SQL: Lenguaje de consulta estructurado

Introducción

SQL (Structured Query Language), lenguaje de consulta estructurado permite realizar una serie de operaciones en bases de datos relacionales.
Se compone de tres partes o sublenguajes:

- LDD (DDL en inglés) o lenguaje de definición de datos.
- LMD(DML en inglés) o lenguaje de manipulación de datos.
- LCD(DCL en inglés) o lenguaje de control de datos.

Las primeras implementaciones de SQL se realizaron en 1975, con el nombre de SEQUEL.
En 1979, ORACLE implementa el primer sistema comercial de gestión de base de datos relacionales basado en SQl. Posteriormente aparecen otros sistemas basados en SQL como DB2, SYBASE, INFORMIX, INGRES, ADABAS.

En 1986, el grupo ANSI aprueba una primera normalización del lenguaje. En 1992 se aprueba una nueva versión de SQL conocida como **SQL-92** en la que se incrementa la capacidad semántica del esquema relacional y se añaden nuevos operadores. También, se incluyen normas para el SQL embebido.
En 1999 se aprueba una nueva versión conocida como **SQL2000** en el que entre otras cosas se agregan las expresiones regulares para ser usadas en búsquedas.
En los años 2003 y 2005 se aprueban dos revisiones que incorporan funcionalidades relacionadas con el uso de datos XML.
La última revisión hasta la fecha es de 2016, en ésta se incluyen funcionalidades relacionadas con el uso de datos en formato JSON.

Dicho esto, hay que decir que los productos comerciales no implementan de manera completa los estándares propuestos por el grupo ANSI, por tanto, habrá que leer detenidamente los manuales del producto comercial concreto. En esta unidad se estudiará el SQL de Oracle.

Características de SQL.

Sentencias SQL

Una sentencia SQL se compone de una palabra clave que la define y de una o varias cláusulas (algunas obligatorias, otras opcionales) que actúan sobre los objetos o elementos de la base de datos.

Podemos clasificar las sentencias SQL en tres tipos:

- **Sentencias de definición de datos.**

  Se corresponderían con el sublenguaje LDD (DDL). Permiten crear y borrar las tablas y vistas de la base de datos, así como modificar la estructura de las tablas.

- **Sentencias de manipulación de datos.**

  Se corresponderían con el sublenguaje LMD(DML). Permiten manipular y consultar la información contenida en las tablas.

- **Sentencias de control de datos.**

  Se corresponderían con el sublenguaje LCD (DCL). Se utilizan para gestionar la confidencialidad y seguridad de la base de datos (ej.: creación de usuarios y permisos). Las sentencias para el control de transacciones caerían también bajo este tipo.

Tipos de datos

El estándar ANSI define una serie de tipos de datos que suelen ser un subconjunto de los que incorporan la mayoría de los sistemas comerciales.
Enumeraremos primero los tipos de datos del estándar y luego comentaremos los que incorpora Oracle para su versión 11.

Tipos de datos estándar.

Numéricos: Pueden contener números con o sin punto decimal y signo.

- Números enteros. INTEGER
- Números enteros pequeños SMALLINT
- Números decimales DECIMAL(P,S) p= precisión, s=escala
- Números en coma flotante FLOAT(p)
- De baja precisión REAL
- De alta precisión DOUBLE PRECISION

Alfanuméricos: Almacenan cadenas de caracteres

- Cadenas de longitud fija CHAR(N) n= longitud
- Cadenas de longitud variable CHARACTER VARYING(N)

Tipos de datos en Oracle versión 11.

CHAR(N) Campo de longitud fija, con un máximo de 2000 caracteres. (No tenía este rango en versiones anteriores). La longitud es opcional si no se especifica nada la longitud es 1.

VARCHAR(N) En la actualidad idéntico a VARCHAR2, aunque su funcionalidad puede cambiar en versiones futuras. Por tanto, para almacenar cadenas de caracteres de longitud variable, debe utilizarse el tipo de datos VARCHAR2.

VARCHAR2(N) Alfanúmerico de longitud variable, con una longitud máxima de 4000 caracteres.

DATE Campo de longitud fija 7 bytes, que se utiliza para almacenar fechas. Un dato de tipo DATE almacena: el siglo, año, mes, día, hora, minutos y segundos de una fecha. Permiten almacenar fechas de la era Juliana en el rango: 1 de Enero de 4712 antes de Cristo hasta el 9999 de nuestra era.

NUMBER Almacenan números fijos y en coma flotante. Se manejan los siguientes rangos:

- Números positivos en el rango que va desde 1×10^{-130} hasta $9{,}99...9 \times 10^{125}$ con hasta 38 dígitos significativos.
- Números negativos en el rango que va desde -1×10^{-130} hasta $9{,}99...99 \times 10^{125}$ con hasta 38 dígitos significativos.
- Cero.

BINARY_FLOAT Numérico de simple precisión de 32 bits. Cada dato de este tipo requiere 5 bytes, incluyendo uno para la longitud.

BINARY_DOUBLE Numérico de doble precisión de 64 bits. Cada dato de este tipo require 9 bytes, incluyendo uno para la longitud.

LONG Campo de longitud variable, con una longitud máxima de 2Gb. Se mantienen por compatibilidad hacia versiones pasadas. Oracle recomienda ahora el uso de datos de tipo LOB (BLOB, CLOB, NLOB y BFILE)

LONG RAW Campo de longitud variable para datos binarios, de una longitud máxima de 2Gb. Se recomienda el uso de datos de tipo LOB.

BLOB Objeto binario no estructurado de gran tamaño, hasta 128 Terabytes, dependiendo del tamaño del bloque de la base de datos. Se pueden usar para guardar imágenes, videos, archivos de audio. Aunque es bastante normal usar el sistema de archivos del sistema operativo para almacenar este tipo de información, guardándose en la base de datos una referencia a la ubicación de dichos archivos.

CLOB Permite almacenar hasta 128 Terabytes de caracteres. Usa el conjunto de caracteres de la base de datos. Este conjunto de caracteres se puede definir en el proceso de instalación del software.

NLOB Permite almacenar hasta 128 Terabytes de caracteres. Usa el conjunto de caracteres Unicode. Este conjunto de caracteres se puede definir en el proceso de instalación del software.

BFILE El tipo de dato BFILE almacena datos binarios no estructurados en ficheros del sistema operativo, externos a la base de datos. Una column BFILE guarda un localizador del fichero que contiene los datos.

Por compatibilidad hacia versiones anteriores y al estándar ANSI Oracle mantiene los tipos numéricos vistos en el apartado tipos de datos estándar.

Operadores, expresiones y funciones

Describiremos en este apartado los principales operadores dejando para apartados posteriores las expresiones y funciones.

Operadores aritméticos (por orden de preferencia)	Descripción	Ejemplos
+ -	Proporciona carácter positivo o negativo a una expresión	-1
* /	Multiplicación y división	Pts*hora
+ -	Sumar y restar	Sueldo+comision

Operadores Relacionales	Descripción	Ejemplos
=	Comprobación de igualdad	Sueldo=100000
!= <>	Distinto de	Sueldo <> 100000
>	Mayor que	
<	Menor que	
>=	Mayor o igual que	
<=	Menor o igual que	
In	Pertenencia a una lista	Oficio in('analista','operador')
Not(in)	No pertenencia a una lista	Oficio not in('analista','operador')
Between a and b	Evalúa verdadero si el valor está comprendido entre a y b	Sueldo between 2000 and 5000
IS [NOT] NULL	Testea si un item es nulo	Direccion IS NULL

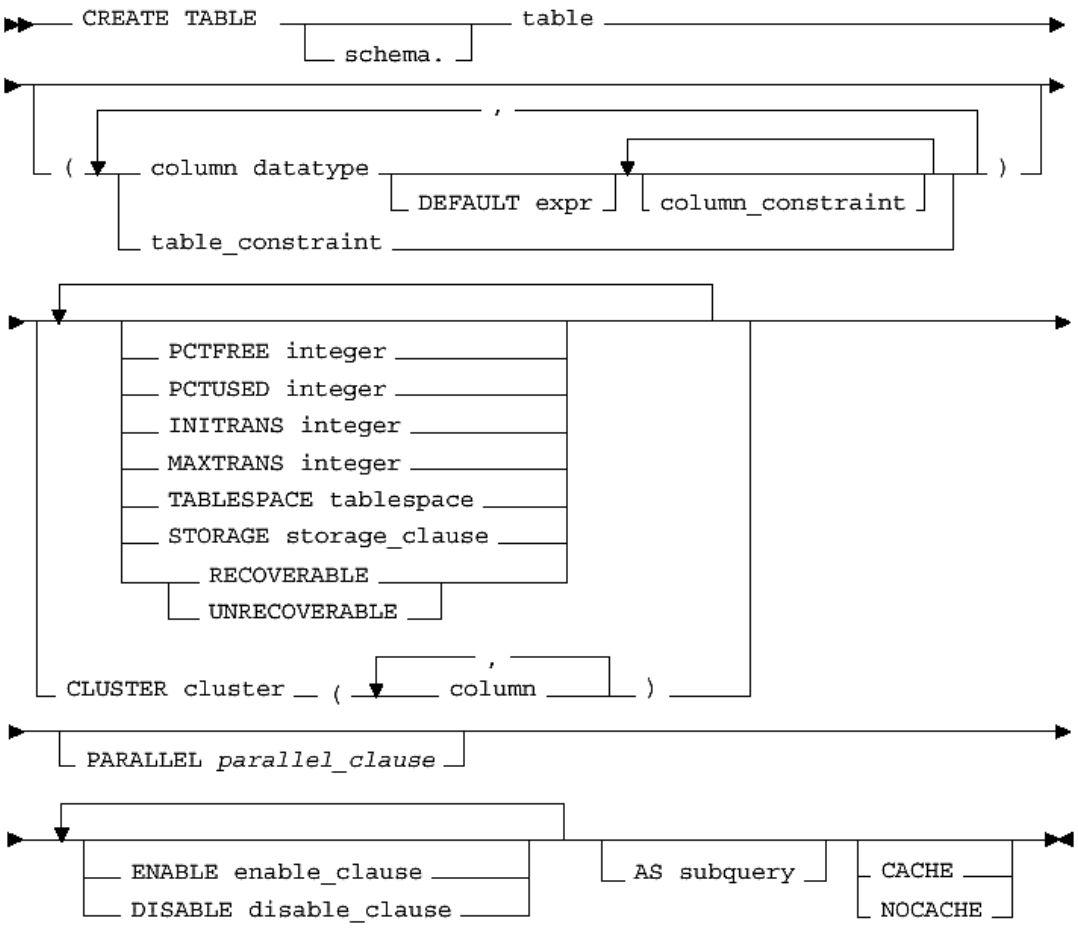
Operadores Logicos	Descripcion	Ejemplo
NOT	Evalúa verdadero si la condición es falsa	Not(sueldo>1000)
AND	Evalúa verdadero si las dos condiciones son verdaderas	(Sueldo>1000) AND (Comision >2000)
OR	Evalúa verdadero si una de las dos condiciones es verdadera	(Sueldo>1000) OR (Comision >2000)

Lenguaje de definición de datos (DDL)

Permite definir los distintos objetos de la base de datos(tablas, vistas, índices...).

Creación de tablas.

La sentencia SQL para crear una tabla es **CREATE TABLE**. Mediante ella podemos especificar la definición de columnas que describen la tabla, las restricciones de integridad, así como información relativa al almacenamiento de dicha tabla.
La sintaxis de la orden es la que se muestra a continuación mediante este grafo sintáctico



Donde:

Schema: Identifica el esquema que contendrá la tabla. Cada usuario de la base de datos tendrá un esquema con el mismo nombre de su usuario en el que podrá crear sus objetos (tablas, vistas, índices...) siempre que tenga permisos para ello.
Si un usuario quisiera crear una tabla en un esquema distinto deberá tener permisos adecuados para ello.
Decir que Oracle implementa el concepto de esquema relacional precisamente mediante esta cláusula de la orden. Así por ejemplo si nosotros quisiéramos implementar el esquema relacional correspondiente al modelado de una biblioteca, lo ideal sería crear un usuario llamado biblioteca (con los correspondientes permisos para poder crear tablas). Una vez hecho esto, trasladaríamos el esquema relacional obtenido en la fase de diseño, creando las tablas correspondientes a las relaciones que conforman el esquema relacional.

Table: Es el nombre de la tabla que se va a crear.

Column: Especifica el nombre de una columna de la tabla. Una tabla puede contener hasta 254 columnas. Se pueden omitir las columnas únicamente cuando la tabla se crea usando la cláusula de subconsulta.

Datatype: Es el tipo de datos de cada una de las columnas. Los tipos se han definido en el apartado 2.2 de esta unidad.

Default: Especifica un valor que será asignado a la columna cuando en una inserción en dicha tabla se omita la columna.

Column_Constraint: Define una restricción de integridad como parte de la definición de columna.

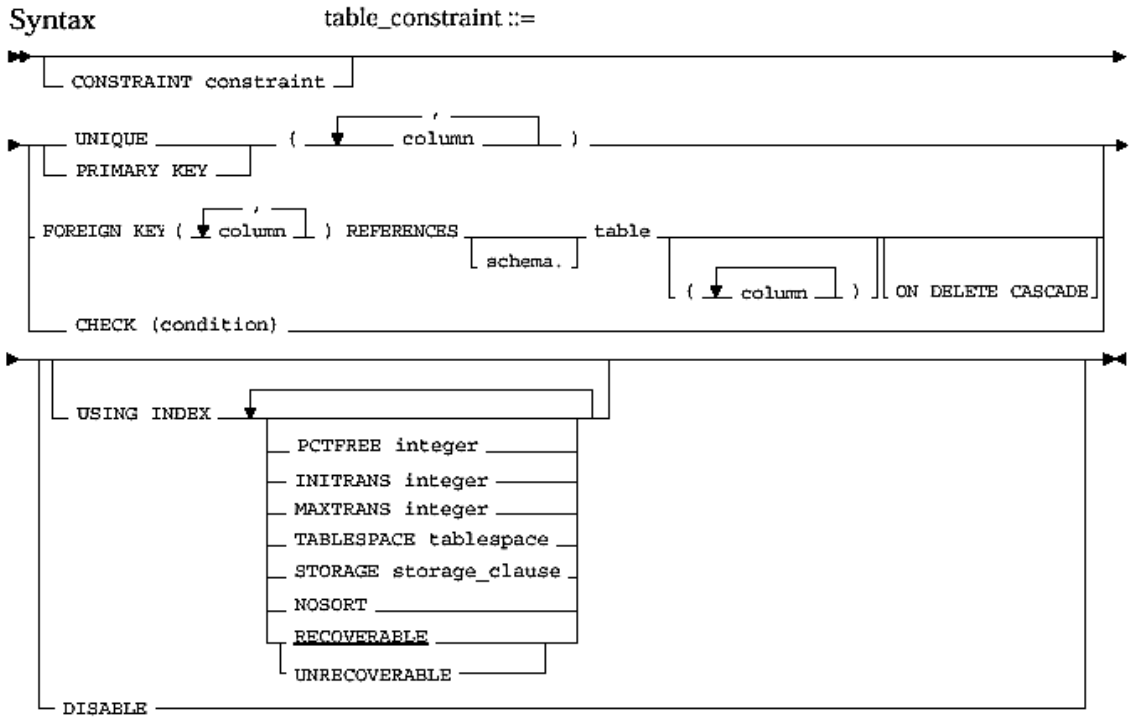
Table_Constraint: Define una restricción de integridad como parte de la definición de la tabla.

AS subquery: Inserta las filas devueltas por una subconsulta sobre la tabla que se está creando. Si se usa esta cláusula no es necesario definir las columnas de la tabla, ya que éstas tendrían el mismo nombre y el mismo tipo de datos que las obtenidas mediante la consulta.
El resto de parámetros de la orden están relacionados con las estructuras de almacenamiento y no se van a detallar. Es una labor más propia de los administradores de la base de datos.

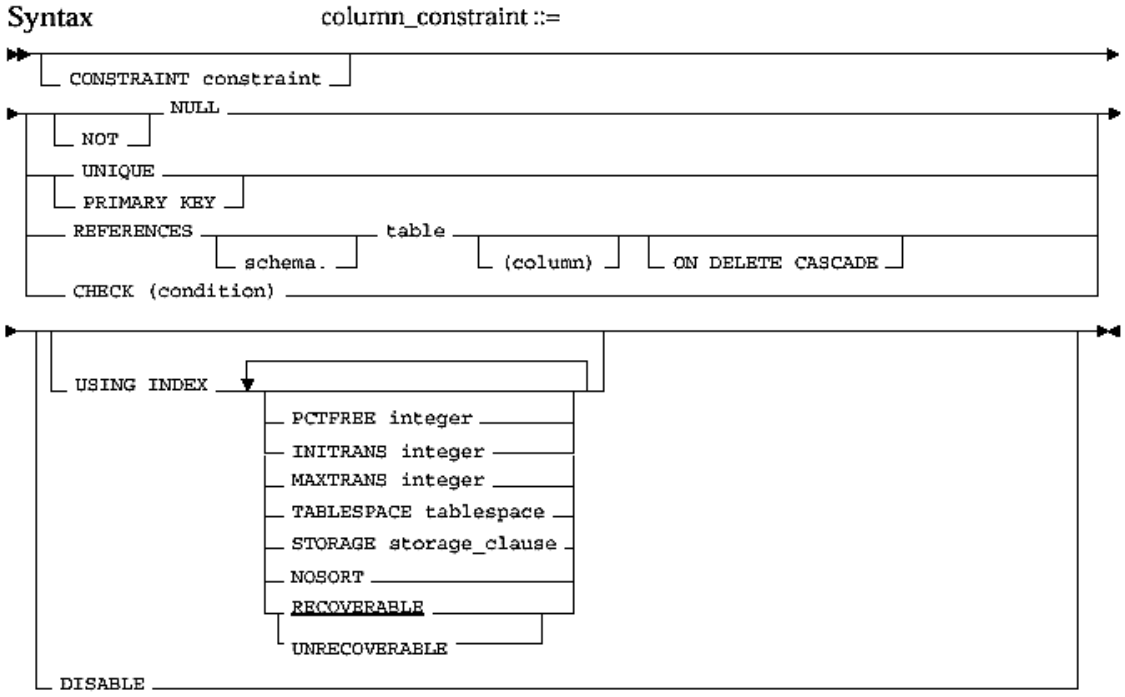
Restricciones de integridad.

En el apartado anterior hemos visto como crear tablas. Como parte de la sintaxis de la orden aparecían dos elementos (column_constraint y table_constraint) relacionados con la definición de las restricciones de integridad de una tabla. Definiremos en este apartado la sintaxis de cada uno de estos elementos que nos permitirán definir las restricciones de clave primaria, claves ajenas, y restricciones de verificación (claúsulas check). Estas restricciones las podemos escribir de dos formas al crear la tabla: bien como parte de la definición de la

columna (column_constraint) o al final de la definición de las columnas de la tabla (table_constraint). En algunos casos sólo es posible utilizar una de las dos alternativas, así por ejemplo si la clave primaria de una tabla es compuesta, lo deberemos indicar mediante una restricción de tabla ya que la de columna en este caso no lo permite.Sintaxis para la cláusula constraint a nivel de tabla:



Sintaxis para la cláusula constraint a nivel de columna



Significado de los parámetros:

Constraint: Sirve para dar un nombre a la restricción de integridad. Oracle almacena el nombre en el diccionario de datos. Si se omite Oracle asigna un identificador propio para dar nombre a la restricción. Este identificador es de la forma: SYS_Cn donde n es un entero que asegura la unicidad dentro de la base de datos. Es interesante asignar un identificador propio, ya que éste formará parte de los mensajes de error que nos devolverá Oracle cuando se intente insertar datos en las tablas que violen la restricción.

Null: Especifica que una columna puede tomar valores nulos. Es la opción por defecto y no hay que especificarlo.

Not null: Especifica que una columna no puede contener valores nulos.

Unique: Especifica que una columna (constraint a nivel de columna) o una combinación de ellas (constraint a nivel de tabla) es una clave alternativa.

Primary Key: Designa una columna o combinación de columnas como clave primaria de la tabla.

Foreign key: Designa una columna o combinación de columnas como clave como clave ajena en una restricción de integridad referencial.

References: Identifica la clave primaria o ajena que es referenciada por una clave ajena en una restricción de integridad referencial. Aquí Oracle a diferencia de otros SGDB admite que la clave ajena referencia a claves alternativas.

On delete cascade: Especifica que Oracle mantiene la integridad referencial borrando automáticamente los valores de clave ajena dependiente si se hace un borrado de la clave primaria o ajena referenciada. Si esta opción no se especifica Oracle no dejara borrar la clave primaria referenciada si ésta tiene claves ajenas relacionadas.

Check: Especifica una condición que cada fila de la tabla debe satisfacer.

Veamos a continuación algunos ejemplos de uso:

Ejemplo1:

```
CREATE TABLE CLIENTE(
      NUM NUMBER(4) CONSTRAINT PK_CLIENTE PRIMARY KEY,
      DNI VARCHAR2(15)  CONSTRAINT UQ_DNICLIENTE UNIQUE,
      NOMBRE VARCHAR2(50),
      FECHAALTA DATE DEFAULT SYSDATE
 );
```

En este ejemplo las restricciones de clave primaria y clave alterna se han definido utilizando el formato de restricción a nivel de columna.

Ejemplo2:

```
CREATE TABLE CLIENTE(
      NUM NUMBER,
      DNI VARCHAR2(15),
      NOMBRE VARCHAR2(50),
      FECHAALTA DATE DEFAULT SYSDATE,
      CONSTRAINT PK_CLIENTE
            PRIMARY KEY(NUM),
      CONSTRAINT UQ_CLIDNIUNICO
            UNIQUE(DNI)
);
```

La misma tabla con el formato de restricción a nivel de tabla.

Ejemplo3:

Las siguientes definiciones de tablas corresponderían a un esquema en el cual una persona puede tener varias aficiones y una afición la puede tener más de una persona, esto es, una relación N:M .

```
CREATE TABLE PERSONA(
 COD_PER NUMBER(4),
 NOMBRE VARCHAR2(20),
      CONSTRAINT PK_PERSONA
            PRIMARY KEY(COD_PER)
)
;
CREATE TABLE AFICION(
COD_AF NUMBER(4),
      DESCRIPCION VARCHAR2(30),
      CONSTRAINT PK_AFICION
            PRIMARY KEY(COD_AF)
)
;
CREATE TABLE AFICIONESPERSONALES(
      PERSONA NUMBER(4),
      AFICION NUMBER(4),
      CONSTRAINT PK_AFICIONESPERSONALES
            PRIMARY KEY(PERSONA,AFICION),
      CONSTRAINT FK_AFICIONESPERSONALES_AFICION
            FOREIGN KEY (AFICION)
            REFERENCES AFICION(COD_AF) ON DELETE CASCADE,
      CONSTRAINT FK_AFICIONESPERSONALES_PERSONA
            FOREIGN KEY(PERSONA)
            REFERENCES PERSONA(COD_PER) ON DELETE CASCADE
)
;
```

En la tabla aficionespersonales la restricción de clave primaria es obligatoria hacerla con el formato de restricción de tabla por ser una clave primaria compuesta.

Ejemplo4:

Crearemos utilizando la cláusula as subquery una copia de la tabla empleado.

```
CREATE TABLE COPIAEMPLEADO
AS SELECT * FROM EMPLEADO;
```

Restricciones de verificación (Check)

La restricción check define explícitamente una condición. Para satisfacerla, cada fila en la tabla debe hacer la condición verdadera o desconocida (debido a un valor null). La condición de una restricción check puede referirse a cualquier columna de la tabla, pero no puede referenciar a columnas de otras tablas. Las restricciones check no admiten las siguientes construcciones:

- Consultas sobre valores de otras filas
- Llamadas a las funciones SYSDATE, UID, USER o USERENV
- Las seudocolumnas CURRVAL, NEXTVAL, LEVEL o ROWNUM
- Constantes de fecha que no estén completamente especificadas.

Al contrario de otras restricciones una restricción check definida con la sintaxis de column_constraint puede imponer reglas sobre cualquier columna de la tabla.
Siempre que Oracle evalúa una restricción check para una fila particular, cualquier nombre de columna que aparece en la condición se referirá a los valores de la columna de esa fila.

Ejemplos de uso de la restricción check:

Ejemplo1:

```
CREATE TABLE EMPLEADO(
      NUMEMP NUMBER(4),
      NOMBRE VARCHAR2(15),
      SEXO CHAR CONSTRAINT CK_EMPSEXO CHECK(SEXO IN('V','M')),
      FECHANAC DATE,
      FECHAING DATE,
      EMPLEO VARCHAR2(20),
      CONSTRAINT PK_EMPLEADO
            PRIMARY KEY(NUMEMP),
      CONSTRAINT CONTROLFECHAS_EMPLEADO
            CHECK((MONTHS_BETWEEN(FECHAING,FECHANAC)/12)>=18)
);
```

En esta tabla aparecen dos restricciones check una a nivel de columna para controlar que el sexo de un empleado pueda tomar únicamente los valores V o M y otra a nivel de tabla que permite controlar que un empleado tenga cumplidos 18 años cuando ingresa en la empresa.

Ejemplo 2:

Este ejemplo se establece la siguiente restricción en la definición de las tablas:
Si el tipo de canal es 'menu' el atributo menu no puede tomar valores nulos y el atributo coste deberá tomar valores nulos.
Si el tipo de canal es 'independiente' el atributo menu deberá tomar valores nulos mientras que el atributo coste no podrá tomar valores nulos.

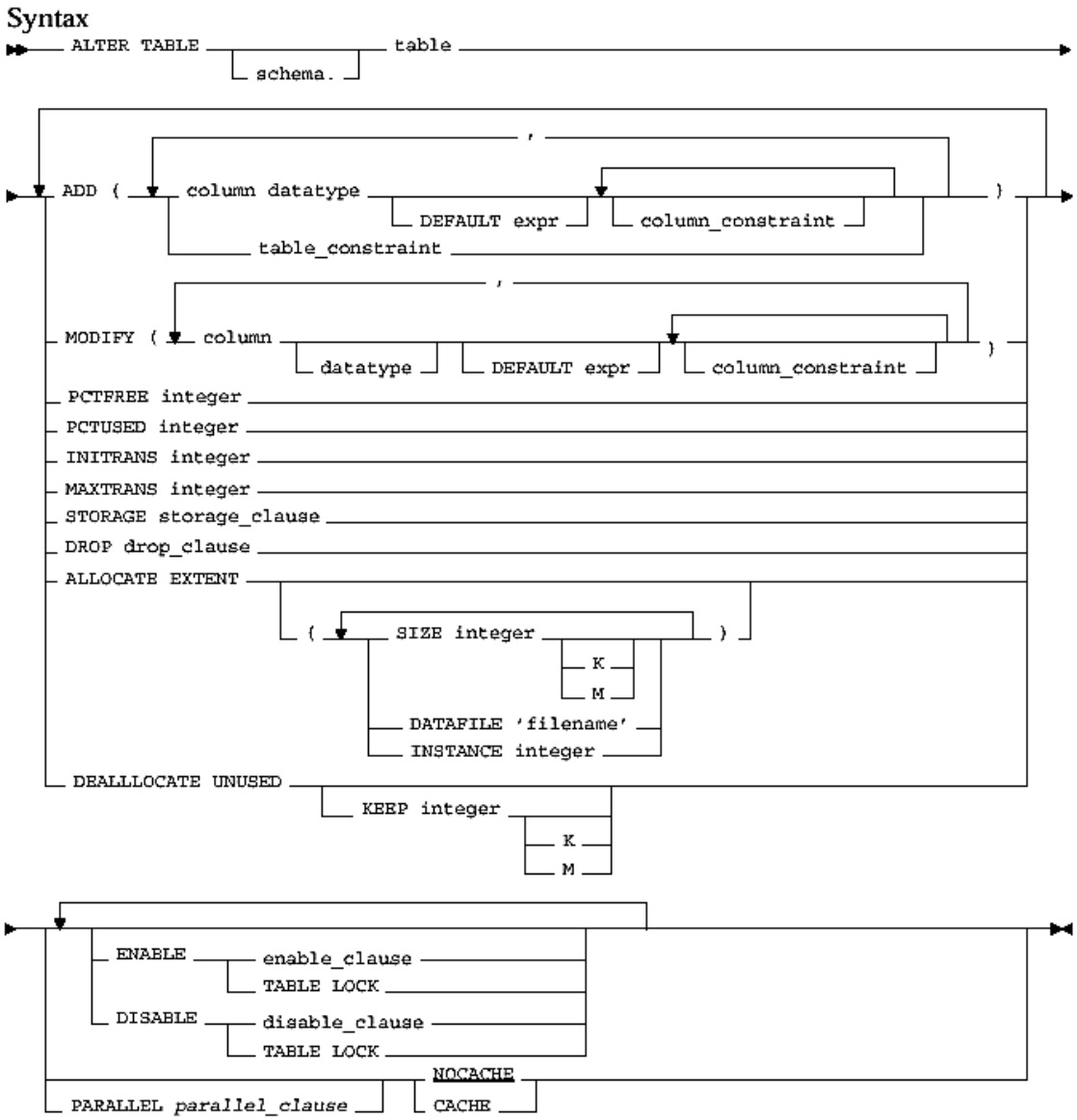
```
CREATE TABLE MENU(
      COD_MENU NUMBER(2),
      DESCRIPCION VARCHAR2(15),
      COSTE NUMBER(4,2),
      CONSTRAINT PK_MENU
            PRIMARY KEY(COD_MENU)
)
;
CREATE TABLE CANAL(
      COD_CANAL NUMBER(3) CONSTRAINT PK_CANAL PRIMARY KEY,
      DESCRIPCION VARCHAR2(15),
      TIPO VARCHAR2(13)
      CONSTRAINT CHECK_TIPO_CANAL
         CHECK (TIPO IN('MENU','INDEPENDIENTE')),
      COSTE NUMBER(4,2),
      MENU NUMBER(2),
      CONSTRAINT FK_CANAL
      FOREIGN KEY(MENU)
            REFERENCES MENU(COD_MENU),
      CONSTRAINT CHECK_ATRIBOBLIGA_CANAL
       CHECK((TIPO='MENU'AND MENU IS NOT NULL AND COSTE IS NULL)
                  OR
      (TIPO='INDEPENDIENTE'AND MENU IS NULL AND COSTE IS NOT NULL))
)
;
```

Modificación de tablas

Es posible alterar la definición de una tabla de una de las siguientes formas:

- Añadir una columna
- Añadir una restricción de integridad
- Eliminar una restricción de integridad
- Redefinir una columna (tipo de dato, tamaño, valor por defecto)

La sentencia que nos permite hacerlo es **ALTER TABLE.** La sintaxis de la orden es la siguiente:



Significado de los parámetros:

- **Table**: Nombre de la tabla que va a ser alterada.
- **Add**: Permite añadir una columna o restricción de integridad
- **Modify**: Modifica la definición de una columna ya existente.
- **Column**: Nombre de la columna que va a ser añadida o modificada.
- **Datatype**: Especifica un tipo de dato para una nueva columna o un nuevo tipo de dato para una columna ya existente.
- **Drop**: Permite eliminar una restricción de integridad.

Ejemplo1:

Queremos alterar la tabla empleado para añadir el campo sueldo con un tipo de dato number(6,2).

```
ALTER TABLE empleado
      add(sueldo number(6,2));
```

Ejemplo2:

Vemos que esta especificación se queda pequeña.

```
ALTER TABLE empleado modify(sueldo number(7,2));
```

Ejemplo3:

Queremos añadir una restricción a nivel de tabla para que el atributo sueldo sea mayor que 900.

```
ALTER TABLE EMPLEADO
     add constraint ck_sueldominimo check(sueldo>900)
```

Ejemplo4:

Como no podemos modificar restricciones si yo quisiera ahora que el sueldo fuese siempre mayor que 1000 haríamos lo siguiente:

Paso1: Primero eliminamos la restricción

```
ALTER TABLE empleado
    drop constraint ck_sueldominimo
```

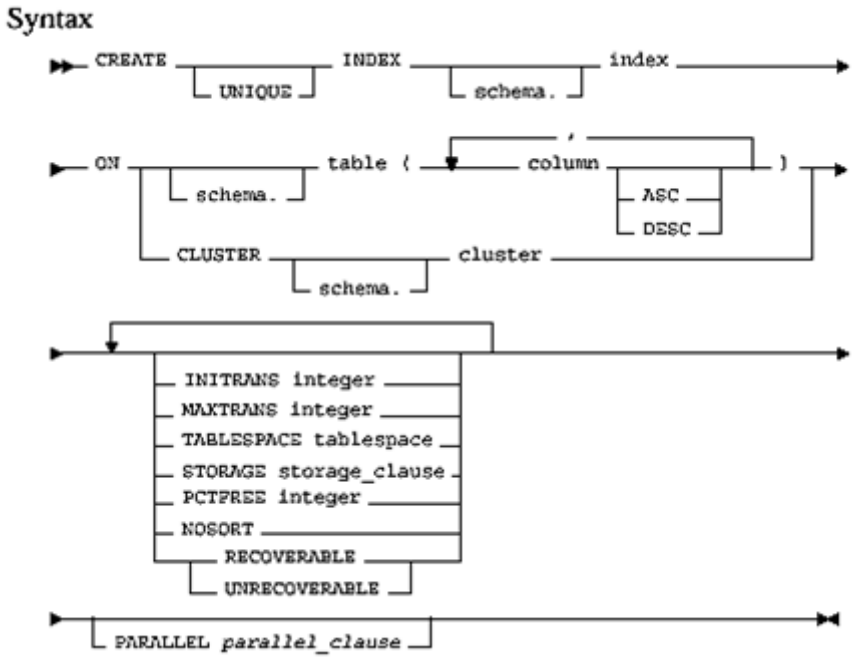
Paso2: Recreamos la restricción check

```
ALTER TABLE EMPLEADO
      add constraint ck_sueldominimo check(sueldo>1000)
```

Si al intentar activar esta nueva restricción hay filas que la incumplen Oracle informará de ello y habría que modificar los valores de estas filas para poder activar la nueva restricción.

Creación de índices.

Un índice es un concepto muy sencillo. Se trata de una lista de palabras clave acompañada de la localización de la información sobre un determinado tema. Por ejemplo, en algunos libros suelen aparecer al final una lista de palabras indicando el número o números de página en la que se encuentran referencias a dichas palabras. De esta manera el lector tiene una forma más rápida de acceder a la información.
Este mismo concepto lo podemos aplicar a los índices que Oracle crea sobre las tablas. Oracle permite crear índices sobre una o varias columnas de las tablas para acelerar la búsqueda de una fila en particular. Una vez creado el índice Oracle actúa como el lector en el libro, busca en el índice la localización exacta de la fila sobre la que se hace una consulta obteniendo la información de una manera más rápida.
Las columnas que aparecen frecuentemente en una cláusula where de una select son buenas candidatas para definir un índice sobre ellas. De igual manera, las consultas que combinan dos tablas se aceleran si las columnas que están relacionadas (por la cláusula where) tienen definido un índice.
Oracle indexa de manera automática la columna o columnas definidas como clave primaria de una tabla.
Sintaxis de la orden:



Significado de los parámetros principales:

- **Unique:** Especifica que el valor de la columna o combinación de columnas en la tabla indexada debe ser único. Aunque la tabla no contenga una restricción unique Oracle la hará cumplir por haberlo especificado en la creación del índice.
- **Schema:** Identifica el esquema que contendrá la tabla.
- **Index:** Es el nombre del índice que va a ser creado.
- **Table:** Es el nombre de la tabla para la cual el índice se crea.

- **Column:** Es el nombre de la columna de la tabla que se indexa. Un índice puede contener hasta 16 columnas. La columna indexada no puede ser del tipo long o long row.
- **Asc, Desc** se permiten por compatibilidad con DB2 aunque los índices siempre se crean en orden ascendente.

Oracle permite ver los índices creados mediante la vista user_indexes

Ejemplo:

```
CREATE UNIQUE  INDEX canal_descripcion on canal(descripcion);
```

Borrado de objetos.

Para borrar cualquier objeto de la base de datos(tablas, índices, usuarios, vistas...) se emplea siempre la misma orden. El formato genérico de esta orden es:
DROP *tipo_objeto [schema.]nombre_objeto*

Borrado de tablas.

Para el caso de tablas admite un modificador más:

```
DROP TABLE [schema.] table [CASCADE CONSTRAINTS]
```

La opción cascade constraints elimina todas las restricciones de integridad referencial referidas tanto de clave primaria como de clave única sobre la tabla borrada. Si esta opción se omite y existiera una restricción de integridad del tipo anterior, Oracle devuelve un error y no borra la tabla.

Ejemplos de uso:

Para borrar la tabla empleado escribiríamos:

```
DROP TABLE empleado
```

Si quisiéramos borrar la tabla menú escribiríamos:

```
DROP TABLE menu CASCADE CONSTRAINTS.
```

La opción drop table borra la información y la definición de la tabla.
Existe otra orden que elimina las filas, pero mantiene la tabla.

La orden es **TRUNCATE TABLE** nombre_tabla.

Borrado de índices.

DROP INDEX nombre_indice
Ejemplo:
DROP INDEX canal_descripcion

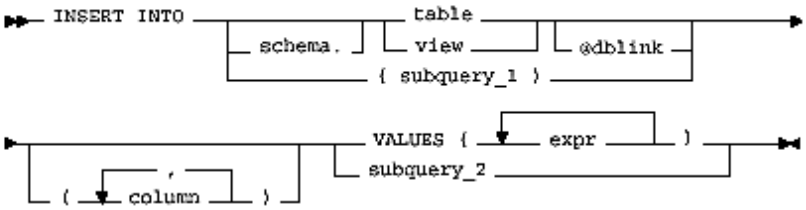
Lenguaje de manipulación de datos (DML)

Una vez que sabemos crear las tablas es preciso tener métodos para introducir información (**INSERT**), modificar la información almacenada (**UPDATE**), borrar información (**DELETE**) y obtener información (**SELECT**).

Inserción de datos.

La orden para insertar filas en una tabla es INSERT. La sintaxis de la orden es la siguiente:

Syntax



Schema: Es el esquema (usuario) que contiene la tabla o vista. Si se omite Oracle admite que la tabla o vista está en el esquema del usuario que hace la inserción.

Table, view: Es el nombre de la tabla donde las filas van a ser insertadas. Si se especifica vista, Oracle insertará las filas en la tabla base de la vista.

Column: Es una columna de la tabla o vista. En la fila insertada, cada columna de la lista se le asigna un valor de la cláusula VALUES o de la subconsulta. Si la lista de columnas se omite los valores se asociarán según el orden que aparecía en la definición de la tabla.

VALUES: Especifica una fila de valores para ser insertados dentro de la tabla o vista. Se debe especificar un valor en la cláusula VALUES para cada columna especificada en la lista de columnas. Si la lista de columnas se omitiera habría que especificar un valor para cada una de las columnas de la tabla, en caso contrario Oracle devolvería un error.

Subquery_2: Es una subconsulta que devuelve filas que serán insertadas en la tabla. La lista seleccionada en la subconsulta debe tener el mismo número de columnas que la lista de columnas de la sentencia INSERT.

Ejemplos de uso:

Supuesta esta definición de tablas:

```
CREATE TABLE ACTOMEDICO(
      COD_ACT NUMBER(2),
      DESCRIPCION VARCHAR2(15),
      PRECIOMEDICO NUMBER,
      CONSTRAINT PK_ACTOMEDICO
            PRIMARY KEY(COD_ACT)
);
CREATE TABLE COMPANIA(
      COD_COMP NUMBER(2),
      NOMBRE VARCHAR2(15),
      CONSTRAINT PK_COMPANIA
            PRIMARY KEY(COD_COMP)
)
;
CREATE TABLE TARIFA(
      COD_ACT NUMBER(2),
      COD_COMP NUMBER(2),
      PRECIO NUMBER,
      CONSTRAINT PK_TARIFA
            PRIMARY KEY(COD_ACT,COD_COMP),
      CONSTRAINT FK_TARIFA_COMPANIA
            FOREIGN KEY(COD_COMP) REFERENCES COMPANIA(COD_COMP)
)
;
```

Ejemplo1:

Vamos a añadir un nuevo acto médico

```
INSERT INTO ACTOMEDICO VALUES(1,'ACTOMEDICO1',7500);
```

Ejemplo2:

Si no quisiéramos dar valores a todas las columnas especificamos en la sentencia sobre que columnas queremos añadir valores. Supongamos que el médico todavía no tiene decidido el precio para el acto médico 'ACTOMEDICO2'. En este caso la inserción la haríamos así:

```
INSERT INTO ACTOMEDICO(COD_ACT,DESCRIPCION) VALUES(2,'ACTOMEDICO2');
```

En este caso la columna preciomedico quedaría con valor NULL.
Esto se puede hacer siempre que la columna preciomedico no esté definida con una restricción NOT NULL.

Ejemplo3:

Queremos añadir ahora la siguiente información: la compañía 'COMPAÑÍA UNO' cubre el actomedico 'ACTOMEDICO UNO' y paga por ello al médico que se lo aplica a un paciente 60€. La orden sería la siguiente:

```
INSERT INTO TARIFA VALUES(1,1,6500);
```

Ejemplo4:

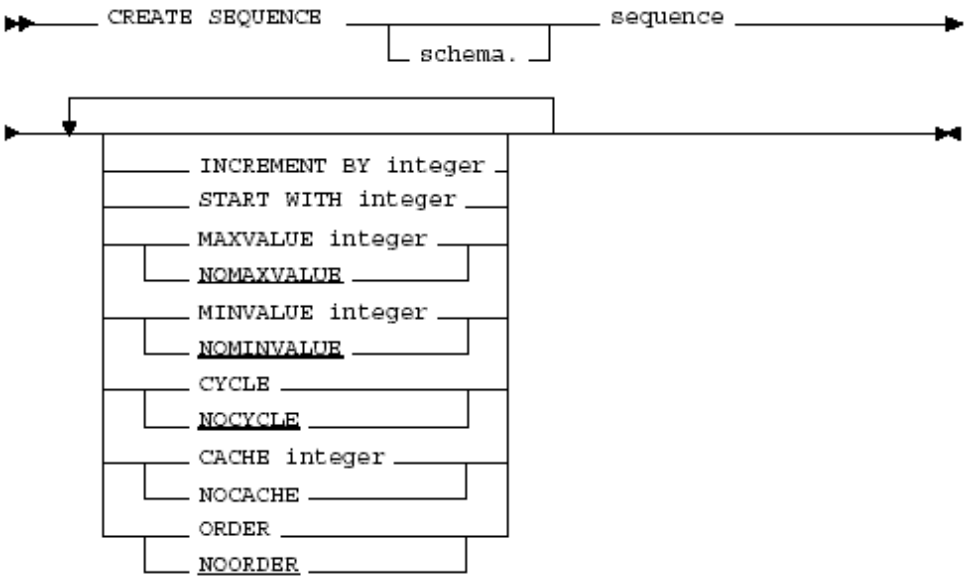
Supongamos que tenemos la tabla tarifa vacía. Supongamos también que todas las compañías con las que trabaja el médico cubren todos los actos médicos, cada una con un precio distinto. El hecho de asociar a cada compañía todos los actos médicos se convertiría en una tarea bastante tediosa. Con la cláusula subquery es posible hacer esto en una sola orden:

```
INSERT INTO TARIFA(COD_ACT,COD_COMP)
SELECT COD_ACT,COD_COMP
FROM
ACTOMEDICO,COMPANIA
;
```

Introduciremos en esta sección un nuevo objeto de la base de datos que puede ser útil asociado a la inserción de datos.

Secuencia

Una secuencia es un objeto de la base de datos, que permite a múltiples usuarios generar números enteros únicos. Podemos usar las secuencias para generar automáticamente valores de clave primaria. La orden que permite crear una secuencia es **CREATE SEQUENCE**, para poder llevarla a cabo debemos tener el privilegio del sistema **CREATE SEQUENCE**, si vamos a crear el objeto en nuestro esquema o bien el privilegio **CREATE ANY SEQUENCE** nos permitirá crear una secuencia en cualquier esquema.
El formato se muestra en el siguiente grafo sintáctico:



Donde:

- **Schema:** Es el esquema que contendrá la secuencia. Si se omite Oracle creará la secuencia en nuestro propio esquema.

- **Sequence:** Es el nombre de la secuencia. Tened en cuenta que el espacio de nombres para las secuencias es el mismo que para las tablas, vistas, procedimientos, paquetes, etc.

- **Increment by:** Fija el intervalo entre la secuencia de números. El incremento puede ser negativo o positivo.

- **Minvalue:** Especifica el menor valor para la secuencia.

- **Nominvalue:** Especifica un mínimo de 1 para secuencias ascendentes y $-(10^{26})$ para una secuencia descendente. Es el valor por defecto

- **Maxvalue:** Especifica el valor máximo que una secuencia puede generar. Puede tener 28 dígitos o menos.

- **Nomaxvalue:** Establece un valor máximo de 10^{27} para una secuencia ascendente o –1 para una descendente. Es el valor por defecto.

- **Start with:** Especifica el primer número de la secuencia a generar.

- **Cycle:** Establece que la secuencia continuará generando valores después de alcanzar el valor máximo, volviendo a comenzar por el valor mínimo después de esto. Si el valor mínimo no se ha especificado comenzará por 1 para secuencias ascendentes.

- **Nocycle:** La secuencia no generará más valores después de alcanzar el máximo. Es el valor por defecto.

- **Cache:** Especifica cuantos valores de secuencia Oracle prealoja en memoria para un acceso más rápido.

- **Nocache:** Para indicar que los valores de secuencia no se prealojan en memoria. Si se omite el parámetro cache y nocache oracle prealoja 20 números en memoria.

- **Order:** Garantiza que la secuencia de números se genera en orden de petición.

- **Noorder:** No garantiza que la secuencia se hará en orden de petición.

Ejemplo 5:

Creamos la secuencia SEQ_CODACT, comenzará a generar valores empezando en 3 con un incremento de uno.

```
CREATE SEQUENCE SEQ_CODACT
START WITH 3
INCREMENT BY 1
;
```

Una vez que se ha creado la secuencia, podemos acceder a los valores a través de las seudocolumnas **NEXTVAL**, para obtener el siguiente valor de la secuencia y **CURRVAL**, para obtener el valor actual.

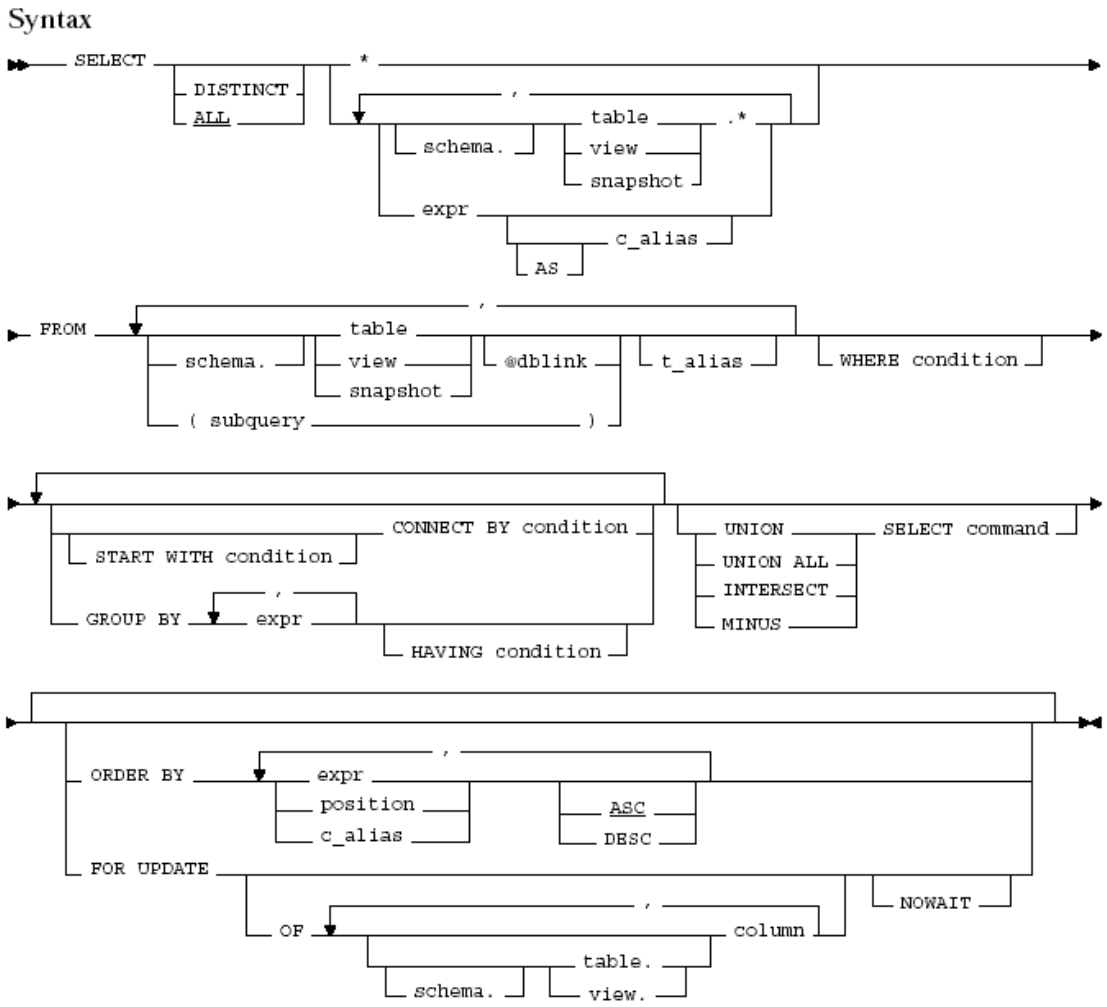
Ejemplo 6:

Como hasta ahora solo teníamos 2 actos médicos, hemos impuesto que la secuencia comience por 3.
Podemos generar ahora un nuevo acto médico utilizando la secuencia SEQ_CODACT para generar un nuevo valor de clave primaria para la columna cod_act.

```
INSERT INTO ACTOMEDICO VALUES (SCOD_ACT.NEXTVAL,'ACTOMEDICOTRES',10500);
```

Consulta de datos. (SELECT)

La sentencia select permite recuperar datos de una o más tablas, o vistas. Para poder seleccionar datos de una tabla o vista, ésta deberá estar en nuestro esquema o bien deberemos tener el privilegio de selección sobre la tabla , o sobre la tabla base asociada a la vista.
La sintaxis de la orden es la siguiente:



La sentencia select es compleja y permite muchas posibilidades. Para ir viéndolas nos apoyaremos en distintos esquemas en los que podremos explotar las distintas posibilidades.

El primer esquema en el que nos vamos a apoyar viene dado por el siguiente script:

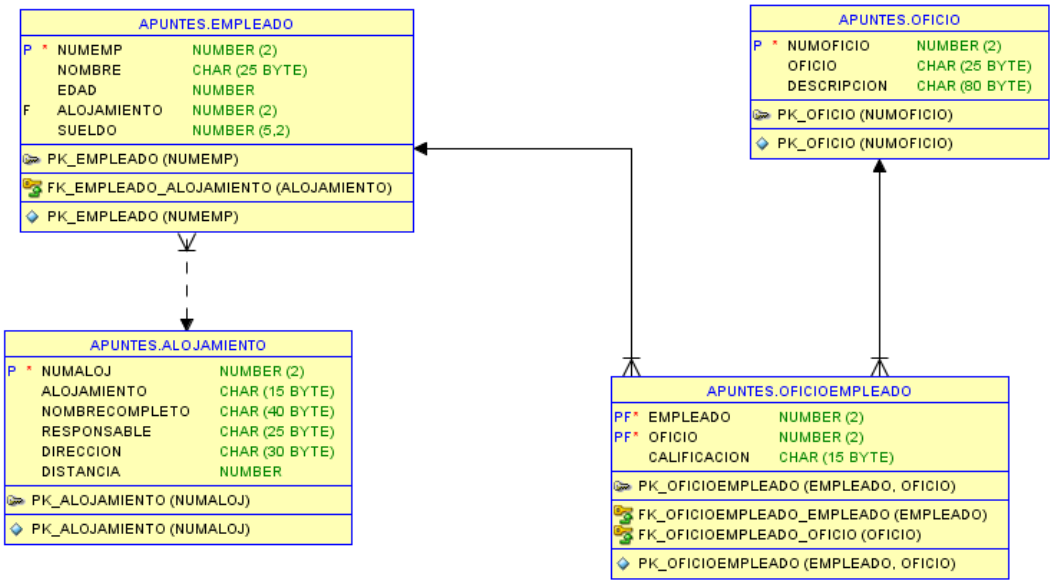
```
CREATE TABLE ALOJAMIENTO(
      NUMALOJ NUMBER(2),
      ALOJAMIENTO CHAR(15),
      NOMBRECOMPLETO CHAR(40),
      RESPONSABLE CHAR(25),
      DIRECCION CHAR(30),
      DISTANCIA NUMBER,
      CONSTRAINT PK_ALOJAMIENTO
            PRIMARY KEY (NUMALOJ)
)
;
CREATE TABLE OFICIO(
      NUMOFICIO NUMBER(2),
      OFICIO CHAR(25),
      DESCRIPCION CHAR(80),
      CONSTRAINT PK_OFICIO
            PRIMARY KEY (NUMOFICIO)
)
;
CREATE TABLE EMPLEADO(
      NUMEMP NUMBER(2),
      NOMBRE CHAR(25),
      EDAD NUMBER,
      ALOJAMIENTO NUMBER(2),
      SUELDO NUMBER(5,2),
      CONSTRAINT PK_EMPLEADO
            PRIMARY KEY(NUMEMP),
      CONSTRAINT FK_EMPLEADO_ALOJAMIENTO
            FOREIGN KEY(ALOJAMIENTO)
            REFERENCES ALOJAMIENTO(NUMALOJ)
)
;
CREATE TABLE OFICIOEMPLEADO(
      EMPLEADO NUMBER(2),
      OFICIO NUMBER(2),
      CALIFICACION CHAR(15),
      CONSTRAINT PK_OFICIOEMPLEADO
            PRIMARY KEY(EMPLEADO,OFICIO),
      CONSTRAINT FK_OFICIOEMPLEADO_EMPLEADO
            FOREIGN KEY(EMPLEADO)
            REFERENCES EMPLEADO(NUMEMP),
      CONSTRAINT FK_OFICIOEMPLEADO_OFICIO
            FOREIGN KEY(OFICIO)
            REFERENCES OFICIO(NUMOFICIO)
)
;
```

En el que se reflejan los datos de los empleados de una empresa. Los supuestos semánticos reflejados en este esquema son los siguientes:

- Un empleado puede tener cero o varios oficios.
- Un oficio lo pueden tener varios empleados.
- Un empleado vive en un determinado alojamiento.
- En un alojamiento pueden vivir varios empleados.

El esquema Entidad/Relación es el siguiente:

Una vez normalizado obtenemos el siguiente esquema relacional, implantado en el gestor Oracle. El grafo relacional puede obtenerse mediante el programa SqlDeveloper. El esquema relacional muestra la relación que existe entre las diferentes tablas que conforman nuestro esquema.



Las tablas tienen los siguientes datos:

TABLA ALOJAMIENTO

NUMALOJ	ALOJAMIENTO	NOMBRECOMPLETO	RESPONSABLE	DIRECCION	DISTANCIA
1	CRAMMER	CRANMER RETREAT HOUSE	THOM CRANMER	HILL ST BERKELEY	32
2	MATTS	MATTS LONG BUNK HOUSE	ROLAND BRANDT	3 MILE RD KEENE	25
3	MULLERS	MULLERS COED LODGING	KEN MULLER	120 MAIN EDMESTON	10
4	PAPA KING	PAPA KING ROOMING	WILLIAM KING	127 MAIN EDMESTON	10
5	ROSE HIL	ROSE HILL FOR MEN	JOHN PELETIER	RFD 3 N.EDMESTON	12
6	WEITBROCHT	WEITBROCHT ROOMING	EUNICE BENSON	320 GENEVA KEENE	26

TABLA EMPLEADO

NUMEMP	NOMBRE	EDAD	ALOJAMIENTO	SUELDO
1	ADAH TALBOT	23	4	1
2	ANDREW DYE	29	5	0,75
3	BART SARJEANT	22	1	0,75
4	DICK JONES	18	5	1
5	DONALD ROLLO	16	2	0,75
6	ELBERT TALBOT	43	6	
7	GEORGE OSCAR	41	5	1,25
8	GERHARDT KENTGEN	55	4	1,5
9	HELEN BRANDT	15		1,5
10	JED HOPKINS	33	2	1,75
11	JOHN PEARSON	27	5	4
12	KAY Y PALMER WALLBOM		5	
13	PAT LAVAY	21	5	1,25
14	PETER LAWSON	25	1	1,25
15	RICHARD KOCH Y HERMANOS		6	
16	ROLAND BRANDT	35	2	0 ,5
17	VICTORIA LYNN	32	3	1
18	WILFRED LOWELL	67		0,5
19	WILLIAM SWING	15	1	1

TABLA OFICIO

NUMOFICIO	OFICIO	DESCRIPCION
1	LEÑADOR	MARCAR Y TALAR ARBOLES,CORTAR,APILAR
2	CONDUCTOR DE SEGADORA	CONDUCIR,AJUSTAR MAQUIA
3	HERRERO	PREPARAR FUEGO,CORTAR,HERRAR CABALLOS
4	PICADOR	CAVAR,PICAR,PALEAR,TAPAR
5	LABRADOR	APAREJAR CABALLOS,PROFUNDIDAD DE REJA
6	OBRERO	TRABAJO EN GENERAL

TABLA OFICIOEMPLEADO

EMPLEADO	OFICIO	CALIFICACION
1	6	BUENO
4	3	EXCELENTE
6	5	LENTO
9	2	MUY RAPIDO
11	2	
11	1	BUENO
11	3	NORMAL
17	3	PRECISO
18	6	NORMAL
18	5	NORMAL

A partir de los datos de estas tablas iremos construyendo distintas consultas para aprovechar las distintas posibilidades de la orden select.

Recuperación de los datos de una sola tabla.

```
SELECT * FROM <nombre_tabla>
```

Seleccionaría todo el contenido de una tabla, es decir, todas sus filas con todas sus columnas.

Ejemplo 1:

Para seleccionar todos los datos de la tabla empleado escribiríamos:

```
SELECT * FROM EMPLEADO;
```

Si no queremos recuperar la información de todas las columnas especificaremos las columnas deseadas después de la palabra select, separando cada columna excepto la última mediante una coma.

Ejemplo 2:

Recuperar el nombre y la edad de todos los empleados.

```
SELECT NOMBRE,EDAD FROM EMPLEADO;
```

Podemos restringir el número de filas devueltas imponiendo una condición mediante la cláusula **where**.

Ejemplo 3:

Obtener el nombre y edad para aquellos empleados que tengan más de 40 años.

```
SELECT NOMBRE,EDAD
FROM EMPLEADO
WHERE EDAD > 40;
```

También podemos obtener que el resultado de la consulta salga ordenado en función de la columna o columnas que deseemos

Ejemplo 4:

Obtener el nombre y edad para aquellos empleados que tengan más de 40 años ordenados de menor a mayor edad.

```
SELECT NOMBRE,EDAD
FROM EMPLEADO
WHERE EDAD > 40
ORDER BY EDAD;
```

Por defecto la cláusula order by ordena de menor a mayor. Si quisiéramos obtener la información anterior ordenada de mayor a menor edad escribiríamos:

```
SELECT NOMBRE,EDAD
FROM EMPLEADO
WHERE EDAD > 40
ORDER BY EDAD DESC;
```

Con estos cuatro primeros ejemplos tendríamos el formato básico de la orden select para una tabla.

```
SELECT columna1,....,columnan
FROM tabla
WHERE condicion
ORDER BY columna1[asc,desc],....
```

Se realiza una proyección sobre las columnas que aparecen a continuación de select, (recordar que con * haríamos mención a todas las columnas de la tabla).

- **Tabla** es el nombre de la tabla de la que cogemos los datos.
- **Where** condición es la condición que deben satisfacer las filas para que aparezcan en la selección. La condición se construirá mediante los operadores relacionales y los operadores lógicos.
- **Order by** especifica la columna o columnas por las que queremos que se ordenen las filas devueltas.

Existen algunos operadores interesantes que se pueden usar en la condición de la cláusula **where**:

Operador Between:

Para expresar un intervalo de valores.

Ejemplo 5:

Obtener los nombres y edades de los empleados cuya edad este sea mayor o igual que 40 y menor o igual que 50.

```
SELECT NOMBRE,EDAD
FROM EMPLEADO
WHERE EDAD BETWEEN 40 AND 50;
```

Esa consulta la podíamos haber escrito también así:

```
SELECT NOMBRE,EDAD
FROM EMPLEADO
WHERE EDAD>=40 AND EDAD<=50;
```

Operador like

Este operador permite comparar patrones. **Like** permitirá buscar en las filas, columnas que cumplan un cierto patrón. Se usan dos tipos diferentes de caracteres para señalar el tipo de comparación que se desea hacer:

- El símbolo de % llamado comodín, representa una serie de espacios o caracteres
- El símbolo de subrayado representa una posición.

Ejemplo 6:

Seleccionar los nombres de los empleados cuya tercera letra sea una C. (Cuidado porque se distinguen entre mayúsculas y minúsculas).

```
SELECT NOMBRE
FROM EMPLEADO
WHERE NOMBRE LIKE '__C%';
```

Ejemplo 7:

Obtener los nombres de los empleados cuyo nombre empieza por 'WIL'

```
SELECT NOMBRE
FROM EMPLEADO
WHERE NOMBRE LIKE 'WIL%';
```

Operador IS [NOT] NULL

Comprueba si el valor de una columna es o no es nulo.

Ejemplo 8:

Queremos obtener los nombres de los empleados sobre los que no tenemos almacenada su edad.

```
SELECT NOMBRE
FROM EMPLEADO
WHERE EDAD IS NULL;
```

Operador IN [NOT IN]

Actúa igual que los operadores lógicos pero trabajando sobre una lista de valores.

Ejemplo 9:

Obtener los nombres y edad de todos los empleados cuya edad es 22, 23, 25, o 35 años.

```
SELECT NOMBRE,EDAD
FROM EMPLEADO
WHERE EDAD IN(22,23,25,35)
```

Recuperación de información procedente de más de una tabla

En la mayor parte de las consultas necesitaremos datos procedentes de más de una tabla. Esto es una consecuencia de haber normalizado nuestros datos.
Tendremos en estos casos dos formas de trabajar:
una de ellas será utilizar **subconsultas**(consulta dentro de otra consulta) y la otra será utilizar la **combinación** de varias tablas.

Combinación de varias tablas (Join).

Supongamos que queremos obtener el nombre del empleado y el nombre del alojamiento en el que están hospedados. Para obtener esta información se necesitan la tabla empleado y la tabla alojamiento.

Ejemplo 10:

```
SELECT NOMBRE,ALOJAMIENTO.ALOJAMIENTO
FROM EMPLEADO,ALOJAMIENTO
WHERE EMPLEADO.ALOJAMIENTO=NUMALOJ;
```

Siempre que en la cláusula **from** escribamos más de un nombre de tabla, se realizará el **producto cartesiano** de las dos o más tablas.

Si en el **ejemplo 10** hubiésemos omitido la cláusula where habríamos obtenido el producto cartesiano de las dos tablas proyectado sobre las columnas nombre(cuyo origen es la tabla empleado) y alojamiento (cuyo origen es la tabla alojamiento), es decir tendríamos una tabla en la que cada empleado estaría relacionado con todos los alojamientos.
Este sería el resultado ordenado por el nombre del empleado y el nombre del alojamiento. Observad como cada empleado se combina con todos los alojamientos disponibles.

```
SELECT NOMBRE,ALOJAMIENTO.ALOJAMIENTO
FROM EMPLEADO,ALOJAMIENTO
ORDER BY NOMBRE,ALOJAMIENTO.ALOJAMIENTO
```

Resultado de la consulta:

NOMBRE	ALOJAMIENTO
ADAH TALBOT	CRAMMER
ADAH TALBOT	MATTS
ADAH TALBOT	MULLERS
ADAH TALBOT	PAPA KING
ADAH TALBOT	ROSE HILL
ADAH TALBOT	WEITBROCHT
ANDREW DYE	CRAMMER
ANDREW DYE	MATTS
ANDREW DYE	MULLERS
ANDREW DYE	PAPA KING
ANDREW DYE	ROSE HILL
ANDREW DYE	WEITBROCHT
BART SARJEANT	CRAMMER
BART SARJEANT	MATTS
BART SARJEANT	MULLERS
BART SARJEANT	PAPA KING
…….	…….

De todos los alojamientos con los que se ha combinado un empleado sólo nos interesa aquel en el que realmente está hospedado y que viene impuesto por la columna alojamiento de la tabla empleado.

La columna alojamiento de la tabla empleado es una clave ajena cuyos valores permitidos son los de la columna numaloj de la tabla alojamiento. Esa es la información en común que comparten ambas tablas; por tanto, si en la cláusula where imponemos la columna alojamiento de la tabla empleado sea igual a la columna numaloj de la tabla alojamiento obtendremos realmente el alojamiento de cada empleado.

```
SELECT NOMBRE,ALOJAMIENTO.ALOJAMIENTO
FROM EMPLEADO,ALOJAMIENTO
WHERE EMPLEADO.ALOJAMIENTO=NUMALOJ;
```
(Clave ajena igual a clave primaria)

Una última aclaración con respecto a la consulta anterior:

Vemos que la columna alojamiento de la cláusula select viene calificada con el nombre de tabla (hemos escrito alojamiento.alojamiento). La razón es que está columna pertenece a dos tablas (empleado y alojamiento), para que el analizador de la orden SQL sepa que es lo que realmente queremos sacar hemos de especificarlo calificando la tabla origen de los datos. Si no lo especificamos ORACLE nos retornará el siguiente error:

column ambiguously defined.

Podemos escribir esta misma consulta mediante lo que comúnmente se denomina "Inner Join", haciendo uso del operador Join. La consulta anterior se escribiría así:

```
SELECT NOMBRE,ALOJAMIENTO.ALOJAMIENTO
FROM EMPLEADO JOIN ALOJAMIENTO
ON EMPLEADO.ALOJAMIENTO=ALOJAMIENTO.NUMALOJ
```

La condición se escribe mediante la cláusula ON. Si queremos filtrar las filas mediante alguna condición adicional, añadiríamos la cláusula where.

Si en el listado anterior, solo queremos mostrar aquellos empleados cuyo nombre empiece por G, añadiríamos la cláusula WHERE para obtener el resultado deseado.

```
SELECT NOMBRE,ALOJAMIENTO.ALOJAMIENTO
FROM EMPLEADO JOIN ALOJAMIENTO
ON EMPLEADO.ALOJAMIENTO=ALOJAMIENTO.NUMALOJ
WHERE EMPLEADO.NOMBRE LIKE 'G%'
```

Ejemplo 11:

Obtener el nombre y alojamiento de los empleados hospedados en 'CRAMMER'

```
SELECT NOMBRE,ALOJAMIENTO.ALOJAMIENTO
FROM EMPLEADO,ALOJAMIENTO
WHERE EMPLEADO.ALOJAMIENTO=NUMALOJ
AND ALOJAMIENTO.ALOJAMIENTO='CRAMMER';
```

Mediante el operador Join.

```
SELECT NOMBRE,ALOJAMIENTO.ALOJAMIENTO
FROM EMPLEADO JOIN ALOJAMIENTO
ON EMPLEADO.ALOJAMIENTO=NUMALOJ
WHERE ALOJAMIENTO.ALOJAMIENTO='CRAMMER';
```

En esta consulta no haría falta sacar el nombre del alojamiento que ya sabemos que es CRAMMER. Sacamos esta columna para comprobar el buen funcionamiento de la consulta.

Subconsultas.

Hemos dicho antes que para obtener consultas que afecten a más de una tabla teníamos dos formas de trabajar. Aquí presentamos la segunda de ellas.

La condición que aparece en la cláusula where puede tomar este aspecto:
Column=subquery, donde column es igual a una columna de una tabla y subquery es una subconsulta que genera un valor simple.

La sintaxis de una subconsulta (ver grafo sintáctico) es prácticamente idéntico al de una consulta. La diferencia esencial es que la subconsulta está dentro de otra consulta. Aprovechando este formato de la orden, vamos a reescribir la consulta anterior mediante subconsulta.

Ejemplo 12:

Obtener el nombre de los empleados hospedados en 'CRAMMER'

Vamos a hacerlo por pasos:

Paso 1:

Primero nos enteraremos de cuál es el valor de numaloj para el alojamiento 'CRAMMER'.

```
SELECT NUMALOJ
FROM ALOJAMIENTO
WHERE ALOJAMIENTO='CRAMMER';
```

Si ejecutáramos esta consulta obtendriamos 1.
Es decir todos los empleados alojados en 'CRAMMER' tienen un 1 como valor de la columna alojamiento.

Paso 2:

Si ahora escribiéramos:

```
SELECT NOMBRE
FROM EMPLEADO
WHERE ALOJAMIENTO=1;
```

Obtendríamos el resultado requerido.

Uniendo los dos pasos anteriores en una única consulta obtendremos el mismo resultado utilizando el formato de subconsulta

```
SELECT NOMBRE
FROM EMPLEADO
WHERE ALOJAMIENTO=(SELECT NUMALOJ
                   FROM ALOJAMIENTO
                   WHERE ALOJAMIENTO='CRAMMER');
```

Para cada empleado se comprueba si su alojamiento es igual al número de alojamiento de 'CRAMMER'.

Si la expresión viene dada por el operador '=', la subconsulta debe devolver un valor simple, en caso contrario ORACLE daría el siguiente error:

```
ORA-01427: la subconsulta de una sola fila devuelve más de una fila
01427. 00000 -  "single-row subquery returns more than one row"
```

La cláusula where de una subconsulta a su vez puede ser otra subconsulta con lo cual podremos anidar subconsultas, lo que nos permitirá obtener consultas más complejas.

Subconsultas con los operadores de comparación in, any, some , exists y all

Ejemplo 13:

Supongamos ahora que queremos obtener los nombres de los empleados que están alojados en 'MULLERS' o 'PAPA KING'. Realizaremos esta consulta mediante la técnica de subconsultas utilizando los operadores anteriores. La consulta puede escribirse de la siguiente manera:

```
SELECT NOMBRE
FROM EMPLEADO
WHERE ALOJAMIENTO= ANY(SELECT NUMALOJ
                       FROM ALOJAMIENTO
                       WHERE ALOJAMIENTO IN ('MULLERS','PAPA KING'))
```

La subconsulta en este caso retorna más de una fila, (los códigos correspondientes a los dos alojamientos solicitados).

No podemos igualar alojamiento al resultado de la subconsulta ya que obtendríamos el error anteriormente citado(**single-row subquery returns more than one row.**).

Utilizamos en este caso el operador any que permite comparar alojamiento con la lista de valores obtenida en la subconsulta, si alojamiento es igual a algún valor de la lista la expresión se evalúa como **verdadera**; si la subconsulta no retorna ninguna fila o alojamiento no coincide con ningún valor de la subconsulta la condición se evalúa como **falsa**.

=Any, in y =some son operadores equivalentes.

La consulta funcionaría de igual forma si en lugar de any escribimos el operador some. También obtendríamos lo mismo si la formulamos en función del operador in. En este último caso la consulta tiene una pequeña diferencia en la escritura. La consulta anterior escrita con in quedaría:

```
SELECT NOMBRE
FROM EMPLEADO
WHERE ALOJAMIENTO IN (SELECT NUMALOJ
                      FROM ALOJAMIENTO
                      WHERE ALOJAMIENTO IN ('MULLERS','PAPA KING'))
```

Exists es un operador relacional que se evalúa como cierto si la subconsulta devuelve al menos una fila. Para escribir la consulta anterior utilizando el operador exists hay que hacer algún pequeño cambio en la escritura de la sentencia. La misma consulta con exists quedaría de la siguiente forma:

Ejemplo 13-bis

Obtener los nombres de los empleados de los que tenemos algún oficio registrado.

```
SELECT NOMBRE
FROM EMPLEADO
WHERE EXISTS(SELECT EMPLEADO
             FROM OFICIOEMPLEADO
             WHERE OFICIOEMPLEADO.EMPLEADO=EMPLEADO.NUMEMP)
```

Usamos aquí un formato especial de subconsulta, llamado **subconsulta corralacionada** (del que hablaremos más adelante) y el operador exists.
La subconsulta correlacionada, se ejecuta una vez por cada fila de la tabla empleado. Verifica si existe al menos una entrada en la tabla OFICIOEMPLEADO para cada uno de los empleados de nuestra tabla EMPLEADO. En este caso, la subconsulta se ejecutará tantas veces como empleados haya en la tabla EMPLEADO.

Comparador All

All compara un valor con cada valor de una lista de valores separados por comas o una lista de valores obtenida a través de una subconsulta.

All debe estar precedido por un operador relacional. All se evalúa como verdadero si el resultado de la comparación es verdadero. También se evalúa como verdadero si la subconsulta no retorna ninguna fila.

Ejemplo de uso de all:

Obtener el nombre de los empleados cuyo número de alojamiento es mayor que el número de alojamiento de 'MULLERS' y de 'PAPA KING'. (Serían los empleados alojados en los alojamientos 5 y 6).

```
SELECT NOMBRE
FROM EMPLEADO
WHERE ALOJAMIENTO>ALL(SELECT NUMALOJ
                      FROM ALOJAMIENTO
                      WHERE ALOJAMIENTO IN ('MULLERS','PAPA KING'))
```

Consultas en las que se necesite trabajar con más de dos tablas.

En los ejemplos que hemos utilizado hasta ahora para obtener información de más de una tabla, hemos trabajado con dos tablas.

Evidentemente el número se puede ampliar a más de dos. Habrá consultas en las que necesitemos obtener información que esté dispersa en tres o más tablas.
La filosofía de trabajo sigue siendo la misma en estos casos pudiendo trabajar mediante combinación de tablas o subconsultas según nuestras necesidades.

Ejemplo 14:

Queremos obtener ahora el nombre del empleado y el nombre de sus oficios.
La información que necesitamos está en las tablas **EMPLEADO**, **OFICIO** y **OFICIOEMPLEADO**.

Habrá que combinar estas tres tablas para obtener la información solicitada. Tres tablas se combinan de la misma forma que se combinan dos. Las columnas comunes se igualan en la cláusula where.
Para combinar tres tablas, se necesita que dos de ellas se puedan combinar con una tercera.
La consulta anterior la escribiríamos de la siguiente manera:

```
SELECT NOMBRE,OFICIO.OFICIO
FROM EMPLEADO,OFICIO,OFICIOEMPLEADO
WHERE EMPLEADO.NUMEMP=OFICIOEMPLEADO.EMPLEADO
AND OFICIOEMPLEADO.OFICIO=OFICIO.NUMOFICIO
```

Observad que en la cláusula where hemos unido la tabla **EMPLEADO** con **OFICIOEMPLEADO** igualando el número del empleado y la tabla **OFICIO** con **OFICIOEMPLEADO** igualando el número de oficio.
Al igual que hicimos en el caso de la recuperación de dos tablas, podemos reescribir esta consulta mediante el uso del operador Join.

```
SELECT NOMBRE,OFICIO.OFICIO
FROM EMPLEADO JOIN OFICIOEMPLEADO
ON EMPLEADO.NUMEMP=OFICIOEMPLEADO.EMPLEADO
JOIN OFICIO ON OFICIOEMPLEADO.OFICIO=OFICIO.NUMOFICIO
```

Observamos de nuevo, que las condiciones de unión tienen que ver con las claves primarias y foráneas de las tablas combinadas.

Si omitimos en la cláusula where las condiciones por las que se deben combinar las tablas, obtendremos como se dijo anteriormente el producto cartesiano, en este caso de tres tablas. Se uniría cada fila de la tabla empleado con todas las filas de la tabla oficioempleado y este resultado se uniría con todas las filas de la tabla oficio.

Escribimos de forma errónea la consulta y mostramos un extracto de lo que obtendríamos, para que nos demos cuenta del problema que se genera si no combinamos adecuadamente las tablas.

```
SELECT EMPLEADO.NOMBRE,OFICIOEMPLEADO.EMPLEADO,OFICIO.OFICIO
FROM EMPLEADO,OFICIOEMPLEADO,OFICIO
ORDER BY EMPLEADO.NOMBRE,OFICIOEMPLEADO.EMPLEADO,OFICIO.OFICIO
```

NOMBRE	OFICIOEMPLEADO.OFICIO	OFICIO.OFICIO
ADAH TALBOT	11	OBRERO
ADAH TALBOT	11	PICADOR
ADAH TALBOT	11	PICADOR
ADAH TALBOT	11	PICADOR
ADAH TALBOT	17	CONDUCTOR DE SEGADORA
ADAH TALBOT	17	HERRERO
ADAH TALBOT	17	LABRADOR
ADAH TALBOT	17	LEÑADOR
ADAH TALBOT	17	OBRERO
ADAH TALBOT	17	PICADOR
ADAH TALBOT	18	CONDUCTOR DE SEGADORA
ADAH TALBOT	18	CONDUCTOR DE SEGADORA
ADAH TALBOT	18	HERRERO
ADAH TALBOT	18	HERRERO
ADAH TALBOT	18	LABRADOR
ADAH TALBOT	18	LABRADOR
ADAH TALBOT	18	LEÑADOR
ADAH TALBOT	18	LEÑADOR
ADAH TALBOT	18	OBRERO
ADAH TALBOT	18	OBRERO
ADAH TALBOT	18	PICADOR
ADAH TALBOT	18	PICADOR
ANDREW DYE	1	CONDUCTOR DE SEGADORA
ANDREW DYE	1	HERRERO
ANDREW DYE	1	LABRADOR
ANDREW DYE	1	LEÑADOR

Consejo:
Siempre que combinemos varias tablas, asegurémonos primero de que están correctamente combinadas uniendo sus claves foráneas con las primarias y después añadamos las condiciones adicionales.

Ejemplo 15:

Obtener el nombre de los empleados cuyo oficio es 'LEÑADOR'.

```
SELECT NOMBRE
FROM EMPLEADO,OFICIO,OFICIOEMPLEADO
WHERE NUMEMP=EMPLEADO
AND OFICIOEMPLEADO.OFICIO=NUMOFICIO
AND OFICIO.OFICIO='LEÑADOR';
```

Una vez que hemos combinado las tablas adecuadamente, añadimos la condición de que el nombre del oficio sea 'LEÑADOR'

Como solo necesitamos obtener el nombre del empleado que es leñador y ninguna información adicional que está localizada en las tablas relacionadas, podemos reescribir esta consulta mediante subconsultas de la siguiente forma:

Volvamos a realizar esta consulta mediante subconsultas

```
SELECT NOMBRE
FROM EMPLEADO
WHERE NUMEMP IN(SELECT EMPLEADO
                FROM OFICIOEMPLEADO
                WHERE OFICIO=(SELECT NUMOFICIO
                              FROM OFICIO
                              WHERE OFICIO='LEÑADOR'))
```

Veamos un par de ejemplos más.

Ejemplo 16:

Queremos obtener el nombre de los conductores de segadora que estén alojados en la ciudad de EDMESTON.

Realizaremos primero la consulta mediante combinación de tablas

La información que necesitamos en este caso está dispersa en las tablas:

- **EMPLEADO** (aquí tenemos el nombre y el código de su alojamiento)
- **OFICIOEMPLEADO**(aquí tenemos los códigos de los oficios de los empleados),
- **OFICIO** (tengo el nombre del oficio que nos hará falta porque lo que buscamos son 'CONDUCTOR DE SEGADORA')

- **ALOJAMIENTO** ya que contiene las direcciones de los alojamientos y nosotros buscamos en concreto uno localizado en EDMESTON).

Para unir estas cuatro tablas hemos de tener cuidado con la condición a imponer en la cláusula where.

Tendremos que ver la información que comparten las tablas para construir correctamente la sentencia. El conocimiento del esquema relacional es fundamental para hacer esto de forma correcta. La relación entre claves primarias y foráneas es nuestra guía para combinar las tablas de forma adecuada.

Habrá que imponer que el número de empleado de la tabla **EMPLEADO** sea igual al empleado de la tabla **OFICIOEMPLEADO**.

Con esta primera condición: **empleado.numemp=oficioempleado.empleado**
tendremos cada empleado ligado con sus oficios reales.

De igual forma asociaremos a los empleados con su alojamiento real imponiendo que **empleado.alojamiento=alojamiento.numaloj.**

Además el oficio de oficioempleado debe estar unido solamente con su oficio correspondiente de la tabla oficio, es decir, **oficioempleado.oficio=oficio.numoficio**.

La siguiente consulta muestra el nombre, nombre del alojamiento, la dirección del alojamiento y el nombre del oficio de cada empleado de la empresa.

```
SELECT
NOMBRE,ALOJAMIENTO.ALOJAMIENTO,ALOJAMIENTO.DIRECCION,OFICIO.OFICIO
FROM EMPLEADO,OFICIO,OFICIOEMPLEADO,ALOJAMIENTO
WHERE NUMEMP=EMPLEADO
AND EMPLEADO.ALOJAMIENTO=NUMALOJ
AND NUMOFICIO=OFICIOEMPLEADO.OFICIO
```

Como únicamente estamos interesados en el nombre de los conductores de segadora que viven en Edmeston, añadiremos estas restricciones a la consulta anterior para obtener la información deseada:

```
SELECT
NOMBRE,ALOJAMIENTO.ALOJAMIENTO,ALOJAMIENTO.DIRECCION,OFICIO.OFICIO
FROM EMPLEADO,OFICIO,OFICIOEMPLEADO,ALOJAMIENTO
WHERE NUMEMP=EMPLEADO
AND EMPLEADO.ALOJAMIENTO=NUMALOJ
AND NUMOFICIO=OFICIOEMPLEADO.OFICIO
AND OFICIO.OFICIO='CONDUCTOR DE SEGADORA'
AND ALOJAMIENTO.DIRECCION LIKE '%EDMESTON%';
```

Alias

Si observamos el grafo sintáctico de la sentencia select vemos que aparecen dos elementos c_alias y t_alias. Vamos a estudiar qué son y en que situaciones nos pueden ser útiles.
Un alias es un nombre temporal asignado a una tabla o columna dentro de una sentencia SQL y utilizado para referenciarlo en otra parte de la misma sentencia .

Alias de columna

Supongamos que queremos sacar el nombre y la edad de los empleados:
Escribiríamos:

```
SELECT NOMBRE,EDAD
FROM EMPLEADO
```

En el listado que ofrece un cliente como SqlDeveloper aparecen los datos solicitados. El nombre que se le da en el listado a las columnas es el que hemos puesto en la sentencia select y que deben coincidir con el nombre de las columnas de la tabla. Los nombres que se utilizan en la definición de las tablas a veces utilizan abreviaturas u otras codificaciones que no son muy aclaratorias para un usuario final. Podemos en este caso utilizar el alias de columna para que el listado tenga un aspecto más claro.

Ejemplo 17:

Queremos obtener la misma información anterior pero queremos que en listado la en lugar de nombre como identificador de columna aparezca EMPLEADO, y en lugar de edad aparezca años. Escribiríamos lo siguiente:

```
SELECT NOMBRE EMPLEADO,EDAD AÑOS
FROM EMPLEADO
```

Empleado es un alias para la columna nombre y años es un alias para la columna edad.
Oracle incluso permite que el alias de columna pueda ser una cadena de caracteres

```
SELECT NOMBRE "NOMBRE DEL EMPLEADO",EDAD "EDAD ACTUAL"
FROM EMPLEADO
```

El alias entrecomillado es lo que utilizará SqlDeveloper para sacar el listado. Esta opción tiene solo utilidad para presentar datos, no sería apropiada si ejecutamos la consulta de forma embebida en cualquier lenguaje de programación.
También utilizaremos los alias de columna en la creación de vistas.

Alias de tabla

Un alias de tabla es un nombre temporal asignado a una tabla dentro de una sentencia SQL y utilizado para referenciarlo en otra parte de la misma sentencia.

Usos:

Un primer uso podría ser para hacer la escritura de determinadas consultas un poco más cómoda. Para ver un ejemplo vamos a reescribir el ejemplo 10:

Obtener el nombre del empleado y el nombre del alojamiento en el que están hospedados.

Para obtener esta información se necesitan la tabla empleado y la tabla alojamiento. La solución la escribíamos así:

```
SELECT NOMBRE,ALOJAMIENTO.ALOJAMIENTO
FROM EMPLEADO,ALOJAMIENTO
WHERE EMPLEADO.ALOJAMIENTO=NUMALOJ;
```

Como alojamiento era común a las dos tablas había que especificar tanto en la cláusula select como la where a que tabla nos referíamos, si a empleado o a alojamiento. Había por tanto que arrastrar los nombres de la tabla. Para que estos nombres sean más cortos a la hora de escribir la sentencia podemos utilizar el alias de tabla quedando la consulta así:

```
SELECT E.NOMBRE,A.ALOJAMIENTO
FROM EMPLEADO E ,ALOJAMIENTO A
WHERE E.ALOJAMIENTO=A.NUMALOJ
```

A es un alias para la tabla alojamiento y E es un alias para la tabla empleado.

Un segundo uso mucho más interesante resulta de la necesidad en determinadas consultas de combinar una tabla consigo misma. Para ilustrar este ejemplo supongamos que tenemos la tabla **TRABAJADOR** creada con el siguiente script:

```
CREATE TABLE TRABAJADOR(
      COD_EMP NUMBER(4),
      NOMBRE VARCHAR2(15),
      COD_JEFE NUMBER(4),
      CONSTRAINT PK_TRABAJADOR
            PRIMARY KEY(COD_EMP),
      CONSTRAINT FK_TRABAJADOR_COD_EMP
            FOREIGN KEY(COD_JEFE)
            REFERENCES TRABAJADOR(COD_EMP)
);
```

Para cada trabajador aparece almacenamos su código de empleado, su nombre y el código de su jefe.

Supongamos que la tabla inicialmente tiene estos valores

COD_EMP	NOMBRE	COD_JEFE
7839	KING	
7698	BLAKE	7839
7782	CLARK	7839
7566	JONES	7839
7654	MARTIN	7698
7499	ALLEN	7698
7844	TURNER	7698
7900	JAMES	7698
7521	WARD	7698
7902	FORD	7566
7369	SMITH	7902
7788	SCOTT	7566
7876	ADAMS	7788
7934	MILLER	7782

Ejemplo 18:

Queremos obtener ahora un listado en el que aparezca el nombre del empleado y el nombre de su jefe. Necesitamos una combinación de la tabla trabajador uniendo una fila de un trabajador con la fila de su jefe, así podremos sacar los nombres de los dos. Para darle también uso a los alias de columna haremos que la salida tenga el siguiente aspecto en los nombres de sus columnas: EMPLEADO Y JEFE. La columna empleado tendrá los nombres de los empleados y la columna jefe tendrá los nombres del jefe.

```
SELECT T.NOMBRE EMPLEADO,J.NOMBRE JEFE
FROM TRABAJADOR T,TRABAJADOR J
WHERE T.COD_JEFE=J.COD_EMP
```

El listado tendría este aspecto:

EMPLEADO	JEFE
BLAKE	KING
CLARK	KING
JONES	KING
MARTIN	BLAKE
ALLEN	BLAKE
TURNER	BLAKE
JAMES	BLAKE
WARD	BLAKE
FORD	JONES
SMITH	FORD
SCOTT	JONES
ADAMS	SCOTT
MILLER	CLARK

En este ejemplo hemos hecho uso tanto del alias de tabla como del alias de columna.

Subconsultas Correlacionadas

Una subconsulta correlacionada es una subconsulta que se evalúa una vez por cada fila procesada por la sentencia padre. La sentencia padre puede ser una SELECT, UPDATE o DELETE. Los siguientes ejemplos muestran la sintaxis general de una subconsulta correlacionada:

```
SELECT  lista_de_selección
FROM tabla1 t_aliastabla1
WHERE expresion operador
(SELECT  listacolumnas
 FROM tabla2 t_aliastabla2
WHERE t_aliastabla1.columna operador t_aliastabla2.columna)

UPDATE tabla1 t_aliastabla1
      SET columna=
            (SELECT expresion
                  FROM tabla2 t_aliastabla2
                  WHERE t_aliastabla1.columna=t_aliastabla2.columna)
DELETE FROM tabla1 t_aliastabla1
      WHERE columna operador
            (SELECT expresion
                  FROM tabla2 t_aliastabla2
                  WHERE t_aliastabla1.columna=t_aliastabla2.columna)
```

Explicaremos el funcionamiento para la sentencia SELECT, aunque la explicación es se aplica a subconsultas correlacionadas en las sentencias UPDATE y DELETE.

Podemos usar una subconsulta correlacionada para responder a una pregunta con varias partes, cuya respuesta depende del valor de cada fila procesada por la sentencia padre. Por ejemplo, dada la siguiente tabla de empleados:

```
CREATE TABLE EMPLEADO (
      NUMERO               NUMBER(4) NOT NULL,
      NOMBRE               VARCHAR2(10),
      OFICIO               VARCHAR2(9),
      JEFE                 NUMBER(4) CONSTRAINT EMP_SELF_KEY
                                    REFERENCES EMP (EMPNO),
      FECHA_ALTA           DATE,
      SUELDO               NUMBER(7,2),
      COMISION             NUMBER(7,2),
      DEPARTAMENTO         NUMBER(2) NOT NULL,
      CONSTRAINT EMP_FOREIGN_KEY
            FOREIGN KEY (DEPTNO) REFERENCES DEPT (DEPTNO),
      CONSTRAINT EMP_PRIMARY_KEY
            PRIMARY KEY (EMPNO));
```

Podemos construir una consulta que nos muestre aquellos empleados cuyo sueldo es mayor que la media de los sueldos de la gente que trabaja en su departamento. La consulta se escribiría de la siguiente manera:

```
SELECT departamento,nombre,sueldo
FROM  empleado e
WHERE sueldo>(SELECT AVG(sueldo)
                FROM empleado t
                WHERE e.departamento=t.departamento)
```

La sentencia funciona de la siguiente manera:

Para cada fila de la tabla empleado, la sentencia padre usa la subconsulta correlacionada para calcular la media del sueldo de los miembros del mismo departamento. Si el sueldo del empleado de la fila de la tabla padre es superior a la media del sueldo de los trabajadores de ese departamento esa fila es devuelta.

La subconsulta se evalúa una vez por cada fila de la tabla empleado.

Siempre que una subconsulta contiene una referencia a una tabla de la consulta padre, la subconsulta se convierte en subconsulta correalecionada.

Productos externos (Outer joins)

La tabla EMPLEADO contiene todos los empleados de nuestra hipotética empresa. La tabla OFICIOEMPLEADO contiene únicamente aquellos empleados que tienen alguna especialidad registrada. Imaginemos que queremos sacar un listado de todos los empleados y su especialidad independientemente de que tengan alguna especialidad o no.
Si intentamos obtener esta información combinando las tablas:

```
SELECT NOMBRE,OFICIO
FROM EMPLEADO E, OFICIOEMPLEADO O
WHERE E.NUMEMP=O.EMPLEADO
```

Vemos que solo aparecen en el listado los empleados que tienen alguna especialidad.
Para forzar a que en el listado aparezcan todos los empleados, se utiliza el operador **LEFT OUTER JOIN**. Este operador fuerza a crear una fila para todos aquellos empleados que no pueden satisfacer la condición expresada en la cláusula **ON** . Es como si estas filas se combinaran con una fila nula de la tabla OFICIOEMPLEADO.
La nueva consulta se escribiría así:

```
SELECT NOMBRE,OFICIO
FROM EMPLEADO E LEFT JOIN OFICIOEMPLEADO O
ON E.NUMEMP=O.EMPLEADO
```

Existe un formato más antiguo para escribir esto mismo. Este formato se considera desfasado y Oracle recomienda el anterior.

```
SELECT NOMBRE,OFICIO
FROM EMPLEADO E, OFICIOEMPLEADO O
WHERE E.NUMEMP=O.EMPLEADO(+)
```

El operador (+) debe ir colocado en la columna de la tabla donde queremos forzar la hipotética fila nula.
Si quisiéramos que apareciera el nombre del oficio escribiríamos la consulta de la siguiente manera:

```
SELECT E.NOMBRE,O.OFICIO
FROM EMPLEADO E LEFT JOIN OFICIOEMPLEADO OE
ON E.NUMEMP=OE.EMPLEADO LEFT JOIN OFICIO O
ON OE.OFICIO=O.NUMOFICIO
```

Podemos usar la función NVL (Null Value), para sustuir el valor Null que se genera en el oficio para aquellos empleados sin oficio registrado, por un texto más descriptivo.

```
SELECT E.NOMBRE,NVL(O.OFICIO,'SIN OFICIO REGISTRADO')OFICIO
FROM EMPLEADO E LEFT JOIN OFICIOEMPLEADO OE
ON E.NUMEMP=OE.EMPLEADO LEFT JOIN OFICIO O
ON OE.OFICIO=O.NUMOFICIO
ORDER BY E.NOMBRE
```

El listado mostrará ahora 'SIN OFICIO REGISTRADO' para aquellos empleados de los que no se tiene registrado ningún oficio.

El mismo caso se nos presenta en el ejemplo 18, King por no tener jefe no aparecía en el listado. La solución en este caso se escribiría así:

```
SELECT T.NOMBRE EMPLEADO,NVL(J.NOMBRE,'SIN JEFE') JEFE
FROM TRABAJADOR T LEFT JOIN TRABAJADOR J
ON T.COD_JEFE=J.COD_EMP
```

Agrupación de elementos.

Supongamos que tenemos la siguiente vista en donde se recogen los diferentes pedidos de una empresa. A efectos didácticos hemos convertido esta vista en una tabla que tendría el siguiente formato:

```
CREATE TABLE DETALLEPEDIDO(
  NUMPEDIDO NUMBER(4),
  LINEADETALLE NUMBER(2),
  PRODUCTO VARCHAR2(30),
  PRECIOACTUAL NUMBER(8),
  CANTIDAD NUMBER(4),
  CONSTRAINT PK_DETALLEPEDIDO
  PRIMARY KEY(NUMPEDIDO,LINEADETALLE)
)
;
```

NUMPEDIDO	LINEADETALLE	PRODUCTO	PRECIOACTUAL	CANTIDAD
601	1	SB ENERGY BAR-6 PACK	2	1
602	1	ACE TENNIS BALLS-3 PACK	3	20
603	2	ACE TENNIS RACKET I	56	4
604	1	ACE TENNIS NET	58	3
604	2	ACE TENNIS RACKET II	42	2
604	3	ACE TENNIS RACKET I	44	10
605	1	ACE TENNIS RACKET II	45	100
605	2	ACE TENNIS BALLS-3 PACK	3	500
605	3	ACE TENNIS NET	58	5
605	4	SP TENNIS RACKET	24	50
605	5	SP JUNIOR RACKET	9	100
605	6	RH: "GUIDE TO TENNIS"	3	10
606	1	RH: "GUIDE TO TENNIS"	3	1
607	1	ACE TENNIS BALLS-6 PACK	6	1
608	1	SP TENNIS RACKET	24	1
608	2	ACE TENNIS BALLS-6 PACK	6	2
609	1	ACE TENNIS RACKET II	35	1
609	2	ACE TENNIS BALLS-3 PACK	3	5
609	3	ACE TENNIS NET	50	1
610	1	ACE TENNIS RACKET I	35	1
610	2	ACE TENNIS BALLS-3 PACK	3	3
610	3	ACE TENNIS NET	58	1
611	1	ACE TENNIS RACKET II	45	1
612	1	ACE TENNIS RACKET I	30	100
612	2	ACE TENNIS RACKET II	41	20
612	3	SP JUNIOR RACKET	10	150
612	4	ACE TENNIS BALLS-6 PACK	6	100
613	1	ACE TENNIS BALLS-6 PACK	6	100
613	2	SP TENNIS RACKET	24	200
613	3	SB VITA SNACK-6 PACK	4	150
613	4	SB ENERGY BAR-6 PACK	2	200
614	1	ACE TENNIS RACKET I	35	444
614	2	ACE TENNIS BALLS-3 PACK	3	1000
614	3	ACE TENNIS BALLS-6 PACK	6	1000
615	1	ACE TENNIS RACKET II	45	4
615	2	ACE TENNIS BALLS-3 PACK	3	100
615	3	ACE TENNIS BALLS-6 PACK	5	50
616	1	ACE TENNIS RACKET II	45	10
616	2	ACE TENNIS BALLS-3 PACK	3	50
616	3	ACE TENNIS NET	58	2
616	4	RH: "GUIDE TO TENNIS"	3	10
616	5	SB ENERGY BAR-6 PACK	2	10
617	1	ACE TENNIS RACKET I	35	50
617	2	ACE TENNIS RACKET II	45	100
617	3	ACE TENNIS BALLS-3 PACK	3	500
617	4	ACE TENNIS BALLS-6 PACK	6	500
617	5	ACE TENNIS NET	58	500
617	6	SP TENNIS RACKET	24	100
617	7	SP JUNIOR RACKET	13	200
617	8	RH: "GUIDE TO TENNIS"	3	100
617	9	SB ENERGY BAR-6 PACK	2	200
617	10	SB VITA SNACK-6 PACK	4	300
618	1	ACE TENNIS RACKET I	35	23
618	2	ACE TENNIS RACKET II	45	50
618	3	ACE TENNIS BALLS-3 PACK	45	10
619	1	SB VITA SNACK-6 PACK	4	100
619	2	SB ENERGY BAR-6 PACK	2	100
619	3	RH: "GUIDE TO TENNIS"	3	100
619	4	ACE TENNIS BALLS-6 PACK	6	50
620	1	ACE TENNIS RACKET I	35	10
620	2	SB ENERGY BAR-6 PACK	2	1000
620	3	RH: "GUIDE TO TENNIS"	3	500
621	1	ACE TENNIS RACKET II	45	10
621	2	ACE TENNIS BALLS-3 PACK	3	100

El objetivo de esta sección es estudiar las cláusulas group by y having de la sentencia select. Previo a este paso vamos a estudiar algunas de las funciones más usuales para grupos de valores.

Funciones para grupos de valores.

Son aquellas funciones estadísticas tales como SUM(SUMA), AVG(MEDIA ARITMÉTICA), COUNT(CUENTA),MIN(MINIMO), MAX(MAXIMO) que dicen algo de un grupo de valores tomado como un todo.
Por ejemplo, el precio medio de los artículos, la suma de todos los artículos vendidos etc.
Las funciones de grupos de valores ignoran los valores NULL y realizan los cálculos sin contar con ellos.

Ejemplo 19:

Supongamos que queremos saber el total de articulos vendidos. Para ello sumaríamos la columna CANTIDAD de la tabla detalle pedido. La sentencia se escribiría así:

```
SELECT SUM(CANTIDAD) TOTALARTICULOSVENDIDOS
FROM DETALLEPEDIDO
```

Hemos utilizado el alias de columna para renombrar el resultado.

Ejemplo 20:

Supongamos que queremos saber el número de pedidos que ha tenido nuestra empresa.
Utilizaremos para ello la función COUNT que cuenta el número de valores de una determinada columna en la tabla. Por lo comentado antes si la columna en cuestión presentara valores nulos en alguna de las filas, estos no formarían parte del recuento.
Como la columna numpedido se repite para cada una de las líneas de detalle de dicho pedido tendremos que utilizar count con la opción distinct. La consulta quedaría así:

```
SELECT COUNT(DISTINCT NUMPEDIDO) TOTALPEDIDOS
FROM DETALLEPEDIDO;
```

Ejemplo 21:

Si queremos calcular la cantidad media de cada línea de detalle escribiríamos:

```
SELECT AVG(CANTIDAD)
FROM DETALLEPEDIDO;
```

Ejemplo 22:

Cantidad mínima solicitada en una línea de detalle de un pedido.

```
SELECT MIN(CANTIDAD)
FROM DETALLEPEDIDO
```

Ejemplo 23:

Cantidad máxima solicitada en una línea de detalle de un pedido.

```
SELECT MAX(CANTIDAD)
FROM DETALLEPEDIDO;
```

Ejemplo 24:

Número de artículos solicitados en el pedido 617.

```
SELECT COUNT(*)  ARTICULOSPEDIDO617
FROM DETALLEPEDIDO
WHERE NUMPEDIDO=617
```

La opción count(*) la utilizamos cuando no queremos contar ocurrencias de ninguna columna en particular sino del total de filas afectadas por la condición.

Ejemplo 25:

Obtener el listado de los productos cuya cantidad solicitada en un pedido sea igual a la cantidad máxima solicitada en todos los pedidos.

```
SELECT DISTINCT PRODUCTO,CANTIDAD
FROM DETALLEPEDIDO D
WHERE CANTIDAD=(SELECT MAX(CANTIDAD)
                FROM DETALLEPEDIDO);
```

El distinct es obligatorio ya que se podría dar el caso que un mimo artículo hubiese sido solicitado en una cantidad máxima en más de un pedido.

Ejemplo 26:

Obtener el total de 'ACE TENNIS BALLS-6 PACK' vendidas.

```
SELECT SUM(CANTIDAD)
FROM DETALLEPEDIDO
WHERE PRODUCTO='ACE TENNIS BALLS-6 PACK';
```

Cláusulas group by y having

Supongamos que queremos sacar un listado en el que aparezca cada artículo con el total de unidades vendidas de él. Con las funciones de grupo de valores podríamos hacer una consulta del tipo del ejemplo 26 para cada uno de los productos. Esta labor sería bastante engorrosa. Podemos hacer que las funciones anteriores trabajen sobre grupos que nosotros establezcamos mediante la cláusula group by.
La consulta con la que iniciamos este apartado se resolvería así:

```
SELECT PRODUCTO,SUM(CANTIDAD) UNIDADESVENDIDAS
FROM DETALLEPEDIDO
GROUP BY PRODUCTO
```

Las filas obtenidas en la cláusula select son agrupadas por la cláusula group by. Se crea un grupo con aquellas filas que tienen igual valor para la columna o columnas que aparecen en la cláusula group by. Una vez hechos los grupos las funciones de grupo actúan sobre cada uno de los grupos.
En el ejemplo anterior se crean tantos grupos como productos diferentes hayamos vendido. Después se suman las cantidades vendidas de cada producto.

La cláusula having actúa como la cláusula where solo que a nivel de grupos.
Supongamos que queremos que en listado solo salgan aquellos productos que tengan más de 1500 unidades vendidas. Escribiríamos lo siguiente:

```
SELECT PRODUCTO,SUM(CANTIDAD) UNIDADESVENDIDAS
FROM DETALLEPEDIDO
GROUP BY PRODUCTO
HAVING SUM(CANTIDAD) >1500;
```

El resultado se puede ordenar mediante la cláusula order by.

```
SELECT PRODUCTO,SUM(CANTIDAD) UNIDADESVENDIDAS
FROM DETALLEPEDIDO
GROUP BY PRODUCTO
HAVING SUM(CANTIDAD) >1500
ORDER BY UNIDADESVENDIDAS
```

En la cláusula order by si podemos referenciar al alias de columna; sin embargo esto no funcionaría en la cláusula having.

Una sentencia select con agrupamiento por tanto tendría de forma general este aspecto:

```
SELECT column1,....
FROM tabla1,....,
WHERE condicion
GROUP BY columna1,....
HAVING condición
ORDER BY columna1,....
```

Las reglas para llevar a cabo una sentencia de este tipo son las siguientes:

- Selección de las filas según la cláusula where.
- Agrupamiento de estas filas según la cláusula group by.
- Cálculo de los resultados de las funciones de grupo para cada grupo.
- Selección y eliminación de grupos según la cláusula having.
- Ordenación de los grupos según los resultados de las funciones de grupo mediante order by. La cláusula order by debe usar funciones de grupo o columnas de la cláusula group by.

Ejemplos de agrupamiento y funciones de grupo sobre nuestro caso de estudio.

Ejemplo 27:

Listar los oficios indicando cuántos trabajadores tenemos de cada uno.

```
SELECT O.OFICIO, COUNT(*) TRABAJADORES
FROM OFICIO O,OFICIOEMPLEADO OE
WHERE NUMOFICIO=OE.OFICIO
GROUP BY O.OFICIO
```

Ejemplo 28:

Listado en el que aparezca el trabajador y el número de oficios que posee, pero sólo para aquellos trabajadores que tengan más de un oficio.

```
SELECT O.OFICIO, COUNT(*) TRABAJADORES
FROM OFICIO O,OFICIOEMPLEADO OE
WHERE NUMOFICIO=OE.OFICIO
GROUP BY O.OFICIO
HAVING COUNT(*)>1
```

Ejemplo 29:

Listado en el que aparezcan el nombre, edad y alojamiento donde se hospeda para los trabajadores de mayor edad en cada alojamiento.
Para sacar la información de la máxima edad por alojamiento escribiríamos:

```
SELECT ALOJAMIENTO,MAX(EDAD)
FROM EMPLEADO GROUP BY ALOJAMIENTO
```

Para incorporar esto a una subconsulta aprovechamos el concepto de consulta correlacionada. La columna E.ALOJAMIENTO está correlacionada con la columna E.ALOJAMIENTO del select externo.

```
SELECT NOMBRE,EDAD,A.ALOJAMIENTO
FROM EMPLEADO E, ALOJAMIENTO A
WHERE E.ALOJAMIENTO=NUMALOJ
AND EDAD=(SELECT MAX(EDAD)
          FROM EMPLEADO
          WHERE E.ALOJAMIENTO=ALOJAMIENTO);
```

También se podría haber escrito utilizando subconsulta con dos atributos.

```
SELECT NOMBRE,EDAD,A.ALOJAMIENTO
FROM EMPLEADO E, ALOJAMIENTO A
WHERE E.ALOJAMIENTO=NUMALOJ AND
(E.ALOJAMIENTO,EDAD) IN (SELECT ALOJAMIENTO,MAX(EDAD)
                         FROM EMPLEADO
                         GROUP  BY ALOJAMIENTO);
```

Ejemplo 30:

Wilfred Lowel es el trabajador de mayor edad, no sale en los listados anteriores por no tener un alojamiento definido. Para forzar a que aparezca en el listado hacemos uso del LEFT OUTER JOIN y de la función nvl.

```
SELECT NOMBRE,EDAD,A.ALOJAMIENTO
FROM EMPLEADO E LEFT OUTER JOIN ALOJAMIENTO A
ON E.ALOJAMIENTO=A.NUMALOJ
WHERE EDAD=(SELECT MAX(EDAD)
            FROM EMPLEADO
            WHERE NVL(E.ALOJAMIENTO,0)=NVL(ALOJAMIENTO,0));
```

Observése como se ha creado un alojamiento ficticio con código 0 para aquellos empleados que no tienen alojamiento, para que cuando la consulta padre esté procesando a Wilfred Lowell, la subconsulta opere sobre los empleados que no tienen alojamiento. Eso se consigue mediante la condición:

```
WHERE NVL(E.ALOJAMIENTO,0)=NVL(ALOJAMIENTO,0));
```

Si quitamos el uso del NVL, la subconsulta evaluaría una condición como ésta:

```
WHERE E.ALOJAMIENTO=ALOJAMIENTO
```

Que en el caso de Wilfred Lowell sería where null=null. Ese null igual a null no produce ninguna fila. ¿Valor desconocido=Valor desconocido? genera siempre un falso y por tanto ninguna fila.
El resultado de la consulta sería el siguiente:

NOMBRE	EDAD	ALOJAMIENTO
ELBERT TALBOT	43	WEITBROCHT
GEORGE OSCAR	41	ROSE HILL
GERHARDT KENTGEN	55	PAPA KING

PETER LAWSON	25	CRAMMER
ROLAND BRANDT	35	MATTS
VICTORIA LYNN	32	MULLERS
WILFRED LOWELL	67	SIN ALOJAMIENTO

Ejemplo 31

Obtener el nombre de los empleados y sus oficios para aquellos trabajadores que tengan más de un oficio.

```
SELECT NOMBRE,O.OFICIO
FROM EMPLEADO E, OFICIO O, OFICIOEMPLEADO OE
WHERE NUMEMP=EMPLEADO
AND OE.OFICIO=NUMOFICIO
AND NUMEMP IN(SELECT NUMEMP
              FROM EMPLEADO,OFICIOEMPLEADO
              WHERE NUMEMP=EMPLEADO
              GROUP BY NUMEMP
             HAVING COUNT(OFICIO)>1);
```

Union, intersect y minus

Unión

La unión de dos relaciones compatibles en su esquema es otra relación definida sobre el mismo esquema cuya extensión estará constituida por las tuplas que pertenezcan a una de las dos relaciones o a ambas(se eliminan las tupas duplicadas).

Ejemplo 31:

Nombre de los trabajadores que cuyo oficio sea 'LEÑADOR' o 'CONDUCTOR DE SEGADORA'.

```
SELECT NOMBRE
FROM EMPLEADO E,OFICIO O, OFICIOEMPLEADO OE
WHERE NUMEMP=EMPLEADO
AND OE.OFICIO=NUMOFICIO
AND O.OFICIO='LEÑADOR'
UNION
SELECT NOMBRE  PEPE
FROM EMPLEADO E,OFICIO O, OFICIOEMPLEADO OE
WHERE NUMEMP=EMPLEADO
AND OE.OFICIO=NUMOFICIO
AND O.OFICIO='CONDUCTOR DE SEGADORA'
```

Esta consulta también se podría haber escrito de esta otra forma:

```
SELECT  DISTINCT NOMBRE
FROM EMPLEADO E,OFICIO O, OFICIOEMPLEADO OE
WHERE NUMEMP=EMPLEADO
AND OE.OFICIO=NUMOFICIO
AND (O.OFICIO='LEÑADOR' OR O.OFICIO='CONDUCTOR DE SEGADORA');
```

O usando el operador in la podemos escribir de esta otra forma:

```
SELECT  DISTINCT NOMBRE
FROM EMPLEADO E,OFICIO O, OFICIOEMPLEADO OE
WHERE NUMEMP=EMPLEADO
AND OE.OFICIO=NUMOFICIO
AND (O.OFICIO IN ('LEÑADOR', 'CONDUCTOR DE SEGADORA'))
```

Intersect (Intersección).

La intersección de dos relaciones compatibles en su esquema es otra relación definida sobre el mismo esquema cuya extensión estará constituida por las tuplas que pertenezcan a las dos relaciones.

Ejemplo 32:

Nombre de los trabajadores que tienen registrado poder trabajar como leñadores y como conductores de segadora.

```
SELECT NOMBRE
FROM EMPLEADO E,OFICIO O, OFICIOEMPLEADO OE
WHERE NUMEMP=EMPLEADO
AND OE.OFICIO=NUMOFICIO
AND O.OFICIO='LEÑADOR'
INTERSECT
SELECT NOMBRE
FROM EMPLEADO E,OFICIO O, OFICIOEMPLEADO OE
WHERE NUMEMP=EMPLEADO
AND OE.OFICIO=NUMOFICIO
AND O.OFICIO='CONDUCTOR DE SEGADORA';
```

Minus(diferencia).

La diferencia de dos relaciones compatibles en su esquema es otra relación definida sobre el mismo esquema cuya extensión estará constituida por las tupas que pertenezcan a una relación pero no pertenezcan a la otra.

Ejemplo 33:

Nombres de los conductores que sean conductores de segadora pero que no sean leñadores.

```
SELECT NOMBRE
FROM EMPLEADO E,OFICIO O, OFICIOEMPLEADO OE
WHERE NUMEMP=EMPLEADO
AND OE.OFICIO=NUMOFICIO
AND O.OFICIO='CONDUCTOR DE SEGADORA'
MINUS
SELECT NOMBRE
FROM EMPLEADO E,OFICIO O, OFICIOEMPLEADO OE
WHERE NUMEMP=EMPLEADO
AND OE.OFICIO=NUMOFICIO
AND O.OFICIO='LEÑADOR';
```

Consultas jerárquicas.

Si la tabla contiene datos jerárquicos, podemos seleccionar filas en un orden jerárquico utilizando las siguientes cláusulas que podemos ver en el grafo sintáctico de la orden select:

- **START WITH:** Podemos especificar la fila(s) padre de la jerarquía usando esta cláusula.
- **CONNECT BY condición:** Para especificar la relación entre las filas padre y las filas hijos de la jerarquía. En alguna parte de la condición se debe usar el operador **PRIOR** para referirse a la fila padre. La parte de la condición que contenga el operador PRIOR debe adoptar una de las siguientes formas:
    - PRIOR expr operador_comparación expr
    - Expr operador_comparación PRIOR expr

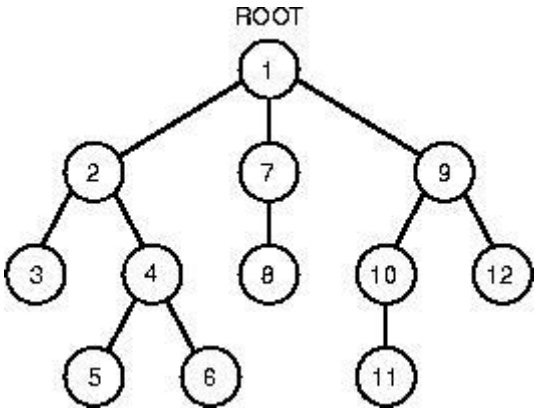
- **WHERE:** Podemos restringir las filas devueltas por la consulta.

Las sentencias select que contienen consultas jerárquicas pueden contener la seudocolumna **LEVEL**.
LEVEL devuelve el valor 1 para un nodo raíz, 2 para un nodo hijo del nodo raíz, 3 para un nieto, etc. El número de niveles de una consulta jerárquica puede estar limitada por la memoria disponible.

A partir de ahí Oracle ejecuta la consulta jerárquica siguiendo los siguientes pasos:

- Oracle selecciona la fila o filas padre de la jerarquía (aquellas que satisfacen la condición de la cláusula START WITH).
- Oracle selecciona las filas hijo de cada fila raíz. Cada fila hijo debe satisfacer la condición que aparece en la cláusula CONNECT BY con respecto a una de las filas raíz.

- Oracle selecciona nuevas generaciones de filas hijo. Oracle primero selecciona los hijos de las filas devueltas en el paso 2, y luego los hijos de esos hijos y así sucesivamente. Oracle siempre selecciona hijos evaluando la condición de la cláusula CONNECT BY con respecto a la actual fila padre.
- Si la consulta contiene una cláusula WHERE, Oracle elimina todas las filas de la jerarquía que no satisfacen la condición de la cláusula WHERE. Oracle evalúa esta condición de cada fila individualmente, en lugar de eliminar todos los hijos de una fila que no satisface la condición.
- Oracle devuelve las filas en el orden mostrado en la siguiente figura. En el diagrama los hijos aparecen bajo sus padres.



Si en la cláusula **CONNECT BY** se produce un **bucle en la jerarquía**, Oracle devuelve un error. Un bucle sucede si una fila es a la vez padre (o abuelo o un antecesor directo) e hijo (o descendiente directo) de otra fila.

Ejemplos

La siguiente cláusula CONNECT BY define una relación jerárquica en la cual el valor de COD_EMP de la fila padre es igual al valor de COD_JEFE de la fila hija.
La siguiente sentencia devuelve a todos los trabajadores de la tabla trabajador en orden jerárquico. La fila padre se define para aquellos trabajadores cuyo empleo sea 'PRESIDENTE'.

```
SELECT LPAD('    ',2*(LEVEL-1))||NOMBRE TRABAJADOR,COD_EMP,COD_JEFE
FROM TRABAJADOR
START WITH NOMBRE='KING'
CONNECT BY PRIOR COD_EMP=COD_JEFE
```

Que mostraría la siguiente salida;

	TRABAJADOR	COD_EMP	COD_JEFE
1	KING	7839	(null)
2	JONES	7566	7839
3	SCOTT	7788	7566
4	ADAMS	7876	7788
5	FORD	7902	7566
6	SMITH	7369	7902
7	BLAKE	7698	7839
8	ALLEN	7499	7698
9	WARD	7521	7698
10	MARTIN	7654	7698
11	TURNER	7844	7698
12	JAMES	7900	7698
13	CLARK	7782	7839
14	MILLER	7934	7782

Los espacios en blanco se consiguen con `LPAD('  ',2*(LEVEL-1));`

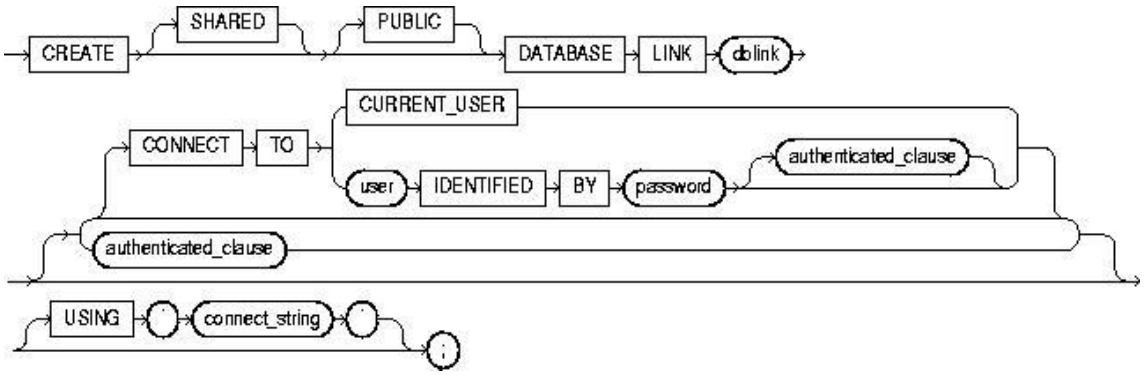
Enlace de base de datos

Nos van a permitir acceder a los datos y objetos de un esquema a través de un sistema de base de datos distribuidos. Ello nos permitirá acceder a datos de una tabla situada en una base de datos remota.
Para ello el administrador de la base de datos debe crear los enlaces de base de datos necesarios. Una vez creado el enlace de base de datos, el acceso a los datos remotos a través de una sentencia select se haría añadiendo el sufijo @enlace de base de datos detrás del nombre de la tabla, tal y como se ve en el grafo sintáctico de la orden select.
Hay tres tipos de enlace de base de datos:

- **Privado**: Solo el dueño del enlace o los subprogramas de su esquema lo pueden utilizar.
- **Público**: Utilizable por todos los usuarios y los subprogramas de la base de datos en la que se ha definido.
- **Global**: Utilizable por todos los usuarios y subprogramas del sistema de base de datos distribuido.

Para poder crear un en enlace privado de base de datos, debemos tener el privilegio de sistema **CREATE DATABASE LINK**.

Para crear un **enlace de base de datos publico** debemos tener el privilegio de sistema **CREATE PUBLIC DATATABASE LINK.** Además, debemos tener el privilegio **CREATE SESSION** en la base de datos remota. La sintaxis para crear un enlace de base de datos es la siguiente:



Describiremos a continuación los parámetros más usados:
PUBLIC: Para crear un enlace de base de datos publico. Si se omite el enlace que creamos es privado.
Dblink: Es el nombre que le daremos al enlace de base de datos.
CONNECT TO: Habilita la conexión a la base de datos remota.
User IDENTIFIED BY password: es el usuario y la contraseña usada para conectarse a la base de datos remota.
Connect_string: Especifica el nombre del servicio de la base de datos remota. Este parámetro será sustituido por la cadena de conexión que tenemos en el archivo tnsnames.ora para acceder a la base de datos remota. Observar que el parámetro connect_string va entre comillas simples.

Supongamos que tenemos instalado una base de datos Oracle en nuestro ordenador personal y queremos acceder a las tablas del usuario **CICLO** que está en el servidor del instituto.
Deberíamos hacer lo siguiente:
Primero:
Añadir la descripción de la conexión al archivo tnsnames.ora con los parámetros que identifican la conexión remota.
El archivo tnsnames.ora quedaría configurado de la siguiente manera:

ORT: representa el nombre lógico de la conexión a la base de datos local. Observad en la descripción del HOST la palabra: **localhost**. Este queda configurado en la instalación de la bse de datos local, no habría que crearlo.

```
ORT =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)(HOST = localhost)(PORT = 1521))
    (CONNECT_DATA =
      (SERVER = DEDICATED)
      (SERVICE_NAME = ort)
    )
  )
```

```
ORADELL =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)(HOST = 10.0.1.12)(PORT = 1521))
    (CONNECT_DATA =
      (SERVER = DEDICATED)
      (SERVICE_NAME = ORADAI)
    )
  )
```

ORADELL: representa el nombre lógico del servidor remoto. En este caso el servidor está alojado en la red local del instituto, pero podría estar en cualquier lugar del mundo.

```
EXTPROC_CONNECTION_DATA =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = IPC)(KEY = EXTPROC1))
    )
    (CONNECT_DATA =
      (SID = PLSExtProc)
      (PRESENTATION = RO)
    )
  )
```

Segundo

Una vez hecho esto desde un usuario de la base de datos local con privilegio CREATE DATABASE LINK escribiríamos lo siguiente:

```
CREATE DATABASE LINK CICLODELL
CONNECT TO CICLO IDENTIFIED BY CICLO USING 'ORADELL';
```

Es obvio que, para definir el enlace de base de datos, debemos tener el usuario y la contraseña con permisos sobre algún esquema de la base de datos remota del que vayamos a obtener datos.
A partir de ahí si yo quisiera desde mi sesión local acceder a la tabla CICLO de la base de datos remota escribiría:

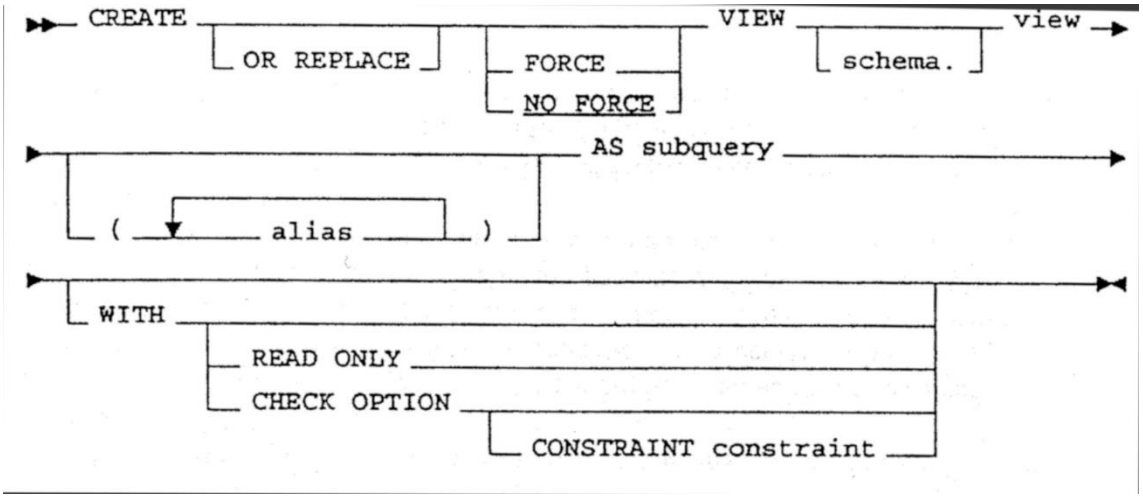
```
SELECT * from CICLO@CICLODELL;
```

Vistas

Aunque la creación de vistas es una orden perteneciente al lenguaje de definición de datos (DDL) hemos preferido posponer su aparición hasta este punto, debido a que la manera de crear una vista es mediante una sentencia **select**.
Una vista define una tabla virtual basada en una o más tablas o vistas. Esta tabla virtual se almacena permanentemente en la base de datos, generando al igual que las tablas, una entrada en el diccionario de datos.
Existe una vista del diccionario de datos para poder consultar las vistas de nuestro esquema. Esta vista es **USER_VIEWS**.
Las vistas se utilizan con los siguientes propósitos:

- Organizar los datos de diferentes formas, de modo que los usuarios los ven desde diferentes perspectivas (recordar el nivel externo de la arquitectura ANSI).
- Añadir un nivel adicional de seguridad ya que podemos restringir el acceso a los datos, permitiendo que determinados usuarios, a través de las vistas, sólo puedan acceder a determinadas filas y columnas de las tablas.
- Para ocultar la complejidad de los datos. Podemos usar una vista como si fuese una única tabla cuando realmente hay varias tablas implicadas en la construcción del resultado.

Desde el punto de vista de usuario, la vista es como una tabla real, con estructura de filas y columnas; pero, a diferencia de éstas, sus datos no se almacenan físicamente en la base de datos. Las filas y columnas visibles para el usuario son consecuencia de la consulta que se realiza en las tablas reales sobre las que se definió la vista.
Para crear una vista en nuestro esquema debemos tener el privilegio del sistema **CREATE VIEW**. Para crear una vista en otro esquema de usuario, debemos tener el privilegio de sistema **CREATE ANY VIEW**.
El propietario del esquema que contiene la vista debe tener los privilegios necesarios para seleccionar, insertar, actualizar o borrar filas de las tablas o vistas sobre las que se basa la definición de la vista.
El formato de la sentencia que permite crear vistas es el siguiente:



Significado de los parámetros y palabras clave:

- **Or Replace:** Permite recrear la vista si esta ya existe. Podemos usar esta opción para cambiar la definición de una vista existente sin tenerla que borrarla previamente.
- **Force:** Crea la vista sin comprobar si las tablas base sobre las que se construye la vista existen o si el propietario del esquema que contiene la vista tiene privilegios en ellas. Hay que tener en cuenta que estas condiciones serán comprobadas y deben ser ciertas antes de hacer cualquier select, insert, update o delete a través de la vista.
- **Noforce:** Crea la vista únicamente si las tablas base existen y el propietario del esquema que contiene la vista tiene privilegios sobre ellas. Esta es la opción por defecto.
- **Schema:** Es el esquema que contendrá la vista. Si se omite la vista se crea en el propio esquema.
- **View:** Es el nombre de la vista. El espacio de nombres para las vistas es el mismo que para las tablas; por tanto, en un mismo esquema el nombre de una vista tiene que ser distinto al de una tabla.
- **Alias:** Especifica los nombres para las expresiones seleccionadas mediante la consulta constructora de la vista. El número de alias debe ser el mismo que el número de expresiones seleccionadas mediante la consulta.
- **As subquery:** Identifica las columnas y filas de la tabla o tablas sobre las que se basa la vista. La consulta constructora de la vista puede ser cualquier sentencia **select** sin las cláusulas **order by** y **for update**.
- **With read only:** Especifica no se permiten borrados, inserciones ni modificaciones a través de la vista.
- **With check option:** Especifica que las inserciones y actualizaciones realizadas a través de la vista deben verificar la consulta de selección sobre la que está construida la vista. Si esta condición no fuese satisfecha la inserción o modificación no tendría efecto. Esta cláusula no se puede garantizar si hay una subconsulta en la definición de la consulta de esta vista o en cualquier otra vista en la cual se base la construcción de nuestra vista.
- **Constraint:** Es el nombre asignado a la restricción check option. Si omitimos el identificador de restricción, Oracle automáticamente asigna a la restricción un nombre de la forma Sys_Cn, donde n es un entero que garantiza que el nombre de la restricción sea único.

Ejemplo 34

Creación de una vista que solo contenga el número de empleado, el nombre y el alojamiento de los empleados.

```
CREATE VIEW DATOSEMPLEADO AS
SELECT NUMEMP,NOMBRE,ALOJAMIENTO
FROM EMPLEADO
```

Podemos ahora consultar estos datos a través de la vista:

```
SELECT * FROM DATOSEMPLEADO
```

Podemos también insertar datos en la tabla empleado a través de la vista:

```
INSERT INTO DATOSEMPLEADO
VALUES(35,'PEPITO',3)
```

Podemos también eliminar datos de la tabla empleado a través de la vista:

```
DELETE FROM DATOSEMPLEADO
WHERE NUMEMP=35;
```

Con respecto a las inserciones, modificaciones y borrados a través de las vistas hay que tener en cuenta lo siguiente:

- Si la vista no contiene en su definición alguna columna de la tabla base que tenga una restricción not null, o bien, no entra en la definición la clave primaria, no serán posibles las inserciones a través de la vista.
- No serán posibles las inserciones, modificaciones ni borrados a través de la vista si la consulta a través de la cual se ha construido, contiene alguna de las siguientes construcciones:
    - Joins (producto cartesiano)
    - Operadores de conjunto
    - Funciones de grupo
    - Claúsulas group by, connect by o start with
    - El operador distinct.
- Si la vista contiene expresiones, sólo se pueden modificar aquellas columnas que no hagan referencia a las expresiones.
- Tampoco será posible insertar, modificar o borrar a través de una vista si ésta contiene en su definición la cláusula with read only.

Ejemplo 35:

Creación de una vista que solo contenga del empleado y el nombre del alojamiento donde se hospeda. Forzaremos a que salgan en el listado aquellos trabajadores que no tienen un alojamiento asignado mediante el operador **left outer join**.

```
CREATE OR REPLACE VIEW ALOJAMIENTOEMPLEADO AS
SELECT NOMBRE EMPLEADO, A.ALOJAMIENTO
FROM EMPLEADO E LEFT OUTER JOIN ALOJAMIENTO A
ON E.ALOJAMIENTO=A.NUMALOJ;
```

Podemos ahora consultar los alojamientos de los empleados simplemente a través de esta consulta:

```
SELECT EMPLEADO,NVL(ALOJAMIENTO,'** NO REGISTRADO **')ALOJAMIENTO
FROM ALOJAMIENTOEMPLEADO;
```

Ejemplo 36:

Nos podemos apoyar en vistas para hacer que determinadas consultas se puedan construir de una forma más sencilla. Supongamos que queremos obtener el nombre y el sueldo de aquellos empleados que tienen la especialidad de 'HERRERO' siempre que el sueldo sea superior a 1. La consulta podría formularse de esta forma:

```
SELECT E.NOMBRE,SUELDO
FROM EMPLEADO E, OFICIO O,OFICIOEMPLEADO OE
WHERE E.NUMEMP=OE.EMPLEADO
AND O.NUMOFICIO=OE.OFICIO
AND O.OFICIO='HERRERO'
AND SUELDO >1
```

Supongamos que tenemos una vista con el numero de empleado y el nombre de los trabajadores que tienen como especialidad 'HERRERO'. Su construcción vendría definida de la siguiente manera:

```
CREATE VIEW HERRERO AS
SELECT E.NUMEMP,NOMBRE
FROM EMPLEADO E, OFICIO O,OFICIOEMPLEADO OE
WHERE E.NUMEMP=OE.EMPLEADO
AND O.NUMOFICIO=OE.OFICIO
AND O.OFICIO='HERRERO'
```

Apoyándonos en esta vista la consulta anterior se podría escribir de esta otra forma:

```
SELECT E. NOMBRE,SUELDO
FROM EMPLEADO E,HERRERO H
WHERE E.NUMEMP=H.NUMEMP
AND SUELDO>1
```

Ejemplo 37:

Veamos un ejemplo más potente del uso de las vistas para construir consultas complejas. Vovamos a nuestra tabla **DETALLEPEDIDO**.
Queremos obtener un listado en el que aparezca el nombre del producto, el total de ventas de ese producto, y el porcentaje de las ventas del producto sobre las ventas totales ordenado de mayor a menor porcentaje.
Para hacer esta consulta de una manera sencilla nos apoyaremos en dos vistas:

Vista 1. TOTALVENTAS.
Vista que obtiene el importe total de las ventas de la empresa. La vista se crearía con la siguiente orden:

```
CREATE VIEW TOTALVENTAS AS
SELECT SUM(PRECIOACTUAL*CANTIDAD) VENTASTOTALES
FROM DETALLEPEDIDO
```

Vista 2. VENTASARTICULO.
Vista en la que tendremos el nombre del producto y el importe total de todas las ventas hechas de ese producto. La vista se crearía de la siguiente manera:

```
CREATE VIEW VENTASARTICULO AS
SELECT PRODUCTO, SUM(PRECIOACTUAL*CANTIDAD) TOTALARTICULO
FROM DETALLEPEDIDO
GROUP BY PRODUCTO
```

La consulta final es ahora una tarea muy sencilla apoyándonos en estas dos vistas:

```
SELECT PRODUCTO,TOTALARTICULO,(TOTALARTICULO/VENTASTOTALES) PORCENTAJE
FROM VENTASARTICULO,TOTALVENTAS
ORDER BY PORCENTAJE DESC;
```

La consulta se podría haber realizado sin apoyarnos en la segunda vista así:

```
SELECT PRODUCTO, SUM(PRECIOACTUAL*CANTIDAD) TOTALARTICULO,
                 SUM(PRECIOACTUAL*CANTIDAD)/VENTASTOTALES PORCENTAJE
FROM DETALLEPEDIDO,TOTALVENTAS
GROUP BY PRODUCTO,VENTASTOTALES
ORDER BY PORCENTAJE DESC;
```

Es obligatorio incluir en el group by a totalventas para poder incluir en la cláusula select la operación que calcula el porcentaje.

Select from (subconsulta).

Si observamos el grafo sintáctico de la orden select, vemos que una de las posibilidades sigue este formato:

```
SELECT  FROM (subquery) t_alias
```

Donde subquery es cualquier consulta válida. La parte (subquery) t_alias puede repetirse siempre que se separen por comas las distintas ocurrencias, tomando este aspecto:

```
SELECT FROM
(subquery1)t_alias1,(subquery2)t_alias2,...
```

T_alias1 y t_alias2 **funcionan como si fueran tablas** que contienen los valores aportados por las subconsultas respectivas.

Este formato de orden select nos da la misma versatilidad a la hora de estructurar consultas complejas que la que nos ofrecía el trabajo con vistas visto en el apartado anterior.
Esta opción además tiene la ventaja de que no hay que crear ningún objeto.
Otra de las ventajas de esta forma de estructurar las consultas es la posibilidad de filtrar el volumen de datos que entran en la cláusula from, obteniendo consultas más eficientes.

Ejemplo: Vamos a reescribir la última consulta vista en el apartado de vistas, para ver que tendremos la misma versatilidad utilizando este formato de la orden.

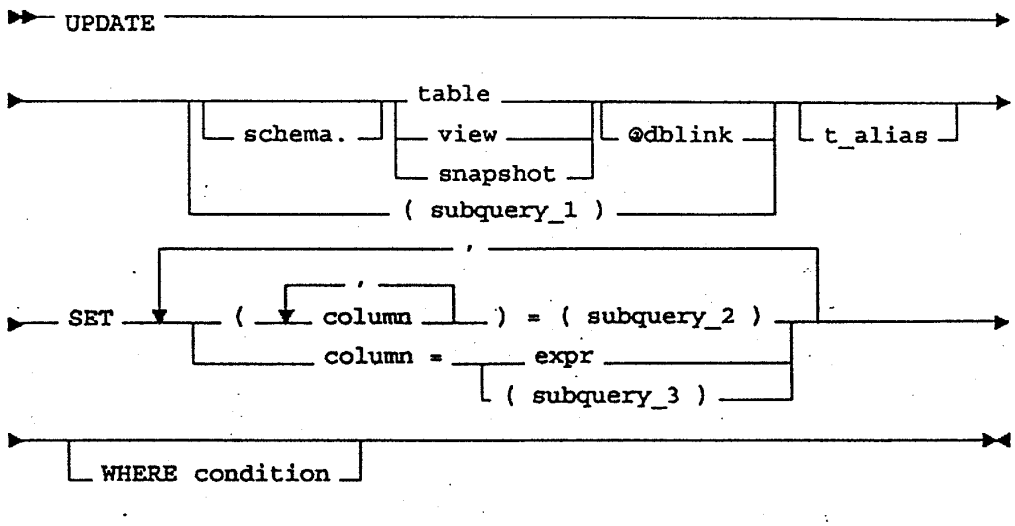
```
SELECT PRODUCTO,TOTALARTICULO,(TOTALARTICULO/VENTASTOTALES)*100
PORCENTAJE_VENTAS
FROM
  (SELECT PRODUCTO,SUM(PRECIOACTUAL*CANTIDAD)TOTALARTICULO
   FROM DETALLEPEDIDO
   GROUP BY PRODUCTO)A,
   (SELECT SUM(PRECIOACTUAL*CANTIDAD)VENTASTOTALES
    FROM DETALLEPEDIDO)B
ORDER BY PORCENTAJE_VENTAS DESC
```

Aunque aquí no aparezca cláusula where en la orden select externa, en la mayoría de los casos sI que existirá. Habrá que ver que **atributos tienen en común** las subconsultas usadas para realizar la combinación (join) de las diferentes subconsultas de una forma correcta. Por defecto, como siempre, se realiza el producto cartesiano de los resultados obtenidos por cada una de las subconsultas.

Modificación de datos. (Update).

Permite cambiar los valores de una tabla o en la tabla base de una vista. Para poder modificar los valores de la tabla ésta debe pertenecer a nuestro esquema o bien tener el privilegio update sobre esa tabla. Para poder modificar los valores en la tabla base de una vista (modificaciones a través de vistas), el propietario del esquema que contiene la vista debe tener el privilegio update en la tabla base.
También si la vista está en un esquema distinto el de nuestra propiedad, debemos tener el privilegio update sobre la vista. El privilegio **update any table** permite modificar los valores en cualquier tabla o vista de la base de datos.
Aparte de estas reglas las modificaciones de tablas base a través de las vistas deberán seguir las reglas vistas en el apartado anterior.

La sintaxis de la sentencia **update** es la siguiente:



Significado de los parámetros y palabras clave de uso más común:

- **Schema:** Es el nombre del esquema que contiene la tabla o la vista. Si se omite Oracle asume que la tabla o vista está en nuestro propio esquema.
- **Table, view:** Nombre de la tabla a ser modificada. Si se especifica el nombre de una vista Oracle modificará la tabla base.
- **Alias:** Para dotar de un nombre distinto a la tabla, vista o subconsulta para ser referenciado en cualquier otro lugar en la sentencia.
- **Column:** Es el nombre de la columna de la tabla o vista que va a ser modificado.
- **Expr:** Es el nuevo valor asignado a la columna correspondiente.
- **Subquery2:** Es la subconsulta que devuelve nuevos valores que serán asignados a las correspondientes columnas.
- **Subquery3:** La subconsulta que devuelve un nuevo valor que será asignado a la columna correspondiente.

- **Where:** Restringe el número de filas que se actualizarán a aquellas para las que la condición especificada en la cláusula where es verdadera. Si la cláusula where se **omite** todas las filas se actualizarán.

Ejemplo 38:

Incrementar la edad de los trabajadores en un año.

```
UPDATE EMPLEADO
SET EDAD=EDAD+1
```

Ejemplo 39:

El alojamiento 'CRAMMER' cuyo número de alojamiento es 1 ha cerrado y todos los empleados alojados aquí se alojarán ahora en el alojamiento 2.

```
UPDATE EMPLEADO
SET ALOJAMIENTO=2
WHERE ALOJAMIENTO=1
```

Ejemplo 40:

Incrementar el sueldo de los empleados de más de 60 años en un 10%.

```
UPDATE EMPLEADO
SET SUELDO=SUELDO+SUELDO*0.1
WHERE EDAD >60;
```

Ejemplo 41

Incrementar el sueldo de los empleados con la especialidad de 'HERRERO' en dos unidades.

```
UPDATE EMPLEADO
SET SUELDO=SUELDO +2
WHERE NUMEMP IN(SELECT NUMEMP
                FROM EMPLEADO E, OFICIOEMPLEADO OE,OFICIO O
                WHERE E.NUMEMP=OE.EMPLEADO
                AND OE.OFICIO=O.NUMOFICIO
                AND O.OFICIO='HERRERO');
```

Ejemplo 42

Actualizar el sueldo de 'ELBERT TALBOT' con el sueldo más alto de la empresa.

```
UPDATE EMPLEADO
SET SUELDO=(SELECT MAX(SUELDO)
            FROM EMPLEADO)
WHERE NOMBRE='ELBERT TALBOT';
```

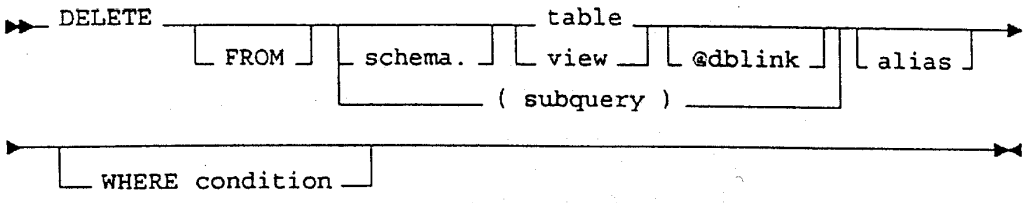
Ejemplo 43

Incrementar el sueldo de los empleados con la especialidad de 'HERRERO' con el sueldo más alto de la empresa.

```
UPDATE EMPLEADO
SET SUELDO=(SELECT MAX(SUELDO)
                        FROM EMPLEADO)
WHERE NUMEMP IN(SELECT NUMEMP
                  FROM EMPLEADO E, OFICIOEMPLEADO OE,OFICIO O
                  WHERE E.NUMEMP=OE.EMPLEADO
                  AND OE.OFICIO=O.NUMOFICIO
                  AND O.OFICIO='HERRERO');
```

Borrado de datos(Delete)

La orden **delete** permite eliminar filas de una tabla o de la tabla base asociada a una vista. Para poder eliminar filas de una tabla ésta debe pertenecer a nuestro esquema o bien tener el privilegio delete sobre esa tabla. Para poder eliminar filas de la tabla base de una vista (borrados a través de vistas), el propietario del esquema que contiene la vista debe tener el privilegio delete en la tabla base. También si la vista está en un esquema distinto el de nuestra propiedad, debemos tener el privilegio delete sobre la vista.
El privilegio **delete any table** permite eliminar filas en cualquier tabla o vista de la base de datos.
Aparte de estas reglas los borrados de filas de tablas base a través de las vistas deberán seguir las reglas vistas en el apartado dedicado a vistas.
La sintaxis de la orden es la siguiente:



Significado de los parámetros y palabras clave de uso más común:

- **Schema:** Es el nombre del esquema que contiene la tabla o la vista. Si se omite Oracle asume que la tabla o vista está en nuestro propio esquema.
- **Table, view:** Nombre de la tabla de la que se van a eliminar filas. Si se especifica el nombre de una vista Oracle eliminará las filas de la tabla base.
- **Alias:** Para dotar de un nombre distinto a la tabla, vista o subconsulta para ser referenciado en cualquier otro lugar en la sentencia. En la orden delete se usan generalmente con consultas subconsultas correlacionadas.
- **Subquery:** Es la subconsulta que selecciona los datos que van a ser eliminados. Este formato no es muy utilizado.

- **Where:** Se eliminarán únicamente las filas que satisfagan la condición. Si la cláusula where se omite todas las filas se actualizarán. La condición puede referenciar la tabla y puede contener una subconsulta. Si se omite la cláusula where eliminaríamos todas las filas de la tabla.

Ejemplo 44

Wilfred Lowell se jubila. Eliminémoslo de la base de datos. La consulta se escribiría así:

```
DELETE FROM EMPLEADO
WHERE NOMBRE='WILFRED LOWELL';
```

Esta acción sin embargo daría el siguiente error:

ORA-02292: integrity constraint (APUNTES.FK_OFICIOEMPLEADO_EMPLEADO)violated - child record found

Por tanto para borrar a Wilfred tendremos que borrar antes sus filas asociadas en la tabla oficioempleado. Procederíamos así:

```
DELETE
FROM OFICIOEMPLEADO
 WHERE EMPLEADO=(SELECT NUMEMP
                 FROM EMPLEADO
                       WHERE NOMBRE='WILFRED LOWELL')
```

Una vez hecho esto podemos escribir la primera orden.
Si en la definición de la foreign key, hubiésemos escrito la cláusula on delete cascade, la primera orden habría eliminado automáticamente las filas asociadas en la tabla oficioempleado.

Ejemplo 45

Eliminar a todos los herreros.

```
DELETE
FROM EMPLEADO
WHERE NUMEMP IN(SELECT NUMEMP
                FROM EMPLEADO E, OFICIOEMPLEADO OE,OFICIO O
                 WHERE E.NUMEMP=OE.EMPLEADO
                 AND OE.OFICIO=O.NUMOFICIO
                 AND O.OFICIO='HERRERO');
```

El resultado de esta orden daría el mismo error que la anterior, pero con una cláusula on delete cascade en la definición de la tabla oficioempleado habría funcionado perfectamente.

Lenguaje de control de datos (DCL)

Se correspondería con el sublenguaje LCD (DCL). Se utiliza para gestionar la confidencialidad y seguridad de la base de datos (ej.: creación de usuarios y permisos). Las sentencias para el control de transacciones caerían también bajo este tipo.

Usuarios, privilegios, roles y perfiles.

Cada usuario Oracle tiene un nombre y una clave de acceso, y posee sus propias tablas, vistas y otros recursos que cree.
Un **rol** Oracle es un conjunto de privilegios que se asigna a un usuario en función de sus responsabilidades. Podremos asignar privilegios directamente a los usuarios o bien, asignar los privilegios a un rol y posteriormente asignar dicho rol a un usuario concreto.
Los privilegios permitirán a los usuarios que los posean ejecutar determinadas acciones sobre la base de datos. Dentro de los privilegios hemos de distinguir dos tipos:

Privilegios del sistema.

Permiten ejecutar un conjunto de operaciones sobre la base de datos. Por ejemplo el privilegio Create Table permite crear tablas, el privilegio Create User permite crear usuarios.
La siguiente tabla contiene una referencia completa del conjunto de privilegios del sistema para una base de datos Oracle.

System Privilege	Operations Authorized
ALTER ANY CLUSTER	Allows grantee to alter any cluster in any schema.
ALTER ANY INDEX	Allows grantee to alter any index in any schema
ALTER ANY PROCEDURE	Allows grantee to alter any stored procedure, function, or package in any schema.
ALTER ANY ROLE	Allows grantee to alter any role in the database.
ALTER ANY SEQUENCE	Allows grantee to alter any sequence in the database.
ALTER ANY SNAPSHOT	Allows grantee to alter any snapshot in the database.
ALTER ANY TABLE	Allows grantee to alter any table or view in the schema.
ALTER ANY TRIGGER	Allows grantee to enable, disable, or compile any database trigger in any schema.
ALTER DATABASE	Allows grantee to alter the database.
ALTER PROFILE	Allows grantee to alter profiles.
ALTER RESOURCE COST	Allows grantee to set costs for session resources.
ALTER ROLLBACK SEGMENT	Allows grantee to alter rollback segments.
ALTER SESSION	Allows grantee to issue ALTER SESSION statements.
ALTER SYSTEM	Allows grantee to issue ALTER SYSTEM statements.
ALTER TABLESPACE	Allows grantee to alter tablespaces.
ALTER USER	Allows grantee to alter any user. This privilege authorizes the grantee to change another user's password or authentication method, assign quotas on **any** tablespace, set default and temporary tablespaces, and assign a profile and default roles.
ANALYZE ANY	Allows grantee to analyze any table, cluster, or index in any schema.
AUDIT ANY	Allows grantee to audit any object in any schema using AUDIT (Schema Objects) statements.
AUDIT SYSTEM	Allows grantee to issue AUDIT (SQL Statements) statements.
BACKUP ANY TABLE	Allows grantee to use the Export utility to incrementally export objects from the schema of other users.
BECOME USER	Allows grantee to become another user. (Required by any user performing a full database import.)

Table 4 – 11 System Privileges

System Privilege	Operations Authorized
COMMENT ANY TABLE	Allows grantee to comment on any table, view, or column in any schema.
CREATE ANY CLUSTER	Allows grantee to create a cluster in any schema. Behaves similarly to CREATE ANY TABLE.
CREATE ANY INDEX	Allows grantee to create an index in any schema on any table in any schema.
CREATE ANY PROCEDURE	Allows grantee to create stored procedures, functions, and packages in any schema.
CREATE ANY SEQUENCE	Allows grantee to create a sequence in any schema.
CREATE ANY SNAPSHOT	Allows grantee to create snapshots in any schema.
CREATE ANY SYNONYM	Allows grantee to create private synonyms in any schema.
CREATE ANY TABLE	Allows grantee to create tables in any schema. The owner of the schema containing the table must have space quota on the tablespace to contain the table.
CREATE ANY TRIGGER	Allows grantee to create a database trigger in any schema associated with a table in any schema.
CREATE ANY VIEW	Allows grantee to create views in any schema.
CREATE CLUSTER	Allows grantee to create clusters in own schema.
CREATE DATABASE LINK	Allows grantee to create private database links in own schema.
CREATE PROCEDURE	Allows grantee to create stored procedures, functions, and packages in own schema.
CREATE PROFILE	Allows grantee to create profiles.
CREATE PUBLIC DATABASE LINK	Allows grantee to create public database links.
CREATE PUBLIC SYNONYM	Allows grantee to create public synonyms.
CREATE ROLE	Allows grantee to create roles.
CREATE ROLLBACK SEGMENT	Allows grantee to create rollback segments.
CREATE SEQUENCE	Allows grantee to create sequences in own schema.
CREATE SESSION	Allows grantee to connect to the database.
CREATE SNAPSHOT	Allows grantee to create snapshots in own schema.
CREATE SYNONYM	Allows grantee to create synonyms in own schema.
CREATE TABLE	Allows grantee to create tables in own schema. To create a table, the grantee must also have space quota on the tablespace to contain the table.

Table 4 – 11 (continued) System Privileges

System Privilege	Operations Authorized
CREATE TABLESPACE	Allows grantee to create tablespaces.
CREATE TRIGGER	Allows grantee to create a database trigger in own schema.
CREATE USER	Allows grantee to create users. This privilege also allows the creator to assign quotas on **any** tablespace, set default and temporary tablespaces, and assign a profile as part of a CREATE USER statement.
CREATE VIEW	Allows grantee to create views in own schema.
DELETE ANY TABLE	Allows grantee to delete rows from tables or views in any schema or truncate tables in any schema.
DROP ANY CLUSTER	Allows grantee to drop clusters in any schema.
DROP ANY INDEX	Allows grantee to drop indexes in any schema.
DROP ANY PROCEDURE	Allows grantee to drop stored procedures, functions, or packages in any schema.
DROP ANY ROLE	Allows grantee to drop roles.
DROP ANY SEQUENCE	Allows grantee to drop sequences in any schema.
DROP ANY SNAPSHOT	Allows grantee to drop snapshots in any schema.
DROP ANY SYNONYM	Allows grantee to drop private synonyms in any schema.
DROP ANY TABLE	Allows grantee to drop tables in any schema.
DROP ANY TRIGGER	Allows grantee to drop database triggers in any schema.
DROP ANY VIEW	Allows grantee to drop views in any schema
DROP PROFILE	Allows grantee to drop profiles.
DROP PUBLIC DATABASE LINK	Allows grantee to drop public database links.
DROP PUBLIC SYNONYM	Allows grantee to drop public synonyms.
DROP ROLLBACK SEGMENT	Allows grantee to drop rollback segments.
DROP TABLESPACE	Allows grantee to drop tablespaces.
DROP USER	Allows grantee to drop users.
EXECUTE ANY PROCEDURE	Allows grantee to execute procedures or functions (stand–alone or packaged) or reference public package variables in any schema.
FORCE ANY TRANSACTION	Allows grantee to for the commit or rollback of any in–doubt distributed transaction in the local database. Also allows the grantee to induce the failure of a distributed transaction.

Table 4 – 11 (continued) System Privileges

System Privilege	Operations Authorized
FORCE TRANSACTION	Allows grantee to force the commit or rollback of own in-doubt distributed transactions in the local database.
GRANT ANY PRIVILEGE	Allows grantee to grant any system privilege.
GRANT ANY ROLE	Allows grantee to grant any role in the database.
INSERT ANY TABLE	Allows grantee to insert rows into tables and views in any schema.
LOCK ANY TABLE	Allows grantee to lock tables and views in any schema.
MANAGE TABLESPACE	Allows grantee to take tablespaces offline and online and begin and end tablespace backups.
READUP	Allows grantee to query data having an access class higher than the grantee's session label. This privilege is only available in Trusted Oracle7.
RESTRICTED SESSION	Allows grantee to logon after the instance is started using the Server Manager STARTUP RESTRICT command.
SELECT ANY SEQUENCE	Allows grantee to reference sequences in any schema.
SELECT ANY TABLE	Allows grantee to query tables, views, or snapshots in any schema.
UNLIMITED TABLESPACE	Allows grantee to use an unlimited amount of any tablespace. This privilege overrides any specific quotas assigned. If you revoke this privilege from a user, the grantee's schema objects remain but further tablespace allocation is denied unless authorized by specific tablespace quotas. You cannot grant this system privilege to roles.
UPDATE ANY TABLE	Allows grantee to update rows in tables and views in any schema.
WRITEDOWN	Allows grantee to create, alter, and drop schema objects and to insert, update, and delete rows having access classes lower than the grantee's session label. This privilege is only available in Trusted Oracle7.
WRITEUP	Allows grantee to create, alter, and drop schema objects and to insert, update, and delete rows having access classes higher than the grantee's session label. This privilege is only available in Trusted Oracle7.

Table 4 – 11 (continued) System Privileges

Oracle crea automáticamente algunos roles y les asigna determinados privilegios. La siguiente tabla muestra los roles creados automáticamente por Oracle, así como los privilegios asignados a cada uno de ellos. Oracle mantiene estos roles por compatibilidad hacia versiones antiguas, pero no recomienda su uso directo, en lugar de esto propone definir nuestros propios roles asignando los privilegios estrictamente necesarios.

Role	System Privileges and Roles Granted
CONNECT	ALTER SESSION CREATE CLUSTER CREATE DATABASE LINK CREATE SEQUENCE CREATE SESSION CREATE SYNONYM CREATE TABLE CREATE VIEW
RESOURCE	CREATE CLUSTER CREATE PROCEDURE CREATE SEQUENCE CREATE TABLE CREATE TRIGGER
DBA	All systems privileges WITH ADMIN OPTION EXP_FULL_DATABASE role IMP_FULL_DATABASE role
EXP_FULL_DATABASE	SELECT ANY TABLE BACKUP ANY TABLE INSERT, UPDATE, DELETE ON sys.incexp sys.incvid sys.incfil
IMP_FULL_DATABASE	BECOME USER WRITEDOWN (in Trusted Oracle7)

Table 4 – 12 Roles

Si asignamos o revocamos el rol **RESOURCE** o **DBA** a un usuario Oracle automáticamente asigna o revoca el privilegio de sistema **UNLIMITED TABLESPACE** para el usuario. Si a un usuario se le asigna cualquiera de estos dos roles y tuviese previamente cualquier restricción de espacio sobre un tablespace concreto, esta restricción automáticamente desaparecería, ya que el privilegio UNLIMITED TABLESPACE permite usar una cantidad ilimitada de espacio en cualquier tablespace.

Aunque el privilegio DBA tiene todos los privilegios, una percepción común es que no se necesitan privilegios adicionales para administrar privilegios sobre objetos de otros esquemas de la base de datos. Aunque esto se puede considerar como cierto hay que tener cuidado en algunas situaciones. Por ejemplo, si un usuario ADMIN que tuviese el privilegio DBA, quisiese crear una restricción de clave ajena que referenciase una tabla de un usuario USUARIO1, debería tener explícitamente concedido el privilegio REFERENCES sobre la tabla de USUARIO1 a la que ADMIN quisiese referenciar en una restricción de clave ajena de una de sus tablas. La concesión de este privilegio la tendría que dar USUARIO1, o bien ADMIN debería conectarse como USUARIO1 para otorgar este privilegio. ADMIN siempre podría conectarse como USUARIO1 ya que tiene el privilegio de sistema ALTER USER que le permitiría cambiar la clave de acceso de USUARIO1.

Privilegios sobre objetos.

Permiten realizar determinadas operaciones sobre objetos de un determinado esquema de la base de datos. Por ejemplo, el privilegio **SELECT** permite seleccionar filas de una determinada tabla. El privilegio del sistema **SELECT ANY TABLE** permitiría seleccionar filas de cualquier tabla de cualquier esquema.
Los privilegios sobre objetos son los siguientes:

- **ALTER:** permite alterar una tabla, vista o secuencia.
- **DELETE:** permite eliminar filas de una tabla o vista.
- **EXECUTE:** Permite ejecutar procedimientos, funciones y paquetes de un determinado esquema.
- **INDEX:** Permite crear índices sobre una tabla.
- **REFERENCES:** Permite definir restricciones de clave ajena sobre una tabla.
- **SELECT:** Permite seleccionar filas de una tabla, vista o secuencia.
- **UPDATE:** Permite modificar columnas de una tabla o vista.

El propietario de los objetos siempre tiene todos estos privilegios sobre los objetos que ha creado en su esquema, así como la posibilidad de conceder los privilegios a otros usuarios.

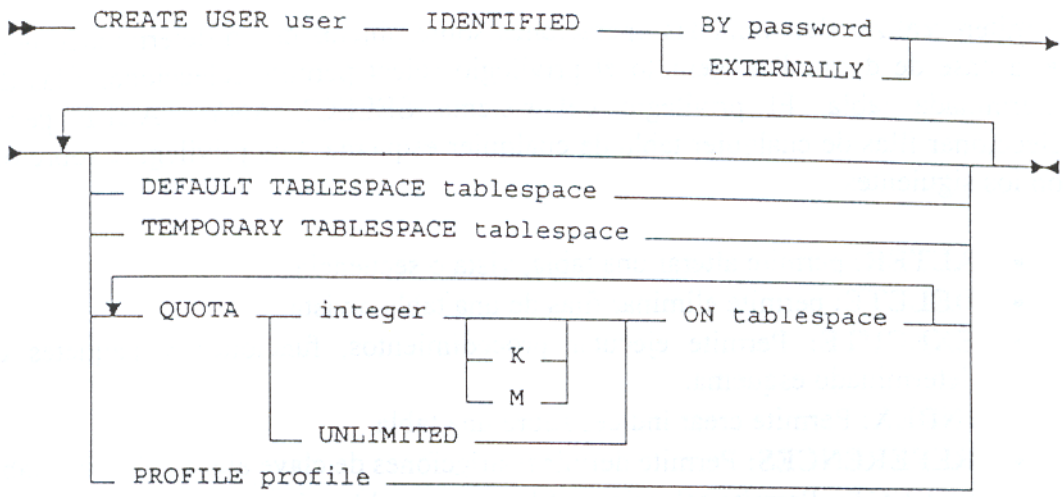
Perfiles.

Un perfil es un conjunto de limitaciones sobre los recursos de la base de datos. Si a un usuario se le asigna un determinado perfil, el usuario no podrá exceder estos límites.
Por ejemplo, podemos asignar a un perfil la limitación de sesiones concurrentes que un usuario puede tener mediante la cláusula de la orden **CREATE PROFILE**, **SESSION_PER_USER**.
Para poder crear un perfil se debe poseer el privilegio del sistema **CREATE PROFILE**.

Una vez vistos estos conceptos básicos, vamos a analizar las ordenes que nos permitirán crear usuarios, roles, perfiles y la asignación de privilegios tanto de sistema como sobre objetos.

Creación de usuarios.

La sentencia que nos permite crear usuarios es CREATE USER. Para poder crear usuarios se debe poseer el privilegio del sistema CREATE USER. La sintaxis de la orden es la siguiente:



Palabras clave y parámetros:

- **User:** Es el nombre del usuario que va a ser creado.
- **Identified:** Indica como permitirá Oracle el acceso de dicho usuario:
    - **By password:** El usuario especifícamente dará una clave de acceso para conectarse al sistema.
    - **Externally:** Oracle verificará que el nombre de usuario dentro de una cuenta del sistema operativo se corresponde con el nombre de usuario especificado en una conexión de la base de datos.
- **Default tablespace:** Identifica el espacio de tablas por defecto para los objetos que el usuario crea. Si se omite la cláusula, el espacio de tablas por defecto será USERS.
- **Temporary tablespace:** Identifica el espacio de tablas para los segmentos temporales de usuario. Si se omite el espacio de tablas temporal es TEMP.
- **Quota:** Permite al usuario poseer espacio es en espacio de tablas y establecer un límite que puede ser dado en kbytes o Megabytes. Se puede usar la opción UNLIMITED en cuyo caso el usuario no tendría ninguna restricción de espacio.
- **Profile:** Asigna el perfil llamado profile el usuario. El perfil limitará la cantidad de recursos que el usuario puede usar.

Nota: Un tablespace es una unidad lógica de almacenamiento del gestor Oracle, que está relacionado con uno o varios archivos de datos. Se usan para tener agrupadas las tablas que están relacionadas con un mismo sistema.
Existe un tablespace espacial llamado SYSTEM, donde Oracle sitúa las tablas que componen el diccionario de datos. Este tablespace debe usarse en exclusividad para este cometido y no asignar dicho tablespace a ningún usuario.

Ejemplo 1

```
CREATE USER PEPE IDENTIFIED BY PEPE;
```

Crearíamos un usuario llamado PEPE y clave de acceso es PEPE.

Un usuario así creado no tiene ningún privilegio, ni siquiera podría conectarse a la base de datos; si lo intentáramos obtendríamos el siguiente mensaje de error:

user PEPE lacks CREATE SESSION privilege; logon denied

Oracle nos está diciendo que le falta el privilegio CREATE SESSION y por tanto niega su conexión.
Para añadirle dicho privilegio y que PEPE pueda conectarse le asignaríamos el privilegio CREATE SESSION mediante la orden grant.

```
GRANT CREATE SESSION TO PEPE;
```

Aun así, PEPE no podría hacer prácticamente nada: no podría crear tablas, seleccionar datos de ninguna tabla de ningún esquema, etc. Si podría por ejemplo cambiar su clave de acceso, mediante la orden ALTER USER.

Mediante la asignación de privilegios tanto de sistema como sobre objetos el usuario PEPE podrá ir haciendo cada vez más cosas.

El espacio de tablas asignado a PEPE sería USERS ya que es el espacio de tablas asignado por defecto.

Ejemplo 2

```
CREATE USER PEPE
IDENTIFIED BY PEPE
DEFAULT TABLESPACE GESTION
TEMPORARY TABLESPACE TEMP
QUOTA 20M ON GESTION
QUOTA 20M ON TEMP
```

Hemos creado un usuario PEPE con clave de acceso PEPE, con un límite de 20 megabyte en el espacio de tablas GESTION y un límite de 20 megabyte en el espacio de tablas TEMP.

Para que PEPE pueda conectarse y crear tablas en su espacio asignamos los privilegios necesarios mediante la orden GRANT.

```
GRANT CREATE SESSION,CREATE TABLE TO PEPE;
```

PEPE ya podría crear tablas en su esquema que se alojarían en el espacio de tablas GESTION. Seguiría sin poder crear por ejemplo vistas u otros objetos para los que no tiene permisos.

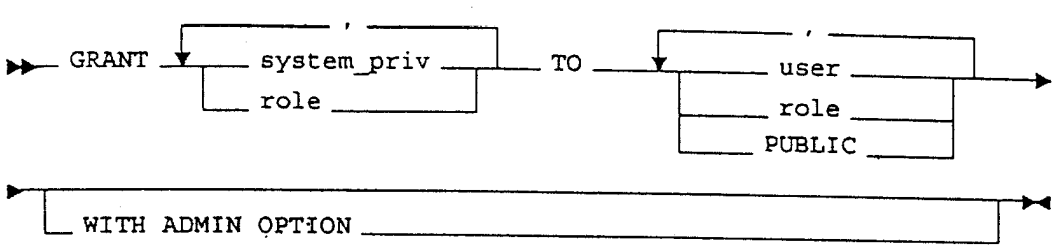
Asignación de permisos.

Hemos visto en el apartado anterior como crear un usuario y en algún ejemplo hemos visto como asignar permisos. En este apartado estudiaremos la orden para conceder permisos con mayor profundidad.
La orden para asignar permisos (GRANT) tiene dos formatos distintos, según se vayan a asignar privilegios de sistema o privilegios sobre objetos.

GRANT(PRIVILEGIOS DE SISTEMA Y ROLES)

Este primer formato permite asignar privilegios de sistema y roles a un usuario, a un rol o a todos los usuarios.
Para poder asignar un privilegio de sistema a un usuario o rol, el usuario debe haber recibido el privilegio con la cláusula **with admin option,** o bien poseer el privilegio del sistema **grant any privilege.**

El formato de la orden es el siguiente:



Palabras clave y parámetros:

- **System_priv:** Es el privilegio del sistema que va a ser asignado.
- **Role:** Es el rol que vamos a asignar.
- **To:** Identifica quien va a ser el receptor del privilegio del sistema o rol a asignar. Puede ser un usuario, otro rol o PUBLIC(afectaría a todos los usuarios).
- **With admin option:** Permite al usuario receptor de los privilegios o roles, asignar los privilegios o roles recibidos a otros usuarios. Si se concede un rol a un usuario con esta cláusula, el usuario receptor puede alterar o borrar el rol.

Ejemplo 1

Asignamos los privilegios connect y resource al usuario PEPE.

```
GRANT CONNECT,RESOURCE TO PEPE;
```

Ejemplo 2:

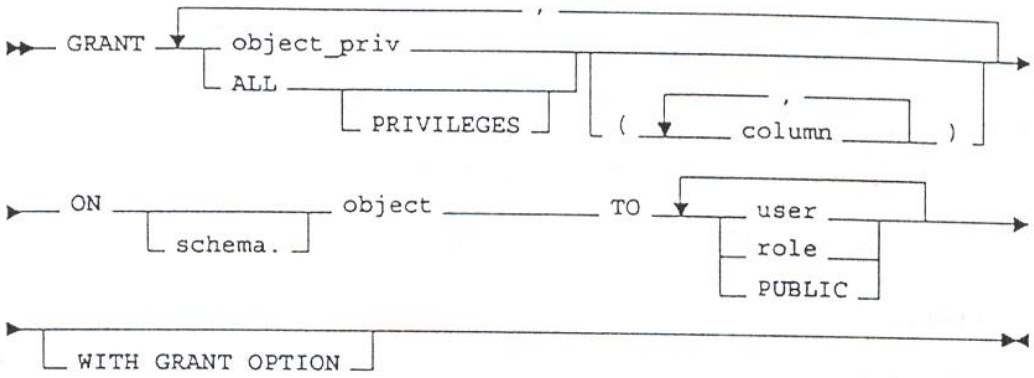
Pepe puede seleccionar ahora datos de cualquier tabla y además conceder este privilegio a otros usuarios.

```
GRANT SELECT ANY TABLE TO PEPE
WITH ADMIN OPTION;
```

GRANT (PRIVILEGIOS SOBRE OBJETOS)

Este segundo formato permite asignar privilegios para un objeto particular a usuarios y roles. Para poder asignar un privilegio sobre un objeto de un determinado esquema, se debe ser el propietario del objeto o bien haber recibido el privilegio con la opción **with grant option.**

El formato de la orden es el siguiente:



Palabras clave y parámetros:

- **Object_priv:** Es el privilegio de objeto a ser asignado.
- **All privilegies:** Permite conceder todos los privilegios sobre un objeto, posibilitando al usuario receptor concederlos a otros usuarios (como si se hubiese incluido la cláusula with grant option).
- **Column:** Especifica las columnas sobre una tabla o vista sobre las que se conceden los privilegios. Unicamente se pueden especificar columnas cuando estamos concediendo los privilegios **insert, references o update.** Si no se especifica la lista de columnas, la asignación de privilegios afecta a todas las columnas de la tabla o vista.
- **On:** Identifica el objeto sobre el que se están asignando los privilegios. Puede ser cualquiera de los siguientes: tabla, vista, secuencia, procedimiento, función o paquete, sinónimos para una tabla, vistas procedimientos, funciones o paquetes.
- **To:** Identifica el objeto receptor de los privilegios. Puede ser un usuario, un rol o PUBLIC (todos los usuarios).
- **With grant option:** Permite al receptor asignar los privilegios recibidos a otros usuarios o roles. El receptor de esta cláusula debe ser un usuario o PUBLIC, es decir, esta cláusula no se puede asignar a un rol.

Ejemplo 1:

Asignamos el privilegio de seleccionar datos de nuestra tabla empleado a todos los usuarios.

```
GRANT SELECT ON EMPLEADO TO PUBLIC
```

Para poder dar este privilegio la tabla empleado debe pertenecer al usuario que concede el privilegio o tener el privilegio del sistema **GRANT ANY OBJECT PRIVILEGE**. Si concedemos un privilegio sobre un objeto del que no somos propietario, y lo hacemos por disponer del privilegio **GRANT ANY OBJECT PRIVILEGE** en ese caso estamos actuando como si fuésemos el propietario del obeto.
Ejemplo:
Damos privilegio de selección sobre la tabla EMPLEADO del esquema VS1DAWAPUNTES1 a todos los usuarios de la base de datos

```
GRANT SELECT ON EUROVISION.CANTANTE TO PUBLIC
```

Una vez dado el privilegio de selección sobre la tabla empleado del usuario VS1DAWAPUNTES1.EMPLEADO a PUBLIC, es decir, a todos los usuarios, cualquier usuario de la base de datos podría seleccionar los datos de la tabla mediante la orden:

```
SELECT * FROM EUROVISION.CANTANTE
```

Ejemplo 2:

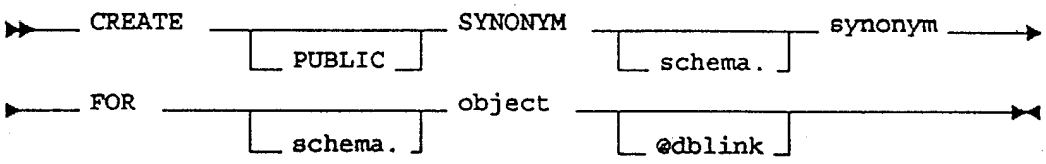
```
GRANT REFERENCES(NUMALOJ),UPDATE(ALOJAMIENTO,DIRECCION)
ON ALOJAMIENTO
TO PUBLIC;
```

Damos la posibilidad de referenciar numaloj en una restricción de clave ajena, y actualizar las columnas alojamiento y dirección a todos los usuarios.

Sinónimos

Introducimos en este punto otro nuevo objeto de la base de datos, los sinónimos, puesto que es aquí donde tiene sentido su uso.
Un sinónimo es un nombre alternativo para una tabla, vista, secuencia, procedimiento, función u otro sinónimo.
Para crear un sinónimo en nuestro esquema debemos tener el privilegio del sistema **CREATE SYNONYM.**
Para crear un sinónimo en un esquema distinto al nuestro debemos tener el privilegio del sistema **CREATE ANY SYNONYM**.
Para crear un sinónimo publico debemos tener el privilegio del sistema **CREATE PUBLIC SYNONYM**.

La sintaxis de la orden para crear sinónimos es:



Palabras clave y parámetros:

- **Public:** Crea un sinónimo público. Es accesible a todos los usuarios.
- **Schema:** Denota el esquema que contendrá el sinónimo. Si el sinónimo es público no se puede especificar el esquema. Si se omite, el sinónimo se creará dentro del esquema en el que estemos conectados.
- **Synonym:** Es el nombre dado al sinónimo.
- **For:** Identifica el objeto para el que se ha creado el sinónimo. El objeto puede ser de uno de los siguientes tipos: tabla, vista, secuencia, procedimiento, función o paquete, u otro sinónimo.

Ejemplo de uso

Supongamos que soy el usuario VS1DAWAPUNTES1.

En el apartado anterior se me ha concedido permiso de selección sobre la tabla CANTANTE del usuario EUROVISION.

Puedo ahora crearme un sinónimo para esa tabla de forma que la orden para seleccionar sea más cómoda. Procederíamos de la siguiente manera:

```
CREATE SYNONYM CANTANTEEUROVISION FOR EUROVISION.CANTANTE
```

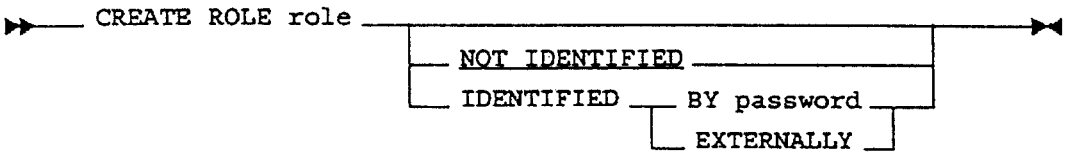
Para seleccionar ahora los datos de la tabla CANTANTE del usuario EUROVISION, el usuario VS1DAWAPUNTES1 podría escribir ahora la orden de la siguiente manera:

```
SELECT * FROM CANTANTEEUROVISION
```

usando el sinónimo en lugar el nombre completo de la tabla.

Creación de Roles.

Para crear un role ejecutaremos la orden CREATE ROLE. Debemos tener el privilegio del sistema CREATE ROLE. La sintaxis de la orden es la siguiente:



Ejemplo:

```
CREATE ROLE CONECTAR – Creamos el rol conectar

GRANT CREATE SESSION,CREATE TABLE TO CONECTAR
```

Asignamos los privilegios de conexión y de creación de tabla al rol conectar.

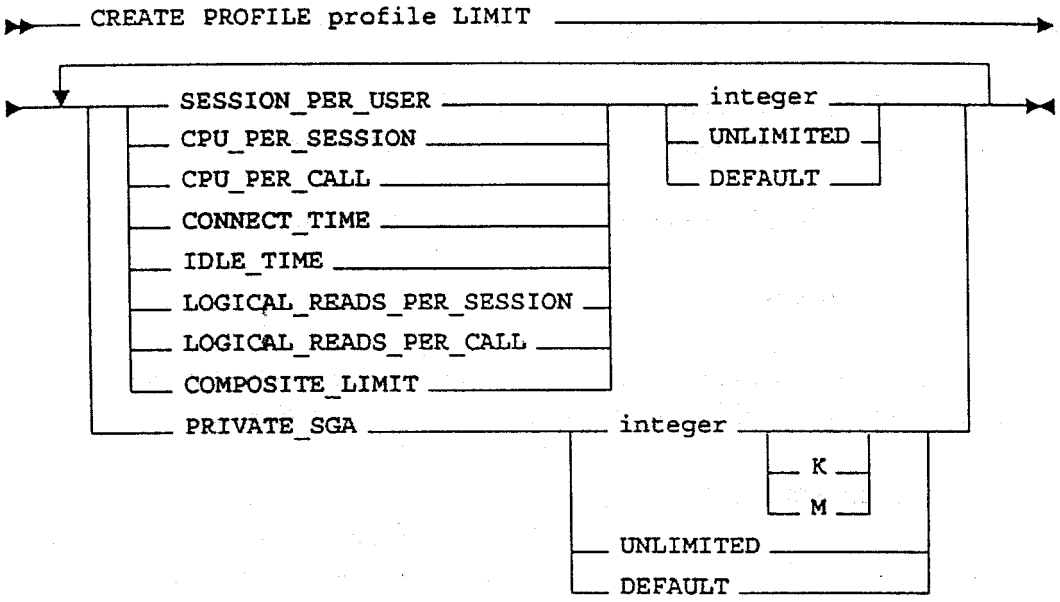
```
CREATE USER PEPE        -- Orden comentada anteriormente
IDENTIFIED BY PEPE
DEFAULT TABLESPACE GESTION
TEMPORARY TABLESPACE TEMP
QUOTA 20M ON GESTION
QUOTA 20M ON TEMP

GRANT CONECTAR TO PEPE;
```

Asignamos el rol conectar a PEPE. Pepe ahora podrá conectar y crear sus tablas.
Esta es la forma de trabajo habitual, crear un rol, asignar los permisos estrictamente necesarios a ese rol, y asignar los permisos a usuarios con necesidades equivalentes a través de los roles específicamente diseñados para ese grupo de usuarios.

Creación de perfiles

Permiten establecer un conjunto de límites sobre los recursos de la base de datos. Si asignamos el perfil a un usuario, éste tendrá que trabajar con los límites impuestos al perfil. Para poder crear un perfil debemos tener el privilegio del sistema **CREATE PROFILE**. La sintaxis es la siguiente:




```
CREATE PROFILE  CONEXIONESSIMULTANEAS3 LIMIT SESSIONS_PER_USER 3;
```

Creamos un perfil de forma que un usuario con este perfil no puede tener más de tres sesiones abiertas simultáneamente.

Para asignárselo a PEPE utilizaríamos la orden ALTER USER de la siguiente manera:

```
ALTER USER PEPE
PROFILE CONEXIONESSIMULTANEAS3;
```

Para que la limitación de recursos sea efectiva, se debe habilitar el parámetro de inicilización RESOURCE_LIMIT.
Para ello, y teniendo los privilegios suficientes, escribiríamos:

```
ALTER  SYSTEM SET resource_limit = TRUE scope = BOTH
```

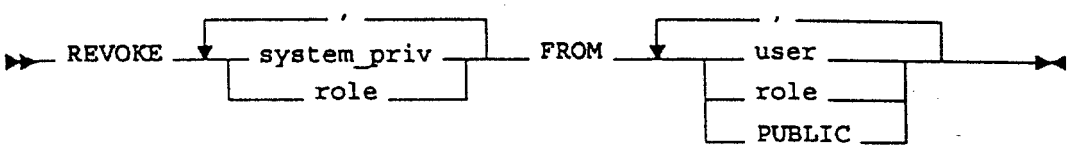
Retirada de privilegios a usuarios o roles.

Al igual que hemos concedido privilegios, podemos retirarlos mediante la orden **REVOKE**. Al igual que teníamos dos formatos para la orden grant, uno para conceder privilegios de sistema y otra para conceder privilegios sobre objetos, tenemos dos formatos para la orden revoke.

Retirada de privilegios del sistema. Revoke(privilegios del sistema).

La orden revoke se utiliza para retirar los privilegios dados a usuarios y roles. Para poder retirar un privilegio del sistema a un usuario, se nos ha debido conceder el privilegio del sistema o rol con la cláusula with admin option, o bien tener el privilegio del sistema GRANT ANY ROLE.

El formato de la orden es muy sencillo:



Ejemplo 1

Para retirar el rol conectar a Pepe, escribiríamos:

```
REVOKE CONECTAR FROM PEPE;
```

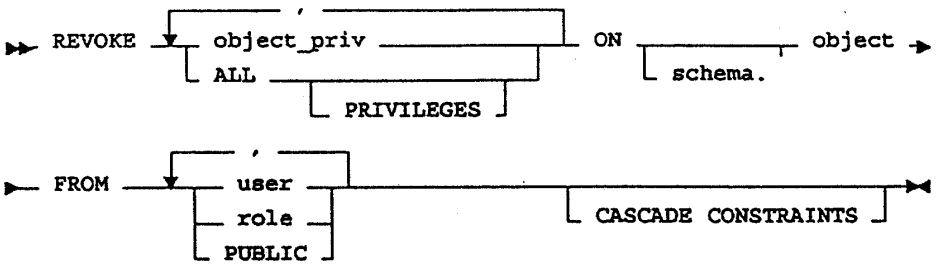
Hay que tener cuidado en una cosa, si los privilegios se han concedido a través de un rol, no los podremos retirar de forma individual. Podríamos quitar el privilegio al rol y afectaría a todos los usuarios que tuviesen ese rol.

Ejemplo 2

```
REVOKE CREATE TABLE FROM CONECTAR;
```

Retirada de privilegios sobre objetos. Revoke(privilegios sobre objetos)

Para retirar privilegios previamente concedidos sobre un objeto particular a usuarios y roles. La sintaxis de la orden es la siguiente:

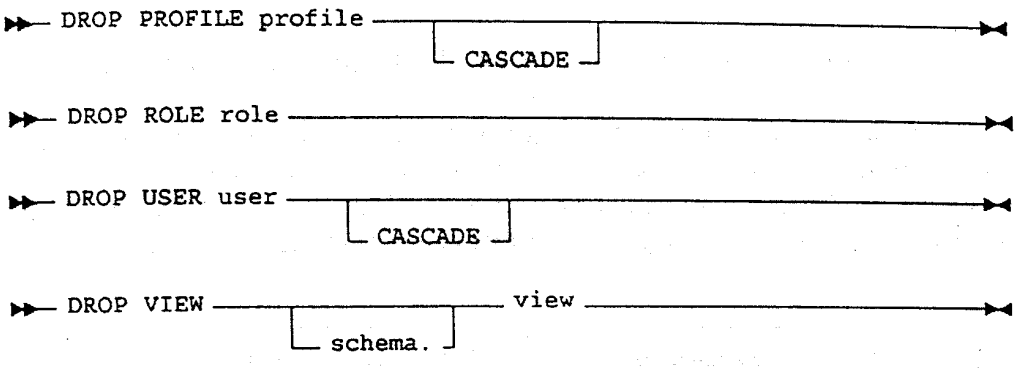


Ejemplo1:

Retirar el privilegio de selección sobre nuestra tabla CANTANTE al usuario PEPE. Escribiríamos

```
REVOKE SELECT ON EMPLEADO FROM PEPE;
```

Eliminación de perfiles, roles, usuarios y vistas.



ANEXO I. Funciones de manejo de cadenas.

1. Concatenación (||)

Sintaxis:
cadena || cadena

Ejemplo:

```
SELECT 'Me llamo '
          || RTRIM(nombre)
          ||  ' y tengo '
          || edad
          || ' años' "Ejemplo de concatenación"
FROM EMPLEADO;
```

Ejemplo de concatenación

Me llamo ADAH TALBOT y tengo 23 años
Me llamo ANDREW DYE y tengo 29 años
Me llamo BART SARJEANT y tengo 22 años
Me llamo DICK JONES y tengo 18 años
...

2. RPAD y LPAD

Aumentan el tamaño de una cadena por la derecha y por la izquierda respectivamente con cualquier conjunto de caracteres.

Sintaxis:
RPAD(cadena, longitud,[,'conjunto'])
LPAD(cadena, longitud,[,'conjunto'])

Ejemplos

```
SELECT RPAD(TRIM(nombre),30,'#')NOMBRE, edad
FROM EMPLEADO
```

NOMBRE	EDAD
ADAH TALBOT##################	23
ANDREW DYE####################	29
BART SARJEANT#################	22
…….	……

```
SELECT LPAD(TRIM(nombre),30,'#') NOMBRE, EDAD
 FROM EMPLEADO;
```

NOMBRE	EDAD
##################ADAH TALBOT	23
##################ANDREW DYE	29
#################BART SARJEANT	22
……	……

3. LOWER, UPPER y INITCAP

Sintaxis:
LOWER(cadena) → Convierte a minúsculas
UPPER(cadena) → Convierte a mayúsculas
INITCAP(cadena) → Pone la inicial de palabra a mayúscula

Ejemplo

```
SELECT INITCAP(LOWER(nombre)) NOMBRE
FROM EMPLEADO;
```

NOMBRE
Adah
Talbot
Andrew
Dye
Bart
Sarjeant
Dick
Jones

4. LTRIM y RTRIM

Sintaxis:
LTRIM(cadena [, 'conjunto']) → Recorta por la izquierda la cadena
RTRIM(cadena [, 'conjunto']) → Recorta por la derecha la cadena

Donde cadena es una columna de una tabla (o literal cadena) y conjunto es la colección de caracteres que se quieren eliminar. Si no se especifica conjunto la función suprime los espacios.

Ejemplo:

```
SELECT nombre, LTRIM(RTRIM(RTRIM(nombre),'T'),'A')
FROM EMPLEADO
```

NOMBRE	RECORTADO
ADAH TALBOT	DAH TALBO
ANDREW DYE	NDREW DYE
BART SARJEANT	BART SARJEAN
……	….

El RTRIM más interno le quita los espacios en blanco finales al nombre. El siguiente RTRIM suprime la T del final del nombre y el LTRIM le quita la A al inicio del nombre. Si se especifican más de un carácter en el conjunto no quiere decir que elimina la palabra sino cualquiera de los caracteres indicados en cualquier orden hasta que se encuentre un carácter que no esté en el conjunto. Así por ejemplo si utilizamos la función LTRIM(cadena,'LOS'), esto no quiere decir que queremos eliminar del inicio de la cadena la palabra LOS, sino que eliminará L, O o S. Si la cadena empezara por LATA eliminaría la L y si empezara por SLAVO eliminaría la SL. Si quisiéramos eliminar palabras deberíamos usar otras funciones como INSTR, SUBSTR, e incluso DECODE.

5. LENGTH

Sintaxis:
LENGTH(cadena) → devuelve la longitud en caracteres de la cadena.

Ejemplo:

```
SELECT NOMBRE, LENGTH(RTRIM(nombre)) NUMEROCARACTERES
FROM EMPLEADO
```

NOMBRE	NUMCARACTERES
ADAH TALBOT	11
ANDREW DYE	10
BART SARJEANT	13
……	….

6. SUBSTR

Sintaxis:
SUBSTR(cadena, comienzo [, cuenta])→
Recorta la cadena desde el comienzo carácter tantos caracteres como cuenta indica.

Ejemplo:

```
SELECT SUBSTR(nombre,3,10) SUBCADENA
FROM EMPLEADO;
```

NOMBRE
AH TALBOT
DREW DYE
RT SARJEAN
…

7. INSTR

Sintaxis:
INSTR(cadena, conjunto [,comienzo [, ocurrencia]]) → Busca la posición donde se encuentra conjunto en cadena comenzando la búsqueda desde la posición del carácter comienzo siempre que se encuentre ocurrencias veces el conjunto en la cadena.

Ejemplo:

```
SELECT NOMBRE, INSTR(nombre,'A',1,2) "Posición de la 2ª A"
FROM EMPLEADO;
```

NOMBRE	Posición de la 2ª A
ADAH TALBOT	3
ANDREW DYE	0
BART SARJEANT	7
……	….

8. REPLACE

Sintaxis:
REPLACE(cadena, cadenaabuscar, cadenareemplaza) → Busca cadenaabuscar en cadena y si la encuentra la sustituye por cadenareemplaza.

Ejemplo

```
SELECT DESCRIPCION, REPLACE(descripcion,'AR', 'AARR') "AARR por AR"
FROM OFICIO
```

DESCRIPCION	AARR por AR
MARCAR Y TALAR ARBOLES, CORTAR, APILAR	MAARRCAARR Y TALAARR AARRBOLES, CORTAARR, APILAARR
CONDUCIR, AJUSTAR MAQUIA	CONDUCIR, AJUSTAARR MAQUIA
PREPARAR FUEGO, CORTAR, HERRAR CABALLOS	PREPAARRAARR FUEGO, CORTAARR, HERRAARR CABALLOS
……	….

En este ejemplo hemos sustituido la palabra AR por AARR en la descripción de la tabla eaoficio.

9. TRANSLATE

Sintaxis:

TRANSLATE(cadena, si, entonces) →

Esta función mira cada carácter de la cadena y la busca en la cadena si, si no lo encuentra el carácter en cadena no se sustituirá, pero si lo encuentra sustituye el carácter de cadena por el carácter de entonces de la posición encontrada en si.

Ejemplo:

```
SELECT NOMBRE, TRANSLATE(Nombre, 'EIOU', 'UOIE')
FROM EMPLEADO
```

NOMBRE	Posición de la 2ª A
ADAH TALBOT	ADAH TALBIT
ANDREW DYE	ANDRUW DYU
BART SARJEANT	BART SARJUANT
……	….

10. DECODE

Sintaxis:
DECODE(valor, si1, entonces1, si2, entonces2, si3, entonces3,...,si no) →
Translate producía una sustitución carácter a carácter, DECODE se puede considerar una sustitución valor a valor. Para cada valor que se ve en un campo, DECODE comprueba si coincide con un si y en ese caso lo cambia por el entonces.

Ejemplos:

```
SELECT nombre, alojamiento,
DECODE(alojamiento, 1,2,2,1,alojamiento) "Nuevo Alojamiento"
FROM EMPLEADO
```

NOMBRE	Posición de la 2ª A	
ADAH TALBOT	4	4
ANDREW DYE	5	5
BART SARJEANT	1	2
DICK JONES	5	5
DONALD ROLLO	2	1
….	….	…..

```
SELECT nombre,edad,sueldo,
DECODE (TRUNC(edad/20),1,sueldo*1.5,NULL) "+50%>20a",
DECODE (TRUNC(edad/20),2,sueldo*2,NULL) "+100%>40a",
DECODE (TRUNC(edad/20),3,sueldo*2.5,NULL) "+150%>60a"
FROM EMPLEADO;
```

NOMBRE	EDAD	SUELDO	"+50%>20a	100%>40	150%>60a
ADAH TALBOT	23	1,5	2,25		
ANDREW DYE	29	1,13	1,695		
BART SARJEANT	22	1,13	1,695		
DICK JONES	18	1,5			
DONALD ROLLO	16	1,13			
ELBERT TALBOT	43				
GEORGE OSCAR	41	1,88		3,76	
GERHARDT KENTGEN	55	2,25		4,5	
HELEN BRANDT	15	2,25			
JED HOPKINS	33	2,63	3,945		
JOHN PEARSON	27	6	9		
KAY Y PALMER WALLBOM					
PAT LAVAY	21	1,88	2,82		
PETER LAWSON	25	1,88	2,82		
RICHARD KOCH Y HERMANOS					
ROLAND BRANDT	35	0,75	1,125		
VICTORIA LYNN	32	1,5	2,25		
WILFRED LOWELL	67	0,75			1,875
WILLIAM SWING	15	1,5			

11. CHR(n) y ASCII(char)

Estas funciones convierten de ASCII a carácter y al contrario.
CHR(65) ← 'A' ASCII('A') ← 65

ANEXO II. Funciones de manejo de fechas

Como dijimos al principio de este tema el tipo de dato DATE almacena una fecha en 7 bytes. Se guarda el siglo, año, mes, día, hora, minutos y segundos de una fecha. Permiten almacenar fechas de la era Juliana en el rango: 1 de Enero de 4712 antes de Cristo hasta el 9999 de nuestra era.

Aritmética de fechas

Cuando sumamos 1 a una fecha, estamos sumando un día completo.
La diferencia entre dos datos de tipo fecha es igual al número de días transcurridos entre esas dos fechas. Por ejemplo: una diferencia de 0,5 indicaría un período de 12 horas.

Funciones relacionadas con los tipos de dato DATE

SYSDATE

Devuelve la fecha del sistema

```
SELECT SYSDATE FROM dual
```

La salida en SqlDeveloper mostraría la fecha en el formato que se muestra a continuación: 14/06/20

Ese formato se puede modificar (Ver más adelante la función TO_CHAR) para mostrar la fecha con otros formatos. Sirva como adelanto la siguiente orden que muestra la fecha y hora del sistema en formato día, mes, año con cuatro cifras y la hora y minutos.

```
SELECT to_char(SYSDATE,'DD-MM-YYYY HH24:MI')FECHA_Y_HORA_ACTUAL
FROM dual
```

FECHA_Y_HORA_ACTUAL
14-06-2020 19:58

ADD_MONTHS

Sintaxis: ADD_MONTHS(d,n) → devuelve la fecha d a la que se ha sumado n meses.

Ejemplo

```
SELECT NOMBRE,FECHANACIMIENTO,
ADD_MONTHS(FECHANACIMIENTO,5) "5 meses+",
ADD_MONTHS(FECHANACIMIENTO,-5) "5 meses-"
FROM ALUMNO
```

NOMBRE	FECHANACIMIENTO	5 meses+	5 meses-
Bargueño Del Alamo, David	31/03/85	31/08/85	31/10/84
Blázquez Rodríguez, Alberto Santiago	07/02/89	07/07/89	07/09/88
Calvo Pérez, Marta	01/10/91	01/03/92	01/05/91

LAST_DAY

Sintaxis: LAST_DAY(d) → Obtiene la fecha del último día del mes de la fecha d.

Ejemplos:

```
SELECT sysdate,
LAST_DAY(sysdate) "Último",
LAST_DAY(sysdate)-sysdate "faltan días"
FROM dual
```

sysdate	Último	faltan días
14/06/20	30/06/20	16

```
SELECT SYSDATE,
LAST_DAY(ADD_MONTHS(SYSDATE,-1))+1 PRIMERODEMES
FROM dual
```

SYSDATE	PRIMERODEMES
14/06/20	01/06/20

MONTHS_BETWEEN

Sintaxis: MONTHS_BETWEEN(d1, d2) → devuelve los meses transcurridos entre dos fechas.

Ejemplo:

```
SELECT  NOMBRE, SYSDATE, FECHANACIMIENTO,
TRUNC(MONTHS_BETWEEN(SYSDATE,FECHANACIMIENTO)/12) EDAD
FROM ALUMNO
```

NOMBRE	SYSDATE	FECHANACIMIENTO	EDAD
Bargueño Del Alamo, David	14/06/20	31/03/85	35
Blázquez Rodríguez, Alberto Santiago	14/06/20	07/02/89	31
Calvo Pérez, Marta	14/06/20	01/10/91	28
Camarma Villar, María de la Merced	14/06/20	24/11/74	45
----------	----------	----------	----------

NEXT_DAY

Sintaxis: NEXT_DAY(fecha,diasemana) → Obtiene la fecha siguiente a la dada en la que coincida con el día de la semana especificado.

Ejemplo: Suponemos que nuestros alumnos sólo pueden celebrar su cumpleaños en sábado. Calculemos la fecha en el que se celebrarán sus próximos cumpleaños.

```
SELECT  NOMBRE, FECHANACIMIENTO,
NEXT_DAY(
ADD_MONTHS(FECHANACIMIENTO,
           TRUNC(MONTHS_BETWEEN(SYSDATE,FECHANACIMIENTO)/12)*12+12)-
1,'SÁBADO')FIESTA
FROM ALUMNO
```

NOMBRE	FECHANACIMIENTO	FIESTA
Bargueño Del Alamo, David	31/03/85	03/04/21
Blázquez Rodríguez, Alberto Santiago	07/02/89	13/02/21
Calvo Pérez, Marta	01/10/91	03/10/20
Camarma Villar, María de la Merced	24/11/74	28/11/20

Hemos calculado inicialmente la edad de los alumnos. Con la edad calculamos los meses vividos en su cumpleaños anterior y le sumamos los nuevos 12 meses que vivirán cuando cumplan de nuevo años en este año. Todos los meses calculados se los añadimos a la fecha de nacimiento para obtener la fecha en la que cumplen años este mismo año. A esta fecha menos un día le calculamos la inmediata que coincida en sábado, que será la del gran día. La razón de restar uno la fecha es por si alguna de ellas ya es sábado y queremos celebrarlo ese mismo día y no al sábado siguiente.

Otra forma de realizar esta consulta:

```
SELECT   NOMBRE,FECHANACIMIENTO,
NEXT_DAY(to_date(to_char(fechanacimiento,'DD-MM')||
    '-'||TO_CHAR(SYSDATE,'yyyy'),'DD-MM-YYYY')-1,'SÁBADO')FIESTACUMPLEAÑOS
FROM ALUMNO
ORDER BY FECHANACIMIENTO,NOMBRE
```

En este caso construimos fecha de cumpleaños concatenando al año en curso, el mes y el día de la fecha de nacimiento de cada alumno. A la fecha obtenida, le aplicamos la función NEXT_DAY para obtener el siguiente sábado a la fecha de cumpleaños.
Para evitar que si el cumpleaños cae en sábado next_day no obtenga el sábado siguiente, a la fecha de cumpleaños del año en curso, le restamos 1.

GREATEST Y LEAST.

La función LEAST selecciona la fecha más temprana de una lista de fechas, tanto si son columnas como si son literales.
GREATEST selecciona la fecha más tardía. Estas funciones se usan igualmente con números y cadenas de caracteres.

Ejemplos

```
SELECT sysdate, sysdate+3,least(sysdate,sysdate+3)
from dual;
```

sysdate	sysdate+3	least(sysdate,sysdate+3)
14/06/20	17/06/20	14/06/20

```
SELECT sysdate, sysdate+3,greatest(sysdate,sysdate+3)
from dual;
```

sysdate	sysdate+3	greatest (sysdate,sysdate+3)
14/06/20	17/06/20	17/06/20

Debemos tener en cuenta que cuando utilizamos literales de fecha en estas funciones, lo que realmente se interpreta son cadenas, no fechas.

Ejemplo

```
SELECT LEAST('15-ENERO-92','15-DICIEMBRE-92') FROM dual
```

LEAST('15-ENERO-92','15-DICIEMBRE-92')
15-DICIEMBRE-92

La función ha operado bien la D es más pequeña que la Ej.
Sin embargo, nosotros queríamos que se interpretaran como fechas y para ello debemos utilizar funciones de conversión:

```
SELECT TO_CHAR(LEAST(TO_DATE('15-ENERO-2020','DD-MONTH-YYYY'),
            TO_DATE('15-DICIEMBRE-2020','DD-MONTH-YYYY')),'DD-MONTH-YYYY')
FROM dual;
```

Salida:

15-ENERO -2020

ROUND Y TRUNC en cálculos de fechas.

Nótese en el siguiente ejemplo como si restamos la fecha de mañana a la de hoy nos aparecen valores como:

```
select to_date('15-06-2020','DD-MM-YYYY')-sysdate from dual
```

El resultado variará en función de la hora a la que se lance la consulta, pero tendrá este aspecto
0,134710648148148148148148148148148148148148148148148148

La razón es que Oracle guarda también horas, minutos y segundos junto con la fecha.
Oracle siempre hace estas suposiciones:

a) Una fecha literal se le da por defecto las 12 A.M. (comienzo del día).

b) Sysdate incluye tanto la fecha como la hora.

c) La función ROUND aplicada sobre cualquier fecha obtiene las 12 A.M. del día si es antes del mediodía y 12 A.M. del día siguiente si es después del mediodía.

d) La función TRUNC por el contrario, siempre pone 12 A.M. del propio día.

Nota: Como síntesis, ROUND redondea la fecha a las 12 A.M. del día más cercano y TRUNC siempre del día actual.

Funciones de conversión y formateo TO_CHAR y TO_DATE.

Sintaxis:

TO_CHAR(fecha/n[,formato][,'parámetrosNLS]

TO_DATE(cadena[,formato][,'parámetrosNLS]

Dependiendo del cliente que estemos usando para acceder a la base de datos (SqlDeveloper, SqlPlus, Programa Java, el formato de la fecha puede cambiar. También es posible cambiar el formato por defecto de un determinado cliente.
En las últimas versiones de SqlDeveloper, el formato por defecto de la aplicación es:
DD/MM/YY

Por último, los llamados parámetros NLS es una cadena que fija la operación NLS_DATE_LANGUAGE para un idioma concreto, distinto del utilizado en la sesión de SQL. Este no debe ser utilizado normalmente, pues Oracle devolverá el día y mes en el idioma que se haya asignado a la sesión.

Todos estos parámetros corresponden a la sección del NLS (National Languaje Support). Se puede consultar en la documentación de Oracle su uso.
Las siguientes tablas muestran los caracteres que se pueden usar en las cadenas de formato. Tenemos caracteres especiales para usar en formatos numéricos y de fecha.

Element	Example	Description
9	9999	Return value with the specified number of digits with a leading space if positive. Return value with the specified number of digits with a leading minus if negative. Leading zeros are blank, except for a zero value, which returns a zero for the integer part of the fixed point number.
0	0999 9990	Return leading zeros. Return trailing zeros.
$	$9999	Return value with a leading dollar sign.
B	B9999	Return blanks for the integer part of a fixed point number when the integer part is zero (regardless of "0"s in the format model).
MI	9999MI	Return negative value with a trailing minus sign "–". Returns positive value with a trailing blank.
S	S9999 9999S	Return negative value with a leading minus sign "–". Return positive value with a leading plus sign "+". Return negative value with a trailing minus sign "–". Return positive value with a trailing plus sign "+".
PR	9999PR	Return negative value in <angle brackets>. Return positive value with a leading and trailing blank.
D	99D99	Return a decimal point (that is, a period ".") in the specified position.
G	9G999	Return a group separator in the position specified.
C	C999	Return the ISO currency symbol in the specified position.

Table 3 – 12 Number Format Elements

Element	Example	Description
L	L999	Return the local currency symbol in the specified position.
, (comma)	9,999	Return a comma in the specified position.
. (period)	99.99	Return a decimal point (that is, a period ".") int the specified position.
V	999V99	Return a value multiplied by 10^n (and if necessary, round it up), where *n* is the number of "9"s after the "V".
EEEE	9.9EEEE	Return a value using in scientific notation.
RN rn	RN	Return a value as Roman numerals in uppercase. Rerturn a value as Roman numerals in lowercase. Value can be an integer between 1 and 3999.
FM	FM90.9	Returns a value with no leading or trailing blanks.

Table 3 – 12 (continued) Number Format Elements

Element	Meaning
- / , . ; : "text"	Punctuation and quoted text is reproduced in the result.
AD A.D.	AD indicator with or without periods.
AM A.M.	Meridian indicator with or without periods.
BC B.C.	BC indicator. with or without periods.
CC SCC	Century; "S" prefixes BC dates with "-".
D	Day of week (1–7).
DAY	Name of day, padded with blanks to length of 9 characters.
DD	Day of month (1–31).
DDD	Day of year (1–366).
DY	Abbreviated name of day.
IW	Week of year (1–52 or 1–53) based on the ISO standard.
IYY IY I	Last 3, 2, or 1 digit(s) of ISO year.
IYYY	4–digit year based on the ISO standard.
HH HH12	Hour of day (1–12).
HH24	Hour of day (0–23).
J	Julian day; the number of days since January 1, 4712 BC. Number specified with 'J' must be integers.
MI	Minute (0–59).
MM	Month (01–12; JAN = 01)
MONTH	Name of month, padded with blanks to length of 9 characters.
MON	Abbreviated name of month.
RM	Roman numeral month (I–XII; JAN = I).

Table 3 – 14 Date Format Elements

Element	Meaning
Q	Quarter of year (1, 2, 3, 4; JAN–MAR = 1)
RR	Last 2 digits of year; for years in other countries.
WW	Week of year (1–53) where week 1 starts on the first day of the year and continues to the seventh day of the year.
W	Week of month (1–5) where week 1 starts on the first day of the month and ends on the seventh.
PM P.M.	Meridian indicator with and without periods.
SS	Second (0–59).
SSSSS	Seconds past midnight (0–86399).
Y YYY	Year with comma in this position.
YEAR SYEAR	Year, spelled out; "S" prefixes BC dates with "–".
YYYY SYYYY	4–digit year; "S" prefixes BC dates with "–".
YYY YY Y	Last 3, 2, or 1 digit(s) of year.

Table 3 – 14 (continued) Date Format Elements

Table 3 – 16 lists suffixes that can be added to date format elements:

Suffix	Meaning	Example Element	Example Value
TH	Ordinal Number	DDTH	4TH
SP	Spelled Number	DDSP	FOUR
SPTH or THSP	Spelled, ordinal number	DDSPTH	FOURTH

Table 3 – 16 Date Format Element Suffixes

Ejemplos

```
SELECT NOMBRE,TO_CHAR(FECHANACIMIENTO,'DD-MON-YYYY')FECHANACIMIENTO
FROM ALUMNO
```

NOMBRE	FECHANACIMIENTO
Bargueño Del Alamo, David	31-MAR-1985
Blázquez Rodríguez, Alberto Santiago	07-FEB-1989
Calvo Pérez, Marta	01-OCT-1991
----------	----------

Listado de alumnos nacidos en Mayo:

```
SELECT NOMBRE,TO_CHAR(FECHANACIMIENTO,'DD-MON-YYYY')NACIDOSENMAYO
FROM ALUMNO
WHERE TO_CHAR(FECHANACIMIENTO,'MM')='05'
```

La tabla de la página siguiente muestra algunos ejemplos de uso de las cadenas de formato aplicadas sobre valores numéricos.

number	*'fmt'*	*Result*
-1234567890	9999999999S	'1234567890-'
0	99.99	' 0.00'
+0.1	99.99	' .10'
-0.2	99.99	' -.20'
0	90.99	' 0.00'
+0.1	90.99	' .10'
-0.2	90.99	' -0.20'
0	9999	' 0'
1	9999	' 1'
0	B9999	' '
1	B9999	' 1'
0	B90.99	' '
+123.456	999.999	' 123.456'
-123.456	999.999	'-123.456'
+123.456	FM999.009	'123.456'
+123.456	9.9EEEE	' 1.2E+02'
+1E+123	9.9EEEE	' 1.0E+123'
+123.456	FM9.9EEEE	'1.23E+02'
+123.45	FM999.009	'123.45'
+123.0	FM999.009	'123.00'
+123.45	L999.99	' $123.45'
+123.45	FML99.99	'$123.45'
+1234567890	9999999999S	'1234567890+'

Table 3 – 17 Results of Example Number Conversions

ANEXO III. FUNCIONES NUMERICAS. SQL

FUNCIÓN	SINTAXIS	COMENTARIO
ABS	ABS(n)	Returns the absolute value of *n*
ACOS	ACOS(n)	Returns the arc cosine of *n*. Inputs are in the range of –1 to 1, and outputs are in the range of 0 to pi and are expressed in radians.
ASIN	ASIN(n)	Returns the arc sine of *n*. Inputs are in the range of –1 to 1, and outputs are in the range of –pi/2 to pi/2 and are expressed in radians.
ATAN	ATAN(n)	Returns the arc tangent of *n*. Inputs are in an unbounded range, and outputs are in the range of –pi/2 to pi/2 and are expressed in radians.
ATAN2	ATAN2(n, m)	Returns the arc tangent of *n* and *m*. Inputs are in an unbounded range, and outputs are in the range of –pi to pi, depending on the signs of x and y, and are expressed in radians. Atan2(x,y) is the same as atan2(x/y)
CEIL	CEIL(n)	Returns smallest integer greater than or equal to n.
COS	COS(n)	Returns the cosine of *n* (an angle expressed in radians).
COSH	COSH(n)	Returns the hyperbolic cosine of n.
EXP	EXP(n)	Returns *e* raised to the nth power; e = 2.71828183 ...
FLOOR	FLOOR(n)	Returns largest integer equal to or less than n.
LN	LN(n)	Returns the natural logarithm of n, where *n* is greater than 0.
LOG	LOG(m,n)	Returns the logarithm, base m, of n. The base *m* can be any positive number other than 0 or 1 and *n* can be any positive number.
MOD	MOD(m,n)	Returns remainder of *m* divided by n. Returns *m* if *n* is 0.
POWER	POWER(m, n)	Returns *m* raised to the nth power. The base *m* and the exponent *n* can be any numbers, but if *m* is negative, *n* must be an integer.
ROUND	ROUND(n[,m])	Returns *n* rounded to *m* places right of the decimal point; if *m* is omitted, to 0 places. *m* can be negative to round off digits left of the decimal point. *m* must be an integer.
SIGN	SIGN(n)	If n<0, the function returns –1; if n=0, the function returns 0; if n>0, the function returns 1.
SIN	SIN(n)	Returns the sine of *n* (an angle expressed in radians).
SINH	SINH(n)	Returns the hyperbolic sine of n.
SQRT	SQRT(n)	Returns square root of n. The value *n* cannot be negative. SQRT returns a "real" result.
TAN	TAN(n)	Returns the tangent of *n* (an angle expressed in radians).
TANH	TANH(n)	Returns the hyperbolic tangent of n.
TRUNC	TRUNC(n[,m])	Returns *n* truncated to *m* decimal places; if *m* is omitted, to 0 places. *m* can be negative to truncate (make zero) *m* digits left of the decimal point.

Extensiones de la cláusula group by mediante rollup, cube y conjuntos de agrupaciones.

Se comentarán estas extensiones a través de ejemplos obtenidos de la documentación de la versión 10 de Oracle.

Usaremos como esquema de referencia el esquema SH que viene preinstalado pero que está bloqueado. Para desbloquear la cuenta utilizaríamos la siguiente orden (actuando como DBA):

```
ALTER USER SH ACCOUNT UNLOCK
```

Después le cambiaríamos la contraseña de acceso:

```
ALTER USER SH IDENTIFIED BY SH (por ej.)
```

El núcleo central de los ejemplos se sitúa en torno a la tabla sales (ventas) y las dimensiones principales sobre las que queremos hacer un análisis a partir de los datos almacenados usando las nuevas extensiones de la claúsula group by son:

- CHANNELS(canal de distribución) que dispone entre otros de : Venta directa, Internet, tele venta, etc.
- COUNTRIES: (Países a los que se distribuyen las ventas). En los ejemplos se trabaja con el código ISO del país.
- TIMES: Tiene que ver con las fechas en las que se realizan las ventas. La tabla times tiene una entrada por cada día del año.

Veamos a continuación las extensiones con las que vamos a trabajar a través de algunos ejemplos:

Extensión Rollup

Operación útil en el cálculo de acumulados parciales en función de las dimensiones con las que trabajemos:

```
SELECT channels.channel_desc, countries.country_iso_code,
       TO_CHAR(SUM(amount_sold), '9,999,999,999') SALES$
FROM sales, customers, times, channels, countries
WHERE sales.time_id=times.time_id
AND sales.cust_id=customers.cust_id
AND sales.channel_id= channels.channel_id
AND channels.channel_desc IN ('Direct Sales', 'Internet')
AND times.calendar_month_desc='2000-09'
AND customers.country_id=countries.country_id
AND countries.country_iso_code IN ('US','FR')
GROUP BY ROLLUP(channels.channel_desc, countries.country_iso_code);
```

Trabajamos con dos dimensiones para la obtención de los parciales que nos interesan. En el ejemplo hemos trabajado con 2 dimensiones: Canal de distribución y país.

Una sentencia ROLLUP que trabaja con n dimensiones produce n+1 niveles de resultados parciales. En nuestro caso son como muestra el resultado los siguientes:

CHANNEL_DESC	CO	SALES$
Internet	FR	9, 597
Internet	US	124, 224
Internet	(correspondería a los dos países)	133,821
Direct Sales	FR	61,202
Direct Sales	US	638,201
Direct Sales	(correspondería a los dos países)	699,403
----------	---------- total ventas	833,224

1. Parcial por (Canal de distribución, país)
2. Parcial por (Canal de distribución, todos los países)
3. Total general (Todos los canales, todos los países).

Todo ello en una única sentencia.

Extensión Cube

Toma un conjunto de columnas de agrupamiento (las dimensiones que nos interesan) y crea parciales para todas las posibles combinaciones entre ellas. Si tenemos n columnas en el agrupamiento mediante la operación cube se obtendrán 2 elevado a n niveles de parciales. Si tomamos la sentencia anterior y hacemos el agrupamiento mediante la operación cube:

```
SELECT channels.channel_desc, countries.country_iso_code,
  TO_CHAR(SUM(amount_sold), '9,999,999,999') SALES$
FROM sales, customers, times, channels, countries
WHERE sales.time_id=times.time_id
AND sales.cust_id=customers.cust_id
AND sales.channel_id= channels.channel_id
AND channels.channel_desc IN ('Direct Sales', 'Internet')
AND times.calendar_month_desc='2000-09'
AND customers.country_id=countries.country_id
AND countries.country_iso_code IN ('US','FR')
GROUP BY CUBE(channels.channel_desc, countries.country_iso_code);
```

Cube genera todas las posibles combinaciones de totales. El resultado sería este:

CANAL	PAIS	SALES$
		833,224
	FR	70,799
	US	762,425
Internet		133,821
Internet	FR	9,597
Internet	US	124,224
Direct Sales		699,403
Direct Sales	FR	61,202
Direct Sales	US	638,201

Funciones AGRUPACIÓN

Dos retos se plantean con el uso de **ROLLUP y CUBE**. En primer lugar, ¿cómo se puede determinar mediante programación qué filas de resultados corresponden a resultados parciales, y cómo encontrar el nivel exacto de acumulado para un resultado parcial?

A menudo tendremos que utilizar los resultados parciales en cálculos, como por ejemplo: el tanto por ciento sobre el total, por lo que necesitamos una manera fácil de determinar qué filas son los parciales. En segundo lugar, ¿qué sucede si los resultados de la consulta contienen tanto valores NULL como los valores "NULL" creados a partir del uso de ROLLUP o CUBE? ¿Cómo se puede diferenciar entre los dos?
Para ello utilizaremos la función **grouping**.

Función GROUPING

GROUPING maneja estos problemas. Usando una sola columna como argumento, **GROUPING** devuelve 1 cuando se encuentra un valor NULL creado por una operación ROLLUP o CUBE. Es decir, si el NULL indica que la fila es un parcial, **GROUPING** devuelve un 1. Cualquier otro tipo de valor, incluyendo un NULL almacenado, devuelve un 0.

GROUPING aparece en la parte de lista de selección de una orden SELECT.

Su forma es la siguiente:

```
SELECT ... [GROUPING  (dimension_column) ...] ...
GROUP BY ... (CUBE | ROLLUP | AGRUPACIÓN SETS) (dimension_column).
```

El uso de la función **GROUPING** junto con DECODE nos permite obtener un listado más detallado de los totales. A continuación, se muestra la sentencia select junto con el resultado obtenido:

```
SELECT DECODE (GROUPING (channels.channel_desc),1,'Todos los
Canales',channels.channel_desc)Canal,
       DECODE(GROUPING (countries.country_iso_code),1,'Los dos
paises',countries.country_iso_code)Pais,
  TO_CHAR(SUM(amount_sold), '9,999,999,999') SALES$
FROM sales, customers, times, channels, countries
WHERE sales.time_id=times.time_id
AND sales.cust_id=customers.cust_id
AND sales.channel_id= channels.channel_id
AND channels.channel_desc IN ('Direct Sales', 'Internet')
AND times.calendar_month_desc='2000-09'
AND customers.country_id=countries.country_id
AND countries.country_iso_code IN ('US','FR')
GROUP BY CUBE(channels.channel_desc, countries.country_iso_code);
```

CANAL	PAIS	SALES$
Todos los Canales	Los dos países	833,224
Todos los Canales	FR	70,799
Todos los Canales	US	762,425
Internet	Los dos paises	133,821
Internet	FR	9,597
Internet	US	124,224
Direct Sales	Los dos paises	699,403
Direct Sales	FR	61,202
Direct Sales	US	638,201

Otro uso de grouping permite la ordenación de filas de parciales, y el filtrado de ellas.

Ejemplo:

```
SELECT channel_desc, calendar_month_desc, country_iso_code, TO_CHAR(
SUM(amount_sold), '9,999,999,999') SALES$, GROUPING(channel_desc) CH,
GROUPING
  (calendar_month_desc)  MO, GROUPING(country_iso_code) CO
FROM sales, customers, times, channels, countries
WHERE sales.time_id=times.time_id AND sales.cust_id=customers.cust_id
AND customers.country_id = countries.country_id
AND sales.channel_id= channels.channel_id
AND channels.channel_desc IN ('Direct Sales', 'Internet')
AND times.calendar_month_desc IN ('2000-09', '2000-10')
AND country_iso_code IN ('GB', 'US')
GROUP BY CUBE(channel_desc, calendar_month_desc, country_iso_code)
HAVING (GROUPING(channel_desc)=1 AND GROUPING(calendar_month_desc)= 1
AND GROUPING(country_iso_code)=1) OR (GROUPING(channel_desc)=1
AND GROUPING (calendar_month_desc)= 1) OR
(GROUPING(country_iso_code)=1
AND GROUPING(calendar_month_desc)= 1);
```

Esto sacaría las ventas totales de Estados Unidos y Gran Bretaña independientemente del canal de distribución y el mes y por otro lado sacaría las ventas totales por canal de

distribución independientemente del país y del mes y finalmente un total de ventas. Se han filtrado todas las posibles combinaciones de cube en la cláusula having.
He aquí el resultado de la consulta:

CHANNEL_DESC	C	CO	SALES$	CH	MO	CO
		US	1,581,775	1	1	0
		GB	208,257	1	1	0
Direct Sales			1,497,646	0	1	1
Internet			292,387	0	1	1
			1,790,032	1	1	1

El poder generar mediante la función grouping las columnas (CH, MO, CO), es cómodo para después tratarlo mediante un programa de aplicación, ya que donde aparezca un uno sabemos que son totales parciales de los que estamos interesados.

Expresión GROUPING SETS

Podemos seleccionar específicamente el conjunto de grupos que queremos usar dentro de una cláusula GROUP BY mediante la expresión GROUPING SETS. Esto permite precisar la especificación a través de las múltiples dimensiones sin computar el cubo entero (todas las combinaciones).
Por ejemplo:

```
SELECT channel_desc, calendar_month_desc, country_iso_code,
       TO_CHAR(SUM(amount_sold), '9,999,999,999') SALES$
FROM sales, customers, times, channels, countries
WHERE sales.time_id=times.time_id
AND sales.cust_id=customers.cust_id
AND sales.channel_id= channels.channel_id
AND channels.channel_desc IN ('Direct Sales', 'Internet')
AND times.calendar_month_desc IN ('2000-09', '2000-10')
AND country_iso_code IN ('GB', 'US')
GROUP BY GROUPING SETS(
      (channel_desc, calendar_month_desc, country_iso_code),
      (channel_desc, country_iso_code),
      (calendar_month_desc, country_iso_code));
```

Esta sentencia produce parciales sobre tres agrupamientos definidos:

- (channel_desc, calendar_month_desc, country_iso_code)
- (channel_desc, country_iso_code)
- (calendar_month_desc, country_iso_code)

El resultado de la sentencia sería este:

CHANNEL_DESC	CALENDAR	CO	SALES$
Internet	2000-09	GB	228,241
Direct Sales	2000-09	GB	1,217,808
Internet	2000-09	US	228,241
Direct Sales	2000-09	US	1,217,808
Internet	2000-10	GB	239,236
Direct Sales	2000-10	GB	1,225,584
Internet	2000-10	US	239,236
Direct Sales	2000-10	US	1,225,584
	2000-09	GB	1,446,049
	2000-09	US	1,446,049
	2000-10	GB	1,464,821
	2000-10	US	1,464,821
Direct Sales		GB	2,443,392
Internet		GB	467,478
Direct Sales		US	2,443,392
Internet		US	467,478

Elegir los agrupamientos que nos interesan aumenta el rendimiento, ya que, el uso de Cube implica un procesamiento pesado.

Análisis e informe: Estudio de algunas funciones analíticas

Orientadas a la mejora del procesamiento analítico a través de SQL.
Se pueden clasificar en los siguientes grupos:

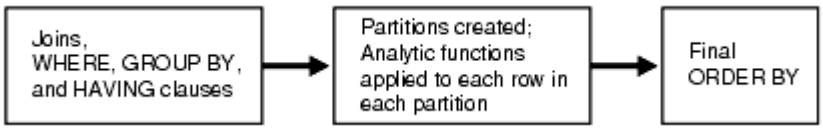
Tabla 1 Funciones analíticas y sus usos

Tipo	Usado Para
Ranking	Obtener clasificaciones (por ej: tres productos más vendidos), percentiles, y n-cubos de los valores en un conjunto de resultados.
Windowing Ventanas	Cálculo de acumulados en conjuntos que se desplazan en función de cómo se elige la ventana. Funciona con las siguientes funciones: SUM, AVG, MIN, MAX, COUNT, VARIANCE, STDDEV, FIRST_VALUE, LAST_VALUE, y nuevas funciones estadísticas. Hay que tener en cuenta que la palabra clave DISTINCT no es compatible con ventanas, excepto para las funciones MAX y MIN.
Reporting Presentación de informes	Cálculo de las cuotas, por ejemplo, cuota de mercado. Funciona con las siguientes funciones: SUM, AVG, MIN, MAX, COUNT (con / sin DISTINCT), VARIANCE, STDDEV, RATIO_TO_REPORT, y nuevas funciones estadísticas. Hay que tener en cuenta que la palabra clave DISTINCT puede ser utilizado en las funciones de informes siempre que distinct se pueda usar en las funciones anteriores trabajando en modo cálculo de acumulados
LAG/LEAD	Encontrar un valor en una fila teniendo en cuenta un desplazamiento especificado, a partir de la fila actual. Por ejemplo: supuesto que tenemos una tabla de empleados con sus sueldos, puedo sacar en una consulta, sin hacer un join, el sueldo de un empleado, y el del empleado con el sueldo inmediatamente anterior o posterior a él.
FIRST/LAST PRIMERA/ÚLTIMA	Primer o último valor en un grupo ordenado.
Regresión lineal	Cálculo de regresión lineal y otras estadísticas (pendiente, interceptar, y así sucesivamente).
Inversa Percentil	El valor de un conjunto de datos que corresponde a un percentil determinado.
Ranking hipotético y Distribución	El ranking o percentil que tendría una hipotética fila si se inserta en un conjunto de datos especificados.

Para realizar estas operaciones, las funciones analíticas añaden varios elementos nuevos en el procesamiento de SQL. Estos elementos construidos sobre el lenguaje SQL ya existente permien expresiones de cálculo flexible y potente.

Con sólo unas pocas excepciones, las funciones analíticas tienen estos nuevos elementos. El flujo de transformación está representado en Figura 21-1.

Figura 1 Orden de procesamiento



Descripción de la "Figura 1 Procesamiento de Orden"

Lo conceptos esenciales utilizados en las funciones analíticas son:

- **Orden de procesamiento**

  El procesamiento de consultas mediante funciones de análisis se lleva a cabo en tres etapas.

  En **primer lugar**, se realizan todas las combinaciones, y cláusulas WHERE, GROUP BY y HAVING.

  En **segundo lugar**, el resultado obtenido en el primer paso, estará disponible para que las funciones analíticas operen sobre él.

  En **tercer lugar**, si la consulta tiene una cláusula ORDER BY en su extremo, el ORDER BY se procesa para permitir la salida ordenada.

  La orden de procesamiento se muestra en la Figura 1.

- **Particiones del conjunto de resultados**.

  Las funciones analíticas permiten a los usuarios dividir el resultado de la consulta obtenido en el primer paso en grupos de filas llamadas **particiones**.

  A lo largo de este capítulo, el término se refiere a las particiones relacionadas con las funciones analíticas. **Las particiones se crean después de los grupos definidos mediante las cláusulas GROUP BY**, de modo que estén disponibles para los resultados globales, tales como sumas y promedios. Las divisiones de partición podrán basarse en las columnas que desee o expresiones. Un conjunto de resultados de consulta puede dividirse en una sola partición que contiene todas las filas, un puñado de grandes particiones, o muchas particiones pequeñas que tienen un par de filas cada uno.

- **Ventana**

  Para cada fila de una partición, puede definir una ventana deslizante de los datos. En esta ventana se determina el rango de filas utilizados para realizar los cálculos de la fila actual. Los tamaños de ventanas pueden basarse en un número físico de filas o un intervalo de lógica, como el tiempo. La ventana tiene una fila de partida y una fila

final. Dependiendo de su definición, la ventana puede moverse a uno o ambos extremos.

Por ejemplo, en una ventana que se define para un cálculo de suma acumulada de cada fila con las anteriores, habría una fila de inicio en la primera fila de su partición, y su fila de final se deslizaría desde el punto de partida hasta la última fila de la partición.

El siguiente ejemplo, obtenido de la documentación oficial de Oracle expresa este uso. Calculamos el salario y el salario acumulado en función del jefe que tengan los empleados. La palabra reservda UNBOUNDED PRECEDING significa que la ventana tendrá en cuenta todas las filas anteriores dentro de la misma partición.

```
SELECT manager_id, last_name, salary,
   SUM(salary) OVER (PARTITION BY manager_id ORDER BY salary
   rows UNBOUNDED PRECEDING) l_csum
         FROM employees;
```

Manager_id	Last_Name	Salary	Salary_Acumulado
100	Hartstein	13000	93900
100	Partners	13500	107400
100	Russell	14000	121400
100	De Haan	17000	138400
100	Kochhar	17000	155400
101	Whalen	4400	4400
101	Mavris	6500	10900
101	Baer	10000	20900
101	Higgins	12008	32908
101	Greenberg	12008	44916
102	Hunold	9000	9000
103	Lorentz	4200	4200
103	Austin	4800	9000
103	Pataballa	4800	13800
103	Ernst	6000	19800

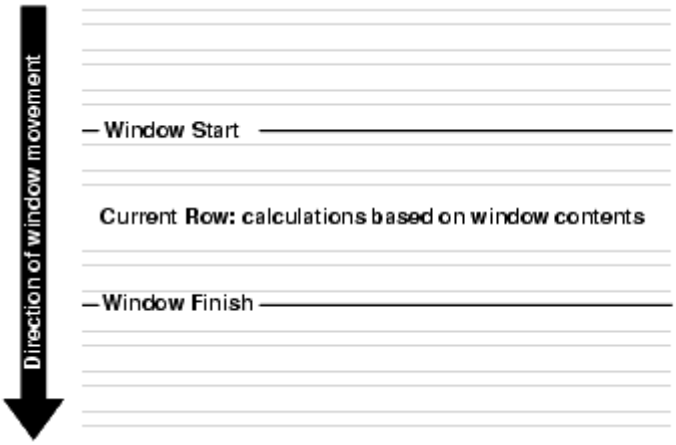
Una ventana se puede ajustar tan grande como todas las filas de una partición o una ventana deslizante de una sola fila dentro de una partición. Cuando una ventana se encuentra cerca de la frontera, la función devuelve los resultados para sólo las filas disponibles, en lugar de un aviso de que los resultados no son lo que usted desea.

Al utilizar las funciones de la ventana, la fila actual se incluye en los cálculos, por lo que sólo debería especificar (*n*-1) Cuando se trata de *n* artículos.

- **La fila actual**

  Cada cálculo realizado con una función analítica se basa en una fila actual dentro de una partición. De la fila actual sirve como punto de referencia para determinar el inicio y el final de la ventana. Por ejemplo, cálculo del promedio se podría definir con una ventana que contiene la fila actual, las seis filas anteriores, y las siguientes seis filas. Esto crearía una ventana deslizante de 13 filas, como se muestra en Figura 2.

Figura 2 Ejemplo de ventana deslizante



Veamos ejemplos de uso de algunas de las funciones más interesantes:

Funciones de clasificación (Ranking)

Una función de clasificación calcula el orden de un registro en comparación con otros registros dentro del conjunto de datos, basándonos en los valores de un conjunto de medidas.

Ejemplos de algunas de ellas:

Las funciones RANK and DENSE_RANK permiten clasificar los elementos de un grupo, por ejemplo, la búsqueda de los tres principales productos que se vendieron en California el año pasado. Hay dos funciones que realizan el ranking. Esta es su sintaxis:

RANK () OVER ([query_partition_clause] order_by_clause)
DENSE_RANK () OVER ([query_partition_clause] order_by_clause)

La diferencia entre RANK y DENSE_RANK es que DENSE_RANK no deja huecos en la secuencia de clasificación cuando hay empates. Es decir, si estuviéramos obteniendo la clasificación de una competición usando DENSE_RANK y hubiera tres personas empatadas para el segundo lugar, diríamos que los tres ocuparon el segundo lugar y que la siguiente persona entró en tercero. La función RANK diría también que tres personas entraron en segundo lugar, pero la tercera persona lo haría en quinto.

Cuestiones a tener en cuenta con las funciones de ranking:

- El orden de ordenación es ascendente, a menos que lo queramos cambiar.
- Las expresiones en la cláusula opcional PARTITION BY divide el conjunto de resultados en grupos dentro de los cuales las funciones de ranking operan. Es decir, las funciones se reinician siempre que el grupo cambie.
- Si no existe la cláusula PARTITION BY, las clasificaciones se calculan sobre todo el conjunto de resultados de la consulta.

- La cláusula ORDER BY especifica las medidas sobre las que se establece la clasificación y define el orden en el que las filas son ordenadas en cada grupo o partición. Una vez que los datos son ordenados en cada partición, se asigna el orden de clasificación de cada fila, comenzando en uno.
- La cláusula NULLS FIRST|NULLS LAST indican la posición de los nulos en la secuencia ordenada, o primero o últimos.
- Si la cláusula NULLS FIRST|NULLS LAST se omite, los órdenes de los valores nulos dependen de los argumentos ASC o DESC. Los valores nulos se consideran mayores que cualquier otro valor, por tanto, si la ordenación es ASC, entonces los nulos aparecerán los últimos, en otro caso aparecerán primero.

Ejemplo:

La cláusula query_partition no se usa, así que el ranking se hace en función de la cláusula order by de la función analítica.

```
SELECT channel_desc, TO_CHAR(SUM(amount_sold), '9,999,999,999')
SALES$,
   RANK() OVER (ORDER BY SUM(amount_sold)) AS default_rank,
   RANK() OVER (ORDER BY SUM(amount_sold) DESC NULLS LAST) AS
custom_rank
FROM sales, products, customers, times, channels, countries
WHERE sales.prod_id=products.prod_id
AND sales.cust_id=customers.cust_id
AND customers.country_id = countries.country_id
AND sales.time_id=times.time_id
AND sales.channel_id=channels.channel_id
AND times.calendar_month_desc IN ('2000-09', '2000-10')
AND country_iso_code='US'
GROUP BY channel_desc;
```

CHANNEL_DESC	SALES$	DEFAULT_RANK	CUSTOM_RANK
Direct Sales	1,320,497	3	1
Partners	800,871	2	2
Internet	261,278	1	3

El ranking se puede basar en más de una expresión, en este caso lo hacemos sobre la columna ventas totales y unidades vendidas. Si hay empate a ventas utilizaríamos la segunda columna para deshacer empates.

```
SELECT channel_desc, calendar_month_desc,
TO_CHAR(TRUNC(SUM(amount_sold),-5),
  '9,999,999,999') SALES$, TO_CHAR(SUM(quantity_sold),
'9,999,999,999')
  SALES_Count, RANK() OVER (ORDER BY TRUNC(SUM(amount_sold), -5)
  DESC, SUM(quantity_sold) DESC) AS col_rank
FROM sales, products, customers, times, channels
WHERE sales.prod_id=products.prod_id
AND sales.cust_id=customers.cust_id
AND sales.time_id=times.time_id
AND sales.channel_id=channels.channel_id
AND times.calendar_month_desc IN ('2000-09', '2000-10')
AND channels.channel_desc<>'Tele Sales'
GROUP BY channel_desc, calendar_month_desc;
```

CHANNEL_DESC	CALENDAR_MONTH_DESC	SALES$	SALES_COUNT	COL_RANK
Direct Sales	2000-10	1,200,000	12,584	1
Direct Sales	2000-09	1,200,000	11,995	2
Partners	2000-10	600,000	7,508	3
Partners	2000-09	600,000	6,165	4
Internet	2000-09	200,000	1,887	5
Internet	2000-10	200,000	1,450	6

Diferencia de RANK and DENSE_RANK

A través de este ejemplo se vería la diferencia entre las dos funciones.

```
SELECT channel_desc, calendar_month_desc,
   TO_CHAR(TRUNC(SUM(amount_sold),-5), '9,999,999,999') SALES$,
      RANK() OVER (ORDER BY TRUNC(SUM(amount_sold),-5) DESC) AS RANK,
DENSE_RANK() OVER (ORDER BY TRUNC(SUM(amount_sold),-5) DESC) AS DENSE_RANK
FROM sales, products, customers, times, channels
WHERE sales.prod_id=products.prod_id
AND sales.cust_id=customers.cust_id
AND sales.time_id=times.time_id
AND sales.channel_id=channels.channel_id
AND times.calendar_month_desc IN ('2000-09', '2000-10')
AND channels.channel_desc<>'Tele Sales'
GROUP BY channel_desc, calendar_month_desc;
```

CHANNEL_DESC	CALENDAR_MONTH_DESC	SALES$	RANK	DENSE_RANK
Direct Sales	2000-10	1,200,000	1	1
Direct Sales	2000-09	1,200,000	1	1
Partners	2000-10	600,000	3	2
Partners	2000-09	600,000	3	2
Internet	2000-10	200,000	5	3
Internet	2000-09	200,000	5	3

Ranking por grupo.

La función de clasificación se puede hacer para operar dentro de los grupos, es decir, el rango se restablece cada vez que cambia de grupo.
Esto se logra con la cláusula **PARTITION BY**. Las expresiones de grupo de la cláusula PARTITION BY dividen el conjunto de datos en grupos dentro de los cuales la función de clasificación opera.
Por ejemplo, para clasificar los productos dentro de cada canal por sus ventas en dólares, se podría emitir la siguiente declaración.

```
SELECT channel_desc, calendar_month_desc, TO_CHAR(SUM(amount_sold),
 '9,999,999,999') SALES$, RANK() OVER (PARTITION BY channel_desc
  ORDER BY SUM(amount_sold) DESC) AS RANK_BY_CHANNEL
FROM sales, products, customers, times, channels
WHERE sales.prod_id=products.prod_id
AND sales.cust_id=customers.cust_id
AND sales.time_id=times.time_id
AND sales.channel_id=channels.channel_id
AND times.calendar_month_desc IN ('2000-08', '2000-09', '2000-10',
'2000-11')
AND channels.channel_desc IN ('Direct Sales', 'Internet')
GROUP BY channel_desc, calendar_month_desc;
```

El resultado sería:

CHANNEL_DESC	CALENDAR_MONTH_DESC	SALES$	RANK_BY_CHANNEL
Direct Sales	2000-08	1,236,104	1
Direct Sales	2000-10	1,225,584	2
Direct Sales	2000-09	1,217,808	3
Direct Sales	2000-11	1,115,239	4
Internet	2000-11	284,742	1
Internet	2000-10	239,236	2
Internet	2000-09	228,241	3
Internet	2000-08	215,107	4

Estamos agrupando por canal y mes. La clasificación se reinicia cada vez que hay un cambio de canal (cláusula PARTITION BY channel_desc).

Un bloque de consulta puede contener más de una función de clasificación, cada una particionando los datos en distintos grupos (es decir, restablecer en particiones distintas). Los grupos pueden ser mutuamente excluyentes.

El siguiente ejemplo muestra esta circunstancia:

```
SELECT channel_desc, calendar_month_desc, TO_CHAR(SUM(amount_sold),
  '9,999,999,999') SALES$, RANK() OVER (PARTITION BY calendar_month_desc
  ORDER BY SUM(amount_sold) DESC) AS RANK_WITHIN_MONTH, RANK() OVER
(PARTITION
 BY channel_desc ORDER BY SUM(amount_sold) DESC) AS RANK_WITHIN_CHANNEL
FROM sales, products, customers, times, channels, countries
WHERE sales.prod_id=products.prod_id
AND sales.cust_id=customers.cust_id
AND customers.country_id = countries.country_id
AND sales.time_id=times.time_id
AND sales.channel_id=channels.channel_id
AND times.calendar_month_desc IN ('2000-08', '2000-09', '2000-10', '2000-11')
AND channels.channel_desc IN ('Direct Sales', 'Internet')
GROUP BY channel_desc, calendar_month_desc;
```

CHANNEL_DESC	CALENDAR_MONTH_DESC	SALES$	RANK_WITHIN_MONTH	RANK_WITHIN_CHANNEL
Direct Sales	2000-08	1,236,104	1	1
Internet	2000-08	215,107	2	4
Direct Sales	2000-09	1,217,808	1	3
Internet	2000-09	228,241	2	3
Direct Sales	2000-10	1,225,584	1	2
Internet	2000-10	239,236	2	2
Direct Sales	2000-11	1,115,239	1	4
Internet	2000-11	284,742	2	1

Función ROW_NUMBER().

La función Row_number asigna un número único a cada fila para la que se aplica (cada fila en una partición, o cada fila devuelta por la consulta) en la secuencia ordenada de filas especificada por la cláusula order by, comenzando por 1. Si se usa Row_number junto con la cláusula PARTITION BY la secuencia se reiniciará cada vez que haya un cambio de partición.

Tiene la siguiente sintaxis:

```
Row_number () OVER ([query_partition_clause] order_by_clause).
```

Ejemplo:

```
SELECT channel_desc, calendar_month_desc,
   TO_CHAR(TRUNC(SUM(amount_sold), -5), '9,999,999,999') SALES$,
   ROW_NUMBER() OVER (ORDER BY TRUNC(SUM(amount_sold), -6) DESC) AS
ROW_NUMBER
FROM sales, products, customers, times, channels
WHERE sales.prod_id=products.prod_id AND
sales.cust_id=customers.cust_id
  AND sales.time_id=times.time_id AND
sales.channel_id=channels.channel_id
  AND times.calendar_month_desc IN ('2001-09', '2001-10')
GROUP BY channel_desc, calendar_month_desc;
```

Resultado:

CHANNEL_DESC	CALENDAR_MONTH_DESC	SALES$	ROW_NUMBER
Direct Sales	2001-10	1,000,000	1
Direct Sales	2001-09	1,100,000	2
Internet	2001-09	500,000	3
Partners	2001-09	600,000	4
Partners	2001-10	600,000	5
Internet	2001-10	700,000	6

Informes mediante funciones de grupo

Después de que una consulta haya sido procesada, se pueden calcular los valores asociados a funciones de grupo, como pueden ser: el número de filas resultantes o un valor medio de una columna, pueden calcularse fácilmente dentro de una partición y ponerse a disposición de otras funciones orientadas a informes.

Las funciones de grupo para informes devuelven el valor mismo para cada fila de una partición. Su comportamiento con respecto a la NULL es lo mismo que las funciones de grupo de SQL.

La sintaxis es:

(SUM | AVG | MAX | MIN | COUNT | STDDEV | VARIANZA ...)
([ALL | DISTINCT] (*valor de expression1* | *))
OVER ([PARTITION BY *valor de expression2*[,...]])

Ejemplo :

```
SELECT SUBSTR(p.prod_category,1,8) AS prod_category,
co.country_region,
 SUM(amount_sold) AS sales,
MAX(SUM(amount_sold)) OVER (PARTITION BY prod_category) AS
MAX_REG_SALES
FROM sales s, customers c, countries co, products p
WHERE s.cust_id=c.cust_id
AND c.country_id=co.country_id
AND s.prod_id =p.prod_id
AND s.time_id = TO_DATE('11-OCT-2001','DD-MON-YYYY')
GROUP BY prod_category, country_region
```

El resultado de esta consulta daría:

PROD_CATEGORY	COUNTRY_REGION	SALES	MAX_REG_SALES
Hardware	Americas	925,93	925,93
Peripher	Europe	4290,38	4290,38
Peripher	Asia	2616,51	4290,38
Peripher	Oceania	940,43	4290,38
Peripher	Americas	3084,48	4290,38
Software	Oceania	890,25	4445,7
Software	Europe	3288,83	4445,7
Software	Asia	1408,19	4445,7
Software	Americas	4445,7	4445,7

Que representa las ventas totales agrupadas por categoría y región junto con el máximo por categoría.
El procesamiento analítico nos permite disponer de la función de grupo sum(amount_sold) para ser usada dentro de la misma consulta, en el cálculo del máximo por categoría haciendo uso de la cláusUla PARTITON BY:

```
(MAX(SUM(amount_sold)) OVER (PARTITION BY prod_category)
```

La diferencia con la concepción de esta misma consulta que hemos tenido hasta ahora es que disponemos en la misma select del máximo por grupo, cosa que tendríamos que haber calculado antes en una subconsulta. Tendríamos que haber realizado la consulta en varías partes:

Consulta 1: Ventas por categoría y región.

```
SELECT SUBSTR(p.prod_category,1,8) AS prod_category,
co.country_region,
 SUM(amount_sold) AS sales
FROM sales s, customers c, countries co, products p
WHERE s.cust_id=c.cust_id
AND c.country_id=co.country_id
AND s.prod_id =p.prod_id
AND s.time_id = TO_DATE('11-OCT-2001','DD-MON-YYYY')
GROUP BY prod_category, country_region
ORDER BY prod_category, country_region
```

Obtendríamos el siguiente resultado parcial:

PROD_CATEGORY	COUNTRY_REGION	SALES
Electron	Americas	581,92
Hardware	Americas	925,93
Peripher	Americas	3084,48
Peripher	Asia	2616,51
Peripher	Europe	4290,38
Peripher	Oceania	940,43
Software	Americas	4445,7
Software	Asia	1408,19
Software	Europe	3288,83
Software	Oceania	890,25

Consulta 2: Máximo de vetas por categoría dentro de cada región

```
SELECT prod_category,MAX(SALES)MAX_REG_SALES
FROM(
SELECT SUBSTR(p.prod_category,1,8) AS prod_category,
co.country_region,
SUM(amount_sold) AS sales
FROM sales s, customers c, countries co, products p
WHERE s.cust_id=c.cust_id
AND c.country_id=co.country_id
AND s.prod_id =p.prod_id
AND s.time_id = TO_DATE('11-OCT-2001','DD-MON-YYYY')
 GROUP BY prod_category, country_region)
 GROUP BY PROD_CATEGORY
```

PROD_CATEGORY	MAX_REG_SALES
Hardware	925,93
Software	4445,7
Electron	581,92
Peripher	4290,38

Finalmente habría que unir estos dos resultados parciales mediante el uso de:
Select from (cosulta1) t_alias2, (consulta2)t_alias2

```
SELECT A.prod_category,country_region,sales,MAX_REG_SALES
FROM (
SELECT SUBSTR(p.prod_category,1,8) AS prod_category,
co.country_region,
 SUM(amount_sold) AS sales
FROM sales s, customers c, countries co, products p
WHERE s.cust_id=c.cust_id
AND c.country_id=co.country_id
AND s.prod_id =p.prod_id
AND s.time_id = TO_DATE('11-OCT-2001','DD-MON-YYYY')
GROUP BY prod_category, country_region)A,
(
SELECT prod_category,MAX(SALES)MAX_REG_SALES
FROM(
SELECT SUBSTR(p.prod_category,1,8) AS prod_category,
co.country_region,
SUM(amount_sold) AS sales
FROM sales s, customers c, countries co, products p
WHERE s.cust_id=c.cust_id
AND c.country_id=co.country_id
AND s.prod_id =p.prod_id
AND s.time_id = TO_DATE('11-OCT-2001','DD-MON-YYYY')
 GROUP BY prod_category, country_region)
 GROUP BY PROD_CATEGORY)B
 WHERE A.prod_category=B.prod_category
ORDER BY prod_category, country_region
```

Como podemos ver esta construcción es más compleja que la primera, que hacía uso del procesamiento analítico.

Utilizando esto podemos obtener la región que obtiene las ventas mayores por categoría. Obsérvese como hay que montar una subconsulta sobre la consulta anterior. La razón está en el orden de procesamiento, la parte del `MAX(SUM(amount_sold)) OVER (PARTITION BY prod_category)` se ejecuta después del primer agrupamiento, por tanto, la cláusula where ya se ha procesado y no podemos volver a usarla para la condición que se queda con las filas en las que la suma coincide con el máximo `WHERE sales = MAX_REG_SALES;` Aun así, el ahorro en la escritura es notable, además de una mejora en la eficiencia en la ejecución de la consulta.

```
SELECT prod_category, country_region, sales
FROM (SELECT SUBSTR(p.prod_category,1,8) AS prod_category,
co.country_region,
 SUM(amount_sold) AS sales,
MAX(SUM(amount_sold)) OVER (PARTITION BY prod_category) AS
MAX_REG_SALES
FROM sales s, customers c, countries co, products p
WHERE s.cust_id=c.cust_id
AND c.country_id=co.country_id
AND s.prod_id =p.prod_id
AND s.time_id = TO_DATE('11-OCT-2001')
GROUP BY prod_category, country_region)
WHERE sales = MAX_REG_SALES;
```

El resultado sería:

PROD_CATEGORY	COUNTRY_REGION	SALES
Electron	Americas	581,92
Hardware	Americas	925,93
Peripher	Europe	4290,38
Software	Americas	4445,7

Función RATIO_TO_REPORT.

Calcula la proporción de un valor con respecto a la suma de un conjunto de valores. Si el valor de la expresión (expr) se evalúa como NULL, RATIO_TO_REPORT también se evalúa como NULL.

Su sintaxis es:

```
RATIO_TO_REPORT (expr) OVER ([query_partition_clause])
```

En este sentido, se aplica lo siguiente:

- expr puede ser cualquier expresión válida que haga referencia a columnas o funciones de grupo de valores.
- La cláusula PARTITION BY define los grupos sobre los que la función RATIO_TO_REPORT será calculada. Si la cláusula PARTITION BY está ausente,

entonces la función se calcula sobre el conjunto completo del resultado de la consulta.

Ejemplo: Ver el porcentaje de ventas de cada canal de distribución en la fecha indicada:

```
SELECT
ch.channel_desc,
TO_CHAR(SUM(amount_sold),'9G999G999') AS SALES,
TO_CHAR(SUM(SUM(amount_sold)) OVER (), '9G999G999') AS TOTAL_SALES,
TO_CHAR(RATIO_TO_REPORT(SUM(amount_sold)) OVER (),'9D999') AS RATIO_TO_REPORT
FROM sales s, channels ch
WHERE s.channel_id=ch.channel_id  AND s.time_id=to_DATE('11-OCT-2000')
GROUP BY ch.channel_desc;
```

El resultado sería el siguiente:

CHANNEL_DESC	SALES	TOTAL_SALES	RATIO_TO_REPORT
Partners	8.391	23.183	,362
Direct Sales	14.447	23.183	,623
Internet	345	23.183	,015