ORACLE12c.Lenguaje SQL

Autor: CLEFormación S.L

Localidad y año de impresión: Madrid, 2011

Capítulo 1. Introducción a las Bases de Datos Relacionales

Introducción a Bases de Datos....1
Propuesta de un método estándar de diseño....3
Los sistemas de gestión de bases de datos....5
Objetivos de los SGBDR....6
Visión metodológica de Oracle....6
Definición de base de datos....6
Ficheros de control....6
Ficheros de REDO LOG....8
Ficheros de datos....9
Tablespaces....10
Definición y usos....10
Tipología de tablespaces....11
Los usuarios....11

Capítulo 2. Introducción al Lenguaje SQL

Introducción al lenguaje SQL....1
Estándares SQL. SQL: 99....1
Funcionamiento del lenguaje SQL....1
Componentes del lenguaje SQL....1
DDL (Data Definition Languaje - Lenguaje de Definición de Datos)....2
DCL (Data Control Languaje - Lenguaje de Control de Datos)....2
DML (Data Manipulation Languaje - Lenguaje de Manipulación de Datos)....2
Objetos de base de datos....3
Objetos de un esquema....3
Objetos no pertenecientes a esquemas....3
Reglas para nombrar objetos de base de datos....4
Referenciar objetos de la base de datos....4
Resolución de referencias a objetos....5
Tipos de datos en Oracle....5

Capítulo 3. Lenguaje de Manipulación de Datos. DML

Introducción a DML....1
Sentencia SELECT....1
Selección de todos los datos de una tabla....1
Selección de ciertas columnas de una tabla....2
Cláusula FROM....2
Cláusula WHERE....3
Cláusula ORDER BY....3
Cláusula GROUP BY....4
Operadores SQL....4
Operadores aritméticos....5
Operadores concatenación....5
Operadores de conjuntos....5
Condiciones SQL....7
Condiciones de comparación....7
Condiciones lógicas....8
Condiciones de pertenencia....9
Condiciones de rango de valores....9
Condición NULL....10
Condición EXISTS....10
Condición LIKE....11
Funciones....12
Funciones Numéricas....12
Funciones de cadena de caracteres que devuelven caracteres....12
Funciones de cadena de caracteres que devuelven números....13
Funciones de fecha y hora....13
Funciones de conversión....13
Máscaras de formato. Formatos numéricos....13
Máscaras de formato. Formatos tipo fecha....14
Otras funciones....16
Variables de entorno....16
Funciones de grupo. La cláusula GROUP BY....17
Funciones de grupo. La cláusula HAVING....19
Combinaciones de Tablas en versiones anteriores a la 9i....20
EQUI JOIN....20
OUTER JOIN....21
Uniones de SQL: 1999....23
Uniones cruzadas....23

Uniones naturales ... 23
Predicados de Unión y la Cláusula ON ... 24
OUTER JOIN. Uniones externas ... 25
La expresión CASE ... 26
CASE simple ... 26
CASE buscada ... 26
NULLIF ... 27
COALESCE ... 27
SUBCONSULTAS ... 27
Usando subconsultas de varias Columnas ... 30
subconsultas correlacionadas ... 30
subconsultas escalares ... 31
Operadores Exists,Not Exists,WITH ... 32
Recuperación jerárquica ... 33
DML. Lenguaje de Manipulación de Datos ... 36
INSERT ... 36
UPDATE ... 37
DELETE ... 38
MERGE ... 39
Sentencias Transaccionales ... 41
COMMIT ... 42
SAVEPOINT ... 42
ROLLBACK ... 42

Capítulo 4. Lenguaje de Definición y Lenguaje de Control de Datos. DML y DCL

DDL. Lenguaje de Definicion de Datos ... 1
Comandos DDL ... 1
CREATE TABLE ... 1
Sintaxis CREATE TABLE ... 1
Integridad de Datos ... 3
Reglas de Negocio ... 3
Integridad de Entidades ... 4
Integridad Referencial ... 4
Nombres de las Restricciones ... 5
Cómo Definir Una Restricción ... 6
Restricciones: características ... 6

Comprobación de restricciones aplazada 7
Restricciones obligatorias 8
Restricción UNIQUE o PRIMARY KEY utilizando un índice no único. 9
CREATE INDEX 9
Tablas tipo índice 10
CREATE SYNONYM 11
CREATE VIEW 12
CREATE SEQUENCE 13
ALTER TABLE 14
Añadir Restricciones 14
Desactivar Restricciones 15
Activar Restricciones 16
Excepciones Activando Restricciones 16
Borrar Restricciones 17
Borrar Columnas 18
ALTER SEQUENCE 18
DROP TABLE 18
DROP INDEX 19
DROP VIEW 19
DROP SYNONYM 19
DROP SEQUENCE 19
RENAME 19
TRUNCATE 20
Privilegios 20
Privilegios sobre objetos 20

ANEXO A. Funciones

Introducción 1
Funciones Numéricas 1
Funciones de cadena de caracteres que devuelven caracteres 4
Funciones de cadena de caracteres que devuelven números 7
Funciones de fecha y hora 8
Funciones de conversión 12
Otras funciones. 16

ANEXO B. El interfaz de usuario SQL*PLUS

Introducción 1
El Editor de SQL*PLUS 2
COMANDOS DE EDICION 2
El Comando SPOOL 4
El Comando SAVE 4
El Comando START 4
El Comando GET 5
El Comando EDIT 5
El Comando DESCRIBE 5
El Comando HOST 5
Parámetros de Sustitución 6
Mejoras en SQL*Plus a partir de Oracle 10g 7
Mejoras en el comando DESCRIBE 7
Mejoras en el comando SPOOL 7
Configuración de Sesión 7
Nuevas variables DEFINE 8
Variables de Sustitución en PROMPT 9
Compatibilidad SQL*Plus 9

ANEXO C. El interfaz de usuario iSQL*PLUS

Utlización de iSQL*Plus 1
Introducción a iSQL*Plus 1
Espacio de Trabajo iSQL*Plus 2
Visualización de Resultados 3
Configuración del Interfaz 4
Configuración de Sistema 5
Cambios en Comandos 7

Ejercicios Lenguaje SQL

Modelo de datos 1
Ejercicios sobre Select 1
Select Simples 1
Select Simples con Funciones 2
Subconsultas/joins 3
Ejercicios DML 4
Insert 4
Update 4
Delete 4
Control de transacciones. 4
Ejercicios ddl e integridad de datos. 5
Práctica DDL-DML 5
Implementación de restricciones de integridad durante la creación de una tabla 7
Implementación de restricciones de integridad después de creación de una tabla 7
Utilización de la tabla de excepciones 7
Creación de Vistas 8
Creación de Sinónimos 8
Soluciones 9
Select Simples 9
Select Simples con Funciones 11
Subconsultas / joins 14
Insert 16
Update 16
Delete 17
Práctica DDL-DML 17
Implementación de restricciones de integridad durante la creación de una tabla 21
Utilización de la tabla de excepciones 21
Vistas 22
Sinónimos 22

1

Introducción a las Bases de Datos Relacionales

Tabla de contenidos

Introducción a Base de Datos **1**

Propuesta de un método estándar de diseño **3**

Los sistemas de gestión de bases de datos **6**

Objetivos de los SGBDR **7**

Visión metodológica de Oracle **7**

Definición de base de datos **8**

Ficheros de control 8

Ficheros de REDO LOG 9

Ficheros de datos 10

Tablespaces **12**

Definición y usos 12

Tipología de tablespaces 12

Los usuarios **13**

Introducción a Base de Datos

Un sistema de información formal (S.I.) se establece por el diagrama estructural de la empresa, denominado organizacional y ha de tomar datos externos e internos de la propia empresa. Estos datos posibilitan la obtención de la información para la correcta gestión de la corporación, posibilitando la toma de decisiones y alcanzando así, los objetivos que previamente los directivos de la misma se marcan.

Por lo tanto un Sistema de Información se puede definir como un conjunto de elementos interrelacionados entre sí, personas, procedimientos y equipos, gobernados por ciertas reglas establecidas por la organización, y que obtienen una información necesaria para el cumplimiento de sus fines. Es por tanto necesario recopilar, procesar y almacenar datos, procedentes tanto de fuentes externas como internas, siendo necesarias ciertas facilidades para almacenar, recuperar y presentar dicha información. Estas facilidades nos las proporcionan los Sistemas gestores de Bases de datos (S.G.B.D.).

Estos datos se albergan en una base de datos, por lo tanto, una Base de Datos es una colección de datos interrelacionados entre sí, con redundancia controlada y una estructura que refleja las vinculaciones y restricciones existentes en la realidad. Los datos deben de ser compartidos por diferentes usuarios y aplicaciones, manteniendo la independencia de estos, con una definición y descripción de las estructuras de datos únicos e integrados en los mismos datos. Los procedimientos de manipulación de la información deben preservar la integridad, seguridad y confidencialidad del conjunto de datos.

Al sistema integrado por una serie de programas, procedimientos y lenguajes encargados de definir, manipular, controlar la coherencia, seguridad e integridad de la Base de datos, así como de procesar la información se denomina Sistema Gestor de Base de Datos. Uno de los principales objetivos de los SGBD es conseguir una independencia lógico/física. Esto permite poder alterar las características físicas o lógicas de los datos sin que dichos cambios incidan en los aspectos lógicos o físicos, respectivamente. Al mismo tiempo implica una independencia entre los datos y las aplicaciones, de manera que los futuros cambios tengan una repercusión mínima, además de una versatilidad y flexibilidad ante los avances de las nuevas tecnologías.

Para obtener un buen Sistema Informático, eficiente, flexible, con altos rendimientos y costes relativamente bajos es imprescindible utilizar una metodología. Así conseguiremos una menor especialización del diseñador, una participación más activa por parte del usuario final, facilitando la portabilidad y versatilidad, obligando a documentar las fases y mejorando así, el mantenimiento.

Por otra parte, en la determinación de las fases de una metodología es apropiado utilizar una jerarquía de niveles de abstracción, en el sentido de ser lo suficientemente amplias para que cada nivel le corresponda decisiones de diseño definidas correctamente. Las metodologías tienen tres grandes fases:

- Diseño Conceptual: en el que se realiza un estudio de los recursos de información del sistema independientemente de los usuarios, aplicaciones y eficiencias.
- Diseño Lógico: transforma el conceptual adaptándolo al modelo de datos en el que se apoya el Gestor de Base de Datos a utilizar, es decir, relacional, jerárquico, etc.

- Diseño Físico: consigue una instrumentación lo más eficientemente posible del diseño lógico.

Con los antecedentes señalados, se puede realizar el diseño de las bases de datos. En primer lugar, y acorde con los diferentes niveles de arquitectura de bases de datos reseñados, tiene lugar la construcción del modelo y del esquema conceptual de la base de datos:

- El esquema conceptual→ Puede definirse como una descripción abstracta y general de la parte o sector del universo real, que el contenido de la base de datos va a representar, llamada en ocasiones "universo del discurso".
- En este nivel de análisis se está tratando con una descripción de la realidad, no con datos, y suele contener listas de tipos de entidades, de las relaciones existentes entre esas entidades y de las restricciones de integridad que se aplican sobre ellas. El esquema conceptual de la base de datos puede utilizarse para integrar los intereses de los diferentes usuarios, como herramienta de representación y de formación, así como para prever futuras modificaciones del sistema. En el aspecto de la representación, lo más interesante es utilizar algún tipo de especificación formal en sentido matemático, lo que facilita la consistencia y los análisis lógicos de los esquemas propuestos.
- Del esquema conceptual formalizado pueden derivarse diferentes subesquemas conceptuales, que representan aquellas partes del esquema conceptual de interés para un usuario o grupo de usuarios finales.
- El esquema de la base de datos → Una vez construido el esquema conceptual, el diseño de bases de datos obliga a realizar varias tareas previas a la construcción del esquema lógico global del sistema, también llamado esquema de bases de datos. Por el momento, basta saber que el esquema de la base de datos representa la descripción de los datos de la base de datos, mientras que el esquema conceptual representaba a la realidad.
- La primera de las tareas necesarias es la identificación de los datos requeridos, para obtener como resultado las partes del área de aplicación que deben representarse mediante datos, y en que forma deben presentarse éstos a los usuarios. El siguiente paso es el análisis de datos, consistente en la definición y clasificación de esos datos, su descripción, que suele presentarse en forma de diccionario de datos, como una "metabase de datos".
- Por último, debe hacerse la especificación de los paquetes de entrada y de salida, correspondientes con los datos que deben introducir y obtener como respuesta los usuarios, según sus necesidades. Las tres tareas habrán permitido obtener tres documentos sobre descripción del área de aplicación, definición y clasificación de los datos y especificación de las características de los diversos paquetes, respectivamente. Tomando como punto de partida estos tres elementos, se construye la especificación de esquema de la base de datos, que responderá al contenido total de la base de datos y las características de las vías de acceso requeridas a través de estos datos. Frente al análisis de datos, que es la definición y clasificación de los datos, el esquema se encarga de la utilización de esos datos.
- El diccionario de recursos de información→ La gestión efectiva de los datos involucrados en la base de datos implica necesariamente disponer de alguna herramienta que controle las características y funciones de aquellos. Esta función es

cubierta mediante el diccionario de recursos de información (DRI),que asegura la integración de toda la información contenida en el sistema. Se habla entonces de metadatos, como datos que definen y describen los datos existentes en el sistema. En un primer momento, este tipo de cuestiones eran resueltas a través de los diccionarios de datos, que reunían información sobre los datos almacenados, sus descripciones, significados, restricciones, usos, etc., y los directorios de datos, subsistemas del sistema de gestión, encargados de describir dónde y cómo se almacenaban los datos, Actualmente se aplica el concepto de diccionario de recursos de información, que engloban todo lo señalado anteriormente, dando lugar a lo que ha pasado a llamarse "metabases".

- El enfoque de tratamiento de los datos → La construcción de los modelos conceptual y lógico de las bases de datos requiere la adopción de un determinado enfoque para la descripción y el tratamiento de los datos. Sin embargo, es necesario insistir en que la modelización de datos se orienta al conocimiento en profundidad de los datos que va a manejar la organización, para lograr una implantación óptima. La unión del modelo de datos con el sistema de gestión de base de datos dará como resultado la base de datos real. El modelo de datos será una representación gráfica orientada a la obtención de las estructuras de datos de forma metódica y sencilla, agrupando esos datos en entidades identificables e individualizables, y será reflejo del sistema de información en estudio.

Propuesta de un método estándar de diseño

Con los métodos que se han expuesto, el diseño de una base de datos relacional puede seguir dos caminos. Por una parte, puede crearse tomando como punto de partida la observación del universo en estudio, dando lugar a un conjunto de esquemas de relaciones, que contengan los atributos y sus restricciones. Por otra parte, puede dividirse el diseño en dos fases, la primera de las cuales sería definir el modelo conceptual y su esquema, y la segunda transformar el esquema conceptual en un esquema relacional mediante una transformación realizada de acuerdo a unas reglas dadas.

Sin perjuicio del rigor en el diseño relacional, el diseño de una base de datos no puede limitare a la aplicación exclusiva de la teoría de la normalización. Del mismo modo que se ha visto la existencia de variadas metodologías en el ámbito de los sistemas de información, se encuentra el mismo panorama en el diseño de bases de datos, aunque aquí tampoco aparece una metodología consagrada. De esta forma, Elmasri y Navathe comparan el ciclo de diseño de los sistemas de información y de las bases de datos, y definen el problema de diseñar una base de datos como:

"Desing the logical and physical structure of one or more databases to accommodate the information needs of the users in an organization for a defined set of applications" y señalan la existencia de seis fases en el proceso de diseño de una base de datos:

- Fase 1: Recopilación y análisis de requerimientos → En esta fase se trata de conocer las expectativas del usuario sobre la base de datos. Para ello, se identifican los grupos de usuarios reales y posibles y las áreas de aplicación, se revisa la documentación existente, se analiza el entorno operativo y los requerimientos de procesado, y se realizan entrevistas y cuestionarios con los usuarios. Para todo ello existen técnicas formalizadas de especificación de requerimientos.

- Fase 2: Diseño conceptual de la base de datos → Esta fase se subdivide en otras dos.
  - La Fase 2a corresponde al Diseño del esquema conceptual, esquema de especificación del modelo de datos a alto nivel, independiente de cualquier SGBD, que no puede utilizarse para implementar directamente la estructura de la base de datos. Para obtenerlo puede adoptarse un enfoque de esquema centralizado (en el cual se unen previamente los diferentes requerimientos a la realización del esquema), o un enfoque de integración de vistas (en el cual se unen los esquemas de cada requerimiento en uno global realizado a posteriori).
  - La Fase 2b corresponde al diseño de transacciones, es decir, a aquellas aplicaciones que van a manipular datos contenidos en la base de datos. Se suelen identificar mediante el estudio de las entradas y salidas de datos y su comportamiento funcional. De esta forma se identifican transacciones de recuperación, de actualización y mixtas.
- Fase 3: Elección de un SGBD → Se consideran diferentes factores técnicos, económicos y de beneficio, de servicio técnico y formación de usuarios, organizativos de rendimiento, etc. Sin embargo, resulta difícil la medida y cuantificación ponderada de los diferentes factores.
- Fase 4: Transformación del modelo de datos (o fase de diseño lógico→. En esta fase se crea un esquema conceptual y los esquemas externos necesarios en el modelo de datos del SGBD seleccionado, mediante la transformación de los esquemas de modelo de datos a alto nivel obtenidos en la Fase 2a, al modelo de datos ofrecido por el SGBD.
- Fase 5: Diseño de la base de datos física → Consiste en definir las estructuras de almacenamiento y de acceso para alcanzar una rendimiento óptimo de las aplicaciones de la base de datos. Los criterios adoptados suelen ser el tiempo de respuesta, la utilización de espacio y el volumen de transacciones por minuto.
- Fase 6: Implementación del sistema de base de datos → En esta fase final se hace realidad la base de datos, mediante la creación y la compilación del esquema de bases de datos y de los ficheros de bases de datos, así como de las transacciones, a través de las aplicaciones.

La metodología expuesta, que puede servir como marco de referencia general, puede modificarse según las características del contexto en el que se diseña e implanta el sistema de bases de datos.

En el dinámico entorno de la información almacenada en las bases de datos, las recientes tendencias, derivadas en muchas ocasiones de las propias necesidades, han obligado a completar e incorporar nuevos conceptos y enfoques en el tratamiento de los datos. Por ejemplo, la existencia de relaciones complejas en el mundo real han obligado a la incorporación del modelado semántico, lo que ha dado como resultado la evolución del modelo entidad-relación extendido, con sus conceptos de superclases y subclases, y los procesos de generalización y especialización, así como la importante noción de herencia.

También es necesaria la referencia ineludible al paradigma de la orientación a objetos (BERTINO y MARTINO, 1995), enfoque de tratamiento de la información que cobra cada vez mayor auge en aplicaciones comerciales, y que se configura como la opción de mayor futuro en el desarrollo de aplicaciones. La identificación de los datos y sus procesos como objetos individuales, el encapsulamiento y las propiedades de herencia son las características

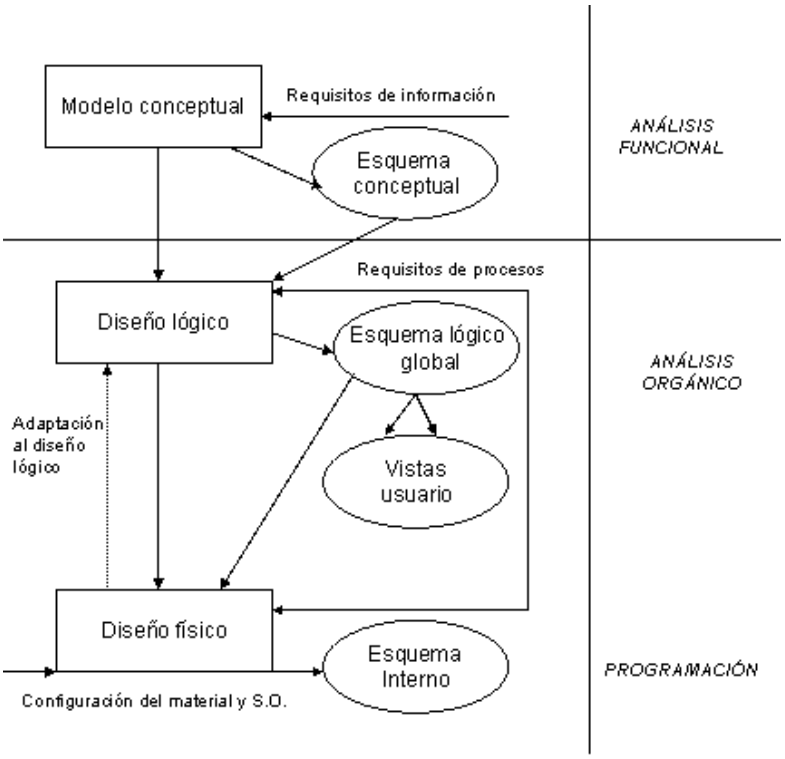
principales del enfoque a objetos. Por último, no puede olvidarse la creciente tendencia entre el enfoque relacional y el modelo de objetos, así como la integración de información referenciada espacialmente en modelos relacionales.

Es innegable que la gestión y la explotación subsiguiente de los registros que contienen datos, y, como consecuencia, información, depende de las herramientas existentes en el campo de la gestión de la información, por una parte, y del cuerpo teórico de la ciencia de la información, por otra. La explotación satisfactoria de esta información, de la misma forma, demanda experiencia en dos áreas de conocimiento: en las técnicas de recuperación de información y en el estudio de las necesidades de los usuarios.

Por lo tanto, una metodología estructura el diseño en una secuencia de pasos, de modo que los resultados de una etapa son las entradas de la siguiente. Cada fase es un proceso iterativo en el que se van produciendo refinamientos sucesivos antes de pasar a la siguiente. Al mismo tiempo existe una realimentación entre las dos últimas fases ya que pueden producirse modificaciones del diseño lógico derivados de requisitos del diseño físico. No es conveniente, sin embargo, que exista realimentación hacia el conceptual, ya que este debe representar las características de recursos de información de la empresa. Las fases de diseño de una base de datos se asemejan a las clásicas de un Sistema de Información:

- Análisis Funcional → integrado en el diseño conceptual, en el que partiendo de los requisitos de la información se define el esquema conceptual.
- Análisis Orgánico → integra los diseños lógicos y físicos y la documentación respectiva. Partiendo del anterior y teniendo en cuenta el modelo de datos del propio SGBD y los requisitos de los procesos, se obtiene:
  - El esquema lógico global, visión general de la base de datos en su propio modelo correspondiente, en el que se incluirá la descripción de los datos y sus interrelaciones, las restricciones de integridad y confidencialidad.
  - Vistas de usuario, es decir, estructuras externas. Derivadas del esquema lógico global que resultan importantes en la utilización del sistema.

La siguiente fase, partiendo del esquema lógico global y teniendo en cuenta los requisitos de los procesos, especificaciones del modelo de datos propio del SGBD, así como las características de los equipos físicos y del sistema operativo, se constituye el esquema interno o vista del sistema. Posteriormente se pasa a la instrumentación de la BD, programación de procesos, carga y explotación.



Los sistemas de gestión de bases de datos

Los SGBD pueden definirse como un paquete generalizado de software, que se ejecuta en un sistema anfitrión, centralizando los accesos a los datos y actuando de interfaz entre los datos físicos y el usuario. Las principales funciones que debe cumplir un SGBD se relacionan con la creación y mantenimiento de la base de datos, el control de accesos, la manipulación de datos de acuerdo con las necesidades del usuario, el cumplimiento de las normas de tratamiento de datos, evitar redundancias e inconsistencias y mantener la integridad.

El SGBD incorpora como herramienta fundamental el lenguaje SQL, para la definición y la manipulación de los datos.

El lenguaje de definición de datos (DDL, Data Definition Language) provee de los medios necesarios para definir los datos con precisión, especificando las distintas estructuras. Acorde con el modelo de arquitectura de tres niveles, habrá un lenguaje de definición de la estructura lógica global, otro para la definición de la estructura interna, y un tercero para la definición de las estructuras externas.

El lenguaje de manipulación de datos (DML, Data Manipulation/ Management Language), que es el encargado de facilitar a los usuarios el acceso y manipulación de los datos. Pueden diferenciarse en procedimentales (aquellos que requieren qué datos se necesitan y cómo obtenerlos) y no procedimentales (que datos se necesitan, sin especificar cómo obtenerlos), y se encargan de la recuperación de los datos almacenados, de la inserción y supresión de datos en la base de datos, y de la modificación de los existentes.

Objetivos de los SGBDR

Todos los gestores considerados como relacionales deberían cumplir los aspectos mencionados a continuación:

- Independencia lógica.
- Independencia física.
- Utilización de un lenguaje estándar de acceso a la información.
- Control de Replicación.
- Control de concurrencia.
- Control de seguridad.
- Eficiencia en el procesamiento de CPU, memoria y E/S.
- Mantenimiento de la integridad de la información.

Visión metodológica de Oracle

La visión de Oracle de los aspectos anteriormente tratados se fundamentan en el ciclo de vida de un sistema de negocios.

- Estrategia: define las áreas de negocio y jerárquicas de la empresa, planificación y control del proyecto, diagrama Entidad/Relación, matriz de Entidades/Funciones, diagrama de flujo de datos, dependencias funcionales, etc.
- Análisis: define con detalle el diagrama Entidad/Relación, la matriz Entidades/Funciones, determina el flujo/almacén de datos, definición de requerimientos, define la biografía de las entidades, los diagramas de transacción y detalla las funciones lógicas.
- Diseño: establece una primera aproximación de la Base de Datos; su tamaño, etc. Y define los módulos de programa, de red, implementa las entidades, define la arquitectura de la red y la máquina manual.
- Construcción/Documentación de usuario: se implementan los programas, se crean los ficheros de la base de datos, se confeccionan los manuales de usuario y la documentación del sistema, realizándose también un test de material.
- Instalación: reingeniería inversa de datos y procesos, conversión de datos, comprobación de la robustez de los datos, test de datos y gestión de la configuración.
- Producción: confeccionamiento de la matriz de análisis de impactos, mantenimiento actualizado tanto del propio sistema como de su documentación.

Es importante emplear el tiempo necesario en cada fase, incluidas las iniciales, y no aventurarse en la etapa siguiente sin tener completamente establecida y estudiada la etapa anterior. Esto significa que, si prestamos poco tiempo en la estrategia tendremos un gasto muy grande en producción. No utilizar aproximaciones puramente teóricas para no limitar la participación del cliente o usuarios final, adaptándolo el rigor a medida que se va progresando en el diseño y teniendo en cuenta los distintos tipos de usuarios involucrados.

Definición de base de datos

Una base de datos en Oracle es un conjunto de información tratado como una unidad y gestionado por la instancia.

Bajo una perspectiva física puede decirse que una base de datos Oracle es un conjunto de ficheros de tres tipos: de datos, de control y de log.

Ficheros de control

El fichero de control de la base de datos es un pequeño fichero binario necesario para el arranque y operación de la base de datos. Este fichero se está actualizando constantemente durante el uso de la base de datos por lo tanto debe estar disponible para su escritura. Si el fichero de control no está accesible, la base de datos no puede funcionar.

El contenido del fichero de control es información sobre la base de datos asociada y es requerida por la instancia en tiempo de arranque y parada. La información del fichero de control solo puede ser modificada por Oracle; no es un fichero que pueda ser editado.

Entre otros aspectos, la información del fichero de control cubre los siguientes:

- Nombre de la base de datos
- Fecha de creación de la base de datos
- Nombre y localización del resto de los ficheros de la base de datos (ficheros de datos y de redo log)
- Información sobre tablespaces
- Rangos de ficheros de datos fuera de línea
- Historia de log
- Información sobre archivado
- Información sobre juegos de backup (backup set) y piezas de backup (backup piece).
- Información sobre backup de ficheros de datos y de redo log.
- Información sobre copia de ficheros de datos.
- Número actual de secuencia de log.
- Información sobre checkpoint.

El nombre de la base de datos y su fecha de creación se originan en tiempo de creación de la base de datos. El nombre se toma del parámetro de inicialización DB_NAME usado cuando se ejecuta la sentencia CREATE DATABASE.

Cada vez que un fichero de datos o de redo log es añadido a la base de datos, renombrado o eliminado, el fichero de control es actualizado para reflejar el cambio en la estructura física de la base de datos. De esta forma, Oracle es capaz de identificar los ficheros necesarios en tiempo de arranque y conocer aquellos ficheros necesarios para realizar una recuperación cuando ésta es precisa.

El fichero de control registra también información sobre checkpoints. Cada tres segundos, el proceso checkpoint (CKPT) registra información en el fichero de control sobre la posición del checkpoint en el redo log. Esta información se utiliza durante el proceso de recovery para especificar a Oracle que las entradas de redo log previas a este punto no son necesarias para

efectuar el recovery de la base de datos ya que están sincronizadas en los ficheros de datos de la base de datos.

Oracle permite multiplexar el fichero de control de forma que múltiples copias de este fichero puedan ser abiertas y gestionadas en paralelo por la misma base de datos. Al tener varias copias del fichero de control multiplexadas sobre diferentes dispositivos de almacenamiento se obtiene una tolerancia a fallos mayor.

Si todos los ficheros de control de la base de datos se pierden, la instancia abortará y será necesaria una recuperación desde un medio de almacenamiento. Esta no es la forma más sencilla de recuperar el fichero de control si el backup del fichero de control que se va a utilizar es antiguo. Por lo tanto, la recomendación de Oracle es:

- Usar ficheros de control multiplexados en cada base de datos
- Almacenar cada copia en un disco físico diferente
- Utilizar mirror de sistema operativo

Ficheros de REDO LOG

La estructura más importante para las operaciones de recovery es el redo log. Este consiste en dos o más ficheros que almacenan todos los cambios realizados en la base de datos a medida que éstos ocurren. Cada instancia de una base de datos Oracle tiene asociado redo log para proteger la base de datos en caso de fallo de la instancia.

Cuando se habla dentro del contexto de múltiples instancias de una base de datos, el redo log relativo a cada instancia se denomina “redo thread”. En configuraciones típicas solo hay una instancia por base de datos, de tal forma que solo hay un redo thread. En entornos Oracle Real Application Clusters (RAC) hay dos o más instancias accediendo de forma concurrente a la misma base de datos y cada instancia tiene su propio redo thread. El hecho de tener un redo thread por instancia evita la contención eliminando un potencial cuello de botella.

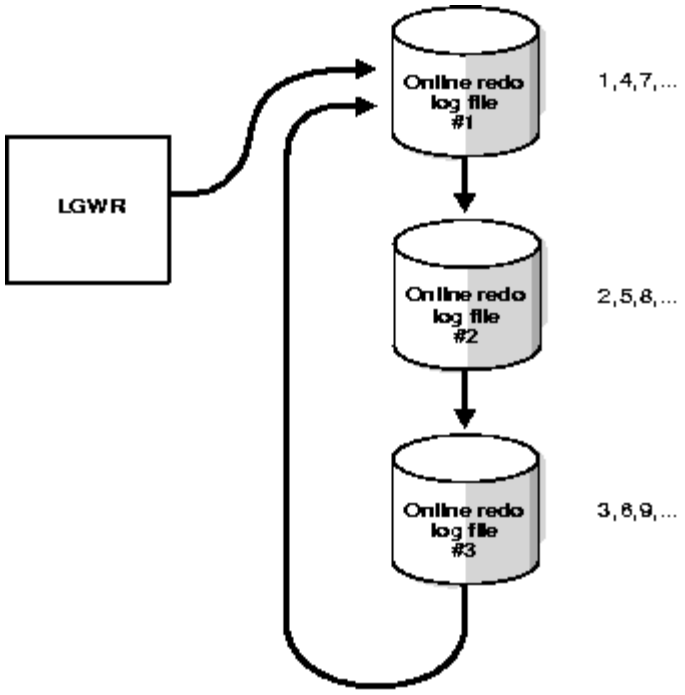
Los ficheros de redo log son llenados a través de registros denominados “redo records”. Un registro de redo, también denominado “redo entry”, está constituido por un grupo de vectores de cambio, cada uno de los cuales describe un cambio realizado en un bloque de la base de datos. Por ejemplo, al modificar el valor del salario en la tabla de empleados, se genera un registro redo que contiene los vectores de cambio que describen los cambios realizados en el bloque del segmento de datos de la tabla, el bloque del segmento de undo y la tabla de transacción del segmento de undo.

Las entradas redo que se utilizan para reconstruir todos los cambios en la base de datos incluyen los segmentos de undo. Por lo tanto, el redo log protege los datos de los que se ha realizado “rollback”. Al hacer una operación de recovery, la base de datos lee los vectores de cambio en los registros redo y aplica los cambios en los bloques correspondientes.

El redo log de una base de datos consiste en dos o más ficheros de redo log. La base de datos necesita un mínimo de dos ficheros para garantizar que uno de ellos siempre está disponible para escrituras mientras que el otro está siendo archivado, si la base de datos está en modo ARCHIVELOG.

El proceso LGWR (Log Writer) escribe en los ficheros de redo log de forma circular. Cuando el fichero actual de redo log se llena, el proceso LGWR comienza a escribir en el siguiente. A

su vez, cuando este se llena, el proceso LGWR retorna al primer fichero y comienza el ciclo de nuevo.



Oracle solo utiliza un redo log a la vez para escribir registros del buffer de redo log. El fichero sobre el que se está escribiendo se denomina fichero “current”.

Los ficheros de redo log necesarios para realizar un recovery de instancia se denominan “active” mientras que aquellos no necesarios para el recovery se denominan “inactive”.

Si la base de datos está en modo ARCHIVELOG, nunca puede reutilizarse un fichero de log marcado como “active” mientras que el proceso archiver (ARCn) no haya copiado su contenido.

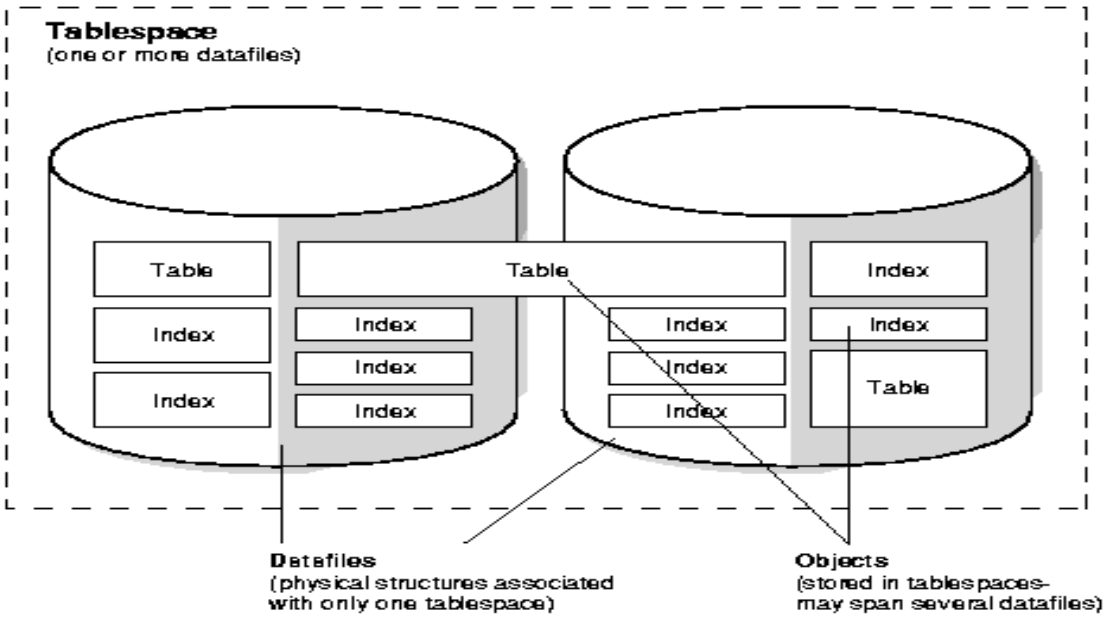
Un cambio de log “log switch” es el punto en el que la base de datos deja de escribir a un fichero de redo log y comienza a escribir en el siguiente. Habitualmente se produce cuando el redo log actual “current” se llena y la escritura ha de continuar en el siguiente. Sin embargo, los cambios en los ficheros de log pueden ser configurados para que sucedan a intervalos regulares aunque el fichero no esté completamente lleno. También se pueden forzar los cambios de fichero de log de forma manual.

Oracle asigna a cada fichero de redo log un nuevo número de secuencia de log cada vez que sucede un cambio de log y el proceso LGWR comienza a escribir sobre él. Cuando se archivan los redo log, éstos mantienen su número de secuencia para poder ser localizados en caso necesario.

Ficheros de datos

Contienen toda la información relativa a los datos de la base de datos. En definitiva son la representación física de los tablespaces de Oracle. Son gestionados a través del proceso background DBWn y su estructura, tanto física como lógica, será tratada en el capítulo “Estructuras de almacenamiento de la base de datos”.

Oracle almacena sus datos de forma lógica e unas estructuras denominadas tablespaces y físicamente en ficheros asociados con su correspondiente tablespace.



Una base de datos, un tablespace y un fichero de datos son términos relativamente cercanos aunque tienen importantes diferencias:

- Una base de datos consiste en una o más estructuras lógicas denominadas tablespaces en las que se almacenan todos los datos de la base de datos.
- Cada tablespace consiste en uno o más ficheros denominados ficheros de datos los cuales son estructuras físicas asociadas al sistema operativo en el que está funcionando la base de datos.
- Los datos de la base de datos son almacenados en los ficheros de datos de los tablespaces de la base de datos.

En el nivel de granularidad menor, Oracle almacena los datos en bloques de datos, también denominados bloques lógicos, bloques Oracle o páginas. Un bloque de datos se corresponde con un número de bytes de espacio en disco.

El siguiente nivel lógico de almacenamiento de la base de datos es la extensión. Una extensión es un número específico de bloques contiguos reservados para un determinado tipo de información.

El mayor nivel lógico de estructuras lógicas de la base de datos es el segmento. Un segmento es un conjunto de extensiones, cada una de las cuales ha sido reservada para una estructura de datos específica y son almacenadas en el mismo tablespace. Por ejemplo, cada tabla es almacenada en su propio segmento de datos mientras que cada índice asociado a la misma tabla es almacenado en un segmento de índice.

Oracle reserva espacio para los segmentos en unidades denominadas extensiones. Cuando las extensiones actuales de un segmento se llenan, Oracle reserva una nueva extensión para el segmento. Debido a que las extensiones se reservan a medida que se van necesitando, pueden no estar contiguas en disco.

Un segmento y todas sus extensiones son almacenados en un tablespace. Dentro del tablespace, un segmento puede tener extensiones en más de un fichero de datos. Sin embargo, una extensión no puede estar contenida en más de un fichero.

Tablespaces

Definición y usos

Un tablespace es un almacén lógico de datos, siendo los ficheros de datos o ficheros de sistema operativo el almacén físico de los mismos. Otra perspectiva para comprender el concepto es pensar en un tablespace como la representación lógica de uno o varios ficheros de sistema operativo.

La existencia de los tablespaces viene dada por un concepto denominado abstracción en niveles. En la abstracción en niveles tiene que existir una separación clara entre el nivel físico y el nivel lógico de las estructuras de la base de datos. De esta forma, cualquier modificación en el nivel físico nunca afectará al nivel lógico. En base a este concepto, los tablespaces son la representación lógica de los ficheros de sistema operativo. De esta manera si un fichero cambia de nombre, de localización o se amplía, el tablespace y sus niveles inferiores no tendrán porque ser conscientes de este cambio.

El uso de múltiples tablespaces proporciona flexibilidad en la gestión de la base de datos. Si una base de datos tiene múltiples tablespaces, es posible:

- Separar los datos de usuario de los del diccionario de datos y así reducir la contención en entrada / salida.
- Separar los datos de las aplicaciones para prevenir que al poner un tablespace offline esto afecte a múltiples aplicaciones.
- Balancear la entrada / salida distribuyendo los ficheros de los tablespaces sobre distintos discos.
- Reservar un tablespace para un determinado uso como gran actividad de actualización, actividad de solo lectura o almacenamiento de segmentos temporales.
- Realizar backup de tablespaces individuales
- Controlar el uso de disco en base a la asignación de cuotas sobre tablespaces.

Tipología de tablespaces

Antes de crear un tablespace ha de generarse la base de datos para contenerlos. El primer tablespace de cualquier base de datos es el tablespace SYSTEM el cual contiene información básica para el funcionamiento de la base de datos tal como el diccionario de datos o el segmento de rollback system. Este tablespace es gestionado como cualquier otro tablespace de la base de datos pero requiere un alto nivel de privilegios y tiene ciertas restricciones tales como que no puede ser renombrado, borrado ni puesto fuera de línea.

El tablespace SYSAUX actúa como un tablespace auxiliar para el tablespace SYSTEM también se genera en tiempo de creación de la base de datos. Contiene información sobre productos y características Oracle de forma que esos productos no requieren un tablespace específico para ellos. La gestión de este tablespace, al igual que la del tablespace SYSTEM requiere un alto nivel de seguridad de forma que no puede ser borrado o renombrado.

Los pasos a seguir en la creación de tablespaces varían en base al sistema operativo con el que se esté trabajando pero el primer paso consiste en crear una estructura de directorios para almacenar los ficheros de datos.

En base a los tipos de información almacenada dentro de un tablespace estos pueden clasificarse en:

- Tablespace SYSTEM
- Tablespaces de datos e índices
- Tablespaces temporales
- Tablespaces de UNDO
- Tablespaces BIGFILE
- Tablespace SYSAUX

Los usuarios

En consonancia con las posibles, y diferentes, vistas externas, se pueden identificar varios tipos de usuarios. En primer lugar, los usuarios finales, que hacen un uso limitado de las capacidades del sistema, normalmente referentes a introducción, manipulación y consulta de los datos. Los usuarios finales pueden ser sofisticados o especializados e ingenuos, dependiendo de su nivel de interacción con el sistema. En segundo lugar hay que citar a los programadores de base de datos, encargados de escribir aplicaciones limitadas, mediante el lenguaje de programación facilitado por el SGBD, normalmente algún lenguaje de cuarta generación, que faciliten la ejecución de tareas por parte de los usuarios finales. Por último, el administrador de base de datos (DBA, Data Base Administrator) cumple las importantes funciones de crear y almacenar las estructuras de la base de datos, definir las estrategias de respaldo y recuperación, vincularse con los usuarios y responder a sus cambios de requerimientos, y definir los controles de autorización y los procedimientos de validación.

2

Introducción al Lenguaje SQL

Tabla de contenidos

Introducción al lenguaje SQL 1

Estándares SQL. SQL: 99 1

Funcionamiento del lenguaje SQL..... 1

Componentes del lenguaje SQL 2

DDL (Data Definition Languaje - Lenguaje de Definición de Datos) 2

DCL (Data Control Languaje - Lenguaje de Control de Datos)..... 2

DML (Data Manipulation Languaje - Lenguaje de Manipulación de Datos)..... 3

Objetos de base de datos 3

Objetos de un esquema..... 3

Objetos no pertenecientes a esquemas 4

Reglas para nombrar objetos de base de datos 4

Referenciar objetos de la base de datos 4

Resolución de referencias a objetos..... 5

Tipos de datos en Oracle 6

Introducción al lenguaje SQL

SQL (Structured Query Language) es un lenguaje utilizado para el acceso y la manipulación de los datos almacenados en una base de datos.

El acceso a los datos se realiza mediante aplicaciones o herramientas que utilizan el lenguaje SQL para mostrar los datos al usuario final.

La historia de SQL (que se pronuncia deletreando en inglés las letras que lo componen, es decir "ese-cu-ele" y no "siquel" como se oye a menudo) empieza en 1974 con la definición en los laboratorios de investigación de 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. Las experimentaciones 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 curso de los años ochenta, numerosas compañías (por ejemplo Oracle y Sybase, sólo por citar algunos) comercializaron productos basados en SQL, que se convierte en el estándar industrial de hecho por lo que respecta a las bases de datos relacionales.

Estándares SQL. SQL: 99

El lenguaje SQL está sujeto a estándares tanto ANSI como ISO.

Oracle cumple con el estándar publicado en Julio de 1999 y llamado SQL: 99.

- ANSI X3.135-1999, "Database Language SQL"
- ISO/IEC 9075:1999, "Database Language SQL

Funcionamiento del lenguaje SQL

El lenguaje SQL es un lenguaje de acceso y manipulación de datos, procesa la información fila a fila y no permite sentencias de control para dirigir el flujo de ejecución.

Existe una extensión al lenguaje SQL, conocida como PL/SQL en el que se incluyen sentencias de control para poder realizar operaciones con datos obtenidos en sentencias anteriores.

El SQL permite:

- Almacenar información en tablas.
- Seleccionar la información que sea necesaria de la base de datos.
- Realizar cambios en la información y estructura de los datos.
- Combinar y calcular datos para conseguir la información necesaria.
- Garantizar la integridad y la consistencia de los datos.

Componentes del lenguaje SQL

Una base de datos relacional almacena datos en una o más tablas, y estas tablas pueden ser relacionadas de varias maneras para poder acceder a la información de manera eficiente. Cumplen las siguientes condiciones:

- Contienen varias tablas.
- Cada tabla es a su vez un conjunto de registros, filas o tuplas.
- Cada registro consta de varias columnas, campos o atributos.
- El nombre de los campos de una tabla es distinto.
- Los valores almacenados en una columna deben ser del mismo tipo de dato.
- Cada registro de la tabla es único.
- Todas las filas de una misma tabla poseen el mismo número de columnas.
- El orden de los registros y de los campos no está determinados.
- Para cada campo existe un conjunto de valores posible.
- La información puede ser recuperada o almacenada por medio de sentencias llamadas consultas.

DDL (Data Definition Languaje - Lenguaje de Definición de Datos)

Las sentencias DDL son aquellas utilizadas para la creación, modificación y eliminación de una base de datos y todos sus componentes: tablas, índices, relaciones, disparadores (triggers), procedimientos almacenados, etc.

- CREATE → Creación de objetos de base de datos.
- ALTER → Modificar los objetos de base de datos ya existentes.
- DROP → Eliminar la definición y los datos los objetos de base de datos ya existentes.
- REPLACE → Reemplazar un objeto de base de datos ya existente.
- RENAME → Renombrar un objeto de base de datos ya existente.
- TRUNCATE → Borrar las filas de una tabla o índice sin eliminar su estructura. El objeto permanece pero vacío.

Las sentencias DDL incorporan COMMIT implícito, no necesitan validación para tener efecto persistente en la base de datos.

DCL (Data Control Languaje - Lenguaje de Control de Datos)

Las sentencias DCL son aquellas que permiten al usuario otorgar y eliminar privilegios sobre objetos y sistema de base de datos.

- GRANT → Conceder privilegios a usuarios o roles de base de datos.
- REVOKE → Suprimir privilegios a usuarios o roles de base de datos

Las sentencias DCL incorporan COMMIT implícito, no necesitan validación para tener efecto persistente en la base de datos.

DML (Data Manipulation Languaje - Lenguaje de Manipulación de Datos)

Las sentencias DML son aquellas utilizadas para insertar, borrar, modificar y consultar los datos de una base de datos.

- SELECT → Consultar datos en la base de datos.
- UPDATE → Actualizar datos en las tablas.
- INSERT → Insertar datos en las tablas.
- DELETE → Borrar datos de las tablas.
- MERGE → Inserta y/o actualiza y/o elimina datos en las tablas.

Todas las sentencias de manipulación de datos necesitan validación para tener efecto duradero. COMMIT o ROLLBACK validan o deshacen los cambios realizados en las tablas.

Objetos de base de datos

Los objetos existentes en una base de datos Oracle se dividen en dos grandes grupos dependiendo si están asociados a un esquema (propietario) específico y los objetos que no están asociados a ningún esquema, es decir, aquellos que son globales a la base de datos.

Objetos de un esquema

Un esquema es una colección de estructuras lógicas de datos. Los esquemas son propiedad de usuarios de base de datos.

El nombre del esquema de base de datos coincide con el nombre de los usuarios.

Todo esquema de base de datos debe de pertenecer a un usuario, pero pueden existir usuarios de base de datos que no tengan su propio esquema.

Los objetos son los siguientes:

Clusters	Restricciones
Enlaces de base de datos	Disparadores
Tablas	Vistas
Sinónimos	Procedimientos almacenados
Funciones almacenadas	
Paquetes	Secuencias
Dimensiones	Librerías de procedimientos
Tablas organizadas como índices	Índices
Clases Java	Recurso Java
Fuente Java	Vistas materializadas
Logs de vistas materializadas	Objetos tabla
Objetos	Objetos vista

Objetos no pertenecientes a esquemas

En Oracle existen otro tipo de objetos que no pertenecen a un esquema específico de base de datos.

Son propiedad de la base de datos y se crean, modifican y eliminan mediante sentencias SQL.

Los objetos son los siguientes:

Contextos	Directorios
Fichero de parámetros SPFILE	Perfiles
Roles	Segmentos UNDO o Rollback
Tablespaces	Usuarios

Reglas para nombrar objetos de base de datos

Se puede especificar el nombre de un objeto de base de datos Oracle de las siguientes formas:

- El nombre dado a un objeto puede ser entrecomillado (‘’), si cuando se crea un objeto se ponen comillas, es necesario acceder al objeto utilizando las (‘’)
- El nombre dado a un objeto no es entrecomillado (‘’), el acceso a dicho objeto se realiza sin comillas. Y el nombre con el que se almacena en las tablas del diccionario es siempre con mayúsculas.

Independientemente de que nombre dado a un objeto se especifique con comillas o sin ella las reglas para nombradlos son las siguientes:

- La longitud del nombre del objeto debe de estar comprendida entre 1 carácter y 30. Las excepciones son:
  - El nombre de la base de datos como máximo puede tener 8 caracteres.
  - Los enlaces de base de datos pueden tener hasta 128 caracteres.
- Si el nombre dado al objeto es entrecomillado puede ser una palabra reservada, en el caso contrario no puede ser una palabra reservada. No es recomendable utilizar palabras reservadas para nombrar objetos de base de datos.
- El nombre del objeto debe de comenzar por un carácter alfabético en el caso de los no entrecomillados.
- Los nombres no entrecomillados pueden empezar por cualquier carácter.
- Los nombres de los objetos deben de ser únicos en el esquema al que pertenecen. Por ejemplo: dos tablas pueden llamarse igual si pertenecen a distintos esquemas.
- El nombre de las columnas de una tabla debe de ser único en la tabla, pero tablas distintas pueden tener los mismos nombres de columnas.

Referenciar objetos de la base de datos

Para acceder a un objeto existente de la base de datos Oracle la sintaxis utilizada es la siguiente:

```
[esquema.]nombre_objeto[.parte objeto][@enlace base de datos]
```

Dónde:

- Esquema → Nombre del esquema que contiene el objeto. Propietario del objeto.
- Nombre_objeto → nombre del objeto
- Parte objeto → Parte del objeto. En el caso de un tabla o vista una columna. No todos los objetos de base de datos tienen partes.
- Enlace de base de datos → nombre del enlace de base de datos en el que reside el objeto que se quiere referenciar.
-

Ejemplo:

```
CONNECT usuario1/contraseña
SELECT * FROM employees;
```

En este caso el usuario conectado a la base de datos es usuario1, esta accediendo a un objeto de su propio esquema, en este caso a la tabla employees.

Si este usuario quisiera acceder a un objeto del usuario usuario2, la sintaxis es la siguiente:

```
CONNECT usuario1/contraseña
SELECT * FROM usuario2.employees;
```

Si el acceso se realiza a más de un objeto, es necesario anteponer el nombre del objeto en las columnas en el caso en el que tuvieran el mismo nombre.

```
SELECT employees.department_id, departments.department_id
FROM employees, departments;
```

Además se pueden utilizar alias tanto para el nombre de las tablas referenciadas en la consulta, como para el nombre de las columnas.

```
SELECT e. department_id departamento, d. department_id n_departamento
FROM employees e, departments d;
```

El alias dado a una columna, aparece como encabezado en el resultado de la consulta.

Resolución de referencias a objetos

Cuando se accede a un objeto de base de datos, Oracle sigue las siguientes reglas para acceder al objeto:

- Una tabla perteneciente al usuario conectado.
- Una vista perteneciente al usuario conectado.
- Un sinónimo privado perteneciente al usuario conectado.
- Un sinónimo público. Estos objetos pertenecen al esquema PUBLIC de la base de datos.

Tipos de datos en Oracle

Las columnas de las tablas de la base de datos tienen una serie de características que se asocia a un tipo de datos.

Los tipos de datos predefinidos en Oracle12c son los siguientes:

Tipo de dato	Descripción
Varchar2(bytes)	Cadena de caracteres de longitud variable. El tamaño está comprendido entre 1 byte y 4.000 bytes. Se debe especificar el tamaño.
Nvarchar2(bytes)	Cadena de caracteres de longitud variable dependiente del juego de caracteres nacional indicado en la creación de la base de datos. El tamaño está comprendido entre 1 byte y 4.000 bytes. Se debe especificar el tamaño.
Number(precisión, decimales)	Número especificando la precisión y los decimales. El rango de la precisión es de 1 a 38. El rango para decimales es de –84 a 127
Long	Cadena de caracteres de longitud variable. Tamaño máximo 2 gigabytes.
Date	Fecha. El rango está comprendido entre el 1 de enero de 4.712 antes de Cristo hasta el 31 de diciembre de 9.999 después de Cristo. El formato depende de las variables NLS
Raw(bytes)	Dato binario de longitud variable. El tamaño máximo es de 2 kilobytes.
Long Raw	Cadena binaria de longitud variable. El tamaño máximo es de 2 gigabytes.
Rowid	Cadena en base hexadecimal que representa la dirección única de una fila dentro de la tabla. La pseudocolumna rowid devuelve la información en este formato.
Char(bytes)	Cadena de caracteres de longitud fija. El tamaño está comprendido entre 1 byte y 2.000 bytes. Se debe especificar el tamaño.

Tipo de dato	Descripción
NChar(bytes)	Cadena de caracteres de longitud fija dependiente del juego de caracteres nacional indicado en la creación de la base de datos. El tamaño está comprendido entre 1 byte y 2.000 bytes. Se debe especificar el tamaño.
Clob	Tipo de dato objeto grande, que contiene caracteres de byte simple. Tamaño máximo 4 gigabytes.
Nclob	Tipo de dato objeto grande, que contiene caracteres UNICODE. Tamaño máximo 4 gigabytes.
Blob	Objeto grande de tipo binario. Tamaño máximo 4 gigabytes.
Bfile	Contiene un localizador a un fichero externo de la base de datos de tipo binario. Tamaño máximo 4 gigabytes.
TIMESTAMP(precisión)	Valores de fecha: día, mes y año y de tiempo: horas minutos y segundos. La precisión especificada es de 0 a 9 y es el número de dígitos usados para representar los segundos.
TIMESTAMP(precisión) WITH TIME ZONE	Valores de fecha: día, mes y año y de tiempo: horas minutos y segundos. Utiliza el desplazamiento para la zona horaria. La precisión especificada es de 0 a 9 y es el número de dígitos usados para representar los segundos.
TIMESTAMP(precisión) WITH LOCAL TIME ZONE	Valores de TIMESTAMP WITH TIME ZONE con las excepciones: Los datos son normalizados al TIME ZONE de la base de datos en el momento de almacenarlos Al seleccionar este tipo de datos se formatean a la zona horaria de la sesión que los visualiza
INTERVAL YEAR (precision) TO MONTH	Almacena un período de tiempo en años y meses. Precisión → el número de dígitos para el año

Tipo de dato	Descripción
INTERVAL DAY (day_precision) TO SECOND (fractional_seconds_precision)	Almacena un periodo de tiempo en días, minutos y segundos. day_precision → máximo numero de dígitos para almacenar el día fractional_seconds_precision → precisión en los segundos
BINARY_FLOAT	Este tipo de datos existe a partir de Oracle 10g, almacenan datos numéricos de coma flotante en formato IEEE 754 de 32 bits de precisión.
BINARY_DOUBLE	Este tipo de datos existe a partir de Oracle 10g, almacenan datos numéricos de coma flotante en formato IEEE 754 de 64 bits de precisión.

Los tipos de datos RAW y LONG ROW se mantienen por compatibilidad, pero se aconseja utilizar los tipos LOB de 4 gigabytes.

3

Lenguaje de Manipulación de Datos. DML

Tabla de contenidos

Introducción a DML **1**

Sentencia SELECT **1**

Selección de todos los datos de una tabla 1

Selección de ciertas columnas de una tabla 2

Cláusula FROM 2

Cláusula WHERE 3

Cláusula ORDER BY 3

Cláusula GROUP BY 4

Operadores SQL. **5**

Operadores aritméticos 5

Operadores concatenación 5

Operadores de conjuntos 6

Operador UNION 6

Operador UNION ALL 6

Operador INTERSECT 6

Operador MINUS 6

Condiciones SQL. **7**

Condiciones de comparación 8

Condiciones lógicas 9

Condiciones de pertenencia. 10

Condiciones de rango de valores. 11

Condición NULL. 11

Condición EXISTS. 12

Condición LIKE 12

Funciones. **12**

Funciones Numéricas 13

Funciones de cadena de caracteres que devuelven caracteres 13

Funciones de cadena de caracteres que devuelven números 13

Funciones de fecha y hora 14

Funciones de conversión 14

Máscaras de formato. Formatos numéricos 14

Máscaras de formato. Formatos tipo fecha. 15

Otras funciones. 17
Variables de entorno. 17
Funciones de grupo. La cláusula GROUP BY 18
Funciones de grupo. La cláusula HAVING 21
Combinaciones de Tablas en versiones anteriores a la 9i 22
EQUI JOIN 22
OUTER JOIN 23
Uniones de SQL: 1999 26
Uniones cruzadas 26
Uniones naturales 26
Predicados de Unión y la Cláusula ON 27
OUTER JOIN. Uniones externas. 28
LEFT OUTER JOIN: 28
RIGHT OUTER JOIN: 29
FULL OUTER JOIN: 29
La expresión CASE 29
CASE simple 30
CASE buscada 30
NULLIF 30
COALESCE 30
SUBCONSULTAS 31
Usando subconsultas de varias Columnas 33
No Pairwise 33
Pairwise 34
subconsultas correlacionadas 34
Otros Ejemplos 34
Updates correlacionados 36
Delete correlacionados 37
subconsultas escalares 37
Ejemplos 37
Operadores Exists,Not Exists,WITH 38
Operador exists 38
Operador not exists 38
Operador with 38
Recuperación jerárquica 39
DML. Lenguaje de Manipulación de Datos 42
INSERT 42

UPDATE 43
DELETE 44
MERGE 45
Extensiones a la sentencia MERGE 46
Sentencias Transaccionales 47
COMMIT 48
SAVEPOINT 48
ROLLBACK 49

Introducción a DML

EL DML (Data Manipulation Lenguage) es un sublenguaje de consulta y manipulación de datos.

Se entiende por manipulación de datos la:

- Recuperación de Información. Sentencia SELECT.
- Inserción de nueva Información. Sentencia INSERT.
- Eliminación (Borrado) de información existente. Sentencia DELETE.
- Modificación de Información Almacenada. Sentencia UPDATE.
- Inserción y/o modificación y/o eliminación de la información nueva o existente. MERGE.

Sentencia SELECT.

La recuperación de información (consulta), se realiza mediante la sentencia SELECT. Básicamente, SELECT se encarga de recoger datos de una o varias tablas. Ahora bien, podemos estar interesados en todos los datos de unas columnas concretas de las tablas, en datos que cumplan una cierta condición... para ello, debemos conocer muy bien su sintaxis que, básicamente, es la siguiente:

```
SELECT [ALL|DISTINCT] {Lista_Columnas_o_Expresiones | * }
FROM   Lista_Tablas
[WHERE Condición_de_búsqueda]
[GROUP BY Lista_Columnas]
[HAVING Condición]
[ORDER BY {expresión|posición}[ASC|DESC];
```

- Lista_Columnas_o_Expresiones → Enumerar cada una de las columnas a seleccionar en la recuperación de datos. En el caso de que la consulta acceda a más de una tabla es necesario anteponer el nombre de la tabla al nombre de la columna, en el caso de nombre duplicados de columnas.

Selección de todos los datos de una tabla

Para seleccionar todos los datos de una tabla se utiliza el signo *.

En este caso el orden de aparición de las columnas coincide con el orden seguido a la hora de crear la tabla. No es necesario conocer la estructura de las columnas de una tabla para seleccionar todas las filas y columnas de la misma.

```
SQL> desc jobs
 Nombre                                    ¿Nulo?   Tipo
 ----------------------------------------- -------- ----------------------
 JOB_ID                                    NOT NULL VARCHAR2(10)
 JOB_TITLE                                 NOT NULL VARCHAR2(35)
 MIN_SALARY                                         NUMBER(6)
 MAX_SALARY                                         NUMBER(6)
SQL> SELECT * FROM jobs;

JOB_ID     JOB_TITLE                           MIN_SALARY MAX_SALARY
---------- ----------------------------------- ---------- ----------
AD_PRES    President                                20000      40000
```

```
AD_VP      Administration Vice President           15000      30000
```

También se pueden especificar todas las columnas en la lista de columnas de la sentencia SELECT, cada una de ellas se separa de la anterior y posterior mediante comas.

En este caso el orden de aparición de las columnas en el resultado es el orden especificado en la sentencia SELECT.

Se pueden utilizar 'alias' para dar el nombre de la cabecera de la columna en el resultado de la sentencia SELECT. Los 'alias' de las columnas se especifican a continuación del nombre de la columna y antes de la coma que lo separa de la siguiente columna.

```
SQL> SELECT job_title Trabajo, job_id Abrev, min_salary salario_minimo,
  2  max_salary salario_maximo
  3  FROM jobs;

TRABAJO                            ABREV      SALARIO_MINIMO SALARIO_MAXIMO
---------------------------------- ---------- -------------- --------------
President                          AD_PRES             20000          40000
Administration Vice President      AD_VP               15000          30000
```

Selección de ciertas columnas de una tabla

Es posible seleccionar algunas columnas de una tabla y no seleccionar la información de otras columnas.

Solamente se muestra la información de aquellas columnas especificadas en la sentencia SELECT. El resto de la información de las columnas de la tabla no se muestra al usuario.

```
SQL> SELECT job_id, job_title
  2  FROM jobs;

JOB_ID     JOB_TITLE
---------- ----------------------------------
AD_PRES    President
AD_VP      Administration Vice President
(...)
```

Cláusula FROM

La información que muestra la sentencia SELECT se extrae de una o varias tablas o vistas.

Los nombres de la tabla(s) a las que pertenecen las columnas de la sentencia SELECT deben de aparecer en las tablas especificadas en la cláusula FROM.

El usuario conectado a la base de datos puede consultar datos pertenecientes a tablas de su propio esquema o a tablas de otros esquemas. En este caso es necesario anteponer el propietario de la tabla en la cláusula FROM.

```
SQL> SELECT * FROM usu1.regions;

 REGION_ID REGION_NAME
---------- -------------------------
         1 Europe
         2 Americas
(…)
```

En el caso en el que los datos obtenidos provengan de varias tablas es conveniente anteponer el nombre de la tabla antes del nombre de la columna.

```
SQL> SELECT regions.region_id, regions.region_name, countries.country_name
  2  FROM regions, countries
  3  WHERE regions.region_id = countries.region_id;

 REGION_ID REGION_NAME               COUNTRY_NAME
---------- ------------------------ -------------------------------------
         1 Europe                   United Kingdom
         1 Europe                   Netherlands
```

Se pueden utilizar alias para los nombres de las tablas, de tal manera que se especifica el nombre del alias en vez del nombre de la tabla en la lista de columnas de la sentencia SELECT.

```
SQL> SELECT R.region_id, R.region_name, C.country_name
  2  FROM regions R , countries C
  3  WHERE regions.region_id = countries.region_id;

 REGION_ID REGION_NAME               COUNTRY_NAME
---------- ------------------------ -------------------------------------
         1 Europe                   United Kingdom
         1 Europe                   Netherlands
```

Cláusula WHERE

Es en la cláusula WHERE donde se especifican las condiciones que debe cumplir lo que se está buscando.

Los predicados utilizados en la condición de búsqueda son los encargados de filtrar los datos antes de mostrarlos al usuario. Estos predicados pueden ser simples o complejos y se pueden utilizar operadores y funciones en ellos.

Ejemplo → seleccionar la información de la tabla countries cuyo código sea 'AR'

```
SQL> SELECT * FROM countries
  2  WHERE country_id = 'AR';

CO COUNTRY_NAME                              REGION_ID
-- ---------------------------------------- ----------
AR Argentina                                         2
```

Cláusula ORDER BY

El resultado de la consulta, por defecto, puede aparecer en cualquier orden.

La cláusula ORDER BY se utiliza para determinar el orden en el que se muestra el resultado.

Permite ordenar por más de una columna y se admite la ordenación por posición.

La ordenación por defecto es ascendente, pero se puede cambiar utilizando DESC.

El conjunto de caracteres de ordenación está basado en el parámetro NLS_SORT que si no se especifica toma los valores de NLS_LANGUAGE.

Ejemplo→ Seleccionar todos los empleados ordenando el resultado por nombre y salario.

```
SQL> SELECT * FROM employees
  2  ORDER BY last_name,salary;
```

Ejemplo→ Seleccionar todos los empleados ordenando el resultado por nombres ascendentes y salarios descendentes.

```
SQL> SELECT * FROM employees
  2  ORDER BY last_name ASC, salary DESC;
```

Ejemplo→ Seleccionar todos los empleados ordenando el resultado por la primera y la cuarta columna en orden descendente.

```
SELECT * FROM employees
ORDER BY 1,4 DESC;
```

Se pueden utilizar expresiones en los criterios de ordenación.

Ejemplo→ Seleccionar todos los países ordenados por las dos primeras posiciones del nombre.

```
SELECT country_name FROM countries
ORDER BY substr(country_name,1,2);
```

Las restricciones de la cláusula ORDER BY son:

- No se puede utilizar el operador DISTINCT.
- El número máximo de expresiones a ordenar son 255.
- No está permitido ordenar un tipo LOB, varray o tabla anidada.
- En el caso que la sentencia SELECT incluya una cláusula GROUP BY, las expresiones del ORDER BY están mucho más restringidas.

Las funciones, expresiones y operadores se explican en los siguientes puntos del capítulo.

Cláusula GROUP BY

Los resultados obtenidos de la sentencia SELECT se pueden agrupar y devolver una sola fila resumen de la información solicitada.

Las expresiones en la cláusula GROUP BY pueden contener cualquier columna perteneciente a las tablas o vistas especificadas en la cláusula FROM.

Ejemplo → Obtener el salario mínimo y el máximo para el tipo de trabajo 'ST_MAN' agrupado por departamento.

```
SQL> SELECT department_id, MIN(salary), MAX (salary)
  2  FROM employees
  3  WHERE job_id = 'ST_MAN'
  4  GROUP BY department_id;

DEPARTMENT_ID MIN(SALARY) MAX(SALARY)
------------- ----------- -----------
           50        5800        8200.
```

En la cláusula GROUP BY se utilizan funciones de grupo para obtener informes con totales y subtotales para las columnas.

Operadores SQL.

Los operadores SQL se utilizan para manipular los datos de manera individual y devolver el valor resultante de la operación.

Los operadores o argumentos suelen estar representados por palabras claves o caracteres especiales (EXISTS, >=).

Genéricamente los operadores se pueden dividir en:

- Unarios → Solamente opera con un operando.
- Binarios → Utilizan dos o más operandos para realizar la operación.

Cuando en una misma expresión aparecen más de un operador, Oracle evalúa los operadores siguiendo un orden de preferencia de mayor a menor, y en el caso de los operadores con la misma preferencia los evalúa de izquierda a derecha.

Operadores aritméticos

Los operadores aritméticos se utilizan para sumar, restar, multiplicar, dividir o negar valores numéricos.

El resultado de la operación siempre es un número.

UNARIOS		
Operador	**Propósito**	Ejemplo
+, -	Cambia a positivo o negativo el operando sobre el que actúa.	SELECT * FROM employees WHERE -salary < 0;
BINARIOS		
Operador	**Propósito**	Ejemplo
+, -,*,/	Suman, restan, multiplican, dividen los operandos.	UPDATE employees SET salary = salary * 1.4;

Operadores concatenación

El operador || concatena cadenas de caracteres y tipos de datos CLOB.

El resultado es una cadena de caracteres.

El tipo de datos resultante depende del tipo de datos de los operandos, pudiendo combinar tipos de datos char, varchar2 y clob en la concatenación.

Ejemplo →

```
SQL> SELECT 'País: ' || country_name
  2  FROM countries;

'PAÍS:'||COUNTRY_NAME
-----------------------------------------
```

```
País: Argentina
País: Australia
País: Belgium
País: Brazil
```

Operadores de conjuntos

Los operadores de conjuntos combinan los resultados de dos consultas con datos comunes en un único resultado.

Las restricciones de estos operadores son las siguientes:

- Los operadores de conjuntos no pueden operar con columnas de tipo BLOB, CLOB, BFILE, VARRAY, o tablas anidadas.
- Los operadores UNION, INTERSECT y MINUS no son válidos con columnas LONG.
- Si la lista de columnas que preceden al operador de columnas contiene alguna expresión, hay que calificarla con un alias.
- No está permitido la cláusula FOR UPDATE.
- No se puede utilizar la cláusula ORDER BY en la subconsulta de los operadores de conjuntos.

Operador UNION

Devuelve todas las filas distintas seleccionadas por ambas consultas.

Operador UNION ALL

Devuelve todas las filas seleccionadas por ambas consultas incluyendo duplicados.

Operador INTERSECT

Devuelve todas las filas comunes a ambas consultas eliminando los duplicados.

Operador MINUS

Devuelve todas las filas seleccionadas por la primera consulta que no estén recuperadas en la primera consulta.

Ejemplos:

```
SQL> SELECT * FROM regions;
 REGION_ID REGION_NAME
---------- -------------------------
         1 Europe
         2 Americas
         3 Asia
         4 Middle East and Africa
```

```
SQL> SELECT * FROM regiones;
 REGION_ID REGION_NAME
---------- -------------------------
         1 Europe
         2 Americas
         3 Asia
         4 Middle East and Africa
         5 Australia
```

```
         6 Antartida
```

```
SQL>  SELECT * FROM regions UNION SELECT * FROM regiones;
 REGION_ID REGION_NAME
---------- -------------------------
         1 Europe
         2 Americas
         3 Asia
         4 Middle East and Africa
         5 Australia
         6 Antartida
```

```
SQL>   SELECT * FROM regions UNION ALL SELECT * FROM regiones;
 REGION_ID REGION_NAME
---------- -------------------------
         1 Europe
         2 Americas
         3 Asia
         4 Middle East and Africa
         1 Europe
         2 Americas
         3 Asia
         4 Middle East and Africa
         5 Australia
         6 Antartida
```

```
SQL>  SELECT * FROM regions INTERSECT SELECT * FROM regiones;
 REGION_ID REGION_NAME
---------- -------------------------
         1 Europe
         2 Americas
         3 Asia
         4 Middle East and Africa
```

```
SQL> SELECT * FROM regiones MINUS SELECT * FROM regionS;
 REGION_ID REGION_NAME
---------- -------------------------
         5 Australia
         6 Antartida
```

Condiciones SQL.

Las condiciones son combinaciones de operadores lógicos y expresiones que devuelven un resultado. El resultado puede ser VERDADERO, FALSO o DESCONOCIDO.

Condiciones de comparación

Condiciones de comparación		
Condición	**Propósito**	Ejemplo
=	Comprueba la igualdad	SELECT * FROM employees WHERE salary = 2500;
!= <> ^=	No igualdad, distinto	SELECT * FROM employees WHERE salary != 2500;
>	Mayor que	SELECT * FROM employees WHERE salary > 2500;
<	Menor que	SELECT * FROM employees WHERE salary < 2500;
>=	Mayor o igual que	SELECT * FROM employees WHERE salary >= 2500;
<=	Menor o igual que	SELECT * FROM employees WHERE salary <= 2500;

ANY **SOME**	Compara los valores de la lista devuelta. Debe de ir precedido de =,!=,<,>,>=,<=. El resultado es verdadero si la comparación es verdadera para alguno de los valores devueltos por la SELECT subordinada, y falso si la SELECT devuelve una tabla vacía, o si la comparación devuelve falso en todos los valores de la SELECT subordinada.	SELECT * FROM employees WHERE salary = ANY (SELECT salary FROM employees WHERE department_id = 60);
ALL	Compara los valores de la lista devuelta. Debe de ir precedido de =,!=,<,>,>=,<=. El resultado es verdadero si la comparación es verdadera para todos los valores devueltos por la SELECT subordinada, falso si para algún valor no nulo se obtiene falso en la comparación (también puede serlo si algún valor es nulo), y desconocido si para todo valor no nulo devuelve cierto la comparación.	SELECT * FROM employees WHERE salary >= ALL (2500, 3000);

Condiciones lógicas

Las condiciones lógicas combinan el resultado de dos componentes de una condición para obtener un solo resultado.

Son utilizadas para agrupar condiciones

Condiciones de comparación		
Condición	**Propósito**	***Ejemplo***

NOT	Devolver verdadero en el caso de que la condición sea falsa. Verdadero en el caso contrario.	SELECT * FROM employees WHERE NOT (job_id IS NULL);
AND	Devolver verdadero si los dos componentes de la condición son verdaderos. Falso en caso contrario	SELECT * FROM employees WHERE job_id = 'PU_CLERK' AND department_id = 50;
OR	Devolver verdadero si uno de los componentes de la condición es verdadero. Falso en caso de que ambos componentes sean falsos.	SELECT * FROM employees WHERE job_id = 'PU_CLERK' OR department_id = 50;

Condiciones de pertenencia.

Las condiciones de pertenencia evalúan si los operandos de la parte izquierda de la comparación pertenecen o no a la lista de valores o subconsulta situada en la parte derecha.

Condiciones de pertenencia		
Condición	**Propósito**	Ejemplo
IN	Averiguar si el resultado de la evaluación de una expresión está incluido en la lista de valores especificado tras la cláusula IN. Si expresión no es nulo y pertenece a la lista de valores (o el SELECT), obtendremos verdadero, si no pertenece falso, y si es nulo, desconocido. Equivalente al operador de comparación ANY	SELECT * FROM employees WHERE salary IN(2500, 3000, 2000; SELECT * FROM employees WHERE salary IN (SELECT salary FROM employees WHERE department_id = 60);
NOT IN	Averiguar si el resultado de la evaluación de una expresión no está incluido en la lista de valores especificado tras la cláusula NOT IN. Si expresión no es nulo y no pertenece a la lista de valores (o el SELECT), obtendremos verdadero, si	SELECT * FROM employees WHERE salary NOT IN (2500, 3000, 2000); SELECT * FROM employees WHERE salary NOT IN (SELECT

	pertenece falso, y si es nulo, desconocido. Equivalente al operador de comparación !=ALL.	salary FROM employees WHERE department_id = 60);

Condiciones de rango de valores.

Las condiciones de rango de valores evalúan si los operandos de la parte izquierda de la comparación pertenecen o no al rango de valores especificado en la parte derecha de la condición.

Condiciones de rango de valores		
Condición	**Propósito**	Ejemplo
BETWEEN x AND y	Averiguar si el resultado de la condición es mayor o igual que x y menor o igual que y.	SELECT * FROM employees WHERE salary BETWEEN 2000 AND 3000;
NOT BETWEEN x AND y	Averiguar si el resultado de la condición no es mayor o igual que x y menor o igual que y.	SELECT * FROM employees WHERE salary NOT BETWEEN 2000 AND 3000;

Condición NULL.

La condición NULL evalúa si el operando de la parte izquierda es un valor nulo.

Condición NULL		
Condición	**Propósito**	Ejemplo
IS NULL	Evaluar si el operando es el valor NULL. El resultado será verdadero en caso de que la columna tenga el valor NULL (NO es lo mismo que 0 o cadena vacía) y falso si no es así.	SELECT * FROM employees WHERE salary IS NULL;
IS NOT NULL	Evaluar si el operando no es el valor NULL. El resultado será verdadero en caso de que la columna no tenga el valor NULL (NO es lo mismo que 0 o cadena vacía) y falso si no es así.	SELECT * FROM employees WHERE salary IS NOT NULL;

Condición EXISTS.

La condición EXISTS evalúa si el operando existe en el las filas retornadas por la subconsulta de la parte derecha.

Condición EXITS		
Condición	**Propósito**	Ejemplo
EXISTS	Devuelve verdadero si el resultado del subselect es una tabla con una o más filas, y falso si el resultado del subselect es una tabla vacía.	SELECT salary FROM employees e WHERE EXISTS (SELECT * FROM departments d WHERE d.department_id = e.department_id);

Condición LIKE.

Con la condición LIKE se establece un patrón de búsqueda para el caso de las columnas con valores que sean de tipo carácter.

Hay dos caracteres que son especiales, '_' y '%'.

El primero tiene el significado de ser sustituido por un carácter cualquiera, y el segundo por una subcadena completa.

Condición LIKE		
Condición	**Propósito**	Ejemplo
X LIKE Y	Devuelve verdadero si el resultado de la comparación de las cadenas de caracteres coincide con el patrón de búsqueda especificado.	SELECT salary FROM employees WHERE last_name LIKE 'R%';
X NOT LIKE Y	Devuelve verdadero si el resultado de la comparación de las cadenas de caracteres no coincide con el patrón de búsqueda especificado.	SELECT salary FROM employees WHERE last_name NOT LIKE 'R_';

Funciones.

Una función SQL es similar a un operador ya que manipula los datos de entrada y devuelve un valor.

Las funciones SQL convierten el tipo de dato que reciben como argumento si ese es distinto al que tiene definido como argumento de entrada.

Las funciones SQL se pueden dividir en dos grandes grupos:

- Funciones de fila simple → Son las funciones que devuelven un resultado único para cada una de las filas sobre las que actúan.
- Funciones de grupo → Devuelven un resultado único para un conjunto de filas de entrada.

En el Anexo D se incluye un gran número de funciones SQL, clasificadas según su pertenencia a los grupos que a continuación se relacionan.

Funciones Numéricas

Son funciones de fila simple, aceptan como argumentos de entrada valores numéricos y devuelven un resultado numérico.

Ejemplo:

```
SQL> SELECT ABS(-25) FROM DUAL;
  ABS(-25)
----------
        25
```

Funciones de cadena de caracteres que devuelven caracteres

Este tipo de funciones reciben como argumento una cadena de caracteres y una vez aplicada la función devuelven una cadena de caracteres.

Son funciones de fila simple.

Las restricciones son las siguientes:

- Las funciones que devuelven una cadena de caracteres tipo CHAR están limitadas a 2.000 caracteres. En el caso de que el valor de retorno supere los 2.000 caracteres se trunca el resultado a 2.000 y no se genera error.
- Las funciones que devuelven una cadena de caracteres tipo VARCHAR2 están limitadas a 4.000 caracteres. En el caso de que el valor de retorno supere los 4.000 caracteres se trunca el resultado a 4.000 y no se genera error.
- Las funciones que devuelven una cadena de caracteres tipo CLOB están limitadas a 4 GB. En el caso de que el valor de retorno supere los 4 GB se genera un error y no se devuelven datos.

Ejemplo:

```
SQL>  SELECT initcap('el nombre') FROM DUAL;
INITCAP('
---------
El Nombre
```

Funciones de cadena de caracteres que devuelven números

Son funciones de fila simple que retornan un valor numérico, tendiendo como argumento de entrada una cadena de caracteres.

Ejemplo:

```
SQL> SELECT LENGTH('hola mundo') FROM DUAL;
```

```
LENGTH('HOLAMUNDO')
-------------------
                 10
```

Funciones de fecha y hora

Las funciones de fecha y hora realizan operaciones sobre tipos de datos DATE y devuelven un valor de tipo DATE o INTERVAL.

Son funciones de fila simple.

Ejemplo:

```
SQL> SELECT NEXT_DAY(sysdate,'DOMINGO') FROM DUAL;
NEXT_DAY(SYSDATE,'DO
--------------------
16-MAR-2003 11:20:41
```

Funciones de conversión

Este tipo de funciones convierten un tipo de datos en otro. El primer tipo de datos se convierte en el segundo.

Son funciones de fila simple.

Ejemplo:

```
SQL> SELECT TO_CHAR(SYSDATE, 'Month DD, YYYY') "Hoy" FROM DUAL;

Hoy
-------------------
Diciembre  11, 2008
```

Máscaras de formato. Formatos numéricos

- Deben proporcionarse las máscaras de conversión en formato numérico adecuadas cuando:
  - En una función TO_CHAR se cambie un tipo de dato NUMBER a un tipo de dato VARCHAR2
  - En una función TO_NUMBER se cambie un tipo de dato CHAR o VARCHAR2 a NUMBER.
- Los formatos numéricos están compuestos de uno o elementos de formatos:

Elemento	Ejemplo	Descripción
9	9999	El número de "9" especifica la cantidad de dígitos que se visualizan. Se visualizan blancos para los ceros a la izquierda.
0	0999	Visualiza los ceros a la izquierda en esa posición
	9990	Como ceros en vez de como blancos, o les da valor de 0 en lugar de blanco.
$	$9999	Antepone como prefijo el signo de dólar.

B	B9999	Devuelve los ceros como blancos, a menos que se indique "0" en la máscara de formato
Elemento	Ejemplo	Descripción
MI	9999MI	Muestra un "-" después de los valores negativos. Para los valores positivos muestra un espacio en blanco.
S	S9999	Muestra un "+" para los valores positivos y "-" para los negativos en la posición indicada.
PR	9999PR	Muestra los valores negativos entre los signos de menor y mayor <>.
D(*)	99D99	Muestra el carácter decimal en la posición indicada. Separa la parte entera y fraccionaria de un número.
G(*)	9G999	Muestra el separador de grupo en la posición indicada.
C(*)	C9999	Muestra el símbolo ISO de la moneda, en la posición indicada.
L(*)	L999	Muestra el símbolo de la moneda local en la posición indicada.
,(coma)	9,999	Muestra una coma en la posición indicada.
.(punto)	99.99	Muestra un punto en la posición indicada y separa la parte entera y la decimal de un número.
EEEE	9.999EEEE	Muestra el valor en notación científica.
RN	RN	Devuelve en mayúsculas o minúsculas (rn) el valor del número en romano. El número tiene que ser un entero entre 1 y 3999

- MI y PR solo pueden estar en la última posición de un formato numérico. Y el formato S sólo en la primera o la última posición.

(*) Los caracteres que devuelven los formatos indicados con un asterisco son especificados por los siguientes parámetros de inicialización de la Base de Datos.

D	Carácter Decimal	NLS_NUMERIC_CHARACTERS
G	Separador Grupo	NLS_NUMERIC_CHARACTERS
C	Símbolo moneda ISO	NLS_ISO_CURRENCY
L	Símbolo moneda	NLS_CURRENCY

Máscaras de formato. Formatos tipo fecha.

Los formatos de fecha se utilizan en las siguientes situaciones:

- En la función TO_CHAR para convertir un valor tipo DATE a un formato diferente al formato por defecto.
- En la función TO_DATE para convertir un valor tipo carácter en un formato diferente que el de por defecto.

La fecha por defecto se especifica de manera explícita con el parámetro NLS_DATE_FORMAT o implícitamente con el parámetro de inicialización NLS_TERRITORY. Se puede sobrescribir el valor de estos parámetros para la sesión actual con el comando ALTER SESSION.

Elemento	Significado
SCC ó CC	Indica el siglo. Si se indica S en las fechas de antes de Xto. Aparece el prefijo "-".
YYYY ó SYYYY	Visualiza los 4 dígitos del año. S antepone un guión "-" si la fecha es antes de Xto.
IYYY	Los 4 dígitos del año en formato ISO estándar.
YYY ó YY ó Y	Los últimos 3, 2 o 1 dígitos del año.
IYY ó IY ó I	Los últimos 3, 2 o 1 dígito del año en formato ISO estándar.
Y,YYY	El año con una coma en la posición indicada.
SYEAR ó YEAR	El número de año nombrado. El prefijo S antepone un guión si es antes de Xto.
RR	Los últimos dos dígitos del año. Se utiliza para años de otros siglos.
BC ó AD	Indicador antes/después de Xto.
B.C. ó A.D.	Indicador antes/después de Xto. Separado por puntos.
Q	Trimestre del año. P.e. de Enero a Marzo = 1.
MM	Número de Mes. P.e. Enero=1
RM	Número de Mes en romano. P.e. Julio=VII
MONTH	Nombre del mes en letra, alineado con blancos de hasta 9 caracteres de longitud.
MON	Nombre del día abreviado a tres letras.
WW	Número de la semana del año.
IW	Número de la semana del año en ISO estándar.
W	Semana del mes.
DDD	Número de día del año. (1-366).
DD	Número de día del mes. (1-31).
D	Número de día de la semana. (1-7)
DAY	Nombre del día de la semana hasta 9 caracteres.
DY	Nombre del día de la semana abreviado.

J	Fecha en Juliano. Los números especificados deben ser enteros.
AM ó PM	Indicador de Meridiano.
A.M. ó P.M.	Indicador de Meridiano separado por puntos.
HH ó HH12	Hora del día (1-12)
HH24	Hora del día (0-23)
MI	Minutos (0-59).
SS	Segundos (0-59)
SSSSS	Segundos transcurridos desde medianoche. (0-86399).

- Las especificaciones de algunos formatos, (Month, Mon, day, Dy, BC/AD, B.C./A.D., AM/PM) dependen del lenguaje utilizado en la B.D. indicados en los parámetros de inicialización:
  - NLS_DATE_LENGUAGE
  - NLS_TERRITORY
  - NLS_LANGUAGE
- Las máscaras de formato YEAR y SYEAR devuelven el nombre del año SIEMPRE en inglés.

Otras funciones.

Hay multitud de funciones no encuadradas en ninguno de los grupos de funciones relacionados anteriormente que aportan gran potencia y funcionalidad a las sentencias SQL. Algunas de ellas son imprescindibles como NVL o aportan gran potencia como DECODE o permiten el manejo de tipos de datos LOB como EMPTY_[B|C]LOB o BFILENAME.

También existen funciones de usuario desarrolladas en PL/SQL y almacenadas en el Servidor de B.D. Se utilizan cuando la funcionalidad que deseamos no está disponible en SQL y sus funciones. Para crearlas se utiliza CREATE FUNCTION o pueden residir dentro de un paquete creado con CREATE PACKAGE.

Variables de entorno.

UID Sintaxis UID

Propósito Devuelve un entero que es el identificador único de el usuario conectado.

Ejemplo

```
SELECT UID FROM DUAL
```

USER

Sintaxis USER

Propósito Devuelve un VARCHAR2 que contiene el nombre del usuario conectado en la base de datos local.

Ejemplo

```
SELECT USER, UID FROM DUAL
       USER          UID
       ----------    ------
       USU1                9
```

USERENV

Sintaxis USERENV (opción)

Propósito Devuelve información en un tipo de dato VARCHAR2 acerca de la sesión actual. El argumento opción puede tener los valores:

'ENTRYID'. Devuelve un identificador de la sentencia ejecutada para auditorías.

'LANGUAGE'. Devuelve el lenguaje y el territorio utilizado durante la sesión en formato:

lenguaje_territorio_juegocaracteres

'SESSIONID'. Devuelve un identificativo de la sesión del usuario para auditorías.

'TERMINAL'. Devuelve el identificativo del sistema operativo para el terminal utilizado.

Ejemplo

```
SELECT USERENV('LANGUAGE') "Idioma" FROM DUAL
              Idioma
       -------------------------
       AMERICAN_AMERICA.US7ASCII
```

Funciones de grupo. La cláusula GROUP BY

- Las funciones de grupo devuelven un resultado basado en un grupo de filas, más que sobre una fila solamente.
- Algunas funciones de grupo aceptan las siguientes opciones:
  - DISTINCT: Esta opción provoca que la función de grupo considere solamente los distintos valores de la expresión. También puede utilizarse sin funciones de grupo.
- Ejemplo:

```
SQL> SELECT distinct(job_id) FROM employees;
       JOB_ID
       ----------
       CLERK
       SALESMAN
       MANAGER
       ANALYST
       PRESIDENT
```

  - ALL: Esta opción provoca que la función de grupo considere todos los valores recuperados, incluyendo duplicados. Es la opción por defecto.

- Todas las funciones de grupos excepto COUNT(*) ignoran los valores nulos.
- A continuación se muestran algunas de las funciones de grupo más utilizadas:

AVG Sintaxis AVG([DISTINCT|ALL] n)

Propósito Devuelve la media de n.

Ejemplo

```
SELECT AVG(salary) "Media" FROM employees;
       Media
       ----------
       2073.21429
```

COUNT Sintaxis COUNT({* | DISTINCT|ALL] expr})

Propósito Devuelve el numero de filas en la consulta.

Ejemplo

```
SELECT COUNT(*) "Total" FROM employees;
       Total
       -----
        14
```

```
SELECT COUNT(job_id) "Contar" FROM employees;
       Contar
       ------
        14
```

```
SELECT COUNT(DISTINCT job_id)"Trabajos" FROM employees;
       Trabajos
       --------
        4
```

MAX Sintaxis MAX([DISTINCT|ALL] expr)

Propósito Devuelve el valor máximo de la expresión.

Ejemplo

```
SELECT MAX(salary) "Máximo" FROM EMPLOYEES;
       Máximo
       ----------
       5000
```

MIN Sintaxis MIN([DISTINCT|ALL] expr)

Propósito Devuelve el valor mínimo de la expresión.

Ejemplo

```
SELECT MIN(salary) "Mínimo" FROM EMPLOYEES;
       Mínimo
       ----------
        800
```

STDDEV Sintaxis STDDEV([DISTINCT|ALL] x)

Propósito Devuelve desviación estándar de x.

Ejemplo

```
SELECT STTDEV(salary) "Desviación" FROM employees;
       Desviación
       ----------
       1182.50322
```

SUM Sintaxis SUM([DISTINCT|ALL] n)

Propósito Devuelve la suma de los valores de n.

Ejemplo

```
SELECT SUM(salary) "Suma" FROM employees;
       Suma
       ----------
       29025
```

VARIANCE

Sintaxis VARIANCE([DISTINCT|ALL] x)

Propósito Devuelve la varianza de x. Un número.

Ejemplo

```
SELECT VARIANCE(salary) "Variación" FROM employees;
       Variación
       ----------
       1389313.87
```

- Se puede utilizar la cláusula GROUP BY para agrupar las filas seleccionadas y devolver una sola fila resumiendo la información solicitada. La selección de los conjuntos de filas se realizan para calcular propiedades de los conjuntos, basándose en los valores de la expresión(es) especificados en la cláusula GROUP BY.
- Si la sentencia SELECT contiene la cláusula GROUP BY, la lista de la SELECT solo puede contener los siguientes tipos de expresiones:
  - Constantes
  - Funciones de Grupo
  - Expresiones idénticas a las que estarán en la cláusula GROUP BY.
  - Expresiones implícitas en las expresiones mencionadas en la SELECT que evalúen el mismo valor para todas las filas de un grupo.

- Las expresiones en la cláusula GROUP BY pueden contener cualquier columna de una tabla o vista mencionada en el FROM, sin importar que estas columnas estén o no en la lista de la SELECT.
- En el siguiente ejemplo muestra el máximo y el mínimo de los salarios para cada departamento.
- En el ejemplo a continuación veremos como recuperar el mínimo y el máximo de los salarios de las personas cuyo trabajo sea 'CLERK' (empleado) dentro de cada departamento.
-

```
SQL> SELECT    department_id deptno, MAX(salary), MIN(salary)
  2  FROM      employees
  3  GROUP BY  department_id;

    DEPTNO MAX(SALARY) MIN(SALARY)
---------- ----------- -----------
       100       12000        6900
        30       11000        2500
                  7000        7000
        90       24000       17000
        20       13000        6000
```

```
SQL> SELECT    department_id deptno, MAX(salary), MIN(salary)
  2  FROM      employees
  3  WHERE     job_id LIKE '%CLERK'
  4  GROUP BY  department_id;

    DEPTNO MAX(SALARY) MIN(SALARY)
---------- ----------- -----------
        30        4464        3600
        50        6048        3024
```

Funciones de grupo. La cláusula HAVING

- Se puede utilizar la cláusula HAVING para restringir el número de grupos de filas definidas en la GROUP BY que se visualizan. El procesamiento de la sentencia es el siguiente:
  - Si la sentencia contiene una cláusula WHERE, primero se filtran todas las filas que no satisfagan la condición.
  - Se realizan los cálculos especificados y se forman los conjuntos de filas especificados en GROUP BY.
  - Solo se muestran los subconjuntos recuperados que satisfacen la cláusula HAVING

Por ejemplo, visualizar el máximo y el mínimo de los salario de employees cuyo trabajo sea el de empleado ('CLERK') para cada departamento y que además su salario mínimo esté por debajo de $3500.

```
SQL> SELECT     department_id, MAX(salary), MIN(salary)
  2  FROM       employees
  3  WHERE      job_id LIKE '%CLERK'
  4  GROUP BY   department_id
```

```
  5  HAVING MIN(salary)<3500;

DEPARTMENT_ID MAX(SALARY) MIN(SALARY)
------------- ----------- -----------
           50        6048        3024
```

Combinaciones de Tablas en versiones anteriores a la 9i

EQUI JOIN

- Un "join" es una sentencia SELECT que combina filas de dos o más tablas o vistas. Cada fila mostrada contiene datos de 2 o más tablas. La cláusula WHERE es la que determina como se realiza la combinación.
- Si la cláusula WHERE es inexistente o inadecuada puede realizarse un producto cartesiano, que combinará todas las filas de todas las tablas implicadas en la combinación.
- El producto cartesiano genera un número de filas muy elevado. Por ejemplo, si combino dos tablas con 100 filas cada una, sin establecer cláusula where, se generarán 10.000 filas. Por lo tanto, siempre debe incluirse un criterio de combinación.
- El tipo de combinación más utilizado es el llamado "simple join" o "equi join", que devuelve filas de dos o más tablas basándose en una condición de igualdad. El criterio de combinación por lo tanto lo establece el operador "="
- Suponiendo que unimos dos tablas, la sintaxis básica sería la siguiente:

SELECTColumnas de las tablas citadas en la cláusula FROM. Si existen columnas con el mismo nombre en ambas tablas, es conveniente anteponer nombre_tabla.nombre_columna o bien, alias_tabla.nombre_columna.

FROM Tablas de las que se extrae información. Por ejemplo: tabla1 alias1, tabla2 alias2

WHERE Criterio de combinación de las tablas.

Por ejemplo: tabla1.columna = tabla2.columna, o bien,

alias1.columna = alias2.columna

- Se pueden seleccionar TODAS las columnas de ambas tablas. NO es necesario que las columnas mencionadas en el criterio de combinación sean seleccionadas.
-

Ejemplo

En ejemplo siguiente muestra cómo recuperar el nombre y el número de departamento en el que trabaja cada empleado.

```
    SELECT e.first_name,e.department_id,d.department_name
```

```
FROM   employees e, departments d
WHERE  e.department_id = d.department_id;
```

FIRST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
Steven	90	Executive
Neena	90	Executive
Lex	90	Executive
Alexander	60	IT
Bruce	60	IT
David	60	IT
Valli	60	IT
Diana	60	IT
Nancy	100	Finance
Daniel	100	Finance

OUTER JOIN

El outer join o combinación externa amplifica el resultado de una combinación simple. Una combinación externa recupera TODAS las filas de la de la combinación externa y también aquellas que no encuentren su correspondiente fila pareja de una tabla a otra.

Ejemplo:

Seleccionar el número de departamento, el nombre de departamento y el salario mensual para cada departamento sin utilizar combinación externa:

```
SELECT    d.department_id, d.department_name, SUM(salary) "Salario Mensual"
FROM      employees e, departments d
WHERE     e.department_id = d.department_id
GROUP BY  d.department_id, d.department_name
ORDER BY  d.department_id;
```

DEPARTMENT_ID	DEPARTMENT_NAME	Salario Mensual
10	Administration	4400

20	Marketing	19000
30	Purchasing	24900
40	Human Resources	6500
50	Shipping	156400
60	IT	28800
70	Public Relations	10000
80	Sales	304500
90	Executive	58000
100	Finance	51600
110	Accounting	20300

11 filas seleccionadas.

- Para seleccionar el total de salarios mensuales para todos los departamentos independientemente de que tengan empleados o no, tendremos que utilizar un OUTER JOIN.

```
SELECT    d.department_id, d.department_name, SUM(salary) "Salario Mensual"
FROM      employees e, departments d
WHERE     e.department_id(+) = d.department_id
GROUP BY  d.department_id, department_name
ORDER BY  d.department_id;
```

DEPARTMENT_ID	DEPARTMENT_NAME	Salario Mensual
10	Administration	4400
20	Marketing	19000
30	Purchasing	24900
40	Human Resources	6500
50	Shipping	156400
60	IT	28800

70	Public Relations	10000
80	Sales	304500
90	Executive	58000
100	Finance	51600
110	Accounting	20300
120	Treasury	
130	Corporate Tax	
140	Control And Credit	
150	Shareholder Services	
160	Benefits	
170	Manufacturing	
180	Construction	
190	Contracting	
200	Operations	
210	IT Support	
220	NOC	
230	IT Helpdesk	
240	Government Sales	
250	Retail Sales	
260	Recruiting	
270	Payroll	

27 filas seleccionadas.

Sintaxis:

```
SELECT    col1, col2, .... coln
```

FROM tabla1, tabla2, ...

WHERE tabla1.col1 = tabla2.col1(+)

El símbolo (+) debe situarse en el predicado de combinación al lado de la columna o columnas pertenecientes a la tabla en la que hay ausencia de valor. En el ejemplo el departamento 40 (OPERATIONS) existe, pero no tiene empleados.

Uniones de SQL: 1999

A partir de Oracle 9i, se ha implementado la sintaxis completa de uniones de SQL: 1999. La unión se especifica en la cláusula FROM. La nueva sintaxis es más fácil de leer y menos propensa a errores ya que permite separar los predicados de unión de los otros predicados. También es más potente en las uniones externas, ya que soporta uniones externas completas.

Tipos de uniones:

- Uniones cruzadas.
- Uniones naturales.
- Uniones de igualdad y la cláusula USING.
- Uniones externas (completa, izquierda y derecha).

Uniones cruzadas

Son equivalentes al producto cartesiano de dos tablas. La sintaxis a utilizar es CROSS JOIN.

```
SELECT e.last_name, d.department_name
FROM   employees e CROSS JOIN Departments d;
```

El ejemplo anterior es equivalente a:

```
SELECT last_name, department_name
FROM employees, departments;
```

Uniones naturales

La unión natural es una unión de igualdad basada en todas las columnas que tienen el mismo nombre. Los tipos de datos que contienen deben ser compatibles. No es posible utilizar un prefijo de alias para una columna de unión.

La sintaxis a utilizar es NATURAL JOIN.

Si se recuperan todas las columnas mediante la sintaxis SELECT *, las columnas comunes aparecerán una sola vez en los resultados. Por tanto, no es necesario utilizar alias de tabla para cualificar las columnas en las consultas de unión natural, de hecho, si se emplea dará un error.

```
SELECT employee_id, last_name, department_id
FROM employees NATURAL JOIN departments;
```

Esta sentencia sería equivalente a la siguiente:

```
SELECT employee_id, last_name, d.department_id
FROM employees e, departments d
WHERE e.department_id = d.department_id AND
```

```
e.manager_id=d.manager_id;Uniones de igualdad y la cláusula USING.
```

En las uniones naturales se usan todas las columnas con nombres comunes para llevar a cabo la unión de las tablas. Si dos tablas tienen más de un nombre de columna en común, pero se desea especificar sólo una de ellas para llevar a cabo la unión, se empleará la cláusula USING.

- No se puede utilizar alias de tabla en las columnas a las que hace referencia la cláusula USING.
- Las cláusulas NATURAL y USING son mutuamente excluyentes.
- En las uniones naturales y en las uniones de igualdad no es posible utilizar columnas tipo LOB como columnas de unión.

```
SELECT employee_id, last_name, job_history.department_id
FROM employees
JOIN job_history
USING (employee_id);
```

La sentencia anterior es equivalente a la siguiente:

```
SELECT employees.employee_id, last_name, job_history.department_id
FROM employees, job_history
WHERE employees.employee_id=job_history.employee_id;
```

Predicados de Unión y la Cláusula ON

La cláusula ON permite separar los predicados de unión de los que no lo son; esta separación permite clarificar el código de las sentencias SQL y por tanto disminuir la posibilidad de cometer errores.

La cláusula ON permite el uso de cualquier predicado, incluyendo las subconsultas y los operadores lógicos.

```
SELECT employees.employee_id, last_name, job_history.department_id
FROM employees
JOIN job_history
ON (employees.employee_id=job_history.employee_id)
WHERE manager_id=100;
SELECT e.last_name employee, m.last_name manager
FROM employees e
JOIN employees m
ON (e.manager_id=m.employee_id);
```

- Mediante la cláusula ON es posible realizar uniones en Tres Sentidos, es decir una unión de tres tablas. En SQL: 1999 las uniones se llevan a cabo de izquierda a derecha. En el siguiente ejemplo la primera unión que se realiza es la de las tablas EMPLOYEES y DEPARTMENTS pero sin referenciar las columnas de la tabla LOCATIONS. La segunda condición de unión referencia las columnas de las tres tablas.

```
SELECT employee_id, city, department_name
FROM employees e
JOIN departments d
ON d.department_id = e.department_id
JOIN locations l
ON d.location_id = l.location_id;
```

La anterior sentencia sería equivalente a la siguiente:

```
SELECT employee_id, city, department_name
FROM employees, departments, locations
WHERE employees.department_id = departments.department_id
AND departments.location_id = locations.location_id;
```

OUTER JOIN. Uniones externas.

Antes de que existiera el estándar ISO/ANSI para uniones externas ORACLE resolvía estas consultas mediante la sintaxis (+).

La nueva sintaxis mejora la potencia y la legibilidad del código. El (+) sólo permite efectuar uniones externas en un único sentido, izquierdo o derecho.

Tipos de uniones externas:

- LEFT OUTER JOIN
- RIGHT OUTER JOIN
- FULL OUTER JOIN

LEFT OUTER JOIN:

Recupera las filas que tienen valor en la otra tabla y las filas de la tabla de la izquierda que no lo tienen en la tabla de la derecha.

```
SELECT employee_id, department_name
FROM departments d
LEFT OUTER JOIN employees e
ON (e.department_id= d.department_id);
```

La sentencia anterior es equivalente a:

```
SELECT employee_id, department_name
FROM departments d, employees e
WHERE e.department_id(+) = d.department_id;
```

Ampliación de la sintaxis de LEFT OUTER JOIN nativo

En versiones anteriores de Oracle, en una consulta que realizara combinaciones externas de más de dos pares de tablas, solo se podría utilizar la combinación de varias tablas contra una sola en el lado izquierdo (RIGHT OUTER JOIN).

```
SELECT COUNT(*) FROM employees e, departments d, locations l
WHERE d.department_id = e.department_id(+)
AND d.location_id = l.location_id(+);

  COUNT(*)
----------
       123
```

Antes de Oracle Database 12c, tener una tabla en el lado izquierdo que ampliara los valores con varias tablas en el lado derecho de una combinación externa, era ilegal y daba lugar al siguiente error.

```
SELECT COUNT(*) FROM employees e, departments d, locations l
WHERE d.department_id(+) = e.department_id
AND d.location_id(+) = l.location_id;

WHERE d.department_id(+) = e.department_id
                         *
```

```
ERROR en línea 2:
ORA-01417: una tabla sólo puede estar unida externamente como máximo a otra
tabla
```

En esta versión esta circunstancia se ha mejorado ya que una sola tabla puede obtener valores nulos generados por varias tablas (LEFT OUTER JOIN).

```
SELECT COUNT(*) FROM employees e, departments d, locations l
WHERE d.department_id(+) = e.department_id
AND d.location_id(+) = l.location_id;

  COUNT(*)
----------
      2461
```

RIGHT OUTER JOIN:

Recupera las filas que tienen valor en la otra tabla y las filas de la tabla de la derecha que no lo tienen en la tabla de la izquierda.

```
SELECT name, o.custid, orderdate
    FROM ord o
    RIGHT OUTER JOIN customer c
    ON (o.custid = c.custid);
```

La sentencia anterior es equivalente a:

```
SELECT name, o.custid, orderdate
    FROM ord o, customer c
    WHERE c.custid = o.custid(+);
```

FULL OUTER JOIN:

Recupera las filas que tienen valor en la otra tabla y las que no, tanto las de la tabla de la izquierda como las de la derecha.

Este es un nuevo tipo de join, que no existía en las versiones previas.

```
SELECT l.city, c.country_name
    FROM locations l
    FULL OUTER JOIN countries c
    ON (l.country_id = c.country_id);
```

La sentencia anterior sería equivalente a una sentencia con la función de conjunto UNION:

```
    SELECT l.city, c.country_name
    FROM locations l, countries c
    WHERE l.country_id = c.country_id(+)
    UNION
    SELECT l.city, c.country_name
    FROM locations l, countries c
    WHERE l.country_id(+) = c.country_id;
```

La expresión CASE

En el SQL: 1999 existen cuatro tipos de expresiones CASE:

- CASE simple.

- CASE buscada.
- NULLIF.
- COALESCE.

CASE simple

La expresión CASE simple es similar a la función DECODE. Es empleada para buscar y reemplazar valores resultantes según una expresión predefinida. Se puede especificar un valor devuelto por cada valor buscado. No se pueden emplear operadores de comparación.

```
SELECT employee_id, CASE department_id
                        WHEN 10 THEN 'Diez'
                    WHEN 20 THEN 'Veinte'
                    WHEN 30 THEN 'Treinta'
                    ELSE 'Otro'
                        END "Departamento"
FROM employees;
```

Esta sentencia es equivalente a:

```
    SELECT employee_id,
           DECODE(department_id,10,'Diez',20,'Veinte',30,'Treinta', 'Otro')
           "Departamento"
    FROM employees;
```

CASE buscada

La CASE buscada (searched CASE) es similar al IF..THEN..ELSE, permite buscar y reemplazar valores condicionalmente.

Cada cláusula WHEN puede ser diferente y los operadores lógicos pueden ser usados para combinar condiciones múltiples. Está permitido el uso de los operadores de comparación.

```
SELECT employee_id,
          CASE
          WHEN salary >=20000 THEN 'High'
          WHEN salary < 10000 THEN 'Low'
          ELSE 'Medium'
          END "Salary"
    FROM employees;
```

NULLIF

Devuelve NULL si el primer argumento es igual al segundo; en caso contrario, se devuelve el valor del primer argumento.

```
SELECT employee_id,
    NULLIF (commission_pct,.1) "new commission"
    FROM employees;
```

COALESCE

Devuelve el primer argumento (comenzando por la izquierda) que sea NOT NULL. Es similar a la función NVL, pero puede tener múltiples valores alternativos. Se acepta cualquier cantidad de argumentos.

Si la evaluación de todas las expresiones es NULL, la función COALESCE devuelve un valor NULL.

```
SELECT employee_id,
    COALESCE(to_char(commission_pct),'none') commission
    FROM employees;
```

SUBCONSULTAS

Las subconsultas son sentencias SELECT que se encuentran anidadas dentro de una CONSULTA PADRE. A las subconsultas a veces se las denomina CONSULTAS ANIDADAS. La sentencia que contiene una subconsulta es denominada CONSULTA PADRE. Las filas que devuelve la subconsulta son utilizadas por la consulta padre; de esta forma se consigue establecer condiciones sin conocer previamente los valores que las determinan.

La sintaxis básica en la sentencia SELECT es la siguiente:

```
    SELECT Lista_Columnas_o_Expresiones|*
    FROM Lista_Tablas
    WHERE expresión operador (SELECT Lista_Columnas_o_Expresiones|*
                              FROM Lista_Tablas
                              WHERE .....)
```

La expresión delante del operador en la consulta padre se debe corresponder en tipo y cardinalidad con la lista de columnas o expresiones de la subSELECT.

Es necesario que el operador sea el adecuado para tratar el número de filas devueltas por la subselect (p.e. elección del operador = ó IN: si devuelve más de una fila para = se produciría un error. Si es posible esperar más de un valor devuelto se emplearía el operador IN).

- Las subconsultas se utilizan para responder una pregunta multi-parte.
- Una subconsulta se ejecuta UNA sola vez, independientemente de las filas que devuelva la consulta principal o consulta padre.
- Una subconsulta puede así mismo contener otra subconsulta. No hay límite en el nivel de anidaciones de consultas.

El uso de las subconsultas va más allá de su uso en la sentencia SELECT. Se utiliza para:

- Definir el conjunto de filas que se desea insertar en la tabla destino en una sentencia INSERT o CREATE TABLE.
- Definir el conjunto de filas que se desea incluir la vista en una sentencia CREATE VIEW.
- Definir uno o más valores a asignar en las filas existentes en una sentencia UPDATE .... SET.
- Proporcionar valores para condiciones en las cláusulas WHERE y HAVING de las sentencias SELECT, UPDATE y DELETE.
-

Ejemplos de subconsultas de FILA SIMPLE:

```
SELECT last_name
FROM employees
WHERE salary >
              (SELECT salary
               FROM employees
               WHERE last_name = 'Abel');
```

```
SELECT last_name, job_id
FROM employees
WHERE job_id =
              (SELECT job_id
               FROM employees
               WHERE employee_id = 141);
SELECT last_name, job_id, salary
FROM employees
WHERE job_id =
              (SELECT job_id
               FROM employees
               WHERE employee_id = 141)
AND salary >
              (SELECT salary
               FROM employees
               WHERE employee_id = 143);
```

Usando funciones de grupo en una subconsulta:

```
SELECT last_name, job_id, salary
FROM employees
WHERE salary =
              (SELECT MIN(salary)
               FROM employees);
```

```
SELECT department_id, MIN(salary)
FROM employees
GROUP BY department_id
HAVING MIN(salary) >
                    (SELECT MIN(salary)
                     FROM employees
                     WHERE department_id = 50);
```

En la siguiente sentencia se produce un error al ser devueltas más de una fila por la subconsulta y no ser adecuado el operador para el tratamiento de más de una fila.

```
SELECT employee_id, last_name
FROM employees
WHERE salary =
              (SELECT MIN(salary)
               FROM employees
               GROUP BY department_id);


ERROR at line 4:
ORA-01427: single-row subquery returns more than one row
```

El operador adecuado para corregir el error sería IN en lugar de =.

Ejemplos con subconsultas que devuelven más de una fila:

```
SELECT last_name, salary, department_id
FROM employees
WHERE salary IN (SELECT MIN(salary)
                 FROM employees
```

```
                GROUP BY department_id);
```

Esta sentencia, teniendo en cuenta el contenido de las tablas, sería equivalente a la siguiente:

```
SELECT last_name, salary, department_id
FROM employees
WHERE salary IN
     (11088,3600,10080,24480,9936,8640,11952,3024,8784,9360,6048,6336);
SELECT employee_id, last_name, job_id, salary
FROM employees
WHERE salary < ANY
                  (SELECT salary
                   FROM employees
                   WHERE job_id = 'IT_PROG')
                   AND job_id <> 'IT_PROG';
```

Ejemplos de tratamiento de NULL en las subconsultas:

La siguiente sentencia muestra los empleados que tienen subordinados. El operador IN es adecuado para el tratamiento de nulos entre los valores devueltos por la subconsulta.

```
SELECT emp.last_name
FROM employees emp
WHERE emp.employee_id IN (SELECT mgr.manager_id FROM employees mgr);
```

En la siguiente sentencia se recuperan los empleados que no tienen subordinados.

```
SELECT last_name FROM employees
WHERE employee_id NOT IN
                        (SELECT manager_id FROM employees
                         WHERE manager_id IS NOT NULL);
```

Siempre que haya valores nulos entre los valores devueltos por una subconsulta no emplear el operador NOT IN pues es equivalente a <>ALL.

```
SELECT emp.last_name
FROM employees emp
WHERE emp.employee_id NOT IN
                           (SELECT mgr.manager_id
                            FROM employees mgr);
```

Usando subconsultas de varias Columnas

No Pairwise

El valor de cada columna es chequeada simultáneamente.

```
SELECT employee_id, manager_id, department_id
FROM employees
 WHERE manager_id IN
            (SELECT manager_id
               FROM employees
              WHERE first_name = 'John')
AND department_id IN
            (SELECT department_id
               FROM employees
```

```
              WHERE first_name = 'John')
AND first_name != 'John';
```

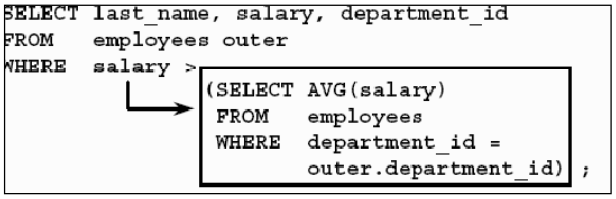
Pairwise

Se chequean las dos columnas simultáneamente.

```
SELECT employee_id, manager_id, department_id
FROM employees
WHERE (manager_id, department_id) IN
              (SELECT manager_id, department_id
                 FROM employees
                WHERE first_name = 'John')
AND first_name != 'John';
```

subconsultas correlacionadas

El servidor de Oracle ejecuta una subconsulta correlacionada cuando la subconsulta hace referencia a una columna de la consulta padre. Se pueden usar los operadores ANY y ALL en este tipo de consultas. El siguiente ejemplo muestra los nombre de las personas que más ganan más que la media de su respectivo departamento. Para cada una de las filas de la consulta outer, la subconsulta es evaluada.



Otros Ejemplos

Se ejecutan para cada una de las filas de la consulta padre.

```
SELECT department_id, (Select max(Salary) from employees where
department_id=d.department_id) max_sal
FROM departments d
```

Uno de los problemas fundamentales de las consultas correlacionadas, es que la subconsulta defininida en la cláusula SELECT devolviera más de una fila.En este caso se define antes de la misma, la cláusula CURSOR.

```
SELECT department_id,CURSOR(Select First_name from employees where
department_id=d.department_id) FROM departments d
```

En algunos casos, este tipo de consultas permiten definir consultas sin la cláusula GROUP BY mejorando los tiempos de respuesta de la misma , y siempre con ayuda del optimizador.

Limitando el número de filas en una consulta (Top-N)

Existen diversos métodos en las versiones anteriores para que de forma indirecta se pueden recuperar N cantidad de registros de los resultados de una consulta. En Oracle12c, esta recuperación se simplifica y se convierte en una sencilla operación mediante la nueva cláusula **FETCH** del comando SELECT.

El formato de esta cláusula es el siguiente:

```
[ OFFSET desplazamiento { ROW | ROWS } ]
[ FETCH { FIRST | NEXT } [ porcentaje PERCENT ]
    { ROW | ROWS } { ONLY | WITH TIES } ]
```

Opciones:

- **OFFSET**. Permite identificar la posición de comienzo de la salida de resultados de la consulta (se inicia en la posición 0). Es como el desplazamiento de la posición de inicio de los resultados.

  Si se utiliza solo esta cláusula en la consulta aparecen los registros desde la posición indicada hasta el final.

- **FETCH**. Establece la forma en que se recuperan los registros. El conjunto de filas puede estar limitado por la cláusula OFFSET.
  - **FIRST**. Los primeros registros del resultado.
  - **NEXT**. Las siguientes filas del conjunto. En ocasiones estas dos cláusulas (ésta y FIRST) funcionan de igual manera.
- **PERCENT**. Indica un porcentaje de los registros que visualizarán.
- **ROWS** { **ONLY** | **WITH TIES**}. Si se establece el parámetro ROWS ONLY sólo recuperará la cantidad de filas indicada. Pero si configuras ROWS WITH TIES puede resultar que aparezcan más registros de los indicados si varias filas coinciden con el valor de la fila enésima.

Ejemplos:

1. Como ejemplo se solicita recuperar los dos empleados que tienen mayor sueldo de la tabla emp.

   En las versiones previas para realizar dicho requerimiento se tendría que construir una consulta similar a esta:

   ```
   SELECT ename,sal
   FROM ( SELECT * FROM emp ORDER BY sal DESC )
   WHERE ROWNUM<=2;
   ```

   Esta nueva versión de Oracle nos ofrece la cláusula FETCH para simplificar la consulta que cumpliría con el requerimiento.

   ```
   SELECT ename,sal
   FROM emp ORDER BY sal DESC
   FETCH FIRST 2 ROWS ONLY;
   ```

2. Debes obtener la tercera y cuarta persona que más sueldo ganen.

En este caso se debe indicar a Oracle que ordene la tabla por el campo sal y que se sitúe en el registro número 3 (OFFSET 2, las filas están numeradas a partir de la posición 0) y recupere 2 registros a partir de esa posición.

```
SELECT ename,sal
FROM emp ORDER BY sal DESC
OFFSET 2 ROWS FETCH NEXT 2 ROWS ONLY;
```

3. Por último se necesita recuperar el 75% de las filas de la tabla emp.

```
SELECT ename,sal
FROM emp ORDER BY sal DESC
FETCH FIRST 75 PERCENT ROWS WITH TIES;
```

Updates correlacionados

En el caso de la sentencia UPDATE, se puede usar una sub consulta correlacionada para actualizar filas en una tabla con base a las filas de otra tabla. En el ejemplo que se muestra a continuación se añade los nombres de los departmentos a los cuales está asociado cada uno de los empleados.

```
ALTER TABLE employees ADD (department_name VARCHAR2(14));
UPDATE employees E SET department_name= (SELECT department_name FROM
departments D WHERE E.department_id=D.department_id)
```

Problemas con la sentencia.

Use una sub consulta correlacionada para actualizar filas en la tabla EMPLOYEES basándose en las filas de la tabla REWARDS:

Este ejemplo usa la tabla REWARDS. La tabla REWARDS tiene la columna EMPLOYEE_ID, PAY_RAISE y PAYRAISE_DATE. Cada vez que un empleado tiene un aumento de sueldo, un registro con el detalle del empleado, la cantidad de incremento y la fecha es insertada en esta tabla. La tabla REWARDS puede contener más de un registro para un empleado. La columna PAYRAISE_DATE es utilizada para identificar el aumento más reciente recibido por un empleado.En el ejemplo, la columna SALARY en la tabla EMPLOYEES es actualizada para reflejar el último aumento recibido para el empleado. Esto es realizado incrementando al salario actual el incremento otorgado.

```
UPDATE employees E
SET salary=
(select employees.salary+rewards.pay_raise
FROM rewards
where employee_id=employees.employee_id
and payraise_date =
              (select max(payraise_date)
                   from rewards
                   where employee_id=employees.employee_id)
    )
WHERE employees.employee_id in (select employee_id from rewards);
```

Delete correlacionados

En el caso de la sentencia DELETE, se puede usar una subconsulta correlacionada para eliminar solo aquellas filas que también existan en otra tabla. El ejemplo siguiente muestra cómo eliminar todos los empleados que coinciden con los existentes en la tabla emp_history.

```
DELETE FROM employees E WHERE employee_id=(SELECT employee_id FROM emp_history
WHERE employee_id= E.employee_id)
```

subconsultas escalares

Son aquellas que obtienen el valor de una columna y una fila concreta. El valor de una expresión en una subconsulta escalar es el valor del elemento de la lista seleccionado de la sub consulta. Si la subconsulta obtiene 0 filas, el valor de la expresión de la subconsulta escalar es nulo. Si la subconsulta obtiene más de una fila, el servidor de Oracle muestra un error.

El uso de una subconsulta escalar ha sido mejorado en

Oracle9i. Ahora se pueden usar sub consultas escalares en:

Condiciones y parte de expresiones de funciones DECODE y CASE

Todas las cláusulas del SELECT excepto GROUP BY

En el lado izquierdo del operador en una cláusula SET y WHERE de una

sentencia UPDATE

Sin embargo, las subconsultas escalares no son expresiones válidas en los siguientes lugares:

- Como valor por defecto para columnas y expresiones para clusters
- En la cláusula RETURNING de sentencias DML
- Como base de una función base indexada
- En la cláusula GROUP BY, constraints CHECK, condiciones WHEN
- Cláusulas HAVING
- En cláusulas START WITH y CONNECT BY
- En sentencias que no son relacionados con consultas, como CREATE PROFILE

Ejemplos

Usando una consulta escalar en la cláusula CASE.

```
SELECT employee_id, last_name,
       (CASE
        WHEN department_id =                    20
             (SELECT department_id FROM departments
              WHERE location_id = 1800)
        THEN 'Canada' ELSE 'USA' END) location
FROM   employees;
```

Usando una consulta escalar en la cláusula Order by

```
SELECT   employee_id, last_name
FROM     employees e
ORDER BY (SELECT department_name
          FROM departments d
          WHERE e.department_id = d.department_id);
```

Operadores Exists,Not Exists,WITH

Operador exists

Este operador es frecuentemente usado en sub consultas correlacionadas para verificar cuando un valor recuperado por la consulta externa existe en el conjunto de resultados obtenidos por la consulta interna. Si la subconsulta obtiene al menos una fila, el operador obtiene el valor TRUE. Si el valor no existe, se obtiene el valor FALSE. El operador EXISTS se asegura que la búsqueda en la consulta interna no continúe cuando al menos una correspondencia sea encontrada para el jefe y número de empleado en la condición:

```
WHERE manager_id = outer.employee_id.
```

Ejemplo

```
SELECT employee_id,last_name,job_id,department_id
FROM employees outer
WHERE EXISTS (select 'x' from employees where manager_id=outer.employees_id)
```

Note que el SELECT de la consulta interna no necesita obtener un valor específico, de tal manera que una constante puede ser seleccionada. Desde el punto de vista de la ejecución, es más rápido seleccionar una constante que una columna. Teniendo EMPLOYEE_ID en la cláusula SELECT de la consulta interna, causa una búsqueda de esa columna en la tabla. Remplazando esta por una literal X, o cualquier constante, mejora el rendimiento.

Operador not exists

Consecuentemente, NOT EXISTS verifica cuando un valor recuperado por la consulta externa no es parte del conjunto de resultados obtenidos por la consulta interna.

Operador with

También conocida como “subquery_factoring_clausula” .La cláusula WITH, permite reutilizar la misma consulta cuando es de alto coste evaluar el bloque de la consulta y ocurre más de una vez en una consulta compleja. Usando la cláusula WITH, el servidor de Oracle recupera los resultados de un bloque de la consulta y los almacena en un tablespace temporal del usuario. Esto puede mejorar el rendimiento de las mismas.

Beneficios

Beneficios de la cláusula WITH:

• Hace que la consulta sea fácil de leer

• Evalúa una cláusula una sola vez, aun si esta aparece muchas veces en la consulta, por esta razón aumenta el desempeño

Restricciones

Restricciones de la cláusula WITH:

• No puede consultas anidadas realizadas con WITH

• No puede hacer referencia mediante operadores, a la cláusula WITH pero si se puede hacer referencia en la cláusula FROM

Ejemplo

```
col M:S format a10
WITH t AS (
  SELECT 100 s FROM DUAL
  UNION ALL
  SELECT 7201 FROM DUAL);
SELECT s,TRUNC(s/60)||':'||MOD(s,60) "M:S"
FROM t;
```

Recuperación jerárquica

Utilizando recuperaciones jerárquicas se pueden recuperar datos basados en relaciones jerárquicas entre las filas de una tabla.

En el caso de la tabla EMPLOYEES, la relación existente entre los jefes y sus empleados se establece a través de la columna MANAGER_ID y EMPLOYEE_ID. Un jefe tiene varios empleados, y éstos a su vez pueden ser jefes de otros empleados.

Sintaxis:

```
SELECT [LEVEL], columna, expr...
FROM tabla
[WHERE condición(es)]
[START WITH condición(es)]
[CONNECT BY PRIOR condición(es))];
```

- Level: Es el nivel que ocupa el valor dentro de la estructura jerárquica. Retorna un number.
- Start with: Por donde queremos que empiece la jerarquía.
- Connect by: En esta cláusula, se vincula los dos campos que tienen relación jerárquica.
- Prior: Establece niveles de jerarquía empezando por START WITH.

```
SELECT employee_id, last_name, job_id, manager_id
FROM employees
START WITH employee_id = 101
CONNECT BY PRIOR manager_id = employee_id;
```

EMPLOYEE_ID	LAST_NAME	JOB_ID	MANAGER_ID
101	Kochhar	AD_VP	100
100	King	AD_PRES	

En el siguiente ejemplo. Visualizar el número de empleado, nombre, trabajo, identificador del jefe y nivel, realizando una recuperación jerárquica ordenada por niveles:

```
SELECT employee_id, last_name, job_id, manager_id,level
FROM employees
START WITH manager_id is null
CONNECT BY  manager_id = PRIOR employee_id
ORDER BY level;
```

EMPLOYEE_ID	LAST_NAME	JOB_ID	MANAGER_ID	LEVEL
100	King	AD_PRES		1
101	Kochhar	AD_VP	100	2
102	De Haan	AD_VP	100	2
121	Fripp	ST_MAN	100	2
120	Weiss	ST_MAN	100	2
114	Raphaely	PU_MAN	100	2
122	Kaufling	ST_MAN	100	2
124	Mourgos	ST_MAN	100	2
146	Partners	SA_MAN	100	2
201	Hartstein	MK_MAN	100	2
149	Zlotkey	SA_MAN	100	2

148	Cambrault	SA_MAN	100	2
147	Errazuriz	SA_MAN	100	2
145	Russell	SA_MAN	100	2
123	Vollman	ST_MAN	100	2
108	Greenberg	FI_MGR	101	3
204	Baer	PR_REP	101	3
203	Mavris	HR_REP	101	3
200	Whalen	AD_ASST	101	3

...

1Es posible formatear la salida en sqlplus utilizando LEVEL:

```
COLUMN org_chart FORMAT A12          -- Formato a columna

SELECT LPAD(last_name, LENGTH(last_name)+(LEVEL*2) -2,'_') AS org_chart
FROM employees
START WITH last_name='King'
CONNECT BY PRIOR employee_id=manager_id;
```

ORG_CHART
King
King
__Kochhar
____Greenber g
______Faviet
______Chen
______Sciarr a
______Urman

_____Popp
____Whalen
____Mavris

DML. Lenguaje de Manipulación de Datos

Cuando se añaden, actualizan o borran datos en una base de datos, se está ejecutando una sentencia DML. Una colección de sentencias DML conforman una unidad lógica de trabajo llamada transacción.

INSERT

- El propósito de la sentencia INSERT es añadir filas a una tabla o vista.
- Como requisitos para añadir filas en una tabla, la tabla debe ser propiedad del usuario que intenta realizar la inserción o bien tener privilegio de INSERT sobre ella.

Sintaxis:

```
INSERT INTO tabla [(columna [, columna...])]
VALUES (valor [, valor...]);
```

- Tabla: Nombre de tabla o vista donde se realiza la inserción.
- Columna: Columna/s sobre la/s que se desea realizar la inserción. Si se omiten los nombres y el orden de las columnas, se espera que se inserten valores para TODAS las columnas de la tabla o vista y en el orden en que fue creada.
- VALUES: Especifica un valor para la fila que se desea insertar.

En lugar de la cláusula VALUES se puede utilizar una subconsulta que recupere datos coordinados con las columnas a insertar.

```
INSERT INTO tabla [(columna [, columna...])]
SUBCONSULTA;
```

Ejemplos de INSERT:

Insertar una fila.

```
INSERT INTO departments(department_id, department_name, manager_id,
                        location_id)
VALUES (280, 'Jobbes', 100, 1700);
```

2

Insertar una fila con valores nulos.

```
INSERT INTO departments (department_id,department_name )
VALUES (290, 'Art');

INSERT INTO departments
VALUES (300, 'Tecnology', NULL, NULL);
```

Insertar valores especiales como la fecha del sistema.

```
INSERT INTO employees (employee_id, first_name, last_name, email,
                       phone_number, hire_date, job_id, salary,
                       commission_pct, manager_id, department_id)
VALUES (207,'Louis', 'Kunts','LKUNT', '515.124.4567',SYSDATE,'AC_ACCOUNT',
        6900,NULL, 205, 100);
```

Insertar datos específicos tipo fecha.

```
INSERT INTO employees
VALUES (214,'Den', 'Marck','DMARCK', '515.127.4561',
        TO_DATE('FEB 3, 1999', 'MON DD, YYYY'),'AC_ACCOUNT', 11000,
        NULL, 100, 30);
```

Insertar filas procedentes de otras tablas.

```
INSERT INTO emp(empno, ename , sal, deptno)
     SELECT employee_id, last_name, salary, 40
     FROM employees
     WHERE job_id LIKE '%REP%';
```

UPDATE

- El propósito de la sentencia UPDATE es modificar los valores existentes en una tabla o vista.
- Como requisitos para actualizar filas de una tabla, la tabla debe ser propiedad del usuario que intenta realizar la actualización o bien tener privilegio de UPDATE sobre ella.

Sintaxis:

```
UPDATE tabla
SET columna = value [, column = valor, ...]
[WHERE condición];
```

- Tabla: Nombre de tabla o vista donde se realiza la actualización.
- Columna: Columna/s sobre la/s que se desea realizar la actualización. Si se omite el nombre de alguna columna de la tabla/vista, esta permanecerá inalterable.
- WHERE: La condición que deben cumplir las filas que se desea actualizar. Si se omite se actualizan TODAS.

La sentencia UPDATE admite especificar subconsultas que devolverá/n los valores nuevos que se asignarán a la columna/s correspondiente/s.

Sintaxis:

```
UPDATE table
SET column =
            (SELECT columna
             FROM tabla
             WHERE condición)
 [, column =
            (SELECT columna
             FROM tabla
             WHERE condición)]
[WHERE condición];
```

Ejemplos de sentencias UPDATE:

Actualizar una fila específica.

```
UPDATE employees
SET department_id = 70
WHERE employee_id = 113;
```

Actualizar todas las filas de una tabla.

```
UPDATE employees
SET salary = salary*1.2;
```

Actualizar una tabla mediante subconsultas.

```
UPDATE employees
SET job_id = (SELECT job_id
              FROM employees
              WHERE employee_id = 205),
    salary = (SELECT salary
              FROM employees
              WHERE employee_id = 205)
WHERE employee_id = 114;
```

DELETE

- El propósito de esta sentencia, es borrar filas de una tabla o vista.
- Como requisitos para borrar filas de una tabla, la tabla debe ser propiedad del usuario que intenta realizar el borrado o bien, tener privilegio de DELETE sobre ella.

Sintaxis:

```
DELETE [FROM] tabla
[WHERE condición];
```

- Tabla: Nombre de tabla o vista donde se realiza el borrado de las filas.
- WHERE: La condición que deben cumplir las filas que se desean borrar. Si se omite se borran TODAS. Esta cláusula admite subconsultas.
-

Ejemplos de sentencias DELETE:

Eliminar una fila especificando una condición.

```
DELETE FROM departments
WHERE department_name = 'Operations';
```

Eliminar todas las filas de una tabla.

```
DELETE FROM salgrade;
```

Eliminar filas utilizando una subconsulta.

```
DELETE FROM employees
WHERE department_id =
                     (SELECT department_id
                      FROM departments
                      WHERE department_name LIKE '%Public%');
```

MERGE

El lenguaje SQL ha sido ampliado para incluir la sentencia MERGE. Utilizando esta sentencia es posible actualizar o insertar una fila condicionalmente. La decisión de actualizar o insertar la fila está basada en una condición especificada en la cláusula ON.

- Realiza un UPDATE si la fila existe y un INSERT si es una nueva fila:
  - Evita UPDATES separados.
  - Aumenta su funcionalidad y es fácil de usar.
  - Es útil para aplicaciones de data warehousing.

Sintaxis:

```
MERGE INTO nombre_tabla alias_tabla
USING (tabla|vista|sub_consulta) alias
ON (condicion de join)
WHEN MATCHED THEN
     UPDATE SET
     col1 = valor,
     col2 = valor
WHEN NOT MATCHED THEN
     INSERT (lista de columnas)
     VALUES (valores);
```

- INTO: especifica la tabla o vista objetivo sobre la que se actualiza o inserta.
- USING: identifica el origen de los datos para la actualización o inserción. Puede ser una tabla, vista o subconsulta.
- ON: especifica la condición que utilizará MERGE para escoger la operación de actualización o inserción.
- WHEN MATCHED | WHEN NOT MATCHED: especifica al servidor cómo actuar ante los resultados de la condición de join. (cuando hay coincidencia o cuando no hay coincidencia).

Ejemplo de sentencia MERGE:

- Actualizar o insertar filas en una tabla copia de EMPLOYEES llamada COPY_EMP para hacerla coincidir las filas comunes con las de la tabla EMPLOYEES e insertar las que no se encuentren.
-

```
MERGE INTO copy_emp c
USING employees e
ON (c.employee_id = e.employee_id)
WHEN MATCHED THEN
UPDATE SET
c.first_name = e.first_name,c.last_name = e.last_name,
c.email = e.email,c.phone_number = e.phone_number,
c.hire_date = e.hire_date,c.job_id = e.job_id,
c.salary = e.salary,c.commission_pct = e.commission_pct,
c.manager_id = e.manager_id,c.department_id = e.department_id
 WHEN NOT MATCHED THEN
```

```
INSERT VALUES(e.employee_id, e.first_name, e.last_name,
e.email, e.phone_number, e.hire_date, e.job_id,
e.salary, e.commission_pct, e.manager_id,e.department_id);
```

Extensiones a la sentencia MERGE

Hay dos mejoras importantes en la sentencia MERGE a partir de Oracle10g:

- Nuevas cláusulas condicionales, así como mejoras de las cláusulas ya existentes, haciendo que la sentencia sea más fácil de usar y agilizando su ejecución.
  - Cláusulas UPDATE o INSERT opcionales. Ahora es posible omitir alguna de las cláusulas.
  - Permite cláusulas UPDATE condicionales.
  - Permite cláusulas INSERT condicionales.
  - Reconocimiento y evaluación en tiempo de compilación de predicados ON, que incluyen condiciones especiales.
- Cláusula opcional DELETE.

Cláusulas INSERT o UPDATE opcionales

Es posible omitir las cláusulas UPDATE o INSERT en una sentencia MERGE. En el ejemplo siguiente, no se incluye la sentencia INSERT.

```
SQL> create table emp_hist as select * from employees where 1=0;
Table created.
```

```
SQL> merge into emp_hist h using employees e
  2  on (h.employee_id = e.employee_id)
  3  when matched then update set h.salary=e.salary;
SQL> merge into emp_hist h using employees e
  2  on (h.employee_id = e.employee_id)
  3  when not matched then
  4    insert (employee_id, first_name, last_name, email, hire_date, job_id,
  5  department_id)
  6    values (e.employee_id, e.first_name, e.last_name, e.email, e.hire_date,
  7  e.job_id, e.department_id);
```

Cláusulas UPDATE condicionales

Es posible añadir una condición WHERE a la sentencia UPDATE, que compruebe una condición cuando se desee no realizar la operación de UPDATE.

```
SQL> merge into emp_hist h using employees e
  2  on (h.employee_id = e.employee_id)
  3  when matched then update set h.salary=e.salary
  4                    where h.salary<>e.salary;
```

Condiciones ON Especiales

Para insertar todas las filas de la tabla origen en otra tabla sin unir ambas tablas origen y destino, se puede utilizar el predicado ON constante.

En el siguiente ejemplo, la tabla NEW_PRODUCT será una copia vacía de la tabla PRODUCTS excepto una columna renombrada.

```
MERGE INTO new_products np
  USING products p
  ON (1=0)
WHEN NOT MATCHED THEN
  INSERT (np.prod_id, np.prod_new_status)
  VALUES (p.prod_id, p.prod_status);
```

INSERT Condicionales

Es posible añadir una condición WHERE a la sentencia INSERT, que compruebe una condición cuando se desee no realizar la operación.

```
MERGE INTO emp_hist h USING employees e
ON (h.employee_id = e.employee_id)
WHEN NOT MATCHED THEN
  INSERT (employee_id, first_name, last_name, email, hire_date, job_id,
department_id)
  VALUES (e.employee_id, e.first_name, e.last_name, e.email, e.hire_date,
e.job_id, e.department_id)
  WHERE e.commission_pct IS NOT NULL;
```

Cláusula DELETE Opcional

Es posible vaciar una tabla a la vez que se insertan o modifican datos mediante una sentencia MERGE.

```
MERGE INTO emp_hist h USING employees e
ON (h.employee_id = e.employee_id)
WHEN MATCHED THEN
  UPDATE SET h.salary=e.salary WHERE e.commission_pct IS NOT NULL
  DELETE WHERE e.commission_pct IS NULL
WHEN NOT MATCHED THEN
  INSERT (employee_id, first_name, last_name, email, hire_date, job_id,
          department_id)
  VALUES (e.employee_id, e.first_name, e.last_name, e.email, e.hire_date,
          e.job_id, e.department_id)
  WHERE e.commission_pct IS NOT NULL;
```

Sentencias Transaccionales

- Oracle Server asegura la consistencia de los datos basándose en transacciones. Las transacciones permiten flexibilidad y control durante los cambios de los datos, y aseguran la consistencia en el caso de que algún proceso de usuario o del sistema falle.
- Una transacción de base de datos está provocada por una o varias sentencias tipo DML, por sólo una tipo DDL o DCL.
- CONSISTENCIA EN LECTURA: los usuarios acceden a la base de datos de dos formas: con operaciones de lectura (SELECT) o de escritura (DML). En función de estos tipos de operaciones y teniendo en cuenta la concurrencia de sesiones que acceden a los mismas tablas es necesario garantizar la consistencia de los datos ante las siguientes consideraciones:
  - Las lecturas y escrituras en la base de datos deben asegurar una vista consistente de los datos.

  - Las lecturas de una sesión no deben ver datos que estén en proceso de cambio en otra sesión diferente.
  - Las escrituras tienen que tener asegurado que los cambios que realicen sobre la base de datos se llevan a cabo sobre datos consistentes.
  - Los cambios hechos por una operación de escritura no debe entrar en conflicto con cambios no consolidados llevados a cabo desde otra sesión.
  - El objeto de la consistencia en lectura es asegurar que antes de que se ejecute una sentencia DML en la sesión, cada usuario ve los datos consistentes, es decir, como quedaron desde el último commit que los validó en la base de datos.
- BLOQUEO: los bloqueos son mecanismos que previenen una interacción destructiva entre transacciones accediendo a un mismo recurso.
- Oracle implementa el bloqueo de forma automática sin requerir una acción por parte del usuario. Este bloqueo se lleva a cabo para todas las sentencias SQL en las que sea necesario de forma automática, excepto para la SELECT, que requerirá de un bloqueo explícito por parte del usuario (… for update of column).
- Una transacción comienza cuando la primera sentencia SQL DML es ejecutada durante una sesión y finaliza cuando se produce alguno de estos eventos:
  - Se ejecuta una sentencia transaccional COMMIT o ROLLBACK.
  - Se ejecuta una sentencia DDL o DCL (se produce un COMMIT implícito).
  - La sesión del usuario finaliza de manera anormal (fallo de sistema) o de forma voluntaria (salida de la herramienta).
- Durante una sesión de usuario únicamente una transacción puede estar en vigor.
- Las sentencias transaccionales son:
  - COMMIT
  - ROLLBACK
  - SAVEPOINT

COMMIT

Finaliza la transacción actual, valida todos los cambios de datos haciéndolos permanentes.

Sintaxis:

```
COMMIT [WORK];
```

- WORK: es opcional y no aporta ninguna funcionalidad.
- Al ejecutar un COMMIT, se liberan todos los o puntos de salvaguarda, SAVEPOINT indicados hasta el momento.
- La ejecución de sentencias DDL y DCL llevan consigo un commit implícito (commit automático)

SAVEPOINT

- Los puntos de salvaguarda son marcas lógicas que va poniendo el usuario durante la transacción actual.
- Estas marcas permiten deshacer los cambios por partes en vez de deshacer toda la transacción.

- Al ser marcas lógicas no es posible listar los puntos de salvagurarda creados.

```
SAVEPOINT <punto de salvaguarda>;
```

- Si se reutiliza un nombre de punto de salvaguarda durante la transacción, el anterior con el mismo nombre se pierde.
- Al hacer el ROLLBACK sin parámetros o el COMMIT, se eliminan todos los puntos de salvaguarda.
- Al hacer el ROLLBACK TO solo se borran los puntos posteriores al especificado.
- SAVEPOINT no es ESTÁNDAR ANSI SQL.

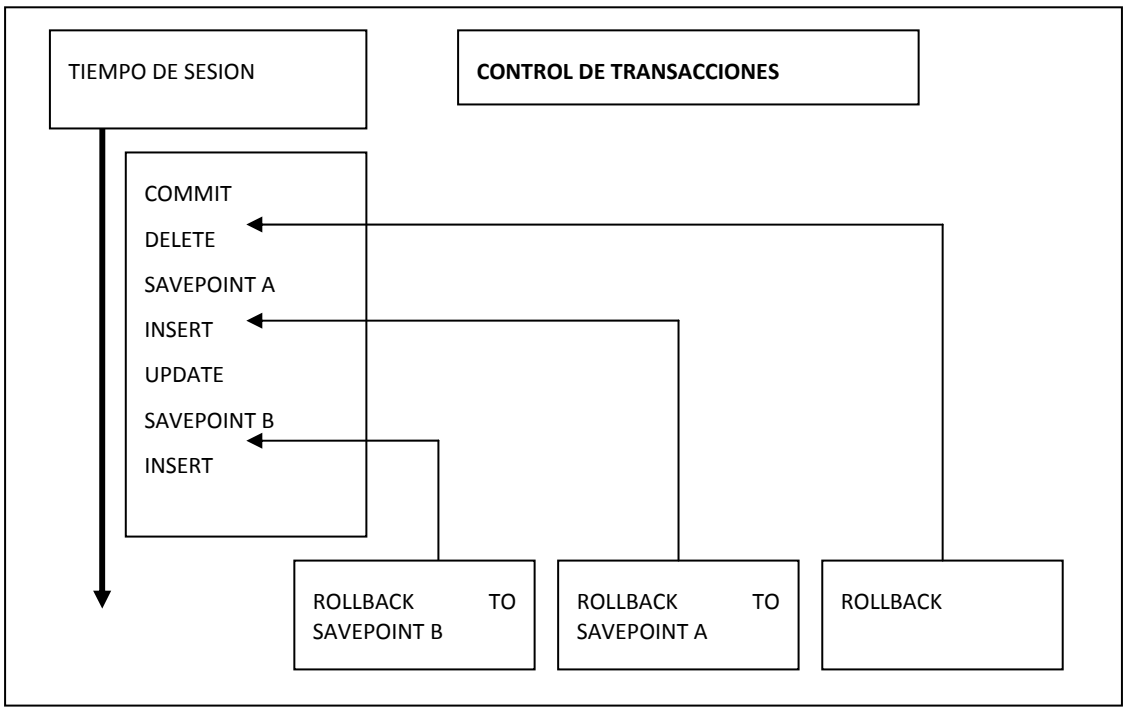
ROLLBACK

- Finaliza la transacción actual y deshace todos los cambios de datos pendientes.

Sintaxis:

```
ROLLBACK [WORK] [TO [SAVEPOINT] punto_salvaguarda];
```

- ROLLBACK: deshace los cambios pendientes.
- WORK: es opcional y no aporta ninguna funcionalidad.
- TO [SAVEPOINT] <punto_salvaguarda>: deshace sólo los cambios efectuados desde el punto de salvaguarda indicado.
- Una terminación anormal de la sesión o un fallo del sistema lleva consigo un rollback implícito (rollback automático).
-



-

4

Lenguaje de Definición y Lenguaje de Control de Datos. DDL y DCL

Tabla de contenidos

DDL. Lenguaje de Definicion de Datos 1

Comandos DDL 1

CREATE TABLE 1

Sintaxis CREATE TABLE 1

Integridad de Datos 4

Reglas de Negocio 4

Integridad de Entidades 4

Integridad Referencial 5

Nombres de las Restricciones 6

Cómo Definir Una Restricción 7

Restricciones: características. 8

Comprobación de restricciones aplazada 8

Restricciones obligatorias 9

Restricción UNIQUE o PRIMARY KEY utilizando un índice no único. 10

CREATE INDEX 10

Tablas tipo índice 11

Creación de varios índices sobre la misma columna en una tabla 13

CREATE SYNONYM 13

CREATE VIEW 14

CREATE SEQUENCE 15

Campos cuyo valor DEFAULT referencia a una secuencia 16

Secuencias a nivel de sesión 17

Campos autonuméricos generados a través de IDENTITY 18

ALTER TABLE 20

Añadir Restricciones 21

Desactivar Restricciones 21

Activar Restricciones 22

Excepciones Activando Restricciones 23

Borrar Restricciones 24

Borrar Columnas 24

Columnas invisibles 25

ALTER SEQUENCE **27**
DROP TABLE **27**
DROP INDEX **27**
DROP VIEW **28**
DROP SYNONYM **28**
DROP SEQUENCE **28**
RENAME **28**
TRUNCATE **29**
Privilegios **29**
Privilegios sobre objetos 30

DDL. Lenguaje de Definicion de Datos

El lenguaje DDL pertenece al lenguaje SQL y nos permite crear, modificar, renombrar y borrar objetos dentro de la Base de Datos. Hasta ahora solo habíamos manipulado objetos ya creados previamente.

Para poder ejecutar los comandos que veremos a continuación es necesario tener una serie de privilegios, que el administrador de la Base de Datos ha debido concedernos previamente.

Llevan implícito la validación o COMMIT.

Comandos DDL

CREATE	El comando CREATE permite al usuario que lo ejecuta crear un objeto (Tablas, índices, vistas, sinónimos, secuencias, etc.)
ALTER	El comando ALTER permite al usuario que lo ejecuta modificar la estructura de un objeto previamente creado.
DROP	El comando DROP permite al usuario que lo ejecuta eliminar un objeto (no sólo las columnas) de la Base de Datos.
RENAME	Permite al usuario que lo ejecuta modificar el nombre de un objeto.
TRUNCATE	Permite al usuario que lo ejecuta borrar TODAS las filas de una tabla o índice. Tiene ciertas diferencias con DELETE o DROP que veremos próximamente.

CREATE TABLE

Permite crear una tabla, que es la estructura básica de una base de datos.

En el momento de la creación de la tabla se puede especificar la siguiente información:

- Definiciones de columnas. (tipos de datos y longitud).
- Restricciones de Integridad.
- Características de Almacenamiento del objeto.
- Si la tabla se forma partiendo de una sentencia SELECT.

Sintaxis CREATE TABLE

```
CREATE TABLE [esquema.]nombre_tabla (
COL1         tipo_Dato_col1 (longitud) [restricción_int] [,
COL2         tipo_Dato_col2 (longitud) [restricción_int],
COL3         tipo_Dato_col3 (longitud) [restricción_int],
…
COLn         tipo_Dato_coln (longitud) DEFAULT [expresión]]
[, Declaración opcional de mas restricciones de integridad]      )
[PARAMETROS DE ALMACENAMIENTO]
[ AS subconsulta]
```

El siguiente ejemplo muestra la creación de una tabla sin indicar ningún tipo de restricción o cláusula de almacenamiento. Solamente se indican los nombres de las columnas, a qué tipos de dato pertenecen y la longitud de las columnas.

```
CREATE TABLE productos (
Codigo_prod      NUMBER(4),
Descripcion      VARCHAR2(80));
```

El siguiente ejemplo muestra la creación de una tabla basada en una sentencia SELECT

```
CREATE TABLE control_salarios (trabajo, maxsal, minsal)
    AS SELECT job, MAX(salario), MIN(salario) FROM emp GROUP BY job;
```

Los tipos de datos que podemos definir en las columnas de nuestras tablas son:

Tipo Dato	Descripción
VARCHAR2(tam)	Cadena de caracteres de longitud variable. Tamaño máximo es 4000 bytes y el mínimo uno. Es obligatorio especificar precisión.
NVARCHAR2(tam)	Cadena de caracteres de longitud variable teniendo un máximo de longitud expresado en caracteres o bytes, depende del juego de caracteres nacional indicado. El tamaño máximo se determina por el número de bytes requerido para almacenar cada carácter. El límite superior es 4000 bytes. Es obligatorio especificar precisión.
NUMBER (n,d)	Numero de tamaño 'n' (hasta 38) conteniendo 'd' decimales (de -84 a 127)
LONG	Cadena de caracteres de longitud variable. Tamaño hasta 2 Giga bytes. Solo puede haber uno por tabla. No pueden tener restricciones de integridad. No pueden utilizarse en cláusulas WHERE, GROUP BY, CONNECT BY, DISTINCT, ni consultas anidadas ni operadores de conjuntos.
DATE	Fecha mostrada con la máscara 'DD-MES-AA”. Depende variables "NLS". Siete bytes numéricos.' Rangos válidos son desde 1 de Enero de 4712 Después de Cristo a 31 de Diciembre de 4712 Antes de Cristo.
RAW(tam)	Dato binario de longitud variable. Tamaño máximo 2000 bytes. Se debe indicar precisión.

LONG RAW	Cadena binaria de longitud variable de hasta 2 Gigabytes.
ROWID	Cadena hexadecimal que representa la dirección única de una fila en una tabla. Este tipo de dato se utiliza para almacenar los valores devueltos par la pseudocolumna ROWID.
CHAR(tam)	Cadena alfanumérica de longitud fija. Tamaño máximo 2000 bytes. Si no se indica precisión el valor mínimo y valor por defecto es 1 byte.
NCHAR(tam)	Cadena alfanumérica de longitud fija. Teniendo un máximo de longitud expresado en caracteres o bytes, depende del juego de caracteres nacional indicado. El tamaño máximo se determina por el número de bytes requerido para almacenar cada carácter. El límite superior es 2000 bytes. Si no se indica precisión el valor mínimo y valor por defecto es 1 byte.
MLSLABEL	Formato binario y etiquetado bajo el sistema operativo. Ese tipo de dato aparece por compatibilidad hacia trastas en “Oracle Trusted”.
CLOB	Objeto grande tipo carácter conteniendo caracteres de byte simple. No se soportan juegos de caracteres de tamaño variable. Máximo 4 gigabytes.
NCLOB	Objeto grande tipo carácter conteniendo caracteres multibyte de tamaño fijo. No se soportan juegos de caracteres de tamaño variable. Máximo 4 gigabytes
BLOB	Un objeto grande binario. Tamaño máximo 4 Gigabytes.
BFILE	Contiene un localizador a un fichero binario almacenado fuera de la base de datos. Tamaño máximo 4 gigabytes.

BINARY_FLOAT	Este tipo de datos existe a partir de Oracle 10g, almacenan datos numéricos de coma flotante en formato IEEE 754 de 32 bits de precisión.
BINARY_DOUBLE	Este tipo de datos existe a partir de Oracle 10g, almacenan datos numéricos de coma flotante en formato IEEE 754 de 64 bits de precisión.

La información acerca de las tablas y columnas en el diccionario de datos, se encuentra en las siguientes vistas del diccionario accesibles para cualquier usuario:

- USER_TABLES y ALL_TABLES
- USER_TAB_COLUMNS y ALL_TAB_COLUMNS

Integridad de Datos

Antes de definir una tabla debemos tener en cuenta ciertos aspectos acerca de la información a almacenar que nos evitan tener que programar de forma redundante ciertas comprobaciones.

La declaración de restricciones de integridad nos ayuda a definir ciertas características de la tabla. Por ejemplo, si una columna puede estar vacía o no, si puede contener valores duplicados, si está relacionada con alguna columna de otra tabla, etc.

La integridad de datos por lo tanto es:

- Asegurarse que los datos en las tablas se ajustan a una serie de reglas predefinidas.
- Las condiciones se cumplen en todas las transacciones y desde todas las herramientas y programas.
- Se definen una sola vez y no necesitan mantenimiento especial.

Conceptos de integridad que se pueden definir:

- Reglas negocio.
- Integridad de entidades.
- Integridad referencial.

Reglas de Negocio

Específica cada área de negocio. Por ejemplo: Sólo cuando se venden 10 productos o más hay derecho a descuento.

Integridad de Entidades

Reglas a las que deben ajustarse las columnas de mis tablas. Pueden definirse las siguientes:

- NOT NULL. La columna a la que se aplique esta restricción debe contener valores. Por ejemplo: la columna número de Identificación fiscal de la tabla facturas.
- Primary Key. Identificador único de una fila en una tabla. Solo puede haber una por tabla, pero puede estar compuesto por más de una columna. Se utiliza de forma

habitual para ser referenciada desde otra tabla. Al definir una columna como clave primaria de forma implícita, se crea un índice asociado y la columna quedará definida como única y obligatoria si no se crea desactivado. Lleva incluida la restricción NOT NULL.

Por ejemplo, el código de un producto es su identificativo en la tabla de los productos y además le ayuda a relacionarse con la tabla que contiene los precios de los productos o con la de almacén que contiene la cantidad de un producto que nos queda en stock.

- UNIQUE. No puede haber más de dos filas con el mismo valor en aquella columna/s que tenga dicha restricción. Puede ser simple o compuesta de varias columnas. A diferencia de la clave primaria, no es obligatoria y además puede haber más de una por tabla. Si se quiere trabajar bajo normativa ANSI/ISO la columna que lleve esta restricción deberá tener adicionalmente la restricción NOT NULL.

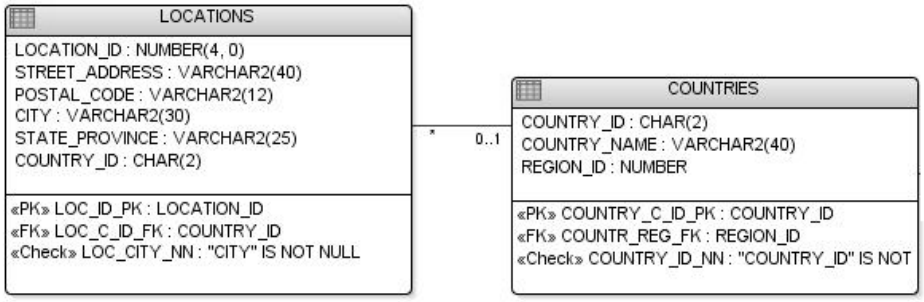
Por ejemplo, una restricción de unicidad sobre el código de una película y el número de copia en la tabla prestamos_videos.

- CHECK. Restricción que realiza un filtro sobre las filas que no cumplan la condición impuesta. Es similar a la cláusula WHERE pero se implementa en la definición del objeto. No puede incluir subconsultas ni variables o pseudocolumnas (user, sysdate, etc).

Por ejemplo: Comprobar que el precio de un producto es mayor que cero.

Integridad Referencial

Es la que determina las relaciones existentes entre los objetos. Por ejemplo existe una relación entre las tablas Locations y Countries.



- PRIMARY KEY (Clave primaria). Identificador único de una fila en una tabla. Hemos visto anteriormente que este concepto pertenece a la integridad de entidades.
- Foreign Key. Columna normalmente asociada a una clave primaria o única de otra tabla o de la misma tabla (denominado auto-referencia). Puede haber tantas como se desee por tabla. Su valor puede ser nulo o el valor de la clave referenciada. Garantiza consistencia de relaciones entre los datos.

Por ejemplo: Toda línea de factura debe tener su cabecera, o lo que es lo mismo, debe pertenecer a una línea de factura.

Estableciendo estas relaciones entre claves ajenas y primarias logramos:

- Evitar borrados, inserciones o actualizaciones no deseadas: Por ejemplo, no se puede borrar una factura que tiene líneas. No se puede dar de alta una línea de factura sin

su correspondiente cabecera. No se puede actualizar el identificativo de una línea de factura.

Propagar Modificaciones. Lo contrario a lo anterior. Se pueden definir acciones de borrado en cascada. Por ejemplo, si se borra una cabecera de factura que se borren las correspondientes líneas usando ON DELETE CASCADE. ON DELETE SET NULL coloca nulos todas las claves secundarias relacionadas con la borrada.

DEFAULT. No es una restricción en sí, pero que proporciona la posibilidad de indicar un valor por defecto a columnas que no se mencionan en un INSERT. Admite variables del sistema como USER, UID o SYSDATE. Por ejemplo, la fecha de emisión de una factura es una columna tipo DATE y si no se le indica un valor explícitamente, tendrá el de la fecha del sistema.

La sentencia CREATE TABLE se utiliza para establecer todas las relaciones y reglas anteriormente citadas. Es la llamada sintaxis declarativa. Véase el siguiente ejemplo.

```
CREATE TABLE depart (
          Deptno NUMBER(2)    CONSTRAINT cp_dept  PRIMARY KEY,
          Dname  VARCHAR2(13) UNIQUE NOT NULL,
          Loc    VARCHAR2(13) );

CREATE TABLE emple (
          Empno  NUMBER(4)    CONSTRAINT cp_emp   PRIMARY KEY,
          Ename  VARCHAR2(13),
          …
          Deptno NUMBER(2)    CONSTRAINT ca_emp_deptno
                              REFERENCES depart(deptno) ON DELETE CASCADE);
```

Se puede establecer la especificación de restricciones según el siguiente ejemplo. Para una gestión bancaria crear la tabla préstamos con todas sus restricciones y reglas.

```
CREATE TABLE prestamos (
    Nocuenta     NUMBER(6)    CONSTRAINT  nn_pres_nocuenta  NOT NULL,
    Noprestamo   NUMBER(6)    CONSTRAINT  nn_pres_noprestamo NOT NULL,
    Tipopres     VARCHAR2(8)  CONSTRAINT  ck_pres_tipopres
                        CHECK (tipopres IN ('PERS', 'CASA', 'COCHE')),
    cantidad     NUMBER(8,0)  CONSTRAINT nn_prest_cantidad NOT NULL,
    fechapres    DATE         DEFAULT sysdate,
    aprobadopor  VARCHAR2(15) CONSTRAINT ca_prest_aprobadopor
                              REFERENCES jefes(nombre_dir),
    CONSTRAINT cp_pres        PRIMARY KEY (nocuenta, noprestamo),
    CONSTRAINT ca_pres_cuenta FOREIGN KEY (nocuenta)
                        REFERENCES  clientes (nocuenta));
```

Nombres de las Restricciones

- Es un nombre único, definido bien por el propietario del objeto o por el sistema. Si lo define el propietario del objeto en el momento de definir la restricción, debe ir precedido de la palabra CONSTRAINT nombre_restricción tipo_restricción.
- Si no se le asigna en el momento de la creación, el sistema generará uno por defecto: SYS_C00n.
- En el ejemplo anterior se ha utilizado una notación para nombrar las restricciones.

La sintaxis es la siguiente:

```
Codigorestricción_nombretabla_nombrecolumna
```

Donde el Código de restricción puede ser:

- CP = CLAVE PRIMARIA
- UQ = UNICIDAD
- CA = CLAVE AJENA
- CK = CHECK
- NN = OBLIGATORIO

El nombre de la restricción aparecerá en los mensajes de error, en la documentación, al activarla y desactivarla temporalmente o para borrarla.

La información acerca de las restricciones definidas por el usuario, y sobre qué columnas han definido se encuentra en las siguientes tablas del diccionario de datos USER_CONSTRAINTS o USER_CONS_COLUMNS

Cómo Definir Una Restricción

Hay dos formas de definir restricciones: "En línea" o "No en línea".

- Definir una restricción "En línea", significa que la restricción va a continuación del nombre de la columna, el tipo de dato y la precisión. En este caso, no es obligatorio darles un nombre con la palabra clave CONTRAINT, pero se generará uno por defecto. Por ejemplo:
-

```
CREATE TABLE facturas (
…
    Num_fra  NUMBER(6)  [CONSTRAINT nombre] PRIMARY KEY,
…);
```

- Definir una restricción "No en línea" significa que la restricción/es va/n antes o después de empezar a definir las columnas. En este caso es obligatorio preceder la restricción con la palabra clave CONSTRAINT para que el gestor distinga entre lo que es la definición de la columna o la declaración de la restricción. En este caso la sintaxis varía, y se define de este modo:

```
CONSTRAINT nombre_restricción TIPO_RESTRICCIÓN (col1 [,…,coln])
```

Al definir una restricción NO EN LINEA, hay que indicar a qué columna/s de la tabla se aplica la restricción, puesto que no está a continuación de la definición. En ciertos casos, como cuando una clave primaria está compuesta de varias columnas, o cuando una restricción se añade a posteriori, es obligatorio definirlas como NO EN LINEA.

A continuación se muestra un ejemplo de restricción NO EN LINEA:

```
CREATE TABLE facturas (
 numero_factura     NUMBER(6),
 columnaN           TIPO_DATO(PRECISION),
     CONSTRAINT cp_facturas_numerofactura PRIMARY KEY (numero_factura),
         CONSTRAINT …
                        …);
```

Restricciones: características.

- La comprobación de restricciones puede aplazarse hasta el final de cada transacción, permitiendo violaciones temporales de las restricciones.
- Cuando una restricción es diferida, el sistema comprueba si satisface la condición en tiempo de validación (commit). Si se viola la restricción la validación causa que la transacción entera se deshaga.
- Cuando una restricción es inmediata, se comprueba para cada sentencia. Si la restricción se viola, la sentencia se deshace.
- Las restricciones pueden forzarse instantáneamente. Los usuarios pueden leer y modificar datos mientras se activa la comprobación de la restricción.
- Aunque una restricción esté activada o desactivada, las restricciones pueden activarse, desactivarse o forzarse. Una restricción forzada se comporta como si la restricción estuviera activa, pero no garantiza que todos los antiguos datos cumplan la restricción.
- Una restricción activa está comprobada y validada.
- Las claves primarias y únicas pueden implementarse con índices no únicos. Las comprobaciones se comprueban al final de cada sentencia o de cada transacción. La columna clave primaria o única debe ser la parte conductora del índice.

Comprobación de restricciones aplazada

- Cuando se crea una tabla, las restricciones pueden utilizar:

Claves que determinan si una restricción puede comprobarse en diferido o no:

  - NOT DEFERRABLE (valor por defecto)
  - DEFERRABLE

Claves que determinan el comportamiento por defecto de una restricción.

  - INITIALLY IMMEDIATE (valor por defecto)
  - INITIALLY DEFERRED

- Cuando una restricción es diferida, el sistema comprueba si satisface la condición en tiempo de validación (commit). Si se viola la restricción la validación causa que la transacción entera se deshaga. Cuando una restricción es inmediata, se comprueba para cada sentencia. Si la restricción se viola, la sentencia se deshace. Cuando se define una restricción puede crearse DEFERRABLE o NOT DEFERRABLE.

Si una restricción se ha definido como NOT DEFERRABLE, no puede cambiarse a diferida.

Si una restricción se define como DEFERRABLE, pero es INITIALLY IMMEDIATE dentro de cada transacción, se puede utilizar la nueva sentencia SET CONSTRAINT nombre_restricción/ALL DEFERRED para aplazar la comprobación de restricciones.

Si una restricción se define como DEFERRABLE, pero es INITIALLY DEFERRED, se puede utilizar el comando SET CONSTRAINT nombre_restricción IMMEDIATE para que la restricción no se compruebe aplazada.

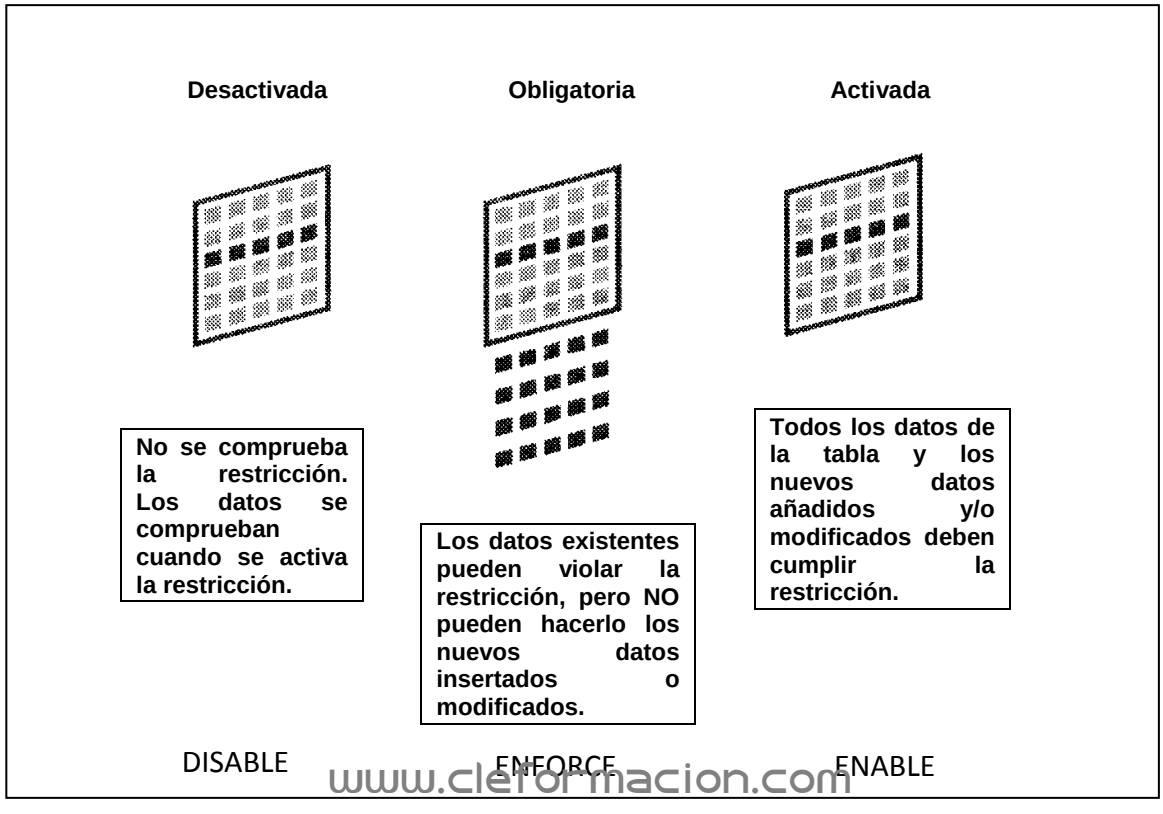
- El comando SET CONSTRAINT afecta solo a la transacción actual. Existe también el comando ALTER SESSION SET CONSTRAINTS=DEFERRED para aplicarlo a toda la sesión.
- El comportamiento por defecto de las restricciones es el siguiente:
  - INITIALLY IMMEDIATE. La comprobación de la restricción no es diferida. Significa que si se desea, hay que indicar bien SET CONSTRAINT o bien ALTER SESSION para aplazar la comprobación.
  - INITIALLY DEFERRED realiza comprobación aplazada, por lo que no se necesita ni SET CONSTRAINT ni ALTER SESSION.

```
-- COMPROBACIÓN APLAZADA DE RESTRICCIONES
CREATE TABLE cuentas(
    no_cta       NUMBER(10)   CONSTRAINT CP_NOCTA PRIMARY KEY
                              DEFERRABLE INITIALLY IMMEDIATE,
    cliente_id   NUMBER(10));
CREATE TABLE ventas(
    no_cta             NUMBER(10)   CONSTRAINT CA_NOCTA REFERENCES cuentas
                                    DEFERRABLE INITIALLY IMMEDIATE,
    persona            VARCHAR2(30),
    cantidad_vendida   NUMBER(8)    NOT NULL,
    no_semana          NUMBER(2)    NOT NULL);
```

```
ALTER SESSION SET CONSTRAINT=DEFERRED;
INSERT INTO ventas VALUES (1000,'JOSE M.',1000000,32);
INSERT INTO cuentas VALUES (1000, 2777);
COMMIT;
```

- El ejemplo muestra como aplazar la comprobación de restricciones violando una relación clave ajena/clave primaria.
- Se crean dos tablas ambas con la opción de aplazar la comprobación de restricciones en la definición de las claves ajenas e inmediatas.

Restricciones obligatorias



- Una restricción puede estar desactivada, activada, o obligatoria. Una restricción obligatoria (enforced) actúa como una activa, solo que no garantiza la integridad de la información ya existente.
- Para que una restricción sea obligatoria se debe utilizar la sentencia ALTER TABLE ENFORCE CONSTRAINT nombre_restricción.
- Una vez que la restricción esta obligatoria, se puede activar sin bloqueos. Puede activarse mientras que los usuarios están actualizando o consultando las tablas involucradas.
- Se pueden fijar restricciones obligatorias múltiples concurrentemente.
- Las restricciones obligatorias se utilizan para restricciones de unicidad con índices existentes únicos o no únicos.

Restricción UNIQUE o PRIMARY KEY utilizando un índice no único.

```
CREATE TABLE cuentas(
    no_cta       NUMBER(10),
    cliente_id   NUMBER(10),
    comentario   VARCHAR2(200),
    CONSTRAINT cp_cid_aid PRIMARY KEY(cliente_id, no_cta)DISABLE);
```

```
CREATE INDEX  I_NCTA_CLI_COMENT
ON cuentas(no_cta, cliente_id, comentario);
```

- Una clave primaria o única, puede obligarse a implementarse mediante un índice no único.
- El índice no único debe crearse con las columnas que componen la clave primaria (en cualquier orden) como la primera columna de un índice no único. Esto elimina la necesidad de índices redundantes e incrementa la velocidad al activar el índice de una clave primaria o única.
- El índice no único no tiene que seguir el orden de las columnas en la clave primaria o única. El índice utilizado para forzar una clave primaria o única no puede borrarse.
- Cuando se desactiva una restricción de unicidad o clave primaria implementada por un índice no único, el índice no se borra.
- Oracle puede crear índices no únicos para claves primarias y únicas.

Ejemplo:

- Cuando la restricción está activa no tiene creado un índice único pero puede obligar la unicidad mediante el índice no único I_NCTA_CLI_COMENT.

CREATE INDEX

Los índices se utilizan para acceder más rápidamente en lectura a un dato. Para que una columna se indexe debe cumplir ciertos requisitos:

- Ser consultada con frecuencia
- No sufrir alteraciones de operadores o funciones cuando se consulta

- Contener un volumen importante de información.
- Tener muchos valores diferentes....

Los índices pueden ser simples o compuestos de varias columnas.

Los índices pueden ser únicos o no. Si un índice se define como único, significa que no puede tener valores repetidos.

Las restricciones UNIQUE y PRIMARY KEY llevan implícita la creación del índice si no se crean desactivadas y que en ambos casos es único.

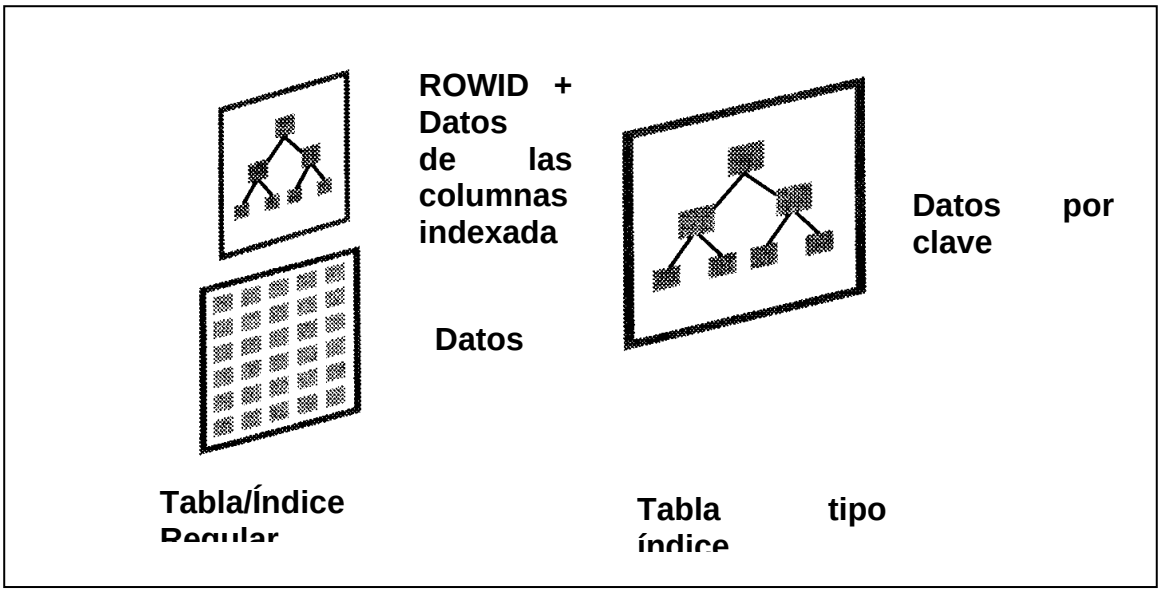
La sintaxis es la siguiente:

```
CREATE [UNIQUE] INDEX nombre_indice ON tabla (col1 [,col2])
STORAGE … (cláusula de almacenamiento)
CREATE INDEX ape_idx ON employees (last_name)
```

Las tablas del diccionario de datos con información acerca de índices accesibles para el usuario son:

- USER_INDEXES y ALL_INDEXES
- USER_IND_COLUMNS y ALL_IND_COLUMNS

Tablas tipo índice



En la mayoría de las tablas en Oracle:

- Los datos se almacenan en bloques de datos, sin orden.
- Los índices se mantienen como estructuras en árbol B.

- Las entradas del índice contienen.
  - El valor indexado.
  - El ROWID de las filas que contienen el valor.

- Esta organización es muy eficiente con la mayor parte de los datos, pero tiene algunas desventajas.
  - Oracle debe leer primero el bloque de índices y después los datos.
  - Los valores claves se almacenan en ambos sitios (tabla e índice). Esto es especialmente costoso en sistemas con gran cantidad de índices concatenados, donde los valores clave son grandes.
- Las aplicaciones que utilizan datos complejos tienen diferentes requerimientos de índices.
- Empleando tablas tipo índice:
  - Guarda todos los bloques en una estructura en árbol B. No hay bloques de datos separados, sólo de índices.
  - Este tipo de tablas se utiliza en aplicaciones que acceden y manipulan datos complejos. Aplicaciones informativas, aplicaciones espaciales, y aplicaciones que realizan procesos analíticos.

Sintaxis:

```
CREATE TABLE esquema.nombre_tabla (
COL1          tipo_Dato_col1 (longitud) [restricción_int],
COL2          tipo_Dato_col2 (longitud) [restricción_int],
COL3          tipo_Dato_col3 (longitud) [restricción_int],
…
COLn          tipo_Dato_coln (longitud) DEFAULT [restricción_int]
[, declaración opcional de mas restricciones de integridad]       )
[ORGANIZATION HEAP|INDEX|EXTERNAL]
[ PARAMETROS DE ALMACENAMIENTO ]
[ AS subconsulta]
```

- Para crear una tabla tipo índice solamente es necesario añadir la clave ORGANIZATION INDEX a la sentencia CREATE TABLE. La opción por defecto es ORGANIZATION HEAP. (Apilada).
- La tabla está indexada, y después los datos se ordenan por la columna que se haya definido como clave primaria.

Nota: No es la sintaxis completa de CREATE TABLE. La opción EXTERNAL hace referencia a la creación de una tabla de solo lectura ubicada fuera de la base de datos.

Ejemplo:

```
CREATE TABLE docindice (
llamada          CHAR(20),
doc_oid          NUMBER,
numero_llamada   NUMBER,
fecha_llamada    VARCHAR2(512),
CONSTRAINT pk_docindice PRIMARY KEY (llamada, doc_oid))
ORGANIZATION INDEX;
```

Las tablas tipo índice tienen una serie de restricciones:

- Deben tener una clave primaria. Es el identificador único que se utiliza como base para la ordenación. No hay ROWID para actuar como identificador único.

- Esta característica significa que no se puede actualmente crear otros índices sobre la tabla, o utilizar restricciones de unicidad, que también se implementan con índices.
- Las tablas tipo índice no pueden incluir columnas tipo LONG, pero SI pueden contener LOBs.
- Las aplicaciones NO pueden recuperar o manipular vía ROWID, porque no hay ROWID.

NOTA: Existen otras restricciones sobre estas tablas que afectan al administrador de B.D. al realizar un ajuste de la misma, o que pertenecen a otras opciones de compra adicionales al Gestor Oracle.

Creación de varios índices sobre la misma columna en una tabla

Antes de la versión Oracle12c, no se podía crear varios índices en la misma columna o conjunto de columnas (índice múltiple).

Por ejemplo, si tenías un índice en la columna {a} o en las columnas {a, b}, no se podía crear otro índice en la misma columna o columnas. Oracle mostraba el siguiente mensaje:

```
ORA-01408: esta lista de columnas ya está indexada
```

En Oracle12c, puedes tener varios índices en la misma columna, siempre y cuando el tipo del índice sea diferente. Pero sólo uno de estos índices se puede utilizar en un momento dado. Debes establecer invisibles el resto de índices para la misma columna.

Ejemplo:

```
CREATE INDEX emp_ind1 ON empleados(empno,ename);

CREATE BITMAP INDEX emp_ind2 ON empleados(empno,ename) INVISIBLE;
```

Si durante el trabajo estableces VISIBLE alguno de los índices invisibles de la misma columna aparece el siguiente error Oracle:

```
ORA-14147: Hay un índice VISIBLE existente definido en el mismo juego de cols.
```

Recuerda que con el fin de poder utilizar los índices que se encuentran invisibles, es necesario establecer el parámetro **OPTIMIZER_USE_INVISIBLE_INDEXES** a TRUE.

```
ALTER SESSION SET optimizer_use_invisible_indexes = true;
```

CREATE SYNONYM

Un sinónimo es un nombre alternativo a una tabla, vista o procedimiento almacenado. A diferencia del alias, el sinónimo no es temporal y permanece hasta que es borrado.

Independientemente de la creación del sinónimo es necesario tener acceso al objeto sobre el que se desea crear dicho sinónimo, o bien ser su propietario.

Existen dos tipos de sinónimos: públicos y privados. Los públicos están accesibles por cualquier usuario de la Base de Datos y para crearlos se necesita el privilegio CREATE PUBLIC

SYNONYM. Los privados son accesibles sólo para aquel usuario que lo crea y se necesita el privilegio CREATE SYNONYM.

La sintaxis es la siguiente:

```
CREATE [PUBLIC] SYNONYM sinónimo FOR [esquema.]objeto
    CREATE SYNONYM miemp FOR pepe.employees
```

Las tablas del diccionario de datos con información acerca de índices accesibles para el usuario son USER_SYNONYMS y ALL_SYNONYMS

CREATE VIEW

Una vista es una tabla lógica basada en una o varias tablas. La tabla no se crea físicamente sino que lo que se almacena es la SELECT de creación de la vista.

Cuando se recuperan filas de una vista, ocurre que se accede a la sentencia SELECT que la compone y que se encuentra almacenada, y se ejecuta.

La sintaxis es la siguiente:

```
CREATE [OR REPLACE] [FORCE|NOFORCE] VIEW nombre_vista
[(col1, coln.....)]
AS subconsulta
[ WITH CHECK OPTION | WITH READ ONLY ]
```

- OR REPLACE - Recrea la vista si ya existe. Esto permite cambiar la definición de la vista sin tener que borrarla y volver a crearla.
- FORCE - Crea la vista incluso si hay problemas de acceso por insuficientes privilegios a los objetos de la subconsulta en que se basa la creación de la vista.
- NOFORCE - Es la opción por defecto y no crea la vista si se producen errores en la definición.
- WITH CHECK OPTION - Si la vista permite inserciones, comprobará la condición WHERE impuesta, no solo en el momento de componer la vista, sino también a la hora de realizar inserciones sobre ella.
-

A continuación se muestra un ejemplo:

```
CREATE VIEW sueldos AS SELECT employee_id, salary
FROM employees
WHERE department_id between 10 and 60
WITH CHECK OPTION;
```

Sobre una vista no existen restricciones en consulta, pero sí en actualización inserción y borrado. Si se cumplen las condiciones impuestas la actualización, inserción o borrado se lleva a cabo sobre la tabla base sobre la que se definió la vista.

Para que una vista sea actualizable (permita UPDATE), NINGUNA de las columnas que forman la vista puede estar modificada por una expresión. P.e. Substr(first_name, 1,3).

Para que la vista permita inserciones (INSERT INTO vista), la vista en su definición debe contener TODAS las columnas obligatorias de la tabla que la forma.

Para que la vista permita borrado de filas (DELETE from vista), la vista debe estar creada sobre UNA sola tabla (no admite join). No puede incorporar cláusulas DISTINCT ni GROUP BY. No puede ser definida con funciones de grupo ni pseudocolumnas (SUM(sal) o rowid).

Las tablas del diccionario de datos con información acerca de índices accesibles para el usuario son USER_VIEWS y ALL_VIEWS

CREATE SEQUENCE

Una secuencia es un objeto que permite generar números secuenciales enteros y únicos. Esto puede ser muy útil, por ejemplo, para conseguir claves primarias de forma automática. Además y dado que es un objeto como otro cualquiera de la Base de Datos, puede ser utilizado por múltiples usuarios. Para poder crear una secuencia, se debe poseer un privilegio denominado CREATE SEQUENCE.

La sintaxis es la siguiente:

```
CREATE SEQUENCE [esquema.]nombre_de_secuencia
          START WITH entero
          INCREMENTE BY entero
          MAXVALUE entero
          NOMAXVALUE
          MINVALUE entero
          NOMINVALUE
          CYCLE
          NOCYCLE
          CACHE | NOCACHE ;
```

- START WITH - Determina el primer número secuencial que será generado.
- INCREMENT BY - Determina el salto entre los números secuenciales. Puede ser un entero negativo o positivo. De ese modo, hacemos secuencias ascendentes o descendentes. No puede ser 0.
- MINVALUE Determina el valor mínimo de la secuencia.
- NOMINVALUE Valor de 1 para las secuencias ascendentes y -(1026) para las descendentes.
- MAXVALUE - Valor máximo que genera la secuencia.
- NOMAXVALUE - Valor de 1027 para las ascendentes y -1 para las descendentes.
- CYCLE- La secuencia entra en un ciclo cuando alcanza su valor máximo o mínimo.
- NOCYCLE - Si se alcanza el valor máximo o mínimo, no se pueden generar más números.
- CACHE - Almacena o no un número determinado de valores en memoria cache

Por ejemplo, para crear una secuencia llamada seq1 que comience por 100 y vaya generando número de cinco en cinco se usa la orden:

```
CREATE SEQUENCE seq1 START WITH 100 INCREMENT BY 5;
```

Una vez que se ha creado la secuencia puede ser accedida utilizando dos pseudocolumnas:

- CURRVAL devuelve el valor actual de la consulta

- NEXTVAL incrementa el valor de la secuencia

La siguiente orden incrementa la secuencia y devuelve el valor actual. CURRVAL devuelve el valor de la secuencia, pero sin incrementar la misma.

```
SELECT seq1.nextval FROM DUAL;
```

Ambas funciones pueden ser utilizadas en:

- Una consulta SELECT que no lleve DISTINCT, ni grupos, ni sea parte de una vista, ni sea subconsulta de otro SELECT, UPDATE o DELETE
- Una subconsulta SELECT en una instrucción INSERT
- La cláusula VALUES de la instrucción INSERT
- La cláusula SET de la instrucción UPDATE

No se puede utilizar como valor para la cláusula DEFAULT de un campo de tabla.

Evidentemente, puede ser más útil en órdenes de tipo INSERT, como por ejemplo:

```
INSERT INTO expedientes (codigo) VALUES (ejemplo.nextval);
```

Las tablas del diccionario de datos con información acerca de índices accesibles para el usuario son USER_SEQUENCES y ALL_SEQUENCES

Campos cuyo valor DEFAULT referencia a una secuencia

En las versiones anteriores de Oracle si se deseaba que un campo tuviera los valores que generaba una secuencia; se realizaba mediante *triggers* que se dispararían antes de la operación INSERT y obtendrían el NEXTVAL de la secuencia.

En esta nueva versión, Oracle12c nos permite de manera sencilla indicar que el valor por defecto (DEFAULT) de un campo será el NEXTVAL de una secuencia sin la necesidad de crear ningún *trigger*.

Ejemplo:

1. En el ejemplo se crea una secuencia con su configuración por defecto (se inicia en 1 e incrementa de 1 en 1).

   ```
   CREATE SEQUENCE seq_contador;
   ```

2. Crea una tabla llamada **producto** cuyo campo **contador** estará asociado al NEXTVAL de la secuencia **seq_contador** en caso no se le especifique en la operación de INSERT un valor.

   ```
   CREATE TABLE producto (cod NUMBER,
                          contador NUMBER DEFAULT seq_contador.NEXTVAL);
   ```

3. Comprueba los valores que tienen por defecto los campos de la tabla. Consulta la vista del diccionario USER_TAB_COLS.

   ```
   SQL> COLUMN column_name FORMAT a20
   SQL> COLUMN data_default FORMAT a40
   ```

```
SQL> SELECT column_name,DATA_DEFAULT FROM user_tab_cols
  2  WHERE table_name='PRODUCTO';

COLUMN_NAME          DATA_DEFAULT
-------------------- ----------------------------------------
COD
CONTADOR             "SCOTT"."SEQ_CONTADOR"."NEXTVAL"
```

4. Introduce un registro en la tabla sin especificar un valor para en el campo contador y comprueba que por defecto ha sido poblado con el valor NEXTVAL de la secuencia.

```
SQL> INSERT INTO producto(cod) VALUES(10);

SQL> INSERT INTO producto VALUES(20,DEFAULT);

SQL> COMMIT;

SQL> SELECT * FROM producto;

       COD   CONTADOR
---------- ----------
        10          1
        20          2
```

Secuencias a nivel de sesión

En la base de datos Oracle 12c existe una nueva funcionalidad que se implementa para las secuencias. Esta novedad permite crear un objeto *sequence* cuyos valores se encuentran a nivel de sesión.

Las secuencias a nivel sesión generan una serie única de valores que están limitados dentro de cada sesión, no a través de varias sesiones. Una vez que la sesión finaliza, el estado de las secuencias de sesión se elimina.

Sintaxis:

```
CREATE SEQUENCE nombre_secuencia [opciones] SESSION;
```

Observaciones:

- Las cláusulas CACHE, NOCACHE, ORDER y NOORDER son ignoradas en las secuencias creadas a nivel de sesión.
- Estos tipos de secuencias son más útiles y adecuados sobre las tablas temporales ya que los datos de éstas tienen su duración a nivel de sesión.

Ejemplos:

5. El siguiente ejemplo muestra la creación de una secuencia a nivel de sesión que comienza en 1 y se incrementa de 1 en 1.

```
CREATE SEQUENCE seq1 START WITH 1 INCREMENT BY 1 SESSION;
```

6. Utiliza la secuencia anterior en una tabla temporal.

```
CREATE GLOBAL TEMPORARY TABLE tabla_temporal ( id NUMBER, seq NUMBER )
ON COMMIT PRESERVE ROWS;
```

```
INSERT INTO tabla_temporal VALUES (1, seq1.NEXTVAL);
INSERT INTO tabla_temporal VALUES (2, seq1.NEXTVAL);
```

La definición del tipo de secuencia se encuentra en el campo **SESSION_FLAG** de la vista del diccionario de datos **USER/ALL/DBA_SEQUENCES**.

```
SQL> DESCRIBE user_sequences
 Nombre                                    ¿Nulo?   Tipo
 ----------------------------------------- -------- ----------------------------
 SEQUENCE_NAME                             NOT NULL VARCHAR2(128)
 MIN_VALUE                                          NUMBER
 ...
 SESSION_FLAG                                       VARCHAR2(1)
 ...
```

```
SQL> COLUMN sequence_name FORMAT a20
SQL> COLUMN session_flag FORMAT a20;
SQL> SELECT sequence_name,session_flag FROM user_sequences;

SEQUENCE_NAME        SESSION_FLAG
-------------------- --------------------
SEQ1                 Y
```

Se puede cambiar el nivel de utilización de la secuencia mediante el comando:

```
ALTER SEQUENCE nombre_secuencia { GLOBAL | SESSION };
```

Ejemplo:

7. Cambia el alcance de la secuencia a nivel global.

```
SQL> ALTER SEQUENCE seq1 GLOBAL;

SQL> SELECT sequence_name,session_flag FROM user_sequences;

SEQUENCE_NAME        SESSION_FLAG
-------------------- --------------------
SEQ1                 N
```

Campos autonuméricos generados a través de IDENTITY

Esta opción nos permite crear un campo que tenga un valor autonumérico generado para cada fila introducida; en versiones previas a Oracle12c se implementaba a través de *triggers* y secuencias.

Para su creación se utiliza la siguiente sintaxis:

```
nombre_columna tipo_dato
  GENERATED [ALWAYS | BY DEFAULT [ON NULL]] AS IDENTITY [(opciones)]
```

Este formato incorpora los siguientes parámetros:

- *nombre_columna*: Es el nombre de la columna autonumérica.
- *tipo_dato*: Es obligatorio y se utiliza para definir el tipo de dato de la columna.

- **GENERATED AS IDENTITY**: Identifica que esa columna va a utilizar valores autogenerados. Solo se permite una columna por tabla que tenga configurada la opción IDENTITY.
- **ALWAYS**: Esta cláusula permite identificar que los números se generan siempre. No debes introducir valor sobre el campo autogenerado. En caso contrario se producirá el siguiente error:

```
ORA-32795: no se puede insertar una columna de identidad siempre generada
```

- **BY DEFAULT [ON NULL]**: Permite la entrada de valores sobre la columna autogenerada. No pudiendo introducir valores nulos en la columna. Si se intenta la entrada de un valor nulo se presenta el siguiente mensaje:

```
ORA-01400: no se puede realizar una inserción NULL en ("SCOTT"."PERSONA".)
```

- Si se añade la cláusula **ON NULL** si se permite introducir valores nulos en el campo.
- *opciones*: Identifica las características de la serie numérica que se va a generar (sigue la nomenclatura de las secuencias).

Ejemplos:

Crea una tabla en la que exista un campo autonumérico (IDENTITY) con las características por defecto.

```
SQL> CREATE TABLE persona (
  2    cod NUMBER GENERATED AS IDENTITY,
  3    nombre VARCHAR2(30),
  4    edad NUMBER(3) );
```

Comprueba que internamente Oracle ha creado una secuencia, consultando la vista del diccionario USER_SEQUENCES:

```
SQL> SELECT sequence_name,increment_by FROM user_sequences;

SEQUENCE_NAME        INCREMENT_BY
-------------------- ------------
ISEQ$$_92544                    1
```

Introduce un registro y comprueba los valores de los campos.

```
SQL> INSERT INTO persona (nombre,edad) VALUES ('Rosa',45);
SQL> COMMIT;

SQL> SELECT * FROM persona;

       COD NOMBRE                                EDAD
---------- ------------------------------ ----------
         1 Rosa                                   45
```

Podemos también indicar el número con que comenzará el valor autogenerado y como será su incremento En el siguiente ejemplo se define que el primer valor generado será 10 y que se incrementará sucesivamente de 10 en 10 hasta llegar al

valor de 1000. Después de este valor ya no se podrán realizar más operaciones de generación de números.

```
SQL> CREATE TABLE persona (
  2    cod NUMBER GENERATED AS IDENTITY
  3              (START WITH 10 INCREMENT BY 10 MAXVALUE 1000 NOCYCLE),
  4    nombre VARCHAR2(30),
  5    edad NUMBER(3)
```

ALTER TABLE

El comando ALTER TABLE permite modificar la estructura de una tabla para:

- Añadir columnas a una tabla ya creada
- Modificar el tipo de dato o la precisión de una columna.
- Añadir, activar o desactivar temporalmente y borrar restricciones de integridad referencial sobre la tabla.
- Borrar columnas de una tabla.

La sintaxis es la siguiente:

```
ALTER TABLE [esquema.]nombre_tabla
ADD | MODIFY | DROP {CONSTRAINT | COLUMN | (columna1[…,
columnan])INVISIBLE|VISIBLE}|
{DISABLE | ENABLE} CONSTRAINT | RENAME COLUMN | …
```

Mediante el siguiente ejemplo, es posible añadir una columna a la tabla de departamentos que contenga el número de empleados que trabaja en un departamento.

```
ALTER TABLE employees
ADD (no_empleados NUMBER(4));
```

Modificar la longitud de la columna “ename” en “emp” de 10 posiciones a 14. Añadir la característica de obligatoriedad a esta misma columna.

```
ALTER TABLE emp
MODIFY (ename VARCHAR2(14) NOT NULL);
```

Para realizar esta operación debo asegurarme de que la columna ename no tiene valores nulos. Si lo estuviera el comando ALTER TABLE devuelve un error.

Eliminar una restricción de unicidad llamada UQ_DNAME sobre la columna DNAME mediante dos variaciones del comando ALTER TABLE.

```
ALTER TABLE dept
DROP CONSTRAINT uq_dname;
    ó
    ALTER TABLE depat
DROP CONSTRAINT UNIQUE(dname);
```

Añadir Restricciones

Se pueden añadir restricciones de integridad referencial a posteriori. En el momento que se añade, se bloquea toda la tabla y se produce la comprobación de la restricción que se desea incorporar. Si alguna fila no la cumple, la restricción no se añade.

Mediante el siguiente ejemplo añadir una restricción a la tabla DEPT. La situación de los departamentos (LOC) debe ser única.

```
ALTER TABLE departments
ADD CONSTRAINT uq_dept_loc UNIQUE (location_id);
```

La sintaxis para añadir restricciones una vez creada la tabla es similar a la de las restricciones "No en línea". Esto significa que se debe de indicar tras el comando ALTER TABLE la palabra clave CONSTRAINT seguida de un nombre (obligatorio) luego el tipo de restricción a aplicar, y por último sobre qué columna de la tabla se desea aplicar dicha restricción.

Añadir una restricción en la tabla línea de facturas. Cada vez que se borre una cabecera de factura se borran las líneas asociadas en cascada. Nótese que la restricción no se

Aplica en la tabla maestra (en este caso la de cabeceras de facturas) sino en la DETALLE (en este caso la de líneas)

```
ALTER TABLE linea_fac
ADD CONSTRAINT ca_li_fact_idfact FOREIGN KEY (idfact)
REFERENCES facturas ON DELETE CASCADE;
```

Cuando se borre una factura, (DELETE from facturas where idfactura=2344) se visualiza el mensaje "N rows deleted" que se refiere a las cabeceras, aunque se hayan borrado una cabecera y sus siete líneas.

Cuando se define una clave ajena se puede hacer referencia explícita a la tabla referenciada REFERENCES tabla (col), o implícita REFERENCES tabla. En el caso del borrado en cascada la sintaxis sólo permite referencia implícita.

Desactivar Restricciones

El desactivar una restricción permite que dicha restricción no se compruebe temporalmente. NO ES IGUAL que borrar la restricción.

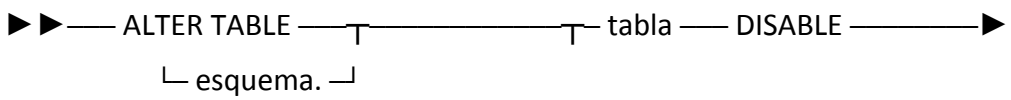
Es la forma de incrementar la velocidad en cargas de datos masivas.

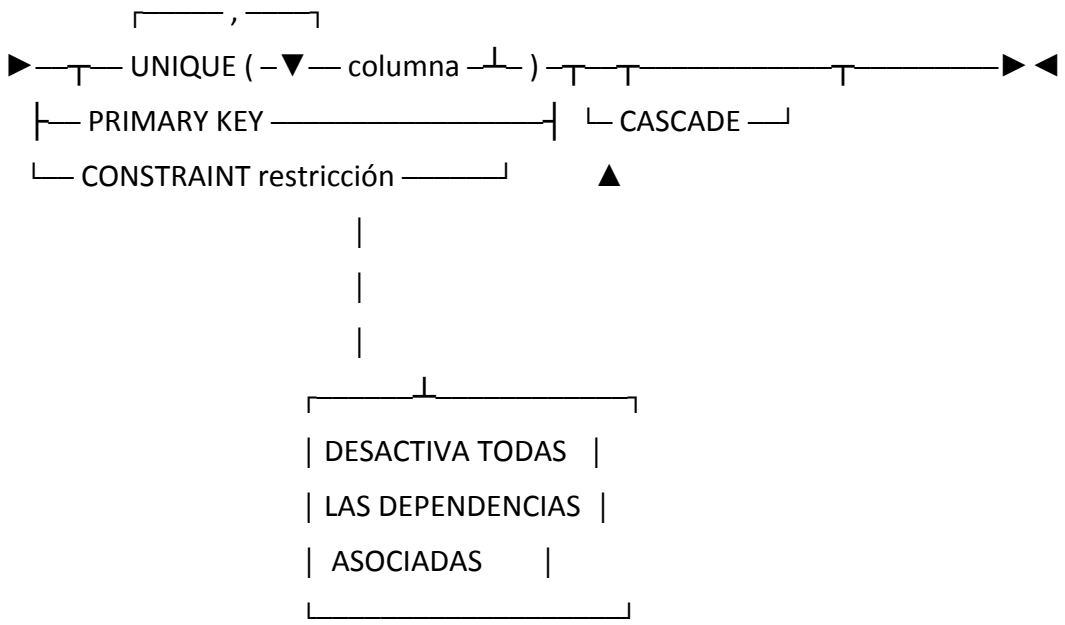
Loader (la herramienta de carga ORACLE), deshabilita automáticamente con su método directo «direct path».

Cuando se intenta desactivar una clave ajena o clave única (que pueden ser referenciadas), primero habrá que desactivar las claves ajenas que le referencian o utilizar DISABLE CASCADE.

Al desactivar una clave primaria o única también se destruyen los índices asociados.

La sintaxis es la siguiente:





Activar Restricciones

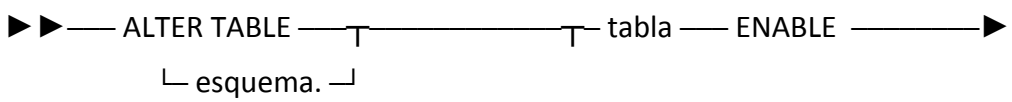
Activa la comprobación de la restricción. En el momento de la creación de una restricción se activa a no ser que se indique la cláusula DISABLE.

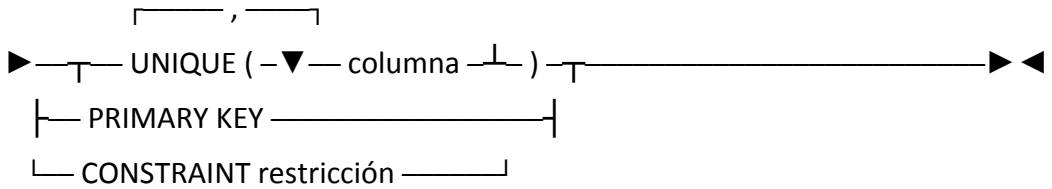
Cuando se activa posteriormente:

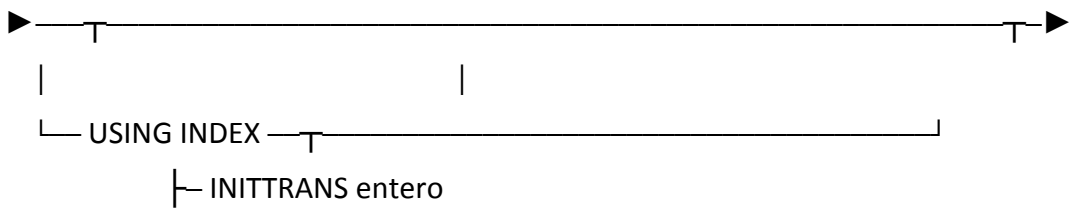
- Bloquea la tabla completa hasta que se termine la comprobación.
- Comprueba una a una las filas.
- Si alguna de las filas no cumple la restricción, no podrá llegar a activarse.

Al habilitar una clave primaria o única se vuelven a construir los índices asociados.

La sintaxis es la siguiente:







├─ MAXTRANS entero

├─ TABLESPACE tablespace

├─ STORAGE almacenamiento

├─ PCTFREE entero

└─ PCTUSED entero

Excepciones Activando Restricciones

Cuando se intenta activar una restricción y no se puede, por cada fila que no cumpla la restricción se puede guardar en una tabla creada previamente la siguiente información:

- Rowid.
- Nombre de restricción que no se cumple.
- Nombre de la tabla en la que se encuentra la restricción.
- Usuario propietario de la tabla.

La tabla EXCEPTIONS se ha creado previamente en SQL*Plus utilizando un fichero que se encuentra en la instalación y se llama UTLXCPT.SQL y cuya estructura es:

```
CREATE TABLE exceptions (
row_id     rowid,
owner            varchar2(30),
table_name       varchar2(30),
constraint       varchar2(30) );
```

En el siguiente ejemplo se ha desactivado la clave primaria que existía sobre la tabla préstamos. Después se ha introducido 1 fila duplicada. A continuación se intenta activar de nuevo la restricción:

```
ALTER TABLE  prestamos
    ENABLE CONSTRAINT cp_prest
    EXCEPTIONS  INTO exceptions;

SELECT * FROM exceptions WHERE CONSTRAINT = 'CP_PREST';
ROWID                  OWNER  TABLE_NAME   CONSTRAINT
------------------     -----  ----------   ----------
AAAAeDAACAAAAERAAA     PEPE   PRESTAMOS    CP_PREST
AAAAeDAACAAAAERAAF     PEPE   PRESTAMOS    CP_PREST
```

La tabla sobre la cual se han definido las restricciones y no pueden activarse, puede actualizarse a partir de los resultados almacenados en la tabla EXCEPTIONS.

Mediante el siguiente ejemplo, es posible averiguar qué filas están duplicadas en la tabla borrando todas las filas duplicadas de la tabla préstamos.

```
SELECT    p.rowid, p.nocuenta, p.noprestamo
FROM  prestamos p, exceptions WHERE p.rowid = exceptions.row_id;
ROWID                   NOCUENTA      NOPRESTAMO
------------------      --------      ----------
AAAAeDAACAAAAERAAA      7499          2495
AAAAeDAACAAAAERAAF      7499          2244
```

A continuación debemos borrar los duplicados de la tabla préstamo. Si las dos filas son EXACTAMENTE iguales, la única forma de eliminar duplicados es utilizar el ROWID de la fila en la tabla.

```
DELETE FROM prestamos WHERE noprestamo=2244; o
DELETE FROM prestamos WHERE rowid='AAAAeDAACAAAAERAAF';
```

Insertar un valor admisible en campos con valor desconocido. Al crear la tabla la restricción se ha creado desactivada:

```
CREATE TABLE prestamos (
…
CONSTRAINT ck_pres_tipo CHECK (
tipopres IN ('PERS', 'CASA', 'NOVALIDO') DISABLE,
…);
```

A continuación se insertan datos en la tabla, y después se activa la restricción utilizando:

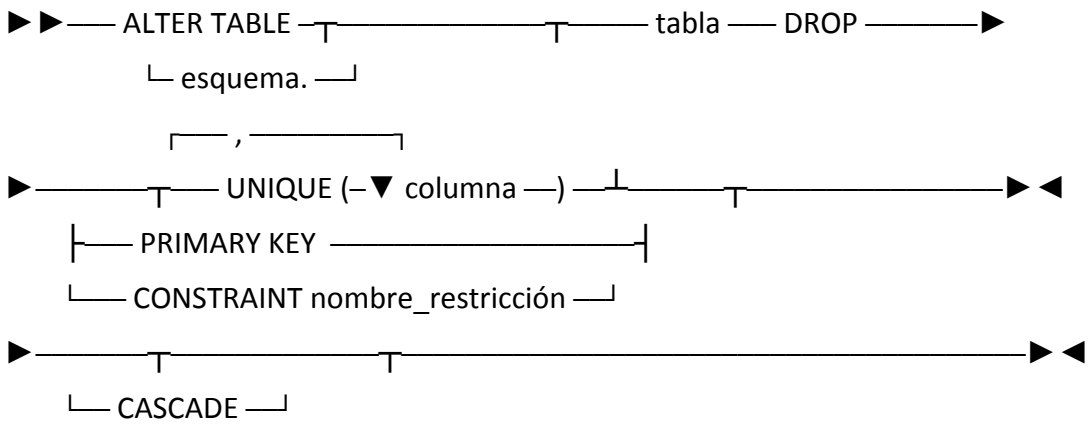
```
ALTER TABLE prestamos ENABLE CONSTRAINT ck_pres_tipo
EXCEPTIONS INTO exceptions;
```

Después se depura la información actualizando los datos de la tabla de la siguiente manera:

```
UPDATE prestamos SET tipopres='NOVALIDO'
WHERE prestamos.rowid IN (SELECT row_id FROM exceptions
                              WHERE constraints='CK_PRES_TIPO';
```

Borrar Restricciones

La sintaxis es la siguiente:



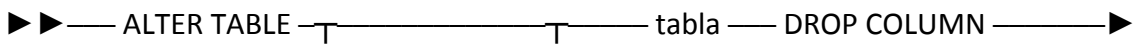
Véase el siguiente ejemplo:

```
ALTER TABLE prestamos DROP CONSTRAINT ca_pres_aprobadopor;
ALTER TABLE prestamos DROP PRIMARY KEY CASCADE;
```

La utilización de la opción CASCADE al borrar una restricción, permite borrar dicha restricción y además elimina todas las claves ajenas que apuntaran a la clave.

Borrar Columnas

La sintaxis es la siguiente:



└ esquema. ┘

Borra una columna de una tabla. La columna no puede contener CONSTRAINT Primary Key o Foreign Key.

```
ALTER TABLE prestamos DROP COLUMN sal;
Para borrar varias columnas la sintaxis es:
ALTER TABLE emp DROP (hire_date,job);
```

Columnas invisibles

En Oracle12c R1, se puede tener una columna no visible en una tabla. Cuando una columna se define como invisible, la columna no aparecerá en las consultas genéricas, a menos que el campo se mencione expresamente en la sentencia SQL.

Es posible ocultar una columna y luego volverla a presentar con la finalidad de asegurarnos que un campo no estará disponible a los usuarios durante el tiempo que creamos oportuno. También lo podemos utilizar para desplegar un nuevo aplicativo que explícitamente consultará las nuevas columnas invisibles creadas en la tabla en pro de no generar interferencia en el servicio a las aplicaciones ya existentes.

Cualquier columna de la tabla se puede establecer **INVISIBLE**, esto implica que el campo no será usado de manera habitual, las operaciones que no consideran la columna invisible son las siguientes:

- La sentencia SELECT * FROM.
- El comando DESCRIBE en SQL*Plus.
- El tipo de datos %ROWTYPE en una declaración de PL/SQL.
- La sentencia INSERT INTO SELECT *.

Es bastante fácil establecer o modificar una columna para que sea visible o invisible. Durante la creación de la tabla se puede establecer que columna es invisible, mediante el siguiente formato:

```
CREATE TABLE nombre_tabla (
  columna1 tipo_datos,
  columna2 tipo_datos INVISIBLE,
  ...
);
```

Una columna que se deje como INVISIBLE se puede establecer como visible nuevamente mediante el comando ALTER TABLE. Su formato es el siguiente.

```
ALTER TABLE nombre_tabla MODIFY nombre_columna { INVISIBLE|VISIBLE } ;
```

Ejemplos:

8. Creación de una tabla que incorpora una columna invisible. Comprueba la estructura de la tabla mediante el comando DESCRIBE.

```
SQL> CREATE TABLE tabla_col_ocultas(
  2    campo1 NUMBER,
  3    campo2 VARCHAR2(30) INVISIBLE,
  4    campo3 DATE
  5   );
```

```
Tabla creada.

SQL> DESCRIBE tabla_col_ocultas
 Nombre                            ¿Nulo?   Tipo
 --------------------------------- -------- ------------
 CAMPO1                                     NUMBER
 CAMPO3                                     DATE
```

9. Si una columna se encuentra en estado INVISBLE puede ser consultada si se coloca explícitamente el nombre de la columna en la sentencia SELECT.

```
SELECT campo2 FROM tabla_col_ocultas;
```

10. A continuación se presenta un ejemplo donde la columna *campo2* de la tabla *tab_col_ocultas* se colocará en estado INVISIBLE, se realizarán operaciones DML sobre la tabla y por último se regresará al estado visible donde veremos la información modificada.

```
ALTER TABLE tabla_col_ocultas MODIFY campo2 INVISIBLE;

INSERT INTO tabla_col_ocultas VALUES (1,SYSDATE);

COMMIT;

SELECT * FROM tabla_col_ocultas;

UPDATE tabla_col_ocultas SET campo2='Rosa Alvarez' WHERE campo1=1;

COMMIT;

SELECT * FROM tabla_col_ocultas;

ALTER TABLE tabla_col_ocultas MODIFY campo2 VISIBLE;

SELECT * FROM tabla_col_ocultas;

    CAMPO1 CAMPO3   CAMPO2
---------- -------- ------------------------------
         1 20/02/14 Rosa Alvarez
```

Observaciones:

- Esta funcionalidad no aplica a tablas externas, temporales, cluster y campos cuyo tipo de dato fue definido por el usuario.
- Cuando una columna es establecida VISIBLE cuando anteriormente no lo era, automáticamente se coloca en la última posición de las columnas de la tabla.

```
SQL> DESCRIBE tabla_col_ocultas
 Nombre                            ¿Nulo?   Tipo
 --------------------------------- -------- ------------
 CAMPO1                                     NUMBER
 CAMPO3                                     DATE
 CAMPO2                                     VARCHAR2(30)
```

- A través de SQL*Plus se puede configurar el hecho de que aparezca una columna invisible en las operaciones que implícitamente no sea especificada, para ello utiliza el parámetro **COLINVISIBLE**.

```
SET COLINVI[SIBLE] ON | OFF
```

ALTER SEQUENCE

El comando ALTER SEQUENCE permite modificar las características y el comportamiento de una secuencia. Se puede modificar:

- El incremento del número secuencial.
- Establecer los valores máximos y mínimos.

La secuencia debe pertenecer al usuario o bien este debe poseer el privilegio ALTER sobre la secuencia. La sintaxis es muy parecida a la de creación:

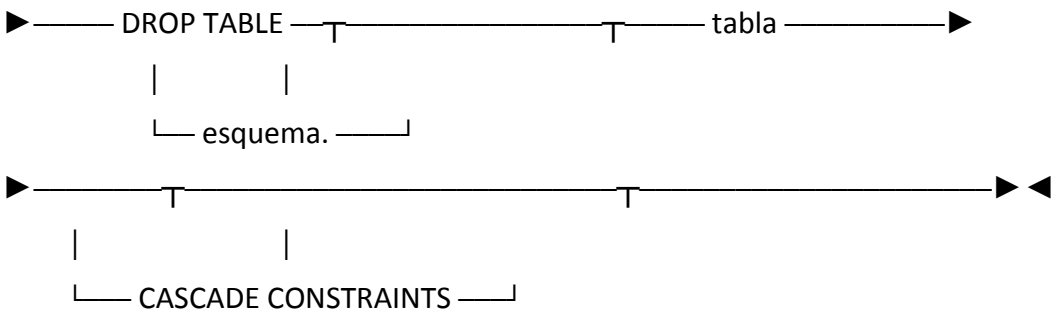
La sintaxis es la siguiente:

```
ALTER SEQUENCE [esquema.]Nombre_de_secuencia
          INCREMENT BY entero
          MAXVALUE entero
          NOMAXVALUE entero
          MINVALUE entero
          NOMINVALUE
          CYCLE
          NOCYCLE;
```

El significado de cada uno de los valores está definido en el comando CREATE SEQUENCE.

DROP TABLE

Permite borrar una tabla. La sintaxis es la siguiente:



Mediante el siguiente ejemplo, es posible eliminar la tabla PRODUCTOS con todas sus filas y las claves ajenas que apunten a cualquier columna de PRODUCTOS.

```
DROP TABLE productos CASCADE CONSTRAINTS;
```

DROP INDEX

Se puede eliminar un índice de una tabla utilizando el siguiente comando:

```
DROP INDEX nombre_indice;
```

A partir de este momento la columna en cuestión ya NO está indexada. En el comando DROP INDEX es indiferente que el índice sea único o no.

La información acerca de los índices está almacenada en el diccionario en las tablas USER_INDEXES y USER_IND_COLUMNS

Para borrar un índice se debe ser el propietario de la tabla en la que están los índices o tener un privilegio llamado DROP ANY INDEX.

DROP VIEW

Se puede eliminar un índice de una tabla utilizando el siguiente comando:

```
DROP VIEW nombre_vista;
```

DROP SYNONYM

Se puede borrar un sinónimo sobre una tabla, procedimiento, vista, etc. utilizando el siguiente comando.

```
DROP [PUBLIC] SYNONYM nombre_sinónimo;
```

Con la utilización de este comando no se borra el objeto, sino solamente el alias permanente del objeto.

La información referente a sinónimos se encuentra en las siguientes tablas: USER_SYNONYMS y ALL_SYNONYMS

Para borrar un sinónimo basta con ser el propietario, con haberlo creado. En el caso de que el sinónimo que se desea borrar sea público, es necesario tener el privilegio DROP PUBLIC SYNONYM.

DROP SEQUENCE

El comando DROP SEQUENCE sirve para eliminar una secuencia de la BD. El comando a utilizar es:

```
DROP SEQUENCE [Esquema.]Nombre_de_Secuencia
```

Suele ser el método utilizado para resetear una secuencia. Se borra y luego se vuelve a crear con los valores deseados. Véase el siguiente ejemplo:

```
DROP SEQUENCE seq1;
```

RENAME

El comando RENAME cambia el nombre de un objeto (tabla, procedimiento, vista, etc.) por otro de forma permanente, de la siguiente forma:

```
RENAME nombre_antiguo_objeto TO nombre_nuevo_objeto;
```

La información sobre los objetos de un usuario se encuentra en el diccionario en la vista USER_OBJECTS. Para renombrar un objeto se debe ser propietario del mismo.

TRUNCATE

El comando TRUNCATE borra filas de una tabla o índice sin eliminar la estructura del objeto. Es similar a DELETE, pero no hay posibilidad de deshacer la transacción (ROLLBACK), ni de hacer un borrado restrictivo (cláusula WHERE).

```
TRUNCATE TABLE nombre_tabla
[DROP STORAGE] [REUSE STORAGE]
```

- Para truncar un objeto, se debe ser propietario del mismo o tener el privilegio DELETE ANY TABLE.
- La opción DROP STORAGE libera el espacio que el objeto ha tomado a la base de datos, opción por defecto.
- La opción REUSE STORAGE mantiene reservado el espacio previamente adquirido para dicho objeto.
- Al truncar una tabla se truncan de forma implícita los índices de dicha tabla

El comando TRUNCATE TABLE con la opción CASCADE en Oracle12c trunca los registros en la tabla maestra y automáticamente inicia también una eliminación recursiva en las tablas relacionadas.

```
TRUNCATE TABLE nombre_tabla CASCADE;
```

Observaciones:

- El campo clave externa de la tabla relacionada (REFERENCES) tiene que estar definido con la cláusula ON DELETE CASCADE. Si no es así cuando se utilice el comando TRUNCATE aparecerá el siguiente error:

```
ORA-14705: claves únicas o primarias a las que hacen referencia las claves
ajenas activadas en la tabla
```

- No hay límite en el número de niveles recursivos, ya que se aplicará a todos.
- La cláusula CASCADE también se puede aplicar en las particiones de tabla y sub-particiones.

```
TRUNCATE TABLE nombre_tabla PARTITION nombre_partición CASCADE;
```

- Esta nueva funcionalidad elimina la condición previa de truncar los registros secundarios antes de truncar una tabla maestra.

Privilegios

Un privilegio de usuario es el permiso para ejecutar un tipo particular de sentencia SQL, o para el uso de un objeto determinado que pertenezca a otro usuario.

Oracle dispone de sistemas para agrupar los privilegios, de manera que los que son más comúnmente asociados a los usuarios, puedan ser concedidos o revocados de manera sencilla.

De manera genérica existen dos tipos de privilegios:

- Privilegios de sistema
- Privilegios sobre objetos

Privilegios sobre objetos

Los privilegios sobre objetos conceden el derecho a efectuar un determinado tipo de acción sobre un objeto específico tales como Tablas, Vistas, Secuencias o Procedimientos.

Para conceder o revocar un privilegio sobre un objeto determinado, es necesario hallarse en al menos una de las siguientes situaciones:

- Ser el propietario del objeto (el objeto está en el esquema del usuario que concede o revoca el privilegio).
- Haberle sido concedido al usuario ese privilegio que a su vez va a conceder o revocar con la cláusula WITH GRANT OPTION.

Cada objeto, maneja como es natural distintos tipos de privilegios como en la tabla que sigue:

Privilegio	TABLA	VISTA	SECUENCIA	PROC. ALMAC
ALTER	X		X	
DELETE	X	X		
EXECUTE				X
INDEX	X*			
INSERT	X	X		
REFERENCES	X*			
SELECT	X	X**	X	
Privilegio	TABLA	VISTA	SECUENCIA	PROC. ALMAC
UPDATE	X	X		

(*) Este privilegio no puede ser concedido a un rol.

(**) El privilegio puede concederse también a vistas materializadas (snapshots).

El privilegio UPDATE puede concederse columna por columna, si no se especifica nada entonces el privilegio se aplica a todas las columnas de la tabla.

Para conceder los privilegios sobre objetos la sintaxis es más o menos similar a la de privilegios del sistema:

```
GRANT privilegio [ (columna1,columna2,...) ] ON nombreobjeto
      TO nombreusuario | nombrerol | PUBLIC
            [ WITH GRANT OPTION ]
```

Algunos ejemplos:

```
GRANT delete, insert ON tabla1 TO juan WITH GRANT OPTION;
GRANT insert ON tabla2 TO PUBLIC;
GRANT update (columna1,columna2) ON tabla3 TO Rol1;
GRANT insert,update, delete ON tabla4 TO juan,pepe;
```

En el primero de los ejemplos se han concedido los privilegios “delete” e “insert” (eliminar y añadir) sobre la tabla “tabla1” al usuario Juan con la posibilidad de que este pueda a su vez conceder o revocar estos privilegios sobre la tabla “tabla1” a otros usuarios.

En el segundo caso se concede el privilegio, añadir registros a la tabla “tabla2” a todos los usuarios de la base de datos.

En el tercero se concede el privilegio modificar tan solo las columnas “columna1” y “columna2” de la tabla “tabla3” al rol “rol1”.

En el cuarto supuesto se conceden los privilegios de añadir, modificar y eliminar registros sobre la tabla “tabla4” a los usuarios Juan y Pepe.

Como se ha visto la sintaxis es similar a la empleada para la concesión de privilegios del sistema, sin embargo, hay dos puntos de distinción entre esta sintaxis y la de privilegios del sistema:

- En primer lugar los privilegios sobre objetos siempre se conceden o revocan sobre un objeto, de ahí que sea necesario asignar el objeto mediante la cláusula ON nombreobjeto, de esta manera los privilegios señalados serán asignados a ese objeto únicamente, esta cláusula es obligatoria no pudiendo por tanto establecer privilegios de manera genérica a todos los objetos.
- La segunda de las diferencias, se basa en la sintaxis utilizada para permitir que el usuario que recibe el privilegio pueda a su vez conceder o revocar ese privilegio, sobre ese objeto concretamente, para ello en este caso se utiliza la cláusula WITH GRANT OPTION a diferencia de la utilizada para los privilegios del sistema (WITH ADMIN OPTION).

Es importante reseñar que la cláusula WITH GRANT OPTION, no puede ser utilizada cuando el privilegio es concedido a un rol de la base de datos, a diferencia de la cláusula WITH ADMIN OPTION, que si puede ser concedida a roles de la base de datos.

En el caso de los receptores del privilegio, de igual manera que los privilegios del sistema, pueden ser concedidos a usuarios nominales (para ello utilizamos la cláusula TO nombreusuario), roles de la base de datos (TO nombrerol), o a todos los usuarios de la base de datos (TO PUBLIC).

Eliminar privilegios sobre objetos

Los privilegios sobre objetos pueden ser eliminados por un usuario que posea el privilegio con la opción WITH GRANT OPTION.

Para eliminar el privilegio se utiliza la sentencia:

```
REVOKE privilegio | ALL ON nombreobjeto FROM nombreusuario | nombrerol |
PUBLIC CASCADE CONSTRAINTS
```

Privilegio | ALL: El privilegio sobre el objeto a suprimir. ALL sustituye a todos los privilegios concedidos sobre el objeto.

Nombreobjeto: Nombre del Objeto sobre el que se suprimen los privilegios.

nombreusuario | nombrerol | PUBLIC: como hemos visto anteriormente al conceder privilegios, estos pueden ser revocados para un usuario, para un rol o para todos los usuarios de la base de datos.

CASCADE CONSTRAINTS: Elimina todas las restricciones de integridad referencial que hayan sido definidas utilizando el privilegio REFERENCES. Si no se indica esta cláusula y existen estas referencias, se produce un error.

```
REVOKE create view, create table FROM juan
REVOKE ALL ON Tabla1 FROM juan
```

Al contrario de lo que ocurre al conceder privilegios, estos no pueden ser revocados de manera selectiva a columnas. Para revocar un privilegio sobre una columna determinada de una tabla es necesario, revocar en primer lugar los privilegios sobre todas las columnas, y entonces volver a conceder los privilegios sobre las columnas sobre las que no se quiere modificar los privilegios.

```
REVOKE UPDATE ON TABLA1 TO PUBLIC
GRANT UPdATE (CAMPO1) ON TABLA1 TO PUBLIC
```

Visualizar los privilegios sobre objetos

Al igual que se ha visto en numerosas ocasiones, se puede obtener información a través de vistas del sistema, acerca de los privilegios concedidos sobre objetos de la base de datos.

Las vistas más usuales son:

Vista	Descripción
DBA_COL_PRIVS	Privilegios sobre columnas de todos los usuarios
ALL_COL_PRIVS	Privilegios sobre columnas para los que el usuario o PUBLIC es el receptor
USER_COL_PRIVS	Privilegios del usuario sobre columnas
ALL_COL_PRIVS_MADE	Privilegios sobre columnas de las que el usuario es propietario o concede
USER_COL_PRIVS_MADE	Todos los privilegios sobre columnas del usuario
ALL_COL_PRIVS_RECD	Privilegios sobre columnas para las que el usuario o PUBLIC es receptor del privilegio
USER_COL_PRIVS_RECD	Privilegios sobre columnas en los que el usuario es el receptor

Vista	Descripción
DBA_TAB_PRIVS	Privilegios sobre objetos de todos los usuarios
ALL_TAB_PRIVS	Privilegios sobre objetos de todos los usuarios
USER_TAB_PRIVS	Privilegios sobre objetos de los que el usuario es propietario, concede, o receptor
ALL_TAB_PRIVS_RECD	Privilegios sobre objetos para los que el usuario o PUBLIC es el receptor
USER_TAB_PRIVS_RECD	Privilegios sobre objetos de los que el usuario es receptor
ALL_TAB_PRIVS_MADE	Privilegios del usuario y privilegios sobre objetos del usuario
DBA_SYS_PRIVS	Privilegios de todos los usuarios sobre el sistema.
COLUMN_PRIVILEGES	Privilegios sobre columnas de las que el usuario o PUBLIC es propietario, concede o receptor del privilegio.
USER_SYS_PRIVS	Privilegios del usuario sobre el sistema

```
SELECT * FROM DBA_SYS_PRIVS;
GRANTEE            PRIVILEGE                        ADM
-------------      -------------------------------- ---
SECURITY_ADMIN     ALTER PROFILE                    YES
SECURITY_ADMIN     AUDIT ANY                        YES
SECURITY_ADMIN     AUDIT SYSTEM                     YES
SECURITY_ADMIN     BECOME USER                      YES
SECURITY_ADMIN     CREATE PROFILE                   YES
SECURITY_ADMIN     CREATE ROLE                      YES
SECURITY_ADMIN     DROP USER                        YES
SECURITY_ADMIN     GRANT ANY ROLE                   YES
MIGUEL             CREATE SESSION                   NO
JUAN               CREATE SESSION                   NO
```

```
SELECT TABLE_NAME, PRIVILEGE, GRANTABLE FROM DBA_TAB_PRIVS
    WHERE GRANTEE = 'JUAN';

TABLE_NAME   PRIVILEGE    GRANTABLE
-----------  ------------ ----------
EMP          SELECT       NO
EMP          DELETE       NO
```

```
SELECT GRANTEE, TABLE_NAME, COLUMN_NAME, PRIVILEGE
    FROM DBA_COL_PRIVS;
```

```
GRANTEE      TABLE_NAME     COLUMN_NAME      PRIVILEGE
-----------  ------------   -------------    --------------
MIGUEL       EMP            CODEMPLEADO      INSERT
MIGUEL       EMP            TRABAJO          INSERT
JUAN         EMP            CODEMPLEADO      INSERT
JUAN         EMP            TRABAJO          INSERT
```

ANEXO A. Funciones SQL

Tabla de contenidos

Introducción 1

Funciones Numéricas 1

Funciones de cadena de caracteres que devuelven caracteres 7

Funciones de cadena de caracteres que devuelven números 12

Funciones de fecha y hora 13

Funciones de conversión 21

Otras funciones. 26

Introducción

A continuación se relacionan las funciones SQL más comunes clasificadas según los grupos siguientes:

- Funciones Numéricas
- Funciones de cadena de caracteres que devuelven caracteres
- Funciones de cadena de caracteres que devuelven números
- Funciones de fecha y hora
- Funciones de conversión
- Otras funciones

Funciones Numéricas

ABS	Sintaxis	ABS(n)
	Propósito	Devuelve el valor absoluto de n.
	Ejemplo	SQL> SELECT ABS(-25) FROM DUAL; ABS(-25) ---------- 25
ACOS	Sintaxis	ACOS(n)
	Propósito	Devuelve el coseno del arco de n.
	Ejemplo	SQL> SELECT ACOS(.25) FROM DUAL; ACOS(.25) ---------- 1,31811607
ASIN	Sintaxis	ASIN(n)
	Propósito	Devuelve el seno del arco de n.

	Ejemplo	SQL> SELECT ASIN (.25) FROM DUAL; ASIN(.25) ---------- ,252680255
ATAN	Sintaxis	ATAN(n)
	Propósito	Devuelve la tangente del arco de n.
	Ejemplo	SQL> SELECT ATAN(.25) FROM DUAL; ATAN(.25) ---------- ,244978663
ATAN2	Sintaxis	ATAN2(N,M)
	Propósito	Devuelve la tangente del arco de n y m.
	Ejemplo	SQL> SELECT ATAN2(.25,.3) FROM DUAL; ATAN2(.25,.3) ---------- ,694738276
ATAN2	Sintaxis	ATAN2(N,M)
	Propósito	Devuelve la tangente del arco de n y m.
	Ejemplo	SQL> SELECT ATAN2(.25,.3) FROM DUAL; ATAN2(.25,.3) ---------- ,694738276
BITAND	Sintaxis	BITAND(arg1,arg2)
	Propósito	Devuelve el resultado de la operación lógica AND entre argumento1 y argumento2, expresados en notación binaria. Se suele utilizar con la función DECODE.

	Ejemplo	SQL> SELECT bitand(4,7) FROM DUAL; BITAND(4,7) 100 111 ---------- 4
CEIL	Sintaxis	CEIL(n)
	Propósito	Devuelve el entero superior o igual a n.
	Ejemplo	SQL> SELECT ceil(12.1) FROM DUAL; CEIL(12.1) ---------- 13
COS	Sintaxis	COS(n)
	Propósito	Devuelve el coseno de n en radianes.
	Ejemplo	SQL> SELECT cos(180) FROM DUAL; COS(180) ---------- -,59846007
COSH	Sintaxis	COSH(n)
	Propósito	Devuelve el coseno hiperbólico de n.
	Ejemplo	SQL> SELECT cosh(0) FROM DUAL; COSH(0) ---------- 1
EXP	Sintaxis	EXP(n)
	Propósito	Devuelve el número e elevado a la potencia de n
	Ejemplo	SQL> SELECT exp(3) FROM DUAL; EXP(3) ---------- 20,0855369
FLOOR	Sintaxis	FLOOR(n)

	Propósito	Devuelve el entero inferior o igual a n.
	Ejemplo	SQL> SELECT floor(13.6) FROM DUAL; FLOOR(13.6) ---------- 13
LN	Sintaxis	LN(n)
	Propósito	Devuelve el logaritmo neperiano de n.
	Ejemplo	SQL> SELECT ln(27) FROM DUAL; LN(27) ---------- 3,29583687
LOG	Sintaxis	Log(M,N)
	Propósito	Devuelve el logaritmo en base m de n. M y n deben de ser enteros positivos y m!=0 y m!=1
	Ejemplo	SQL> SELECT log(10,1000) FROM DUAL; LOG(10,1000) ---------- 3
MOD	Sintaxis	MOD(m,n)
	Propósito	Devuelve el resto de m dividido entre n.
	Ejemplo	SQL> SELECT mod(4,3) FROM DUAL; MOD(4,3) ---------- 1
POWER	Sintaxis	POWER(m,n)
	Propósito	Devuelve m elevado a la potencia n. Si m es negativo, n debe de ser un numero entero.

	Ejemplo	SQL> SELECT power(10,3) FROM DUAL; POWER(10,3) ---------- 1000
ROUND	Sintaxis	ROUND(n,[m])
	Propósito	Devuelve n redondeado a m. Si se omite m por defecto es 0.
	Ejemplo	SQL> SELECT round(12.345,2) FROM DUAL; ROUND(12.345,2) --------------- 12,35
SIGN	Sintaxis	SIGN(n)
	Propósito	Si n>0 devuelve1. Si n=0 devuelve 0. Si n<0 devuelve –1.
	Ejemplo	SQL> SELECT sign(-14) FROM DUAL; SIGN(-14) ---------- -1
SIN	Sintaxis	SIN(n)
	Propósito	Devuelve el seno de n.
	Ejemplo	SQL> SELECT sin(90) FROM DUAL; SIN(90) ---------- ,893996664
SINH	Sintaxis	SINH(n)
	Propósito	Devuelve el seno hiperbólico de n
	Ejemplo	SQL> SELECT sinh(1) FROM DUAL; SINH(1) ---------- 1,17520119

SQRT	Sintaxis	SQRT(n)
	Propósito	Devuelve la raíz cuadrada de n. n>0
	Ejemplo	SQL> SELECT sqrt(16) FROM DUAL; SQRT(16) ---------- 4
TAN	Sintaxis	TAN(n)
	Propósito	Devuelve la tangente de n.
	Ejemplo	SQL> SELECT tan(45) FROM DUAL; TAN(45) ---------- 1,61977519
TANH	Sintaxis	TANH(n)
	Propósito	Devuelve la tangente hiperbólica de n.
	Ejemplo	SQL> SELECT tanh(45) FROM DUAL; TANH(45) ---------- 1
TRUNC	Sintaxis	TRUNC(n,[m])
	Propósito	Devuelve n truncado a m decimales. Si se omite m el valor por defecto es 0.
	Ejemplo	SQL> SELECT trunc(12.345,2) FROM DUAL; TRUNC(12.345,2) --------------- 12,34
WIDTH_BUCK	Sintaxis	WIDTH_BUCKET (expr min, max , num)

ET	Propósito	Utilizado para la construcción de histogramas. Compara los valores de expresión con los valores mínimo y máximo. En el caso de superar se asigna el valor num.
	Ejemplo	SQL> SELECT FIRST_NAME,SALARY, WIDTH_BUCKET(salary, 100, 10000,30) "Grupo" FROM employees where department_id =30 order by salary; FIRST_NAME SALARY Grupo ---------- ---------- ---------- Karen 2500 8 Guy 2600 8 Sigal 2800 9 Shelli 2900 9 Alexander 3100 10 Den 11000 31

Funciones de cadena de caracteres que devuelven caracteres

CHR	Sintaxis	CHR(n, USING NCHAR_CS)
	Propósito	Devuelve el carácter cuyo número correspondiente en binario es n. La transformación es en el juego de caracteres de la base de datos o en el juego de caracteres nacional.
	Ejemplo	SQL> SELECT chr(68) FROM DUAL; C - D
CONCAT	Sintaxis	CONCAT(cadena1, cadena2)
	Propósito	Devuelve la cadena1 concatenada con la cadena2. Es equivalente al operador ||.

	Ejemplo	SQL> SELECT concat('Mi nombre es: ',first_name) Nombre FROM employees; NOMBRE ---------- Mi nombre es: Steven
INITCAP	Sintaxis	INITCAP(cadena)
	Propósito	Devuelve la primera letra de cada palabra en mayúsculas. Palabras delimitadas por blancos.
	Ejemplo	SQL> SELECT initcap('el nombre') FROM DUAL; INITCAP(' --------- El Nombre
LOWER	Sintaxis	LOWER(cadena)
	Propósito	Devuelve la cadena de caracteres con todas las letras en minúsculas
	Ejemplo	SQL> SELECT lower('holA MUNdo') FROM DUAL; LOWER('HOL ---------- hola mundo
LPAD	Sintaxis	LPAD(cadena1,n,[cadena2])
	Propósito	Devuelve la cadena1 rellenada con el carácter por la izquierda indicado en cadena2, hasta la longitud indicada en n. Si se omite la cadena2 se rellena hasta la longitud con blancos.
	Ejemplo	SQL> SELECT lpad('Base',10,'*') FROM DUAL; LPAD('BASE ---------- ******Base
LTRIM	Sintaxis	LTRIM(cadena, muestra)

	Propósito	Borra los caracteres a la izquierda de la cadena de caracteres hasta encontrar el primer carácter que no está en la muestra.
	Ejemplo	SQL> SELECT ltrim('pspspEl perro','ps') FROM DUAL; LTRIM('P -------- El perro
NLS_INITCAP	Sintaxis	NLS_INITCAP(cadena, parámetros NLS)
	Propósito	Devuelve la cadena de caracteres la primera letra de cada palabra en mayúsculas. Dependiente de del parámetro de ordenación NLS
	Ejemplo	SQL> SELECT nls_initcap('ijs','NLS_SORT = XDutch') FROM DUAL; IJs SQL> SELECT nls_initcap('ijs') FROM DUAL; Ijs
NLS_LOWER	Sintaxis	NLS_LOWER(cadena, parámetros NLS)
	Propósito	Devuelve la cadena de caracteres en minúsculas. Dependiente de del parámetro de ordenación NLS
	Ejemplo	SQL> SELECT nls_lower('HOla','NLS_SORT = XDutch') FROM DUAL; NLS_ ---- hola
NLSSORT	Sintaxis	NLSSORT(cadena, parámetros NLS)
	Propósito	Devuelve los datos ordenados dependiendo de los parámetros del lenguaje NLS.

		SELECT * FROM test ORDER BY NLSSORT(nombre); NOMBRE ---------- Gaaciar Gabia
	Ejemplo	SELECT * FROM test ORDER BY NLSSORT(nombre, 'NLS_SORT = XDanish'); NOMBRE ---------- Gabia Gaaciar
NLS_UPPER	Sintaxis	NLS_UPPER(cadena, parámetros NLS)
	Propósito	Devuelve la cadena de caracteres en mayúsculas. Dependiente de del parámetro de ordenación NLS
	Ejemplo	SQL> SELECT NLS_UPPER('HOla','NLS_SORT = XDutch') FROM DUAL; NLS_ ---- HOLA
REPLACE	Sintaxis	REPLACE(cadena, c1,[c2]))
	Propósito	Busca en la cadena el carácter c1 y lo remplaza por c2.
	Ejemplo	SQL> SELECT REPLACE('perro','rr','l') FROM DUAL; REPL ---- pelo
RPAD	Sintaxis	RPAD(cadena1,n,[cadena2])
	Propósito	Devuelve la cadena1 rellenada por la derecha con el carácter indicado en cadena2, hasta la longitud indicada en n. Si se omite la cadena2 se rellena hasta la longitud con blancos.

	Ejemplo	SQL> SELECT RPAD('Base',10,'*') FROM DUAL; RPAD('BASE ---------- Base******
RTRIM	Sintaxis	RTRIM(cadena, muestra)
	Propósito	Borra los caracteres a la derecha de la cadena de caracteres hasta encontrar el primer carácter que no está en la muestra.
	Ejemplo	SQL> SELECT RTRIM('El perropspsp','ps') FROM DUAL; RTRIM('P -------- El perro
SOUNDEX	Sintaxis	SOUNDEX(cadena)
	Propósito	Devuelve una cadena de caracteres conteniendo la representación fonética de los caracteres. Basado en el idioma inglés.
	Ejemplo	SELECT last_name FROM hr.employees WHERE SOUNDEX(last_name) = SOUNDEX('SMYTHE'); LAST_NAME -------------- Smith
SUBSTR	Sintaxis	SUBSTR(cadena,M,[N])
	Propósito	Devuelve una parte de la cadena, empezando por la posición indicada en M y tantos caracteres como se indica en N. Existen las funciones SUBSTRB, SUBSTRC
	Ejemplo	SQL> SELECT SUBSTR('papagayo',1,4) FROM DUAL; SUBS ---- papa

TRANSLATE	Sintaxis	TRANSLATE(cadena, c1,c2)
	Propósito	Devuelve la cadena de caracteres con todas las ocurrencias de c1 reemplazadas por c2.
	Ejemplo	SQL> SELECT TRANSLATE('NUM123', '0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ', '9999999999XXXXXXXXXXXXXXXXXXXXXX') FROM DUAL; TRANSL ------ XXX999
TRIM	Sintaxis	TRIM(cadena,FROM cadena origen)
	Propósito	Elimina los caracteres indicados tanto en la parte derecha como en la parte izquierda de la cadena de caracteres origen.
	Ejemplo	SELECT TRIM (0 FROM 0009872348900) "TRIM Example" FROM DUAL; TRIM Example ------------ 98723489
UPPER	Sintaxis	UPEER(cadena)
	Propósito	Devuelve la cadena de caracteres con todas las letras en MAYÚSCULAS
	Ejemplo	SQL> SELECT UPPER('holA MUNdo') FROM DUAL; LOWER('HOL ---------- HOLA MUNDO

-

Funciones de cadena de caracteres que devuelven números

ASCII	Sintaxis	ASCII(cadena)

	Propósito	Devuelve la representación numérica del carácter indicado.
	Ejemplo	SQL> SELECT ASCII('A') FROM DUAL; ASCII('A') ---------- 65
INSTR	Sintaxis	INSTR(cadena, c1,[N][,M]))
	Propósito	Busca en la cadena el carácter c1 empezando a buscar por N y devuelve la posición M en la que aparece. Existe también INSTRB.
	Ejemplo	SQL> SELECT INSTR('zaragoza','a',1,2) FROM DUAL; INSTR('ZARAGOZA','A',1,2) ------------------------- 4
LENGTH	Sintaxis	LENGTH(cadena1)
	Propósito	Devuelve la longitud de la cadena, incluyendo los blancos..
	Ejemplo	SQL> SELECT LENGTH('hola mundo') FROM DUAL; LENGTH('HOLAMUNDO') ------------------- 10

Funciones de fecha y hora

ADD_MONTHS	Sintaxis	ADD_MONTHS(fecha,n)
	Propósito	Devuelve la fecha mas los meses indicados en n

	Ejemplo	SQL> SELECT ADD_MONTHS('27/05/78',6) FROM DUAL; ADD_MONT -------- 27/11/78
CURRENT_DATE	Sintaxis	CURRENT_DATE
	Propósito	Devuelve la fecha actual, teniendo en cuenta la zona horaria. TIME_ZONE
	Ejemplo	ALTER SESSION SET TIME_ZONE = '-5:0'; SQL> SELECT CURRENT_DATE FROM DUAL; CURRENT_DATE ---------- 12-MAR-2003 12:31:23 SQL> ALTER SESSION SET TIME_ZONE = '-8:0'; SQL> SELECT CURRENT_DATE FROM DUAL; CURRENT_DATE ---------- 12-MAR-2003 09:31:41
CURRENT_TIMESTAMP	Sintaxis	CURRENT_TIMESTAMP(precision)
	Propósito	Devuelve la fecha y la hora actual en la zona horaria de la sesión. La precisión para los segundos por defecto es 6.
	Ejemplo	SQL> ALTER SESSION SET TIME_ZONE = '-5:0'; SQL> SELECT CURRENT_TIMESTAMP FROM DUAL; CURRENT_TIMESTAMP ---------- 13/03/03 04:19:22,007000 -05:00
DBTIMEZONE	Sintaxis	DBTIMEZONE

	Propósito	Devuelve la zona horaria de la base de datos. Este valor se establece en el momento de creación de la base de datos. Puede ser modificado con ALTER DATABASE.
	Ejemplo	SQL> SELECT DBTIMEZONE FROM DUAL; DBTIME ------ +01:00

EXTRACT	Sintaxis	EXTRACT ({ { AÑO | MES | DIA | HORA | MINUTOS | SEGUNDOS } | { TIMEZONE_HOUR | TIMEZONE_MINUTE } | { TIMEZONE_REGION | TIMEZONE_ABBR } } FROM { expresión de fecha | expresión interval })
	Propósito	Devuelve el valor especificado de una expresión de fecha o intervalo.
	Ejemplo	SQL> SELECT EXTRACT(YEAR FROM DATE '1995-04-29') FROM DUAL; EXTRACT(YEARFROMDATE'1995-04-29') ---------- 1995

FROM_TZ	Sintaxis	FROM_TZ (timestamp_value , time_zone_value)
	Propósito	Devuelve un valor timestamp conevertidoa a timestamp con TIME_ZONE
	Ejemplo	SQL> SELECT FROM_TZ(TIMESTAMP '2003-03-28 08:00:00', '5:00') 2 FROM DUAL; FROM_TZ(TIMESTAMP'2003-03-2808:00:00','5:00') --- 28/03/03 08:00:00,000000000 +05:00
LAST_DAY	Sintaxis	LAST_DAY(fecha)
	Propósito	Devuelve el ultimo día del mes que contenga la fecha especificada como argumento.
	Ejemplo	SQL> SELECT SYSDATE,LAST_DAY(SYSDATE) FROM DUAL; SYSDATE LAST_DAY(SYSDATE) --- --- 13-MAR-2003 10:49:00 31-MAR-2003 10:49:00
LOCAL TIMESTAMP	Sintaxis	LOCALTIMESTAMP(precision)
	Propósito	Devuelve la fecha y la hora actual con TIME ZONE en el formato de TIMESTAMP.
	Ejemplo	SQL> SELECT CURRENT_TIMESTAMP, LOCALTIMESTAMP FROM DUAL; CURRENT_TIMESTAMP LOCALTIMESTAMP --- --- 13/03/03 04:52:34,092000 -05:00 13/03/03 04:52:34,092000

MONTHS_ BETWEEN		MONTHS_BETWEEN(fecha1, fecha2)
	Sintaxis	
	Propósito	Devuelve el número de meses entre las dos fechas que recibe como argumentos.
	Ejemplo	SQL> SELECT MONTHS_BETWEEN(SYSDATE+34,SYSDATE) FROM DUAL; MONTHS_BETWEEN(SYSDATE+34,SYSDATE) ---------- 1,09677419
NEW_TIME	Sintaxis	NEW_TIME (fecha , zona1 , zona2)
	Propósito	Devuelve la fecha en la zona horaria especificada en el argumento zona2.
	Ejemplo	SQL> SELECT SYSDATE, NEW_TIME(SYSDATE, 'AST','PST') FROM DUAL; SYSDATE NEW_TIME(SYSDATE,'AS ---------- ---------- 13-MAR-2003 11:16:04 13-MAR-2003 07:16:04
NEXT_DAY	Sintaxis	NEXT_DAY (fecha , char)
	Propósito	Devuelve la fecha del día de la semana indicado en char a partir de la fecha indicada en fecha.
	Ejemplo	SQL> SELECT NEXT_DAY(SYSDATE,'DOMINGO') FROM DUAL; NEXT_DAY(SYSDATE,'DO ---------- 16-MAR-2003 11:20:41
NUMTODSINTERVAL	Sintaxis	NUMTODSINTERVAL (n , 'char_expr')
	Propósito	Convierte el numero n en el tipo literal INTERVAL DAY TO SECOND. 'DAY' ,'HOUR' ,'MINUTE' , 'SECOND'

	Ejemplo	SQL> SELECT NUMTODSINTERVAL (4,'DAY') FROM DUAL; NUMTODSINTERVAL(4,'DAY') --- +000000004 00:00:00.000000000
NUMTOYMINTERVAL	Sintaxis	NUMTOYMINTERVAL (n , 'char_expr')
	Propósito	Convierte el numero n en el tipo literal INTERVAL YEAR TO MONTH. 'YEAR' ,'MONTH'
	Ejemplo	SQL> SELECT NUMTOYMINTERVAL (4,'YEAR') FROM DUAL; NUMTOYMINTERVAL(4,'YEAR') --- +000000004-00
ROUND	Sintaxis	ROUND (FECHA [, formato])
	Propósito	Devuelve la fecha correspondiente al formato especificado. Si se omite se muestra la fecha del día siguiente.
	Ejemplo	SQL> SELECT SYSDATE,ROUND(SYSDATE, 'YEAR') FROM DUAL; SYSDATE ROUND(SYSD ---------- ---------- 13-03-2003 01-01-2003
SESSIONTIMEZONE	Sintaxis	SESSIONTIMEZONE
	Propósito	Devuelve el TIME ZONE (zona horaria) de la sesión.
	Ejemplo	SQL> SELECT SESSIONTIMEZONE FROM DUAL; SESSIONTIMEZONE --- -05:00
SYS_EXTRACT_UTC	Sintaxis	SYS_EXTRACT_UTC (datetime_with_timezone

	Propósito	Devuelve la fecha UTC, de la fecha que recibe como argumento con desplazamiento de zona horaria.
	Ejemplo	SQL> SELECT SYS_EXTRACT_UTC(TIMESTAMP '2000-03-28 11:30:00.00 -08:00') FROM DUAL; SYS_EXTRACT_UTC --- 28/03/00 19:30:00,000000000
SYSDATE	Sintaxis	SYSDATE
	Propósito	Devuelve la fecha y la hora actual. Devuelve un tipo de datos DATE
	Ejemplo	SQL> SELECT SYSDATE FROM DUAL; SYSDATE --- 13-03-2003
SYSTIMESTAMP	Sintaxis	SYSTIMESTAMP
	Propósito	Devuelve la fecha y la hora del sistema y la zona horaria. El tipo de datos devuelto es TIMESTAMP WITH TIME ZONE
	Ejemplo	SQL> SELECT SYSTIMESTAMP FROM DUAL; SYSTIMESTAMP --- 13/03/03 11:49:27,380000 +01:00
TO_DSINTERVAL	Sintaxis	TO_DSINTERVAL (char ['nlsparam'])
	Propósito	Convierte la cadena de caracteres en el tipo de datos INTERVAL DAY TO SECOND.
	Ejemplo	SQL> SELECT TO_DSINTERVAL('100 10:00:00') FROM DUAL; TO_DSINTERVAL('10010:00:00') --- +000000100 10:00:00.000000000
TO_TIMESTAMP	Sintaxis	TO_TIMESTAMP (char [, fmt ['nlsparam']]

	Propósito	Convierte la cadena de caracteres en el tipo de dato TIMESTAMP.
	Ejemplo	SQL> SELECT TO_TIMESTAMP ('1999-12-01 11:00:00', 'YYYY-MM-DD HH:MI:SS') FROM DUAL; TO_TIMESTAMP('19990','YYYY-MM-DDHH:MI:SS') --- 01/12/99 11:00:00,000000000
TO_TIMESTAMP_TZ	Sintaxis	TO_TIMESTAMP_TZ (char[,fmt 'nlsparam']])
	Propósito	Convierte la cadena de caracteres en el tipo de dato TIMESTAMP WITH TIME ZONE.
	Ejemplo	SQL> SELECT TO_TIMESTAMP_TZ(SYSDATE) FROM DUAL; TO_TIMESTAMP_TZ(SYSDATE) --- 13/03/20 03:12:26 -05:00
TO_YMINTERVAL	Sintaxis	TO_YMINTERVAL (char)
	Propósito	Convierte la cadena de caracteres en el tipo de datos INTERVAL YEAR TO MONTH.
	Ejemplo	SQL> SELECT hire_date, hire_date + TO_YMINTERVAL('01-02') FROM employees; HIRE_DATE --- --- 17-JUN-1987 00:00:00 17-AGO-1988 00:00:00 21-SEP-1989 00:00:00 21-NOV-1990 00:00:00
TRUNC	Sintaxis	TRUNC (FECHA [, fmt])
	Propósito	Devuelve la fecha truncada al formato especificado.

	Ejemplo	SQL> SELECT SYSDATE, TRUNC(SYSDATE,'YEAR') FROM DUAL; SYSDATE TRUNC(SYSDATE,'YEAR' -------------------- -------------------- 13-MAR-2003 12:31:24 01-ENE-2003 00:00:00
TZ_OFFSET	Sintaxis	TZ_OFFSET ({ 'time_zone_name' | '{ + | - } hh : mi' | SESSIONTIMEZONE | DBTMEZONE })
	Propósito	Devuelve el desplazamiento de la zona horaria especificada.
	Ejemplo	SQL> SELECT TZ_OFFSET('US/Eastern') FROM DUAL; TZ_OFFS ------- -05:00

Funciones de conversión

ASCIISTR	Sintaxis	ASCIISTR(cadena)
	Propósito	Convierte la cadena de caracteres en ASCII
	Ejemplo	SELECT ASCIISTR('ABÄCDE') FROM DUAL; ASCIIS ------ ABDCDE
BIN_TO_NUM	Sintaxis	BIN_TO_NUM(expr[,expr]...)
	Propósito	Convierte una cadena binaria de bits en un número.

	Ejemplo	SQL> SELECT BIN_TO_NUM(1,0,1,1,0) FROM DUAL; BIN_TO_NUM(1,0,1,1,0) ---------- 22
CAST	Sintaxis	CAST({expr | (subquery)| MULTISET(subquery)} AS type_name)
	Propósito	Convierte un tipo de datos en otro tipo de datos.
	Ejemplo	SQL> SELECT CAST('22-OCT-1997' AS TIMESTAMP WITH LOCAL TIME ZONE) FROM DUAL; CAST('22-OCT-1997'ASTIMESTAMPWITHLOCALTIMEZONE) --- 22/10/97 00:00:00,000000
CHARTOROWID	Sintaxis	CHARTOROWID(char)
	Propósito	Convierte la cadena de caracteres en tipo de datos ROWID
	Ejemplo	SQL> SELECT last_name FROM employees WHERE ROWID = CHARTOROWID('AAAGwDAAEAAAABUAAA'); LAST_NAME --- King
COMPOSE	Sintaxis	COMPOSE('string')
	Propósito	La cadena de caracteres la convierte en una cadena de caracteres UNICODE.
	Ejemplo	SQL> SELECT COMPOSE ('o' || UNISTR('\0308')) FROM DUAL; C - ö

CONVERT	Sintaxis	CONVERT(char,dest_char_set[,source_char_set])
	Propósito	Convierte una cadena en un juego de caracteres a otro.
	Ejemplo	SQL> SELECT CONVERT('España', 'we8dec','us7ascii') FROM DUAL; CONVER ------ Espaqa
DECOMPOSE	Sintaxis	DECOMPOSE('string' [CANONICAL | COMPATIBILITY])
	Propósito	Valida solo para cadenas de caracteres UNICODE. Devuelve la cadena de caracteres de entrada en UNICODE.
	Ejemplo	SELECT DECOMPOSE ('Châteaux') FROM DUAL; DECOMPOSE --------- Cha^teaux
HEXTORAW	Sintaxis	HEXTORAW (char)
	Propósito	Convierte una cadena que contiene dígitos en hexadecimal a un valor raw.
	Ejemplo	CREATE TABLE test (raw_col RAW(10)); INSERT INTO test VALUES (HEXTORAW('7E6D'));
RAWTOHEX	Sintaxis	RAWTOHEX (raw)
	Propósito	Convierte un valor raw una cadena que contiene dígitos en hexadecimal.
	Ejemplo	SQL> SELECT RAWTOHEX(raw_col) FROM test; RAWTOHEX(RAW_COL) -------------------- 7E6D
RAWTONHEX	Sintaxis	RAWTONHEX (raw)
	Propósito	Convierte un valor raw una cadena de juego de caracteres nacional que contiene dígitos en hexadecimal.

		SQL> SELECT RAWTNOHEX(raw_col) FROM test; RAWTOHEX(RAW_COL) ---------- 7E6D
	Ejemplo	
ROWIDTOCHAR	Sintaxis	ROWIDTOCHAR(rowid)
	Propósito	Convertir un tipo de dato ROWID a VARCHAR2. De la conversión resulta siempre una cadena de 18 caracteres de longitud.
	Ejemplo	SELECT ROWID FROM oficinas WHERE ROWIDTOCHAR(rowid) LIKE '%Br1AAB%' ROWID ---------- AAAAZ6AABAAABr1AAB
TO_CHAR, conversión de fechas	Sintaxis	TO_CHAR(f, [,formato [, 'parametros_nls']])
	Propósito	Convierte una fecha f tipo DATE en un VARCHAR2 con la máscara indicada en formato. Si se omite formato, la fecha se convierte a en un valor VARCHAR2 en el formato de fecha por defecto. Más adelante veremos los "modelos de formato de fecha". Parametros_nls especifica el lenguaje en el cual se visualizan los nombres y abreviaturas de los meses y días. El parámetro se indica con: 'NLS_DATE_LANGUAGE = 'idioma'. Si se omite toma el idioma por defecto para la sesión.
	Ejemplo	SELECT TO_CHAR(SYSDATE, 'Month DD, YYYY') "Hoy" FROM DUAL Hoy ---------- Julio 10, 2003
TO_CHAR,	Sintaxis	TO_CHAR(número[,formato [,'parametros_nls']])

conversión de números	Propósito	Convierte número de tipo NUMBER en un tipo de datos VARCHAR2 con la máscara indicada en formato. Si se omite formato, la fecha se convierte a en un valor VARCHAR2 lo suficientemente grande para contener los dígitos más significativos.. Más adelante veremos los "modelos de formato numéricos". 'Parámetros_nls' especifica los caracteres y elementos de formatos de números que se visualiza (separador de decimales, de grupo, símbolo de la moneda local e ISO).
	Ejemplo	SELECT TO_CHAR(-10000, 'L99G999D99MI') "Cantidad" FROM DUAL; Cantidad ---------- $10,000.00-
	Sintaxis	TO_DATE(char,[,formato] [,'parametros_nls']])
TO_DATE	Propósito	Convierte una cadena tipo CHAR o VARACHAR2 en un valor tipo DATE. El formato es la máscara de la fecha. Si se omite toma el formato por defecto. Si el formato es J para juliano char debe ser un entero. 'Parámetros_nls' tiene la misma función que para TO_CHAR en conversión a fechas. No utilizar con TO_DATE un valor tipo DATE como argumento. La fecha que devuelve puede tener un valor diferente que el char original, dependendo del formato indicado o el activo por defecto.
	Ejemplo	SELECT TO_DATE('January 15, 1997', ,'Month dd, YYYY') Hoy FROM DUAL Hoy ---------- January 15, 1997

	Sintaxis	TO_MULTI_BYTE (char)
TO_MULTI_BYTE	Propósito	Devuelve char con todos sus caracteres de byte simple convertidos en sus correspondientes caracteres multibyte. Solo se utiliza en B.D. que utilicen ambos tipos de juegos de caracteres.

	Ejemplo	SELECT TO_MULTI_BYTE('1234567') "Numero" FROM DUAL Numero ---------- 1234567
TO_SINGLE_BYTE	Sintaxis	TO_SINGLE_BYTE (char)
	Propósito	Devuelve char con todos sus caracteres multibyte convertidos en sus correspondientes caracteres de byte simple. Solo se utiliza en B.D. que utilicen ambos tipos de juegos de caracteres.
	Ejemplo	SELECT TO_SINGLE_BYTE('1234567') "Numero" FROM DUAL Numero ---------- 1234567
TO_NUMBER	Sintaxis	TO_NUMBER(char[,formato [,'parametros_nls']])
	Propósito	Convierte char de un valor CHAR o VARCHAR2 a un tipo de dato NUMBER con el formato especificado.
	Ejemplo	UPDATE emp SET sal = sal + TO_NUMBER('100.00','9G999D99') WHERE ename='BLAKE'

Otras funciones.

DUMP Sintaxis DUMP(expr, [,formato [,posic_comienzo [,long]]])

Propósito Devuelve un VARCHAR2 conteniendo el código del tipo de dato, la longitud en bytes y la representación interna del dato. El argumento formato indica la notación en la que se visualiza el resultado. Formato puede tener los siguientes valores:

8 Devuelve el resultado en notación octal.

10 Devuelve el resultado en notación decimal.

16 Devuelve el resultado en notación hexadecimal.

17 Devuelve el resultado en formato carácter simple.

Ejemplo

```
SELECT DUMP(ename, 8, 3, 2) "Ascii" FROM emp  WHERE ename='SCOTT'
          Ascii
          -------------------------
          Type=1 Len=5: 79,84
```

```
SELECT DUMP(ename, 16) "Hexad" FROM emp where empno=7894
          Hexad
          -------------------------
          Type=1 Len=5:4f,54
```

EMPTY_[B|C]LOB

Sintaxis EMPTY_[B|C]LOB()

Propósito Se muestra un localizador de LOB que se utiliza para inicializar el valor de una variable, o para inicializar una columna LOB en una sentencia UPDATE o INSERT y dejarla vacía. No se puede utilizar el localizador que devuelve como parámetro en el paquete DBMS_LOB.

Ejemplo

```
UPDATE tab_lob1
    SET col_clob = EMPTY_CLOB();
```

BFILENAME

Sintaxis BFILENAME('directorio','fichero')

Propósito Devuelve el localizar BFILE asociado con el fichero binario físico LOB que reside en el sistema de ficheros del S.O. El directorio es un objeto de la base de datos que identifica ficheros externos a dicha B.D. 'Fichero' es el nombre del fichero en el sistema operativo. Se debe asociar primero al BFILE con un fichero físico antes de realizar operaciones con SQL, PL/SQL, el paquete DBMS_LOB u OCI.

Ejemplo

```
INSERT INTO tbl_ficheros VALUES(BFILENAME('dir1_lob' 'image1.gif'));
```

GREATEST

Sintaxis GREATEST(expr [,expr2]....)

Propósito Devuelve el mayor valor de una lista de expresiones. Compara carácter a carácter. El tipo de dato es siempre VARCHAR2.

Ejemplo

```
SELECT GREATEST('GUTIEZ','GUTIERREZ') "Mayor" FROM DUAL;
          Mayor
          -----------------
              GUTIEZ
```

LEAST Sintaxis LEAST(expr [,expr2]...)

Propósito Devuelve menor valor de una lista de expresiones. Compara carácter a carácter. El tipo de dato es siempre VARCHAR2.

Ejemplo

```
SELECT LEAST('GUTIEZ','GUTIERREZ') "Menor"
    FROM DUAL;
          Menor
          -----------------
          GUTIERREZ
```

NLS_CHARSET_DECL_LEN

Sintaxis NLS_CHARSET_DECL_LEN(bytecnt, csid)

Propósito Indica la longitud de declaración (en número de caracteres) de una columna NCHAR. El parámetro bytecnt es la longitud de la columna. El parametro csid es el identificativo del juego de caracteres utilizado en la columna.

Ejemplo

```
SELECT NLS_CHARSET_DECL_LEN(200, nls_charset_id('jaa16eucfixed')) COLUMNA_NLS
FROM DUAL;
          COLUMNA_NLS
          -----------
                 100
```

NLS_CHARSET_ID

Sintaxis NLS_CHARSET_ID(texto)

Propósito Muestra el identificativo del juego de caracteres utilizado correspondiente a el juego indicado en texto. El parámetro texto es un VARCHAR2. Si se le indica como texto 'CHAR_CS' devuelve el número identificativo del juego de caracteres de la b.d. (servidor). Si se le indica como parámetro 'NCHAR_CS' muestra el número identificativo del juego de carácter nacional. Los juegos de caracteres no válidos devuelven NULL.

Ejemplo1

```
SELECT NLS_CHARSET_ID('char_cs') FROM DUAL;
```

```
          NLS_CHARSET_ID('CHAR_CS')
          -------------------------
                       2
```

Ejemplo

```
SELECT NLS_CHARSET_ID('nchar_cs') FROM DUAL;
          NLS_CHARSET_ID('NCHAR_CS')
          --------------------------
                       1001
```

NLS_CHARSET_NAME

Sintaxis NLS_CHARSET_NAME(n)

Propósito Muestra el juego de caracteres NLS correspondiente al número que se indica como parámetro. Si n no es un valor válido, se devuelve un nulo.

Ejemplo

```
SELECT NLS_CHARSET_NAME(2) FROM DUAL;
          NLS_CH
          ------
          WE8DEC
```

NVL

Sintaxis NVL(expr1 ,expr2)

Propósito Indica el valor que ha de tener expr1 cuando en su contenido aparezcan nulos. El tipo de dato que retorna expr2 es siempre el mismo que tenga expr1.

Ejemplo

```
SELECT codigo_prod, NVL(precio, 0) Precio FROM productos
          CODIGO_PROD    Precio
          -----------    -----------------
                10             40
                20             200
                30             0
                40             340
                50             0
```

VSIZE

Sintaxis VSIZE(expr)

Propósito Devuelve el número de bytes que ocupa la representación interna del dato.

Ejemplo

```
SELECT ename, VSIZE(ename) "Bytes" FROM emp
          Ename        Bytes
          ----------   ------
          PEPE              4
          LOPEZ             5
```

DECODE

Sintaxis Decode(expr, val1, cod1, ...valn, codn, codf)

Propósito Dentro de una expr evalúa los valores de la lista y los cambia por el código correspondiente.

Ejemplo

```
SELECT codigo_prod, DECODE(Codigo_prod,20,'Bebidas'
    ,40, 'Infantil','General') "Códigos"
FROM productos;
          CODIGO_PROD  Códigos
          -----------  -------------
                10     General
                20     Bebidas
                30     General
                40     Infantil
```

B

ANEXO B. El interfaz de usuario SQL*PLUS

Tabla de contenidos

Introducción **1**

El Editor de SQL*PLUS **1**

COMANDOS DE EDICION 1

El Comando SPOOL 3

El Comando SAVE 3

El Comando START 3

El Comando GET 4

El Comando EDIT 4

El Comando DESCRIBE 4

El Comando HOST 4

Parámetros de Sustitución **5**

Mejoras en SQL*Plus a partir de Oracle 10g **6**

Mejoras en el comando DESCRIBE 6

Mejoras en el comando SPOOL 6

Configuración de Sesión 6

Nuevas variables DEFINE 7

Variables de Sustitución en PROMPT 8

Compatibilidad SQL*Plus 8

Introducción

- SQL*Plus es una de las herramientas desde las que se ejecutan comandos SQL.
- Desde ciertas herramientas, solo están permitidos los comandos DML (Manipulación), en Plus, en cambio también se permiten como hemos visto, comandos DDL (definición).
- Desde SQL*PLUS se permite:
  - Trabajar con SQL.
  - Disponer de ayuda para comandos. Por ejemplo HELP SELECT.
  - Los comandos se ejecutan de forma inmediata, o bien, pueden ser almacenados en un buffer o en un fichero del S.O. y ser ejecutados a posteriori.
  - También se considera a PLUS una herramienta para generar informes sencillos y tabulares. Permite realización de cálculos y de rupturas.
  - Utilizar el editor propio de la herramienta (que es un editor de línea), o trabajar desde PLUS con el editor del sistema operativo.
- Para entrar en la herramienta SQL*PLUS ya hemos visto antes que hay que introducir el siguiente comando desde el indicador de Sistema operativo (en UNIX):

```
$ sqlplus [usuario/password]
```

- Si no indicamos el usuario/password en la línea de comandos, la propia herramienta lo preguntará.

```
$ sqlplus

Enter user-name:
Enter password:
```

El Editor de SQL*PLUS

- SQL*Plus dispone de editor de líneas. Este editor permite modificar, examinar, o reejecutar un comando SQL, sin tener que reescribirlo:
- Los comandos SQL*Plus deben introducirse después del prompt "SQL>" y SIN punto y coma al final.

COMANDOS DE EDICION

Operador	Reducido	Propósito
APPEND	A texto	Añadir texto al final de una línea
CHANGE	C/texto/	Elimina el texto de la línea.
CHANGE	C/antiguo/nuevo	Cambia el texto "antiguo" por el "nuevo" en la línea indicada

CLEAR BUFFER	CL BUFF	Borra todas las líneas del buffer.
DEL	(ninguna)	Si primero me posiciono en una línea con L4 y luego indico. DEL, sólo borra la línea sobre la que me he posicionado
INPUT	I	Añade una línea a partir del número de línea en que me encuentro.
LIST	L	Muestra todas las líneas del buffer
LIST n	L n	Sólo mostrará la línea "n"
LIST n m	L n m	Si se indica L "n" "m", muestra las líneas comprendidas entre ambos números n y m inclusive.
RUN	R	Reejecuta la sentencia actual

Ejemplos:

- Visualizar la línea uno del buffer:

```
SQL>  list 1
1*    SELECT dptno, dname
```

- Cambiar dptno por deptno.

```
SQL>  C/dptno/deptno
```

- Reejecutar el comando

```
SQL>  RUN
1     SELECT  deptno, dname
2     FROM    DEPT
3*    WHERE   deptno=10

DEPTNO     DNAME             LOC
------     -------------     ------------------
10         ACCOUNTING        NEW YORK
```

- Añadir una línea al final:

```
SQL>  LIST 3
3*    WHERE deptno=10
```

```
SQL>  I
4*    order by deptno (por ejemplo)
```

```
SQL>  LIST
    1     SELECT deptno, dname
    2     FROM DEPT
    3     WHERE deptno=10
    4*    order by deptno
```

- Añadir "desc" al final de la línea 4

```
    SQL>  L 4
    4*    order by deptno
```

```
    SQL>  AP  desc
    4*    order by deptno des
```

El Comando SPOOL

- El comando SPOOL se utiliza para almacenar el RESULTADO de la ejecución de una select en un fichero:

```
    SQL>  SPOOL  nombre_fichero
    SQL>  run
    SQL>  SPOOL  OFF;
```

- Esto genera un fichero con una extensión por defecto .lst en la que esta el resultado de la select ejecutada.
- Enviar a la impresora:

```
    SQL>  SPOOL  on
    SQL>  run
    SQL>  SPOOL  Out;
```

El Comando SAVE

El comando SAVE guarda en un fichero el CONTENIDO de la sentencia SQL (solo una) actualmente en el buffer:

```
    SQL>  SELECT * FROM dept
    SQL>  Save departamentos
```

Esto crea un fichero de sistema operativo llamado departamentos.sql con la select escrita anteriormente.

El Comando START

Ejecuta el fichero de sistema operativo que se indique:

```
SQL>  start departamentos
    SQL>  select * from dept;

    DEPTNO     DNAME              LOC
    ------     -------------      --------------------
    10         ACCOUNTING         NEW YORK
    20         RESEARCH           DALLAS
    30         SALES              CHICAGO
```

```
40         OPERATIONS         BOSTON
```

Si el fichero tiene una extensión distinta a "sql" hay que indicarlo.

El comando START me permite ejecutar un fichero de comandos creado previamente y que contenga más de una sentencia SQL.

El Comando GET

- El comando get, lleva al buffer SQL el fichero indicado, por ejemplo, para modificarlo. Si la extensión del nombre del fichero a editar no es SQL hay que indicar el nombre completo.

```
SQL>  get departamentos[.extension]
SQL>  select * from dept;
SQL>  i
2*    order by dname
SQL>  SAVE departamentos [REPLACE]
```

- La opción replace del comando SAVE reemplaza el fichero antiguo por el nuevo.

El Comando EDIT

- Llama al editor activo del sistema operativo. Se puede elegir el editor deseado utilizando:

```
SQL> define_editor= vi
```

- Esto implica que cuando el usuario ejecute:

```
SQL> ed
```

Se invoca el editor "vi" del sistema operativo.

- Si después de "ed" no se indica un nombre de fichero se creará un fichero temporal llamado "afiedt.buf". Si después de "ed" se le especifica un nombre, genera en el sistema operativo un fichero con ese nombre y una extensión .sql.

El Comando DESCRIBE

- El comando DESCRIBE o DESC muestra la estructura de una tabla:

```
SQL>  DESC dept

Name                    Null              Type
------------------     ---------     --------------
DEPTNO                  NOT NULL         NUMBER(2)
DNAME                                    VARCHAR2(14)
LOC                                      VARCHAR2(13)
```

El Comando HOST

- El comando HOST permite ejecutar un comando de sistema operativo sin salir de la sesión de SQL*Plus. En algunos sistemas operativos su equivalente es el signo de cerrar admiración (UNIX), y en otros el símbolo del dólar (NT). Veamos un ejemplo a continuación utilizando las tres posibilidades.

```
SQL> HOST ls -l

SQL> ! ls -l

SQL> $ DIR /s
```

- Si se ejecuta solo HOST, se sale temporalmente al sistema operativo. Para regresar a SQL*Plus, basta utilizar el comando EXIT que nos devuelve a la sesión de base de datos.

Parámetros de Sustitución

- Se pueden incluir variables de sustitución en un programa en SQL*Plus:

  Por ejemplo:

```
SQL>  SELECT * FROM dept
2     WHERE    deptno = &nodepartamento;

Enter value for nodepartamento: 10

DEPTNO     DNAME              LOC
------     -------------      -------------------
10         ACCOUNTING         NEW YORK
```

- Si salvamos el fichero, luego podemos ejecutarlo pasando el valor de la variable que el programa espera, en el momento de invocarlo:

```
SQL>  save departamentos
SQL>  start departamentos 10
```

- Se puede guardar, utilizando un doble ampersand "&&" el contenido de una variable durante la sesión para utilizarse en otra sentencia SQL. Por ejemplo

```
SQL>  SELECT * FROM dept
2     WHERE    deptno = &&nodepartamento;
Enter value for nodepartamento: 10

DEPTNO     DNAME              LOC
------     -------------      -------------------
10         ACCOUNTING         NEW YORK

SQL>  SELECT * FROM EMP
2     WHERE    deptno = &nodepartamento;
```

- En esta ocasión no pregunta por el valor de la variable, porque se mantiene al haberla definido con doble ampersand.

```
EMPNO ENAME          JOB            MGR     HIREDATE  SAL  COMM DEPTNO
----- -------------- ----------     ----    --------- ---- ---- ------
7782  CLARK          MANAGER        7839    09-JUN-81 2450        10
7739  KING           PRESIDENT              17-NOV-81 5000        10
7934  MILLER         CLERK          7782    23-JAN-82 1300        10
```

Mejoras en SQL*Plus a partir de Oracle 10g

Mejoras en el comando DESCRIBE

El comando SQL* Plus DESCRIBE se ha mejorado, y ahora puede describir reglas, conjuntos de reglas y contextos de evaluación de reglas. Esto hace mucho más fácil mostrar las especificaciones de reglas comparado con las consultas a las vistas del diccionario USER_RULES o USER_RULE_SET_RULES.

Las reglas, conjuntos de reglas y contextos de evaluación de reglas se adhieren a una topología específica. Un conjunto de reglas es un grupo de reglas relacionadas agrupadas conjuntamente, y una regla puede ser parte de uno o varios conjuntos de reglas.

Debido a ello, el objeto regla descrito puede ser de nivel superior o no. Si es un objeto de nivel superior, como un conjunto de reglas, debe contener la lista de reglas y un contexto de evaluación.

El comando DESCRIBE podrá mostrar los datos de detalle mediante técnicas drill-down. La sintaxis permanece invariable:

```
DESC[RIBE] {[esquema.]objeto[@connect]}
```

Mejoras en el comando SPOOL

El comando SPOOL ha sido mejorado, y permite añadir a un fichero, o reemplazar un fichero existente. En versiones anteriores solo era posible utilizar SPOOL para crear (y reemplazar) un fichero. La opción por defecto es REPLACE. La nueva sintaxis es:

```
SPOOL {file_name[.ext] [CRE[ATE]] | [REP[LACE]] | [APP[END]] | OFF | OUT}
```

Es posible generar un fichero HTML válido utilizando uno o varios comandos SPOOL APPEND, construyendo manualmente las cabeceras y pies de página.

```
SPOOL fichero.htm
PROMPT <HTML><BODY>
SET MARKUP HTML ON
SELECT * FROM departments;
SET MARKUP HTML ENTMAP OFF
PROMPT </BODY></HTML>
SPOOL OFF
```

Configuración de Sesión

Es posible utilizar ficheros de perfiles para controlar el entorno SQL*Plus:

- El fichero de sistema GLOGIN.SQL permite definir un entorno común a todos los usuarios de una base de datos particular.
- El fichero de usuario LOGIN.SQL permite personalizar una sesión específica.

En Oracle, el fichero de perfil de sistema se ejecuta tras cada conexión satisfactoria, ya sea mediante SQLPLUS o mediante el comando CONNECT. El fichero de usuario se ejecuta tras el fichero de sistema.

El fichero de perfil del sistema contiene fundamentalmente asignaciones de variables de entorno, mientras que el fichero de configuración de usuario puede contener comandos SQL*Plus, comandos SQL o incluso bloques PL/SQL.

Por defecto, el fichero de perfil de sistema está localizado en el directorio $ORACLE_HOME\sqlplus\admin. SQL*Plus busca el fichero de perfil de usuario en el directorio actual, y si no en algún directorio de la variable de entorno SQLPATH.

Es posible utilizar la opción SQLPLUSCOMPATIBILITY para forzar compatibilidad hacia atrás. Si toma un valor menor que 10, los ficheros glogin.sql y login.sql no se leerán tras los comandos CONNECT sucesivos.

Nuevas variables DEFINE

El comando DEFINE incorpora tres nuevas variables _DATE, _PRIVILEGE y _USER. Estas tres nuevas variables tienen valores predefinidos. Adicionalmente _DATE puede tomar como valor una cadena de texto definida por el usuario.

- _DATE – es una variable dinámica y muestra la fecha actual.

El formato de fecha se especifica en NLS_DATE_FORMAT. Es posible establecer un valor diferente para _DATE mediante un comando DEFINE.

```
SQL> SET SQLPROMPT _DATE>
09/03/05>DEFINE _DATE='01/01/2009'
01/01/2009>
```

Es posible reactivar el comportamiento dinámico por defecto mediante:

```
01/01/2004>DEFINE _DATE=''
19/12/2008>
```

- _PRIVILEGE – indica el nivel de privilegio de la conexión actual mediante los valores AS SYSDBA, AS SYSOPER o mediante una cadena vacía para conexiones normales, o cuando no hay conexión.

```
SQL>conn / as sysdba
Connected.
SQL>SET SQLPROMPT _PRIVILEGE>
AS SYSDBA>
```

- _USER – contiene el nombre de usuario de la sesión actual.

```
SQL>CONN scott/tiger
Connected.
SQL>SET SQLPROMPT _USER>
SCOTT>
```

Pueden ser consultadas y modificadas al igual que cualquier otra variable DEFINE, y pueden utilizarse en TTITLE, como variables de sustitución '&' o como prompt de línea de comandos SQL*Plus. Pueden utilizarse, por ejemplo, para modificar el SQLPROMPT en los ficheros de inicialización GLOGIN.SQL y LOGIN.SQL.

Variables de Sustitución en PROMPT

Es posible utilizar variables en el prompt SQL*Plus, permitiendo la sustitución de valores de las variables en tiempo de ejecución. Es posible utilizar las variables predefinidas _USER, _DATE y _PRIVILEGE así como cualquier otra variable definida.

Dichas variables no tienen porqué venir precedidas por un ampersand &, y actúan como cualquier otra variable DEFINE, pudiendo ser consultadas y modificadas igualmente.

- No es necesario prefijarlas con &
- Pueden ser consultadas y modificadas como otras variables DEFINE.

Algunos ejemplos de variables de sustitución pueden ser:

- Predefinidas

```
SQL>SET SQLPROMPT '_USER  _DATE>'
SCOTT  09/03/05>ALTER SESSION SET NLS_DATE_FORMAT = 'HH24:MI:SS';
Session altered.
SCOTT  16:24:44>
```

- No Predefinidas

```
SQL>DEFINE con=PEDRO
SQL>SET SQLPROMPT 'con>'
PEDRO>
```

Compatibilidad SQL*Plus

SQL*Plus tiene una nueva opción de línea de comandos, llamada COMPATIBILITY. Esta opción nueva permite establecer la variable de sistema SQLPLUSCOMPATIBILITY a una versión de SQL*Plus especificada a continuación, mediante la sintaxis:

```
SQLPLUS [-C[OMPATIBILITY]] {X.Y[.Z]}
```

La opción de línea de comandos permite establecer la variable de sistema antes de ejecutar los ficheros de configuración de sistema o de usuario.

También es posible especificar el valor mediante el comando SQL*Plus:

```
SET SQLPLUSCOMPATIBILITY {X.Y.[Z]}
```

El ejemplo siguiente muestra cómo se evalúan las variables de sustitución en función del valor de la variable de sistema SQLPLUSCOMPATIBILITY.

```
SQL>SET SQLPLUSCOMPATIBILITY 9.2.0
SQL>SET SQLPROMPT '_USER  _DATE>'
_USER  _DATE>SET SQLPLUSCOMPATIBILITY 10.1
SCOTT  16:32:28>
```

Ejercicios Lenguaje SQL.

Tabla de contenidos

Modelo de datos **1**

Ejercicios sobre Select **1**

Select Simples 1

Select Simples con Funciones 2

Subconsultas/joins 4

Ejercicios DML **5**

Insert 5

Update 5

Delete 5

Control de transacciones. **6**

Ejercicios ddl e integridad de datos. **6**

Práctica DDL-DML 6

Implementación de restricciones de integridad durante la creación de una tabla 8

Implementación de restricciones de integridad después de creación de una tabla 9

Utilización de la tabla de excepciones 9

Creación de Vistas 9

Creación de Sinónimos 9

Soluciones **9**

Select Simples 9

Select Simples con Funciones 12

Subconsultas / joins 14

Insert 16

Update 16

Delete 17

Práctica DDL-DML 17

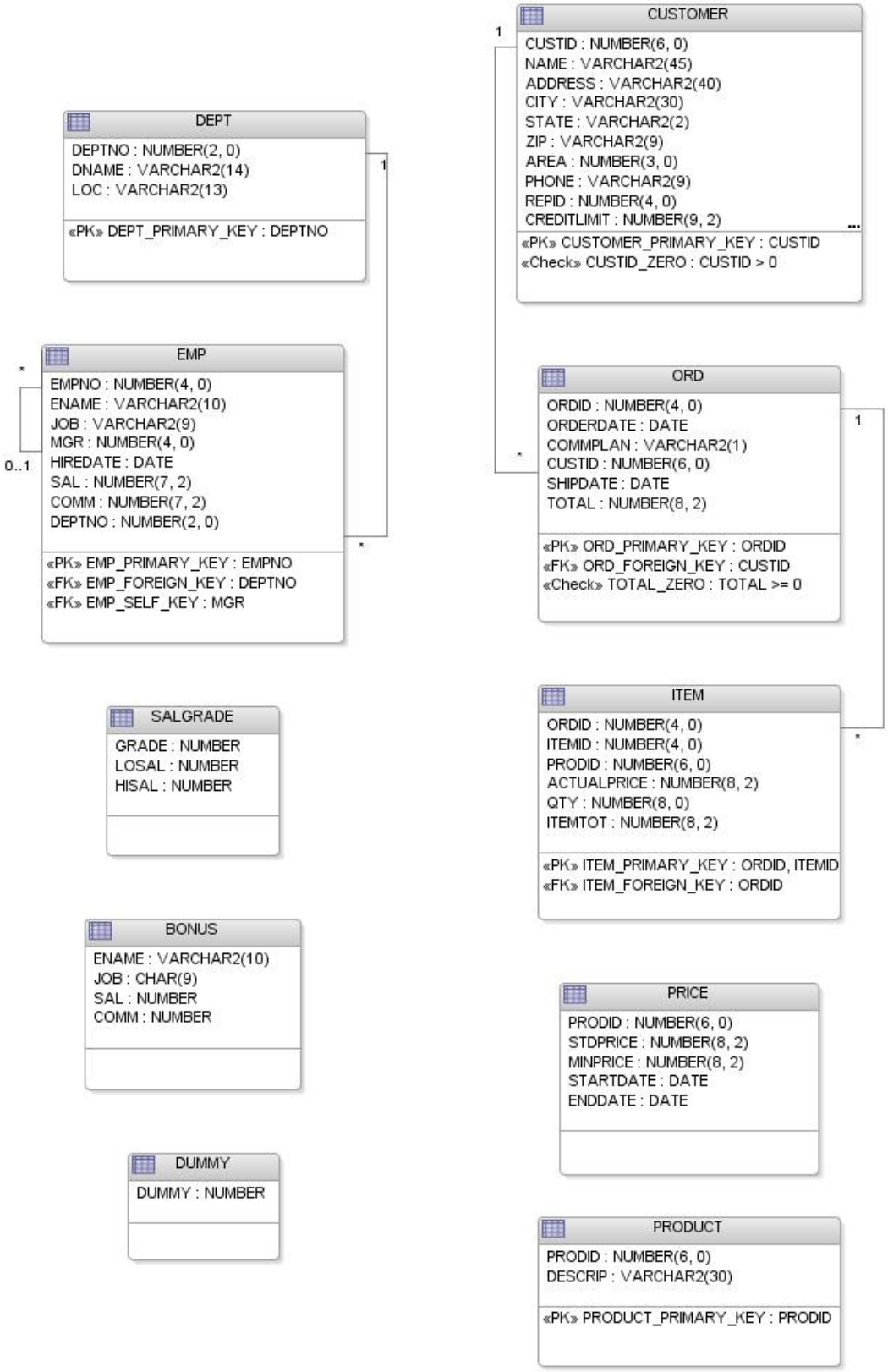
Implementación de restricciones de integridad durante la creación de una tabla 21

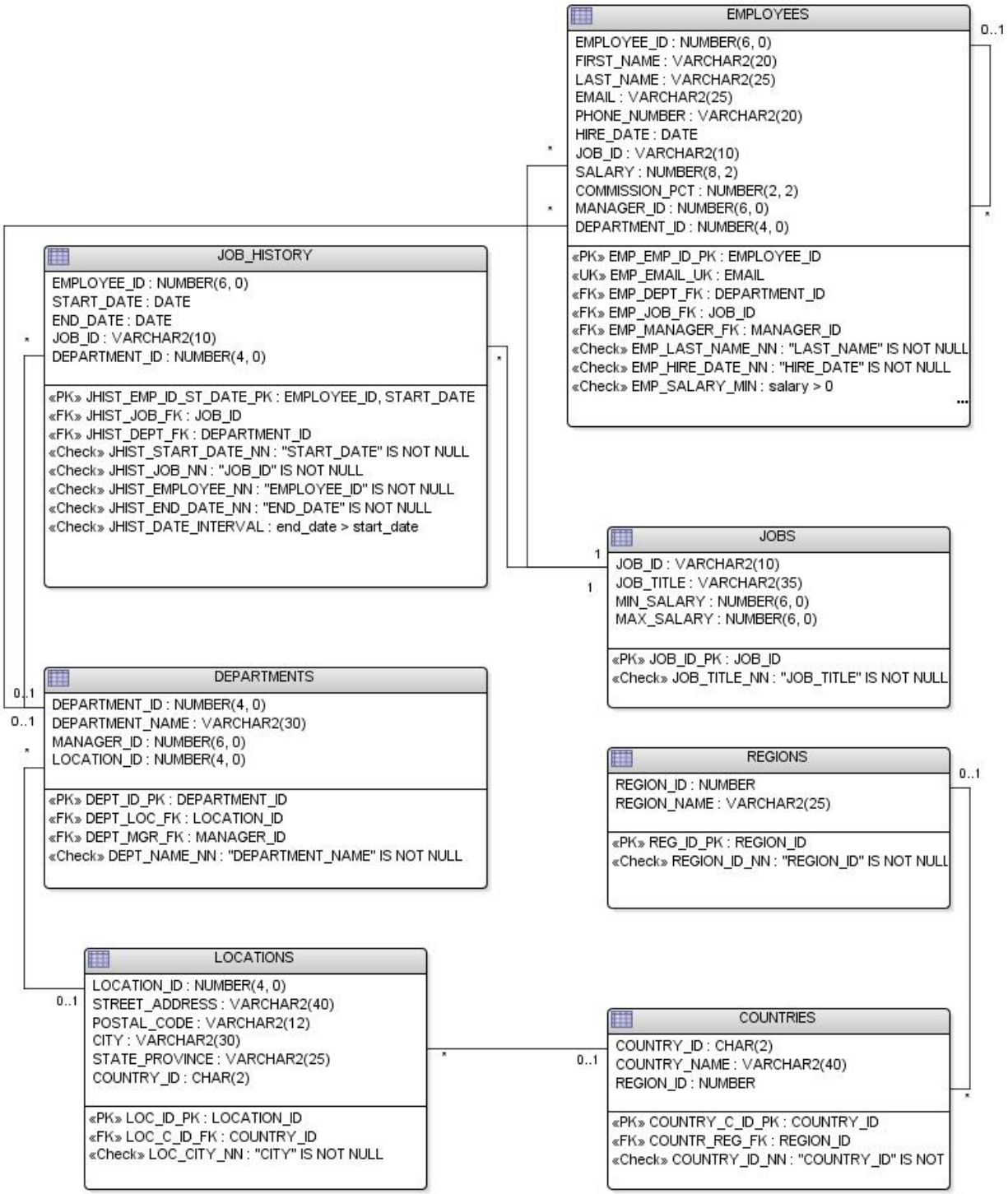
Utilización de la tabla de excepciones 21

Vistas 22

Sinónimos 22

Modelo de datos





Ejercicios sobre Select

Select Simples

1. Describir la tabla de empleados.

2. Nombre, apellido, salario de los empleados ordenado por el apellido en orden ascendente.

3. Nombre, apellido, salario de los empleados ordenado por salario ascendente, nombre en orden descendente.

4. Mostrar los códigos de trabajos distintos de los empleados que existen en el departamento 30.

5. Número de empleado, nombre, apellido, número de departamento de los empleados cuyo nombre empieza con 'J' y su apellido por 'L', ordenados por número de departamento.

6. Número de empleado, nombre, apellido, número de departamento de los empleados cuyo nombre NO empieza con 'J' y contiene la cadena 'en' en su apellido, ordenados por nombre.

7. Número de empleado, nombre, apellido, número de departamento de los empleados que no cobran comisión.

8. Mostrar la fecha actual.

9. Nombre, número de departamento de los empleados que su oficio empiece por "Ad".

10. Nombre y apellido de los empleados de algún departamento (variable de sustitución). Utilizar como etiquetas de columna las palabras "Nombre" y "Apellido".

11. Nombre, apellido y número de días que llevan trabajando en la empresa.

12. Nombre de los empleados cuyo nombre acabe en 'n'.

13. Nombre, apellido, junto con su salario incrementado un 7,5%.

14. Nombre, apellido, salario, comisión de los que cobran comisión;

15. Mostrar el código de la ciudad de Oxford.

16. Mostrar los datos de los departamentos que se encuentran en la localidad 2500.

17. Mostrar un listado llamado “Empleados”, de la siguiente forma “Número de empleado. Apellido, Nombre”.

18. Dirección, código postal, nombre de ciudad. De todas las ciudades del estado de Washington.

19. Nombre, salario, comisión, fecha de alta, número de departamento de los empleados que trabajan en los departamentos 50, 80 y 100. Su salario está entre 3000 y 10000 incluidos, o no tienen comisión. Además, haber entrado a trabajar desde el 2000. Ordenados por fecha de alta ascendentemente y salario descendentemente.

20. Mostrar nombre del propietario y nombre de las tablas a las que tiene algún permiso.

21. Nombre del usuario con el que está conectado.

22. Averiguar si “Himuro” es nombre o apellido, mostrando un listado de nombres y apellidos de los empleados, a los cuales les coincida el nombre o el apellido con dicha palabra.

23. Listar el nombre de vistas del diccionario de datos que contengan la palabra “TAB”.

Select Simples con Funciones

1. Mostrar en una sola columna: los nombres de los empleados del departamento 30 y el número de meses que llevan trabajando para la empresa. Los nombres alineados a la izquierda y el número de meses a la derecha.

2. Mostrar el nombre del empleado y la posición en la que aparece la primera letra “E” en dicho nombre, para aquellos empleados cuya comisión sea nula.

3. Mostrar el nombre del empleado y su fecha de alta en la empresa con formato Día (nombre del día) / Mes (nombre del mes) / YYYY (cuatro dígitos del año) que desempeñen el trabajo de “IT_PROG”.

4. ¿Qué día de la semana naciste?

5. Mostrar el nombre del empleado y el número de su departamento. Si es el 10 debe aparecer el literal “Contabilidad”, si es el 20 “Investigación”, si es el 30 “Ventas” y si es el 40 “Operaciones”.

6. Mostrar las tres primeras letras del nombre del empleado en mayúsculas y una columna que contenga el salario multiplicado por la comisión cuya cabecera sea “Bono”, si el empleado no tiene comisión asignarle 0 en la columna ‘Bono’.

7. Mostrar el salario medio de aquellos empleados cuyo trabajo sea el de “IT_PROG”.

8. Encontrar, entre los empleados, el salario más alto, el más bajo y la diferencia entre ambos.

9. Hallar el número de trabajos distintos que existen en el departamento 30.

10. Encontrar que departamentos tienen al menos dos trabajos distintos.

11. Encontrar el número de trabajos diferentes, que se desempeñan en el departamento 30, para aquellos empleados cuyo salario se comprende entre 1.000 y 7.800$.

12. Contar el número de personas que realizan un trabajo determinado, dentro de cada departamento.

13. Nombre de los empleados que su nombre sea único.

14. Mostrar del salario; la suma, media, máximo y mínimo. Todos éstos valores con separadores de miles y redondeados a dos dígitos. Además el número de empleados total y el número de empleados que se les paga un salario. Condicionado para un número de departamento.

15. Número de empleados que fueron dados de alta un sábado y los que fueron dados de alta un domingo.

16. Listar el nombre, apellido de los empleados junto con el literal correspondiente al salario. “Bajo” menor que 3000, “Medio” de 3000 a 8000, “Alto” mayor a 8000. Además mostrar un literal para el departamento, si es el 90 “Administrativo”, 80 “Ventas” y el resto solo mostrar el número.

Subconsultas/joins

1. Nombre de los empleados; número y nombre de departamento; número y los tres primeros caracteres de la localidad donde se encuentra el departamento. Para aquellos empleados que trabajen en el departamento 20 o que sean vendedores (que su código de trabajo comience por ‘SA’).

2. Nombre del empleado, identificador de trabajo, el nombre y código del departamento en el que trabaja.

3. En la consulta anterior existen departamentos que no tienen empleados. Mostrar también esta información.

4. Nombre completo (en una sola columna) del empleado que más gana.

5. Nombre, código de trabajo y el salario de todos los empleados que tienen un salario superior al salario más bajo del departamento 30. Ordenar la recuperación por orden descendente de salarios.

6. Nombre y apellido de los trabajadores que tienen por jefe a ‘King’.

7. Nombre del día de la semana en el que más empleados entraron a la empresa.

8. Nombre, trabajo (title), salario y fecha de alta de aquellos empleados que cobren más que cualquier vendedor (job_id ‘SA…’). Ordenar la salida por orden descendente de salarios.

9. Nombre, departamento (name), salario y trabajo (title) de aquellos empleados que trabajan en el mismo departamento que ‘Irene’, y cuyo salario es menor a la media de los salarios del departamento 10.

10. Nombre del empleado y la localidad donde trabaja, para los que tengan comisión o que trabajen en el mismo departamento que 'Irene'.

11. Recuperar los empleados cuyos salarios exceden la media de sus respectivos departamentos.

12. Recuperar el empleado que más tarde entró para cada uno de los departamentos existentes en la empresa.

Ejercicios DML

Insert

1. Dar de alta un departamento llamado 'Ayuda Social', cuyo jefe será el jefe del departamento 'Finance'.
2. Insertar los siguientes datos en las tablas correspondientes.
3. Un empleado para la ciudad de 'Madrid', que trabaje en el departamento de 'Formación' y su trabajo sea 'Formador Oracle'.

Update

1. Establecer al empleado 'Jones' trabajo IT_PROG, en el departamento 20 y con un incremento del 13% de su salario, (suponemos un único Jones).

2. Dejar la fecha de alta del empleado igual a la fecha del sistema para aquellos empleados de la ciudad con código 1500 y cuyo salario sea superior a la media de los salarios de analistas de la empresa.

3. Actualizar el salario de los empleados que trabajen en el departamento donde trabaja 'Matos'. Incrementarles el salario en un 15% respecto a la media de los salarios del departamento 20.

Delete

Borrar los países y regiones que estén sin relacionar.

Control de transacciones.

Si las inserciones, modificaciones y borradores se han realizado correctamente, validar las sentencias anteriores.

Ejercicios ddl e integridad de datos.

Práctica DDL-DML

1. Se desea tener información de cursos que se impartirán de forma gratuita. Por lo tanto solo se permite que una persona reciba un solo curso.
2. De las personas que se inscriban en los cursos se almacenará su dni, nombre, apellidos, dirección, sexo, fecha de nacimiento.
3. Un curso se identifica por un código numérico (inferior a 3 dígitos), se almacenará un nombre de curso, la fecha de inicio y fin. Un determinado curso lo imparte sólo un profesor. Tendrá un número máximo de alumnos (inferior a 3 dígitos) y el número de horas (inferior a 4 dígitos).
4. Un profesor puede impartir varios cursos, se almacenará su dni, nombre, apellidos, dirección, titulo y lo que gana por hora (inferior a 3 dígitos).
5. Generar las siguientes tablas.

ALUMNOS		CURSOS		PROFESORES	
NOMBRE	VARCHAR2(30)	NOMBRE_CURSO	VARCHAR2(40)	NOMBRE	VARCHAR2(30)
APELLIDO1	VARCHAR2(20)	COD_CURSO	NUMBER(2)	APELLIDO1	VARCHAR2(20)
APELLIDO2	VARCHAR2(20)	NIF_PROFESOR	VARCHAR2(9)	APELLIDO2	VARCHAR2(20)
NIF	VARCHAR2(9)	MAXIMO_ALUMNOS	NUMBER(2)	NIF	VARCHAR2(9)
DIRECCIÓN	VARCHAR2(40)	FECHA_INICIO	DATE	DIRECCIÓN	VARCHAR2(40)
SEXO	VARCHAR2(1)	FECHA_FIN	DATE	TITULO	VARCHAR2(30)
FECHA_NACIMIENTO	DATE	NUM_HORAS	NUMBER(3)	GANA	NUMBER(4,2)
CURSO	NUMBER(2)				

Establecer las siguientes restricciones:

Identificamos las tuplas de las tablas; CURSOS mediante el atributo CODIGO, PROFESORES y ALUMNOS usando el DNI.

La información del número de horas del curso es imprescindible para almacenarlo.

No es posible dar de alta un alumno si no se matricula en un curso.

El campo GANA de la tabla PROFESORES no puede estar en ningún caso vacío.

Dos cursos no pueden llamarse igual, lo mismo le pasa a los profesores (nombre completo).

Cumplir con la relación normal entre fecha comienzo y fecha fin (orden cronológico).

Los valores para el atributo sexo son sólo M o F (en mayúsculas).

Se ha de mantener la regla de integridad referencial.

NOTA: Algunos ejercicios provocan errores, intentar corregir uno a uno para ver el tipo y mensaje de error para corregir. (Dudas, preguntar al formador)

Insertar las siguientes tuplas:

Profesores

NOMBRE	APELLIDO1	APELLIDO2	NIF	DIRECCION	TITULO	GANA
Miguel	Torres	García	12345678X	Argentina, 6	Programador	40
Lucia	Benítez	Suarez	13334444A	Av. La Paz, 20.	Ing. Sistemas.	58

Cursos

NOMBRE	COD_CURSO	NIF_PROF	MAX_ALU	FECHA_INI	FECHA_FIN	NUM_HORAS
SQL	0	12345678X	15	05-MAY-11	23-JUN-11	120
SQL Tuning	1	13334444A			01-09-11	80
PLSQL	2	12345678X	12			100
Administración I	3	33334444A	10	12-06-11	31-06-11	100

Alumnos

NOMBRE	APELLIDO 1	APELLIDO 2	NIF	DIRECCION	SEXO	FECHA_NAC	CURSO
Lucas	Salas	García	12345678X		V		1
Lucia	Benítez	Suarez	13334444A	Av. La Paz, 20.	F	23-FEB-1969	
Antonia	Reyes	Pérez	87654321A	Canadá, 3.	M		0

Manuel	Guerra	Toro	123123123 L	Canadá, 3.			1

Insertar la siguiente tupla en ALUMNOS:

NOMBRE	APELLIDO 1	APELLIDO 2	NIF	DIRECCION	SEXO	FECHA_NAC	CURSO
Sergio	Aguirre	Moscoso	12345678 X		m		

Añadir el campo edad de tipo numérico a la tabla PROFESORES.

Añadir las siguientes restricciones:

La edad de los profesores está entre 18 y 65 años, incluidos.

Ningún curso su número máximo de alumnos será menor que 10.

A partir de ahora el número de horas de los cursos debe ser mayor que 100.

Eliminar la restricción que controla que los cursos no pueden llamarse igual.

Eliminar la restricción NOT NULL del atributo GANA.

La fecha de nacimiento de la alumna Antonia Reyes está equivocada. La verdadera es 23 de diciembre de 1949.

Cambiar a Antonia Reyes al curso de código 5.

Eliminar la profesora Laura Jiménez.

Cambiar la clave primaria de PROFESORES al nombre y apellidos.

Implementación de restricciones de integridad durante la creación de una tabla

Crear las siguientes tablas, implementar las siguientes reglas:

TABLA HABITACIONES

```
Nº HABITACIÓN    NUMBER(2)    Clave primaria
TIPO             VARCHAR2(15) Check
PRECIO           NUMBER(9)
```

TABLA CLIENTES

```
Nº CLIENTE       NUMBER(4)    Clave primaria
NIF              NUMBER(8)    Clave única
NOMBRE_CLIENTE   VARCHAR2(90)
NVISA            NUMBER(4)
TELÉFONO         NUMBER(9)
```

TABLA RESERVAS

```
Nº RESERVAS      NUMBER(2)    Clave primaria
Nº CLIENTE       NUMBER(4)    Foreign Key (tabla clientes)
Nº HABITACION    NUMBER(2)    Foreign Key (tabla Habitaciones)
PAGAGO           CHAR(2)      Check
```

Implementación de restricciones de integridad después de creación de una tabla

TABLA HABITACIONES

```
Nº HABITACION          Primary key
TIPO                   Solo recogerá los valores de Simple o doble.
```

TABLA CLIENTES

```
Nº CLIENTE              Primary key
NIF                    Clave única.
```

TABLA RESERVAS

```
Nº RESERVAS      Primary key
Nº CLIENTE       Foreign key. Este campo está relacionado con tabla CLIENTES.
Nº HABITACIÓN    Foreign Key. Este campo está relacionado con HABITACIONES.
PAGADO           Solo admitirá los valores de   ‘s’,’n’ (‘ si ‘o ‘no’)
```

Utilización de la tabla de excepciones

Desactivar la restricción PRIMARY KEY para el campo NÚMERO DE CLIENTES

Insertar un nuevo número de cliente duplicado.

Crear la tabla exceptions mediante la ejecución del fichero utlexcpt.sql

Intentar activar la restricción primary key insertando las filas que no cumplan la restricción en la tabla de excepciones.

Depurar la información eliminando duplicados.

Intentar activar la restricción.

Creación de Vistas

Crear una vista ‘EMPLEADOS’ que muestre: número de empleado; ‘apellido, nombre’ llamada FULL_NAME; email; teléfono; nombre del departamento en el que trabaja; dirección; ciudad; provincia; código postal y nombre del país.

Creación de Sinónimos

Crear sinónimos públicos en español para las tablas: Employees, Departments, Locations, Jobs, Countries, Regions.

Soluciones

Select Simples

Solución al ejercicio 1

```
desc       employees
```

Solución al ejercicio 2

```
select     first_name, last_name, salary
from       employees
order by   last_name;
```

Solución al ejercicio 3

```
select     first_name, last_name, salary
from       employees
order by   salary, first_name desc;
```

Solución al ejercicio 4

```
select     distinct job_id
from       employees
where      department_id=30;
```

Solución al ejercicio 5

```
select     employee_id,first_name,last_name, department_id
from       employees
where      first_name like 'J' and last_name like 'L%'
order by   department_id;
```

Solución al ejercicio 6

```
select     employee_id,first_name,last_name, department_id
from       employees
where      first_name not like 'J' and last_name like '%en%'
order by   2;
```

Solución al ejercicio 7

```
select     employee_id,first_name,last_name, department_id
from       employees
where      commission_pct is null;
```

Solución al ejercicio 8

```
Select     sysdate from dual;
```

Solución al ejercicio 9

```
Select     first_name,department_id
from       employees
where      job_id like 'Ad%';
```

Solución al ejercicio 10

```
Select     first_name Nombre,last_name Apellido
from       employees
where      department_id=&num_dep;
```

Solución al ejercicio 11

```
Select     first_name, last_name, sysdate-hire_date “Dias Trabajados”
from       employees;
```

Solución al ejercicio 12

```
Select     first_name
from       employees
where      first_name like ‘%n’;
```

Solución al ejercicio 13

```
select     first_name, last_name, salary*1.75
from       employees;
```

Solución al ejercicio 14

```
select     first_name, lart_name, salary, commission_pct
from       employees
where      commission_pct is not null;
```

Solución al ejercicio 15

```
select     location_id
from       locations
where      city=’Oxford’;
```

Solución al ejercicio 16

```
SELECT     *
FROM       departments
WHERE      location_id=2500;
```

Solución al ejercicio 17

```
SELECT     employee_id||’. ‘||last_name||’, ‘||first_name
FROM       employees;
```

Solución al ejercicio 18

```
select     street_address, postal_code, city
from       locations
where      state_province=’Washington’;
```

Solución al ejercicio 19

```
select     first_name, salary, commission_pct, hire_date, department_id
from       employees
where      department_id in (50,80,100)
and (salary between 3000 and 10000
or commission_pct is null)
and hire_date between ‘01/01/2000’ and sysdate
order by   4, 2 desc;
```

Solución al ejercicio 20

```
select     owner, table_name
from       all_tables;
```

Solución al ejercicio 21

```
select     user from dual;
```

Solución al ejercicio 22

```
select     first_name, last_name
from       employees
where      'Himuro' in (first_name, last_name);
```

Solución al ejercicio 23

```
select     table_name
from       DICT
where      table_name like '%TAB%';
```

Select Simples con Funciones

Solución al ejercicio 1

```
SELECT     rpad(last_name,15) ||
lpad(ROUND(MONTHS_BETWEEN (SYSDATE, hire_date)),6)
"Meses Transcurridos"
FROM       employees
WHERE      department_id = 30;
```

Solución al ejercicio 2

```
SELECT     first_name "Nombre", INSTR(UPPER(first_name), 'E', 1) "Posicion
primera letra E"
FROM       employees
WHERE      Commission_pct IS NULL;
```

Solución al ejercicio 3

```
SELECT     first_name "Nombre", last_name "Apellido",
           TO_CHAR (Hire_date, 'DAY/MONTH/YYYY') "Fecha de Alta"
FROM       employees
WHERE      job_id = 'IT_PROG';
```

Solución al ejercicio 4

```
SELECT     TO_CHAR(TO_DATE('30 junio 1966','DD/MM/YYYY'),'DAY')
FROM       DUAL;
```

Solución al ejercicio 5

```
SELECT     first_name "Nombre", department_id "Departamento",
           DECODE (department_id, 10, 'Contabilidad', 20,
           'Investigación', 30, 'Ventas', 40, 'Operaciones')  "Nombre Depto"
FROM       employees;
```

Solución al ejercicio 6

```
SELECT     UPPER(SUBSTR(first_name,1,3)) "Nombre",
Salary*NVL(Commission_pct, 0) "Bono"
FROM       employees;
```

Solución al ejercicio 7

```
SELECT     AVG(salary) "Media Salario"
FROM       employees
WHERE      job_id = 'IT_PROG';
```

Solución al ejercicio 8

```
SELECT     MAX (Salary) "Salario Máximo",
           MIN (Salary) "Salario Mínimo",
            MAX (Salary) - MIN (Salary) "Diferencia"
FROM       employees;
```

Solución al ejercicio 9

```
SELECT     COUNT(DISTINCT(Job_id)) cantidad
FROM       employees
WHERE      department_id = 30 ;
```

Solución al ejercicio 10

```
SELECT     department_id,COUNT(DISTINCT(Job_id)) "cantidad"
FROM       employees
GROUP BY   department_id
HAVING     COUNT(DISTINCT(Job_id))>=2;
```

Solución al ejercicio 11

```
SELECT     COUNT(DISTINCT Job_id) “Trabajos”
FROM       employees
WHERE      department_id = 30 AND
           salary BETWEEN 1000 AND 7800;
```

Solución al ejercicio 12

```
SELECT     department_id, job_id, COUNT(*) “Personas”
FROM       employees
GROUP BY   department_id, job_id;
```

Solución al ejercicio 13

```
select     first_name
from       employees
group by   first_name
having     COUNT(employee_id)=1;
```

Solución al ejercicio 14

```
select     to_char(sum(salary),'999g999G999d99') Suma,
to_char(avg(salary),'999g999G999d99') Media,
to_char(max(salary),'999g999G999d99') Maximo,
to_char(min(salary),'999g999G999d99') Minimo,
```

```
count(*) “Total Empleados”,
count(salary) “Emp Asalariados”
from       employees
where      department_id=&d;
```

Solución al ejercicio 15

```
select     to_char(hire_date,’Day’) Dia,count(*) Empleados
from       employees
where      rtrim(to_char(hire_date,’Day’)) in (‘Sábado’,’Domingo’)
group by   to_char(hire_date,’Day’);
```

Solución al ejercicio 16

```
SELECT  first_name,
last_name,
case
when salary<3000 then 'Bajo'
when salary between 3000 and 8000 then 'Medio'
when salary>8000 then 'Alto'
end Salario,
case department_id
when 90 then 'Administrativo'
when 80 then 'Ventas'
else TO_CHAR(department_id)
end Departamento
FROM  employees
order by 1,2;
```

Subconsultas / joins

Solución al ejercicio 1

```
SELECT     first_name "Nombre", E.Department_id "Numero de
Departamento",department_name "Departamento", SUBSTR(Location_id, 1, 3)
FROM       EMPLOYEES E, DEPARTMENTS D
WHERE      (E.Department_id = D. Department_id) AND ( (E. Department_id = 20)
OR (job_id like 'SA%') );
```

Solución al ejercicio 2

```
SELECT     first_name “Nombre”, job_id “Trabajo”,
           Department_name “Departamento”, D.department_id “Número Dept”
FROM       EMPLOYEES E, DEPARTMENTS D
WHERE      E.Department_id = D. Department_id;
```

Solución al ejercicio 3

```
SELECT     first_name “Nombre”, Job_id “Trabajo”,
           Department_name “Departamento”, D.department_id “Número Dept”
FROM       EMPLOYEES E, DEPARTMENTS D
WHERE      E.Department_id(+) = D. Department_id;
```

Solución al ejercicio 4

```
SELECT      LPAD(' ',3*LEVEL) ||FIRST_NAME
FROM        EMPLOYEES
START WITH MANAGER_ID IS NULL
CONNECT BY MANAGER_ID = PRIOR  EMPLOYEE_ID
```

Solución al ejercicio 5

```
SELECT     first_name||' '||last_name
FROM       EMPLOYEES
WHERE      Salary = (SELECT    MAX(Salary)
                     FROM      EMPLOYEES);
```

Solución al ejercicio 6

```
SELECT     first_name "Nombre", job_id "Trabajo", salary "Salario"
FROM       EMPLOYEES
WHERE      salary > (   SELECT MIN (Salary)
                        FROM   EMPLOYEES
                        WHERE  department_id = 30 )
ORDER BY   salary DESC;
```

Solución al ejercicio 7

```
SELECT     first_name||' '||last_name
FROM       EMPLOYEES
WHERE      manager_id IN (SELECT     employee_id
                          FROM       EMPLOYEES
                          WHERE      last_name = 'King');
```

Solución al ejercicio 8

```
SELECT     TO_CHAR(hire_date, 'DAY')
FROM       EMPLOYEES
GROUP BY   TO_CHAR(Hire_date, 'DAY')
HAVING     (COUNT(TO_CHAR(hire_date, 'DAY')))
        = (SELECT       MAX(COUNT(TO_CHAR(hire_date, 'DAY')))
           FROM         EMPLOYEES
           GROUP BY     TO_CHAR(hire_date, 'DAY'));
```

Solución al ejercicio 9

```
SELECT     first_name||' '||last_name, job_id, salary, hire_date
FROM       EMPLOYEES
WHERE      salary > (SELECT    MAX (salary)
                     FROM      EMPLOYEES
                     WHERE     job_id LIKE 'SA%')
ORDER BY   salary DESC;
```

Solución al ejercicio 10

```
SELECT     first_name||' '||last_name, department_name, salary, job_id
FROM       EMPLOYEES E, DEPARTMENTS D
WHERE      E.department_id=D.department_id
AND        E.department_id = ( SELECT department_id
                               FROM   EMPLOYEES
                               WHERE  first_name = 'Irene' )
AND        salary < (   SELECT AVG(salary)
                        FROM   EMPLOYEES
                        WHERE  department_id = 10 );
```

Solución al ejercicio 11

```
SELECT     first_name, location_id
FROM       EMPLOYEES E, DEPARTMENTS D
WHERE      E.department_id = D.department_id
```

```
AND       (Commission_pct IS NOT NULL OR
           E.department_id = (SELECT department_id
                              FROM  EMPLOYEES
                              WHERE first_name = 'Irene'));
```

Solución al ejercicio 12

```
SELECT    department_id, first_name||' '||last_name, salary
FROM      EMPLOYEES x
WHERE     salary > (SELECT AVG(salary)
                    FROM   EMPLOYEES
                    WHERE  x.department_id = department_id )
ORDER BY  department_id ;
```

Solución al ejercicio 13

```
SELECT    department_name, first_name||' '||last_name, job_id,
TO_CHAR(hire_date, 'DD/MM/YY HH24:MI:SS') dia
FROM      EMPLOYEES x
WHERE     hire_date = (SELECT MAX(hire_date) )
                       FROM  EMPLOYEES
                       WHERE x.department_id = department_id )
ORDER BY  department_id;
```

Insert

Solución al ejercicio 1

```
insert into departments(department_id,department_name,manager_id)
values(280,'Ayuda Social',(select manager_id from departments where
department_name='Finance'));
```

Solución al ejercicio 2

```
INSERT INTO COUNTRIES VALUES('ES','España',1);

INSERT INTO LOCATIONS(LOCATION_ID,STREET_ADDRESS,POSTAL_CODE,CITY,
                      STATE_PROVINCE,COUNTRY_ID)
VALUES    (3300,'Dr. Esquerdo, 34',28022,'Madrid','Madrid','ES');

INSERT INTO DEPARTMENTS(DEPARTMENT_ID,DEPARTMENT_NAME,MANAGER_ID,LOCATION_ID)
VALUES    (280,'Formacion',100,3300);

INSERT INTO JOBS(JOB_ID,JOB_TITLE,MIN_SALARY,MAX_SALARY)
VALUES    ('FO_ORA','Formador Oracle',2000,10000);

INSERT INTO EMPLOYEES(EMPLOYEE_ID,FIRST_NAME,LAST_NAME,EMAIL,PHONE_NUMBER,
HIRE_DATE,JOB_ID,SALARY,COMMISSION_PCT,MANAGER_ID,DEPARTMENT_ID)
VALUES    (210,'Juan','Lopez','jlopez@cleformacion.com',913243675,sysdate,
'FO_ORA', 5000,.56,100,28);
```

Update

Solución al ejercicio 1

```
UPDATE    EMPLOYEES
SET       job_id = 'IT_PROG', salary = salary * 1.13
WHERE     last_name = 'Jones';
```

Solución al ejercicio 2

```
UPDATE EMPLOYEES
SET    hire_date = SYSDATE
WHERE  department_id = (SELECTdepartment_id
                        FROM   DEPARTMENTS
                        WHERE  location_id = 1500 )
AND salary > (SELECT    AVG (salary)
                  FROM          EMPLOYEES
                  WHERE         job_id = 'IT_PROG');
```

Solución al ejercicio 3

```
UPDATE     EMPLOYEES
SET        salary = salary + (SELECT AVG (salary) * 0.15
                              FROM   EMPLOYEES
                              WHERE  department_id = 20)
WHERE      department_id = (   SELECT department_id
                              FROM   EMPLOYEES
                              WHERE  last_name = 'Matos');
```

Delete

Solución al ejercicio 1

```
delete countries
where country_id not in (select country_id from locations);

delete regions
where region_id not in (select region_id from countries);
```

Práctica DDL-DML

1

```
-- Tabla PROFESORES
-- Almacena los datos de los profesores que imparten los distintos cursos
-- la tabla CURSOS tiene una foreign key que hace referencia a ésta tabla
create table profesores(
nombre           varchar2(30),
apellido1        varchar2(20),
apellido2        varchar2(20),
nif              varchar2(9),
direccion        varchar2(40),
titulo           Varchar2(30),
gana             number(4,2)         constraint nn_profesores_gana not null,
constraint pk_profesores_nif         primary key(nif),
constraint uq_profesores_fullname    unique(nombre,apellido1,apellido2));
```

```
-- Tabla CURSOS
-- Almacena la información de cada curso que se impartirá
-- la tabla ALUMNOS tiene una foreign key que hace referencia a ésta tabla.
create table cursos(
nombre_curso     varchar2(40),
cod_curso        number(2),
nif_profesor     varchar2(9),
maximo_alumnos   number(2),
fecha_inicio     date,
fecha_fin        date,
num_horas        number(3)    constraint nn_cursos_horas not null,
```

```
constraint pk_cursos_cod      primary key(cod_curso),
constraint uq_cursos_nombre   unique(nombre_curso),
constraint ck_cursos_fechas   check (fecha_inicio<fecha_fin));
```

```
-- Tabla ALUMNOS
-- Almacena la información de los alumnos que asistan a algún curso
create table alumnos(
nombre           varchar2(30),
apellido1        varchar2(20),
apellido2        varchar2(20),
nif              varchar2(9),
direccion        varchar2(40),
sexo             varchar2(1),
fecha_nacimiento date,
curso            number(2)    constraint nn_alumnos_curso not null,
constraint pk_alumnos_nif     primary key(nif),
constraint ck_alumnos_sexo    check(sexo in ('F','M')));
```

```
-- Relacionar las tablas
alter table cursos
add constraint fk_cursos_profesor foreign key(nif_profesor)
    references profesores(nif);
alter table alumnos
add constraint fk_alumnos_curso foreign key(curso)
    references cursos(cod_curso);
```

2

```
-- Se inserta algunas tuplas en la tabla PROFESORES
insert into profesores
values('Miguel', 'Torres','García','12345678X','Argentina,
6','Programador',40);
insert into profesores
values('Lucia','Benítez','Suárez','13334444A','Av. La Paz, 20','Ing.
Sistemas',52.55);
```

```
-- Se inserta algunas tuplas en la tabla CURSOS
insert into cursos
values('SQL',0,'12345678X',15,'05-MAY-11','23-JUN-11',120);

insert into cursos
values('SQL Tuning',1,'13334444A',null,null,'01-09-11',80);

insert into cursos
values('PLSQL',2,'12345678X',12,null,null,100);
insert into cursos
values('Administración I',3,'13334444A',10,'12-06-11','31-06-11',100);
-- Falla por que no existe el 31 de Junio. No insertarlo.
```

```
-- Se insertan alunas tuplas en la tabla ALUMNOS
insert into alumnos
values('Lucas','Salas','García','12345678X',null,'V',null,1);
-- ORA-02290: restricción de control (ALUMNO1M.CK_ALUMNOS_SEXO) violada
-- Recordad que el sexo solo puede ser H o M
insert into alumnos
```

```
values('Lucas','Salas','García','12345678X',null,'M',null,1);

insert into alumnos
values('Lucia','Benítez','Suárez','13334444A','Av. La Paz, 20','F','23-FEB-
1969',null);
-- ORA-01400: no se puede realizar una inserción NULL en
("ALUMNO1M"."ALUMNOS"."CURSO")
-- Al no darnos la información del curso no podemos darle de alta.
insert into alumnos
values('Lucia','Benítez','Suárez','13334444A','Av. La Paz, 20','F','23-FEB-
1969',0);

insert into alumnos
values('Antonia','Reyes','Pérez','87654321A','Canadá, 3','M',null,0);


insert into alumnos
values('Manuel','Guerra','Pérez','123123123L','Canadá, 3',null,null,1);
-- ORA-12899: el valor es demasiado grande para la columna
"ALUMNO1M"."ALUMNOS"."NIF" (real: 10, máximo 9)
-- El nif ha sido mal escrito, no cabe. Insertarlo con nif 12312312L
insert into alumnos
values('Manuel','Guerra','Pérez','12312312L','Canadá, 3',null,null,1);
```

3

```
-- Intentar ir corrigiendo cada error uno a uno para ver los mensajes que da.
--
insert into alumnos
values('Sergio','Aguirre','Moscoso','12345678X',null,'m',null,null);
-- ORA-01400: no se puede realizar una inserción NULL en
("ALUMNO1M"."ALUMNOS"."CURSO")
-- dar de alta en el curso 0
insert into alumnos
values('Sergio','Aguirre','Moscoso','12345678X',null,'m',null,0);
-- ORA-02290: restricción de control (ALUMNO1M.CK_ALUMNOS_SEXO) violada
-- sexo solo puede ser H o M
insert into alumnos
values('Sergio','Aguirre','Moscoso','12345678X',null,'M',null,0);
-- ORA-00001: restricción única (ALUMNO1M.PK_ALUMNOS_NIF) violada
-- ya existe un alumno con ese nif!
-- corregir al nif 12345678C
insert into alumnos
values('Sergio','Aguirre','Moscoso','12345678C',null,'M',null,0);
```

4

```
alter table profesores
add edad number(2);
```

5

```
-- La edad de los profesores está entre 18 y 65 años, incluidos.
alter table profesores
add constraint ck_profesores_edad check(edad between 18 and 65);
```

```
-- Ningún curso su número máximo de alumnos será menor de 10.
alter table cursos
add constraint ck_cursos_maxalu check(maximo_alumnos>=10);
```

```
-- A partir de ahora el número de horas de los cursos debe ser mayor que 100.
alter table cursos
add constraint ck_cursos_horas check(num_horas>100) enforce;
```

6

```
alter table cursos
drop constraint uq_cursos_nombre;
```

7

```
alter table profesores
drop constraint nn_profesores_gana;
```

8

```
update alumnos
set fecha_nacimiento='23-12-1949'
where nombre='Antonia' and apellido1='Reyes';
```

9

```
update alumnos
set curso=5
where nombre='Antonia' and apellido1='Reyes';
-- ORA-02291: restricción de integridad (ALUMNO1M.FK_ALUMNOS_CURSO) violada -
clave principal no encontrada.
-- Al no existir el curso no se le da de alta.
```

10

```
delete profesores
where nombre='Laura' and apellido1='Jiménez';
-- Al no existir se elimininan 0 filas.
```

11

```
-- Eliminar la foreign key que hace referencia a la primary key
alter table cursos
drop constraint fk_cursos_profesor;

-- Eliminar la primary key ya que solo se tiene una por tabla
alter table profesores
drop constraint pk_profesores_nif;
-- Eliminar la restricción de unicidad que existe.
-- ya que no son compatibles una unique con una primary key
-- formada por las mismas columnas.
alter table profesores
drop constraint uq_profesores_fullname;

-- A este punto podemos crear la primary key
alter table profesores
add constraint pk_profesores_fullname primary key(nombre,
apellido1,apellido2);
```

```
-- para mantener la integridad referencial
-- entre las tablas CURSOS y PROFESORES
-- se necesita de los mismos campos para hacer referencia
-- a la primary key
alter table cursos
add (nombre      varchar2(30),
apellido1  varchar2(20),
apellido2  varchar2(20));

-- creación de la foreign key
alter table cursos
add constraint fk_cursos_profesor foreign key(nombre,apellido1,apellido2)
    references profesores(nombre,apellido1,apellido2);
-- Posteriormente habría que actualizar los datos.
```

Implementación de restricciones de integridad durante la creación de una tabla

1

```
CREATE TABLE habitaciones (num_hab   NUMBER(2) CONSTRAINT pk_num PRIMARY KEY,
                           Tipo      varchar2(15) CONSTRAINT ck_tipo
                                                  CHECK(tipo IN('simple','doble')),
                           Precio    NUMBER(9));
```

2

```
CREATE TABLE clientes (n_cliente NUMBER(4) CONSTRAINT pk_num_clie PRIMARY KEY,
                       nif       NUMBER(8) CONSTRAINT uq_nif  unique,
                       nombre_cliente varchar2(90),
                       nvisa     NUMBER(4),
                       telefóno  NUMBER(9));
```

3

```
CREATE TABLE reservas(  n_reservas NUMBER   CONSTRAINT pk_reservas PRIMARY KEY,
    n_cliente  NUMBER(4)CONSTRAINT fk_client REFERENCES
                                  clientes(n_cliente) ON DELETE CASCADE,
                        num_hab    NUMBER(2) CONSTRAINT fk_numh REFERENCES
                habitaciones(num_hab)ON DELETE CASCADE,
          pagado     char(2)   CONSTRAINT ck_pagado CHECK
                                                  (pagado IN('s','n')));
```

Utilización de la tabla de excepciones

1

```
ALTER TABLE clientes
DISABLE PRIMARY KEY CASCADE;
```

2

```
INSERT INTO clientes values(1,'3456755','Jose Garcia',9959,695356778);
```

3

```
create table exceptions (
  row_id         rowid,
  owner          varchar2(30),
  table_name     varchar2(30),
```

```
  CONSTRAINT     varchar2(30));
```

4

```
ALTER TABLE  clientes
    ENABLE CONSTRAINT pk_num_clie
    EXCEPTIONS  INTO exceptions;
```

5

```
SELECT * FROM exceptions WHERE CONSTRAINT = 'PK_NUM_CLIE';

SELECT * FROM clientes WHERE rowid= '  '    -- rowid de la busqueda anterior

UPDATE clients set n_cliente =              /*con este update modificar el
numero de cliente duplicado.*/
```

6

```
ALTER TABLE  clientes
    ENABLE CONSTRAINT pk_num_clie;
```

Vistas

Solución al ejercicio 1

```
create view empleados as
select employee_id, last_name||','||first_name "full_name", email,
phone_number, department_name, street_address, city, state_province,
postal_code, country_name
from employees join departments using(department_id)
join locations using(location_id)
join countries using(country_id)
order by 1;
```

Sinónimos

Solución al ejercicio 1

```
create public synonym empleados      for employees;
create public synonym departamentos  for departments;
create public synonym puestos        for jobs;
create public synonym localidades    for locations;
create public synonym paises         for countries;
create public synonym regiones       for regions;
create public synonym historial_trabajo for job_history;
```