RESUMEN GENERAL

SEMANA 1 - TEORÍA DE CONJUNTOS, DATOS E INFORMACIÓN

Teoría de Conjuntos y Relaciones:

- **Conjunto**: Colección de objetos bien definidos (elementos).
  - Ejemplo: A={1,2,3,...,n}.

    A={1,2,3,...,n}A = \{1, 2, 3, \dots, n\}
  - **Cardinalidad**: Número de elementos en un conjunto, denotada por $|A|$.

    $|A|$ $|A|$
- **Subconjuntos**: Conjuntos que contienen algunos o todos los elementos de otro conjunto.
- **Producto cartesiano**: Combinaciones ordenadas entre los elementos de dos conjuntos, representado como A×B.

  A×BA \times B
- **Relaciones**: Subconjunto del producto cartesiano entre dos o más conjuntos.
  - **Cardinalidad mínima y máxima**: Define las relaciones mínimas o máximas entre los elementos de dos conjuntos.

Relación inversa y recursiva:

- Una **relación inversa** invierte el orden de los elementos en la relación.
- Una **relación recursiva** es cuando un conjunto se relaciona consigo mismo.

Clasificación y generalización:

- **Clasificación**: Definir subconjuntos a partir de un conjunto genérico (total, parcial, exclusiva, inclusiva).
- **Generalización**: Proceso inverso a la clasificación.

Datos, Información y Bases de Datos

- **Datos**: Descripción objetiva de hechos sin interpretación (ej. temperatura, cantidad de asistentes).
- **Información**: Procesamiento de datos que reduce la incertidumbre y tiene un valor subjetivo.
- **Conocimiento**: Resultado de la interpretación de la información.

Atributos de la Información:

- **Correcta**, **oportuna**, **disponible**, **concisa**, **relevante** y **completa**.

Base de Datos:

- Conjunto de datos persistentes utilizados por aplicaciones organizacionales.
- **SGBD/DBMS**: Sistema que permite crear, manipular y gestionar bases de datos.
- **Objetivo**: Proveer un ambiente conveniente y eficiente para almacenar y extraer información.

Proceso y Procedimiento

- Un **proceso** es un conjunto de actividades organizadas para obtener un producto final.
- Un **procedimiento** es la descripción detallada de cómo se lleva a cabo un proceso.

SEMANA 2 - CONCEPTOS BÁSICOS

Modelos de Datos

Un modelo de datos es una representación simplificada de la realidad que destaca los aspectos más importantes de un sistema mientras deja de lado lo irrelevante. Sirve para describir elementos de un problema y las relaciones entre ellos. Los modelos de datos proporcionan una abstracción que facilita el entendimiento y manejo de los datos.

Existen tres tipos principales de modelos de datos:

1. **Modelos lógicos basados en objetos**: incluyen el modelo entidad-relación, el orientado a objetos y el modelo binario.
2. **Modelos lógicos basados en registros**: incluyen el modelo relacional, el modelo de red y el modelo jerárquico.
3. **Modelos físicos de datos**: describen la forma en que los datos se almacenan físicamente en el sistema, como el modelo unificado o de memoria de trama.

Independencia de Datos

Este concepto se refiere a la capacidad de modificar la estructura de la base de datos en diferentes niveles sin afectar otros niveles. Hay dos tipos:

1. **Independencia de datos física**: la capacidad de modificar el esquema físico sin tener que reescribir los programas.
2. **Independencia de datos lógica**: permite modificar el esquema conceptual sin afectar a los programas que usan esos datos.

Fases del Diseño de una Base de Datos

El diseño de una base de datos sigue cuatro fases principales:

1. **Dominio del problema**: Se identifican las entidades y relaciones dentro del problema a resolver.
2. **Diseño conceptual**: Se crean descripciones detalladas de entidades, relaciones y restricciones sin considerar aspectos técnicos.
3. **Diseño lógico**: Se traduce el diseño conceptual a un modelo de datos como el relacional u objeto-relacional.
4. **Diseño físico**: Se especifican detalles de almacenamiento, índices y rutas de acceso para optimizar el rendimiento.

Modelo Entidad-Relación (ER)

Este modelo es utilizado para representar visualmente las entidades y sus relaciones dentro de una base de datos. Algunos conceptos clave incluyen:

- **Entidad**: Un objeto que existe y es distinguible dentro del sistema, como un cliente o un producto.
- **Atributos**: Características o propiedades de una entidad. Los atributos pueden ser simples, compuestos, monovalor o multivalor.
- **Clave primaria**: Un atributo que identifica de manera única a cada instancia de una entidad.
- **Relaciones**: Asociaciones entre entidades, como un cliente que hace un pedido. Pueden ser uno a uno, uno a muchos o muchos a muchos.

Dependencia Existencial

Se dice que una entidad depende existencialmente de otra cuando para que exista, debe estar relacionada con una entidad de otro conjunto. Existen dos tipos:

1. **Dependencia por existencia**: no requiere la clave de otra entidad para existir.
2. **Dependencia por identidad**: requiere la clave de otra entidad para identificarse, como una entidad débil que depende de una entidad fuerte.

SEMANA 3 - METODOLOGÍAS DE MODELAMIENTO

Modelo Relacional

El modelo relacional organiza los datos en tablas (relaciones), donde cada fila es una "tupla" y cada columna representa un dominio de valores. Los elementos clave del modelo relacional son:

- **Relación**: Representada como una tabla que agrupa datos relacionados.
- **Tupla**: Cada fila de una tabla, que corresponde a un conjunto de valores.
- **Atributos**: Las columnas de una tabla, que corresponden a los dominios de los valores.

- **Clave primaria (PK)**: Un atributo o conjunto de atributos que identifica de manera única cada fila en una tabla.
- **Clave foránea (FK)**: Un atributo que se refiere a la clave primaria de otra tabla para establecer relaciones entre tablas.

El modelo relacional se basa en tres principios fundamentales:

1. **Estructura**: Los datos se representan como colecciones de relaciones (tablas).
2. **Manipulación**: Incluye operaciones para consultar y modificar los datos.
3. **Integridad**: Conjunto de reglas que garantizan la consistencia de los datos.

Lenguajes Formales en el Modelo Relacional

El modelo relacional utiliza dos lenguajes formales para trabajar con datos:

- **Álgebra Relacional**: Es un lenguaje de consulta procedimental. Proporciona un conjunto de operaciones como selección, proyección, unión, intersección, producto cartesiano y diferencia, entre otras.
- **Cálculo Relacional**: Es un lenguaje no procedimental, donde el usuario describe lo que desea obtener sin especificar los pasos exactos.

Operaciones del Álgebra Relacional:

- **Selección (σ)**: Extrae las tuplas que cumplen un cierto predicado.
- **Proyección (π)**: Selecciona columnas específicas de una tabla.
- **Producto cartesiano (×)**: Combina cada tupla de una tabla con cada tupla de otra tabla.
- **Unión (∪)**: Combina los resultados de dos tablas.
- **Diferencia (–)**: Devuelve las tuplas que están en una tabla pero no en la otra.
- **Intersección (∩)**: Devuelve las tuplas comunes entre dos tablas.
- **Join (⋈)**: Combina tuplas de dos tablas basándose en una condición de igualdad.

Restricciones de Integridad en el Modelo Relacional

Estas restricciones aseguran que los datos en la base de datos sean coherentes y válidos:

1. **Integridad de dominio**: Los valores deben pertenecer a un dominio específico.
2. **Integridad de columna**: Impone restricciones adicionales a nivel de columna.
3. **Integridad de entidad**: Ningún componente de una clave primaria puede ser nulo.
4. **Integridad referencial**: Toda clave foránea debe tener un valor correspondiente en la clave primaria de otra tabla.
5. **Integridad definida por el usuario**: Reglas personalizadas que reflejan las necesidades específicas del negocio.

Técnicas de Modelamiento de Datos

Se presentaron dos notaciones principales para el modelamiento de datos:

Notación Barker

Desarrollada inicialmente por la consultora CACI y promovida por Richard Barker, esta notación es ampliamente utilizada en la industria, especialmente por Oracle. Las convenciones principales son:

- **Entidades**: Se representan con cuadros de esquinas redondeadas y nombres en mayúsculas.
- **Atributos**: Los atributos obligatorios se marcan con un asterisco (*), los opcionales con un círculo (o), y los identificadores únicos con una almohadilla (#).
- **Relaciones**: Se representan con líneas que conectan entidades, y pueden ser obligatorias o opcionales.

Notación IDEF1X

Es otra técnica utilizada para modelar datos, con un enfoque más estructurado y formal en comparación con Barker.

SEMANA 4 - SQL

Lenguaje SQL

SQL (Structured Query Language) es el lenguaje estándar utilizado para gestionar bases de datos relacionales. Es un lenguaje declarativo y orientado a conjuntos que permite interactuar con bases de datos de manera libre, sin distinguir entre mayúsculas y minúsculas.

SQL se usa para:

- **Crear** y gestionar bases de datos y sus estructuras (DDL - Data Definition Language).
- **Consultar** datos mediante sentencias SELECT.
- **Manipular** datos mediante INSERT, UPDATE y DELETE.
- **Controlar** el acceso a los datos.

SQL se estandarizó en 1986 (SQL-86) y sigue evolucionando, con la versión más reciente siendo SQL-2016.

Comandos de DDL (Data Definition Language)

Los comandos DDL en SQL se utilizan para definir la estructura de las bases de datos, incluyendo la creación y modificación de tablas, vistas, índices y esquemas. Los comandos clave son:

CREATE TABLE

El comando **CREATE TABLE** se utiliza para crear una nueva tabla, especificando:

- **Nombre de la tabla** y su esquema.
- **Columnas** con sus nombres, tipos de datos y restricciones.
- **Restricciones** como claves primarias, claves foráneas, valores por defecto y verificaciones (CHECK).

Ejemplo:

```
CREATE TABLE Persona (
  IdPersona CHAR(8) PRIMARY KEY,
  IdDepartamento CHAR(2) REFERENCES Departamento(IdDepartam
ento),
  Npersona VARCHAR2(40) NOT NULL,
  Jefe CHAR(8) REFERENCES Persona(IdPersona),
```

```
  Sueldo NUMBER(6,2) DEFAULT 930
);
```

ALTER TABLE

El comando **ALTER TABLE** permite modificar una tabla existente:

- **Agregar** o eliminar columnas.
- **Modificar** las propiedades de las columnas, como cambiar su tamaño o valor por defecto.
- **Definir restricciones** sobre las columnas (por ejemplo, CHECK o UNIQUE).

Ejemplo:

```
ALTER TABLE Persona ADD (FechaIngreso DATE, Sexo CHAR(1) NO
T NULL);
ALTER TABLE Persona MODIFY Sueldo NUMBER(7,2) DEFAULT 1025;
```

DROP TABLE

El comando **DROP TABLE** elimina permanentemente una tabla y sus datos asociados. Es una operación irreversible.

Ejemplo:

```
DROP TABLE Persona CASCADE CONSTRAINTS;
```

Restricciones en Tablas

SQL permite definir varias restricciones para garantizar la integridad de los datos:

1. **PRIMARY KEY**: Identifica de forma única cada fila en una tabla.
2. **FOREIGN KEY**: Establece relaciones entre tablas, referenciando claves primarias de otras tablas.
3. **UNIQUE**: Asegura que los valores en una columna sean únicos.
4. **NOT NULL**: Impide valores nulos en una columna.
5. **CHECK**: Establece reglas personalizadas para los valores de una columna.

Ejemplo de restricciones:

```
CREATE TABLE scott.emp (
  empno NUMBER CONSTRAINT pk_emp PRIMARY KEY,
  ename VARCHAR2(10) CONSTRAINT nn_name NOT NULL CONSTRAINT
upper_name CHECK(ename = UPPER(ename)),
  sal NUMBER(10,2) CONSTRAINT ck_sal CHECK(sal > 500)
);
```

Índices en SQL

Los **índices** mejoran el rendimiento de las consultas, acelerando la recuperación de datos en una tabla. Existen dos tipos:

- **Índices únicos**: Se crean automáticamente para columnas que son PRIMARY KEY o UNIQUE.
- **Índices no únicos**: Son creados manualmente para mejorar el acceso a las filas.

Ejemplo:

```
CREATE INDEX index_name ON table_name (column1, column2);
```

Índices compuestos

Son aquellos creados en varias columnas de una tabla. Se usan para mejorar las consultas que involucran más de una columna en condiciones WHERE o JOIN.

Ejemplo:

```
CREATE INDEX emps_name_idx ON employees (first_name, last_n
ame);
```

Vistas en SQL

Una **vista** es una representación lógica de una tabla o consulta. No almacena datos propios, sino que muestra los datos de las tablas base. Las vistas se utilizan para simplificar consultas y mejorar la seguridad al limitar el acceso a datos sensibles.

Ejemplo:

```
CREATE VIEW view_employees AS
SELECT employee_id, first_name, last_name
FROM employees
WHERE employee_id BETWEEN 100 AND 124;
```

SEMANA 5 - SQL-DML

Manipulación de Datos con SQL (DML)

Las sentencias DML (Data Manipulation Language) en SQL se utilizan para manipular los datos almacenados en las bases de datos. Los comandos principales de DML incluyen SELECT, INSERT, DELETE y UPDATE.

1. Sentencia SQL SELECT

La sentencia **SELECT** es utilizada para recuperar datos de una o más tablas o vistas, devolviendo una tabla de resultados que puede ser utilizada en otras operaciones.

Tipos de consultas:

- **Consultas simples**: Recuperan datos básicos.
- **Consultas con funciones agregadas**: Como `SUM`, `COUNT`, `AVG`.
- **Consultas con `GROUP BY` y `HAVING`**: Agrupan resultados y aplican condiciones a grupos.
- **Consultas con `ORDER BY`**: Ordenan resultados en orden ascendente o descendente.
- **Consultas de unión (`UNION`)**: Combinan resultados de varias consultas.
- **Consultas anidadas**: Consultas dentro de otras.
- **Consultas correlacionadas**: Subconsultas que dependen de la consulta externa.

Sintaxis básica de SELECT:

```
SELECT columnas
FROM tablas
WHERE condición;
```

- **DISTINCT** elimina filas duplicadas.
- Los operadores de comparación como `=`, `!=`, `<>`, `>`, `<`, etc., se utilizan en la cláusula `WHERE` para filtrar filas.

Ejemplo:

```
SELECT DISTINCT job FROM emp;
```

Operadores comunes en SELECT:

- **IN**: Verifica si un valor pertenece a una lista de valores.
- **BETWEEN**: Verifica si un valor está en un rango.
- **EXISTS**: Verifica si una subconsulta devuelve algún resultado.
- **LIKE**: Realiza búsquedas de patrones con comodines.
- **IS NULL**: Verifica si un valor es nulo.

Ordenación de resultados con ORDER BY:

Puedes ordenar los resultados de una consulta en orden ascendente (`ASC`) o descendente (`DESC`).

Ejemplo:

```
SELECT ciudad, ventas
FROM oficinas
ORDER BY ventas DESC;
```

---

2. Sentencia SQL INSERT

La sentencia **INSERT** se utiliza para agregar nuevas filas a una tabla.

Sintaxis básica:

```
INSERT INTO tabla (columna1, columna2, ...)
VALUES (valor1, valor2, ...);
```

Si se omiten los nombres de las columnas, se deben proporcionar valores para todas las columnas en el orden en que están definidas.

Ejemplo:

```
INSERT INTO emp (empno, ename, hiredate, sal, deptno)
VALUES (1235, 'Jorge', '01-JAN-2024', 2500, 30);
```

---

3. Sentencia SQL DELETE

La sentencia **DELETE** elimina filas de una tabla que cumplan con una condición especificada en `WHERE`. Si no se especifica una condición, se eliminan todas las filas.

Sintaxis básica:

```
DELETE FROM tabla
WHERE condición;
```

Ejemplo:

```
DELETE FROM Persona WHERE idpersona = '0002';
```

---

4. Sentencia SQL UPDATE

La sentencia **UPDATE** se utiliza para modificar los valores en las columnas de una tabla. A menos que se especifique una condición en `WHERE`, todas las filas serán actualizadas.

Sintaxis básica:

```
UPDATE tabla
SET columna1 = valor1, columna2 = valor2, ...
WHERE condición;
```

Ejemplo:

```
UPDATE emp
SET job = 'salesman', sal = sal * 1.5, deptno = 30
WHERE deptno = (SELECT deptno FROM emp WHERE empno = 7788);
```

---

5. Prioridad de operadores

En SQL, el operador **AND** tiene mayor prioridad que el operador **OR**. Es importante tener esto en cuenta para evitar resultados inesperados en consultas que combinan estos operadores.

Ejemplo:

```
SELECT last_name, salary * 1.05 AS aumento
FROM employees
WHERE department_id IN(50, 80)
AND first_name LIKE 'C%' OR last_name LIKE '%s%';
```

SEMANA 6 - SQL-DML:

Operadores

Los operadores en SQL se utilizan para realizar diversas manipulaciones sobre los datos. Algunos ejemplos importantes incluyen:

- **ROWNUM**: Devuelve el número de la fila de una consulta. Se utiliza comúnmente en consultas que requieren limitar el número de filas devueltas, como obtener los primeros n registros ordenados.

  Ejemplo:

  ```
  SELECT rownum AS posicion, saldo, nomproveedor, razonsoc
  ial
  FROM (SELECT saldo, nomproveedor, razonsocial
        FROM Proveedor
        ORDER BY saldo DESC)
  WHERE ROWNUM <= 5;
  ```

- **Expresiones aritméticas con fechas**: SQL permite realizar operaciones aritméticas con fechas. Por ejemplo, se puede sumar o restar días a una fecha, o calcular la diferencia entre dos fechas para obtener el número de días.

  Ejemplo:

```
SELECT TO_CHAR(fecha_ingreso, 'MM/YYYY') AS Mes_Ingreso
FROM Empleados
WHERE primer_apellido = 'Salazar';
```

Funciones

SQL proporciona un conjunto de funciones que pueden aplicarse a los datos. Se dividen en tres tipos principales:

1. **Funciones aritméticas**:
   - **ABS**: Devuelve el valor absoluto.
   - **FLOOR**: Devuelve el mayor entero menor o igual que el valor numérico.
   - **MOD**: Devuelve el resto de una división.
   - **ROUND** y **TRUNC**: Redondean o truncan un número a una precisión específica.
2. **Funciones de fecha y hora**:
   - **SYSDATE**: Devuelve la fecha y hora actuales.
   - **ADD_MONTHS**: Suma o resta meses a una fecha.
   - **NEXT_DAY**: Devuelve el siguiente día de la semana después de una fecha específica.
3. **Funciones de cadena**:
   - **UPPER** y **LOWER**: Convierten cadenas a mayúsculas o minúsculas.
   - **SUBSTR**: Extrae una subcadena de una cadena.
   - **LENGTH**: Devuelve el número de caracteres de una cadena.
4. **Funciones de agregación**:
   - **COUNT**: Devuelve el número de filas.

- **SUM**: Suma los valores de una columna.
- **AVG**: Calcula el promedio de los valores.
- **MAX** y **MIN**: Devuelven el valor máximo o mínimo.

Ejemplo:

```
SELECT COUNT(*), AVG(salario), MAX(salario)
FROM Empleados;
```

Consultas Agrupadas

Las consultas agrupadas en SQL permiten sumarizar los resultados usando la cláusula **GROUP BY** junto con funciones de agregación. Esto es útil para obtener subtotales o resúmenes basados en los valores de una o más columnas.

- **GROUP BY** agrupa los resultados en función de una columna o expresión.
- **HAVING** filtra los grupos creados por `GROUP BY`, similar a cómo `WHERE` filtra filas.

  Ejemplo:

  ```
  SELECT IdDepartamento, MIN(Sueldo), AVG(Sueldo), MAX(Sue
  ldo)
  FROM Persona
  GROUP BY IdDepartamento;
  ```

Condiciones de Búsqueda de Grupo

La cláusula **HAVING** se utiliza para filtrar los resultados después de agruparlos con **GROUP BY**. Esto permite aplicar condiciones de filtrado a los grupos y no a las filas individuales.

Ejemplo:

```
SELECT REP, AVG(IMPORTE)
FROM PEDIDOS
GROUP BY REP
HAVING SUM(IMPORTE) > 30000;
```

Outer Join (Conjunción Lateral)

El **Outer Join** extiende el resultado de un join normal, devolviendo no solo las filas que cumplen con la condición del join, sino también todas las filas de una tabla, incluso aquellas sin coincidencias en la otra tabla. Las columnas correspondientes a la tabla sin coincidencia tendrán valores nulos.

Ejemplo:

```
SELECT D.IdDepartamento, D.NDepartamento, AVG(P.Sueldo), SU
M(P.Sueldo)
FROM Departamento D, Persona P
WHERE D.IdDepartamento (+) = P.IdDepartamento
GROUP BY D.IdDepartamento, D.NDepartamento;
```

En este ejemplo, se devuelven todos los departamentos, incluso los que no tienen empleados, con valores nulos en las columnas de sueldos.

SEMANA 7 - DML

Lenguaje de Manipulación de Datos (DML)

El DML (Data Manipulation Language) es utilizado para la manipulación de los datos en las bases de datos relacionales. Las principales operaciones incluyen:

- **SELECT**: Recupera datos de las tablas.
- **INSERT**: Inserta nuevas filas en las tablas.
- **UPDATE**: Actualiza los valores existentes.
- **DELETE**: Elimina datos.

Índices

Un **índice** es un objeto que mejora la velocidad de las consultas al proporcionar acceso rápido a las filas. Existen varios tipos de índices, incluyendo:

- **Índice B-Tree (árbol balanceado)**: El tipo más común, adecuado para consultas que buscan un valor único.

- **Índice Bitmap**: Eficiente para columnas con un número limitado de valores distintos.
- **Índices particionados**: Dividen el índice en particiones según un criterio específico (por rango, lista, hash).
- **Índice basado en función**: Aplica una función o expresión a las columnas indexadas.

Ejemplo:

```
CREATE INDEX idx_apellido ON EMPLOYEES (LAST_NAME);
```

Los índices se crean después de cargar los datos y es recomendable no indexar columnas con muchos valores nulos. Para eliminar un índice, se utiliza `DROP INDEX`.

Vistas

Una **vista** es una tabla lógica que no contiene datos por sí misma, sino que se basa en tablas o vistas existentes. Se utilizan para:

- Proporcionar un nivel adicional de seguridad.
- Ocultar la complejidad de las tablas subyacentes.
- Presentar los datos desde diferentes perspectivas.

Existen varios tipos de vistas:

- **Vistas simples**: Basadas en una sola tabla y generalmente modificables.
- **Vistas complejas**: Involucran múltiples tablas y agregaciones, generalmente no modificables.
- **Vistas materializadas**: Almacenan los datos físicamente para mejorar el rendimiento en consultas costosas.

Ejemplo:

```
CREATE VIEW VLOCALIDADES AS
SELECT LOCATION_ID, CITY, COUNTRY_ID FROM LOCATIONS;
```

Secuencias

Las **secuencias** generan valores enteros secuenciales únicos, generalmente utilizados para las claves primarias. Una secuencia garantiza que los valores no se repitan, y puede ser configurada para reiniciarse al alcanzar un valor máximo (CYCLE) o no (NOCYCLE).

Sintaxis:

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

Para utilizar una secuencia:

- `NEXTVAL`: Retorna el siguiente valor de la secuencia.
- `CURRVAL`: Retorna el valor actual de la secuencia.

Subconsultas

Una **subconsulta** es un comando `SELECT` anidado dentro de otro comando SQL. Existen dos tipos principales:

- **Subconsulta escalar**: Retorna un solo valor.
- **Subconsulta correlacionada**: Se evalúa una vez por cada fila procesada en la consulta externa.

Ejemplo de subconsulta correlacionada:

```
SELECT P.NPersona, P.Sueldo
FROM Persona P
WHERE Sueldo > (SELECT AVG(Sueldo) FROM Persona R WHERE P.I
dDepartamento = R.IdDepartamento);
```

Las subconsultas también pueden aparecer en la cláusula `FROM` de una consulta, actuando como una tabla virtual.

Existencia (EXISTS)

El operador `EXISTS` verifica si una subconsulta devuelve al menos una fila. Si es así, retorna TRUE; de lo contrario, retorna FALSE.

Ejemplo:

```
SELECT nDepartamento
FROM Departamento D
WHERE EXISTS (SELECT * FROM Persona P WHERE D.IdDepartament
o = P.IdDepartamento);
```

Seguridad y Control de Acceso (DCL)

El **DCL (Data Control Language)** se utiliza para controlar el acceso a los objetos de la base de datos mediante los comandos **GRANT** y **REVOKE**:

- **GRANT**: Otorga privilegios a un usuario o rol.
- **REVOKE**: Revoca los privilegios previamente otorgados.

Ejemplo:

```
GRANT SELECT, INSERT ON Persona TO rl_prueba;
```

También se pueden crear **roles**, que son conjuntos de privilegios, y asignarlos a usuarios específicos.