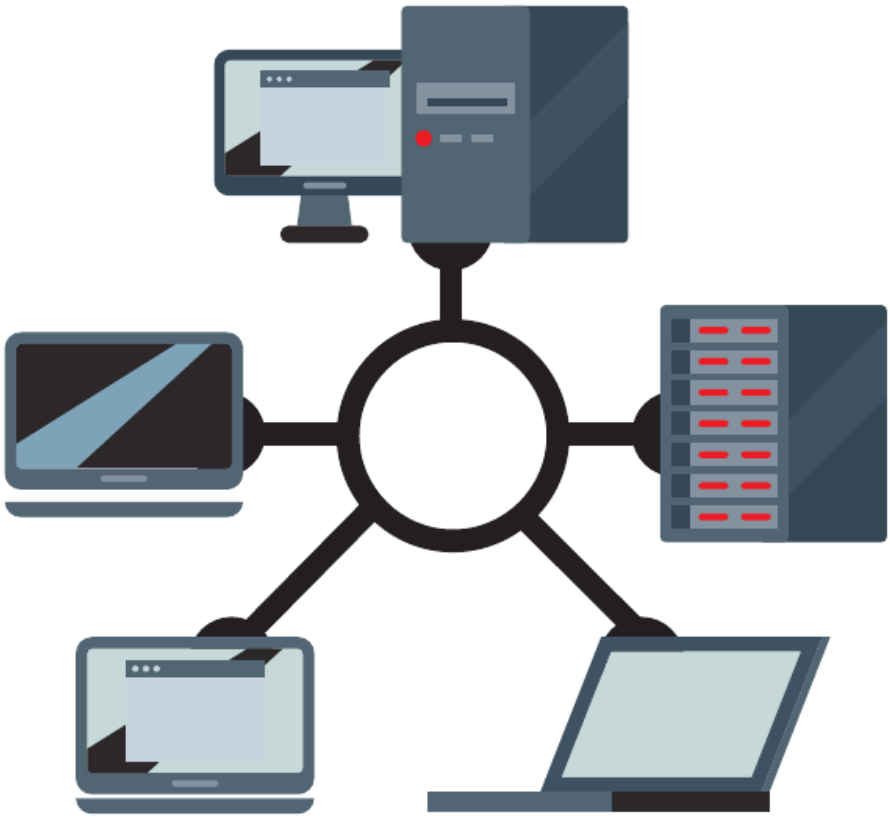
ORACLE 12c SQL

Curso práctico de formación



Incluye:

12 supuestos prácticos totalmente resueltos además de multitud de ejemplos
45 cuestiones resueltas para la preparación de los exámenes de certificación de Oracle en lenguaje SQL: 1Z0-047, 1Z0-051, 1Z0-061, IZ0-071 y 1Z0-117

Antolín Muñoz Chaparro

Alfaomega

libros R

Oracle 12c SQL

Curso práctico de formación

Oracle 12c SQL

Curso práctico de formación

Antolín Muñoz Chaparro

Alfaomega

libros RC

Diseño de colección y preimpresión:
Grupo RC
Diseño cubierta: Cuadratín

Datos catalográficos
Muñoz, Antolín
Oracle 12c SQL. Curso práctico de formación
Primera Edición
Alfaomega Grupo Editor, S.A. de C.V., México

ISBN: 978-607-538-225-8

Formato: 17 x 23 cm Páginas: 356

Oracle 12c SQL. Curso práctico de formación
Antolín Muñoz Chaparro
ISBN: 978-84-947170-4-8 edición original publicada por RC Libros, Madrid, España.

Primera edición: Alfaomega Grupo Editor, México, mayo 2018

Dr. Isidoro Olvera (Eje 2 sur) No. 74, Col. Doctores, 06720, Ciudad de México.

Miembro de la Cámara Nacional de la Industria Editorial Mexicana
Registro No. 2317

Pág. Web: **http://www.alfaomega.com.mx**
E-mail: **atencionalcliente@alfaomega.com.mx**

ISBN: 978-607-538-225-8

Nota importante:
La información contenida en esta obra tiene un fin exclusivamente didáctico y, por lo tanto, no está previsto su aprovechamiento a nivel profesional o industrial. Las indicaciones técnicas y programas incluidos, han sido elaborados con gran cuidado por el autor y reproducidos bajo estrictas normas de control. ALFAOMEGA GRUPO EDITOR, S.A. de C.V. no será jurídicamente responsable por: errores u omisiones; daños y perjuicios que se pudieran atribuir al uso de la información comprendida en este libro, ni por la utilización indebida que pudiera dársele.

Edición autorizada para venta en México y todo el continente americano.

Impreso en México. Printed in Mexico.

Empresas del grupo:

México: Alfaomega Grupo Editor, S.A. de C.V. – Dr. Isidoro Olvera (Eje 2 sur) No. 74, Col. Doctores, C.P. 06720, Del. Cuauhtémoc, Ciudad de México – Tel.: (52-55) 5575-5022 – Fax: (52-55) 5575-2420 / 2490. Sin costo: 01-800-020-4396 – E-mail: atencionalcliente@alfaomega.com.mx

Colombia: Alfaomega Colombiana S.A. – Calle 62 No. 20-46, Barrio San Luis, Bogotá, Colombia, Tels.: (57-1) 746 0102 / 210 0415 – E-mail: cliente@alfaomega.com.co

Chile: Alfaomega Grupo Editor, S.A. – Av. Providencia 1443. Oficina 24, Santiago, Chile Tel.: (56-2) 2235-4248 – Fax: (56-2) 2235-5786 – E-mail: agechile@alfaomega.cl

Argentina: Alfaomega Grupo Editor Argentino, S.A. – Av. Córdoba 1215, piso 10, CP: 1055, Buenos Aires, Argentina, Tel./Fax: (54-11) 4811-0887 y 4811 7183 – E-mail: ventas@alfaomegaeditor.com.ar

ÍNDICE

PRÓLOGO

Este libro tiene como objeto actualizar a la versión 12c de la base de datos de Oracle el libro Oracle 11g SQL, también publicado por esta editorial en el año 2011.

Se ha mantenido con respecto a la versión 11g del libro todo lo relacionado con el lenguaje SQL que sigue siendo compatible en la versión 12c de la base de datos de Oracle, añadiendo, además, las nuevas características del lenguaje SQL que se han puesto en funcionamiento para la versión 12c. Estas nuevas características se pueden identificar fácilmente tanto en el índice del libro como a lo largo de los capítulos del mismo con el símbolo New 12c.

Por último se ha actualizado la información sobre el proceso de Certificación de Oracle, así como los enlaces a las distintas páginas de la web de Oracle donde puede encontrar más información.

EL AUTOR

Antolín Muñoz Chaparro nació en Madrid en 1970. Diplomado en Ingeniería Técnica en Informática de Gestión por la Universidad Carlos III de Madrid, se especializó en la rama de bases de datos relacionales, y más en concreto en la base de datos Oracle.

Su trayectoria profesional ha estado vinculada desde el año 1990 al servicio de la administración pública, donde actualmente trabaja como Jefe de Servicio de

Aplicaciones Informáticas. Desde su incorporación a la administración ha trabajado a diario con la base de datos Oracle en sus distintas versiones, desde la versión 6 hasta la versión 12c que actualmente maneja y para la que está enfocada este libro. Su experiencia con la base de datos Oracle abarca desde la administración y migración de la misma, pasando por la programación en los lenguajes asociados: SQL y PL/SQL, y por último en el diseño de aplicaciones con las herramientas Designer, Forms y Reports de Oracle, utilizando plataformas hardware basadas en sistemas Microsoft y Unix.

Además de la experiencia profesional en el sector público, también ha realizado colaboraciones en el sector privado impartiendo cursos de formación de administración y programación de la base de datos Oracle. Fruto de esta experiencia le ha permitido elaborar este y otros manuales de ayuda práctica para la programación y diseño de aplicaciones contra la base de datos Oracle.

1 CONCEPTOS DE BASES DE DATOS RELACIONALES

¿QUÉ ES UNA BASE DE DATOS?

Es un conjunto de información almacenada en una estructura de ficheros en disco, que será consultada y modificada por los usuarios que acceden a la misma, siendo el Sistema Gestor de Base de Datos (en adelante SGBD) el encargado de llevar físicamente a cabo estas operaciones.

Gráficamente, una base de datos se representa con un cilindro.

Dentro del mismo se ubicará la información que contenga.

El proceso de consulta de una base de datos quedaría representado en la *figura 1-1*, donde un usuario ejecuta desde su ordenador una sentencia (`select * from tabla`), contra la base de datos.

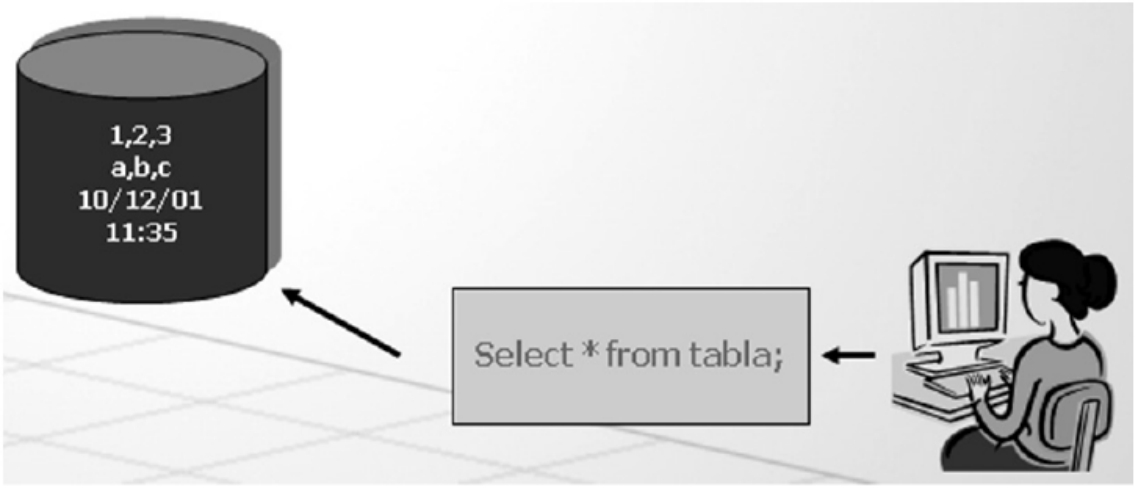


Fig. 1-1 Ejecución de una sentencia de consulta.

La base de datos convierte la lógica de la sentencia ejecutada por el usuario en una búsqueda física de la información dentro de la base de datos. Para ello, tendrá que acceder a los índices que contenga la tabla consultada, para ubicar la zona física del disco donde se encuentra la información.

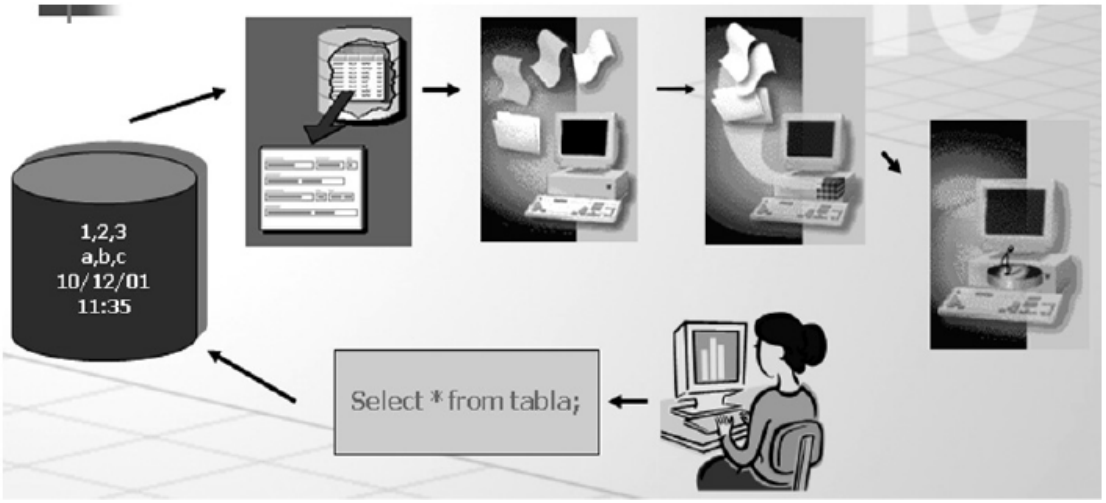


Fig. 1-2 Búsqueda física de la información en la base de datos.

Completada la búsqueda de la información física, el SGBD devuelve la información al usuario de forma estructurada, según el tipo de consulta que haya realizado.

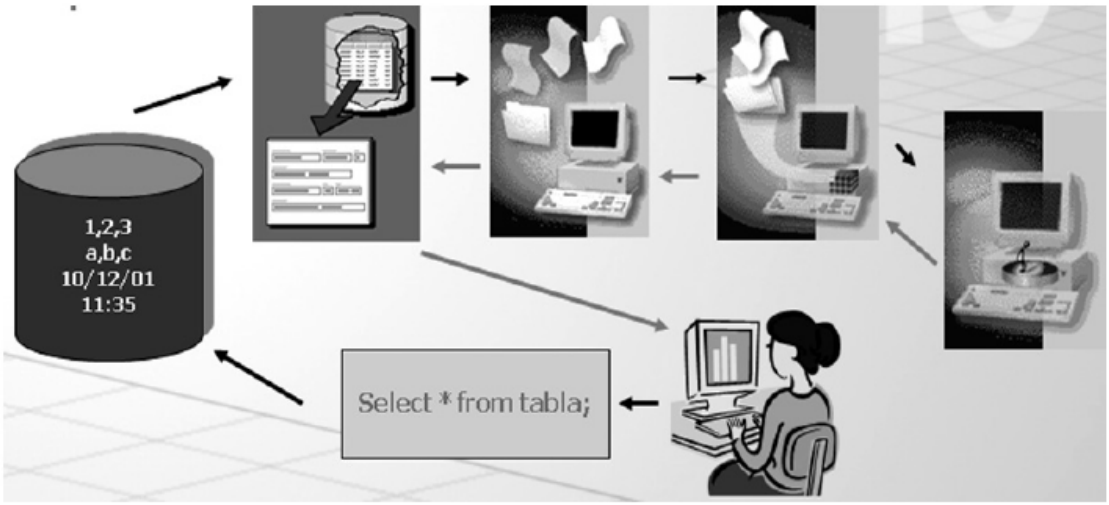


Fig. 1-3 Búsqueda física de la información en la base de datos.

¿QUÉ SIGNIFICA BASE DE DATOS RELACIONAL?

Cuando hablamos de base de datos, tendemos a unir el concepto relacional a su significado, pero realmente; ¿qué significa base de datos relacional? Y ¿cuándo podemos considerar a una base de datos como relacional?

- Una base de datos relacional es aquella que se fundamenta o utiliza las teorías del modelo relacional de Codd.
- Oracle implementa el modelo relacional en sus SGBD desde su primera versión. Actualmente ha extendido este modelo (sin abandonarlo), para añadir funcionalidades de los entornos orientados a objeto.
- Otros SGBD que utilizan el modelo relacional son SQLSERVER, SYSBASE y MySQL.

INTRODUCCIÓN AL MODELO RELACIONAL

El modelo relacional fue enunciado por Codd a finales de los años 60.

Está basado en la teoría de las relaciones, donde se contemplan como elementos principales: las entidades y las relaciones.

El elemento básico es la **relación**: nexo entre 2 o más entidades.

Elementos del modelo relacional

Los elementos del modelo relacional son:

- Entidades.
- Relaciones.
- Atributos.
- Tuplas.
- Dominios.

ENTIDADES

Una entidad es un objeto del mundo real que tiene interés para el sistema, y del cual se puede extraer una serie de atributos.

Ejemplos de entidades son: una casa, un coche, una oficina, un hospital, una persona, etc.

RELACIONES

Una relación es el nexo (vínculo de unión) entre 2 o más entidades a través de alguno/s de sus atributos.

Por ejemplo, si tomamos como referencia 2 entidades: cliente y factura, ambas se pueden relacionar por el código de cliente (atributo perteneciente a ambas entidades).

ATRIBUTOS

Un atributo es la característica o propiedad que define una entidad, y que la diferencia de otra.

Ejemplos de atributos para la entidad casa serían: ubicación, número de habitaciones, metros cuadrados, tipo de vivienda, etc.

TUPLAS

Una tupla es la unión de todos los atributos de una entidad en una sola estructura independiente.

Un ejemplo de tupla sería: Pepito, Pérez, 26, C/ Madrid, 913332211

DOMINIOS

Un dominio es un conjunto de valores que puede adoptar un atributo de una entidad.

Un ejemplo del dominio correspondiente al atributo metros cuadrados de una casa sería: 1 m^2 - 30 m^2, 31 m^2 – 60 m^2, 61 m^2 – 100 m^2, 100 m^2 – 200 m^2, > 200 m^2.

Tipos de clave en el modelo relacional

Una clave es un conjunto de valores de una entidad que distingue unívocamente esta información de otro conjunto de valores de la misma entidad.

Por ejemplo, si consideramos una entidad alumno que incluya como atributos: DNI y nombre de alumno, la clave de la misma sería el DNI, que es el único dato que hace diferenciar un alumno de otro.

En el modelo relacional se consideran 3 tipos de claves:

- Primaria.
- Alternativa.
- Ajena.

CLAVE PRIMARIA

Una clave primaria corresponde con el/los atributos de una entidad cuyo conjunto de valores distingue unívocamente una tupla de otra.

En el modelo relacional solo se permite una clave primaria por cada entidad.

Un ejemplo de clave primaria para la entidad trabajador sería el DNI.

CLAVE ALTERNATIVA

Una clave alternativa es aquel atributo o conjunto de atributos que podrían haber formado una clave primaria, pero que al ya existir una definida, no pueden hacerlo.

Un ejemplo de clave alternativa para la entidad trabajador sería el número de la Seguridad Social.

CLAVE AJENA

Una clave ajena es aquel atributo o conjunto de atributos de una entidad que se relaciona con la clave primaria de otra entidad.

Por ejemplo, si tomamos como referencia 2 entidades: cliente y factura, que tienen como atributo en común el DNI de un cliente, podemos definir como clave primaria de cliente el atributo DNI y como clave ajena de factura también el mismo atributo de DNI. De esta forma podemos relacionar factura con cliente.

Restricciones en el modelo relacional

En el modelo relacional se permite la determinación de restricciones sobre los atributos de una entidad.

Una restricción se define sobre un atributo y consiste en una limitación que se le aplica al mismo.

Una clave es por definición una restricción que limita la repetición de valores para la clave (en el caso de las primarias o alternativas), o la asignación de valores distintos a los de la clave primaria relacionada (para el caso de las claves ajenas).

Así mismo, un dominio definido sobre un atributo también lo es, porque limita los valores posibles que se pueden almacenar.

Por último, también se pueden definir restricciones que no sean del tipo clave o del tipo dominio. Por ejemplo, cuando queremos definir que un atributo concreto de una entidad no pueda almacenar valores vacíos.

Teorías de normalización

Son normas y reglas que permiten convertir un modelo conceptual (Entidad/Relación) en un modelo relacional.

La aplicación de estas teorías nos permite convertir el modelo relacional en un modelo físico de construcción de la base de datos, donde existirán elementos propios del tipo: tablas, columnas, índices y constraints.

RELACIÓN ENTRE LOS ELEMENTOS DEL MODELO RELACIONAL Y UNA BASE DE DATOS

Para poder utilizar el modelo relacional (diagrama de Entidad/Relación) en un SGBD, es necesaria la conversión de los elementos del modelo relacional a elementos propios de una base de datos relacional.

A continuación, se muestra una tabla con la conversión de elementos:

Modelo conceptual Entidad/Relación (Modelo relacional)	Base de datos
Entidades	Tablas
Atributos	Columnas
Tuplas	Filas
Dominios	Restricciones
Claves	Restricciones e índices

Una **tabla** es un conjunto de filas y columnas donde se almacena la información.

Una **columna** representa un único valor o atributo de una tabla.

Una **fila** representa la unión de un conjunto de valores repartidos entre las columnas de la tabla.

Una **restricción (constraint)** en inglés, es aquella limitación que se impone a los valores que puede llegar a tomar una columna de una tabla. Así mismo se pueden imponer restricciones a nivel de toda la tabla, usando claves.

Un **índice** es una estructura de acceso rápido que utiliza el SGBD para localizar más rápido la información física contenida en la base de datos.

COMPOSICIÓN DE UN ÍNDICE

Un índice se crea para poder acceder a la información física de manera más rápida por parte del SGBD, y se compone de un conjunto de columnas que son la clave de acceso a través del índice + la identificación de la posición física dentro del fichero, donde se encuentra la fila para recuperar la información correspondiente.

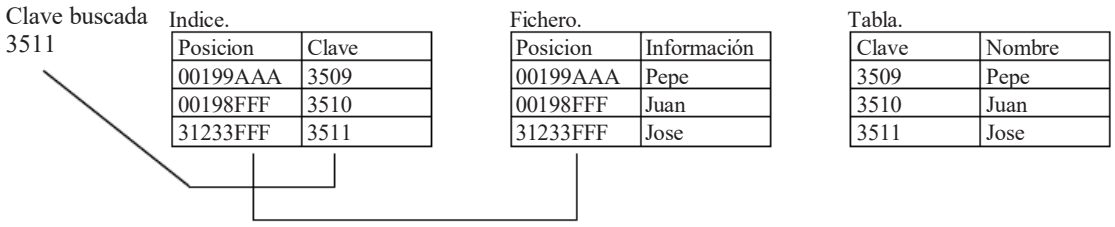


Fig. 1-4 Esquema de un índice y método de localización del dato físico.

CREACIÓN DE UN DIAGRAMA RELACIONAL

En este apartado se orienta sobre la creación de un diagrama relacional, los pasos a seguir y los elementos a contemplar en el mismo.

La teoría del modelo de Codd parte de la idea de creación de un diagrama Entidad/Relación, donde únicamente se especificarán las Entidades (con sus atributos clave y no clave), y las Relaciones entre las mismas, especificando cardinalidades y grado de las relaciones. Este tipo de diagramas queda fuera del estudio de este curso; no obstante, al final de este apartado se muestra el diagrama Entidad/Relación del que se partiría para poder llegar a obtener el modelo relacional descrito a continuación como práctica de este curso.

El supuesto práctico que se expresa en las distintas fases de este apartado quiere reflejar, en un modelo relacional, la estructura de un comercio que vende productos informáticos. Evidentemente esta descripción es muy abierta, por lo que el diagrama que se propone podría ser diferente dependiendo de la interpretación que quiera darle cada alumno a este enunciado.

Fase 1. Definición de entidades

En la primera fase del diseño del diagrama tenemos que identificar las entidades, sus atributos y las restricciones que pueden imponerse sobre las mismas, si las hubiera.

Fig. 1-5 Esquema de entidades de un comercio que vende productos informáticos.

Fase 2. Identificación de claves primarias y alternativas

En la segunda fase hay que definir los atributos de cada entidad que forman la clave primaria y, si las hubiera, las posibles claves alternativas.

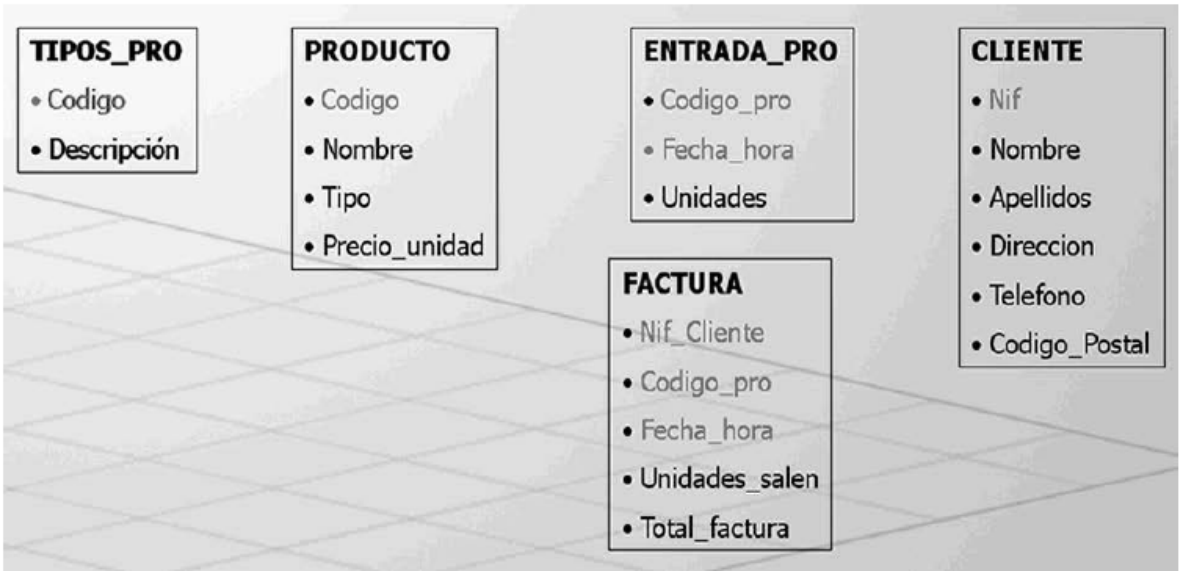


Fig. 1-6 Claves primarias de las entidades.

Fase 3. Identificación de claves ajenas y relaciones

En la última fase del diseño hay que definir las claves ajenas que relacionan unas entidades con otras y los atributos que se ven afectados por las mismas.

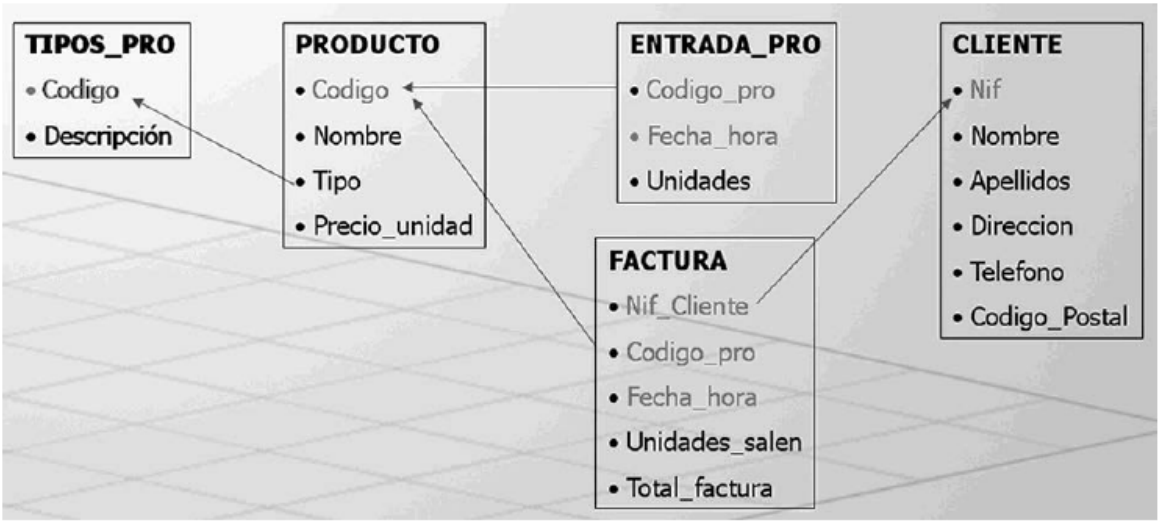


Fig. 1-7 Diagrama relacional completo.

Diseño Entidad/Relación

Como se ha comentado al comienzo del apartado, un diseño relacional requiere de un diseño Entidad/Relación previo. Para nuestro supuesto, el diagrama Entidad/Relación sería el siguiente:

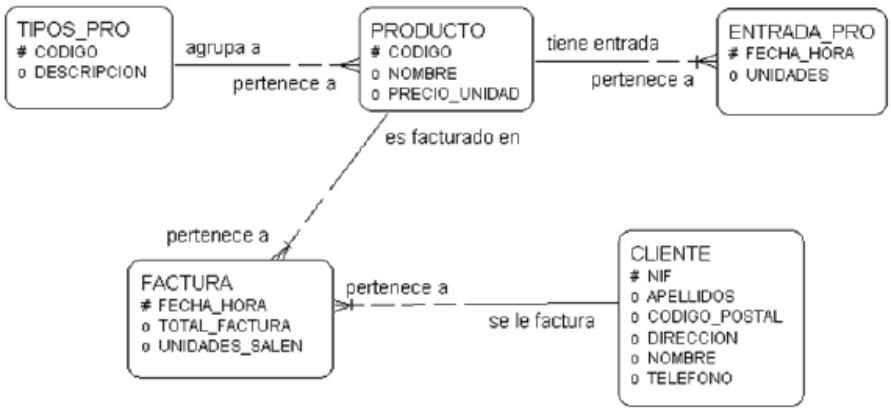
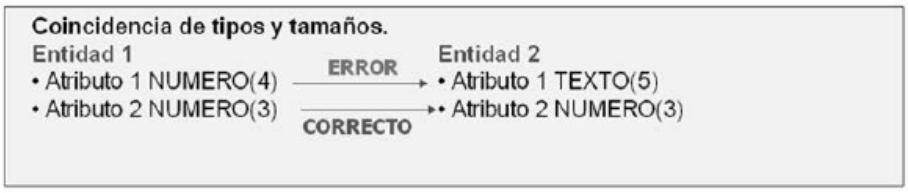


Fig. 1-8 Modelo Entidad/Relación de un comercio que vende productos informáticos.

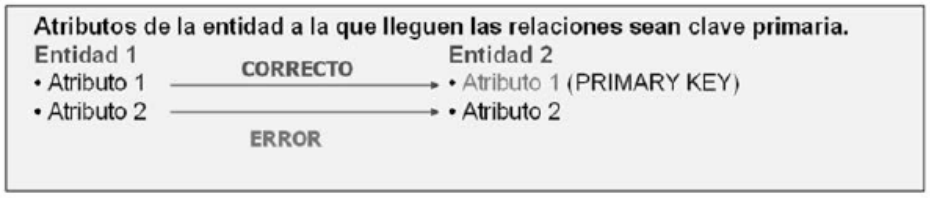
REGLAS PARA LA CREACIÓN DE CLAVES AJENAS

Para crear una relación entre entidades o claves ajenas, es necesario que se cumplan una serie de condiciones:

- Que el tipo de datos y tamaño entre los atributos coincida.



- Que el/los atributos de la entidad destino de la relación sean clave primaria o clave única.



- La relación con la entidad destino debe ser completa (con todos los atributos que forman la clave primaria de la misma).

OPERACIONES DE CONSULTA

En el modelo relacional de Codd se definen las siguientes operaciones de consulta sobre entidades:

- Selección.
- Proyección.
- Unión.
- Intersección.
- Diferencia.
- Producto cartesiano.
- Join.

Operación de selección

Es una operación de consulta que obtiene un conjunto de tuplas de una sola entidad con o sin condiciones.

ATRI1	ATRI2	ATRI3	ATRI4	ATRI5
1	2	3	4	6
7	8	9	10	11
12	13	14	15	16

Fig. 1-9 Resultado de una operación de selección.

Operación de proyección

Es una operación de consulta que permite obtener valores de uno o varios atributos de una sola entidad.

ATRI1	ATRI2	ATRI3	ATRI4	ATRI5
1	2	3	4	6
7	8	9	10	11
12	13	14	15	16

Fig. 1-10 Resultado de una operación de proyección.

Operación de unión

Para poder realizar una operación de consulta de unión entre entidades, tiene que coincidir el número de atributos de ambas y el tipo.

El resultado de esta operación es la suma de tuplas de ambas entidades para los atributos que se unen.

A	B	C
1	1	11
2	2	22
3	3	33

∪

A	D	C
4	467	44
5	578	55
6	678	66

=

A	C
1	11
2	22
3	33
4	44
5	55
6	66

Fig. 1-11 Resultado de una operación de unión.

Operación de intersección

Para realizar una operación de consulta con intersección entre entidades, tiene que coincidir el número de atributos de ambas y el tipo.

El resultado de esta operación es la aparición únicamente de las tuplas con el mismo contenido de ambas entidades.

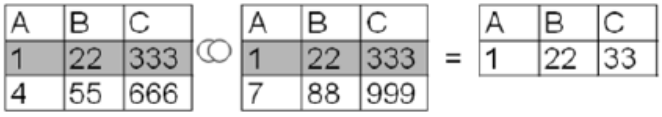


Fig. 1-12 Resultado de una operación de intersección.

Operación de diferencia

Para realizar una operación de consulta con diferencia entre entidades, tiene que coincidir el número de atributos de ambas y el tipo.

El resultado de esta operación es la aparición de aquellas tuplas de la primera entidad que no coincidan con las tuplas de la segunda.

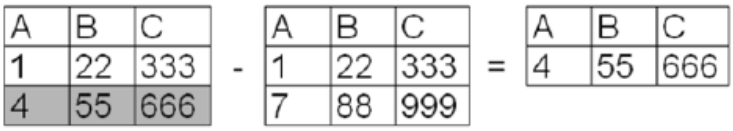


Fig. 1-13 Resultado de una operación de diferencia.

Operación de producto cartesiano

La operación de consulta del producto cartesiano permite obtener un resultado que será una tupla por cada combinación entre cada tupla de la primera entidad y todas las tuplas de la segunda.

A	B	C
1	2	3

X

D	E
44	55
66	77

=

A	B	C	D	E
1	2	3	44	55
1	2	3	66	77

Fig. 1-14 Resultado del producto cartesiano.

Operación de Join

La operación de consulta Join permite unir tuplas de dos entidades a través de algún atributo en común.

El resultado de esta operación es la suma de los atributos que se quieren mostrar.

A	B	C
1	22	33

J

A	E	F
1	565	32
2	751	12

=

A	B	C	E	F
1	22	33	565	32

Fig. 1-15 Resultado de una operación de Join.

SUPUESTO PRÁCTICO 0: ***Resolución en el Anexo I.***

Diseñar un esquema de modelo relacional en el que se vean reflejados las entidades, atributos mínimos, relaciones y las claves primarias, alternativas y ajenas que considere oportunas, de acuerdo con el enunciado que se especifica a continuación. Opcionalmente, podrá diseñar, previo al modelo relacional, el modelo Entidad/Relación del que se derivaría el modelo relacional.

- Representar la estructura de un centro de enseñanza que posee delegaciones en varias provincias de España.
- Será necesario reflejar para el centro de enseñanza los siguientes atributos:

  - Código de 1 a 99.
  - Nombre.
  - Dirección.
  - Provincia.
  - Teléfono.
  - Código postal.
- Para las provincias habrá que reflejar:
  - Código provincia de 1 a 52.
  - Nombre.
- El código del centro es unívoco y distinto para todos los centros estatales.
- En cada centro se imparten una serie de cursos con los siguientes atributos:
  - Código de curso de 1 a 99999.
  - Nombre de curso.
  - Código del centro donde se imparte.
- Cada curso es impartido por una serie de profesores cualificados con los siguientes atributos:
  - DNI.
  - Nombre.
  - Apellidos.
  - Dirección.
  - Teléfono.
  - Código postal.
  - Código de curso.
  - Código de centro.
  - Nº de la Seguridad Social.
- El código del curso puede repetirse en centros distintos, por lo que no es unívoco. Un curso de un centro solo puede ser impartido por un profesor.
- Por último, a estos cursos asisten una serie de alumnos con los siguientes atributos:
  - DNI.
  - Nombre.
  - Apellidos.
  - Dirección.
  - Teléfono.
  - Código postal.
  - Código de curso.

- Código de centro.
- Fecha de comienzo del curso.
- Fecha de fin del curso.

- Un alumno en un mismo centro puede asistir a varios cursos, pero al mismo curso solo podrá asistir en fechas de comienzo distintas.

INTRODUCCIÓN AL LENGUAJE SQL 2

El lenguaje SQL es el conjunto de comandos que todos los programas y usuarios deben utilizar en sus accesos a una base de datos de Oracle.

En este capítulo vamos a ver los siguientes apartados:

- Historia del lenguaje SQL.
- Lenguaje SQL estándar.
- Beneficios en la utilización del lenguaje SQL.
- Lenguaje SQL embebido.
- Convenciones léxicas.

HISTORIA DEL LENGUAJE SQL

El nombre originario que se le dio a SQL fue SEQUEL y nació como un prototipo de IBM entre los años 1974 y 1975. Se basaba en el modelo relacional teórico de Codd.

Posteriormente, el lenguaje recibió el nombre de SEQUEL II cuando fueron añadidas nuevas funcionalidades.

En 1979 aparece el primer gestor de base de datos basado en SQL, que lo adopta la compañía Oracle.

En 1982 el comité ANSI (American National Standard Institute) presenta un lenguaje estándar basado en el SQL propio de IBM y lo denomina SQL/ANS.

Más tarde en 1989, con nuevas mejoras, el estándar de ANSI pasa a denominarse Addendum.

En 1992 tras incrementos semánticos del lenguaje Addendum, basándose en el esquema relacional, se aprueba el estándar de ANSI: SQL2. Este estándar es el que utiliza Oracle en su base de datos de la versión 8.

En 1996 ANSI define SQL/PSM como una extensión a SQL, para proporcionar programación procedimental (fue la inspiración para la creación de PL/SQL).

La extensión SQL/PSM es definida por ISO/IEC 9075-4:2003. SQL/PSM estandariza extensiones procesales para SQL, incluyendo el flujo de control, el manejo de condiciones, declaración de variables, cursores y variables locales, y la asignación de expresiones a variables y parámetros. Además, SQL/PSM formaliza la declaración y el mantenimiento de rutinas de lenguaje de base de datos persistentes (por ejemplo, procedimientos almacenados).

En 1999 se aprueba una nueva versión oficial estándar de ANSI, conocida como SQL3 o SQL3-PSM, que añade comparación de expresiones regulares, consultas recursivas, triggers, soporte para procedimientos, sentencias de control de flujo, tipos de datos no escalares y algunas de las características de la programación orientada a objetos.

En 2003 se define SQL 2003, que introduce características del lenguaje XML, secuencias estandarizadas y columnas con valores autogenerados.

En 2006 se presenta una nueva versión del lenguaje estandarizada por ISO/IEC 9075-14:2006, que define los modos en los que el lenguaje SQL puede ser usado en conjunción con el lenguaje XML. Define modos de importación y almacenamiento de datos XML en una base de datos SQL, manipulación y publicación de los datos en XML y SQL convencional en formularios XML.

En 2008 se realiza una revisión del estándar anterior para añadir el uso de la sentencia `ORDER BY` en la definición de cursores externos, adición de la instrucción `INSTEAD OF` en los triggers y adición de la sentencia `TRUNCATE.` Esta versión del lenguaje SQL es la última que ha aparecido hasta el momento y es la que se encuentra implementada en la versión 12c de Oracle.

SQL STANDARD

En este apartado se realiza una breve descripción de las características de las distintas versiones estandarizadas del lenguaje SQL.

SQL/86

La historia de SQL (Structured Query Language) empieza en 1974 con la definición por parte de Donald Chamberlin y de otras personas que trabajaban en los laboratorios de investigación IBM, de un lenguaje para la especificación de las características de las bases de datos que adoptaban el modelo relacional. Este lenguaje se llamaba SEQUEL (Structured English Query Language) y se implementó en un prototipo llamado SEQUEL-XRM entre 1974 y 1975. Los experimentos con ese prototipo condujeron entre 1976 y 1977 a una revisión del lenguaje (SEQUEL/2), que a partir de ese momento cambió de nombre por motivos legales, convirtiéndose en SQL.

El prototipo (System R) basado en este lenguaje se adoptó y utilizó internamente en IBM, y lo adoptaron algunos de sus clientes elegidos. Gracias al éxito de este sistema, que no estaba todavía comercializado, también otras compañías empezaron a desarrollar sus productos relacionales basados en SQL. A partir de 1981 IBM comenzó a entregar sus productos relacionales, y en 1983 empezó a vender DB2. En el transcurso de los años ochenta numerosas compañías (por ejemplo, Oracle y Sybase, solo por citar algunas), comercializaron productos basados en SQL, que se convierte en el estándar industrial de hecho, por lo que respecta a las bases de datos relacionales.

En 1986 el ANSI adoptó SQL (sustancialmente adoptó el dialecto SQL de IBM) como estándar para los lenguajes relacionales, y en 1987 se transformó en estándar ISO. Esta versión del estándar tenía el nombre de SQL/86.

SQL/89

En 1989 ANSI definió el SQL/89, basado en el anterior, pero con una serie de mejoras: definición de claves primarias, integridad de los datos, etc. Una característica importante definida era la posibilidad de utilizarse a través de dos interfaces: interactivamente o dentro de programas de aplicación.

En su primera versión del SQL-89 se tienen tres partes:

- **El lenguaje de definición de datos (LDD).** Contiene todas las instrucciones para definir el esquema de una base de datos, como son: `create`, `alter` y `drop`.
- **El lenguaje de manipulación de datos (LMD).** Contiene las instrucciones de manejo de las tablas, como son: `select`, `insert`, `delete` y `update`, y para control de concurrencia, como son: `commit` y `rollback`.
- **El lenguaje de control de datos (LCD).** Contiene aquellas instrucciones para dar y revocar permisos de acceso a los datos de la base de datos, como son: `grant` y `revoke`.

Todas las instrucciones pueden ir embebidas en programas escritos en otros lenguajes de programación, como: Cobol, Fortran, Pascal y PL/1.

Todas las sentencias SQL comienzan con un verbo, una palabra clave que describe lo que la sentencia hace: `CREATE`, `INSERT`, `DELETE`, `COMMIT` son verbos típicos.

La sentencia continúa con una o más cláusulas. Una cláusula puede especificar los datos sobre los que debe actuar la sentencia, o proporcionar más detalles acerca de lo que la sentencia debe hacer. Todas las cláusulas comienzan también con una palabra clave, tal como `WHERE`, `FROM`, `INTO` y `HAVING`. Algunas cláusulas son opcionales y otras necesarias. La estructura y el contenido específico varían de una cláusula a otra. Muchas cláusulas contienen nombres de tablas o columnas; algunas pueden contener palabras claves adicionales, constantes o expresiones.

SQL/92

SQL/92 fue desarrollado por el comité técnico NCITS H2 sobre bases de datos. SQL/92 fue diseñado para ser un estándar en los sistemas manejadores de bases de datos relacionales (RDBMS) y está basado en su antecesor SQL/89.

En 1992 aparece SQL2 o SQL/92, la versión hoy en día más difundida ([ISO/IEC 1992] [ANSI 1992] [ISO/IEC 1994]). Con la aparición de la segunda versión del estándar (SQL2) en 1992, prácticamente todos los RDBMS, incluso los no relacionales, incluían soporte a SQL. Actualmente, SQL se ha convertido en el lenguaje de consulta más utilizado.

SQL, además de permitirnos consultas en la base de datos, contiene primitivas de definición de tablas, actualización de la base de datos, definición de vistas, otorgamientos de privilegios, etc.

SQL3

El lenguaje estándar llamado SQL3 prometió ser un aumento de la segunda generación de SQL (comúnmente conocido como SQL92).

SQL3 fue originalmente planeado para su uso en el año 1996, pero tardó 7 años en desarrollarse, en vez de los tres o cuatro que se pensaba iba a tardar.

SQL3 se podría definir como "SQL orientado a objetos" y es la base de algunos sistemas de manejo de bases de datos orientadas a objetos (incluyendo ORACLE, Informix Universal Server, IBM's DB Universal Database y Cloudscape, además de otros).

SQL3 envuelve características adicionales que se consideran herencia de los SQL relacionales, así como también una reestructuración de los documentos de los estándares, con vista a una mayor progresión hacia normas más efectivas.

Las dos organizaciones que se involucraron en la estandarización de SQL, y por lo tanto en el desarrollo de SQL3 son ANSI e ISO. Más específicamente, la comunidad internacional de trabajos, mediante ISO/IEC JTC1 (Joint Technical Committee 1), un comité formado por la organización internacional de estandarización junto con la comisión internacional electrotécnica.

Los aspectos más relevantes de esta versión del lenguaje SQL se dividen en "aspectos relacionales" y "aspectos relacionados con objetos".

- Aspectos relacionales:
  - Nuevos tipos de datos: LARGE OBJECT (LOB), BOOLEAN.
  - Nuevos tipos de datos compuestos: ARRAY, ROW.
  - Nuevos predicados: SIMILAR, DISTINCT.
  - Nueva semántica: ampliación de las clases de vistas, llamadas recursivas, locators (localizadores), savepoints (puntos de ruptura para deshacer operaciones).
  - Mejoras de seguridad: privilegios otorgados por roles.
  - Base de datos activa: triggers (disparadores).

Además de las características discutidas hasta ahora, SQL3 se caracteriza porque fue desarrollado principalmente para manejar objetos. Algunas de las características que están dentro de esta categoría fueron definidas en el estándar SQL/PSM publicado en 1996, específicamente para llamadas a funciones y procedimientos desde SQL.

SQL3 mejora esta capacidad que llamó SQL-invoked routines, para añadir una tercera clase de rutina conocida como método.

- Aspectos relacionados con objetos:
  - Tipos de estructuras definidas por el usuario: CREATE TYPE.
  - Funciones versus métodos: SQL3 hace una importante distinción entre las típicas llamadas a funciones y las llamadas a métodos. En resumen, un método es una función con varias restricciones y aumentos. Dejaremos resumidas las diferencias entre los dos tipos de rutina: Los métodos están estrechamente limitados a un simple tipo definido por el usuario. Las funciones pueden ser polimórficas (sobrecargadas).
  - Notaciones funcionales y de punto.
  - Objetos: tipo REF.

SQL/2006

ISO/IEC 9075-14:2006 define las maneras en las que el lenguaje SQL se puede utilizar conjuntamente con el lenguaje XML.

Define los mecanismos para importar y guardar datos XML en una base de datos SQL, manipulándolos dentro de la misma, y publicando los datos XML y los datos SQL convencionales en forma XML.

Además, proporciona facilidades que permiten a las aplicaciones integrar dentro de su código SQL el uso de Xquery; lenguaje de consulta XML publicado por el W3C (World Wide Web Consortium) para acceso concurrente a datos ordinarios SQL y documentos XML.

BENEFICIOS DEL LENGUAJE SQL

La creación del lenguaje SQL ha permitido (con sus distintas estandarizaciones) una serie de beneficios en los accesos a las bases de datos relacionales, que se resumen en los siguientes puntos:

- Se pueden procesar conjuntos de datos como grupos en vez de como unidades individuales.
- Se proporcionan mecanismos para la navegación por los datos.
- Existen instrucciones muy potentes individualmente que proporcionan tareas variopintas:
    - Consulta de datos.
    - Inserciones, modificaciones y borrado de filas de una tabla.
    - Creación, reemplazo, alteración y borrado de objetos.
    - Control de acceso a la base de datos y sus objetos.
    - Garantizadores de consistencia e integridad.
- Permite el uso de SQL embebido (se puede incluir dentro de otros lenguajes de programación).

ESTRUCTURA DEL LENGUAJE SQL

El lenguaje SQL se agrupa en las siguientes categorías de instrucciones:

- DDL (Data Definition Language).
- DML (Data Manipulation Language).
- Órdenes de control de transacciones.
- Órdenes de control de sesión.
- Órdenes de control del sistema.

Sublenguaje DDL

Permite la creación y manipulación de objetos y derechos de la base de datos, y consta de las siguientes instrucciones básicas:

- CREATE: Creación de objetos.
- DROP: Borrado de objetos.
- ALTER: Alteración de objetos existentes.
- EXECUTE: Ejecución de código.
- GRANT: Asignación de permisos.
- REVOKE: Revocación de permisos.

Sublenguaje DML

Permite la manipulación de los datos contenidos en las tablas de una base de datos (consulta, modificación, inserción y borrado). Consta de las siguientes instrucciones básicas:

- SELECT: Consulta de elementos.
- INSERT: Inserción de nuevos elementos.
- DELETE: Borrado de elementos existentes.
- UPDATE: Actualización de elementos existentes.

Sublenguaje de control de transacciones

Son las órdenes encargadas de garantizar la consistencia de los datos en las operaciones concurrentes sobre la base de datos. Consta de las siguientes instrucciones básicas:

- COMMIT: Validar los cambios.
- ROLLBACK: Rechazar los cambios.
- SAVEPOINT: Punto sin retorno.

Sublenguaje de control de sesión

Son las órdenes que permiten cambiar las opciones de una conexión determinada a la base de datos. Consta de las siguientes instrucciones básicas:

- ALTER SESSION: Cambio de parámetros.
- SET ROLE: Asignación de un role de permisos.

Sublenguaje de control del sistema

Son las órdenes que afectan a la base de datos completa, como por ejemplo la activación o desactivación del archivado definitivo. Consta de la siguiente instrucción básica:

- ALTER SYSTEM: Cambios en la configuración del SGBD.

CONVENCIONES LÉXICAS

Las convenciones léxicas hacen referencia a la implementación de sentencias SQL.

Las expresiones que se diseñan con instrucciones SQL permiten a parte de las propias palabras reservadas del lenguaje, los siguientes elementos léxicos:

- Tabuladores.
- Retornos de línea.
- Espacios.
- Comentarios.

Por ejemplo la siguiente instrucción es válida:

```
SELECT ENAME, SAL*12, MONTHS_BETWEEN(HIREDATE,SYSDATE)
FROM EMP
```

Y la siguiente también, aunque incluya tabuladores, retornos de línea y espacios.

```
SELECT ENAME,
     SAL*12,
          MONTHS_BETWEEN(HIREDATE,SYSDATE)
FROM
     EMP;
```

ELEMENTOS DE SQL 3

INTRODUCCIÓN

En el lenguaje SQL se definen los siguientes elementos básicos:

- Literales o valores constantes.
- Textos.
- Enteros.
- Números.
- Tipos de datos.
- Nulos.
- Pseudocolumnas.
- Comentarios.
- Objetos de base de datos.
- Esquemas de nombres de objeto y calificadores.
- Referenciación a esquemas de objetos.
- Referenciación a base de datos remotas.

LITERALES O VALORES CONSTANTES

Los términos "literal" o "valor constante" son sinónimos y se refieren a valores de datos fijos. Se tiende a llamar a los valores constantes de tipo carácter con el nombre de "literal" y a los "valores constantes" de tipo numérico como tales.

Por ejemplo, los siguientes datos se consideran "literales" o "valores constantes":

- `'JOSE', '101'` son "literales" de tipo carácter.
- `5000, -50, 35.21` son "valores constantes" de tipo numérico.

Como se puede comprobar por los ejemplos anteriores, todo "literal" de tipo carácter tiene que ser encerrado entre comillas simples (').

TEXTOS

Un texto especifica un conjunto de caracteres o un literal de tipo carácter.

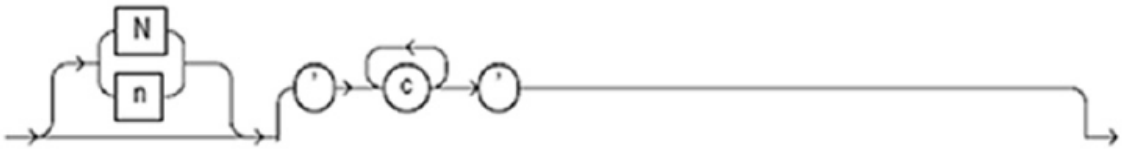


Fig. 3-1 Sintaxis de un texto.

N: especifica que el texto que se encierre a continuación de las comillas automáticamente será traducido por Oracle, al lenguaje nacional que se haya indicado en la instalación del SGBD.

C: indica el carácter o conjunto de caracteres introducidos en el texto.

Por último, el texto se encierra entre comillas: '.

Un texto se asimila en cuanto a tipos de datos Oracle a un tipo CHAR o VARCHAR2.

El tamaño máximo de un texto es de 4.000 bytes.

Ejemplos de textos válidos son los siguientes:

- `'Hola'`
- `'Oracle.system'`
- `'09-mar-99'`
- `N'hello'` En este caso, el texto `'hello'` será traducido al lenguaje que se haya definido para el SGBD de Oracle.

La configuración del lenguaje que se utilice para el acceso a la información de la base de datos se realiza tanto en el cliente que interroga la base de datos, como en los parámetros internos del SGBD.

Dentro del cliente hay que configurar el **Regedit** de Windows, en la sección `HKEY_LOCAL_MACHINE`. Dentro de esta sección encontraremos una carpeta `SOFTWARE`, y dentro de la misma una de `ORACLE`. En cada carpeta de esta última habrá que configurar el parámetro `NLS_LANG`, al valor **SPANISH_SPAIN.WE8MSWIN1252**. Este juego de caracteres es el que habitualmente se utiliza para la lengua española, incluyéndose dentro de dicho juego la Ñ y símbolos especiales como el €.

En la propia base de datos el juego de caracteres se indica en el parámetro de inicialización de la misma, denominado `NLS_CHARACTERSET`.

Para comprobar el valor que tiene este parámetro, tenemos que ejecutar la siguiente sentencia SQL:

```
SELECT VALUE FROM NLS_DATABASE_PARAMETERS
WHERE PARAMETER = 'NLS_CHARACTERSET';
```

Lo ideal es tener el mismo juego de caracteres tanto en la base de datos como en el cliente que accede a la misma. O bien tener un juego de caracteres compatibles entre ambos. Por tanto, si quisiéramos igualar el juego de caracteres de la base de datos al del cliente indicado anteriormente, el valor de `NLS_CHARACTERSET` tendría que ser: WE8MSWIN1252.

ENTEROS

Un entero es un valor numérico sin decimales, con un máximo de 38 dígitos de precisión.

Un número entero puede ir precedido de signo positivo (por defecto si no se indica nada) o negativo.

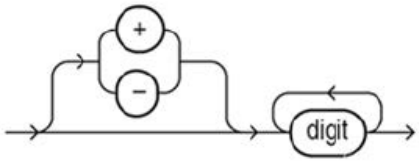


Fig. 3-2 Sintaxis de un entero.

Donde **digit** corresponde a un dígito entre los siguientes: 0, 1, 2, 3, 4, 5, 6, 7, 8 o 9.

Ejemplos de enteros válidos son los siguientes:

```
7, +255, -300
```

NÚMEROS

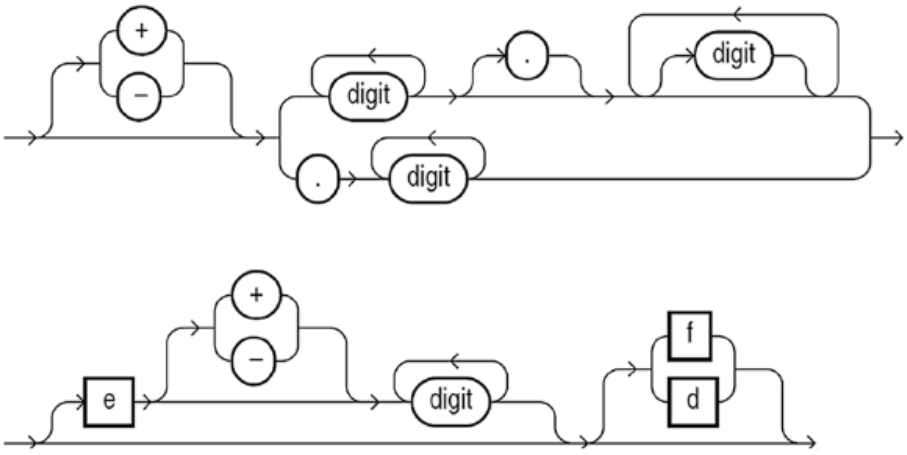


Fig. 3-3 Sintaxis de un número.

Donde el símbolo . (punto) indica el separador decimal de un número, siempre y cuando se haya definido dentro de la configuración de instalación de Oracle. Si se ha

indicado notación castellana, será la "coma" (**,**) el símbolo que se utilizará como separador decimal.

E o **e** indican que el dígito que viene a continuación está en notación exponencial. El rango de un exponente va de -130 a 125. **F** o **f** indica que es un número en coma flotante de 32 bits, mientras que **D** o **d** indica que es de 64 bits.

Un número puede almacenar hasta 38 dígitos de precisión.

Ejemplos de números válidos son los siguientes:

```
25    +6.34    0.5    25e-03    -1    25f    0.5d
```

TRATAMIENTO DE VALORES NULOS

Las normas para la interpretación y el tratamiento de los valores nulos (vacíos) en una base de datos Oracle son muy importantes, para poder implementar correctamente aplicaciones contra la misma.

Las normas a considerar son las siguientes:

- Las columnas de una tabla que no tienen valores (están vacías), en realidad tienen almacenado el valor **NULL**.
- El valor 0 (cero) de una columna es distinto del valor NULL.
- La función para la conversión de los valores nulos de una columna es `NVL`, cuya sintaxis es la siguiente:

  ```
  NVL(columna, valor_convertido_si_es_nulo).
  ```

  Donde **columna** es el nombre de la columna que se está evaluando para identificar si tiene un valor nulo o no, y **valor_convertido_si_es_nulo** es el valor en el que se convierte el contenido de la columna si estaba vacío (tenía un NULL).

  Un ejemplo de uso de la función `NVL` es la siguiente:

  ```
  NVL(salario,0)
  ```

  En este ejemplo si la columna salario tiene en alguna de sus filas un valor NULL (vacío) se convertirá automáticamente a valor 0 (cero).

- Las operaciones aritméticas con valores nulos devuelven siempre el valor NULL como resultado.
- Las operaciones de comparación con valores nulos devuelven siempre el valor DESCONOCIDO.

El valor NULL aparece en aquellas columnas que no posean todavía un valor asignado siempre y cuando no se hayan restringido en la creación de la tabla, mediante una restricción (constraint) de tipo NOT NULL o PRIMARY KEY.

Un ejemplo de operación aritmética que puede verse afectada si existe un valor nulo es el siguiente:

```
1000 + salario / 2
```

En el ejemplo anterior si la columna salario tiene en alguna fila un valor NULL (vacío), el resultado de la operación aritmética será NULL. En el resto de los casos se devolverá el resultado de la operación.

Un ejemplo de operación de comparación entre valores que puede verse afectada por un valor nulo es el siguiente:

```
Salario > 1000
```

La siguiente tabla muestra los posibles resultados que ofrecería esta operación de comparación, dependiendo de los valores de la columna salario, donde FALSE sería falso y TRUE verdadero.

Si SALARIO es	Condición	Resultado devuelto
1000	SALARIO > 1000	FALSE
> 1000	SALARIO > 1000	TRUE
NULL	SALARIO > 1000	DESCONOCIDO
< 1000	SALARIO > 1000	FALSE

Para completar las casuísticas que nos podemos encontrar al evaluar una operación de comparación en la que puede haber valores nulos, se muestra la siguiente tabla.

Si SALARIO es	Condición	Resultado devuelto
1000	SALARIO IS NULL	FALSE
1000	SALARIO IS NOT NULL	TRUE
NULL	SALARIO IS NULL	TRUE
NULL	SALARIO IS NOT NULL	FALSE
1000	SALARIO = NULL	DESCONOCIDO
1000	SALARIO <> NULL	DESCONOCIDO
NULL	SALARIO = NULL	DESCONOCIDO
NULL	SALARIO <> NULL	DESCONOCIDO
NULL	SALARIO = 1000	DESCONOCIDO
NULL	SALARIO <> 1000	DESCONOCIDO

Cuando se quiere realizar una operación de comparación para comprobar si un valor es nulo, no se puede utilizar la expresión del tipo SALARIO = NULL, sino que hay que utilizar la expresión SALARIO IS NULL.

Lo mismo ocurre con la negación; si queremos comprobar si un valor no es nulo, no se puede utilizar la expresión del tipo SALARIO <> NULL, sino que hay que utilizar la expresión SALARIO IS NOT NULL.

PSEUDOCOLUMNAS

Una pseudocolumna actúa como una columna, pero no se almacena en una tabla. Se pueden realizar consultas (SELECT) sobre pseudocolumnas, pero no se pueden insertar, modificar o borrar sus valores.

Existen los siguientes tipos de pseudocolumnas:

- Para el uso de secuencias se utilizan CURRVAL y NEXTVAL.
- Para el uso de consultas jerárquicas se utiliza LEVEL.
- Para el uso de indicadores de fila de una tabla se utilizan ROWID y ROWNUM.

Currval y Nextval

Estas pseudocolumnas están ligadas al uso de los objetos secuencia (SEQUENCE).

Una secuencia es un esquema de objeto perteneciente a un esquema, que permite generar de forma automática un contador de números. Estos valores a menudo se utilizan como claves primarias o claves únicas.

- CURRVAL: nos devuelve el valor actual que posee la secuencia consultada.
- NEXTVAL: nos devuelve el siguiente valor para la secuencia consultada.

Antes de cada una de las pseudocolumnas, hay que especificar el nombre de la secuencia que se está consultando.

En el siguiente ejemplo se puede comprobar esta nomenclatura:

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

También se puede hacer referencia a secuencias que sean propiedad de otro usuario, siempre y cuando tengamos derecho sobre ellas, anteponiendo el nombre del usuario al de la secuencia. En el siguiente ejemplo se puede comprobar esta nomenclatura:

```
SELECT CURSOSQL.SEQ_CONTEO.CURRVAL FROM DUAL;
SELECT CURSOSQL.SEQ_CONTEO.NEXTVAL FROM DUAL;
```

Y por último podemos hacer referencia a secuencias que estén en bases de datos remotas, utilizando un enlace que se haya definido previamente. En el siguiente ejemplo se puede comprobar esta nomenclatura:

```
SELECT CURSOSQL.SEQ_CONTEO.CURRVAL@BDREMOTA2 FROM DUAL;
SELECT CURSOSQL.SEQ_CONTEO.NEXTVAL@BDREMOTA2 FROM DUAL;
```

Para completar el aprendizaje sobre el manejo de las secuencias, en el siguiente ejemplo se muestra el código de creación de una secuencia y sus posteriores consultas utilizando pseudocolumnas:

```
CREATE SEQUENCE SEQ_CONTEO START WITH 1 INCREMENT BY 1;

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

Level

Esta pseudocolumna define un tipo de consulta ramificada en la que existen nodos padres y nodos hijos. La pseudocolumna LEVEL nos irá indicando en qué nivel del árbol nos encontramos.

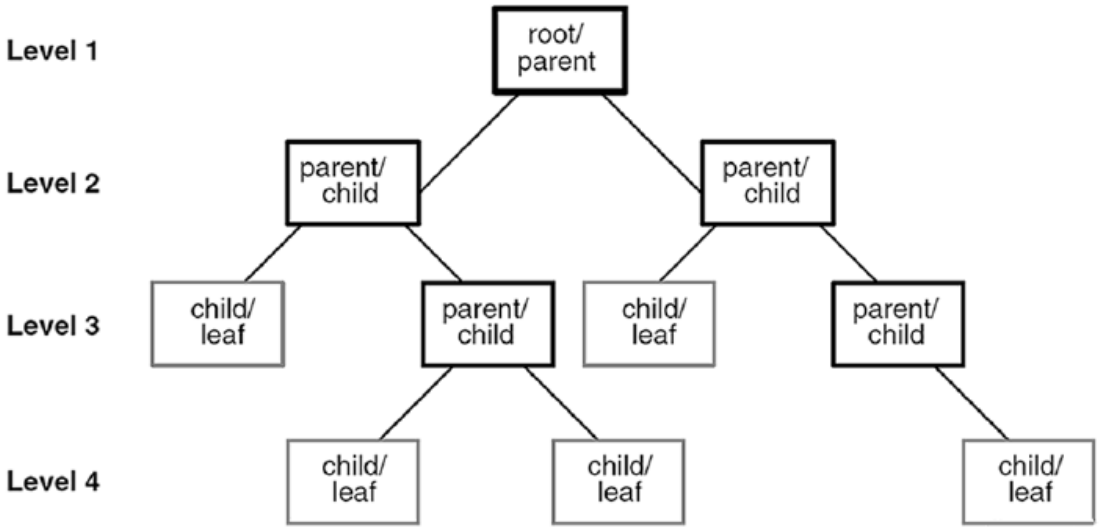


Fig. 3-4 Esquema de ejecución de una consulta jerárquica usando LEVEL.

El tipo de estructura representado en la figura se genera utilizando una instrucción del tipo SELECT, combinada con las cláusulas START WITH y CONNECT BY.

Rowid

Esta pseudocolumna contiene la dirección lógica de acceso a la información física de la fila consultada de la tabla en la base de datos.

En el capítulo 4 se especificarán con más detalle su significado y su contenido, al ser también un tipo de datos.

Su importancia radica en que:

- Es un método más rápido de obtener una fila de una tabla.
- Muestra cómo está almacenada la fila.
- Es un identificador único de una fila de la tabla, por lo que si no existe una clave primaria, siempre podremos recurrir a esta pseudocolumna para identificar unívocamente cada fila de la tabla.

Un ejemplo de consulta de esta pseudocolumna es el siguiente:

```
SELECT ROWID, cod_cliente FROM empleado
WHERE cod_departamento = 20;
```

Rownum

Esta pseudocolumna contiene el número indicativo del orden en que Oracle está devolviendo las filas de una consulta.

Un ejemplo de consulta de esta pseudocolumna es el siguiente:

```
SELECT ROWNUM, cod_cliente FROM empleado
WHERE ROWNUM < 10;
```

En este ejemplo se van a seleccionar las 9 primeras filas de la tabla empleado, mostrándose el número de fila y el código de cliente.

COMENTARIOS

Dentro del código SQL o PL/SQL se pueden añadir comentarios para mejorar o facilitar la comprensión del mismo, cuando vaya a ser revisado por el programador u otra persona. Un comentario presenta las siguientes características:

- Se puede incluir en el código SQL y también en el lenguaje PL/SQL.
- Puede ocupar 1 o más líneas.

Cuando el comentario ocupa 1 línea habrá que utilizar la notación: `--texto`. A continuación, se muestra un ejemplo de esta notación:

```
SELECT codigo, nombre_empleado   -- código es el dni
FROM empleado
WHERE salario > 1000.
```

Cuando el comentario ocupa varias líneas habrá que utilizar la notación: `/* texto */`. A continuación, se muestra un ejemplo de esta notación:

```
/* Esta consulta nos devuelve
   el dni y el nombre del empleado */

SELECT codigo, nombre
FROM empleado
WHERE salario > 1000.
```

OBJETOS DE LA BASE DE DATOS

En una base de datos de Oracle nos encontraremos objetos agrupados por esquemas y objetos sin esquema.

Las características de los esquemas son:

- Un esquema es una colección de estructuras lógicas de datos o de otros esquemas de objetos.
- Son propiedad de un usuario de la base de datos.
- Cada usuario de base de datos tiene su propio esquema.
- Cada esquema se crea y manipula mediante instrucciones SQL.

Los tipos de objetos que puede contener un esquema son:

- Clusters.
- Constraints (restricciones).
- Enlaces de base de datos.
- Dimensiones.
- Librerías de procedimientos externos.
- Tablas de índices organizados.
- Índices.
- Tipos Índice.
- Clases, recursos y código fuente de Java.
- Vistas materializadas.
- Logs de vistas materializadas.
- Modelos de Data Mining.
- Tablas Objeto.
- Tipos Objeto.
- Vistas Objeto.
- Operadores.
- Paquetes.
- Secuencias.
- Funciones y procedimientos almacenados.
- Sinónimos.
- Tablas.
- Vistas.

Los objetos que no forman parte de ningún esquema son:

- Contextos.
- Directorios.
- Ediciones.
- Puntos de restauración.
- Roles.
- Segmentos Rollback.
- Tablespaces.
- Usuarios.

NOMBRES DE OBJETO Y CALIFICADORES

Para la identificación de objetos y elementos dentro de una base de datos, se utilizan nombres y calificadores. Las reglas que deben cumplir son las siguientes:

- Los nombres de objetos pueden tener de 1 a 30 caracteres, excepto:
    - Los nombres de base de datos, que solo pueden tener de 1-8 caracteres.
    - Los nombres de enlace a otras bases de datos, que solo pueden tener hasta 128 caracteres.
- Los nombres no pueden llevar comillas como parte del mismo.
- No afecta el uso de mayúsculas/minúsculas salvo que se encierre todo el nombre entre comillas dobles (").
- Un nombre siempre tiene que comenzar por un carácter alfabético.
- Solo se admite como composición de los nombres los caracteres alfanuméricos, y los signos _ $ # . @
- No se admiten palabras reservadas (tampoco la palabra reservada DUAL).
- Cada objeto tiene un espacio donde se almacena su nombre.
- En los espacios que se reservan para los nombres de objetos solo se admite un mismo nombre de objeto.
- No se admiten iguales nombres de columna en la misma tabla o vista, pero sí en distintas.
- Los objetos encerrados entre dobles comillas (") admiten espacios y resto de signos especiales.

Los espacios de nombres de objeto aglutinan una serie de nombres de objeto, en algunos casos de forma independiente, y en otros común para distintos nombres de objetos diferentes.

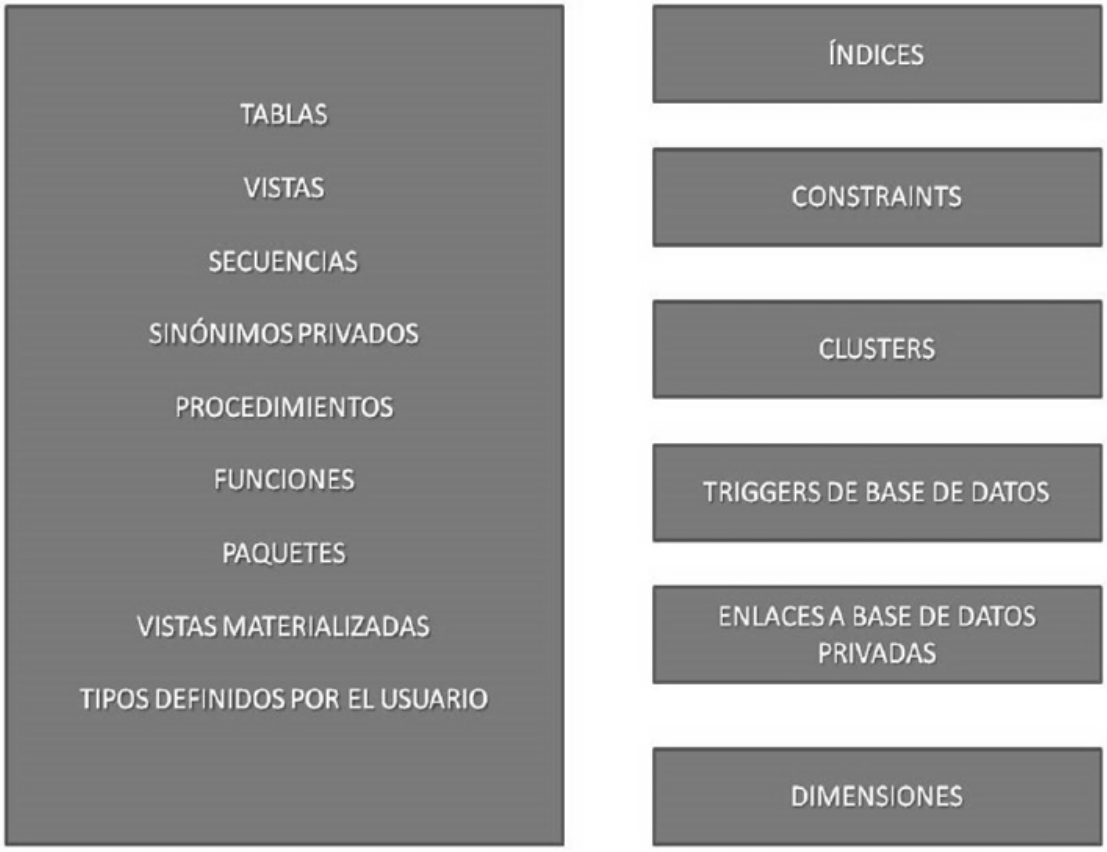


Fig. 3-5 Namespaces (Esquemas de nombre) de objetos de un esquema.

Dentro del mismo espacio de nombres no se puede repetir un nombre de objeto aunque sea de distinto tipo.

Por ejemplo, una tabla y una vista no se pueden denominar con el mismo nombre, porque comparten el mismo esquema de nombres. En cambio una restricción y un trigger sí se pueden llamar igual, porque cada uno de estos objetos del esquema de base de datos poseen esquemas independientes.

Fig. 3-6 Namespaces de objetos que no pertenecen a un esquema.

Ejemplos de nombres de esquema válidos son los siguientes:

```
EMP
"Emp"
SCOTT.FECHAALTA
"INCLUSO ESTO & VALE!"
UN_NOMBRE_LARGO_Y_VALIDO
```

REFERENCIANDO A ESQUEMAS DE OBJETOS

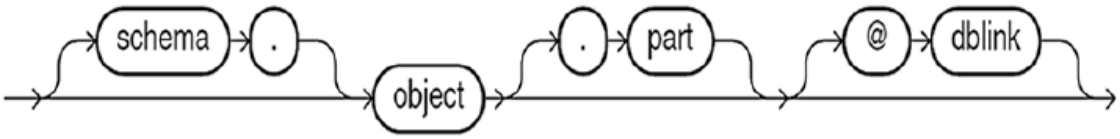


Fig. 3-7 Sintaxis para referenciar un esquema.

Schema es el nombre del esquema referenciado.

Object es el nombre del objeto perteneciente al esquema que se está referenciando. Cuando el objeto se encuentra en el mismo esquema al que pertenece el usuario que realiza la consulta, el nombre del esquema se puede omitir. Cuando se indica nombre de esquema, el nombre del objeto va antepuesto por un punto.

Part es un elemento del objeto consultado, como por ejemplo una columna de una tabla. Cuando se utiliza hay que anteponerlo con un punto.

Dblink es el nombre de una base de datos que contiene un objeto. Cuando no se indica, por defecto se accede a la propia base de datos a la que se encuentra conectado el usuario.

Ejemplos de referenciación de esquemas son los siguientes:

```
SELECT MIESQUEMA.EMPLEADO.COL1 FROM MIESQUEMA.EMPLEADO;
SELECT MIESQUEMA.EMPLEADO.COL1
FROM MIESQUEMA.EMPLEADO@ENLACEBD.
```

REFERENCIANDO A BASES DE DATOS REMOTAS

Para poder referenciar a una base de datos remota, primero hay que crear un enlace a la misma, dentro de la propia base de datos desde la que se va a acceder.

Fig. 3-8 Sintaxis para la creación de un enlace.

Database es el nombre de la base de datos remota (el SID de la misma).

Domain especifica el dominio de conexión a la base de datos remota. Si se omite, se asume el dominio que tenga la base de datos actual desde la cual creamos el enlace.

Connect_descriptor es el descriptor completo del ordenador donde reside la base de datos.

Ejemplos de código para la creación de un enlace a una base de datos externa son los siguientes:

```
CREATE DATABASE LINK mi_enlace
sales.us.america.acme_auto.com;

CREATE DATABASE LINK mi_enlace2 CONNECT TO hr
IDENTIFIED BY mipassword USING 't:host1.db1';
```

En el último ejemplo se crea un enlace `mienlace2` a una base de datos `db1` del servidor `host1`, mediante un protocolo TCP/IP (indicado con la letra `t`), conectándonos a la misma con el usuario `hr` y con la password `mipassword`.

En el siguiente ejemplo se muestra cómo se puede utilizar un enlace creado en base de datos contra otra base de datos externa:

```
SELECT * FROM empleado@mi_enlace2;
```

SUPUESTO PRÁCTICO 1: ***Resolución en el Anexo I.***

Extraer las entidades del supuesto con indicación de los atributos, dominios, relaciones y todas las claves que existan, diseñando un modelo relacional, y opcionalmente, si se desea, diseñar previamente un modelo entidad/relación.

- Se necesitan reflejar los datos de los hospitales que hay en la geografía española con la siguiente información básica:
  - Un código de hospital único con independencia de la localización geográfica del centro. Este código almacenará valores de 1 a 99.
  - El nombre del hospital de 12 caracteres alfabéticos.
  - La dirección de 50 caracteres alfanuméricos.
  - El teléfono de 9 dígitos.
  - El número de camas del hospital con valores de 1 a 9.999.

- A cada hospital asisten diariamente enfermos con diversas afecciones de los que queremos reflejar la siguiente información:
  - El código del hospital al que asisten.
  - El número de inscripción del hospital en cuestión, que será único para cada asistencia que se produzca dentro de un hospital. Los valores del número variarán entre 1 y 99.999.999.
  - Nombre de 20 caracteres alfabéticos.
  - Apellidos de 40 caracteres alfabéticos.
  - Dirección de 50 caracteres alfanuméricos.
  - Fecha de nacimiento.
  - Sexo que solo podrá ser M o F.
  - Número de la Seguridad Social de 9 dígitos.
  - Fecha de inscripción en el hospital.

- Cada hospital cuenta con una serie de salas donde se aplican tratamientos diversos. De ellas se quiere almacenar la siguiente información:
  - Código del hospital en el que se encuentra la sala.
  - Código de la sala que será único dentro del mismo hospital y que variará de 1 a 99.
  - Nombre de la sala de 30 caracteres alfanuméricos.
  - Número de camas de las que dispone la sala y que variará de 0 a 99.

- En cada hospital trabajan una serie de doctores que aplican terapias a sus pacientes, siendo necesario que el esquema refleje datos de los mismos:
  - El código del hospital en el que trabajan o ha trabajado.
  - Nif de 9 caracteres alfanuméricos.
  - Nombre de 20 caracteres alfabéticos.
  - Apellidos de 40 caracteres alfabéticos.
  - Especialidad en la que trabaja el doctor de 20 caracteres alfabéticos.

- Aparte de los doctores, en cada hospital trabaja una plantilla sanitaria que cubre el resto de áreas de sanidad del hospital. Para cada uno de ellos se quiere reflejar:
  - El código del hospital en el que trabajan o han trabajado.
  - El código de la sala en la que trabajan o han trabajado.
  - NIF de 9 caracteres alfanuméricos.
  - Nombre de 20 caracteres alfabéticos.
  - Apellidos de 40 caracteres alfabéticos.
  - Función desempeñada en la sala donde trabaja (de 20 caracteres alfabéticos).
  - Turno que tienen asignado en la sala en la que se encuentra. Solo podrá ser M,T,N.
  - Salario anual que cobran en euros (5 dígitos enteros y 2 decimales).

- Además de las salas, un hospital también consta de departamentos donde trabaja el personal no sanitario. Queremos diseñar una lista de departamentos universales que sirva para cualquier hospital, y para ello necesitamos la siguiente información:
  - Código del departamento único de 1 a 99.
  - Nombre del departamento de 20 caracteres alfabéticos.

- También existe una plantilla no sanitaria que trabaja en el hospital desempeñando otras tareas que mantienen en funcionamiento todos los servicios del hospital y que abarcan desde la dirección del hospital hasta el servicio de limpieza del mismo. Queremos incluir en nuestro esquema la siguiente información al respecto.
  - El código del hospital en el que trabajan o han trabajado.
  - NIF de 9 caracteres alfanuméricos.
  - Nombre de 20 caracteres alfabéticos.
  - Apellidos de 40 caracteres alfabéticos
  - Oficio que desempeñan de 20 caracteres alfanuméricos.
  - NIF de la persona de la que dependen funcionalmente.
  - Fecha de alta en la empresa.
  - Salario mensual en euros (4 enteros y 2 decimales).
  - Comisión en euros (4 enteros y 2 decimales).
  - Código del departamento en el que trabaja o ha trabajado.

TIPOS DE DATOS 4

TIPOS NUMÉRICOS

Existen 3 tipos de datos básicos para el manejo de los números:

- NUMBER
- BINARY_FLOAT
- BINARY_DOUBLE

Number

El tipo NUMBER puede contener valores enteros o decimales.

La sintaxis es: `NUMBER(P,S)`

Donde **P** es la precisión o parte entera del número que admite un máximo de 38 dígitos.

Donde **S** es la escala o parte decimal del número que admite un rango entre -84 y 127.

Un ejemplo de código SQL para la definición de tipos de datos `NUMBER` y asignación de valores a los mismos es el siguiente:

```
DECLARE
   Valor1     NUMBER;
   Valor2     NUMBER(3);
   Valor3     NUMBER(3,2);
BEGIN
   Valor1 := 123.89;
   Valor2 := 123;
   Valor3 := 123.89;
   Valor2 := 123.45;     -- El sistema almacena 123
   Valor2 := 123.55;     -- El sistema almacena 124
```

Como se puede apreciar en el ejemplo anterior, cuando se asigna un valor numérico con decimales a un tipo de dato numérico que solo admite valores enteros, como es el caso de la variable VALOR2, el SGBD automáticamente hace un redondeo de la parte decimal a la unidad.

Binary_Float

El tipo BINARY_FLOAT puede contener valores numéricos de coma flotante de hasta 32 bits. Para su almacenamiento requiere 4 bytes.

Los valores máximo y mínimo permitidos son los siguientes:

- Valor mínimo admitido: `1.17549E-38F`.
- Valor máximo admitido: `3.40282E+38F`.

Binary_Double

El tipo BINARY_DOUBLE puede contener valores numéricos de coma flotante de hasta 64 bits. Para su almacenamiento requiere 8 bytes.

Los valores máximo y mínimo permitidos son los siguientes:

- Valor mínimo admitido: `22507485850720E-308`.
- Valor máximo admitido: `1.79769313486231E+308`.

TIPOS CARÁCTER (TIPO TEXTO)

Existen 3 tipos de datos básicos para el manejo de los caracteres o textos:

- VARCHAR2
- CHAR
- LONG

Varchar2

El tipo VARCHAR2 puede contener cadenas de texto de longitud variable en la que se especifique una longitud máxima prefijada.

La sintaxis es: `VACHAR2(L) [CHAR|BYTE]`

Donde **L** es la longitud máxima de una cadena de caracteres admitida. Es obligatorio indicar un valor para este parámetro, y admite valores en el rango de 1 a 32.767 bytes (32K) tanto en PL/SQL como en el tipo asociado a una columna de tabla (esto es una ampliación que se ha producido en la versión 12c de Oracle, dado que anteriormente una columna de una tabla de tipo `VARCHAR2` solo admitía hasta 4.000 bytes).

Para poder extender la capacidad de almacenamiento de las columnas de una tabla de tipo VARCHAR2 hasta los 32k es necesario habilitar el parámetro `MAX_STRING_SIZE` en la inicialización de la base de datos.

```
ALTER SYSTEM SET MAX_STRING_SIZE = ENABLED
```

Se puede especificar el modo de almacenamiento del contenido en formato `CHAR` o `BYTE`. Por defecto, si no se especifica nada se almacena en formato `BYTE.`

Existe un subtipo de datos que se define a partir de este tipo de dato que es el tipo NVARCHAR2.

Char

El tipo CHAR contiene cadenas de texto de longitud fija en la que se puede especificar la longitud de la misma.

La sintaxis es: `CHAR(L) [CHAR|BYTE]`

Donde **L** es la longitud máxima de una cadena de caracteres admitida. Este parámetro no es obligatorio, y si no se especifica longitud, se asume que será 1. El valor máximo permitido es 2.000 bytes.

Se puede especificar el modo de almacenamiento del contenido en formato CHAR o BYTE. Por defecto, si no se especifica nada se almacena en formato BYTE.

Existe un subtipo de datos que se define a partir de este tipo de dato que es el tipo NCHAR.

DIFERENCIA ENTRE EL USO DE LOS TIPOS CHAR Y VARCHAR2

En el caso de usar CHAR, cada cadena ocupa el máximo de la longitud permitida para la columna, mientras que en el caso de un VARCHAR2, cada cadena ocupa su longitud real.

En la tabla siguiente podemos observar la forma en la que el SGBD almacena la misma cadena de texto en formato CHAR y VARCHAR2.

Valor	Almacenamiento en CHAR(5)	Almacenamiento en VARCHAR2(5)
'JJ'	'JJ '	'JJ'
'TACON'	'TACON'	'TACON'
'PEP'	'PEP '	'PEP'

Como se puede observar en la tabla, cuando una cadena tiene menos caracteres que el tamaño de una columna CHAR, el SGBD rellena la misma con espacios en blanco hasta completar el tamaño máximo.

Long

El tipo LONG admite cadenas de caracteres de hasta 2 gigabytes de datos en su almacenamiento dentro de una columna de una tabla de la base de datos. Mientras que en PL/SQL solo admite cadenas de caracteres en el rango de 1 a 32.760 bytes.

Oracle ha mantenido este tipo de datos por compatibilidad con las versiones anteriores de base de datos, pero recomienda que las columnas definidas con este tipo de datos se conviertan a tipos LOB (CLOB, NCLOB o BLOB).

El uso de este tipo de datos presenta una serie de restricciones que se enumeran a continuación:

- Una tabla solo puede contener una columna de tipo LONG.
- No se permite la creación de tipos objeto con un atributo de tipo LONG.
- Las columnas de tipo LONG no pueden aparecer en las condiciones de la cláusula WHERE o en restricciones de integridad (CONSTRAINTS), a excepción de las restricciones de tipo NULL y NOT NULL.
- Las columnas de tipo LONG no pueden estar indexadas.
- Los datos de las columnas de tipo LONG no se pueden utilizar en expresiones regulares.
- Una función almacenada en base de datos no puede devolver un valor de tipo LONG.
- Se admite la declaración de variables o argumentos de una unidad de programa PL/SQL (paquete, procedimiento o función), usando el tipo de dato LONG, pero no se pueden llamar estas unidades de programa desde SQL.
- Las columnas de tipo LONG no pueden ser utilizadas en sentencias SQL distribuidas y tampoco pueden ser replicadas.
- Una columna de tipo LONG no puede aparecer en las siguientes cláusulas de una sentencia SQL: GROUP BY, ORDER BY, CONNECT BY y DISTINCT.
- Una columna de tipo LONG no puede aparecer en el operador UNIQUE de una sentencia SQL.
- Una columna de tipo LONG no puede aparecer en la lista de columnas de una sentencia CREATE CLUSTER.
- Una columna de tipo LONG no puede aparecer en cláusula CLUSTER de una sentencia CREATE MATERIALIZED VIEW.
- Una columna de tipo LONG no puede aparecer en consultas SELECT que contengan subconsultas o consultas combinadas con alguna de las siguientes cláusulas: UNION, INTERSECT o MINUS.
- Una columna de tipo LONG no puede aparecer en la lista de columnas de la sentencia SELECT cuando se utiliza CREATE TABLE…AS SELECT.
- Una columna de tipo LONG no puede aparecer en una sentencia ALTER TABLE…MOVE.
- Una columna de tipo LONG no puede aparecer en la lista de columnas de una sentencia SELECT que se utilice en una subconsulta de la sentencia INSERT.

TIPOS FECHA Y HORA

Existen los siguientes tipos de datos para el manejo de los tipos fecha y hora:

- DATE
- TIMESTAMP
- TIMESTAMP WITH TIME ZONE
- TIMESTAMP WITH LOCAL TIME ZONE
- INTERVAL YEAR TO MONTH
- INTERVAL DAY TO SECOND

Date

El tipo DATE permite almacenar fechas y horas, e incluso ambas a la vez.

Dentro de este tipo de datos se almacenan los siguientes atributos correspondientes a la fecha y la hora:

- Siglo.
- Año.
- Mes.
- Día.
- Hora.
- Minuto.
- Segundo.

El tamaño reservado en base de datos para este tipo de datos es de 7 bytes, reservándose 1 byte por cada uno de los atributos definidos anteriormente.

Timestamp

El tipo TIMESTAMP almacena solo fechas y horas (incluso ambas a la vez).

La sintaxis es: `TIMESTAMP(dígitos)`

Donde **dígitos** especifica la precisión en la parte fraccional de los segundos que se van a utilizar. Por defecto, el valor es 6 y admite de 0 a 9 dígitos.

Timestamp with time zone

El tipo TIMESTAMP WITH TIME ZONE almacena lo mismo que el tipo TIMESTAMP, más los valores de fecha y hora correspondientes a la zona horaria específica para la base de datos.

La sintaxis es: `TIMESTAMP(dígitos) WITH TIME ZONE`

Donde **dígitos** especifica la precisión en la parte fraccional de los segundos que se van a utilizar. Por defecto, el valor es 6 y admite de 0 a 9 dígitos.

Timestamp with local time zone

El tipo TIMESTAMP WITH LOCAL TIME ZONE almacena lo mismo que el tipo TIMESTAMP, más los valores de fecha y hora correspondientes a la zona horaria donde se encuentre físicamente el servidor de base de datos.

La sintaxis es: `TIMESTAMP(dígitos) WITH LOCAL TIME ZONE`

Donde **dígitos** especifica la precisión en la parte fraccional de los segundos que se van a utilizar. Por defecto, el valor es 6 y admite de 0 a 9 dígitos.

Interval year to month

El tipo INTERVAL YEAR TO MONTH almacena un período de tiempo en años y meses.

La sintaxis es: `INTERVAL YEAR(dígitos) TO MONTH`

Donde **dígitos** especifica la precisión correspondiente al año. Por defecto, el valor es 2 y admite de 0 a 9 dígitos.

Interval day to second

El tipo INTERVAL DAY TO SECOND almacena un período de tiempo en días, horas, minutos y segundos.

La sintaxis es: `INTERVAL DAY(dígitos) TO SECOND (dígitos2)`

Donde **dígitos** especifica la precisión correspondiente al día. Por defecto, el valor es 2 y admite de 0 a 9 dígitos.

Donde **dígitos2** especifica la precisión correspondiente a la parte fraccional de los segundos. Por defecto, el valor es 6 y admite de 0 a 9 dígitos.

EJEMPLOS DE TIPO FECHA Y HORA

A continuación, se muestran ejemplos de los tipos fecha y hora vistos anteriormente:

```
TIMESTAMP '1997-01-31 09:26:50.124'
TIMESTAMP '1997-01-31 09:26:56.66 +02:00'
TIMESTAMP '1999-04-15 8:00:00 US/Pacific'
CREATE TABLE time_table (
start_time      TIMESTAMP,
duration_1      INTERVAL DAY (6) TO SECOND (5),
duration_2      INTERVAL YEAR TO MONTH);
```

TIPO ROWID

El tipo ROWID almacena la dirección lógica de ubicación del registro en la tabla. El valor se representa en formato carácter con base 64.

Se puede consultar el ROWID de una fila concreta de una tabla utilizando la pseudocolumna `ROWID`. A continuación, se muestra un ejemplo de cómo hacerlo:

```
SELECT COD_EMPLEADO, ROWID
FROM EMPLEADOS WHERE COD_EMPLEADO = 9999;
```

El resultado de esta consulta sería similar al siguiente:

```
COD_ENTIDAD ROWID
----------- ------------------
       9999 AAA058AAEAAFkJ9AAu
```

La cadena de caracteres que devuelve ROWID contiene la siguiente información:

- El **bloque de datos** del fichero que contiene el registro (la fila consultada en la tabla).
- La **fila** dentro del bloque de datos.

- El **fichero de base de datos** que contiene la fila. El primer fichero de datos tiene el número 1.
- El **número del objeto de datos**, que es un identificativo asignado a cada segmento de la base de datos.

TIPO BOOLEAN

Es un tipo que admite únicamente los siguientes valores:

- TRUE: verdadero.
- FALSE: falso.
- NULL: vacío.

TIPOS LOB

Existen los siguientes tipos LOB:

- CLOB
- NCLOB
- BLOB
- BFILE

Clob

El tipo CLOB almacena objetos de tipo carácter de gran longitud.

Permite almacenar objetos de hasta (4 gigabytes – 1) * (tamaño del bloque de la base de datos).

Existe un subtipo de datos que se define a partir de este tipo de dato que es el tipo NCLOB.

Blob

El tipo BLOB almacena objetos de tipo binario de gran longitud.

Permite almacenar objetos de hasta (4 gigabytes – 1) * (tamaño del bloque de la base de datos).

Bfile

El tipo BFILE almacena un localizador a un fichero binario externo a la base de datos.

Permite enlazar ficheros de hasta un máximo de 4 gigabytes.

TIPOS DEFINIDOS POR EL USUARIO

Existen los siguientes tipos de datos definidos por el usuario:

- Tipos objeto.
- Tipos de datos REF.
- VARRAYS.
- Tablas anidadas.

Tipos objeto

Los tipos objeto son abstracciones de entidades del mundo real, como por ejemplo: reservas de libros.

Un tipo objeto es un esquema de objetos con 3 clases de componentes:

- Un **nombre** que identifica unívocamente el tipo objeto dentro del esquema.
- **Atributo/s**: que serán tipos de datos de los vistos anteriormente, u otros tipos de datos definidos por el usuario. Los atributos modelan la estructura de la entidad del mundo real.
- **Métodos**: que son funciones y procedimientos escritos en PL/SQL y almacenados en la base de datos, o escritos en un lenguaje como C o Java y almacenados externamente. Los métodos implementan operaciones que las aplicaciones pueden realizar en la entidad del mundo real.

Tipos de datos REF

Un tipo de datos REF es un identificador único de un objeto que permite referenciarlo desde otro objeto o incluso desde tablas de la base de datos. Es decir, un tipo de datos REF es un puntero a otro objeto.

Varrays

Un array es un conjunto ordenador de elementos donde todos los elementos son del mismo tipo.

Cada elemento tiene un índice que es un número que identifica la posición del elemento dentro del array.

El número de elementos en un array constituye el propio tamaño del array; por tanto, son de longitud variable, y por ello se les identifica como VARRAY (Variable Array).

Cuando se declara un VARRAY hay que especificar el tamaño máximo de elementos que puede contener. La declaración del VARRAY no supone que se asigne y reserve el espacio.

Un tipo VARRAY se puede utilizar de la siguiente forma:

- Como un tipo de dato de una columna de una tabla de la base de datos.
- Como un atributo de un tipo objeto.
- Como una variable, parámetro o valor de retorno de una función de PL/SQL.

Tablas anidadas

Un tipo de tabla anidada modela un conjunto de elementos desordenados.

Los elementos pueden ser de los tipos de datos vistos anteriormente (incluidos los tipos de datos de usuario).

Los tipos tabla anidada se pueden visualizar como una tabla de una única columna, o si la tabla anidada es un tipo objeto, como una tabla multicolumnas con una columna por cada atributo del tipo objeto.

La declaración de un tipo tabla anidada no supone que se asigne y reserve el espacio.

Un tipo tabla anidada se puede utilizar de la siguiente forma:

- Como un tipo de dato de una columna de una tabla de la base de datos.
- Como un atributo de un tipo objeto.
- Como una variable, parámetro o valor de retorno de una función de PL/SQL.

TIPOS SUMINISTRADOS POR ORACLE

Aparte de los tipos de datos vistos hasta el momento, que derivan directamente del estándar de SQL, Oracle suministra una serie de tipos de datos propios para dar cobertura a las nuevas necesidades en la información almacenada en base de datos, así como nuevos tipos aparecidos en el último estándar de SQL publicado.

Los tipos de datos suministrados por Oracle son:

- Tipos ANY.
- Tipos XML.
- Tipos espaciales.
- Tipos multimedia.

Tipos ANY

Los tipos ANY proporcionan un modelado altamente flexible de parámetros (correspondientes a procedimientos) y columnas (de tablas), donde no se conoce el tipo actual.

Estos tipos de datos permiten dinámicamente, encapsular y acceder a la descripción de los tipos, instancias de datos y conjunto de instancias de datos de cualquier otro tipo de dato SQL.

Los tipos ANY tienen interfaces de construcción y acceso a la información mediante rutinas OCI y PL/SQL.

Tipos XML

Los tipos XML se utilizan para almacenar una consulta de datos XML sobre la base de datos.

Los tipos XML tienen funciones que permiten acceder, extraer y consultar los datos XML usando expresiones XPATH.

Tipos espaciales

Los tipos espaciales están diseñados para que sea más sencillo el manejo de los datos de carácter geográfico, y que resulte más intuitivo para los usuarios el manejo de aplicaciones GIS (Sistemas de Información Geográfica).

Después de que los datos espaciales se hayan almacenado en una base de datos Oracle, se pueden fácilmente manipular, recuperar y relacionar con el resto de datos almacenados en la base de datos.

Tipos multimedia

Este tipo de datos son utilizados por el complemento Oracle Multimedia instalado en el SGBD, y son similares a las claves Java y C++ que describen datos multimedia.

Una instancia de este tipo de objetos tiene atributos (incluyendo metadatos y datos multimedia) y métodos.

Este tipo de datos debe ser creado en el esquema ORDSYS de la base de datos y existen sinónimos públicos para todos los tipos de datos, de forma que puedan acceder al mismo dentro de su esquema particular.

GESTIÓN DE USUARIOS 5

INTRODUCCIÓN

Todos los objetos que se crean en la base de datos se almacenan en espacios físicos denominados DATAFILES o archivos de datos.

Los DATAFILES son aquellos ficheros que están asociados a un TABLESPACE, o unidad lógica de almacenamiento de información de una base de datos Oracle.

Cada objeto de dicha base de datos se asocia a un usuario, que es aquel que lo crea. Este usuario se hace poseedor de los objetos, y además puede administrarlos hacia otros usuarios de la base de datos a través de permisos.

En toda instalación de una base de datos Oracle se instalan al menos 2 usuarios: SYS y SYSTEM, con sus correspondientes password por defecto.

En la tabla que se muestra a continuación se pueden consultar las password por defecto que se asignan en las instalaciones de bases de datos Oracle, que, por supuesto, pueden ser cambiados durante el transcurso de la misma.

Nombre de usuario	Password	Derechos sobre objetos de la base de datos
SYS	CHANGE_ON_INSTALL	DBA (Derechos de administración sobre todos los objetos de la base de datos). Es el poseedor de todas las vistas y tablas del sistema. Es el usuario con más permisos y privilegios del SGBD.
SYSTEM	MANAGER	DBA

MODOS DE CONEXIÓN A LA BASE DE DATOS

Para conectarse una base de datos Oracle, se utilizan 3 modalidades (perfiles) de conexión:

- NORMAL: es el valor por defecto y no hace falta indicarlo en la conexión.
- SYSDBA: es el modo de administración con más permisos.
- SYSOPER: es un modo de administración restringido.

Privilegios asociados al modo de conexión SYSDBA

El perfil de conexión SYSDBA tiene asociados los siguientes privilegios para el manejo de la base de datos:

- STARTUP: arranque de la base de datos.
- SHUTDOWN: parada de la base de datos.
- ALTER DATABASE OPEN/MOUNT/BACKUP/CHARACTER SET: alteración de la base de datos para operaciones de apertura, montado, salvaguarda y cambio del juego de caracteres.
- CREATE DATABASE: creación de una base de datos.
- CREATE SPFILE: creación de un fichero de parámetros de arranque.
- ARCHIVELOG y RECOVERY: autoarchivado de operaciones y elementos de base de datos y recuperación de los mismos en caso de error.
- RESTRICTED SESSION: permite restringir los accesos a la base de datos.

Privilegios asociados al modo de conexión SYSOPER

El perfil de conexión SYSOPER tiene asociados los siguientes privilegios para el manejo de la base de datos:

- STARTUP: arranque de la base de datos.
- SHUTDOWN: parada de la base de datos.
- ALTER DATABASE OPEN/MOUNT/BACKUP: alteración de la base de datos para operaciones de apertura, montado y salvaguarda.
- CREATE SPFILE: creación de un fichero de parámetros de arranque.
- ARCHIVELOG y RECOVERY: autoarchivado de operaciones y elementos de base de datos y recuperación de los mismos en caso de error.
- RESTRICTED SESSION: permite restringir los accesos a la base de datos.

ARRANQUE Y PARADA DE UNA B.D. POR COMANDOS

Los métodos de arranque y parada de una base de datos Oracle han ido variando a medida que han ido apareciendo nuevas versiones.

En este apartado vamos a revisar de manera histórica estos métodos, desde la versión 7 de Oracle hasta la actual 12c.

Versión 7

El arranque y parada se hace con la herramienta denominada SQL*DBA.

Para invocar la ejecución de esta herramienta en línea de comandos del sistema operativo Windows, hay que buscar el fichero denominado `SQLDBA.EXE` y ejecutarlo.

Versión 8

El arranque y parada se hace con la herramienta denominada SQL*DBA.

Para invocar la ejecución de esta herramienta en línea de comandos del sistema operativo Windows, hay que buscar el fichero denominado `SVRMGR30.EXE` y ejecutarlo.

Versión 8i

El arranque y parada se hace con la herramienta denominada SQL*DBA.

Para invocar la ejecución de esta herramienta en línea de comandos del sistema operativo Windows, hay que buscar el fichero denominado SVRMGRL.EXE y ejecutarlo.

USUARIO DE CONEXIÓN PARA ARRANQUE Y PARADA EN VERSIONES 7, 8 Y 8I

Una vez ejecutada la herramienta especificada anteriormente para cada una de las versiones de Oracle, el SGBD nos devuelve como prompt SQLDBA>, lo cual indica que nos hemos conectado correctamente al SGBD de Oracle.

Parra arrancar o parar la base de datos hay que conectarse como usuario INTERNAL utilizando el comando CONNECT INTERNAL.

Si este usuario solicitara password, habría que indicarle ORACLE, si no se especificó otra diferente durante la instalación del producto.

Versiones 9i, 10g, 11g y 12c

En las versiones 9i, 10g, 11g y 12c el programa SQL*DBA ha desaparecido, al igual que el propio usuario INTERNAL.

En estas versiones, toda la gestión por comandos se realiza con el intérprete de comandos genérico de Oracle: SQL*PLUS.

Asimismo, se puede realizar la gestión integrada de la base de datos a través de herramientas visuales, que se arrancan desde el explorador de Windows.

Para invocar la ejecución de esta herramienta en línea de comandos del sistema operativo Windows, hay que buscar el fichero denominado SVRMGRL.EXE y ejecutarlo.

METODO DE CONEXIÓN PARA ARRANQUE Y PARADA EN VERSIONES 9I, 10G, 11G Y 12C

Para efectuar operaciones de arranque y parada de la base de datos en las versiones 9i, 10g, 11g y 12c, hay que conectarse al SGBD mediante el intérprete de comandos SQL*PLUS sin conectarse a la base de datos. Esto se realiza mediante el siguiente comando:

```
SQLPLUS /NOLOG
```

A continuación, el SGBD nos devolverá el prompt SQL> y es en ese momento cuando nos conectaremos con el usuario SYS y con el modo de conexión SYSDBA para realizar la operación de arranque o parada (según corresponda). El comando para efectuar la conexión es el siguiente:

```
SQL> CONNECT SYS/password AS SYSDBA
```

Donde sustituiremos **password** por la correspondiente asignada para el usuario SYS. Hay que recordar que si no se especificó una diferente durante la instalación del producto o posteriormente a la misma, la password por defecto de SYS es CHANGE_ON_INSTALL.

Comando para el arranque de la B.D. en todas las versiones

En todas las versiones de Oracle vistas hasta el momento se ha mantenido el mismo comando para el arranque de la base de datos.

Una vez conectado al SGBD hay que ejecutar el comando STARTUP.

Comando para la parada de la B.D. en todas las versiones

En todas las versiones de Oracle vistas hasta el momento se ha mantenido el mismo comando para la parada de la base de datos.

Una vez conectado al SGBD hay que ejecutar el comando SHUTDOWN.

El comando de parada SHUTDOWN admite 3 parámetros:

- `SHUTDOWN NORMAL` o `SHUTDOWN:` es la opción por defecto. Ejecuta la parada de la base de datos cuando los usuarios conectados a la misma cierran todas las sesiones de conexión.

- `SHUTDOWN IMMEDIATE:` ejecuta la parada de la base de datos cerrando las conexiones (sesiones) abiertas por los usuarios, y realizando un ROLLBACK de las transacciones en curso.

- `SHUTDOWN ABORT:` ejecuta la parada de la base de datos cerrando las conexiones (sesiones) abiertas por los usuarios, sin realizar ROLLBACK de las transacciones en curso. Este es el método de parada más agresivo, que solo debe ejecutarse ante riesgos graves contra la base de datos que obliguen a una parada inmediata de la misma.

Comando para salir de SQL*DBA o SQL*PLUS

Una vez concluidas las operaciones de arranque o parada de la base de datos, para salir de los programas SQL*DBA o SQL*PLUS, dependiendo de la versión de Oracle que hayamos utilizado para las operaciones, indicaremos el comando `EXIT`.

CREACIÓN DE UN USUARIO

Desde el intérprete de comandos SQL*PLUS, conectado como usuario SYSTEM o con un usuario con privilegios DBA, hay que ejecutar la siguiente sentencia para invocar al SGBD para que cree un nuevo usuario:

```
CREATE USER nombre_usuario IDENTIFIED BY password;
```

Donde **nombre_usuario** corresponde con el nombre del nuevo usuario que queremos crear en la base de datos, y **password** con la correspondiente contraseña asociada al mismo.

GESTIÓN DE ROLES

Los ROLES (papeles) son las características de utilización de la base de datos que tiene cada usuario. Equivale al perfil de trabajo contra la base de datos, es decir, qué operaciones se les permite realizar a cada usuario dentro de la base de datos.

Como mínimo, un usuario tendrá que tener el ROLE de conexión para poder conectarse a la base de datos, dado que con crear un usuario no se hace nada más que almacenar un nuevo objeto en la base de datos de tipo usuario, sin ningún privilegio o ROLE asignado.

Para asignar el ROLE de conexión, así como cualquier otro ROLE a un usuario, se utiliza la siguiente sentencia:

```
GRANT nombre_role TO nombre_usuario [IDENTIFIED BY password
[WITH ADMIN OPTION]];
```

Donde **nombre_role** corresponde con el nombre de uno de los ROLES preestablecido por el SGBD de Oracle, **nombre_usuario** será el usuario destino de la asignación del ROLE, y **password** será la contraseña asociada al mismo.

La introducción de la **password** del usuario es opcional.

La opción `WITH ADMIN OPTION` del comando también es opcional, y permite al usuario destino de la asignación del ROLE, poder igualmente asociarle los ROLES recibidos a otros usuarios de la base de datos.

Roles preestablecidos

A continuación, se muestra una tabla con los distintos ROLES preestablecidos por la versión 12c de Oracle que se pueden asociar a un usuario.

Nombre del ROLE	Descripción
ADM_PARALLEL_EXECUTE_TASK	Proporciona privilegios para actualizar en paralelo los datos de las tablas usando el paquete PL/SQL DBMS_PARALLEL_EXECUTE.

Nombre del ROLE	Descripción
APEX_ADMINISTRATOR_ROLE	Proporciona privilegios para actualizar los datos de la tabla en paralelo usando el paquete pl/sql DBMS_PARALLEL_EXECUTE
APEX_GRANTS_FOR_NEW_USERS_ROLE	Contiene múltiples privilegios del SYS
AQ_ADMINISTRATOR_ROLE	Proporciona privilegios para administrar la Cola Avanzada (Advanced Queuing). Incluye los siguientes privilegios sobre las tablas de la Cola Avanzada: ENQUEUE ANY QUEUE DEQUEUE ANY QUEUE MANAGE ANY QUEUE SELECT Además, incluye el privilegio EXECUTE sobre los paquetes de la Cola Avanzada.
AQ_USER_ROLE	Es un ROLE obsoleto que se mantiene por compatibilidad con la versión 8 de Oracle. Proporciona el privilegio EXECUTE en los paquetes DBMS_AQ y DBMS_AQIN.
AUDIT_ADMIN	Proporciona privilegios para crear políticas de auditoría. Usa las sentencias AUDIT y NOAUDIT.
AUDIT_VIEWER	Proporciona privilegios para ver y analizar las auditorías de datos.
AUTHENTICATEDUSER	Usado por los protocolos XDB para definir a cualquier usuario que se haya conectado al sistema.
CAPTURE_ADMIN	Proporciona privilegios necesarios para crear y gestionar privilegios de análisis de políticas.
CDB_DBA	Proporciona los privilegios requeridos para administrar un CBD, tal como un: SET CONTAINER SELECT ON PDB_PLUG_IN_VIOLATIONS SELECT ON CDB_LOCAL_ADMIN_PRIVS
CONNECT	Proporciona el privilegio del sistema CREATE SESSION. Este ROLE se mantiene por compatibilidad con versiones anteriores de Oracle.

Nombre del ROLE	Descripción
CSW_USR_ROLE	Proporciona privilegios de usuario para gestionar el Catálogo de Servicios para la Web (CSW – Catalog Services for the Web), que es un componente de Oracle Spatial.
CTXAPP	Proporciona privilegios para crear índices e índices preferentes de Oracle Text, y usar paquetes de PL/SQL. Este ROLE debería ser asignado por usuarios de Oracle Text.
DATAPUMP_EXP_FULL_DATABASE	Proporciona privilegios para exportar datos de una base de datos Oracle, usando Oracle Data Pump.
DATAPUMP_IMP_FULL_DATABASE	Proporciona privilegios para importar datos a una base de datos Oracle, usando Oracle Data Pump.
DBA	Proporciona todos los privilegios del sistema que fueron creados con la opción ADMIN. Este ROLE se mantiene por compatibilidad con versiones anteriores de Oracle.
DBFS_ROLE	Proporciona acceso a los objetos y paquetes del Filesystem de la Base de Datos.
DBHADOOP	Creado en la instalación pero no tiene asociados roles o privilegios.
DELETE_CATALOG_ROLE	Proporciona el privilegio DELETE en la tabla de auditoría del sistema (AUD$).
DV_ACCTMGR	Utilice este role para crear y mantener cuentas y perfiles de base de datos.
DV_ADMIN	Controla los ODV (Oracle Database Vault) de los paquetes pl/sql.
DV_AUDIT_CLEANUP	Proporciona permisos para limpiar los ODV en entornos de auditoría.
DV_GOLDENGATE_ADMIN	Para proporcionar permisos a los usuarios que deseen configurar Oracle GoldenGate.

Nombre del ROLE	Descripción
DV_GOLDENGATE_REDO_ACCESS	Para proporcionar permisos a los usuarios que estén usando Oracle GoldenGate con los métodos siguientes: TRANLOGOPTIONS DBLOGREADER Para acceder a los redo logs en entornos de uso de ODV.
DV_MONITOR	Habilita al agente de Oracle Enterprise Manager Grid Control a monitorizar ODV.
DV_OWNER	Contiene los privilegios de los roles: DV_ADMIN DV_SECANALYST
DV_PATCH_ADMIN	Proporciona permisos para administrar la base de datos con objeto de añadir parches de software a la misma o añadir lenguaje al ODV.
DV_PUBLIC	Para proporcionar privilegios en los entornos ODV al esquema DVSYS de forma pública, primero hay que asociar el role DV_PUBLIC a PUBLIC.
DV_REALM_OWNER	Permite gestionar objetos de base de datos en múltiples esquemas que definen un REALM (ámbito, esfera o reino).
DV_REALM_RESOURCE	Concede privilegios para operaciones tales como la creación de tablas, vistas, triggers, sinónimos y otros objetos que se usan habitualmente en un REALM.
DV_SECANALYST	Permite consultar objetos del esquema DVSYS a través de vistas de ODV.
DV_STREAMS_ADMIN	Concede permisos a un usuario para la configuración de la replicación de Streams en ODV.
DV_XSTREAM_ADMIN	Concede permisos a un usuario para la configuración de la replicación de XStreams en ODV.

Nombre del ROLE	Descripción
EJBCLIENT	Proporciona privilegios para conectarse a EJBs desde procedimientos almacenados en Java.
EM_EXPRESS_ALL	Proporciona privilegios para conectarse a Oracle Enterprise Manager Express y usar todo la funcionalidad de la herramienta.
EM_EXPRESS_BASIC	Proporciona privilegios para conectarse a Oracle Enterprise Manager Express y ver las páginas en modo solo lectura.
EXECUTE_CATALOG_ROLE	Proporciona el privilegio EXECUTE sobre objetos del diccionario de datos.
EXP_FULL_DATABASE	Proporciona los privilegios necesarios para realizar exportaciones completas e incrementales de la base de datos, usando la utilidad de exportación (posteriormente sustituida por Oracle Data Pump). Incluye los siguientes privilegios: SELECT ANY TABLES, BACKUP ANY TABLE, EXECUTE ANY PROCEDURE, EXECUTE ANY TYPE, ADMINISTER RESOURCE MANAGER e INSERT, DELETE y UPDATE en las tablas SYS.INCVID, SYS.INCFIL y SYS.INCEXP. También incluye los siguientes ROLES: EXECUTE_CATALOG_ROLE y SELECT_CATALOG_ROLE.
GATHER_SYSTEM_STATISTICS	Proporciona privilegios para actualizar las estadísticas del sistema que son recuperadas usando el procedimiento DBMS_STATS.GATHER_SYSTEM_STATISTICS.
GDS_CATALOG_SELECT	Proporciona acceso a 10 objetos propiedad de GSMADMIN_INTERNAL
GLOBAL_AQ_USER_ROLE	Proporciona privilegios para establecer una conexión al servidor LDAP, para usarlo con Oracle Streams AQ.

Nombre del ROLE	Descripción
GSMADMIN_ROLE	Proporciona los siguientes roles: AQ_ADMINISTRATOR_ROLE CONNECT Incluido el privilegio EXECUTE en DBMS_GSM_UTILITY y recursos relacionados.
GSMUSER_ROLE	Proporciona el role CONNECT. Incluido el privilegio EXECUTE en DBMS_GSM_DBADMIN.
GSM_POOLADMIN_ROLE	Proporciona el role CONNECT. Incluido el privilegio EXECUTE en DBMS_GSM_POOLADMIN.
HS_ADMIN_EXECUTE_ROLE	Proporciona el privilegio EXECUTE para usuarios que quieren usar los paquetes PL/SQL de HS (Heterogeneous Services).
HS_ADMIN_ROLE	Proporciona privilegios para usar tanto los paquetes de HS, como consultas a las vistas del diccionario de datos de HS.
HS_ADMIN_SELECT_ROLE	Proporciona privilegios para consultar las vistas del diccionario de datos de HS.
IMP_FULL_DATABASE	Proporciona los privilegios necesarios para importar una base de datos completa usando la utilidad de importación (posteriormente sustituida por Oracle Data Pump). Incluye una lista extensa de privilegios del sistema y los siguientes ROLES: EXECUTE_CATALOG_ROLE y SELECT_CATALOG_ROLE.
JAVADEBUGPRIV	Proporciona privilegios para ejecutar el debugger de las aplicaciones Java de la base de datos Oracle.
JAVASYSPRIV	Proporciona permisos amplios para usar Java2, incluyendo la actualización de paquete Oracle JVM protegidos.
JAVAUSERPRIV	Proporciona permisos limitados para usar Java2.

Nombre del ROLE	Descripción
JAVA_ADMIN	Proporciona permisos de administración para actualizar las tablas de políticas para las aplicaciones Java de la base de datos Oracle.
JAVA_DEPLOY	Proporciona privilegios para desarrollar DLLs ncomp en el directorio java/admin, usando las utilidades ncomp y deplyns. Sin este ROLE, no es posible acceder a los directorios java/admin y java/deploy.
JMXSERVER	Proporciona privilegios para arrancar y mantener un agente JMX en una sesión de base de datos.
LBAC_DBA	Proporciona permisos para usar los paquetes PL/SQL de SA_SYSDBA.
LOGSTDBY_ADMINISTRATOR	Proporciona privilegios de administración para manejar el entorno SQL Apply.
OEM_ADVISOR	Proporciona los privilegios para usar los paquetes PL/SQL DBMS_SQLTUNE y ADVISOR.
OEM_MONITOR	Proporciona los privilegios necesarios para monitorizar y gestionar la base de datos a través del componente Management Agent de Oracle Enterprise Manager.
OLAP_DBA	Proporciona privilegios de administración para crear objetos dimensionados en diferentes esquemas para Oracle OLAP.
OLAP_USER	Proporciona privilegios para desarrollo de aplicaciones que crean objetos dimensionados en los esquemas propios de dichas aplicaciones para Oracle OLAP.
OLAP_XS_ADMIN	Proporciona privilegios para administrar la seguridad de Oracle OLAP.

Nombre del ROLE	Descripción
OPTIMIZER_PROCESSING_RATE	Proporciona privilegios para ejecutar los siguientes procedimientos del paquete DBMS_STATS: GATHER_PROCESSING_RATE SET_PROCESSING_RATE DELETE_PROCESSING_RATE
ORDADMIN	Proporciona privilegios para administrar Oracle Multimedia DICOM.
OWB$CLIENT	Proporciona privilegios para realizar tareas relacionadas con el cliente, para Oracle Warehouse Builder, tales como crear proyectos, módulos, tablas, vistas, mapas, etc.
OWB_DESIGNCENTER_VIEW	Proporciona privilegios a nivel de base de datos, para cualquier usuario registrado de Oracle Warehouse Builder que quiera consultar las vistas públicas de Warehouse Builder, tales como ALL_IV_PROJECTS.
OWB_USER	Proporciona privilegios para crear y hacerse propietario de un espacio de trabajo (workspace) en Oracle Warehouse Builder.
PDB_DBA	Autoriza automáticamente al usuario creado con la instalación de una nueva base de datos conectable (Pluggable DataBase - PDB) desde la semilla (seed) PDB. No se proporcionan privilegios con este role.
PLUSTRACE	Asocia los privilegios requeridos en la vista V$ para usar AUTOTRACE.
PROVISIONER	Proporciona privilegios para registrar y actualizar las retrollamadas globales de las sesiones de Oracle Database Real Application.
PUBLIC	-

Nombre del ROLE	Descripción
RECOVERY_CATALOG_OWNER	Proporciona privilegios para el propietario del catálogo de recuperación (recovery). Incluye: CREATE SESSION, ALTER SESSION, CREATE SYNONYM, CREATE VIEW, CREATE DATABASE LINK, CREATE TABLE, CREATE CLUSTER, CREATE SEQUENCE, CREATE TRIGGER y CREATE PROCEDURE.
RESOURCE	Proporciona los siguientes privilegios del sistema: CREATE CLUSTER, CREATE INDEXTYPE, CREATE OPERATOR, CREATE PROCEDURE, CREATE SEQUENCE, CREATE TABLE, CREATE TRIGGER, CREATE TYPE. Este ROLE se mantiene por compatibilidad con versiones anteriores de Oracle.
SCHEDULER_ADMIN	Permite la asignación de permisos para la ejecución de procedimientos del paquete DBMS_SHEDULER. Incluye todos los privilegios del sistema de la cola de trabajo (job scheduler) y está incluido en el ROLE DBA.
SELECT_CATALOG_ROLE	Proporciona el privilegio SELECT a objetos del diccionario de datos.
SPATIAL_CSW_ADMIN	Proporciona privilegios administrativos para gestionar CSW.
SPATIAL_WFS_ADMIN	Proporciona privilegios administrativos para gestionar WFS (Web Feature Service), que es un componente de Oracle Spatial.
TKPROFER	Proporciona el permiso SELECT en las vistas dinámicas para TKPROF.
WFS_USR_ROLE	Proporciona privilegios de usuario para WFS.
WM_ADMIN_ROLE	Proporciona privilegios para gestionar Oracle Workspace. Permite que los usuarios puedan ejecutar cualquier procedimiento DBMS_WM.

Nombre del ROLE	Descripción
XDBADMIN	Permite tanto la asignación de permisos para registrar globalmente un esquema XML, como la denegación de permisos, de forma que sea registrado para uso o acceso solo por parte del usuario propietario.
XDB_SET_INVOKER	Permite la asignación de permisos para el manejo de los triggers del repositorio XML que son invocados.
XDB_WEBSERVICES	Permite la asignación de permisos para acceder a los servicios de Oracle Database Web sobre HTTPS.
XDB_WEBSERVICES_OVER_HTTP	Permite la asignación de permisos para acceder a los servicios de Oracle Database Web sobre HTTPS.
XDB_WEBSERVICES_WITH_PUBLIC	Permite la asignación de permisos para acceder a los objetos públicos a través de los servicios de Oracle Database Web.
XS_CACHE_ADMIN	Se requiere para cachear la política de seguridad en el nivel mid-tier para autorizar sobre el método de la clase XSAccessController. Se debe asociar este role al usuario conectado a la aplicación o al "dispatcher" del Real Application Security.
XS_NAMESPACE_ADMIN	En Oracle Database Real Application Security, habilita los permisos para gestionar y manipular el espacio de nombres y atributos de una sesión.
XS_RESOURCE	En Oracle Database Real Application Security, habilita los permisos para gestionar objetos en el esquema asociado, a través del paquete pl/sql XS_ACL.
XS_SESSION_ADMIN	En Oracle Database Real Application Security, habilita los permisos para gestionar el ciclo de vida de una sesión incluida la posibilidad de crear, asociar, desasociar y destruir la sesión.

GESTIÓN DE PRIVILEGIOS SOBRE OBJETOS

Mientras que los ROLES otorgaban permisos a los usuarios para realizar operaciones sobre objetos de su propiedad, la gestión de privilegios resuelve un aspecto pendiente, que es la necesidad de que un usuario pueda manipular o ejecutar objetos de otros usuarios.

Para realizar estas operaciones, existen privilegios que se otorgan también con la instrucción GRANT, de la siguiente manera:

```
GRANT privilegio ON objeto TO usuario [WITH GRANT OPTION];
```

Donde **privilegio** corresponde con el nombre del privilegio que se quiera otorgar, **objeto** es el nombre del objeto de la base de datos sobre el que se otorga el privilegio y **usuario** será el usuario destino de la asignación del privilegio.

La opción `WITH ADMIN OPTION` del comando es opcional y permite al usuario destino de la asignación del privilegio, poder igualmente asociárselo a otros usuarios de la base de datos.

Los privilegios siempre los tiene que otorgar el usuario poseedor del objeto, o bien el administrador del sistema (o cualquier usuario con el role DBA).

Si queremos asignar de una tacada todos los privilegios posibles de un objeto a un usuario, podemos utilizar en vez de uno de los privilegios concretos, la cláusula `ALL PRIVILEGES` que asigna todos los privilegios que se puedan dar sobre el objeto. Para hacer esto, utilizaremos la siguiente sintaxis:

```
GRANT ALL [PRIVILEGES] ON objeto TO usuario [WITH GRANT OPTION];
```

Por último, y aunque no es muy común, se pueden asignar privilegios a columnas de un objeto de tipo tabla en vez de a todo el objeto entero. Un ejemplo de este tipo de asignación es el siguiente:

```
GRANT UPDATE(empno, salario), REFERENCES(empno) ON emp TO DEMO;
```

En el ejemplo anterior se asignan privilegios de actualización al usuario `DEMO`, únicamente sobre las columnas `empno` y `salario` de la tabla `emp`, y además permite crear una restricción (constraint) de tipo `FOREIGN KEY` sobre la columna `empno` de esa misma tabla.

Privilegios sobre objetos

A continuación, se muestra una tabla con los distintos privilegios que se pueden otorgar sobre objetos de la base de datos, agrupándolos en las siguientes categorías:

- Privilegios sobre objetos DIRECTORY
- Privilegios sobre objetos EDITION
- Privilegios sobre objetos INDEXTYPE
- Privilegios sobre objetos FLASHBACK DATA ARCHIVE
- Privilegios sobre objetos LIBRARY
- Privilegios sobre objetos MATERIALIZED VIEW
- Privilegios sobre objetos MINING MODEL
- Privilegios sobre objetos OBJECT TYPE
- Privilegios sobre objetos OLAP
- Privilegios sobre objetos OPERATOR
- Privilegios sobre objetos PROCEDURE, FUNCTION o PACKAGE
- Privilegios sobre objetos SCHEDULER
- Privilegios sobre objetos SEQUENCE
- Privilegios sobre objetos SYNONYM
- Privilegios sobre objetos TABLE
- Privilegios sobre objetos VIEW

PRIVILEGIOS SOBRE OBJETOS DIRECTORY

Los siguientes privilegios sobre objetos DIRECTORY proporcionan seguridad para acceder a los ficheros almacenados en un directorio del sistema operativo, al cual apunta el objeto DIRECTORY almacenado en la base de datos.

El objeto DIRECTORY contiene el nombre de la ruta completa del directorio del sistema operativo, donde residen los ficheros.

Nombre del privilegio	Descripción
READ	Permite leer ficheros en el directorio.
WRITE	Permite escribir ficheros en el directorio.
EXECUTE	Permite ejecutar un programa preprocesado que resida en el directorio.

PRIVILEGIOS SOBRE OBJETOS EDITION

Los siguientes privilegios sobre objetos EDITION autorizan el uso de un editor.

Nombre del privilegio	Descripción
USE	Permite usar un editor.

PRIVILEGIOS SOBRE OBJETOS INDEXTYPE

Los siguientes privilegios sobre objetos INDEXTYPE autorizan operaciones en un INDEXTYPE.

Nombre del privilegio	Descripción
EXECUTE	Permite referenciar un INDEXTYPE.

PRIVILEGIOS SOBRE OBJETOS FLASHBACK DATA ARCHIVE

Los siguientes privilegios sobre objetos FLASHBACK DATA ARCHIVE autorizan operaciones en archivos de datos FLASHBACK.

Nombre del privilegio	Descripción
FLASHBAK ARCHIVE	Permite habilitar o deshabilitar el histórico de trazas para una tabla.

PRIVILEGIOS SOBRE OBJETOS LIBRARY

Los siguientes privilegios sobre objetos LIBRARY autorizan operaciones en una librería.

Nombre del privilegio	Descripción
EXECUTE	Permite usar y referenciar objetos específicos e invocar sus métodos.

PRIVILEGIOS SOBRE OBJETOS MATERIALIZED VIEW

Los siguientes privilegios sobre objetos MATERIALIZED VIEW autorizan operaciones en una vista materializada.

Los privilegios DELETE, INSERT y UPDATE solo se pueden asignar sobre vistas materializadas actualizables.

Nombre del privilegio	Descripción
ON COMMIT REFRESH	Permite crear una vista materializada con autorrefresco de datos al realizar un commit sobre los datos a los que accede la vista.
QUERY REWRITE	Permite crear una vista materializada para consulta de reescritura usando una tabla específica.
SELECT	Permite consultar una vista materializada con la sentencia SELECT.

PRIVILEGIOS SOBRE OBJETOS MINING MODEL

Los siguientes privilegios sobre objetos MINING MODEL autorizan operaciones en un modelo de minería. Para modelos dentro del esquema del propio usuario no es necesario asignar estos privilegios.

Nombre del privilegio	Descripción
ALTER	Permite cambiar el nombre del modelo de minería o la matriz de coste asociada usando los procedimientos aplicados de DBMS_DATA_MINING.
SELECT	Permite anotar o ver el modelo de minería. La anotación se realiza con la familia de funciones SQL denominada PREDICTION o con el procedimiento DBMS_DATA_MINING.APPLY. La visualización del modelo se realiza con el procedimiento DBMS_DATA_MINING.GET_MODEL_DETAILS_*.

PRIVILEGIOS SOBRE OBJETOS OBJECT TYPE

Los siguientes privilegios sobre objetos OBJECT TYPE autorizan operaciones en tipos objeto de la base de datos.

Nombre del privilegio	Descripción
DEBUG	Permite acceder a través del debugger a todas las variables públicas y no públicas, métodos y tipos definidos en el tipo objeto.
EXECUTE	Permite usar y referenciar los objetos específicos e invocar sus métodos.

UNDER	Permite crear un subtipo a partir del tipo objeto concreto. Solo se pueden dar permisos a este objeto (subtipo) si se dispone del privilegio UNDER ANY TYPE con la opción WITH GRANT OPTION asociada al supertipo inmediatamente superior a este subtipo.

PRIVILEGIOS SOBRE OBJETOS OLAP

Los siguientes privilegios sobre objetos OLAP son válidos si se está usando una base de datos Oracle con el complemento OLAP instalado en la misma.

Nombre del privilegio	Descripción
INSERT	Permite insertar miembros en la dimensión del cubo OLAP, o medidas en la carpeta de medidas.
ALTER	Permite cambiar la definición de las dimensiones del cubo OLAP o del propio cubo.
DELETE	Permite borrar miembros de la dimensión del cubo OLAP, o medidas desde la carpeta de medidas.
SELECT	Permite visualizar o consultar el cubo OLAP o la dimensión del cubo.
UPDATE	Permite actualizar los valores de medida del cubo OLAP o valores de atributos de la dimensión del cubo.

PRIVILEGIOS SOBRE OBJETOS OPERATOR

Los siguientes privilegios sobre objetos OPERATOR autorizan operaciones en operadores definidos por el usuario.

Nombre del privilegio	Descripción
EXECUTE	Permite referenciar un operador.

PRIVILEGIOS SOBRE OBJETOS PROCEDURE, FUNCTION O PACKAGE

Los siguientes privilegios sobre objetos PROCEDURE, FUNCTION o PACKAGE autorizan operaciones en procedimientos, funciones y paquetes.

Estos privilegios también son aplicables sobre código fuente, clases o recursos de Java que trata la base de datos Oracle.

Nombre del privilegio	Descripción
DEBUG	Permite acceder a través del debugger a todas las variables públicas y no públicas, métodos y tipos definidos en los objetos.
EXECUTE	Permite ejecutar los procedimientos y funciones directamente, o acceder a cualquier objeto de programa que se haya declarado en la especificación de un paquete.

PRIVILEGIOS SOBRE OBJETOS SCHEDULER

Los objetos de trabajo del SCHEDULER se crean usando el paquete DBMS_SCHEDULER.

Después de haber creado estos objetos, se pueden asignar los siguientes privilegios.

Nombre del privilegio	Descripción
EXECUTE	Permite operaciones en clases, programas, cadenas y credenciales de un trabajo.
ALTER	Permite modificaciones de trabajos, programas, cadenas, credenciales y planificadores (schedules).

PRIVILEGIOS SOBRE OBJETOS SEQUENCE

Los siguientes privilegios sobre objetos SEQUENCE autorizan operaciones en una secuencia.

Nombre del privilegio	Descripción
ALTER	Permite cambiar la definición de la secuencia con la sentencia ALTER SEQUENCE.
SELECT	Permite examinar e incrementar valores de la secuencia con las pseudocolumnas CURRVAL y NEXTVAL.

PRIVILEGIOS SOBRE OBJETOS SYNONYM

Los privilegios de un sinónimo (SYNONYM) son los mismos que tenga el objeto origen sobre el que se ha creado el sinónimo.

PRIVILEGIOS SOBRE OBJETOS TABLE

Los siguientes privilegios sobre objetos TABLE autorizan operaciones en una tabla.

Nombre del privilegio	Descripción
ALTER	Permite cambiar la definición de la tabla con la sentencia ALTER TABLE.
DELETE	Permite borrar filas de una tabla con la sentencia DELETE.
DEGUB	Permite acceder a través del debugger a: • Código PL/SQL en el cuerpo de un trigger definido en una tabla. • Información de la sentencia SQL que referencia directamente a la tabla.
INDEX	Permite crear un índice en la tabla con la sentencia CREATE INDEX.
INSERT	Permite añadir nuevas filas en la tabla con la sentencia INSERT.
REFERENCES	Permite crear una CONSTRAINT (restricción) que referencia a la tabla. No es posible dar este privilegio a un ROLE.
SELECT	Permite consultar la tabla con la sentencia SELECT.
UPDATE	Permite cambiar datos de la tabla con la sentencia UPDATE.

PRIVILEGIOS SOBRE OBJETOS VIEW

Los siguientes privilegios sobre objetos VIEW autorizan operaciones en una vista.

Nombre del privilegio	Descripción
DEGUB	Permite acceder a través del debugger a: • Código PL/SQL en el cuerpo de un trigger definido en una vista. • Información de la sentencia SQL que referencia directamente a la vista.
DELETE	Permite borrar filas de una VISTA con la sentencia DELETE.
INSERT	Permite añadir nuevas filas en la vista con la sentencia INSERT.

MERGE	Este privilegio de objeto tiene el mismo comportamiento que el privilegio del sistema MEGER ANY VIEW, excepto que el privilegio está limitado a la vista específica que se haya indicado en la cláusula ON.
REFERENCES	Permite definir una constraint de tipo FOREIGN KEY en la vista.
SELECT	Permite consultar la vista con la sentencia SELECT.
UPDATE	Permite cambiar datos de la tabla con la sentencia UPDATE.
UNDER	Permite crear una subvista sobre la vista. Solo se puede dar este privilegio si se dispone del privilegio UNDER ANY VIEW con la opción WITH GRANT OPTION asociada a la supervista inmediatamente superior a esta subvista.

ANULACIÓN DE PRIVILEGIOS Y ROLES

Para quitar privilegios o roles a un usuario, se utiliza la sentencia REVOKE, con dos sintaxis diferentes. A continuación, se presenta la primera:

```
REVOKE role FROM usuario;
```

Esta primera sintaxis permite quitar un **role** concreto a un **usuario**. Para quitar varios roles al mismo usuario, se separará el nombre de cada uno de los roles con una coma.

La segunda sintaxis es la siguiente:

```
REVOKE {privilegio | ALL [PRIVILEGES]} ON objeto FROM usuario
[CASCADE CONSTRAINT];
```

En esta segunda sintaxis se quita un **privilegio**, o varios si se separan por comas, o todos si se incluye la cláusula ALL, sobre un **objeto**, a un **usuario**. Si además se incluye la cláusula CASCADE CONSTRAINT, se eliminan las restricciones (constraint) del tipo FOREIGN KEY que haya creado dicho usuario desde otros objetos al que se le quitan los privilegios.

BORRAR UN USUARIO

Para poder borrar un usuario, hay que conectarse a la base de datos con un usuario administrador (role DBA) y ejecutar la siguiente sintaxis de comando:

```
DROP USER usuario [CASCADE];
```

Esta primera sintaxis permite quitar un **role** concreto a un **usuario**. Para quitar varios roles al mismo usuario, se separará el nombre de cada uno de los roles con una coma.

USUARIO PUBLIC

Los perfiles y roles se pueden asignar en vez de a un usuario concreto, al usuario PUBLIC, que es una especie de grupo que engloba al resto de usuario creados en el sistema.

En la práctica cuando se otorgan derechos al usuario PUBLIC se están otorgando los derechos a todos los usuarios del sistema.

Por ejemplo, la siguiente instrucción:

```
GRANT INSERT ON CLIENTES TO PUBLIC
```

En este ejemplo se está asignando el privilegio de inserción sobre la tabla clientes a todos los usuarios del sistema.

SUPUESTO PRÁCTICO 2: ***Resolución en el Anexo I.***

Crearse un usuario mediante código SQL que tenga las siguientes características particulares de cada alumno que lo crea:

- El nombre de usuario deberá componerse de la forma que se expresa debajo, sin dejar espacios en blanco y sin utilizar símbolos especiales, solo letras alfabéticas:

```
NOMBREDEPILA+Iniciales de los apellidos
```

- Un ejemplo válido de nombre de usuario sería: PEDROMR
- La password de cada usuario se deja a libre elección por el alumno, prestando especial precaución cuando la inserte para recordarla, dado que la necesitará a lo largo de este curso.
- Además de crear el usuario, se le deberá dotar de capacidad suficiente para poderse conectar a la base de datos, utilizando el nombre y password creados con anterioridad, más la posibilidad de crear y ejecutar procedimientos (objeto PROCEDURE), así como crear vistas (objeto VIEW). Para hacerlo se deberán asignar los privilegios necesarios de los que se han visto durante este capítulo.

Para un mejor aprovechamiento del curso con objeto de poder probar los resultados de los supuestos prácticos que se proponen en el libro, se recomienda la instalación de la versión Oracle 11g XE. Puede encontrar información sobre los enlaces para su descarga, así como un tutorial de la instalación en el Anexo IV de este libro.

EL SUBLENGUAJE DDL 6

INTRODUCCIÓN

El lenguaje SQL se compone de los siguientes elementos básicos:

- Comandos.
- Cláusulas.
- Operadores.
- Funciones de agregado.

Dentro del lenguaje SQL se definen dos sublenguajes:

- DDL (Data Definition Language).
- DML (Data Manipulation Language).

Los comandos utilizados en el sublenguaje DDL son principalmente los siguientes:

Comando	Descripción
CREATE	Se utiliza para crear objetos: tablas, vistas, índices, etc.
DROP	Se utiliza para eliminar objetos.
ALTER	Se utiliza para modificar la estructura de los objetos.
EXECUTE	Se utiliza para la ejecución de estructuras de código PL/SQL.
GRANT	Se utiliza para la asignación de privilegios y roles a un usuario.
REVOKE	Se utiliza para anular privilegios y roles de un usuario.

CREACIÓN DE UNA TABLA (CREATE TABLE)

Al crear una tabla con el comando CREATE TABLE, podemos indicar los siguientes objetos pertenecientes a la misma:

- Nombre de Columnas o campos (propiedades) de la tabla.
- Tipo y tamaño de cada columna.
- Clave primaria (PRIMARY KEY).
- Claves alternativas (UNIQUE).
- Claves ajenas (FOREIGN KEY).
- Restricciones de usuario (CONSTRAINTS).

Se permite la creación de 3 tipos de tablas:

- Tablas relacionales. Son las tablas habituales con filas y columnas.
- Tablas objeto.
- Tablas de tipo XML.

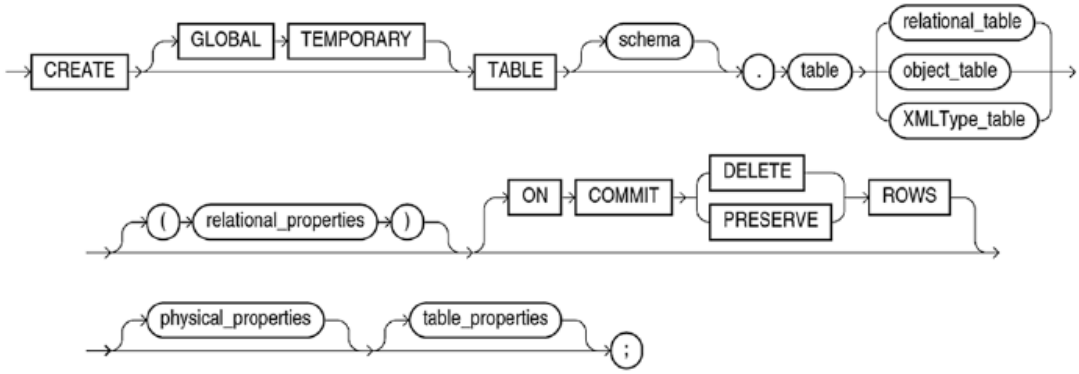


Fig. 6-1 Sintaxis para la creación de una tabla relacional.

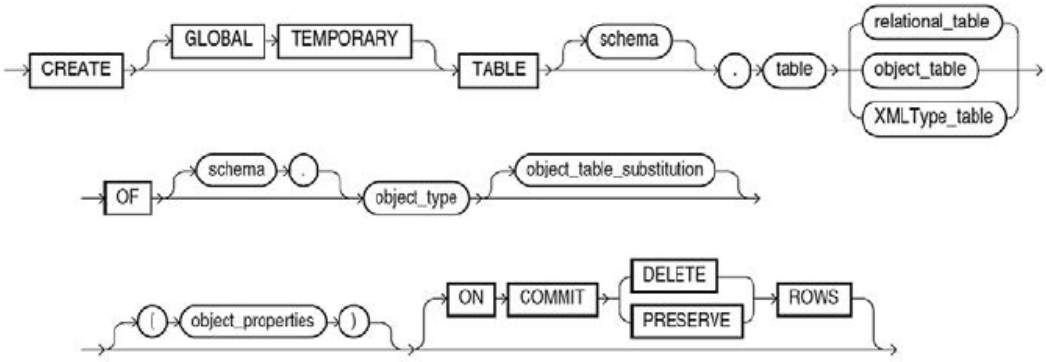


Fig. 6-2 Sintaxis para la creación de una tabla objeto.

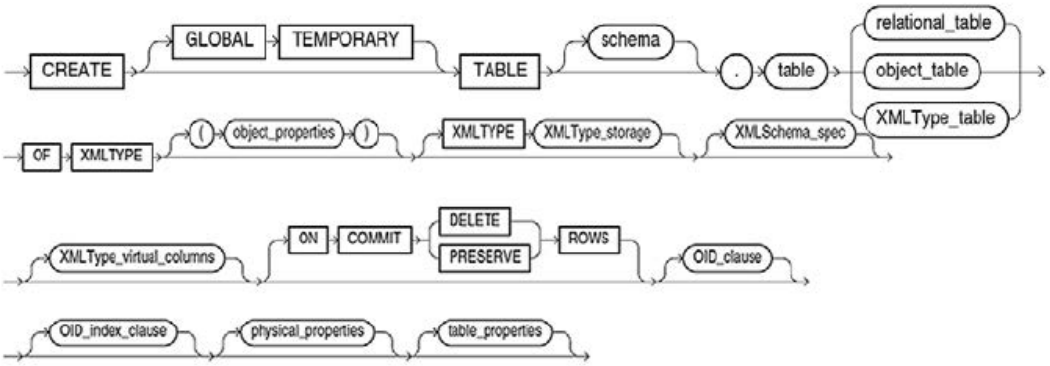


Fig. 6-3 Sintaxis para la creación de una tabla de tipo XML.

A continuación, se muestran en diversas figuras los elementos que completan la sintaxis de la creación de una tabla relacional:

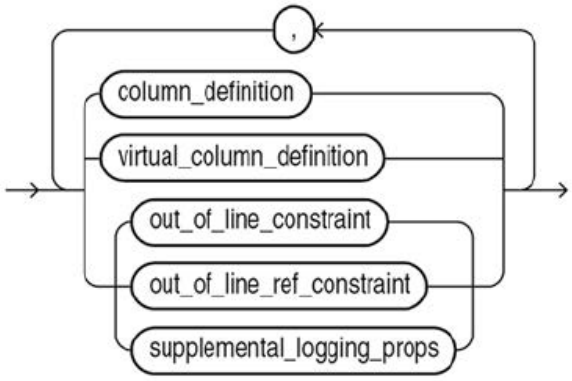


Fig. 6-4 Sintaxis del elemento RELATIONAL_PROPERTIES de la creación de una tabla relacional.

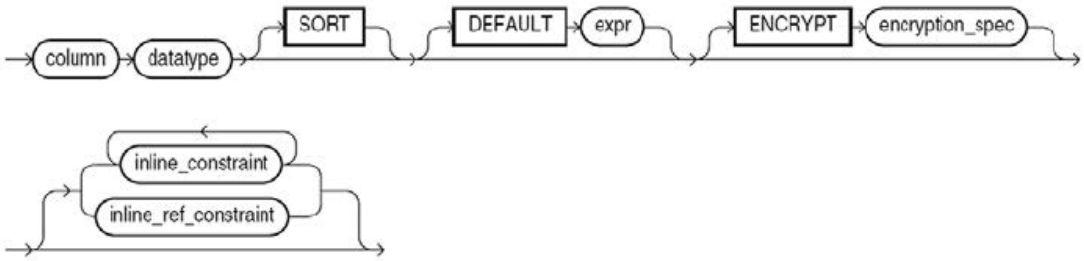


Fig. 6-5 Sintaxis del elemento COLUMN_DEFINITION del elemento RELATIONAL_PROPERTIES.

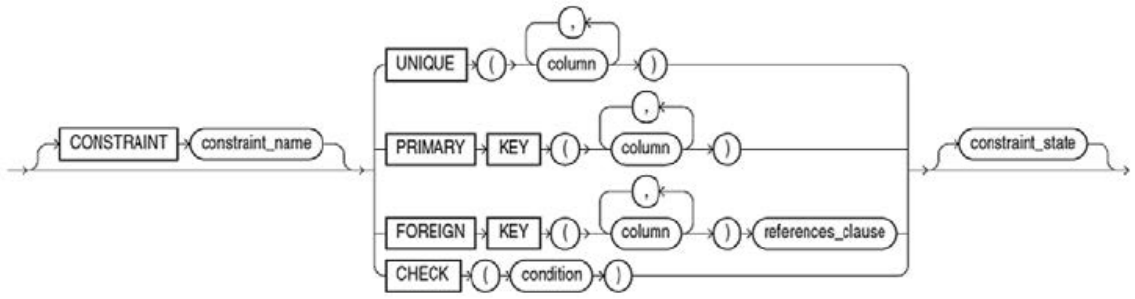


Fig. 6-6 Sintaxis del elemento OUT_OF_LINE_CONSTRAINT del elemento RELATIONAL_PROPERTIES.

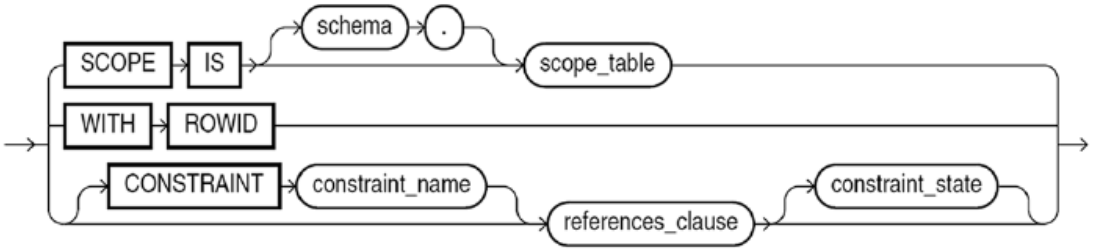


Fig. 6-7 Sintaxis del elemento INLINE_REF_CONSTRAINT del elemento RELATIONAL_PROPERTIES.

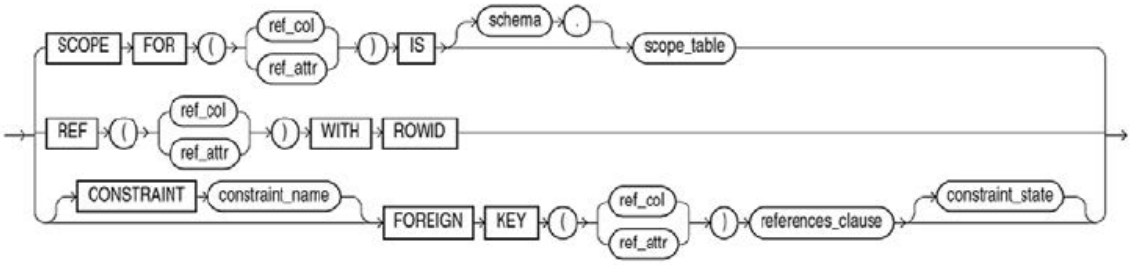


Fig. 6-8 Sintaxis del elemento OUT_OF_LINE_REFCONSTRAINT del elemento RELATIONAL_PROPERTIES.

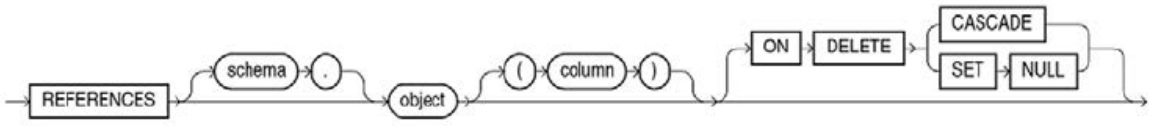


Fig. 6-9 Sintaxis del elemento REFERENCES_CLAUSE de los elementos CONSTRAINT vistos anteriormente.

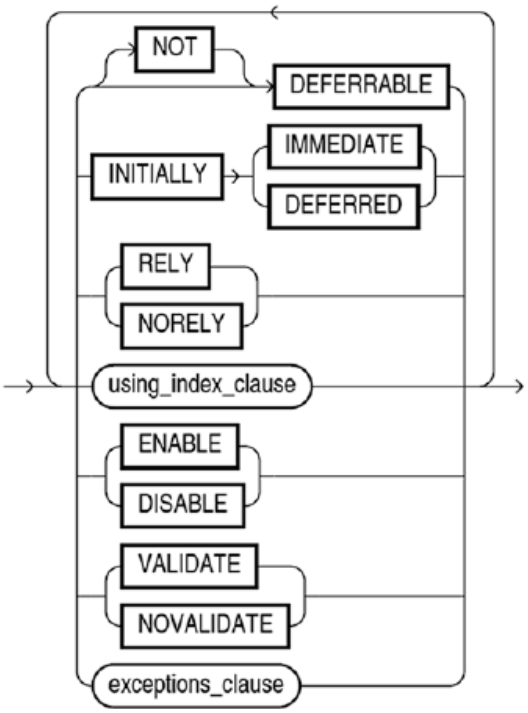


Fig. 6-10 Sintaxis del elemento CONSTRAINT_STATE de los elementos CONSTRAINT vistos anteriormente.

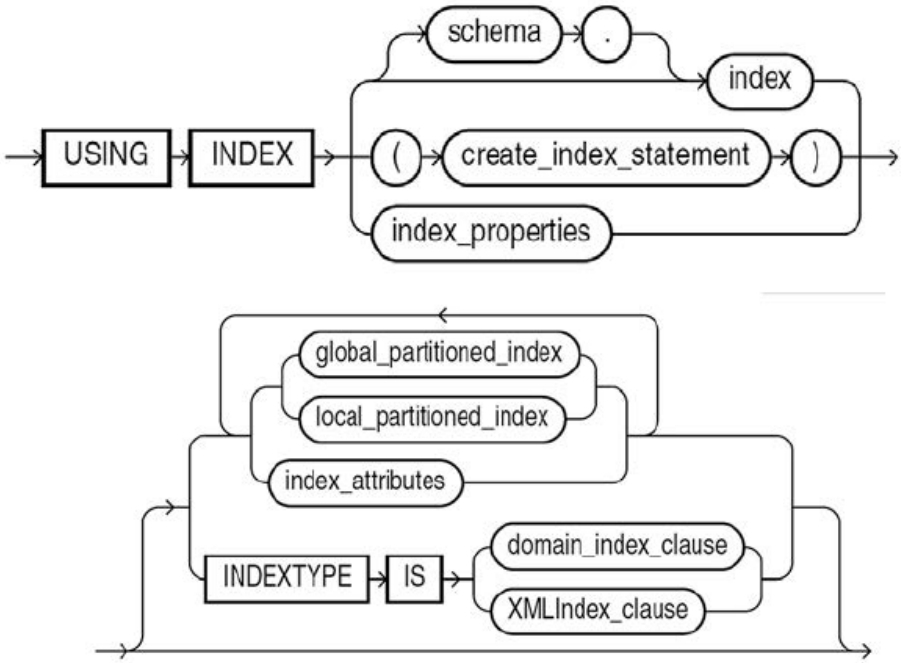


Fig. 6-11 Sintaxis del elemento USING_INDEX_CLAUSE del elemento CONSTRAINT_STATE e INDEX_PROPERTIES del elemento USING_INDEX_CLAUSE.

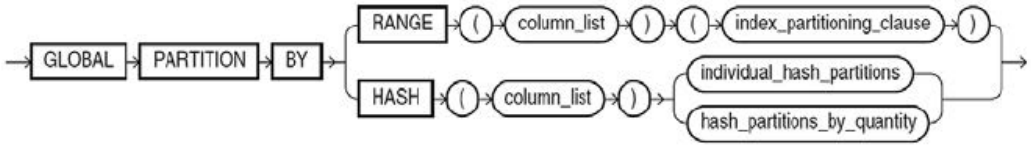


Fig. 6-12 Sintaxis del elemento GLOBAL_PARTITIONED_INDEX del elemento USING_INDEX_CLAUSE.

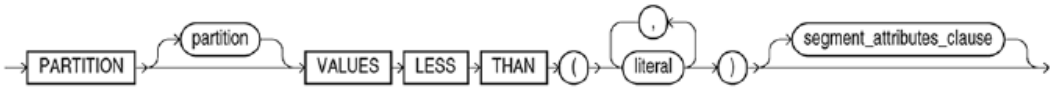


Fig. 6-13 Sintaxis del elemento INDEX_PARTITIONING_CLAUSE del elemento GLOBAL_PARTITIONED_INDEX.

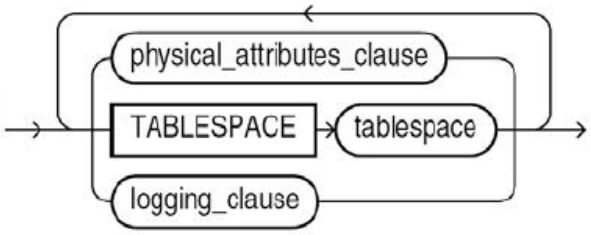


Fig. 6-14 Sintaxis del elemento SEGMENT_ATTRIBUTES_CLAUSE del elemento INDEX_PARTITIONING_CLAUSE.

El elemento PHYSICAL_ATTRIBUTES_CLAUSE también posee una sintaxis desgranada de elementos, pero queda fuera del estudio de este curso.

EJEMPLO 1. CREACIÓN DE UNA TABLA SIN RESTRICCIONES

```
CREATE TABLE ejemplo1
(empno        NUMBER,
nombre        VARCHAR2(10),
profesion     VARCHAR2(9),
fechaalta     DATE,
salario       NUMBER(10,2)
);
```

Este es el método más sencillo para crear una tabla que no posee ningún tipo de restricción.

Pero esto no es lo habitual, más bien todo lo contrario. Una tabla, por lo general, siempre lleva asociado algún tipo de restricción (constraint). En los siguientes ejemplos que se muestran en este capítulo se pueden observar los comandos necesarios para crear tablas con restricciones.

EJEMPLO 2. CREACIÓN DE UNA TABLA CON CLAVE PRIMARIA

```
CREATE TABLE ejemplo2
(empno        NUMBER CONSTRAINT    pk_ejemplo1
                     PRIMARY KEY,
nombre        VARCHAR2(10),
profesion     VARCHAR2(9),
fechaalta     DATE,
salario       NUMBER(10,2)
);
```

En este ejemplo creamos la misma tabla que en el ejemplo anterior, pero además, le indicamos cuál es la clave primaria de la misma.

La clave primaria es la composición de una o más columnas de la tabla creada, que hacen que podamos identificar unívocamente cada registro de la misma, y así diferenciarlos de los demás, de forma que no pueda haber 2 registros iguales. En este caso la clave primaria la forma una sola columna, que sería el número de empleado (empno).

EJEMPLO 3. CREACIÓN DE UNA TABLA CON CLAVE PRIMARIA (2 COLUMNAS)

```
CREATE TABLE ejemplo3
(coddepartamento   NUMBER(2),
codarea            NUMBER(2),
num_empleados      NUMBER(5),
salario            NUMBER(10,2),
CONSTRAINT pk_ejemplo3 PRIMARY KEY
(coddepartamento,codarea)
);
```

En este ejemplo creamos una tabla con indicación de una clave primaria compuesta de 2 columnas: el código del departamento (coddepartamento) y el código del área de dicho departamento (codarea).

Dado que un mismo departamento puede tener varias áreas de trabajo, la única forma de diferenciar un registro de otro es la composición de las 2 columnas para formar la clave primaria.

EJEMPLO 4. CREACIÓN DE UNA TABLA CON DOS RESTRICCIONES

```
CREATE TABLE ejemplo4
(empno        NUMBER     CONSTRAINT pk_ejemplo4
                         PRIMARY KEY,
nombre        VARCHAR2(10),
profesion     VARCHAR2(9),
fechaalta     DATE,
salario       NUMBER(10,2),
num_segsocial NUMBER(9)  CONSTRAINT uniq_ss UNIQUE
);
```

En este ejemplo además de indicar una clave primaria, indicamos que existe otra columna indexada dentro de la tabla: es la columna que almacena el número de la Seguridad Social (`num_segsocial`), que al ser un número que no se repite nunca, habrá de indicarse la restricción UNIQUE para evitar errores de grabación (duplicidad de números). Por tanto, esta restricción indica que los valores de dicha columna son únicos.

EJEMPLO 5. CREACIÓN DE UNA TABLA CON MÚLTIPLES RESTRICCIONES

```
CREATE TABLE ejemplo5
(empno        NUMBER       CONSTRAINT pk_ejemplo5
                           PRIMARY KEY,
nombre        VARCHAR2(10) NOT NULL,
profesion     VARCHAR2(9)  CONSTRAINT nn_profe
                           NOT NULL,
fechaalta     DATE,
salario       NUMBER(10,2),
num_segsocial NUMBER(9)    CONSTRAINT uniq_ss UNIQUE
);
```

En este ejemplo se aprecian dos formas distintas de indicar que los valores de una columna no pueden dejarse en blanco cuando se realiza la inserción de datos en la tabla. Esta indicación se consigue mediante la cláusula `NOT NULL`.

Se puede aplicar directamente sobre la columna, en cuyo caso el sistema asigna automáticamente un nombre de restricción, o dándole el propio usuario un nombre de restricción asociado a la restricción como es el caso de `nn_profe`.

EJEMPLO 6. CREACIÓN DE UNA TABLA CON VALORES POR DEFECTO

```
CREATE TABLE ejemplo6
(empno        NUMBER       CONSTRAINT pk_ejemplo6
                           PRIMARY KEY,
nombre        VARCHAR2(10) NOT NULL,
profesion     VARCHAR2(9)  CONSTRAINT nn_profe
                           NOT NULL,
fechaalta     DATE         CONSTRAINT df_fechalta
                           DEFAULT SYSDATE,
salario       NUMBER(10,2),
num_segsocial NUMBER(9)    CONSTRAINT uk_ejem6_ss
                           UNIQUE
);
```

En este ejemplo se muestra la forma de indicar un valor por defecto a un campo de una tabla para el supuesto de que no se asigne un valor para el mismo en el momento de la inserción. En ese caso se almacenará el valor por defecto que se indique a la derecha de la cláusula DEFAULT que se haya indicado en la creación de la tabla.

En nuestro caso si no se indica fecha de alta del empleado en el momento de insertar sus datos, se almacena la fecha del día, mediante la variable del sistema SYSDATE.

EJEMPLO 7. CREACIÓN DE UNA TABLA CON UNA RESTRICCIÓN DE TIPO CHECK

```
CREATE TABLE ejemplo7
(empno        NUMBER       CONSTRAINT pk_ejemplo6
                           PRIMARY KEY,
nombre        VARCHAR2(10) NOT NULL,
profesion     VARCHAR2(9)  CONSTRAINT nn_profe
                           NOT NULL,
fechaalta     DATE         CONSTRAINT df_fechalta
                           DEFAULT SYSDATE,
salario       NUMBER(10,2) CONSTRAINT ck_salario
                           CHECK(salario > 15000),
num_segsocial NUMBER(9)    CONSTRAINT uk_ejem6_ss
                           UNIQUE
);
```

Este ejemplo muestra la forma de restringir los valores de un campo dentro de un rango concreto.

Para nuestro caso concreto restringimos que para el campo salario no se puedan introducir valores inferiores o iguales a 15.000 euros.

La cláusula CHECK asociada con un criterio permite evaluar los datos antes de ser introducidos, a fin de determinar si cumplen o no las condiciones del criterio indicado.

EJEMPLO 8. CREACIÓN DE UNA TABLA CON RELACIONES A OTRAS

```
CREATE TABLE estados
(codigo   NUMBER(1)      CONSTRAINT pk_estados
                         PRIMARY KEY,
Descripción VARCHAR2(10) NOT NULL);


CREATE TABLE ejemplo7
(empno        NUMBER       CONSTRAINT pk_ejemplo8
                           PRIMARY KEY,
nombre        VARCHAR2(10) NOT NULL,
profesion     VARCHAR2(9)  NOT NULL,
fechaalta     DATE         CONSTRAINT df_fechalta
                           DEFAULT SYSDATE,
salario       NUMBER(10,2) CONSTRAINT ck_salario
                           CHECK(salario > 15000),
num_segsocial NUMBER(9)    CONSTRAINT uk_ejem6_ss
                           UNIQUE,
cod_estado    NUMBER(1)    NOT NULL,
CONSTRAINT fk_ejemplo8 FOREING KEY(cod_estado
REFERENCES estados(codigo) ON DELETE CASCADE
);
```

Este ejemplo muestra uno de los aspectos más importantes en cuanto al concepto relacional de una base de datos, y en cuanto a la integridad de la misma.

En primer lugar tenemos una tabla de estados (p.e.: “soltero”, “viudo”, “casado”, etc.), cuya clave primaria es el campo `codigo`.

Por otro lado hemos creado una tabla `ejemplo8`, en la que tenemos un campo denominado `cod_estado` que se encuentra referenciado al campo `codigo` de la tabla `estados`.

Cuando se relaciona el campo de una tabla con el campo de la otra mediante la cláusula FOREIGN KEY, se está indicando que los valores admitidos para el campo de la tabla que posee la instrucción FOREIGN KEY, serán únicamente los valores que existan en la tabla a la que se hace referencia. Con esto se consigue el concepto de integridad dado que la tabla `ejemplo8` no tendrá ningún código de estado ajeno a los existentes en la tabla `estados`.

La cláusula FOREIGN KEY puede formarse como combinación de una o más columnas, pero en todos los casos, las columnas a las que se hace referencia con la cláusula REFERENCES, tienen que ser columnas clave (clave primaria o UNIQUE) en la tabla correspondiente.

Además, la opción ON DELETE CASCADE permite que cuando se borre una fila de la tabla padre, y el campo de unión con la tabla hijo tenga un valor igual al borrado, también se borran en la tabla hijo, el/las filas con dicho valor.

INTEGRIDAD REFERENCIAL

El concepto de integridad referencial se aplica dentro de la creación de una tabla mediante la cláusula FOREIGN KEY. Es decir, cuando manejamos claves ajenas.

Este concepto nos va a permitir mantener una consistencia entre los datos relacionados en la tabla padre e hijo, de manera que las columnas que forman la clave ajena en la tabla hija no podrán tener valores distintos a sus homólogas referenciadas en la tabla padre.

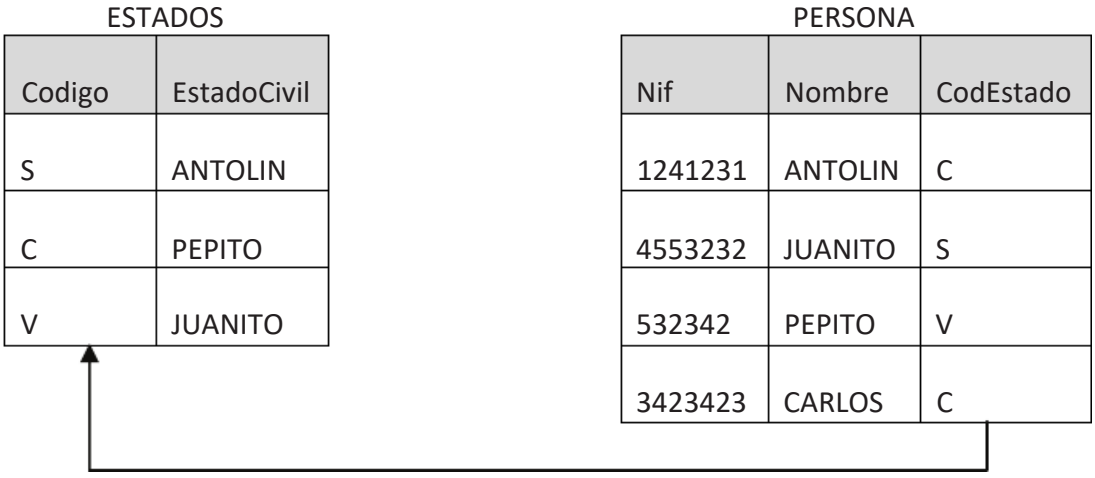
En el lenguaje estándar SQL se aprobaron en su momento las posibilidades de integridad referencial que se podían dar entre columnas de una tabla padre e hijo, pero no todas estas opciones están implementadas en los SGBD de Oracle. A continuación, vemos un cuadro con las opciones que se aprobaron en el lenguaje SQL y las que contiene el SGBD de Oracle.

Enunciadas en SQL estándar	Implementadas en Oracle
ON DELETE RESTRICT	SÍ
ON DELETE CASCADE	SÍ
ON UPDATE RESTRICT	NO
ON UPDATE CASCADE	NO implementada directamente.

Vamos a ver a continuación el significado de cada una, y cómo afectan a los datos. Asimismo, podremos enunciar cómo se indican mediante instrucciones del lenguaje SQL de Oracle.

On delete restrict

Esta cláusula impide que se borren datos en la tabla padre, siempre y cuando existan en la tabla hija datos en las columnas referenciadas, que coincidan con dicho valor.



Tomando el ejemplo anterior vemos que existe una tabla padre que es `Estados` y una tabla hija `Persona` donde existe una clave ajena que es `Persona.CodEstado` sobre la columna `Estados.Codigo`. Para crear dicha clave ajena, mediante código SQL de Oracle, utilizaríamos un constraint al final de la definición de la tabla `Persona` de la siguiente manera:

```
CONSTRAINT FK_PERSONA_ESTADOS FOREIGN KEY (codestado)
REFERENCES estados(codigo)
```

Si ahora quisiéramos borrar la línea de la tabla Estados que contiene en la columna codigo el valor de 'C' (casado), utilizaríamos la siguiente instrucción:

```
DELETE ESTADOS WHERE CODIGO = 'C';
```

El sistema nos devolverá un error indicándonos que no es posible llevar a cabo la operación, porque existe información de una tabla hija que referencia el valor de esta columna. En concreto, en nuestro ejemplo hay 2 columnas con el contenido 'C' (casado) en la columna CodEstado, que son la primera y la última.

On update restrict

Esta cláusula impide que se modifiquen datos en la tabla padre, siempre y cuando existan en la tabla hija datos en las columnas referenciadas, que coincidan con dicho valor.

La indicación de este tipo de restricción mediante sentencia SQL de Oracle es la misma que en la opción anterior:

```
CONSTRAINT FK_PERSONA_ESTADOS FOREIGN KEY (codestado)
REFERENCES estados(codigo)
```

Si ahora quisiéramos actualizar la línea de la tabla Estados que contiene en la columna codigo el valor de 'V' (viudo), utilizaríamos la siguiente instrucción:

```
UPDATE ESTADOS SET CODIGO = 'X' WHERE CODIGO = 'V';
```

El sistema nos devolverá un error indicándonos que no es posible llevar a cabo la operación, porque existe información de una tabla hija que referencia el valor de esta columna. En concreto, en nuestro ejemplo la 3ª fila tiene el valor 'V' en la columna CodEstado.

On delete cascade

Esta cláusula permite que se borren valores en la tabla padre aunque existan valores iguales al borrado en la hija. Pero para mantener la integridad referencial en los datos, también borra todas las filas de la tabla hija que contengan la clave de la fila borrada en el padre.

La indicación de este tipo de restricción mediante sentencia SQL de Oracle es la siguiente:

```
CONSTRAINT FK_PERSONA_ESTADOS FOREIGN KEY (codestado)
REFERENCES estados(codigo)
ON DELETE CASCADE
```

Si ahora quisiéramos borrar la línea de la tabla `Estados` que contiene en la columna `codigo` el valor de 'C' (casado), utilizaríamos la siguiente instrucción:

```
DELETE ESTADOS WHERE CODIGO = 'C';
```

El resultado que obtendremos en cuanto a las tablas de nuestro ejemplo sería el siguiente:

ESTADOS

Codigo	EstadoCivil
C	PEPITO
V	JUANITO

PERSONA

Nif	Nombre	CodEstado
4553232	JUANITO	S
532342	PEPITO	V

On update cascade

Esta cláusula permite que se actualicen valores en la tabla padre aunque existan valores iguales al borrado en la hija. Para mantener la integridad referencial en los datos, también se actualizan todas las filas de la tabla hija que contengan el valor actualizado en el padre.

Esta opción no tiene implementación en Oracle mediante una instrucción directa asociada a un CONSTRAINT, pero se puede llegar a realizar una implementación que posibilite la operación de ON UPDATE CASCADE, creando un TRIGGER asociado a la tabla padre.

SUPUESTO PRÁCTICO 3: ***Resolución en el Anexo I.***

Crear las estructuras de tablas, claves, restricciones, etc., del diagrama relacional resuelto en el Supuesto 1, por el que se diseñó la estructura de un hospital. Para ello se deberán realizar las siguientes acciones conducentes a obtener un diseño óptimo:

1. Convertir los elementos del modelo relacional a elementos de base de datos Oracle.
2. Codificar mediante sentencias SQL de Oracle la creación de los elementos necesarios para la creación de la base de datos de nuestro hospital: tablas, columnas, tipos de datos, tamaños, claves, relaciones, restricciones, etc.
3. Indicar el orden de creación de las tablas.
4. Ejecutar el código para la creación de cada uno de los elementos conectándose con el usuario creado en el supuesto anterior.

ALTERACIÓN DE UNA TABLA (ALTER TABLE)

El comando ALTER TABLE permite modificar una tabla creada para realizar las siguientes operaciones:

- Añadir una nueva columna.
- Modificar el nombre, tipo, tamaño y restricciones de una columna.
- Renombrar una tabla, una columna o una restricción.
- Alteración de los parámetros de almacenamiento de la tabla.
- Habilitar y deshabilitar o borrar una restricción.
- Añadir o modificar tipos objeto y restricciones sobre los tipos objeto.

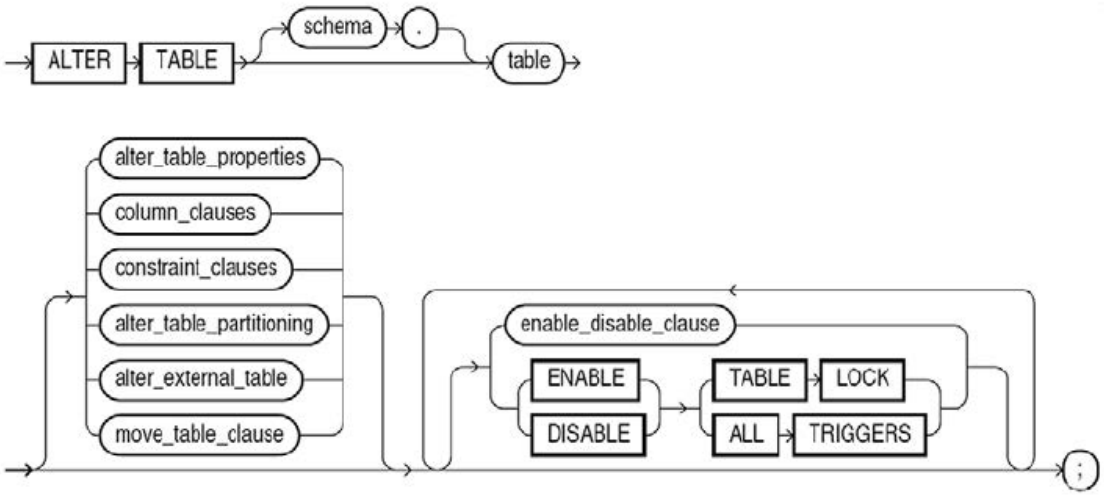


Fig. 6-15 Sintaxis para la alteración de una tabla.

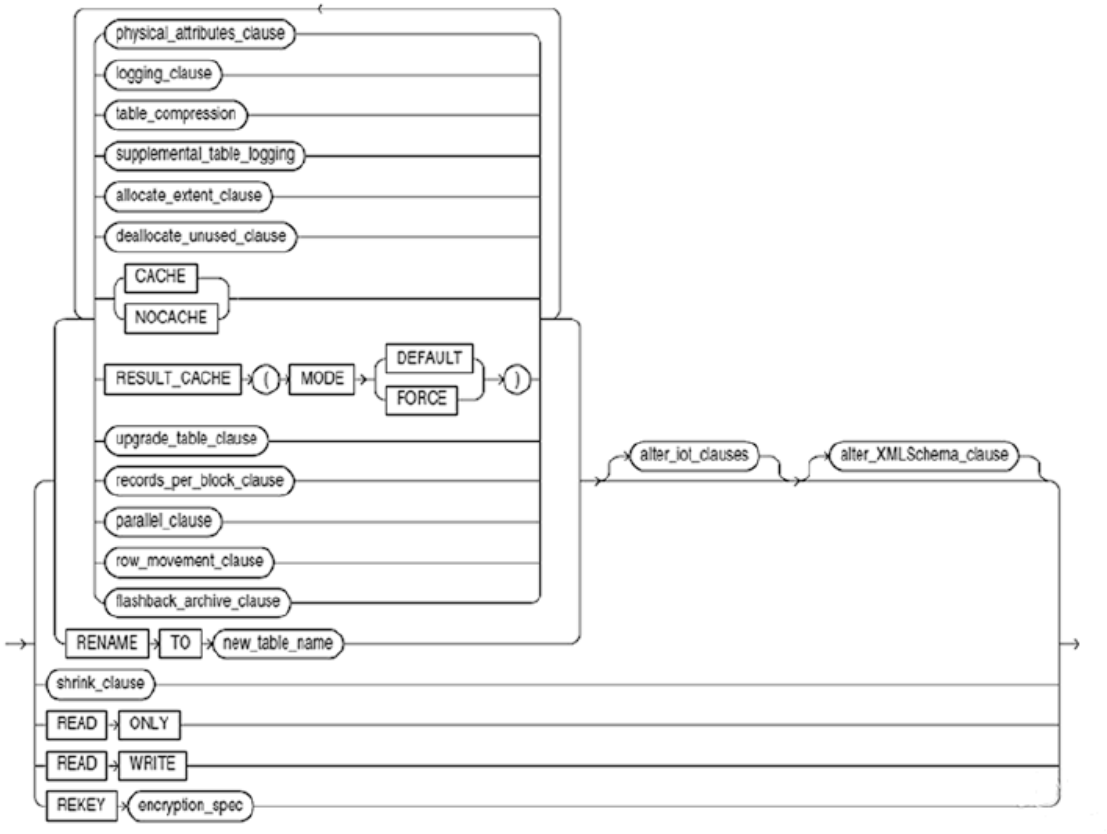


Fig. 6-16 Sintaxis del elemento ALTER_TABLE_PROPERTIES para la alteración de una tabla.

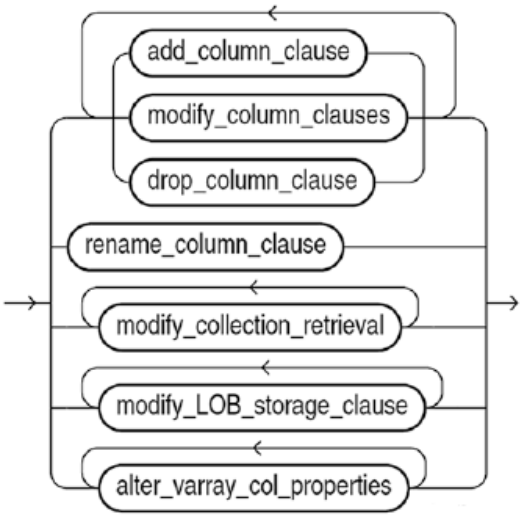


Fig. 6-17 Sintaxis del elemento COLUMN_CLAUSES para la alteración de una tabla.

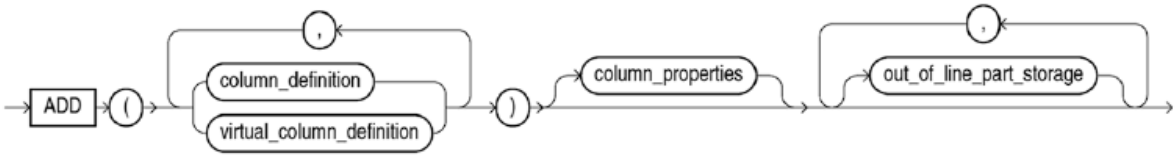


Fig. 6-18 Sintaxis del elemento ADD_COLUMN_CLAUSE del elemento COLUMN_CLAUSES.

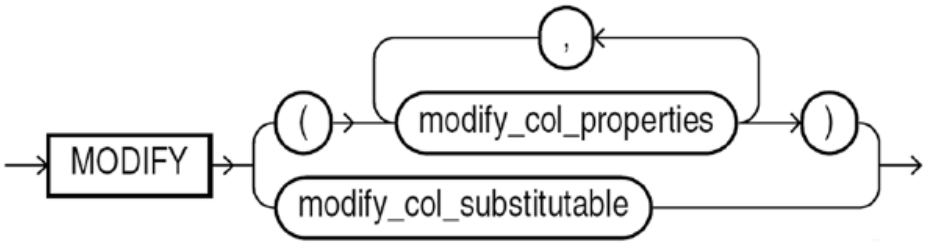


Fig. 6-19 Sintaxis del elemento MODIFY_COLUMN_CLAUSE del elemento COLUMN_CLAUSES.

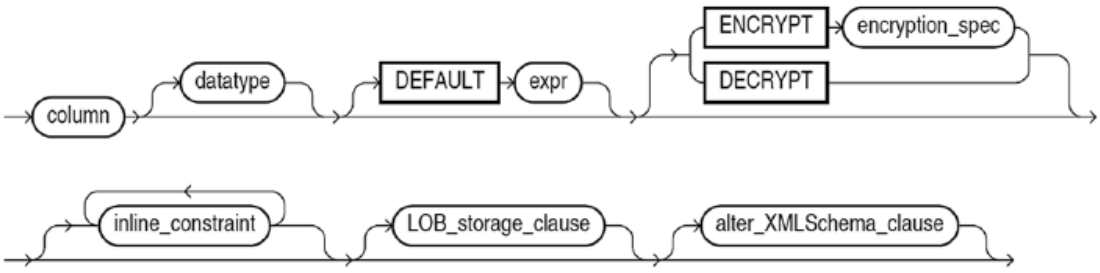


Fig. 6-20 Sintaxis del elemento MODIFY_COL_PROPERTIES del elemento MODIFY_COLUMN_CLAUSE

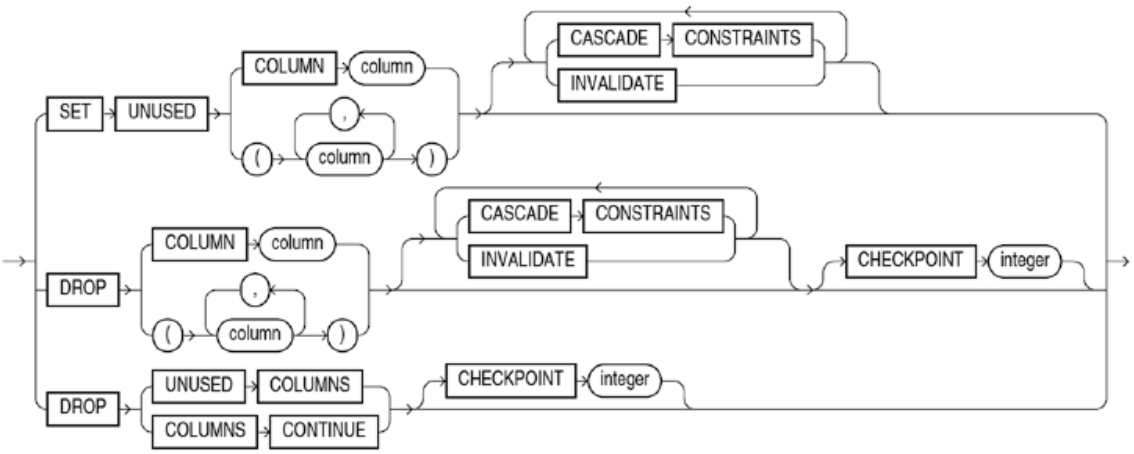


Fig. 6-21 Sintaxis del elemento DROP_COLUMN_CLAUSE del elemento COLUMN_CLAUSE

Fig. 6-22 Sintaxis del elemento RENAME_COLUMN_CLAUSE del elemento COLUMN_CLAUSE

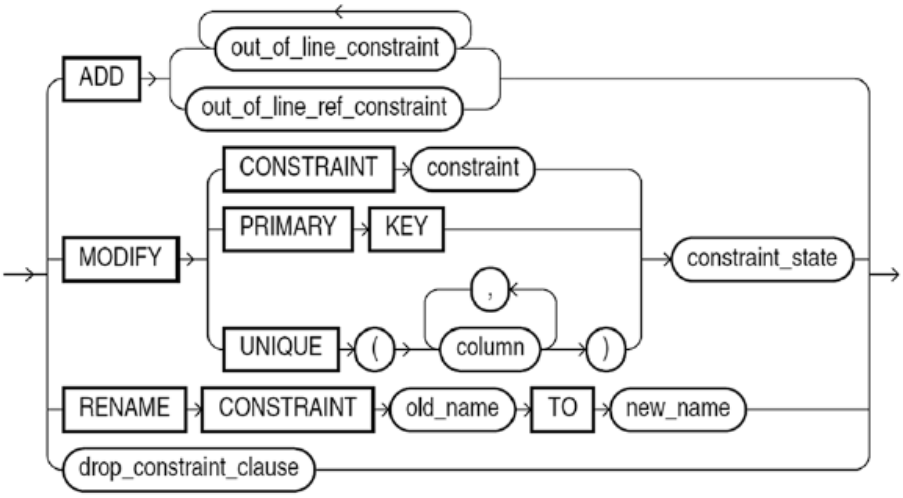


Fig. 6-23 Sintaxis del elemento CONSTRAINT_CLAUSES de alteración de una tabla.

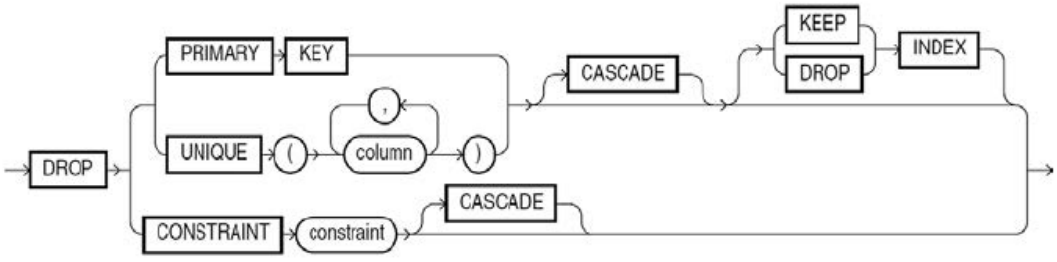


Fig. 6-24 Sintaxis del elemento DROP_CONSTRAINT_CLAUSE del elemento CONSTRAINT_CLAUSES.

EJEMPLO 1. ALTERACIÓN DE UNA TABLA PARA AÑADIR UNA NUEVA COLUMNA

```
ALTER TABLE ejemplo1
ADD (num_segsocial NUMBER(9) CONSTRAINT uk_ejem1_ss
                             UNIQUE);
```

En este ejemplo alteramos la tabla `ejemplo1,` añadiéndola una nueva columna denominada `numsegsocial,` que además va a tener una restricción de tipo clave única (no se pueden repetir valores en esta nueva columna).

EJEMPLO 2. ALTERACIÓN DE UNA TABLA PARA AÑADIR UNA NUEVA RESTRICCIÓN

```
ALTER TABLE ejemplo1
ADD CONSTRAINT pk_ejemplo1 PRIMARY KEY (empno);
```

En este ejemplo alteramos la tabla `ejemplo1,`añadiéndola una nueva restricción de tipo clave primaria, denominada `pk_ejemplo1.`

Hay que tener en cuenta que cuando se añade una nueva restricción a una tabla que ya contiene información, si los valores de la misma no cumplen las condiciones válidas para la restricción que se pretende añadir, esta no se creará. En nuestro caso, si existen valores para la columna `empno` vacíos (NULL) o repetidos, la restricción de clave primaria no se podrá crear, porque por definición una clave primaria no admite valores nulos ni repetidos para el/las columnas que forman dicha clave.

EJEMPLO 3. ALTERACIÓN DE UNA TABLA PARA MODIFICAR EL TIPO DE DATO DE UNA COLUMNA

```
ALTER TABLE ejemplo1
MODIFY (codigo NUMBER(20));
```

En este ejemplo alteramos la tabla `ejemplo1,` modificando el tipo de dato de la columna `codigo,` para que ahora sea de tipo numérica (NUMBER) de 20 dígitos.

Al modificar el tipo de dato de una columna que ya contiene información, hay que tener en cuenta que si los valores actuales no cumplen las condiciones del nuevo tipo de dato que se quiere dar para la misma, esta modificación no se podrá llevar a cabo. En nuestro caso, si existen valores para la columna `codigo` alfabéticos, alfanuméricos o con símbolos especiales, no se podrá realizar la modificación dado que por definición, un tipo de dato NUMBER únicamente admite valores numéricos.

EJEMPLO 4. ALTERACIÓN DE UNA TABLA PARA RENOMBRARLA

```
ALTER TABLE ejemplo1
RENAME TO eje1;
```

En este ejemplo alteramos la tabla `ejemplo1` para renombrarla, y que ahora pase a llamarse `eje1`.

EJEMPLO 5. ALTERACIÓN DE UNA TABLA PARA RENOMBRAR UNA COLUMNA

```
ALTER TABLE ejemplo1
RENAME COLUMN codigo_alum TO codigo_alumno
```

En este ejemplo alteramos la tabla `ejemplo1` para cambiar el nombre de la columna `codigo_alum` al nuevo nombre: `codigo_alumno.`

EJEMPLO 6. ALTERACIÓN DE UNA TABLA PARA RENOMBRAR UNA RESTRICCIÓN

```
ALTER TABLE ejemplo1
RENAME CONSTRAINT pk_ejemplo1 TO
clave_primaria_ejemplo1;
```

En este ejemplo alteramos la tabla `ejemplo1` para cambiar el nombre de la restricción `pk_`ejemplo1 al nuevo nombre: `clave_primaria_ejemplo1.`

EJEMPLO 7. ALTERACIÓN DE UNA TABLA PARA DESHABILITAR UNA CONSTRAINT

```
ALTER TABLE ejemplo1
DISABLE CONSTRAINT pk_ejemplo1;
```

En este ejemplo alteramos la tabla `ejemplo1` para deshabilitar la restricción `pk_ejemplo1`.

Con la deshabilitación de una restricción conseguimos, sin borrar la misma de la tabla, que las condiciones que se estaban imponiendo a través de la misma, dejen de operar y no se validen contra los nuevos datos introducidos en la tabla.

EJEMPLO 8. ALTERACIÓN DE UNA TABLA PARA HABILITAR UNA CONSTRAINT

```
ALTER TABLE ejemplo1
ENABLE CONSTRAINT pk_ejemplo1;
```

En este ejemplo alteramos la tabla `ejemplo1` para habilitar la restricción `pk_ejemplo1` que estaba deshabilitada.

Cuando se habilita una restricción, automáticamente el SGBD valida las condiciones de la misma contra toda la información que existe en la tabla asociada, para las columnas que operan sobre la restricción. Si alguna información contenida en la tabla no cumple las condiciones que impone la restricción, no se podrá habilitar la restricción.

EJEMPLO 9. ALTERACIÓN DE UNA TABLA PARA BORRAR UNA CLAVE PRIMARIA Y CLAVES AJENAS ASOCIADAS

```
ALTER TABLE ejemplo1
DROP PRIMARY KEY CASCADE;
```

En este ejemplo alteramos la tabla `ejemplo1` para borrar la clave primaria de la tabla, y además todas las claves ajenas que se hayan definido sobre las columnas de la clave primaria eliminada.

EJEMPLO 10. ALTERACIÓN DE UNA TABLA PARA BORRAR UNA RESTRICCIÓN Y CLAVES AJENAS ASOCIADAS

```
ALTER TABLE ejemplo1
DROP CONSTRAINT pk_dept CASCADE;
```

En este ejemplo alteramos la tabla `ejemplo1` para borrar la restricción `pk_dept` y además todas las claves ajenas que se hayan definido sobre las columnas de dicha restricción.

EJEMPLO 11. ALTERACIÓN DE UNA TABLA PARA LIBERAR EL ESPACIO SIN UTILIZAR UNA TABLA

```
ALTER TABLE ejemplo1
DEALLOCATE UNUSED;
```

En este ejemplo alteramos la tabla `ejemplo1` para liberar todo el espacio sin utilizar que tiene asociado la tabla.

EJEMPLO 12. ALTERACIÓN DE UNA TABLA PARA BORRAR UNA COLUMNA

```
ALTER TABLE ejemplo1
DROP (codigo);
```

En este ejemplo alteramos la tabla `ejemplo1` para borrar la columna `codigo`.

Hay que tener en cuenta que podremos borrar una columna de una tabla cuando la misma no forme parte de ninguna clave o esté referenciada por claves ajenas desde otras tablas.

EJEMPLO 13. ALTERACIÓN DE UNA TABLA PARA BORRAR UNA COLUMNA Y RESTRICCIONES REFERENCIADAS A LA MISMA

```
ALTER TABLE ejemplo1
DROP (codigo) CASCADE CONSTRAINTS;
```

En este ejemplo alteramos la tabla `ejemplo1` para borrar la columna `codigo` y todas aquellas restricciones que haya sobre la misma, o que provengan de una FOREIGN KEY hacia dicha columna, desde otras tablas.

Este sería el método para solventar el problema expuesto en el ejemplo 12 en el que se diera el caso de querer borrar una columna con restricciones o claves ajenas que la referencian.

EJEMPLO 14. ALTERACIÓN DE UNA TABLA PARA BORRAR MÚLTIPLES COLUMNAS

```
ALTER TABLE ejemplo1
DROP (codigo, nombre, apellidos);
```

En este ejemplo alteramos la tabla `ejemplo1` para borrar las columnas `codigo`, `nombre` y `apellidos`.

BORRADO DE UNA TABLA (DROP TABLE)

El comando DROP TABLE permite borrar una tabla y todos los elementos asociados a la misma (índices, restricciones, columnas, valores de las mismas, etc.).

En el único caso que el SGBD no permite borrar una tabla es cuando existen dependencias con otras (es la tabla padre de otra), en ese caso habrían de borrarse primero la/las tablas hijo, y luego la tabla padre, o bien anular las restricciones de tipo FOREIGN KEY que existiesen en las tablas hijo.

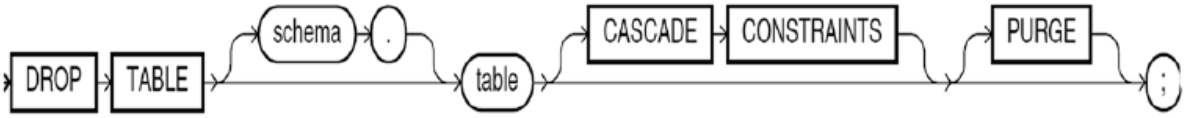


Fig. 6-25 Sintaxis para el borrado de una tabla.

La opción CASCADE CONSTRAINTS borra también todas las dependencias con otras tablas; es decir, evita los errores al intentar borrar una tabla que posea sobre sus claves restricciones de tipo FOREIGN KEY desde otras.

EJEMPLO 1. BORRADO BÁSICO DE UNA TABLA

```
DROP TABLE ejemplo1;
```

En este ejemplo borramos la tabla `ejemplo1,` sus propias restricciones y sus valores. Si hubiese restricciones desde otras tablas (FOREIGN KEY), no se podría borrar la tabla con este comando.

EJEMPLO 2. BORRADO DE UNA TABLA Y TODAS SUS RESTRICCIONES PROPIAS Y AJENAS

```
DROP TABLE ejemplo1 CASCADE CONSTRAINTS
```

En este ejemplo borramos la tabla `ejemplo1,` sus propias restricciones, sus valores y cualquier otra restricción ajena que se derive desde otras tablas hacia ella.

SUPUESTO PRÁCTICO 4: ***Resolución en el Anexo I.***

Supuesto para el manejo de los comandos de alteración sobre tablas y restricciones.

- **Tabla ESTADOS_CIVILES**
  - **Codigo CHAR(1) CLAVE PRIMARIA**
  - **Descripcion VARCHAR2(50) NO NULA**

- **Tabla PERSONA**
  - **Nif VARCHAR2(9) CLAVE PRIMARIA**
  - **Codestadocivil NUMBER(1) NO NULA**
  - **Nombre VARCHAR2(100) NO NULA**

Se han de realizar mediante código SQL las siguientes operaciones:

1. Crear las tablas que aparecen en el enunciado de este supuesto, según la descripción de columnas y restricciones que aparecen, sin añadir más restricciones de las que se indican.
2. Alterar la tabla `estados_civiles` para añadir una nueva restricción sobre la columna `codigo`, de manera que únicamente se permita introducir en dicha columna los siguientes valores ('S','C','V','O').
3. Alterar la tabla `persona` de manera que se modifique el tipo del campo `codestadocivil` que tendrá que pasar a ser de tipo CHAR(1).
4. Añadir una nueva restricción a la tabla `persona` de manera que se restrinjan los valores de la columna `codestadocivil` a los mismos que pueda tener la columna `codigo` de la tabla `estados_civiles`.
5. Alterar la tabla `persona` para deshabilitar la restricción creada en el punto anterior.

MANEJO DE ÍNDICES

Un índice se crea automáticamente mediante la indicación de la clave primaria (PRIMARY KEY), alternativa (UNIQUE) o ajena (FOREIGN KEY).

Además, existe la posibilidad de que el usuario cree nuevos índices, indicando si los mismos son de tipo único (UNIQUE) o no.

Durante este apartado trataremos tres operaciones sobre los índices:

- Creación de índices: CREATE INDEX.
- Alteración de la estructura de un índice: ALTER INDEX.
- Borrado de un índice: DROP INDEX.

Creación de índices (CREATE INDEX)

Al crear un índice podemos indicar que sea de uno de los siguientes tipos:

- Único: indica que los valores del índice no se pueden repetir por cada fila.
- No único: indica que los valores del índice se pueden repetir por fila.

Los índices siempre se crean con almacenamiento en orden ascendente de sus datos.

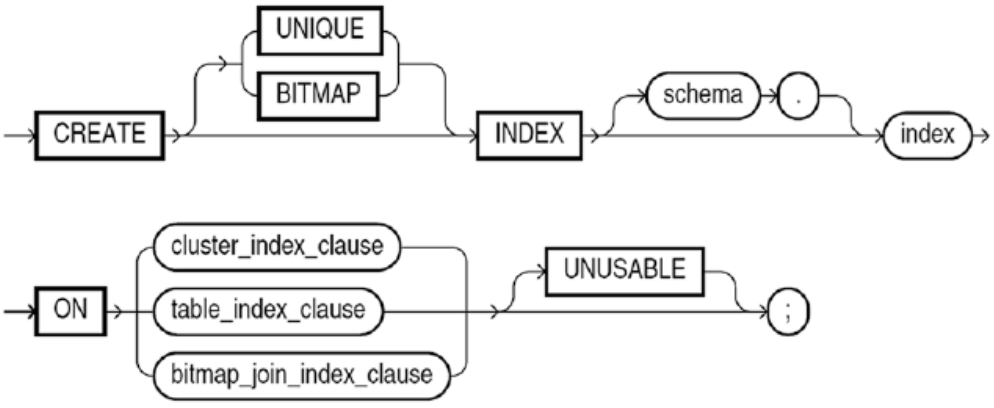


Fig. 6-26 Sintaxis para la creación de un índice.

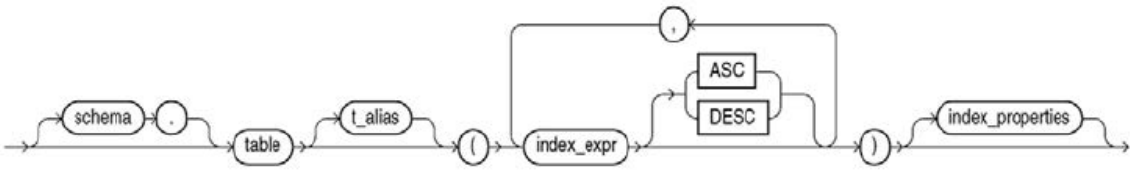


Fig. 6-27 Sintaxis del elemento TABLE_INDEX_CLAUSE para la creación de un índice.

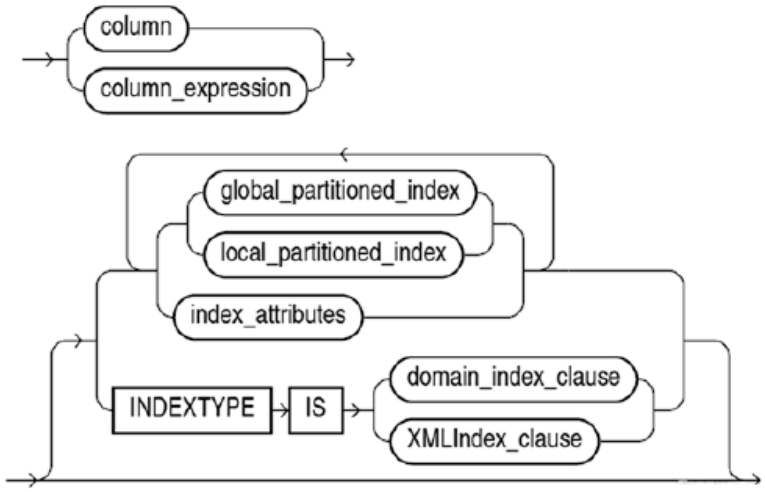


Fig. 6-28 Sintaxis del elemento INDEX_EXPR e INDEX_PROPERTIES del elemento TABLE_INDEX_CLAUSE.

EJEMPLO 1. CREACIÓN DE UN ÍNDICE CON REPETICIONES (NO ÚNICO)

```
CREATE INDEX inx_ejemplo7 ON ejemplo7 (nombre);
```

En este ejemplo creamos un índice denominado `inx_ejemplo7` sobre la tabla `ejemplo7`, para la columna `nombre`, admitiéndose repeticiones de valores.

EJEMPLO 2. CREACIÓN DE UN ÍNDICE SIN REPETICIONES (ÚNICO)

```
CREATE UNIQUE INDEX inx_u_ejemplo7
ON ejemplo7 (nombre);
```

En este ejemplo creamos un índice denominado `inx_u_ejemplo7` sobre la tabla `ejemplo7` para la columna `nombre`, no admitiéndose repeticiones de valores. Por tanto en la práctica, a parte del índice, estamos creando una nueva restricción por la que se impide que se inserten valores repetidos en la columna `nombre` de la tabla `ejemplo7`. Esto sería equivalente a haber creado la tabla de la siguiente forma:

```
CREATE TABLE ejemplo7
(…
, nombre VARCHAR2(10) CONSTRAINT inx_u_ejemplo7
                      UNIQUE,
…
);
```

También sería equivalente a la alteración de la tabla para añadir una nueva restricción de la siguiente forma:

```
ALTER TABLE ejemplo7
ADD CONSTRAINT inx_u_ejemplo7 UNIQUE;
```

Alteración de índices (ALTER INDEX)

La alteración de un índice permite realizar las siguientes operaciones sobre un índice existente:

- Modificar la estructura de almacenamiento del índice.
- Renombrar el índice.
- Inutilizar un índice para que no se aplique o no se utilice sobre la tabla.

Cuando se crea un índice no se pueden alterar los siguientes elementos del mismo:

- El/las columnas que forman parte del índice.
- El tipo de índice (único o no único).

Cuando es necesario alterar alguno de los elementos anteriores, la única posibilidad es borrar el índice y volverlo a crear con las nuevas modificaciones.

Fig. 6-29 Sintaxis para la alteración de un índice.

EJEMPLO 1. ALTERACIÓN DE UN ÍNDICE PARA CAMBIARLO DE NOMBRE

```
ALTER INDEX ejemplo1_inx RENAME TO inx_ejemplo1;
```

En el ejemplo alteramos el índice `ejemplo1_inx` para cambiarle su nombre, pasándose a llamar ahora: `inx_ejemplo1`.

EJEMPLO 2. ALTERACIÓN DE UN ÍNDICE PARA INUTILIZARLO

```
ALTER INDEX ejemplo1_inx UNUSUABLE;
```

En el ejemplo alteramos el índice `ejemplo1_inx` para inutilizarlo. De esta forma deja de funcionar el índice sobre la tabla que fue definido, pero no se borra de la base de datos.

Borrado de índices (DROP INDEX)

El borrado de un índice permite eliminarlo de la base de datos.

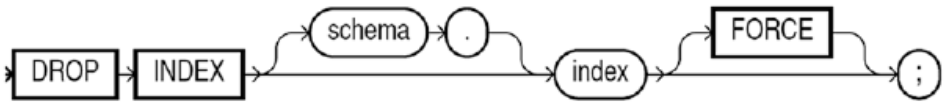


Fig. 6-30 Sintaxis para el borrado de un índice.

EJEMPLO 1. BORRADO DE UN ÍNDICE

```
DROP INDEX ejemplo1_inx;
```

En el ejemplo borramos el índice `ejemplo1_inx` de la base de datos.

SUPUESTO PRÁCTICO 5: ***Resolución en el Anexo I.***

A fin de poder realizar búsquedas de enfermos por su nombre de pila o por el apellido del mismo, se quiere indexar la tabla `enfermo`.

1. Crear un índice para el campo `nombre` que admita repeticiones.
2. Crear un índice para el campo `apellidos` que admita repeticiones.

También se tiene que indexar la tabla `departamentos` a fin de crear un índice único para el campo `nombre`, de manera que puedan repetirse los nombres de departamentos.

MANEJO DE VISTAS

Las vistas son máscaras que se definen sobre una o varias tablas y están ligadas a operaciones de consulta (SELECT) sobre las mismas.

Durante este apartado trataremos tres operaciones sobre las vistas:

- CREATE VIEW: Creación de una vista.
- ALTER VIEW: Alteración de la estructura de una vista.
- DROP VIEW: Borrado de una vista.

Creación de una vista (CREATE VIEW)

Una vista se crea a partir de una sentencia de consulta (SELECT) sobre una o varias tablas.

De no especificarse nada en su creación admite la manipulación de los datos (inserción, borrado o modificación), siempre y cuando la consulta que se haya diseñado también lo permita.

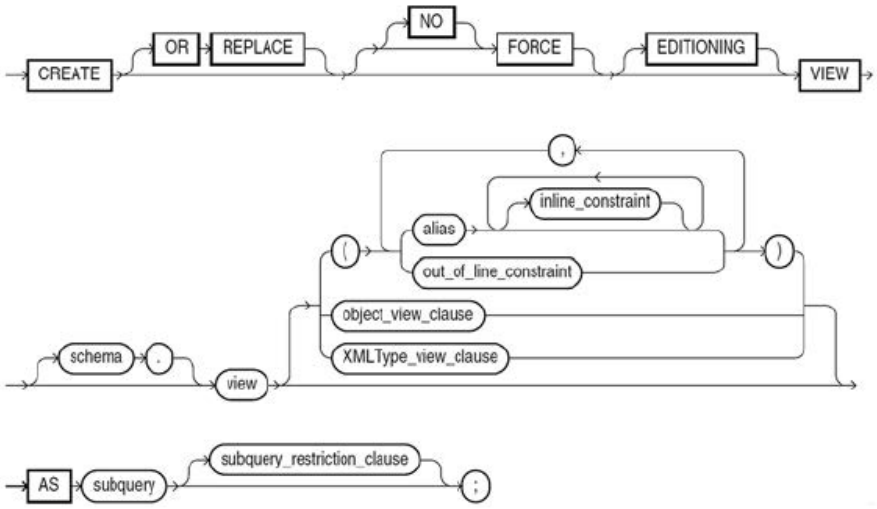


Fig. 6-31 Sintaxis para la creación de una vista.

Fig. 6-32 Sintaxis del elemento SUBQUERY_RESTRICTION_CLAUSE de la creación de una vista.

Cuando creamos una vista podemos utilizar dos tipos de sintaxis:

- `CREATE:` crea la vista y en caso de existir una vista con el mismo nombre, nos da un error por existencia de un objeto nominado de igual forma.

- `CREATE OR REPLACE:` crea la vista y en caso de existir una vista con el mismo nombre reemplaza su contenido por el nuevo.

Hay que considerar también que ocurre con las tablas y permisos que afectan a la/las tablas de la consulta asociada:

- `FORCE:` crea la vista con independencia de la existencia de las tablas, o aunque el usuario que crea la vista no tenga permisos sobre las tablas

afectadas. Pero esta vista no presentará datos hasta que existan las tablas y el usuario tenga permisos sobre las mismas.

- `NO FORCE:` es la opción predeterminada al crear una vista aunque no se indique expresamente, y supone que solo se llevará a cabo si las tablas existen y el usuario que crea la vista tiene permisos de acceso a las mismas.

- `WITH READ ONLY:` impide que se puedan realizar operaciones de inserción, borrado y modificación de datos sobre las tablas asociadas a la vista.

- `WITH CHECK OPTION:` obliga a que los datos que se inserten o actualicen en la vista, cumplan con las condiciones de la consulta asociada. A este tipo de restricción el SGBD de Oracle le asigna un nombre por defecto, a menos que se añada la cláusula CONSTRAINT y nombre de restricción.

EJEMPLO 1. CREACIÓN DE UNA VISTA SOBRE UNA TABLA

```
CREATE OR REPLACE VIEW departamento20
AS SELECT descridpto, numempleados
   FROM empleados
   WHERE numdpto = 20;
```

En este ejemplo creamos una vista sobre la tabla `empleados` que únicamente nos muestra los datos del departamento 20. La opción `OR REPLACE` permite que cuando insertemos de nuevo una vista con el nombre `departamento20`, si ya existe una con ese nombre la reemplace por la nueva.

EJEMPLO 2. CREACIÓN DE UNA VISTA RENOMBRANDO COLUMNAS Y APLICANDO RESTRICCIÓN EN LOS VALORES DE ENTRADA

```
CREATE OR REPLACE VIEW administrativos
(cod_empleado, nombre, departamento, puesto)
AS SELECT empno, ename, deptno, job
FROM empleados
WHERE job = ´ADMINISTRATIVO´
WITH CHECK OPTION CONSTRAINT
ck_vista_administrativos;
```

En este ejemplo creamos una vista sobre la tabla `empleados` con el fin de mostrar los datos de trabajadores que ocupen puestos de 'ADMINISTRATIVO'. Para ello renombramos los campos de la tabla `empleados` para que sean comprensibles.

Además, se ha añadido la cláusula WITH CHECK OPTION. La función de la misma es evitar que se añadan o modifiquen datos en la vista (es decir, en la tabla empleados) que sean distintos al puesto de 'ADMINISTRATIVO', dado que si incluyésemos valores distintos a este, no sería posible verlos en esta vista. A la restricción del tipo CHECK OPTION le indicamos como nombre de restricción ck_vista_administrativos.

EJEMPLO 3. CREACIÓN DE UNA VISTA FORZADA

```
CREATE OR REPLACE FORCE VIEW v_empleado
AS SELECT nif, nombre FROM empleado
WHERE cod_empleado = 10;
```

En este ejemplo creamos una vista llamada v_empleado en la que forzamos la creación de la misma, con independencia de la existencia de la tabla empleado y los permisos que pudiera tener el usuario que crea la vista.

EJEMPLO 4. CREACIÓN DE UNA VISTA DE SOLO LECTURA

```
CREATE OR REPLACE VIEW departamento20
AS SELECT descridpto, numempleados
FROM empleados
WHERE numdpto = 20
WITH READ ONLY;
```

En este ejemplo creamos una vista sobre la tabla empleados que únicamente nos muestra los datos del departamento 20.

Esta vista no nos permite hacer operaciones de inserción, modificación y borrado de datos, sobre la tabla empleados.

Alteración de una vista (ALTER VIEW)

La alteración de una vista no permite modificar la estructura creada mediante la instrucción CREATE VIEW; es decir, no permite modificar la consulta asociada, sino que únicamente permite recompilarla para determinar si es válida.

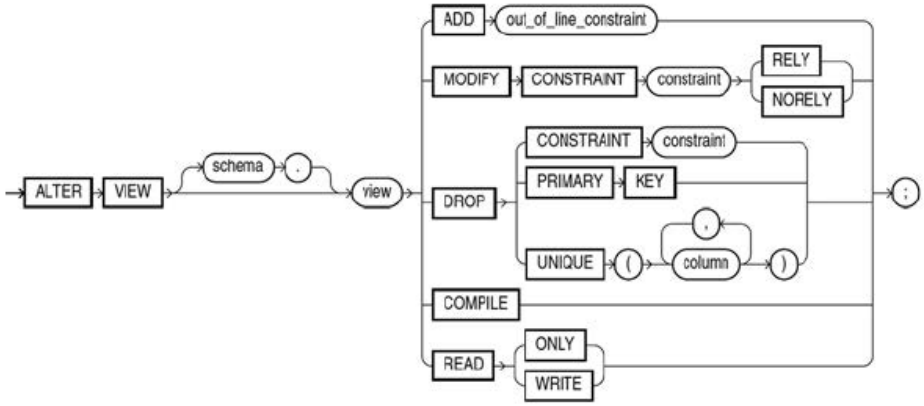


Fig. 6-33 Sintaxis para la alteración de una vista.

EJEMPLO 1. ALTERACIÓN DE UNA VISTA

```
ALTER VIEW vista_cliente COMPILE;
```

En este ejemplo alteramos la vista `vista_cliente` para compilar el código SQL que tiene asociado, es decir, la consulta.

Borrado de una vista (DROP VIEW)

El borrado de una vista lo elimina de la base de datos.

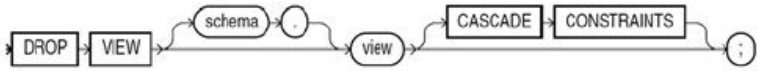


Fig. 6-34 Sintaxis para el borrado de una vista.

EJEMPLO 1. BORRADO DE UNA VISTA

```
DROP VIEW vw_administrativos;
```

En este ejemplo borramos y eliminamos de la base de datos la vista `vw_administrativos.`

SUPUESTO PRÁCTICO 6: ***Resolución en el Anexo I.***

Crear las siguientes vistas de solo lectura para que los usuarios las consulten. Se indica, para cada vista, las consultas (querys) a utilizar.

1. Crear una vista que muestre los pacientes de un hospital concreto, solicitando el nombre del hospital. La consulta que hay que añadir es:

```
SELECT e.nombre, e.apellidos, e.direccion,
e.sexo, e.fnacimiento,
e.numsegsocial,he.finscripcion
FROM hospital h, enfermo e, hospital_enfermo he
WHERE
    h.nombre = '&Indique_el_nombre_de_hospital'
AND he.hosp_codigo = h.codigo
AND  e.numsegsocial = he.enf_numsegsocial
ORDER BY e.apellidos, e.nombre, he.finscripcion;
```

2. Crear una vista de solo lectura que pueda mostrar todos los miembros de la plantilla según el turno que indique el usuario. La consulta que hay que añadir es:

```
SELECT p.nombre AS "NOMBRE EMPLEADO",
p.apellidos, p.funcion, s.nombre AS "NOMBRE
SALA", h.nombre AS "NOMBRE HOSPITAL"
FROM hospital h, sala s, plantilla p,
plantilla_sala ps
WHERE
    p.turno = '&Indique_un_turno'
AND ps.plan_nif = p.nif
AND s.hosp_codigo = ps.sala_hosp_codigo
AND s.codigo = ps.sala_codigo
AND h.codigo = s.hosp_codigo;
WITH READ ONLY;
```

SECUENCIAS

Una secuencia es un objeto de la base de datos que como tal se almacena, siendo poseedor un usuario de la misma, y que permite la generación automática de números dado un número de comienzo y un incremento.

Es la alternativa al tipo de dato autonumérico que existe en otras bases de datos, como, por ejemplo, Microsoft Access.

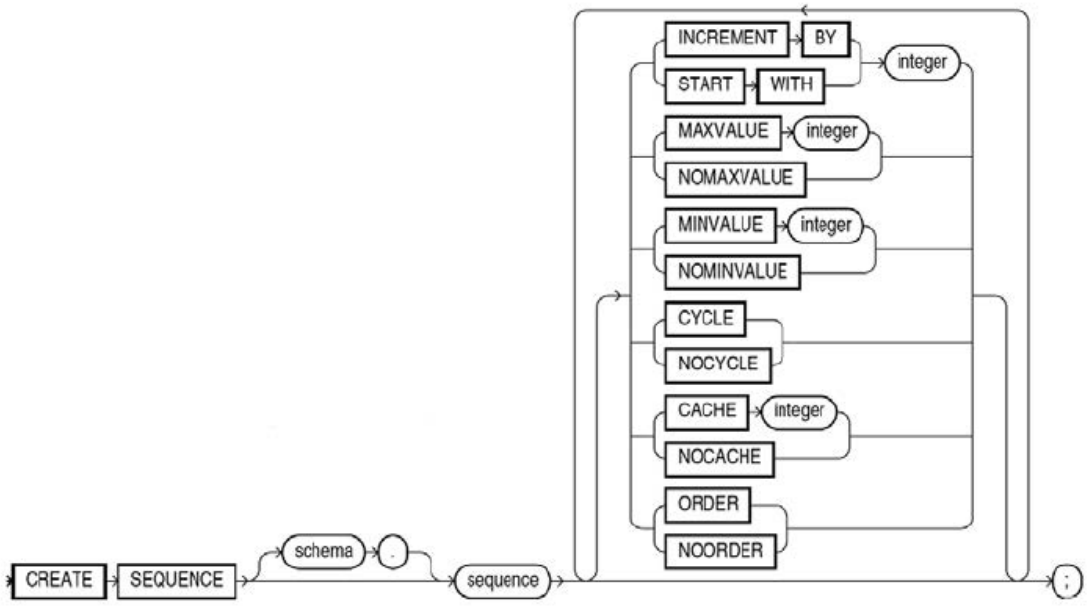


Fig. 6-35 Sintaxis para la creación de una secuencia.

La sintaxis textual en modo comando SQL sería la siguiente:

```
CREATE SEQUENCE nombre
[INCREMENT BY integer]
START WITH integer
[MAXVALUE integer
MINVALUE integer]
[CYCLE / NOCYCLE]
[ORDER / NOORDER]
[CACHE entero / NOCACHE];
```

Parámetros de una secuencia

Los valores `integer` asociados a la sintaxis textual vista anteriormente solo admiten números enteros de un máximo de 28 dígitos.

A continuación, se define cada uno de los parámetros disponibles para la creación de la secuencia:

- `INCREMENT BY`: especifica el crecimiento que va a tener la secuencia. Si el valor es positivo, la secuencia ascenderá según el incremento indicado, y si el valor es negativo, decrecerá. Si se omite, el valor por defecto es 1. No está permitido asociarle un valor.

- `START WITH:` especifica el valor de comienzo de la secuencia.

- MAXVALUE: especifica el valor máximo que puede alcanzar la secuencia. Si se indica, también habrá que especificar contenido para el parámetro MINVALUE.

- MINVALUE: especifica el valor mínimo que pueda alcanzar la secuencia. Si se indica, también habrá que especificar contenido para el parámetro MAXVALUE.

- CYCLE: se utiliza para especificar que cuando la secuencia alcance su valor máximo o mínimo (dependiendo de que sea incremento ascendente o descendente), automáticamente vuelva a comenzar por el valor asignado en el parámetro START WITH.

- NOCYCLE: se utiliza para especificar que cuando la secuencia alcance su valor máximo o mínimo (dependiendo de que sea incremento ascendente o descendente), se pare y no vuelva a reiniciarse. Esta es la opción por defecto cuando no se especifica ni este parámetro ni su contrario: CYCLE.

- CACHE: se utiliza para indicar cuántos valores de la secuencia se mantienen en memoria del sistema para un acceso más rápido. Como valor mínimo se le puede asociar 2. Es la opción por defecto de creación de una secuencia cuando no se indica ni este parámetro ni su contrario: NOCACHE. Si no se indica valor, por defecto almacena 20 números en memoria.

- NOCACHE: se utiliza para indicar que los valores no se mantienen en memoria. Esta opción es la que presenta los valores de la secuencia más veraces, dado que en algunas versiones de Oracle ha fallado la opción CACHE, no incrementando ni presentando correctamente el valor actual de la secuencia.

Cómo se maneja una secuencia

Para manejar la secuencia existen 2 pseudocolumnas que se pueden añadir al nombre de la misma, que son:

- CURRVAL: que devuelve el valor actual de la secuencia.
- NEXTVAL: que incrementa/decrementa (si el número es negativo) la secuencia, y devuelve el valor después de realizar esta operación.

Cuando se crea una secuencia antes de poder utilizar la pseudocolumna CURRVAL hay que ejecutar al menos una vez la pseudocolumna NEXTVAL, que inicializa la secuencia al valor correspondiente al parámetro START WITH.

EJEMPLO 1. CREACIÓN Y MANIPULACIÓN DE UNA VISTA

```
SQL> CREATE SEQUENCE cuenta1
        INCREMENT BY 10
        START WITH 1;

SQL> SET SERVEROUTPUT ON;

SQL> DECLARE
        VAR1 NUMBER;
     BEGIN
        VAR1:= CUENTA1.NEXTVAL;
        DBMS_OUTPUT.PUT_LINE('Inicializada secuencia'||
                             ' a valor '||VAR1);
        SELECT CUENTA1.NEXTVAL INTO VAR1 FROM DUAL;
        DBMS_OUTPUT.PUT_LINE('Incrementada secuencia'||
                             ' valor '||VAR1);
        DBMS_OUTPUT.PUT_LINE('Valor actual secuencia'||
                             ' es '||CUENTA1.CURRVAL);
     END;
```

En este ejemplo en primer lugar creamos una secuencia (CREATE SEQUENCE…) con valor de comienzo 1 que se irá incrementando de 10 en 10. El resto de parámetros adoptarán su valor por defecto, es decir, la secuencia no será cíclica (NOCYCLE) y almacenará 20 números en memoria (CACHE).

En segundo lugar activamos un parámetro de entorno de la sesión (SET SERVEROUTPUT ON) que permite devolver resultados a pantalla mediante el uso de DBMS_OUTPUT.PUT_LINE.

Por último, en el bloque PL/SQL definido para la manipulación de la secuencia (DECLARE…BEGIN…END), se realizan las siguientes operaciones:

- Se declara una variable VAR1 de tipo numérica (DECLARE…).
- Dentro del bloque (BEGIN…END) se realizan el resto de operaciones:

- Se inicializa la secuencia `CUENTA1` asignando el valor a la variable (`VAR1 := CUENTA1.NEXTVAL`). Por tanto, `VAR1` tendrá el valor 1. Este método de asignación de valores de una secuencia a una variable solo está permitido a partir de la versión de base de datos 11g.
- Se visualiza en pantalla el valor de la variable (`DBMS_OUTPUT.PUT_LINE('Inicializada…`).
- Se incrementa la secuencia `CUENTA1` asignando el valor a la variable (`SELECT CUENTA1.NEXTVAL INTO…`). Por tanto, `VAR1` tendrá el valor 11. Tanto la inicialización de la secuencia como el incremento se pueden realizar de la primera forma (`VAR1 := CUENTA1.NEXTVAL`) o bien de esta última forma, utilizando una consulta contra la tabla virtual `DUAL`. Este método de asignación de valores de una secuencia a una variable mediante una sentencia SELECT, es el único permitido en las versiones de base de datos Oracle 10g y anteriores.
- Se visualiza en pantalla el valor de la variable (`DBMS_OUTPUT.PUT_LINE('Incrementada…`).
- Por último, se visualiza en pantalla el valor actual de la secuencia (`CUENTA1.CURRVAL`).

SUPUESTO PRÁCTICO 7: ***Resolución en el Anexo I.***

Crear una secuencia que comience en 1 y se vaya incrementando de 1 en 1 a fin de poder utilizarla para el número de inscripción de un enfermo.

- Indicar como valor mínimo: 1.
- Indicar como valor máximo: 99.999.999.
- Indicar que no sea cíclica.
- Indicar que no se almacenen valores en la caché.

SUPUESTO PRÁCTICO 8: ***Resolución en el Anexo I.***

Supuesto repaso de instrucciones del sublenguaje DDL.

1. Crear la tabla `ejemplo1` que se indica a continuación utilizando instrucciones SQL/DDL.

Tabla EJEMPLO1

EMP_NO: Número de empleado (4 dígitos numéricos). Índice Primario.
APELLIDO: Apellido (16 caracteres).
FECHA_ALT: Fecha de alta en la empresa (por defecto la del día).
SALARIO: Salario mensual (9 dígitos numéricos).

2. Alterar la tabla `ejemplo1` que se ha creado con anterioridad para que se recojan estos nuevos campos:

   COD_DOMI: Codificación del domicilio (3 caracteres). Únicamente admite los valores siguientes: 'CL', 'AVD', 'RD', 'PSO'.
   SEGSOCIAL: Nº de la Seguridad Social (9 dígitos numéricos). No admite valores nulos y por defecto tiene que tener el valor 0.

3. Crear la tabla `ejemplo2` según se indica a continuación utilizando instrucciones SQL/DDL:

   Tabla EJEMPLO2

   EMP_NO: Número de empleado (4 dígitos numéricos). Índice Primario.

4. Borrar la tabla `ejemplo2.`

5. Crear un índice para la tabla `ejemplo1` que englobe al campo `APELLIDO` y que se llame `iapellido_ejemplo1.`

6. Crear un índice único en la tabla `ejemplo1` que englobe al campo `SEGSOCIAL` y que se llame `iss_ejemplo1.`

7. Alterar la tabla `ejemplo1` para añadir una restricción (constraint) que impida la repetición de los valores del campo `SEGSOCIAL.`

8. Crear una vista llamada `sal_empleado` sobre la tabla `ejemplo1` que realice la siguiente consulta:

```
SELECT EMP_NO, SALARIO FROM EJEMPLO1;
```

9. Borrar la vista anterior.

NEW 12C COLUMNAS DE IDENTIDAD

Con la nueva versión 12c de Oracle, se crea un nuevo tipo de columnas denominadas "columnas de identidad" (identity column). Este tipo de columnas generan un valor por sí mismas, de forma que ya no sería necesario crear una secuencia para manejar valores de una columna y luego asociar los valores de la secuencia a la misma.

En las columnas de identidad, Oracle crea implícitamente una secuencia por defecto. Por cada operación de inserción que se realice en la tabla donde se encuentra la columna de identidad, se incrementa el valor de dicha columna automáticamente a partir de la secuencia que está manejando Oracle.

La sintaxis textual en modo comando SQL sería la siguiente:

```
CREATE TABLE nombre
(nom_columna    NUMBER
                GENERATED BY DEFAULT AS IDENTITY
                (START WITH integer
                 INCREMENT BY integer),
….)
```

A continuación, se define cada uno de los parámetros disponibles para la creación de una columna de identidad:

- `Nom_columna`: nombre que se le asigna a la columna de identidad. El tipo tendrá que ser NUMBER o INTEGER
- `INCREMENT BY`: especifica el crecimiento que va a tener la secuencia en formato entero. Si el valor es positivo, la secuencia ascenderá según el incremento indicado, y si el valor es negativo, decrecerá. Si se omite, el valor por defecto es 1.
- `START WITH:` especifica el valor de comienzo de la secuencia en formato entero.

EJEMPLO. CREACIÓN DE UNA TABLA CON COLUMNA DE IDENTIDAD. INSERCIÓN Y CONSULTA

```
SQL> CREATE TABLE prueba
         (COL1   NUMBER
                 GENERATED BY DEFAULT AS IDENTITY
                 (START WITH 1
                 INCREMENT BY 1)
                 PRIMARY KEY,
         COL2   VARCHAR2(10)
     );

SQL> INSERT INTO prueba (col2) values ('PRUEBA 1');
     INSERT INTO prueba (col2) values ('PRUEBA 1');
     INSERT INTO prueba values (default, 'PRUEBA 3');
     COMMIT;

SQL> SELECT * FROM prueba;

          COL1 COL2
   ---------- ----------
             1 PRUEBA 1
             2 PRUEBA 2
             3 PRUEBA 3
```

NEW 12C COLUMNAS INVISIBLES

Con la nueva versión 12c de Oracle, se permite la creación de columnas invisibles que no participan de las operaciones de la tabla.

Una columna invisible no se muestra en la definición de la tabla ni en las consultas SELECT * FROM TABLA, que se realicen sobre la misma. Además, una columna invisible no participa en las transacciones de la tabla.

Para seleccionar o insertar un valor en una columna invisible, se debe especificar su nombre explícitamente.

Para hacer aparecer todas las columnas invisibles en las descripciones de la estructura de la tablas hay que utilizar la siguiente instrucción:

```
SET COLINVISIBLE ON
```

La sintaxis textual en modo comando SQL sería la siguiente:

```
CREATE TABLE nombre
(nom_columna    tipo_dato
                INVISIBLE),
….)
```

EJEMPLO. CREACIÓN DE UNA TABLA CON COLUMNA INVISIBLE. DESCRIPCIÓN Y CONSULTA

```
SQL> CREATE TABLE prueba
        (COL1   NUMBER
                INVISIBLE
                PRIMARY KEY,
        COL2   VARCHAR2(10)
     );

SQL> DESC prueba;

Nombre Nulo Tipo
------ ---- ------------
COL2        VARCHAR2(10)

SQL> INSERT INTO PRUEBA(COL1,COL2) VALUES (1,'A');
     INSERT INTO PRUEBA(COL1,COL2) VALUES (2,'B');
     COMMIT;

SQL> SELECT * FROM prueba;

COL2
----------
A
B

SQL> SELECT COL1, COL2 FROM prueba;

      COL1 COL2
---------- ----------
         1 A
         2 B
```

NEW 12C MANEJANDO VALORES POR DEFECTO EN COLUMNAS

Con la nueva versión 12c de Oracle, se aumentan los métodos de asignación de valores por defecto a las columnas de una tabla. En la nueva versión de Oracle se admite la asociación de una secuencia como valor por defecto de una columna, así mismo se permita que una columna asuma un valor solo cuando la misma no reciba ningún dato (es decir, venga a NULL en un proceso de inserción de datos).

Con anterioridad a la versión 12c, para poder asociar el valor de una secuencia a una columna, era necesario programar un trigger donde se implementase la asociación del valor de la secuencia a una columna.

EJEMPLO. SECUENCIA Y VALOR POR DEFECTO DE UNA COLUMNA

```
SQL> CREATE SEQUENCE cuenta1
        INCREMENT BY 10
        START WITH 1;


SQL> CREATE TABLE prueba
   (x  NUMBER DEFAULT cuenta1.nextval
       PRIMARY KEY,
    y  NUMBER);


SQL> INSERT INTO PRUEBA (y) VALUES (333);
     INSERT INTO PRUEBA (x,y) VALUES (2,334);
     INSERT INTO PRUEBA (y) VALUES (400);
     commit;


SQL> SELECT * FROM PRUEBA;

         X          Y
---------- ----------
         1        333
         2        334
        11        400
```

EJEMPLO. DEFINIR UNA COLUMNA CON VALOR NULL POR DEFECTO

```
SQL> CREATE TABLE prueba
   (x  NUMBER
       PRIMARY KEY,
    y  NUMBER,
    Z  NUMBER DEFAULT ON NULL 13);


SQL> INSERT INTO PRUEBA (x,y,z) VALUES (1,333,NULL);
     INSERT INTO PRUEBA (x,y) VALUES (2,334);
     INSERT INTO PRUEBA VALUES (11,400,-3);
     commit;


SQL> SELECT * FROM PRUEBA;

         X          Y          Z
---------- ---------- ----------
         1        333         13
         2        334         13
        11        400         -3
```

7 INSERCIÓN DE DATOS

INTRODUCCIÓN AL SUBLENGUAJE DML

Los comandos utilizados en el sublenguaje DML de SQL son básicamente los siguientes:

Comando	Descripción
INSERT	Se utiliza para cargar información en una tabla.
SELECT	Se utiliza para consultar información de una tabla.
UPDATE	Se utiliza para actualizar información de una tabla.
DELETE	Se utiliza para borrar información de una tabla.

Las cláusulas son condiciones de modificación utilizadas para definir los datos que se desean seleccionar o manipular, y que se aplican a alguno de los comandos anteriores. A continuación, se muestran las cláusulas básicas:

Comando	Descripción
FROM	Se utiliza para especificar la/s tabla/s desde las que se va a seleccionar información.
WHERE	Se utiliza para especificar condiciones que deben cumplir los registros a los que se accede en la/s tabla/s.
GROUP BY	Se utiliza para separar los registros seleccionados en grupos específicos que cumplan las mismas condiciones.

Comando	Descripción
HAVING	Se utiliza para expresar la condición que debe de satisfacer un grupo acotado previamente con la cláusula GROUP BY.
ORDER BY	Se utiliza para ordenar los registros seleccionados de acuerdo a la/s columna/s que se indiquen.

INSERCIÓN DE INFORMACIÓN (INSERT)

En este apartado vamos a conocer los métodos para poder insertar información dentro de una tabla.

Como único comando existente en el lenguaje DML tenemos el comando INSERT que permite introducir datos en nuestras tablas de dos formas:

- Indicando directamente los valores que queremos introducir.
- Utilizando una instrucción SELECT que recupere previamente información de otra/s tabla/s.

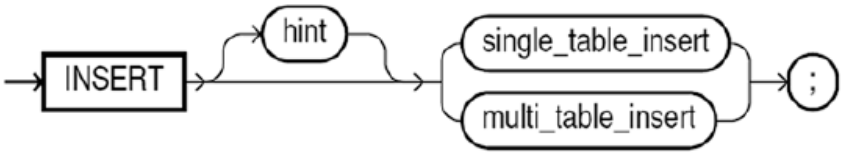


Fig. 7-1 Sintaxis para la inserción de información en una tabla.

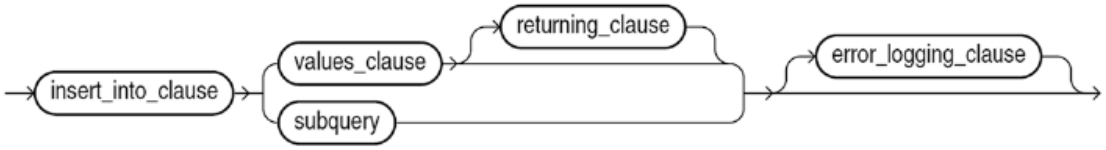


Fig. 7-2 Sintaxis del elemento SINGLE_TABLE_INSERT para insertar información.

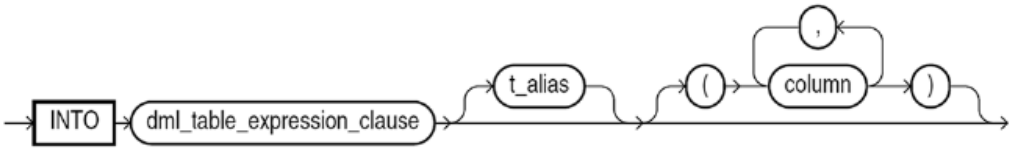


Fig. 7-3 Sintaxis del elemento INSERT_INTO_CLAUSE del elemento SINGLE_TABLE_INSERT.

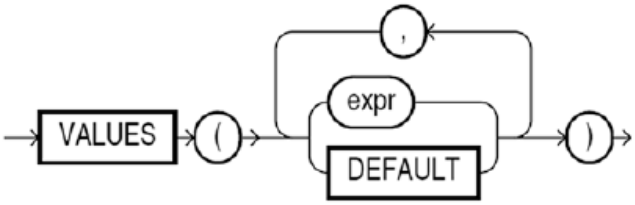


Fig. 7-4 Sintaxis del elemento VALUES_CLAUSE del elemento SINGLE_TABLE_INSERT.

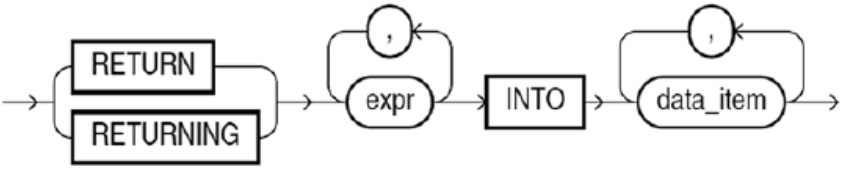


Fig. 7-5 Sintaxis del elemento RETURNING_CLAUSE del elemento SINGLE_TABLE_INSERT.

Ejemplo 1. Inserción de datos para toda la tabla

```
INSERT INTO DEPT
VALUES (50, 'PRODUCCION','SAN FRANCISCO');
```

Esta forma de insertar valores en una tabla es la más simple. Se inserta información para todas las columnas de las que se compone la tabla.

Ejemplo 2. Inserción de datos en columnas concretas

```
INSERT INTO emp (empno, ename, job, sal, comm, deptno)
VALUES (7890, 'JINKS', 'CLERK', 1.2E3, NULL, 40);
```

Este método inserta valores para las columnas específicas que aparecen después de la cláusula `INTO` y antes de `VALUES`.

Ejemplo 3. Inserción de datos en columnas concretas utilizando una consulta

```
INSERT INTO (select empno, ename, job, sal, comm,
             deptno from emp)
VALUES (7890, 'JINKS', 'CLERK', 1.2E3, NULL, 40);
```

Este método produce el mismo efecto que el caso anterior, insertando valores en columnas concretas; aquellas que se han indicado en la instrucción SELECT.

Ejemplo 4. Inserción de datos obtenidos de una consulta

```
INSERT INTO bonus
   SELECT ename, job, sal, comm
   FROM emp
   WHERE comm > 0.25 * sal
   OR job IN ('PRESIDENTE', 'DIRECTOR');
```

Este método de inserción de datos copia en la tabla BONUS, todos aquellos empleados cuyo trabajo sea PRESIDENTE O DIRECTOR o cuya comisión sea superior al 25% de su salario.

Ejemplo 5. Inserción de datos en una tabla de otro usuario y de una base de datos remota

```
INSERT INTO scott.accounts@sales (acc_no, acc_name)
VALUES (5001, 'BOWER');
```

Este método inserta valores concretos en la tabla ACCOUNTS del usuario SCOTT para la base de datos remota SALES.

Ejemplo 6. Inserción de datos devolviendo resultados

```
INSERT INTO emp VALUES (empseq.nextval, 'LEWIS',
                        'CLARK',7902, SYSDATE, 1200,
                        NULL, 20)
RETURNING salario*12, trabajo INTO :bnd1, :bnd2;
```

Este método inserta valores concretos en la tabla EMP y retorno, además de los valores (SALARIO * 12) y TRABAJO a las variables :BND1 y :BND2.

SUPUESTO PRÁCTICO 9: ***Resolución en el Anexo I.***

A) Realizar la carga de los datos en las tablas creadas para el modelo de hospital, de acuerdo a los datos especificados en el Anexo II de este manual.

B) Una vez realizada la carga de datos anteriores, hay que insertar, mediante instrucciones DML, los siguientes empleados en la plantilla no sanitaria del hospital. Las inserciones las efectuaremos asociadas al hospital con código 7.

- **DIRECTOR**
  - Nif: **12345678B**
  - Nombre: **Juan**
  - Apellidos: **Lopez Z.**
  - No depende de ningún otro empleado.
  - Entró a formar parte de la plantilla del hospital el **11 de enero de 1970** y cobra un salario de **3.000 €** sin que cobre comisiones por realizar su trabajo.
  - El director del hospital desarrolla su labor dentro del **departamento con código 4.**

- **GERENTE**
  - Nif: **87654321A**
  - Nombre: **Fermin**
  - Apellidos: **Garcia L.**
  - Esta persona depende directamente del DIRECTOR. Entró a formar del hospital el **12 de enero de 1975**, cobra un salario de **2.000 €** que se ve complementado con una comisión fija de **1.000 €**.
  - El director del hospital desarrolla su labor dentro del **departamento con código 4.**

- **ADMINISTRADOR**
  - Nif: **64328285C**
  - Nombre: **Rosa**
  - Apellidos: **Miranda R.**
  - Esta persona depende directamente del GERENTE. Entró a formar parte del hospital el **13 de enero de 1979**, cobra un salario de **1.500 €** sin recibir comisiones por su trabajo.
  - El director del hospital desarrolla su labor dentro del **departamento con código 4**.

- **CONTABLE**

    - Nif: **83253235F**
    - Nombre: **Miguel**
    - Apellidos: **Soria T.**
    - Esta persona depende directamente del ADMINISTRADOR. Entró a formar parte del hospital el **14 de enero de 1980**, cobra un salario de **1.000 €** y si realiza bien su trabajo cobra una comisión de **300 €.**
    - El director del hospital desarrolla su labor dentro del **departamento con código 4.**

CONSULTA DE DATOS 8

CONSULTA DE INFORMACIÓN (SELECT)

En este apartado conoceremos los distintos métodos para consultar datos de tablas y vistas, teniendo en cuenta que existe un único comando como en el caso de la inserción de datos, que es el comando SELECT, pero al que se le aplican una serie de cláusulas que nos permiten formular distintos tipos de consultas.

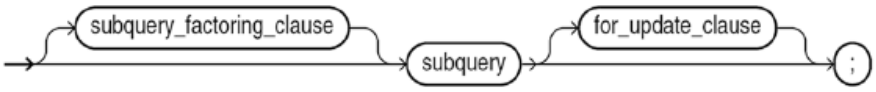


Fig. 8-1 Sintaxis para la consulta de datos de una tabla.

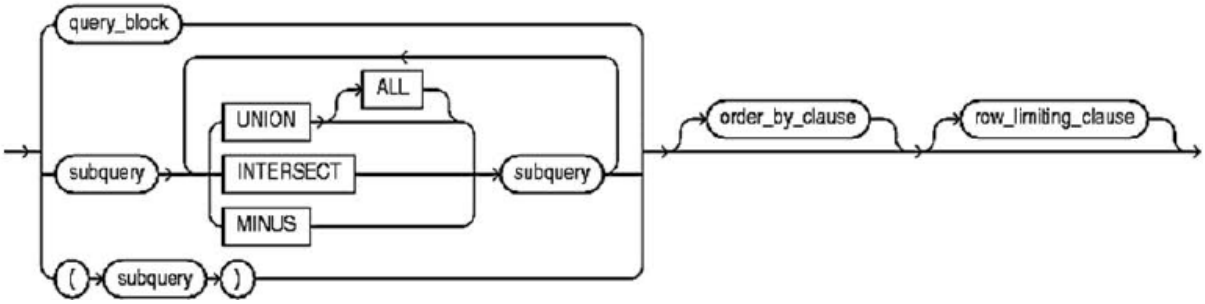


Fig. 8-2 Sintaxis del elemento SUBQUERY de la consulta de datos.

La sintaxis textual del comando de selección de datos de una/s tabla/s es la siguiente:

```
SELECT [hint] [{DISTINCT | UNIQUE } | ALL] lista_col
FROM tabla1 [,tabla2]…
[clausula_where]
[clausula_de_consulta_jerárquica]
[clausula_group_by]
[HAVING condición]
[{UNION | UNION ALL | INTERSECT | MINUS] (subconsulta)]
[clausula_order_by]
[clausulas_limitación_registros]
```

CONSULTAS BÁSICAS

La sintaxis básica de una consulta de selección es la siguiente:

```
SELECT campos FROM tabla;
```

En donde **campos** es la lista de nombres de columnas, expresiones o valores que se desean recuperar de la **tabla** o tablas.

Ejemplo

```
SELECT NOMBRE, TELEFONO FROM CLIENTES;
```

Esta consulta devuelve un conjunto de filas (RECORDSET) con la información de las columnas `nombre` y `teléfono` de la tabla `clientes`.

ORDENAR LOS REGISTROS

Opcionalmente se puede especificar el orden en que se desean recuperar los registros de las tablas mediante la cláusula `ORDER BY` *Lista de Campos*. En donde **Lista de campos** representa las columnas o expresiones a ordenar. En este caso la sintaxis de la sentencia `SELECT` sería la siguiente:

```
SELECT Campos FROM Tabla
[ORDER BY Lista de campos];
```

Ejemplo ordenando por una columna

```
SELECT CodigoPostal, Nombre, Telefono
FROM Clientes
ORDER BY Nombre
```

Esta consulta devuelve las columnas `CodigoPostal, Nombre` y `Teléfono` de la tabla `Clientes` ordenados por la columna `Nombre`.

Ejemplo ordenando por varias columnas

```
SELECT CodigoPostal, Nombre, Telefono
FROM Clientes
ORDER BY CodigoPostal, Nombre
```

Esta consulta muestra un ejemplo de ordenación de registros por más de una columna. Se ordenan los datos por el `CodigoPostal` y dentro del mismo código por el `Nombre`.

Ejemplo ordenando ascendente y descendente

```
SELECT CodigoPostal, Nombre, Telefono
FROM Clientes
ORDER BY CodigoPostal DESC , Nombre ASC
```

Las ordenaciones de valores siempre se hacen de forma ascendente por defecto (si no se especifica otra cosa). En este ejemplo se ordena la información `CodigoPostal` de forma descendente (incluyendo la cláusula DESC), y la columna `Nombre` de forma ascendente (se ha incluido la cláusula ASC que es redundante aunque si no se especifica también se ordena de esta forma).

CONSULTA CON PREDICADOS

El predicado se incluye entre la cláusula SELECT y el primer nombre del campo a recuperar.

Los posibles predicados son:

Comando	Descripción
*	Devuelve todas las columnas de una tabla.
DISTINCT	Omite las filas repetidas para la combinación de columnas que se incluyan después del predicado.

Predicado *

Por norma general no se incluye como predicado, dado que se asume por defecto al realizar una consulta y el SGBD selecciona todos los registros que cumplen las condiciones de la instrucción SQL.

No es conveniente abusar de este predicado ya que obligamos al motor de la base de datos a analizar la estructura de la tabla para averiguar las columnas que contiene. Es mucho más rápido (y mejora el rendimiento de la consulta) indicar el listado de columnas que se quiere consultar.

EJEMPLO

```
SELECT * FROM Empleados;
```

Esta consulta muestra todas las columnas de la tabla `Empleados`, y al no tener otro tipo de restricciones muestra también todas las filas de la tabla.

Predicado DISTINCT

Omite las tuplas (conjunto de filas) repetidas para la consulta indicada. La repetición se tendrá que dar en el conjunto de columnas que forman parte de la SELECT y que aparecen a la derecha del predicado.

EJEMPLO

```
SELECT DISTINCT Apellido FROM Empleados;
```

Esta consulta muestra únicamente la columna `Apellido` de todas aquellas filas de la tabla `Empleados`, mostrando en caso de repetición del mismo contenido en la columna `Apellido`, una sola fila, es decir, un solo apellido por cada repetición. Por tanto, si dos filas tienen el contenido "LOPEZ" en la columna `Apellido`, únicamente se mostrará una vez la fila con contenido "LOPEZ" de las 2 posibles.

EL CONCEPTO DE ALIAS

En determinadas circunstancias es necesario asignar un nombre “virtual” a alguna columna determinada, como encabezado del resultado de un conjunto de valores

devuelto, por simple capricho o por otras circunstancias, como la interpretación de la información más asequible por parte del cliente.

Para resolver estas circunstancias, tenemos la palabra reservada AS, que se encarga de asignar el nombre virtual que deseamos a la columna indicada.

Ejemplo de alias sobre una columna

Tomando como referencia el ejemplo anterior podemos hacer que la columna devuelta por la consulta, en lugar de llamarse apellido (igual que el campo devuelto) se llame Empleado. En este caso procederíamos de la siguiente forma:

```
SELECT DISTINCT Apellido AS Empleado FROM Empleados;
```

Asimismo, algunas veces se presenta la necesidad de asignar un alias al nombre de la tabla, para poder manejarla más cómodamente, bien porque es un nombre de tabla larga, bien porque se accede a la misma en subconsultas. En estos casos también se utiliza un alias, pero no se utiliza la cláusula AS, sino que se deja un espacio en blanco entre el nombre verdadero de la tabla y el alias.

Ejemplo de alias sobre una tabla

En el ejemplo que se muestra a continuación se puede observar que a la tabla empleados se le ha asignado el alias emp:

```
SELECT emp.Apellido FROM Empleados emp;
```

CRITERIOS DE SELECCIÓN

Anteriormente hemos visto la forma de recuperar los registros de las tablas. En todas ellas se devolvían todas las filas de la mencionada tabla (salvo en el caso del DISTINCT).

A lo largo de este apartado se estudiarán las posibilidades de filtrar las filas que devuelve la consulta, con el fin de recuperar solamente aquellas que cumplan una condición preestablecida.

La cláusula WHERE

La cláusula WHERE puede usarse para determinar qué filas de la/s tabla/s enumeradas en la cláusula FROM aparecerán en los resultados de la instrucción SELECT. Después de escribir esta cláusula se deben especificar las condiciones o criterios de selección.

Si no se emplea esta cláusula, la consulta devolverá todas las filas de la tabla. Por tanto, la cláusula WHERE es opcional, pero cuando aparece debe ir a continuación de la cláusula FROM, por lo que la sintaxis de la cláusula SELECT quedaría de la siguiente forma:

```
SELECT Campos FROM Tabla
WHERE Condiciones de filtro
[ORDER BY Lista de campos];
```

EJEMPLO

```
SELECT Apellidos, Salario
FROM Empleados
WHERE Salario > 210000;
```

Esta consulta nos devuelve los datos de los apellidos y salarios de los empleados cuyo salario sea mayor a 210.000 euros.

Uso de operadores lógicos

Los operadores lógicos soportados por SQL son los siguientes:

- AND
- OR
- NOT

A excepción de los dos últimos, todos poseen la siguiente sintaxis:

<expresión1> operador <expresión2>

En donde expresión1 y expresión2 son las condiciones a evaluar.

Si a cualquiera de las anteriores condiciones le anteponemos el operador NOT, el resultado de la operación será el contrario al devuelto sin el operador NOT.

EJEMPLO

```
SELECT * FROM Empleados
WHERE Edad > 25
AND Edad < 50;
```

Esta consulta nos devuelve todas las columnas de la tabla Empleados, para aquellos empleados que tengan una edad comprendida entre 26 y 49 años (ambos inclusive).

EJEMPLO

```
SELECT * FROM Empleados
WHERE (Edad > 25 AND Edad < 50)
OR Sueldo = 1000;
```

Esta consulta nos devuelve todas las filas de la tabla Empleados, para aquellos empleados que tengan una edad comprendida entre 26 y 49 años (ambos inclusive), o bien aquellos empleados cuyo sueldo sea mayor a 1.000 €.

EJEMPLO

```
SELECT * FROM Empleados
WHERE NOT Estado = 'Soltero';
```

Esta consulta nos devuelve todas las filas de la tabla Empleados, para aquellos empleados cuyo estado social sea distinto a Soltero.

EJEMPLO

```
SELECT * FROM Empleados
WHERE (Sueldo > 1000 AND Sueldo < 5000)
OR (Provincia = 'Madrid' AND Estado = 'Casado');
```

Esta consulta nos devuelve todos los datos de la tabla Empleados, para aquellos empleados cuyo sueldo se encuentre comprendido entre 1.001 € y 4.999 € (ambos inclusive), o bien aquellos empleados cuya provincia sea “Madrid” y además estén casados.

Uso de intervalos de valores

Para indicar que deseamos recuperar las filas de una tabla según el intervalo de valores de una columna emplearemos preferentemente el operador BETWEEN en vez de utilizar una combinación de operadores lógicos como hemos visto anteriormente.

La sintaxis del mismo es:

columna [NOT] BETWEEN valor1 AND valor2

En este caso de aplicar este filtro de selección dentro de la cláusula WHERE, la consulta devolvería las filas que contengan en "columna" un valor incluido en el intervalo "valor1" y "valor2" (ambos inclusive). Si anteponemos la condición NOT (que es opcional), devolverá aquellos valores no incluidos en el intervalo.

EJEMPLO

```
SELECT * FROM Pedidos
WHERE CodPostal Between 28000 And 28999;
```

Esta consulta devuelve todas las filas de la tabla Pedidos para aquellos códigos postales entre 28000 y 28999 (ambos inclusive).

Uso del operador LIKE

Se utiliza para comparar una expresión de cadena con un modelo dentro de una expresión SQL que aparece tras la cláusula WHERE.

Su sintaxis es: expresión LIKE modelo

En donde “expresión” es una cadena en la que se busca el “modelo”. Se puede utilizar el operador LIKE para encontrar valores en las columnas de la tabla que coincidan con el modelo especificado.

En la parte del “modelo” se puede especificar un valor completo (como por ejemplo 'Ana María'), o se pueden utilizar caracteres comodines para encontrar un rango de valores. Por ejemplo, si introduce LIKE 'C%' en una consulta SQL, la consulta devuelve todos los valores dentro de las columnas seleccionadas que comiencen por la letra 'C'.

Por tanto, el símbolo '%' es uno de los comodines que se pueden utilizar para indicarle a una consulta que utilice el operador LIKE, que busque cualquier carácter en el lugar que se incluya este símbolo.

EJEMPLO

```
SELECT * FROM vuelos
WHERE COMPAÑIA LIKE 'A%'
```

En este ejemplo devolvemos todas las filas de la tabla `vuelos` para las compañías que empiecen por 'A', como por ejemplo: AVIACO, AIR EUROPA, etc.

EJEMPLO

```
SELECT * FROM vuelos
WHERE COMPAÑIA LIKE 'A _ _ _ _ _'
```

En este ejemplo devolvemos todas las filas de la tabla `vuelos` para las compañías que empiecen por 'A', y que además tengan obligatoriamente 5 letras, como por ejemplo: AVIACO.

EJEMPLO

```
SELECT * FROM vuelos
WHERE COMPAÑIA LIKE '%A%'
```

En este ejemplo devolvemos todas las filas de la tabla `vuelos` para las compañías que contengan una 'A' en cualquier posición, como por ejemplo: AVIACO, SPAIN AIR, OSAKA AIRLINES, etc.

Uso del operador IN

Este operador devuelve aquellas filas cuya columna indicada coincide con alguno de los valores de una lista.

Su sintaxis es: expresión `[NOT] IN(valor1, valor2,…)`

EJEMPLO

```
SELECT * FROM Pedidos
WHERE Provincia In ('Madrid', 'Barcelona', 'Sevilla');
```

Esta consulta nos devuelve todos los campos de la tabla `Pedidos` para las provincias: “Madrid”, “Barcelona” o “Sevilla”.

Uso de operadores de comparación

Los operadores de comparación que se pueden utilizar en consultas SELECT son los siguientes:

```
>, <, >=, <=, =, <>, !=
```

EJEMPLO

```
SELECT * FROM Pedidos
WHERE Importe >= 1200.35;
```

Esta consulta nos devuelve todas las filas de la tabla `Pedidos`, para aquellos pedidos cuyo importe sea superior o igual a 1.200,35 €.

Uso del operador de concatenación

El operador de concatenación que nos permite unir 2 cadenas dentro de una consulta SELECT es el siguiente: `||`

EJEMPLO

```
SELECT 'Antolin'||' Muñoz' FROM DUAL;
```

Esta consulta nos devuelve la expresión 'Antolin Muñoz' después de concatenar las expresiones: 'Antolin' y 'Muñoz'.

FUNCIONES DE CONVERSIÓN

Las funciones que se explican en este apartado permiten realizar la mayoría de las conversiones de datos necesarias para representar la información que se obtiene de las tablas, o bien para poder realizar filtros sobre las mismas.

En SQL se utilizan 3 tipos de funciones genéricas para la conversión de tipos:

- TO_NUMBER: Conversiones a formato numérico.
- TO_CHAR: Conversiones a formato de texto.
- TO_DATE: Conversiones a formato de fecha y hora.

Función TO_NUMBER

Esta función permite realizar la conversión de expresiones a formato numérico.

La sintaxis de la función es: `TO_NUMBER(expresión[,formato])`

Para que se pueda obtener un resultado tras esta conversión, el contenido de la expresión deberá ser numérico.

EJEMPLO

```
SELECT TO_NUMBER('10') FROM DUAL;
```

Esta consulta nos devuelve convertido a número la cadena '10'.

Función TO_CHAR

Esta función permite realizar la conversión de expresiones a formato texto.

La sintaxis de la función es: `TO_CHAR(expresión[,formato])`

EJEMPLO

```
SELECT TO_CHAR(10) FROM DUAL;
```

Esta consulta nos devuelve, convertido a una cadena de texto, el valor numérico 10.

Función TO_DATE

Esta función permite realizar la conversión de expresiones a formato fecha y/o hora.

La sintaxis de la función es: `TO_DATE(expresión[,formato])`

EJEMPLO

```
SELECT TO_DATE('10-12-08','DD-MM-YY') FROM DUAL;
```

Esta consulta nos devuelve la conversión en fecha de la cadena '10-12-08', utilizando como formato de interpretación de la cadena: 'dd-mm-yy': 2 dígitos para el día, 'guion', 2 dígitos para el mes, 'guion' y 2 dígitos para el año.

MODIFICADORES PARA LAS FUNCIONES DE CONVERSIÓN

Para el manejo de las funciones de conversión, existen una serie de modificadores de formato que especifican la estructura de conversión de la expresión.

Modificadores para la conversión a/desde formato numérico

Los modificadores que se pueden utilizar en el apartado "formato" de las funciones de conversión a/desde formato numérico son los siguientes:

- 9 : para indicar un dígito. Si no aparece, se sustituye por un espacio.
- 0 : para indicar un dígito. Si no aparece, se sustituye por un cero.
- D : para indicar el separador decimal.
- G : para indicar el separador de miles.

EJEMPLO

```
SELECT TO_CHAR(10000,'99G990') FROM DUAL;
```

Esta consulta nos devuelve ' 10.000' después de convertir el número 10000 con formato.

Hay que tener presente que la cadena devuelta tiene un espacio delante del 1, porque se reserva esta posición para el signo: + o -. El signo positivo no aparece porque se sobrentiende, pero igualmente el espacio en blanco queda reservado.

EJEMPLO

```
SELECT TO_CHAR(10000.9,'99G990D00') FROM DUAL;
```

Esta consulta nos devuelve ' 10.000,90' después de convertir el número 10000.9 con formato. Al igual que en el caso anterior, la cadena devuelta tiene un espacio en blanco delante del 1.

EJEMPLO

```
SELECT TO_NUMBER('1.001','9G999') FROM DUAL;
```

Esta consulta nos devuelve 1001 después de convertir la cadena '1.001' con formato.

EJEMPLO

```
SELECT TO_NUMBER('-1.001,33','9G999D99') FROM DUAL;
```

Esta consulta nos devuelve -1001.33 después de convertir la cadena '-1.001,33' con formato.

Modificadores para la conversión a/desde formato fecha/hora

Los modificadores que se pueden utilizar en el apartado "formato" de las funciones de conversión a/desde formato fecha/hora son los siguientes:

- AM: para indicar si la hora se encuentra antes (AM) o después de las 12 (PM) del mediodía.
- BC: para indicar si la fecha corresponde a un período anterior (AC) o posterior (BC) al nacimiento de Cristo.
- d: para indicar el día de la semana (1 – 7).
- dd: para indicar el día del mes (1-31).
- ddd: para indicar el día del año (1-366).
- DAY: para indicar el nombre del día (lunes – domingo).
- dy: para indicar el nombre del día de forma abreviada.
- hh: para indicar la hora en formato de 2 posiciones (1-12).
- hh12: para indicar la hora en formato 12 hs.
- hh24: para indicar la hora en formato 24 hs.
- iw: para indicar la semana dentro del año.
- mi: para indicar los minutos (0 – 59).
- mm: para indicar el mes (1 – 12).
- MON: para indicar el nombre del mes en formato abreviado (ENE-DIC).
- MONTH: para indicar el nombre del mes (ENERO-DICIEMBRE).
- q: para indicar el número correspondiente al cuatrimestre (1,2,3,4).
- rm: para indicar el mes en números romanos.
- ss : para indicar los segundos (0 – 59).
- SCC: para indicar el siglo.
- ww: para obtener la semana del año.
- w: para obtener la semana del mes.
- y: para indicar una posición para el año (admite desde y hasta yyyy).
- year: para obtener la fecha traducida a texto en letras.

EJEMPLO

```
SELECT TO_DATE('01/01/2008','dd/mm/yyyy') FROM DUAL;
```

Esta consulta nos devuelve la fecha 01/01/08 después de convertir la cadena '01/01/2008' con formato 'dd/mm/yyyy': 2 dígitos para el año, '/', 2 dígitos para el mes, '/' y 4 dígitos para el año.

EJEMPLO

```
SELECT TO_DATE('01-ENE-08 22:13:00',
               'dd-MON-yy hh24:mi:ss') FROM DUAL;
```

Esta consulta nos devuelve la fecha 01/01/08, después de convertir la cadena '01-ENE-08 22:13:00' con formato 'dd-MON-yy hh24:mi:ss.

EJEMPLO

```
SELECT TO_CHAR(SYSDATE,'DAY dd-MONTH-yyyy') FROM DUAL;
```

Esta consulta nos devuelve la cadena ' VIERNES 21-AGOSTO -2009', después de convertir la fecha del día del sistema (SYSDATE) con formato.

FUNCIONES DE CARACTERES

En este apartado se definen una serie de funciones representativas para el uso de cadenas de caracteres.

Función	Descripción
ASCII	Código ASCII de un carácter.
ASCIISTR	Convierte en ASCII una cadena de caracteres.
CHARTOROWID	Convierte una cadena en un tipo Rowid.
CHR	Carácter correspondiente a un código ASCII.
COALESCE	Primera cadena no nula.
COMPOSE	Compone cadenas de juegos distintos en una sola.
CONCAT	Concatena cadenas.
CONVERT	Convierte una cadena de un juego de caracteres a otro.
DECODE	Decodificación de valores.
GREATEST	La mayor de las cadenas.
INITCAP	Cadena de texto en formato título.
INSTR	Búsqueda de una cadena dentro de otra.
LEAST	La menor de las cadenas.
LENGTH	Longitud de una cadena.
LNNVL	Compara una condición que puede tener NULL a ambos lados.
LOWER	Cadena de texto en minúsculas.
LPAD	Ajuste de cadenas a la izquierda.
LTRIM	Elimina blancos por la izquierda.

Función	Descripción
NULLIF	Comparador de cadenas buscando nulos.
NVL2	Comparación de una cadena para ver si es NULL o NOT NULL.
REPLACE	Reemplazo de caracteres.
RPAD	Ajuste de cadenas a la derecha.
RTRIM	Elimina blancos por la derecha.
SUBSTR	Corta una cadena.
TRANSLATE	Sustituye caracteres múltiples de una cadena.
TRIM	Elimina blancos a la derecha e izquierda de la cadena.
UPPER	Cadena de texto en mayúsculas.

Función ASCII(cadena)

Devuelve el valor en número del código ASCII de una cadena.

EJEMPLO

```
SELECT ASCII('%') FROM DUAL;
```

Esta función nos devuelve el símbolo número 37.

Función ASCIISTR(cadena)

Toma como argumento una cadena o una expresión y devuelve una versión ASCII de la cadena.

EJEMPLO

```
SELECT ASCIISTR('ABÄCDE') FROM DUAL;
```

Esta función nos devuelve la cadena ' AB\017DCDE'.

Función CHARTOROWID(X)

Convierte una cadena en un tipo ROWID.

EJEMPLO

```
SELECT APELLIDO2 FROM EMPLEADO
WHERE ROWID = CHARTOROWID('AAAFd1AAFAAAABSAA/');
```

Esta función nos devuelve al apellido2 del empleado cuya fila se encuentra almacenada físicamente en la dirección (ROWID) `'AAAFd1AAFAAAABSAA/'`.

Función CHR(X)

Devuelve el carácter ASCII correspondiente al valor de X donde X es un número.

EJEMPLO

```
SELECT CHR(37) FROM DUAL;
```

Esta función nos devuelve el símbolo %.

Función COALESCE(cadena[,cadena…]

Esta función devuelve la primera cadena de la lista indicada en la misma, que no sea nula.

Al menos una de las cadenas incluidas en la función tiene que ser un literal diferente a NULL.

EJEMPLO

```
SELECT COALESCE('ANTOLIN',NULL,'MUÑOZ') FROM DUAL;
```

Esta función nos devuelve la cadena 'ANTOLIN'.

Función COMPOSE(cadena)

Toma como argumento una cadena o una expresión que se compone en una única cadena.

EJEMPLO

```
SELECT COMPOSE ('o' || UNISTR('\0308') ) FROM DUAL;
```

Esta función nos devuelve la cadena 'ö'.

Función CONCAT(cadena1, cadena2)

Concatena 2 cadenas, es decir, las une de la misma manera que el símbolo ||.

EJEMPLO

```
select CONCAT ('Antolin',' Muñoz') FROM DUAL;
```

Esta función nos devuelve la cadena 'Antolin Muñoz'.

Función CONVERT(cadena,juego_destino[,juego fuente])

Convierte una cadena de un juego de caracteres a otro.

EJEMPLO

```
SELECT CONVERT('ANTOLÍN MUÑOZ',
               'US7ASCII','WE8MSWIN1252') FROM DUAL;
```

Esta función nos devuelve la cadena 'ANTOL??N MUAYOZ'.

Función DECODE(expresión, comparacion1, valor1, comparacion2, valor2, ..., predeterminado)

Codifica la expresión a un valor concreto o si no al predeterminado, dependiendo de que se cumpla cada supuesto de comparación.

EJEMPLO

```
 SELECT DECODE(cargo,'Administrativo',1
                    ,'Autonomo',2
                    ,3) FROM empleados;
```

Esta función devolverá 1 si el cargo de la tabla `empleados` equivale a 'Administrativo'. Devolverá 2 si equivale a 'Autonomo' y en caso de no equivaler a ninguna de estas dos cadenas, devolverá 3.

Función GREATEST(expresion1,expresion2,….)

Devuelve el mayor de los valores de la lista de expresiones. Oracle usa el tipo de las expresiones para determinar qué tipo de valor tiene que devolver.

EJEMPLO

```
SELECT GREATEST('HARRY', 'HARRIOT', 'HAROLD')
FROM DUAL;
```

Esta función nos devuelve la cadena 'HARRY'.

Función INITCAP(cadena)

Devuelve la misma cadena pero con la primera letra en mayúscula y el resto en minúsculas.

EJEMPLO

```
SELECT INITCAP('ANTOLIN') FROM DUAL;
```

Esta función nos devuelve la cadena 'Antolin'.

Función INSTR(cadena1, cadena2 [posición, [,aparición]])

Devuelve la posición de la cadena1 donde está contenida la cadena2. La cadena se explora desde la izquierda comenzando con el valor "posición".

Si "posición" es negativo entonces se comienza desde la derecha. "Aparición" indica qué número de aparición queremos reflejar (la primera aparición, la segunda aparición, etc.).

EJEMPLO

```
SELECT INSTR('Hola caracola','la',1,2) FROM dual;
```

Esta función nos devuelve el número 12 correspondiente a la posición donde se encuentra la 2ª aparición de la cadena "la" comenzando por la posición 1 de la cadena "Hola caracola".

Función LEAST(expresion1,expresion2,….)

Devuelve el menor de los valores de la lista de expresiones. Oracle usa el tipo de las expresiones para determinar qué tipo de valor tiene que devolver.

EJEMPLO

```
SELECT LEAST('HARRY', 'HARRIOT', 'HAROLD')
FROM DUAL;
```

Esta función nos devuelve la cadena 'HAROLD'.

Función LENGTH(cadena)

Nos devuelve la longitud de la cadena.

EJEMPLO

```
SELECT LENGTH('Antolin') FROM DUAL;
```

Esta función nos devuelve el número 7.

Función LNNVL(condición)

Proporciona un método de evaluación de una condición cuando uno o ambos operandos de la comparación de la condición pueden ser nulos.

EJEMPLO

```
SELECT COUNT(*) FROM estudiantes
WHERE nota > mediacurso;
```

Esta función nos devuelve el conteo de los estudiantes cuya nota es mayor que la media del curso, pero si por ejemplo la media del curso aún no ha sido calculada y tiene un valor null, esta consulta no devolverá nada. Si por el contrario tenemos un valor para la media del curso y algún estudiante aún no se le ha calculado su nota, estos últimos no se incluirán en el conteo final.

```
SELECT COUNT(*) FROM estudiantes
WHERE LNNVL(nota > mediacurso);
```

Esta función nos devuelve el mismo conteo que la SELECT anterior, pero también incluye en el conteo los estudiantes cuya nota esté vacía.

Función LOWER(cadena)

Devuelve toda la cadena en minúsculas.

EJEMPLO

```
SELECT LOWER('ANTOLIN') FROM DUAL;
```

Esta función nos devuelve la cadena 'antolin'.

Función LPAD(cadena1,x,[cadena2])

Rellena por la izquierda la cadena 1 con los caracteres de la cadena 2, o si no se indica, con blancos.

EJEMPLO

```
SELECT LPAD('Short String',15) as prueba from dual;
```

Esta función nos devuelve el siguiente resultado:

```
Prueba
--------------
   Short String
```

Función LTRIM(cadena)

Elimina todos los espacios en blanco que haya a la izquierda del primer carácter de la cadena, distinto de blanco.

EJEMPLO

```
SELECT LTRIM(TO_CHAR(1000,'9G999')) FROM DUAL;
```

Esta función nos devuelve la cadena '1.000', eliminando el espacio que se reserva para el signo a la izquierda del número 1.

Función NULLIF(expresion1,expresion2)

Esta función compara la expresion1 con la expresion2. Si ambas son iguales devuelve NULL, en caso contrario devuelve la expresion1.

La expresion1 no puede ser el literal NULL.

EJEMPLO

```
SELECT NULLIF('Antolin',NULL) FROM DUAL;
```

Esta función nos devuelve la cadena 'Antolin'.

Función NVL2(cadena1, cadena2, cadena3)

Determina el valor a devolver consultando el contenido de "cadena1". Si no está vacío, devolverá "cadena2", en caso contrario, devolverá "cadena3".

EJEMPLO

```
SELECT NVL2(comision,salario+comision,salario)
FROM EMPLEADO;
```

Si el empleado no tiene comisión (IS NULL), la consulta devolverá solo el salario. En caso contrario, devolverá la suma del salario + la comisión.

Función REPLACE(cadena1,cadena2[,cadena3])

Reemplaza de la "cadena1" aquellos caracteres que se encuentren en la "cadena2" de la siguiente forma:

- Si se indica "cadena3", los caracteres indicados en cadena2 son sustituidos dentro de la cadena1, por los que se indiquen en la cadena2.
- Si no se indica "cadena3", los caracteres indicados en cadena2 son eliminados de la cadena1.

EJEMPLO

```
SELECT REPLACE('ANTOLIN','NT') FROM DUAL;
```

Esta función nos devuelve la cadena 'AOLIN'.

EJEMPLO

```
SELECT REPLACE('ANTOLIN','LIN','NIO') FROM DUAL;
```

Esta función nos devuelve la cadena 'ANTONIO'.

Función RPAD(cadena1,x,[cadena2])

Rellena por la derecha la cadena 1 con los caracteres de la cadena 2, o si no se indica con blancos.

EJEMPLO

```
select RPAD('Short String',15) as prueba from dual;
```

Esta función nos devuelve la cadena '`Short String   `'.

Función RTRIM(cadena)

Elimina todos los espacios en blanco que haya a la derecha del último carácter de la cadena distinto de blanco.

EJEMPLO

```
select RTRIM('Pepito   ') FROM DUAL;
```

Esta función nos devuelve la cadena 'Pepito' eliminando los 3 espacios en blanco que había a la derecha de la letra o.

Función SUBSTR(cadena, posición[,caracteres])

Corta de la cadena una longitud concreta de caracteres, comenzando por la posición que se indique en "posición". Si no se indica "caracteres", la función cortará desde "posición" hasta el final de la cadena, en cambio si se indica "caracteres", cortará únicamente el número de caracteres especificados.

EJEMPLO

```
SELECT SUBSTR('abc123def',5,4) prueba FROM dual;
```

Esta función nos devuelve la cadena '23d3e' después de extraer de la cadena 'abc123def' 4 caracteres comenzando en la posición 5ª de la cadena.

EJEMPLO

```
SELECT SUBSTR('abc123def',5) prueba FROM dual;
```

Esta función nos devuelve la cadena '3def' después de extraer todos los caracteres que había en la misma desde la posición 6 (inclusive).

Función TRANSLATE(expresión,cadena1,cadena2)

Devuelve "expresión" cuando todas las ocurrencias de cada carácter de "cadena1" es reemplazado por su correspondiente carácter en la "cadena2". Los caracteres de "expresión" que no están en la "cadena1" no se reemplazan.

EJEMPLO

```
SELECT TRANSLATE('Fichero.sql,19 12/2011',
                 '., /','____') FROM DUAL;
```

Esta función nos devuelve la cadena 'Fichero_sql_19_12_2011'.

Función TRIM(cadena)

Elimina todos los espacios en blanco que haya a la derecha del último carácter de la cadena distinto de blanco y delante del primer carácter de la cadena distinto de blanco.

EJEMPLO

```
select TRIM('   Pepito   ') FROM DUAL;
```

Esta función nos devuelve la cadena 'Pepito' eliminando los 3 espacios en blanco que había a la derecha de la letra 'o' y a la izquierda de la letra 'P'.

Función UPPER(cadena)

Devuelve la cadena en mayúsculas.

EJEMPLO

```
SELECT UPPER('antolin') FROM DUAL;
```

Esta función nos devuelve la cadena 'ANTOLIN'.

FUNCIONES DE NÚMERO

En este apartado se definen una serie de funciones representativas para el uso de los números.

Función	Descripción
ABS	Valor absoluto.
ACOS	Arco coseno.
ASIN	Arco seno.
ATAN	Arco tangente de 1 número.
ATAN2	Arco tangente de 2 números.
BIN_TO_NUM	Convierte un código binario en un número.
CEIL	Número entero más pequeño, >= a otro.
COS	Coseno.
SIN	Seno.

Función	Descripción
TAN	Tangente.
COSH	Coseno hiperbólico.
SINH	Seno hiperbólico.
TANH	Tangente hiperbólica.
EXP	Exponente neperiano.
FLOOR	Número entero más grande, <= a otro.
LN	Logaritmo natural / neperiano.
LOG	Logaritmo de un número con base.
MOD	El resto de una división.
NANVL	Para devolver un número en lugar de otro que no lo sea.
POWER	Potencia de un número.
ROUND	Redondeo de números.
SIGN	Devuelve el signo de un número.
SQRT	Raíz cuadrada de un número.
TRUNC(Number)	Truncado de números.
VSIZE	Devuelve el nº de bytes que ocupa una expresión.

Función ABS(x)

Devuelve el valor absoluto de un número.

EJEMPLO

```
SELECT ABS(-15.2) FROM dual;
```

Esta función nos devuelve el número 15.2.

Funciones ACOS(x), ASIN(x), ATAN(x)

Nos devuelven el arco coseno, seno y tangente, respectivamente, de un número.

EJEMPLO

```
SELECT ACOS(.3) FROM dual;
```

Esta consulta nos devuelve el número 1.2610367.

Funciones ATAN2(n1,n2), ATAN2(n1/n2)

Nos devuelven el arco tangente de n1 y n2.

EJEMPLO

```
SELECT ATAN2(.3,.2) FROM dual;
```

Esta consulta nos devuelve el número .982793723.

Función BIN_TO_NUM(bit1,bit2,…)

Convierte un conjunto de códigos bit en un número.

EJEMPLO

```
SELECT BIN_TO_NUM(1,0,1,0) FROM dual;
```

Esta consulta nos devuelve el número 10.

Función CEIL(x)

Nos devuelve el número entero más pequeño que sea mayor o igual a X.

EJEMPLO

```
SELECT CEIL(268651.8) FROM dual;
```

Esta consulta nos devuelve el número 268652.

Funciones COS(x), SIN(x), TAN(x)

Nos devuelven el coseno, seno y tangente, respectivamente, de un número.

EJEMPLO

```
SELECT COS(180) FROM dual;
```

Esta consulta nos devuelve el número -,59846007.

Funciones COSH(x), SINH(x), TANH(x)

Nos devuelven el coseno, seno y tangente hiperbólicas, respectivamente, de un número.

EJEMPLO

```
SELECT COSH(180) FROM dual;
```

Esta consulta nos devuelve el número 7,4469E+77.

Función EXP(n)

Nos devuelve el número e (constante de neper) elevado a la potencia n, donde e=2.71828183….

EJEMPLO

```
SELECT EXP(4) FROM DUAL;
```

Esta consulta nos devuelve el número 54.59815.

Función FLOOR(x)

Nos devuelve el número entero más grande que sea igual o menor a X.

EJEMPLO

```
SELECT FLOOR(268651.8) FROM dual;
```

Esta consulta nos devuelve el número 268651.

Función LN(x)

Nos devuelve el logaritmo natural o neperiano de un número X.

EJEMPLO

```
SELECT LN(95) FROM DUAL;
```

Esta consulta nos devuelve el número 4.55387689.

Función LOG(n,x)

Nos devuelve el logaritmo de un número X en base n. La base n puede ser cualquier valor positivo distinto a 0 o 1.

EJEMPLO

```
SELECT LOG (10,100) FROM DUAL;
```

Esta consulta nos devuelve el número 2.

Función MOD(x,y)

Nos devuelve el resto de dividir x por y.

EJEMPLO

```
SELECT MOD(7,2) FROM dual;
```

Esta consulta nos devuelve el resto 1.

Función NANVL(n2,n1)

Se utiliza únicamente con números de coma flotante de tipo BINARY_FLOAT o BINARY_DOUBLE. Devuelve el valor alternativo "n1" si el valor de "n2" no es un número (NaN). Si "n2" es un número, entonces devolverá "n2".

EJEMPLO

Si tomamos como referencia la siguiente tabla (FLOAT_POINT) con los valores que se indican:

```
DEC_NUM  BIN_DOUBLE  BIN_FLOAT
-------  ----------  ---------
1234.56  1.235e+003 1.235E+003
      0         Nan        Nan

SELECT bin_float, NANVL(bin_float,0) FROM float_point;
```

Esta consulta nos devolvería los siguientes resultados:

```
BIN_FLOAT      NANVL(BIN_FLOAT,0)
----------     ------------------
1.235e+003              1.235E+003
       Nan                       0
```

Función POWER(n2,n1)

Devuelve "n2" elevado a la potencia "n1".

EJEMPLO

```
SELECT POWER(3,2) FROM DUAL;
```

Esta consulta nos devuelve el número 9.

Función ROUND(x,[,y])

Devuelve x redondeado a y posiciones a la derecha del punto decimal. Si no se indica, se eliminan todos los valores decimales.

EJEMPLO

```
SELECT ROUND(133.287,2) FROM DUAL;
```

Esta consulta nos devuelve el número 133,29.

Función SIGN(n)

Devuelve el signo de "n", siendo "n" un número. Los posibles valores a devolver por la función son los siguientes:

- -1: si el valor de "n" es negativo.
- 0: si el valor de "n" es igual a cero.
- 1: si el valor de "n" es positivo y > que 0.

EJEMPLO

```
SELECT SIGN(-15) FROM DUAL;
```

Esta consulta nos devuelve el resultado de -1.

Función SQRT(n)

Devuelve la raíz cuadrada del número "n" que se indique.

EJEMPLO

```
SELECT SQRT(26) FROM DUAL;
```

Esta consulta nos devuelve el número 5.09901951.

Función TRUNC(n1[,n2])

Devuelve n1 truncada n2 lugares decimales. Si se omite n2, entonces n1 se trunca 0 lugares. N2 puede ser negativo para truncar (hacer cero) n2 dígitos a la izquierda del punto decimal.

EJEMPLO

```
SELECT TRUNC(15.79,1) FROM DUAL;
```

Esta consulta nos devolvería el número 15.7.

EJEMPLO

```
SELECT TRUNC(15.79,-1) FROM DUAL;
```

Esta consulta nos devolvería el número 10.

Función VSIZE(expresión)

Devuelve el número de bytes con el que se representa internamente la "expresión" indicada.

EJEMPLO

```
SELECT VSIZE('ANTOLIN') FROM DUAL;
```

Esta consulta nos devolvería el número 7.

FUNCIONES DE FECHA

En este apartado se definen una serie de funciones representativas para el uso de las fechas y las horas.

Función	Descripción
ADD_MONTHS	Añadir mes/es a una fecha.
CURRENT_DATE	Fecha actual en la zona horaria de la sesión.
CURRENT_TIMESTAMP	Fecha y hora actual en la zona horaria de la sesión.
DBTIMEZONE	Zona horaria de la base de datos.
EXTRACT(datetime)	Extrae elementos de una fecha o fecha/hora.
LAST_DAY	Último día del mes de la fecha.
MONTHS_BETWEEN	Mes/es entre 2 fechas.
NEXT_DAY	Fecha del primer día que coincida con el indicado.
ROUND(date)	Redondea una fecha.
SYSDATE	Fecha actual del SGBD de Oracle.
SYSTIMESTAMP	Fecha, hora, segundos y zona horaria del SGBD de Oracle.

Función ADD_MONTHS(d,x)

Devuelve la fecha "d" más "x" meses que se le añadan.

EJEMPLO

```
SELECT ADD_MONTHS('01-01-08',3) FROM DUAL;
```

Esta consulta nos devuelve la fecha 01/04/08.

Función CURRENT_DATE

Devuelve la fecha actual en la zona horaria donde se encuentra abierta la sesión de conexión contra la base de datos. El valor devuelto es de tipo DATE.

EJEMPLO

```
SELECT CURRENT_DATE FROM DUAL;
```

Esta consulta nos devuelve la fecha actual de acuerdo a la zona horaria que tenga la sesión con la que nos hemos conectado a la base de datos Oracle.

Función CURRENT_TIMESTAMP

Devuelve la fecha actual y la hora en la zona horaria donde se encuentra abierta la sesión de conexión contra la base de datos. El valor devuelto es de tipo TIMESTAMP WITH TIME ZONE.

EJEMPLO

```
SELECT CURRENT_TIMESTAMP FROM DUAL;
```

Esta consulta nos devuelve la fecha y hora actual de acuerdo a la zona horaria que tenga la sesión con la que nos hemos conectado a la base de datos Oracle.

Función DBTIMEZONE

Devuelve el valor de la zona horaria que tiene la base de datos.

EJEMPLO

```
SELECT DBTIMEZONE FROM DUAL;
```

Esta consulta podría devolvernos por ejemplo: +00:00.

Función EXTRACT(datetime)

Extrae y devuelve el valor de uno de los campos de fecha y hora que se especifique.

Los posibles valores a extraer de una fecha son:

- YEAR: Extrae y devuelve el año de la fecha.
- MONTH: Extrae y devuelve el mes de la fecha.
- DAY: Extrae y devuelve el día de la fecha.
- HOUR: Extrae y devuelve la hora de la fecha y hora.
- MINUTE: Extrae y devuelve los minutos de la fecha y hora.
- SECOND: Extrae y devuelve los segundos de la fecha y hora.
- TIMEZONE_HOUR: Extrae y devuelve la hora en la zona horaria.
- TIMEZONE_MINUTE: Extrae y devuelve los minutos en la zona horaria.
- TIMEZONE_REGION: Extrae y devuelve la región de la zona horaria.
- TIMEZONE_ABBR: Extrae y devuelve la abreviatura de la zona horaria.

Los valores YEAR, MONTH, DAY, HOUR, MINUTE y SECOND se pueden extraer de cualquier formato de fecha y fecha/hora, mientras que los valores TIMEZONE solo se pueden extraer de valores de tipo TIMESTAMP WITH TIME ZONE.

La sintaxis de la función es: EXTRACT(valor a extraer FROM fecha).

EJEMPLO

```
SELECT EXTRACT(YEAR FROM SYSDATE)FROM DUAL;
```

Esta consulta nos devolvería 2011 si estamos dentro de este año cuando la ejecutemos.

EJEMPLO

```
SELECT EXTRACT(MONTH FROM DATE '1998-03-07')FROM DUAL;
```

Esta consulta nos devolvería 3.

Función LAST_DAY(fecha)

Devuelve la última fecha del último día del mes que contiene la fecha indicada. El valor devuelto siempre es una fecha.

EJEMPLO

```
SELECT LAST_DAY(SYSDATE) FROM DUAL;
```

Esta consulta nos devuelve el último día del mes correspondiente a la fecha actual (fecha del sistema).

Función MONTHS_BETWEEN(fecha1, fecha2)

Devuelve el número de meses existentes entre fecha1 y fecha2.

EJEMPLO

```
SELECT MONTHS_BETWEEN('31-12-09','31-05-08') FROM DUAL;
```

Esta consulta nos devuelve el número de meses 19.

Función NEXT_DAY(date,char)

Devuelve la fecha correspondiente al primer día que coincida con "char", y que sea mayor que la fecha "date" indicada.

EJEMPLO

```
SELECT NEXT_DAY('02-FEB-2001','MARTES') FROM DUAL;
```

Esta consulta nos devuelve 06/02/01 correspondiente al primer martes posterior a la fecha 2/2/01.

Función ROUND(fecha[,formato])

Devuelve una fecha redondeada a la unidad que se haya especificado en formato.

Si no se indica formato, devuelve la fecha correspondiente al día más cercano a la fecha especificada.

EJEMPLO

```
SELECT    ROUND(TO_DATE('15-12-09    12:12:00','DD-MM-YY
HH24:MI:SS')) FROM DUAL;;
```

Esta consulta nos devuelve la fecha '16/12/09'.

EJEMPLO

```
SELECT ROUND(TO_DATE('15-12-09'),'YEAR') FROM DUAL;
```

Esta consulta nos devuelve la fecha '01/01/10'.

Función SYSDATE

Esta función devuelve la fecha y hora actual del sistema operativo en el que reside la base de datos. El formato devuelto es de tipo DATE.

EJEMPLO

```
SELECT   TO_CHAR(SYSDATE,'DD-MM-YYYY   HH24:MI:SS')   FROM
DUAL;
```

Esta consulta nos devuelve la cadena ' 25-08-2009 12:36:22'.

Función SYSTIMESTAMP

Esta función devuelve la fecha y hora actual del sistema incluyendo segundos fraccionales y zona horaria del sistema operativo en el que reside la base de datos. El formato de la fecha devuelto es de tipo TIMESTAMP WITH TIME ZONE.

EJEMPLO

```
SELECT SYSTIMESTAMP FROM DUAL;
```

Esta consulta nos devuelve '23/05/11 21:00:36,533000 +02:00', correspondiente a la fecha y hora actual del sistema donde reside la base de datos en el momento en el que se ejecutó dicha sentencia.

FUNCIONES DE SESIÓN

En este apartado se definen una serie de funciones representativas para el uso de la sesión de conexión a la base de datos.

Función	Descripción
UID	Identificador de sesión.
USER	Nombre de usuario de sesión.
USERENV	Información de la sesión.

Función UID

Devuelve un número que identifica unívocamente la sesión del usuario que se conectó a la base de datos.

EJEMPLO

```
SELECT UID FROM DUAL;
```

Esta consulta nos devolvería un número que identifica la sesión donde se ejecuta esta sentencia.

Función USER

Esta función devuelve el nombre del usuario de sesión que ejecuta la sentencia.

EJEMPLO

```
SELECT USER FROM DUAL;
```

Esta consulta nos devolvería el nombre del usuario con el que se ha conectado a la sesión y que está ejecutando en este momento la instrucción.

Función USERENV('parámetro')

Esta función devuelve información sobre la sesión actual, para ello utiliza una serie de parámetros que nos retornará la información. Estos parámetros son:

- CLIENT_INFO: Devuelve 64 bytes de información de sesión del usuario.
- ENTRYID: Devuelve el número de entradas actuales que han sido auditadas.
- ISDBA: Devuelve 'TRUE' (verdadero) si el usuario ha sido autenticado con privilegios de DBA. En caso contrario, devuelve 'FALSE'.
- LANG: Devuelve la abreviatura del lenguaje utilizado en la sesión actual.
- LANGUAGE: Devuelve el lenguaje y el territorio usado en la sesión actual.
- SESSIONID: Devuelve el identificador de sesión auditado.
- TERMINAL: Devuelve el identificador del sistema operativo para el terminal de la sesión actual.

EJEMPLO

```
SELECT USERENV('LANGUAGE') FROM DUAL;
```

Esta función nos devolvería por ejemplo "SPANISH_SPAIN.WE8MSWIN1252", si tenemos instalado este juego de caracteres en la base de datos.

SUPUESTO PRÁCTICO 10: ***Resolución en el Anexo I.***

Realice las consultas que se le formulan en los siguientes puntos.

1. Seleccione el nombre y código de hospital de todos los hospitales.
2. Encuentre a todos los miembros del personal de plantilla cuyo nombre comience por A y cuyo apellido comience por una letra < M. Se quiere conocer de cada uno de ellos, el nombre, apellidos y turno. Ordenar el listado por el nombre y apellidos .

3. Haga un listado de las enfermeras que ganen entre 2.000.000 y 2.500.000 pts. (1€ = 166.386 pts). Visualizar nombre y salario de la plantilla sanitaria.
4. Muestre el nombre de los hospitales con más de 100 camas, ordenados descendentemente.
5. Muestre la fecha del sistema con el formato "<día de la semana en letras> - <día en número> - <mes en letra>-<año en 4 dígitos>- <siglo>- <hora en 24>: <minutos> ".

FUNCIONES DE AGRUPAMIENTO

Son funciones que operan contra el conjunto de filas que recupera la consulta. Su resultado es un único valor y su funcionalidad se expresa en el siguiente cuadro:

Comando	Descripción
MIN	Calcula el valor mínimo de una columna.
MAX	Calcula el valor máximo de una columna.
AVG	Calcula la media aritmética de una columna.
SUM	Calcula la suma de todos los campos de una columna.
COUNT	Cuenta el número de filas de una columna.
COVAR_SAMP	Calcula la covarianza simple.
STDDEV	Calcula la desviación estándar.
STDDEV_SAMP	Calcula la desviación estándar aplicando la raíz cuadrada de la varianza simple.
VAR_SAMP	Calcula la varianza simple.
VARIANCE	Calcula la varianza.

Para utilizar las funciones de agrupamiento, podemos utilizar dos tipos diferentes de sintaxis de consulta SELECT dependiendo de si además de estas funciones aparecen o no columnas de una tabla:

Sintaxis utilizando únicamente funciones de agrupamiento:

```
SELECT Funciones de agrupamiento FROM Tabla
[WHERE Condiciones de filtro]
[ORDER BY Lista de campos];
```

Sintaxis utilizando funciones de agrupamiento y columnas de tabla:

```
SELECT Funciones de agrupamiento, campos FROM Tabla
[WHERE Condiciones de filtro]
GROUP BY campos
[ORDER BY Lista de campos];
```

Cuando se realiza una consulta en la que se van a mostrar resultados de funciones de agrupamiento y columna/s de una o varias tablas, es obligatorio incluir la cláusula GROUP BY seguida de todas las columnas incluidas en el SELECT que no sean funciones de agrupamiento.

Funciones MAX y MIN

Devuelven el máximo y el mínimo de un conjunto de valores contenidos en una columna específica de una consulta.

Su sintaxis es:

```
MIN (expresión)
MAX (expresión)
```

Donde `expresión` es la columna sobre la que se desea realizar el cálculo. `Expresión` puede incluir el nombre de una columna de una tabla, una constante o una función (la cual puede ser intrínseca o definida por el usuario, pero no otra función de columna).

EJEMPLO

```
SELECT MIN(VELO_CRUCE), MAX(VELO_CRUCE)
FROM AVIONES;
```

Esta consulta nos devuelve la velocidad crucero mínima y máxima de todos los aviones contenidos en la tabla AVIONES.

Función AVG

Esta función calcula la media aritmética de un conjunto de valores contenidos en la columna especificada en la consulta.

Su sintaxis es la siguiente:

```
AVG(expresión)
```

En donde `expresión` representa la columna que contiene los datos numéricos para los que se desea calcular la media, o una expresión que realiza un cálculo, utilizando los datos de dicha columna. La media calculada por AVG es la media aritmética (la suma de los valores dividido por el número de valores).

La función AVG no incluye ningún campo NULL en el cálculo.

EJEMPLO

```
SELECT AVG(Gastos) AS Promedio
FROM Pedidos
WHERE Gastos > 1000;
```

Esta consulta nos devuelve la media de los gastos mayores a 1.000 euros para la tabla pedido.

Función SUM

Esta función devuelve la suma del conjunto de valores contenidos en una columna o columnas de una consulta.

Su sintaxis es: `SUM(expresión)`

En donde `expresión` representa el nombre de la columna que contiene los datos que desean sumarse, o una expresión que realiza el cálculo utilizando los datos de varias columnas.

EJEMPLO

```
SELECT SUM(Compras + Otros Gastos) AS TotalGastos
FROM Pedidos;
```

Esta consulta nos devuelve el total de gastos que se producen en la tabla pedidos para cada una de las columnas implicadas (`compras` y `Otros Gastos`).

Función COUNT

Calcula el número de registros devueltos por una consulta.

Su sintaxis es: `COUNT(expresión)`

Donde `expresión` contiene el nombre de la columna que se desea contar.

Aunque `expresión` puede realizar un cálculo sobre una columna, COUNT simplemente cuenta el número de registros sin tener en cuenta qué valores se almacenan en las filas. La función COUNT no cuenta los registros que tienen NULL a menos que `expresión` sea el carácter comodín asterisco (*). Si lo utiliza, COUNT calcula el número total de registros incluyendo aquellos que contienen valores NULL.

El uso de COUNT(*) es considerablemente más rápido que COUNT(columna).

EJEMPLO

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

Esta consulta nos devuelve el número total de registros almacenados en la tabla Pedidos.

Función COVAR_SAMP(expresion1,expresion2)

Nos devuelve la covarianza simple de 2 conjuntos de valores.

EJEMPLO

```
SELECT COVAR_SAMP(salario,comision) FROM empleado;
```

Esta consulta nos devolverá la covarianza simple entre los valores de `salario` y `comision` de todos los empleados de la tabla.

Función STDDEV([Distinct] columna)

Devuelve la desviación estándar de una columna con un conjunto de valores.

EJEMPLO

```
SELECT STDDEV(sueldo) FROM empleado;
```

Esta consulta nos devolvería la desviación estándar de la columna `sueldo` de todos los empleados.

Función STDDEV_SAMP(columna)

Devuelve el cálculo correspondiente a la acumulación de la desviación estándar de la "columna", una vez aplicada la raíz cuadrada de la varianza simple.

EJEMPLO

```
SELECT STDDEV_SAMP(sueldo) FROM empleado;
```

Esta consulta nos devolvería la desviación estándar de la columna sueldo una vez aplicada la raíz cuadrada de la varianza simple sobre la misma.

Función VAR_SAMP(columna)

Devuelve la varianza simple de un conjunto de valores que normalmente estarán en una columna.

EJEMPLO

```
SELECT VAR_SAMP(sueldo) FROM empleado;
```

Esta consulta nos devolvería la varianza simple de la columna `sueldo` de los empleados.

Función VARIANCE([Distinct] columna)

Devuelve la varianza de un conjunto de valores que normalmente estarán en una columna.

EJEMPLO

```
SELECT VARIANCE(sueldo) FROM empleado;
```

Esta consulta nos devolvería la varianza de la columna `sueldo` de todos los empleados.

AGRUPAMIENTO DE REGISTROS

El agrupamiento de registros es un concepto de SQL que permite dividir los resultados de una consulta en grupos de filas que comparten características comunes.

Para hacerlo tenemos que utilizar la cláusula GROUP BY a la que también se le puede aplicar la cláusula HAVING.

Cláusula GROUP BY

Para hacer un agrupamiento por columnas, hay que utilizar la cláusula GROUP BY después de indicar las condiciones WHERE y antes del ordenamiento de registros ORDER BY.

La sintaxis quedaría de la siguiente forma:

```
SELECT Funciones de agrupamiento, campos FROM Tabla
[WHERE Condiciones de filtro]
GROUP BY campos
[ORDER BY Lista de campos];
```

La cláusula GROUP BY debe aparecer obligatoriamente entre la cláusula SELECT y FROM, y solo se podrá aplicar cuando en la consulta SELECT aparezca al menos una función de agrupamiento y al menos también una columna.

Después de la cláusula GROUP BY debe aparecer obligatoriamente todas las columnas de la cláusula SELECT que no sean funciones de agrupamiento.

Los valores NULL en los campos GROUP BY se agrupan y no se omiten. No obstante, los valores NULL no se evalúan en ninguna de las funciones de columna.

EJEMPLO

```
SELECT DESTINO, MIN(HORA_SALIDA)
FROM VUELOS
GROUP BY DESTINO
```

En esta consulta se visualiza para cada `destino` de la tabla `vuelos`, la menor de las horas de salida entre todos los registros del mismo destino.

EJEMPLO

```
SELECT ORIGEN, COUNT(*)
FROM VUELOS
GROUP BY ORIGEN
```

En esta consulta se visualiza el número de vuelos que existen para cada uno de los orígenes de la tabla `vuelos`.

Cláusula HAVING

Esta cláusula se combina inseparablemente con la cláusula GROUP BY, y se utiliza para seleccionar dentro de los grupos obtenidos con la cláusula GROUP BY, aquellos que cumplan un criterio especificado con esta cláusula.

EJEMPLO

```
SELECT ORIGEN, MIN(COSTE_VUELO)
FROM VUELOS
GROUP BY ORIGEN
HAVING MIN(HORA_SALIDA) < 20000
```

En esta consulta se agrupan los datos por el mínimo coste, para cada origen distinto de la tabla `vuelos`, y de esos solo se muestran aquellos que tengan un coste menor a 20.000 euros.

EJEMPLO

```
SELECT ORIGEN, COUNT(*)
FROM VUELOS
GROUP BY ORIGEN
HAVING COUNT(*) > 2
```

En esta consulta se visualizan los orígenes que tengan más de dos vuelos.

RECUPERACIÓN JERÁRQUICA

El uso de la función LPAD combinada con las cláusulas START WITH y CONNECT BY PRIOR resulta muy útil en aquellas consultas sobre tablas en las que existen campos dependientes de otros.

Ejemplos de estas tablas serían aquellas en las que en una misma tabla se han creado 2 campos NIF (evidentemente con nombres distintos), uno para identificar a la propia persona y otro para identificar cuál es el NIF del padre o familiar.

También en la misma dinámica se podrían haber creado 2 campos NIF, uno para identificar a la persona, y otro para identificar el NIF de la persona de la que dependemos funcionalmente en la empresa.

Ejemplo

```
SELECT LPAD(' ',2*(LEVEL-1)) || ename org_chart, empno,
       mgr, job
FROM emp
START WITH job = 'PRESIDENT'
CONNECT BY PRIOR empno = mgr;
```

Con esta consulta se hace lo siguiente:

1. Se comprueba si la tabla existe (cláusula FROM emp).
2. Se recorre la tabla para localizar el empleado cuyo trabajo es 'PRESIDENT' (cláusula START WITH...).
3. Se hace un segundo recorrido de la tabla para extraer todos los registros cuyo campo empno = mgr (cláusula CONNECT BY PRIOR...).
4. Si se omite la cláusula PRIOR no habrá un segundo recorrido.
5. Se lista en pantalla la información ajustándola a la izquierda con la función LPAD por el nivel de recuperación de los registros (2*(LEVEL – 1)).

SUPUESTO PRÁCTICO 11: ***Resolución en el Anexo I.***

Realice las consultas que se le formulan en los siguientes puntos.

1. Encontrar el salario medio de los empleados no sanitarios agrupados por el oficio que desempeñan. Mostrar el conteo de personas que trabajan; por oficio, salario medio del oficio y nombre del oficio. La media del salario se habrá de mostrar con el siguiente formato: redondeado a 2 decimales y con separados de miles. Los títulos a mostrar para las 3 columnas serán: Nº Pers., Salario M, Oficio.
2. Diseñar una consulta que muestre el nombre de aquellos doctores que comiencen por la letra 'I' que se repitan en la tabla doctores más de una vez.
3. Diseñar una consulta que muestre el nombre, apellidos, edad y sexo de todos los enfermos de la tabla enfermo que sean del sexo femenino o bien que siendo del sexo masculino tengan más de 40 años. No se deben repetir filas con la misma información en el listado que se muestre mediante la consulta.

TRATAMIENTO DE NULOS

Uno de los aspectos más delicados en el uso de la información en las bases de datos es el tratamiento de los valores nulos. Cada base de datos maneja estos valores de diferente forma, por lo que hay que tener siempre muy claro cómo se usan.

Dentro de las bases de datos de Oracle los valores nulos se identifican con la palabra reservada `NULL`.

Para preguntar en SQL (de Oracle) si una columna posee un valor nulo, no se hace mediante igualación sino preguntando por `IS NULL`. A continuación, podemos ver dos ejemplos, el primero corresponde con la forma correcta de preguntar por valores nulos, y el segundo, con la forma incorrecta.

Ejemplo (forma correcta de preguntar por valores nulos)

```
SELECT * FROM EMPLEADOS
WHERE ACTIVIDAD IS NULL;
```

Esta consulta devuelve todos los datos de la tabla `empleado` para aquellas filas cuya columna `actividad` esté vacía.

Ejemplo (forma incorrecta de preguntar por valores nulos)

```
SELECT * FROM EMPLEADOS
WHERE ACTIVIDAD = "";
```

Esta consulta es incorrecta en cuanto a la búsqueda de valores nulos. Para preguntar si una columna tiene valores NO VACÍOS, se pregunta por `IS NOT NULL`.

Ejemplo (forma correcta de preguntar por valores no nulos)

```
SELECT * FROM EMPLEADOS
WHERE ACTIVIDAD IS NOT NULL;
```

Esta consulta devuelve todos los datos de la tabla `empleado` para aquellas filas cuya columna actividad NO esté vacía.

Operaciones aritméticas con valores nulos

Cualquier operación aritmética sobre un campo nulo nos devolverá como resultado un valor nulo.

Tomemos como ejemplo la siguiente tabla:

COL_A	COL_B
15	10
35	35
140	NULL
NULL	100
NULL	NULL
7	110
33	60

COL_A	COL_B
NULL	NULL
NULL	NULL
NULL	NULL

Si sobre la tabla anterior realizamos la siguiente consulta:

```
SELECT COL_A + COL_B
FROM NULOS
```

El resultado que obtenemos es el siguiente para cada fila consultada:

COL_A+COL_B
25
70
NULL
NULL
NULL
117
93
NULL
NULL
NULL

Operaciones de comparación con valores nulos

Dos valores nulos no son iguales ni son distintos, sino indeterminados. Por tanto, cualquier operación de comparación con un valor nulo no devolverá ni verdadero (TRUE) ni falso (FALSE).

Tomemos como ejemplo la siguiente consulta sobre los valores de la tabla comentada en el apartado anterior:

```
SELECT *
FROM NULOS
WHERE COL_A = COL_B
```

El resultado que obtendríamos sería el siguiente:

COL_A	COL_B
35	35

Esto indica que para el SGBD de Oracle los únicos valores idénticos de ambas columnas son los mostrados, y no así los valores NULL equivalentes entre las columnas A y B en varias de las filas de la tabla.

LA FUNCIÓN NVL

Esta función nos permite evaluar expresiones para determinar si son nulas, y automáticamente convertirlas a un valor.

Su sintaxis es la siguiente: `NVL(expresion1,expresion2)`

Ejemplo de consulta convirtiendo valores nulos

```
SELECT NVL(salario,0) FROM EMPLEADOS
```

Esta consulta nos devuelve todos los salarios de la tabla `empleados`, convirtiendo a valor 0 todos aquellos valores nulos que encuentre en la columna `salario`. El resto de valores se muestran tal cual estén grabados en la columna.

Ejemplo de operación aritmética segura con posibles valores nulos

```
SELECT SUM(NVL(salario,0))+ MAX(NVL(comision,0)) FROM
EMPLEADOS
```

Esta consulta nos devuelve el sumatorio de `salario` (habiendo convertido los posibles valores nulos a 0), sumado al valor máximo de la `comisión` (convirtiendo un posible valor nulo a 0). De esta forma aseguramos que la operación nos devolverá al menos el valor 0 y no un nulo.

Ejemplo de operación de comparación segura con posibles valores nulos

```
SELECT * FROM EMPLEADO
WHERE nvl(salario,0) <= nvl(comision,0)
```

Esta consulta nos devuelve todos los empleados cuyo salario sea inferior o igual a las comisiones que cobra, sustituyendo para la consulta, aquellos valores nulos en ambas columnas, por 0.

SUBCONSULTAS

Las subconsultas nos permiten crear consultas dentro de otras, bajo ciertas premisas de unión o concatenación de las mismas.

Concatenación básica de subconsultas

Las subconsultas son consultas SELECT anidadas y responden a la siguiente sintaxis:

```
SELECT <lista de columnas>
FROM <lista de tablas>
WHERE <nombre de columna> <CONCATENARDOR>
                                    (SELECT <nombre de columna>
                                     FROM <lista de tablas>
                                     WHERE <Predicado>)
```

El contenido de **concatenador** es un operador de comparación o la cláusula IN. Recordemos que los operadores de comparación son: >, <, >=, <=, =, <>, !=...

Al realizar una subconsulta hay que diseñar bien la misma, de forma que el resultado que devuelva la subconsulta sea un único valor, para poderlo comparar con el valor de la columna de la consulta principal, a no ser que utilicemos los concatenadores IN, ANY, ALL en cuyo caso la subconsulta sí que puede devolver más de un valor, pero para una única columna.

Cada SELECT se ejecuta una única vez desde la más interna hasta la más externa.

EJEMPLO

```
SELECT PLAZAS_LIBRE
FROM RESERVAS
WHERE FECHA_SALIDA='20-FEB-92'
```

```
AND NUM_VUELO IN ( SELECT NUM_VUELO
                   FROM VUELOS
                   WHERE ORIGEN='MADRID'
                   AND DESTINO='LONDRES')
```

Esta consulta recupera las plazas libres que hay en cada vuelo MADRID-LONDRES del día 20/02/1992.

EJEMPLO

```
SELECT PLAZAS_LIBRE
FROM RESERVAS
WHERE FECHA_SALIDA='20-FEB-92'
AND TIPO_AVION = ( SELECT TIPO_AVION
                   FROM VUELOS
                   WHERE ORIGEN='MADRID'
                   AND DESTINO='LONDRES'
AND FECHA_SALIDA = TO_DATE('01012002','DDMMYYYY'))
```

Esta consulta recupera las plazas libres que hay en los tipos de aviones que salen el 20/2/92 y que sean idénticos al que sale de MADRID el 1/1/2002 y llega a LONDRES.

Cláusula ANY, ALL

Esta cláusula se usa para poder utilizar operadores de comparación con subconsultas que nos devuelvan más de un valor único como resultado. Pero siempre recordando que todos los valores devueltos en la subconsulta corresponden a una única columna.

La sintaxis es la siguiente:

```
SELECT <lista de columnas>
FROM <lista de tablas>
WHERE <nombre de columna> <CONCATENAR> {ANY/ALL}
                         (SELECT <nombre de columna>
                          FROM <lista de tablas>
                          WHERE <Predicado>)
```

Una expresión ANY es cierta si lo es para algún valor de los que devuelve la subconsulta. Equivale a utilizar la cláusula IN cuando se utiliza un operador de igualdad.

Una expresión ALL es cierta si lo es para todos los valores que devuelve la subconsulta.

Por ejemplo, veamos el resultado de los siguientes procesos de comparación:

Valor 1	Operador	Valor 2	Resultado
3	>	ANY(2,5,7)	Cierto
3	=	ANY(2,5,7)	Falso
3	>	ALL(2,5,7)	Falso
3	<	ALL(9,10,11)	Cierto

EJEMPLO

```
SELECT *
FROM AVIONES
WHERE LONGITUD > ALL (SELECT ENVERGADURA
                      FROM AVIONES)
```

O bien también se podría poner así:

```
SELECT *
FROM AVIONES
WHERE LONGITUD > ( SELECT MAX(ENVERGADURA)
                   FROM AVIONES)
```

Esta consulta nos devuelve toda la información de la tabla aviones cuya longitud sea mayor que la envergadura de todos ellos.

Alias para subconsultas

El concepto de alias ya se definió con anterioridad en este manual, pero es dentro de las subconsultas donde se suele utilizar más, dado que es habitual realizar consultas anidadas sobre la misma tabla, de forma que hay que definir alias que diferencien los predicados de esa tabla para cada una de las consultas.

EJEMPLO

```
SELECT *
FROM RESERVAS A
```

```
WHERE A.PLAZAS_LIBRES > (SELECT AVG(B.PLAZAS_LIBRES)
                         FROM RESERVAS B
                        WHERE B.NUM_VUELO = A.NUM_VUELO)
```

Esta consulta recupera las reservas cuyo número de plazas libres sea mayor que la media para ese mismo vuelo.

Cláusula EXISTS-NOT EXISTS

Se define para comprobar la existencia o ausencia del valor devuelto por una subconsulta. Una expresión con EXIST devuelve verdadero (TRUE) si la subconsulta nos devuelve al menos un valor.

La sintaxis es la siguiente:

```
SELECT <lista de columnas>
FROM <lista de tablas>
WHERE EXISTS / NOT EXISTS (SELECT NULL / *
                           FROM <lista de tablas>
                           WHERE <Predicado>)
```

En esta sintaxis se especifica para la subconsulta, que los valores a devolver son NINGUNO o *. El motivo de realizarlo así es que la misión de la subconsulta es evaluar que haya resultados disponibles de acuerdo a los criterios indicados en la misma, mientras que la consulta principal evalúa que existan los mismos o que no existan (no haya resultados), pero le da igual qué tipo de resultados sean. Lo más normal es utilizar en la subconsulta la cláusula NULL cuando se utilice EXISTS o NOT EXISTS.

EJEMPLO

```
SELECT *
FROM VUELOS
WHERE ORIGEN = 'MADRID'
AND EXISTS ( SELECT NULL
             FROM RESERVAS
             WHERE PLAZAS_LIBRES > 0
             AND NUM_VUELO = VUELOS.NUM_VUELO)
```

Esta consulta obtiene todos los datos de la tabla `vuelo` para los aviones que partan de 'MADRID' y para los cuales existan plazas libres.

Cláusulas de conjuntos (UNION, INTERSECT, MINUS)

Esta cláusula se define para recuperar información que se obtiene a partir de más de una consulta concatenada en una única instrucción, a través de las sentencias UNION, INTERSECT o MINUS.

La sintaxis es la siguiente:

```
SELECT <clausulas select>
{UNION | INTERSECT | MINUS
SELECT <cláusulas select>
```

Cada SELECT devuelve un conjunto de filas.

Condiciones de cada estructura SELECT:

- Todas deben ser iguales, o compatibles una a una. Esto supone que por cada columna tengamos un único tipo de dato.
- Pueden ser completas (WHERE, GROUP BY,...), exceptuando las cláusulas ORDER BY que se ubicarán al final de la última SELECT.

La funcionalidad de las cláusulas UNION, INTERSECT y MINUS es la siguiente:

Comando	Descripción
UNION	Muestra el conjunto de filas de ambas consultas SELECT.
INTERSECT	Muestra el conjunto de filas que son idénticas en ambas SELECT.
MINUS	Muestra el conjunto de filas de la primera consulta que no se encuentran en la segunda.

EJEMPLO

```
SELECT apellido FROM enfermo WHERE s = 'M'
UNION
SELECT apellido FROM enfermo WHERE s = 'F'
ORDER BY apellido;
```

Esta consulta nos lista los apellidos de todos los enfermos cuyo sexo sea MASCULINO, y todos aquellos apellidos de los enfermos cuyo sexo sea FEMENINO. Al final ordena toda la lista por el `apellido`.

EJEMPLO

```
SELECT apellido FROM enfermo WHERE s = 'M'
INTERSECT
SELECT apellido FROM enfermo WHERE s = 'F'
ORDER BY apellido;
```

Esta consulta nos lista los apellidos de todos los enfermos cuyo sexo sea MASCULINO, y todos aquellos apellidos de los enfermos cuyo sexo sea FEMENINO en el que coincidan los apellidos.

EJEMPLO

```
SELECT apellido FROM enfermo WHERE s = 'M'
MINUS
SELECT apellido FROM enfermo WHERE s = 'F'
ORDER BY apellido;
```

Esta consulta nos lista los apellidos de todos los enfermos cuyo sexo sea MASCULINO, y todos aquellos apellidos de los enfermos cuyo sexo sea FEMENINO en el que los apellidos de los enfermos masculinos no coincida con los apellidos de los enfermos femeninos.

CONSULTAS JOIN

Las consultas de JOIN no son más que consultas que combinan los valores de varias tablas a través de columnas iguales.

Hay 3 tipos de JOINS:

- Producto cartesiano.
- Join de igualdad.
- Join externo.

Producto cartesiano

Nos muestra la información combinada de columnas de varias tablas en una única consulta.

EJEMPLO

```
SELECT apellido, d.dept_no, dnombre
FROM emp, dept d;
```

Esta consulta nos saca un informe con el apellido del empleado, el número de departamento al que pertenece y el nombre del departamento. Este tipo de JOIN no implica necesariamente que haya algún campo común en ambas tablas.

Join de igualdad

Es el JOIN más simple y se utiliza en la comparación de campos iguales de 2 tablas para sacar resultados combinados.

EJEMPLO

```
SELECT e.apellido, d.dept_no, d.dnombre
FROM emp, dept d
WHERE e.dept_no = d.dept_no;
```

Esta consulta nos saca un informe con el apellido del empleado, el número de departamento al que pertenece y el nombre del departamento, teniendo como campo de unión el campo DEPT_NO.

Join externo

Consiste en especificar con un signo + en aquella columna que forma parte del predicado de igualdad y corresponde a una columna que le faltan valores (puede tener nulos).

EJEMPLO

```
SELECT e.code_no, experimento, res1, res2
FROM experimento e, resultados r
WHERE e.code_no = r.code_no(+)
```

NEW 12C CONSULTAS CON EXPRESIONES CASE

La expresión CASE permite utilizar la lógica del IF … THEN … ELSE en una sentencia SQL sin tener que invocar a procedimientos y funciones almacenadas en base de datos. La sintaxis es:

Fig. 8-3 Sintaxis de la expresión CASE.

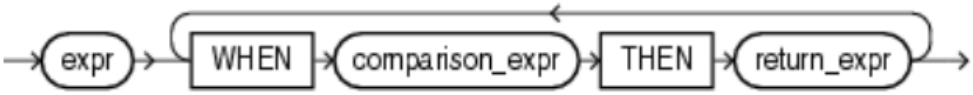


Fig. 8-4 Sintaxis de SIMPLE_CASE_EXPRESSION.

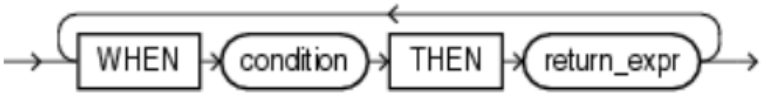


Fig. 8-5 Sintaxis de SEARCHED_CASE_EXPRESSION.

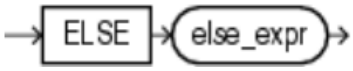


Fig. 8-6 Sintaxis de ELSE_CLAUSE.

En una expresión CASE, la base de datos de Oracle recorre cada cláusula WHEN…THEN indicada en la sentencia, comenzando por la primera, hasta que se cumplan las condiciones de alguna de ellas. En el caso de que ninguna cláusula WHEN cumpla las condiciones especificadas, entonces la consulta ejecutará el código indicado en la sentencia ELSE.

Si no se incluye sentencia ELSE y no se cumple ninguna condición WHEN, entonces la base de datos devolverá un NULL como resultado.

El máximo número de argumentos en una expresión CASE es de 255 (incluyendo las expresiones CASE y ELSE. Los pares WHEN…THEN cuentan como dos argumentos.

Los tipos de datos permitidos para las expresiones (`expr`), las expresiones de comparación (`comparasion_expr`) y los resultados (`return_expr`) son: `CHAR`, `VARCHAR2`, `NCHAR` o `NVARCHAR2`, `NUMBER`, `BINARY_FLOAT` o `BINARY_DOUBLE`.

EJEMPLO

```
SELECT nombre_cliente
CASE limite_credito WHEN 100 THEN 'Bajo'
                    WHEN 5000 THEN 'Alto'
                    ELSE 'Medio'
END AS 'LIMITE'
FROM clientes;
```

Esta consulta nos saca un informe con el nombre del cliente y el límite de crédito del que dispone indicándolo de forma textual con los siguientes valores: Bajo, Alto o Medio, dependiendo del valor que haya en la columna *limite_credito*.

```
NOMBRE_CLIENTE              LIMITE
--------------------------  ------------
PEDRO                       Medio
JOSE                        Alto
ANDRÉS                      Bajo
JACINTO                     Medio
…
```

EJEMPLO

```
SELECT AVG(CASE WHEN e.salario > 2000 THEN e.salario
           ELSE 2000 END) "SALARIO MEDIO"
FROM empleados e;
```

Esta consulta nos saca un informe con el salario medio de los empleados de la tabla EMPLEADOS teniendo en cuenta que cualquier salario inferior o igual a 2.000, se considerará como que fuese equiparado a 2.000.

```
SALARIO MEDIO
-------------------------
6350.65
```

EJEMPLO

```
SELECT NUMERO_SOLICITUD,
CASE
WHEN CODIGO_EMPLEADO IS NOT NULL THEN 'TIPO SOLICITUD:
De un Empleado'
WHEN CODIGO_DEPARTAMENTO IS NOT NULL THEN 'TIPO
SOLICITUD: De un Departamento'
```

```
  WHEN CODIGO_EXTRANJERO IS NOT NULL THEN 'TIPO
SOLICITUD: De una Oficina en el Extranjero'
  ELSE 'TIPO SOLICITUD: Sin origen identificado'
  end AS ORIGEN
  from SOLICITUDES
  ORDER BY 3;
```

Esta consulta nos saca un informe todas las solicitudes contenidas en la tabla SOLICITUDES, indicando el número de solicitud y el origen. Se introduce una sentencia CASE para identificar y codificar con un texto los distintos orígenes de las solicitudes que se determinan por la existencia de contenido en distintos campos.

```
NUMERO_SOLICITUD    ORIGEN
----------------    ------------
123456              TIPO SOLICITUD: De un Empleado
564452              TIPO SOLICITUD: De un Departamento
567742              TIPO SOLICITUD: De una Oficina en
                    el Extranjero
65455424            TIPO SOLICITUD: Sin origen
                    identificado
…
```

NEW 12C CONSULTAS DE REGISTROS LIMITADOS

Oracle 12c introduce nuevas cláusulas en las consultas que permiten limitar el número de registros que devuelve una consulta.

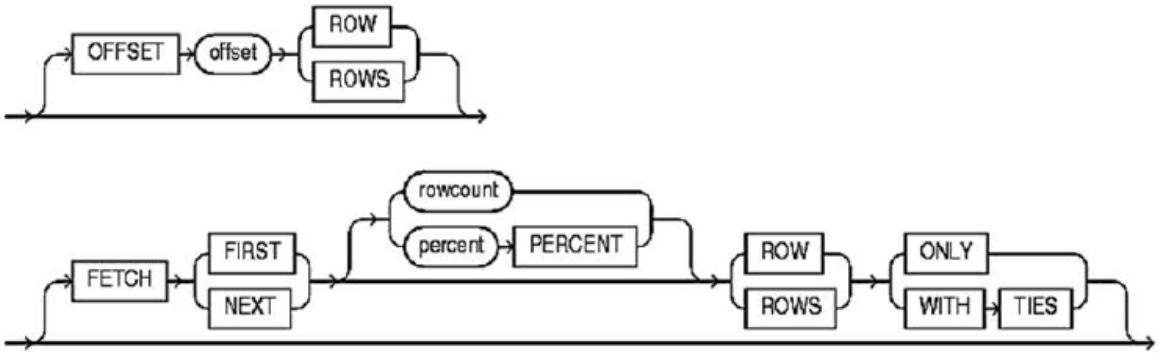


Fig. 8-7 Sintaxis de las cláusulas para limitar registros.

Se pueden especificar un desplazamiento (OFFSET) y un número de registros (FETCH) o porcentaje de registros (PERCENT) que se van a devolver en la consulta.

Además, las cláusulas de recuperación de registros limitados se pueden combinar con la cláusula de ordenación ORDER BY con objeto de asegurar la ordenación de los resultados devueltos por la consulta.

Offset

Esta cláusula se utiliza para especificar un número de registros que hay que saltar antes de comenzar a recuperar las filas que se hayan limitado.

OFFSET debe ser un número o una expresión que devuelva un valor numérico. Si se especifica un valor negativo, la cláusula OFFSET lo tratará como un valor 0. Si se especifica un valor NULL o un número mayor o igual que el número de filas totales que devuelve la consulta, entonces se devolverán 0 registros. Si OFFSET incluye una fracción, entonces la porción fraccional será truncada.

Si no se especifica esta cláusula, entonces OFFSET es cero y la consulta limitada comenzará por la primera fila.

Row / Rows

Estas palabras clave se pueden utilizar indistintamente y se proporcionan con objeto de tener más claridad semántica en la instrucción de consulta.

Fetch

Esta cláusula se utiliza para especificar el número de registros o el porcentaje de registros que tiene que devolver la consulta.

Si no se especifica esta cláusula, entonces la consulta devolverá todos los registros, comenzando en la fila OFFSET + 1

First / Next

Estas palabras claves se pueden utilizar indistintamente y se proporcionan con objeto de tener más claridad semántica en la instrucción de consulta.

Rowcount / Percent PERCENT

Se utiliza ROWCOUNT para especificar el número de registros que se quiere devolver en la consulta. ROWCOUNT debe ser un número o una expresión que devuelve un valor numérico. Si se especifica un valor negativo, entonces ROWCOUNT será tratado como si tuviese valor 0. Si ROWCOUNT es mayor que el número de registros disponibles comenzando por la fila OFFSET + 1, entonces se devolverán todos los registros disponibles. Si ROWCOUNT incluye una fracción, entonces la porción fraccional será truncada. Si ROWCOUNT es NULL, entonces no se devolverá ninguna fila en la consulta.

Se utiliza `percent` PERCENT para especificar el porcentaje sobre el total de número de filas seleccionadas que se quieren devolver. `Percent` debe ser un número o una expresión que devuelva un valor numérico. Si se especifica un valor negativo, entonces `Percent` será tratado como si tuviese valor 0. Si `Percent` es NULL, entonces no se devolverá ninguna fila en la consulta.

Only / With Ties

La cláusula ONLY se utiliza para devolver exactamente el número o porcentaje de registros.

La cláusula WITH TIES se utiliza para devolver registros adicionales con el mismo patrón de ordenación que el último registro recuperado. Si se especifica WITH TIES, entonces se deberá especificar la cláusula ORDER BY. Si no se especifica la cláusula ORDER BY, entonces no se recuperará ningún registro adicional.

Restricciones de uso de las cláusulas de limitación de registros

Las cláusulas para limitar los registros devueltos en una consulta también presentan restricciones de uso que se indican a continuación:

- No se pueden utilizar estas cláusulas en las consultas FOR UPDATE.
- Si se especifican estas cláusulas, entonces no pueden aparecer en la consulta las pseudocolumnas CURRVAL o NEXTVAL.
- A las vistas materializadas no se les puede asociar la opción de refresco incremental, si contienen en la consulta cláusulas de limitación de registros.

- Si la consulta contiene columnas con idéntico nombre y se especifican cláusulas de limitación, entonces se producirá el error ORA-00918. Para evitar este error se pueden utilizar alias de columna.

EJEMPLO

```
SELECT id_empleado, nombre_empleado
FROM empleados
ORDER BY id_empleado
FETCH FIRST 5 ROWS ONLY;
```

Esta consulta devuelve los 5 primeros empleados con el código de empleado (id_empleado) más bajo.

```
 ID_EMPLEADO          NOMBRE_EMPLEADO
 -----------------    ------------
 10                   JACINTO
 11                   BENITO
 12                   RODRIGO
 13                   ALFREDO
 14                   JUAN
```

EJEMPLO

```
SELECT id_empleado, nombre_empleado
FROM empleados
ORDER BY id_empleado
OFFSET 5 ROWS FETCH NEXT 5 ROWS ONLY;
```

Esta consulta devuelve los siguientes 5 empleados con el código de empleado (id_empleado) más bajo, saltando los 5 primeros.

```
 ID_EMPLEADO          NOMBRE_EMPLEADO
 ----------------     ------------
 15                   JORGE
 16                   EMILIO
 17                   VICENTE
 18                   PEDRO
 19                   RAFAEL
```

EJEMPLO

```
SELECT id_empleado, salario
FROM empleados
ORDER BY salario
FETCH FIRST 5 PERCENT ROWS ONLY;
```

Esta consulta devuelve los primeros 5 empleados con el porcentaje de salario más bajo.

```
ID_EMPLEADO         SALARIO
----------------    ------------
10                  1000
16                  1010
20                  1100
25                  1200
33                  1300
```

EJEMPLO

```
SELECT id_empleado, salario
FROM empleados
ORDER BY salario
FETCH FIRST 5 PERCENT ROWS WITH TIES;
```

Esta consulta devuelve los primeros 5 empleados con el porcentaje de salario más bajo, pero además al haber introducido la cláusula WITH TIES, también devuelve el resto de empleados cuyo salario sea equivalente al salario del último empleado recuperado en los primeros 5 registros.

```
ID_EMPLEADO         SALARIO
----------------    ------------
10                  1000
16                  1010
20                  1100
25                  1200
33                  1300
40                  1300
51                  1300
61                  1300
62                  1300
65                  1300
```

SUPUESTO PRÁCTICO 12: ***Resolución en el Anexo I.***

Realice las consultas que se le formulan en los siguientes puntos.

1. ¿Cuál es la plantilla que trabaja en el turno de mañana en el hospital 3? Visualizar nombre del empleado de la plantilla y su función.
2. Recuperar todo el personal de la plantilla sanitaria del hospital con código 1, indicando el nombre de cada persona, función y turno. La información se mostrará ordenada por turno, nombre y función.
3. Mostrar el nombre y oficio de todos los miembros del personal no sanitario del hospital con código 1 que no cobren comisiones. Ordenar el resultado por el salario que cobren.
4. Seleccionar las distintas especialidades que imparten los doctores del hospital 1.
5. Realizar una consulta para extraer todos los doctores del hospital con código 1. Se deberá devolver en la consulta: el nif, nombre, especialidad que ocupa y una codificación de la especialidad de acuerdo a la siguiente conversión:

   - Cardiología: código 1.
   - Psiquiatría: código 2.
   - Pediatría: código 3.
   - El resto: código 4.

   Los resultados se mostrarán ordenados por la codificación de la especialidad.
6. Realizar una consulta que nos muestre el número de salas distintas de las que consta cada hospital.
7. Confeccionar una consulta que nos muestre, agrupada por hospital y turno, las funciones que se desarrollan y el número de personas que las realizan.
8. Realizar una consulta que nos muestre agrupados por código de hospital, el conteo del número de enfermos de cada hospital, teniendo en cuenta que solo se tendrá que incluir en la consulta aquellos pacientes cuyo año de nacimiento sea igual o mayor a 1955. Además, la consulta solo mostrará aquellos hospitales cuyo número de enfermos sea mayor o igual a 10. La columna con el conteo de enfermos se deberá llamar "Num.Enfermos".

9. Diseñar una consulta que muestre el organigrama del personal no sanitario. Para cada miembro se incluirá la siguiente función: LPAD('',2*(LEVEL-1)), concatenada con el nombre de la persona, '-' el oficio que realiza el mismo, ' NIVEL(', el nivel en el que se encuentra dentro del organigrama del hospital y ')'. A todo esto se le denominará con el alias "Organización hospital".
10. Diseñar una consulta que muestre el código del hospital, código del departamento, suma del salario en euros con el nombre "Sal. euros.", suma del salario en pesetas con el nombre "Sal. Pts.", y suma de las comisiones en euros que se llamará "Comisión eur.", acumulando los valores por hospital y departamento, siempre y cuando la fecha de alta de los empleados no asalariados sea mayor o igual al 1 de enero de 2002, el código de hospital sea 1,3,5 o 7 y el código de departamento se encuentre entre 1 y 4. De todos los valores que se podrían mostrar solo nos interesan aquellos cuya suma de salarios en pesetas, sea mayor a las 500.000 pts., y la suma de comisiones en euros menor a 2.000 €. Como valor de referencia para la conversión a pesetas tomaremos 166.386.
11. Diseñar una consulta que permita mostrar el nombre de los hospitales que poseen menos camas que la suma de camas que posee cada una de las salas del mismo.
12. Diseñar una consulta que nos muestre el nombre y apellidos de todos aquellos enfermos que hayan asistido a más de 2 hospitales distintos. El título de las columnas será: nombre, apellidos y nº asistencias.
13. Diseñar una consulta que nos muestren los distintos nombres y apellidos de todos aquellos enfermos cuyos apellidos comiencen por las letras A o M, que hayan asistido al hospital con código 1 y que además hayan asistido también al hospital con código 2.
14. Diseñar una consulta que nos muestre los distintos nombres y apellidos de doctores que han sido fieles a la institución que les paga y consecuentemente solo han trabajado en un hospital a lo largo de su carrera.
15. Diseñar una consulta que nos muestre el nombre, apellidos y nif de todos los miembros que trabajan en el hospital número 1 (doctores, plantilla no sanitaria y plantilla sanitaria), ordenando el listado por apellidos, nombre y nif.

16. Diseñar una consulta que nos muestre el nombre, teléfono del hospital número 3, así como el nombre de las salas del mismo, ordenando el listado por el nombre de la sala. El nombre de las respectivas columnas a mostrar será: Hospital, Tlf., y Sala.
17. Diseñar un listado que nos muestre para cada miembro de la plantilla sanitaria que trabaje en el turno de noche (N) y que sea enfermo, el nif, nombre, salario que cobra, nombre de sala en la que trabaja, y nombre del hospital en el que trabaja. El título de las columnas será: Nif, Nombre, Salario, Nombre Sala y Nombre Hospital. El listado se ordenará por el nombre del miembro de la plantilla, nombre del hospital y nombre de la sala.
18. Diseñar un listado que nos muestre el nombre de los departamentos en los que están trabajando más de 5 personas. El título de las columnas será: Departamento y Nº Trabajadores. El listado se ordenará por el nombre del departamento.
19. Diseñar un listado que nos muestre todo el personal sanitario incluidos los doctores que trabajan en el hospital número 4. Para este listado se mostrará la siguiente información:

    - En el caso de que sea un miembro de la plantilla sanitaria: nombre del hospital, nombre de la sala, nombre de la persona, función que tiene y el literal fijo '(SANITARIO)'. Utilizar como alias de columnas las siguientes: Hospital, Sala, Nombre, Función, Tipo de empleado.
    - En el caso de que sea un doctor: nombre del hospital, nulo, nombre del doctor, especialidad y el literal fijo '(DOCTOR)'. Utilizar como alias de columnas los que se han indicado en el punto anterior.

    Todo el listado se ordenará por el nombre del hospital, nombre de la sala y nombre del miembro sanitario.

ACTUALIZACIÓN DE DATOS 9

ACTUALIZACIÓN DE INFORMACIÓN (UPDATE)

El comando UPDATE permite realizar todo tipo de actualizaciones sobre la información contenida en las distintas tablas de la base de datos.

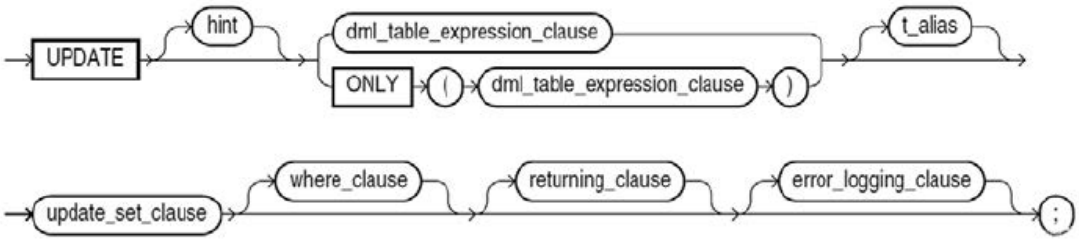


Fig. 9-1 Sintaxis para la actualización de una tabla.

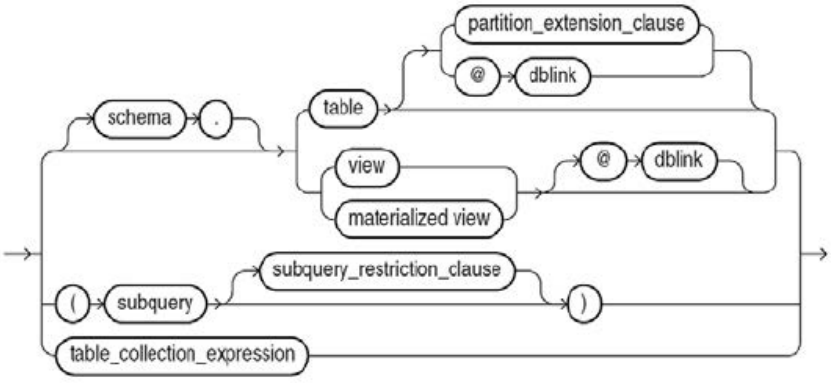


Fig. 9-2 Sintaxis del elemento DML_TABLE_EXPRESSION_CLAUSE para la actualización de una tabla.

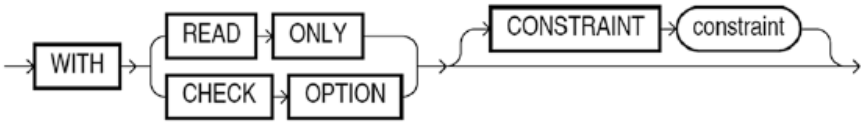


Fig. 9-3 Sintaxis del elemento SUBQUERY_RESTRICTION_CLAUSE del elemento DML_TABLA_EXPRESSION_CLAUSE.

Fig. 9-4 Sintaxis del elemento WHERE_CLAUSE para la actualización de una tabla.

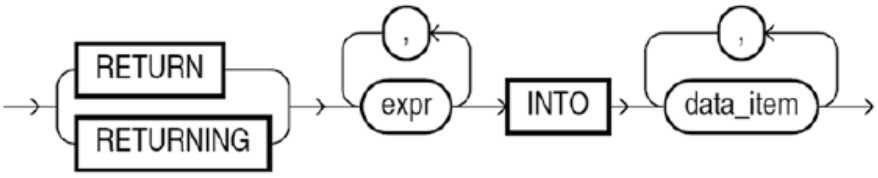


Fig. 9-5 Sintaxis del elemento RETURNING_CLAUSE para la actualización de una tabla.

ACTUALIZACIÓN GENERAL

Permite modificar de una sola vez todos los registros de una tabla.

Ejemplo

```
UPDATE EMP
SET EMPNO = 10;
```

Esta instrucción de actualización modificaría el contenido de todas las filas de la tabla `emp`, cambiando el valor del campo `empno` a valor 10.

ACTUALIZACIÓN CON CRITERIOS

Permite modificar registros de nuestra tabla indicando criterios de actualización, al igual que se pueden indicar criterios de consulta. Es decir, la parte del WHERE de este comando se puede componer de la misma manera que la parte WHERE de una consulta como las que vimos en el capítulo anterior.

Ejemplo

```
UPDATE emp
SET
   Empno = 10
WHERE
   Apellido = 'Higueras D.';
```

Este ejemplo actualizaría todas las filas de nuestra tabla emp, que cumplan que el apellido sea 'Higueras D.' cambiando el valor del campo empno a 10.

ACTUALIZACIÓN MEDIANTE SELECT

Permite modificar registros de nuestra tabla indicando o no criterios de actualización, pero los datos no se indican de forma estática (mediante literales numéricos o de carácter), sino mediante una consulta.

Ejemplo

```
UPDATE emp a
SET deptno = (SELECT deptno FROM dept
              WHERE loc = 'BOSTON'),
(sal, comm) = (SELECT 1.1*AVG(sal), 1.5*AVG(comm)
               FROM emp b
               WHERE a.deptno = b.deptno)
WHERE
deptno IN (SELECT deptno FROM dept
           WHERE loc = 'DALLAS' OR loc = 'DETROIT');
```

Este ejemplo nos actualizaría todas las filas de nuestra tabla emp que cumplan que el campo deptno se encuentre dentro de uno de los valores que devuelve la subconsulta por departamentos que estén localizados en 'DALLAS' o 'DETROIT'.

Los datos a actualizar son en primer lugar el propio campo deptno, asignándole el valor que nos devuelva la subconsulta de departamentos localizados en Boston (teniendo en cuenta que esta consulta sabemos que solo nos devolverá un valor porque en caso contrario fallaría la actualización).

Y por último actualizamos los campos `sal` y `comm` a los valores de la media aritmética de salarios por 1.1, y la media aritmética de comisión por 1.5, respectivamente.

ACTUALIZACIÓN DEVOLVIENDO VALORES

Este formato del comando UPDATE permite modificar registros de nuestra tabla indicando o no criterios de actualización, y además nos devuelve valores a variables de PL/SQL.

Ejemplo

```
UPDATE emp
SET job ='MANAGER'
,sal = sal + 1000
, deptno = 20
WHERE ename = 'JONES'
RETURNING sal*0.25, ename, deptno
INTO :bnd1, :bnd2, :bnd3;
```

Este ejemplo nos actualizaría los campos `job, sal` y `deptno` de la tabla `emp`, para aquellos registros cuyo `ename` sea 'JONES', devolviendo sobre las variables PL/SQL `bnd1, bnd2 y bnd3`, los valores de `sal*0.25, ename` y `deptno`, respectivamente.

BORRADO DE DATOS 10

BORRADO DE INFORMACIÓN (DELETE)

El comando DELETE permite realizar todo tipo de borrados sobre la información que existe en las tablas.

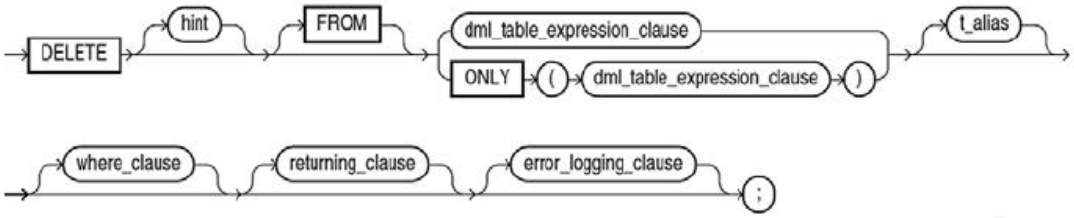


Fig. 10-1 Sintaxis para borrado de información de una tabla.

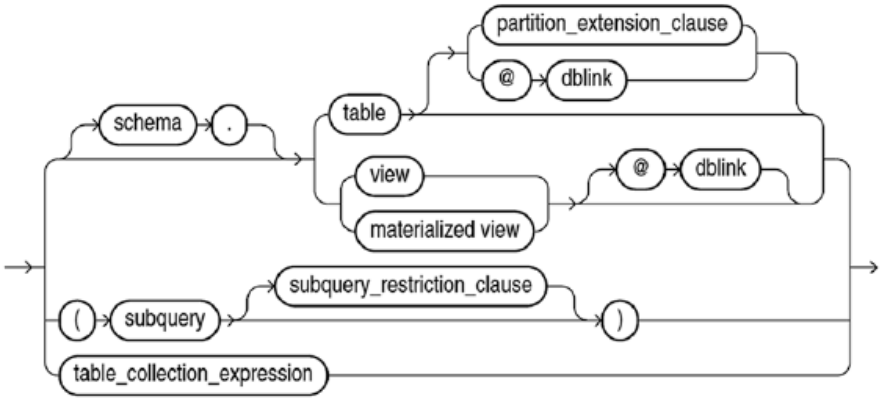


Fig. 10-2 Sintaxis del elemento DML_TABLE_EXPRESSION_CLAUSE de la actualización de datos.

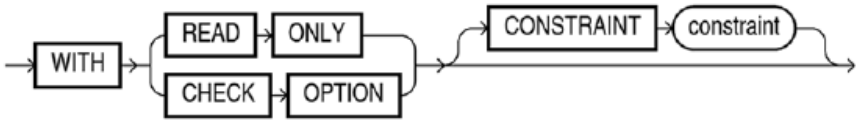


Fig. 10-3 Sintaxis del elemento SUBQUERY_RESTRICTION_CLAUSE del elemento DML_TABLE_EXPRESSION_CLAUSE.

Fig. 10-4 Sintaxis del elemento WHERE_CLAUSE de la actualización de datos.

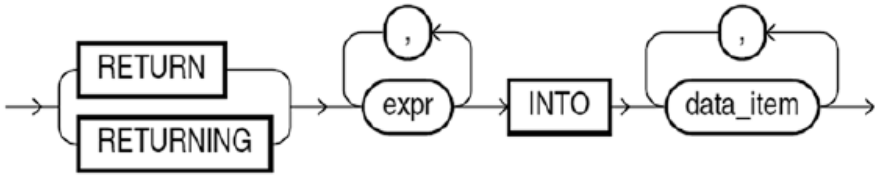


Fig. 10-5 Sintaxis del elemento RETURNING_CLAUSE de la actualización de datos.

BORRADO GENERAL

Permite borrar de una sola vez todos los registros de la tabla.

Ejemplo

```
DELETE [FROM] emp;
```

Esta instrucción borraría todas las filas de la tabla `emp`. La cláusula FROM es opcional.

BORRADO CON CRITERIOS

Permite borrar registros de nuestra tabla indicando criterios al igual que se pueden indicar criterios de consulta. Es decir, la parte del WHERE de este comando se puede componer de la misma manera que la parte WHERE de una consulta mediante SELECT.

Ejemplo

```
DELETE [FROM] emp
WHERE Emp_no = 10;
```

Nos borraría todas las filas de nuestra tabla `emp` cuyo campo `emp_no` sea igual a 10.

BORRADO DEVOLVIENDO VALORES A VARIABLES

Permite borrar registros de nuestra tabla, indicando o no criterios, devolviéndonos además valores a variables de PL/SQL.

Ejemplo

```
DELETE FROM emp
WHERE job = 'SALESMAN'
AND COMM < 100
RETURNING sal INTO :bnd1;
```

Esta instrucción borra todos los registros de la tabla `emp` cuyo `job` sea 'SALESMAN' y `COMM` sea menor a 100, devolviendo el campo `sal` sobre la variable PL/SQL `bnd1`.

CERTIFICACIONES DE ORACLE 11

INTRODUCCIÓN

Las certificaciones de Oracle están reconocidas por la industria, y son la mejor carta de presentación para ayudarle a tener éxito en su carrera profesional dentro del sector de las Tecnologías de la Información.

Las certificaciones de Oracle son un certificado que prueba la educación y experiencia recibidas en los diversos productos de Oracle, y pueden acelerar su desarrollo profesional, mejorar su productividad y su credibilidad.

CERTIFICACIONES DE ORACLE DISPONIBLES

Toda la información sobre las certificaciones de Oracle la puede encontrar actualizada en el siguiente enlace:

```
http://education.oracle.com/pls/web_prod-plq-
dad/db_pages.getpage?page_id=39
```

Identificar el camino correcto para la certificación deseada

La experiencia técnica es provechosa y recomendada a la hora de afrontar un examen de certificación. Una combinación de experiencia de uso del producto, la propia experiencia técnica y la certificación asegurarán más amplias oportunidades de carrera para usted.

Elija su camino preferido, y repase las exigencias deseadas para afrontar la certificación que le gustaría obtener. Las exigencias de certificación varían para cada certificación. Acceda a la información completa sobre la cCertificación seleccionada desde la propia página de Certificaciones de Oracle, donde se informa de todos los requisitos y el temario necesario para afrontar el examen de certificación.

Prepararse para el examen

Existe un buen número de exámenes de muestra para la preparación de las distintas Certificaciones de Oracle que se pueden obtener online, introduciendo el código de examen en un buscador de Internet.

También, Oracle pone a disposición de los candidatos a obtener una certificación, un conjunto de recursos para la preparación a sus certificaciones. Pueden acceder a estos recursos en el siguiente enlace de Internet:

```
http://education.oracle.com/pls/web_prod-plq-
dad/db_pages.getpage?page_id=618
```

Registrarse y completar el examen

Visite la página web de registro de exámenes para ver las instrucciones sobre la petición de una cita en un Centro examinador de Oracle (Oracle Testing Center), o a través de un Centro examinador autorizado Pearson VUE (Pearson VUE Authorized Test Center). También es posible realizar el examen de certificación online. Toda la información la pueden encontrar en el enlace siguiente:

```
http://education.oracle.com/pls/web_prod-plq-
dad/db_pages.getpage?page_id=156
```

Completar los requerimientos de tu certificación

La dirección que usted facilite a Pearson VUE, en el momento del registro del examen de certificación, será usada por Oracle para enviarle el Kit del Programa de Éxito de la Certificación de Oracle. Asegúrese de mantener al día los datos del domicilio y correo electrónico. Si necesita cambiar cualquier información de contacto, puede visitar la página www.pearsonvue.com/oracle.

Además, confirme que usted completó todos los requerimientos de la certificación correspondientes al número de identificación del Test. Puede confirmar su histórico de requerimientos de certificación de manera online, en el enlace certview.oracle.com.

Dentro de las 6-8 semanas posteriores a pasar todos los exámenes requeridos, usted recibirá el Kit del Programa de Éxito de la Certificación de Oracle.

PREGUNTAS TIPO EXAMEN DE CERTIFICACIÓN SQL

En esta sección se ha recopilado de diversas fuentes públicas para la preparación de los exámenes de Certificación de Oracle una serie de preguntas que le pueden orientar y servir de práctica a la hora de afrontar un examen de certificación en el lenguaje SQL.

En concreto, actualmente existen los siguientes códigos de examen de certificación en SQL:

- 1Z0-061 (Oracle Database 12c: SQL Fundamentals)
- 1Z0-051 (Oracle Database 11g: SQL Fundamentals I)
- 1Z0-117 (Oracle Database 11g Release 2: SQL Tuning)
- 1Z0-047 (Oracle Database SQL Expert)
- 1Z0-071 (Oracle Database 12c SQL)

Es importante reseñar que todos los exámenes se realizan en inglés, por lo que debe de estar familiarizado con la terminología técnica en dicho idioma.

Las preguntas aquí detalladas en algunos casos se han traducido al castellano para facilitar su comprensión, pero no debe confundir al candidato que se presente al examen, dado que no las encontrará así.

La resolución a las preguntas que se plantean a continuación las podrá encontrar en el Anexo II de este libro.

Cuestión 1

Se propone gestionar todos los empleados que reciben un incremento del 10% en el salario y una comisión que es un 20% mayor al salario incrementado. Se pide escribir una consulta que visualice adecuadamente el salario, la comisión y la compensación total.

Evalúa las siguientes sentencias SQL:

```
1. SELECT sal * 1.1 "Salary", sal * 1.1 * .2 "Commission", sal
   * 1.1 + sal * 1.1 * .2 "Compensation"      FROM    emp;
2. SELECT sal * 1.1 "Salary", sal * 1.1 * .2 "Commission", (sal
   * 1.1) + (sal * .2) "Compensation"      FROM    emp;
3. SELECT sal * 1.1 "Salary", sal * .2 "Commission", sal * 1.1
   + sal * 1.1 * .2 "Compensation"      FROM    emp;
```

¿Cuál es el resultado de estas sentencias?

A. 2 sentencias devuelven el resultado deseado.
B. Solo una sentencia devuelve el resultado deseado.
C. Solo 2 sentencias devuelven el resultado deseado.
D. Solo 3 sentencias devuelven el resultado deseado.
E. Ninguna de las sentencias devuelve el resultado deseado.

Cuestión 2

¿Qué sentencia relacionada con la actualización de una tabla sin cláusula WHERE es verdadera?

A. La sentencia no se ejecutará.
B. Solo las filas especificadas serán actualizadas.
C. Se actualizarán todas las filas de la tabla.
D. La sentencia se ejecutará pero no provocará ningún cambio.

Cuestión 3

La tabla PRODUCT contiene las siguientes columnas:

```
ID          NUMBER(7)        PK
SALE_PRICE  NUMBER(7,2)
```

Evalúa las siguientes sentencias SQL:

```
1.  SELECT  MAX(sale_price), MIN(sale_price), AVG(sale_price)
    FROM product;
2.  SELECT  ROUND(MAX(sale_price),2), ROUND(MIN(sale_price),2),
    ROUND(AVG(sale_price),2)
    FROM          product
    GROUP BY      sale_price;
```

¿En qué se diferencian los resultados?

A. Una de las sentencias generará un error.
B. La consulta 2 solo mostrará una fila como resultado y la consulta 1 podría mostrar más de una fila como resultado.
C. La consulta 1 mostrará 3 valores, mientras que la consulta 2 mostrará 3 valores por cada SALE PRICE.
D. La consulta 1 mostrará un resultado por cada SALE PRICE, mientras que la consulta 2 mostrará un resultado por cada PRODUCT.

Cuestión 4

Revisa la siguiente sentencia SQL:

```
SELECT  department "Departments", MAX(salary) "Top Salaries"
FROM          employee
WHERE   department IN(200, 300, 400)
GROUP BY      Departments
HAVING  MAX(salary) > 60000;
```

Esta sentencia falla cuando se ejecuta. ¿Qué cambio corregiría el problema?

A. Quitar la función de agrupamiento de la cláusula HAVING.
B. Añadir la condición "MAX(salary) > 60000" a la cláusula WHERE.
C. Reemplazar el alias de la columna en la cláusula GROUP BY por el nombre de la columna.
D. Añadir la función de agrupamiento utilizada en la SELECT dentro de la cláusula GROUP BY.

Cuestión 5

Dada la siguiente información sobre la tabla EMPLOYEE.

EMPLOYEE

Column Name	EMPLOYEE_ID	NAME	JOB	MANAGER	DATE_HIRED	SALARY	BONUS	DEPTARTMENT_ID
Key Type	PK							
Nulls/Unique	NN, U				NN			NN
FK Table								
FK Column								
Datatype	NUM	VARCHAR2	VARCHAR2	NUM	DATE	NUM	NUM	NUM
Length	6	20	9	6		7,2	7,2	2

Indicar qué sentencia SQL se ejecutaría correctamente:

```
A. CREATE FORCE VIEW last_first_vu
   AS SELECT name ||', '|| job "Employee & Position"
   FROM employee
   WHERE deptartment_id = 30
B. CREATE FORCE VIEW last_first_vu
   AS SELECT name ||', '|| job "Employee & Position",
   deptartment_id
   FROM employee
   ORDER BY "Employee & Position";
C. CREATE FORCE VIEW last_first_vu
   AS SELECT name ||', '|| job "Employee & Position",
   deptartment_id "department"
   FROM employee
   WHERE deptartament = 10;
D. CREATE FORCE VIEW last_first_vu
   AS SELECT name ||', '|| job "Employee & Position",
   deptartment_id "department"
   FROM employee
   ORDER BY name
   GROUP BY deptartment_id;
```

Cuestión 6

¿Cuál de las sentencias que aparecen utilizarías para borrar la restricción de clave primaria EMPLOYEE_ID_PK, y todas las restricciones dependientes de la tabla EMPLOYEE?

```
A. ALTER TABLE employee
   DROP PRIMARY KEY CASCADE;
```

```
B. ALTER TABLE employee
   DELETER PRIMARY KEY CASCADE;
C. MODIFY TABLE employee
   DROP CONSTRAINT employee_id_pk CASCADE;
D. ALTER TABLE employee
   DELETE PRIMARY KEY employee_id_pk CASCADE;
```

Cuestión 7

Se quieren visualizar los nombres de empleados que contengan una "a" como segunda letra. ¿Cuál de las sentencias que aparecen utilizarías?

```
A. SELECT name
   FROM employee
   WHERE name LIKE '_a%'
B. SELECT lower(name)
   FROM employee
   WHERE name = '_A%'
C. SELECT name
   FROM employee
   WHERE lower(name) LIKE '_A%'
D. SELECT name
   FROM employee
   WHERE name = '_a%'
   OR         name = '_A%'
```

Cuestión 8

Revisa la siguiente sentencia SQL:

```
SELECT  ename, empno, sal
FROM     emp
WHERE  deptno = (SELECT  deptno
                 FROM     dept
                 WHERE  UPPER(loc) = UPPER('&loc'))
```

Cuando ejecutas esta sentencia, qué podría ocurrir. (Elegir 2 respuestas).

A. La sentencia se ejecuta correctamente si la columna LOC en la tabla DEPT tiene valores únicos.
B. La sentencia falla porque no se puede usar un símbolo & en una subquery de una sentencia SELECT.

C. La sentencia falla si la subquery devuelve más de una fila y se pretende comparar mediante igualdad con el campo DEPTNO.
D. La sentencia se ejecuta correctamente, pero no devuelve los resultados esperados a causa de usar dos veces la función UPPER.

Cuestión 9

¿Bajo qué circunstancias se debería crear un índice en una tabla?

A. Cuando la tabla es pequeña.
B. Cuando la tabla se actualiza frecuentemente.
C. Cuando los valores de una columna son estáticos y contienen un rango pequeño de valores.
D. Cuando se utilizan 2 columnas en una cláusula WHERE de una SELECT mediante un Join.

Cuestión 10

Revisa la siguiente sentencia SQL:

```
SELECT  ename, emp_number, salary
FROM     employee
WHERE  dept_number = (SELECT  dept_number
                      FROM    department
                      WHERE location IN('CHICAGO',
                                        'ATLANTA'))
```

¿Por qué puede devolver un error esta sentencia?

A. Porque una subquery de múltiples filas devuelve una fila.
B. Porque una subquery de múltiples columnas devuelve una columna.
C. Porque una subquery de una única fila devuelve más de una fila.
D. Porque una query de múltiples filas usa una subquery de una única fila.
E. Porque una query de una única fila usa una subquery de múltiples filas que devuelve solo una fila.

Cuestión 11

La tabla EMPLOYEE contiene las columnas siguientes:

```
FIRST_NAME        VARCHAR2(25)
```

```
LAST_NAME          VARCHAR2(25)
JOB                VARCHAR2(25)
SALARY             NUMBER(7,2)
DEPT_ID            NUMBER(3)
```

Se necesitan visualizar "first_name" y "last_name" de aquellos empleados con los siguientes datos:

1. El "last_name" solo puede ser: Brown, Chan, o Lindsey.
2. El "job" solo puede ser: Manager, Technician o Clerk.
3. El "salary" solo puede ser: > 30000.

Alguien nos presenta esta SQL para resolver el enunciado:

```
SELECT   first_name, last_name
FROM     employee
WHERE    UPPER(last_name) IN ('BROWN', 'CHAN', 'LINDSEY')
AND      UPPER(job) IN ('MANAGER', 'TECHNICIAN', 'CLERK')
AND      salary <= 30000;
```

¿Qué empleados se visualizarían?

A. Aquellos que cumplan 1 solo requerimiento de los indicados.
B. Aquellos que cumplan 2 requerimientos de los indicados.
C. Aquellos que cumplan los 3 requerimientos indicados.
D. Aquellos que no cumplan ninguno de los requerimientos.

Cuestión 12

¿Cuál de las sentencias SQL es una query con Join de igualdad entre 2 tablas?

```
A. SELECT   region.region_name, employee.salary
   FROM     region, employee
   WHERE    region.id = employee.region_no
B. SELECT   region.region_name, employee.salary
   FROM     region, employee
   WHERE    region.id = employee.region_no(+)
C. SELECT   region.region_name, employee.salary
   FROM     region, employee
   WHERE    employee.salary BETWEEN region.avg_salary AND
   region.max_salary
D. SELECT   region.region_name, employee info.last_name
   FROM     employee region, employee info
   WHERE    employee info.id >= region.manager_id
```

Cuestión 13

La tabla SALE contiene estas columnas:

```
ID          NUMBER(9)   PK
SALE_DATE   DATE
```

Se necesita crear un fichero script de sentencias SQL que solicite al usuario introducir un id. La fecha de venta para el número id proporcionado debería ser actualizada con la fecha del día.

¿Cuál de los scripts SQL*Plus usarías para obtener los resultados deseados?

```
A. UPDATE   sale
   SET      sale_date = sysdate
   /
B. UPDATE   sale
   SET      sale_date = sysdate
   WHERE    id = &id
   /
C. UPDATE   sale
   SET      sale_date = &sysdate
   WHERE    id = &id
D. UPDATE   sale(sale_date)
   SET      sale_date = sysdate
   WHERE    id = _&id
```

Cuestión 14

¿Cuál de las query usarías para visualizar los nombres de todas las tablas a las que tienes acceso?

```
A. SELECT table_name
   FROM user_tables;
B. SELECT table_name
   FROM all_user_tables;
C. SELECT tname
   FROM tab
   WHERE tabtype = 'TABLE';
D. SELECT object_name
   FROM all_objects
   WHERE object_type = 'TABLE';
```

Cuestión 15

David works as a Database Administrator for uCerty Inc. The company uses an oracle database. The database contains two tables, named Departments and Employees. The DepartmentID column of the Employees table references the DepartmentID column of the Departments table. The Departments and Employees tables contain the following records:

Departments:

DepartmentID	DepartmentName
10	Administration
20	Marketing
30	Finance
40	Human Resources

Employees:

EmployeeID	EmployeeName	DepartmentID	ManagerID	JobID	Salary
101	David	20	120	SA_REP	14000
102	Sam	10	105	CLERK	12500
103	Andrew	20	120	FIN_ADMIN	14200
104	Adrian	30	108	MAR_CLERK	12500
105	Maria	30	108	FIN_ADMIN	15000
106	Tracy	40	110	AD_ASST	13000
108	Kate	30	110	FIN_DIR	16500
110	Anne	40	120	EX_DIR	18000
120	Fran	20	110	SA_DIR	16500

David writes the following statement:

```
DELETE FROM Departments
WHERE DepartmentID = 40;
```

What will be the result of the statement when it is executed?

A. The statement will fail because column names are no specified in the DELETE statement.
B. All records associated with department ID 40 will be deleted from both the Departments and Employees tables.

C. The statement will fail because both the Employees and Departments tables contain records associated with department ID 40.
D. The record associated with department ID 40 will be deleted from the Departments table.
E. The records associated with department ID 40 will be deleted from the Employees table.

Cuestión 16

You work as a Database Administrator for UCertify INC. The company uses an Oracle database. The database contains two tables, named Employees and Departments. The DepartmentID column of the Employees table references the DepartmentID column (PRIMARY KEY) of the Departments table. The database also contains a view named EmpView that is based on the Employees and Departments tables. The view contains three columns, named EmployeeID, EmployeeName, and DepartmentName. You want to add a new column named ManagerID to the view. Which of the following statements will you use to accomplish this?

```
A. ALTER VIEW EmpView AS
   SELECT EmployeeID, EmployeeName, DepartmentName, ManagerID
   FROM Employees e, Departments d
   WHERE e.DeparmentID = d.DepartmentID;
B. CREATE OR REPLACE VIEW EmpView AS
   SELECT EmployeeID, EmployeeName, DepartmentName, ManagerID
   FROM Employees e, Departments d
   WHERE e.DeparmentID = d.DepartmentID;
C. MODIFY VIEW EmpView (ADD ManagerID NUMBER);
D. ALTER VIEW EmpView (ADD ManagerID NUMBER);
E. MODIFY VIEW EmpView AS
   SELECT EmployeeID, EmployeeName, DepartmentName, ManagerID
   FROM Employees e, Departments d
   WHERE e.DeparmentID = d.DepartmentID;
```

Cuestión 17

Samantha works as a Database Administrator for Blue Well Inc. The company uses an Oracle database. The database contains a table named Employees. Following is the structure of the table.

```
EmployeeID      NUMBER(5) PRIMARY KEY
EmployeeName    VARCHAR2(50)
DepartmentID    NUMBER(5) NOT NULL
Salary          NUMBER(7,2)
```

Samantha wants to display the identification numbers of departments the pay their employees above $10000. She also wants to display the maximum salaries paid by the departments. She writes the following SELECT statement to accomplish this:

```
SELECT DepartmentID DepID, MAX(Salary) MaxSal
FROM Employees
WHERE Salary > 10000
GROUP BY DeptID
ORDER BY MaxSal;
```

The statement generates an error on execution. Which of the following clauses of the statement is causing the error?

```
A. WHERE
B. GROUP BY
C. SELECT
D. FROM
E. ORDER BY
```

Cuestión 18

View the Exhibit and examine the structure of the LOCATIONS and DEPARTMENTS tables. You need to display all those cities that have only one department. Which query gives the correct output?

```
A. SELECT location_id, city
   FROM locations l
   WHERE 1 = (SELECT COUNT(*)
   FROM departments
   WHERE location_id = l.location_id);
B. SELECT location_id, city
   FROM locations WHERE EXISTS (SELECT COUNT(*)
   FROM departments
   GROUP BY location_id HAVING COUNT(*) = 1);
C. SELECT location_id, city
   FROM locations WHERE
   1 = (SELECT COUNT(*) FROM departments
   GROUP BY location_id);
D. SELECT l.location_id, city
   FROM locations l JOIN departments d ON (l.location_id =
   d.location_id)
   WHERE EXISTS (SELECT COUNT(*)
```

```
FROM departments d
WHERE l.location_id =d.location_id);
```

Tables"Exhibit"

LOCATIONS

Name	Null?	Type
LOCATION_ID	NOT NULL	NUMBER(4)
STREET_ADDRESS		VARCHAR2(40)
POSTAL_CODE		VARCHAR2(12)
CITY		VARCHAR2(30)
STATE_PROVINCE		VARCHAR2(25)
COUNTRY_ID		CHAR(2)

-

DEPARTMENTS

Name	Null?	Type
DEPARTMENT_ID	NOT NULL	NUMBER(4)
DEPARTMENT_NAME	NOT NULL	VARCHAR2(30)
MANAGER_ID		NUMBER(6)
LOCATION_ID		NUMBER(4)

Cuestión 19

View the Exhibit and examine the structure of the EMP table. You want to display the names and salaries of only those employees who earn the highest salaries in their departments. Which two SQL statements give the required output? (Choose two.)

```
A. SELECT ename, sal
   FROM emp e
   WHERE sal = (SELECT MAX(sal)
   FROM emp
   WHERE deptno = e.deptno);
```

```
B. SELECT ename, sal
   FROM emp
   WHERE sal = ALL (SELECT MAX(sal)
   FROM emp
   GROUP BY deptno);
C. SELECT ename, sal
   FROM emp e
   WHERE EXISTS (SELECT MAX(sal)
   FROM emp WHERE deptno = e.deptno);
D. SELECT ename, sal
   FROM emp
   NATURAL JOIN (SELECT deptno, MAX(sal) sal
   FROM emp
   GROUP BY deptno);
```

Tables "Exhibit"

Name	Null?	Type
EMPNO	NOT NULL	NUMBER(4)
ENAME		VARCHAR2(10)
JOB		VARCHAR2(9)
HIREDATE		DATE
SAL		NUMBER(7,2)
DEPTNO		NUMBER(2)

Cuestión 20

Evaluate the following SQL statement:

(Note that the numbers 2,3 etc in the SQL statement are line numbers and not part of the syntax)

```
SQL> CREATE TABLE product
2 (prod_id NUMBER(3),
3 prod_name VARCHAR2(25),
4 qty NUMBER(7,2),
5 price NUMBER(10,2),
6 CONSTRAINT prod_id_pk PRIMARY KEY(prod_id),
7 CONSTRAINT prod_name_uq UNIQUE (prod_name),
8 CONSTRAINT price_nn NOT NULL (price));
```

What is the outcome of executing this command?

A. It generates an error at line 6.
B. It generates an error at line 7.
C. It generates an error at line 8.
D. It executes successfully and creates the PRODUCTS table.

Cuestión 21

Examine the structure of the DEPT table:

Name	Null?	Type
DEPTNO	NOT NULL	NUMBER(2)
DNAME		VARCHAR2(14)
LOC		VARCHAR2(13)

You successfully execute the following SQL statement:

```
SQL>CREATE TABLE emp
(emp_no NUMBER(3) PRIMARY KEY,
emp_name VARCHAR2(25) UNIQUE,
job_id VARCHAR2(10) NOT NULL,
deptno   NUMBER(2) REFERENCES dept(deptno),
salary NUMBER(10,2) CHECK (salary > 0));
```

For which columns would an index be generated automatically? (Choose all that apply)

A. EMP_NO
B. SALARY
C. JOB_ID
D. DEPT_NO
E. EMP_NAME

Cuestión 22

Sam works as a Database Administrator for Gentech Inc. The company uses an Oracle database. The database contains two tables, named Employees and Departments. Following are the structures of the tables:

Employees:

```
EmployeeID      NUMBER(5)
EmployeeName    VARCHAR2(25)
```

Departments:

```
DepartmentID    NUMBER(6)
EmployeeID      NUMBER(5)
DepartmentName  VARCHAR2(25)
```

Sam queries the Employees and the Departments tables with the following SQL statement:

```
SELECT EmployeeName, DepartmentName
FROM Employees e, Departments d
WHERE Employees.EmployeeID = Departments.EmployeeID
ORDER BY 1,2
```

The statement generates an error on execution. Which of the following clauses in the statement is causing the error?

A. WHERE
B. SELECT
C. FROM
D. ORDER BY

Cuestión 23

Tom works as a Database Administrator for Tech Mart library. The library uses an Oracle database. The database contains a table named Books. Following is the structure of the table:

```
BookID          NUMBER(4)
BookTitle       VARCHAR2(25)
Price           NUMBER(5,2)
```

```
PurchaseDate     DATE
AuthorName       VARCHAR2(30)
```

Tom wants to display the titles of books that meet the following criteria:

- Purchased before January 17, 2003
- Price is either than $500 or greater than $1000

He also wants to sort book titles by their date of purchase, starting with the most recently purchased book. Which of the following statements will he use to accomplish this?

```
A. SELECT BookTitle
   FROM Books
   WHERE (Price < 500 or > 1000) AND (PurchaseDate < '17-
   JAN-2003')
   ORDER BY PurchaseDate DESC;
B. SELECT BookTitle
   FROM Books
   WHERE (Price IN (500,1000) AND (PurchaseDate < '17-JAN-
   2003')
   ORDER BY PurchaseDate ASC;
C. SELECT BookTitle
   FROM Books
   WHERE (Price < 500 or Price > 1000) AND (PurchaseDate <
   '17-JAN-2003')
   ORDER BY PurchaseDate DESC;
D. SELECT BookTitle
   FROM Books
   WHERE (Price BETWEEN 500 AND 1000) AND (PurchaseDate <
   '17-JAN-2003')
   ORDER BY PurchaseDate;
```

Cuestión 24

Which of the following statements about a role are true?

Each correct answer represents a complete solution. Choose all that apply.

A. A role is created using the CREATE ROLE statement.
B. A role is a named group of related privileges, which can only be assigned to users.
C. A role can comprise a maximum of one hundred privileges.

D. Privileges are assigned to a role by using the GRANT statement.
E. A role can be assigned to a maximum of 1000 users.
F. A user can be assigned several roles, and a single role can be assigned to several users.

Cuestión 25

Bob created the ROOM table that contains these columns:

```
ID          NUMBER(9) PK
NAME        VARCHAR2(25)
LOCATION    VARCHAR2(15)
SQ_FT       NUMBER(3)
```

Bob granted all users the INSERT privilege on the ROOM table. You inserted three records into the ROOM table without committing the changes. Bob issues this statement:

```
SELECT * FROM room;
```

Which statements regarding the visibility of records are true? (Choose all that apply).

A. Bob will be able to access the ROOM table.
B. Bob will be able to insert the same records into the ROOM table.
C. Bob will see the three records you inserted into the ROOM table.
D. Bob will NOT be able to access the ROOM table.
E. Bob will NOT see the three records you inserted into the ROOM table.
F. Bob will NOT be able to insert the same records into the ROOM table.

Cuestión 26

The SUPPLIER table contains these columns:

```
S_ID              NUMBER PK
NAME              VARCHAR2(30)
LOCATION_ID       NUMBER
ORDER_DT          DATE
ORDER_AMOUNT      NUMBER(8,2)
```

Which clauses represent valid uses of aggregate functions? (Choose all that apply.)

```
A. FROM MAX(order_dt)
B. SELECT SUM(order_dt)
C. SELECT SUM(order_amount)
D. SELECT MAX(AVG(order_amount))
E. WHERE MIN(order_amount) = order_amount
F. SELECT location_id, order_dt, MAX(order_amount)
```

Cuestión 27

Line_item

LINE_ITEM_ID	NUMBER(9)	NOT NULL, Primary Key
ORDER_ID	NUMBER(9)	NOT NULL, Primary Key, Foreign Key to ORDER_ID column of the CURR_ORDER table
QUANTITY	NUMBER(9)	

Click the Exhibit(s) button to examine the structure of the LINE_ITEM table.

You want to display order id numbers, product id numbers, and the quantity of the product ordered with these desired results:

1. The volume of the item ordered must be 50 or greater.
2. The display must be sorted from the lowest to the highest by order number and then by product number.
3. The items must belong to order numbers ranging from 1800 to 1900.

Evaluate this SQL script:

```
SELECT order_id, product_id, quantity
FROM line_item
WHERE quantity >= 50
AND order_id IN (1800,1900)
ORDER BY order_id, product_id;
```

What does the proposed solution provide?

A. One of the desired results.
B. Two of the desired results.
C. All of the desired results.
D. An error statement.

Cuestión 28

Which three statements concerning explicit data type conversions are true. (Choose three.)

A. A number value may be converted to a date value using the TO_DATE function.
B. A date value may be converted to a number value using the TO_NUMBER function.
C. A character value may be converted to a date value using the TO_DATE function.
D. A date value may be converted to a character value using the TO_DATE function.
E. A date value may be converted to a character string using the TO_CHAR function.
F. A number value may be converted to a character string using the TO_CHAR function.
G. A number value may be converted to a character value using the TO_NUMBER function.

Cuestión 29

Evaluate this SQL statement:

```
SELECT i.id_number, m.id_number
FROM inventory i, manufacturer m
WHERE i.manufacturer_id = m.id_number
ORDER BY 1;
```

Which clause prevents all the rows in the INVENTORY table from being joined to all the rows in the MANUFACTURER table?

```
A. ORDER BY 1;
B. SELECT i.id_number, m.id_number
C. FROM inventory i, manufacturer m
D. WHERE i.manufacturer_id = m.id_number
```

Cuestión 30

The STUDENT table contains these columns:

```
ID              NUMBER(9) PK
FIRST_NAME      VARCHAR2(25)
LAST_NAME       VARCHAR2(25)
ENROLL_DATE     DATE
```

You need to create a script to display a student's enrollment date and projected graduation date.

These are the desired results:

1. Prompt the user for a student id.
2. Display the student's first name, last name and date of enrollment.
3. Display the student's projected graduation date by adding four years to the enrollment date value.

Evaluate this SQL*Plus script:

```
SELECT           CONCAT(first_name,           last_name),
TO_CHAR(enroll_date, 'fmDD MONTH YYYY'), enroll_date + 4
grad_date
FROM student
WHERE id = &id
/
```

What does the proposed solution provide?

A. One of the desired results.
B. Two of the desired results.
C. All of the desired results.
D. A syntax error.

Cuestión 31

What will be the output of the following query?

```
SELECT product_name
FROM Products
WHERE price IN (SELECT price FROM Products
                WHERE product_category = 3 OR
                      product_category = 5);
```

A. It will display the name of all products having the same price as products of category 3.
B. It will display the name of all products having the same price as products of category 3 or 5.
C. It will display the name of all products having the same price as products of category 3 and 5.
D. It will display the name of all products having the same price as products of category 5.

Cuestión 32

Martin works as a Database Administrator for MTech Inc. He designs a database that has a table named Products. He wants to create a report listing different product categories. He does not want to display any duplicate row in the report. Which of the following SELECT statements will Martin use to create the report?

```
A. SELECT Product_No, Prod_Category
   FROM Products;
B. SELECT Product_No, Prod_Category
   FROM Products
   GROUP BY Product_No ORDER BY Product_No;
C. SELECT Product_No, Prod_Category
   FROM Products
   GROUP BY Product_No;
D. SELECT DISTINCT Product_No, Prod_Category
   FROM Products;
```

Cuestión 33

Martha writes the following statement to create a view:

```
1. CREATE VIEW My_View
2. AS SELECT Acc_ID, Acc_Description
3. FROM Accounts
4. WHERE Acc_ID = 457
5. ORDER BY Acc_Description;
```

Which of the following statements is correct about the view?

A. Line 4 will generate an error on execution.
B. Line 1 will generate an error on execution.
C. Line 5 will generate an error on execution.
D. There is no error.

Cuestión 34

Which of the following statements are correct?

A. A correlated subquery is evaluated for each row processed by the parent statement.

B. A correlated subquery is evaluated once and the result is used in the parent statement.
C. A correlated subquery is executed and each row from the result is evaluated against the parent statement.
D. When a correlated subquery is evaluated, the parent statement is evaluated for each row processed by the subquery.

Cuestión 35

Maria works as a Database Administrator for Blue Well Inc. She wants to give User1 the privilege to modify only the column "column1" of a table named table1, in the database. Which of the following statements will she use to accomplish this task?

```
A. GRANT UPDATE ON table1 columns(column1) TO User1;
B. GRANT UPDATE ON table1.column1 TO User1;
C. GRANT UPDATE ON table1(column1) TO User1;
D. GRANT UPDATE (column1) ON table1 TO User1;
```

Cuestión 36

Which of the following statements is used to enable PRIMARY KEY constraint on the Emp_ID column of the Employees table?

```
A. ALTER TABLE Employees
   ENABLE CONSTRAINT
   Emp_no_pk PRIMARY KEY (Emp_ID);
B. ALTER TABLE Employees
   ENABLE PRIMARY KEY;
C. ENABLE CONSTRAINT
   Emp_no_pk PRIMARY KEY (Emp_ID);
D. UPDATE TABLE Employees
   MODIFY CONSTRAINT
   Emp_no_pk PRIMARY KEY (Emp_ID);
```

Cuestión 37

Martha writes the following query to display information from the Ordinates table:

```
SELECT Ord_No, DISTINCT Ord_Name, Salary
FROM Ordinates;
```

Which of the following statements is true about this query?

A. The query will display all values for Ord_No, unique values for Ord_Name, and all values for Salary.
B. DISTINCT is not a valid keyword in Oracle.
C. The query will display all values for Ord_No, unique values for Ord_Name, and unique values for Salay.
D. The query will give an error on execution.

Cuestión 38

Patrick wants to create a sequence for primary columns of a table. He wants the sequence to start at 2000, increment by 400, and generate a maximum value of 20000. He does not want the sequence to repeat numbers after reaching the maximum value. Which of the following statements will Patrick use to create the sequence?

```
A. CREATE SEQUENCE New_Sequence
   START WITH 2000
   INCREMENT BY 400
   MAXVALUE 20000
   CYCLE;
B. CREATE SEQUENCE New_Sequence
   START WITH 2000
   INCREMENT BY 400
   MAXVALUE 20000
   NOCYCLE;
C. CREATE SEQUENCE New_Sequence
   START WITH 2000
   MAXVALUE 20000
   NOCYCLE;
D. CREATE SEQUENCE New_Sequence
   INCREMENT BY 400
   MAXVALUE 20000
   NOCYCLE;
```

Cuestión 39

Samantha works a Database Administrator for SamTech Inc. She creates a database that contains a table named Ordinate. Which of the following SQL

statements will Samantha use to display the name of all the ordinates that not have any subordinate?

```
A. SELECT Ord.Ord_Name
   FROM Ordinate Ord
   WHERE Ord.Ord_No <> (SELECT Subord.SubOrd_No
                        FROM Ordinate Subord);
B. SELECT Ord.Ord_Name
   FROM Ordinate Ord
   WHERE Ord.Ord_No NOT IN (SELECT Subord.SubOrd_No
                            FROM Ordinate Subord);
C. SELECT Ord.Ord_Name
   FROM Ordinate Ord
   WHERE Ord.Ord_No != (SELECT Subord.SubOrd_No
                        FROM Ordinate Subord);
D. SELECT Ord.Ord_Name
   FROM Ordinate Ord
   WHERE Ord.Ord_No NOT HAVING (SELECT Subord.SubOrd_No
                                FROM Ordinate Subord);
```

Cuestión 40

What is the use of index in a database?

Each correct answer represents a complete solution. Choose two.

A. Index enforces entity integrity.
B. Index reduces data storage size.
C. Index enforces referential integrity.
D. Index improves data retrieval performance.

Cuestión 41

Patrick writes and executes the following query;

```
SELECT deptno, deptname
FROM dept
WHERE deptno = 101 or deptno = 202;
```

Which of the following operators will Patrick use to replace the OR condition in the WHERE clause?

```
A. <=;
B. BETWEEN…AND…;
C. LIKE
D. IN
E. >=
```

Cuestión 42

Which of the following statements will you use to create a view named My_view?

```
A. CREATE My_view VIEW…;
B. CREATE VIEW My_view…;
C. CREATE OR REPLACE VIEW My_view…
D. VIEW My_view…
```

Cuestión 43

Which of the following statements are true about views?

Each correct answer represents a complete solution. Choose two.

A. A view can be used to restrict a user to specific columns in a table.
B. A view is used to speed up data retrieval.
C. A view is a brief description of a particular database.
D. A view represents a subset of table attributes and is designed to facilitate a particular circumstance.

Cuestión 44

What is the relationship between foreign key and primary key?

Each correct answer represents a complete solution. Choose all that apply.

A. A foreign key constraint works in conjunction with a primary key constraint to enforce referential integrity among related entities.
B. A foreign key and primary key create a link between two entities.
C. A foreign key ties attribute(s) of a entity to the primary key of another entity, for the purpose of creating a dependency.
D. There is no relationship between a primary key and a foreign key.

Cuestión 45

Allen works as a Database Administrator for SamTech Inc. He creates a database containing a table, named Employees. The Employees table contains the following data:

Emp_ID	Last_Name	First_Name	Salary	Dept_No
9	Brown	William	21000	155
6	Robert	William	20000	234
4	Peterson	David	25000	102
3	Thomas	Samantha	30000	155
2	Jackson	David	22000	234
5	Krey	Vivian	23000	233
1	Patrick	Arnold	25000	234
7	Oliver	John	31000	102

He writes the following query to retrieve records from the Employees table:

```
SELECT Dept_No, Last_Name, SUM(Salary)
FROM Employees
WHERE Salary < 30000
GROUP BY Dept_No
ORDER BY Last_Name;
```

Allen gets an error on executing the query. Which of the following is the cause of the error?

```
A. WHERE Salary < 30000
B. GROUP BY Dept_No
C. ORDER BY Last_Name
D. FROM Employees
```

RESOLUCIÓN DE SUPUESTOS PRÁCTICOS

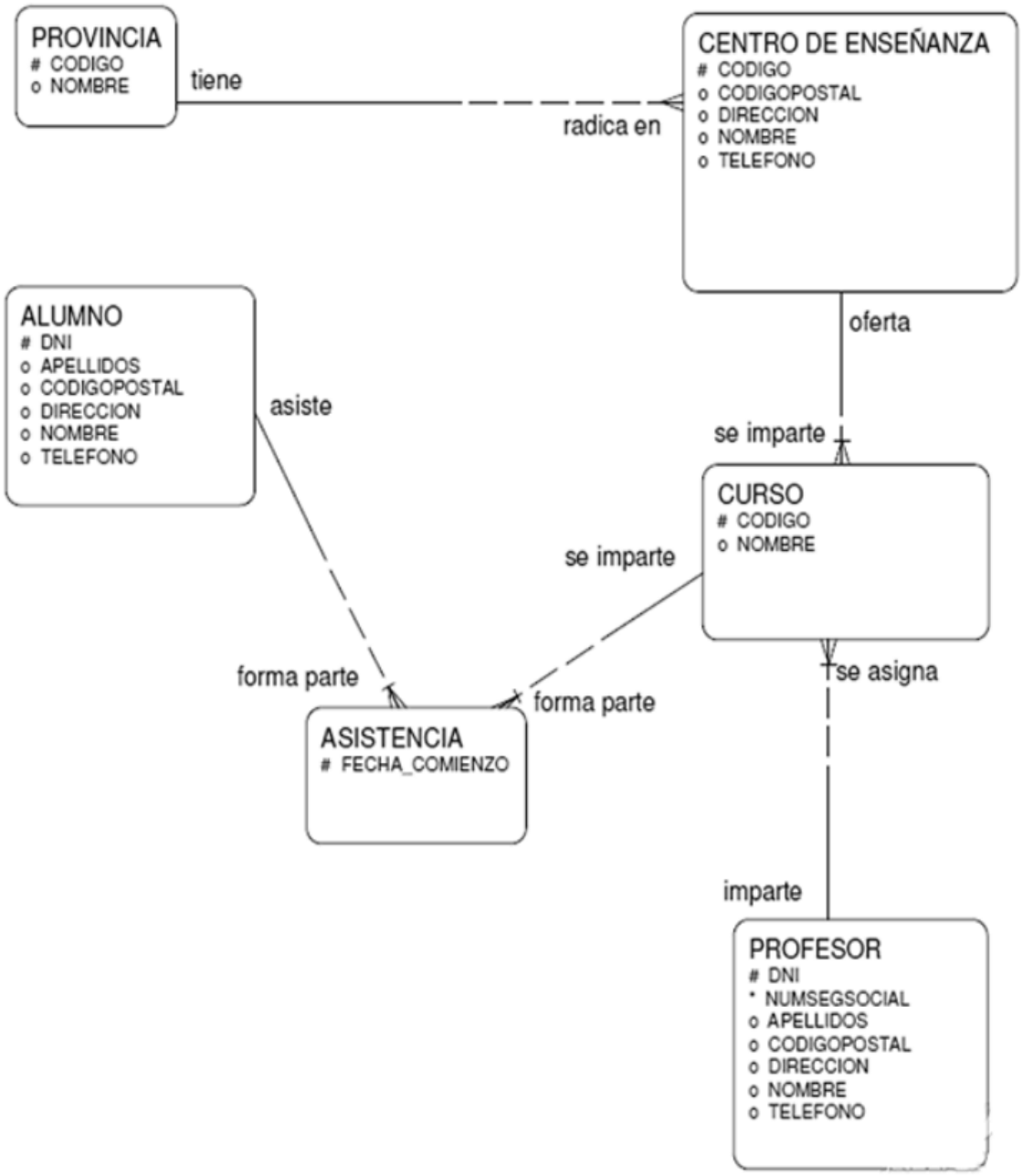
SUPUESTO PRÁCTICO 0

Diseñar un esquema de modelo relacional en el que se vean reflejados las entidades, atributos mínimos, relaciones y las claves primarias, alternativas y ajenas que considere oportunas, de acuerdo al enunciado que se especifica a continuación. Opcionalmente, podrá diseñar, previo al modelo Relacional, el modelo Entidad Relación del que se derivaría el modelo Relacional.

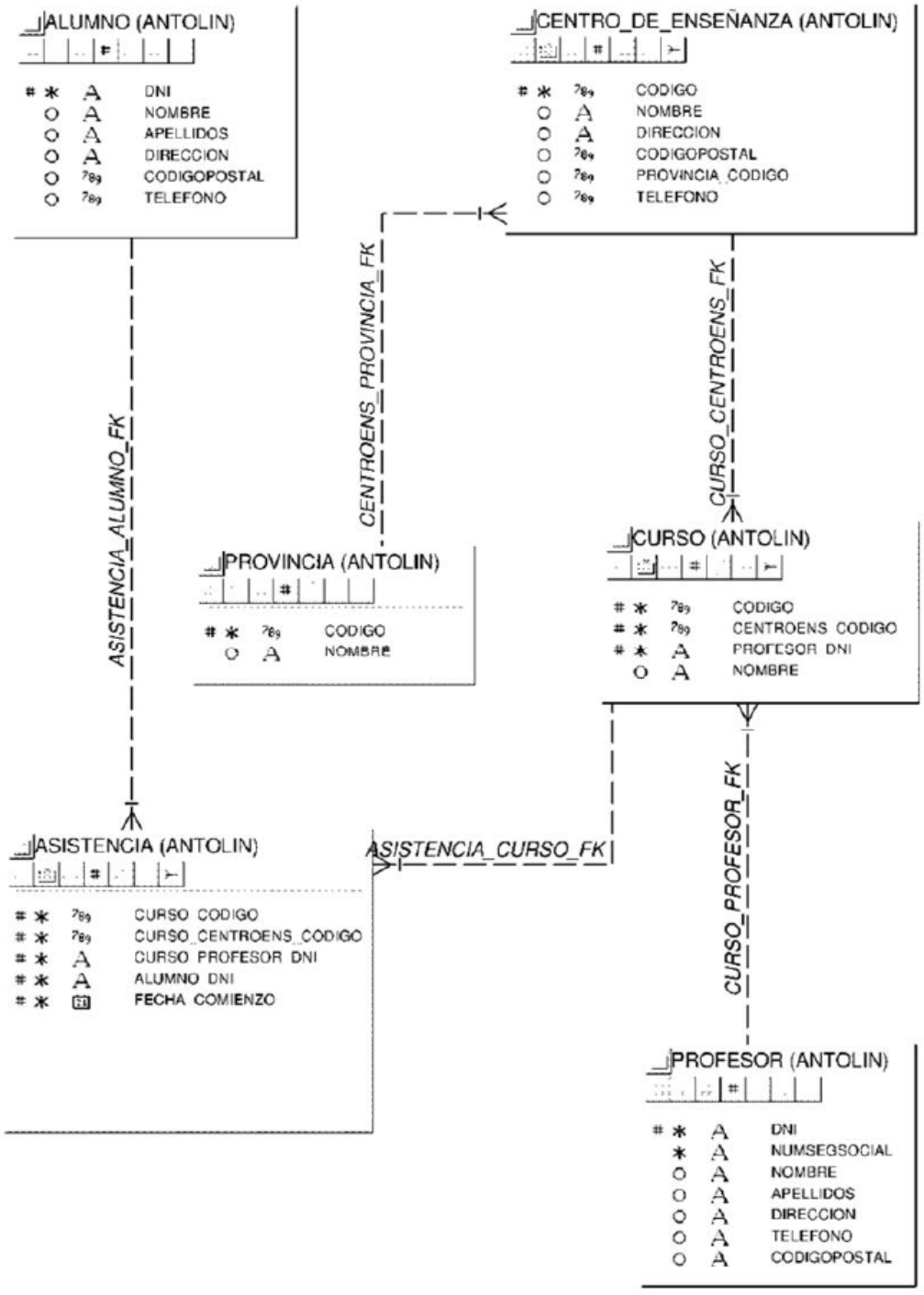
- Representar la estructura de un centro de enseñanza que posee delegaciones en varias provincias de España.
- Será necesario reflejar para el centro de enseñanza los siguientes atributos:
    - Código de 1 a 99.
    - Nombre.
    - Dirección.
    - Provincia.
    - Teléfono.
    - Código postal.
- Para las provincias habrá que reflejar:
    - Código provincia de 1 a 52
    - Nombre.
- El código del centro es unívoco y distinto para todos los centros estatales.

- En cada centro se imparten una serie de cursos con los siguientes atributos:
    - Código de curso de 1 a 99999.
    - Nombre de curso.
    - Código del centro donde se imparte.
- Cada curso es impartido por una serie de profesores cualificados con los siguientes atributos:
    - Dni.
    - Nombre.
    - Apellidos.
    - Dirección.
    - Teléfono.
    - Código postal.
    - Código de curso.
    - Código de centro.
    - Nº de la Seguridad Social.
- El código del curso puede repetirse en centros distintos, por lo que no es unívoco. Un curso de un centro solo puede ser impartido por un profesor.
- Por último, a estos cursos asisten una serie de alumnos con los siguientes atributos:
    - Dni.
    - Nombre.
    - Apellidos.
    - Dirección.
    - Teléfono.
    - Código postal.
    - Código curso.
    - Código centro.
    - Fecha de comienzo del curso.
    - Fecha de fin del curso.
- Un alumno en un mismo centro puede asistir a varios cursos, pero al mismo curso solo podrá asistir en fechas de comienzo distintas.

Resolución del supuesto (modelo entidad/relación)



Resolución del supuesto (modelo relacional)



Relación de claves de las entidades:

Entidad Alumno:

- Clave Primaria: DNI
- Clave Ajena: de DNI (Alumno) a ALUMNO_DNI (Asistencia)

Entidad Centro de Enseñanza:

- Clave Primaria: CODIGO
- Clave Ajena: de PROVINCIA_CODIGO (Centro de Enseñanza) a CODIGO (Provincia)

Entidad Provincia:

- Clave Primaria: CODIGO
- Clave Ajena: de DNI (Alumno) a ALUMNO_DNI (Asistencia)

Entidad Curso:

- Clave Primaria: CODIGO, CENTROENS_CODIGO, PROFESOR_DNI
- Clave Ajena: de CENTROENS_CODIGO (Curso) a CODIGO (Centro de Enseñanza), de PROFESOR_DNI(Curso) a DNI (Profesor)

Entidad Asistencia:

- Clave Primaria: CURSO_CODIGO, CURSO_CENTROENS_CODIGO, CURSO_PROFESOR_DNI, ALUMNO_DNI, FECHA_COMIENZO
- Clave Ajena: de CURSO_CODIGO, CURSO_CENTROENS_CODIGO,CURSO_PROFESOR_DNI (Asistencia) a CODIGO, CENTROENS_CODIGO, PROFESOR_DNI (Curso)

Entidad Profesor:

- Clave Primaria: DNI
- Clave Alternativa: NUMSEGSOCIAL

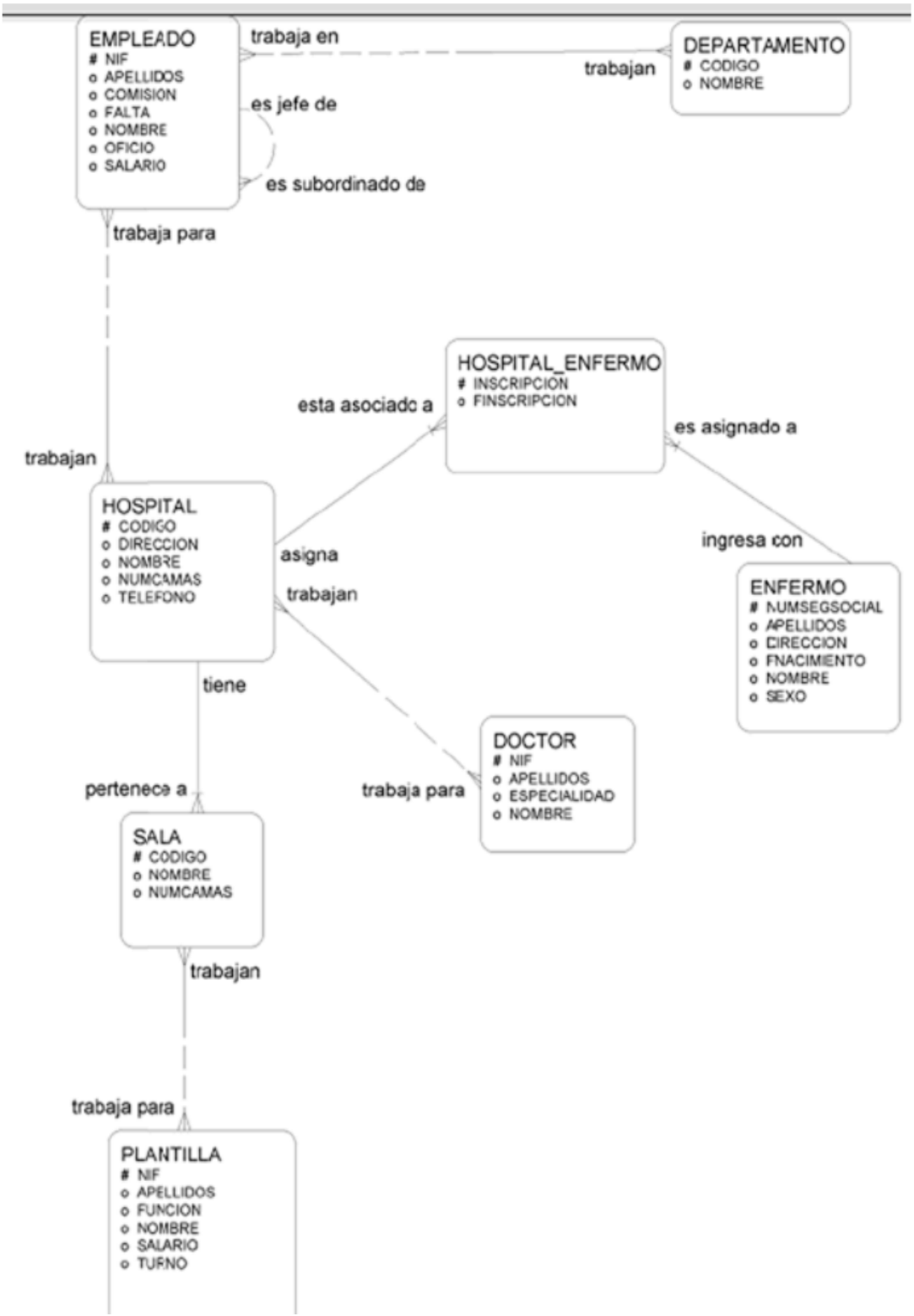
SUPUESTO PRÁCTICO 1

Extraer las entidades del supuesto con indicación de los atributos, dominios, relaciones y todas las claves que existan, diseñando un modelo Relacional y opcionalmente si se desea, diseñar previamente un modelo Entidad/Relación.

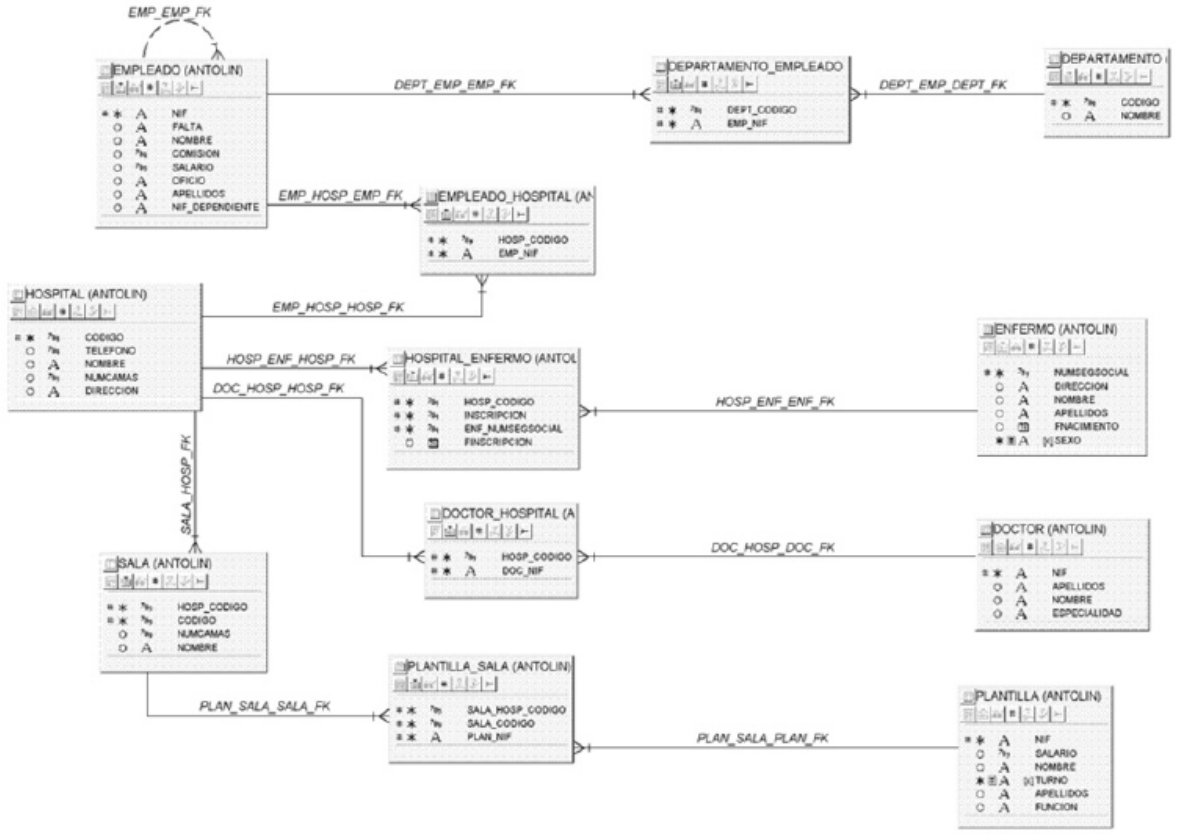
- Se necesitan reflejar los datos de los hospitales que hay en la geografía española con la siguiente información básica:
  - Un código de hospital único con independencia de la localización geográfica del centro. Este código almacenará valores de 1 a 99.
  - El nombre del hospital de 12 caracteres alfabéticos.
  - La dirección de 50 caracteres alfanuméricos.
  - El teléfono de 9 dígitos.
  - El número de camas del hospital con valores de 1 a 9999.
- A cada hospital asisten diariamente enfermos con diversas afecciones, de los que queremos reflejar la siguiente información:
  - El código del hospital al que asisten.
  - El número de inscripción del hospital en cuestión, y que será único para cada asistencia que se produzca dentro de un hospital. Los valores del número variarán entre 1 y 99.999.999.
  - Nombre de 20 caracteres alfabéticos.
  - Apellidos de 40 caracteres alfabéticos.
  - Dirección de 50 caracteres alfanuméricos.
  - Fecha de nacimiento.
  - Sexo que solo podrá ser M o F.
  - Número de la Seguridad Social de 9 dígitos.
  - Fecha de inscripción en el hospital.
- Cada hospital cuenta con una serie de salas donde se aplican tratamientos diversos. De las mismas se quiere almacenar la siguiente información:
  - Código del hospital en el que se encuentra la sala.
  - Código de la sala que será único dentro del mismo hospital y que variará de 1 a 99.
  - Nombre de la sala de 30 caracteres alfanuméricos.
  - Número de camas de las que dispone la sala y que variará de 0 a 99.
- En cada hospital trabajan una serie de doctores que aplican terapias a sus pacientes, siendo necesario que el esquema refleje datos de los mismos:
  - El código del hospital en el que trabajan o ha trabajado.
  - Nif de 9 caracteres alfanuméricos.
  - Nombre de 20 caracteres alfabéticos.
  - Apellidos de 40 caracteres alfabéticos.
  - Especialidad en la que trabaja el doctor de 20 caracteres alfabéticos.

- Aparte de los doctores, en cada hospital trabaja una plantilla sanitaria que cubre el resto de áreas de sanidad del hospital. Para cada uno de ellos se quiere reflejar:
  - El código del hospital en el que trabajan o han trabajado.
  - El código de la sala en la que trabajan o han trabajado.
  - NIF de 9 caracteres alfanuméricos.
  - Nombre de 20 caracteres alfabéticos.
  - Apellidos de 40 caracteres alfabéticos.
  - Función desempeñada en la sala donde trabaja de 20 caracteres alfabéticos.
  - Turno que tienen asignado en la sala en la que se encuentra. Solo podrá ser M,T,N.
  - Salario anual que cobran en euros (5 dígitos enteros y 2 decimales).
- Además de las salas un hospital, también consta de departamentos donde trabaja el personal no sanitario. Queremos diseñar una lista de departamentos universales que sirva para cualquier hospital, y para ello necesitamos la siguiente información:
  - Código del departamento único de 1 a 99.
  - Nombre del departamento de 20 caracteres alfabéticos.
- También existe una plantilla no sanitaria que trabaja en el hospital desempeñando otras tareas que mantienen en funcionamiento todos los servicios del hospital y que abarcan desde la dirección del hospital hasta el servicio de limpieza del mismo. Queremos incluir en nuestro esquema la siguiente información al respecto.
  - El código del hospital en el que trabajan o han trabajado.
  - NIF de 9 caracteres alfanuméricos.
  - Nombre de 20 caracteres alfabéticos.
  - Apellidos de 40 caracteres alfabéticos
  - Oficio que desempeñan de 20 caracteres alfanuméricos.
  - NIF de la persona de la que dependen funcionalmente.
  - Fecha de alta en la empresa.
  - Salario mensual en euros (4 enteros y 2 decimales)
  - Comisión en euros (4 enteros y 2 decimales)
  - Código del departamento en el que trabaja o ha trabajado.

Resolución del supuesto (modelo entidad/relación)



Resolución del supuesto (modelo relacional)



Relación de claves de las entidades:

Entidad Empleado:

- Clave Primaria: NIF
- Clave Ajena: de NIF_DEPENDIENTE (Empleado) a NIF (Empleado)

Entidad Departamento-Empleado:

- Clave Primaria: DEPT_CODIGO, EMP_NIF
- Clave Ajena: de EMP_NIF (Departamento) a NIF (Empleado)

Entidad Departamento:

- Clave Primaria: CODIGO

Entidad Hospital:

- Clave Primaria: CODIGO

Entidad Empleado-Hospital:

- Clave Primaria: HOSP_CODIGO, EMP_NIF
- Clave Ajena: de HOSP_CODIGO (Empleado-Hospital) a CODIGO (Hospital), de EMP_NIF (Empleado-Hospital) a NIF (Empleado)

Entidad Hospital-Enfermo:

- Clave Primaria: HOSP_CODIGO, INSCRIPCION, ENF_NUMSEGSOCIAL
- Clave Ajena: de HOSP_CODIGO (Hospital-Enfermo) a CODIGO (Hospital), de ENF_NUMSEGSOCIAL (Hospital-Enfermo) a NUMSEGSOCIAL(Enfermo)

Entidad Enfermo:

- Clave Primaria: NUMSEGSOCIAL

Entidad Sala:

- Clave Primaria: HOSP_CODIGO, CODIGO
- Clave Ajena: de HOSP_CODIGO (Sala) a CODIGO (Hospital)

Entidad Doctor:

- Clave Primaria: NIF

Entidad Doctor-Hospital:

- Clave Primaria: HOSP_CODIGO, DOC_NIF
- Clave Ajena: de HOSP_CODIGO (Doctor-Hospital) a CODIGO (Hospital), de DOC_NIF (Doctor-Hospital) a NIF (Doctor)

Entidad Plantilla:

- Clave Primaria: NIF

Entidad Plantilla-Sala:

- Clave Primaria: SALA_HOSP_CODIGO, SALA_CODIGO, PLAN_NIF
- Clave Ajena: de SALA_HOSP_CODIGO, SALA_CODIGO (Plantilla-Sala) a HOSP_CODIGO, CODIGO (Sala), de PLAN_NIF (Plantilla-Sala) a NIF (Plantilla)

SUPUESTO PRÁCTICO 2

Crearse un usuario mediante código SQL que tenga las siguientes características particulares de cada alumno que lo crea:

- El nombre de usuario deberá componerse de la forma que se expresa debajo, sin dejar espacios en blanco y sin utilizar símbolos especiales, solo letras alfabéticas:

```
NOMBREDEPILA+Iniciales de los apellidos
```

- La password de cada usuario será de libre elección por el alumno.

Además de crear el usuario, se le deberá dotar al mismo de capacidad suficiente para poderse conectar a la base de datos utilizando el nombre y password creados con anterioridad, más la posibilidad de crear y ejecutar procedimientos (PROCEDURE), así como crear vistas (VIEW).

Resolución del supuesto

Para la resolución de este supuesto se ha tomado como ejemplo un usuario que se llama PEPE Rodríguez Fernández y que elige como password PEPITO.

```
1. Conectarse con uno de los usuarios administradores a Oracle
   (SYSTEM, SYS).
2. SQL> CREATE USER PEPERF IDENTIFIED BY PEPITO;
3. SQL> GRANT CONNECT, RESOURCE TO PEPERF;
4. SQL> GRANT CREATE VIEW TO PEPERF;
```

SUPUESTO PRÁCTICO 3

Crear las estructuras de tablas, claves, restricciones, etc., del diagrama relacional resuelto en el Supuesto 1, por el que se diseñó la estructura de un hospital. Para ello se deberán realizar las siguientes acciones conducentes a obtener un diseño óptimo:

1. Convertir los elementos del modelo relacional a elementos de base de datos Oracle.
2. Codificar mediante sentencias SQL de Oracle la creación de los elementos necesarios para la creación de la base de datos de nuestro hospital (tablas, campos, tipos de datos, tamaños, claves, relaciones, restricciones, etc.).
3. Indicar el orden de creación de las tablas.
4. Ejecutar el código para la creación de cada uno de los elementos, conectándose con el usuario creado en el supuesto anterior.

Resolución del supuesto

```
CREATE TABLE EMPLEADO
 (NIF             VARCHAR2(9) NOT NULL
 ,FALTA           VARCHAR2(240)
 ,NOMBRE          VARCHAR2(20)
 ,COMISION        NUMBER(6,2)
 ,SALARIO         NUMBER(6,2)
 ,OFICIO          VARCHAR2(20)
 ,APELLIDOS       VARCHAR2(40)
 ,NIF_DEPENDIENTE VARCHAR2(9)
 ,CONSTRAINT EMP_PK PRIMARY KEY (NIF)
 ,CONSTRAINT EMP_EMP_FK FOREIGN KEY (NIF_DEPENDIENTE)
                        REFERENCES EMPLEADO (NIF))
/

CREATE TABLE HOSPITAL
 (CODIGO    NUMBER(2,0) NOT NULL
 ,TELEFONO  NUMBER(9)
 ,NOMBRE    VARCHAR2(12)
 ,NUMCAMAS  NUMBER(4,0)
 ,DIRECCION VARCHAR2(50)
,CONSTRAINT HOSP_PK PRIMARY KEY (CODIGO))
/

CREATE TABLE ENFERMO
 (NUMSEGSOCIAL  NUMBER(9,0) NOT NULL
 ,DIRECCION     VARCHAR2(50)
 ,NOMBRE        VARCHAR2(20)
 ,APELLIDOS     VARCHAR2(40)
 ,FNACIMIENTO   DATE
 ,SEXO          CHAR(1) NOT NULL
 ,CONSTRAINT ENF_PK PRIMARY KEY (NUMSEGSOCIAL)
 ,CONSTRAINT AVCON_1247567854_SEXO_000 CHECK (SEXO IN
('M', 'F')))
/
```

```
CREATE TABLE PLANTILLA
 (NIF       VARCHAR2(9) NOT NULL
 ,SALARIO   NUMBER(7,2)
 ,NOMBRE    VARCHAR2(20)
 ,TURNO     CHAR(1) NOT NULL
 ,APELLIDOS VARCHAR2(40)
 ,FUNCION   VARCHAR2(20)
 ,CONSTRAINT PLAN_PK PRIMARY KEY (NIF)
 ,CONSTRAINT AVCON_1247567854_TURNO_000 CHECK (TURNO
IN ('M','T','N')))
/

CREATE TABLE SALA
 (HOSP_CODIGO   NUMBER(2,0) NOT NULL
 ,CODIGO        NUMBER(2,0) NOT NULL
 ,NUMCAMAS      NUMBER(2,0)
 ,NOMBRE        VARCHAR2(30)
 ,CONSTRAINT SALA_PK PRIMARY KEY (CODIGO,HOSP_CODIGO)
 ,CONSTRAINT SALA_HOSP_FK FOREIGN KEY (HOSP_CODIGO)
                          REFERENCES HOSPITAL
(CODIGO))
/

CREATE TABLE PLANTILLA_SALA
 (SALA_HOSP_CODIGO   NUMBER(2,0) NOT NULL
 ,SALA_CODIGO        NUMBER(2,0) NOT NULL
 ,PLAN_NIF           VARCHAR2(9) NOT NULL
 ,CONSTRAINT PLAN_SALA_PK PRIMARY KEY
(PLAN_NIF,SALA_CODIGO,SALA_HOSP_CODIGO)
 ,CONSTRAINT PLAN_SALA_SALA_FK FOREIGN KEY
(SALA_CODIGO,SALA_HOSP_CODIGO) REFERENCES SALA
(CODIGO,HOSP_CODIGO)
 ,CONSTRAINT PLAN_SALA_PLAN_FK FOREIGN KEY (PLAN_NIF)
REFERENCES PLANTILLA (NIF))
/
```

```
CREATE TABLE HOSPITAL_ENFERMO
 (HOSP_CODIGO         NUMBER(2,0) NOT NULL
 ,INSCRIPCION         NUMBER(8,0) NOT NULL
 ,ENF_NUMSEGSOCIAL    NUMBER(9,0) NOT NULL
 ,FINSCRIPCION        DATE
 ,CONSTRAINT HOSP_ENF_PK PRIMARY KEY
(INSCRIPCION,HOSP_CODIGO,ENF_NUMSEGSOCIAL)
 ,CONSTRAINT HOSP_ENF_HOSP_FK FOREIGN KEY
(HOSP_CODIGO) REFERENCES HOSPITAL (CODIGO)
 ,CONSTRAINT HOSP_ENF_ENF_FK FOREIGN KEY
(ENF_NUMSEGSOCIAL) REFERENCES ENFERMO (NUMSEGSOCIAL))
/

CREATE TABLE DOCTOR
 (NIF            VARCHAR2(9) NOT NULL
 ,APELLIDOS      VARCHAR2(20)
 ,NOMBRE         VARCHAR2(20)
 ,ESPECIALIDAD   VARCHAR2(40)
 ,CONSTRAINT DOC_PK PRIMARY KEY (NIF))
/

CREATE TABLE DOCTOR_HOSPITAL
 (HOSP_CODIGO    NUMBER(2,0) NOT NULL
 ,DOC_NIF        VARCHAR2(9) NOT NULL
 ,CONSTRAINT DOC_HOSP_PK PRIMARY KEY
(DOC_NIF,HOSP_CODIGO)
 ,CONSTRAINT DOC_HOSP_HOSP_FK FOREIGN KEY
(HOSP_CODIGO) REFERENCES HOSPITAL (CODIGO)
 ,CONSTRAINT DOC_HOSP_DOC_FK FOREIGN KEY (DOC_NIF)
REFERENCES DOCTOR (NIF))
/

CREATE TABLE EMPLEADO_HOSPITAL
 (HOSP_CODIGO    NUMBER(2,0) NOT NULL
 ,EMP_NIF        VARCHAR2(9) NOT NULL
 ,CONSTRAINT EMP_HOSP_PK PRIMARY KEY
(EMP_NIF,HOSP_CODIGO)
 ,CONSTRAINT EMP_HOSP_HOSP_FK FOREIGN KEY
(HOSP_CODIGO) REFERENCES HOSPITAL (CODIGO)
 ,CONSTRAINT EMP_HOSP_EMP_FK FOREIGN KEY (EMP_NIF)
REFERENCES EMPLEADO (NIF))
/
```

```
CREATE TABLE DEPARTAMENTO
 (CODIGO   NUMBER(2,0) NOT NULL
 ,NOMBRE   VARCHAR2(20)
 ,CONSTRAINT DEPT_PK PRIMARY KEY (CODIGO))
/

CREATE TABLE DEPARTAMENTO_EMPLEADO
 (DEPT_CODIGO   NUMBER(2,0) NOT NULL
 ,EMP_NIF       VARCHAR2(9) NOT NULL
 ,CONSTRAINT DEPT_EMP_PK PRIMARY KEY
(EMP_NIF,DEPT_CODIGO)
 ,CONSTRAINT DEPT_EMP_DEPT_FK FOREIGN KEY
(DEPT_CODIGO) REFERENCES DEPARTAMENTO (CODIGO)
 ,CONSTRAINT DEPT_EMP_EMP_FK FOREIGN KEY (EMP_NIF)
REFERENCES EMPLEADO (NIF))
/
```

SUPUESTO PRÁCTICO 4

Supuesto para el manejo de comandos de alteración sobre tablas y restricciones.

- Tabla ESTADOS_CIVILES

Código	CHAR(1)	CLAVE PRIMARIA
Descripción	VARCHAR2(50)	NO NULA

- Tabla PERSONA

Nif	VARCHAR2(9)	CLAVE PRIMARIA
Codestadocivil	NUMBER(1)	
Nombre	VARCHAR2(100)	NO NULO

Se han de realizar mediante código SQL las siguientes operaciones:

1. Crear las tablas que aparecen en el enunciado de este supuesto, según la descripción de columnas y restricciones que aparecen, sin añadir más restricciones de las que se indican.
2. Alterar la tabla estados_civiles para añadir una nueva restricción sobre la columna código de manera que únicamente se permita introducir en dicha columna los siguientes valores ('S','C','V','O').
3. Alterar la tabla persona de manera que se modifique el tipo del campo codestadocivil que tendrá que pasar a ser de tipo CHAR(1).

4. Añadir una nueva restricción a la tabla persona de manera que se restrinjan los valores de la columna codestadocivil a los mismos que pueda tener la columna código de la tabla estados_civiles.
5. Alterar la tabla persona para deshabilitar la restricción creada en el punto anterior.

Resolución del supuesto

```
-- PUNTO 1
create table estados_civiles
(codigo         char(1),
descripcion     varchar2(50) constraint
nn_estados_civiles not null,
constraint pk_estados_civiles primary key(codigo));

create table persona
(nif            varchar2(9),
codestadocivil  number(1),
nombre          varchar2(100) constraint
nn_nombre_persona not null,
constraint pk_personal primary key(nif));

-- PUNTO 2
alter table estados_civiles
add constraint ck_codigo_estados_civiles check (codigo
in ('S','C','V','O'));

-- PUNTO 3
alter table persona
modify (codestadocivil     char(1));

-- PUNTO 4
alter table persona
add constraint fk_persona_estados_civiles foreign key
(codestadocivil) references estados_civiles(codigo);

-- PUNTO 5
alter table persona
disable constraint fk_persona_estados_civiles;
```

SUPUESTO PRÁCTICO 5

A fin de poder realizar búsquedas de enfermos por su nombre de pila o por el apellido del mismo, se quiere indexar la tabla enfermo. Y para ello habrán de realizarse las siguientes operaciones:

1. Crear un índice para el campo nombre que admita repeticiones.
2. Crear un índice para el campo apellidos que admita repeticiones.

También se ha de indexar la tabla departamentos a fin de crear un índice único para el campo nombre de manera que no se pueda repetir los nombres de departamentos.

Resolución del supuesto

```
-- PUNTO 1
CREATE INDEX ix1_enfermo ON enfermo(nombre);

-- PUNTO 2
CREATE INDEX ix2_enfermo ON enfermo(apellidos);

-- Párrafo segundo
CREATE UNIQUE INDEX ix1_departamento ON
departamento(nombre);
```

SUPUESTO PRÁCTICO 6

Crear las siguientes vistas de solo lectura para que los usuarios las consulten. Se indica para cada vista, las consultas (querys) a utilizar.

1. Crear una vista que muestre los pacientes de un hospital concreto, solicitando el nombre del hospital. La consulta a añadir sería la siguiente:

```
SELECT e.nombre, e.apellidos, e.direccion,
e.sexo, e.fnacimiento,
e.numsegsocial,he.finscripcion
FROM hospital h, enfermo e, hospital_enfermo he
WHERE
    h.nombre = '&Indique_el_nombre_de_hospital'
AND he.hosp_codigo = h.codigo
AND  e.numsegsocial = he.enf_numsegsocial
ORDER BY e.apellidos, e.nombre, he.finscripcion;
```

2. Crear una vista de solo lectura que pueda mostrar todos los miembros de la plantilla según el turno que indique el usuario. La consulta a añadir sería la siguiente:

```
SELECT p.nombre AS "NOMBRE EMPLEADO",
p.apellidos, p.funcion, s.nombre AS "NOMBRE
SALA", h.nombre AS "NOMBRE HOSPITAL"
FROM hospital h, sala s, plantilla p,
plantilla_sala ps
WHERE
    p.turno = '&Indique_un_turno'
AND ps.plan_nif = p.nif
AND s.hosp_codigo = ps.sala_hosp_codigo
AND s.codigo = ps.sala_codigo
AND h.codigo = s.hosp_codigo;
WITH READ ONLY;
```

Resolución del supuesto

```
-- PUNTO 1
CREATE OR REPLACE VIEW v_enfermos_hospital
AS
SELECT e.nombre, e.apellidos, e.direccion, e.sexo,
e.fnacimiento,
      e.numsegsocial,he.finscripcion
FROM hospital h, enfermo e, hospital_enfermo he
WHERE
    h.nombre = '&Indique_el_nombre_de_hospital'
AND he.hosp_codigo = h.codigo
AND  e.numsegsocial = he.enf_numsegsocial
ORDER BY e.apellidos, e.nombre, he.finscripcion;

-- PUNTO 2
CREATE OR REPLACE VIEW v_plantilla_hospitales
AS
SELECT p.nombre AS "NOMBRE EMPLEADO", p.apellidos,
p.funcion, s.nombre AS "NOMBRE SALA", h.nombre AS
"NOMBRE HOSPITAL"
FROM hospital h, sala s, plantilla p, plantilla_sala
ps
WHERE
    p.turno = '&Indique_un_turno'
AND ps.plan_nif = p.nif
AND s.hosp_codigo = ps.sala_hosp_codigo
AND s.codigo = ps.sala_codigo
```

```
AND h.codigo = s.hosp_codigo;
WITH READ ONLY;
```

SUPUESTO PRÁCTICO 7

Crear una secuencia que comience en 1 y se vaya incrementando de 1 en 1 a fin de poder utilizarla para el número de inscripción de un enfermo.

Indicar el valor mínimo 1 y máximo (99.999.999), así como que no sea cíclica y que no se almacenen los valores en la caché.

Resolución del supuesto

```
CREATE SEQUENCE seq_inscripcion
START WITH 1
INCREMENT BY 1
MINVALUE 1
MAXVALUE 99999999
NOCYCLE
NOCACHE;
```

SUPUESTO PRÁCTICO 8

Supuesto repaso de instrucciones sublenguaje DDL.

1. Crear la tabla ejemplo1 que se indica a continuación mediante código SQL.
2. Alterar la tabla ejemplo1 que se ha creado con anterioridad para que se recojan estos nuevos campos:

   - COD_DOMI: Codificación del domicilio (3 caracteres). Puede ser un valor de la siguiente lista (‘CL’,’AVD’,’RD’,’PSO’).
   - SEGSOCIAL: Número de la seguridad social (9 dígitos numéricos). No puede ser un valor nulo y por defecto tiene que tener valor 0.

3. Crear la tabla ejemplo2 según se indica a continuación mediante código SQL.
4. Borrar la tabla ejemplo2.
5. Crear un índice en la tabla ejemplo1 para el campo APELLIDO que se llame “iapellido_ejemplo1”.
6. Crear un índice único en la tabla ejemplo1 para el campo SEGSOCIAL que se llame “iss_ejemplo1”.

7. Alterar la tabla ejemplo1 para añadir una restricción (constraint) que impida la repetición de los valores del campo SEGSOCIAL.
8. Crear una vista llamada "sal_empleado" sobre la tabla ejemplo1 que realice la siguiente consulta:
```
SELECT EMP_NO, SALARIO FROM EJEMPLO1
```
9. Borrar la vista anterior.

- Tabla EJEMPLO1

EMP_NO:	Número de empleado (4 dígitos numéricos) INDICE PRIMARIO
APELLIDO:	Apellido (16 caracteres)
FECHA_ALT	Fecha de alta en la empresa (por defecto la del día)
SALARIO:	Salario mensual (9 dígitos numéricos)

- Tabla EJEMPLO2

EMP_NO:	Número de empleado (4 dígitos numéricos) INDICE PRIMARIO.

Resolución del supuesto

```
-- PUNTO 1
create table ejemplo1
(EMP_NO         NUMBER(4) primary key
,APELLIDO       VARCHAR2(16)
,FECHA_ALT           DATE default SYSDATE
,SALARIO        NUMBER(9));

-- PUNTO 2
alter table ejemplo1
ADD (COD_DOMI   CHAR(1)     constraint ck_cod_domi
 check (cod_domi in ('CL','AVD','RD','PSO'))
,SEGSOCIAL           NUMBER(9)  default 0);

-- PUNTO 3
create table ejemplo2
(EMP_NO    NUMBER(4) primary key
);

-- PUNTO 4
drop table ejemplo2;
```

```
-- PUNTO 5
create index iapellido_ejemplo1 on ejemplo1(apellido);

-- PUNTO 6
create unique index iss_ejemplo1 on
ejemplo1(SEGSOCIAL);

-- PUNTO 7
alter table ejemplo1
add (constraint c_ss_ejemplo1 unique(SEGSOCIAL));

-- PUNTO 8
create view sal_empleado as
SELECT EMP_NO, SALARIO FROM EJEMPLO1;

-- PUNTO 9
Drop view sal_empleado;
```

SUPUESTO PRÁCTICO 9

A) Realizar la carga de los datos en las tablas creadas para el modelo de Hospital de acuerdo a los datos especificados en el Anexo II del libro.

B) Una vez realizada la carga de datos anteriores, insertar mediante instrucciones SQL los siguientes empleados en la plantilla no sanitaria del hospital con código 7:

- DIRECTOR

Nif: 12345678B
Nombre: Juan
Apellidos: Lopez Z.
No depende de ningún otro empleado.
Entró a formar parte de la plantilla del hospital el 11 de enero de 1970 y cobra un salario de 3.000 € sin que cobre comisiones por realizar su trabajo.
El director del hospital desarrolla su labor dentro del departamento con código 4.

- GERENTE

  Nif: 87654321A
  Nombre: Fermin
  Apellidos: Garcia L.
  Esta persona depende directamente del DIRECTOR. Entró a formar del hospital el 12 de enero de 1975, cobra un salario de 2.000 € que se ve complementado con una comisión fija de 1.000 €.
  El director del hospital desarrolla su labor dentro del departamento con código 4.

- ADMINISTRADOR

  Nif: 64328285C
  Nombre: Rosa
  Apellidos: Miranda R.
  Esta persona depende directamente del GERENTE. Entró a formar parte del hospital el 13 de enero de 1979, cobra un salario de 1.500 € sin recibir comisiones por su trabajo.
  El director del hospital desarrolla su labor dentro del departamento con código 4.

- CONTABLE

  Nif: 83253235F
  Nombre: Miguel
  Apellidos: Soria T.
  Esta persona depende directamente del ADMINISTRADOR. Entró a formar parte del hospital el 14 de enero de 1980, cobra un salario de 1.000 € y si realiza bien su trabajo cobra una comisión de 300 €.
  El director del hospital desarrolla su labor dentro del departamento con código 4.

Resolución del supuesto (apartado a)

```
-- Inserciones en la tabla HOSPITAL

INSERT INTO hospital
values (1,'916644264','Provincial',502,'O Donell 50');
```

```
INSERT INTO hospital
values (2,'915953111','General',987,'Atocha s/n');

INSERT INTO hospital
values (3,'919235411','La Paz',412,'Castellana 100');

INSERT INTO hospital
values (4,'915971500','San Carlos',845,'Ciudad
Universitaria');

INSERT INTO hospital
values (5,'915971500','Gr. Marañon',300,'Francisco
Silvela');

INSERT INTO hospital
values (6,'915971500','Doce Octubre',200,'Avda.
Cordoba');

INSERT INTO hospital
values (7,'915971500','La Zarzuela',100,'Moncloa');

-- Inserciones en la tabla ENFERMO

INSERT INTO enfermo
VALUES(280862482,'Goya20','Jose','M.M.',to_date('16051
956','ddmmyyyy'),'M');

INSERT INTO enfermo
VALUES(280862481,'Granada
35','Javier','R.R.',to_date('16081970','ddmmyyyy'),'M'
);

INSERT INTO enfermo
VALUES(280862480,'Sevilla
32','Ruben','S.S.',to_date('10091971','ddmmyyyy'),'M')
;

INSERT INTO enfermo
VALUES(280862483,'Toledo
1','Rocio','K.K.',to_date('10091968','ddmmyyyy'),'F');

INSERT INTO enfermo
VALUES(280862484,'Malaga
2','Laura','J.J.',to_date('10091971','ddmmyyyy'),'F');
```

```
INSERT INTO enfermo
VALUES(280862485,'Barcelona
2','Beatriz','A.A.',to_date('10091988','ddmmyyyy'),'M'
);

-- Inserciones en la tabla HOSPITAL_ENFERMO

INSERT INTO hospital_enfermo
VALUES(1,seq_inscripcion.nextval,280862482,to_date('01
012002','ddmmyyyy'));

INSERT INTO hospital_enfermo
VALUES(1,seq_inscripcion.nextval,280862482,to_date('02
012002','ddmmyyyy'));

INSERT INTO hospital_enfermo
VALUES(1,seq_inscripcion.nextval,280862482,to_date('03
012002','ddmmyyyy'));

INSERT INTO hospital_enfermo
VALUES(1,seq_inscripcion.nextval,280862482,to_date('04
012002','ddmmyyyy'));

INSERT INTO hospital_enfermo
VALUES(1,seq_inscripcion.nextval,280862482,to_date('05
012002','ddmmyyyy'));

INSERT INTO hospital_enfermo
VALUES(1,seq_inscripcion.nextval,280862481,to_date('01
012002','ddmmyyyy'));


INSERT INTO hospital_enfermo
VALUES(1,seq_inscripcion.nextval,280862481,to_date('02
012002','ddmmyyyy'));

INSERT INTO hospital_enfermo
VALUES(1,seq_inscripcion.nextval,280862481,to_date('03
012002','ddmmyyyy'));


INSERT INTO hospital_enfermo
VALUES(1,seq_inscripcion.nextval,280862480,to_date('01
102002','ddmmyyyy'));
```

```
INSERT INTO hospital_enfermo
VALUES(1,seq_inscripcion.nextval,280862480,to_date('02
102002','ddmmyyyy'));

INSERT INTO hospital_enfermo
VALUES(1,seq_inscripcion.nextval,280862480,to_date('03
102002','ddmmyyyy'));

INSERT INTO hospital_enfermo
VALUES(1,seq_inscripcion.nextval,280862483,to_date('03
102002','ddmmyyyy'));

INSERT INTO hospital_enfermo
VALUES(1,seq_inscripcion.nextval,280862484,to_date('04
102002','ddmmyyyy'));

INSERT INTO hospital_enfermo
VALUES(1,seq_inscripcion.nextval,280862485,to_date('03
112002','ddmmyyyy'));

INSERT INTO hospital_enfermo
VALUES(2,seq_inscripcion.nextval,280862482,to_date('02
012002','ddmmyyyy'));

INSERT INTO hospital_enfermo
VALUES(2,seq_inscripcion.nextval,280862482,to_date('03
012002','ddmmyyyy'));

INSERT INTO hospital_enfermo
VALUES(2,seq_inscripcion.nextval,280862482,to_date('04
012002','ddmmyyyy'));

INSERT INTO hospital_enfermo
VALUES(2,seq_inscripcion.nextval,280862482,to_date('05
012002','ddmmyyyy'));

INSERT INTO hospital_enfermo
VALUES(2,seq_inscripcion.nextval,280862481,to_date('02
012002','ddmmyyyy'));


INSERT INTO hospital_enfermo
VALUES(2,seq_inscripcion.nextval,280862481,to_date('03
012002','ddmmyyyy'));
```

```
INSERT INTO hospital_enfermo
VALUES(2,seq_inscripcion.nextval,280862480,to_date('02
102002','ddmmyyyy'));

INSERT INTO hospital_enfermo
VALUES(2,seq_inscripcion.nextval,280862480,to_date('03
102002','ddmmyyyy'));

INSERT INTO hospital_enfermo
VALUES(2,seq_inscripcion.nextval,280862484,to_date('04
102002','ddmmyyyy'));

INSERT INTO hospital_enfermo
VALUES(2,seq_inscripcion.nextval,280862485,to_date('03
112002','ddmmyyyy'));

INSERT INTO hospital_enfermo
VALUES(3,seq_inscripcion.nextval,280862482,to_date('03
012002','ddmmyyyy'));

INSERT INTO hospital_enfermo
VALUES(3,seq_inscripcion.nextval,280862482,to_date('04
012002','ddmmyyyy'));

INSERT INTO hospital_enfermo
VALUES(3,seq_inscripcion.nextval,280862482,to_date('05
012002','ddmmyyyy'));

INSERT INTO hospital_enfermo
VALUES(3,seq_inscripcion.nextval,280862480,to_date('03
102002','ddmmyyyy'));

INSERT INTO hospital_enfermo
VALUES(3,seq_inscripcion.nextval,280862485,to_date('03
112002','ddmmyyyy'));

INSERT INTO hospital_enfermo
VALUES(4,seq_inscripcion.nextval,280862482,to_date('04
012002','ddmmyyyy'));

INSERT INTO hospital_enfermo
VALUES(4,seq_inscripcion.nextval,280862482,to_date('05
012002','ddmmyyyy'));
```

```
INSERT INTO hospital_enfermo
VALUES(4,seq_inscripcion.nextval,280862485,to_date('03
112002','ddmmyyyy'));

INSERT INTO hospital_enfermo
VALUES(5,seq_inscripcion.nextval,280862482,to_date('05
012002','ddmmyyyy'));

INSERT INTO hospital_enfermo
VALUES(5,seq_inscripcion.nextval,280862485,to_date('03
112002','ddmmyyyy'));

INSERT INTO hospital_enfermo
VALUES(6,seq_inscripcion.nextval,280862485,to_date('03
112002','ddmmyyyy'));

-- Inserciones en la tabla SALA

INSERT INTO sala
VALUES (1,1,24,'Maternidad');
INSERT INTO sala
VALUES (1,2,21,'Cuidados intensivos');

INSERT INTO sala
VALUES (1,3,67,'Psiquiatrico');

INSERT INTO sala
VALUES (1,4,53,'Cardiologia');

INSERT INTO sala
VALUES (1,5,10,'Recuperacion');

INSERT INTO sala
VALUES (2,1,88,'Maternidad');

INSERT INTO sala
VALUES (2,2,88,'Cuidados intensivos');

INSERT INTO sala
VALUES (2,3,88,'Psiquiatrico');

INSERT INTO sala
VALUES (2,4,88,'Cardiologia');
```

```
INSERT INTO sala
VALUES (2,5,88,'Recuperacion');

INSERT INTO sala
VALUES (3,5,99,'Maternidad');

INSERT INTO sala
VALUES (3,4,99,'Cuidados intensivos');

INSERT INTO sala
VALUES (3,3,99,'Psiquiatrico');

INSERT INTO sala
VALUES (3,2,99,'Cardiologia');

INSERT INTO sala
VALUES (3,1,99,'Recuperacion');

INSERT INTO sala
VALUES (4,1,10,'Maternidad');

INSERT INTO sala
VALUES (4,2,11,'Cuidados intensivos');

INSERT INTO sala
VALUES (4,3,12,'Psiquiatrico');

INSERT INTO sala
VALUES (4,4,13,'Cardiologia');

INSERT INTO sala
VALUES (5,1,10,'Maternidad');

INSERT INTO sala
VALUES (5,2,11,'Cuidados intensivos');

INSERT INTO sala
VALUES (5,3,12,'Psiquiatrico');

INSERT INTO sala
VALUES (6,1,10,'Maternidad');

INSERT INTO sala
VALUES (6,2,11,'Cuidados intensivos');
```

```
INSERT INTO sala
VALUES (7,1,99,'Maternidad');

-- Inserciones en la tabla DOCTOR

INSERT INTO doctor
VALUES ('12345678A','Gutierrez
J.','Raimundo','Cardiologia');


INSERT INTO doctor
VALUES ('12345678F','Gutierrez
J.','Perico','Cardiologia');


INSERT INTO doctor
VALUES ('12345678J','Gutierrez
T.','Iñaqui','Cardiologia');


INSERT INTO doctor
VaLUES ('12345678B','Soledad
B.','Ines','Ginecologia');

INSERT INTO doctor
VaLUES ('12345678K','Casas
B.','Bartolome','Ginecologia');

INSERT INTO doctor
VALUES ('12345678C','Moreno D.','Rosa','Pediatria');

INSERT INTO doctor
VALUES ('12345678L','Moreno D.','Maria','Pediatria');

INSERT INTO doctor
VALUES ('12345678M','Moreno
D.','Isidoro','Pediatria');

INSERT INTO doctor
VALUES ('12345678N','Moreno
D.','Antonio','Pediatria');

INSERT INTO doctor
VALUES ('12345678D','Del Toro
D.','Ramiro','Psiquiatria');
```

```
INSERT INTO doctor
VALUES ('22345678A','Fermin
J.','Edmunto','Cardiologia');
INSERT INTO doctor
VALUES ('22345678J','Lopez
T.','Iñaqui','Cardiologia');

INSERT INTO doctor
VaLUES ('22345678B','Acaso B.','Ines','Ginecologia');

INSERT INTO doctor
VaLUES ('22345678K','Torres
B.','Bartolome','Ginecologia');

INSERT INTO doctor
VALUES ('22345678C','Moreno D.','Rosa','Pediatria');

INSERT INTO doctor
VALUES ('32345678A','Fernandez
J.','Loli','Cardiologia');

INSERT INTO doctor
VALUES ('32345678P','Fermin
J.','Jorge','Cardiologia');

INSERT INTO doctor
VALUES ('32345678J','Lopez
T.','Samuel','Cardiologia');

INSERT INTO doctor
VaLUES ('32345678B','Acaso B.','Maria','Ginecologia');

INSERT INTO doctor
VaLUES ('32345678K','Torres
B.','Tirano','Ginecologia');

INSERT INTO doctor
VALUES ('42345678A','Fernandez
J.','Ramon','Cardiologia');

INSERT INTO doctor
VALUES ('42345678M','Fermin J.','Fede','Cardiologia');
```

```
INSERT INTO doctor
VALUES ('42345678J','Lopez T.','Loles','Cardiologia');

INSERT INTO doctor
VaLUES ('42345678B','Acaso B.','Maica','Ginecologia');

INSERT INTO doctor
VaLUES ('42345678K','Torres
B.','Toñin','Ginecologia');

INSERT INTO doctor
VALUES ('52345678A','Fernandez
J.','Ramon','Cardiologia');

INSERT INTO doctor
VALUES ('52345678T','Fermin J.','Fede','Cardiologia');

INSERT INTO doctor
VALUES ('52345678J','Lopez T.','Loles','Cardiologia');

INSERT INTO doctor
VaLUES ('52345678B','Acaso B.','Maica','Ginecologia');

INSERT INTO doctor
VALUES ('62345678A','Fernandez
J.','Rocio','Cardiologia');

INSERT INTO doctor
VALUES ('62345678J','Lopez
T.','Carlos','Cardiologia');

INSERT INTO doctor
VaLUES ('62345678K','Torres B.','Juan','Ginecologia');

INSERT INTO doctor
VALUES ('72345678J','Lopez
T.','JuanMa','Cardiologia');

-- Inserciones en la tabla DOCTOR_HOSPITAL

INSERT INTO doctor_hospital
VALUES (1,'12345678A');
```

```
INSERT INTO doctor_hospital
VALUES (1,'12345678F');

INSERT INTO doctor_hospital
VALUES (1,'12345678J');

INSERT INTO doctor_hospital
VaLUES (1,'12345678B');

INSERT INTO doctor_hospital
VaLUES (1,'12345678K');

INSERT INTO doctor_hospital
VALUES (1,'12345678C');

INSERT INTO doctor_hospital
VALUES (1,'12345678L');

INSERT INTO doctor_hospital
VALUES (1,'12345678M');

INSERT INTO doctor_hospital
VALUES (1,'12345678N');

INSERT INTO doctor_hospital
VALUES (1,'12345678D');

INSERT INTO doctor_hospital
VALUES (2,'12345678A');

INSERT INTO doctor_hospital
VALUES (2,'22345678A');

INSERT INTO doctor_hospital
VALUES (2,'22345678J');

INSERT INTO doctor_hospital
VaLUES (2,'22345678B');

INSERT INTO doctor_hospital
VaLUES (2,'22345678K');

INSERT INTO doctor_hospital
VALUES (2,'22345678C');
```

```
INSERT INTO doctor_hospital
VALUES (3,'32345678A');

INSERT INTO doctor_hospital
VALUES (3,'32345678P');

INSERT INTO doctor_hospital
VALUES (3,'32345678J');

INSERT INTO doctor_hospital
VaLUES (3,'32345678B');

INSERT INTO doctor_hospital
VaLUES (3,'32345678K');

INSERT INTO doctor_hospital
VALUES (3,'22345678C');

INSERT INTO doctor_hospital
VALUES (4,'42345678A');

INSERT INTO doctor_hospital
VALUES (4,'42345678M');

INSERT INTO doctor_hospital
VALUES (4,'42345678J');

INSERT INTO doctor_hospital
VaLUES (4,'42345678B');

INSERT INTO doctor_hospital
VaLUES (4,'42345678K');

INSERT INTO doctor_hospital
VALUES (5,'52345678A');

INSERT INTO doctor_hospital
VALUES (5,'52345678T');

INSERT INTO doctor_hospital
VALUES (5,'52345678J');

INSERT INTO doctor_hospital
VaLUES (5,'52345678B');
```

```
INSERT INTO doctor_hospital
VALUES (6,'62345678A');

INSERT INTO doctor_hospital
VALUES (6,'62345678J');

INSERT INTO doctor_hospital
VaLUES (6,'62345678K');

INSERT INTO doctor_hospital
VALUES (7,'62345678A');

INSERT INTO doctor_hospital
VALUES (7,'72345678J');


-- Inserciones en la tabla DEPARTAMENTO

INSERT INTO departamento
VALUES(1,'CONTABILIDAD');

INSERT INTO departamento
VALUES(2,'INVESTIGACION');

INSERT INTO departamento
VALUES(3,'FACTURACION');

INSERT INTO departamento
VALUES(4,'ADMINISTRACION');

INSERT INTO departamento
VALUES(5,'FARMACIA');

INSERT INTO departamento
VALUES(6,'LIMPIEZA');


-- Inserciones en la tabla EMPLEADO

INSERT INTO empleado
VALUES('10000000A',TO_DATE('10012002','DDMMYYYY'),'Jor
ge',1000.22,3000.11,'DIRECTOR','Perez Sala',NULL);
```

```
INSERT INTO empleado
VALUES('20000000B',TO_DATE('11012002','DDMMYYYY'),'Jav
ier',500.22,2000.22,'GERENTE','Sala
Rodriguez','10000000A');

INSERT INTO empleado
VALUES('30000000C',TO_DATE('11012002','DDMMYYYY'),'Sol
edad',500.33,2000.33,'ADMISTRADOR','Lopez
J.','10000000A');

INSERT INTO empleado
VALUES('40000000D',TO_DATE('12012002','DDMMYYYY'),'Son
ia',NULL,1800.44,'JEFE FARMACIA','Moldes
R.','20000000B');

INSERT INTO empleado
VALUES('50000000E',TO_DATE('12012002','DDMMYYYY'),'Ant
onio',300.44,1800.44,'JEFE LABORATORIO','Lopez
A.','20000000B');

INSERT INTO empleado
VALUES('60000000F',TO_DATE('12012002','DDMMYYYY'),'Car
los',500.55,1800.55,'CONTABLE','Roa D.','30000000C');

INSERT INTO empleado
VALUES('70000000G',TO_DATE('13012002','DDMMYYYY'),'Lol
a',NULL,1000,'ADMINISTRATIVO','Sanchez
D.','60000000F');

INSERT INTO empleado
VALUES('80000000L',TO_DATE('13012002','DDMMYYYY'),'Ang
el',NULL,1000,'ADMINISTRATIVO','Perez','60000000F');

INSERT INTO empleado
VALUES('90000000M',TO_DATE('12012002','DDMMYYYY'),'Ram
on',NULL,1500,'JEFE LIMPIEZA','Maria
Casas','20000000B');

INSERT INTO empleado
VALUES('11000000P',TO_DATE('14012002','DDMMYYYY'),'Lui
s',NULL,700,'HIGIENE','Sanchez D.','90000000M');
```

```
INSERT INTO empleado
VALUES('12000000Q',TO_DATE('14012002','DDMMYYYY'),'Ros
a',NULL,700,'HIGIENE','Torres A.','90000000M');

INSERT INTO empleado
VALUES('10000000N',TO_DATE('14012002','DDMMYYYY'),'Sar
a',200,1000,'INVESTIGADOR','Gomez A.','50000000E');


-- Inserciones en la tabla EMPLEADO_HOSPITAL

INSERT INTO empleado_hospital
VALUES(1,'10000000A');

INSERT INTO empleado_hospital
VALUES(1,'20000000B');

INSERT INTO empleado_hospital
VALUES(1,'30000000C');

INSERT INTO empleado_hospital
VALUES(1,'40000000D');

INSERT INTO empleado_hospital
VALUES(1,'50000000E');

INSERT INTO empleado_hospital
VALUES(1,'60000000F');

INSERT INTO empleado_hospital
VALUES(1,'70000000G');

INSERT INTO empleado_hospital
VALUES(1,'80000000L');

INSERT INTO empleado_hospital
VALUES(1,'90000000M');

INSERT INTO empleado_hospital
VALUES(1,'11000000P');

INSERT INTO empleado_hospital
VALUES(1,'12000000Q');
```

```
INSERT INTO empleado_hospital
VALUES(1,'10000000N');

INSERT INTO empleado_hospital
VALUES(2,'10000000A');

INSERT INTO empleado_hospital
VALUES(2,'20000000B');

INSERT INTO empleado_hospital
VALUES(2,'30000000C');

INSERT INTO empleado_hospital
VALUES(2,'40000000D');

INSERT INTO empleado_hospital
VALUES(2,'50000000E');

INSERT INTO empleado_hospital
VALUES(2,'70000000G');

INSERT INTO empleado_hospital
VALUES(2,'80000000L');

INSERT INTO empleado_hospital
VALUES(2,'90000000M');

INSERT INTO empleado_hospital
VALUES(2,'11000000P');

INSERT INTO empleado_hospital
VALUES(2,'12000000Q');

INSERT INTO empleado_hospital
VALUES(2,'10000000N');

INSERT INTO empleado_hospital
VALUES(3,'10000000A');

INSERT INTO empleado_hospital
VALUES(3,'20000000B');

INSERT INTO empleado_hospital
VALUES(3,'30000000C');
```

```
INSERT INTO empleado_hospital
VALUES(3,'40000000D');

INSERT INTO empleado_hospital
VALUES(3,'80000000L');

INSERT INTO empleado_hospital
VALUES(3,'90000000M');

INSERT INTO empleado_hospital
VALUES(3,'11000000P');

INSERT INTO empleado_hospital
VALUES(3,'12000000Q');

INSERT INTO empleado_hospital
VALUES(3,'10000000N');

INSERT INTO empleado_hospital
VALUES(4,'10000000A');
INSERT INTO empleado_hospital
VALUES(4,'20000000B');

INSERT INTO empleado_hospital
VALUES(4,'30000000C');

INSERT INTO empleado_hospital
VALUES(4,'40000000D');

INSERT INTO empleado_hospital
VALUES(4,'80000000L');

INSERT INTO empleado_hospital
VALUES(4,'90000000M');

INSERT INTO empleado_hospital
VALUES(4,'12000000Q');

INSERT INTO empleado_hospital
VALUES(4,'10000000N');

INSERT INTO empleado_hospital
VALUES(5,'10000000A');
```

```
INSERT INTO empleado_hospital
VALUES(5,'20000000B');

INSERT INTO empleado_hospital
VALUES(5,'30000000C');

INSERT INTO empleado_hospital
VALUES(5,'40000000D');

INSERT INTO empleado_hospital
VALUES(5,'80000000L');

INSERT INTO empleado_hospital
VALUES(5,'12000000Q');

INSERT INTO empleado_hospital
VALUES(5,'10000000N');

INSERT INTO empleado_hospital
VALUES(6,'10000000A');

INSERT INTO empleado_hospital
VALUES(6,'20000000B');

INSERT INTO empleado_hospital
VALUES(6,'30000000C');

INSERT INTO empleado_hospital
VALUES(6,'40000000D');

INSERT INTO empleado_hospital
VALUES(6,'12000000Q');

INSERT INTO empleado_hospital
VALUES(6,'10000000N');


-- Inserciones en la tabla PLANTILLA

INSERT INTO plantilla
VALUES('11111111A',15000.22,'Alejandro','M','A.A.','EN
FERMERO');
```

```
INSERT INTO plantilla
VALUES('11111111B',15000.22,'Bartolome','T','B.B.','EN
FERMERO');

INSERT INTO plantilla
VALUES('11111111C',15000.22,'Carlos','N','C.C.','ENFER
MERO');

INSERT INTO plantilla
VALUES('22222222A',15000.22,'Adriana','M','A.A.','ENFE
RMERA');

INSERT INTO plantilla
VALUES('22222222B',15000.22,'Bibiana','T','B.B.','ENFE
RMERA');

INSERT INTO plantilla
VALUES('22222222C',15000.22,'Casilda','N','C.C.','ENFE
RMERA');

INSERT INTO plantilla
VALUES('33333333A',15000.22,'Alberto','M','A.A.','ENFE
RMERO');

INSERT INTO plantilla
VALUES('33333333B',15000.22,'Bonifacio','T','B.B.','EN
FERMERO');

INSERT INTO plantilla
VALUES('33333333C',15000.22,'Casimiro','N','C.C.','ENF
ERMERO');

INSERT INTO plantilla
VALUES('44444444A',15000.22,'Amelia','M','A.A.','ENFER
MERA');

INSERT INTO plantilla
VALUES('44444444B',15000.22,'Bony','T','B.B.','ENFERME
RA');

INSERT INTO plantilla
VALUES('44444444C',15000.22,'Casandra','N','C.C.','ENF
ERMERA');
```

```
INSERT INTO plantilla
VALUES('55555555A',15000.22,'Armando','M','A.A.','ENFE
RMERO');

INSERT INTO plantilla
VALUES('55555555B',15000.22,'Benicio','T','B.B.','ENFE
RMERO');

INSERT INTO plantilla
VALUES('55555555C',15000.22,'Ciceron','N','C.C.','ENFE
RMERO');

-- Inserciones en la tabla PLANTILLA_SALA

INSERT INTO plantilla_sala
VALUES(1,1,'11111111A');

INSERT INTO plantilla_sala
VALUES(1,1,'11111111B');

INSERT INTO plantilla_sala
VALUES(1,1,'11111111C');

INSERT INTO plantilla_sala
VALUES(1,2,'22222222A');

INSERT INTO plantilla_sala
VALUES(1,2,'22222222B');

INSERT INTO plantilla_sala
VALUES(1,2,'22222222C');

INSERT INTO plantilla_sala
VALUES(1,3,'33333333A');

INSERT INTO plantilla_sala
VALUES(1,3,'33333333B');

INSERT INTO plantilla_sala
VALUES(1,3,'33333333C');

INSERT INTO plantilla_sala
VALUES(1,4,'44444444A');
```

```
INSERT INTO plantilla_sala
VALUES(1,4,'44444444B');

INSERT INTO plantilla_sala
VALUES(1,4,'44444444C');

INSERT INTO plantilla_sala
VALUES(1,5,'55555555A');

INSERT INTO plantilla_sala
VALUES(1,5,'55555555B');

INSERT INTO plantilla_sala
VALUES(1,5,'55555555C');

INSERT INTO plantilla_sala
VALUES(2,1,'11111111A');

INSERT INTO plantilla_sala
VALUES(2,1,'11111111B');

INSERT INTO plantilla_sala
VALUES(2,2,'22222222A');
INSERT INTO plantilla_sala
VALUES(2,2,'22222222B');

INSERT INTO plantilla_sala
VALUES(2,3,'33333333A');

INSERT INTO plantilla_sala
VALUES(2,3,'33333333B');

INSERT INTO plantilla_sala
VALUES(2,4,'44444444A');

INSERT INTO plantilla_sala
VALUES(2,4,'44444444B');

INSERT INTO plantilla_sala
VALUES(2,5,'55555555A');

INSERT INTO plantilla_sala
VALUES(2,5,'55555555B');
```

```
INSERT INTO plantilla_sala
VALUES(3,1,'11111111A');

INSERT INTO plantilla_sala
VALUES(3,2,'22222222A');

INSERT INTO plantilla_sala
VALUES(3,3,'33333333A');

INSERT INTO plantilla_sala
VALUES(3,4,'44444444A');

INSERT INTO plantilla_sala
VALUES(3,5,'55555555A');

INSERT INTO plantilla_sala
VALUES(4,1,'11111111A');

INSERT INTO plantilla_sala
VALUES(4,2,'22222222A');

INSERT INTO plantilla_sala
VALUES(4,3,'33333333A');
INSERT INTO plantilla_sala
VALUES(4,4,'44444444A');

INSERT INTO plantilla_sala
VALUES(6,1,'11111111A');

INSERT INTO plantilla_sala
VALUES(6,2,'22222222A');

INSERT INTO plantilla_sala
VALUES(7,1,'11111111A');


-- Inserciones en la tabla DEPARTAMENTO_EMPLEADO

INSERT INTO departamento_empleado
VALUES(4,'10000000A');

INSERT INTO departamento_empleado
VALUES(4,'20000000B');
```

```
INSERT INTO departamento_empleado
VALUES(4,'30000000C');

INSERT INTO departamento_empleado
VALUES(5,'40000000D');

INSERT INTO departamento_empleado
VALUES(2,'50000000E');

INSERT INTO departamento_empleado
VALUES(1,'60000000F');

INSERT INTO departamento_empleado
VALUES(1,'70000000G');

INSERT INTO departamento_empleado
VALUES(1,'80000000L');

INSERT INTO departamento_empleado
VALUES(6,'90000000M');

INSERT INTO departamento_empleado
VALUES(6,'11000000P');

INSERT INTO departamento_empleado
VALUES(6,'12000000Q');

INSERT INTO departamento_empleado
VALUES(2,'10000000N');

commit;
```

Resolución del supuesto (apartado b)

```
-- Inserciones en la tabla EMPLEADO

INSERT INTO empleado
VALUES('12345678B',TO_DATE('11011970','DDMMYYYY'),'Jua
n',NULL,3000,'DIRECTOR','Lopez Z.',NULL);

INSERT INTO empleado
VALUES('87654321A',TO_DATE('12011975','DDMMYYYY'),'Fer
min',1000,2000,'GERENTE','Garcia L.','12345678B');
```

```
INSERT INTO empleado
VALUES('64328285C',TO_DATE('13011979','DDMMYYYY'),'Ros
a',NULL,1500,'ADMINISTRADOR','Miranda
R.','87654321A');

INSERT INTO empleado
VALUES('83253235F',TO_DATE('14011980','DDMMYYYY'),'Mig
uel',300,1000,'CONTABLE','Soria T.','64328285C');

-- Inserciones en la tabla DEPARTAMENTO_EMPLEAO

INSERT INTO departamento_empleado
VALUES(4,'12345678B');

INSERT INTO departamento_empleado
VALUES(4,'87654321A');

INSERT INTO departamento_empleado
VALUES(4,'64328285C');

INSERT INTO departamento_empleado
VALUES(4,'83253235F');

Commit;
```

SUPUESTO PRÁCTICO 10

Realice las consultas que se le formulan en los siguientes puntos.

1. Seleccione el nombre y código de hospital de todos los hospitales.
2. Encuentre a todos los miembros del personal de plantilla cuyo nombre comience por A y cuyo apellido comience por una letra anterior a la M. Se quiere conocer de cada uno de ellos, el nombre, apellidos y turno. Ordenar el listado por el nombre y apellidos.
3. Haga un listado de las enfermeras que ganen entre 2.000.000 y 2.500.000 pts. (1€ = 166.386 pts). Visualizar nombre y salario de la plantilla sanitaria.
4. Muestre el nombre de los hospitales con más de 100 camas, ordenados descendentemente.
5. Muestre la fecha del sistema con el formato “<DIA de la semana en letras> - <DIA en número> - <mes en letra>-<año en 4 dígitos>- <siglo>-<hora en 24>: <minutos> ”.

Resolución del supuesto (apartado 1)

```
select nombre, codigo from hospital;

NOMBRE           CODIGO
------------ ----------
Provincial            1
General               2
La Paz                3
San Carlos            4
Gr. Marañon           5
Doce Octubre          6
La Zarzuela           7

7 filas seleccionadas.
```

Resolución del supuesto (apartado 2)

```
select nombre, apellidos, funcion, turno from
plantilla where nombre like 'A%' and
substr(apellidos,1,1) < 'M' order by nombre,
apellidos;


NOMBRE          APELLIDOS                   FUNCION        T
--------------- --------------------------- -------- -------
Adriana         A.A.                        ENFERMERA      M
Alberto         A.A.                        ENFERMERO      M
Alejandro       A.A.                        ENFERMERO      M
Amelia          A.A.                        ENFERMERA      M
Armando         A.A.                        ENFERMERO      M
```

Resolución del supuesto (apartado 3)

```
select nombre, salario from plantilla where
UPPER(funcion) = 'ENFERMERA'
AND SALARIO*166.386 BETWEEN 2000000 AND 2500000;

NOMBRE                SALARIO
-------------------- ----------
Adriana                15000,22
Bibiana                15000,22
Casilda                15000,22
```

```
Amelia                    15000,22
Bony                      15000,22
Casandra                  15000,22

6 filas seleccionadas.
```

Resolución del supuesto (apartado 4)

```
select nombre from hospital
where numcamas > 100
order by nombre desc;

NOMBRE
------------
San Carlos
Provincial
La Paz
Gr. Marañon
General
Doce Octubre

6 filas seleccionadas.
```

Resolución del supuesto (apartado 5)

```
select to_char(sysdate,'DAY-DD-MONTH-YYYY-SCC-HH24:MI')
from dual;

TO_CHAR(SYSDATE,'DAY-DD-MONTH-YYYY-SCC
--------------------------------------
VIERNES  -29-MAYO    -2011- 21-20:05
```

SUPUESTO PRÁCTICO 11

Realice las consultas que se le formulan en los siguientes puntos.

1. Encontrar el salario medio de los empleados no sanitarios agrupados por el oficio que desempeñan. Mostrar conteo de personas que trabajan por oficio, salario medio del oficio y nombre del oficio. La media del salario se habrá de mostrar con el siguiente formato: redondeado a 2 decimales y con separador de miles. Los títulos a mostrar para las 3 columnas serán: Nº Pers., Salario M, Oficio.

2. Diseñar una consulta que muestre el nombre de aquellos doctores que comiencen por la letra "I" que se repitan en la tabla doctores más de vez.
3. Diseñar una consulta que muestre el nombre, apellidos, edad y sexo de todos los enfermos de la tabla enfermo que sean del sexo femenino o bien, que siendo del sexo masculino tengan más de 40 años. No se deben repetir filas con la misma información en el listado que se muestre mediante la consulta.

Resolución del supuesto (apartado 1)

```
SELECT OFICIO, COUNT(NIF) "Nº Pers.",
TO_CHAR(ROUND(AVG(SALARIO),2),'9G999D99') "Salario M"
FROM EMPLEADO GROUP BY OFICIO;

OFICIO               Nº Pers. Salario M
-------------------- ---------- ---------
GERENTE                       2  2.000,11
CONTABLE                      2  1.400,28
ADMINISTRATIVO                2  1.000,00
INVESTIGADOR                  1  1.000,00
HIGIENE                       2    700,00
DIRECTOR                      2  3.000,06
JEFE FARMACIA                 1  1.800,44
ADMISTRADOR                   1  2.000,33
JEFE LABORATORIO              1  1.800,44
JEFE LIMPIEZA                 1  1.500,00
ADMINISTRADOR                 1  1.500,00

11 filas seleccionadas.
```

Resolución del supuesto (apartado 2)

```
SELECT NOMBRE, COUNT(*) FROM DOCTOR
WHERE NOMBRE LIKE UPPER('I%')
GROUP BY NOMBRE
HAVING COUNT(*) > 1;

NOMBRE               COUNT(*)
-------------------- ----------
Ines                          2
Iñaqui                        2
```

Resolución del supuesto (apartado 3)

```
SELECT DISTINCT NOMBRE, APELLIDOS,
ROUND(MONTHS_BETWEEN(SYSDATE,FNACIMIENTO)/12) "Edad",
sexo FROM ENFERMO
WHERE (SEXO = 'F')
OR (SEXO = 'M' AND
ROUND(MONTHS_BETWEEN(SYSDATE,FNACIMIENTO)/12) > 40)
order by SEXO, "Edad" DESC, apellidos, nombre;
```

```
NOMBRE               APELLIDOS                                   Edad S
-------------------- ---------------------------------------- ---------- -
Rocio                K.K.                                          43 F
Laura                J.J.                                          40 F
Jose                 M.M.                                          55 M
Javier               R.R.                                          41 M
```

SUPUESTO PRÁCTICO 12

Realice las consultas que se le formulan en los siguientes puntos.

1. ¿Cuál es la plantilla que trabaja en el turno de mañana, en el hospital 3? Visualizar nombre del empleado de la plantilla y su función.
2. Recuperar todo el personal de la plantilla sanitaria del hospital con código1, indicando el nombre de cada persona, función y turno, realizando una ordenación por turno, nombre y función.
3. Mostrar el nombre y oficio de todos los miembros del personal no sanitario del hospital con código 1 que no cobren comisiones y ordenar el resultado por el salario que cobren.
4. Seleccionar las distintas especialidades que imparten los doctores del hospital 1.
5. Realizar una consulta para extraer todos los doctores del hospital con código1. Se deberá devolver en la consulta, el nif, nombre, especialidad que ocupan y una codificación de las especialidad como se indica a continuación (cardiología1, psiquiatría 2, pediatría 3, el resto 4). Ordenar los resultados por la codificación de la especialidad.
6. Realizar una consulta que nos muestre el número de salas distintas de las que consta cada hospital.
7. Confeccionar una consulta que nos muestre agrupada por hospital y turno, las funciones que se desarrollan y el número de personas que las realizan.
8. Realizar una consulta que nos muestre agrupados por código de hospital, el conteo del número de enfermos de cada hospital, teniendo en cuenta que solo se tendrá que incluir en la consulta, aquellos pacientes cuyo año de nacimiento

sea igual o mayor a 1955. Además, la consulta solo mostrará aquellos hospitales cuyo número de enfermos sea mayor o igual a 10. La columna con el conteo de enfermos se deberá llamar "Num. Enfermos".

9. Diseñar una consulta que muestre el organigrama del personal no sanitario. Para cada miembro, se incluirá la siguiente función: LPAD(' ',2*(LEVEL – 1)), concatenada con el nombre de la persona, ' –' el oficio que realizar el miembro,' NIVEL(', el nivel en el que se encuentra dentro del organigrama del hospital y ')'. A todo esto se le denominará con el alias "Organización hospital".
10. Diseñar una consulta que muestre el código del hospital, código del departamento, suma del salario en euros "Sal euros.", suma del salario en pesetas "Sal. Pts." Y suma de las comisiones en euros "Comisión eur.", acumulados los valores por hospital y departamento, siempre y cuando la fecha de alta de los empleados no asalariados, sea mayor o igual al 1 de enero de 2002, el código de hospital sea 1,3,5 o 7 y el código de departamento se encuentre entre 1 y 4. De todos los valores que se podrían mostrar, solo nos interesan aquellos cuya suma de salarios en pesetas, sea mayor a las 500.000 pts. y la suma de comisiones en euros, menor a 2.000 €. Como valor de referencia para la conversión a pesetas tomaremos 166.386.
11. Diseñar una consulta que permita mostrar el nombre de los hospitales que poseen menos camas que la suma de camas que poseen cada una de las salas del mismo.
12. Diseñar una consulta que nos muestre el nombre y apellidos de todos aquellos enfermos que hayan asistidos a más de 2 hospitales distintos. El título de las columnas será, nombre, apellidos y nº asistencias
13. Diseñar una consulta que nos muestre los distintos nombres y apellidos de todos aquellos enfermos cuyos apellidos comiencen por las letras A o M, que hayan asistido al hospital con código 1 y que además hayan asistido también al hospital con código 2.
14. Diseñar una consulta que nos muestre los distintos nombres y apellidos de doctores que han sido fieles a la institución que les paga y consecuentemente solo han trabajado en un hospital a lo largo de su carrera.
15. Diseñar una consulta que nos muestre el nombre, apellidos y nif de todos los miembros que trabajan en el hospital número 1 (doctores, plantilla no sanitaria, plantilla sanitaria), ordenado el listado por apellidos, nombre y nif.
16. Diseñar una consulta que nos muestre el nombre, teléfono del hospital, número 3, así como el nombre de las salas del mismo, ordenado el listado por el nombre de la sala. El nombre de las respectivas columnas a mostrar será: "Hospital", "Tlf","Sala“
17. Diseñar un listado que nos muestre para cada miembro de la plantilla sanitaria que trabaje en el turno de noche (N) y que sea enfermero, el nif, nombre, salario que cobra, nombre de sala en la que trabaja y nombre del hospital en el que

trabaja. El título de las columnas será: NIF, NOMBRE, SALARIO, "Nombre Sala", "Nombre Hospital". El listado se ordenará por el nombre del miembro de la plantilla, nombre del hospital y nombre de la sala.

18. Diseñar un listado que nos muestre el nombre de los departamentos en los que están trabajando más de 5 personas. El título de las columnas será "Departamento", "Nº Trabajadores". El listado se ordenará por el nombre del departamento.
19. Diseñar un listado que nos muestre todo el personal sanitario incluidos los doctores que trabajan en el hospital número 4. Para este listado se mostrará la siguiente información:

    - En el caso de que sea un miembro de la plantilla sanitaria: nombre del hospital, nombre de la sala, nombre de la persona, función que tiene y el literal fijo *'(SANITARIO)'.*Utilizar como alias de columnas los siguientes: "Hospital", "Sala", "Nombre", "Funcion", "Tipo empleado"
    - En el caso de que sea un doctor: Nombre del hospital, nulo, nombre del doctor, especialidad y el literal fijo *'(DOCTOR)'*. Utilizar como alias de columnas los que se han indicado anteriormente.

    Todo el listado se ordenará por el nombre del hospital, nombre de la sala y nombre del miembro sanitario.

Resolución del supuesto (apartado 1)

```
select nombre, funcion from plantilla, plantilla_sala
where sala_hosp_codigo = 3 and nif = plan_nif and turno =
'M';

NOMBRE               FUNCION
-------------------- --------------------
Alejandro            ENFERMERO
Adriana              ENFERMERA
Alberto              ENFERMERO
Amelia               ENFERMERA
Armando              ENFERMERO
```

Resolución del supuesto (apartado 2)

```
select nombre, funcion, turno
from plantilla, plantilla_sala
where sala_hosp_codigo = 1
and nif = plan_nif;
```

```
NOMBRE               FUNCION              T
-------------------- -------------------- -
Alejandro            ENFERMERO            M
Bartolome            ENFERMERO            T
Carlos               ENFERMERO            N
Adriana              ENFERMERA            M
Bibiana              ENFERMERA            T
Casilda              ENFERMERA            N
Alberto              ENFERMERO            M
Bonifacio            ENFERMERO            T
Casimiro             ENFERMERO            N
Amelia               ENFERMERA            M
Bony                 ENFERMERA            T
Casandra             ENFERMERA            N
Armando              ENFERMERO            M
Benicio              ENFERMERO            T
Ciceron              ENFERMERO            N

15 filas seleccionadas.
```

Resolución del supuesto (apartado 3)

```
select nombre, oficio,salario from empleado,
empleado_hospital
where hosp_codigo = 1
and nif = emp_nif
and comision is null
order by salario;

NOMBRE               OFICIO                  SALARIO
-------------------- -------------------- ----------
Rosa                 HIGIENE                    700
Luis                 HIGIENE                    700
Lola                 ADMINISTRATIVO            1000
Angel                ADMINISTRATIVO            1000
Ramon                JEFE LIMPIEZA             1500
Sonia                JEFE FARMACIA           1800,44

6 filas seleccionadas.
```

Resolución del supuesto (apartado 4)

```
select distinct especialidad from doctor, doctor_hospital
where hosp_codigo = 1 and nif = doc_nif;

ESPECIALIDAD
----------------------------------------
Pediatria
Cardiologia
Ginecologia
Psiquiatria
```

Resolución del supuesto (apartado 5)

```
select nif, nombre, especialidad,
decode(upper(especialidad),'CARDIOLOGIA',1
,'PSIQUIATRIA',2,'PEDIATRIA',3,4) as codigo from doctor,
doctor_hospital
where hosp_codigo = 1 and nif = doc_nif
order by codigo;


NIF       NOMBRE               ESPECIALIDAD        CODIGO
--------- -------------------- ------------------ --------------
12345678F Perico               Cardiologia         1
12345678J Iñaqui               Cardiologia         1
12345678A Raimundo             Cardiologia         1
12345678D Ramiro               Psiquiatria         2
12345678N Antonio              Pediatria           3
12345678C Rosa                 Pediatria           3
12345678L Maria                Pediatria           3
12345678M Isidoro              Pediatria           3
12345678B Ines                 Ginecologia         4
12345678K Bartolome            Ginecologia         4


10 filas seleccionadas.
```

Resolución del supuesto (apartado 6)

```
SELECT hosp_codigo, count(codigo) "Num. Salas" from sala
group by hosp_codigo;
```

```
HOSP_CODIGO Num. Salas
----------- ----------
          1          5
          6          2
          2          5
          4          4
          5          3
          3          5
          7          1

7 filas seleccionadas.
```

Resolución del supuesto (apartado 7)

```
select sala_hosp_codigo, turno, funcion, count(*) from
plantilla, plantilla_sala
where plan_nif = nif
group by sala_hosp_codigo, turno, funcion;
SALA_HOSP_CODIGO T FUNCION               COUNT(*)
---------------- - -------------------- ----------
               4 M ENFERMERO                    2
               6 M ENFERMERO                    1
               2 M ENFERMERO                    3
               7 M ENFERMERO                    1
               2 T ENFERMERA                    2
               1 N ENFERMERA                    2
               1 M ENFERMERA                    2
               2 M ENFERMERA                    2
               1 M ENFERMERO                    3
               1 N ENFERMERO                    3
               1 T ENFERMERA                    2
               1 T ENFERMERO                    3
               2 T ENFERMERO                    3
               6 M ENFERMERA                    1
               3 M ENFERMERO                    3
               3 M ENFERMERA                    2
               4 M ENFERMERA                    2

17 filas seleccionadas.
```

Resolución del supuesto (apartado 8)

```
select hosp_codigo, count(*) "Num. Enfermos" FROM
ENFERMO, HOSPITAL_ENFERMO
WHERE NUMSEGSOCIAL = ENF_NUMSEGSOCIAL AND
TO_NUMBER(TO_CHAR(FNACIMIENTO,'YYYY')) >= 1955
GROUP BY HOSP_CODIGO
HAVING COUNT(*) >= 10;

HOSP_CODIGO Num. Enfermos
----------- -------------
          1            14
          2            10
```

Resolución del supuesto (apartado 9)

```
SELECT LPAD(' ',2*(LEVEL - 1))||NOMBRE||'-'||OFICIO||'
NIVEL ('||LEVEL||')' "Organizacion Hospital"
FROM EMPLEADO
START WITH UPPER(OFICIO) = 'DIRECTOR'
CONNECT BY PRIOR NIF = NIF_DEPENDIENTE;
Organizacion Hospital
--------------------------------------------------------------------
Jorge-DIRECTOR   NIVEL (1)
  Javier-GERENTE   NIVEL (2)
    Sonia-JEFE FARMACIA   NIVEL (3)
    Antonio-JEFE LABORATORIO   NIVEL (3)
      Sara-INVESTIGADOR   NIVEL (4)
    Ramon-JEFE LIMPIEZA   NIVEL (3)
      Luis-HIGIENE   NIVEL (4)
      Rosa-HIGIENE   NIVEL (4)
  Soledad-ADMISTRADOR   NIVEL (2)
    Carlos-CONTABLE   NIVEL (3)
      Lola-ADMINISTRATIVO   NIVEL (4)
      Angel-ADMINISTRATIVO   NIVEL (4)
Juan-DIRECTOR   NIVEL (1)
  Fermin-GERENTE   NIVEL (2)
    Rosa-ADMINISTRADOR   NIVEL (3)
      Miguel-CONTABLE   NIVEL (4)

16 filas seleccionadas.
```

Resolución del supuesto (apartado 10)

```
SELECT EH.HOSP_CODIGO, DE.DEPT_CODIGO,
to_CHAR(SUM(E.SALARIO),'9G999D99') "Sal euros.",
TO_CHAR(ROUND(SUM(E.SALARIO)*166.386),'9G999G999') "Sal.
pts." , SUM(E.COMISION) "Comision eur."
FROM EMPLEADO E, EMPLEADO_HOSPITAL EH,
DEPARTAMENTO_EMPLEADO DE
WHERE EH.HOSP_CODIGO IN (1,3,5,7)
AND DE.EMP_NIF = EH.EMP_NIF
AND DE.DEPT_CODIGO BETWEEN 1 AND 4
AND E.NIF = EH.EMP_NIF
AND E.FALTA >= TO_DATE('01012002','DDMMYYYY')
GROUP BY EH.HOSP_CODIGO, DE.DEPT_CODIGO
HAVING ROUND(SUM(E.SALARIO)*166.386) > 500000
AND SUM(E.COMISION) < 2000;

HOSP_CODIGO DEPT_CODIGO Sal euros Sal. pts.  Comision eur.
----------- ----------- --------- ---------- -------------
          1           1  3.800,55    632.358        500,55
```

Resolución del supuesto (apartado 11)

```
SELECT NOMBRE
FROM HOSPITAL
WHERE NUMCAMAS < (SELECT SUM(NUMCAMAS)
                  FROM SALA
                  WHERE HOSP_CODIGO = HOSPITAL.CODIGO);

NOMBRE
------------
La Paz
```

Resolución del supuesto (apartado 12)

```
SELECT  NOMBRE, APELLIDOS, COUNT(DISTINCT HOSP_CODIGO) "Nº
asistencias"
FROM ENFERMO, HOSPITAL_ENFERMO
WHERE NUMSEGSOCIAL = ENF_NUMSEGSOCIAL
GROUP BY NOMBRE, APELLIDOS
HAVING COUNT(DISTINCT HOSP_CODIGO) > 2;
```

```
NOMBRE               APELLIDOS                                Nº asistencias
-------------------- ---------------------------------------- --------------
Ruben                S.S.                                                3
Beatriz              A.A.                                                6
Jose                 M.M.                                                5
```

Resolución del supuesto (apartado 13)

Esta solución utiliza un JOIN entre las tablas.

```
SELECT DISTINCT E1.NOMBRE, E1.APELLIDOS
FROM ENFERMO E1, HOSPITAL_ENFERMO EH
WHERE (E1.APELLIDOS LIKE 'A%' OR E1.APELLIDOS LIKE 'M%')
AND EH.HOSP_CODIGO = 1
AND E1.NUMSEGSOCIAL = EH.ENF_NUMSEGSOCIAL
AND EXISTS (SELECT NULL
           FROM ENFERMO E2, HOSPITAL_ENFERMO EH2
           WHERE EH2.HOSP_CODIGO = 2
           AND E2.NUMSEGSOCIAL = EH2.ENF_NUMSEGSOCIAL
           AND E2.NOMBRE = E1.NOMBRE
           AND E2.APELLIDOS = E1.APELLIDOS);
```

```
NOMBRE               APELLIDOS
-------------------- ----------------------------------------
Beatriz              A.A.
Jose                 M.M.
```

Resolución del supuesto (apartado 13)

Esta solución utiliza una cláusula INSERT entre dos consultas para resolver el problema con el mismo resultado que en la solución anterior.

```
SELECT DISTINCT E1.NOMBRE, E1.APELLIDOS
FROM ENFERMO E1, HOSPITAL_ENFERMO EH
WHERE (E1.APELLIDOS LIKE 'A%' OR E1.APELLIDOS LIKE 'M%')
AND EH.HOSP_CODIGO = 1
AND E1.NUMSEGSOCIAL = EH.ENF_NUMSEGSOCIAL
INTERSECT
SELECT DISTINCT E2.NOMBRE, E2.APELLIDOS
FROM ENFERMO E2, HOSPITAL_ENFERMO EH2
WHERE EH2.HOSP_CODIGO = 2
AND E2.NUMSEGSOCIAL = EH2.ENF_NUMSEGSOCIAL;
```

```
NOMBRE               APELLIDOS
-------------------- ----------------------------------------
Beatriz              A.A.
Jose                 M.M.
```

Resolución del supuesto (apartado 14)

```
SELECT DISTINCT D1.NOMBRE, D1.APELLIDOS
FROM DOCTOR D1, DOCTOR_HOSPITAL DH1
WHERE D1.NIF = DH1.DOC_NIF
AND NOT EXISTS (SELECT NULL
                FROM DOCTOR D2, DOCTOR_HOSPITAL DH2
                WHERE
                     DH2.HOSP_CODIGO != DH1.HOSP_CODIGO
                AND  D2.NIF = DH2.DOC_NIF
                AND  D2.NOMBRE = D1.NOMBRE
                AND  D2.APELLIDOS = D1.APELLIDOS);


NOMBRE               APELLIDOS
-------------------- --------------------
Iñaqui               Lopez T.
Tirano               Torres B.
Antonio              Moreno D.
Bartolome            Torres B.
Loli                 Fernandez J.
Jorge                Fermin J.
Samuel               Lopez T.
Toñin                Torres B.
Iñaqui               Gutierrez T.
Maria                Moreno D.
Edmunto              Fermin J.
JuanMa               Lopez T.
Carlos               Lopez T.
Juan                 Torres B.
Ines                 Acaso B.
Ines                 Soledad B.
Ramiro               Del Toro D.
Perico               Gutierrez J.
Bartolome            Casas B.
Isidoro              Moreno D.
Maria                Acaso B.

21 rows selected.
```

Resolución del supuesto (apartado 15)

```
SELECT NOMBRE, APELLIDOS, NIF
FROM DOCTOR, DOCTOR_HOSPITAL
WHERE HOSP_CODIGO = 1
AND NIF = DOC_NIF
UNION
SELECT NOMBRE, APELLIDOS, NIF
FROM PLANTILLA, PLANTILLA_SALA
WHERE SALA_HOSP_CODIGO = 1
AND NIF = PLAN_NIF
UNION
SELECT NOMBRE, APELLIDOS, NIF
FROM EMPLEADO, EMPLEADO_HOSPITAL
WHERE HOSP_CODIGO = 1
AND NIF = EMP_NIF
ORDER BY APELLIDOS, NOMBRE, NIF;
```

NOMBRE	APELLIDOS	NIF
Adriana	A.A.	22222222A
Alberto	A.A.	33333333A
Alejandro	A.A.	11111111A
Amelia	A.A.	44444444A
Armando	A.A.	55555555A
Bartolome	B.B.	11111111B
Benicio	B.B.	55555555B
Bibiana	B.B.	22222222B
Bonifacio	B.B.	33333333B
Bony	B.B.	44444444B
Bartolome	Casas B.	12345678K
Carlos	C.C.	11111111C
Casandra	C.C.	44444444C
Casilda	C.C.	22222222C
Casimiro	C.C.	33333333C
Ciceron	C.C.	55555555C
Ramiro	Del Toro D.	12345678D
Sara	Gomez A.	10000000N
Perico	Gutierrez J.	12345678F
Raimundo	Gutierrez J.	12345678A
Iñaqui	Gutierrez T.	12345678J
Antonio	Lopez A.	50000000E
Soledad	Lopez J.	30000000C
Ramon	Maria Casas	90000000M
Sonia	Moldes R.	40000000D
Antonio	Moreno D.	12345678N
Isidoro	Moreno D.	12345678M
Maria	Moreno D.	12345678L
Rosa	Moreno D.	12345678C
Angel	Perez	80000000L

```
Jorge             Perez Sala                          10000000A
Carlos            Roa D.                              60000000F
Javier            Sala Rodriguez                      20000000B
Lola              Sanchez D.                          70000000G
Luis              Sanchez D.                          11000000P
Ines              Soledad B.                          12345678B
Rosa              Torres A.                           12000000Q

37 filas seleccionadas.
```

Resolución del supuesto (apartado 16)

```
SELECT H.NOMBRE "Hospital", H.TELEFONO "Tlf", S.NOMBRE
"Sala"
FROM HOSPITAL H, SALA S
WHERE
     H.CODIGO = 3
AND  S.HOSP_CODIGO = H.CODIGO
ORDER BY "Sala";
Hospital            Tlf Sala
------------ ---------- ------------------------------
La Paz        919235411 Cardiologia
La Paz        919235411 Cuidados intensivos
La Paz        919235411 Maternidad
La Paz        919235411 Psiquiatrico
La Paz        919235411 Recuperacion
```

Resolución del supuesto (apartado 17)

```
SELECT P.NIF, P.NOMBRE, P.SALARIO, S.NOMBRE "Nombre
Sala", H.NOMBRE "Nombre Hospital"
FROM PLANTILLA P, SALA S, HOSPITAL H, PLANTILLA_SALA PS
WHERE TURNO = 'N'
AND UPPER(FUNCION) = 'ENFERMERO'
AND S.HOSP_CODIGO = PS.SALA_HOSP_CODIGO
AND S.CODIGO = PS.SALA_CODIGO
AND P.NIF = PS.PLAN_NIF
AND H.CODIGO = S.HOSP_CODIGO
ORDER BY P.NOMBRE, "Nombre Hospital", "Nombre Sala";

NIF       NOMBRE       SALARIO    Nombre Sala       Nombre Hospi
--------- ------------ ---------- ----------------- ------------
11111111C Carlos      15000,22   Maternidad          Provincial
33333333C Casimiro    15000,22   Psiquiatrico        Provincial
55555555C Ciceron     15000,22   Recuperacion        Provincial
```

Resolución del supuesto (apartado 18)

```
SELECT D.NOMBRE "Departamento", COUNT(E.EMP_NIF) "Nº
Trabajadores"
FROM DEPARTAMENTO_EMPLEADO E, DEPARTAMENTO D
WHERE D.CODIGO = E.DEPT_CODIGO
GROUP BY D.NOMBRE
HAVING COUNT(E.EMP_NIF) > 5;

Departamento         Nº Trabajadores
-------------------- ---------------
ADMINISTRACION                     7
```

Resolución del supuesto (apartado 19)

```
SELECT H.NOMBRE "Hospital", S.NOMBRE "Sala", P.NOMBRE
"Nombre", P.FUNCION "Funcion" , '(SANITARIO)' "Tipo
empleado"
FROM HOSPITAL H, PLANTILLA P, SALA S, PLANTILLA_SALA PS
WHERE PS.SALA_HOSP_CODIGO = 4
AND P.NIF = PS.PLAN_NIF
AND S.HOSP_CODIGO = PS.SALA_HOSP_CODIGO
AND S.CODIGO = PS.SALA_CODIGO
AND H.CODIGO = S.HOSP_CODIGO
UNION
SELECT H.NOMBRE "Hospital", NULL "Sala", D.NOMBRE
"Nombre", D.ESPECIALIDAD "Funcion", '(DOCTOR)' "Tipo
empleado"
FROM HOSPITAL H, DOCTOR D, DOCTOR_HOSPITAL DH
WHERE DH.HOSP_CODIGO = 4
AND D.NIF = DH.DOC_NIF
AND H.CODIGO = DH.HOSP_CODIGO
ORDER BY "Hospital","Sala","Nombre";

Hospital     Sala                           Nombre
------------ ------------------------------ --------------------
Funcion                                Tipo emplea
-------------------------------------- -----------
San Carlos   Cardiologia                    Amelia
ENFERMERA                              (SANITARIO)

San Carlos   Cuidados intensivos            Adriana
ENFERMERA                              (SANITARIO)
```

```
San Carlos   Maternidad                    Alejandro
ENFERMERO                                 (SANITARIO)

San Carlos   Psiquiatrico                  Alberto
ENFERMERO                                 (SANITARIO)

San Carlos                                 Fede
Cardiologia                               (DOCTOR)

San Carlos                                 Loles
Cardiologia                               (DOCTOR)

San Carlos                                 Maica
Ginecologia                               (DOCTOR)

San Carlos                                 Ramon
Cardiologia                               (DOCTOR)

San Carlos                                 Toñin
Ginecologia                               (DOCTOR)


9 filas seleccionadas.
```

RESOLUCIÓN DE CUESTIONES DE CERTIFICACIÓN

CUESTIÓN 1

Se propone gestionar todos los empleados que reciben un incremento del 10% en el salario y una comisión que es un 20% mayor al salario incrementado. Se pide escribir una consulta que visualice adecuadamente el salario, la comisión y la compensación total.

Evalúa las siguientes sentencias SQL:

```
1. SELECT sal * 1.1 "Salary", sal * 1.1 * .2 "Commission", sal
   * 1.1 + sal * 1.1 * .2 "Compensation"      FROM    emp;
2. SELECT sal * 1.1 "Salary", sal * 1.1 * .2 "Commission", (sal
   * 1.1) + (sal * .2) "Compensation"      FROM    emp;
3. SELECT sal * 1.1 "Salary", sal * .2 "Commission", sal * 1.1
   + sal * 1.1 * .2 "Compensation"      FROM    emp;
```

¿Cuál es el resultado de estas sentencias?

La sentencia 1 es la correcta.

A. 2 sentencias devuelven el resultado deseado.
B. Solo una sentencia devuelve el resultado deseado.
C. Solo 2 sentencias devuelven el resultado deseado.
D. Solo 3 sentencias devuelven el resultado deseado.
E. Ninguna de las sentencias devuelve el resultado deseado.

CUESTIÓN 2

¿Qué sentencia relacionada con la actualización de una tabla sin cláusula WHERE es verdadera?

A. La sentencia no se ejecutará.
B. Solo las filas especificadas serán actualizadas.
C. Se actualizarán todas las filas de la tabla.
D. La sentencia se ejecutará pero no provocará ningún cambio.

CUESTIÓN 3

La tabla PRODUCT contiene las siguientes columnas:

ID	NUMBER(7)	PK
SALE_PRICE	NUMBER(7,2)	

Evalúa las siguientes sentencias SQL:

```
1.   SELECT     MAX(sale_price), MIN(sale_price),
     AVG(sale_price) FROM  product;

2.   SELECT     ROUND(MAX(sale_price),2),
     ROUND(MIN(sale_price),2), ROUND(AVG(sale_price),2)
     FROM       product
     GROUP BY   sale_price;
```

¿En qué se diferencian los resultados?

A. Una de las sentencias generará un error.
B. La consulta 2 solo mostrará una fila como resultado y la consulta 1 podría mostrar más de una fila como resultado.
C. La consulta 1 mostrará 3 valores, mientras que la consulta 2 mostrará tres valores por cada SALE PRICE.
D. La consulta 1 mostrará un resultado por cada SALE PRICE, mientras que la consulta 2 mostrará un resultado por cada PRODUCT.

CUESTIÓN 4

Revisa la siguiente sentencia SQL:

```
SELECT  department "Departments",
        MAX(salary) "Top Salaries"
FROM   employee
WHERE  department IN(200, 300, 400)
GROUP BY Departments
HAVING  MAX(salary) > 60000;
```

Esta sentencia falla cuando se ejecuta. ¿Qué cambio corregiría el problema?

A. Quitar la función de agrupamiento de la cláusula HAVING.
B. Añadir la condición "MAX(salary) > 60000" a la cláusula WHERE.
C. Reemplazar el alias de la columna en la cláusula GROUP BY por el nombre de la columna.
D. Añadir la función de agrupamiento utilizada en la SELECT dentro de la cláusula GROUP BY.

CUESTIÓN 5

Dada la siguiente información sobre la tabla EMPLOYEE.

EMPLOYEE

Column Name	EMPLOYEE_ID	NAME	JOB	MANAGER	DATE_HIRED	SALARY	BONUS	DEPTARTMENT_ID
Key Type	PK							
Nulls/Unique	NN, U				NN			NN
FK Table								
FK Column								
Datatype	NUM	VARCHAR2	VARCHAR2	NUM	DATE	NUM	NUM	NUM
Length	6	20	9	6		7,2	7,2	2

Indicar qué sentencia SQL se ejecutaría correctamente:

```
A. CREATE FORCE VIEW last_first_vu
   AS SELECT name ||', '|| job "Employee & Position"
   FROM employee
   WHERE deptartment_id = 30
B. CREATE FORCE VIEW last_first_vu
   AS SELECT name ||', '|| job "Employee & Position",
   deptartment_id
   FROM employee
   ORDER BY "Employee & Position";
```

No se puede utilizar el alias en el ORDER BY.

```
C. CREATE FORCE VIEW last_first_vu
   AS SELECT name ||', '|| job "Employee & Position",
   deptartment_id "department"
   FROM employee
   WHERE deptartament = 10;
D. CREATE FORCE VIEW last_first_vu
   AS SELECT name ||', '|| job "Employee & Position",
   deptartment_id "department"
   FROM employee
   ORDER BY name
   GROUP BY deptartment_id;
```

El nombre del campo es DEPTARTAMENT_ID.

No se puede agrupar porque no hay funciones de agrupamiento en la SELECT.

CUESTIÓN 6

¿Cuál de las sentencias que aparecen, utilizarías para borrar la restricción de clave primaria EMPLOYEE_ID_PK y todas las restricciones dependientes de la tabla EMPLOYEE?

```
A. ALTER TABLE employee
   DROP PRIMARY KEY CASCADE;
B. ALTER TABLE employee
   DELETER PRIMARY KEY CASCADE;
C. MODIFY TABLE employee
   DROP CONSTRAINT employee_id_pk CASCADE;
D. ALTER TABLE employee
   DELETE PRIMARY KEY employee_id_pk CASCADE;
```

CUESTIÓN 7

Se quieren visualizar los nombres de empleados que contengan una "a" como segunda letra. ¿Cuál de las sentencias que aparecen utilizarías?

```
A. SELECT name
   FROM employee
   WHERE name LIKE '_a%'
B. SELECT lower(name)
   FROM employee
   WHERE name = '_A%'
C. SELECT name
   FROM employee
   WHERE lower(name) LIKE '_A%'
```

```
D. SELECT name
   FROM employee
   WHERE name = '_a%'
   OR         name = '_A%'
```

CUESTIÓN 8

Revisa la siguiente sentencia SQL:

```
SELECT  ename, empno, sal
FROM     emp
WHERE  deptno = (SELECT  deptno
                 FROM     dept
                 WHERE  UPPER(loc) = UPPER('&loc'))
```

Cuando ejecutas esta sentencia, qué podría ocurrir. (Elegir 2 respuestas).

A. La sentencia se ejecuta correctamente si la columna LOC en la tabla DEPT tiene valores únicos.
B. La sentencia falla porque no se puede usar un símbolo & en una subquery de una sentencia SELECT.
C. La sentencia falla si la subquery devuelve más de una fila y se pretende comparar mediante igualdad con el campo DEPTNO.
D. La sentencia se ejecuta correctamente, pero no devuelve los resultados esperados a causa de usar dos veces la función UPPER.

CUESTIÓN 9

¿Bajo qué circunstancias se debería crear un índice en una tabla?

A. Cuando la tabla es pequeña.
B. Cuando la tabla se actualiza frecuentemente.
C. Cuando los valores de una columna son estáticos y contienen un rango pequeño de valores.
D. Cuando se utilizan 2 columnas en una cláusula WHERE de una SELECT mediante un join.

CUESTIÓN 10

Revisa la siguiente sentencia SQL:

```
SELECT  ename, emp_number, salary
FROM     employee
WHERE  dept_number = (SELECT  dept_number
                      FROM    department
                      WHERE  location IN('CHICAGO',
                                         'ATLANTA'))
```

¿Por qué puede devolver un error esta sentencia?

A. Porque una subquery de múltiples filas devuelve una fila.
B. Porque una subquery de múltiples columnas devuelve una columna,
C. Porque una subquery de una única fila devuelve más de una fila.
D. Porque una query de múltiples filas usa una subquery de una única fila.
E. Porque una query de una única fila usa una subquery de múltiples filas que devuelve solo una fila.

CUESTIÓN 11

La tabla EMPLOYEE contiene las columnas siguientes:

FIRST_NAME	VARCHAR2(25)
LAST_NAME	VARCHAR2(25)
JOB	VARCHAR2(25)
SALARY	NUMBER(7,2)
DEPT_ID	NUMBER(3)

Se necesitan visualizar "first_name" y "last_name" de aquellos empleados con los siguientes datos:

1. El "last_name" solo puede ser: Brown, Chan, o Lindsey.
2. El "job" solo puede ser: Manager, Technician o Clerk.
3. El "salary" solo puede ser: > 30000.

Alguien nos presenta esta SQL para resolver el enunciado:

```
SELECT first_name, last_name
FROM employee
WHERE UPPER(last_name) IN ('BROWN', 'CHAN', 'LINDSEY')
```

```
AND UPPER(job) IN ('MANAGER', 'TECHNICIAN', 'CLERK')
AND salary <= 30000;
```

Se cumplen los requisitos 1 y 2. El requisito 3 no se cumple porque busca valores de salario <= 30000 y no mayores a 30000 que es lo que pide el enunciado.

¿Qué empleados se visualizarían?

A. Aquellos que cumplan un solo requerimiento de los indicados.
B. Aquellos que cumplan 2 requerimientos de los indicados.
C. Aquellos que cumplan los 3 requerimientos indicados.
D. Aquellos que no cumplan ninguno de los requerimientos.

CUESTIÓN 12

¿Cuál de las sentencias SQL es una query con join de igualdad entre 2 tablas?

```
A. SELECT   region.region_name, employee.salary
   FROM     region, employee
   WHERE    region.id = employee.region_no
B. SELECT   region.region_name, employee.salary
   FROM     region, employee
   WHERE    region.id = employee.region_no(+)
C. SELECT   region.region_name, employee.salary
   FROM     region, employee
   WHERE    employee.salary BETWEEN region.avg_salary AND
   region.max_salary
D. SELECT   region.region_name, employee info.last_name
   FROM     employee region, employee info
   WHERE    employee info.id >= region.manager_id
```

Esto es un JOIN

Esto no es un JOIN.

CUESTIÓN 13

La tabla SALE contiene estas columnas:

ID	NUMBER(9) PK
SALE_DATE	DATE

Se necesita crear un fichero script de sentencias SQL que solicite al usuario introducir un id. La fecha de venta para el número id proporcionado debería ser actualizada con la fecha del día.

¿Cuál de los scripts SQL*Plus usarías para obtener los resultados deseados?

```
A. UPDATE   sale
   SET      sale_date = sysdate
   /
B. UPDATE   sale
   SET      sale_date = sysdate
   WHERE    id = &id
   /
C. UPDATE   sale
   SET      sale_date = &sysdate
   WHERE    id = &id
D. UPDATE   sale(sale_date)
   SET      sale_date = sysdate
   WHERE    id = _&id
```

CUESTIÓN 14

¿Cuál de las query usarías para visualizar los nombres de todas las tablas a las que tienes acceso?

```
A. SELECT table_name
   FROM user_tables;
B. SELECT table_name
   FROM all_user_tables;
C. SELECT tname
   FROM tab
   WHERE tabtype = 'TABLE';
D. SELECT object_name
   FROM all_objects
   WHERE object_type = 'TABLE';
```

CUESTIÓN 15

David works as a Database Administrator for uCerty Inc. The company uses an oracle database. The database contains two tables, named Departments and Employees. The DepartmentID column of the Employees table references the DepartmentID column of the Departments table. The Departments and Employees tables contain the following records:

Departments:

DepartmentID	DepartmentName
10	Administration
20	Marketing
30	Finance
40	Human Resources

Employees:

EmployeeID	EmployeeName	DepartmentID	ManagerID	JobID	Salary
101	David	20	120	SA_REP	14000
102	Sam	10	105	CLERK	12500
103	Andrew	20	120	FIN_ADMIN	14200
104	Adrian	30	108	MAR_CLERK	12500
105	Maria	30	108	FIN_ADMIN	15000
106	Tracy	40	110	AD_ASST	13000
108	Kate	30	110	FIN_DIR	16500
110	Anne	40	120	EX_DIR	18000
120	Fran	20	110	SA_DIR	16500

David writes the following statement:

```
DELETE FROM Departments
WHERE DepartmentID = 40;
```

What will be the result of the statement when it is executed?

A. The statement will fail because column names are no specified in the DELETE statement.
B. All records associated with department ID 40 will be deleted from both the Departments and Employees tables.
C. The statement will fail because both the Employees and Departments tables contain records associated with department ID 40.
D. The record associated with department ID 40 will be deleted from the Departments table.
E. The records associated with department ID 40 will be deleted from the Employees table.

CUESTIÓN 16

You work as a Database Administrator for UCertify INC. The company uses an Oracle database. The database contains two tables, named Employees and Departments. The DepartmentID column of the Employees table references the DepartmentID column (PRIMARY KEY) of the Departments table. The database also contains a view named EmpView that is based on the Employees and Departments tables. The view contains three columns, named EmployeeID, EmployeeName, and DepartmentName. You want to add a new column named ManagerID to the view. Which of the following statements will you use to accomplish this?

```
A. ALTER VIEW EmpView AS
   SELECT EmployeeID, EmployeeName, DepartmentName, ManagerID
   FROM Employees e, Departments d
   WHERE e.DeparmentID = d.DepartmentID;
B. CREATE OR REPLACE VIEW EmpView AS
   SELECT EmployeeID, EmployeeName, DepartmentName, ManagerID
   FROM Employees e, Departments d
   WHERE e.DeparmentID = d.DepartmentID;
C. MODIFY VIEW EmpView (ADD ManagerID NUMBER);
D. ALTER VIEW EmpView (ADD ManagerID NUMBER);
E. MODIFY VIEW EmpView AS
   SELECT EmployeeID, EmployeeName, DepartmentName, ManagerID
   FROM Employees e, Departments d
   WHERE e.DeparmentID = d.DepartmentID;
```

CUESTIÓN 17

Samantha works as a Database Administrator for Blue Well Inc. The company uses an Oracle database. The database contains a table named Employees. Following is the structure of the table.

```
EmployeeID      NUMBER(5) PRIMARY KEY
EmployeeName    VARCHAR2(50)
DepartmentID    NUMBER(5) NOT NULL
Salary          NUMBER(7,2)
```

Samantha wants to display the identification numbers of departments the pay their employees above $10000. She also wants to display the maximum salaries paid by the departments. She writes the following SELECT statement to accomplish this:

```
SELECT DepartmentID DepID, MAX(Salary) MaxSal
FROM Employees
WHERE Salary > 10000
GROUP BY DeptID
ORDER BY MaxSal;
```

The statement generates an error on execution. Which of the following clauses of the statement is causing the error?

A. WHERE
B. GROUP BY
C. SELECT
D. FROM
E. ORDER BY

No se puede utilizar el alias de una columna para indicar el agrupamiento de la misma.

CUESTIÓN 18

View the Exhibit and examine the structure of the LOCATIONS and DEPARTMENTS tables. You need to display all those cities that have only one department. Which query gives the correct output?

A.
```
SELECT location_id, city
FROM locations l
WHERE 1 = (SELECT COUNT(*)
FROM departments
WHERE location_id = l.location_id);
```

B.
```
SELECT location_id, city
FROM locations WHERE EXISTS (SELECT COUNT(*)
FROM departments
GROUP BY location_id HAVING COUNT(*) = 1);
```

C.
```
SELECT location_id, city
FROM locations WHERE
1 = (SELECT COUNT(*) FROM departments
GROUP BY location_id);
```

D.
```
SELECT l.location_id, city
FROM locations l JOIN departments d ON (l.location_id =
d.location_id)
WHERE EXISTS (SELECT COUNT(*)
FROM departments d
WHERE l.location_id =d.location_id);
```

Tables"Exhibit"

LOCATIONS

Name	Null?	Type
LOCATION_ID	NOT NULL	NUMBER(4)
STREET_ADDRESS		VARCHAR2(40)
POSTAL_CODE		VARCHAR2(12)
CITY		VARCHAR2(30)
STATE_PROVINCE		VARCHAR2(25)
COUNTRY_ID		CHAR(2)

DEPARTMENTS

Name	Null?	Type
DEPARTMENT_ID	NOT NULL	NUMBER(4)
DEPARTMENT_NAME		VARCHAR2(30)
MANAGER_ID	NOT NULL	NUMBER(6)
LOCATION_ID		NUMBER(4)

CUESTIÓN 19

View the Exhibit and examine the structure of the EMP table. You want to display the names and salaries of only those employees who earn the highest salaries in their departments. Which SQL statement give the required output?

```
A. SELECT ename, sal
   FROM emp e
   WHERE sal = (SELECT MAX(sal)
   FROM emp
   WHERE deptno = e.deptno);
B. SELECT ename, sal
   FROM emp
   WHERE sal = ALL (SELECT MAX(sal)
   FROM emp
   GROUP BY deptno);
```

```
C. SELECT ename, sal
   FROM emp e
   WHERE EXISTS (SELECT MAX(sal)
   FROM emp WHERE deptno = e.deptno);
D. SELECT ename, sal
   FROM emp
   NATURAL JOIN (SELECT deptno, MAX(sal) sal
   FROM emp
   GROUP BY deptno);
```

Tables "Exhibit"

Name	Null?	Type
EMPNO	NOT NULL	NUMBER(4)
ENAME		VARCHAR2(10)
JOB		VARCHAR2(9)
HIREDATE		DATE
SAL		NUMBER(7,2)
DEPTNO		NUMBER(2)

CUESTIÓN 20

Evaluate the following SQL statement: (Note that the numbers 2,3 etc in the SQL statement are line numbers and not part of the syntax)

```
SQL> CREATE TABLE product
2    (prod_id NUMBER(3),
3    prod_name VARCHAR2(25),
4    qty NUMBER(7,2),
5    price NUMBER(10,2),
6    CONSTRAINT prod_id_pk PRIMARY KEY(prod_id),
7    CONSTRAINT prod_name_uq UNIQUE (prod_name),
8    CONSTRAINT price_nn NOT NULL (price));
```

What is the outcome of executing this command?

La restricción NOT NULL es la única que no permite que se le indique un nombre con CONSTRAINT.

A. It generates an error at line 6.
B. It generates an error at line 7.
C. It generates an error at line 8.
D. It executes successfully and creates the PRODUCTS table.

CUESTIÓN 21

Examine the structure of the DEPT table:

Name	Null?	Type
DEPTNO	NOT NULL	NUMBER(2)
DNAME		VARCHAR2(14)
LOC		VARCHAR2(13)

You successfully execute the following SQL statement:

```
SQL>CREATE TABLE emp
(emp_no NUMBER(3) PRIMARY KEY,
emp_name VARCHAR2(25) UNIQUE,
job_id VARCHAR2(10) NOT NULL,
deptno   NUMBER(2) REFERENCES dept(deptno),
salary NUMBER(10,2) CHECK (salary > 0));
```

For which columns would an index be generated automatically? (Choose all that apply)

A. EMP_NO
B. SALARY
C. JOB_ID
D. DEPT_NO
E. EMP_NAME

CUESTIÓN 22

Sam works as a Database Administrator for Gentech Inc. The company uses an Oracle database. The database contains two tables, named Employees and Departments. Following are the structures of the tables:

Employees:

```
EmployeeID      NUMBER(5)
EmployeeName    VARCHAR2(25)
```

Departments:

```
DepartmentID    NUMBER(6)
EmployeeID      NUMBER(5)
DepartmentName  VARCHAR2(25)
```

Sam queries the Employees and the Departments tables with the following SQL statement:

```
SELECT EmployeeName, DepartmentName
FROM Employees e, Departments d
WHERE Employees.EmployeeID = Departments.EmployeeID
ORDER BY 1,2
```

The statement generates an error on execution. Which of the following clauses in the statement is causing the error?

A. WHERE
B. SELECT
C. FROM
D. ORDER BY

Cuando se crean ALIAS para los nombres de tabla en una SELECT ya no se pueden nominar los campos con el nombre real de la tabla sino con su alias.

CUESTIÓN 23

Tom works as a Database Administrator for Tech Mart library. The library uses an Oracle database. The database contains a table named Books. Following is the structure of the table:

```
BookID          NUMBER(4)
BookTitle       VARCHAR2(25)
Price           NUMBER(5,2)
PurchaseDate    DATE
AuthorName      VARCHAR2(30)
```

Tom wants to display the titles of books that meet the following criteria:

- Purchased before January 17, 2003
- Price is either than $500 or greater than $1000

He also wants to sort book titles by their date of purchase, starting with the most recently purchased book. Which of the following statements will he use to accomplish this?

```
A. SELECT BookTitle
   FROM Books
   WHERE (Price < 500 or > 1000) AND (PurchaseDate < '17-
   JAN-2003')
   ORDER BY PurchaseDate DESC;
B. SELECT BookTitle
   FROM Books
   WHERE (Price IN (500,1000) AND (PurchaseDate < '17-JAN-
   2003')
   ORDER BY PurchaseDate ASC;
C. SELECT BookTitle
   FROM Books
   WHERE (Price < 500 or Price > 1000) AND (PurchaseDate <
   '17-JAN-2003')
   ORDER BY PurchaseDate DESC;
D. SELECT BookTitle
   FROM Books
   WHERE (Price BETWEEN 500 AND 1000) AND (PurchaseDate <
   '17-JAN-2003')
   ORDER BY PurchaseDate;
```

CUESTIÓN 24

Which of the following statements about a role are true?

Each correct answer represents a complete solution. Choose all that apply.

A. A role is created using the CREATE ROLE statement.
B. A role is a named group of related privileges, which can only be assigned to users.
C. A role can comprise a maximum of one hundred privileges.
D. Privileges are assigned to a role by using the GRANT statement.
E. A role can be assigned to a maximum of 1000 users.
F. A user can be assigned several roles, and a single role can be assigned to several users.

CUESTIÓN 25

Bob created the ROOM table that contains these columns:

```
ID          NUMBER(9) PK
NAME        VARCHAR2(25)
LOCATION    VARCHAR2(15)
SQ_FT       NUMBER(3)
```

Bob granted all users the INSERT privilege on the ROOM table. You inserted three records into the ROOM table without committing the changes. Bob issues this statement:

```
SELECT * FROM room;
```

Which statements regarding the visibility of records are true? (Choose all that apply).

A. Bob will be able to access the ROOM table.
B. Bob will be able to insert the same records into the ROOM table.
C. Bob will see the three records you inserted into the ROOM table.
D. Bob will NOT be able to access the ROOM table.
E. Bob will NOT see the three records you inserted into the ROOM table.
F. Bob will NOT be able to insert the same records into the ROOM table.

CUESTIÓN 26

The SUPPLIER table contains these columns:

```
S_ID              NUMBER PK
NAME              VARCHAR2(30)
LOCATION_ID       NUMBER
ORDER_DT          DATE
ORDER_AMOUNT      NUMBER(8,2)
```

Which clauses represent valid uses of aggregate functions? (Choose all that apply.)

```
A. FROM MAX(order_dt)
B. SELECT SUM(order_dt)
C. SELECT SUM(order_amount)
D. SELECT MAX(AVG(order_amount))
E. WHERE MIN(order_amount) = order_amount
F. SELECT location_id, order_dt, MAX(order_amount)
```

CUESTIÓN 27

Line_item

LINE_ITEM_ID	NUMBER(9)	NOT NULL, Primary Key
ORDER_ID	NUMBER(9)	NOT NULL, Primary Key, Foreign Key to ORDER_ID column of the CURR_ORDER table
QUANTITY	NUMBER(9)	

Click the Exhibit(s) button to examine the structure of the LINE_ITEM table.

You want to display order id numbers, product id numbers, and the quantity of the product ordered with these desired results:

1. The volume of the item ordered must be 50 or greater.
2. The display must be sorted from the lowest to the highest by order number and then by product number.
3. The items must belong to order numbers ranging from 1800 to 1900.

Evaluate this SQL script:

```
SELECT order_id, product_id, quantity
FROM line_item
WHERE quantity >= 50
AND order_id IN (1800,1900)
ORDER BY order_id, product_id;
```

What does the proposed solution provide?

A. One of the desired results.
B. Two of the desired results.
C. All of the desired results.
D. An error statement.

Se cumplen los puntos 1 y 2 del enunciado. El punto 3 no se cumple porque según está la consulta, solo buscaría pedidos con nº 1800 o 1900 y el enunciado solicitaba buscar pedidos desde el 1800 al 1900.

CUESTIÓN 28

Which three statements concerning explicit data type conversions are true. (Choose three.)

A. A number value may be converted to a date value using the TO_DATE function.
B. A date value may be converted to a number value using the TO_NUMBER function.
C. A character value may be converted to a date value using the TO_DATE function.
D. A date value may be converted to a character value using the TO_DATE function.
E. A date value may be converted to a character string using the TO_CHAR function.
F. A number value may be converted to a character string using the TO_CHAR function.
G. A number value may be converted to a character value using the TO_NUMBER function.

CUESTIÓN 29

Evaluate this SQL statement:

```
SELECT i.id_number, m.id_number
FROM inventory i, manufacturer m
WHERE i.manufacturer_id = m.id_number
ORDER BY 1;
```

Which clause prevents all the rows in the INVENTORY table from being joined to all the rows in the MANUFACTURER table?

```
A. ORDER BY 1;
B. SELECT i.id_number, m.id_number
C. FROM inventory i, manufacturer m
D. WHERE i.manufacturer_id = m.id_number
```

CUESTIÓN 30

The STUDENT table contains these columns:

```
ID              NUMBER(9) PK
FIRST_NAME      VARCHAR2(25)
LAST_NAME       VARCHAR2(25)
ENROLL_DATE     DATE
```

You need to create a script to display a student's enrollment date and projected graduation date.

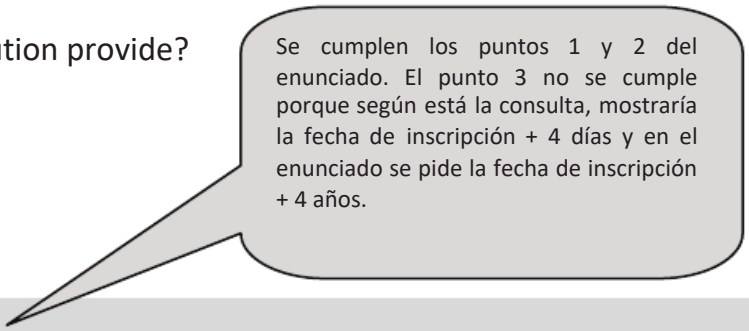
These are the desired results:

1. Prompt the user for a student id.
2. Display the student's first name, last name and date of enrollment.
3. Display the student's projected graduation date by adding four years to the enrollment date value.

Evaluate this SQL*Plus script:

```
SELECT           CONCAT(first_name,           last_name),
TO_CHAR(enroll_date, 'fmDD MONTH YYYY'), enroll_date + 4
grad_date
FROM student
WHERE id = &id
/
```

What does the proposed solution provide?



A. One of the desired results.
B. Two of the desired results.
C. All of the desired results.
D. A syntax error.

CUESTIÓN 31

What will be the output of the following query?

```
SELECT product_name
FROM Products
WHERE price IN (SELECT price FROM Products
                WHERE product_category = 3 OR
                      product_category = 5);
```

A. It will display the name of all products having the same price as products of category 3.
B. It will display the name of all products having the same price as products of category 3 or 5.
C. It will display the name of all products having the same price as products of category 3 and 5.
D. It will display the name of all products having the same price as products of category 5.

CUESTIÓN 32

Martin works as a Database Administrator for MTech Inc. He designs a database that has a table named Products. He wants to create a report listing different product categories. He does not want to display any duplicate row in the report. Which of the following SELECT statements will Martin use to create the report?

A.
```
SELECT Product_No, Prod_Category
FROM Products;
```

```
B. SELECT Product_No, Prod_Category
   FROM Products
   GROUP BY Product_No ORDER BY Product_No;
C. SELECT Product_No, Prod_Category
   FROM Products
   GROUP BY Product_No;
D. SELECT DISTINCT Product_No, Prod_Category
   FROM Products;
```

CUESTIÓN 33

Martha writes the following statement to create a view:

```
1. CREATE VIEW My_View
2. AS SELECT Acc_ID, Acc_Description
3. FROM Accounts
4. WHERE Acc_ID = 457
5. ORDER BY Acc_Description;
```

Which of the following statements is correct about the view?

A. Line 4 will generate an error on execution.
B. Line 1 will generate an error on execution.
C. Line 5 will generate an error on execution.
D. There is no error.

CUESTIÓN 34

Which of the following statements are correct?

A. A correlated subquery is evaluated for each row processed by the parent statement.
B. A correlated subquery is evaluated once and the result is used in the parent statement.
C. A correlated subquery is executed and each row from the result is evaluated against the parent statement.
D. When a correlated subquery is evaluated, the parent statement is evaluated for each row processed by the subquery.

CUESTIÓN 35

Maria works as a Database Administrator for Blue Well Inc. She creates a user, named User1. She wants to give User1 the privilege to modify only the column "column1" of a table named table1, in the database. Which of the following statements will she use to accomplish this task?

```
A. GRANT UPDATE ON table1 columns(column1) TO User1;
B. GRANT UPDATE ON table1.column1 TO User1;
C. GRANT UPDATE ON table1(column1) TO User1;
D. GRANT UPDATE (column1) ON table1 TO User1;
```

CUESTIÓN 36

Which of the following statements is used to enable PRIMARY KEY constraint on the Emp_ID column of the Employees table?

```
A. ALTER TABLE Employees
   ENABLE CONSTRAINT
   Emp_no_pk PRIMARY KEY (Emp_ID);
B. ALTER TABLE Employees
   ENABLE PRIMARY KEY;
C. ENABLE CONSTRAINT
   Emp_no_pk PRIMARY KEY (Emp_ID);
D. UPDATE TABLE Employees
   MODIFY CONSTRAINT
   Emp_no_pk PRIMARY KEY (Emp_ID);
```

CUESTIÓN 37

Martha writes the following query to display information from the Ordinates table:

```
SELECT Ord_No, DISTINCT Ord_Name, Salary
FROM Ordinates;
```

Which of the following statements is true about this query?

A. The query will display all values for Ord_No, unique values for Ord_Name, and all values for Salary.

B. DISTINCT is not a valid keyword in Oracle.
C. The query will display all values for Ord_No, unique values for Ord_Name, and unique values for Salay.
D. The query will give an error on execution.

La instrucción DISTINCT debe englobar a toda la SELECT y ser la primera después de la cláusula SELECT.

CUESTIÓN 38

Patrick wants to create a sequence for primary columns of a table. He wants the sequence to start at 2000, increment by 400, and generate a maximum value of 20000. He does not want the sequence to repeat numbers after reaching the maximum value. Which of the following statements will Patrick use to create the sequence?

```
A. CREATE SEQUENCE New_Sequence
   START WITH 2000
   INCREMENT BY 400
   MAXVALUE 20000
   CYCLE;
B. CREATE SEQUENCE New_Sequence
   START WITH 2000
   INCREMENT BY 400
   MAXVALUE 20000
   NOCYCLE;
C. CREATE SEQUENCE New_Sequence
   START WITH 2000
   MAXVALUE 20000
   NOCYCLE;
D. CREATE SEQUENCE New_Sequence
   INCREMENT BY 400
   MAXVALUE 20000
   NOCYCLE;
```

CUESTIÓN 39

Samantha works a Database Administrator for SamTech Inc. She creates a database that contains a table named Ordinate. Which of the following SQL

statements will Samantha use to display the name of all the ordinates that not have any subordinate?

```
A. SELECT Ord.Ord_Name
   FROM Ordinate Ord
   WHERE Ord.Ord_No <> (SELECT Subord.SubOrd_No
                        FROM Ordinate Subord);
B. SELECT Ord.Ord_Name
   FROM Ordinate Ord
   WHERE Ord.Ord_No NOT IN (SELECT Subord.SubOrd_No
                            FROM Ordinate Subord);
C. SELECT Ord.Ord_Name
   FROM Ordinate Ord
   WHERE Ord.Ord_No != (SELECT Subord.SubOrd_No
                        FROM Ordinate Subord);
D. SELECT Ord.Ord_Name
   FROM Ordinate Ord
   WHERE Ord.Ord_No NOT HAVING (SELECT Subord.SubOrd_No
                                FROM Ordinate Subord);
```

CUESTIÓN 40

What is the use of index in a database?

Each correct answer represents a complete solution. Choose two.

A. Index enforces entity integrity.
B. Index reduces data storage size.
C. Index enforces referential integrity.
D. Index improves data retrieval performance.

CUESTIÓN 41

Patrick writes and executes the following query;

```
SELECT deptno, deptname
FROM dept
WHERE deptno = 101 or deptno = 202;
```

Which of the following operators will Patrick use to replace the OR condition in the WHERE clause?

```
A. <=;
B. BETWEEN…AND…;
C. LIKE
D. IN
E. >=
```

CUESTIÓN 42

Which of the following statements will you use to create a view named My_view?

```
A. CREATE My_view VIEW…;
B. CREATE VIEW My_view…;
C. CREATE OR REPLACE VIEW My_view…
D. VIEW My_view…
```

CUESTIÓN 43

Which of the following statements are true about views?

Each correct answer represents a complete solution. Choose two.

A. A view can be used to restrict a user to specific columns in a table.
B. A view is used to speed up data retrieval.
C. A view is a brief description of a particular database.
D. A view represents a subset of table attributes and is designed to facilitate a particular circumstance.

CUESTIÓN 44

What is the relationship between foreign key and primary key?

Each correct answer represents a complete solution. Choose all that apply.

A. A foreign key constraint works in conjunction with a primary key constraint to enforce referential integrity among related entities.
B. A foreign key and primary key create a link between two entities.
C. A foreign key ties attribute(s) of a entity to the primary key of another entity, for the purpose of creating a dependency.
D. There is no relationship between a primary key and a foreign key.

CUESTIÓN 45

Allen works as a Database Administrator for SamTech Inc. He creates a database containing a table, named Employees. The Employees table contains the following data:

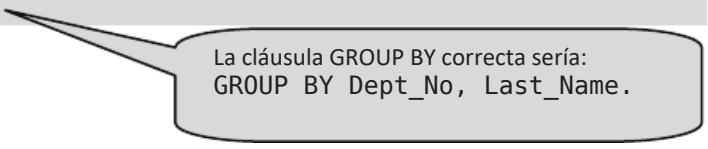
Emp_ID	Last_Name	First_Name	Salary	Dept_No
9	Brown	William	21000	155
6	Robert	William	20000	234
4	Peterson	David	25000	102
3	Thomas	Samantha	30000	155
2	Jackson	David	22000	234
5	Krey	Vivian	23000	233
1	Patrick	Arnold	25000	234
7	Oliver	John	31000	102

He writes the following query to retrieve records from the Employees table:

```
SELECT Dept_No, Last_Name, SUM(Salary)
FROM Employees
WHERE Salary < 30000
GROUP BY Dept_No
ORDER BY Last_Name;
```

Allen gets an error on executing the query. Which of the following is the cause of the error?

```
A. WHERE Salary < 30000
B. GROUP BY Dept_No
C. ORDER BY Last_Name
D. FROM Employees
```



REFERENCIAS Y MATERIAL ANEXO EN INTERNET

REFERENCIAS UTILIZADAS PARA EL CURSO

Para la elaboración de la documentación contenida en este libro se han utilizado como fuentes de referencia técnica:

- Documentación oficial que proporciona Oracle en su web para la versión 12c. En el siguiente apartado se indican enlaces a la misma.
- Preguntas y respuestas de ejemplo para la presentación a las certificaciones de Oracle.
- Documentación elaborada durante el desempeño de la actividad profesional, fruto de la experiencia de más de 20 años, adquirida en los distintos puestos de trabajo desempeñados con el manejo de diversas versiones de los productos de Oracle desde la versión Oracle 7 a la versión Oracle 12c.
- Documentación que recoge la experiencia formativa, sugerencias y preguntas técnicas surgidas durante la impartición de este curso en distintas academias de formación técnica y en empresas privadas.

ENLACES A ORACLE

Si necesita ampliar información, conocer nuevos productos y descargar software de Oracle, puede acudir a los enlaces oficiales de la web de Oracle que se indican a continuación.

PÁGINA OFICIAL DE ORACLE INTERNACIONAL

La web oficial de Oracle a nivel internacional se encuentra en el enlace siguiente:

```
http://www.oracle.com/
```

PÁGINA OFICIAL DE ORACLE ESPAÑA

La web oficial de Oracle en España se encuentra en el enlace siguiente:

```
http://www.oracle.com/es/index.html
```

RED TECNOLÓGICA DE ORACLE

La web que aglutina la comunidad de desarrolladores, DBAs y arquitectos de Oracle se encuentra en el enlace siguiente:

```
http://www.oracle.com/technetwork/index.html
```

Esta página es el mejor punto para ampliar información y descargar software oficial de Oracle. Asimismo, encontrará tutoriales y ejemplos de scripts y código de los diversos productos de Oracle. También podrá compartir información y experiencias en los foros disponibles para desarrolladores, DBAs y arquitectos.

PROGRAMA DE CERTIFICACIÓN EN PRODUCTOS DE ORACLE

La web que informa sobre todo el proceso de certificación en los productos de Oracle se encuentra en el siguiente enlace:

```
http://education.oracle.com/pls/web_prod-plq-
dad/db_pages.getpage?page_id=39
```

UNIVERSIDAD DE ORACLE

La web para la formación oficial de Oracle en sus productos se encuentra en el siguiente enlace:

```
http://education.oracle.com/pls/web_prod-plq-
dad/db_pages.getpage?page_id=3
```

DOCUMENTACIÓN OFICIAL DE ORACLE DE LA VERSIÓN 12C

La web donde puede encontrar toda la documentación oficial de Oracle sobre la versión 12c de base de datos, se encuentra en el enlace siguiente:

```
http://docs.oracle.com/en/database/database.html
```

GUÍA DE INSTALACIÓN DE ORACLE 11G XE

IV

INTRODUCCIÓN

Para el mejor aprovechamiento de este curso se aconseja la instalación de la base de datos de Oracle 11g Express Edition, dado que es una versión reducida de libre distribución que consume pocos recursos de máquina y que permite llevar a cabo todas las prácticas y supuestos que se especifican en este libro.

Puede descargar este producto desde el siguiente enlace de la web de Oracle previo registro gratuito:

http://www.oracle.com/technetwork/database/express-edition/downloads/index.html

A la finalización de la redacción de este libro, la última versión que ofrece Oracle sobre la base de datos Express Edition es la 11.2 Express Edition Beta.

En este anexo podrá encontrar una guía práctica para la instalación de este producto Oracle en su PC.

REQUERIMIENTOS MÍNIMOS

Los requerimientos de máquina para la instalación de Oracle 11.2 Express Edition Beta son los siguientes:

- Sistema operativo Windows 32 bits:
  - Windows 2000 Service Pack 4 o posterior.
  - Windows Server 2003.
  - Windows XP Professional Service Pack 1 o posterior.
  - Compatible con Windows Vista y Windows 7.
- Protocolo TCP/IP:
  - El equipo debe de disponer de una tarjeta de red fija o inalámbrica.
- Espacio en disco necesario:
  - 1,6 GB.
- Memoria RAM:
  - 265 MB mínimo. Se recomienda 512 MB.
- Microsoft Windows Installer (MSI) disponible en el equipo:
  - MSI versión 2.0 o posterior.
  - Si no dispone de este componente, lo puede descargar desde la página web http://msdn.microsoft.com.

TUTORIAL DE INSTALACIÓN

A continuación, se muestran los pasos a seguir para la instalación de Oracle 11.2 Express Edition Beta.

Paso 1: Descarga del producto

Descargue Oracle 11.2 Express Edition Beta desde la web de Oracle.

Paso 2: Configuración del equipo para la instalación

Antes de comenzar la instalación, debe de seguir estos pasos para preparar su equipo para la instalación:

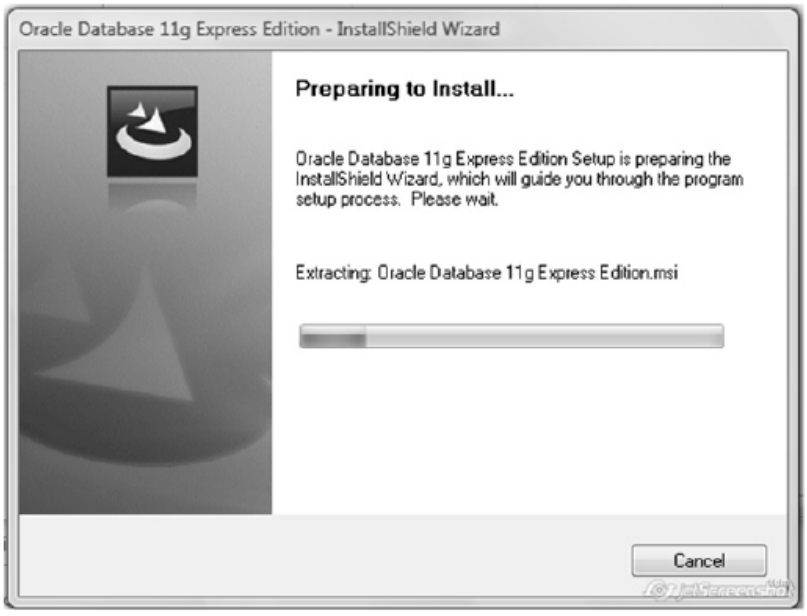
- Cree una carpeta con el nombre *OracleXE11gR2*.
- Extraiga el fichero de instalación *win32_11gR2_OracleXE.zip* en la carpeta *OracleXE11gR2* creada en el punto anterior.

Paso 3: Ejecute la instalación

Para comenzar la instalación debe de ejecutar el fichero *setup.exe* que encontrará en la carpeta *\OracleXE11gR2\DISK1*.

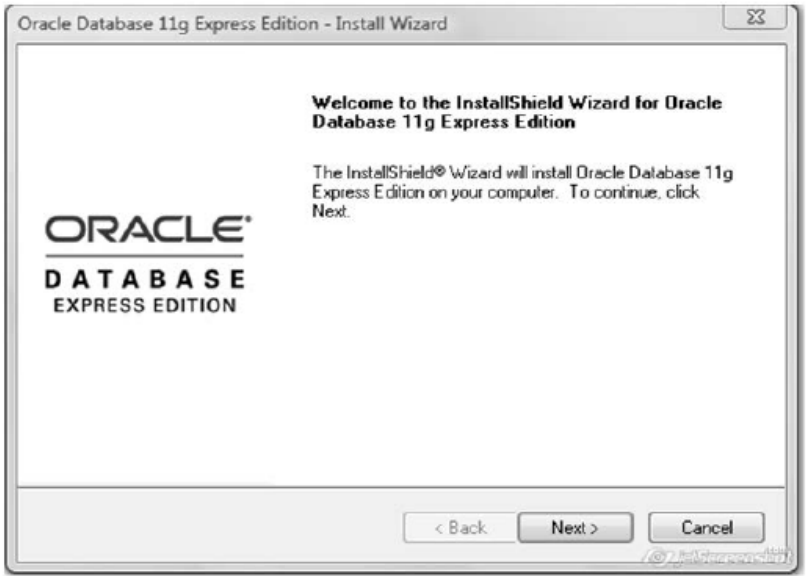
Paso 4: Comienza el proceso de instalación

Una vez ejecutado el fichero de instalación se mostrará una pantalla con el progreso de la preparación para la instalación.



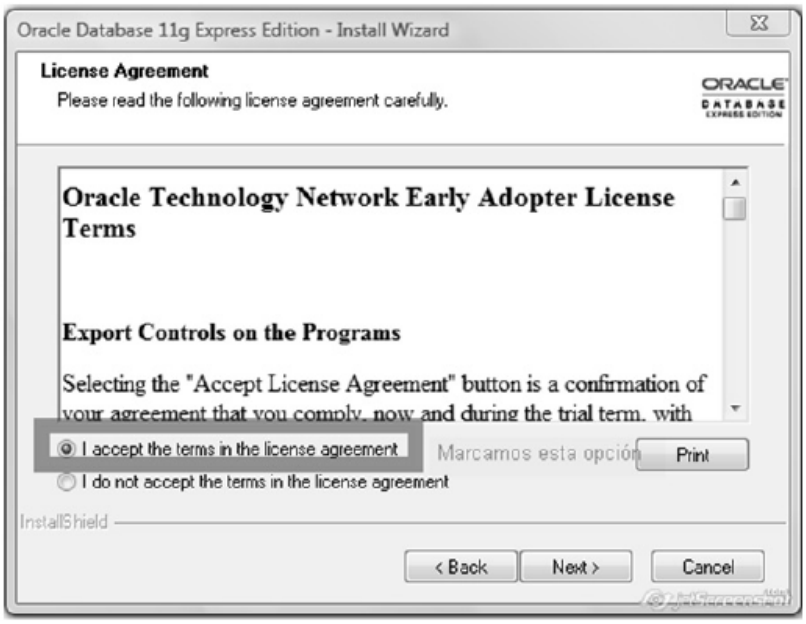
Paso 5: Pantalla de bienvenida

Concluida la preparación para comenzar la instalación se mostrará una pantalla de bienvenida en la que deberá pulsar el botón *Next* para comenzar el proceso de instalación de la base de datos en su equipo.



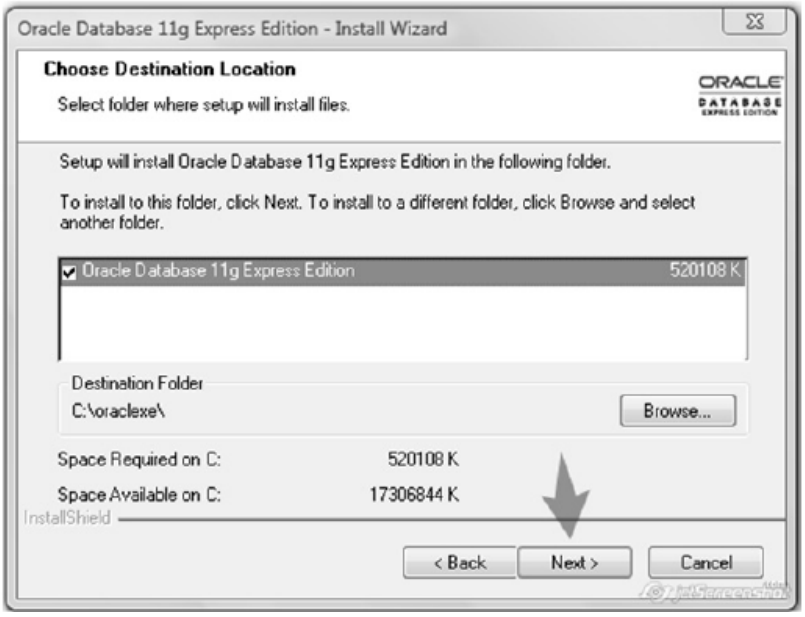
Paso 6: Términos de la licencia

A continuación, se le mostrarán los términos de la licencia de uso del producto para su aceptación. Debe marcar la opción *I accept the terms in the license agreement* y posteriormente pulsar el botón *Next.*



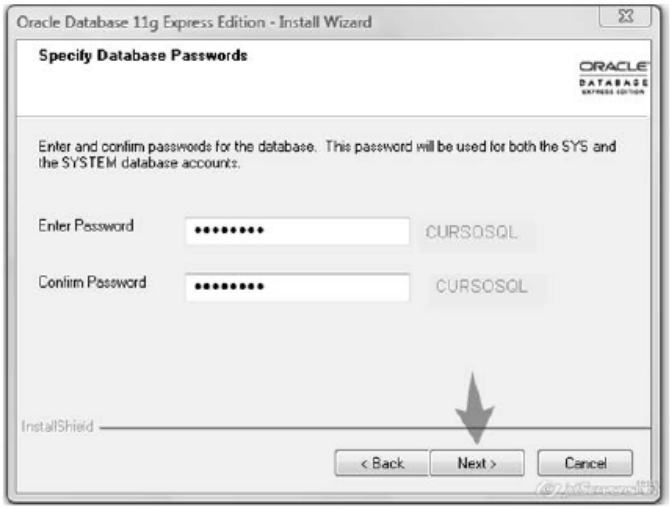
Paso 7: Ubicación para la instalación

Mantenga la ubicación por defecto de la instalación y pulse el botón *Next.*



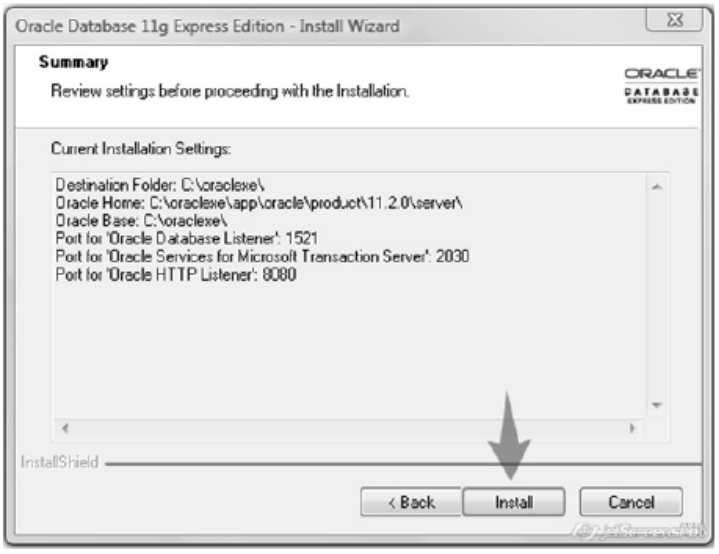
Paso 8: Contraseña para los usuarios SYS y SYSTEM

En este punto de la instalación el sistema nos solicita que introduzcamos una contraseña para el acceso a la base de datos de los usuarios administradores SYS y SYSTEM. Debe introducir la misma contraseña 2 veces. Se recomienda para el seguimiento de este curso que la contraseña que introduzca sea *CURSOSQL* y a continuación pulse el botón *Next.*



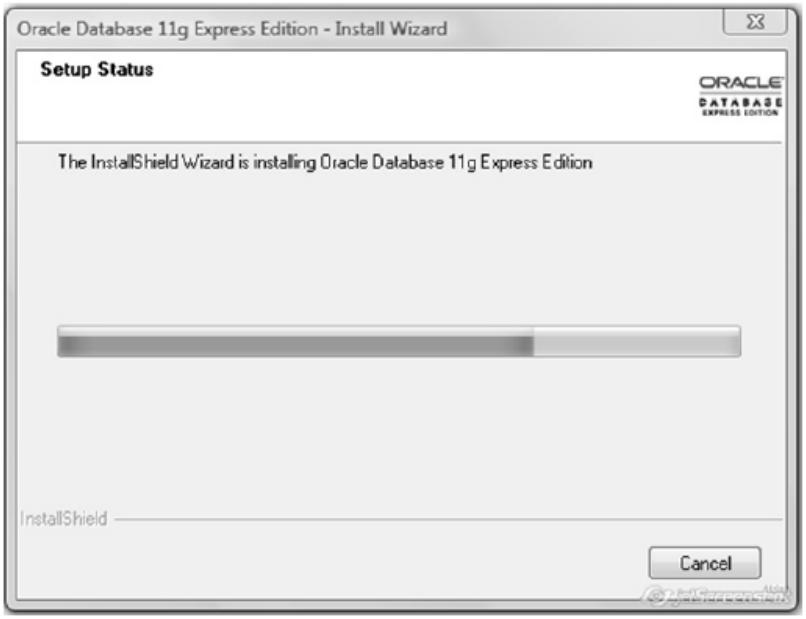
Paso 9: Resumen previo a la instalación

Justo antes de comenzar la instalación física de acuerdo a los parámetros introducidos en las pantallas previas, se presenta una pantalla resumen con las opciones que se van a instalar, así como los puertos que se habilitarán después de la instalación, para poder acceder a la base de datos. Pulse el botón *Install*.



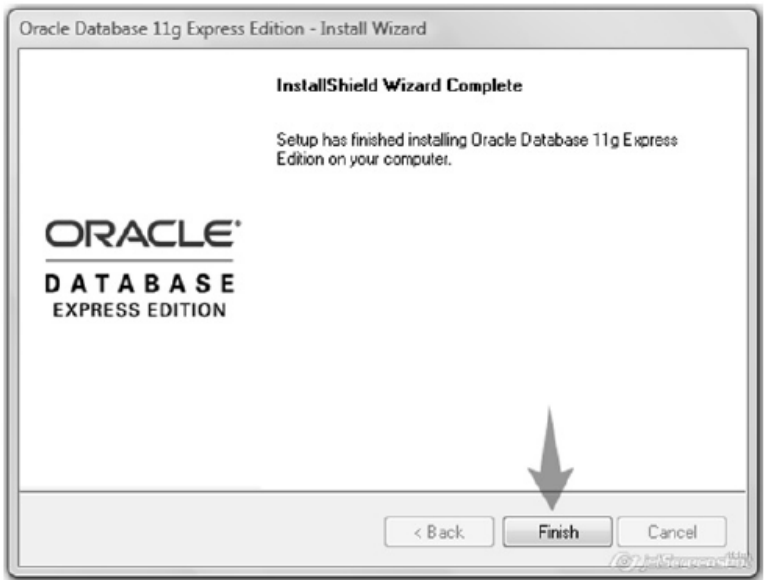
Paso 10: Progreso de la instalación

En esta fase el usuario recibirá en pantalla una serie de imágenes con el progreso de la instalación en su equipo. No debe pulsar ningún botón hasta que concluya la instalación.



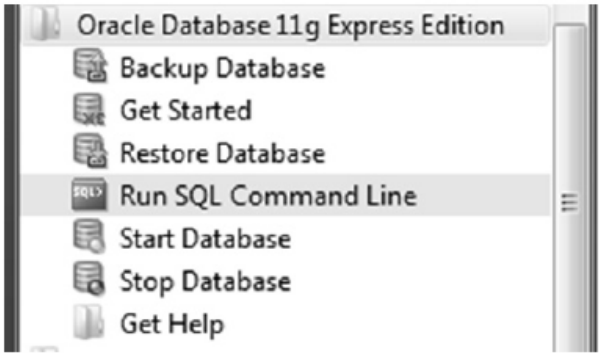
Paso 11: Finalización de la instalación

La instalación finalizará cuando se presente una pantalla como la que se muestra a continuación, con un botón *Finish*, que deberá pulsar para concluirla.



Paso 12: Familiarizándose con los elementos instalados

Finalizada la instalación se encontrará con el siguiente grupo de elementos instalado en su equipo.



E igualmente le aparecerá el siguiente icono en su escritorio.

El funcionamiento básico de estos elementos es el siguiente:

BACKUP DATABASE

Permite realizar un backup o salvaguarda de la base de datos.

GET STARTED

Abre el sistema de gestión administrativa de la base de datos, a través del explorador de su equipo.

RESTORE DATABASE

Permite realizar un restore o recuperación de una salvaguarda que se haya realizado de la base de datos, con la utilidad backup comentada anteriormente.

RUN SQL COMMAND LINE

Permite ejecutar sentencias y scripts contra la base de datos. Es en esta herramienta donde deberá probar la ejecución de las sentencias SQL que se han explicado durante este curso, al igual que la usará para la resolución de los supuestos prácticos que se le han sugerido en el libro.

START DATABASE

Permite arrancar la base de datos para poderse conectar con ella. Normalmente esta utilidad queda configurada en el equipo para que se ejecute automáticamente cuando arranque el ordenador.

STOP DATABASE

Permite parar la base de datos.

ÍNDICE ANALÍTICO

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

X