UNIDAD DE TRABAJO 4: DEFINICIÓN de Datos

Conceptos

- Introducción
- Objetos de la base de datos
- Tipos de datos
- La sentencia CREATE TABLE
- Referencias a tablas de otros usuarios
- La opción DEFAULT
- Tablas de una base de datos Oracle
- Consultas al Diccionario de Datos
- Tipos de datos
- Creación de una tabla por medio de una subconsulta
- La sentencia ALTER TABLE
- Añadir una columna
- Modificar una columna
- Eliminación de una columna
- Opción SET UNUSED
- La sentencia DROP TABLE
- Cambiar el nombre a una tabla
- Truncar una tabla
- Añadir comentarios a una tabla
- ¿Qué son las restricciones?
- Definición de restricciones
- La restricción NOT NULL
- La restricción UNIQUE Key
- La restricción PRIMARY KEY
- La restricción FOREIGN KEY
- La restricción CHECK
- Operaciones con restricciones
- Añadir una restricción
- Eliminar una restricción
- Desactivar restricciones

- **Activar restricciones**
- **Consulta de restricciones en el Diccionario de Datos**
- **Vistas**

1. *Introducción*

El lenguaje SQL (*Structured Query Language*) permite la comunicación con el SGBD. Fue desarrollado sobre un prototipo de SGBD relacional denominado SYSTEM R y diseñado por IBM a mediados de los años setenta. En 1979 Oracle Corporation presentó la primera implementación comercial de SQL, que estuvo disponible antes que otros productos de IBM. Por su parte, IBM desarrolló productos herederos del prototipo SYSTEM R, como DB2 y SQL/DS.

El instituto ANSI (*American National Standard Institute*) adoptó el lenguaje SQL como estándar para la gestión de bases de datos relacionales en octubre de 1986. En 1987 lo adopta ISO (*International Standardization Organization*).

SQL es tan conocido en bases de datos que muchos lenguajes de programación incorporan sentencias SQL como parte de su repertorio; tal es el caso de Visual Basic para acceder a bases de datos relacionales.

Entre las principales características de SQL podemos destacar:

- **Es un lenguaje para todo tipo de usuarios (administradores, desarrolladores y usuarios normales), el usuario que emplea SQL especifica qué quiere, no cómo ni dónde, permite hacer cualquier consulta de datos.**
- **Es un lenguaje no procedimental. Permite al usuario solicitar un resultado sin preocuparse de los medios necesarios para obtener dicho resultado.**
- **Permite manipular un registro o un conjunto de registros , por tanto, no son necesarias estructuras de control.**
- **Universal, puede utilizarse en todos los niveles de la gestión de una bbdd, administración de la bbdd, desarrollo, gestion de datos,etc**

Podemos clasificar las sentencias SQL en 2 tipos:

- **Sentencias DDL (*Data Definition Language*). Con este lenguaje se crea y mantiene la estructura de la bbdd. Sirve para realizar las siguientes tareas:**

    - **Crear un objeto de la bbdd, tablas, vistas, procedimientos...(Create)**
    - **Eliminar un objeto de la bbdd (Drop)**
    - **Modificar un objeto de la bbdd (Alter)**
    - **Conceder privilegios sobre un objeto de la bbdd (Grant)**
    - **Retirar privilegios sobre un objeto de la bbdd. (revoke)**

- **Sentencias DML (Data Manipulation Language) Conjunto de sentencias sirve para manipular los datos contenidos en la bbdd. Es posible hacer:**

    - **Insert. Insertar filas en la bbdd**
    - **Update. Actualizar filas de datos**
    - **Delete. Eliminar filas de datos.**
    - **Select. Recuperar filas de datos.**
    - **Bloqueo de tablas**

Existen otro grupo de sentencias SQL no tan extendidas como las anteriores y son:

- **Lenguaje de control de transacción, gestiona las modificaciones realizadas por las instrucciones DML:**

    - **Características de la transacción**
    - **Validación y anulación de la modificación**

COMMIT, SAVEPOINT, ROLLBACK, SET TRANSACTION

En la presente unidad se va a contemplar el sublenguaje DDL (*Data Definition Language*) de SQL. Se mostrará cómo crear los objetos principales del modelo relacional: las tablas, así como las restricciones de los datos. Se cierra la línea que iniciamos en la unidad anterior iniciada en el diseño conceptual, seguida del diseño lógico, terminando en el diseño físico con la creación de las tablas y las restricciones en el SGBDR Oracle.

Por último, indicar que las sentencias DDL tienen incluido el comando COMMIT, que finaliza y lleva a cabo todas las transacciones pendientes hasta el lanzamiento de una sentencia DDL.

2. *Objetos de la base de datos*

Una base de datos Oracle puede contener múltiples estructuras de datos. Cada estructura debería definirse en el diseño de la base de datos, para que pueda ser creada durante la etapa de construcción del desarrollo de la base de datos.

Principales objetos:

- **Tabla → Unidad básica de almacenamiento, compuesta de registros y columnas.**
- **Vista → representación lógica de un subconjunto de una o mas tablas. Puede manipularse como una tabla (tabla virtual)**
- **Secuencia → Genera valores para la clave primaria. (Similar a campo autonumérico de Access)**
- **Índice → Columna o conjunto de columnas que permiten realizar búsquedas más rápidas. Sirven para mejorar el rendimiento de algunas consultas, en las que se incluya.**
- **Sinónimo → Da nombres alternativos a los objetos de almacenamiento. Nombres alternativos a tablas o vistas.**

Notas sobre tablas en Oracle:

- ✓ **Pueden ser creadas en cualquier momento, incluso mientras los usuarios usan la BD.**
- ✓ **No se necesita especificar el tamaño de ninguna tabla. El tamaño es definido por la cantidad de espacio en la BD según va creciendo. Es importante, sin embargo, estimar cuánto espacio utilizará la tabla.**
- ✓ **La estructura de una tabla puede ser modificada on-line.**

Reglas para los nombres de tablas columnas y demás objetos de la BD Oracle:

- ❖ **Deben comenzar por una letra.**
- ❖ **Pueden tener una longitud de 1-30 caracteres de largo.**
- ❖ **Caracteres permitidos: A-Z, a-z, 0-9, , $, # (No valen los espacios)**
- ❖ **No deben duplicar el nombre de otro objeto que sea propiedad del mismo usuario.**
- ❖ **No debe ser una palabra reservada del servidor Oracle.**

Nota: los nombres no son sensibles a mayúsculas/minúsculas. Es lo mismo DEPT que dept que Dept.

3. Tipos de datos

CHAR(tamaño)	**cadena de caracteres de longitud fija con longitud "tamaño". Por defecto vale 1.** **1 >= tamaño <= 2000** **Si se introduce una cadena de menor longitud a la establecida lo rellena con espacios en blanco en la derecha.**
VARCHAR2(tamaño)	**cadena de caracteres de longitud variable, máximo el "tamaño".** **1 >= tamaño <= 4000 Apellido Varchar2(20)** **Si se introduce una cadena superior al tamaño definido Oracle dará un error**
NUMBER(P,S) **NUMBER (n)**	**Representa datos numericos tanto enteros como reales y con signo.**

NUMBER	**Cuando especifico p,s es porque puedo poner máximo número de dígitos p (entre 1 y 38)** **S representa el número de dígitos de la parte fraccionaria (hasta 127)** **Ejemplo: Sal_Medio NUMBER(9,2) podría guardar 4566770.86** **Cuando pongo n, es por que trabajo con números enteros y en decimal supone 0** **Salario Number(10)** **Cuando solo pongo NUMBER se guarda en decimal o entero según se teclee** **El carácter por defecto para separar la parte fija y la fraccionaria es el punto. Se puede modificar en NLS_NUMERIC_CHARACTERS(ALTER SESSION)**
DATE	**Fecha y horas. Para cada tipo Date se almacena Siglo/Año/Mes/día/Hora/Minutos/segundos** **El formato de la fecha está en NLS_DATE_FORMAT . Por defecto es dd//mm/yy** **Ejemplo: fecha_edición date**
TIMESTAMP(P)	**Datos tipo Date en los que podemos indicar la precisión para las fracciones de segundo. Por defecto es 6**
BLOB	**datos binarios, hasta 4 GBytes**
BFILE	**Datos binarios almacenados en archivos externos a la bbdd (4 GB máximo)**
LONG	**Cadenas de caracteres de longitud variable (2 GB máximo) Para almacenar textos grandes.** **Solo se puede definir una columna por tabla**

	No pueden aparecer en restricciones de integridad (constraints) ni para indexar, ni en where,etc
RAW(N)	**Datos binarios de longitud variable no interpretable por Oracle. La diferencia con los varchar2 es que maneja cadenas de caracteres en lugar de cadenas de n bytes (n <2000 bytes)**
LONG RAW(N)	**Datos binarios de longitud variable no interpretable por Oracle Se emplea para almacenamiento de graficos. Utiliza (4 Gb como máximo)**

4. La sentencia CREATE TABLE

Esta sentencia forma parte del sublenguaje DDL de SQL y es utilizada para crear tablas. Tiene un efecto inmediato sobre la base de datos y graba información en el Diccionario de Datos.

Para crear una tabla, el usuario debe tener el privilegio CREATE TABLE y un área de almacenamiento para crear objetos

Es posible definir hasta 1000 columnas por tabla

Sintaxis:

CREATE [GLOBAL TEMPORARY] TABLE [schema.]nombretabla (columna tipo_datos [DEFAULT expr] [NOT NULL]) [TABLESPACE espacio_de_tabla] ;

- **GLOBAL TEMPORARY: especifica que la tabla es temporal y que su definición es visible para todas las sesiones. Los datos en una tabla temporal son visibles sólo para la sesión que inserta datos en la tabla.**

- **Schema. : propietario**
- **DEFAULT expr: específica un valor por defecto si se omite en la sentencia insert.**
- **NOT NULL la columna debe contener alguna información. Nunca puede ser nula cuando insertemos una nueva fila**
- **Se puede indicar el TABLESPACE donde se debe almacenar la tabla. Este debe existir. Si se omite se inserta en el tablespace que tenga asignado el usuario.**

El usuario debe especificar el nombre de la tabla y los nombres y tipos de las columnas, siguiendo las reglas para la definición de nombres.

Ejercicio: Crear la tabla alumnos con num_mat, nombre, fecha_nac, dirección, localidad en el tablespace USER_DATA

```
CREATE TABLE ALUMNOS(
NUM_MAT NUMBER(6) NOT NULL,
NOMBRE VARCHAR2(15) NOT NULL,
FECHA_NAC DATE,
DIRECCION VARCHAR2(30),
LOCALIDAD VRCHAR2(15)
) TABLESPACE USER_DATA;
```

Referencias a tablas de otros usuarios:

Un esquema (*schema*) es una colección de objetos. Los objetos de un esquema son las estructuras lógicas que hacen referencia directa a los datos en una base de datos. Incluyen tablas, vistas, sinónimos, secuencias, procedimientos almacenados, índices, clusters y database links.

Las tablas propiedad de otros usuarios no pertenecen a nuestro esquema. Si deseamos hacer referencia a ellas, debemos anteponer el nombre del usuario propietario al de la tabla separando ambos

vocablos con un punto.

Podremos acceder a esas tablas si el propietario lo permite con los permisos oportunos.

La opción DEFAULT:

Usando la opción DEFAULT se puede dar un valor por defecto a una columna. Esta opción previene la introducción de valores nulos en aquellas filas que no especifican un valor para esa columna. El valor por defecto puede ser un literal, una expresión o una función SQL, pero el valor no puede ser el nombre de otra columna. La expresión por defecto debe ser del mismo tipo que el de la columna.

Ejemplo:

```
CREATE TABLE dept2
(numdept NUMBER(2),
nombre dept VARCHAR2(14),
ubicacion VARCHAR2(13));
```

5. Tablas de una base de datos Oracle

Las tablas de usuario son tablas creadas por el usuario.

Hay otro conjunto de tablas y vistas en una base de datos Oracle conocido como *Diccionario de Datos* (DD). Este conjunto de tablas es creado y mantenido por Oracle y contiene información sobre la base de datos.

Todas las tablas del DD son propiedad del usuario SYS.

Los usuarios raramente acceden a las tablas base porque la información que hay en ellas no es fácil de entender.

No se puede escribir directamente sobre él.

Los usuarios normalmente acceden a vistas del diccionario de datos porque aquí la información se presenta en un formato más fácil de

entender.

La información que se almacena en el DD incluye nombres de los usuarios de Oracle Server, privilegios concedidos a los usuarios, nombres de los objetos de la BD, restricciones de integridad e información de auditoría.

Consultas al Diccionario de Datos:

Se consultan las tablas del DD para localizar información acerca de los objetos que contiene. Algunas de las tablas del diccionario utilizadas con mayor frecuencia son:

- **USER_TABLES**
- **USER_OBJECTS**
- **USER_CATALOG: Tablas, vistas, sinónimos, y secuencias propiedad del usuario.**

Ejemplos:

- **Tablas propiedad del usuario:**

select table_name from user_tables;

En este caso me visualiza los nombres de las tablas de usuario, es decir, mis tablas

select * from user_tables;

Visualiza toda la información posible de todas mis tablas

USER_TABLES es una vista cuyo propietario es SYS, por eso ponemos:

select table_name from sys.user_tables;

Visualiza el nombre de las tablas creadas igual que anteriormente.

También podemos poner:

Desc SYS.USER_TABLES; Me visualiza todos los campos de la vista USER_TABLES

- **Distintos objetos propiedad del usuario:**

select distinct object_type from user_objects;

En el caso de scott me dice que los únicos objetos que tiene son vistas y tablas

select object_type, object_name from user_objects;

Nos dice de cada objeto su nombre y el tipo que es.

select object_type, substr (object_name, 1, 10) from user_objects;

Como los nombres salían en un formato poco legible cogemos del nombre solo los 10 primeros caracteres.

- **Tablas, vistas, sinónimos y secuencias propiedad del usuario:**

select * from user_catalog;

6. Creación de una tabla por medio de una subconsulta

Un segundo método para crear una tabla, es aplicar una cláusula AS como subconsulta, para crear tanto la tabla, como para llenarla con los registros devueltos por la subconsulta.

Sintaxis:

CREATE TABLE [columna (,columna…)] AS subconsulta;

Ejemplo:

CREATE TABLE dept50 AS select * from dept where deptno=50;

La tabla se creará con los nombres de columnas especificadas y los

registros recuperados por la sentencia SELECT serán insertados en la tabla. Si se dieran nombres de columnas, su número será igual al número de columnas especificadas en la sentencia SELECT de la subconsulta.

Cuando creo una tabla así no copia las restricciones de la tabla original.

7. La sentencia ALTER TABLE

Después de haber creado una tabla, puede necesitar cambiar su estructura, porque tal vez omitió una columna, o la definición ha cambiado. Esto puede hacerse utilizando la sentencia ALTER TABLE.

Se utiliza para:

- **Añadir una nueva columna (o atributo)**
- **Modificar una columna existente**
- **Dar un valor por defecto a una nueva columna**

Añadir una columna:

Sintaxis:

```
ALTER TABLE tabla
ADD  (columna tipo  datos [DEFAULT expr]
[, columna tipo  datos]…);
```

Ejemplo:

```
  ALTER TABLE dept30
  ADD (puesto VARCHAR2(9));
```

La columna añadida aparecerá la última de la tabla.

Modificar una columna:

Se utiliza para cambiar el tipo de datos de una columna, su tamaño o su valor por defecto. En este último caso, el valor por defecto se aplicará sólo a posteriores inserciones en la tabla.

Sintaxis:

```
ALTER TABLE tabla
MODIFY (columna tipo_datos [DEFAULT expr]
[, columna tipo_datos]...);
```

Ejemplo:

```
ALTER TABLE Dept
MODIFY (Provincia VARCHAR2(15));
```

Eliminación de una columna:

Esta sentencia se utiliza para borrar una columna de una tabla cada vez. La columna podrá o no contener datos no pudiéndose recuperar una vez se haya eliminado (no se puede recuperar con ROLLBACK puesto que lleva COMMIT implícito). En la tabla debe quedar como mínimo una columna después de ALTER.

Sintaxis:

```
ALTER TABLE tabla DROP COLUMN columna;
```

Ejemplo:

ALTER TABLE Dept DROP COLUMN Provincia;

Renombrar una columna:

ALTER TABLE nombre tabla
RENAME COLUMN old name to new name;

Opción SET UNUSED (Se utiliza conALTER):

La opción SET UNUSED marca una o más columnas como "no usadas" para que puedan ser eliminadas cuando se necesiten recursos en el sistema. Esta es una prestación propia de Oracle. Especificando esta cláusula no se eliminan realmente las columnas de cada fila de la tabla, es decir, no restaura el espacio usado por estas columnas. Sin embargo, el tiempo de respuesta es mejor que si se ejecuta la cláusula de borrado DROP.

Las columnas "unused" son tratadas como si hubieran sido borradas, aunque los datos permanecen en la tabla. Después de marcar una columna como "unused", no tenemos acceso a ella. Una sentencia de consulta no recuperará los datos de estas columnas. Además, los nombres y tipos de datos de estas columnas no se mostrarán con DESCRIBE y podemos añadir a la tabla una nueva columna con el mismo nombre que otra columna "unused".

Sintaxis:

ALTER TABLE tabla SET UNUSED (columna);

Ejemplo:

ALTER TABLE Dept30 SET UNUSED (ename);

En un segundo paso, eliminaremos las columnas "unused" para

conseguir el espacio en disco que ocupan. Si la tabla no contiene este tipo de columnas, la sentencia no devuelve error.

Sintaxis:

ALTER TABLE tabla DROP UNUSED COLUMNS;

8. La sentencia DROP TABLE

Sirve para eliminar las tablas (y también sus datos). Se borra la definición de la tabla en el DD y todos los índices asociados. Como toda sentencia DDL no se puede hacer ROLLBACK de la sentencia.

Solamente el propietario de la tabla u otro usuario con el permiso DROP ANY TABLE puede eliminar una tabla.

Sintaxis:

DROP TABLE tabla;

9. Cambiar el nombre de un objeto

Para cambiar el nombre de una tabla, vista, secuencia o sinónimo, se utiliza la instrucción RENAME. Ha de ser el propietario del objeto que renombra. Es una instrucción propia de Oracle.

Sintaxis:

RENAME nombre1 TO nombre2;

Ejemplo:

RENAME dept TO departamento;

10. Truncar una tabla

Es propia de Oracle y se utiliza para borrar todos los registros de una tabla y liberar así todo el espacio de almacenamiento ocupado por la tabla. No se puede hacer ROLLBACK. Es una alternativa a la sentencia DELETE de DML pero ésta si puede efectuar ROLLBACK. DELETE no libera espacio.

El usuario que trunque una tabla debe ser el propietario de la misma o tener el privilegio del sistema DELETE TABLE.

Sintaxis:

TRUNCATE TABLE tabla;

11. Añadir comentarios a una tabla

Es propio de Oracle y no de SQL. Sirve para añadir comentarios de hasta 2000 bytes sobre una columna, tabla o vista. El comentario se almacena en el Diccionario de Datos y puede ser visualizado por medio de la columna COMMENTS de una de las siguientes vistas del DD:

Para columnas:

- **ALL_COL_COMMENTS**
- **USER_COL_COMMENTS**

Para tablas:

- **ALL_TAB_COMMENTS**
- **USER_TAB_COMMENTS**

Sintaxis:

COMMENT ON TABLE tabla IS 'comentario descriptivo';

COMMENT ON COLUMN tabla.col IS 'comentario descriptivo';

12. *¿Qué son las restricciones?*

Cuando almacenamos datos en nuestras tablas, se ajustan a una serie de restricciones:

- **Integridad de datos**
  - **Que una columna no puede almacenar valores negativos**
  - **Que se almacenen las cadenas en mayúsculas**
  - **Que una columna no permita 0**

  La integridad hace referencia al hecho de que los datos cumplan ciertas restricciones.

  Integridad: Regla que restringe el rango de valores de una o más columnas.
- **Integridad referencial**
  - **Garantiza que los valores de una columna de una tabla dependan de los valores de otra columna de otra tabla.**

  El servidor Oracle utiliza las restricciones (*constraints*) para prevenir la introducción de datos no validos en una tabla.

  Se pueden usar para
    - **Garantizar el cumplimiento de reglas a nivel de tablas, en el momento que una fila se inserta, se actualiza o se borra. La restricción debe ser cumplida para que la operación tenga éxito.**
    - **Impedir la eliminación de una tabla si existen dependencias desde otras tablas.**

RESTRICCIONES DE INTEGRIDAD EN ORACLE

CONSTRAINT	***DESCRIPCIÓN***
NOT NULL	**Especifica que una columna no tenga valores nulos.**
UNIQUE	**Especifica que una**

	columna o combinación de ellas, tenga valores irrepetibles.
PRIMARY KEY	Especifica la clave principal.
FOREIGN KEY	Especifica una clave foránea.
CHECK	Especifica que una condición debe ser verdadera.

Se recomienda asignarles un nombre descriptivo a las restricciones. De lo contrario, Oracle le dará uno del tipo SYS_CX_ donde X es un número entero.

Se pueden crear en:

- En el momento de crear la tabla.
- Después de que la tabla haya sido creada.

Las restricciones se pueden definir tanto a nivel de columna como a nivel de tabla. Se pueden ver las restricciones en el Diccionario de Datos, en la tabla USER_CONSTRAINTS.

13. *Definición de restricciones*

Las restricciones se crean normalmente a la vez que se crea la tabla pero pueden ser añadidas después de haber creado la tabla y también pueden desactivarse temporalmente.

Sintaxis:

```
CREATE TABLE tabla
(columna1 tipo_datos [CONSTRAINT_NOMBRERESTRICCION][NOT NULL] [CONSTRAINT_NOMBRERESTRICCION][UNIQUE] [CONSTRAINT_NOMBRERESTRICCION][PRIMARY KEY]
```

[CONSTRAINT NOMBRERESTRICCION][DEFAULT VALOR][REFERENCES NOMBRE TABLA][CHECK CONDICION]
columna2 tipo datos [CONSTRAINT NOMBRERESTRICCION][NOT NULL][UNIQUE][PRIMARY KEY][DEFAULT VALOR][REFERENCES NOMBRE TABLA][CHECK CONDICION]

...
[constraint tabla]);

Ejemplo (restricción de columna sin nombre explícito):

```
CREATE TABLE empleados
  (empno NUMBER (4),
   ename VARCHAR2 (10),
   deptno NUMBER (2) NOT NULL,
   CONSTRAINT empleados_empno_pk PRIMARY KEY (empno));
```

Ejemplo (restricción de columna con nombre explícito):

```
CREATE TABLE empleados
  (empno NUMBER (4),
   ename VARCHAR2 (10),
   deptno NUMBER (2) CONSTRAINT empleados_deptno_nn NOT NULL,
   CONSTRAINT empleados_empno_pk PRIMARY KEY (empno));
```

La restricción NOT NULL:

La restricción NOT NULL asegura que en la columna no se permitirán valores nulos. Las columnas sin la restricción NOT NULL pueden contener valores nulos, por defecto.

Esta restricción se aplica solamente a nivel de columna, no a nivel de tabla.

La restricción UNIQUE Key:

Esta restricción requiere que cada valor en una columna o conjunto de columnas sean únicas, es decir, dos registros de una tabla no tendrán valores duplicados para determinada columna o conjunto de columnas.

Permite la entrada de nulos, a menos que se especifique también la restricción NOT NULL para las mismas columnas.

Una columna o conjunto de columnas con restricciones UNIQUE y NOT NULL definen una clave alternativa en una tabla.

Sintaxis:

… CONSTRAINT nombre_uk UNIQUE (columna [,columna]…)….

Ejemplo:

```
CREATE TABLE dept
(deptno NUMBRE(2),
 dname VARCHAR2(14),
 loc VARCHAR2(13),
 CONSTRAINT dept_dname_uq UNIQUE (dname));
```

La restricción PRIMARY KEY:

Crea una clave principal en una tabla. Se puede crear solamente una Primay Key por tabla. Esta restricción consiste en una columna o conjunto de columnas que identifican unívocamente a cada fila de una tabla. Garantiza la unicidad de la columna (o combinación de ellas) y asegura que ninguna columna que sea parte de la Primary Key pueda contener un valor nulo.

Puede definirse a nivel de columna o de tabla. Oracle crea automáticamente un índice para la columna (o columnas) de la clave principal.

A nivel de tabla la sintaxis será:

… CONSTRAINT nombre_pk PRIMARY KEY (columna [,columna]…)….

A nivel de columna la sintaxis será:

```
CREATE TABLE nombredetabla
(col1 tipocol [CONSTRAINT nombre] PRIMARY KEY;
```

Ejemplo:

```
CREATE TABLE dept
(deptno NUMBER(2),
 dname VARCHAR2(14),
 loc VARCHAR2(13),
 CONSTRAINT dept_dname_uq UNIQUE (dname),
 CONSTRAINT dept_deptno_pk PRIMARY KEY (deptno));
```

Cuando una tabla está formada por una clave principal de varias columnas, no podremos utilizar la sintaxis de columnas, tendremos que definirla a nivel de tabla.

La restricción FOREIGN KEY:

La restricción FOREIGN KEY, o de integridad referencial, designa a una columna o combinación de ellas como una clave extranjera y establece una relación con una Primary Key de la misma tabla o de otra. Se pueden definir a nivel de tabla o columna.

El valor de la Foreign Key debe coincidir con uno de la tabla padre o ser NULL.

Las Foreign Key están basadas en valores de datos y son puramente lógicos, no son punteros físicos.

Podemos definir tantas claves ajenas como sea necesario.

Sintaxis a nivel de tabla:

```
... CONSTRAINT nombre_fk FOREIGN KEY (columna [,columna]...)
    REFERENCES tabla (columna [,columna]...) ....
```

Sintaxis a nivel de columna:

```
Columna tipo  columna [CONSTRAINT nombrerestriccion]
REFERENCES nombretabla[(columna)][ ON DELETE CASCADE];
```

Ejemplo (a nivel de tabla):

```
CREATE TABLE empleados
(empno NUMBER(4),
 ename VARCHAR2(10) NOT NULL,
 job VARCHAR2 (9),
 deptno NUMBER (3) NOT NULL,
 CONSTRAINT emp_deptno_fk FOREIGN KEY (deptno)
REFERENCES dept (deptno));
```

Ejemplo (a nivel de columna):

```
CREATE TABLE empleados
(empno NUMBER(4),
```

ename VARCHAR2(10) NOT NULL,
job VARCHAR2 (9),
deptno NUMBER (3) CONSTRAINT emp_deptno_fk REFERENCES dept (deptno));

La palabra reservada FOREIGN KEY no es necesaria en este último caso.

A esta restricción se le puede añadir la opción ON DELETE CASCADE. Indica que cuando la fila es borrada en la tabla padre, se borrarán todas las filas dependientes en la tabla hija.

Sin la opción ON DELETE CASCADE, la fila en la tabla padre no puede ser borrada mientras haya referencias a ella en la tabla hija. (EJEMPLO DE DEPT EN ORACLE)

Sintaxis:

… CONSTRAINT nombre_fk FOREIGN KEY (columna [,columna]…)
REFERENCES tabla (columna [,columna]…) ON DELETE CASCADE …

Ejemplo:

CONSTRAINT emp_deptno_fk FOREIGN KEY (deptno) REFERENCES dept (deptno) ON DELETE CASCADE);

EJEMPLO:

Crear la tabla PERSONAS con la siguiente información:

NOMBRE COLUMNA	TIPO	RESTRICCION

DNI	**NUMBER(8)**	**PRIMARIA**
NOMBRE	**VARCHAR2(15)**	
DIRECCIÓN	**VARCHAR2(15)**	
POBLACIÓN	**VARCHAR2(15)**	
COD_PROV	**NUMBER(2)**	**AJENA**

Crear la tabla PROVINCIAS con la siguiente información:

NOMBRE COLUMNA	**TIPO**	**RESTRICCION**
COD_PROV	**NUMBER(2)**	**PRIMARIA**
NOMBRE	**VARCHAR2(15)**	

```
CREATE TABLE PERSONAS
(....
FOREIGN KEY (COD_PROV) REFERENCES PROVINCIAS
)
```

Si creamos primero la tabla PERSONAS al meter esta sentencia nos dirá que no existe la tabla PROVINCIAS, por este motivo tendremos que crear antes la tabla PROVINCIAS.

Igualmente si queremos borrar las tablas, habrá que borrar antes PERSONAS que PROVINCIAS , pues de lo contrario Oracle dá error al borrar PROVINVIAS pues te dice que tiene claves primarias que son referenciadas en otra tabla.

Si quisiéramos borrar alguna provincia en concreto, y automáticamente borrasemos las personas asociadas, pondríamos ON DELETE CASCADE.

```
FOREIGN KEY (COD_PROV) REFERENCES PROVINCIAS ON DELETE CASCADE.
```

RESUMEN DE RESTRICCIONES:

- **RESTRICCIONES DE BORRADO: Una fila no se puede borrar si es referenciada por un clave ajena. Tampoco se puede actualizar la clave primaria .**
- **RESTRICCIONES EN CASCADA: Si borramos una fila de la tabla maestra (PROVINCIAS), todas las filas de la tabla detalle (PERSONAS) que sean referenciadas se borraran automáticamente. Cuando salga el mensaje de n rows deleted solo se refiere a las de la tabla maestra.**

La restricción CHECK:

Define una condición que cada fila debe satisfacer. La condición puede usar la misma sintaxis que las condiciones de las consultas (ej. WHERE color = red)

Una misma columna puede tener más de una restricción CHECK. Se pueden definir a nivel de tabla o de columna.

Sintaxis:

… CONSTRAINT nombre_ck CHECK (condición) …

Ejemplo a nivel de tabla:

```
CREATE TABLE emp
(deptno NUMBER (2),
 CONSTRAINT emp_deptno_ck CHECK (deptno BETWEEN 10 AND 99));
```

Ejemplo:
Crear la tabla BLOQUEDEPISOS

14. *Operaciones con restricciones*

Las operaciones que pueden realizarse con las restricciones son:

- **Añadir**
- **Eliminar**
- **Activar**
- **Desactivar**
- **Visualización en el Diccionario de Datos.**

Añadir una restricción:

Se pueden añadir restricciones para tablas ya existentes, usando la sentencia ALTER TABLE con la cláusula ADD.

Sintaxis:

```
  ALTER TABLE tabla
  ADD [CONSTRAINT restricción] tipo  restricción (columna)
```

En el caso de querer añadir una restricción del tipo NOT NULL a una columna existente, debe utilizarse la sentencia ALTER TABLE con la cláusula MODIFY. Esto sólo puede hacerse si la tabla no contiene filas.

Ejemplo:

```
ALTER TABLE empleados
ADD CONSTRAINT empleados_mgr_fk FOREIGN KEY (mgr)
REFERENCES empleados(empno);
```

Eliminar una restricción:

Se utiliza para borrar una restricción creada previamente. Cuando se borra una restricción de integridad, esa restricción ya no es parte del servidor Oracle y por tanto, no está disponible en el

Diccionario de Datos.

Sintaxis

ALTER TABLE tabla DROP CONSTRAINT nombre_constraint;

Ejemplos:

ALTER TABLE emp DROP CONSTRAINT emp_mgr_fk;

ALTER TABLE dept DROP PRIMARY KEY CASCADE;

En el último ejemplo, se borra la restricción de Primary Key de la tabla dept y también la restricción de Foreign Key asociada en la columna emp.deptno.

Desactivar restricciones:

Puede desactivarse una restricción sin borrarla. Se utiliza la sentencia ALTER TABLE con la cláusula DISABLE. Con la opción CASCADE se desactivan las restricciones de integridad dependientes.

Puede utilizarse la cláusula DISABLE también en la sentencia CREATE TABLE

Sintaxis:

ALTER TABLE tabla
DISABLE CONSTRAINT nombre_restricción [CASCADE];

Ejemplo:

ALTER TABLE empleados
DISABLE CONSTRAINT empleados_empno_pk CASCADE;

Activar restricciones:

Permite activar restricciones de integridad actualmente desactivadas en la definición de la tabla por medio de la cláusula ENABLE.

Si se activa una restricción UNIQUE o PRIMARY KEY se creará automáticamente un índice para la clave UNIQUE o PRIMARY KEY.

Puede utilizarse la cláusula ENABLE también en la sentencia CREATE TABLE

Sintaxis:

ALTER TABLE tabla
ENABLE CONSTRAINT nombre_restricción;

Ejemplo:

ALTER TABLE emp ENABLE CONSTRAINT emp_empno_pk;

15. Consulta de restricciones en el Diccionario de Datos

Tras crear una tabla, puede verificarse su existencia haciendo uso del comando DESCRIBE. La única restricción que puede verificar es NOT NULL. Para ver todas las restricciones sobre una tabla, hay que consultar la vista USER_CONSTRAINTS del Diccionario de Datos.

En la vista USER_CONSTRAINTS se ven todos los nombres y definiciones de restricciones.

En la vista USER_CONS_COLUMNS se ven todas las columnas asociadas con los nombres de las restricciones. Esta vista es especialmente útil para restricciones a las que el propio sistema ha asignado un nombre.

Ejemplos:

```
SELECT constraint_name, constraint_type, search_condition
FROM USER_CONSTRAINTS
WHERE table_name = 'EMP';
```

```
SELECT constraint_name, column_name
FROM USER_CONS_COLUMNS
WHERE table_name = 'EMP';
```

16. Vistas

Una vista representa lógicamente un subconjunto de una o más tablas (llamadas tablas base). Una vista no contiene datos en sí misma pero es como una ventana a través de la cual se pueden ver o cambiar los datos de las tablas.

La vista se almacena como una sentencia SELECT en el diccionario de datos.

Las vistas se utilizan para:

- **Restringir el acceso a los datos**
- **Realizar consultas complejas fácilmente. Por ejemplo, para que los usuarios consulten datos de varias tablas sin necesidad de conocer cómo escribir sentencias de unión o JOIN.**
- **Independencia de los datos para usuarios y aplicaciones: una vista puede ser usada para recuperar datos de diversas tablas**
- **Presentar diferentes vistas de los mismos datos, ofreciendo acceso a los datos a grupos de usuarios de acuerdo a su criterio particular**

Tipos de vistas:

- **Simples: utilizan una sola tabla, no contienen funciones ni grupos de datos y pueden hacer DML a través de la vista.**
- **Complejas: utilizan una o más tablas, contienen funciones o grupos de datos y no siempre pueden realizar DML a través de la vista.**

Sentencia CREATE VIEW

Podemos crear una vista embebiendo una subconsulta dentro de la sentencia CREATE VIEW.

Es una sentencia perteneciente al sublenguaje DDL (Data Definition Language)

Sintaxis:

```
CREATE [OR REPLACE] [FORCE | NOFORCE] VIEW nombre_vista
[(columna[,columna]...)]
AS subconsulta
[WITH CHECK OPTION [CONSTRAINT restricción]]
[WITH READ ONLY];
```

- **OR REPLACE: recrea la vista si ya existe**
- **FORCE: crea la vista sin importar que la tabla base exista o no**
- **NOFORCE: crea la vista únicamente si la tabla base existe. Es la opción por defecto.**
- **alias: especifica nombres para las expresiones seleccionadas en la consulta de la vista. El número de alias de las columnas debe coincidir con el de expresiones seleccionadas por la subselect. Cuando no se ponen nombres de columnas**
- **subconsulta: es una sentencia SELECT completa. Se pueden usar los alias para las columnas de la lista del SELECT. No puede contener la cláusula ORDER BY.**
- **WITH CHECK OPTION: especifica que solamente las filas accesibles a la vista pueden ser insertadas o actualizadas.**
- **restricción: es el nombre asignado a la restricción CHECK OPTION**
- **WITH READ ONLY: asegura que ninguna operación DML pueda**

realizarse sobre esta vista.

- **Si se suprime una tabla la vista asociada es invalida**

Creación de vistas sencillas

- **Las vistas mas sencillas son las que unicamente tocan una tabla**

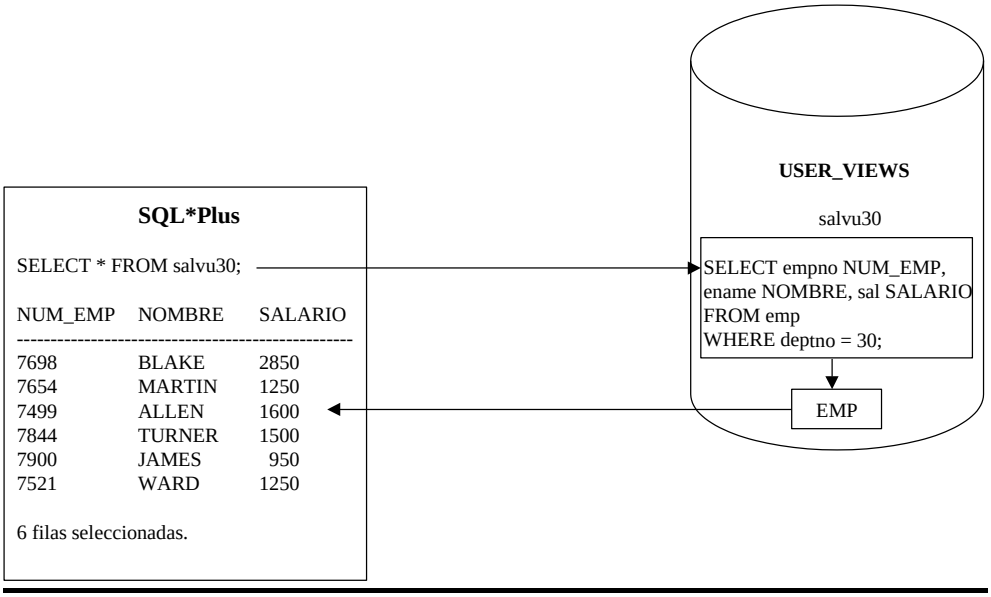
Ejemplos: Vista de los ermpleados del deparatmento 30

```
CREATE VIEW empvu10
AS SELECT empno, ename, job
FROM emp
WHERE deptno = 10;

CREATE VIEW salvu30
AS SELECT empno NUM  EMP, ename NOMBRE, sal SALARIO
FROM emp
WHERE deptno = 30;
```

Recuperación de datos de una vista

Podemos recuperar datos desde una vista al igual que lo hacemos desde una tabla. Podemos visualizar el contenido de la vista al completo o sólo ver columnas y filas específicas.

```
  SELECT * FROM salvu30;
```



Modificación de una vista

La opción OR REPLACE permite que se cree una vista incluso si ya existe una con ese nombre, reemplazando de esta forma la antigua versión de la vista. Esto significa que la vista puede alterarse sin hacer DROP, volver a crear ni volver a conceder privilegios sobre el objeto.

Ejemplo:

```
CREATE OR REPLACE VIEW empvu10
        (num_empleado, nombre_empleado, empleo)
AS SELECT empno, ename, job
FROM emp
WHERE deptno = 10;
```

Nota: los alias de columna (en el ejemplo: num_empleado, nombre_empleado, empleo) en la cláusula CREATE VIEW deben aparecer en el mismo orden que las columnas en la subconsulta (en el ejemplo: empno,, ename, job).

Ejemplo de vista compleja:

CREATE VIEW deptsumvu

(nombre_dep, minimo_sueldo, maximo_sueldo, media_sueldo)

AS SELECT d.dname, MIN(e.sal), MAX(e.sal), AVG(e.sal)

FROM emp e, dept d

WHERE e.deptno = d.deptno

GROUP BY d.dname;

Reglas para realizar operaciones DML sobre vistas

Cuando creamos una vista con todas las columnas de la tabla asociada, podremos insertar filas en la vista sin ningún problema.

Podemos realizar operaciones DML sobre los datos a través de una vista siempre que esas operaciones sigan ciertas reglas.

Se puede eliminar una fila de una vista salvo que la misma contenga:

- **Funciones de grupo (SUM, AVG, COUNT, etc.)**
- **Una cláusula GROUP BY**
- **La cláusula DISTINCT**

Se pueden modificar los datos de una vista salvo que contenga una de las condiciones mencionadas y tenga columnas definidas por expresiones (por ejemplo: sal*12).

Se pueden agregar datos a través de una vista a menos que la misma contenga cualquiera de las condiciones anteriores y además no existan columnas NOT NULL en la tabla base no seleccionada por la vista. Todos los valores requeridos deben estar presentes en la vista. Recordar que se están agregando valores directamente en la tabla subyacente *a través* de la vista.

La cláusula WITH CHECK OPTION

Es posible realizar chequeos de integridad referencial a través de las vistas. Podemos reforzar las restricciones al nivel de la base de datos. La vista puede utilizarse para proteger la integridad de los datos pero el uso es muy limitado.

La cláusula WITH CHECK OPTION especifica que los INSERTs y UPDATEs realizados a través de las vistas no pueden crear filas que la vista no pueda seleccionar y, por lo tanto, esto permite las restricciones de integridad y chequeos en la validación de los datos para imponerse sobre los datos que se están insertando o actualizando.

Si hay un intento de realizar operaciones DML sobre las filas que la vista no ha seleccionado, se visualiza un error, con el nombre de la restricción si se ha especificado.

Ejemplo:

```
CREATE OR REPLACE VIEW empvu20
AS SELECT *
FROM EMP
WHERE deptno = 20
WITH CHECK OPTION CONSTRAINT empvu20_ck;

UPDATE empvu20
SET deptno = 22
WHERE empno = 7788;
```

SQL> ERROR: violación de la cláusula WITH CHECK OPTION. Si se intenta cambiar el número de departamento para cualquier fila, la sentencia fallará porque viola la restricción de CHECK OPTION.

La opción WITH READ ONLY

Podemos denegar operaciones DML sobre determinada vista creándola con la opción WITH READ ONLY.

Ejemplo:

```
CREATE OR REPLACE VIEW empvu10
   (num_empleado, nombre_empleado, empleo)
AS SELECT empno, ename, job
FROM emp
WHERE deptno = 10
WITH READ ONLY;

DELETE FROM empvu10
WHERE num_empleado = 7782;
```

SQL> ERROR: cualquier intento de quitar una fila de la vista, dará un error.

La sentencia DROP VIEW

Se utiliza la sentencia DROP VIEW para eliminar una vista. La ejecución de esta sentencia provoca la eliminación de la definición de la vista de la base de datos. La eliminación de una vista no afecta a las tablas sobre las que se basa la vista. Las vistas o las aplicaciones basadas en la vista eliminada se convierten en inválidas.

Únicamente el creador o un usuario con el privilegio DROP ANY VIEW puede eliminar una vista.

Sintaxis:

DROP VIEW nombre_vista;

Ejemplo:

DROP VIEW empvu10;

Vistas inline

- **Una vista inline es una subconsulta con un alias (nombre correlacionado) que podemos usar dentro de la sentencia SQL.**
- **Es similar a usar una subconsulta en la cláusula FROM de la consulta principal.**
- **Una vista inline no es un objeto del esquema.**

Ejemplo:

```
SELECT a.ename, a.sal, a.deptno, b.maxsal
FROM emp a, (SELECT deptno, max(sal) maxsal
        FROM emp
        GROUP BY deptno) b
WHERE a.deptno = b.deptno
AND a.sal < b.sal;
```

PARTE 5 DML

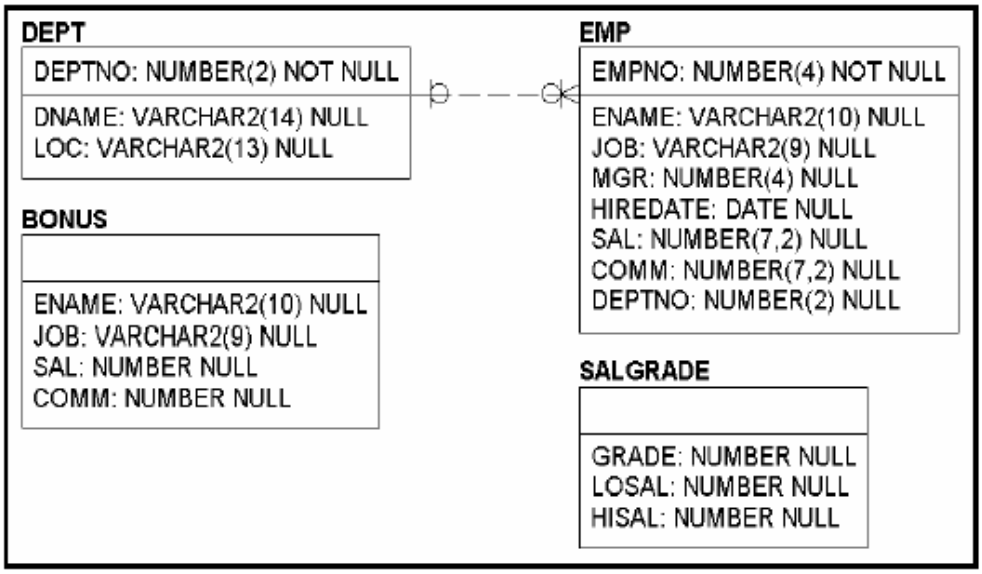
UNIDAD DE TRABAJO 5: Consulta y Manipulación de Datos

Esquema SCOTT

Para poder iniciar una sesión en el esquema de scout debemos utilizar los siguientes datos:

Usuario	scott
Contraseña	tiger

Su esquema es el siguiente:



1. La sentencia SELECT

Una sentencia SELECT recupera información de la base de datos. Utilizando una sentencia SELECT, podemos hacer lo siguiente:

- *Selección*: podemos seleccionar los registros de la tabla que queramos, devueltos por una consulta. Podemos utilizar varios criterios para restringir de manera selectiva los registros que queremos ver.
- *Proyección*: podemos seleccionar las columnas de la tabla que deseemos, devueltas por una consulta. Podemos seleccionar tantas tablas como queramos.
- *Unión*: para extraer los datos almacenados en diferentes tablas mediante la creación de enlaces entre una columna y las dos tablas compartidas.

Sintaxis:

SELECT [**DISTINCT**] {*, columna, … }

FROM tabla [,tabla …]
[WHERE condicion];

En la forma más simple una sentencia SELECT incluye las columnas a consultar (* indica todas) y una cláusula FROM que especifica la tabla que contiene las columnas descritas en la cláusula SELECT. Adicionalmente, puede incluir una cláusula WHERE que filtra las filas o registros que devolverá la consulta. Si no hay cláusula WHERE, se mostrarán todas las filas de la tabla.

La opción DISTINCT sirve para eliminar filas duplicadas en el resultado de una consulta.

Expresiones aritméticas, alias de columna, concatenación y strings

Puede ser necesario modificar la forma en que se visualizan los datos o se realizan los cálculos. Esto es posible utilizando expresiones aritméticas detrás de la cláusula SELECT. Una expresión aritmética puede contener nombres de columna, valores numéricos constantes y operadores aritméticos.

Los **operadores aritméticos disponibles en SQL son: + - * /**

Ejemplo:

```
SELECT ename, sal, sal+300
FROM emp;
```

La precedencia de estos operadores es: * / + - Podemos sobreescribir las reglas de precedencia utilizando paréntesis y así especificar el orden en que se ejecutan los operadores.

Cuando una expresión aritmética contiene un NULL se evalúa a NULL. Un NULL es un valor inaccesible o desconocido. No representa ni un cero ni un espacio en blanco. El cero es un número y el espacio es un carácter.

Si se desea renombrar el encabezamiento de una columna resultado se pueden utilizar alias. Es útil especialmente en cálculos. *Sigue inmediatamente al nombre de la columna y opcionalmente puede situarse en medio la palabra clave AS. Si se desea que contenga espacios en blanco o caracteres especiales se encierra entre comillas dobles.*

Ejemplo:

```
SELECT sal AS salario, ename "nombre del empleado"
FROM emp;
```

Con el operador de concatenación || se pueden concatenar columnas con otras columnas, expresiones aritméticas o valores constantes con el fin de crear una expresión de caracteres. Las columnas de cualquiera de los lados del operador se combinan para formar una sola columna resultante.

Ejemplo:

```
SELECT ename || job AS "Empleados"
FROM emp;
```

Un literal es un carácter, expresión o número incluido en la lista de la cláusula SELECT. Por cada registro devuelto se imprime una cadena de caracteres. Las cadenas de caracteres con formato de texto libre se pueden incluir en el resultado de la consulta y se tratan como una columna de la lista SELECT. Los valores literales de tipo fecha y carácter deben estar encerrados dentro de comillas simples ' ' mientras que los literales de tipo número no.

Ejemplo:

```
SELECT ename || ` es un ` || job AS Empleados
FROM emp;
```

En Oracle, para visualizar la estructura o definición de una tabla se utiliza el comando DESCRIBE.

Ejemplo:

```
DESCRIBE dept
```

Restricción y clasificación de los datos: condiciones en la cláusula WHERE

Podemos restringir las filas recuperadas usando la cláusula WHERE. Una cláusula WHERE contiene una condición que se debe cumplir y se escribe a continuación de la cláusula FROM.

La cláusula WHERE puede comparar los valores en las columnas, valores literales, expresiones aritméticas o funciones. La cláusula WHERE consta de tres elementos:

- Nombre de la columna
- Operador de comparación
- Nombre de columna, constante o lista de valores

Ejemplo:

```
SELECT ename, job, deptno
FROM emp
WHERE job='CLERK';
```

Tabla resultado devuelta:

ENAME	JOB	DEPTNO
JAMES	CLERK	30
SMITH	CLERK	20
ADAMS	CLERK	20
MILLER	CLERK	10

En el ejemplo, la sentencia SELECT recupera el nombre, oficio y número de departamento de todos los empleados cuyo oficio es CLERK.

Podemos observar que el oficio CLERK ha sido especificado en mayúsculas para asegurarse de que se corresponde con el valor de la columna JOB en la tabla EMP, que están todos en mayúsculas**. Las cadenas de caracteres son, por tanto, "case sensitive", es decir, distingue entre mayúsculas y minúsculas.**

Cadenas de caracteres y fechas

Las cadenas de caracteres y las fechas se encierran entre comillas simples en la cláusula WHERE. Sin embargo, los valore numéricos no se encierran entre comillas.

Si la consulta anterior se hubiera formulado como:

```
SELECT ename, job, deptno
FROM emp
WHERE job='clerck';
```

No se habría devuelto ningún valor en la tabla resultado porque todos los valores están almacenados en mayúsculas.

En cuanto **a las fechas, Oracle las almacena en un formato numérico interno, representado por el siglo, año, mes, día, hora, minutos y segundos. La fecha por defecto se visualiza como YYY-MM-DD.**
Evitar Comparaciones Directas con Texto o Valores Parciales: No uses LIKE, ni valores como '1982' o 1982, ya que no coinciden con el formato completo de un tipo DATE.

1. **Usa el formato estándar DATE 'YYYY-MM-DD' para escribir fechas literales:**

   WHERE HIREDATE = DATE '1982-06-15';

2. **Comparar con Rangos de Fechas: Usa el operador BETWEEN o condiciones de rango para filtrar fechas:**
   SELECT ENAME, HIREDATE
   FROM EMP
   WHERE HIREDATE BETWEEN DATE '1982-01-01' AND DATE '1982-12-31';
3. **Si necesitas trabajar con otro formato (como DD-MM-YYYY), debes convertirlo explícitamente usando TO_DATE**
   Alternativa con TO_DATE (solo si necesitas usar otro formato):

   SELECT ENAME, HIREDATE
   FROM EMP
   WHERE HIREDATE BETWEEN TO_DATE('01-01-1982', 'DD-MM-YYYY') AND TO_DATE('31-12-1982', 'DD-MM-YYYY')
4.

Operadores de comparación

Se utilizan en las condiciones en las que se compara una expresión con otra. Tienen el siguiente formato:

WHERE *expresion operador valor*

Los operadores de comparación clásicos son los siguientes:

=	Igual a
>	Mayor que
<	Menor que
>=	Mayor o igual que
<=	Menor o igual que
<>	Distinto o no igual a

Ejemplo:

```
SELECT ename, sal, comm
FROM emp
WHERE sal<=comm;
```

Otros operadores de comparación que pueden aparecer en una condición son:

- **BETWEEN ... AND ...**

Se utiliza para visualizar registros basados en un rango de valores. Los valores especificados como valor superior e inferior se incluyen en la operación.

Ejemplo:

```
SELECT ename, sal
FROM emp
WHERE sal BETWEEN 1000 AND 1500;
```

La cláusula WHERE se podría haber expresado también como:

```
WHERE sal >= 1000 AND sal <= 1500;
```

También se puede utilizar este operador de forma negativa precediéndola de NOT:

```
NOT BETWEEN … AND …
```

- **IN (lista ...)**

Se utiliza este operador para localizar valores coincidentes con una determinada lista. Puede utilizarse con cualquier tipo de dato.

Ejemplos:

```
SELECT empno, ename, sal, mgr
FROM emp
WHERE mgr IN (7902, 7566, 7788);

SELECT empno, ename, mgr, deptno
FROM emp
WHERE ename IN ('FORD', 'ALLEN');
```

Admite la versión negativa del operador: NOT IN (lista ….)

- **LIKE**

Se utiliza para realizar búsquedas en cadenas de caracteres. No siempre se conoce el valor exacto a buscar. Se pueden seleccionar filas que coincidan con un patrón de caracteres usando el operador LIKE. La

operación de coincidencia se conoce como una búsqueda que incluye *comodines*:

% representa cualquier secuencia de cero o más caracteres
_ (guión bajo) representa un solo carácter

Ejemplos:

- Empleados cuyo nombre empieza por S:

```
SELECT ename
FROM emp
WHERE ename LIKE 'S%';
```

- Empleados que se contrataron en 1981:

```
SELECT ename, hiredate
FROM emp
WHERE hiredate LIKE '%1981';
```

- Empleados cuyo nombre tenga una A como segunda letra:

```
SELECT ename
FROM emp
WHERE ename LIKE '_A%';
```

En el caso de necesitar una coincidencia exacta para los comodines, se utiliza la opción ESCAPE. Dicha opción especifica cual es el carácter ESCAPE. Si quisiéramos hacer una consulta sobre el departamento HEAD_QUARTERS, lo haríamos de la siguiente manera:

```
SELECT * FROM dept
WHERE dname LIKE '%\_%' ESCAPE '\';
```

Devolverá la tabla resultado:

DEPTNO	DNAME	LOC
50	HEAD_QUARTERS	ATLANTA

La opción ESCAPE identifica la barra (\) como el carácter ESCAPE. En el patrón de búsqueda, el carácter ESCAPE precede al _. Esto hace que Oracle interprete el _ literalmente.

- **IS NULL**

Verifica la presencia de valores nulos. Un valor nulo significa que el valor es inaccesible, sin valor, desconocido o inaplicable. Por lo tanto, **no se puede comprobar con el símbolo =**, porque un valor nulo no puede ser igual o distinto a otro valor.

Ejemplos:

```
SELECT ename, mgr
FROM emp
WHERE mgr IS NULL;

SELECT ename, job, comm
FROM emp
```

```
WHERE comm IS NULL;
```

Operadores lógicos

Un operador lógico combina el resultado de dos condiciones para producir un único resultado o para invertir el resultado de una condición simple. Hay tres operadores lógicos disponibles en SQL:

ANDdevuelve TRUE si ambas condiciones son TRUE
ORdevuelve TRUE si alguna de las condiciones es TRUE
NOTdevuelve TRUE si la siguiente condición es FALSE

Los operadores lógicos AND y OR nos permiten utilizar varias condiciones en una cláusula WHERE.

Ejemplos:

```
SELECT empno, ename, job, sal
    FROM emp
WHERE sal >= 1100 AND job = 'CLERK';

SELECT empno, ename, job, sal
    FROM emp
WHERE sal >= 1100 OR job = 'CLERK';

SELECT ename, job
    FROM emp
WHERE job NOT IN ('CLERK', MANAGER', ANALYST');

SELECT empno, ename, job,
    FROM emp
WHERE comm IS NOT NULL;
```

Reglas de Precedencia

Cuando aparece más de un operador en una condición, se evalúan en el orden que digan las reglas de precedencia.

ORDEN EVALUADO	OPERADOR
1	Todos los operadores de comparación
2	NOT
3	AND
4	OR

Se pueden utilizar los paréntesis para modificar las reglas de precedencia.

Ejemplo:

```
SELECT ename, job, sal
    FROM emp
WHERE job = 'SALESMAN' OR job='PRESIDENT' AND sal>1500;
```

Hay 2 condiciones:

- La primera es que el oficio sea PRESIDENT y el salario mayor que 1500.

- La segunda es que el oficio sea SALESMAN.

Por lo tanto, la sentencia SELECT interpretará lo siguiente: "Seleccionar los registros de los empleados cuyo oficio sea PRESIDENT y gane más de 1500 o si su oficio es SALESMAN".

En el caso de haber utilizado paréntesis como en el siguiente ejemplo, la condición sería totalmente distinta:

```
SELECT ename, job, sal
FROM emp
WHERE (job = 'SALESMAN' OR job='PRESIDENT') AND sal>1500;
```

"Seleccionar los registros de los empleados cuyo oficio sea PRESIDENT o SALESMAN y ganen más de 1500".

La cláusula ORDER BY

El orden de las filas recuperadas por el resultado de una consulta es indefinido. La cláusula ORDER BY se utiliza para ordenar las filas. Si se usa, será la última cláusula de la sentencia SELECT.

La ordenación se puede hacer de forma ascendente o de forma descendente, poniendo las palabras clave ASC (ascendente) o DESC (descendente) al final de la cláusula.

Ejemplos:

```
SELECT ename, job, deptno, hiredate
FROM emp
ORDER BY hiredate DESC;
```

Utilizando un alias de columna:

```
SELECT empno, ename, sal*12 sal_anual
FROM emp
ORDER BY sal_anual;
```

Ordenando por más de una columna. Se puede ordenar por una columna no seleccionada:

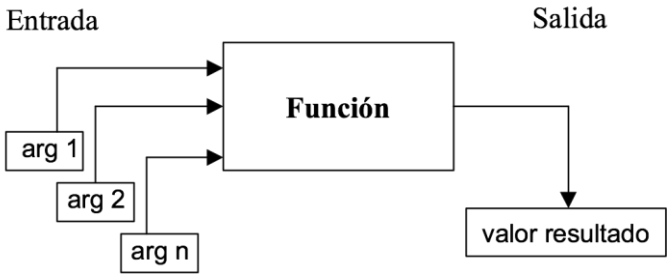
```
SELECT ename, deptno, sal
FROM emp
ORDER BY deptno, sal DESC;
```

2. Funciones a nivel de fila

Las funciones constituyen una característica muy poderosa de SQL y se pueden utilizar para:

- Realizar cálculos sobre datos
- Modificar ítems de datos individuales
- Manipular la salida de grupos de filas
- Modificar formatos de los datos para su presentación
- Convertir los tipos de datos de las columnas

Las funciones SQL aceptan argumentos y devuelven un valor.



Existen 2 tipos distintos de funciones:

- A nivel de fila
- A nivel de grupos de filas

Las funciones a nivel de fila operan sólo sobre filas y devuelven un único resultado por cada una de ellas:

- Funciones de caracteres
- Funciones de número
- Funciones de fecha
- Funciones de conversión

Las funciones a nivel de fila se usan para manipular ítems de datos. Aceptan uno o más argumentos y devuelven un valor por cada fila que recupera la consulta. Un argumento puede ser:

- ❑ Una constante suministrada por el usuario
- ❑ Un valor de variable
- ❑ Un nombre de columna
- ❑ Una expresión

Las funciones a nivel de fila pueden actuar sobre cada fila recuperada por la consulta, devuelven un resultado por cada fila, pueden devolver un dato de diferente tipo que el referenciado, pueden aceptar uno o más argumentos del usuario, se pueden anidar y se pueden usar en las cláusulas SELECT, WHERE y ORDER BY.

Sintaxis:

nombre_de_función (columna | expresión, [arg1, arg2,…])

Funciones de caracteres:

a) LOWER (columna | expresión) -> Pasa a minúsculas el campo que se le pase.

b) UPPER (columna | expresión) -> Pasa a mayúsculas el campo.

c) INITCAP (columna | expresión) -> Pasa a mayúsculas la primera letra de cada palabra y deja las demás en minúsculas.

d) CONCAT (columna1 | expresión1, columna2 | expresión2) -> Concatena las 2 cadenas. Es equivalente al operador ||

e) SUBSTR (columna | expresión, m, [n]) -> devuelve la cadena de caracteres empezando por el carácter en la

posición m hasta la n. Si n se omite (opcional), devuelve los caracteres hasta el final. Si m es negativo, el contador empieza desde el final de la cadena.

f) LENGTH (columna | expresión) -> longitud de la cadena que se le pase.

g) INSTR (columna | expresión, m) -> devuelve la posición del carácter m dentro de la cadena.

h) LPAD (columna | expresión, n, 'string') -> rellena la cadena por la IZQUIERDA con el carácter ESPECIFICADO ('string') hasta alcanzar un tamaño total de n posiciones. Esto hace que los datos de una columna sean de tamaño fijo.

- **columna | expresión**: La cadena original que quieres rellenar. COLUMNA O EXPRESION
- **n**: El tamaño total que tendrá la cadena resultante (incluyendo el relleno).
- **'string'**: El carácter o los caracteres que se usarán para rellenar la cadena por la izquierda. Si no se especifica, por defecto se usa un espacio en blanco.

·trabaja con cadenas .tienes que cambiar si te da error con funciones tipo to_char... ...

```
SELECT LPAD(codigo, 6, '0') AS codigo_fijo
FROM tabla;
```

RPAD SU INVERSA DERECHA

i) TRIM (parámetro, carácter_a_borrar FROM campo) -> sirve para eliminar caracteres del principio (leading), del final (trailing) o ambos (both).

El parámetro tiene tres valores posibles:

- ❑ leading: de una cadena de caracteres, borras el principio
- ❑ trailing: de una cadena de caracteres, borras el final
- ❑ both: de una cadena de caracteres, borras tantos el principio como el final.

Ejemplos (sacar de la tabla de empleados (emp) información como la siguiente):

- 'El trabajo de' nombre_del_empleado_en_mayúsculas 'es' 'Puesto_en_minúsculas'

```
SELECT 'El trabajo de ' || INITCAP (ename) || ' es ' || LOWER (job) FROM emp;
```

- Visualizar el numero de empleado, nombre y departamento de blake:

```
SELECT ename, empno, deptno
FROM emp
WHERE ename = UPPER ('blake');
```

- Visualizar el nombre del empleado, nombre del empleado concatenado a su puesto de trabajo, longitud del nombre y posición de la letra "A" en el nombre:

```
SELECT ename, ename ||empno, LENGTH (ename), INSTR (ename, 'A')
FROM emp;
```

Funciones numéricas:

a) ROUND (columna | expresión, [n]) -> Redondea la columna, expresión o valor a n posiciones decimales. Si se omite n, no se redondea con lugares decimales. Si n es negativo, los números a la izquierda del punto decimal se redondean.

b) TRUNC (columna | expresión, [n]) -> Trunca la columna o valor en la n-ésima posición decimal. Si se omite n, sin lugares decimales. Si n es negativo, los números a la izquierda del punto decimal se truncan a cero.

c) MOD (m, n) -> Devuelve el resto de la división de m entre n.

Ejemplos:

ROUND (45.926, 2) ➔ 45.93
TRUNC (45.926, 2) ➔ 45.92
MOD (1600, 300) ➔ 100

- Calcular el ratio (resto) del salario respecto a la comisión de cada empleado, cuyo oficio sea "SALESMAN":

```
SELECT ename, sal, comm, MOD (sal, comm)
FROM emp
WHERE job='SALESMAN';
```

- Visualizar el valor 45.923 redondeado a centenas, 0 y 10 posiciones decimales:

```
SELECT ROUND (45.923, 2), ROUND (45.923, 0), ROUND (45.923, -1)
FROM SYS.DUAL;
```

Tabla resultado devuelta:

ROUND (45.923, 2)	ROUND (45.923, 0)	ROUND (45.923, -1)
45.92	46	50

- Visualizar el valor 45.923 truncado a centenas, 0 y 10 posiciones decimales:

```
SELECT TRUNC (45.923, 2), TRUNC (45.923, 0), TRUNC (45.923, -1)
FROM SYS.DUAL;
```

Tabla resultado devuelta:

TRUNC (45.923, 2)	TRUNC (45.923, 0)	TRUNC (45.923, -1)
45.92	45	40

Funciones de fecha:

Internamente Oracle las almacena como números, que incluye: Siglo, año, mes, día, horas, minutos y segundos.

Va desde el 1 de enero de 4712 A.C. hasta el 31 de diciembre de 9999 D.C.

SELECT SYSDATE FROM SYS.DUAL; para saber la fecha actual. DUAL es una tabla virtual de la BD .

Recordemos que el formato de fecha es YYY-MM-DD.

A las fechas, se les puede aplicar operadores aritméticos (+, -):

- Fecha + Número ➔ Fecha (agrega una cantidad de días a la fecha)
- Fecha – Número ➔ Fecha (resta una cantidad de días a partir de una fecha)
- Fecha – Fecha ➔ Numero de días (resta una fecha de otra)
- Fecha + Numero/24 ➔ Fecha (agrega una cantidad de horas a la fecha)

Ejemplo: Número de semanas que llevan contratados los empleados.

```
SELECT ename, (SYSDATE - hiredate)/7
FROM emp;
```

Las funciones de fecha son las siguientes:

- ➢ MONTHS_BETWEEN ('fecha1', 'fecha2'): devuelve un número de meses entre dos fechas dadas.
- ➢ ADD_MONTHS ('fecha', n): devuelve una fecha en la que se añadirá 'n' meses.
- ➢ NEXT_DAY ('fecha', 'caracteres'): calcula la fecha posterior a la dada. Los caracteres pueden ser un número representando un día o una cadena de caracteres Ejemplo: NEXT_DAY (04-10-04, 'MONDAY') ➔ '08-10-04'
- ➢ LAST_DAY ('fecha'): te devuelve el último día del mes de la fecha especificada.
- ➢ ROUND ('fecha', [formato de fecha]): devuelve la fecha redondeada a la unidad especificada por el formato. Ejemplos de formato: 'month', 'year'.
- ➢ TRUNC ('fecha', [formato de fecha]): devuelve la fecha con la porción del día truncado en la unidad especificada por el modelo de formato de fecha. Si se omite el formato, la fecha se trunca en el día más próximo.
- ➢ CEIL: que redondean hacia arriba Funcionan parecido a Round y Trunc.
- ➢ FLOOR: que redondean hacia abajo Funcionan parecido a Round y Trunc. https://blogs.oracle.com/connect/post/from-floor-to-ceiling-and-other-functional-cases (Ejemplos de Ceil, Floor, Round Trunc)

Ejemplos:

- Comparar las fechas de contratación de todos los empleados que empezaron en 1987. Visualizar el número de empleados, fecha de contratación y el mes en que el empezaron, utilizando funciones ROUND y TRUNC.

```
SELECT empno, hiredate, ROUND (hiredate,  'month'), TRUNC (hiredate, 'month')
FROM emp
WHERE hiredate LIKE '%87';
```

- Para todos los trabajadores empleados por menos de 200 meses, visualizar número de empleado, fecha de contratación, fecha de revisión de 6 meses, primer viernes después de la fecha de contratación y el mismo día del mes en que se contrataron.

```
SELECT     empno, hiredate, ADD_MONTHS (hiredate, 6),
           NEXT_DAY (hiredate, 'VIERNES'), LAST_DAY (hiredate)
FROM emp
WHERE MONTHS_BETWEEN (sysdate, hiredate) <300;
```

Tipos y conversión de tipos:

Recordemos los tipos de datos existentes en Oracle:

- NUMBER (n, d) n: numero máximo de dígitos
  - d: numero de decimales
  - dígitos enteros = n – d

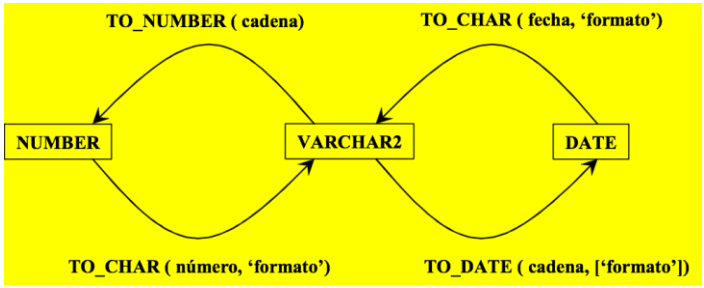
- VARCHAR2 (s) s: cadena de caracteres de tamaño máximo s (tamaño variable)
- DATE
- CHAR (s) s: cadena de caracteres de tamaño fijo s

Las conversiones de datos pueden realizarse de 2 maneras:

1. Implícitas: las hace Oracle, sin intervención del usuario.

De	***A***
VARCHAR2 o CHAR	NUMBER
VARCHAR2 o CHAR	DATE
NUMBER	VARCHAR2
DATE	VARCHAR2

2. Explícitas: las que el usuario hace explícitamente.



TO_CHAR
1. de fecha a caracteres.
2. de número a caracteres.

1. TO_CHAR (fecha, 'formato'): de fecha a caracteres
   Ejemplo: TO_CHAR (hiredate, 'MM/YY').

Formato:
a. Encerrado entre comillas simples
b. Es case sensitive Y ≠ y
c. Elementos del formato
   i. YYYY : año completo en numero
   ii. YEAR : año en letras
   iii. MM : número del mes con 2 dígitos
   iv. MONTH : nombre completo del mes
   v. DY : abreviatura de 3 letras del día de la semana
   vi. DAY : nombre completo del día
   vii. S : siglo
   viii. SCC : pone AC antes de fecha
   ix. Q : pone el trimestre del año
   x. RM : pone el mes en números romanos
   xi. J: día juliano (el numero de días desde el 31 de diciembre del 4712 a.c.)
   xii. fm : quita el relleno en blanco o los 0 a la izquierda

2. TO_CHAR (numero, 'formato'): de número a caracteres
   Ejemplo: SELECT TO_CHAR (sal, '99,999L') FROM emp;

Nota: Mirar el libro McGRAW

9: Digito
TO_NUMBER (cadena_de_caracteres): convierte de caracteres a números.

TO_DATE (cadena_de_caracteres [,formato]): cambia de caracteres a fecha
Ej.: SELECT TO_DATE ('01111996','DDMMYYYY')
FROM SYS.DUAL;

Hay un formato de fecha: RR (Como YY, pero sensible a cambios de siglo)

		Fecha especificada (2 dígitos)	
		0-49	50-99
Fecha actual (2 dígitos)	0-49	Mismo siglo	Siglo anterior
	50-99	Siglo posterior	Mismo siglo

Algunos ejemplos que utilizan este formato RR comparado con el YY:

Año Actual	**Fecha especificada**	**Formato RR**	**Formato YY**
1995	27-OCT-95	1995	1995
1995	27-OCT-17	2017	1917
2001	27-OCT-17	2017	2017
2001	27-OCT-95	1995	2095

Funciones de Oracle

https://sites.google.com/site/josepando/home/funciones-sql/funciones-que-devuelven-una-valor-nico-para-cada-fila-de-una-consulta-o-vista/funciones-de-tipo-fecha

Otras funciones de Oracle:

Función NVL: Convierte un nulo a un valor concreto

✓ Es aplicable a cualquier tipo de datos: fechas, caracteres y números.
✓ Los tipos de datos de ambas expresiones deben coincidir.

Sintaxis: *NVL (expresion1, expresion2)*

Cada vez que la *expresión1* equivale a nulo, la sustituye por la *expresion2*.

Ejemplos:

- Numérico: NVL (comm, 0)

- Fecha: NVL (hiredate, '01-ene-82')
- Cadena de caracteres: NVL (job, 'Sin trabajo')
- Calcular la compensación anual de todos los empleados, es decir, multiplicar el salario mensual por 12 y añadirle la comisión:

```
SELECT ename, sal, comm, (sal*12) + NVL (comm, 0)
FROM emp;
```

Función DECODE: hace las veces de sentencias CASE o IF-THEN-ELSE para facilitar consultas condicionales.

Esta función descifra una expresión después de compararla con cada valor de búsqueda. Si la expresión coincide con el valor de búsqueda, devuelve el resultado. Si se omite el valor por defecto (default), se devolverá un valor nulo en el caso de que la búsqueda no coincida con ninguno de los valores resultantes.

Sintaxis:

```
DECODE (columna | expresión, valor1, resultado1
                             [, valor2, resultado2, …, ]
                              [, default ]) ;
```

Interpretación:

```
IF (columna | expresión) = valor1
        THEN resultado1
IF (columna | expresión) = valor2
  THEN resultado2
IF (columna | expresión) = valor3
  THEN resultado3
ELSE resultado 4
```

Ejemplo:

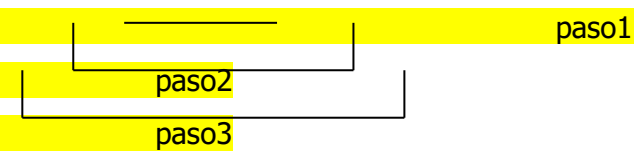
Mostrar el puesto de trabajo y el salario de los empleados de tal manera que si el puesto de trabajo (JOB) es ANALYST, el salario (SAL) se incrementa en un 10%; si JOB es CLERK, el salario se incrementa en un 15%; si JOB es MANAGER, el salario se incrementa en un 20%. El resto no percibe incremento de salario.

```
SELECT job, sal,
  DECODE (job, 'ANALYST', sal *1.1,
               'CLERK', sal *1.15,
               'MANAGER', sal*1.20,
                          sal)
FROM emp;
```

Anidamiento de funciones:

Las funciones a nivel de fila pueden ser anidadas hasta cualquier profundidad. Estas se evalúan desde el nivel más interno hasta el más externo.

F3 (F2 (F1 (col, arg1), arg2), arg3)



Ejemplos:

```
SELECT ename, NVL (TO_CHAR (mgr), ' Sin jefe')
                FROM emp
               WHERE mgr IS NULL;
```

```
SELECT TO_CHAR (NEXT_DAY (ADD_MONTHS (hiredate, 6), 'VIERNES'),
                'fmDay, Month ddth, YYYY') "Prox revision de 6 meses"
  FROM emp
 ORDER BY hiredate;
```

4. Obtención de datos de múltiples tablas

En ocasiones necesitamos utilizar datos de más de una tabla al mismo tiempo. Por ejemplo, podríamos necesitar datos que estén en la tabla EMP y en la tabla DEPT. Para obtenerlos necesitamos unir ambas tablas o, lo que es lo mismo, realizar una JOIN.

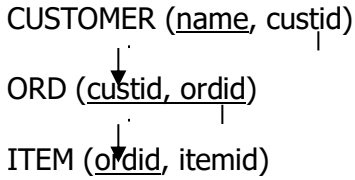
La unión o JOIN de dos tablas se realiza a través de valores comunes, esto es, con las columnas de clave extranjera y la de clave primaria. Para ello, se escribe una condición en la cláusula WHERE de la sentencia SELECT.

Sintaxis:

```
SELECT tabla1.columna1, tabla2.columna2, …
  FROM tabla1, tabla2
 WHERE tabla1.columnaFK = tabla2.columnaPK;
```

Para combinar tablas se necesita como mínimo una cantidad de condiciones de JOIN equivalentes a la cantidad de tablas menos uno. Por lo tanto, para combinar 4 tablas, se necesitarían un mínimo de 3 condiciones. **Esta regla no se puede aplicar si la tabla tiene una clave primaria combinada**, en cuyo caso se pueden requerir más de una columna para identificar unívocamente a cada fila.

Ejemplo: Supongamos el siguiente esquema relacional:




```
SELECT customer.name, ord.ordid, item.itemid
  FROM customer, ord, item
 WHERE customer.custid = ord.custid
   AND ord.ordid = i.ordid;
```

Cuando una condición de **JOIN no es válida o se omite completamente, el resultado es un producto cartesiano,** en el cual se muestran las combinaciones de todas las filas. Se combinan todas las filas de la primera tabla con todas las filas de la segunda.

Un producto cartesiano tiende a generar una gran cantidad de filas y este tipo de resultado es raramente útil. Se debería incluir una condición de JOIN válida en la cláusula WHERE, a menos que exista la necesidad específica de combinar todas las filas de todas las tablas.

Ejemplo:

```
SELECT emp.empno, emp.ename, emp.deptno, dept.deptno, dept.loc
  FROM emp, dept
 WHERE emp.deptno = dept.deptno;
```

Para evitar la ambigüedad, es necesario calificar los nombres de las columnas en la cláusula WHERE usando los nombres

de las tablas. Sin los prefijos, la columna deptno podría ser tanto de la tabla dept como de la tabla emp.

Si no existen nombres repetidos en las tablas que se unen, no es necesario calificar las columnas. Sin embargo, se ganará una mejora del rendimiento con el uso de prefijos, ya que de esta manera le decimos exactamente a Oracle a donde ir para encontrar las columnas.

Los requerimientos para calificar a los nombres de columna ambiguos también son aplicables para las columnas que también podrían ser ambiguas en otras cláusulas como SELECT u ORDER BY.

Además del JOIN, se pueden añadir criterios adicionales para la cláusula WHERE. Por ejemplo, para visualizar el número de empleado de King, el nombre, número de departamento y localidad, necesitamos una condición adicional en la cláusula WHERE.

```
SELECT empno, ename, emp.deptno, loc
    FROM emp, dept
  WHERE emp.deptno = dept.deptno
  AND ename = UPPER ('King');
```

Alias de tabla

Calificar las tablas con los nombres de tabla puede llevar mucho tiempo, especialmente si los nombres son largos. Para ello existen los alias de tabla. Así como los alias de columna dan a una columna otro nombre, los alias de tablas dan a una tabla otro nombre. Los alias de tablas ayudan a mantener reducido el código SQL, utilizando por tanto menos memoria.

Ejemplo:

```
SELECT e.empno, e.ename, e.deptno, d.deptno, d.loc
    FROM emp e, dept d
  WHERE e.deptno = d.deptno;
```

Las reglas de utilización de los alias de tabla son las siguientes:

✓ Pueden tener hasta 30 caracteres de longitud pero es preferible que sean cortos
✓ Si se usa un alias para un nombre de tabla en particular en la cláusula FROM, entonces debe ser sustituida a través de toda la sentencia SELECT.
✓ Los alias de tablas deben ser significativos
✓ El alias de tabla es válido sólo en la sentencia SELECT actual.

Otros tipos de JOINS (Mirar el anexo 1 de tipos de JOINS)

A las JOINS realizadas utilizando las claves primarias y extranjeras vistas hasta ahora se las denomina JOINS simples (EQUIJOINS o INNER JOINS), en las que los valores de las columnas comparadas son iguales.

En algunos casos las columnas a unir pueden estar en la misma tabla, correspondiéndose con relaciones reflexivas. Por ejemplo, el jefe de un empleado (MGR) puede combinarse con el número de empleado (ENAME) que supervisa.

```
SELECT trabajador.ename || ` trabaja para ' || jefe.ename as currantes
    FROM emp trabajador, emp jefe
  WHERE trabajador.mgr = jefe.empno;
```

Existen otros tipos de JOINS:

- NON-EQUIJOINS

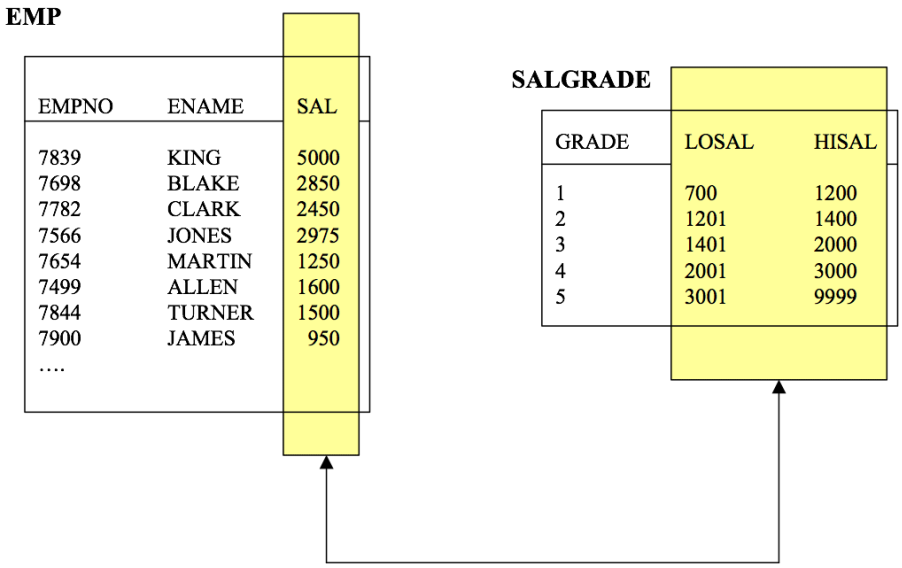
- OUTER JOINS

NON-EQUIJOINS:

Se dan **cuando no existe una correspondencia directa entre las tablas que se unen en el JOIN.** La relación se obtiene utilizando un operador distinto del igual (=).

Ejemplo:

```
SELECT e.ename, e.sal, s.grade
FROM emp e, salgrade s
WHERE e.sal BETWEEN s.losal AND s.hisal;
```



La relación entre las tablas EMP y SALGRADE es un NON-EQUIJOIN, lo cual significa que ninguna columna en la tabla EMP se corresponde directamente con una columna en la tabla SALGRADE. **La relación entre las dos tablas es que la columna SAL en la tabla EMP está entre las columnas LOSAL y HISAL de la tabla SALGRADE.**

Se produce la siguiente tabla-resultado:

ENAME	SAL	GRADE
JAMES	950	1
SMITH	800	1
ADAMS	1100	1
....		

Se está evaluando el grado de salario de un empleado, el cual debe estar entre cualquier par de los rangos de salario alto y bajo.

Es importante destacar que todos los empleados aparecen exactamente una vez cuando se ejecuta la consulta. Ningún empleado se repite en la lista. Hay dos razones para esto:

- Ninguna de las filas en la tabla del grado de salario contiene grados que se solapen, lo que se traduce en que el valor del salario de un empleado sólo puede estar entre el salario más bajo y el más alto de una de las filas en la tabla del grado del salario
- Todos los salarios de los empleados están dentro de los límites proporcionados por la tabla del grado del salario, lo que significa que ningún empleado gana menos del valor más bajo contenido en la columna LOSAL ni más del valor más alto contenido en la columna HISAL.

OUTER-JOINS:

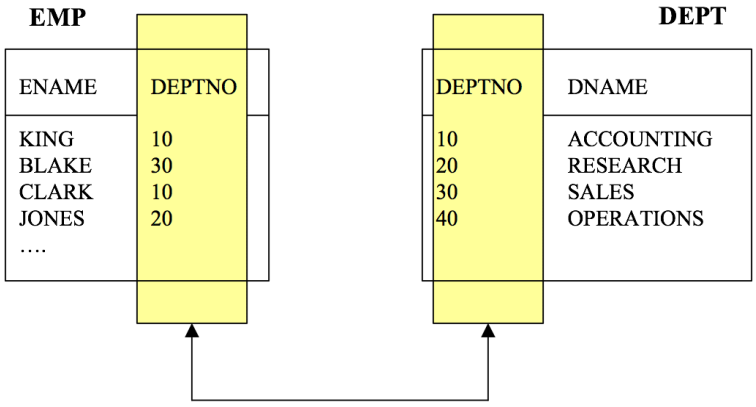
Si una fila no satisface una condición de JOIN, no aparecerá en la tabla-resultado de la consulta. Por ejemplo, si relacionamos las tablas EMP y SALGRADE por la columna DEPTNO, no aparecerá el departamento OPERATIONS porque nadie trabaja en él. La(s) filas que faltan pueden ser recuperadas si en la condición de JOIN se usa un operador de OUTER-JOIN: **(+). Este operador se sitúa en el "lado" del JOIN que es deficiente en información. Tiene el efecto de crear una o más filas NULL para que aquellas filas de la tabla sin valores coincidentes en la otra puedan ser combinadas.**

Sintaxis:

```
SELECT tabla1.columna, tabla1.columna2, … tabla2.columnax
        FROM tabla1, tabla2
    WHERE tabla1.columnai (+) = tabla2.columnaj;

SELECT tabla1.columna, tabla1.columna2, … tabla2.columnax
        FROM tabla1, tabla2
    WHERE tabla1.columnai  = tabla2.columnaj (+);
```

Ejemplo:

```
SELECT e.name, d.deptno, d.dname
    FROM emp e, dept d
  WHERE e.deptno (+) = d.deptno
    ORDER BY e.deptno;
```



Utilizando una OUTER-JOIN se produce la siguiente tabla resultado:

ENAME	DEPTNO	DNAME
KING	10	ACCOUNTING
CLARK	10	ACCOUNTING
MILLER	10	ACCOUNTING
….		
	40	OPERATIONS

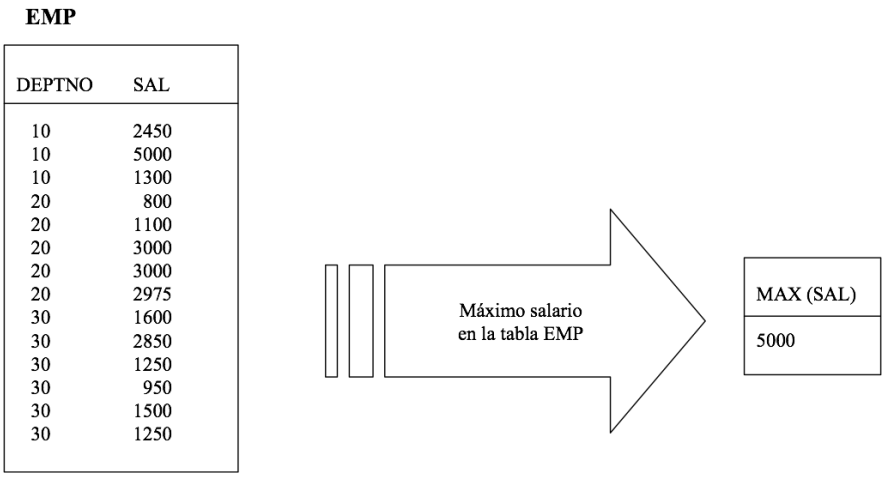
Como se observa, el departamento OPERATIONS, que no tiene ningún empleado, también se visualiza.

A la hora de utilizar una OUTER-JOIN hay que tener en cuenta las siguientes restricciones:

- El operador (+) puede aparecer sólo en un lado de la expresión, aquel lado al que le falta la información. Éste recupera aquellas filas de una tabla que no tiene correspondencia directa en otra tabla.
- **Una condición que incluye un OUTER-JOIN no puede utilizar el operador IN o unirse a otra condición por el operador OR.**

3. Datos Agregados por medio de Funciones de Grupo

A diferencia de las funciones a nivel de fila estudiadas en apartados anteriores, las ***las funciones de grupo*** operan sobre conjuntos de filas para dar un resultado por cada uno de ellos. Dichos grupos pueden estar constituidos por la tabla entera o por partes de la misma.



Funciones de grupo:

Estas funciones aceptan un argumento y se clasifican en:

- **AVG ([DISTINCT|ALL] n)**: calcula el valor promedio del argumento n, ignorando los valores nulos.
- **COUNT (* | [DISTINCT|ALL] expr)**: devuelve la cantidad de filas cuando expr no resulta en un valor nulo. Para contar las filas se usa *, el mismo incluye filas duplicadas o aquellas que contienen valores nulos.
- **MAX ([DISTINCT|ALL] expr)**: calcula el valor máximo de la expr que se le pase.
- **MIN ([DISTINCT|ALL] expr)**: calcula el valor mínimo del argumento expr pasado.
- **SUM ([DISTINCT|ALL] n)**: devuelve la suma de los valores del argumento pasado, ignorando los nulos.
- **STDDEV ([DISTINCT|ALL] n):** calcula la desviación estándar estadística de n, ignorando los valores nulos.
- **VARIANCE ([DISTINCT|ALL] n)**: calcula la varianza estadística de n, ignorando los nulos.

Uso de las Funciones de grupo:

Sintaxis:

```
SELECT [columna, ...] función_de_grupo (columna)
FROM tabla
[WHERE condición]
[GROUP BY columna1 [, columna2...]]
[ORDER BY columna1 [, columna2...]];
```

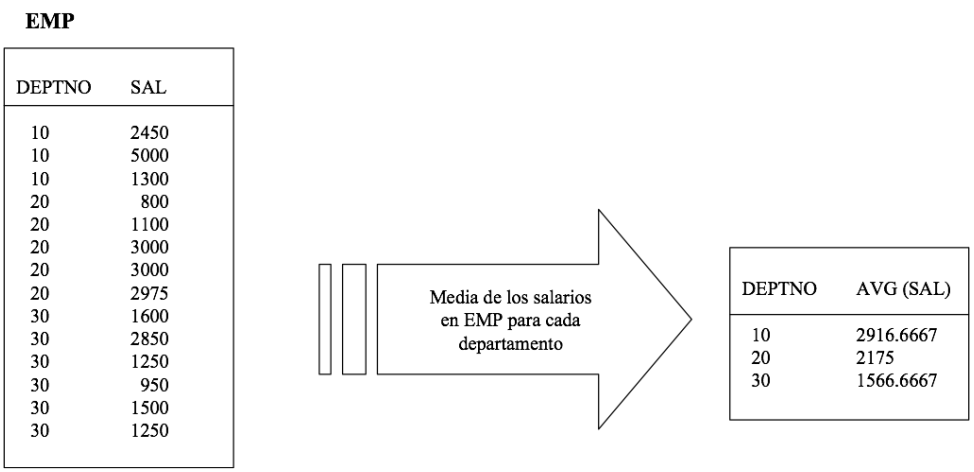
✓ El uso de DISTINCT, hace que la función no considere los valores duplicados. ALL provoca que se consideren todos los valores, incluyendo los duplicados. Por defecto, si no se especifica nada, se sobreentiende ALL.
✓ Los tipos de datos de los parámetros en pueden ser CHAR, VARCHAR2, NUMBER o DATE.
✓ Las funciones AVG, SUM, VARIANCE y STDDEV sólo pueden ser utilizadas con valores numéricos.
✓ Todas las funciones de grupo excepto COUNT(*) ignoran los valores nulos. Para sustituir NULL por un valor concreto, use la función NVL.
✓ Oracle, por defecto, ordena el resultado en orden ascendente cuando se usa una cláusula GROUP BY. Se puede realizar la ordenación descendente incluyendo la opción DESC en la cláusula ORDER BY.

Ejemplos:

```
SELECT AVG(SAL), MAX(SAL), MIN(SAL), SUM(SAL)
FROM EMP
WHERE JOB LIKE 'SALES%';

SELECT MIN(hiredate), MAX(hiredate)
FROM EMP;

SELECT COUNT(*)
FROM EMP
WHERE deptno = 30;

SELECT COUNT(comm)
FROM EMP
WHERE deptno = 30;

SELECT COUNT(deptno)
FROM EMP;

SELECT COUNT(DISTINCT(deptno))
FROM EMP;

SELECT AVG(comm)
FROM EMP;

SELECT AVG(NVL(comm,0))
FROM EMP;
```

La cláusula GROUP BY:

Hasta ahora, todas las funciones de grupo trataban la tabla como un gran grupo de información. A veces, es necesario dividir la tabla de información en grupos más pequeños, asunto que se resuelve utilizando la cláusula GROUP BY.



La cláusula GROUP BY se utiliza con el propósito de repartir las filas de una tabla en grupos más pequeños. Las funciones de grupo se pueden utilizar para devolver información resumida para cada grupo.

Para utilizar esta cláusula hay que respetar unas reglas:

Todas las columnas mendionadas en la SELECT que no son funciones de grupo, tienen que estar en la cláusula GROUP BY

- ✓ Si se incluye una función de grupo en una cláusula SELECT, no se pueden seleccionar resultados individuales a menos que la columna aparezca en la cláusula GROUP BY. Si no se incluyen las columnas correspondientes en la lista, se recibirá un mensaje de error.
- ✓ Con el uso de la cláusula WHERE se pueden excluir filas antes de la división en grupos.
- ✓ No se pueden utilizar alias de columna en la cláusula GROUP BY.
- ✓ Por defecto, las filas se ordenan en forma ascendente de acuerdo a la lista GROUP BY. Puede modificarse con ORDER BY.

Ejemplo:

```
SELECT deptno, AVG (sal)
FROM emp
GROUP BY deptno;

SELECT AVG (sal)
FROM emp
GROUP BY deptno;

SELECT deptno, AVG (sal)
FROM emp
GROUP BY deptno
ORDER BY AVG(sal);
```

Algunas veces, es necesario ver los resultados de grupos dentro de grupos. Esto se realiza incluyendo más de una columna en la cláusula GROUP BY.

Ejemplo:

```
SELECT deptno, job, SUM (sal)
FROM emp
GROUP BY deptno, job;
```

Ejemplos de consultas no válidas que utilizan funciones de grupo:

```
SELECT deptno, COUNT (ename)
FROM emp;
```

ERROR >> columna no especificada en la cláusula GROUP BY

```
SELECT deptno, COUNT (ename)
FROM emp
GROUP BY deptno;
```

CORRECTO

```
SELECT deptno, AVG (sal)
FROM emp
WHERE AVG (sal) > 2000
```

```
GROUP BY deptno;
```

ERROR >> no puede usar la cláusula WHERE para restringir grupos

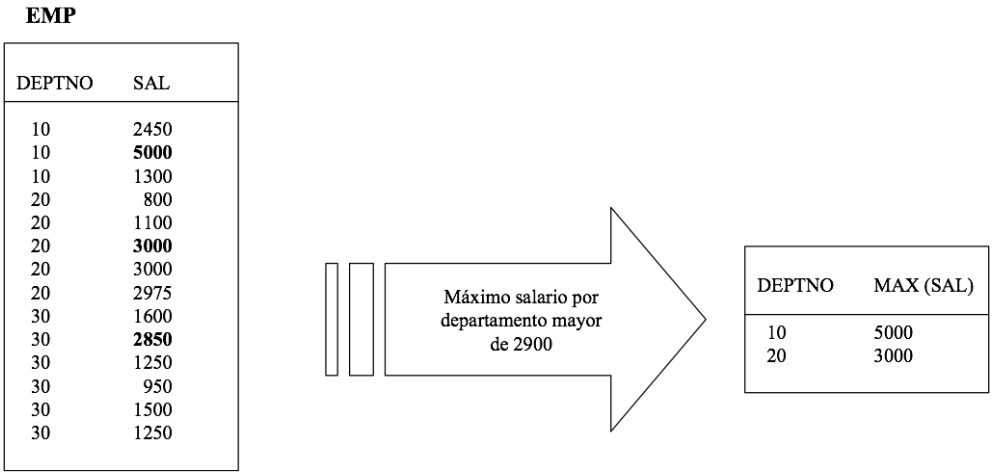
```
SELECT deptno, AVG (sal)
FROM emp
GROUP BY deptno
HAVING AVG (sal) > 2000;
```

CORRECTO

Cláusula HAVING o Exclusión de resultados de un grupo:

De la misma manera que se utiliza la cláusula WHERE para restringir los registros que se selecciona, se utiliza la cláusula HAVING para restringir grupos.

Por ejemplo, se desea localizar el máximo salario de cada departamento pero mostrar sólo aquellos cuyo máximo es superior a 2900. Para ello habrá que averiguar el máximo salario de cada departamento agrupando por nº de departamento y limitar los grupos a aquellos departamentos que superen 2900 su salario máximo.



Sintaxis:

```
SELECT [columna, …] función_de_grupo (columna)
FROM tabla
[WHERE condición]
[GROUP BY columna1 [, columna2…]]
[HAVING condición_de_grupo]
[ORDER BY columna1 [, columna2…]];
```

Ejemplos:

```
SELECT deptno, MAX (sal)
FROM emp
GROUP BY deptno
HAVING MAX (sal) > 2900;
```

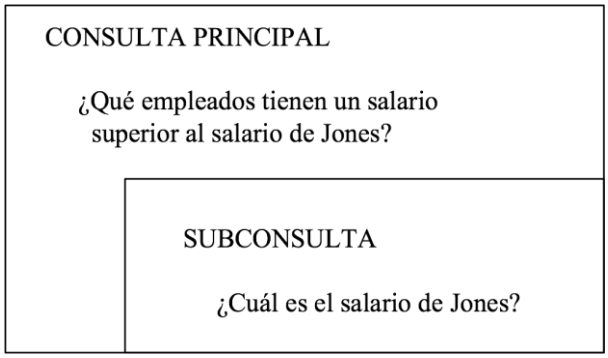
```
SELECT job, SUM (sal)
FROM emp
WHERE job NOT LIKE 'SALES%'
```

```
GROUP BY job
HAVING SUM (sal) > 5000
ORDER BY SUM (sal);

SELECT MAX (AVG (sal))
FROM emp
GROUP BY deptno;
```

4. Subconsultas

Supongamos que se quiere escribir una consulta para localizar aquellos empleados que gana un salario superior al de Jones. Para realizar este problema, se necesitan 2 consultas:

- Una que averigüe el salario de Jones
- Otra que encuentre a los empleados que ganan más de esta cantidad



La consulta más interna o subconsulta devuelve una valor que será utilizado por la consulta principal (la consulta más externa). Usar una subconsulta es equivalente a realizar 2 consultas secuencialmente y utilizar el resultado de la primera que se ejecuta (subconsulta) como valor para la segunda consulta que se ejecuta o consulta principal.

Sintaxis:

```
SELECT columna1, ..., columnax
FROM tabla
WHERE expresión operador (SELECT columna1, ..., columnaz
                          FROM tabla);
```

Una subconsulta es una sentencia SELECT que está incluida en una cláusula de otra sentencia SQL. Se pueden construir comandos simples y potentes usando subconsultas. **Pueden ser útiles cuando se necesita seleccionar filas de una tabla con una condición que depende de los datos que están en la misma tabla.**

Se puede poner una subconsulta en cláusulas WHERE, HAVING y FROM de una SELECT o un DELETE.

El operador que precede a la subconsulta es un operador de comparación.

Ejemplo:

```
SELECT ename
FROM emp
WHERE sal > (SELECT sal
             FROM emp
             WHERE empno = 7566);
```

Para usar subconsultas se deben contemplar unas reglas:

- ✓ Encierre las subconsultas entre paréntesis
- ✓ Coloque la subconsulta a la derecha del operador
- ✓ No agregar una cláusula ORDER BY a una subconsulta. Se puede tener solamente una cláusula ORDER BY para una sentencia SELECT y específicamente debe ser la última cláusula en la sentencia SELECT principal.
- ✓ Para las subconsultas se utilizan 2 clases de operadores: a nivel de fila para subconsultas que devuelven sólo una fila (Operadores mono-registro) y a nivel de grupo para subconsultas que devuelven más de una fila (Operadores multi-registro).

OPERADORES MONO-REGISTRO	**OPERADORES MULTI-REGISTRO**
= > >= < <= <>	IN: igual a los valores de cierta lista ANY: compara los valores con cada valor devuelto por la subconsulta ALL: compara los valores con cada uno de los valores devueltos por la subconsulta

Ejemplos de subconsultas mono-registro:

```
SELECT ename, job
FROM emp
WHERE job = (SELECT job
                                                    FROM emp
                                                WHERE empno = 7369);

SELECT ename, job
FROM emp
WHERE job = (SELECT job
                                                    FROM emp
                                                WHERE empno = 7369)
AND sal > (SELECT sal
           FROM emp
           WHERE empno = 7876);

SELECT ename, job, sal
FROM emp
WHERE sal = (SELECT MIN(sal)
```

```
                                                    FROM emp);

SELECT deptno, MIN (sal)
FROM emp
GROUP BY deptno
HAVING MIN (sal) > (SELECT MIN (sal)
                                                        FROM emp
                                                        WHERE deptno = 20);
```

Puesto de trabajo con la media más baja de salarios:

```
SELECT job, AVG (sal)
FROM emp
GROUP BY job
HAVING AVG (sal) = (SELECT MIN (AVG (sal))
                                                        FROM emp
                                                        GROUP BY job);
```

Consultas erróneas:

```
SELECT empno, ename
FROM emp
WHERE sal = (SELECT MIN(sal)
                                                    FROM emp
                                                GROUP BY deptno);
```

>> ERROR: la subconsulta devuelve más de una fila (el operador = sólo admite un valor). Se resolvería si se utilizara IN (operador multiregistro).

```
SELECT ename, job
FROM emp
WHERE job = (SELECT job
                                                    FROM emp
                                            GROUP BY ename = 'SMYTHE');
```

>> No se devuelve ninguna fila pues SMYTHE no existe. La consulta principal recibe un NULL de la subconsulta y no existe ningún empleado que tenga empleo nulo.

Ejemplos de subconsultas multi-registro:

```
SELECT ename, sal, deptno
FROM emp
WHERE sal IN (SELECT MIN (sal)
                                                    FROM emp
                                                GROUP BY deptno);

SELECT empno, ename, job
FROM emp
WHERE sal < ANY (SELECT sal
                                                        FROM emp
                                                        WHERE job = 'CLERK');
```

Nota:

< ANY significa menos que el máximo
> ANY significa más que el mínimo

= ANY es equivalente a IN

```
SELECT empno, ename, job
FROM emp
WHERE sal > ALL (SELECT AVG (sal)
                 FROM emp
                 GROUP BY deptno);
```

Nota:

> ALL significa más que el máximo
> ALL significa menos que el mínimo

El operador NOT puede ser utilizado con los operadores IN, ANY y ALL.

5. OPERADORES DE UNION, INTERSECT Y MINUS

- Son operadores de conjuntos , por tanto se utilizaran entre dos select que serán las encargadas de seleccionar los dos conjuntos

  Select… from …where
  Operador_de_conjunto
  Select …from..where

- UNION: Combina los resultados de las dos consultas. Elimina los valores repetidos

  Select… from …where
  UNION
  Select …from..where

  Select nombre from alumnos UNION select nombre from nuevos;

- UNION ALL : Visualiza también las filas duplicadas
- INTERSECT: Devuelve las filas que sean comunes en ambas tablas

  Select… from …where
  INTERSECT
  Select …from..where

- MINUS: Devuelve las filas que están en la primera select y no están en la segunda

  Select… from …where
  MINUS
  Select …from..where

  A veces los programadores prefieren utilizar los operadores IN,AND, OR en lugar de los vistos anteriormente.

  Select nombre from alumnos where nombre in (select nombre from nuevos)
  En este caso anterior estoy haciendo una intersección

- <u>Reglas de utilización de operadores de conjuntos</u>

  1. Las columnas de las dos consultas se relacionan en orden , de izquierda a derecha
  2. Los nombres de columna de la primera select no tiene porque ser los mismos que los nombres de la segunda select

3. La select necesitan tener el mismo número de columnas
4. Los tipos de datos deben coincidir, aunque la longitud no tiene porque ser la misma.

6.Manipulación de datos

El lenguaje de manipulación de datos o DML (*Data Manipulation Language*) es una parte esencial de SQL. Cuando se quiere agregar, actualizar o eliminar datos, se ejecuta una sentencia DML. Un conjunto de sentencias DML que aún no se han hecho permanentes se denomina **transacción** o unidad lógica de trabajo.

Imagine una base de datos de un banco. Cuando un cliente del banco transfiere dinero de una cuenta a otra, la transacción consiste en tres operaciones separadas: disminuir el saldo, incrementar el de la otra cuenta y registrar la transacción. Oracle debe garantizar que las tres sentencias SQL se ejecutan para mantener las cuentas con el saldo adecuado. Cuando una de las sentencias no puede ejecutarse, el resto de las sentencias deben deshacerse.

La sentencia INSERT

Permite añadir nuevos registros a una tabla.

Sintaxis:

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

Mediante esta sintaxis, sólo se inserta un registro o fila al mismo tiempo.

Ejemplo:

```
INSERT INTO DEPT (deptno, dname, loc)
VALUES (50, 'DESARROLLO', 'MADRID');
```

Es preciso listar los valores en el orden por defecto de las columnas de la tabla.

Opcionalmente se pueden listar las columnas en las cláusula INSERT.

Si se desean insertar valores nulos, se pueden utilizar dos métodos:

- Método implícito: omitir la columna en la lista.

```
INSERT INTO DEPT (deptno, dname)
VALUES (60, 'MI DEPARTAMENTO');
```

- Método explícito: especificar la palabra clave NULL.

```
INSERT INTO DEPT
VALUES (60, 'MI DEPARTAMENTO', NULL);
```

Antes hay que asegurarse que la columna permite valores nulos (comprobarlo con DESCRIBE).

Ejemplos:

```
INSERT INTO EMP (empno, ename, job, mgr, hiredate, sal, comm, deptno)
VALUES (7196, 'HIGUERAS', 'SALESMAN', 7782, SYSDATE, 2000, NULL, 10);

INSERT INTO EMP
VALUES (2296, 'ORTEGA', 'SALESMAN', 7782,
        TO_DATE ('FEB 3, 97', 'MON DD, YY'), 1300, NULL, 10);
```

Copia de registros de otra tabla

Se pueden usar la sentencia INSERT para agregar filas a una tabla donde los valores se derivan de otras tablas ya existentes. En lugar de la cláusula VALUES, utilizamos una subconsulta.

```
INSERT INTO JEFES (id, nombre, salario, alta)
SELECT empno, ename, sal, hiredate FROM EMP WHERE job = 'MANAGER';
```

Cuidar la concordancia de columnas de la cláusula INSERT y las de la subconsulta.

La sentencia UPDATE

Permite modificar los registros existentes. Puede afectar a más de un registro al mismo tiempo.

Sintaxis:

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

Los registros a modificar se especifican en la cláusula WHERE.

```
UPDATE EMP
```

```
SET deptno = 20
WHERE empno = 7782;
```

Si se omite la cláusula WHERE, se modificarán **todos** los registros de la tabla.

```
UPDATE EMPLEADOS
SET deptno = 20;
```

Ejemplos:

```
UPDATE EMP
SET (job, deptno) =    (SELECT job, deptno
                  FROM EMP
                  WHERE empno =7499)
WHERE empno = 7698;
```

```
UPDATE EMP
SET deptno = 55
WHERE deptno = 10;
```

SQL>> ERROR: violación de restricción de integridad (el departamento 55 no existe en la tabla padre DEPT).

La sentencia DELETE

Permite borrar registros de una tabla.

Sintaxis:

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

Los registros a eliminar se especifican en la cláusula WHERE.

```
DELETE FROM DEPARTAMENTOS
WHERE dname = 'DEVELOPMENT';
```

Si se omite la cláusula WHERE se borrarán **todos** los registros de la tabla.

```
DELETE FROM DEPARTAMENTOS;
```

Ejemplos:

```
DELETE FROM EMPLEADOS
WHERE deptno = (SELECT deptno
                  FROM DEPT
                  WHERE dname = 'SALES');
```

```
DELETE FROM DEPT
WHERE deptno = 10;
```

SQL>> ERROR: violación de restricción de integridad (no se puede eliminar un registro que contiene una clave primaria, usada como clave extranjera en otra tabla).

7.Transacciones en la base de datos

El servidor Oracle asegura la consistencia de los datos basándose en transacciones. Las transacciones otorgan más

flexibilidad y control cuando cambian los datos y aseguran en los datos ante un fallo eventual en el proceso del usuario o un fallo del sistema.

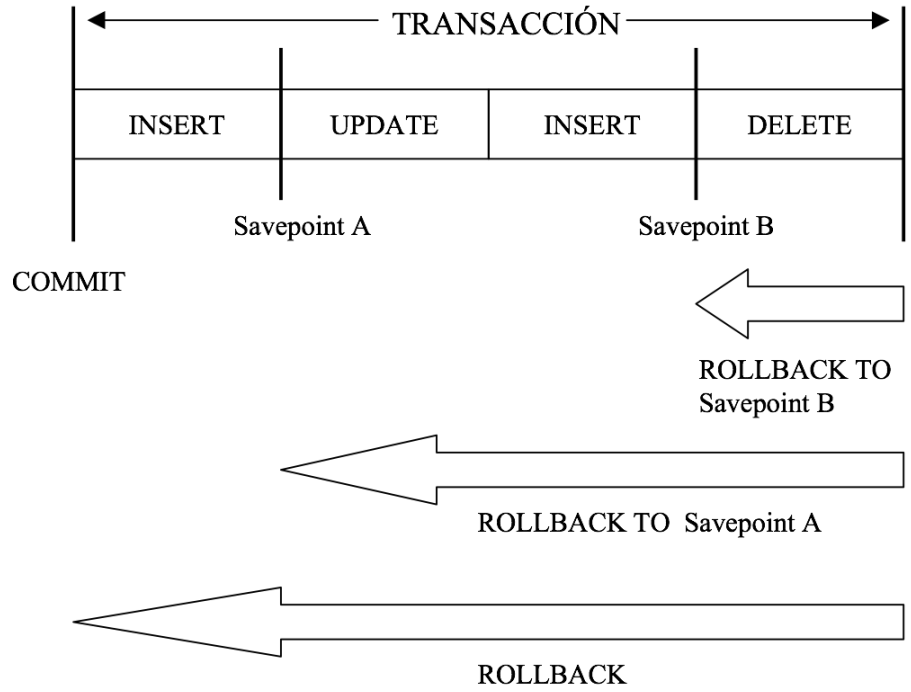
Las transacciones consisten de comandos DML que llevan a cabo un cambio en los datos de manera consistente. Por ejemplo, una transferencia de fondos entre dos cuentas, debiera incluir un crédito en una cuenta y un débito en la otra por la misma cantidad. Ambas acciones debieran tener éxito o fallar como una sola operación. El crédito no debería confirmarse sin el débito correspondientes.

Tipos de transacciones:
- Un conjunto de sentencias DML que Oracle tratará como una unidad simple
- Una única sentencia DDL
- Una única sentencia DCL

Las transacciones comienzan cuando se ejecuta el primer comando SQL.

Las transacciones finalizan con uno de los siguientes eventos:
- ➢ COMMIT o ROLLBACK
- ➢ Ejecución de un comando DCL o DCL (COMMIT automático)
- ➢ Salida del usuario
- ➢ Caída del sistema

Las sentencias COMMIT y ROLLBACK aseguran la consistencia de los datos, permiten visualizar los cambios sobre los datos antes de hacerlos permanentes y agrupan lógicamente tareas relacionadas entre sí.



- ➢ *COMMIT*: finaliza la transacción actual haciendo permanente todos los cambios
- ➢ *SAVEPOINT nombre*: Marca un Savepoint en la transacción actual
- ➢ *ROLLBACK [TO SAVEPOINT nombre]*: ROLLBACK finaliza la transacción actual deshaciendo todos los cambios pendientes. ROLLBACK TO SAVEPOINT deshace la transacción actual hasta el Savepoint especificado, deshaciendo por lo tanto el Savepoint y todos los cambios posteriores. Si se omite esta cláusula, la sentencia ROLLBACK deshace la transacción entera.

Se produce un COMMIT automático cuando se aplica un comando DDL o DCL o una salida normal desde SQL*Plus (sin realizar un COMMIT o ROLLBACK explícitamente) con el comando quit.

SQL*Plus dispone de un tercer comando: AUTOCOMMIT, que puede tomar valores ON u OFF. Si es ON, cada sentencia DML individual es confirmada (se le hace COMMIT) tan rápido como es ejecutada. Los cambios no son reversibles (sin ROLLBACK).

Se produce un ROLLBACK automático cuando ocurre una terminación anormal de SQL*Plus o un fallo del sistema. Esto previene que el error cause cambios no deseados a los datos y devuelve a las tablas su estado inicial desde el último COMMIT. De esta manera, SQL*Plus protege la integridad de las tablas.

Estado de los datos antes de COMMIT o ROLLBACK

- El estado previo de los datos puede ser recuperado.
- El usuario actual puede revisar los resultados de sus operaciones DML usando la sentencia SELECT.
- Otros usuarios no pueden ver los resultados de las sentencias DML ejecutadas por el usuario actual.
- Las filas afectadas son bloqueadas; otros usuarios no pueden cambiar los datos pertenecientes a esas filas.

Estado de los datos después de COMMIT

- Los cambios se hacen permanentes en la base de datos.
- Los datos anteriores se pierden definitivamente.
- Todos los usuarios pueden ver los resultados.
- Se liberan los bloqueos aplicados a las filas afectadas; esas filas están ahora disponibles para que otros usuarios las manipulen.
- Se borran todos los SAVEPOINTS.

Ejemplos:

```
UPDATE EMP SET deptno = 10 WHERE empno = 7782;
INSERT INTO DEPT VALUES (60, 'MARKETING', 'BARCELONA');
COMMIT;

DELETE FROM TEST;
ROLLBACK;
DELETE FROM TEST WHERE ID = 100;
SELECT * FROM TEST WHERE ID = 100;
COMMIT;
```

Hacer ROLLBACK hasta una marca

Se puede crear una marca en una transacción usando el SAVEPOINT. En consecuencia, la transacción puede dividirse en dos secciones más pequeñas. Esto permite que se descarten los cambios hechos hasta la marca usando la sentencia ROLLBACK TO SAVEPOINT.

Si se crea un segundo SAVEPOINT con el mismo nombre que un SAVEPOINT anterior, este último es eliminado.

Ejemplo:

```
UPDATE DEPARTAMENTOS SET loc = 'MIAMI' WHERE deptno = 40;
SAVEPOINT update_hecho;
INSERT INTO DEPARTAMENTOS VALUES (80, 'PUBLICIDAD', 'PARIS');
ROLLBACK TO update_hecho;
```

- Si una única sentencia DML falla durante su ejecución, entonces se hace ROLLBACK sobre esa sentencia solamente.

- El usuario debería terminar explícitamente las transacciones usando una sentencia COMMIT o ROLLBACK.

Consistencia en Lectura

Los usuarios de la base de datos realizan dos tipos de acceso a la base de datos:
- Operaciones de lectura (sentencia SELECT)
- Operaciones de escritura (sentencias INSERT, UPDATE, DELETE)

La consistencia en lectura es necesaria para que:
- Las lecturas y escrituras a la base de datos aseguren una vista consistente de los datos
- Las lecturas no vean datos que están en proceso de cambio
- Las escrituras aseguren que los cambios a la base de datos se hacen de forma consistente
- Los cambios realizados por una escritura no crean conflictos con los cambios que otra escritura está realizando

El propósito de la consistencia en lectura es asegurar que cada usuario ve los datos tal como están desde el último COMMIT, antes de que comience una operación DML.

Implementación de la consistencia en lectura

La consistencia en lectura es una implementación automática. Mantiene una copia parcial de la base de datos en segmentos de rollback. Cuando una operación DML se realiza contra la base de datos, el servidor Oracle hace una copia de los datos antes de su cambio y la escribe en un segmento de rollback.

Todas las lecturas, excepto aquella que realiza el cambio, ven la base de datos tal como estaba; ven la imagen de los segmentos de rollback, que es como una foto de los datos.

Antes de hacer el COMMIT de los cambios en la base de datos, sólo el usuario que modifica los datos ve los cambios; el resto lee la imagen del segmento de rollback. Esto garantiza que las lecturas son consistentes. Al hacer COMMIT, el cambio realizado a la base de datos se convierte en visible a cualquiera que ejecute una sentencia SELECT.

Si la transacción experimenta un rollback, los cambios "se deshacen":
- La versión original, antigua de los datos, en el segmento de rollback se escriben de nuevo a la tabla
- Todos los usuarios ven la base de datos tal como estaba antes de comenzar la transacción.

Bloqueos

Los bloqueos son mecanismos que previenen conflictos entre transacciones, que acceden a los mismos recursos, bien sea un objeto de usuario (como tablas o registros), o un objeto del sistema no visible a los usuarios (como estructuras de datos compartidas y registros del diccionario de datos).

El bloqueo en Oracle es completamente automático y no requiere acción por parte del usuario. El bloqueo implícito ocurre para todas las sentencias SQL. El bloqueo por defecto de Oracle es un mecanismo automático que utiliza el nivel más bajo aplicable de restricción, portano, ofrece el grado más alto de concurrencia y máxima integridad de datos. Oracle también permite que el usuario bloquee los datos manualmente.

Oracle utiliza dos modelos de bloqueo:
- Exclusivo: previene la compartición de un recurso. La primera transacción bloquea un recurso exclusivamente, es la única que puede alterarlo, hasta liberar el recurso.
- Compartido: permite la compartición de un recurso. Múltiples usuarios leyendo datos pueden compartir los datos, manteniendo bloqueos para prevenir acceso concurrente por una escritura (que necesita un bloqueo exclusivo). Varias transacciones pueden adquirir bloqueos compartidos sobre el mismo recurso.

PARTE 6 INTRODUCCION

- PL/SQL es la extensión estructurada y procedimental (permite crear funciones y procedimientos) del lenguaje SQL implementada por Oracle junto a la versión 6.

- Con los scripts de SQL se tienen limitaciones como uso de variables, modularidad,etc.

- Con PL/SQL se pueden usar sentencias SQL para acceder a bases de datos Oracle y sentencias de control de flujo para procesar los datos, se pueden declarar variables y constantes, definir procedimientos, funciones, subprogramas, capturar y tratar errores en tiempo de ejecución, etc...

- En un programa escrito en PL/SQL se pueden utilizar sentencias SQL de tipo LMD (Lenguaje de Manipulación de Datos) directamente y sentencias de tipo LDD (Lenguaje de Definición de Datos) mediante la utilización de paquetes.

- Oracle incorpora un gestor PL/SQL en el servidor de bbdd y en las principales herramientas, Forms , Reports, Graphics, etc.

- Basado en Ada incorpora todas las características propias de los lenguajes de tercera generación; manejo de variables, estructura modular (procedimientos y funciones) , estructuras de control, control de excepciones,etc

- El código PL/SQL puede estar almacenado en la base de datos (procedimientos, funciones, disparadores y paquetes) facilitando el acceso a todos los usuarios autorizados.

- La ejecución de los bloques PL/SQL puede realizarse interactivamente desde herramientas como SQL*Plus, Oracle Forms, etc..., o bien, cuando el S.G.B.D. detecte determinados eventos, también llamados disparadores.

- Nosotros vamos a utilizarlos en SQL Worksheet de Oracle Live

SQL para poder probar nuestro Pl/SQL

- Los programas se ejecutan en el servidor, con el consiguiente ahorro de recursos en los clientes y disminución de tráfico en la red.

ARQUITECTURA DE PL/SQL

La arquitectura de PL/SQL consiste principalmente en los tres componentes que se citan a continuación:

1. El bloque PL/SQL
2. El motor PL/SQL
3. El servidor de la base de datos

El bloque PL/SQL:

- Es el componente que contiene el código de pl/sql
- Consiste en distintas secciones que forman la lógica del bloque
- También contiene instrucciones de sql que interactuar con el servidor de base de datos
- Hay diferentes tipos de bloques o Units de Pl/sql y son :

1. Bloques anonimos
2. Procedimientos
3. Paquetes
4. Trigger

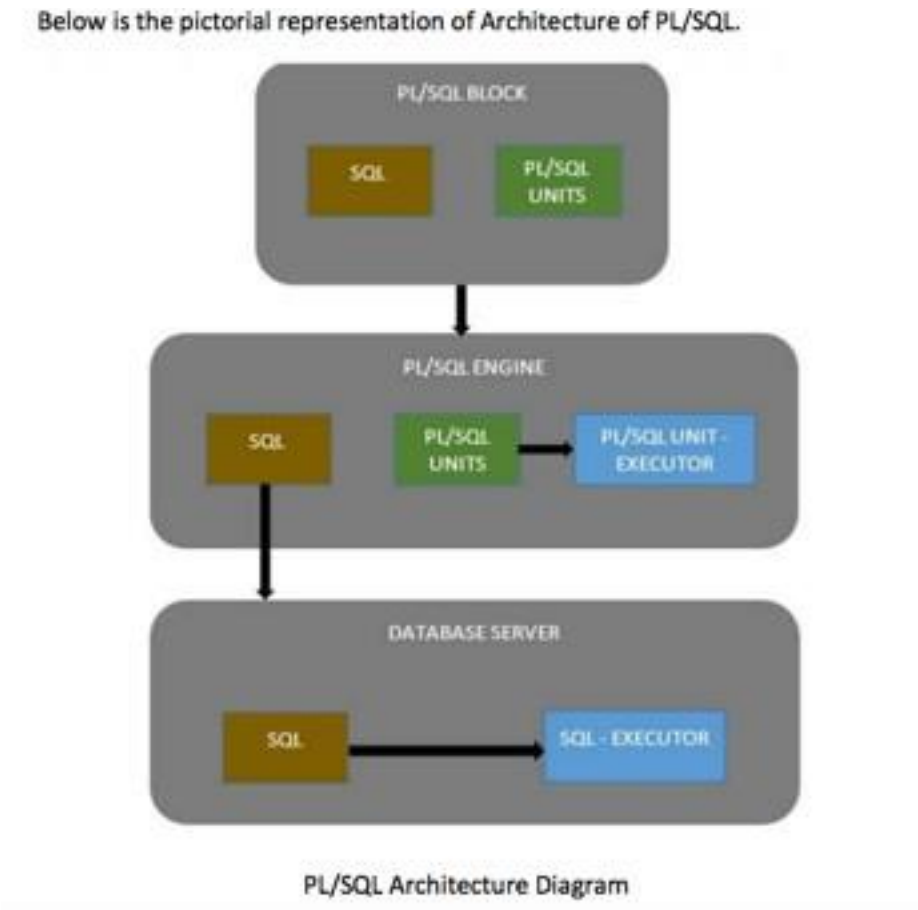
El motor PL/SQL

Es el componente donde se procesa el código
El motor PL/SQL separa las unidades de PL/SQL y las partes de SQL y este motor será el encargado de manejar los bloques PL/SQL

La parte de SQL será enviada al servidor donde interactua con la base de datos

El servidor de base de datos

Es la parte más importante pues es la que contiene los datos
El motor de The PL/SQL usa el SQL de los bloques PL/SQL para interactuar con el servidor de base de datos



PL/SQL Architecture Diagram

Diferencias entre SQL y PL/SQL

sql	pl/sql
SQL es una consulta sencilla usada para desarrollar operaciones DML and DDL.	PL/SQL es un bloque de codigo usado para escribir programas enteros con procedimientos, bloques y funciones
Es declarativo, es decir, se dice que cosas queremos más que como se hacen	Es prodedimental, por tanto, decimos como se hacen las cosas.
Ejecuta una sentencia sencilla	Ejecuta todo un bloque
Interactua con el servidor de la base de datos	No interactua con les hervor de base de datos
No puede contener código PL/SQL	Contiene SQL en sus bloques

CARACTERÍSTICAS DEL LENGUAJE. BLOQUES

En PL/SQL, tel código no es ejecutado como una linea, siempre es ejecutado como un grupo de sentencia denominado BLOQUE.

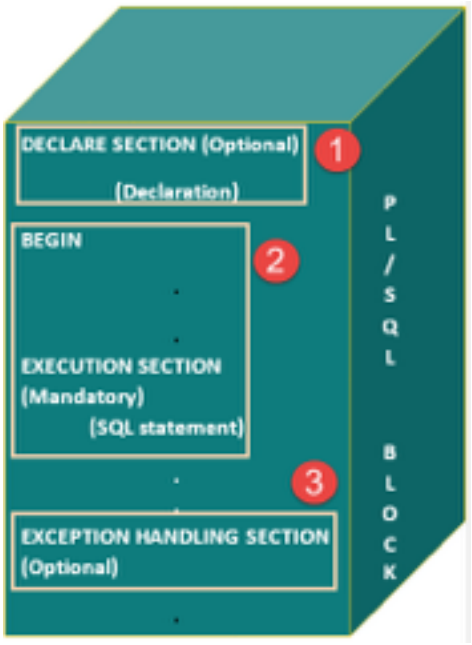
Los bloques contienen instrucciones tanto de PL/SQL como de SQL.
La estructura básica del lenguaje es el bloque.

Todos los programas se escriben en bloques de código que se dividen en diferentes secciones independientes.

Se pueden anidar bloques formando estructuras de programación más complejas

BLOQUES ANÓNIMOS: La estructura de un bloque sin nombre es la siguiente:

```
[DECLARE
      Declaraciones;]
BEGIN
      Sentencias;
  [EXCEPTION
      Excepciones;]
END;
```



- El bloque consta de tres secciones:

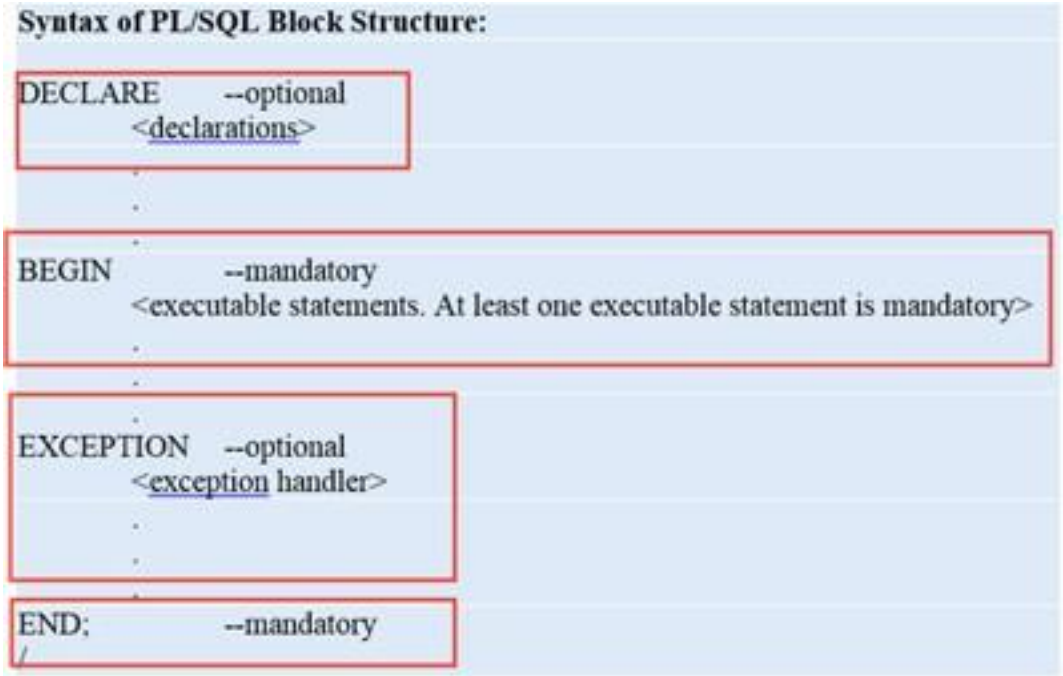
  DECLARE: Sección de declaración. En ella se declaran todos los objetos que vayamos a usar en el programa
  Es opcional, salvo cuando se deba realizar algún tipo de declaraciones de

elementos, tales como variables, constantes, cursores, etc…
Si se pone esta sección debe ser la primera.
Solo se utiliza en Bloques anónimos y en triggers pero los procedimientos no tienen esta parte Declare.

BEGIN:
Sección que contiene las sentencias que se van a ejecutar.
Es la única parte obligatoria , ya que es la parte ejecutable. Comienza con la palabra BEGIN y finaliza con END;
En esta sección es donde especificamos el código que se va a ejecutar

EXCEPTION:
Sección que contiene las sentencias para el manejo de errores.
También contiene sentencias PL/SQL .
Es opcional, salvo cuando tengamos la necesidad de tratar los errores.
Normalmente se utiliza para capturar errores ya predefinidos y hay que especificar las acciones a realizar en caso de que se produzca un error.

Por último se especifica END que indica final del bloque. Especifica final de la zona ejecutable aunque vaya después de la zona de excepciones cuando la incluya.



- Los bloques pueden contener sub-bloques, es decir, podemos tener bloques anidados. Las anidaciones se pueden realizar en la parte ejecutable y en la de manejo de excepciones, pero no en la declarativa.

```
[DECLARE
      Declaraciones;]
BEGIN

      [DECLARE
            Declaraciones;]
      BEGIN
            Sentencias:
             [EXCEPTION

            Excepciones;]

      END;

 [EXCEPTION

      Excepciones;]

END;
```

- A continuación tenemos un ejemplo de bloque PL/SQL:

```
DECLARE
sueldo NUMBER(8);
BEGIN
SELECT salario INTO sueldo
      FROM plantilla
      WHERE apellido LIKE 'MORENO';

IF sueldo < 100000 THEN
      sueldo := sueldo + 20000;
END IF;
UPDATE plantilla
      SET salario = sueldo WHERE apellido LIKE 'MORENO';
COMMIT;
END;
/

EJEMPLO UPDATE CON NULL
UPDATE EMP
SET SAL = NVL(SAL, 0) * (1 + X_PORCENTAJE / 100)
WHERE EMPNO = N_EMPLEADO;
CONTROLAR NULLCON VL
```

- Con este programa actualizamos el sueldo del empleado con apellido 'MORENO', incrementándolo en 20.000 en caso de que dicho sueldo no supere 100.000 pesetas.

- El programa tiene parte declarativa, en la que se define una variable, y parte ejecutable.

- Podemos observar que hay sentencias SQL, como SELECT y UPDATE, sentencias de control como IF... THEN y sentencias de asignación. El símbolo ':=' representa el

operador de asignación.

- El bloque de código se ejecuta al encontrar el operador de ejecución '/'.

TIPOS DE BLOQUES

Se reconocen dos tipos de bloques en PL/SQL:
- Bloques anónimos: Bloques sin nombre
- Bloques nombrados: Que a su vez se subdividen en procedimientos y funciones.

Bloques anonimos:

Son bloques que no tienen nombre y necesitan construirse en una sesión para probar algún código.
Al no tener nombre no serán almacenados en la base de datos.

Son escritos y ejecutados directamente y se compilan y ejecutan en el mismo proceso.

Bloques nombrados:

Los bloques con nombre son almacenados como objetos en la base de datos.

Como son guardados en el servidor , podremos invocarlos y ejecutarlos cuando lo necesitemos.

La compilación se produce cuando los creamos y después se ejecutaran en caso de ser necesario.

Características de estos bloques:
- Pueden ser invocados por otros bloques
- Pueden tener otros bloques anidados como dijimos antes
- Se dividen en Procedimientos y Funciones, que se verán mas adelante.

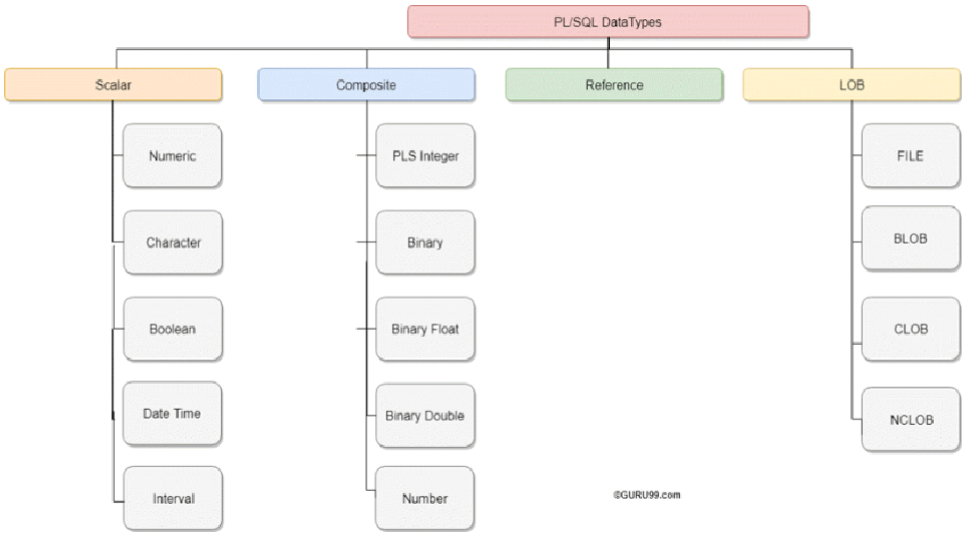
TIPOS DE DATOS

¿Qué son los tipos de datos PL/SQL?

Tipos de datos en PL/SQL se utilizan para definir cómo los datos serán almacenados, manejados y tratados por Oracle durante el almacenamiento y procesamiento de datos. Los tipos de datos están asociados con el formato de almacenamiento específico y las restricciones de rango. En Oracle, a cada valor o constante se le asigna un tipo de datos.

La principal diferencia entre PL/SQL y SQL tipos de datos es, el tipo de datos SQL se limita a la columna de la tabla, mientras que los tipos de datos PL/SQL se utilizan en la bloques PL/SQL

A continuación se muestra el diagrama de diferentes Oracle Tipos de datos PL/SQL:



Diferentes tipos de datos en PL/SQL

Tipo de datos de carácter PL/SQL

Este tipo de datos básicamente almacena caracteres alfanuméricos en formato de cadena.

Los valores literales siempre deben estar entre comillas simples al asignarlos al tipo de datos CHARACTER.

Este tipo de datos de carácter se clasifica además de la siguiente manera:

- CHAR Tipo de datos (tamaño de cadena fijo)
- VARCHAR2 Tipo de datos (tamaño de cadena variable)
- VARCHAR Tipo de datos
- NCHAR (tamaño de cadena fijo nativo)

Tipo de datos	Descripción	Sintaxis
CHAR	Este tipo de datos almacena el valor de la cadena y el tamaño de la cadena se fija en el momento de declarar el variable. • Oracle La variable se rellenaría en blanco si la variable no ocupara todo el tamaño que se ha declarado para ella, por lo tanto Oracle asignará la memoria para el tamaño declarado incluso si la variable no la ocupó por completo. • La restricción de tamaño para este tipo de datos es de 1 a 2000 bytes. • El tipo de datos CHAR es más apropiado para usar siempre que se maneje el tamaño de datos fijo.	grade CHAR; manager CHAR (10):= 'guru99'; **Explicación de sintaxis:** • La primera declaración declaró la variable "grado" del tipo de datos CHAR con el tamaño máximo de 1 byte (valor predeterminado). • La segunda declaración declaró la variable 'administrador' del tipo de datos CHAR con el tamaño máximo de 10 y asignó el valor 'guru99' que es de 6 bytes. Oracle asignará la memoria de 10 bytes en lugar de 6 bytes en este caso.
VARCHAR2	Este tipo de datos almacena la cadena, pero la longitud de la cadena no es fija. • La restricción de tamaño para este tipo de datos es de 1 a 4000 bytes para el tamaño de la columna de la tabla y de 1 a 32767 bytes para las variables. • El tamaño se define para cada variable en el momento de la declaración de la variable. • Pero Oracle asignará memoria sólo después de que se defina la variable, es decir, Oracle considerará solo la longitud real de la cadena que está almacenada en una variable para la asignación de memoria en lugar del tamaño que se ha dado para una variable en la parte de declaración. • Siempre es bueno utilizar VARCHAR2 en lugar del tipo de datos CHAR para optimizar el uso de la memoria.	manager VARCHAR2(10) := ‘guru99'; **Explicación de sintaxis:** • La declaración anterior declaró la variable 'administrador' del tipo de datos VARCHAR2 con el tamaño máximo de 10 y le asignó el valor 'guru99' que es de 6 bytes. Oracle asignará memoria de sólo 6 bytes en este caso.
VARCHAR	Esto es sinónimo del tipo de datos VARCHAR2.	manager VARCHAR(10) := ‘guru99'; **Explicación de sintaxis:**

Tipo de datos	Descripción	Sintaxis
	• Siempre es una buena práctica utilizar VARCHAR2 en lugar de VARCHAR para evitar cambios de comportamiento.	• La declaración anterior declaró la variable 'administrador' del tipo de datos VARCHAR con el tamaño máximo de 10 y asignó el valor 'guru99' que es de 6 bytes. Oracle asignará memoria de sólo 6 bytes en este caso. (Similar a VARCHAR2)
	• NVARCHAR2 (tamaño de cadena variable nativa) • LARGO y LARGO CRUDO	
NCHAR	Este tipo de datos es el mismo que el tipo de datos CHAR, pero el juego de caracteres será el juego de caracteres nacional. • Este juego de caracteres se puede definir para la sesión utilizando NLS_PARAMETERS. • El juego de caracteres puede ser UTF16 o UTF8. • La restricción de tamaño es de 1 a 2000 bytes.	native NCHAR(10); **Explicación de sintaxis:** • La declaración anterior declara la variable 'nativa' del tipo de datos NCHAR con un tamaño máximo de 10. • La longitud de esta variable depende del (número de longitudes) por byte tal como se define en el juego de caracteres.
NVARCHAR2	Este tipo de datos es el mismo que el tipo de datos VARCHAR2, pero el juego de caracteres será el juego de caracteres nacional. • Este juego de caracteres se puede definir para la sesión utilizando NLS_PARAMETERS. • El juego de caracteres puede ser UTF16 o UTF8. • La restricción de tamaño es de 1 a 4000 bytes.	Native var NVARCHAR2(10):='guru99'; **Explicación de sintaxis:** • La declaración anterior declara la variable 'Native_var' del tipo de datos NVARCHAR2 con un tamaño máximo de 10.
LARGO y LARGO	Este tipo de datos se utiliza para almacenar texto grande o datos sin procesar hasta un tamaño máximo de 2 GB.	Large_text LONG; Large_raw LONG RAW; **Explicación de sintaxis:**

	• Se utilizan principalmente en el diccionario de datos. • El tipo de datos LONG se utiliza para almacenar datos del juego de caracteres, mientras que LONG RAW se utiliza para almacenar datos en formato binario. • El tipo de datos LONG RAW acepta objetos multimedia, imágenes, etc., mientras que LONG solo funciona con datos que se pueden almacenar utilizando un juego de caracteres.	• La declaración anterior declara la variable 'Large_text' del tipo de datos LONG y 'Large_raw' del tipo de datos LONG RAW. **Nota:** No se recomienda el uso del tipo de datos LARGO Oracle. En su lugar, se debe preferir

NÚMERO PL/SQL Tipo de datos

Este tipo de datos almacena números de punto fijo o flotante de hasta 38 dígitos de precisión. Este tipo de datos se utiliza para trabajar con campos que contendrán solo datos numéricos. La variable se puede declarar con precisión y detalles de dígitos decimales o sin esta información. Los valores no necesitan estar entre comillas al asignarse para este tipo de datos.

A NUMBER(8,2);

B NUMBER(8);

C NUMBER;

Explicación de sintaxis:

- En lo anterior, la primera declaración declara que la variable 'A' es de tipo de datos numéricos con precisión total 8 y dígitos decimales 2.
- La segunda declaración declara que la variable 'B' es de tipo de datos numéricos con precisión total 8 y sin dígitos decimales.
- La tercera declaración es la más genérica, declara que la variable 'C' es de tipo numérico sin restricción en precisión o decimales. Puede tener hasta un máximo de 38 dígitos.

Tipo de datos booleano PL/SQL

Este tipo de datos almacena los valores lógicos. Oracle El tipo de datos booleano representa VERDADERO o FALSO y se utiliza principalmente en declaraciones condicionales. No es necesario que los valores estén entre comillas al asignarlos para este tipo de datos.

Var1 BOOLEAN;

Explicación de sintaxis:

- En lo anterior, la variable 'Var1' se declara como tipo de datos BOOLEANO. La salida del código será verdadera o falsa según la condición establecida.

Tipo de datos de fecha PL/SQL

Este tipo de datos almacena los valores en formato de fecha, como fecha, mes y año. Siempre que una variable se define con el tipo de datos FECHA junto con la fecha, puede contener información de hora y, de forma predeterminada, la información de hora se establece en 12:00:00 si no se especifica. Los valores deben estar entre comillas al asignarse para este tipo de datos.

El Oracle El formato de hora para entrada y salida es 'DD-MON-AA' y nuevamente se establece en NLS_PARAMETERS (NLS_DATE_FORMAT) en el nivel de sesión.

```
newyear DATE:='01-JAN-2015';

current_date DATE:=SYSDATE;
```

Explicación de sintaxis:

- En lo anterior, la variable "año nuevo" se declara como tipo de datos FECHA y se le asigna el valor del 1 de enero.st, fecha 2015.
- La segunda declaración declara la variable current_date como tipo de datos DATE y le asigna el valor con la fecha actual del sistema.
- Ambas variables contienen la información de tiempo.

Tipo de datos LOB PL/SQL

Este tipo de datos se utiliza principalmente para almacenar y manipular grandes bloques de datos no estructurados como imágenes, archivos multimedia, etc. Oracle prefiere LOB en lugar del tipo de datos LONG ya que es más flexible que el tipo de datos LONG. Las siguientes son las principales ventajas del tipo de datos LOB sobre el tipo de datos LARGO.

- El número de columnas en una tabla con tipo de datos LONG está limitado a 1, mientras que una tabla no tiene restricción en el número de columnas con tipo de datos LOB.
- La herramienta de interfaz de datos acepta el tipo de datos LOB de la tabla durante la replicación de datos, pero omite la columna LARGA de la tabla. Estas columnas LARGAS deben replicarse manualmente.
- El tamaño de la columna LARGA es de 2 GB, mientras que LOB puede almacenar hasta 128 TB.

- Oracle está mejorando constantemente el tipo de datos LOB en cada una de sus versiones de acuerdo con los requisitos modernos, mientras que el tipo de datos LONG es constante y no recibe muchas actualizaciones.

Por lo tanto, siempre es mejor utilizar el tipo de datos LOB en lugar del tipo de datos LONG. A continuación, se muestran los diferentes tipos de datos LOB. Pueden almacenar hasta un tamaño de 128 terabytes.

1. BLOB
2. CLOB y NCLOB
3. ARCHIVOB

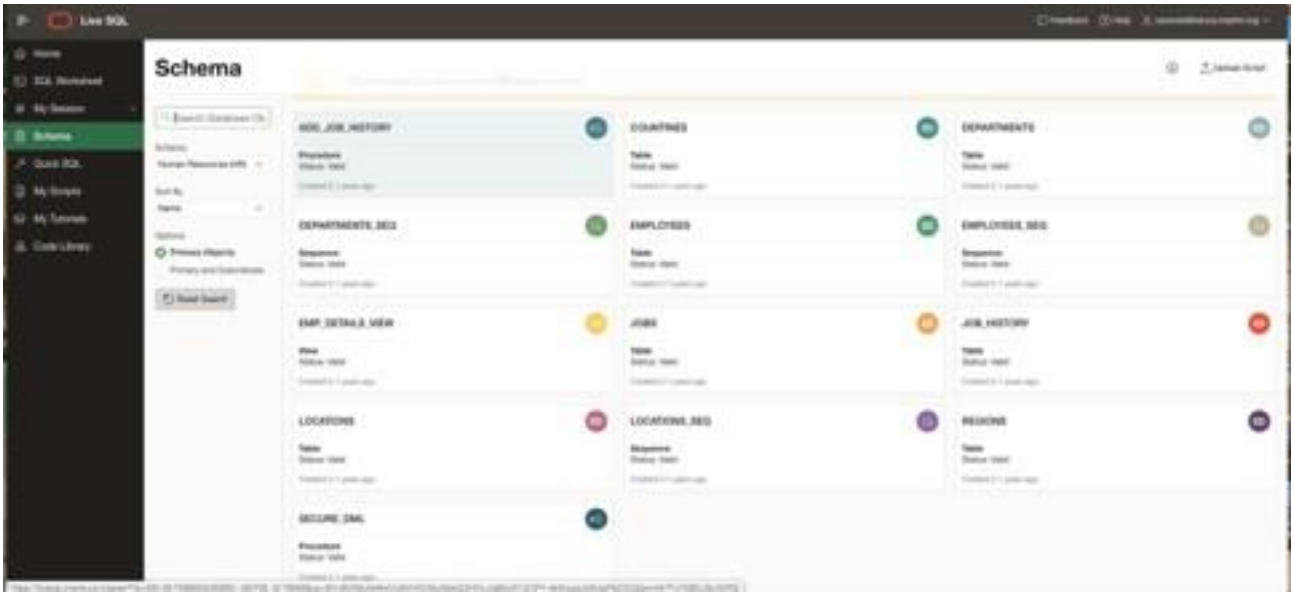
Tipo de datos	Descripción	Sintaxis
BLOB	Este tipo de datos almacena los datos LOB en formato de archivo binario hasta un tamaño máximo de 128 TB. No almacena datos basados en los detalles del conjunto de caracteres, por lo que puede almacenar datos no estructurados, como objetos multimedia, imágenes, etc.	Binary_data BLOB; **Explicación de sintaxis:** • En lo anterior, la variable 'Binary_data' se declara como BLOB.
CLOB y NCLOB	El tipo de datos CLOB almacena los datos LOB en el juego de caracteres, mientras que NCLOB almacena los datos en el juego de caracteres nativo. Dado que estos tipos de datos utilizan almacenamiento basado en juegos de caracteres, no pueden almacenar datos como multimedia, imágenes, etc. que no se pueden poner en una cadena de caracteres. El tamaño máximo de estos tipos de datos es 128 TB.	Charac_data CLOB; **Explicación de sintaxis:** • En lo anterior, la variable 'Charac_data' se declara como tipo de datos CLOB.
ARCHIVOB	• BFILE son los tipos de datos que almacenaron los datos en formato binario no estructurado fuera de la base de datos como un archivo del sistema operativo. • El tamaño de BFILE es para un sistema operativo limitado, son archivos de solo lectura y no se pueden modificar.	

ENTORNO DE DESARROLLO

- Existen diferentes entornos de desarrollo para PL/SQL. Los principales son:
  - SQL*PLUS(ISQLPLUS)
  - Oracle Developer Procedure Builder (herramienta de desarrollo)
  - El primero utiliza el motor en el servidor Oracle y el segundo cuenta con las dos opciones disponibles.
  - Dispondremos de una opciones u otras en relación a la construcción de programas con bloques:
    - Un bloque anónimo se puede utilizar en cualquier entorno PL/SQl. Son un conjunto de instrucciones que se ejecutan en modo local. Normalmente se utilizan para hacer pruebas
    - Bloque nominado, es igual pero tiene una etiqueta. También se ejecuta en modo local

ORACLE LIVE SQL

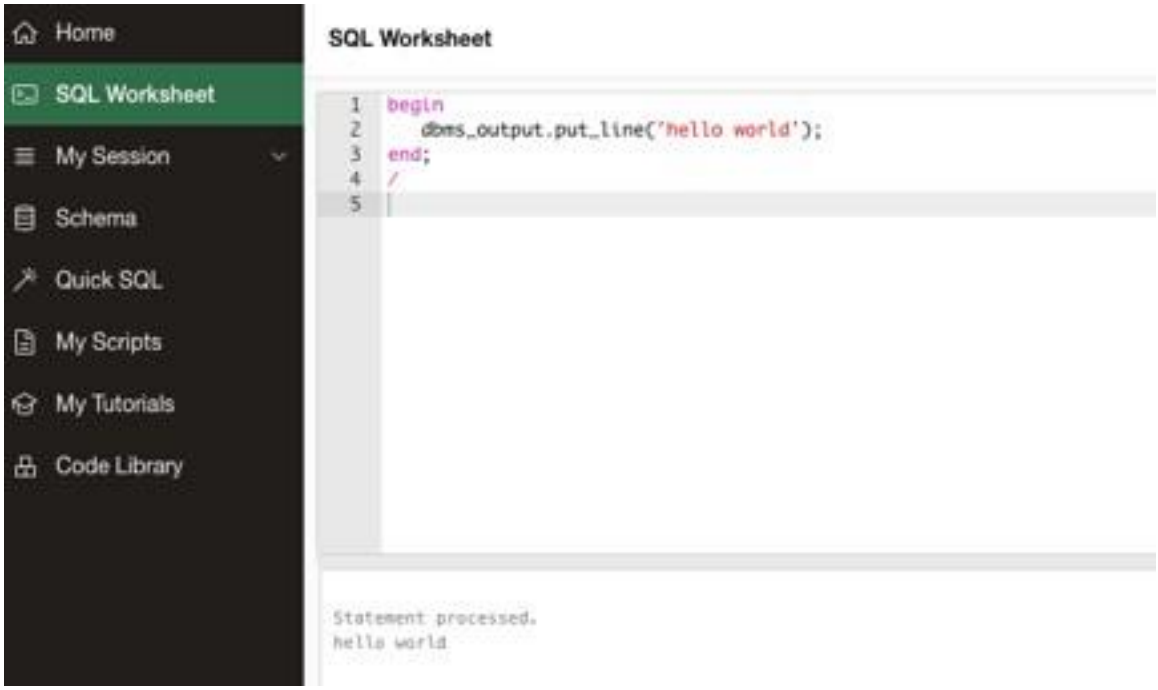
Dada la situación actual vamos a utilizar Live SQL de orca para probar nuestro código. En este entorno entras a SQL



Como veis en sql Worksheet podemos introducir nuestro código sql y PL/sql.

Os he hecho una captura del las tablas del usuario o esquema HR que viene como ejemplo en Oracle

Como vemos a continuación , acabamos de ejecutar un bloque anónimos que visualiza un "Hello World" de prueba.



Ejemplo:

```
declare
    l_today date := sysdate;
begin
    dbms_output.put_line(
          'today is '||to_char(l_today,'Day'));
exception when others then
    dbms_output.put_line(sqlerrm);
end;
/
```

El código anterior nos visualiza que hoy es miércoles (wednesday) y saltaría una excepción que nos visualiza un código de error , si se produjese algún error.

ASIGNACION DE VALORES A VARIABLES.

- La asignación de valores a una variable en la parte ejecutable de un bloque se puede realizar a través del operador := pero también con la cláusula INTO de las sentencias SELECT y con FETCH (Se estudiará posteriormente en el tema).

  Ejemplos:

  ```
  total := unidades * pts_unidad;
  SELECT salario INTO sueldo     FROM plantilla
            WHERE apellido LIKE 'MORENO';
   FECHT cursor1 INTO nombre;
  ```

  Para empezar a familiarizarnos con el SQL dentro de un bloque PL/SQL vemos que las Select ya no hacen la salida en un terminal, sino que lo que recuperan lo almacenan en la variable que pongamos detrás del Into.

  En este caso recuerda de la base de datos el sueldo de "MORENO2 y lo que recupere lo guarda en la variable sueldo.

- También es posible realizar asignaciones de valor a las variables en el momento de su declaración, para ello se utilizará la palabra clave DEFAULT, o bien, el operador de asignación.

  Ejemplos:

  ```
  importe   NUMBER (9) DEFAULT 3000;
  nombre    VARCHAR2(20) := 'Marta Parrado';
  ```

EXPRESIONES.

- Una expresión está formada por un conjunto de operando unidos por operadores, los cuales ya fueron estudiados en el Tema 3.
- Cabe mencionar algún operador que incorpora PL/SQL y la prioridad de los operadores:

Operador	Operación
**,NOT	Exponenciacion , negación
+,-	Operadores unario (positivo, negativo)
*./	Multiplicacion,división
+,-, ||	Suma, resta, concatenación
= , ! = , < > , > , < , > = , < = , I S NULL,LIKE,BETWEEN, IN	Operadores de comparación
AND	Conjunción
OR	Inclusión

VARIABLES Y CONSTANTES.

- PL/SQL permite declarar e inicializar variables y/o constantes.
- Las variables pueden ser de cualquiera de los tipos vistos en SQL o de otros tipos que veremos.
- Todas las variables han de ser declaradas antes de ser utilizadas
- Las variables se pueden utilizar:
  - Para almacenar temporalmente datos de entrada hasta que ya no se necesiten
  - Para realizar operaciones sobre los datos
  - Facilidad de mantenimiento: %ROWTYPE,%TYPE
- Para declarar una variable utilizaremos el siguiente formato:
  Nombre_variable [CONSTANT] tipo [NOT NULL]
  [{:=| DEFAULT } expresión];
- Las palabras clave NOT NULL especifican que la variable no podrá tomar valor nulo.
- La palabra clave DEFAULT permite inicializar una variable a la vez que se define.
- Expresión podrá ser un literal, otra variable o el resultado de una expresión que puede incluir operadores y funciones.
- Para los identificadores de las variables se deben utilizar las mismas reglas que para los objetos de SQL
  - Longitud máxima: 30 caracteres
  - Letras, números y caracteres solo $, _, #
  - El nombre debe empezar por una letra
  - No puede ser una palabra reservada de Oracle
- Para asignar valores por defecto podemos utilizar indistintamente := o DEFAULT
- Una variable no inicializada queda con valor NULL hasta asignarle valor en la ejecución
- Las constantes y variables declaradas como NOT NULL deben ser inicializadas
- Dos objetos pueden tener el mismo nombre siempre y cuando estén declarados en distintos bloques
- No se debe poner el mismo nombre a un identificador de variable que a la columna de la que vaya a recibir los datos. Esto evita confusiones y facilita el mantenimiento del software

Ejemplos:

```
importe     NUMBER (9);
nombre      VARCHAR2(20) NOT NULL;
descuento NUMBER(4,2) NOT NULL DEFAULT 10.27;
nombre char(20) NOT NULL := `MIGUEL';
```

casado BOOLEAN DEFAULT FALSE;

- Podemos utilizar el atributo %TYPE para dar a una variable el tipo de dato de otra variable o de una columna de la base de datos.

Nom_variable tabla.columna%type;

Ejemplo:

importe NUMBER (9);
total importe %TYPE; → Declarada total como NUMBER(9)
nuevonombre emp.nombre%TYPE -> Declara nuevonombre del mismo tipo que la columna nombre de emp

- La principal ventaja del uso de este atributo es la facilidad de mantenimiento de los programas, ya que ante cualquier cambio de tipos en la base de datos no hay que hacer ningún cambio en el código Pl/SQL

- %TYPE también puede utilizarse para declara una variable del mismo tipo que otra declarada previamente.

- **Si una columna tiene la restricción NOT NULL, la variable definida de ese tipo mediante el atributo %TYPE, no asume dicha restricción, pudiendo, por lo tanto, ponerla a NULL, a no ser que se especifique lo contrario en la declaración.**

Ejemplo: planta vecinos.piso %TYPE NOT NULL;
Declarada como NUMBER(2) y no
admitiendo valores nulos.

- También podemos utilizar %ROWTYPE que crea una variable tipo registro para cargar los valores de toda una fila.
Mifila emp%ROWTYPE

Variable tabla%ROWTYPE;

- Se pueden declarar constantes utilizando la palabra reservada CONSTANT, pero teniendo en cuenta que la constante se debe inicializar en el momento de la declaración.

- El valor de una constante, una vez determinada no modificarse. La declaración de constantes facilita la modificación de programas que contienen datos constantes.

- La declaración de una constante es igual a la de una variable, salvo que hay que utilizar CONSTANT e inicializarla.

Ejemplo: iva CONSTANT NUMBER(2) DEFAULT 16;

- En el ejemplo anterior, si no declaramos la constante 'iva', un cambio en el porcentaje del impuesto obligaría a cambiarlo en todas las expresiones donde apareciese.

- Cualquier variable o constante debe ser declarada antes de ser referenciada en otra declaración o sentencia, ya que en caso contrario se produciría un error.

  Ejemplo: total importe %TYPE; → Error
  importe NUMBER(9,2);

Procedimientos y Funciones PL/SQL

INTRODUCCION

- Los procedimientos y funciones son subprogramas que pueden ser invocados por los usuarios.
- En PL/SQL el desarrollador puede definir sus propios subprogramas o bien utilizar funciones predefinidas en PL/SQL.
- Los subprogramas pueden ser de dos tipos:
  - Funciones: subprogramas que devuelven un valor
  - Procedimientos: subprogramas que ejecutan una secuencia de instrucciones pero que el nombre del subprograma en si mismo no devuelve un valor.

FUNCIONES PREDEFINIDAS PL/SQL

PL/SQL proporciona un gran número de funciones muy útiles para ayudar a manipular la información y permite incorporar en sus expresiones casi todas las funciones disponibles en SQL. Se pueden agrupar en categorías:

- caracteres
- numéricas
- fechas
- conversión de tipo de datos
- manejo de nulos
- misceláneas
- error-reporting

Las funciones de agrupación de SQL como por ejemplo AVG, MIN, MAX, COUNT, SUM, STDDEV, y VARIANCE, no están implementadas en PL/SQL, sin embargo se pueden usar en sentencias SQL. Tampoco se pueden usar algunas otras como DECODE, DUMP, y VSIZE.

A continuación vamos a ver algunas de las más utilizadas.

FUNCIONES PREDEFINIDAS CARACTERES

- LENGTH: Devuelve la longitud de un tipo CHAR.

```
resultado := LENGTH('HOLA MUNDO’); -- Devuelve 10
```

- INSTR

Busca una cadena de caracteres (la que se indica en el segundo parámetro pasado) dentro de otra (la que se indica en el primer parámetro) y devuelve la posición de la ocurrencia de la cadena buscada dentro de la cadena. En el tercer parámetro se indica la posición desde la que se comienza a buscar (opcional) y en el cuarto el número de ocurrencia que se busca (opcional).

Su sintaxis es la siguiente: INSTR(<char>, <search_string>, <startpos>, <occurrence>)

```
resultado := INSTR('AQUI ES DONDE SE BUSCA', 'BUSCA', 1, 1); -- Devuelve 18
```

FUNCIONES PREDEFINIDAS CARACTERES

- REPLACE: Reemplaza un texto por otro en una cadena de caracteres.

REPLACE(<expresion>, <busqueda>, <reemplazo>)

El siguiente ejemplo reemplaza la palabra 'HOLA' por 'VAYA' en la cadena 'HOLA MUNDO'.

```
resultado := REPLACE ('HOLA MUNDO','HOLA', 'VAYA'); -- devuelve VAYA MUNDO
```

- SUBSTR: Obtiene una parte de una cadena de caracteres, desde una posición de inicio hasta una determinada longitud.

SUBSTR(<expresion>, <posicion_ini>, <longitud>)

```
resultado := SUBSTR('HOLA MUNDO', 6, 5); -- Devuelve MUNDO
```

FUNCIONES PREDEFINIDAS CARACTERES

- UPPER: Convierte una cadena alfanumérica a mayúsculas.

```
resultado := UPPER('hola mundo'); -- Devuelve HOLA MUNDO
```

- LOWER: Convierte una cadena alfanumérica a minúsculas.

```
resultado := LOWER('HOLA MUNDO'); -- Devuelve hola mundo
```

- RTRIM: Elimina los espacios en blanco a la derecha de una cadena de caracteres.

```
resultado := RTRIM ('Hola Mundo ');
```

- LTRIM: Elimina los espacios en blanco a la izquierda de una cadena de caracteres.

```
resultado := LTRIM (' Hola Mundo');
```

- TRIM: Elimina los espacios en blanco a la izquierda y derecha de una cadena de caracteres.

```
resultado := TRIM (' Hola Mundo ');
```

FUNCIONES PREDEFINIDAS NUMERICAS

- MOD: Devuelve el resto de la división entera entre dos números.

MOD(<dividendo>, <divisor>)

```
resultado := MOD(20,15); -- Devuelve el modulo de dividir 20/15
```

- TRUNC: Trunca un número y devuelve la parte entera.

```
resultado := TRUNC(9.99); -- Devuelve 9
```

- ROUND: Devuelve el entero más próximo.

```
resultado := ROUND(9.99); -- Devuelve 10
```

FUNCIONES PREDEFINIDAS FECHAS

- SYSDATE: Devuelve la fecha del sistema.

```
resultado := SYSDATE;
```

- TRUNC: Trunca una fecha, elimina las horas, minutos y segundos de la misma.

```
resultado := TRUNC(SYSDATE);
```

FUNCIONES PREDEFINIDAS CONVERSION TIPO DATOS

- TO_DATE: Convierte una expresión al tipo fecha.

TO_DATE(<expresion>, [<formato>])

El parámetro opcional formato indica el formato de entrada de la expresión no el de salida.

En este ejemplo se convierte la cadena de caracteres '01/12/2006' a una fecha (tipo DATE). El formato indica que la fecha está escrita como día/mes/año, de forma que la fecha sea el uno de diciembre y no el doce de enero.

```
resultado := TO_DATE('01/12/2006', 'DD/MM/YYYY');
```

El siguiente ejemplo muestra la conversión con formato de día y hora.

```
resultado := TO_DATE('31/12/2006 23:59:59', 'DD/MM/YYYY HH24:MI:SS');
```

FUNCIONES PREDEFINIDAS CONVERSION TIPO DATOS

- TO_CHAR: Convierte una expresión al tipo CHAR.

TO_CHAR(<expresion>, [<formato>])

El parámetro opcional formato indica el formato de salida de la expresión.

```
resultado := TO_CHAR(SYSDATE, 'DD/MM/YYYY HH24:MI:SS');
```

- TO_NUMBER: Convierte una expresión alfanumérica en numérica, se puede especificar el formato de salida (opcional).

TO_NUMBER(<expresion>, [<formato>])

```
resultado := TO_NUMBER ('10.21', ’99.99’); -- resultado: 10,21 (el separador decimal es ,)
```

FUNCIONES PREDEFINIDAS MANEJO DE NULOS

- NVL: Devuelve el valor recibido como parámetro en el caso de que expresión sea NULL o el valor de la expresión en caso contrario.

NVL(<expresion>, <valor>)

El siguiente ejemplo devuelve 0 si el precio es nulo, y el precio cuando está informado:

```
SELECT CO_PRODUCTO, NVL(PRECIO, 0) FROM PRECIOS;
```

FUNCIONES PREDEFINIDAS MISCELANEAS

- DECODE: Proporciona la funcionalidad de una sentencia de control de flujo if-elseif-else.

DECODE(<expr>, <cond1>, <val1>[, ..., <condN>, <valN>], <default>)

Esta función evalúa una expresión "<expr>", si se cumple la primera condición "<cond1>" devuelve el valor1 "<val1>", en caso contrario evalúa la siguiente condición y así hasta que una de las condiciones se cumpla. Si no se cumple ninguna condición se devuelve el valor por defecto (el último parámetro).

Es muy común escribir la función DECODE indentada como si se tratase de un bloque IF.

```
SELECT DECODE (co_pais, /* Expresion a evaluar */
               'ESP', 'ESPAÑA', /* Si co_pais = 'ESP' ==> 'ESPAÑA' */
               'MEX', 'MEXICO', /* Si co_pais = 'MEX' ==> 'MEXICO' */
               'PAIS '||co_pais)/* ELSE ==> concatena */
FROM PAISES;
```

FUNCIONES PREDEFINIDAS

Carácter	Numéricas	Fecha	Conversión	Manejo Nulos	Misceláneas	Error
ASCII	ABS	ADD_MONTHS	CHARTOROWID	NVL	DECODE	SQLCODE
CHR	ACOS	CURRENT_DATE	CONVERT		DUMP	SQLERRM
CONCAT	ASIN	CURRENT_TIMESTAM	HEXTORAW		GREATEST	
INITCAP	ATAN	LAST_DAY	NLS_CHARSET_ID		GREATEST_LB	
INSTR	ATAN2	LOCALTIMESTAMP	NLS_CHARSET_NAME		LEAST	
INSTRB	CEIL	MONTHS_BETWEEN	RAWTOHEX		LEAST_UB	
LENGTH	COS	NEW_TIME	ROWIDTOCHAR		UID	
LENGTHB	COSH	NEXT_DAY	TO_CHAR		USER	
LOWER	EXP	ROUND	TO_DATE		USERENV	
LPAD	FLOOR	SYSDATE	TO_LABEL		VSIZE	
LTRIM	LN	SYSTIMESTAMP	TO_MULTI_BYTE			
NLS_INITCAP	LOG	TRUNC	TO_NUMBER			
NLS_LOWER	MOD		TO_SINGLE_BYTE			
NLS_UPPER	POWER					
NLSSORT	ROUND					
REPLACE	SIGN					
RPAD	SIN					
RTRIM	SINH					
SOUNDEX	SQRT					
SUBSTR	TAN					
SUBSTRB	TANH					
TRANSLATE	TRUNC					
UPPER						

PROCEDIMIENTOS Y FUNCIONES DEFINIDOS POR EL DESARROLLADOR

Los bloques de código anónimos BEGIN .. END, son un mecanismo básico para la programación en PL/SQL, pero no están orientados a la reutilización de SCRIPTS. Por ejemplo, en caso de que se tenga un algoritmo para algún cálculo según determinados parámetros tendríamos que repetirlo cuantas veces sea necesario.

El uso de procedimientos en PL/SQL es un buen mecanismo para la reutilización de código, además de que permite dividir el código en partes funcionales más pequeñas.

Los procedimientos pueden ser declarados en bloques anónimos o almacenarnos en la base de datos.

DECLARACION DE PROCEDIMIENTOS

La creación de un procedimiento en PL/SQL es similar a la creación de un bloque anónimo. La sintaxis de un procedimiento es la siguiente:

```
PROCEDURE nom_proc[(param1[,param2 ...])]
IS
  declaraciones locales;
BEGIN
  sentencias;
[EXCEPTION
         tratamiento_de_excepciones]
END [nom_proc];
```

- nom_proc: Es el nombre del procedimiento, se usará para identificarlo.
- param: son como variables, contienen datos que se pueden especificar en el momento de llamar al procedimiento.
- declaraciones locales: Como en un bloque anónimo se pueden crear variables que sólo pueden ser usadas en código dentro del procedimiento.
- sentencias: Es el código que se ejecuta cuando se llama al procedimiento y que puede hacer uso de las variables declaradas así como de los parámetros.

DECLARACION DE PROCEDIMIENTOS

El IS es el equivalente a DECLARE en los bloques anónimos.

En el IS sí debemos indicar la longitud de las variables locales.

PARAMETROS DE LOS PROCEDIMIENTOS

Tienen la siguiente sintaxis:

Nom_param [IN|OUT|IN OUT] tipo_dato[{:=|DEFAULT }Valor]

- Cuando no se indica, los parámetros se definen por defecto de tipo IN.
- En tipo_dato sólo se especifica el tipo, sin indicar su longitud ni restricciones.
- Si un procedimiento no tienen parámetros, no es necesario poner los paréntesis en la cabecera.

- Un parámetro de entrada permite que pasemos valores al subprograma y no puede ser modificado en el subprograma. El parámetro pasado puede ser una constante o una variable.
- Un parámetro de salida permite devolver valores y en el subprograma actúa como variable no inicializada. El parámetro pasado debe ser una variable.
- Un parámetro de entrada-salida se utiliza para pasar valores al subprograma y/o para recibirlos, por lo que un parámetro formal que actúe como parámetro pasado debe ser una variable.

PROCEDIMIENTOS DENTRO DE UN BLOQUE ANONIMO

En estos casos el procedimientos se debe crear dentro de la sección DECLARE … BEGIN.

```
DECLARE
-- El procedimiento debe ser declarado dentro de la sección DECLARE .. BEGIN
  PROCEDURE registrar_cliente(P_ID NUMBER,
                              P_NOMBRE VARCHAR2,
                              P_APELLIDOS VARCHAR2)
  IS
     declaraciones locales;
  BEGIN
     sentencias;
  [EXCEPTION
           tratamiento_de_excepciones]
  END [nom_proc];
BEGIN
-- Sentencias, código de bloque anónimo
 ...
  REGISTRAR_CLIENTE(1,'Juan', 'Rosales');
  REGISTRAR_CLIENTE(2,'Luis', 'Cabrera');
 ...
END;
/
```

La declaración de procedimientos debe ir al final de la sección DECLARE correspondiente.

PROCEDIMIENTOS ALMACENADOS

PL/SQL permite almacenar los procedimientos para ser usados desde cualquier bloque anónimo (sin que haya la necesidad de declararlo) y también desde otros procedimientos.

Para crear un procedimiento almacenado debemos poner la palabra reservada CREATE y ejecutar el código como si se tratase de un bloque PL/SQL.

El procedimiento almacenado es compilado previamente por el motor PL/SQL y si no da errores quedará almacenado y se podrá llamar.

```
CREATE PROCEDURE registrar_cliente (P_ID NUMBER,
                                    P_NOMBRE VARCHAR2,
                                    P_APELLIDOS VARCHAR2)
IS
  declaraciones locales;
BEGIN
  sentencias;
[EXCEPTION
         tratamiento_de_excepciones]
END [nom_proc];
```

PROCEDIMIENTOS ALMACENADOS

Adicionalmente, se puede añadir las palabras reservadas **OR REPLACE** para evitar errores al intentar compilar un procedimiento que ya ha sido compilado :

CREATE OR REPLACE PROCEDURE REGISTRAR_CLIENTE….

Con esta sentencia creamos un procedimiento. Si ya existía lo reemplaza.

En el caso de los procedimientos almacenados ya no es necesario declarar el procedimiento dentro de la sección DECLARE .. BEGIN de los bloques anónimos.

Un procedimiento **se puede invocar desde un** bloque u otro procedimiento/ función de PL/SQL **llamándolo** simplemente **por su nombre** y pasándole los parámetros.

PROCEDIMIENTOS ALMACENADOS

Se puede invocar un procedimiento también desde SQL*PLUS:

Sql> **execute** registrar_cliente (7902,'Antonio','Alvarez Sánchez');

Si alguno de los parámetros fuera de salida (OUT o IN OUT) se debe invocar con una variable que debe ser definida previamente:

```
// Creación del procedimiento, el segundo parámetro es de salida
CREATE OR REPLACE PROCEDURE calcular_cuadrado_procedure(P_NUMERO NUMBER, P_CUADRADO OUT NUMBER)
IS
BEGIN
  P_CUADRADO := P_NUMERO*P_NUMERO;
END calcular_cuadrado_procedure;
```

```
// Llamada al procedimiento desde consola SQL*PLUS
SQL> var num_cuadrado NUMBER -- Se define la variable Host necesaria para parámetro de salida

SQL> exec calcular_cuadrado_procedure(5,:num_cuadrado) -- Llamada a método, : antes de variable Host

SQL> PRINT num_calculo -- Muestra por pantalla la variable Host
```

METODOS DE PASO DE PARAMETROS

Notación Posicional: Se pasan los valores de los parámetros en el mismo orden en que el procedure los define.

```
BEGIN
  REGISTRAR_CLIENTE(1,'Juan', 'Rosales');
END;
```

Notación Nominal: Se pasan los valores en cualquier orden nombrando explícitamente el parámetro y su valor separados por el símbolo =>.

```
BEGIN
  REGISTRAR_CLIENTE(P_ID => 1,P_NOMBRE => 'Juan', P_APELLIDOS => 'Rosales');
END;
```

Notación Mixta: Combina las dos anteriores.

```
BEGIN
  REGISTRAR_CLIENTE(1,P_NOMBRE => 'Juan', P_APELLIDOS => 'Rosales');
END;
```

DECLARACION DE FUNCIONES

La creación de una función tiene una sintaxis similar a la de un procedimiento:

```
FUNCTION nom_funcion([param1[,param2 ...]])
RETURN [tipo de valor devuelto]
IS
  declaraciones locales;
BEGIN
  sentencias;
  RETURN(expresión);
[EXCEPTION
         tratamiento_de_excepciones]
END [nom_funcion];
```

- Los parámetros tienen la misma sintaxis que en los procedimientos.
- La cláusula RETURN de la cabecera indica el tipo de datos que devuelve la función.
- La cláusula RETURN del cuerpo hace efectivo ese retorno.

FUNCIONES ALMACENADAS

Como para los procedimientos, para crear una función almacenada:

```
CREATE OR REPLACE FUNCTION nom_funcion(([param1[,param2 ...]])
RETURN [tipo de valor devuelto]
IS
   declaraciones locales;
BEGIN
   sentencias;
   RETURN(expresión);
[EXCEPTION
        tratamiento_de_excepciones]
END [nom_funcion];
```

En el caso de las funciones almacenadas ya no es necesario declararla dentro de la sección DECLARE .. BEGIN de los bloques anónimos.

LLAMADAS A FUNCIONES

Una función **se puede invocar desde un** bloque u otro procedimiento / función de PL/SQL **llamándola** simplemente **por su nombre** y pasándole los parámetros requeridos, asignando el valor (mediante :=) a una variable del mismo tipo que devuelve la función:

```
BEGIN
  ...
  num_calculo := calcular_cuadrado(3);
  ...
END;
```

Se puede invocar también desde SQL*PLUS

- Consulta genérica:

```
SQL> select year_of_date(start_date) FROM DUAL;
```

- Utilizando exec y variables Host:

```
SQL> var num_calculo NUMBER -- Se define la variable de Host necesaria para asignar valor a la función
SQL> exec :num_calculo := calcular_cuadrado(3) -- Llamada a la función, usar : antes de variable Host

SQL> PRINT num_calculo -- Muestra por pantalla la variable Host
```

PROCEDIMIENTOS Y FUNCIONES

PL/SQL permite la **sobrecarga** en los nombres de subprogramas (aplica a procedimientos y funciones), es decir, podemos llamar a dos subprogramas con el mismo nombre y los distingue porque sus parámetros deben tener o distinto número o distintos tipo. La sobrecarga de los subprogramas se usa generalmente cuando conceptualmente se ejecuta la misma tarea (o similar) pero con un conjunto de parámetros ligeramente diferente (o con diferente definición).

También permite **programación recursiva** (ejemplo típico cálculo del factorial de un número).

PROCEDIMIENTOS Y FUNCIONES

Se pueden conocer los procedimientos y funciones definidos mediante la siguiente consulta

```
SELECT object_name FROM user_procedures;
```

Se puede eliminar un procedimiento ejecutando

```
drop procedure nombre_procedimiento;
```

Se puede eliminar una función ejecutando

```
drop function nombre_función;
```

SENTENCIAS DE CONTROL.

- PL/SQL dispone de sentencias que permiten construir las tres estructuras de control de la programación estructurada, es decir, secuencial, alternativa y repetitiva, teniendo en cuenta que para las dos últimas estructuras están permitidas las anidaciones, es decir, dentro de in if podremos encontrar otro, o dentro de un while podremos anidar por ejemplo otro while.

SENTENCIAS SECUENCIALES

- Se consideran una secuencia , al conjunto de instrucciones que se ejecutan una detrás de otra.
- Cada sentencia en PL/SQL finaliza con ;

SENTENCIAS ALTERNATIVAS

- Son las sentencias que nos permiten cambiar el flujo del programa dependiendo de que se cumpla o no una condición.
- Oracle cuenta con las siguientes sentencias alternativas

IF-THEN

- Como se aprecia en el primer gráfico que se presenta a continuación, , cuando se cumple la condición, es decir cuando devuelve True, se ejecutan una serie de sentencias (<action_block>) que no se ejecutarán si no se cumple la condición, pero en cualquiera de los dos casos, continúa la ejecución del programa después del END IF;
- La condición por tanto, siempre debe ser evaluada como true o false
- La sintaxis de un If será.

```
IF <condition: returns Boolean>
THEN
 -executed only if the condition returns TRUE
 <action_block>
END if;
```

- Cualquier condición evaluada como 'NULL', será tratada como ‘FALSE'.

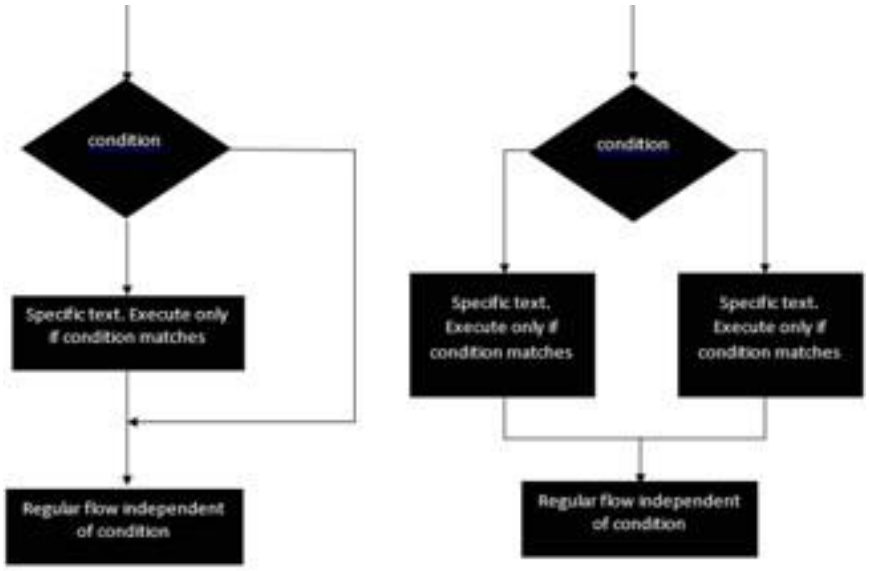
Ejemplo :

```
DECLARE
a CHAR(1) :=’u’;
BEGIN
IF UPPER(a) in ('A’,'E','I','0','U' ) THEN
dbms_output.put_line(‘El carácter es una vocal');
END IF;
END;
/
```

- Como vemos la función que coge el carácter y lo convierte a a mayusculas es UPPER(), exactamente igual que en SQL
- In , permite comprobar si el valor está entre los citados entre los ( )

IF-THEN-ELSE

- Como se aprecia en el segundo gráfico, si se cumple la condición se ejecuta un conjunto de sentencias y si no se cumple se ejecutan otra sentencia o conjunto de ellas, y después en cualquiera de los dos casos continúa con la ejecución del resto del programa



Decision Making Statement Diagram

IF <condition: returns Boolean>
THEN
-executed only if the condition returns TRUE
<action_blockl>
ELSE
-execute if the condition failed (returns FALSE)
<action_block2>
END if;

- Se ejecuta <action_blockl>cuando la condición devuelve true
- Se ejecuta <action_block2> cuando la condición devuelve False

IF-THEN-ELSIF

```
 IF <conditionl: returns Boolean>
THEN
-executed only if the condition returns TRUE <
action_blockl>
ELSIF <condition2 returns Boolean> <
action_block2>
ELSIF <condition3:returns Boolean> <
action_block3>
ELSE —optional
<action_block_else>
END if;
```

- Esta sentencia alternativa se utiliza cuando hay que seleccionar una alternativa entre un conjunto de ellas.
- La primera condición que devuelva verdadero, será la que se ejecute , y el resto no se ejecutará.
- En caso de que no se cumpla ninguna, se ejecutan las sentencias del bloque ELSE si es que existiese.

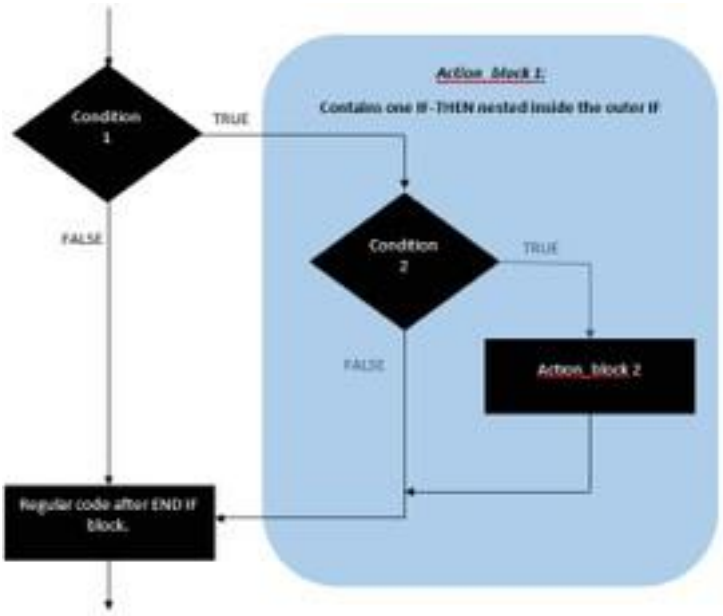
Ejemplo:

```
DECLARE
mark NUMBER :=55;
BEGIN
dbms_output.put_line('Program started.' );
IF( mark >= 70) THEN
dbms_output.put_line('Grade A');
ELSIF(mark >= 40 AND mark < 70) THEN
dbms_output.put_line('Grade B');
ELSIF(mark >=35 AND mark < 40) THEN
dbms_output.put_line('Grade C');
END IF;
dbms_output.put_line('Program completed.');
END;
/
```

Como vemos en este caso, la primera condición no la cumple IF (mark >= 70), por tanto , pasa a evaluar la siguiente . ELSIF(mark >= 40 AND mark < 70) esta si la cumple , por tanto ejecuta dbms_output.put_line('Grade B'); y dbms_output.put_line('Program completed.'); y finaliza el programa.

Bucles anidados

- En PL se pueden incluir unos if dentro de otro, anidando así los if que deseemos



IF <conditionl: returns Boolean>
THEN
—executed only if the condition returns TRUE
<action block1 starts>
IF <condition2: returns Boolean>
THEN
<action_block2>
END IF; —END IF corresponds to condition2
<action_blockl ends>
END IF; —END IF corresponds to condition1

CASE

Es simillar a a IF , pero selecciona un bloque de sentencias en función de la expresión, que ahora no tiene por qué ser un valor Booleano, puede ser un entero, cadena, etc.

El else se ejecuta cuando ninguna de las alternativas es seleccionada.

```
CASE (expression)
 WHEN <valuel> THEN action_blockl;
 WHEN <value2> THEN action_block2;
 WHEN <value3> THEN action_block3;
 ELSE action_block_default;
END CASE;
```

ejemplo:

```
DECLARE
a NUMBER :=55;
b NUMBER :=5;
arth_operation VARCHAR2(20) :='MULTIPLY’;
BEGIN
dbms_output.put_line(‘Program started.' );
CASE (arth_operation)
WHEN ‘ADD’ THEN dbms_output.put_line(‘Addition of the numbers are: '|| a+b );
WHEN ‘SUBTRACT' THEN dbms_output.put_line(‘Subtraction of the numbers are: '||a-b );
WHEN ‘MULTIPLY' THEN dbms_output.put_line(‘Multiplication of the numbers are: '|| a*b
);
WHEN ‘DIVIDE' THEN dbms_output.put_line(‘Division of the numbers are:'|| a/b);
ELSE dbms_output.put_line(‘No operation action defined. Invalid operation');
END CASE;
dbms_output.put_line(‘Program completed.' );
END;
/
EJEMPLO DE CASE Y UPDATE
UPDATE EMP
SET SAL =
    CASE
        WHEN SAL IS NULL THEN 1000  -- Si el salario es NULL, le asignamos un valor
predeterminado
        ELSE SAL * (1 + X_PORCENTAJE / 100)  -- Si el salario no es NULL, aplicamos el
aumento
    END
WHERE EMPNO = N_EMPLEADO;
```

SEARCHED CASE

Es un caso especial del CASE, pero no ponemos expresión en el CASE y las vamos poniendo en el WHEN. Cuando se cumpla una expresión se ejecuta el código asociado y finaliza el CASE.

```
CASE
WHEN <expression1> THEN action_blockl;
WHEN <expression2> THEN action_block2;
WHEN <expression3> THEN action_block3;
ELSE action_block_default;
```

```
END CASE;
```

Ejemplo:

```
DECLARE a NUMBER
:=55; b NUMBER :=5;
arth_operation VARCHAR2(20) :='DIVIDE';
BEGIN
dbms_output.put_line(`Program started.' );
CASE
WHEN arth_operation = 'ADD'
THEN dbms_output.put_line(`Addition of the numbers are: '||a+b );
WHEN arth_operation = `SUBTRACT'
THEN dbms_output.put_line(`Subtraction of the numbers are: '|| a-b);
WHEN arth_operation = `MULTIPLY’
THEN dbms_output.put_line(`Multiplication of the numbers are: '|| a*b );
WHEN arth_operation = ’DIVIDE'
THEN dbms_output.put_line(`Division of the numbers are: '|| a/b ):
ELSE dbms_output.put_line(`No operation action defined. Invalid operation');
END CASE;
dbms_output.put_line(`Program completed.'
); END;
/
```

TIPOS REGISTROS

- Un tipo registro es un tipo de dato complejo , el cual permite al programador agrupar en un registro distintos atributos .

- Con este tipo de datos, podremos agrupar con un único nombre, varias columnas que queramos utilizar.

- Utilizaremos la palabra Type para que el compilador cree un tipo de datos nuevo
- la creación de este nuevo tipo de datos , lo podemos hacer a nivel de toda la base de datos o a nivel de un subprograma, y solo en este nivel se podría crear.
- Para acceder a cada uno de los campos del registro utilizamos . como notación.

Sintaxis

```
CREATE TYPE <type_name_db> IS RECORD (
<column 1> <datatype>,
  );
```

Con esta sintaxis creamos un tipo de datos a nivel de subprograma.

Como recordamos todas las sentencias de PL/SQL terminan con ; y el bloque se ejecuta con / al final.

Ejemplo:

Vamos a ver como crear un tipo registro a nivel de subprograma, es decir, solo está present este registro en este programa, no en la base de datos.

Vamos a crear el tipo registro emp-dept y a continuación nos declaramos la variable empleado, que es una variable de este tipo creado.

Este tipo de datos lo utilizamos exactamente igual que si fuese uno básico , varchar, number, etc

Solución

```
DECLARE
TYPE emp_det IS RECORD (
EMP_NO NUMBER, EMP_NAME VARCHAR2(150), MANAGER NUMBER,
 SAL NUMBER
 );
empleado  emp_det;
BEGIN
empleado.emp_no:= 1001;
empleado.emp_name:='PETER';
empleado.manager:= 1000;
```

```
empleado.sal:=10000;
dbms_output.put_line('Employee Detail');
dbms_output.put_line ('Employee Number: '||empleado.emp_no);
dbms_output.put_line ('Employee Name: '||empleado.emp_name);
dbms_output.put_line ('Employee Salary: ' ||empleado.sal); dbms_output.put_line
('Employee Manager Number: '||empleado.manager); END;
  /
```

COMO CREAR VARIABLES REGISTROS Y CARGARLAS CON DATOS DE TABLAS

```
  DECLARE
TYPE emp_det IS RECORD (
  EMP_NO NUMBER,
  EMP_NAME
  vARCHAR2( 150),
  MANAGER NUMBER,
  SALARY NUMBER
  );
empleado emp_det; BEGIN
INSERT INTO emp (emp_no, emp_name, salary, manager) VALUES (1002,'PETER', 15000,1000);
  COMMIT;
SELECT emp_no, emp_name, salary, manager INTO empleado FROM emp WHERE emp_no=1002;
dbms_output.put_line (‘Employee Detail’);
dbms_output.put_line (‘Employee Number: ‘||empleado. emp_no); dbms_output.put_line (‘Employee
Name: ‘||empleado. emp_name); dbms_output.put_line (‘Employee Salary: '||empleado. salary);
dbms_output.put_line (‘Employee Manager Number: '||empleado.manager); END;
  /
```

Veamos lo que hace

1. como vemos hacemos un insert en emp con los datos de Peter
2. Hacemos una select como veremos más adelante que se hacen en PL/SQL, es decir, recupera un dato que se guarda en la variable empleado
3. A continuación, visualizamos los datos del empleado

```
DECLARE
    -- Definir el tipo de registro
    TYPE emp_registro IS RECORD (
        n_deptno NUMBER
        n_ename  VARCHAR2(70)
    );

    empleado emp_registro;  -- Variable de tipo emp_registro
BEGIN
    -- Seleccionar datos en el registro
    SELECT
        deptno,
        ename
```

```
    INTO
        empleado.n_deptno,  -- Asignar 'deptno' al campo 'n_deptno'
        empleado.n_ename    -- Asignar 'ename' al campo 'n_ename'
    FROM
        emp
    WHERE
        ename = 'ADAMS';  -- Filtrar por 'ename' = 'ADAMS'

    -- Puedes agregar dbms_output para visualizar los resultados
    dbms_output.put_line('Department Number: ' || empleado.n_deptno);
    dbms_output.put_line('Employee Name: ' || empleado.n_ename);

END;
```

Explicación de la corrección:

- Cambié los nombres de los campos en el **registro** para que coincidan con las columnas que vas a seleccionar de la tabla emp (deptno y ename).
- Luego, en la instrucción SELECT INTO, los valores seleccionados (deptno y ename) se asignan explícitamente a los campos del registro (empleado.n_deptno y empleado.n_ename).

Resumen de cambios:

1. Asegúrate de que los campos del tipo **RECORD** coincidan con los nombres de las columnas seleccionadas.
2. Usa **empleado.n_deptno** y **empleado.n_ename** para asignar los valores de las columnas del SELECT INTO.

Este código debería funcionar sin problemas y te permitirá acceder a los datos de la fila seleccionada en el registro empleado.
¿Te queda claro cómo hacer este ajuste?