SQL - *Structured Query Language*

Google

UADE

2024

SQL - *Structured Query Language*

Google

Puesto: Data Analyst / Business Intelligence Analyst

Requisitos de SQL: Google busca personas con experiencia en **consultas avanzadas de SQL**, optimización de consultas, y capacidad para manejar grandes volúmenes de datos usando herramientas como **BigQuery**.

...que buscan las grandes empresas

Preguntas clave de entrevista:

- ¿Cómo realizarías una consulta recursiva en SQL?
- ¿Cómo harías para identificar duplicados en una tabla y eliminarlos?
- Tu consulta SQL funciona pero tarda 10 minutos. ¿Qué tres caminos diferentes probarías para mejorarla?

UADE

2024

SQL - *Structured Query Language*

amazon

Puesto: Data Engineer / Data Scientist

Requisitos de SQL: Amazon busca ingenieros de datos que dominen consultas complejas, así como la habilidad para trabajar con big data utilizando SQL y otras herramientas como Redshift o Athena.

...que buscan las grandes empresas

Preguntas clave de entrevista:

- ¿Cómo usarías SQL para analizar grandes volúmenes de datos almacenados en Amazon Redshift?
- ¿Cómo usarías la cláusula GROUP BY y HAVING para filtrar y agrupar datos?
- ¿Cómo realizarías una operación de JOIN entre varias tablas con millones de filas?

UADE

2024

SQL - *Structured Query Language*

Meta

Puesto: Data Scientist / Data Engineer

Requisitos de SQL: Se valora mucho la capacidad de escribir consultas eficientes, manejar bases de datos distribuidas y analizar grandes volúmenes de datos con herramientas basadas en SQL.

...que buscan las grandes empresas

Preguntas clave de entrevista:

- ¿Cuál es la diferencia entre INNER JOIN y OUTER JOIN? Da un ejemplo.
- ¿Cómo crearías un índice en SQL para mejorar el rendimiento de una consulta?
- Explica cómo manejarías una situación en la que los datos de una tabla están desnormalizados.

UADE

2024

SQL - *Structured Query Language*

Objetivo: Aprender a crear y utilizar vistas para reutilización de consultas y aplicaciones de seguridad, y explorar las funciones del sistema para realizar operaciones específicas dentro de un SGBD.

SQL - *Structured Query Language*

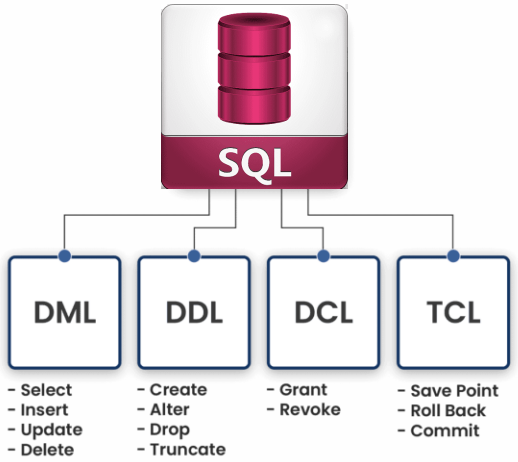
SQL (Structured Query Language) *es un lenguaje de programación diseñado específicamente para gestionar y manipular bases de datos relacionales.*
Es utilizado para realizar diversas operaciones, como la consulta, inserción, actualización, eliminación y gestión de datos en sistemas de gestión de bases de datos (SGBD).

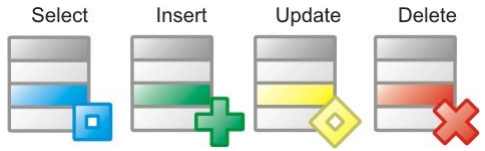
Principales características de SQL:

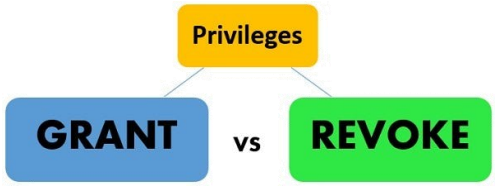
1. **Lenguaje declarativo**: SQL se centra en describir qué se quiere hacer con los datos, sin necesidad de detallar cómo debe realizarse la operación. *No posee estructuras de control*.
2. **Estándar**: Aunque hay variaciones específicas en diferentes sistemas de bases de datos (como SQL Server, MySQL, Oracle), SQL es un estándar que sigue una especificación definida por el ISO/ANSI.
3. **Relacional**: SQL está basado en el modelo relacional de bases de datos, donde los datos se organizan en tablas que pueden estar relacionadas entre sí a través de claves primarias y foráneas.

UADE

2024







SUBLENGUAJES

DDL (Data Definition Language): Lenguaje de definición de datos:

Define y modifica la estructura de las bases de datos, como crear o eliminar tablas, índices, vistas, entre otros objetos.

- **CREATE – DROP – ALTER**

DML (Data Manipulation Language): Lenguaje de manipulación datos:

Manipula los datos dentro de las tablas. Permite realizar operaciones como consultar, insertar, actualizar o eliminar registros.

- **SELECT – INSERT – UPDATE – DELETE**

DCL (Data Control Language): Lenguaje de control de datos:

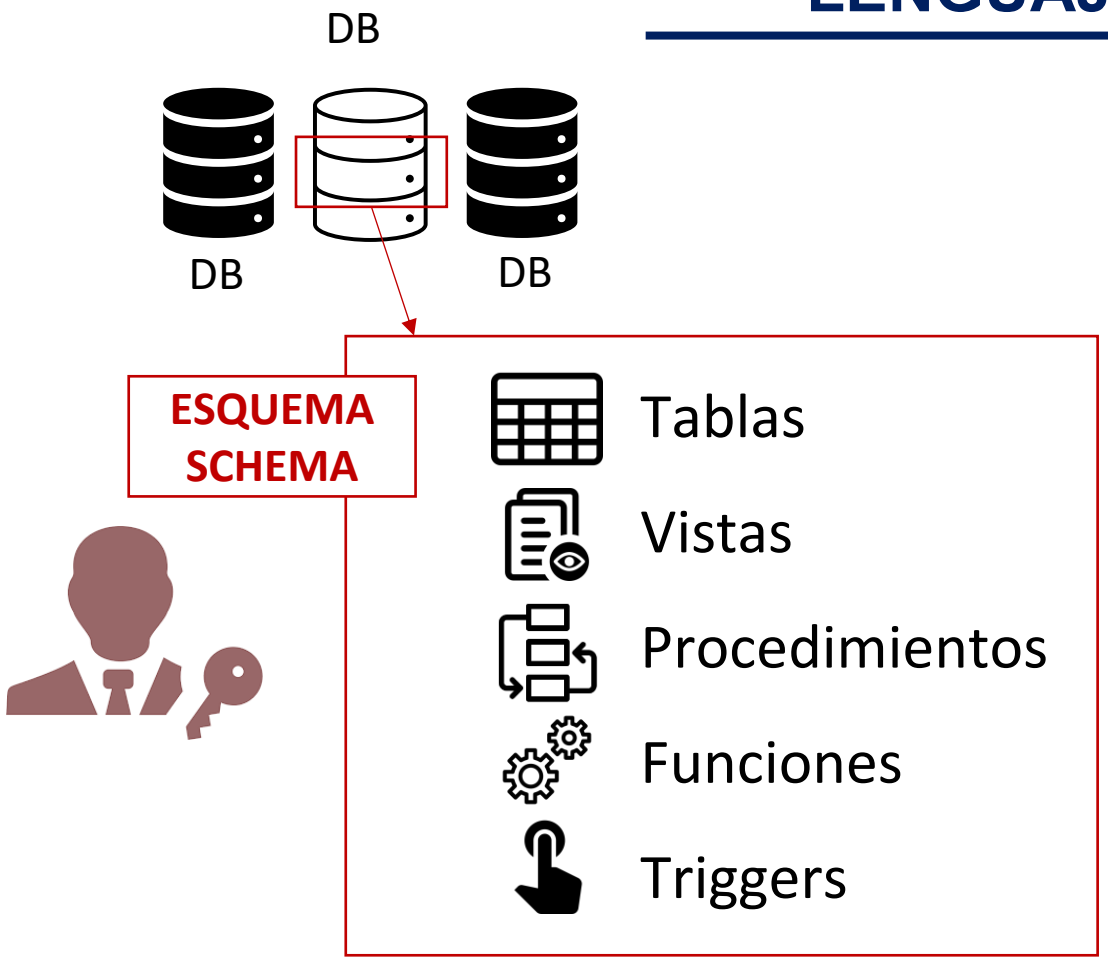
Controla el acceso a los datos y gestiona los permisos de los usuarios. Se utiliza para definir quién puede acceder o modificar los datos de la base de datos.

- **GRANT - REVOKE**

UADE

Garantizar
seguridad, integridad y
disponibilidad de los datos

LENGUAJE DE DEFINICION DE DATOS




```sql
--- List of tables
SELECT * FROM sys.tables;

--- List of schemas
SELECT * FROM sys.schemas;

--- Crear una base de datos (catálogo)
CREATE DATABASE EmpresaDB;

--- Crear un esquema en esa base de datos
USE EmpresaDB;
CREATE SCHEMA Ventas;

--- Crear una tabla dentro del esquema
CREATE TABLE Ventas.Clientes (
    Id INT PRIMARY KEY,
    Nombre NVARCHAR(50),
    Email NVARCHAR(100)
);

--- Consultar la tabla utilizando el nombre cualificado
SELECT * FROM Ventas.Clientes;
```

Permite crear nuevas tablas, índices, vistas, procedimientos almacenados, y otros objetos en la base de datos

CREATE

UADE

DDL

LENGUAJE DE DEFINICION DE DATOS

```
CREATE tipo_de_objeto identificador
(
descripción_del_objeto
)
```

```
--- Crear una base de datos

CREATE DATABASE EmpresaDB;
```

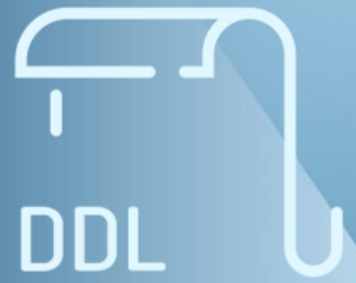
TIPO DE OBJETOS

DATABASE	FUNCTION	SYNONYM
SCHEMA	TRIGGER	TYPE
TABLE	USER	ASSEMBLY
VIEW	ROLE	RULE
INDEX	LOGIN	SERVICE
PROCEDURE	SEQUENCE	AGGREGATE

***Nota:** Estos no son los únicos objetos que pueden ser creados mediante esta sentencia solo algunos de los más comunes.

CREATE

Permite crear nuevas tablas, índices, vistas, procedimientos almacenados, y otros objetos en la base de datos



LENGUAJE DE DEFINICION DE DATOS

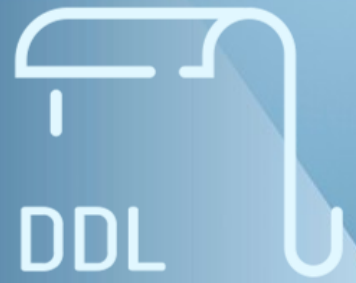
```sql
-- CREATE TABLE:
CREATE TABLE Usuarios (
    ID INT PRIMARY KEY,
    Nombre VARCHAR(100)
);

-- CREATE VIEW:
CREATE VIEW Vista_Empleados AS
SELECT Nombre, Departamento FROM Empleados;

-- CREATE INDEX:
CREATE INDEX idx_Nombre ON RecursosHumanos.Empleados(Nombre);

--CREATE PROCEDURE:
CREATE PROCEDURE ObtenerUsuarios
AS
SELECT * FROM Usuarios;

-- CREATE SCHEMA:
CREATE SCHEMA Finanzas;

-- CREATE DATABASE:
CREATE DATABASE MiBaseDeDatos;
```

```sql
-- CREATE TRIGGER:
CREATE TRIGGER trg_AfterInsert
ON Empleados
AFTER INSERT
AS
BEGIN
    PRINT 'Empleado insertado';
END;

-- CREATE FUNCTION:
CREATE FUNCTION ObtenerEdad(@FechaNacimiento DATE)
RETURNS INT
AS
BEGIN
    RETURN DATEDIFF(YEAR, @FechaNacimiento, GETDATE());
END;

-- CREATE USER:
CREATE USER 'nuevo_usuario' WITH PASSWORD 'contraseña';

-- CREATE ROLE:
CREATE ROLE Gerente;
```

UADE

CREATE

Permite crear nuevas tablas, índices, vistas, procedimientos almacenados, y otros objetos en la base de datos

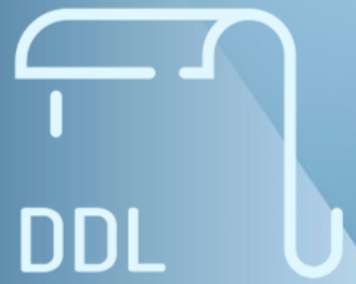


LENGUAJE DE DEFINICION DE DATOS

```
CREATE table Empleados
(       IdEmpleado INT PRIMARY KEY,                         -- Entero como clave primaria
        Nombre NVARCHAR(50) NOT NULL,                       -- Cadena de texto de longitud variable
        FechaNacimiento DATE,                               -- Fecha (año, mes, día)
        Salario DECIMAL(10, 2),                             -- Número decimal con 10 dígitos en total, 2 decimales
        Genero CHAR(1) NOT NULL CHECK (Genero IN ('M', 'F', 'O')),
            Genero CHAR(1) NOT NULL CONSTRAINT chk_genero CHECK (Genero IN ('M', 'F', 'O')),
        o   Genero CHAR(1) NOT NULL,
        o           FOREIGN KEY (Genero) REFERENCES Generos(IdGenero),
                                                            -- Un carácter ('M' masculino, 'F' femenino, 'O' otro)
        Telefono VARCHAR(15) NULL,                          -- Número de teléfono como texto (longitud variable)
        CorreoElectronico VARCHAR(100) UNIQUE,              -- Correo electrónico con restricción de unicidad
        FContratacion DATETIME DEFAULT GETDATE(),           -- Fecha y hora con valor predeterminado
        Estado BIT DEFAULT 1,                               -- Booleano (0 para falso, 1 para verdadero)
        FotoPerfil VARBINARY(MAX),                          -- Almacena datos binarios como una imagen
        Direccion NVARCHAR(200),                            -- Cadena de texto con hasta 200 caracteres
        Comentarios TEXT,                                   -- Texto largo (almacenar comentarios o notas)
        UltimaModificacion TIMESTAMP                        -- Marca temporal (cambia automáticamente al modificar)
                                                            TIMESTAMP / ROWVERSION
);
```

UADE

CREATE



LENGUAJE DE DEFINICION DE DATOS

ALTER tipo_de_objeto *identificador*
tipo de operación;

```
--- Modificar tabla

ALTER TABLE identificador
{ ADD | DROP } objeto [Descripción];
{ ALTER COLUMN } objeto [Descripción];

EXEC sp_rename identificador, objeto;
```

TIPO DE OBJETOS

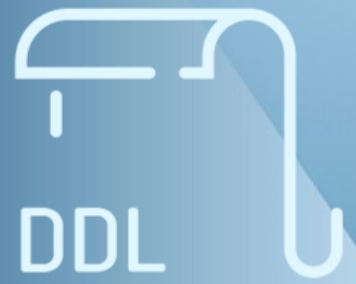
DATABASE	FUNCTION	TYPE
SCHEMA	TRIGGER	ASSEMBLY
TABLE	USER	RULE
VIEW	ROLE	SERVICE
INDEX	LOGIN	AGGREGATE
PROCEDURE	SEQUENCE	

Se utiliza para modificar la estructura de objetos existentes en una base de datos, como tablas, esquemas, vistas, etc. Permite hacer ajustes sin tener que eliminar y volver a crear el objeto.

LENGUAJE DE DEFINICION DE DATOS

```sql
-- Agregar una columna a una tabla existente
ALTER TABLE Empleados
ADD FechaNacimiento DATE;

-- Modificar una columna existente
ALTER TABLE Empleados
ALTER COLUMN Salario DECIMAL(10, 2);

-- Eliminar una columna
ALTER TABLE Empleados
DROP COLUMN Telefono;
```

```sql
-- Renombrar una columna
EXEC sp_rename 'Empleados.Nombre', 'NombreCompleto', 'COLUMN';

-- Renombrar una tabla
EXEC sp_rename 'Empleados', 'Personal', 'OBJECT';

-- Renombrar un índice
EXEC sp_rename 'Empleados.idx_nombre', 'idx_Completo', 'INDEX';
```

LENGUAJE DE DEFINICION DE DATOS

Agregar columna: ALTER TABLE ... ADD

Modificar columna: ALTER TABLE ... ALTER COLUMN

Eliminar columna: ALTER TABLE ... DROP COLUMN

Renombrar columna: EXEC sp_rename ...

Agregar clave foránea: ALTER TABLE ... ADD FOREIGN KEY

Eliminar clave foránea: ALTER TABLE ... DROP CONSTRAINT

Agregar restricción CHECK: ALTER TABLE ... ADD CONSTRAINT

Eliminar restricción CHECK: ALTER TABLE ... DROP CONSTRAINT

Renombrar tabla: EXEC sp_rename ...

Cambiar tipo de dato: ALTER TABLE ... ALTER COLUMN

Agregar valor predeterminado: ALTER TABLE ... ADD CONSTRAINT

Eliminar columna con DEFAULT: ALTER TABLE ... DROP COLUMN

Agregar clave primaria: ALTER TABLE ... ADD PRIMARY KEY

Eliminar clave primaria: ALTER TABLE ... DROP CONSTRAINT

LENGUAJE DE DEFINICION DE DATOS

DROP tipo_de_objeto *identificador;*

```
--- Eliminar tabla

DROP TABLE Empleados;

--- Eliminar vista

DROP VIEW VistaEmpleados;
```

TIPO DE OBJETOS

DATABASE	FUNCTION	TYPE
SCHEMA	TRIGGER	ASSEMBLY
TABLE	USER	RULE
VIEW	ROLE	SERVICE
INDEX	LOGIN	AGGREGATE
PROCEDURE	SEQUENCE	CONSTRAINT

Se utiliza para eliminar objetos de la base de datos, como tablas, vistas, índices, procedimientos, y más. Una vez que se elimina un objeto, los datos y la estructura se pierden permanentemente (a menos que tengas una copia de seguridad).

UADE

DROP

DDL

¿Qué nombre le pondrían a su base de datos si crearan una startup?

Creen una base datos simple

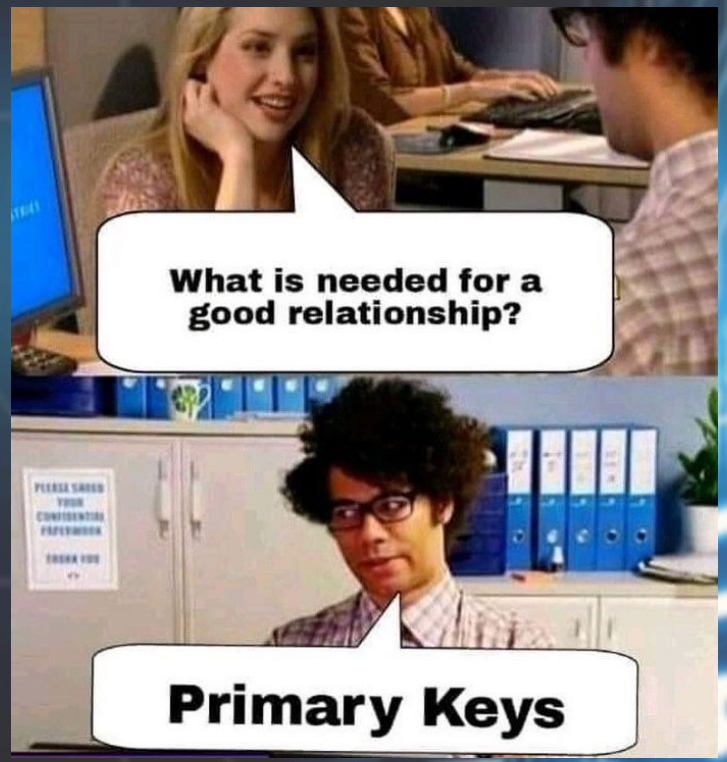
Creen un esquema de Ventas

Creen una tabla de Clientes

Creen una tabla de Productos

UADE

Cuando logras recuperar la base de datos que tú mismo borraste



SE NECESITA PROGRAMADOR

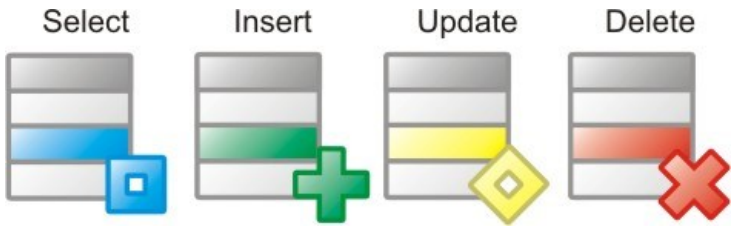
REQUISITOS:
- PHP, C#, JAVA, PASCAL, GO, PYTHON, JS, ARDUINO, AJAX, SQL, BINARIO, ASSEMBLY
- 30 AÑOS DE EXPERIENCIA
- 2 PREMIOS NOBEL DE LA PAZ
- SABER PILOTEAR UN COETE

Cuando la oferta de empleo pide NoSQL y vos no sabes SQL

Usas UPDATE en SQL para arreglar un registro

1.255.399 registros afectados

LENGUAJE DE MANIPULACION DE DATOS



SELECT: Se utiliza para consultar o recuperar datos de una o más tablas. Es el comando más común y flexible para obtener información de las tablas.

INSERT [**INTO**]: Sirve para agregar nuevos registros (filas) a una tabla.

UPDATE: Se utiliza para modificar datos existentes en una o más filas de una tabla.

DELETE: Se usa para eliminar uno o más registros de una tabla, basándose en una condición.

```sql
--- Consultar todos los empleados con un salario mayor a 3000:
SELECT Nombre, Salario
FROM RecursosHumanos.Empleados
WHERE Salario > 3000;

--- Insertar un nuevo empleado en la tabla Empleados:
INSERT INTO RecursosHumanos.Empleados (IdEmpleado, Nombre, Salario,
            IdDepartamento, IdCategoria)
VALUES (1, 'Carlos López', 4000.00, 2, 1);

--- Actualizar el salario de un empleado:
UPDATE RecursosHumanos.Empleados
SET Salario = 5000.00
WHERE IdEmpleado = 1;

--- Eliminar al empleado con el ID 1:
DELETE FROM RecursosHumanos.Empleados
WHERE IdEmpleado = 1;
```

Los comandos DML (Data Definition Language) operan directamente sobre los datos.

UADE

LENGUAJE DE MANIPULACION DE DATOS

```
INSERT [INTO]
        { identificador_tabla | identificador_vista
        [ ( nombre_columna, ..., n ) ] }
VALUES ( { valor | expresion| null} [ , ..., n ] )
```

***Nota**: Debe tener un valor para cada columna especificada en la lista o en la tabla respetando el orden*

```
CREATE TABLE RecursosHumanos.Empleados (
    IdEmpleado INT PRIMARY KEY,
    Nombre NVARCHAR(50) NOT NULL,
    Salario DECIMAL(10, 2) DEFAULT 3000.00,
    IdDepartamento INT,
    IdCategoria INT,
    FechaContratacion DATE DEFAULT GETDATE()
);
```

Toma valores por defecto

```
INSERT INTO RecursosHumanos.Empleados (IdEmpleado, Nombre)
VALUES (1, 'Carlos López');
```

```
INSERT INTO RecursosHumanos.Empleados
VALUES (2, 'Ana Gómez', NULL, 1, 3, NULL);
```

NULL explícito

```
INSERT INTO RecursosHumanos.Empleados
VALUES (2, 'Ana Gómez', NULL, 1, 'C', NULL);
```

ERROR

El comando INSERT INTO se utiliza para insertar nuevos registros en una tabla. Puedes especificar los valores para cada columna o solo algunas columnas de la tabla, en cuyo caso el resto tomará valores predeterminados o NULL, si están permitidos.

LENGUAJE DE MANIPULACION DE DATOS

DELETE [FROM]

{ *identificador_tabla* | identificador_vista

[WHERE < condición >]

***Nota**: Si la condición no se evalúa como verdadera, no se realizará ninguna eliminación y devolverá el mensaje:* `(0 row(s) affected)`

```
-- Eliminar un empleado específico: Elimina al empleado con
IdEmpleado = 3.
DELETE FROM RecursosHumanos.Empleados
WHERE IdEmpleado = 3;

-- Eliminar empleados de un departamento: Elimina a todos
los empleados que pertenecen al Departamento 2.
DELETE FROM RecursosHumanos.Empleados
WHERE IdDepartamento = 2;

-- Eliminar empleados con un salario bajo: Elimina a todos
los empleados cuyo salario sea inferior a 3500.
DELETE FROM RecursosHumanos.Empleados
WHERE Salario < 3500;

-- Eliminar todos los registros: Si deseas vaciar la tabla
Empleados por completo, sin eliminar la estructura de la
tabla:
DELETE FROM RecursosHumanos.Empleados;
```

El comando DELETE se utiliza para eliminar filas de una tabla. Se pueden eliminar una o varias filas a la vez, dependiendo de la condición que se especifique.

UADE

DELETE

LENGUAJE DE MANIPULACION DE DATOS

UPDATE

{ *identificador_tabla* | identificador_vista }

SET columna = { expresión | Default | Null }

[WHERE < condición >]

***Nota**: Cuidado con usar UPDATE sin WHERE ya que puede afectar todas las filas de la tabla. Si la condición no se evalúa como verdadera, devolverá el mensaje de:*
`(0 row(s) affected)`

```sql
--Incrementar el salario de un empleado:
UPDATE RecursosHumanos.Empleados
SET Salario = Salario + 500.00
WHERE IdEmpleado = 3;

-- Actualizar el departamento de todos los empleados cuyo salario es mayor a 4000:
UPDATE RecursosHumanos.Empleados
SET IdDepartamento = 1
WHERE Salario > 4000;

-- Actualizar el salario de los empleados con el salario promedio del departamento:
UPDATE RecursosHumanos.Empleados
SET Salario = (SELECT AVG(Salario) FROM RecursosHumanos.Empleados WHERE IdDepartamento = 1)
WHERE IdDepartamento = 1;

-- Actualizar todas las filas:
UPDATE RecursosHumanos.Empleados
SET Salario = 3000.00;
```

El comando UPDATE en SQL se utiliza para modificar los datos existentes en una o más filas de una tabla. Solo cambia los valores de columnas en filas válidas.

LENGUAJE DE MANIPULACION DE DATOS

5 SELECT { ALL | DISTINCT } *lista de columnas*

[INTO tabla destino]

1 FROM *{ identificador_tabla | identificador_vista }*

2 [WHERE < condición >]

3 [GROUP BY < criterio de agrupamiento >]

4 [HAVING < condición del agrupamiento >]

6 [ORDER BY < ordenamiento { ASC | DES } >]

Cláusula	Finalidad
SELECT	Especifica las columnas de la tabla de resultados. SELECT * es la cláusula predefinida. La sentencia predefinida conserva todas las filas de la tabla de resultados y no elimina los duplicados redundantes. SELECT DISTINCT elimina todos los conjuntos de filas duplicadas de la tabla de resultados, excepto uno.
FROM	Especifica las tablas que se utilizan en la consulta. Puede asignar alias a las tablas para reducir la complejidad o la ambigüedad de una sentencia.
WHERE	Define condiciones que determinan si las filas se incluyen en la tabla de resultados de la sentencia SELECT.
GROUP BY	Describe cómo se agrupan las filas en la tabla de resultados. También puede definir las expresiones de agrupación anidadas.
HAVING	Define condiciones para grupos.
ORDER BY	Controla el orden de las filas que se presentan en la tabla de resultados. Puede seleccionar la expresión de la columna de la cláusula SELECT que desea utilizar para definir el orden de las filas en la tabla de resultados.

SELECT recupera datos de una base de datos y los devuelve en forma de tabla. Esta tabla puede incorporarse en una aplicación o utilizarse de forma interactiva.

UADE

SELECT

LENGUAJE DE MANIPULACION DE DATOS

SELECT { ALL | DISTINCT } [TOP]

{ * | columna | expresión | expresión [AS] alias_columna }

Expresiones:

Agregación: { COUNT() | SUM() | AVG() | MIN() | MAX() }

Funciones Fila: { ROW_NUMBER() | RANK() | DENSE_RANK() }

Lógica condicional: CASE

Subconsultas: SELECT

```
select *

select all nombre, codDpto

select distinct nombre, codDpto

select nombre, horas * valor AS salario

select nombre, horas * valor

SELECT DISTINCT TOP 5
   Nombre + ' ' + Apellido AS NombreCompleto,
   Salario * 1.10 AS SalarioAjustado,
   COUNT(*) OVER () AS TotalEmpleados,
   ROW_NUMBER() OVER (ORDER BY Salario DESC) AS NumeroFila,
   CASE
        WHEN Salario > 3000 THEN 'Alto'
        ELSE 'Bajo'
   END AS NivelSalario
FROM empleados
WHERE Departamento = 'Ventas';
```


LENGUAJE DE MANIPULACION DE DATOS

SELECT *

FROM { tabla [[AS] alias] | vista [[AS] alias] }

- ✓ En la consulta más básica, la cláusula **FROM** especifica una sola tabla de la cual se extraen las columnas listadas en el **SELECT**.
- ✓ Se puede combinar datos de varias tablas utilizando diferentes tipos de **uniones (JOIN)** en la cláusula **FROM** (operaciones de reunión entre tablas).
- ✓ Usar alias para tablas permite acortar los nombres y mejorar la legibilidad de la consulta (múltiples tablas o subconsultas).
- ✓ Se puede incluir una **subconsulta** en la cláusula **FROM**, que crea una tabla temporal o derivada para su uso en la consulta principal. Debe usar siempre un alias.
- ✓ Cuando se indica más de un origen de datos sin especificar condición de unión, realiza un producto cartesiano **CROSS JOIN**

```sql
SELECT IdEmpleado, Nombre, Salario
FROM RecursosHumanos.Empleados

SELECT Empleados.Nombre, Departamentos.NombreDepartamento
FROM Empleados
INNER JOIN Departamentos ON Empleados.IdDepartamento =
Departamentos.IdDepartamento;

SELECT Nombre, SalarioPromedio
FROM (SELECT Nombre, AVG(Salario) AS SalarioPromedio
      FROM RecibosSueldo
      WHERE DataRecibo >= '2024-01-01'
      GROUP BY Nombre) AS SalariosPorEmpleado;

SELECT e.Nombre, d.NombreDepartamento
FROM Empleados e
INNER JOIN Departamentos d ON e.IdDepartamento = d.IdDepartamento;

--- PRODUCTO CARTESIANO
SELECT *
FROM tabla1, tabla2;
```

LENGUAJE DE MANIPULACION DE DATOS

SELECT *

FROM { tabla [[AS] alias] | vista [[AS] alias] }

WHERE [NOT] < expresión_booleana >

[{ AND | OR } [NOT] < expresión_booleana 2 >]

- ✓ Se utiliza para filtrar registros de una consulta basándose en una condición específica, limitando los registros resultados para los cuales un **predicado es verdadero**.
- ✓ Siempre debe ser una **expresión válida** y devolver un valor booleano.
- ✓ NULL → Cuando hay valores nulos en una comparación o expresión, el resultado puede ser verdadero, falso o **desconocido**.

Condiciones en la cláusula WHERE:

- **Operadores de comparación**: =, >, <, >=, <=, <>
- **Operadores lógicos**: AND, OR, NOT.
- **Operadores de rango**: BETWEEN ... AND.
- **Patrones de texto**: LIKE.
- **Valores nulos**: IS NULL, IS NOT NULL.
- **Subconsultas**
  - Escalar - un solo valor.
  - Múltiples Valores: operadores IN, ANY, ALL

UADE

Cláusula WHERE

LENGUAJE DE MANIPULACION DE DATOS

```sql
-- Uso de AND: Para combinar múltiples condiciones:
SELECT Nombre, Apellido
FROM Empleados
WHERE Salario > 3500 AND IdDepartamento = 2;

-- Uso de OR: Para especificar que solo una de varias condiciones debe cumplirse:
SELECT Nombre, Apellido
FROM Empleados
WHERE Salario < 3000 OR IdDepartamento = 1;

-- Uso de LIKE: Para realizar coincidencias de patrones en una cadena de texto.
-- El símbolo % representa cualquier cadena de caracteres.
SELECT Nombre, Apellido
FROM Empleados
WHERE Nombre LIKE 'A%';  -- Selecciona los empleados cuyo nombre empieza con 'A'

-- Uso de BETWEEN: Para seleccionar valores dentro de un rango.
SELECT Nombre, Apellido, Salario
FROM Empleados
WHERE Salario BETWEEN 3000 AND 5000; -- Selecciona empleados con un salario entre 3000 y 5000.

-- Uso de IS NULL y IS NOT NULL: Para trabajar con valores nulos.
SELECT Nombre, Apellido
FROM Empleados
WHERE Telefono IS NULL;  -- Selecciona empleados cuyo campo "Telefono" no está definido
```


LENGUAJE DE MANIPULACION DE DATOS

```sql
-- Ejemplo 1: Subconsulta escalar (retorna un solo valor)
SELECT Nombre, Apellido, Salario
FROM Empleados
WHERE Salario > (SELECT AVG(Salario) FROM Empleados);
```

```sql
-- Ejemplo 2: Subconsulta con múltiples valores (IN)
SELECT Nombre, Apellido
FROM Empleados
WHERE IdDepartamento IN (
    SELECT IdDepartamento
    FROM Empleados
    GROUP BY IdDepartamento
    HAVING COUNT(*) > 5
);
```

```sql
-- Ejemplo 3: Subconsulta con comparación (ANY o ALL)
SELECT Nombre, Apellido, Salario
FROM Empleados
WHERE Salario > ANY (
    SELECT Salario
    FROM Empleados
    WHERE IdDepartamento = 1  -- Supongamos que el Id 1 es Ventas
);
```

UADE

Cláusula WHERE

LENGUAJE DE MANIPULACION DE DATOS

SELECT

COLUMNA, { COUNT() | SUM() | AVG() | MIN() | MAX() }

FROM { tabla [[AS] alias] | vista [[AS] alias] }

GROUP BY COLUMNA [, COLUMNA 2, …]

- ✓ Se utiliza para agrupar filas que tienen valores comunes en una o más columnas, permitiendo aplicar funciones de agregación.
- ✓ Se puede agrupar por más de una columna y siempre deben aparecer en la cláusula **GROUP BY** aunque no es obligatorio que estén en la cláusula **SELECT**.
- ✓ No puede haber columnas en **SELECT** que no estén en **GROUP BY**

```sql
SELECT IdEmpleado, COUNT(IdVenta) AS TotalVentas,
      SUM(Cantidad) AS TotalCantidad,
      AVG(Cantidad) AS PromedioCantidad
FROM Ventas
GROUP BY IdEmpleado;


SELECT IdEmpleado, MAX(Nombre)
FROM Empleados
GROUP BY IdEmpleado;


SELECT IdEmpleado, IdProducto, COUNT(IdVenta) AS TotalVentas
FROM Ventas
GROUP BY IdEmpleado, IdProducto;


SELECT IdEmpleado, COUNT(IdVenta) AS TotalVentas
FROM Ventas
GROUP BY IdEmpleado
HAVING COUNT(IdVenta) > 1;
```

UADE

Cláusula GROUP BY / HAVING

LENGUAJE DE MANIPULACION DE DATOS

SELECT

COLUMNA, { COUNT() | SUM() | AVG() | MIN() | MAX() }

FROM { tabla [[AS] alias] | vista [[AS] alias] }

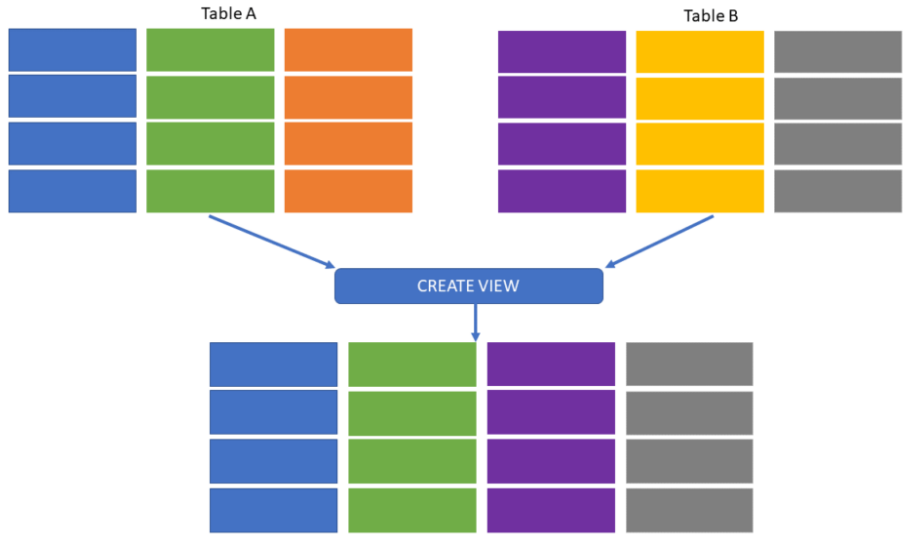
ORDER BY { COLUMNA | expresión } [ASC | DESC] , .. ;

- ✓ Ordena resultados de una consulta SELECT en forma ascendente (predeterminado) o descendente.
- ✓ En lugar de usar los nombres de las columnas, también se puede ordenar usando el número de la posición de la columna en la lista de selección.
- ✓ La expresión sirve para ordenar los resultados en base a una operación, cálculo o transformación de una o más columnas.

```sql
SELECT Nombre, COUNT(IdVenta) AS TotalVentas,
      SUM(Cantidad) AS TotalCantidad,
      AVG(Cantidad) AS PromedioCantidad
FROM Ventas
GROUP BY Nombre
ORDER BY Nombre;


SELECT Nombre, Salario
FROM Empleados
ORDER BY Salario DESC;


SELECT Nombre, Apellido, Salario
FROM Empleados
ORDER BY 3 DESC;


SELECT Nombre, Apellido, Salario
FROM Empleados
ORDER BY Salario * 12 DESC, Nombre + '-' + Apellido ASC;
```

SQL SERVER VIEWS

VISTAS



Las vistas en SQL son objetos virtuales que nos permiten acceder y manipular datos almacenados en una o varias tablas como si fueran tablas individuales.

En esencia, una vista es una consulta predefinida que se guarda en la base de datos y se comporta como una tabla virtual.

Se utilizan para simplificar consultas completas y puede contener proyecciones, filtros, uniones y otras operaciones SQL que nos permiten seleccionar y presentar los datos de una manera específica.

```sql
CREATE VIEW VistaVentas AS
SELECT Nombre, Apellido, Salario
FROM Empleados
WHERE Departamento = 'Ventas';

SELECT * FROM VistaVentas;
```

UADE

DDL

SQL SERVER VIEWS

VISTAS

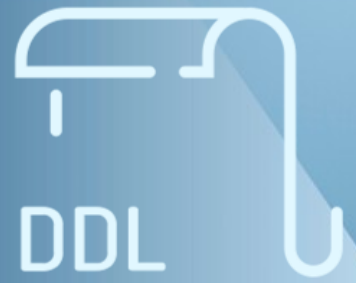
CREATE VIEW *identificador* AS
SELECT *
FROM *tabla*
WHERE *condición;*

1) ***Simplificación de consultas***
2) ***Organización de datos***
3) ***Seguridad y control de acceso***
4) ***Rendimiento mejorado***
5) ***Abstracción de la estructura de datos***

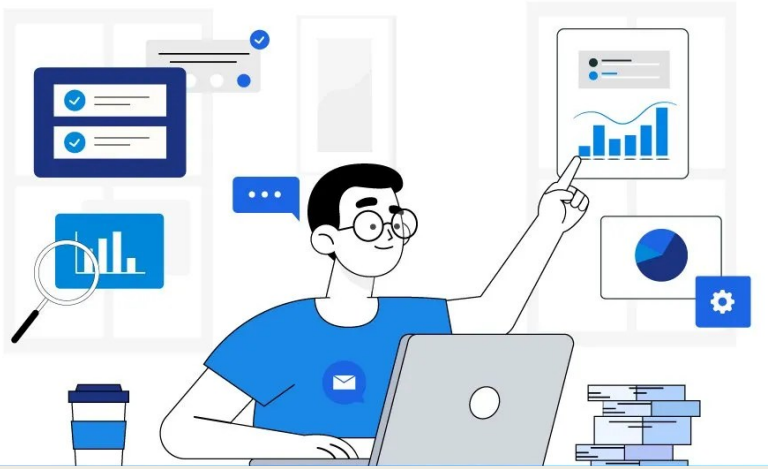
```sql
-- Vistas simples:
CREATE VIEW VistaEmpleados AS
SELECT Nombre, Apellido, Salario
FROM Empleados
WHERE Departamento = 'Ventas';

-- Vistas complejas:
CREATE VIEW VistaVentas AS
SELECT e.Nombre, e.Apellido, SUM(v.Cantidad) AS TotalVentas
FROM Empleados e
JOIN Ventas v ON e.IdEmpleado = v.IdEmpleado
GROUP BY e.Nombre, e.Apellido;

-- Vistas indexadas:
CREATE VIEW VistaSalarios
WITH SCHEMABINDING AS
SELECT IdEmpleado, Salario
FROM dbo.Empleados;
GO
CREATE UNIQUE CLUSTERED INDEX idx_salario
ON VistaSalarios (Salario);

-- Vistas con columnas calculadas:
CREATE VIEW VistaSalarioAnual AS
SELECT Nombre, Apellido, Salario, (Salario * 12) AS SalarioAnual
FROM Empleados;
```

UADE

Ventajas – Beneficios - Tipos



VISTAS



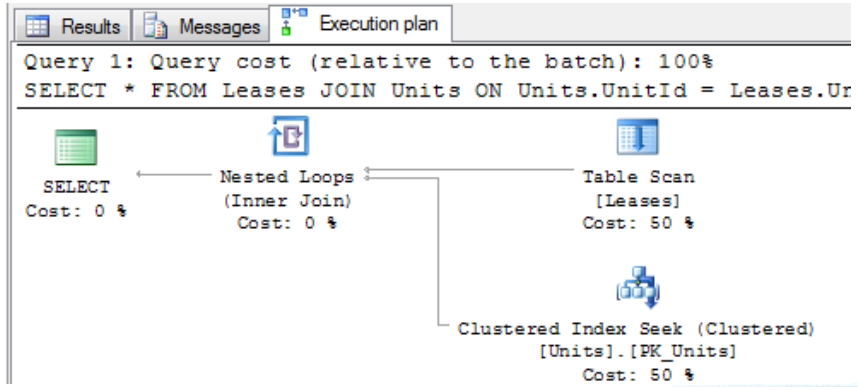
Requisitos para que una vista sea actualizable:

1. **Debe estar basada en una sola tabla**: Las vistas que combinan varias tablas o usan JOINs generalmente no son actualizables.
2. **No debe contener funciones agregadas** (SUM(), COUNT(), etc.).
3. **No debe usar DISTINCT, GROUP BY, o HAVING.**
4. **No debe contener subconsultas**.
5. **Todas las columnas necesarias** para realizar la actualización (por ejemplo, las claves primarias) deben estar incluidas en la vista.

```sql
-- Ejemplo de una vista actualizable:

CREATE VIEW VistaSimple AS
SELECT IdEmpleado, Nombre, Apellido, Salario
FROM Empleados;

--Insertar en la vista:

INSERT INTO VistaSimple (IdEmpleado, Nombre, Apellido,
Salario)
VALUES (6, 'Carlos', 'López', 4500);
```

VISTAS

1. **Identificar y Analizar el Problema:** usar herramientas como **Execution Plan** – Ctrl + M : *Include Actual Execution Plan*



2. **Revisar y simplificar la consulta:**
   Evitar subconsultas innecesarias.
   Evitar SELECT *

3. **Usar indices apropiados:**
   Columnas filtradas
   Indices compuestos

4. **Optimizar Joins y Subconsultas**

5. **Optimizar Funciones Agregadas y GROUP BY**

6. **Limitar datos recuperados:** usar LIMIT o TOP.

7. **Particionamiento de Tablas**

SQL - *Structured Query Language*

Bibliografía:

Introducción a los Sistemas de Bases de Datos - C. J. Date – 7° Edición

- CAPÍTULO 4: **Introducción a SQL**
- CAPÍTULO 5: **Dominios, relaciones y varrels base.**

Sistemas de Bases de Datos – Conceptos Fundamentales - Elmasri / Navathe – 5° Edición.

- **Capítulo 8: SQL-99: definición del esquema, restricciones, consultas y vistas.**

Próximo encuentro

Clase 7: SQL y Seguridad

Instrucciones / Comandos de SQL y su aplicación práctica.

Seguridad: perfiles de usuario y administración de permisos.

Anidamientos y operadores de conjunto: ALL, ANY.

LENGUAJE DE MANIPULACION DE DATOS

```sql
SELECT e.Nombre, d.NombreDepartamento
FROM Empleados e
INNER JOIN Departamentos d ON e.IdDepartamento = d.IdDepartamento;
```

TABLE 1

Products	Price
Kiwis	$6
Onions	$3
Tomatoes	$7

TABLE 2

Products	Quantity
Kiwis	10
Onions	6
Broccoli	5

INNER JOIN

Products	Price	Quantity
Kiwis	$6	10
Onions	$3	6