# 3

# SQL

## Contenido



## Objetivos

- ?.
  -

## 3.1 Introducción

En este capítulo se estudiará el Lenguaje Estructurado de Consultas (Structured Query Language [SQL]), cuyas sentencias se agrupan en sublenguajes. Para que el estudiante aprenda su sintaxis —que se explica en numerosos sitios de la Web— se realizarán algunas sugerencias de estilo para escribir las sentencias SELECT —también denominadas "consultas"— y otras indicaciones útiles para aprovechar todo su poder; de esta manera, se facilitará la obtención de información que, normalmente, se almacena en una o varias tablas de una base de datos y, además, para cubrir las tareas de una aplicación comercial, científica, de un sitio Web, en la que se insertarán, borrarán o modificarán los datos ya almacenados en los motores relacionales que soportan la mayoría de las aplicaciones.

Es importante destacar el rol de SQL como lenguaje especializado en la comunicación con la Base de Datos, que funciona dentro o empotrado en otros lenguajes de uso general que construyen las demás funcionalidades de una aplicación de negocios, como los numerosos lenguajes procedimentales conocidos como COBOL, C, C++, orientados a objetos como Java, PHP y Python. A estos lenguajes se los podría caracterizar como de propósitos múltiples, ya que con ellos se construirán pantallas, menúes, botones, campos listas, mientras que para acceder a la Base de datos para obtener las filas haciendo consultas o inserciones, borrados y actualizaciones, se requerirán las correspondientes sentencias SQL. También hay que mencionar que las bases de datos proveen otros lenguajes —procedimentales o basados en procedimientos que se ejecutan en el servidor de Base de Datos, como PL/SQL de Oracle, SQL PL de DB2, T-SQL de MS Sql Server— que se tratarán en otro capítulo.

## 3.2 Algo de Historia del Lenguaje SQL

En su libro *El modelo Relacional para la administración de Bases de Datos* [Codd, 1990], en el capítulo de "Introducción al Modelo Relacional", E.J. Codd da pistas sobre el origen del lenguaje y lo atribuye a un grupo de Investigación de IBM a finales del año 1972. En esta publicación, el padre del Modelo Relacional, afirma que es inconsistente en diversas formas con las máquinas abstractas del Modelo Relacional V1 (versión 1) y V2 (versión 2), y que algunas características no están implementadas directamente y, otras, no están implementadas correctamente y que, cada una de las inconsistencias, pueden reducir su utilidad. En esta publicación, Codd afirma que el lenguaje QUEL, por Query Language o Lenguaje de Consulta —inventado por Held y Stonebraker de la Universidad de California, Berkeley—, era el más completo pero, pese a esto, el SQL —desarrollado en IBM por Andrew Richardson, Donald C. Messerly y Raymond F. Boyce— ganó preeminencia porque se adoptó como estándar ANSI y, a partir de ello, lo soportan todos los Motores de Base de datos.

En 1970, luego de que el Dr. E.F. Codd introdujera el concepto del Modelo Relacional, IBM desarrolló este lenguaje. Se buscó que fuera fácil de aprender y potente en sus posibilidades expresivas. Se basa en la estructura del Inglés con el objetivo de permitirle a un usuario no técnico armar sus propias consultas para obtener las filas de la base de datos, aunque esta ventaja se consiguió sólo parcialmente, el SQL es fácil de aprender y, con la ayuda de los programas con asistentes, el técnico no especializado puede lograr resultados muy interesantes.

En 1979, casi diez años después de la presentación del modelo relacional de Codd, aparece la primera base de datos comercialmente disponible con SQL como único lenguaje para acceder a los datos; luego, las otras bases de datos empezaron a incorporar esta característica. En 1986, se convirtió en un estándar de la American National Standard Institute (ANSI). En la actualidad, existen siete versiones (SQL86, SQL89, SQL92 o SQL2, SQL99 o SQL 3, SQL2003, SQL2006, SQL2008) [wikipedia, 2009] a las que les agregaron funcionalidades que adoptaron todos los servidores que quisieron cumplir con el último estándar de la industria.

Los avances en motores de Bases de Datos, en paralelo con los del lenguaje, son múltiples: el descubrimiento de Internet como una forma de conectarse a los motores de bases de datos fue uno de los transformadores de la WWW, que de un rol presentadora de documentos estáticos o folletería electrónica, pasó a soportar aplicaciones corporativas que fueron la base para la explosión del e-commerce como soporte de una nueva economía y se transformó en un ejemplo concreto de la formación de la sociedad del conocimiento. En este movimiento, hay que mencionar a MySQL como un protagonista de mucho menor tamaño que los motores tradicionales, pero con las prestaciones suficientes para publicar un sitio web y hacer funcionar los sitios de todo tamaño que aparecieron en esos tiempos. La estrategia de esta empresa fue focalizarse en ofrecer lo que las grandes empresas no hicieron, una implementación en pequeño de las funcionalidades básicas de SQL. [MySQL, 1999]

## 3.3 Características del lenguaje SQL

El lenguaje SQL se considera, por un lado, un lenguaje diseñado específicamente para la comunicación entre usuarios y, por otro, la base de datos para realizar todas las tareas requeridas para resolver los requerimientos como obtener información almacenada, realizar cálculos, modificar lo existente y agregar nuevas filas que contengan información de clientes, productos, transacciones (como ventas, compras, pedidos, reservas, asistencia, ausencias, llegadas tardes, y un casi infinito etcétera), puesto que la mayoría de las aplicaciones actuales usan bases de datos relacionales y, por ende, SQL.

La característica más destacada del lenguaje SQL es que es No-procedimental; es decir, no se indica en sus sentencias cómo realizar la tarea sino que se limita a describir el resultado buscado, y queda todo el trabajo de resolver lo solicitado al servidor de bases de datos que, de acuerdo con su optimizador y con los metadatos (se los denomina así porque son datos acerca de los datos) del diccionario, cumplirá con la sentencia SQL provista.

Por lo antedicho, SQL no tiene sentencias de control de flujo desde su origen aunque, recientemente, se han incluido como partes opcionales aceptadas del estándar de SQL, ISO/IEC 9075-5: 1996.

Es común que estas sentencias de control de flujo —conocidas como “módulos persistentes almacenados”— se consideren como extensiones procedimentales del lenguaje. Los proveedores de Bases de datos tienen estas extensiones (por ejemplo: PL/SQL de Oracle, T-SQL de SQL Server de Microsoft, SQL-PL de DB2 de IBM y PSQL de PostgreSQL). En general, estas extensiones complementan a SQL para construir procedimientos almacenados, paquetes y disparadores, que se explicarán —como ya se anticipó— en el siguiente capítulo.

## 3.4 El lenguaje SQL y su sublenguaje de definición de datos o DDL

Las sentencias SQL se agrupan en sublenguajes de acuerdo con sus propósitos: primero se revisarán las sentencias del Lenguaje de definición de datos o DDL (Data Definition Language), que permitirán crear, borrar o modificar objetos dentro de la base de datos. Por ejemplo, se puede utilizar CREATE, DROP, ALTER y RENAME para crear, eliminar, modificar su estructura y cambiarle el nombre a los objetos de datos de una base de datos como en el caso de las Tablas, las Vistas comunes, las Vistas materializadas, los Índices, las funciones, los procedimientos y los paquetes que pertenecen a un Esquema lógico. También se aplican estas acciones sobre los Usuarios, los Roles, las estructuras de almacenamiento como una Base de datos propiamente dicha, un espacio de tablas (Tablespace), los Segmentos de RollBack (o RollBack Segments) y numerosas estructuras propias de un motor de Base de datos.

TRUNCATE es una sentencia que permite borrar todo el contenido de una tabla, sin eliminar la estructura y sin generar transacciones. Por esta razón, se la nombró aparte.

En DDL se encuentran, también, las sentencias GRANT, REVOKE que permiten que se otorguen privilegios sobre objetos y sobre acciones o de sistema, a los usuarios y a los roles de la base de datos.

Además, COMMENT es una sentencia de DDL que permite guardar comentarios sobre tablas y columnas.

Se cerrará con la sentencia AUDIT y NOAUDIT —presente en algunos motores— que activa o desactiva el registro de todos los cambios hechos sobre los objetos por sesiones, también denominada auditoría automática.

A continuación, se verán algunos ejemplos en los que se utilizarán las tablas normalizadas en los capítulos anteriores.

Para el modelo de datos del sistema de Ventas, se definieron cuatro tablas: Artículos, Clientes, Facturas y Detalle de Factura. Se analizarán las sentencias para su creación. A la tabla Clientes, se le agregará una columna “olvidada” en el momento de creación o agregada por nuevos requerimientos para ejemplificar la sentencia ALTER. Además, se creará una Vista que mostrará las facturas del cliente “A01”. Una vez creadas las tablas y la vista, se creará, además, un usuario para luego darle privilegios de lectura al usuario sobre las tablas. Por último, se agregarán los comentarios sobre las tablas y una columna.

Nótese que las sentencias SQL empiezan siempre con un verbo que indica la acción; luego, el tipo de objeto sigue solo en la creación y en la modificación y, después, el nombre del objeto que se tratará. Se continuará —si es necesario— con otra información que completará la tarea. Si bien SQL no es sensible a las mayúsculas y a las minúsculas, es bueno que se adopte un ‘protocolo’ de escritura que facilite la lectura del código. En este libro, se adoptará el uso de las mayúsculas para las palabras reservadas y de las minúsculas para los nombres de los objetos.

En SQL, los nombres de los objetos tienen una longitud máxima (depende de cada motor), son sensibles a las mayúsculas y a las minúsculas y no soportan espacios intermedios en los nombres; normalmente, se utilizan el carácter guión bajo o ‘_’

para unir las palabras y, de esta manera, evitar los espacios. En Oracle, los nombres, que pueden tener hasta 30 caracteres, deben empezar con una letra y no con número o signo. Los signos permitidos son: '$', '_' y '#'.

Los nombres de los objetos se identificarán con su esquema y con su nombre. Para ello, se utilizará la notación "esquema.nombre". Esta regla se puede resolver, también, con el uso de sinónimos públicos que referenciarán directamente a los objetos. Originalmente, en el modelo de datos se incluyeron todos los acentos de las palabras, pero en la creación no se los utilizó para facilitar el trabajo de escritura de las sentencias SQL.

Las sentencias SQL se pueden extender por varias líneas y finalizan con un punto y coma " ; "; y tienen como separador interno o semántico el espacio ' ' o la coma ','.

Ejemplos de Creación de objetos del modelo de datos de Ventas:

```
CREATE TABLE articulo     (codigo_del_articulo VARCHAR2(8),
nombre_de_articulo VARCHAR2(30) NOT NULL,
precio_unitario NUMBER(8,3));

CREATE TABLE clientes     (codigo_del_cliente VARCHAR2(8),
nombre _del _cliente VARCHAR2(30) NOT NULL);

CREATE TABLE facturas     (sucursal                    NUMBER(2),
número_de_factura         NUMBER(2),
fecha_de_factura TIMESTAMP,
forma_de_pago_factura VARCHAR2(3),
codigo_del_cliente        VARCHAR2(8),
total_de_la_factura       NUMBER(14,2));

CREATE TABLE detalle_de_factura (sucursal                          NUMBER(2),
numero_de_factura                 NUMBER(2),
codigo_del_articulo               VARCHAR2(8),
cantidad_del_artículo  NUMBER(8,3) NOT NULL,
precio_unitario_del_articulo      NUMBER(8,3),
subtotal_del_articulo             NUMBER(9,3));
```

En las columnas en las que se prohíbe, por regla de negocios, que tenga valores null, se ubican las restricciones de la columna con las palabras NOT NULL.

Para destacar, se usan los tipos de datos de Oracle, donde NUMBER es un tipo de dato que soporta hasta 38 dígitos, en los que se incluyen los decimales y sus signos. Para los caracteres alfanuméricos de longitud variable, se utiliza VARCHAR2, que soporta hasta 4096 caracteres. Para la fecha, se emplea Timestamp —que es el

tipo de datos que se ha definido como estándar— que soporta seis posiciones para fracciones de segundo y define la zona horaria, lo que aumenta las prestaciones para el registro de tiempos.

**Ejemplos de modificación de objetos de la Base:**

Una vez creada la tabla Clientes, resultó necesario separar los nombres y los apellidos con la sentencia ALTER TABLE

```
ALTER TABLE cliente ADD apellido_del_cliente VARCHAR2(30));
```

Otra alternativa —si aún no se tiene filas ni restricciones referenciales sobre la tabla que se modificará— sería eliminarla y volverla a crear con todas las columnas o restricciones, según los últimos requerimientos.

```
DROP TABLE clientes;
CREATE TABLE clientes    (codigo_del_cliente VARCHAR2(8),
nombre _del _cliente VARCHAR2(30),
apellido _del _cliente VARCHAR2(30));
```

Se notará que, en el momento de la creación de las tablas, no se incluyó nada acerca de las claves primarias y las foráneas; estos constraints se agregarán con ALTER.

La sentencia ALTER TABLE tiene, como modificadores, las acciones ADD, MODIFY y DROP, que pueden agregar, modificar y eliminar columnas. En algunas implementaciones de lenguajes se permite el uso, dentro de ALTER, de RENAME, que renombra las columnas de una tabla, pero su explicación detallada excede el propósito del libro.

Se agregarán, con ALTER, las restricciones correspondientes a las claves primarias y a las foráneas de las tablas anteriores; luego, se ilustrará, con un ejemplo, cómo se puede hacer, siempre bajo el supuesto de que no tienen filas y que las tablas referenciadas en la clave primaria ya existen.

```
ALTER TABLE clientes
ADD CONSTRAINT cliente_pk PRIMARY KEY (codigo_del_cliente));
```

Después de crear la tabla Artículos se define su clave primaria.

```
ALTER TABLE articulo
ADD CONSTRAINT articulos_pk PRIMARY KEY (codigo_del_articulo));
```

Para crearla simultáneamente la sintaxis tendría estas dos variantes:

Variante uno, como restricción de columna (no se puede usar cuando la clave es múltiple)

```
CREATE TABLE articulo     (codigo_del_articulo VARCHAR2(8) PRIMARY KEY,
nombre_de_articulo VARCHAR2(30),
precio_unitario NUMBER(8,3));
```

Variante dos, como restricción de tabla (es la única que se puede usar cuando la clave es múltiple)

```
CREATE TABLE articulo     (codigo_del_articulo VARCHAR2(8) ,
nombre_de_articulo VARCHAR2(30),
precio_unitario NUMBER(8,3),
CONSTRAINT articulos_pk PRIMARY KEY (codigo_del_articulo));
```

En este último ejemplo, se usó la cláusula CONSTRAINT para que la restricción tuviera un nombre "articulos_pk", que es muy útil cuando una aplicación trata de cargar una fila repetida y el servidor se lo impide. Éste le enviará un mensaje diciendo que esta restricción, "articulos_pk", ha sido atacada, lo que permitirá entender que se ha intentado cargar una fila con estas columnas con null, o con un valor de clave repetido.

Ahora se agregarán las claves foráneas, y se advertirá que la tabla referenciada debe existir. En una implementación real, también se considerará que si las tablas ya existen desde hace tiempo y tienen filas que no cumplen con las relaciones que se introducirán, se le indicará que ignore los incumplimientos previos, pero no se lo verá en la sintaxis porque es un tema avanzado en el estudio de SQL, que excede el alcance de este trabajo.

```
ALTER TABLE factura
ADD CONSTRAINT cliente_fk FOREIGN KEY (codigo_del_cliente)
REFERENCES clientes(codigo_del_cliente);

ALTER TABLE detalle_de_factura
ADD CONSTRAINT articulo_fk FOREIGN KEY (codigo_del_artículo)
REFERENCES artículo(codigo_del_ articulo);

ALTER TABLE detalle_de_factura
ADD CONSTRAINT factura_fk FOREIGN KEY (numero_de_la_factura)
REFERENCES factura(numero_de_la_factura);
```

Para completar las restricciones que permite el lenguaje SQL, se deberá también nombrar el de validación de contenido o CHECK, que impide ingresar un valor en una columna que no cumpla la condición definida en la restricción. Se aplicará un requerimiento que indicará que los precios de los artículos no serán negativos, entonces:

```
ALTER TABLE artículo
ADD CONSTRAINT precio_ck CHECK (precio_unitario_del_articulo > 0);
```

Para ejemplificar otras sentencias DDL, se ha omitido erróneamente una 's' en el nombre de la tabla Artículos, para corregirlo se utilizará RENAME. Será tarea del motor actualizar todos objetos que hayan sido creados con referencias a este objeto para que se mantenga esta relación.

```
RENAME TABLE artículo TO artículos;
```

Ahora es el momento de crear otros objetos de datos: las vistas y los índices:

Aunque aún no se ha visto la sentencia SELECT, se aplicará, para la vista, una sentencia muy simple que, sin analizar los detalles, traerá todas las facturas que tengan en el código de cliente un valor 'A01'.

```
CREATE VIEW v_factura_clienteA01 AS
SELECT numero_de_factura, fecha_de_factura, monto_de_factura

FROM   facturas
WHERE codigo_de_cliente = 'A01'
```

Así, se habrá creado una vista que se usará de la misma forma que una tabla; por eso, se recomienda que en el nombre identifique claramente que se trata de una vista. Por ejemplo, si se utiliza un prefijo como 'v_...', como se mostró en el ejemplo anterior.

Para los índices la sintaxis es:

```
CREATE INDEX factura_fecha_idx ON factura(fecha_de_la_factura);
```

La consecuencia de esta sentencia es que se ha creado un índice —implementado como un segmento de datos— similar a una tabla en la que se guarda el identificador único de las que tienen cada fecha distinta; esto facilitará el acceso a las páginas y a las filas donde estén las fechas buscadas, de acuerdo con el funcionamiento del motor de la base. Habitualmente, estos índices tienen la característica de ser árboles B, lo que garantiza el acceso rápido a las filas que cumplen el criterio de búsqueda que use la columna "fecha_de_la_factura". En otras variantes de sintaxis, se puede indicar que no se permitan las repeticiones ("CREATE UNIQUE INDEX …"), o que arme una estructura de índice que no base en árboles, sino en una estructura en mapas de bit ("CREATE BITMAPPED INDEX …") con características dirigidas a la indexación de columnas con baja selectividad, o con pocos valores distintos, lo que permite que se resuelvan rápidamente las consultas basadas en las columnas con pocos valores diferentes y gran cantidad de filas cada uno.

La creación de Roles, a los que se puede definir como un conjunto de privilegios que se asignan siempre en bloque, se presentará luego de la creación de usuarios.

```
CREATE USER jgomez IDENTIFIED BY lacontraseña;
```

El rol de operador de consultas “op_consultas” se crea de la siguiente manera:

```
CREATE ROLE op_consultas;
```

Los roles se crean para facilitar la administración de privilegios de muchos objetos a grupos de usuarios que necesitan los mismos accesos porque realizan trabajos similares.

En este ejemplo, se le dará todos los privilegios al rol “op_consulta” y, luego, se lo asignará al usuario “jgomez”.

En Oracle, se puede definir un rol con contraseña para que sea activado después de que el usuario que lo posee se conecte, cuando tenga que realizar una tarea que requiera los privilegios del rol y solicite activarlo. A la vez, se le solicita una contraseña específica para que esa activación pueda ser posible.

En algunas bases de datos, con la mera creación del usuario alcanza para poder conectarse inmediatamente; en Oracle, es preciso que se le dé los permisos de conexión “create session”, con el siguiente grupo de sentencias DDL, GRANT y REVOKE. Para esto, se aplicará al usuario un privilegio de acción o de sistema.

```
GRANT CREATE_SESSION TO jgomez;
```

Otro privilegio de acción o de sistema sería darle al usuario la posibilidad de consultar cualquier tabla en la base de datos, que se aplicará a las actuales y a las que no se han creado aún.

```
GRANT SELECT ANY TABLE TO jgomez;
```

Ahora, se le otorgará al rol todos los privilegios y, luego, se los asignaremos todos juntos al usuario con un solo GRANT.

Con los privilegios de objetos, que son SELECT, ALTER, DROP, DELETE, REFERENCE, INDEX y el que abarca a todos, ALL, la sintaxis indicará sobre qué objetos se asignarán los privilegios al rol:

```
GRANT SELECT ON articulos TO op_consulta;
```

Si quisiéramos darle los otros privilegios solamente a jgomez, se hace directamente

```
GRANT DELETE ON articulos TO jgomez;
```

Para no hacer todas las sentencias se puede abreviar con

```
GRANT ALL ON articulo TO jgomez;
```

Luego le otorgaremos el rol de op_consultas al usuario jgomez, de esta manera:

```
GRANT op_consultas TO jgomez;
```

Si por alguna razón se quisiera quitar estos privilegios, se usará la sentencia REVOKE como sigue:

```
REVOKE CREATE_SESSION FROM jgomez;
REVOKE op_consulta FROM jgomez; (aquí le quitamos el rol al usuario)
REVOKE ALL FROM jgomez;
```

Se finalizará el tratamiento de DDL con un ejemplo de la creación de comentarios sobre una tabla y sobre una columna.

```
COMMENT ON articulos ('Contendrá todos la información de los artículos que se venden en este negocio');

COMMENT ON articulos.nombre_del_articulo ('Almacena el nombre completo de cada artículo')
```

### 3.4.1 Sentencias del sublenguaje DML

Este sublenguaje abarca a SELECT —que se tratará inmediatamente— y a INSERT, a DELETE y a UPDATE que sirven —respectivamente— para insertar filas, para borrarlas y para cambiarle el contenido de las columnas en las filas existentes, y para que cumplan con las condiciones de la cláusula WHERE, como se verá en los siguientes ejemplos.

SELECT es la sentencia que permite la obtención de los valores almacenados en las columnas de las filas recuperadas como resultado del acceso a las tablas o a las vistas de la Base de datos. A esta sentencia también se la denomina "Consulta", porque es, justamente, lo que hace el motor: busca las filas de las tablas incluidas en su cláusula FROM, y realiza las operaciones definidas en el Algebra relacional para seleccionar aquellas filas que cumplan las condiciones expresadas.

Esta sentencia se estructura en distintas cláusulas, con el funcionamiento que se describirá a continuación:

**SELECT** <lista de columnas, valores constantes, funciones, subconsultas, etc.>
**FROM** <lista de tablas, vistas, subconsultas>
**WHERE** <condicion> [**AND, OR, NOT**] <condicion>
**GROUP BY** <lista de columnas>
**HAVING** <condicion> [**AND, OR, NOT**] <condicion>
**ORDER BY** <lista de columnas, numero de orden o alias de columna> [**DESC**, **ASC**];

Las cláusulas obligatorias son **SELECT** y **FROM,** en Oracle; en otras bases como MySQL, sólo es obligatoria la cláusula **SELECT**.

En esta cláusula, se puede realizar la operación Project del Álgebra Relacional cuando se enumeran sólo algunas columnas de la tabla que se consultará. Para evitar esta operación, se incluirán todas las columnas, una por una, o se usará el operador asterisco (' * ') que indicará que es necesario la recuperación de todas las columnas de la o las tabla/s del FROM.

En la primera cláusula, se pueden combinar las columnas de las tablas, las vistas o las subconsultas de la cláusula FROM, funciones como las que veremos en los párrafos siguientes y subconsultas. Estas subconsultas devuelven un sólo valor, por lo que se las denomina “Escalares”. También se agregan valores literales que se repetirán por cada fila mostrada. Por ejemplo: la sentencia muestra datos constantes, una columna con el nombre de los empleados, luego el sueldo y, en la columna aumento, el cálculo de sueldo con un 20% de aumento. Es para destacar que en los dos últimos se adoptó un “Alias de Columna” para titular las columnas con un nombre distinto al de las columnas de la tabla.

SELECT ‘El nombre es:’, ename, salario salarioAnterior, salario*1.20 aumento, FROM empleados;

Resulta en

| ‘El nombre es:’ | ename | salarioAnterior | aumento |
|---|---|---|---|
| El nombre es: | Perez | 1100 | 1200 |
| El nombre es: | Lopez | 900 | 1080 |
| El nombre es: | Suarez | 1100 | 1320 |
| El nombre es: | Gianni | 700 | 840 |
| El nombre es: | Vinci | 1500 | 1800 |

## 3.5 Funciones de fila simple

En el ejemplo anterior, se vio el uso de una función de fila simple, el operador multiplicación ‘*’. En este caso, ‘*’ es la función multiplicación que, en otro contexto, significa “todas las columnas de las tablas que figuran en el FROM”.

Las funciones de fila simple retornan un sólo valor por cada fila de una tabla o de una vista consultada. Estas funciones pueden usarse en la lista de columnas del **SELECT**, en las cláusulas **WHERE** y **HAVING**, que forman parte de las condiciones y demás.

## 3.6 Funciones numéricas

La mayoría de las funciones numéricas, que reciben y devuelven valores numéricos, retornan valores NUMBER con exactitud en 38 dígitos decimales.

Las más simples son los operadores matemáticos:

**‘*’** por, **‘/’** dividido, **’+’** más, **’-‘** menos

Las funciones como **COS**, **COSH**, **EXP**, **LN**, **LOG**, **SIN**, **SINH**, **SQRT**, **TAN** y **TANH** tienen una exactitud de 36 de dígitos decimales, y las funciones **ACOS**, **ASIN**, **ATAN** una exactitud de 30 dígitos decimales. Algunas funciones del SQL de ORACLE 10g de ejemplo:

ABS (Valor Absoluto)
ACOS (Arc Coseno)
ASIN (Arc Seno)
ATAN (Arc Tangente)
CEIL (Valor tope)
COS (Coseno)
COSH (Coseno Hiperbólico)
EXP (Exponente)
FLOOR (Valor Mínimo)
LN (Logaritmo Neperiano)
LOG (Logaritmo Decimal)
MOD (Resto de la división)
POWER (elevar a una potencia)
ROUND (number) (Redondeo Numérico)
SIGN (Signo)
SIN (Seno)
SINH (Seno Hiperbólico)
SQRT (Raíz Cuadrada)
TAN (Tangente)
TANH (Tangente Hiperbólico)
TRUNC (number) (Truncar Numérico)

Para una lista completa de funciones de Fila Simple en Oracle, ver [Lorentz, 2009].

Las funciones de carácter son aquellas que reciben argumentos numéricos, de fecha o de caracteres y devuelven valores en formato carácter, y son:

CHR (devuelve un carácter de acuerdo con un argumento numérico, normalmente el valor ASCII).

CONCAT (concatena valores).
INITCAP (pone mayúscula a las primeras letras de cada palabra=).
LOWER (transforma en minúsculas).
LPAD (completa a la izquierda con un carácter hasta una cantidad de caracteres).
LTRIM (recorta una cantidad de caracteres a la izquierda).
REPLACE (reemplaza un carácter por otro).
RPAD (completa a la derecha con un carácter hasta una cantidad de caracteres).
RTRIM (recorta una cantidad de caracteres a la derecha).
SUBSTR (toma una subcadena de una cantidad de caracteres desde una posición determinada).
UPPER (transforma en mayúsculas).

Para una lista completa de funciones de Fila Simple en Oracle, ver [Lorentz, 2009].

Las funciones de fecha y hora son:

ADD_MONTHS (suma meses a una fecha).
CURRENT_DATE (la fecha del sistema).
EXTRACT (datetime) (extrae fecha u hora).
LAST_DAY (último día del mes de la fecha argumento).
MONTHS_BETWEEN (cantidad de meses entre dos fechas argumento).
ROUND (date) (redondeo de fechas, a mes o a año).
SYSDATE (fecha del sistema).
TO_CHAR (datetime) (traducir a caracteres una fecha).
TRUNC (date) (truncar una fecha).

Para una lista completa de funciones de Fila Simple en Oracle, ver [Lorentz, 2009]

## 3.7 Funciones generales de comparación

Las funciones generales de comparación determinan los valores mayores o menores de un conjunto de datos:

GREATEST (mayor)
LEAST (menor)

## 3.8 Funciones de Conversión

Convierten un valor de un tipo de datos (cf. más adelante en este capítulo) a otro tipo de datos:

TO_CHAR (character)
TO_CHAR (datetime)
TO_CHAR (number)
TO_CLOB
TO_DATE
TO_LOB
TO_MULTI_BYTE
TO_NUMBER

## 3.9 Funciones de Grupo

Las funciones de grupo devuelven un sólo resultado por cada grupo de filas que se forman por cada valor distinto de una o varias columnas. Si se considera que el grupo de filas puede tener una sola fila. Las funciones de grupo pueden aparecer en la lista de columnas

en el SELECT y en las condiciones dentro de la cláusula HAVING, pero es un error común incluirlas en la cláusula WHERE, lo que retorna en un error de compilación.

Las funciones de grupo son usadas comúnmente en combinación con la cláusula **GROUP BY**, con la que se divide las filas seleccionadas porque cumplen con las condiciones de filtro del WHERE, en grupos basados en los distintos valores de las columnas incluidas en el **GROUP BY**.

Dada la tabla de Estudiantes

Tabla 3.1 Estudiantes

| Legajo | Apellido | Carrera |
|---|---|---|
| 7898 | Messi | SIS |
| 6677 | García | SIS |
| 6644 | Bremen | SOF |
| 2123 | López | ABO |
| 5522 | Palermo | SOF |
| 2123 | López | ABO |
| 5221 | Vianni | ABO |
| 4512 | Suárez | SOF |

"**GROUP BY** columna" separa las filas del conjunto, en grupos de filas que tienen el mismo valor en la columna

Tabla 3.2 Tabla Estudiantes agrupada por Carrera

| Legajo | Apellido | Carrera |
|---|---|---|
| 7898 | Messi | SIS |
| 6677 | García | SIS |
| 6644 | Bremen | SOF |
| 5522 | Palermo | SOF |
| 4512 | Suárez | SOF |
| 3566 | Perez | ABO |
| 2123 | López | ABO |
| 5221 | Vianni | ABO |

La consulta:

```
SELECT carrera, count(*)
FROM estudiantes
GROUP BY carrera;
```

Nos da como resultado:

```
Carrera  COUNT(*)
-------------------------
ABO              3
SIS               2
SOF              3
```

En el ejemplo, se aplicó la función **COUNT(*),** que cuenta las filas seleccionadas y filtradas por la consulta y muestra el resultado del conteo.

Si se usa una función de grupo sin la cláusula **GROUP BY,** éstas se aplicarán a todo el conjunto de filas seleccionadas. Se usan las funciones de grupo también en la cláusula **HAVING,** para filtrar los grupos que no cumplen con las condiciones. **HAVING** filtra filas mientras que **WHERE** filtra Filas, antes de que se armen los grupos.

Todas las funciones de grupo ignoran los valores nulls; esto es, si en la columna Apellido se va a contar con COUNT(Apellido) y tiene en todas las filas valores null, esta función devolverá 0 (cero); si sólo en algunas tuviera nulls, contará todas las filas con valores distintos de null.

El asterisco en **COUNT(*)** le indica a la función que cuente todas las filas de la o las tablas que están en el FROM. Según la definición de una tabla relacional, ésta no puede tener una fila con todos valores null o fila nula.

Es posible anidar funciones de grupo, dada la tabla NOTAS (se muestran sólo algunas columnas):

Tabla 3: Tabla Notas

| Notas | | |
|---|---|---|
| Legajo | Id_Materia | Nota |
| 7898 | 1 | 7 |
| 6677 | 1 | 7 |
| 6644 | 2 | 8 |
| 2123 | 2 | 9 |
| 5522 | 3 | 10 |
| 2123 | 3 | 10 |
| 5221 | 5 | 2 |
| 4512 | 5 | 4 |

Para calcular el promedio de la nota más alta de cada materia, se puede hacer:

```
SELECT AVG(MAX(nota))
FROM examenes
GROUP BY id_materia;
AVG(MAX(nota))
----------------
7,5
```

Esta consulta evalúa el agrupamiento interno (**MAX**(nota)) trayendo las notas más altas de cada materia, las agrupa nuevamente y, luego, les calcula el promedio.

Tomó la nota máxima de cada material (7, 9, 10, 4) y las promedió (30/4)= 7,5

Algunas de las funciones de grupo más usadas son:

AVG (Average o promedio).
COUNT (conteo).
MAX (máximo).
MIN (mínimo).
STDDEV (desviación estándar).
SUM (suma).
VARIANCE (varianza).

### 3.9.1 La cláusula FROM

En la cláusula **FROM,** se enumeran las tablas, las vistas y las subconsultas que se consultarán para buscar las columnas que se enumeran en el **SELECT**. Cuando hay más de una referencia a tablas, se dice que la consulta es **MULTITABLA**. En este caso, se está frente a la aplicación de la operación Reunión (JOIN) del algebra relacional y, si no se escriben las condiciones de reunión en el WHERE, el optimizador de consultas realizará un Producto Cartesiano que relacionará todas las filas de las tablas reunidas, lo que generará resultados falsos: si se tuvieran cinco alumnos y tres exámenes rendidos, la reunión sin "condiciones de reunión", daría quince filas (5 x 3 filas). Esto produce un gran volumen de filas reunidas que, habitualmente, no se utilizarán.

Lo expresado en el párrafo precedente es en general, pero se observa que, para algunos casos, esta operación —Producto Cartesiano— será necesaria, por ejemplo, en el caso en que se requiera relacionar todos los jugadores de un torneo de tenis para armar los partidos posibles, teniendo en cuenta que se eliminarán los pares en los que se repiten los mismos jugadores. ¿Cómo sería esto? Supongamos que hay catorce empleados y se quiere que estén listos todos los partidos posibles (los imposibles serían, por ejemplo, que un empleado jugara contra sí mismo), para esto se puede hacer el Producto Cartesiano de la tabla empleados jugara consigo misma con distinto alias de tablas, para identificar a qué rol corresponde.

En el primer intento, dejará un Producto Cartesiano ya que se unirá la tabla empleados consigo misma. Esta forma de reunión se denomina, también, "Auto Join".

```
SELECT l.ename Local, ' juega con ',  v.ename Visita
FROM empleados l, empleados v
```

Si esta tabla tiene 14 filas, da un resultado de 196 filas (ver en la Figura 3.2, en la que se muestran, a modo ilustrativo, sólo las primeras).

En la siguiente consulta, se resolverá el requerimiento que armará los partidos de ajedrez en un campeonato interno de la empresa (ver Figura 3).

**SELECT** l.ename Local, ' juega con ', v.ename Visita
**FROM** empleados l, empleados v
**WHERE** l.ename <> v.ename

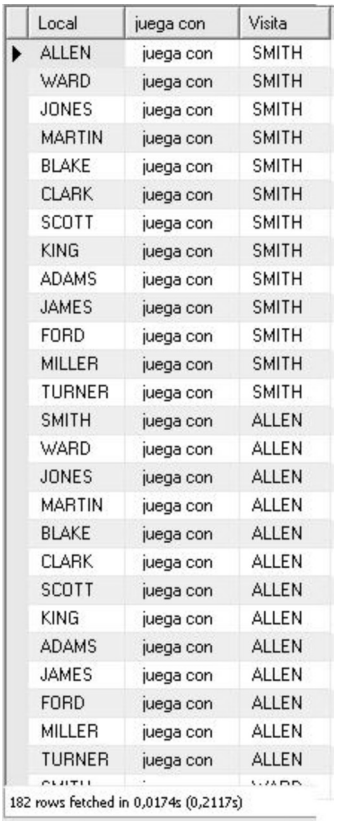
Si bien la intención no es adelantarse al próximo apartado, se observa que hay una cláusula **WHERE** con una condición que evita que se una al mismo empleado como local y como visitante.

Es importante destacar que se ha usado "Alias de Tablas" para identificar las columnas "ename" que, como vienen de la misma tabla, pero repetida en el **FROM**, se la ha calificado con la letra que se escribe, en esta cláusula, a continuación de la tabla. Esta forma de nombrar las columnas de una tabla, en una posición determinada, se denomina

"Alias de tabla" y, como recomendación, se usó la inicial del rol o del significado que tiene cada repetición de la tabla en el SELECT; para este caso, se adoptó la "l" para Local y la "v" para Visitante.

| Local | juega con | Visita |
|---|---|---|
| SMITH | juega con | SMITH |
| ALLEN | juega con | SMITH |
| WARD | juega con | SMITH |
| JONES | juega con | SMITH |
| MARTIN | juega con | SMITH |
| BLAKE | juega con | SMITH |
| CLARK | juega con | SMITH |
| SCOTT | juega con | SMITH |
| KING | juega con | SMITH |
| ADAMS | juega con | SMITH |
| JAMES | juega con | SMITH |
| FORD | juega con | SMITH |
| MILLER | juega con | SMITH |
| TURNER | juega con | SMITH |
| SMITH | juega con | ALLEN |
| ALLEN | juega con | ALLEN |
| WARD | juega con | ALLEN |
| JONES | juega con | ALLEN |
| MARTIN | juega con | ALLEN |
| BLAKE | juega con | ALLEN |
| CLARK | juega con | ALLEN |
| SCOTT | juega con | ALLEN |
| KING | juega con | ALLEN |
| ADAMS | juega con | ALLEN |
| JAMES | juega con | ALLEN |
| FORD | juega con | ALLEN |
| MILLER | juega con | ALLEN |

**Figura 3.1.** Producto cartesiano: se observa que es imposible que SMITH juegue contra SMITH; sin embargo, la operación lo permite.



| Local | juega con | Visita |
|---|---|---|
| ALLEN | juega con | SMITH |
| WARD | juega con | SMITH |
| JONES | juega con | SMITH |
| MARTIN | juega con | SMITH |
| BLAKE | juega con | SMITH |
| CLARK | juega con | SMITH |
| SCOTT | juega con | SMITH |
| KING | juega con | SMITH |
| ADAMS | juega con | SMITH |
| JAMES | juega con | SMITH |
| FORD | juega con | SMITH |
| MILLER | juega con | SMITH |
| TURNER | juega con | SMITH |
| SMITH | juega con | ALLEN |
| WARD | juega con | ALLEN |
| JONES | juega con | ALLEN |
| MARTIN | juega con | ALLEN |
| BLAKE | juega con | ALLEN |
| CLARK | juega con | ALLEN |
| SCOTT | juega con | ALLEN |
| KING | juega con | ALLEN |
| ADAMS | juega con | ALLEN |
| JAMES | juega con | ALLEN |
| FORD | juega con | ALLEN |
| MILLER | juega con | ALLEN |
| TURNER | juega con | ALLEN |

182 rows fetched in 0,0174s (0,2117s)

**Figura 3.2.** Reunión sin Producto cartesiano. Se agregó una restricción en el WHERE y se armó el conjunto de partidos posibles

Nótese que el resultado de 182 filas es porque se excluyeron los partidos entre un empleado contra él mismo; es decir: aquellos partidos 'imposibles' que totalizan 14 partidos.

Entonces, cuando en el **FROM** se unen las tablas, para evitar el producto cartesiano, se escribe una condición de reunión; generalmente, en estas consultas, se unen las tablas con claves foráneas y con claves primarias, que concretan la reunión entre tablas relacionadas. Pero esto no es excluyente, también se pueden escribir otras condiciones de reunión con columnas del mismo dominio de valores; es decir, comparables…

Para reunir las tablas, existen dos alternativas de sintaxis: la primera y más antigua, es nombrarlas en el **FROM** y, luego, se evitará el producto cartesiano con condiciones de reunión en el **WHERE**. En la segunda, se escribe la palabra **JOIN** entre las tablas y, después, en el mismo **FROM, se** escribirá la condición dentro del operador ON. De esta manera, se obtendrá el mismo resultado que la Figura 3.3:

**SELECT** l.ename Local, ' juega con ', v.ename Visita

**FROM** empleados l **JOIN** empleados v **ON** (l.ename <> v.ename)

Esto se considera como estándar desde SQL3. Se adjunta la ayuda de MySQL sobre la sintaxis de esta forma de escribir las reuniones de tablas.

```
  table_reference [INNER | CROSS] JOIN table_factor [join_condition]
| table_reference STRAIGHT_JOIN table_factor
| table_reference STRAIGHT_JOIN table_factor ON condition
| table_reference LEFT [OUTER] JOIN table_reference join_condition
| table_reference NATURAL [LEFT [OUTER]] JOIN table_factor
| table_reference RIGHT [OUTER] JOIN table_reference join_condition
| table_reference NATURAL [RIGHT [OUTER]] JOIN table_factor

join_condition:
  ON conditional_expr
| USING (column_list)
```

En la sintaxis, aparecen operadores que no se estudiarán en este capítulo.

### 3.9.2 La cláusula WHERE

En esta cláusula, se escriben las condiciones de filtro que permiten elegir aquellas filas que se quieren mostrar. La condición es una construcción que tiene tres partes en general y, como excepción, cuando se usa el operador de comparación EXISTS o NOT EXISTS, dos.

Una condición posee como elementos una columna, un operador de comparación y un valor constante, otra columna, una variable o una subconsulta que devuelva uno o varios valores, que se aceptan solamente si el operador es IN, NOT IN, EXISTS y NOT EXISTS, que son los que tratan con lista de valores.

Ejemplos de condiciones: “Una fila es elegida si…

**WHERE** sal > 1000 … es verdad que el salario de la fila es mayor a 1000”.

**WHERE** ename = ‘SMITH’ …es verdad que el apellido de la fila es ‘SMITH’ (en mayúsculas)”.

**WHERE** hiredate < ‘01/03/1998’ … es verdad que la fecha de contratación es anterior al primero de marzo de 1998”.

o negarlas con NOT,

“Una fila es elegida si …

**WHERE NOT** sal > 1000 … NO es verdad que el salario de la fila es mayor a 1000”.

También se pueden combinar las condiciones, lo que permite que se las evalúe, simultáneamente, con los coordinadores ‘AND’ y ‘OR’. De esta manera, se obtendrá que.

“Una fila es elegida si…

**WHERE** (sal > 1000) **AND** (ename = 'SMITH') … es verdad que el salario de la fila es mayor a 1000 y que tiene un valor en apellido igual a 'SMITH' (en mayúsculas).

**WHERE** hiredate < '01/03/1998' **OR** (ename = 'SMITH' ) … es verdad que la fecha de contratación es anterior al primero de marzo de ese año o si el valor de su columna ename es igual a 'SMITH' (en mayúsculas)".

Las cantidad de combinaciones no tiene límites en una cláusula WHERE, que es una de las competencias más usadas para entender esta cláusula y escribir condiciones que permitan encontrar todas las filas que deben encontrarse para cumplir con una tarea.

Los operadores de comparación son

**=** igual

**<** menor que

**>** mayor que

**<>** distinto que

**= ANY/ALL** compara con todos los valores de una lista, =ANY es equivalente a IN

**< ALL** es menor que todos los valores de una lista o subconsulta.

**> ALL** es mayor que todos los valores de una lista o subconsulta.

**< ANY** es menor que algunos de los valores de una lista o subconsulta.

**> ANY** es mayor que algunos de los valores de una lista o subconsulta.

**IN** es igual que al menos uno de los valores de una lista o subconsulta.

**NOT IN** No es igual que al menos uno de los valores de una lista o subconsulta.

**BETWEEN** límite inferior AND límite superior está dentro de un rango inclusivo.

**NOT BETWEEN** límite inferior AND límite superior está fuera de un rango inclusivo.

**LIKE** patrón como % (comodines de múltiples valores y múltiple cantidad) y _ Idem pero solo de una posición.

**NOT LIKE** no cumple con un patrón como % (comodines de múltiples valores y múltiple cantidad) y _ Idem pero sólo de una posición.

**EXISTS** test de existencia en una subconsulta.

**NOT EXISTS** test de No existencia en una subconsulta.

### 3.9.3 La cláusula GROUP BY

Como se vio en la explicación de las funciones de grupo, esta cláusula define sobre en qué valores de columna o columnas se basará el optimizador para agrupar las filas por sus distintos valores.

### 3.9.4 La cláusula HAVING

Una vez definidos los grupos, se puede escribir en HAVING una condición que usa las funciones de grupo para poder filtrar o seleccionar y mostrar sólo aquellos grupos que cumplen con las condiciones presentes de HAVING.

Se la suele comparar con la cláusula WHERE, porque aplica condiciones, pero que quede claro que, WHERE, filtra filas; HAVING, filtra grupos armados en GROUP BY.

### 3.9.5 La cláusula ORDER BY

La última cláusula de la sentencia SELECT es la que permite dar el orden deseado a las filas encontradas que cumplan la condición, junto con los valores de las funciones y otros cálculos o formateos realizados con esta sentencia. Es posible combinar varias columnas para ordenar, y el orden de aparición de estas columnas en la cláusula **ORDER BY** no es trivial, comenzará por la primera columna que aparece luego de la palabra BY.

Existen dos indicadores del sentido en el que se debe ordenar, **DESC** de descendiente o **ASC**, de ascendente. Si no se indica ninguno, se toma por defecto **ASC**.

Así, si se usara **DESC** como indicador del sentido del ordenamiento, mostrará de mayor a menor y, si es **ASC**, de menor a mayor.

## 3.10 Sentencias del sub lenguaje DML, Transacciones, Confirmación y Deshacer.

### 3.10.1 Sentencia INSERT

La primera sentencia, del grupo de sentencias del Data Management Language, es **INSERT**, que permite insertar filas en una tabla, e ingresar columna a columna los valores de cada una de ellas:

**INSERT INTO** empleados (empno, ename, job, mgr, hiredate, sal, comm, deptno)
**VALUES** (8000, ‘Alves, Pedro’, ‘Programa’, 7411, 3500, null, 20)

La lista de columnas se puede obviar, pero por claridad se sugiere no dejar de consignarla.

Otra forma para insertar filas en una tabla —y más de una por ejecución— es la que incluye un select dentro del Insert. A esta sentencia subordinada, habitualmente se la denomina Subconsultas:

**INSERT INTO** empleados_historico (empno, ename, job, mgr, hiredate, sal, comm, deptno, fecha_carga);

**SELECT** empno, ename, job, mgr, hiredate, sal, comm, deptno, current_date
**FROM** empleados
**WHERE** hiredate < ‘01/01/2000’;

En este caso, se parte del supuesto de que estamos insertando, en la tabla de empleados históricos, los empleados contratados antes del primero de enero de2000, y se agrega la fecha actual en la columna “fecha_carga” de la mencionada tabla. En

esta operación, se insertarán más de una fila en la tabla objetivo, que es una de las aplicaciones de las subconsultas que se verá más en detalle en las siguientes lecturas.

### 3.10.2 Sentencia UPDATE

La sentencia para cambiar el valor de una o más columnas en una tabla es **UPDATE** — que tiene la siguiente sintaxis en Oracle—, aumentando el sueldo de todos los empleados de la tabla:

UPDATE *tbl_name*
SET *col_name1=expr1* [, *col_name2=expr2* ...]
[WHERE *where_condition*];

Ejemplo:

```
UPDATE empleados
SET sal = sal * 1.2;
```

Esto provoca un aumento de sueldo de un 20% a todos los integrantes de la empresa; si se deseara que se le aumente a unos pocos, se utilizará la cláusula WHERE, que tiene las mismas características que la sentencia SELECT. Si, por ejemplo, a la empresa le interesara sólo aumentarle el sueldo a los que ganan entre 1000 y 1200, la sintaxis sería esta:

```
UPDATE empleados
SET sal = sal * 1.2
WHERE sal BETWEEN 1000 AND 1200;
```

En esta sentencia UPDATE, se pueden usar subconsultas como argumento del SET y como parte del WHERE, como en cualquier sentencia SELECT.

```
UPDATE empleados
SET sal = (SELECT AVG(sal) FROM empleados)
WHERE sal BETWEEN 1000 AND 1200;
```

En la sentencia anterior, se ha modificado el sueldo de los que ganan entre 1000 y 1200, por el valor del sueldo promedio de todos los empleados.

### 3.10.3 Sentencia MERGE

La sentencia MERGE es una combinación de INSERT y UPDATE y, con la sintaxis que contiene las dos sentencias, al ejecutarse, busca la fila; si la encuentra la modifica de acuerdo con la sintaxis del UPDATE y, si no existe ninguna fila que cumpla con el

criterio del WHERE, la inserta con la sintaxis del INSERT y, si una fila cumple con una condición, la puede borrar.

Ejemplo de la sintaxis en ORACLE 10g [Ora, 2009]:

```
MERGE INTO bonus b
USING (
  SELECT empno, sal, deptno
  FROM empleados
  WHERE deptno =20) e
ON (b.empno = e.empno)
WHEN MATCHED THEN
  UPDATE SET b.sal = e.sal * 0.1
  DELETE WHERE (e.sal < 40000)
WHEN NOT MATCHED THEN
  INSERT (b.ename, b.sal)
  VALUES (e.name, e.sal * 0.05)
  WHERE (e.sal > 40000);
```

### 3.10.4 Sentencia DELETE

Para borrar filas de una tabla, la sentencia DELETE es muy simple: alcanza con completar el nombre de la tabla y, opcionalmente, se puede incorporar la cláusula WHERE, con el funcionamiento que se vio en la sentencia SELECT.

```
DELETE emp1
WHERE sal > 5000;
```

Sin la cláusula WHERE y su condición, borraría todas las filas de la tabla; en cambio, con esa cláusula, en la columna sal, borrará aquellas filas con valores mayores a 5000.

## 3.11 Sentencias del sub lenguaje TCL de Control de Transacciones

### 3.11.1 Concepto de Transacciones

Todos los cambios de datos en una Base de datos se pueden hacer a través de una transacción. Estas transacciones se confirmarán con COMMIT o desharán con ROLLBACK. Estas dos sentencias cierran la transacción. El motor de la base de datos, si la transacción se confirma, asegurará que se encuentren los datos modificados. Si una transacción se deshace, el sistema asegurará que no quede trunca ni ningún dato modificado sin corregir, como si nunca se hubiera realizado el cambio que se borró.

En este cuadro, se observa una transacción [Ora, 2009]

Cuadro 3.1

```
UPDATE cust_accounts
  SET balance = balance - 1500
WHERE
  account_no = '70-490930.1';
COMMIT;
UPDATE cust_accounts
  SET balance = balance + 1500
WHERE
  account_no = '70-909249.1';
COMMIT;
```

Si este código se ejecuta y la base tiene un problema luego del primer COMMIT, las cuentas quedarán desbalanceadas, ya que se han debitado 1500 pesos de la primera cuenta.

En cambio, si se dejara solamente el último commit, estas sentencias se completarían o anularían conjuntamente, que impediría que quedaran desbalanceadas. [Ora, 2009]

### 3.11.2 Sentencia de confirmación (Commit)

La sentencia COMMIT confirma y guarda los cambios de la transacción en curso y libera los recursos bloqueados por cualquier actualización hecha con la transacción actual.

Cuadro 3.2

```
COMMIT [WORK];
```

Si ejecutamos:

Cuadro 3.3

```
DELETE FROM t_pedidos WHERE cod_pedido = 15;
COMMIT;
```

Borra un registro y se guardan los cambios.

### 3.11.3 Sentencia deshacer (Rollback)

**ROLLBACK** es la sentencia que permite deshacer la última transacción. Cuando se ejecuta, vuelve atrás las transacciones en todas las tablas hechas desde el último **COMMIT**. Su sintaxis es muy simple

```
ROLLBACK;
```

### 3.11.4 Sentencia Savepoint

En una transacción, se pueden determinar puntos de salvaguarda o **SAVEPOINTS**, que significa que se hace **ROLLBACK** desde el último punto, como un marcador. Su sintaxis es:

### 3.12.4 Inicio de transacción

```
----
----
SAVEPOINT inicio;
---
INSERT...
UPDATE ...
DELETE ...
...
ROLLBACK TO SAVEPOINT inicio;
```

y devuelve las transacciones desde el savepoint. Es importante mencionar que no se debe ejecutar un **COMMIT** entre el **SAVEPOINT** y el **ROLLBACK**, porque quedan anulados cuando se confirman los cambios.

## 3.12 Procesamiento de Consultas

Las aplicaciones de software funcionan de acuerdo con los requerimientos de negocio y, normalmente, se prueban con volúmenes de datos iniciales, que no dan una idea del crecimiento potencial de las cantidades de filas que se agregarán con el funcionamiento continuo de la operatoria normal del negocio. Esto llevará a que en un razonable período de tiempo, las aplicaciones comiencen a cumplir su tarea más lentamente, a medida que las tablas en las que se basa incrementen la cantidad de filas dentro de las tablas afectadas.

Esta lógica evolución necesita que se entienda cómo funciona el mecanismo de ejecución de consultas para, de esta manera, mejorar la *performance* del sistema en general. Por esta razón, se describirán brevemente los componentes del procesamiento de Consultas y las formas de resolver el plan de ejecución.

Existen diferentes métodos para determinar qué recursos usará el ejecutor para resolver una sentencia SQL y, así, obtener el resultado esperado. Con esta información, se puede decidir qué alternativa realizará el trabajo en el menor tiempo posible.

El ejecutor de sentencias SQL utiliza un proceso denominado optimizador, que se encarga de elegir el método de resolución para encontrar los datos requeridos por la sentencia. Este optimizador tiene dos estrategias o modos de funcionar:

El primero, orientado a la sintaxis con la que está escrito, tiene un *ranking* con las reglas de acceso a los datos. Este modo —llamado Optimizador— se basa en Reglas o RBO y, como se afirmó al comienzo de este párrafo, utiliza la forma de escritura de la sentencia y, con la información del diccionario sobre estructuras de datos que usará, determina el plan de ejecución para responderle. Esta estrategia se desarrolló

en el origen de las primeras bases de datos y como es estático en el modo de resolver el plan de ejecución, en muchos casos se reemplaza por otro que se apoya en las estadísticas de uso de los objetos y que se describe a continuación.

En el Modo Basado en Costos o CBO, el optimizador inspecciona cada sentencia e identifica los caminos posibles de acceso a los datos y establece los costos basado en la cantidad de lecturas lógicas que se realizarán para ejecutar la sentencia. Este modo emplea las estadísticas que se almacenan sobre los objetos afectados por la sentencia SQL para, de esta manera, determinar el plan de ejecución que demande menos recursos.

El modo del optimizador, que se aplicará en las bases de datos Oracle, se fija por parámetros generales o de la base, en el nivel de la sesión, y, también, para cada sentencia en particular.

### 3.12.1 Plan de Consultas

Para cada sentencia, el optimizador prepara un árbol de operaciones denominado "plan de ejecución" que define el orden y los métodos de operaciones que el servidor seguirá para resolverla.

En un motor Oracle existen numerosas herramientas que permiten la elección de un determinado plan de ejecución y la *performance* asociada. A continuación, se enumerarán los principales recursos:

En el paquete STATPACK, un conjunto de Programas obtiene la información de las estadísticas almacenadas sobre los objetos de la base de datos.

El comando SQL, EXPLAIN PLAN genera, en una sesión de base de datos, el plan que se ejecutará con una sentencia SQL dada.

El utilitario TKPROFtoma la salida de una sesión que está generando archivos de pistas de auditoría o TRACE y crea un archivo legible con la información que facilitará el estudio de las características del plan de ejecución.

También es posible contar con AUTOTRACE —una opción de SQL*Plus— que elabora un plan de ejecución y las estadísticas relativas a las operaciones que se realizarán para cumplir con la sentencia.

En una base de datos ORACLE con el cliente SQL*Plus se puede usar el comando EXPLAIN PLAN que, en vez de ejecutar una sentencia SQL, crea el plan de consultas en una tabla especial llamada PLAN_TABLE, que contendrá las operaciones que realizará al ejecutarse la sentencia. Por ejemplo:

```
SQL>    EXPLAIN PLAN FOR
2       SELECT apellido FROM estudiantes;
```

Se crea el plan y la consulta como se ilustra a continuación:

```
SQL>    SELECT *
2       FROM TABLE(dbms_xplan.display);
```

Este método de acceso a la PLAN_TABLE, vía el paquete provisto DBMS_XPLAN, es más efectivo ya que resume la información útil del resultado de EXPLAIN PLAN.

### 3.12.2 Optimización de Consultas

Con la información de los recursos utilizados por las distintas posibilidades de ejecución de una consulta determinada, será posible entender y elegir la manera de resolver la sentencia que mejor se adapte a las circunstancias requeridas y, eventualmente, se podría decidir si se tomarán medidas complementarias, como la creación de índices que optimicen los tiempos de acceso a la información requerida.

Como hay diferentes modos de ejecutar una sentencia SQL —por ejemplo: alterando el orden de acceso a una tabla o a un índice—, el desarrollador puede influir en la manera en la que el optimizador considere los distintos factores en un proceso en particular. Por ejemplo: incluir en la sentencia los HINTS, que son indicaciones específicas que el optimizador utilizará en la realización de determinadas tareas mediante la utilización de un índice especial o mediante la alteración del orden de acceso a las tablas de consulta.

El tratamiento detallado de las distintas técnicas excede el alcance de este libro. Por esta razón, se recomienda al lector que amplíe estos conceptos en la documentación oficial de los principales proveedores de Bases de datos y en las fuentes de información técnica especializada en la materia.