Introducción a SQL

Asignatura	CAPÍTULO 3: SQL DDL
Fecha	@20 de septiembre de 2024

Tipos de Datos

AVG: Promedio.

Query Analyzer: to get detailed information about the database queries executed with each service call.

Numéricos

`NUMBER(p,s)`

- El tipo de dato numérico más común en Oracle. Puede almacenar enteros y decimales con precisión.
- `p` **(precisión)**: Número total de dígitos.
- `s` **(escala)**: Número de dígitos después del punto decimal.
- **Uso**: Para cualquier número, incluyendo enteros y decimales.
- **Ejemplo**:

```
salario NUMBER(10, 2) -- Máximo 10 dígitos, con 2 despué
s del punto decimal
```

`INTEGER` / `INT`

- Sinónimo de `NUMBER`, con la precisión ya definida para enteros.
- **Uso**: Para almacenar enteros.
- **Ejemplo**:

```
edad INTEGER
```

Cadenas de texto

`VARCHAR2(n)`

- Almacena una cadena de longitud variable, donde `n` es el número máximo de caracteres. Oracle recomienda el uso de `VARCHAR2` en lugar de `VARCHAR`.
- **Uso**: Para almacenar textos de longitud variable.
- **Ejemplo**:

```
nombre VARCHAR2(50)
```

`CHAR(n)`

- Almacena una cadena de longitud fija. Si el valor tiene menos caracteres de los especificados, se rellena con espacios en blanco.
- **Uso**: Para códigos o identificadores de longitud fija.
- **Ejemplo**:

```
codigo CHAR(5)
```

`CLOB` (Character Large Object)

- Almacena grandes cantidades de texto.
- **Uso**: Para textos largos como descripciones o artículos.
- **Ejemplo**:

```
descripcion CLOB
```

Fechas y Tiempos

`DATE`

- Almacena tanto la fecha como la hora (día, mes, año, horas, minutos y segundos).

- **Uso**: Para cualquier dato relacionado con fecha y hora.
- **Ejemplo**:

```
fecha_nacimiento DATE
```

`TIMESTAMP`

- Similar a `DATE`, pero con mayor precisión para fracciones de segundo.
- **Uso**: Para almacenar información detallada de fecha y hora.
- **Ejemplo**:

```
fecha_registro TIMESTAMP
```

Booleanos

- **Oracle no tiene un tipo** `BOOLEAN` en SQL estándar. Sin embargo, se puede simular usando otros tipos de datos como `CHAR(1)` o `NUMBER(1)` con valores como `0/1` o `Y/N`.
- **Ejemplo**:

```
es_activo CHAR(1) CHECK (es_activo IN ('Y', 'N'))
```

Otros tipos

`BLOB` (Binary Large Object)

- Almacena datos binarios como imágenes, videos o documentos.
- **Uso**: Para manejar grandes archivos binarios.
- **Ejemplo**:

```
sql
Copiar código
imagen BLOB
```

Restricciones en Oracle

En SQL, un **constraint** (restricción) es una regla aplicada a una columna o conjunto de columnas en una tabla para garantizar la integridad y validez de los datos. Los constraints definen ciertas condiciones que los datos deben cumplir al insertarse o modificarse, y ayudan a mantener la consistencia en una base de datos.

Algunos tipos comunes de **constraints** en SQL son:

`NOT NULL`

- Impide que una columna acepte valores nulos.
- **Ejemplo**:

```
nombre VARCHAR2(50) NOT NULL
```

`UNIQUE`

- Asegura que todos los valores de una columna o grupo de columnas sean únicos.
- **Ejemplo**:

```
correo VARCHAR2(100) UNIQUE
```

`PRIMARY KEY`

- Define una columna o combinación de columnas que identifican de manera única cada fila en una tabla. Implica `UNIQUE` y `NOT NULL` (no permite valores duplicados ni nulos).
- **Ejemplo**:

```
id_empleado NUMBER PRIMARY KEY
```

`FOREIGN KEY`

- Crea una relación entre dos tablas, vinculando una columna en la tabla actual con la clave primaria de otra tabla.

- **Ejemplo**:

```
id_departamento NUMBER,
CONSTRAINT fk_departamento FOREIGN KEY (id_departamento)
REFERENCES DEPARTAMENTO(id)
```

`REFERENCES`

- Se usa para crear una `FK` que establece una relación entre dos tablas, asegurando que los valores de una columna en una tabla existan en la columna referenciada de otra tabla.
- **Ejemplo**:

```
CREATE TABLE PERSONA
( percod number primary key,
perapp varchar2(20) not null,
perapm varchar2(20) not null,
pernom varchar2(20) not null,
persex varchar2(1) not null,
perfna date,
discod number,
constraint fk_discod foreign key (discod) references DISTR
```

`ON DELETE CASCADE`

- Elimina automáticamente las filas relacionadas en una tabla secundaria cuando se elimina una fila en la tabla principal.
- **Ejemplo**:

```
CREATE TABLE PERSONA
( percod number primary key,
perapp varchar2(20) not null,
perapm varchar2(20) not null,
pernom varchar2(20) not null,
persex varchar2(1) not null,
perfna date,
discod number,
```

```
constraint fk_discod foreign key (discod) references DISTR
on delete cascade );
```

`ON DELETE SET NULL`

- Significa que cuando se elimina una fila de la tabla principal, las claves foráneas en la tabla relacionada se vuelven `NULL`. En lugar de eliminar las filas relacionadas, simplemente se eliminan los vínculos entre ellas.
- **Ejemplo**:

```
CREATE TABLE PERSONA
( percod number primary key,
perapp varchar2(20) not null,
perapm varchar2(20) not null,
pernom varchar2(20) not null,
persex varchar2(1) not null,
perfna date,
discod number,
constraint fk_discod foreign key (discod) references DISTR
(discod) on delete set null );
```

`CHECK`

- Impone una condición sobre los valores que pueden almacenarse en una columna.
- **Ejemplo**:

```
salario NUMBER(10, 2) CHECK (salario >= 0)
```

```
CREATE TABLE PERSONA
( percod number primary key,
perapp varchar2(20) not null,
perapm varchar2(20) not null,
pernom varchar2(20) not null,
persex varchar2(1) not null,
perfna date,
```

```
perpeso number,
constraint ch_perpeso CHECK (perpeso > 0) ) ;
```

DEFAULT

- Define un valor por defecto para una columna.
- **Ejemplo**:

```
estado VARCHAR2(10) DEFAULT 'activo'
```

Ejemplo de cómo se implementa un **constraint**:

```
CREATE TABLE usuarios (
id INT PRIMARY KEY,
nombre VARCHAR(50) NOT NULL,
email VARCHAR(100) UNIQUE,
edad INT CHECK (edad >= 18)
);
```

- El campo `id` es la clave primaria.
- El campo `nombre` no puede ser nulo.
- El campo `email` debe tener valores únicos.
- El campo `edad` debe ser mayor o igual a 18.

En SQL, puedes utilizar explícitamente la palabra **CONSTRAINT** para definir una restricción con nombre, lo que es útil para identificarla o modificarla más adelante. Aquí te dejo un ejemplo:

```
CREATE TABLE empleados (
  id INT,
  nombre VARCHAR(50),
  email VARCHAR(100),
  edad INT,
  salario DECIMAL(10, 2),
```

```
  CONSTRAINT pk_empleados_id PRIMARY KEY (id),
  CONSTRAINT u_email UNIQUE (email),
  CONSTRAINT chk_edad CHECK (edad >= 18),
  CONSTRAINT chk_salario CHECK (salario > 0)
);
```

- `pk_empleados_id` es el nombre de la restricción de **PRIMARY KEY** aplicada a la columna `id`.
- `u_email` es el nombre de la restricción **UNIQUE** en la columna `email` para garantizar que no haya correos duplicados.
- `chk_edad` es una restricción **CHECK** que asegura que la edad sea mayor o igual a 18.
- `chk_salario` es otra restricción **CHECK** que asegura que el salario sea un número positivo.

Nombrar las restricciones con **CONSTRAINT** permite gestionarlas más fácilmente, como eliminarlas o modificarlas en el futuro.

Comandos para gestión de tablas

`CREATE TABLE`

- Comando para crear una tabla.
- Sintaxis:

```
CREATE TABLE nombre de la tabla
( columnas );
```

- Ejemplo:

```
CREATE TABLE DISTRITO
( discod number primary key,
disnom varchar2(20) not null );
```

```
CREATE TABLE PERSONA
( percod number primary key,
```

```
perapp varchar2(20) not null,
perapm varchar2(20) not null,
pernom varchar2(20) not null,
persex varchar2(1) not null,
perfna date ) ;
```

```
CREATE TABLE PROFESION
( procod number primary key,
pronom varchar2(20) not null );
```

```
CREATE TABLE TELEFONO
( percod number,
pertel varchar2(10),
primary key ( percod, pertel ),
constraint fk_percod foreign key (percod) references PERSO
```

```
CREATE TABLE PERSONA_PROFESION
( percod number,
procod number,
añotit number,
primary key ( percod, procod ),
constraint fk_percod foreign key (percod) references PERSO
constraint fk_procod foreign key (procod) references PROFE
```

DESCRIBE

- Comando para ver la estructura de una tabla. Muestra información sobre las columnas de la tabla, como sus nombres, tipos de datos, si permiten valores nulos, claves, entre otros detalles.
- Sintaxis:

```
DESCRIBE nombre de tabla;
```

- Ejemplo:

```
DESCRIBE PERSONA;
```

ALTER TABLE

Se utiliza para modificar la estructura de una tabla existente.

ADD

- Comando para agregar una nueva columna a una tabla.
- Sintaxis:

```
ALTER TABLE nombre de tabla
ADD nombre de columna tipo restricción;
```

- Ejemplo:

```
ALTER TABLE PERSONA
ADD pertalla number;
```

```
ALTER TABLE PERSONA
ADD año number;
```

MODIFY

- Comando para modificar una columna de una tabla.
- Sintaxis:

```
ALTER TABLE nombre de tabla
MODIFY nombre de columna tipo;
```

- Ejemplo:

```
ALTER TABLE PERSONA
MODIFY pernom varchar2(30);
```

RENAME COLUMN

- Comando para cambiar el nombre de una columna de una tabla.
- Sintaxis:

```
ALTER TABLE nombre de tabla
RENAME COLUMN nombre actual de columna TO nuevo nombre;
```

- Ejemplo:

```
ALTER TABLE PERSONA
RENAME COLUMN percor TO permail;
```

`DROP COLUMN`

- Comando para eliminar una columna de una tabla.
- Sintaxis:

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

- Ejemplo:

```
ALTER TABLE PERSONA
DROP COLUMN perpeso;
```

`ADD CONSTRAINT`

- Comando para agregar una restricción en una tabla.
- Sintaxis:

```
ALTER TABLE nombre de tabla
ADD CONSTRAINT restricción;
```

- Ejemplo:

```
ALTER TABLE PERSONA
ADD CONSTRAINT ch_pertalla CHECK (pertalla > 1.65);
```

DROP CONSTRAINT

- Comando para eliminar una restricción en una tabla.
- Sintaxis:

```
ALTER TABLE nombre de tabla
DROP CONSTRAINT nombre de restricción;
```

- Ejemplo:

```
ALTER TABLE PERSONA
DROP CONSTRAINT ch_pertalla;
```

DROP TABLE

- Comando para eliminar una tabla y todos sus datos de forma permanente de la base de datos.

CASCADE CONSTRAINTS

- Asegura que se eliminen también las restricciones relacionadas, evitando conflictos.
- Sintaxis:

```
DROP TABLE nombre de tabla CASCADE CONSTRAINTS;
```

- Ejemplo:

```
DROP TABLE DISTRITO CASCADE CONSTRAINTS;
```

RENAME

- Comando para renombrar una tabla.
- Sintaxis:

```
RENAME nombre de tabla TO nuevo nombre;
```

- Ejemplo:

```
RENAME PERSONA TO PEOPLE;
```

`COMMENT ON COLUMN`

- Comando para agregar un comentario descriptivo a una columna específica de una tabla.
- Sintaxis:

```
COMMENT ON COLUMN nombre_tabla.nombre_columna IS 'Descripc
```

- Ejemplo:

```
COMMENT ON COLUMN persona.perapp IS 'Apellido paterno de l
```

Sentencias

`INSERT`

- Se utiliza para insertar nuevas filas en una tabla.
- Sintaxis:

```
INSERT INTO tabla
VALUES ( valor1, valor2, ..., valorN )
```

```
INSERT INTO tabla
( col1, col2, ..., colN )
VALUES ( valor1, valor2, ..., valorN )
```

Inserción de datos para todas las columnas

```
INSERT INTO persona
( percod, perapp, perapm, pernom, persex, perfna, perpeso,
pertalla, percor, discod, año )
VALUES ( 52, 'Valdez', 'Durand', 'Dora' , 'F',
```

```
                to_date('02-07-1991', 'DD-MM-YYYY'), 59,
                1.62, 'dvaldez@gmail.com', 5, 2 );
```

Inserción de datos para algunas columnas

```
INSERT INTO persona
( percod, perapp, perapm, pernom, persex, perpeso )
VALUES ( 53, 'Miranda', ‘Flores', ‘Roberto' , ‘M', 73 );
```

WHERE

- Se utiliza para filtrar registros en una consulta, seleccionando solo aquellos que cumplen con una condición específica
- Sintaxis:

```
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

UPDATE

- Se utiliza para modificar valores en una tabla.
- Sintaxis:

```
UPDATE tabla
SET col1 = valor1, col2 = valor2, ... colN = valorN
WHERE condición
```

  - La cláusula **SET** establece los nuevos valores para las columnas indicadas.
  - La cláusula **WHERE** sirve para seleccionar las filas que se quiere modificar.
    - Si se omite **WHERE** se modificará los valores en todas las filas de la tabla.

Asignar la fecha de nacimiento 17 de mayo de 1999 y el sueldo 3400 soles a la persona cuyo código es 35

```
UPDATE persona
SET perfna = TO_DATE('17-05-1999', 'DD-MM-YYYY'),
        persue = 3400
WHERE percod = 35
```

Incrementar en 100 soles el sueldo de todas las mujeres

```
UPDATE persona
SET persue = persue + 100
WHERE persex = 'F'
```

`TO_DATE`

- Es una función utilizada para convertir una cadena de texto en un valor de tipo fecha. Es especialmente útil para asegurarse de que las fechas se interpreten correctamente en el formato deseado.

Asignar la fecha de nacimiento 17 de mayo de 1999 y el sueldo 3400 soles a la persona cuyo código es 35

```
UPDATE persona
SET perfna = TO_DATE('17-05-1999', 'DD-MM-YYYY'),
        persue = 3400
WHERE percod = 35
```

`DELETE`

- Se utiliza para eliminar filas de una tabla.
- Sintaxis:

  ```
  DELETE FROM tabla
  WHERE condición
  ```

  - La cláusula **WHERE** sirve para seleccionar las filas que se quiere eliminar.
    - Si se omite **WHERE** se eliminarán todas las filas de la tabla.

Eliminar todas las filas de la tabla PERSONA

```
DELETE FROM persona
```

Eliminar los datos de la persona con código 27

```
DELETE FROM persona
WHERE percod = 27
```

Eliminar los datos de las personas que ganan más de 4000 soles

```
DELETE FROM persona
WHERE persue > 4000
```

SELECT

- Es una de las sentencias SQL más importantes, ya que permite realizar consultas sobre los datos almacenados en la base de datos.
- Sintaxis:

  ```
  SELECT * ó Lista de columnas
  FROM tablas
  WHERE condición
  ORDER BY columnas
  ```

  - Se usa * cuando se quiere visualizar todas las columnas.
  - La cláusula **FROM** permite indicar las tablas que serán consultadas.
  - La cláusula **WHERE** permite especificar la condición que deben cumplir las filas que serán consideradas en la consulta.
  - La cláusula **ORDER BY** permite ordenar el resultado de la consulta en base a una o varias columnas.

Mostrar todos los datos de todas las personas

```
SELECT *
FROM persona
```

Mostrar todos los datos de las mujeres

```
SELECT *
FROM persona
WHERE persex = 'F'
```

MAX

- Es una función de agregación que se utiliza para obtener el valor máximo de una columna específica en un conjunto de registros.
- Sintaxis:

```
SELECT MAX(nombre_columna)
FROM nombre_tabla
WHERE condiciones;
```

- Ejemplo:

```
SELECT MAX(persue) AS Sueldo_Maximo
FROM persona;
```

MIN

Se utiliza para obtener el valor mínimo de una columna específica en un conjunto de registros.

- Sintaxis:

```
SELECT MIN(nombre_columna)
FROM nombre_tabla
WHERE condiciones;
```

- Ejemplo:

```
SELECT MIN(persue) AS Sueldo_Minimo
FROM persona;
```

ORDER BY

- Ordena los resultados de una consulta en función de una o más columnas. Permite especificar si el orden será ascendente (`ASC`, que es el valor por defecto) o descendente (`DESC`).

Mostrar todos los datos de las personas que ganan más de 3000 soles ordenados por apellido paterno

```
SELECT *
FROM persona
WHERE persue > 3000
ORDER BY perapp
```

AS

- Asigna un alias a una columna o tabla en los resultados de una consulta. Esto permite dar nombres más descriptivos o legibles a las columnas resultantes.

Mostrar apellidos y nombres de las personas. Cada columna debe tener un título

```
SELECT perapp AS Apellido_Paterno,
            perapm AS Apellido_Materno,
            pernom AS Nombre
FROM persona
```

DISTINCT

- Elimina duplicados de los resultados de una consulta, asegurando que cada valor en la columna especificada aparezca solo una vez.

Mostrar todos los sueldos. Si hay sueldos repetidos solo deben mostrarse una sola vez. Los sueldos deben mostrarse de mayor a menor

```
SELECT DISTINCT persue
FROM persona
ORDER BY persue DESC
```

COUNT

- Se utiliza para contar el número de filas en un conjunto de resultados. Puede contar todas las filas, o solo las filas que tienen un valor no nulo en una columna específica.
- Sintaxis:

```
SELECT COUNT([DISTINCT] columna)
FROM tabla
WHERE condición;
```

- Ejemplos
  - Contar todas las filas de una tabla

```
SELECT COUNT(*) AS total_personas
FROM persona;
```

  - Contar valores no nulos en una columna

```
SELECT COUNT(persue) AS personas_con_sueldo
FROM persona;
```

  - Contar valores distintos en una columna

```
SELECT COUNT(DISTINCT perdist) AS distritos_distintos
FROM persona;
```

`GROUP BY`

- Se utiliza para agrupar filas que tienen los mismos valores en columnas específicas, y es comúnmente usada junto con funciones agregadas como `COUNT`, `SUM`, `AVG`, `MAX`, o `MIN`. De esta forma, puedes obtener resultados agregados (resumidos) para cada grupo de filas que comparten un valor común.
- Sintaxis:

```
SELECT column1, AGGREGATE_FUNCTION(column2)
FROM table_name
```

```
GROUP BY column1;
```

Se desea hallar el impuesto total que se debe pagar por los sueldos, agrupados por departamento. El impuesto es el 10% del sueldo.

```
SELECT a.IdDepartamento||'-'||b.Ndepartamento as Departamento
FROM Persona a, Departamento b
WHERE a.IdDepartamento = b.IdDepartamento
GROUP BY a.IdDepartamento||'-'||b.Ndepartamento;
```

`HAVING`

- Se utiliza para filtrar los resultados de una consulta después de que se han aplicado las funciones agregadas y la agrupación mediante `GROUP BY`. A diferencia de `WHERE`, que filtra las filas antes de la agrupación, `HAVING` filtra los grupos creados por `GROUP BY`.
- Sintaxis:

```
SELECT column1, AGGREGATE_FUNCTION(column2)
FROM table_name
GROUP BY column1
HAVING condition;
```

¿Cuál es el tamaño de pedido promedio para cada vendedor (REP) cuyos pedidos totalizan más de $30,000?

```
SELECT REP, AVG(IMPORTE)
FROM PEDIDOS
GROUP BY REP
HAVING SUM(IMPORTE) > 30,000
```

`OUTER JOIN` (Conjunción lateral)

- **(+)** se utiliza para indicar de qué lado de la combinación externa (outer join) se deben mostrar las filas cuando no hay coincidencias.

  `LEFT OUTER JOIN`

  - Si pones el `(+)` en una columna de la tabla derecha.

- Sintaxis:

```
SELECT columnas
FROM tabla1, tabla2
WHERE tabla1.columna = tabla2.columna(+);
```

Supongamos que deseas obtener todos los registros de la tabla `persona` aunque no haya coincidencia en la tabla `distritos`.

```
SELECT p.perapp, p.perapm, p.pernom, d.distrito_nombre
FROM persona p, distritos d
WHERE p.perdist = d.distrito_id(+);
```

`RIGHT OUTER JOIN`

- Si el `(+)` está en la columna de la tabla izquierda.
- Sintaxis:

```
SELECT columnas
FROM tabla1, tabla2
WHERE tabla1.columna(+) = tabla2.columna;
```

Si se quiere obtener todas las filas de la tabla `distritos`, aunque no haya coincidencias en la tabla `persona`.

```
SELECT p.perapp, p.perapm, p.pernom, d.distrito_nombre
FROM persona p, distritos d
WHERE p.perdist(+) = d.distrito_id;
```

DECODE

- Se utiliza para realizar comparaciones condicionales, similar a una declaración `IF-THEN-ELSE`. Es una función que evalúa una expresión y devuelve un valor basado en el resultado de la evaluación, comparando el valor con una lista de posibles resultados.
- Sintaxis:

```
DECODE(expresión, valor1, resultado1, valor2, resultado2,
-- valor_default: OPCIONAL
```

Ejemplos:

```
SELECT pernom, persue,
       DECODE(persue,
              1000, 'Bajo',
              2000, 'Medio',
              3000, 'Alto',
              'Desconocido') AS nivel_sueldo
FROM persona;
```

```
SELECT pernom,
       DECODE(persue, NULL, 'Sin salario', persue) AS salario
FROM persona;
```

CASE

- La sentencia `CASE` permite realizar las mismas operaciones que las sentencias de control PL/SQL `IF-THEN-ELSIF-ELSE` pero con la particularidad de que pueden utilizarse dentro de una sentencia SQL (`SELECT`, `UPDATE`, etcétera).
- Sintaxis:

```
CASE [ expresion ]
WHEN condicion_1 THEN resultado_1
WHEN condicion_2 THEN resultado_2
...
WHEN condicion_n THEN resultado_n
ELSE resultado
END CASE;
```

- Ejemplos:

```
SELECT pernom, persue,
       CASE persue
           WHEN 1000 THEN 'Bajo'
           WHEN 2000 THEN 'Medio'
           WHEN 3000 THEN 'Alto'
           ELSE 'Desconocido'
       END AS nivel_sueldo
FROM persona;
```

```
SELECT pernom, persue,
       CASE
           WHEN persue < 1000 THEN 'Bajo'
           WHEN persue BETWEEN 1000 AND 2000 THEN 'Medio'
           WHEN persue > 2000 THEN 'Alto'
           ELSE 'Desconocido'
       END AS nivel_sueldo
FROM persona;
```

```
SELECT pernom, perdist,
       CASE
           WHEN perdist = 1 THEN 'Distrito 1'
           WHEN perdist = 2 THEN 'Distrito 2'
           WHEN perdist = 3 THEN 'Distrito 3'
           ELSE 'Otro distrito'
       END AS distrito_nombre
FROM persona;
```

```
SELECT perdist,
       CASE
           WHEN AVG(persue) > 5000 THEN 'Alto'
           WHEN AVG(persue) BETWEEN 3000 AND 5000 THEN 'Me
           ELSE 'Bajo'
       END AS nivel_promedio_sueldo
FROM persona
GROUP BY perdist;
```

INDEX

- Es una estructura que acelera las consultas al permitir acceso rápido a los datos sin recorrer toda la tabla. Los índices se crean en columnas clave y mejoran el rendimiento de consultas, pero pueden ralentizar las operaciones de escritura.
- **Sintaxis:**

```
CREATE INDEX nombre_indice ON nombre_tabla (columna1, co
lumna2);
```

- **Ejemplo:**

```
CREATE INDEX idx_fecha_venta ON ventas (fecha_venta);
```

  Esto acelera las consultas que filtran por `fecha_venta` en la tabla `ventas`.

VIEW

- Es una tabla virtual basada en el resultado de una consulta. No almacena datos, sino que muestra datos en tiempo real desde otras tablas. Las vistas simplifican consultas complejas, mejoran la seguridad y permiten filtrar o reorganizar la información.
- **Sintaxis:**

```
CREATE VIEW nombre_vista AS
SELECT columnas
FROM tabla
WHERE condición;
```

- Para eliminar una vista:

```
DROP VIEW view_name [CASCADE CONSTRAINT];
```

- **Ejemplo:**

```
CREATE VIEW vista_clientes_activos AS
SELECT nombre, estado
```

```
FROM clientes
WHERE estado = 'activo';
```

Consulta a la vista:

```
SELECT * FROM vista_clientes_activos;
```

SEQUENCE

- Se utiliza para generar números enteros secuenciales únicos, generalmente para claves primarias. Garantiza que no haya valores repetidos y puede configurarse para aumentar o disminuir automáticamente.
- **Sintaxis:**

```
CREATE SEQUENCE nombre_secuencia
START WITH valor_inicial
INCREMENT BY valor_incremento
MAXVALUE valor_máximo
MINVALUE valor_mínimo
CYCLE | NOCYCLE;
```

- **Ejemplo:**

```
CREATE SEQUENCE seq_codigo_cliente
START WITH 1
INCREMENT BY 1
MAXVALUE 99999
MINVALUE 1;
```

Para usar la secuencia:

```
INSERT INTO CLIENTE (codigo, nombre)
VALUES (seq_codigo_cliente.NEXTVAL, 'Tienda Azul S.A');
```

Este código genera automáticamente un valor único para la columna `codigo` usando la secuencia.

SUBQUERY

Una **subquery** (subconsulta) es una consulta dentro de otra consulta más grande (consulta principal). Se utiliza para devolver datos que luego serán utilizados por la consulta principal. Las subqueries se colocan dentro de una sentencia `SELECT`, `INSERT`, `UPDATE`, o `DELETE` y pueden estar en cláusulas como `WHERE`, `FROM`, o `SELECT`.

Tipos de subqueries:

1. **Subquery escalar**: Devuelve un solo valor.
2. **Subquery correlacionada**: Depende de la consulta externa para su ejecución.
3. **Subquery anidada**: Se coloca dentro de otras subqueries.

Sintaxis:

```
SELECT columna1, columna2
FROM tabla1
WHERE columna1 = (SELECT columna3
                                    FROM tabla2
                                    WHERE condición);
```

Ejemplo:

Obtener los empleados cuyo salario es mayor al promedio de la compañía:

```
SELECT nombre, salario
FROM empleados
WHERE salario > (SELECT AVG(salario)
                                    FROM empleados);
```

Aquí, la subquery devuelve el salario promedio, y la consulta principal selecciona a los empleados cuyo salario es mayor que ese valor.

Operadores

- Las condiciones de la cláusula `WHERE` se pueden combinar usando los operadores `AND` y `OR`.

- Los operadores `AND` y `OR` se utilizan para filtrar registros en función de más de una condición.

`AND`

- El operador `AND` muestra una fila si **todas las condiciones** separadas por `AND` son **verdaderas**.

Mostrar todos los datos de las mujeres que ganan menos de 2500 soles

```
SELECT *
FROM persona
WHERE persex = 'F' AND persue < 2500
```

`OR`

- El operador `OR` muestra una fila si **alguna de las condiciones** separadas por `OR` es **verdadera**.

Mostrar apellidos y nombre de todos los que se llaman Jorge o Juan

```
SELECT perapp, perapm, pernom
FROM persona
WHERE pernom = 'Juan' OR pernom = 'Jorge'
```

`BETWEEN`

- El operador `BETWEEN` selecciona valores dentro de un rango dado.
- Los valores pueden ser números, texto o fechas.

Mostrar apellidos, nombre, fecha de nacimiento y sueldo de las personas que ganan desde 2000 hasta 4000 soles

```
SELECT perapp, perapm, pernom, perfna, persue
FROM persona
WHERE persue BETWEEN 2000 AND 4000
```

`[NOT] IN`

- El operador `IN` permite especificar un conjunto de valores en una cláusula `WHERE` evitando el uso de múltiples condiciones `OR`.

Mostrar apellidos, nombre y sueldo de las mujeres cuyo apellido paterno sea Torres, Cuba, Diaz o Vargas

```
SELECT perapp, perapm, pernom, persue
FROM persona
WHERE persex = 'F' AND
            perapp IN ( 'Torres', 'Cuba', 'Diaz', 'Vargas' )
```

`[NOT] BETWEEN x AND y`

- El operador `BETWEEN` comprueba valores entre un rango determinado y devuelve los valores coincidentes. La condición `BETWEEN` es inclusiva. Los valores inicial y final están incluidos.

```
SELECT *
FROM persona
WHERE persue BETWEEN 2000 AND 5000;
```

`EXISTS`

- El operador `EXISTS` devuelve verdadero si la subconsulta devuelve alguna fila, de lo contrario, devuelve falso.

```
SELECT perapp, perapm, pernom, persue, perdist
FROM persona p
WHERE EXISTS (
    SELECT *
    FROM empleo e
    WHERE e.percod = p.percod
);
```

`||`

- El operador `||` permite concatenar o juntar varios textos.

Mostrar apellidos y nombre de las personas, todo junto en una sola columna. Ordenar por apellido paterno.

```
SELECT perapp || ' ' || perapm || ' ' || pernom AS "nombre_co
FROM persona
ORDER BY perapp
```

`x [NOT] LIKE`

- El operador `LIKE` se usa en una cláusula `WHERE` para buscar un patrón específico en una columna.
- Hay dos comodines que se usan con frecuencia junto con el operador `LIKE`:
  - `%` representa cero, uno o varios caracteres
  - `_` representa un solo carácter

Mostrar apellidos y nombre de las personas cuyo apellido paterno empieza con la letra F

```
SELECT perapp, perapm, pernom
FROM persona
WHERE perapp LIKE 'F%'
```

Mostrar apellidos y nombre de las personas cuyo nombre tiene una a en la penúltima posición

```
SELECT perapp, perapm, pernom
FROM persona
WHERE pernom LIKE '%a_'
```

`IS [NOT] NULL`

- El operador `NULL` comprobar si un valor en una columna o una expresión es `NULL` o no.

```
SELECT *
FROM persona
WHERE persue IS NULL; -- WHERE persue IS NOT NULL
```

`ROWNUM/SELECT TOP`

- El operador `ROWNUM` es una pseudo-columna que asigna un número secuencial a cada fila devuelta por una consulta. Saca tops (5/10/15 primeros datos). Esta función actualiza sus valores usando subconsultas de modo que la consulta:

```
SELECT * FROM (
    SELECT ROWNUM AS row_num, perapp, perapm, pernom, persue,
    FROM persona
    ORDER BY persue DESC
)
WHERE row_num <= 5;
```

Manejo de fechas

Funciones para el manejo de fechas

- **extract(year from fecha):** retorna el año de una fecha
- **extract(month from fecha):** retorna el mes de una fecha
- **extract(day from fecha):** retorna el día de una fecha

Mostrar apellidos, nombre y fecha de nacimiento de las personas que nacieron el mismo día de navidad después del año 1996

```
SELECT perapp, perapm, pernom, perfna
FROM persona
WHERE extract( day from perfna ) = 25 AND
            extract( month from perfna ) = 12 AND
            extract( year from perfna ) > 1996
```

Mostrar apellidos, nombre y fecha de nacimiento de las personas que nacieron en febrero, abril, junio, agosto o noviembre, ordenado por fecha de nacimiento

```
SELECT perapp, perapm, pernom, perfna
FROM persona
```

```
WHERE extract( month from perfna) in ( 2, 4, 6, 8, 11 )
ORDER BY perfna
```

Expresiones aritméticas con fechas

Desde que las bases de datos almacenan fechas como números, se pueden realizar operaciones usando operadores aritméticos como sumas y restas.

Operación	Resultado	Descripción
Fecha + número	Fecha	Añade un número de días a una fecha.
Fecha - número	Fecha	Resta un número de días a una fecha.
Fecha - Fecha	Número de días	Resta una fecha a otra.
Fecha + número / 24	Fecha	Añade un número de horas a una fecha.

Es posible dar formato a los valores de fecha y de campos numéricos empleando una función de conversión de caracteres y elementos de formato. Sintaxis:

```
TO_CHAR(ColumnaFecha,‘ElementosFormato’)
```

```
TO_CHAR(ColumnaNumérica,‘ElementosFormato’)
```

El valor retornado siempre será un VARCHAR2.

Elementos que dan formato a fechas	Sintaxis	Ejemplo
Día del mes (01 - 31)	DD	1 - 31
Día de la semana (tres primeras letras)	DY	THU
Nombre en mayúsculas del día de la semana (9 caracteres)	DAY	TUESDAY
Nombre en mayúsculas del número del día del mes	DDSPTH	TWELFTH
Mes (01 - 12)	MM	01 - 12
Nombre en mayúsculas del mes (9 caracteres)	MONTH	JANUARY
Año de dos dígitos	YY	91

Elementos que dan formato a fechas	Sintaxis	Ejemplo
Año de cuatro dígitos	YYYY	2011
Horas, minutos y segundos	HH:MM:SS	09:00:00
Supresión de blancos en elementos subsiguientes	FM	

Funciones

Funciones aritméticas

DESCRIPCIÓN	FUNCIÓN	EJEMPLO
Devuelve la potencia absoluta del valor numérico.	ABS(valor numérico)	ABS(-1)
Devuelve el entero más grande que no sea mayor que el valor numérico	FLOOR(valor numérico)	FLOOR(-1,2)
Módulo o resto	MOD(dividendo, divisor)	MOD(7,5) es 2
Raíz cuadrada	SQRT (valor numérico)	SQRT(25) es 5
Redondeo	ROUND(valor numérico, precisión)	ROUND(monto, 2) monto redondeado a 2 decimales
Truncamiento	TRUNC(valor numérico, precisión)	TRUNC(monto, 2) monto truncado a 2 decimales
Potencia	POWER(valor numérico, potencia)	POWER(monto, 3) monto elevado al cubo

Funciones de fecha y hora

DESCRIPCIÓN	FUNCIÓN	EJEMPLO
Agregando meses	ADD_MONTHS (Columnafecha, ±Núméromeses)	ADD_MONTHS(fechaingreso,-6) devuelve 6 meses antes de la fecha de ingreso
Último día del mes de la fecha	LAST_DAY(ColumnaFecha)	LAST_DAY(fechaingreso) devuelve el último día del mes en que ingresó
Siguiente día específico de la semana después de la fecha	NEXT_DAY(ColumnaFecha, DiaSemana)	NEXT_DAY(fechaingreso, ‘ FRIDAY’) devuelve el primer viernes después de que ingresó
Número de meses entre dos fechas	MONTHS_BETWEEN(ColFecha1, ColFecha2)	MONTHS_BETWEEN(sysdate, fechaingreso) devuelve los meses trabajados
Fecha actual	SYSDATE	SYSDATE() devuelve la fecha actual

Funciones de cadena

DESCRIPCIÓN	FUNCIÓN	EJEMPLO
Devolver la primera letra en mayúscula y el resto en minúscula	INITCAP(ColumnaCadena)	INITCAP(nombre) devuelve la primera letra de los nombres en mayúscula y el resto en minúscula
Devolver todas en mayúsculas	UPPER(ColumnaCadena)	UPPER(nombre) devuelve todos los caracteres del nombre en mayúscula
Devolver todas en minúsculas	LOWER(ColumnaCadena)	LOWER(nombre) devuelve todos los caracteres del nombre en minúscula
Devolver desde la posición dada N caracteres	SUBSTR(ColumnaCadena, Posición,N)	SUBSTR(empleo,1,3) devuelve desde la primera letra 3 caracteres del empleo
Devolver el número de caracteres de una cadena	LENGTH(ColumnaCadena)	LENGTH(nombre) devuelve el número de caracteres del nombre
El mayor entre dos valores	GREATEST(Columna1, Columna2)	GREATEST(sal,comi) devuelve el mayor entre salario y comisión
El menor entre dos valores	LEAST(Columna1,Columna 2)	LEAST('Adam','Smith') devuelve el menor: Adam

Funciones de agregación

DESCRIPCIÓN	FUNCIÓN	EJEMPLO
Devuelve el número de filas para las que una o más expresiones proporcionadas son no nulas	COUNT(valor)	COUNT(expresión)
Devuelve el número total de filas recuperadas, incluidas las filas que contienen un valor nulo	COUNT(*)	COUNT(*)
Devuelve el valor máximo del argumento	MAX(valor numérico)	MAX(valorCuota)
Devuelve el valor mínimo del argumento	MIX(valor numérico)	MIN(valorCuota)
Devolver el número de caracteres de una cadena	SUM(valor numérico)	SUM(totalVentas)
Devuelve la media de valores numéricos en una expresión	AVG(valor numérico)	AVG(precioVenta)

NVL

- El operador `NVL` se utiliza para reemplazar un valor `NULL` con otro valor. Es muy útil para asegurarse de que un resultado no sea `NULL` cuando se realizan operaciones o se muestran datos.
- Sintaxis:

```
NVL(expression, replacement_value)
```

- Ejemplo:

```
SELECT perapp, perapm, pernom, NVL(persue, 0)
FROM persona;
```

```
SELECT perapp, perapm, NVL(pernom, 'No especificado') AS n
FROM persona;
```

Consultas multitabla

Permiten recuperar datos de más de una tabla al mismo tiempo, utilizando **uniones** o combinaciones basadas en relaciones entre las tablas.

Mostrar apellidos y nombres de las personas, el distrito donde viven y desde qué año viven en él

```
SELECT perapp, perapm, pernom, disnom, año
FROM persona P, distrito D
WHERE P.discod = D.discod
ORDER BY perapp
```

- Se especifican las tablas `persona` y `distrito`, utilizando alias (`P` para `persona` y `D` para `distrito`) para simplificar la referencia.
- `WHERE P.discod = D.discod` establece la relación entre las tablas, indicando que solo se deben recuperar las filas donde el código de distrito (`discod`) en `persona` coincida con el de `distrito`.

Mostrar apellidos y nombres de los varones que viven en Surco

```
SELECT perapp, perapm, pernom, disnom
FROM persona P, distrito D
WHERE P.discod = D.discod AND persex = 'M'
            AND disnom = 'Surco'
ORDER BY perapp
```

Mostrar apellidos, nombre y teléfonos de las personas cuyos teléfonos terminan en 5

```
SELECT perapp, perapm, pernom, pertel
FROM persona P, telefono T
WHERE P.percod = T.percod AND pertel like '%5'
ORDER BY perapp
```

Mostrar apellidos, nombre y profesión de todas las personas

```
SELECT perapp, perapm, pernom, pronom
FROM persona P, personaprofesion PP, profesion R
WHERE P.percod = PP.percod AND PP.procod = R.procod
ORDER BY perapp
```

Mostrar apellidos y nombre de todos los ingenieros que viven en La Molina

```
SELECT perapp, perapm, pernom, pronom, disnom
FROM persona P, personaprofesion PP, profesion R, distrito D
WHERE P.percod = PP.percod AND PP.procod = R.procod AND
            P.discod = D.discod AND disnom = 'La Molina' AND
            pronom like 'Ingeniero%'
ORDER BY perapp
```

Una **secuencia** en SQL se utiliza para generar números enteros secuenciales únicos, generalmente para claves primarias. Garantiza que no haya valores repetidos y puede configurarse para aumentar o disminuir automáticamente.

Sintaxis:

```
CREATE SEQUENCE nombre_secuencia
START WITH valor_inicial
INCREMENT BY valor_incremento
MAXVALUE valor_máximo
MINVALUE valor_mínimo
CYCLE | NOCYCLE;
```

Ejemplo:

```
CREATE SEQUENCE seq_codigo_cliente
START WITH 1
INCREMENT BY 1
MAXVALUE 99999
MINVALUE 1;
```

Para usar la secuencia:

```
INSERT INTO CLIENTE (codigo, nombre)
VALUES (seq_codigo_cliente.NEXTVAL, 'Tienda Azul S.A');
```

Este código genera automáticamente un valor único para la columna `codigo` usando la secuencia.