CONSULTAS

```
SELECT [ DISTINCT ] listaColumnas
FROM listaTablas
[ WHERE condición para filas]
[ GROUP BY listaColumnas por las que se quiere agrupar
[ HAVING condición para los grupos] ]
[ ORDER BY listaColumnas [ ASC | DESC ] ]
```

GROUP BY - HAVING

Utilizando **GROUP BY** se agrupan filas en categorías o grupos, normalmente combinado con funciones de agregación (**COUNT( )**). Esto crea grupos de filas que tienen los mismos valores en una o más columnas.
Ejemplo:

```
SELECT departamento, COUNT(*)
FROM empleados
GROUP BY departamento;
```

Esto agrupa a los empleados por departamento y cuenta cuántos empleados hay en cada uno.

La condición **HAVING** es similar a la cláusula **WHERE** pero filtra grupos en lugar de filas individuales. Solo se puede utilizar con **GROUP BY**.
Actúa como "mientras pase ... muestra (tablas agrupadas en el group by)".

ARITMÉTICAS

```
COUNT( * ) número de filas
COUNT( [DISTINCT] expr ) número de valores distintos en expr
SUM( [DISTINCT] expr ) suma de todos los valores en expr
AVG( [DISTINCT] expr ) promedio de todos los valores en expr
MIN( expr ) el más pequeño de todos los valores en expr
MAX( expr ) el mayor de todos los valores en expr
```

TABLAS

CREATE TABLE

```
CREATE TABLE nombre_tabla (
codigo int PRIMARY KEY,
descripcion varchar(50),
articulo varchar(50),
tipo char(1),
CONSTRAINT ck_nombre_tabla CHECK (tipo in('D','T','E')));

-- CONSTRAINT --
CONSTRAINT pk_nombre_tabla PRIMARY KEY(codigo) PRIMARY KEY
```

```
CONSTRAINT ck_nombre_tabla CHECK (tipo in('D','T','E'))
CONSTRAINT fk_nombre_tabla FOREIGN KEY(articulo) REFERENCES articulo
```

ALTER TABLE

```
CREATE TABLE temporada (
        nombre varchar2(5),
        categoria varchar2(2),
        constraint pk_nombre primary key);

-- MODIFY --
ALTER TABLE temporada MODIFY nombre
        CONSTRAINT ck_temporada CHECK(nombre IN('BAJA','MEDIA','ALTA'))
ALTER TABLE temporada MODIFY categoria DEFAULT 'D';
ALTER TABLE temporada MODIFY nombre varchar2(5) NOT null;
-- DROP --
ALTER TABLE temporada DROP categoria;
-- RENAME --
ALTER TABLE temporada RENAME TO temporadas;
-- ADD --
ALTER TABLE tempordas ADD tipo varchar2(5);
```

INTEGRIDAD REFERENCIAL

Una clave ajena solo puede alojar valores nulos o los que estén previamente almacenados en la clave primaria a la que apuntan.

```
-- Dadas las siguientes categorias insertadas en la tabla
insert into categoria (nombre, descripcion, supmin)
        values ('D', 'SECADOR, MINIBAR, DOS CAMAS, ADMITE SUPLETORIA', 15);
insert into categoria (nombre, descripcion, supmin)
        values ('DT', 'SECADOR, MINIBAR, AMPLIA, DOS CAMAS, ADMITE SUPLETORIA, TERRAZA AL
PARQUE', 15);
insert into categoria (nombre, descripcion, supmin)
        values ('I', 'UNA CAMA, SECADOR',12);

-- Inserciones
insert into habitacion (numero, categoria) values (1, 'D');
insert into habitacion (numero, categoria) values (2, 'D');
insert into habitacion (numero, categoria) values (3, 'D');
insert into habitacion (numero, categoria) values (1, 'I'); -- no se puede, viola la clave
primaria
insert into habitacion (numero, categoria) values (5, 'DT');
insert into habitacion (numero, categoria) values (6, 'DT');

insert into habitacion (numero, categoria) values (5, 'E');
-- no se puede, viola la restricción de integridad referencial de habitacion a categoria.
No existe la categoria 'E' en la tabla categoria
```

ÍNDICES

- Actúan como "filtro" ante ambigüedad en nombres de columnas y reducen la búsqueda solo a un subconjunto de una tabla.

```
CREATE INDEX idx_cliente ON cliente(provincia);
DROP INDEX nom_índice;
```

idx_cliente -> nombre del índice

EXPLAIN PLAN

- Da detalles sobre como se realizará una consulta específica, como el orden de acceso a tablas, los índices que se utilizarán y operaciones de filtro.
- La salida de explain plan se almacena en la tabla **PLAN_TABLE**, la cual contiene:
  - **Statement_id varchar2(30)** Valor opcional para identificar planes de ejecución.
  - **Operation varchar2(30)** Nombre de la operación interna realizada.
  - **Options varchar2(225)** Detalles sobre la operación.
  - **Object_name varchar2(30)** Nombre de la tabla o índice.
  - **Position numeric** Para la primera fila índice al coste estimado por el optimizador para realizar la sentencia, para el resto índice la posición relativa respecto al padre.

```
EXPLAIN PLAN FOR
SELECT column FROM table_name WHERE condition;
```

VISTAS

Son consultas guardadas que se comportan como tablas virtuales. No almacena datos por si misma, sino que almacena una consulta que se ejecuta cada vez que se accede a la vista.

- Es una tabla lógica que no contiene sus propios datos, pero refleja los de la base de datos de manera dinámica.
- Si se modifica la base de datos también se actualizará la vista.
- "Simplificación" de una tabla para su visualización.
- Se pueden hacer consultas de visualización como las SELECT, JOIN, consultas con funciones... pero no de modificación como INSERT, UPDATE, DELETE, agregación sin GROUP BY...

```
CREATE VIEW nombre_de_vista AS
SELECT column FROM nombre_de_tabla WHERE condición;

CREATE VIEW VistaEjemplo AS
SELECT Nombre, Apellido, Edad FROM Empleados WHERE Departamento = 'Ventas';

-- Se puede acceder a los datos de las vistas como si se tratara de una tabla
SELECT * FROM VistaEjemplo;

DROP view VistaEjemplo;
```

CHECK_OPTION

Garantiza que las filas que se insertan o actualizan sigan cumpliendo con las condiciones definidas en la consulta de la vista.
Cuando se crea una vista que tiene ciertas condiciones (por ejemplo, solo mostrar registros que cumplen con una cláusula `WHERE`), el `WITH CHECK OPTION` asegura que cualquier modificación que hagas a través de la vista **no viole esas condiciones**.

- Sin `WITH CHECK OPTION`, podrías insertar o actualizar datos a través de la vista que no cumplan con los criterios de la vista.
- Con `WITH CHECK OPTION`, si intentas insertar o actualizar una fila que no cumple con las condiciones de la vista, la operación fallará. Esto ayuda a evitar que se introduzcan inconsistencias en los datos.

TRIGGERS

- Funcionan como alarmas en la base de datos.
- Sirven para verificar datos o registrar lo que sucede en la base de datos.
- Se asocian con eventos de la base de datos, como INSERT, UPDATE o DELETE a tablas específicas.
- No se llaman, se activan automáticamente cuando ocurre el evento que están monitoreando (alarmas).

```
CREATE TRIGGER nombre_del_trigger
{BEFORE | AFTER} {INSERT | UPDATE | DELETE} ON nombre_de_tabla
FOR EACH ROW
BEGIN
    -- Código del trigger
END;
```

PROCEDIMIENTOS

- Son listas de tareas que se pueden almacenar, guardar y ejecutar en la base de datos.
- Son bloques de código que se almacenan en la base de datos y se llaman desde una aplicación o consulta SQL.
- Pueden aceptar parámetros de entrada y manipular datos, o realizar cálculos o transacciones.
- No devuelven datos.

```
CREATE PROCEDURE nombre_del_procedimiento
(parámetros)
BEGIN
    -- Código del procedimiento
END;
```

FUNCIONES

- Sirven para hacer cálculos o cambios de datos.
- Pueden aceptar parámetros de entrada y manipular datos, o realizar cálculos o transacciones.
- Devuelven datos.

```
CREATE FUNCTION nombre_de_función
(parámetros)
RETURNS tipo_de_dato
BEGIN
```

```
    -- Código de la función
END;
```

MANEJO DE EXCEPCIONES

```
BEGIN
...
EXCEPTION
        WHEN <nombre_excepción> THEN
                <bloque de sentencias>;
        WHEN <nombre_excepción> THEN
                <bloque de sentencias>;
        ...
        [WHEN OTHERS THEN <bloque de sentencias>;]
END;
```

Excepciones predefinidas

- **too_many_rows**: Se produce cuando SELECT … INTO devuelve más de una fila.
- **no_data_found**: se produce cuando un SELECT… INTO no devuelve ninguna fila.
- **value_error**: se produce cuando hay un error aritmético o de conversión.
- **zero_divide**: se puede cuando hay una división entre 0.
- **dupval_on_index**: se crea cuando se intenta almacenar un valor que crearía duplicados en la clave primaria o en una columna con restricción UNIQUE.
- **invalid_number**: se produce cuando se intenta convertir una cadena a un valor numérico.

Excepciones definidas por el usuario

```
DECLARE
        ...
        pasa_tope exception;
BEGIN
        if total>10 then raise pasa_tope; end if;
        ...
EXCEPTION
        when pasa_tope then
                ...
END;
```

PL-SQL

VARIABLES

Declaración de variables

```
auxvalor valor%TYPE;
-- Crea una variable de nombre "auxvalor" con el tipo de dato de "valor"
```

Asignar valor a una variable

```
-- A través de una expresión o a través de una sentencia select
aux_valor:=25;
SELECT categoria INTO auxcategoria FROM habitacion WHERE ...
```

CURSORES

Reúnen las tablas de una consulta en un objeto/tabla con un nombre asignado que se puede recorrer como una tabla normal.

```
DECLARE
CURSOR c1 IS SELECT num, pSA FROM habitación h, pvptemporada p WHERE
h.categoria=p.categoria;
```