parte de gestión de errores permite indicar las instrucciones que se ejecutarán cuando se encuentre un error en el proceso.

Estas dos secciones (declarativa y errores) son opcionales.

Si la sintaxis de un bloque PL/SQL es la siguiente:

```
[DECLARE
...]
BEGIN
...
...
[EXCEPTION
...]
END;
```

Por ejemplo, si retomamos un ejemplo del capítulo La manipulación de los datos (LMD) - Ejercicios de aplicación.

Seleccionar el título de la película, la fecha de estreno, el apellido y el nombre del director, el apellido y nombre del actor, su fecha de nacimiento, el presupuesto de la película y el número de entradas vendidas en España de las películas que tengan un actor argelino.

Se puede codificar en PL/SQL del siguiente modo:

```
DECLARE
pais_busqueda VARCHAR2(100) := 'ARGELIA';
titulo_pelicula VARCHAR2(100)
fecha_pelicula DATE;
director VARCHAR2(100);
apellido_actor VARCHAR2(100);
nombre_actor VARCHAR2(100);
fecha_nacimiento DATE;
numero_peliculas NUMBER(8);
presupuesto DECIMAL(10,2);
numero_entradas NUMBER(8);
BEGIN
SELECT PELICULAS.TITULO, PELICULAS.FECHA_ESTRENO,
       DIRECTOR.APELLIDO||' '||DIRECTOR.NOMBRE DIRECTOR,
       ACTOR.APELLIDO APELLIDO,
       ACTOR.NOMBRE NOMBRE, ACTOR.FECHA_NACIMIENTO,
       ACTOR.NUM_PELICULAS,ESTA.PRESUPUESTO, ESTA.NUM_ENTRADAS_ESPANA
       ENTRADAS
INTO
       titulo_pelicula,fecha_pelicula,
       director,apellido_actor,
       nombre_actor ,fecha_nacimiento ,
       numero_peliculas,presupuesto,numero_entradas
FROM   PELICULAS PELICULAS, DIRECTOR DIRECTOR, CASTING CAST,
       ACTOR ACTOR, ESTADISTICA ESTA, PAIS PAIS
WHERE
   PELICULAS.IDENT_DIRECTOR  = DIRECTOR.IDENT_DIRECTOR AND
```

```
   PELICULAS.IDENT_PELICULAS = CAST.IDENT_PELICULAS AND
   PELICULAS.IDENT_PELICULAS = ESTA.IDENT_PELICULAS AND
   CAST.IDENT_ACTOR          = ACTOR.IDENT_ACTOR AND
   PAIS.IDENT_PAIS           = ACTOR.NACIONALIDAD AND
   PAIS.DESCRIPCION          = pais_busqueda
ORDER BY PELICULAS.TITULO;
DBMS_OUTPUT.PUT_LINE( 'SQLCODE     : '||TO_CHAR(SQLCODE));
IF SQLCODE = 0 THEN
   DBMS_OUTPUT.PUT_LINE( 'Título de la película: '||titulo_pelicula);
   DBMS_OUTPUT.PUT_LINE( 'Fecha de estreno     : '||fecha_pelicula);
   DBMS_OUTPUT.PUT_LINE( 'Director             : '||director);
   DBMS_OUTPUT.PUT_LINE( 'Apellido actor       : '||apellido_actor);
   DBMS_OUTPUT.PUT_LINE( 'Nombre actor         : '||nombre_actor);
   DBMS_OUTPUT.PUT_LINE( 'Fecha de nacimiento  : '||fecha_nacimiento);
   DBMS_OUTPUT.PUT_LINE( 'Número de películas  : '||numero_peliculas);
   DBMS_OUTPUT.PUT_LINE( 'Presupuesto          : '||presupuesto);
   DBMS_OUTPUT.PUT_LINE( 'Número de entradas   : '||numero_peliculas);
END IF;
EXCEPTION
   WHEN NO_DATA_FOUND THEN
     DBMS_OUTPUT.PUT_LINE( 'No se ha encontrado ninguna fila con el
país: '|| pais_busqueda) ;
   WHEN OTHERS THEN
     DBMS_OUTPUT.PUT_LINE( 'El número de error es: '||
TO_CHAR( SQLCODE )) ;
     DBMS_OUTPUT.PUT_LINE( 'correspondiente a: '||
TO_CHAR( SQLERRM )) ;
END;
/
```

La declaración de las variables se puede hacer indicando el tipo pero también se puede indicar el nombre de una columna de una tabla y así la nueva variable coge el mismo tipo que la columna. Para ello hay que añadir « %type » después del nombre de la variable.

Ejemplos:

```
pais_busqueda VARCHAR2(100) := 'ARGELIA';
titulo_pelicula PELICULAS.TITULO%type;
fecha_pelicula PELICULAS.FECHA_ESTRENO%type;
director VARCHAR2(100);
apellido_actor ACTOR.APELLIDO%type;
nombre_actor ACTOR.NOMBRE%type;
fecha_nacimiento ACTOR.FECHA_NACIMIENTO%type;
numero_peliculas NUMBER(8);
```

DBMS_OUTPUT.PUT_LINE es una función Oracle que permite mostrar información en la línea de comandos.

Para ver los resultados del procedimiento, hay que activar en SQL*Plus la visualización con el comando: **SET SERVEROUTPUT ON**.

*Resultado de la ejecución en SQL*Plus:*

```
SQLCODE              : 0
Título de la película : BIENVENIDOS AL NORTE
Fecha de estreno     : 27/02/08
Director             : BOON DANY
Apellido actor       : MERAD
Nombre actor         : KAD
Fecha de nacimiento  : 27/03/64
Número de películas  : 55
Presupuesto          : 11
Número de entradas   : 21000000

Procedimiento PL/SQL terminado correctamente.
```

Si, por ejemplo, se sustituye 'ARGELIA' por 'BÉLGICA'.

```
l2
c/ARGELIA/BÉLGICA/
/ No se ha encontrado ninguna fila con el país: BÉLGICA

Procedimiento PL/SQL terminado correctamente.
```

El procedimiento nos indica que ninguna fila corresponde a ese país. La excepción NO_DATA_FOUND ha interceptado el error y muestra el mensaje previsto. El SQLCODE correspondiente a NO_DATA_FOUND es +100.

SQLCODE contiene siempre el número de error de la última instrucción SQL ejecutada. Es 0 si no hay ningún error.

SQLERRM contiene la descripción correspondiente al último número de error SQL encontrado.

Si, por el contrario, se buscan las películas para 'FRANCIA' el resultado será:

```
l2
c/BÉLGICA/FRANCIA/

El número de error es: -1422
correspondiente a: ORA-01422: la recuperación exacta devuelve
un número mayor de filas que el solicitado

Procedimiento PL/SQL terminado correctamente.
```

Hay varias filas que corresponden a la selección, y como no se puede recuperar más que una fila en el INTO, la excepción nos muestra el código y la descripción del error correspondiente.

Cuando queremos devolver varias filas, hay que utilizar cursores.

Los cursores

Un cursor es un elemento que permite almacenar una consulta que devuelva varias filas.

Hay que declararlo en la sección declarativa.

Hay que abrirlo con un **OPEN**, ejecutarlo con un **FETCH** y cerrarlo con un **CLOSE**.

En el ejemplo, el país buscado se pasa como parámetro al cursor: CURSOR C_PELICULAS_POR_PAIS (PAISB IN VARCHAR2) IS

PAISB se indica en el OPEN CURSOR con la variable que contiene la descripción del país: OPEN C_PELICULAS_POR_PAIS(pais_busqueda);

Ejemplo con la misma consulta anterior:

```
DECLARE

-- declaración del cursor C_PELICULAS_POR_PAIS
CURSOR C_PELICULAS_POR_PAIS (PAISB IN VARCHAR2) IS
SELECT PELICULAS.TITULO, PELICULAS.FECHA_ESTRENO,
       DIRECTOR.APELLIDO||' '||DIRECTOR.NOMBRE DIRECTOR, ACTOR.APELLIDO
       APELLIDO,
       ACTOR.NOMBRE NOMBRE, ACTOR.FECHA_NACIMIENTO,
       ACTOR.NUM_PELICULAS,ESTA.PRESUPUESTO, ESTA.NUM_ENTRADAS_ESPANA
       ENTRADAS

FROM   PELICULAS PELICULAS, DIRECTOR DIRECTOR, CASTING CAST,
       ACTOR ACTOR, ESTADISTICA ESTA, PAIS PAIS
WHERE
   PELICULAS.IDENT_DIRECTOR  = DIRECTOR.IDENT_DIRECTOR AND
   PELICULAS.IDENT_PELICULAS = CAST.IDENT_PELICULAS AND
   PELICULAS.IDENT_PELICULAS = ESTA.IDENT_PELICULAS AND
   CAST.IDENT_ACTOR          = ACTOR.IDENT_ACTOR AND
   PAIS.IDENT_PAIS           = ACTOR.NACIONALIDAD AND
   PAIS.DESCRIPCION          =  PAISB
ORDER BY PELICULAS.TITULO;

/* declaración de las variables que reciben los datos
   ------------------------------------  */
titulo_pelicula VARCHAR2(100);
fecha_pelicula DATE;
director VARCHAR2(100);
apellido_actor VARCHAR2(100);
nombre_actor VARCHAR2(100);
fecha_nacimiento DATE;
numero_peliculas NUMBER(8);
presupuesto DECIMAL(10,2);
numero_peliculas NUMBER(8);

/* declaración de otras variables
 ---------------------------------- */
pais_busqueda VARCHAR2(100) := 'FRANCIA';

BEGIN
-- Apertura
OPEN C_PELICULAS_POR_PAIS(pais_busqueda);
-- Bucle de lectura
LOOP
-- recuperación de los elementos fila a fila
   FETCH C_PELICULAS_POR_PAIS INTO
       titulo_pelicula,fecha_pelicula,
       director,apellido_actor,
       nombre_actor ,fecha_nacimiento ,
       numero_peliculas,presupuesto,numero_peliculas;
   Exit When C_PELICULAS_POR_PAIS%NOTFOUND;
-- Visualización de los elementos recuperados
   DBMS_OUTPUT.PUT_LINE( 'Título de la película: '||titulo_pelicula);
```

```
   DBMS_OUTPUT.PUT_LINE( 'Fecha de estreno     : '||fecha_pelicula);
   DBMS_OUTPUT.PUT_LINE( 'Director             : '||director);
   DBMS_OUTPUT.PUT_LINE( 'Apellido actor       : '||apellido_actor);
   DBMS_OUTPUT.PUT_LINE( 'Nombre actor         : '||nombre_actor);
   DBMS_OUTPUT.PUT_LINE( 'Fecha de nacimiento  : '||fecha_nacimiento);
   DBMS_OUTPUT.PUT_LINE( 'Número de películas  : '||numero_peliculas);
   DBMS_OUTPUT.PUT_LINE( 'Presupuesto          : '||presupuesto);
   DBMS_OUTPUT.PUT_LINE( 'Número de entradas   : '||numero_entradas);
   DBMS_OUTPUT.PUT_LINE( '--------------------------------- ');
END LOOP;
-- Cierre del cursor (liberación de memoria)
CLOSE C_PELICULAS_POR_PAIS;

EXCEPTION
   WHEN NO_DATA_FOUND THEN
     DBMS_OUTPUT.PUT_LINE( 'No se ha encontrado ninguna fila con el país:
'|| pais_busqueda) ;
   WHEN OTHERS THEN
     DBMS_OUTPUT.PUT_LINE( 'El número del error es: '||
TO_CHAR( SQLCODE )) ;
     DBMS_OUTPUT.PUT_LINE( 'correspondiente a: '||
TO_CHAR( SQLERRM )) ;
END;
/
```

Resultados de la consulta:

```
Título de la película: BIENVENIDOS AL NORTE
Fecha de estreno     : 27/02/08
Director             : BOON DANY
Apellido actor       : BOON
Nombre actor         : DANY
Fecha de nacimiento  : 26/06/66
Número de películas  : 23
Presupuesto          : 11
Número de entradas   : 21000000
---------------------------------
Título de la película: NIKITA
Fecha de estreno     : 21/02/90
Director             : BESSON LUC
Apellido actor       : PARILLAUD
Nombre actor         : ANNE
Fecha de nacimiento  : 06/05/60
Número de películas  : 35
Presupuesto          : 7,6
Número de entradas   : 3787845
---------------------------------
Título de la película: NIKITA
Fecha de estreno     : 21/02/90
Director             : BESSON LUC
Apellido actor       : RENO
Nombre actor         : JEAN
Fecha de nacimiento  : 30/06/48
Número de películas  : 75
Presupuesto          : 7,6
Número de entradas   : 3787845
---------------------------------
Título de la película: SUBWAY
Fecha de estreno     : 10/04/85
Director             : BESSON LUC
Apellido actor       : ADJANI
Nombre actor         : ISABELLE
Fecha de nacimiento  : 27/06/55
Número de películas  : 42
Presupuesto          : 2,6
Número de entradas   : 2917562
---------------------------------
Título de la película: SUBWAY
Fecha de estreno     : 10/04/85
```

```
Director             : BESSON LUC
Apellido actor       : BOHRINGER
Nombre actor         : RICHARD
Fecha de nacimiento  : 16/06/42
Número de películas  : 132
Presupuesto          : 2,6
Número de entradas   : 2917562
---------------------------------
Título de la película: SUBWAY
Fecha de estreno     : 10/04/85
Director             : BESSON LUC
Apellido actor       : GALABRU
Nombre actor         : MICHEL
Fecha de nacimiento  : 27/10/22
Número de películas  : 277
Presupuesto          : 2,6
Número de entradas   : 2917562
```

Los bucles FOR, WHILE, LOOP y la estructura condicional CASE

1. El WHILE

El WHILE permite repetir un trozo de código mientras la condición que se comprueba al principio sea cierta. Si la condición es falsa, se sale directamente del bucle sin ejecutar el código.

Por ejemplo:

```
SET SERVEROUTPUT ON
DECLARE

-- declaración del cursor C_PELICULAS_POR_PAIS

CURSOR C_PELICULAS_POR_PAIS (PAISB IN VARCHAR2) IS
SELECT PELICULAS.IDENT_PELICULAS,PELICULAS.TITULO,
       PELICULAS.FECHA_ESTRENO,
       DIRECTOR.APELLIDO||' '||DIRECTOR.NOMBRE DIRECTOR, ACTOR.APELLIDO
       APELLIDO,
       ACTOR.NOMBRE NOMBRE, ACTOR.FECHA_NACIMIENTO,
       ACTOR.NUM_PELICULAS,ESTA.PRESUPUESTO, ESTA.NUM_ENTRADAS_ESPANA
       ENTRADAS

FROM   PELICULAS PELICULAS, DIRECTOR DIRECTOR, CASTING CAST,
       ACTOR ACTOR, ESTADISTICA ESTA, PAIS PAIS
WHERE
   PELICULAS.IDENT_DIRECTOR  = DIRECTOR.IDENT_DIRECTOR AND
   PELICULAS.IDENT_PELICULAS = CAST.IDENT_PELICULAS AND
   PELICULAS.IDENT_PELICULAS = ESTA.IDENT_PELICULAS AND
   CAST.IDENT_ACTOR          = ACTOR.IDENT_ACTOR AND
   PAIS.IDENT_PAIS           = ACTOR.NACIONALIDAD AND
   PAIS.DESCRIPCION          = PAISB
ORDER BY PELICULAS.IDENT_PELICULAS;

/* declaración de las variables que reciben datos
   ------------------------------------  */
ident_pelicula number := 0;
titulo_pelicula VARCHAR2(100);
fecha_pelicula DATE;
director VARCHAR2(100);
apellido_actor VARCHAR2(100);
nombre_actor VARCHAR2(100);
fecha_nacimiento DATE;
numero_peliculas NUMBER(8);
presupuesto DECIMAL(10,2);
numero_peliculas NUMBER(8);

/* declaración de otras variables
 ------------------------------------ */
pais_busqueda VARCHAR2(100) := 'FRANCIA';


BEGIN
-- Apertura
OPEN C_PELICULAS_POR_PAIS(pais_busqueda);
-- Lectura del primer elemento
FETCH C_PELICULAS_POR_PAIS INTO
    ident_pelicula, titulo_pelicula,fecha_pelicula,
    director,apellido_actor,
    nombre_actor ,fecha_nacimiento ,
    numero_peliculas,presupuesto,numero_peliculas;
-- Bucle de lectura mientras el identificador de la película sea < 3
WHILE ident_pelicula < 3
LOOP
```

```
-- Visualización de los elementos recuperados
      DBMS_OUTPUT.PUT_LINE( 'Identificador película: '||ident_pelicula);
      DBMS_OUTPUT.PUT_LINE( 'Título de la película : '||titulo_pelicula);
      DBMS_OUTPUT.PUT_LINE( 'Fecha de estreno      : '||fecha_pelicula);
      DBMS_OUTPUT.PUT_LINE( 'Director              : '||director);
      DBMS_OUTPUT.PUT_LINE( 'Apellido actor        : '||apellido_actor);
      DBMS_OUTPUT.PUT_LINE( 'Nombre actor          : '||nombre_actor);
      DBMS_OUTPUT.PUT_LINE( 'Fecha nacimiento      : '||fecha_nacimiento);
      DBMS_OUTPUT.PUT_LINE( 'Número de películas   : '||numero_peliculas);
      DBMS_OUTPUT.PUT_LINE( '------------------------------- ');
-- Lectura de los siguientes elementos
   FETCH C_PELICULAS_POR_PAIS INTO
       ident_pelicula, titulo_pelicula,fecha_pelicula,
       director,apellido_actor,
       nombre_actor ,fecha_nacimiento ,
       numero_peliculas,presupuesto,numero_peliculas;
   Exit When C_PELICULAS_POR_PAIS%NOTFOUND;
END LOOP;
-- Cierre del cursor (liberación de memoria)
CLOSE C_PELICULAS_POR_PAIS;
END;
/
```

2. El FOR

Como el WHILE, el FOR permite realizar bucles indicando desde el inicio el número de veces que se ejecuta el código.

Hay que indicar la variable que se comprueba, el valor de inicio y el valor final.

Ejemplo:

```
SET SERVEROUTPUT ON
DECLARE

-- declaración del C_PELICULAS_POR_PAIS

CURSOR C_PELICULAS_POR_PAIS (PAISB IN VARCHAR2) IS
SELECT PELICULAS.IDENT_PELICULAS,PELICULAS.TITULO,
PELICULAS.FECHA_ESTRENO,
       DIRECTOR.APELLIDO||' '||DIRECTOR.NOMBRE DIRECTOR, ACTOR.APELLIDO
       APELLIDO,
       ACTOR.NOMBRE NOMBRE, ACTOR.FECHA_NACIMIENTO,
       ACTOR.NUM_PELICULAS,ESTA.PRESUPUESTO, ESTA.NUM_ENTRADAS_ESPANA
       ENTRADAS

FROM   PELICULAS PELICULAS, DIRECTOR DIRECTOR, CASTING CAST,
       ACTOR ACTOR, ESTADISTICA ESTA, PAIS PAIS
WHERE
   PELICULAS.IDENT_DIRECTOR  = DIRECTOR.IDENT_DIRECTOR AND
   PELICULAS.IDENT_PELICULAS = CAST.IDENT_PELICULAS AND
   PELICULAS.IDENT_PELICULAS = ESTA.IDENT_PELICULAS AND
   CAST.IDENT_ACTOR          = ACTOR.IDENT_ACTOR AND
   PAIS.IDENT_PAIS           = ACTOR.NACIONALIDAD AND
   PAIS.DESCRIPCION          = PAISB
ORDER BY PELICULAS.IDENT_PELICULAS;

/* declaración de las variables que reciben datos
   ----------------------------------  */
ident_pelicula number;
titulo_pelicula VARCHAR2(100);
fecha_pelicula DATE;
director VARCHAR2(100);
apellido_actor VARCHAR2(100);
nombre_actor VARCHAR2(100);
fecha_nacimiento DATE;
numero_peliculas NUMBER(8);
presupuesto DECIMAL(10,2);
numero_peliculas NUMBER(8);
```

```
/* declaración de otras variables
 ---------------------------------- */
pais_busqueda VARCHAR2(100) := 'FRANCIA';
num_lectura number := 0;

BEGIN
-- Apertura
OPEN C_PELICULAS_POR_PAIS(pais_busqueda);

-- Lectura de los cuatro primeros elementos
FOR num_lectura in 1..4
LOOP
   FETCH C_PELICULAS_POR_PAIS INTO
       ident_pelicula, titulo_pelicula,fecha_pelicula,
       director,apellido_actor,
       nombre_actor ,fecha_nacimiento ,
       numero_peliculas,presupuesto,numero_peliculas;
   Exit When C_PELICULAS_POR_PAIS%NOTFOUND;
-- Visualización de los elementos recuperados
     DBMS_OUTPUT.PUT_LINE( 'Identificador película: '||ident_pelicula);
     DBMS_OUTPUT.PUT_LINE( 'Título de la película : '||titulo_pelicula);
     DBMS_OUTPUT.PUT_LINE( 'Fecha de estreno      : '||fecha_pelicula);
     DBMS_OUTPUT.PUT_LINE( 'Director              : '||director);
     DBMS_OUTPUT.PUT_LINE( 'Apellido actor        : '||apellido_actor);
     DBMS_OUTPUT.PUT_LINE( 'Nombre actor          : '||nombre_actor);
     DBMS_OUTPUT.PUT_LINE( 'Fecha nacimiento      : '||fecha_nacimiento);
     DBMS_OUTPUT.PUT_LINE( 'Número de películas   : '||numero_peliculas);
     DBMS_OUTPUT.PUT_LINE( '-------------------------------- ');
END LOOP;
-- Cierre del cursor (liberación de memoria)
CLOSE C_PELICULAS_POR_PAIS;
END;
/
```

Observe que se conserva la comprobación de si se han encontrado filas o no (Exit When C_PELICULAS_POR_PAIS%NOTFOUND;) para salir del bucle si se llega al final de la lectura antes de haber hecho las cuatro lecturas.

Resultado obtenido (sólo se muestran los cuatro primeros elementos):

```
Identificador película: 1
Título de la película : SUBWAY
Fecha de estreno      : 10/04/85
Director              : BESSON LUC
Apellido actor        : RENO
Nombre actor          : JEAN
Fecha de nacimiento   : 30/06/48
Número de películas   : 75
--------------------------------
Identificador película: 1
Título de la película : SUBWAY
Fecha de estreno      : 10/04/85
Director              : BESSON LUC
Apellido actor        : GALABRU
Nombre actor          : MICHEL
Fecha de nacimiento   : 27/10/22
Número de películas   : 277
--------------------------------
Identificador película: 1
Título de la película : SUBWAY
Fecha de estreno      : 10/04/85
Director              : BESSON LUC
Apellido actor        : BOHRINGER
Nombre actor          : RICHARD
Fecha de nacimiento   : 16/06/42
Número de películas   : 132
```

```
--------------------------------
Identificador película: 1
Título de la película : SUBWAY
Fecha de estreno      : 10/04/85
Director              : BESSON LUC
Apellido actor        : LAMBERT
Nombre actor          : CHRISTOPHE
Fecha de nacimiento   : 29/03/57
Número de películas   : 64
--------------------------------

Procedimiento PL/SQL terminado correctamente
```

3. El LOOP

El bucle LOOP no tiene condición de salida al inicio, sino que hay que poner la palabra « exit » en el interior del bucle para salir.

Lo hemos visto anteriormente con la condición de salida del FETCH en una lectura de cursor:

```
... ...
BEGIN
-- Apertura
OPEN C_PELICULAS_POR_PAIS(pais_busqueda);
-- Bucle de lectura
LOOP
-- Recuperación de los elementos fila por fila
   FETCH C_PELICULAS_POR_PAIS INTO
       titulo_pelicula,fecha_pelicula,
       director,apellido_actor,
       nombre_actor ,fecha_nacimiento ,
       numero_peliculas,presupuesto,numero_peliculas;
   Exit When C_PELICULAS_POR_PAIS%NOTFOUND;
-- Visualización de los elementos recuperados
   DBMS_OUTPUT.PUT_LINE( 'Título de la película: '||titulo_pelicula);
   DBMS_OUTPUT.PUT_LINE( 'Fecha de estreno     : '||fecha_pelicula);
   DBMS_OUTPUT.PUT_LINE( 'Director             : '||director);
   DBMS_OUTPUT.PUT_LINE( 'Apellido actor       : '||apellido_actor);
   DBMS_OUTPUT.PUT_LINE( 'Nombre actor         : '||nombre_actor);
   DBMS_OUTPUT.PUT_LINE( 'Fecha de nacimiento  : '||fecha_nacimiento);
   DBMS_OUTPUT.PUT_LINE( 'Número de películas  : '||numero_peliculas);
   DBMS_OUTPUT.PUT_LINE( 'Presupuesto          : '||presupuesto);
   DBMS_OUTPUT.PUT_LINE( 'Número de entradas   : '||numero_entradas);
   DBMS_OUTPUT.PUT_LINE( '-------------------------------- ');
END LOOP;
... ...
```

Si se quiere salir una vez que el identificador de la película sea superior a 3:

```
exit when ident_pelicula > 3;
```

4. El CASE

CASE permite comprobar el valor de una variable y, en función del mismo, realizar una acción u otra. Tan sólo se comprueba una acción, cada condición WHEN es exclusiva.

Por ejemplo, se puede comprobar el valor del PAIS en cada fila en lugar de añadir un WHERE en el cursor. De este modo se leen todas las filas de la tabla pero sólo se muestran las que corresponden al país buscado.

Ejemplo:

```
SET SERVEROUTPUT ON
DECLARE

-- declaración del cursor C_PELICULAS_POR_PAIS
```

```
CURSOR C_PELICULAS_POR_PAIS IS
SELECT PELICULAS.TITULO, PAIS.DESCRIPCION

FROM   PELICULAS PELICULAS, DIRECTOR DIRECTOR, CASTING CAST,
       ACTOR ACTOR, ESTADISTICA ESTA, PAIS PAIS
WHERE
   PELICULAS.IDENT_DIRECTOR  = DIRECTOR.IDENT_DIRECTOR AND
   PELICULAS.IDENT_PELICULAS = CAST.IDENT_PELICULAS AND
   PELICULAS.IDENT_PELICULAS = ESTA.IDENT_PELICULAS AND
   CAST.IDENT_ACTOR          = ACTOR.IDENT_ACTOR AND
   PAIS.IDENT_PAIS           = ACTOR.NACIONALIDAD
ORDER BY PELICULAS.TITULO;

/* declaración de variables que reciben datos
   -------------------------------------  */
titulo_pelicula VARCHAR2(100);
descripcion_pais PAIS.DESCRIPCION%type;

/* declaración de otras variables
 ---------------------------------- */
num_filas_francia number := 0;
num_filas_argelia number := 0;
num_filas_otro    number := 0;

BEGIN
-- Apertura
OPEN C_PELICULAS_POR_PAIS;
-- Bucle de lectura
LOOP
-- Recuperación de elementos fila a fila
   FETCH C_PELICULAS_POR_PAIS INTO
       titulo_pelicula,descripción_pais;
   Exit When C_PELICULAS_POR_PAIS%NOTFOUND;

   CASE DESCRIPCION_PAIS
      WHEN 'FRANCIA' THEN num_filas_francia := num_filas_francia + 1;
      WHEN 'ARGELIA' THEN num_filas_argelia := num_filas_argelia + 1;
      ELSE num_filas_otro := num_filas_otro + 1;
   END CASE;

END LOOP;
-- Cierre del cursor (liberación de memoria)
CLOSE C_PELICULAS_POR_PAIS;

 -- Visualización de los valores
DBMS_OUTPUT.PUT_LINE( 'Num de filas FRANCIA: '||num_filas_francia);
DBMS_OUTPUT.PUT_LINE( 'Num de filas ARGELIA: '||num_filas_argelia);
DBMS_OUTPUT.PUT_LINE( 'Num de filas OTRO   : '||num_filas_otro);

EXCEPTION
   WHEN NO_DATA_FOUND THEN
     DBMS_OUTPUT.PUT_LINE( 'No se ha encontrado ninguna fila en la
selección') ;
   WHEN OTHERS THEN
     DBMS_OUTPUT.PUT_LINE( 'El número de error es: '||
TO_CHAR( SQLCODE )) ;
     DBMS_OUTPUT.PUT_LINE( 'correspondiente a: '||
TO_CHAR( SQLERRM )) ;
END;
/
```

Resultados:

```
Num de filas FRANCIA: 8
Num de filas ARGELIA: 1
Num de filas OTRO   : 4

Procedimiento PL/SQL terminado correctamente
```

Las excepciones más utilizadas

Salvo la excepción « NOT_DATA_FOUND » que hemos visto en los ejemplos anteriores, existen multitud de excepciones. No vamos a citarlas todas en este libro, pero vamos a ver algunas que puede ser útiles.

CURSOR_ALREADY_OPEN: ya se ha realizado el open del cursor. Hay que cerrarlo antes de volver a abrirlo (sqlcode --> 06511)

INVALID_NUMBER: la variable utilizada no contiene un número válido (sqlcode --> 01722)

NOT_LOGGED_ON: el usuario no está conectado a la base de datos (sqlcode --> 01012)

TOO_MANY_ROWS: la selección devuelve varias filas, mientras el select sólo prevé una ocurrencia; se debe hacer un cursor (sqlcode --> 01422)

ZERO_DIVIDE: división por cero (sqlcode --> 01476)

Para tratar cualquier tipo de error, es preferible añadir siempre una comprobación de este tipo para visualizar el error.

```
WHEN OTHERS THEN
   DBMS_OUTPUT.PUT_LINE( ’El número de error es: ’||
TO_CHAR( SQLCODE )) ;
     DBMS_OUTPUT.PUT_LINE( ’correspondiente a: ’||
TO_CHAR( SQLERRM )) ;
```

Creación de un procedimiento almacenado

Cuando se quiere compartir un trozo de código realizado en PL/SQL, se puede grabar en la base de datos y así el resto de programadores pueden acceder a él. Un procedimiento almacenado es un bloque de código compilado y almacenado en la base de datos. Basta con llamarlo por su nombre para ejecutarlo.

La principal ventaja del procedimiento almacenado, es que está guardado en un formato « ejecutable », el servidor de la base de datos no va a interpretar los comandos sino que los ejecuta directamente, con la ganancia de tiempo considerable respecto a la ejecución de la misma consulta desde un programa.

Otra ventaja del procedimiento almacenado es que se le pueden pasar parámetros.

Sintaxis:

```
CREATE OR REPLACE PROCEDURE <nombre procedimiento>
 [(<variable entrada 1> IN <formato>,
   <variable entrada 2> IN <formato>,
   ... ...
   <variable salida> OUT <formato>)]
 IS

BEGIN

... ...

[EXCEPTION
... ...
]
END;
```

Por ejemplo, el siguiente procedimiento calcula la edad de un actor a partir de su año de nacimiento.

```
CREATE OR REPLACE PROCEDURE CALCULO_EDAD_ACTOR
 (FECHA_NACIMIENTO IN DATE, EDAD_ACTOR OUT NUMBER)
 IS

BEGIN

SELECT (SYSDATE - FECHA_NACIMIENTO)/365 INTO EDAD_ACTOR FROM DUAL;

EXCEPTION
   WHEN OTHERS THEN
     DBMS_OUTPUT.PUT_LINE( 'El número de error es: '||
TO_CHAR( SQLCODE )) ;
     DBMS_OUTPUT.PUT_LINE( 'correspondiente a: '||
TO_CHAR( SQLERRM )) ;
END;
/
```

En caso de que el SGBDR le indique problemas de compilación, es posible visualizar el último error encontrado con el comando SHOW ERRORS.

Añadimos la llamada a este procedimiento (CALCULO_EDAD_ACTOR) en el ejemplo anterior:

```
DECLARE

-- declaración del cursor C_PELICULAS_POR_PAIS

CURSOR C_PELICULAS_POR_PAIS (PAISB IN VARCHAR2) IS
SELECT PELICULAS.TITULO, PELICULAS.FECHA_ESTRENO,
       DIRECTOR.APELLIDO||' '||DIRECTOR.NOMBRE DIRECTOR, ACTOR.APELLIDO APELLIDO,
       ACTOR.NOMBRE NOMBRE, ACTOR.FECHA_NACIMIENTO,
       ACTOR.NUM_PELICULAS,ESTA.PRESUPUESTO, ESTA.NUM_ENTRADAS_ESPANA
```

```
ENTRADAS

FROM   PELICULAS PELICULAS, DIRECTOR DIRECTOR, CASTING CAST,
       ACTOR ACTOR, ESTADISTICA ESTA, PAIS PAIS
WHERE
   PELICULAS.IDENT_DIRECTOR  = DIRECTOR.IDENT_DIRECTOR AND
   PELICULAS.IDENT_PELICULAS = CAST.IDENT_PELICULAS AND
   PELICULAS.IDENT_PELICULAS = ESTA.IDENT_PELICULAS AND
   CAST.IDENT_ACTOR          = ACTOR.IDENT_ACTOR AND
   PAIS.IDENT_PAIS           = ACTOR.NACIONALIDAD AND
   PAIS.DESCRIPCION          = PAISB
ORDER BY PELICULAS.TITULO;

/* declaración de las variables que recuperan datos
   ------------------------------------  */
titulo_pelicula VARCHAR2(100);
fecha_pelicula DATE;
director VARCHAR2(100);
apellido_actor VARCHAR2(100);
nombre_actor VARCHAR2(100);
fecha_nacimiento DATE;
numero_peliculas NUMBER(8);
presupuesto DECIMAL(10,2);
numero_peliculas NUMBER(8);

/* declaración de otras variables
 ------------------------------------ */
pais_busqueda VARCHAR2(100) := 'FRANCIA';
edad_actor NUMBER(5);

BEGIN
-- Apertura
OPEN C_PELICULAS_POR_PAIS(pais_busqueda);
-- Bucle de lectura
LOOP
-- Recuperación de los elementos fila a fila
   FETCH C_PELICULAS_POR_PAIS INTO
       titulo_pelicula,fecha_pelicula,
       director,apellido_actor,
       nombre_actor ,fecha_nacimiento ,
       numero_peliculas,presupuesto,numero_peliculas;
   Exit When C_PELICULAS_POR_PAIS%NOTFOUND;
CALCULO_EDAD_ACTOR(fecha_nacimiento, edad_actor);

-- Visualización de los elementos recuperados
   DBMS_OUTPUT.PUT_LINE( 'Título de la película: '||titulo_pelicula);
   DBMS_OUTPUT.PUT_LINE( 'Fecha de estreno     : '||fecha_pelicula);
   DBMS_OUTPUT.PUT_LINE( 'Director             : '||director);
   DBMS_OUTPUT.PUT_LINE( 'Apellido actor       : '||apellido_actor);
   DBMS_OUTPUT.PUT_LINE( 'Nombre actor         : '||nombre_actor);
   DBMS_OUTPUT.PUT_LINE( 'Fecha de nacimiento  : '||fecha_nacimiento);
   DBMS_OUTPUT.PUT_LINE( 'Edad del actor       : '||edad_actor);
   DBMS_OUTPUT.PUT_LINE( 'Número de películas  : '||numero_peliculas);
   DBMS_OUTPUT.PUT_LINE( 'Presupuesto          : '||presupuesto);
   DBMS_OUTPUT.PUT_LINE( 'Número de entradas   : '||numero_entradas);
   DBMS_OUTPUT.PUT_LINE( '--------------------------------- ');
END LOOP;
-- Cierre del cursor (liberación de memoria)
CLOSE C_PELICULAS_POR_PAIS;

EXCEPTION
   WHEN NO_DATA_FOUND THEN
     DBMS_OUTPUT.PUT_LINE( 'No se ha encontrado ninguna fila
con el país: '||pais_busqueda) ;
   WHEN OTHERS THEN
     DBMS_OUTPUT.PUT_LINE( 'El número de error es: '||
TO_CHAR( SQLCODE )) ;
     DBMS_OUTPUT.PUT_LINE( 'correspondiente a: '||
TO_CHAR( SQLERRM )) ;
END;
```

```
/
```

Resultados:

```
Título de la película: BIENVENIDOS AL NORTE
Fecha de estreno     : 27/02/08
Director             : BOON DANY
Apellido actor       : BOON
Nombre actor         : DANY
Fecha de nacimiento  : 26/06/66
Edad del actor       : 45
Número de películas  : 23
Presupuesto          : 11
Número de entradas   : 21000000
---------------------------------
Título de la película: NIKITA
Fecha de estreno     : 21/02/90
Director             : BESSON LUC
Apellido actor       : PARILLAUD
Nombre actor         : ANNE
Fecha de nacimiento  : 06/05/60
Edad del actor       : 51
Número de películas  : 35
Presupuesto          : 7,6
Número de entradas   : 3787845
---------------------------------
Título de la película: NIKITA
Fecha de estreno     : 21/02/90
Director             : BESSON LUC
Apellido actor       : RENO
Nombre actor         : JEAN
Fecha de nacimiento  : 30/06/48
Edad del actor       : 63
Número de películas  : 75
Presupuesto          : 7,6
Número de entradas   : 3787845
---------------------------------
Título de la película: SUBWAY
Fecha de estreno     : 10/04/85
Director             : BESSON LUC
Apellido actor       : ADJANI
Nombre actor         : ISABELLE
Fecha de nacimiento  : 27/06/55
Edad del actor       : 56
Número de películas  : 42
Presupuesto          : 2,6
Número de entradas   : 2917562
---------------------------------
Título de la película: SUBWAY
Fecha de estreno     : 10/04/85
Director             : BESSON LUC
Apellido actor       : BOHRINGER
Nombre actor         : RICHARD
Fecha de nacimiento  : 16/06/42
Edad del actor       : 69
Número de películas  : 132
Presupuesto          : 2,6
Número de entradas   : 2917562
```

Creación de una función almacenada

En el mismo ejemplo, también es posible crear una función en lugar de un procedimiento. ¿Cuál es la diferencia entre una función y un procedimiento? Que la primera devuelve un valor.

Sintaxis:

```
CREATE OR REPLACE FUNCTION <nombre función>
 [(<variable entrada 1> IN <formato>,
   <variable entrada 2> IN <formato>,
   ... ... )]
   RETURN <formato>
 IS
<variable salida> <formato>;
BEGIN

... ...

[EXCEPTION
... ...
]
END;
```

Por ejemplo, la siguiente función devuelve la edad del actor calculada a partir de su año de nacimiento.

```
CREATE OR REPLACE FUNCTION CALCULO_EDAD_ACTOR
 (FECHA_NACIMIENTO IN DATE) RETURN NUMBER
 IS
 EDAD_ACTOR NUMBER(5);

BEGIN

SELECT (SYSDATE - FECHA_NACIMIENTO)/365 INTO EDAD_ACTOR FROM DUAL;
RETURN  (EDAD_ACTOR);

EXCEPTION
   WHEN OTHERS THEN
     DBMS_OUTPUT.PUT_LINE( 'El número de error es: '||
TO_CHAR( SQLCODE )) ;
     DBMS_OUTPUT.PUT_LINE( 'correspondiente a: '||
TO_CHAR( SQLERRM )) ;
END;
/
```

Esta función se puede utilizar directamente en un procedimiento como por ejemplo:

```
   DBMS_OUTPUT.PUT_LINE('Fecha de nacimiento: '||fecha_nacimiento);
   DBMS_OUTPUT.PUT_LINE('Edad del actor     : '||
CALCULO_EDAD_ACTOR(fecha_nacimiento));
   DBMS_OUTPUT.PUT_LINE('Número de películas: '||numero_peliculas);
```

También se puede utilizar directamente en un comando SELECT del siguiente modo:

```
SELECT PELICULAS.TITULO, PELICULAS.FECHA_ESTRENO,
       DIRECTOR.APELLIDO||' '||DIRECTOR.NOMBRE DIRECTOR, ACTOR.APELLIDO
APELLIDO,
       ACTOR.NOMBRE NOMBRE, ACTOR.FECHA_NACIMIENTO,
       ACTOR.NUM_PELICULAS,ESTA.PRESUPUESTO, ESTA.NUM_ENTRADAS_ESPANA
ENTRADAS,
       CALCULO_EDAD_ACTOR(ACTOR.FECHA_NACIMIENTO)
FROM   PELICULAS PELICULAS, DIRECTOR DIRECTOR, CASTING CAST,
       ACTOR ACTOR, ESTADISTICA ESTA, PAIS PAIS
WHERE
   PELICULAS.IDENT_DIRECTOR  = DIRECTOR.IDENT_DIRECTOR AND
```

```
   PELICULAS.IDENT_PELICULAS = CAST.IDENT_PELICULAS AND
   PELICULAS.IDENT_PELICULAS = ESTA.IDENT_PELICULAS AND
   CAST.IDENT_ACTOR          = ACTOR.IDENT_ACTOR AND
   PAIS.IDENT_PAIS           = ACTOR.NACIONALIDAD AND
   PAIS.DESCRIPCION          = 'FRANCIA'
ORDER BY PELICULAS.TITULO;
```

Los packages

La denominación « package » significa que se agrupan bajo un mismo nombre los procedimientos y funciones sobre el mismo tema, y así podemos crear verdaderas aplicaciones.

En un paquete, podemos tener declaraciones de variables públicas o privadas, así como funciones y procedimientos privados que no se pueden ver desde fuera.

En un paquete, hay que crear una zona de declaraciones y una zona donde están las funciones y los procedimientos.

En la zona de declaraciones, se listarán los procedimientos y funciones que se describen en la otra zona. Todas las funciones que estén declaradas aquí serán « públicas ». Las variables funcionan del mismo modo, si están en la zona de declaraciones, son « públicas ».

Sintaxis:

```
CREATE OR REPLACE PACKAGE <nombre paquete> IS
      PROCEDURE <nombre procedimiento 1>;
      FUNCTION <nombre función 1> (<variable 1> IN <formato>) RETURN
<formato>; END;
/
CREATE OR REPLACE PACKAGE BODY <nombre paquete> IS

   FUNCTION <función 1>
     ... ...
   END;

   PROCEDURE <procedimiento 1> IS
     ... ...
   END;
END;
/
```

Podemos agrupar la función CALCULO_EDAD_ACTOR y el procedimiento de visualización de las películas utilizados anteriormente (al que se le puede llamar LISTA_PELICULAS) y crear un package llamado VISUALIZACION_PELICULAS.

```
CREATE OR REPLACE PACKAGE VISUALIZACION_PELICULAS IS
      PROCEDURE LISTA_PELICULAS;
      FUNCTION CALCULO_EDAD_ACTOR (FECHA_NACIMIENTO IN DATE) RETURN
NUMBER;
END;
/
CREATE OR REPLACE PACKAGE BODY VISUALIZACION_PELICULAS IS

   FUNCTION CALCULO_EDAD_ACTOR
    (FECHA_NACIMIENTO IN DATE) RETURN NUMBER
    IS
      EDAD_ACTOR NUMBER(5);

   BEGIN

   SELECT (SYSDATE - FECHA_NACIMIENTO)/365 INTO EDAD_ACTOR FROM DUAL;
   RETURN  (EDAD_ACTOR);

   EXCEPTION
      WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE( 'El número de error es: '||
TO_CHAR( SQLCODE )) ;
      DBMS_OUTPUT.PUT_LINE( 'correspondiente a: '||TO_CHAR( SQLERRM )) ;
   END;

   PROCEDURE LISTA_PELICULAS IS

   -- declaración del cursor C_PELICULAS_POR_PAIS
```

```
   CURSOR C_PELICULAS_POR_PAIS (PAISB IN VARCHAR2) IS
   SELECT PELICULAS.TITULO, PELICULAS.FECHA_ESTRENO,
          DIRECTOR.APELLIDO||' '||DIRECTOR.NOMBRE DIRECTOR, ACTOR.APELLIDO
         APELLIDO,
         ACTOR.NOMBRE NOMBRE, ACTOR.FECHA_NACIMIENTO,
         ACTOR.NUM_PELICULAS,ESTA.PRESUPUESTO,
         ESTA.NUM_ENTRADAS_ESPANA ENTRADAS,
         CALCULO_EDAD_ACTOR(ACTOR.FECHA_NACIMIENTO)
   FROM  PELICULAS PELICULAS, DIRECTOR DIRECTOR, CASTING CAST,
         ACTOR ACTOR, ESTADISTICA ESTA, PAIS PAIS
   WHERE
      PELICULAS.IDENT_DIRECTOR  = DIRECTOR.IDENT_DIRECTOR AND
      PELICULAS.IDENT_PELICULAS = CAST.IDENT_PELICULAS AND
      PELICULAS.IDENT_PELICULAS = ESTA.IDENT_PELICULAS AND
      CAST.IDENT_ACTOR          = ACTOR.IDENT_ACTOR AND
      PAIS.IDENT_PAIS           = ACTOR.NACIONALIDAD AND
      PAIS.DESCRIPCION          = PAISB
   ORDER BY PELICULAS.TITULO;

   /* declaración de las variables que recuperan datos
      ------------------------------------  */
   titulo_pelicula VARCHAR2(100);
   fecha_pelicula DATE;
   director VARCHAR2(100);
   apellido_actor VARCHAR2(100);
   nombre_actor VARCHAR2(100);
   fecha_nacimiento DATE;
   numero_peliculas NUMBER(8);
   presupuesto DECIMAL(10,2);
   numero_peliculas NUMBER(8);

   /* declaración de otras variables
    ------------------------------------ */
   pais_busqueda VARCHAR2(100) := 'FRANCIA';
   edad_actor NUMBER(5);

   BEGIN
   -- Apertura
   OPEN C_PELICULAS_POR_PAIS(pais_busqueda);
   -- Bucle de lectura
   LOOP
   -- Recuperación de los elementos fila a fila
      FETCH C_PELICULAS_POR_PAIS INTO
          titulo_pelicula,fecha_pelicula,
          director,apellido_actor,
          nombre_actor ,fecha_nacimiento ,
          numero_peliculas,presupuesto,numero_peliculas,edad_actor;
      Exit When C_PELICULAS_POR_PAIS%NOTFOUND;
   -- Visualización de los elementos recuperados
   DBMS_OUTPUT.PUT_LINE( 'Título de la película: '||titulo_pelicula);
   DBMS_OUTPUT.PUT_LINE( 'Fecha de estreno     : '||fecha_pelicula);
   DBMS_OUTPUT.PUT_LINE( 'Director             : '||director);
   DBMS_OUTPUT.PUT_LINE( 'Apellido actor       : '||apellido_actor);
   DBMS_OUTPUT.PUT_LINE( 'Nombre actor         : '||nombre_actor);
   DBMS_OUTPUT.PUT_LINE( 'Fecha de nacimiento  : '||fecha_nacimiento);
   DBMS_OUTPUT.PUT_LINE( 'Edad del actor       : '||edad_actor);
   DBMS_OUTPUT.PUT_LINE( 'Número de películas  : '||numero_peliculas);
   DBMS_OUTPUT.PUT_LINE( 'Presupuesto          : '||presupuesto);
   DBMS_OUTPUT.PUT_LINE( 'Número de entradas   : '||numero_entradas);
   DBMS_OUTPUT.PUT_LINE( '-------------------------------- ');
END LOOP;
   -- Cierre del cursor (liberación de memoria)
   CLOSE C_PELICULAS_POR_PAIS;

   EXCEPTION
      WHEN NO_DATA_FOUND THEN
       DBMS_OUTPUT.PUT_LINE( 'No se ha encontrado ninguna fila
con el país:'|| pais_busqueda) ;
      WHEN OTHERS THEN
```

```
        DBMS_OUTPUT.PUT_LINE( 'El número de error es: '||
TO_CHAR( SQLCODE )) ;
        DBMS_OUTPUT.PUT_LINE( 'correspondiente a: '||
TO_CHAR( SQLERRM )) ;
   END;
END;
/
```

Para poder utilizar las funciones del paquete, hay que utilizar la siguiente sintaxis: <nombredelpaquete.nombrefuncion>.

```
SELECT PELICULAS.TITULO, PELICULAS.FECHA_ESTRENO,
       DIRECTOR.APELLIDO||' '||DIRECTOR.NOMBRE DIRECTOR, ACTOR.APELLIDO
       APELLIDO,
       ACTOR.NOMBRE NOMBRE, ACTOR.FECHA_NACIMIENTO,
       ACTOR.NUM_PELICULAS,ESTA.PRESUPUESTO, ESTA.NUM_ENTRADAS_ESPANA
       ENTRADAS,
       VISUALIZACION_PELICULAS.CALCULO_EDAD_ACTOR
       (ACTOR.FECHA_NACIMIENTO)
FROM   PELICULAS PELICULAS, DIRECTOR DIRECTOR, CASTING CAST,
       ACTOR ACTOR, ESTADISTICA ESTA, PAIS PAIS
WHERE
   PELICULAS.IDENT_DIRECTOR  = DIRECTOR.IDENT_DIRECTOR AND
   PELICULAS.IDENT_PELICULAS = CAST.IDENT_PELICULAS AND
   PELICULAS.IDENT_PELICULAS = ESTA.IDENT_PELICULAS AND
   CAST.IDENT_ACTOR          = ACTOR.IDENT_ACTOR AND
   PAIS.IDENT_PAIS           = ACTOR.NACIONALIDAD AND
   PAIS.DESCRIPCION          = 'FRANCIA'
ORDER BY PELICULAS.TITULO;
```

Compilación de un procedimiento, de una función o de un paquete

Sintaxis:

```
ALTER <’PROCEDURE’ o ’FUNCTION’ o ’PACKAGE’> <Nombre procedimiento,
función o paquete> COMPILE;
```

Ejemplo:

```
ALTER FUNCTION CALCULO_EDAD_ACTOR COMPILE;
ALTER PROCEDURE LISTA_PELICULAS COMPILE;
ALTER PACKAGE VISUALIZACION_PELICULAS COMPILE PACKAGE;
ALTER PACKAGE VISUALIZACION_PELICULAS COMPILE BODY;
- compila body y package
ALTER PACKAGE VISUALIZACION_PELICULAS COMPILE;
```

Eliminación de un procedimiento, de una función o de un paquete

Sintaxis:

```
DROP <’PROCEDIMIENTO’, ’FUNCIÓN’ o ’PAQUETE’> <Nombre procedimiento,
función o paquete>;
```

Ejemplo:

```
DROP FUNCTION CALCULO_EDAD_ACTOR;
DROP PROCEDURE LISTA_PELICULAS;
- eliminación de todo el paquete (cuerpo y declaración)
DROP PACKAGE VISUALIZACION_PELICULAS;
- eliminación del cuerpo del paquete
DROP PACKAGE BODY VISUALIZACION_PELICULAS;
```

Introducción

Los errores Oracle de acceso a los datos tienen el formato ORA-nnnnn, y existen miles. En este capítulo veremos aquellos que se producen con más frecuencia.

Existen otros errores Oracle y se dividen por tipos. Por ejemplo, los errores específicos de PLS/SQL son los PLS-nnnnn, los errores vinculados a las copias de seguridad son los RMAN-nnnnn, los errores de carga con SQL*Loader son los SQL*LOADERnnnnn, etc.

En Internet encontrará sitios que recogen todos los errores existentes. A continuación puede ver algunos:

- www.ora-code.com
- http://docs.oracle.com/cd/B28359_01/server.111/b28278/toc.htm
- www.ora-error.com

Sobre el acceso a los datos (LDD/LMD)

CÓDIGO DE ERROR	MENSAJE ORACLE	CAUSA DEL ERROR	SOLUCIÓN
ORA-00001	unique constraint (string.string) violated	Un UPDATE o un INSERT provoca una clave duplicada.	Elimine la restricción UNIQUE en la clave, o corrija el INSERT o el UPDATE.
ORA-00051	timeout occurred while waiting for a resource	Ha sobrepasado el tiempo de espera desde la ejecución del comando. Una fila de una tabla está bloqueada por otro usuario y, pasado un tiempo de espera, Oracle devuelve este error.	Vuelva a ejecutar el comando más tarde.
ORA-00054	resource busy and acquire with NOWAIT specified	La tabla o las filas a las que se quiere acceder están reservadas (bloqueadas) por otro usuario y el parámetro NOWAIT está activado. Esto significa que en este caso Oracle no espera.	Espere unos minutos y vuelva a ejecutar el comando o elimine el parámetro NOWAIT para que Oracle espere a que el recurso sea liberado.
ORA-00060	deadlock detected while waiting for resource	Intenta actualizar una fila que está siendo actualizada por otro usuario.	Una u otra sesión debe realizar un ROLLBACK o un COMMIT para liberar el recurso.
ORA-00100	no data found	El SELECT no ha devuelto ninguna fila.	Compruebe la cláusula WHERE y los datos de la tabla.
ORA-00900	invalid SQL statement	La sintaxis de su consulta o de su procedimiento almacenado no es correcta.	Corrija la sintaxis, compruebe sobretodo el formato de las fechas respecto a la variable NLS_DATE_ FORMAT declarada en la base de datos.
ORA-00900	to ORA-01499 3-7statement. ORDER BY cannot be used to create an ordered view or to insert in a certain order.	El comando SQL ORDER BY no está permitido con un CREATE VIEW o INSERT.	Compruebe la sintaxis del comando.
ORA-00901	invalid CREATE command	El comando CREATE no es correcto.	Corrija la sintaxis.
ORA-00902	invalid datatype	En la creación o modificación de una tabla no ha utilizado un tipo de columna Oracle correcto.	Compruebe los formatos asignados a las columnas (CHAR, NUMBER, etc.). Compruebe el contenido de sus

			variables en un UPDATE o en un INSERT.
ORA-00903	invalid table name	Nombre de tabla incorrecto. Un nombre de tabla debe tener como máximo 30 caracteres y sólo pueden ser alfanuméricos. O la tabla no existe o no puede acceder con sus privilegios.	Compruebe el nombre de la tabla y/o pregunte al DBA si tiene acceso.
ORA-00904	string: invalid identifier	Nombre de columna incorrecto. El nombre de columna debe tener como máximo 30 caracteres y sólo pueden ser alfanuméricos.	Compruebe el nombre de la columna.
ORA-00905	missing keyword	Sintaxis errónea en el comando.	Compruebe la sintaxis del comando.
ORA-00906	missing left parenthesis	Falta un paréntesis a la izquierda	Compruebe la sintaxis del comando.
ORA-00907	missing right parenthesis	Falta un paréntesis a la derecha	Compruebe la sintaxis del comando.
ORA-00908	missing NULL keyword	Falta la palabra NULL en la consulta. Ejemplo: Select toto from table where toto is not;	Compruebe la sintaxis del comando.
ORA-00909	invalid number of arguments	Falta un parámetro en la función.	Compruebe la sintaxis de la función.
ORA-00910	specified length too long for its datatype	El tamaño para una columna sobrepasa el límite máximo autorizado.	Compruebe el tamaño máximo de las columnas por tipo de datos (VARCHAR, CHAR, etc.).
ORA-00911	invalid character	Se ha encontrado un carácter incorrecto en su consulta (por ejemplo un " o un guión (-)).	Compruebe la sintaxis del comando.
ORA-00913	too many values	Error en el número de argumentos en una consulta INSERT por ejemplo. Hay más parámetros que columnas en la tabla.	Compruebe la estructura de la tabla.
ORA-00914	missing ADD keyword	Falta la palabra clave ADD en el comando ALTER TABLE.	Compruebe la sintaxis del comando.
ORA-00917	missing comma	Falta la coma en la lista de valores. En un comando INSERT por ejemplo, los parámetros se deben indicar del	Compruebe la sintaxis del comando.

		siguiente modo: (C,D,E,F, ...).	
ORA-00918	column ambiguously defined	En la unión entre dos tablas, existe el mismo nombre de columna en cada tabla.	Añada un alias delante del nombre de la columna para indicar de qué tabla se trata: Tabla1.col1, Tabla2.col1
ORA-00919	invalid function	La función no existe.	Compruebe la sintaxis del comando.
ORA-00920	invalid relational operator	Operador incorrecto. Debe ser uno de los siguientes =, !=, ^=, <>, >, <, >=, <=,ALL, ANY, [NOT] BETWEEN, EXISTS, [NOT] IN, IS [NOT] NULL o [NOT]LIKE.	Compruebe la sintaxis del comando.
ORA-00921	unexpected end of SQL command	El comando SQL no está completo, faltan elementos al final.	Compruebe la sintaxis del comando.
ORA-00922	missing or invalid option	Se ha indicado una opción incorrecta en la creación o modificación de una columna de una tabla. Por ejemplo, poner la opción NOT NULL cuando hay valores NULL en la tabla.	Compruebe la sintaxis del comando.
ORA-00923	FROM keyword not found where expected	Falta la cláusula FROM en un SELECT.	Compruebe la sintaxis del comando.
ORA-00924	missing BY keyword	Falta la cláusula BY en un SELECT.	Compruebe la sintaxis del comando.
ORA-00925	missing INTO keyword	Falta la cláusula INTO en un SELECT.	Compruebe la sintaxis del comando.
ORA-00926	missing VALUES keyword	Falta el comando VALUES en un INSERT o UPDATE.	Compruebe la sintaxis del comando.
ORA-00927	missing equal sign	Falta el signo = en la consulta.	Compruebe la sintaxis del comando.
ORA-00932	inconsistent datatypes: expected string got string	Comando incompatible entre dos tipos diferentes. Por ejemplo, sumar un carácter a una fecha.	Compruebe la sintaxis del comando.
ORA-00933	SQL command not properly ended	El comando SQL utiliza una cláusula incorrecta. Por ejemplo, una cláusula ORDER BY en un comando CREATE VIEW o INSERT.	Compruebe la sintaxis del comando.
ORA-00934	group function is not	No se puede utilizar una	Añada el GROUP BY en

	allowed here	función del tipo AVG, COUNT, MAX, MIN, SUM, STDDEV o VARIANCE sin utilizar la cláusula GROUP BY.	la columna especificada.
ORA-00936	missing expression	El comando SELECT no está completo o hay un error de sintaxis.	Compruebe la sintaxis del comando.
ORA-00937	not a single-group group function	Una de las columnas está en un GROUP BY, por lo que se debe utilizar con una función del tipo AVG, COUNT, MAX, MIN, SUM, STDDEV o VARIANCE.	Compruebe la sintaxis del comando.
ORA-00938	not enough arguments for function	Está utilizando una función que requiere parámetros complementarios.	Compruebe la sintaxis del comando y consulte la documentación Oracle.
ORA-00939	too many arguments for function	Está utilizando una función que requiere menos argumentos.	Compruebe la sintaxis del comando y consulte la documentación Oracle.
ORA-00940	invalid ALTER command	En el comando ALTER está utilizando un comando desconocido o no permitido.	Compruebe la sintaxis del comando y consulte la documentación Oracle.
ORA-00941	missing cluster name	El nombre del cluster es incorrecto.	El nombre de cluster debe tener como máximo 30 caracteres y solo puede tener caracteres alfanuméricos. O el cluster no existe o no tiene privilegios para acceder.
ORA-00942	table or view does not exist	El nombre de la tabla o vista no existe.	La tabla no existe o no tiene privilegios para acceder.
ORA-00943	cluster does not exist	No existe el nombre del cluster.	El cluster no existe o no tiene privilegios para acceder.
ORA-00946	missing TO keyword	Ha olvidado el TO en el comando GRANT o la sintaxis es incorrecta.	Compruebe la sintaxis del comando y consulte la documentación Oracle.
ORA-00947	not enough values	Falta un valor en un comando INSERT o en un comando SELECT. Ha indicado más columnas que valores en la cláusula VALUE o en la cláusula INTO.	Compruebe que tiene el mismo número de columnas que de valores.
ORA-00950	invalid DROP option	Está utilizando el DROP	Compruebe la sintaxis

		sobre un elemento sobre el que no se puede aplicar. Se aplica en los tipos CLUSTER, DATABASE LINK, INDEX, ROLLBACK SEGMENT, SEQUENCE, SYNONYM, TABLE, TABLESPACE, o VIEW.	del comando y consulte la documentación Oracle.
ORA-00952	missing GROUP keyword	Falta la cláusula GROUP en el comando.	Compruebe la sintaxis del comando y consulte la documentación Oracle.
ORA-00953	missing or invalid index name	Falta el nombre del índice o este no es válido en una cláusula CREATE INDEX o DROP INDEX.	Compruebe la sintaxis del comando y consulte la documentación Oracle.
ORA-00955	name is already used by an existing object	Intenta crear un objeto (tabla, vista, índice, etc.) que ya existe en la base de datos.	Haga un SELECT en ALL_TABLES, ALL_VIEWS o en otra tabla del sistema para comprobar los elementos que existen en la base de datos.
ORA-00957	duplicate column name	El nombre de la columna en una tabla debe ser único.	Cambie el nombre de la columna.
ORA-00967	missing WHERE keyword	Ha olvidado el WHERE en el comando SELECT o la sintaxis es incorrecta.	Compruebe la sintaxis del comando y consulte la documentación Oracle.
ORA-00968	missing INDEX keyword	Ha olvidado el INDEX en el comando CREATE o la sintaxis es incorrecta.	Compruebe la sintaxis del comando y consulte la documentación Oracle.
ORA-00969	missing ON keyword	Ha olvidado el ON en el comando o la sintaxis es incorrecta.	Compruebe la sintaxis del comando y consulte la documentación Oracle.
ORA-00978	nested group function without GROUP BY	Está utilizando una función en una columna pero no hay cláusula GROUP BY asociada.	Compruebe la sintaxis del comando y consulte la documentación Oracle.
ORA-00979	not a GROUP BY expression	Está utilizando una columna en una cláusula GROUP BY. Esta columna no está asociada a una función vinculada al GROUP BY.	Añada una función en la columna de tipo AVG, COUNT, MAX, MIN, SUM, STDDEV, o VARIANCE.
ORA-00984	column not allowed here	Está utilizando una columna de forma incorrecta en una consulta.	Compruebe la sintaxis del comando y consulte la documentación Oracle.

ORA-00996	the concatenate operator is ||, not |	Para concatenar dos columnas, debe utilizar dos "pipe" y no solo uno.	Corrija el comando.
ORA-01002	fetch out of sequence	Ha abierto un cursor. Está realizando FETCH para recorrerlo. El cursor ha llegado al final y ha realizado un FETCH adicional.	Compruebe el valor de fin de lectura (ORA-01403) antes de realizar el FETCH.
ORA-01006	bind variable does not exist	Está utilizando una variable externa que no está declarada.	A menudo se utiliza con lenguajes como C o COBOL. Todas las variables utilizadas en un comando SQL se deben declarar previamente.
ORA-01012	not logged on	Está ejecutando comandos SQL pero no está conectado a la base de datos.	Conéctese a Oracle.
ORA-01329	unable to truncate required build table	No se puede vaciar la tabla, ya que está siendo utilizada por otro usuario.	Espere a que la tabla esté disponible.
ORA-01400	cannot insert NULL into	No puede poner el valor NULL en esta columna.	Modifique el comando INSERT.
ORA-01401	inserted value too large for column	En un comando INSERT o UPDATE está asignando un valor cuyo tamaño no corresponde al de la columna.	Compruebe la definición de las columnas respecto a sus valores.
ORA-01403	no data found	El comando SELECT no ha devuelto ningún dato.	Compruebe la cláusula WHERE o la tabla está vacía.
ORA-01405	fetched column value is NULL	En un cursor, tiene una cláusula INTO entre columnas de la tabla y variables. Una de las columnas tiene valor NULL y no puede asignarla a la variable.	Utilice la función NVL para indicar el valor por defecto que debe tomar la variable cuando la columna esté vacía. Ejemplo: SELECT NVL(COL1, 0) INTO VARIABLE
ORA-01406	fetched column value was truncated	El tamaño de la variable receptora es inferior al tamaño de la columna de la tabla.	Compruebe la definición de las columnas respecto a sus variables.
ORA-01407	cannot update to NULL	No puede poner el valor NULL en esta columna.	Modifique su comando UPDATE.
ORA-01438	value larger than specified precision allows for this column	La columna numérica es más pequeña que el valor utilizado en el comando INSERT o UPDATE.	Adapte el tamaño de sus variables o modifique la columna en la tabla.

ORA-01452	cannot CREATE UNIQUE INDEX-duplicate keys found	No puede crear un índice UNIQUE en esta tabla ya que contiene valores duplicados en las columnas utilizadas en el índice.	Limpie la tabla antes de crear el índice o modifique las columnas del índice.
ORA-01555	snapshot too old: rollback segment number string with name "string"too small	Ha abierto un cursor después de algún tiempo y el contenido de la base de datos ha cambiado. Oracle no puede asegurar la coherencia de los datos.	Modifique el cursor para que devuelva menos filas o ciérrelo regularmente aplicando un COMMIT y volviendo a abrirlo.
ORA-01562	failed to extend rollback segment number string	El comando de actualización (UPDATE, INSERT, DELETE) se está aplicando sobre muchas filas y Oracle no tiene espacio para guardar todas las actualizaciones.	Añada WHERE más restrictivos para disminuir el número actualizaciones simultáneas. Haga COMMIT con más frecuencia y, si es necesario contacte con el DBA.
ORA-01650	unable to extend rollback segment string by string in tablespacestring	El comando de actualización (UPDATE, INSERT, DELETE) se aplica a muchas filas y Oracle no tiene suficiente espacio para guardar todas las actualizaciones.	Añada WHERE más restrictivos para disminuir el número actualizaciones simultáneas. Haga COMMIT con más frecuencia y, si es necesario contacte con el DBA.
ORA-01730	invalid number of column names specified	En una cláusula CREATE TABLE AS SELECT xxx, el número de columnas del SELECT no corresponde al número de columnas de la tabla que recibe los datos.	Compruebe la sintaxis del comando.
ORA-01821	date format not recognized	El formato de la fecha que ha indicado no es válido.	Compruebe la sintaxis del comando.
ORA-01839	date not valid for month specified	El día indicado no corresponde al mes (por ejemplo 30/02).	Corrija el valor.
ORA-01843	not a valid month	Nombre de mes incorrecto.	Corrija el valor
ORA-01847	day of month must be between 1 and last day of month	El día debe corresponder al mes indicado (por ejemplo, 31/02 o 31/04 son incorrectos).	Corrija sus valores
ORA-01848	day of year must be between 1 and 365 (366 for leap year)	El día debe estar entre 1 y 365.	Corrija sus valores
ORA-01849	hour must be between	La hora debe estar entre	Corrija sus valores

	1 and 12	1 y 12.	
ORA-01850	hour must be between 0 and 23	La hora debe estar entre 0 y 23.	Corrija sus valores
ORA-02112	SELECT..INTO returns too many rows	Por ejemplo: ha realizado un SELECT que debería devolver una fila y devuelve varias.	Debe declarar las variables que reciben datos como una matriz o abra un cursor para devolver las filas una a una.
ORA-02289	sequence does not exist	La secuencia no existe o no tiene privilegios para acceder a ella.	Compruebe el nombre de la secuencia.

Sobre las transacciones y las sesiones (TCL/DCL)

CÓDIGO DE ERROR	MENSAJE ORACLE	CAUSA DEL ERROR	SOLUCIÓN
ORA-00018	maximum number of sessions exceeded	Se ha sobrepasado el número de sesiones máximas permitidas.	Aumente el valor del número máximo de sesiones: parámetro SESSIONS alter system set sessions=250 scope=spfile;
ORA-00019	maximum number of session licenses exceeded	Se han utilizado todas las licencias.	Aumente el valor del número máximo de licencias de sesiones: parámetro LICENSE_MAX_ SESSIONS
ORA-00021	session attached to some other process cannot switch session	Una sesión de usuario está siendo utilizada por otro usuario.	Controle las sesiones activas.
ORA-00022	invalid session ID-access denied	La sesión no existe o no tiene privilegios para utilizarla.	Utilice una sesión válida o compruebe los privilegios de usuario.
ORA-00025	failed to allocate string	Falta memoria para una cadena de caracteres.	Aumente el tamaño de la memoria de la SGA.
ORA-00026	missing or invalid session ID	El comando ALTER SYSTEM KILL SESSION indica que el ID no existe.	Vuelva a ejecutar el comando con un ID válido.
ORA-00027	cannot kill current session	No se puede matar la sesión actual.	Utilice otro usuario para matar la sesión.
ORA-00028	your session has been killed	Otro usuario ha matado su sesión.	Vuelva a conectarse.
ORA-00029	session is not a user session	El comando ALTER SYSTEM KILL SESSION indica que el ID no existe.	Utilice un número de ID válido.
ORA-00030	User session ID does not exist.	La sesión ya se ha eliminado.	Compruebe el número de sesión.
ORA-00032	invalid session migration password	Modificación de contraseña incorrecta, quizás es demasiado larga.	Vuelva a intentar con una nueva contraseña con menos de 30 caracteres.
ORA-00150	duplicate transaction ID	Está intentando iniciar una nueva transacción con un número que ya existe.	Cambie el número o pare la sesión del otro usuario.
ORA-00151	invalid transaction ID	El número de transacción no existe.	Cambie el número.
ORA-00987	missing or invalid username(s)	Falta el nombre del usuario en una cláusula	Indique un nombre de usuario correcto.

		GRANT o la sintaxis no respeta los 30 caracteres máximos o no contiene solo caracteres alfanuméricos.	
ORA-00992	invalid format for REVOKE command	El comando REVOKE no respeta la sintaxis.	Compruebe la sintaxis del comando y consulte la documentación Oracle.
ORA-01014	ORACLE shutdown in progress	Oracle se está parando y no puede ejecutar ningún comando SQL.	Espere a que la base de datos vuelva a arrancar.
ORA-01017	invalid username/password-logon denied	El nombre del usuario o la contraseña es incorrecta. No es posible la conexión.	Compruebe su nombre de usuario y/o su contraseña.
ORA-01435	user does not exist	En un comando GRANT o REVOKE está indicando un usuario que no existe.	Compruebe que el usuario sea correcto.

Sobre los componentes internos (memoria, sistema)

CÓDIGO DE ERROR	MENSAJE ORACLE	CAUSA DEL ERROR	SOLUCIÓN
ORA-00058	DB_BLOCK_SIZE must be string to mount this database (not string)	El valor indicado por el parámetro DB_BLOCK_SIZE en el arranque de la base de datos no es el que se utilizó en su creación.	Corrija el parámetro DB_BLOCK_SIZE.
ORA-00059	maximum number of DB_FILES exceeded	Se ha sobrepasado el número máximo de ficheros físicos (DB_FILES) de la base de datos.	Aumente el parámetro DB_FILES y vuelva a iniciar la base de datos.
ORA-00063	maximum number of LOG_FILES exceeded	Se ha sobrepasado el número máximo de ficheros log.	Aumente el parámetro LOG_FILES.
ORA-00065	initialization of FIXED_DATE failed	La variable FIXED_DATE que es la fecha de sistema Oracle no está en formato fecha (yyyy-mm-dd:hh24:mi:ss).	Modifique el parámetro y póngalo en el formato correcto y reinicie la base de datos.
ORA-00483	During shutdown a process abnormally terminated	Al parar la base de datos, un proceso ha acabado con error.	Compruebe los logs para detectar que proceso es el qué ha acabado con error.

Resumen de los principales comandos

1. Los principales comandos del LDD (lenguaje de definición de datos) o DDL (Data Definition Language)

Creación de una tabla

```
CREATE TABLE nombre_de_tabla  (Nombre_columna Tipo_columna,
                               Nombre_columna Tipo_columna,
                               Nombre_columna Tipo_columna,
                              ... );
```

```
CREATE TABLE TELEFONO (NUMERO          INTEGER,
                       TIPO            CHAR(2),
                       MARCA           INTEGER,
                       FECHA_COMPRA    DATE,
                       PRECIO          NUMBER(9,2),
                       NUM_PROPIETARIO INTEGER,
                       COLOR           VARCHAR(25));
```

Creación de una tabla a partir de otra

```
CREATE TABLE <Nueva tabla> AS SELECT <nombre columna1>,<nombre
columna2>... o <*> FROM <Tabla a copiar> WHERE ... ... ;
```

```
CREATE TABLE SAV_TELEFONO AS SELECT NUMERO, NUM_PROPIETARIO,
COLOR FROM TELEFONO WHERE TIPO = 'SP';
```

Añadir un comentario a una tabla o a una columna

```
COMMENT ON <'COLUMN' o 'TABLE'> <Nombre columna o nombre de tabla>
IS
'descripción libre';
```

```
COMMENT ON TABLE TELEFONO IS 'Lista de teléfonos de la empresa';
```

```
COMMENT ON COLUMN TELEFONO.TIPO IS 'Indica si es un teléfono de
tipo SMARTPHONE, CON TAPA u OTRO;
```

Eliminación de una tabla

```
DROP TABLE nombre_de_tabla;
```

```
DROP TABLE TELEFONO;
```

Creación de un sinónimo

```
CREATE SYNONYM <Nombre sinónimo> FOR <Nombre tabla>;
```

```
CREATE SYNONYM TELEFONO FOR ALEXANDRE.TELEFONO;
```

Modificación de una columna o de una restricción de una tabla

```
ALTER TABLE nombre_de_tabla [ADD nombre_de_columna Tipo_columna]
                            [,DROP COLUMN nombre_de_columna]
                            [,ADD CONSTRAINT nombre_restriccion]
                            [,DROP CONSTRAINT nombre_restriccion]
```

```
ALTER TABLE TELEFONO ADD TELEF_NUM_PIN INTEGER
```

Renombrar una tabla

```
RENAME nombre_de_tabla_antigua TO nombre_de_tabla_nueva;
```

```
RENAME TELEFONO TO SAV_TELEFONO;
```

Creación de una secuencia

```
CREATE SEQUENCE <nombre secuencia>;
```

```
CREATE SEQUENCE S_NUMERO START WITH 5 INCREMENT BY 1
MINVALUE 2 MAXVALUE 999999 CYCLE;
```

Creación de una vista

```
CREATE VIEW <Nombre_Vista> AS SELECT ...
```

```
CREATE VIEW TEL_FECHA AS
SELECT TELEFONO.FECHA_COMPRA,TELEFONO.TIPO,TIPO_TEL.DESC_TIPO,
       TELEFONO.MARCA,MARCA_TEL.DESC_MARCA
FROM TELEFONO, TIPO_TEL, MARCA_TEL
WHERE TELEFONO.TIPO   = TIPO_TEL.TIPO AND
      TELEFONO.MARCA  = MARCA_TEL.MARCA;
```

Eliminación de una vista

```
DROP VIEW nombre_vista
```

```
DROP VIEW TEL_FECHA
```

Creación de un índice

```
CREATE [UNIQUE] INDEX <nombre índice> ON <nombre tabla>
<nombre columna 1> [ASC|DESC], <nombre columna 2> [ASC|DESC], ...
...
```

```
CREATE INDEX I3_TIPMAR ON TELEFONO (TIPO, MARCA);
```

Eliminación de un índice

```
DROP INDEX <nombre_indice>
```

2. Los principales comandos del LMD (lenguaje de manipulación de datos) o DML (Data Manipulation Language)

El comando SELECT: seleccionar datos de una o varias tablas

```
SELECT <nombre columna1>, <nombre columna 2> ... ... ,
[SUM/COUNT/AVG/MIN/MAX](<nombre columna 3>), [SUM/COUNT/AVG/MIN/MAX]
(<nombre columna 4>)
FROM <tabla 1> , <tabla 4>,
JOIN <tabla 2> ON <tabla1.columna1> = <tabla2.columna1>
JOIN <tabla 3> ON <tabla1.columna2> = <tabla3.columna2>
WHERE ... ... ...
GROUP BY <nombre columna1>, <nombre columna 2> ... ...
HAVING [SUM/COUNT/AVG/MIN/MAX]<nombre columna 4> <operador aritmético>
<valor> ... ...
ORDER BY <nombre columna1>, <nombre columna 2> ... ...
```

```
SELECT TL.FECHA_COMPRA,
       TL.TIPO,
       TT.DESC_TIPO,
       TL.MARCA,
```

```
       MT.DESC_MARCA
FROM TELEFONO TL JOIN TIPO_TEL TT  ON TL.TIPO  = TT.TIPO
                 JOIN MARCA_TEL MT ON TL.MARCA = MT.MARCA
ORDER BY TL.TIPO ASC, MT.DESC_MARCA DESC;
```

```
SELECT TL.FECHA_COMPRA,
       TL.TIPO,
       TT.DESC_TIPO,
       TL.MARCA,
       MT.DESC_MARCA
FROM TELEFONO TL,
     TIPO_TEL TT ,
     MARCA_TEL MT
WHERE TL.TIPO  = TT.TIPO AND
      TL.MARCA = MT.MARCA;
```

El operador UNION

```
SELECT <columna 1>, <columna 2>, ... FROM <tabla1>
WHERE ...
UNION
SELECT <columna 1>, <columna 2>, ... FROM <tabla2>
WHERE ...
ORDER BY <columna 1>, <columna 2>, ...
```

```
SELECT PAIS FROM MARCA_TEL
UNION
SELECT PAIS FROM MARCA_PC;
```

El operador INTERSECT

```
SELECT <columna 1>, <columna 2>, ... FROM <tabla1>
WHERE ...
INTERSECT
SELECT <columna 1>, <columna 2>, ... FROM <tabla2>
WHERE ...
ORDER BY <columna 1>, <columna 2>, ...
```

```
SELECT DESC_MARCA, PAIS FROM MARCA_TEL
INTERSECT
SELECT DESC_MARCA, PAIS FROM MARCA_PC;
```

El operador EXCEPT

```
SELECT <columna 1>, <columna 2>, ... FROM <tabla1>
WHERE ...
EXCEPT
SELECT <columna 1>, <columna 2>, ... FROM <tabla2>
WHERE ...
ORDER BY <columna 1>, <columna 2>, ...
```

```
SELECT DESC_MARCA, PAIS FROM MARCA_TEL
EXCEPT
SELECT DESC_MARCA, PAIS FROM MARCA_PC;
```

El comando INSERT

```
INSERT INTO <nombre tabla> (<nombre columna1>, <nombre columna 2>
... ...)
VALUE (<valor 1>, <valor 2> ... ... )
```

```
INSERT INTO <nombre tabla> (<nombre columna1>, <nombre columna 2>
... ...)
SELECT <nombre columna1>, <nombre columna 2> ... ...
FROM <tabla 2>
[JOIN <tabla 3> ON <tabla2.columna> = <tabla3.columna1>]
[JOIN <tabla 4> ON <tabla2.columna2> = <tabla4.columna2>]
[WHERE ... ... ... ]
[GROUP BY <nombre columna1>, <nombre columna 2> ... ... ]
[HAVING [SUM/COUNT/AVG/MIN/MAX]<nombre columna 4> <operador
aritmético> <valor> ... ... ]
```

```
[ORDER BY <nombre columna1>, <nombre columna 2> ...]
```

```
INSERT INTO PELICULAS VALUES (3,'STAR WARS 6: EL RETORNO DEL
JEDI','ACCION','SF',TO_DATE('19/10/1983','DD/MM/YYYY'),
2,2,'20th
Century Fox ',' El imperio galáctico es más poderoso que nunca:
la construcción de la nueva arma, la Estrella de la Muerte, amenaza
todo el universo.');
```

El comando DELETE: eliminación de filas de una tabla

```
DELETE FROM <nombre tabla> WHERE ...
```

```
DELETE FROM TELEFONO WHERE COLOR IS NULL AND FECHA_COMPRA <
TO_DATE('31/12/2009','DD/MM/YYYY');
```

El comando TRUNCATE: vaciado de una tabla

```
TRUNCATE TABLE <nombre tabla>;
```

o

```
TRUNCATE TABLE TIPO_TEL;
```

Unión interna

```
SELECT <columna 1>, <columna 2>, etc ...
FROM <tabla izquierda> [INNER] JOIN <tabla derecha 1> ON <criterios1>,
                       [INNER] JOIN <tabla derecha 2> ON <criterios2>,
                        etc ...
```

Unión externa

Unión natural

Unión cruzada

```
SELECT <columna 1>, <columna 2>, etc ...
FROM <tabla izquierda>, <tabla derecha 1>, <tabla derecha 2> etc ...
WHERE <criterios 1> AND <criterios 2> etc ...
```

o

```
SELECT TL.FECHA_COMPRA,
       TL.TIPO,
       TT.DESC_TIPO,
       TL.MARCA,
       MT.DESC_MARCA
FROM TELEFONO TL INNER JOIN TIPO_TEL  TT ON TL.TIPO = TT.TIPO
                 INNER JOIN MARCA_TEL MT ON TL.MARCA = MT.MARCA;
```

```
SELECT <columna 1>, <columna 2>, etc ...
FROM <tabla izquierda> [LEFT|RIGHT|FULL] [OUTER] JOIN <tabla derecha
1> ON <criterios>,
                    [LEFT|RIGHT|FULL] [OUTER] JOIN <tabla derecha
2> ON <criterios>,
                     etc ...
```

```
SELECT TL.FECHA_COMPRA,
       TL.TIPO,
       TT.DESC_TIPO,
       TL.MARCA,
       MT.DESC_MARCA
FROM TELEFONO TL LEFT OUTER JOIN TIPO_TEL TT   ON TL.TIPO   =
TT.TIPO
                 LEFT OUTER JOIN MARCA_TEL MT ON TL.MARCA =
MT.MARCA;
```

```
SELECT <columna 1>, <columna 2>, etc ...
FROM <tabla izquierda> NATURAL JOIN <tabla derecha 1>,
                       NATURAL JOIN <tabla derecha 2>,
                       etc ...

SELECT FECHA_COMPRA,
       TIPO,
       DESC_TIPO,
       MARCA,
       DESC_MARCA
FROM TELEFONO NATURAL JOIN TIPO_TEL
              NATURAL JOIN MARCA_TEL;
```

```
SELECT <columna 1>, <columna 2>, etc ...
FROM <tabla izquierda> CROSS JOIN <tabla derecha 1>,
                       CROSS JOIN <tabla derecha 2>,
                       etc ...
```

```
SELECT <columna 1>, <columna 2>, etc ...
FROM <tabla izquierda>, <tabla derecha 1>, <tabla derecha 2> etc ...
```

```
SELECT TL.FECHA_COMPRA,
       TL.MARCA,
       MT.DESC_MARCA
FROM TELEFONO TL CROSS JOIN MARCA_TEL MT;
```

3. Los principales comandos del LCD (lenguaje de control de datos) o DCL (Data Control Language)

Creación de un usuario (Oracle)

```
CREATE <usuario> IDENTIFIED BY <Contraseña>;
```

```
CREATE USER ASMITH IDENTIFIED BY ASMITH;
```

Creación de un usuario (MySQL)

```
GRANT ALL PRIVILEGES ON *.* TO '<usuario>'@'localhost'
 IDENTIFIED BY '<Contraseña>' WITH GRANT OPTION;
```

Cambiar una contraseña (Oracle)

```
ALTER USER <usuario> IDENTIFIED BY <Nueva contraseña>;
```

```
ALTER USER ASMITH IDENTIFIED BY ABCD12E;
```

Cambiar una contraseña (MySQL)

```
SET PASSWORD FOR '<usuario>'@'<host>' = PASSWORD('mipass');
```

Asignar privilegios

```
GRANT <privilegio1>, <privilegio2>, ...
ON TABLE <nombre tabla>
TO <usuario1>, <usuario2> ...
[ WITH GRANT OPTION];
```

```
GRANT SELECT ON TABLE CLIENT TO ASMITH;
```

Asignar todos los privilegios (MySQL)

```
GRANT ALL PRIVILEGES
ON TABLE <tabla1>, <tabla2>,  ...
[ WITH GRANT OPTION] TO <usuario1>, <usuario2> ...
```

Asignar todos los privilegios (Oracle)

```
GRANT ALL PRIVILEGES
ON <tabla1>, <tabla2>,  ...
[ WITH GRANT OPTION] TO <usuario1>, <usuario2> ...
```

```
GRANT ALL PRIVILEGES ON CLIENTE TO ASMITH;
GRANT ALL PRIVILEGES ON PROVEEDOR TO ASMITH;
GRANT ALL PRIVILEGES ON FACTURA TO ASMITH;
```

Eliminar privilegios

```
REVOKE <privilegio1>, <privilegio2>, ...
ON TABLE <nombre tabla>
FROM <usuario1>, <usuario2> ...;
REVOKE ALL PRIVILEGES ON USUARIOS FROM ASMITH;
```

Crear un rol

Ejemplo Oracle

```
CREATE ROLE <nombre rol>;
```

```
CREATE ROLE CONTROL_GESTION;
```

Asignar un rol a los usuarios

```
GRANT CONTROL_GESTION T
GRANT CONTROL_GESTION T
etc ...
```

Eliminar un rol

Ejemplo Oracle

```
DROP ROLE <nombre rol>;
```

```
DROP ROLE CONTROL_GESTION;
```

4. Los principales comandos del LCT (lenguaje de control de transacciones) o TCL (Transaction Control Language)

Implementación del modo de bloqueo a nivel de sesión

Sintaxis MySQL

```
SET [SESSION | GLOBAL] TRANSACTION ISOLATION LEVEL
{ READ UNCOMMITTED
| READ COMMITTED
| REPEATABLE READ
| SERIALIZABLE
};
```

Sintaxis Oracle

```
SET TRANSACTION ISOLATION LEVEL
{ READ COMMITTED
| SERIALIZABLE
};
```

Implementación del modo de bloqueo a nivel de tabla

Sintaxis MySQL

```
SELECT ... FROM .... WHERE ....
FOR UPDATE;
```

o

Sintaxis Oracle

```
SELECT ... FROM .... WHERE ....
LOCK IN SHARE MODE;
```

```
SELECT ... FROM .... WHERE ....
FOR UPDATE;
```

Validar las modificaciones en una base de datos

```
COMMIT;
```

Anular las modificaciones en una base de datos

```
ROLLBACK TO [SAVEPOINT] <nombre del punto de control>;
```

Crear un punto de sincronización

```
SAVEPOINT <nombre del punto de control>;
```

Ejemplo

```
SAVEPOINT ANTES_INSERT;
```

5. La creación de procedimientos y funciones

Creación de un procedimiento almacenado

```
CREATE OR REPLACE PROCEDURE <nombre procedimiento>
 ([<variable entrada 1> IN <formato>,
   <variable entrada 2> IN <formato>,
   ... ...]
   <variable salida> OUT <formato>)
 IS

BEGIN

... ...

[EXCEPTION
... ...
]
END;
```

Creación de una función almacenada

```
CREATE OR REPLACE FUNCTION <nombre función>
 [(<variable entrada 1> IN <formato>,
   <variable entrada 2> IN <formato>,
   ... ... )]
   RETURN <formato>
 IS
<variable salida> <formato>;
BEGIN

... ...

[EXCEPTION
... ...
]
END;
```

Creación de un paquete

```
CREATE OR REPLACE PACKAGE <nombre paquete> IS
      PROCEDURE <nombre procedimiento 1>;
      FUNCTION <nombre función 1> (<variable 1> IN <formato>) RETURN
<formato>;
END;
/
CREATE OR REPLACE PACKAGE BODY <nombre paquete> IS

   FUNCTION <función 1>
      ... ...
   END;

   PROCEDURE <procedimiento 1> IS
      ... ...
   END;
END;
/
```

Sintaxis para la compilación de un procedimiento, una función o un paquete

```
ALTER <'PROCEDURE', 'FUNCTION' o 'PACKAGE'> <Nombre de procedimiento,
función o paquete> COMPILE;
```

Sintaxis para la eliminación de un procedimiento, una función o un paquete

```
DROP <'PROCEDURE', 'FUNCTION' o 'PACKAGE'> <Nombre de procedimiento,
función o paquete>;
```

Funciones SQL presentadas en este libro

COUNT	Contar filas
SUM	Sumar valores
MAX Y MIN	Valores máximo y mínimo
AVG	Promedio
ABS	Valor absoluto
ASCII	Valor ASCII de un carácter
COS	Coseno
SIN	Seno
MOD(<columna>,<valor>)	Módulo
ROUND(<columna>,[<precisión>])	Redondeo
SQRT	Raíz cuadrada
IN - NOT IN	En la lista o no
EXISTS - NOT EXISTS	Existe o no
BETWEEN	Buscar entre dos valores
LIKE	Buscar los elementos que contienen una parte del valor
CURRENT_DATE	Fecha del día
CURRENT_TIMESTAMP	Fecha y hora del día
LOWER / UPPER / UCASE / LCASE	Transformar los caracteres en minúsculas y mayúsculas
TRIM / LTRIM / RTRIM	Eliminar los blancos a la derecha o izquierda de una cadena de caracteres
TO_CHAR	Transformar un numérico o una fecha en carácter
INSTR	Encontrar la posición de una cadena de caracteres en una cadena
LPAD / RPAD	Añadir caracteres por delante o por detrás de una cadena
SUBSTR	Extraer una parte de una cadena de caracteres
NVL	Comprobar si una columna tiene el valor null
NULLIF	Comparar dos columnas
COALESCE	Comprobar varios valores
CAST	Cambiar el tipo de una columna
DECODE	Condicionar y comprobar los datos
CASE	Condicionar y comprobar los datos

Glosario

SQL	*Structured Query Language*
DDL	*Data Definition Language*
LDD	Lenguaje de definición de datos
DML	*Data Manipulation Language*
LMD	Lenguaje de manipulación de datos
DCL	*Data Control Language*
LCD	Lenguaje de control de datos
TCL	*Transaction Control Language*
LCT	Lenguaje de control de transacciones
BDD o BD	Base de datos
SGBD	Sistema de gestión de bases de datos
SGBDR	Sistema de gestión de bases de datos relacionales
DBA	*DataBase Administrator*
MOA	Gestión de proyectos
MOE	Gestor del proyecto

Recursosinformáticos

SQL

Los fundamentos del lenguaje

Este libro sobre los fudamentos del lenguaje SQL se dirige a **desarrolladores** e **informáticos principiantes** que deban trabajar con un Sistema Gestor de Bases de Datos Relacionales (SGBDR) para almacenar y manipular datos. Su objetivo es describir los **principales comandos más utilizados del lenguaje SQL** (independientemente de las variaciones realizadas por los editores de los diferentes SGBDR) para permitir al lector hacerse cargo rápidamente de una base de datos relacional y ser capaz de **crear tablas**, de **consultarlas**, de **modificarlas**, de **insertar y suprimir registros**.

El libro comienza con una breve historia sobre la creación de la norma SQL y algunas **nociones sobre el modelo relacional**. A continuación, cada capítulo aborda una subdivisión de SQL; la **creación** y la **manipulación de tablas**, y a continuación la **gestión de los datos** de estas tablas. El autor continúa con **las funciones SQL**, la **seguridad de los datos** y las **transacciones** y acaba abordando temas un poco más complejos como las **cargas masivas**, las **importaciones** y **exportaciones de tablas**, los **trigger**, el **PL/SQL** y los **errores que se encuentran con más frecuencia**.

Los ejemplos que se utilizan en este libro se han realizado con la versión Oracle 10g Express Release 10.2.0.1.0 y la versión MySql 5.1.54 y se pueden **descargar** de www.ediciones-eni.com.

Eric Godoc es Director de proyectos informáticos en un departamento de sistemas de información. Sus proyectos entorno al desarrollo y migraciones de aplicaciones en grandes empresas le han reportado una gran experiencia en la manipulación de bases de datos relacionales. Con este libro ofrece a los lectores su experiencia, y sobre todo, proporciona los medios para aprender las bases del lenguaje SQL. Su objetivo es puedan responder a la mayoría de necesidades que puedan tener en la utilización de una base de datos relacional, sea cual sea.

En **www.ediciones-eni.com**:

- Las consultas de gestión de tablas y de gestión de los datos de las tablas.
- Las consultas vinculadas a la seguridad y a la gestión de las transacciones.
- Los ejemplos de scripts (carga, importación/exportación, eliminación de duplicados, generación de comandos SELECT...).

Los capítulos del libro

Para más información :

ISBN : 978-2-7460-9124-5

eni ediciones

www.ediciones-eni.com