Aquí tienes un resumen detallado de la teoría presentada:

**Resumen detallado sobre SQL y su lenguaje DDL**

**1. Introducción a SQL**

SQL (Structured Query Language) es un lenguaje estándar utilizado para la comunicación con Sistemas de Gestión de Bases de Datos (SGBD). Se originó en IBM en los años setenta con el prototipo **SYSTEM R** y, posteriormente, Oracle Corporation lanzó la primera implementación comercial en 1979.

En 1986, el **ANSI (American National Standard Institute)** adoptó SQL como estándar para la gestión de bases de datos relacionales, seguido por **ISO (International Standardization Organization)** en 1987.

SQL se ha convertido en un lenguaje ampliamente utilizado en bases de datos, siendo integrado en diversos lenguajes de programación, como **Visual Basic**, para acceder a bases de datos relacionales.

**2. Características principales de SQL**

* Es un **lenguaje universal** utilizado en administración, desarrollo y gestión de datos en bases de datos.
* Es un **lenguaje declarativo**, lo que significa que el usuario especifica *qué* quiere obtener sin necesidad de indicar *cómo* debe obtenerse.
* No es **procedimental**, permitiendo manipular registros sin necesidad de estructuras de control.
* Facilita la ejecución de **consultas complejas** y manipulación de grandes volúmenes de datos.

**3. Clasificación de las sentencias SQL**

Las instrucciones SQL se dividen en tres grandes grupos:

**a) Sentencias DDL (Data Definition Language)**

Estas sentencias permiten definir y modificar la estructura de la base de datos. Se utilizan para:

* **Crear objetos** como tablas, vistas y procedimientos (CREATE).
* **Eliminar objetos** (DROP).
* **Modificar objetos** (ALTER).
* **Gestionar permisos** sobre los objetos (GRANT, REVOKE).

**b) Sentencias DML (Data Manipulation Language)**

Se encargan de la manipulación de los datos almacenados en la base de datos:

* **INSERT**: Agregar nuevos registros.
* **UPDATE**: Modificar registros existentes.
* **DELETE**: Eliminar registros.
* **SELECT**: Recuperar registros.
* **Bloqueo de tablas** para control de concurrencia.

**c) Lenguaje de Control de Transacciones**

Permite gestionar modificaciones realizadas por instrucciones **DML**, asegurando la integridad de la base de datos mediante:

* **COMMIT**: Confirma cambios realizados.
* **SAVEPOINT**: Crea puntos de restauración en una transacción.
* **ROLLBACK**: Revierte cambios no confirmados.
* **SET TRANSACTION**: Configura propiedades de una transacción.

**4. Aplicación del DDL en SQL**

El estudio del **DDL (Data Definition Language)** abarca la creación de los principales objetos del modelo relacional, especialmente **las tablas y sus restricciones de datos**. Su aplicación se sitúa dentro del proceso de diseño de bases de datos, pasando por:

1. **Diseño conceptual**
2. **Diseño lógico**
3. **Diseño físico** (donde se crean las tablas y restricciones en un SGBD como **Oracle**).

Un aspecto clave de las sentencias DDL es que incluyen **implícitamente el comando COMMIT**, lo que significa que los cambios estructurales en la base de datos son permanentes una vez ejecutadas.

Este resumen presenta los conceptos clave sobre SQL, sus características, tipos de sentencias y el papel del DDL en la gestión de bases de datos relacionales.

**2. Resumen detallado sobre los objetos de la base de datos en Oracle**

En Oracle, una base de datos puede contener múltiples **estructuras de datos**, las cuales deben ser definidas en la fase de diseño para su correcta implementación en la etapa de construcción del desarrollo.

**Principales objetos de la base de datos en Oracle**

1. **Tablas**
   * Son la unidad básica de almacenamiento.
   * Se componen de **registros (filas)** y **columnas (atributos)**.
2. **Vistas**
   * Representaciones lógicas de uno o más conjuntos de datos provenientes de tablas.
   * Funcionan como **tablas virtuales** y pueden ser manipuladas como si fueran tablas reales.
3. **Secuencias**
   * Generan valores automáticos, generalmente usados para claves primarias.
   * Son equivalentes a los campos autonuméricos de Access.
4. **Índices**
   * Se definen sobre una o varias columnas para **optimizar** la velocidad de búsqueda en las consultas.
   * Mejoran el rendimiento en operaciones de selección y filtrado.
5. **Sinónimos**
   * Permiten asignar **nombres alternativos** a tablas, vistas u otros objetos de la base de datos.
   * Facilitan el acceso a objetos sin necesidad de conocer su nombre exacto.

**Notas sobre las tablas en Oracle**

* **Creación dinámica**: Las tablas pueden crearse en cualquier momento, incluso cuando hay usuarios activos en la base de datos.
* **Gestión del espacio**: No es necesario definir el tamaño de una tabla al momento de su creación, ya que crecerá según el espacio disponible en la base de datos. Sin embargo, es recomendable **estimar su tamaño** para una mejor administración.
* **Modificación en línea**: La estructura de una tabla puede ser modificada sin necesidad de detener la base de datos.

**Reglas para nombrar objetos en Oracle**

Para garantizar la consistencia y evitar errores en la base de datos, los nombres de **tablas, columnas y otros objetos** deben seguir estas reglas:

✅ **Deben iniciar con una letra**.  
✅ **Longitud permitida:** Entre **1 y 30 caracteres**.  
✅ **Caracteres permitidos:** Letras (A-Z, a-z), números (0-9), y los símbolos \_, $, #.  
✅ **No pueden tener espacios** ni caracteres especiales no permitidos.  
✅ **No deben duplicar** nombres de otros objetos dentro del mismo esquema de usuario.  
✅ **No pueden ser palabras reservadas** de Oracle.

**Nota:** Los nombres de los objetos en Oracle **no son sensibles a mayúsculas y minúsculas**, por lo que DEPT, dept y Dept son considerados el mismo nombre.

Este resumen detalla los objetos esenciales de una base de datos Oracle, sus características y las reglas fundamentales para su correcta definición.

**Resumen detallado sobre los tipos de datos en Oracle**

En Oracle, los **tipos de datos** determinan el tipo de valores que pueden almacenarse en una columna de una tabla. Se dividen en **cadenas de caracteres, numéricos, fechas y datos binarios**.

**1. Tipos de datos de cadenas de caracteres**

* **CHAR(tamaño)**
  + Almacena cadenas de longitud fija (por defecto, 1).
  + Rango permitido: **1 a 2000 caracteres**.
  + Si el valor ingresado es menor al tamaño definido, se **rellena con espacios en blanco a la derecha**.
* **VARCHAR2(tamaño)**
  + Almacena cadenas de **longitud variable** hasta el tamaño especificado.
  + Rango permitido: **1 a 4000 caracteres**.
  + Si se introduce un valor mayor al tamaño definido, Oracle **genera un error**.
* **LONG**
  + Almacena cadenas de caracteres de longitud variable (hasta **2 GB**).
  + Solo se puede definir **una columna LONG por tabla**.
  + No puede usarse en restricciones de integridad ni para indexar.

**2. Tipos de datos numéricos**

* **NUMBER(P,S)**
  + Representa números enteros o decimales.
  + **P (Precisión):** Máximo número total de dígitos (1 a 38).
  + **S (Escala):** Cantidad de dígitos en la parte decimal (hasta 127).
  + Ejemplo: Sal\_Medio NUMBER(9,2) → Puede almacenar 4566770.86.
* **NUMBER(n)**
  + Si solo se indica n, se trabaja con **números enteros** (los decimales se consideran 0).
  + Ejemplo: Salario NUMBER(10).
* **NUMBER**
  + Puede almacenar tanto enteros como decimales sin restricciones específicas.
  + **El separador decimal por defecto es el punto (.), pero puede modificarse con ALTER SESSION**.

**3. Tipos de datos de fecha y hora**

* **DATE**
  + Almacena fechas y horas con el formato: **Siglo/Año/Mes/Día/Hora/Minutos/Segundos**.
  + El formato por defecto es DD/MM/YY, pero puede modificarse con NLS\_DATE\_FORMAT.
  + Ejemplo: fecha\_edición DATE.
* **TIMESTAMP(P)**
  + Extensión del tipo DATE que permite incluir fracciones de segundo con precisión P (por defecto, 6).

**4. Tipos de datos binarios y especiales**

* **BLOB (Binary Large Object)**
  + Almacena datos binarios hasta **4 GB** (imágenes, videos, etc.).
* **BFILE**
  + Similar a BLOB, pero almacena **referencias a archivos externos**, sin ser gestionados directamente por la base de datos.
  + Límite: **4 GB**.
* **RAW(N)**
  + Almacena datos binarios de longitud variable (hasta **2000 bytes**).
  + No es interpretable por Oracle (se usa para datos en formato binario).
* **LONG RAW**
  + Similar a RAW, pero admite hasta **4 GB**.
  + Se usa para almacenar **gráficos y otros formatos multimedia**.

Este resumen detalla los principales tipos de datos en Oracle, sus características y restricciones para definir correctamente las estructuras de las bases de datos.

**Resumen detallado sobre la sentencia CREATE TABLE en Oracle**

La sentencia CREATE TABLE forma parte del **sublenguaje DDL (Data Definition Language) de SQL** y se utiliza para la **creación de tablas** en una base de datos Oracle. Su ejecución tiene un **efecto inmediato**, registrando la estructura de la tabla en el **Diccionario de Datos**.

**Requisitos para crear una tabla**

Para poder ejecutar la sentencia CREATE TABLE, el usuario debe contar con:  
✅ **Privilegio CREATE TABLE**.  
✅ **Un área de almacenamiento (tablespace) disponible**.  
✅ **Especificar correctamente los nombres de la tabla y sus columnas, siguiendo las reglas de nomenclatura de Oracle**.  
✅ **Hasta 1000 columnas** pueden ser definidas por tabla.

**Sintaxis básica de CREATE TABLE**

CREATE [GLOBAL TEMPORARY] TABLE [schema.]nombretabla

(columna tipo\_datos [DEFAULT expr] [NOT NULL] )

[TABLESPACE espacio\_de\_tabla];

📌 **Descripción de los elementos de la sintaxis:**

* **GLOBAL TEMPORARY**: Crea una **tabla temporal** cuya estructura es visible para todas las sesiones, pero cuyos datos son **visibles solo para la sesión que los inserta**.
* **schema.**: Indica el esquema (propietario) de la tabla.
* **DEFAULT expr**: Permite asignar un valor por defecto si no se especifica uno en la inserción de datos.
* **NOT NULL**: Indica que la columna no puede contener valores nulos.
* **TABLESPACE espacio\_de\_tabla**: Define el tablespace en el que se almacenará la tabla. Si no se especifica, se utilizará el **tablespace por defecto del usuario**.

**Ejemplo de creación de una tabla en Oracle**

Se creará la tabla ALUMNOS con los siguientes atributos:

* NUM\_MAT: Número de matrícula (entero de 6 dígitos, no nulo).
* NOMBRE: Nombre del alumno (cadena de hasta 15 caracteres, no nulo).
* FECHA\_NAC: Fecha de nacimiento (tipo DATE).
* DIRECCION: Dirección del alumno (cadena de hasta 30 caracteres).
* LOCALIDAD: Localidad de residencia (cadena de hasta 15 caracteres).

CREATE TABLE ALUMNOS(

NUM\_MAT NUMBER(6) NOT NULL,

NOMBRE VARCHAR2(15) NOT NULL,

FECHA\_NAC DATE,

DIRECCION VARCHAR2(30),

LOCALIDAD VARCHAR2(15) -- Corregido el error tipográfico (VRCHAR2 → VARCHAR2)

) TABLESPACE USER\_DATA;

**Referencias a tablas de otros usuarios**

En Oracle, un **esquema** es una colección de objetos como tablas, vistas, sinónimos, secuencias, procedimientos almacenados, índices y clusters.

Si una tabla pertenece a otro usuario, debemos **anteponer el nombre del propietario** antes del nombre de la tabla, separado por un punto (.).

SELECT \* FROM usuario.tabla\_externa;

Para poder acceder a estas tablas, el usuario propietario **debe otorgar los permisos adecuados**.

**Uso de la opción DEFAULT**

La opción DEFAULT permite establecer un **valor por defecto** en una columna si no se proporciona uno explícitamente en la inserción de datos.

📌 **Reglas del valor por defecto:**  
✅ Puede ser un **literal, una expresión o una función SQL**.  
✅ **No puede ser el nombre de otra columna**.  
✅ Debe ser del mismo **tipo de datos** que la columna.

📌 **Ejemplo:**

CREATE TABLE dept2 (

numdept NUMBER(2),

nombre\_dept VARCHAR2(14) DEFAULT 'Sin nombre',

ubicacion VARCHAR2(13) DEFAULT 'No especificada'

);

📌 **Explicación del ejemplo:**

* Si no se especifica un valor para nombre\_dept, se insertará 'Sin nombre'.
* Si no se proporciona ubicacion, se insertará 'No especificada'.

Este resumen cubre los aspectos clave de la sentencia CREATE TABLE, incluyendo su sintaxis, requisitos, opciones avanzadas y ejemplos prácticos. 🚀

5 . **Resumen detallado sobre las tablas en una base de datos Oracle**

En una base de datos Oracle, las tablas pueden clasificarse en **tablas de usuario** y **tablas del Diccionario de Datos (DD)**.

**1️⃣ Tablas de usuario**

✅ Son creadas por los usuarios para almacenar información específica de sus aplicaciones.  
✅ Se consultan y gestionan mediante **sentencias SQL estándar** (SELECT, INSERT, UPDATE, DELETE).

**2️⃣ Diccionario de Datos (DD) en Oracle**

🔹 Es un conjunto de **tablas y vistas** creadas y mantenidas por Oracle.  
🔹 Contiene **información interna** sobre la base de datos, como usuarios, objetos, privilegios, restricciones, auditoría, etc.  
🔹 **No se puede modificar manualmente**, solo se puede consultar.  
🔹 **Todas las tablas del DD son propiedad del usuario SYS**.  
🔹 Los usuarios acceden al DD a través de **vistas predefinidasS**, ya que las tablas base son complejas.

📌 **Información almacenada en el Diccionario de Datos**:

* **Usuarios registrados en Oracle Server**.
* **Privilegios y permisos asignados a los usuarios**.
* **Objetos creados dentro de la base de datos (tablas, vistas, secuencias, índices, etc.)**.
* **Restricciones de integridad** aplicadas sobre las tablas.
* **Información de auditoría** de actividades dentro de la BD.

**3️⃣ Consultas comunes al Diccionario de Datos**

El Diccionario de Datos permite obtener información sobre los objetos dentro de la base de datos. Algunas vistas comunes son:

✅ **USER\_TABLES** → Contiene información sobre las tablas del usuario.  
✅ **USER\_OBJECTS** → Muestra todos los objetos del usuario, incluyendo tablas, vistas, secuencias, etc.  
✅ **USER\_CATALOG** → Lista todas las tablas, vistas, sinónimos y secuencias del usuario.

**4️⃣ Ejemplos de consultas al Diccionario de Datos**

📌 **Obtener nombres de las tablas propiedad del usuario**

SELECT table\_name FROM user\_tables;

🔹 Muestra solo los nombres de las tablas que pertenecen al usuario.

SELECT \* FROM user\_tables;

🔹 Muestra información detallada sobre las tablas del usuario.

📌 **Obtener información sobre las tablas del sistema (SYS.USER\_TABLES)**

SELECT table\_name FROM sys.user\_tables;

🔹 Devuelve el mismo resultado que user\_tables, pero especificando que la vista pertenece al esquema SYS.

DESC SYS.USER\_TABLES;

🔹 Muestra la estructura de la vista USER\_TABLES, incluyendo sus columnas y tipos de datos.

📌 **Consultar los diferentes tipos de objetos del usuario**

SELECT DISTINCT object\_type FROM user\_objects;

🔹 Muestra los tipos de objetos propiedad del usuario (por ejemplo, tablas y vistas).

SELECT object\_type, object\_name FROM user\_objects;

🔹 Lista los nombres y tipos de cada objeto propiedad del usuario.

SELECT object\_type, SUBSTR(object\_name, 1, 10) FROM user\_objects;

🔹 Muestra solo los primeros **10 caracteres** de los nombres de los objetos, para facilitar la visualización.

📌 **Consultar todas las tablas, vistas, sinónimos y secuencias del usuario**

SELECT \* FROM user\_catalog;

🔹 Devuelve un listado completo de todos los objetos pertenecientes al usuario actual.

**Conclusión**

* **Las tablas de usuario** contienen los datos específicos de las aplicaciones.
* **El Diccionario de Datos (DD)** almacena información sobre la estructura de la base de datos y sus objetos.
* **Oracle permite consultar el DD a través de vistas como USER\_TABLES, USER\_OBJECTS y USER\_CATALOG**.
* **Los usuarios pueden acceder al DD para obtener detalles sobre sus propios objetos, pero no pueden modificar su contenido**. 🚀

6 . **Creación de una Tabla mediante una Subconsulta en Oracle**

Además del método tradicional (CREATE TABLE con la definición manual de columnas y tipos de datos), Oracle permite crear tablas utilizando una **subconsulta** con la cláusula AS.

**1️⃣ Características de este método**

✅ **Crea una nueva tabla** basada en el resultado de una consulta (SELECT).  
✅ **Llena automáticamente la tabla** con los datos devueltos por la subconsulta.  
✅ **No copia restricciones** como claves primarias, foráneas, índices o NOT NULL.

**2️⃣ Sintaxis**

CREATE TABLE nombre\_tabla [(columna1, columna2, ...)] AS subconsulta;

🔹 nombre\_tabla: Nombre de la nueva tabla.  
🔹 (columna1, columna2, ...): (Opcional) Define nombres personalizados para las columnas.  
🔹 subconsulta: Sentencia SELECT que define la estructura y contenido de la tabla.

**3️⃣ Ejemplo básico**

CREATE TABLE dept50 AS

SELECT \* FROM dept WHERE deptno = 50;

🔹 Crea una tabla dept50 con la misma estructura de dept.  
🔹 Copia todas las filas donde deptno = 50.  
🔹 **No copia restricciones** (PRIMARY KEY, FOREIGN KEY, NOT NULL, etc.).

**4️⃣ Ejemplo con selección de columnas específicas**

CREATE TABLE empleados\_nueva (id, nombre, salario) AS

SELECT empno, ename, sal FROM empleados WHERE deptno = 20;

🔹 Crea una tabla con solo las columnas id, nombre y salario.  
🔹 Copia los datos de empleados donde deptno = 20.

**5️⃣ Consideraciones importantes**

🔹 **Tipos de datos**: La nueva tabla hereda los tipos de datos de la consulta original.  
🔹 **Restricciones**: No se copian, por lo que es necesario agregarlas manualmente (ALTER TABLE).  
🔹 **Índices y claves primarias**: Tampoco se copian, deben definirse después de la creación.

**6️⃣ Agregar restricciones manualmente**

Después de crear la tabla, podemos añadir restricciones como PRIMARY KEY:

ALTER TABLE empleados\_nueva ADD CONSTRAINT pk\_empleados PRIMARY KEY (id);

🔹 Define id como clave primaria en empleados\_nueva.

**Conclusión**

* CREATE TABLE ... AS subconsulta permite crear una tabla **copiando estructura y datos de otra tabla**.
* **No hereda restricciones**, por lo que estas deben agregarse manualmente.
* Es un método **rápido y eficiente** para crear copias parciales de una tabla o realizar backups de datos. 🚀

/////////////////////////////////////////

**ALTER TABLE en Oracle: Modificación de Tablas**

La sentencia ALTER TABLE permite modificar la estructura de una tabla existente en una base de datos Oracle sin necesidad de eliminarla y volverla a crear.

**1️⃣ Usos principales de ALTER TABLE**

✅ **Agregar una nueva columna** (ADD)  
✅ **Modificar una columna existente** (MODIFY)  
✅ **Eliminar una columna** (DROP COLUMN)  
✅ **Renombrar una columna** (RENAME COLUMN)  
✅ **Marcar una columna como no utilizada** (SET UNUSED)  
✅ **Eliminar columnas marcadas como no utilizadas** (DROP UNUSED COLUMNS)

**2️⃣ Agregar una nueva columna (ADD)**

Se usa cuando necesitamos incluir una nueva columna en la tabla.

**📌 Sintaxis:**

ALTER TABLE nombre\_tabla

ADD (columna tipo\_dato [DEFAULT expr]);

**🔹 Ejemplo:**

ALTER TABLE dept30

ADD (puesto VARCHAR2(9));

🔹 Se añade la columna puesto con tipo VARCHAR2(9).  
🔹 La nueva columna aparece al final de la tabla.

**3️⃣ Modificar una columna existente (MODIFY)**

Permite cambiar el tipo de datos, el tamaño o el valor por defecto de una columna.

**📌 Sintaxis:**

ALTER TABLE nombre\_tabla

MODIFY (columna tipo\_dato [DEFAULT expr]);

**🔹 Ejemplo:**

ALTER TABLE dept

MODIFY (provincia VARCHAR2(15));

🔹 Modifica la columna provincia, cambiando su tamaño a VARCHAR2(15).

**4️⃣ Eliminar una columna (DROP COLUMN)**

Elimina completamente una columna de una tabla. **Esta operación es irreversible** y tiene un COMMIT implícito.

**📌 Sintaxis:**

ALTER TABLE nombre\_tabla

DROP COLUMN nombre\_columna;

**🔹 Ejemplo:**

ALTER TABLE dept

DROP COLUMN provincia;

🔹 La columna provincia se elimina permanentemente de la tabla dept.

**5️⃣ Renombrar una columna (RENAME COLUMN)**

Cambia el nombre de una columna sin afectar su contenido ni tipo de datos.

**📌 Sintaxis:**

ALTER TABLE nombre\_tabla

RENAME COLUMN old\_name TO new\_name;

**🔹 Ejemplo:**

ALTER TABLE empleados

RENAME COLUMN apellido TO apellidos;

🔹 La columna apellido se renombra como apellidos.

**6️⃣ Marcar una columna como no utilizada (SET UNUSED)**

En lugar de eliminar directamente una columna, se marca como **"no utilizada"**.  
🔹 **Ventaja**: La operación es más rápida que DROP COLUMN y evita bloqueos en la base de datos.  
🔹 **Desventaja**: No libera el espacio en disco hasta que se ejecute DROP UNUSED COLUMNS.

**📌 Sintaxis:**

ALTER TABLE nombre\_tabla

SET UNUSED (nombre\_columna);

**🔹 Ejemplo:**

ALTER TABLE dept30

SET UNUSED (ename);

🔹 La columna ename ya no está disponible para consultas o modificaciones.

**7️⃣ Eliminar columnas marcadas como no utilizadas (DROP UNUSED COLUMNS)**

Cuando ya no se necesiten las columnas marcadas como UNUSED, se pueden eliminar para recuperar espacio en disco.

**📌 Sintaxis:**

ALTER TABLE nombre\_tabla

DROP UNUSED COLUMNS;

**🔹 Ejemplo:**

ALTER TABLE dept30

DROP UNUSED COLUMNS;

🔹 Todas las columnas marcadas como UNUSED en dept30 se eliminan completamente.

**8️⃣ Consideraciones importantes**

⚠ **ALTER TABLE es una operación que afecta directamente a la estructura de la tabla.**  
⚠ **DROP COLUMN es irreversible y tiene un COMMIT implícito.**  
⚠ **Si se marca una columna como UNUSED, ya no puede ser utilizada en consultas.**  
⚠ **Al modificar el tipo de dato de una columna (MODIFY), los valores existentes deben ser compatibles con el nuevo tipo.**

**Conclusión**

La sentencia ALTER TABLE es fundamental para la administración de bases de datos en Oracle, ya que permite realizar modificaciones estructurales de manera eficiente. Dependiendo de la necesidad, podemos **añadir, modificar, eliminar, renombrar o marcar columnas como no utilizadas**. 🚀

8.9.10.11///////////////////////////////////////////////////////////////////////////

**8. La sentencia DROP TABLE**

La sentencia DROP TABLE se utiliza para eliminar completamente una tabla, incluyendo su definición y todos los datos que contiene. También elimina los índices asociados a la tabla. **Es una sentencia DDL**, lo que significa que no se puede realizar un ROLLBACK (no hay reversión de esta operación).

**⚠️ Importante**: Solo el propietario de la tabla o un usuario con el permiso DROP ANY TABLE puede eliminarla.

**📌 Sintaxis**

DROP TABLE nombre\_tabla;

**🔹 Ejemplo:**

DROP TABLE dept;

🔹 Esto eliminará la tabla dept junto con todos sus datos e índices.

**9. Cambiar el nombre de un objeto (RENAME)**

La sentencia RENAME se utiliza para cambiar el nombre de una tabla, vista, secuencia o sinónimo en Oracle. El **propietario del objeto** es el único que puede renombrarlo.

**📌 Sintaxis**

RENAME nombre\_actual TO nuevo\_nombre;

**🔹 Ejemplo:**

RENAME dept TO departamento;

🔹 Esto cambia el nombre de la tabla dept a departamento.

**10. Truncar una tabla (TRUNCATE)**

La sentencia TRUNCATE se utiliza para eliminar todos los registros de una tabla y liberar el espacio de almacenamiento que ocupan. Es una alternativa a la sentencia DELETE de DML, pero con **dos diferencias clave**:

1. **No se puede hacer ROLLBACK**, es decir, una vez ejecutada, no se puede deshacer.
2. **Libera espacio**, a diferencia de DELETE, que no libera el espacio de almacenamiento.

Para usar TRUNCATE, el usuario debe ser el propietario de la tabla o tener el privilegio DELETE TABLE.

**📌 Sintaxis**

TRUNCATE TABLE nombre\_tabla;

**🔹 Ejemplo:**

TRUNCATE TABLE dept;

🔹 Esto eliminará todos los registros de la tabla dept, y liberará el espacio que ocupaban.

**11. Añadir comentarios a una tabla o columna**

En Oracle, puedes agregar comentarios descriptivos a tablas y columnas. Estos comentarios se almacenan en el Diccionario de Datos y pueden ser visualizados más tarde mediante las vistas ALL\_TAB\_COMMENTS o USER\_TAB\_COMMENTS para tablas, y ALL\_COL\_COMMENTS o USER\_COL\_COMMENTS para columnas.

**📌 Sintaxis para tablas**

COMMENT ON TABLE nombre\_tabla IS 'comentario descriptivo';

**📌 Sintaxis para columnas**

COMMENT ON COLUMN nombre\_tabla.nombre\_columna IS 'comentario descriptivo';

**🔹 Ejemplo para tabla:**

COMMENT ON TABLE dept IS 'Tabla que contiene información de los departamentos';

**🔹 Ejemplo para columna:**

COMMENT ON COLUMN dept.deptno IS 'Número único de cada departamento';

🔹 Esto agrega un comentario descriptivo sobre la tabla dept y la columna deptno.

**Resumen de las sentencias**

1. **DROP TABLE**: Elimina una tabla y sus datos. No puede deshacerse.
2. **RENAME**: Cambia el nombre de una tabla, vista, secuencia o sinónimo.
3. **TRUNCATE**: Elimina todos los registros de una tabla y libera espacio, sin posibilidad de hacer ROLLBACK.
4. **COMMENT**: Añade comentarios descriptivos a tablas y columnas para documentación en el Diccionario de Datos.

Estas sentencias permiten gestionar de manera eficiente la estructura de la base de datos y mejorar la documentación.

12//////////////////////////////

**12. ¿Qué son las restricciones?**

Las **restricciones** en bases de datos son reglas que aseguran la **integridad de los datos**. Estas reglas definen los límites que los datos deben cumplir para ser almacenados correctamente en las tablas, evitando que se ingresen valores no válidos.

**Tipos de integridad que garantizan las restricciones:**

1. **Integridad de datos**: Asegura que los valores almacenados en las columnas sean válidos y consistentes.
   * Ejemplo: Evitar que se almacenen valores negativos o cadenas en minúsculas.
2. **Integridad referencial**: Asegura que las relaciones entre diferentes tablas se mantengan correctas, lo que significa que una columna en una tabla (clave foránea) debe corresponder a un valor válido en otra tabla (clave primaria).

**Restricciones comunes en Oracle:**

| **Restricción** | **Descripción** |
| --- | --- |
| **NOT NULL** | Impide que una columna tenga valores nulos. |
| **UNIQUE** | Asegura que una columna (o combinación de columnas) tenga valores únicos (sin repetirse). |
| **PRIMARY KEY** | Define una clave primaria, que es una combinación de una o más columnas con valores únicos y no nulos. |
| **FOREIGN KEY** | Define una clave foránea que establece una relación entre dos tablas, asegurando que los valores coincidan con la clave primaria de otra tabla. |
| **CHECK** | Define una condición que debe ser verdadera para insertar o actualizar los datos. |

**¿Cómo se definen las restricciones?**

Las restricciones pueden ser definidas en el momento de la creación de la tabla o después de que la tabla ya ha sido creada.

**Se pueden definir a nivel de columna o a nivel de tabla**, dependiendo de la restricción. Por ejemplo, las restricciones **NOT NULL** y **UNIQUE** generalmente se asignan a nivel de columna, mientras que las **FOREIGN KEY** y **PRIMARY KEY** se pueden asignar a nivel de tabla.

**Recomendación**: Es una buena práctica asignarles **nombres descriptivos** a las restricciones. Si no se les asigna un nombre, Oracle asignará un nombre por defecto del tipo SYS\_CX (donde X es un número).

**¿Dónde ver las restricciones?**

Las restricciones se almacenan en el **Diccionario de Datos** de Oracle, específicamente en la vista **USER\_CONSTRAINTS**, donde se puede consultar información sobre las restricciones existentes en las tablas de un usuario.

**Ejemplo de creación de restricciones:**

* **NOT NULL**:

CREATE TABLE empleados (

id NUMBER(6) NOT NULL,

nombre VARCHAR2(50) NOT NULL

);

* **UNIQUE**:

CREATE TABLE productos (

id NUMBER(6) PRIMARY KEY,

codigo\_producto VARCHAR2(50) UNIQUE

);

* **PRIMARY KEY**:

CREATE TABLE clientes (

id NUMBER(6),

nombre VARCHAR2(50),

PRIMARY KEY (id)

);

* **FOREIGN KEY**:

CREATE TABLE pedidos (

id NUMBER(6),

id\_cliente NUMBER(6),

FOREIGN KEY (id\_cliente) REFERENCES clientes(id)

);

* **CHECK**:

CREATE TABLE empleados (

id NUMBER(6),

salario NUMBER(8, 2),

CHECK (salario >= 0)

);

**Resumen:**

Las **restricciones** son esenciales para mantener la calidad y consistencia de los datos en la base de datos. Aseguran que los datos sean correctos, no nulos cuando no debe serlo, y que se respeten las relaciones entre las tablas. Se pueden definir durante la creación de la tabla o después, y son fundamentales para mantener la integridad referencial y de los datos.

13///////////////////////////////////

**13. Definición de restricciones**

Las **restricciones** son reglas que se aplican a los datos almacenados en las tablas para asegurar que se cumplan ciertas condiciones, garantizando la integridad y validez de los mismos. Estas restricciones se crean normalmente cuando se define una tabla, pero también pueden ser añadidas posteriormente. Además, pueden desactivarse temporalmente cuando sea necesario.

**Sintaxis general de creación de restricciones:**

CREATE TABLE tabla

(columna1 tipo\_datos

[CONSTRAINT\_NOMBRERESTRICCION] [NOT NULL]

[CONSTRAINT\_NOMBRERESTRICCION] [UNIQUE]

[CONSTRAINT\_NOMBRERESTRICCION] [PRIMARY KEY]

[CONSTRAINT\_NOMBRERESTRICCION] [DEFAULT VALOR]

[REFERENCES NOMBRE\_TABLA]

[CHECK CONDICION],

columna2 tipo\_datos

[CONSTRAINT\_NOMBRERESTRICCION] [NOT NULL]

[UNIQUE] [PRIMARY KEY]

[DEFAULT VALOR]

[REFERENCES NOMBRE\_TABLA]

[CHECK CONDICION]

...

[constraint\_tabla]);

**Ejemplos de creación de restricciones:**

1. **Restricción sin nombre explícito:**

CREATE TABLE empleados

(empno NUMBER(4),

ename VARCHAR2(10),

deptno NUMBER(2) NOT NULL,

CONSTRAINT empleados\_empno\_pk PRIMARY KEY (empno));

1. **Restricción con nombre explícito:**

CREATE TABLE empleados

(empno NUMBER(4),

ename VARCHAR2(10),

deptno NUMBER(2) CONSTRAINT empleados\_deptno\_nn NOT NULL,

CONSTRAINT empleados\_empno\_pk PRIMARY KEY (empno));

**Tipos de restricciones en Oracle:**

1. **NOT NULL**:  
   Impide que una columna tenga valores nulos. Las columnas sin esta restricción pueden almacenar valores nulos de forma predeterminada.

**Ejemplo:**

CREATE TABLE empleados (empno NUMBER(4), ename VARCHAR2(10) NOT NULL);

1. **UNIQUE**:  
   Asegura que los valores de una columna (o combinación de columnas) sean únicos en toda la tabla. Permite valores nulos, a menos que se use junto con la restricción **NOT NULL**.

**Ejemplo:**

CREATE TABLE dept

(deptno NUMBER(2),

dname VARCHAR2(14),

loc VARCHAR2(13),

CONSTRAINT dept\_dname\_uq UNIQUE (dname));

1. **PRIMARY KEY**:  
   Define una clave primaria que garantiza que los valores de una columna o conjunto de columnas sean únicos y no nulos. Solo puede existir una clave primaria por tabla.

**Ejemplo:**

CREATE TABLE dept

(deptno NUMBER(2),

dname VARCHAR2(14),

loc VARCHAR2(13),

CONSTRAINT dept\_deptno\_pk PRIMARY KEY (deptno));

1. **FOREIGN KEY** (Clave foránea):  
   Establece una relación entre una columna en una tabla (clave foránea) y la clave primaria de otra tabla. Esta restricción asegura que los valores de la clave foránea coincidan con los valores de la clave primaria de la otra tabla, o sean nulos.

**Ejemplo (a nivel de tabla):**

CREATE TABLE empleados

(empno NUMBER(4),

ename VARCHAR2(10) NOT NULL,

job VARCHAR2(9),

deptno NUMBER(3) NOT NULL,

CONSTRAINT emp\_deptno\_fk FOREIGN KEY (deptno) REFERENCES dept (deptno));

**Ejemplo (a nivel de columna):**

CREATE TABLE empleados

(empno NUMBER(4),

ename VARCHAR2(10) NOT NULL,

job VARCHAR2(9),

deptno NUMBER(3) CONSTRAINT emp\_deptno\_fk REFERENCES dept (deptno));

Si se desea borrar automáticamente las filas de la tabla hija cuando se elimina la fila correspondiente en la tabla principal, se puede usar la opción **ON DELETE CASCADE**.

**Ejemplo con CASCADE:**

CONSTRAINT emp\_deptno\_fk FOREIGN KEY (deptno) REFERENCES dept (deptno) ON DELETE CASCADE;

1. **CHECK**:  
   Define una condición que debe cumplirse para que los valores de una columna sean válidos. Puede usarse a nivel de columna o de tabla.

**Ejemplo (a nivel de columna):**

CREATE TABLE emp

(deptno NUMBER(2) CONSTRAINT emp\_deptno\_ck CHECK (deptno BETWEEN 10 AND 99));

**Ejemplo (a nivel de tabla):**

CREATE TABLE emp

(deptno NUMBER(2),

CONSTRAINT emp\_deptno\_ck CHECK (deptno BETWEEN 10 AND 99));

**Ejemplo práctico:**

**Crear la tabla PERSONAS** con restricciones:

CREATE TABLE PERSONAS

(

DNI NUMBER(8) CONSTRAINT pk\_personas PRIMARY KEY,

NOMBRE VARCHAR2(15),

DIRECCION VARCHAR2(15),

POBLACION VARCHAR2(15),

COD\_PROV NUMBER(2),

CONSTRAINT fk\_codprov FOREIGN KEY (COD\_PROV) REFERENCES PROVINCIAS

);

**Crear la tabla PROVINCIAS**:

CREATE TABLE PROVINCIAS

(

COD\_PROV NUMBER(2) CONSTRAINT pk\_provincias PRIMARY KEY,

NOMBRE VARCHAR2(15)

);

En este caso, al intentar crear primero la tabla PERSONAS, Oracle lanzará un error si la tabla PROVINCIAS no ha sido creada antes, ya que la columna COD\_PROV de PERSONAS hace referencia a la clave primaria de PROVINCIAS. Además, si se elimina una provincia de la tabla PROVINCIAS, las personas asociadas a esa provincia también se eliminarán si se ha especificado la opción **ON DELETE CASCADE**.

**Resumen de restricciones:**

* **Restricciones de borrado**: No se puede eliminar una fila que sea referenciada por una clave foránea. Tampoco se puede actualizar la clave primaria si está referenciada.
* **Restricciones en cascada**: Si se borra una fila en la tabla principal (por ejemplo, PROVINCIAS), todas las filas referenciadas en la tabla hija (por ejemplo, PERSONAS) se eliminarán automáticamente si se especifica la opción **ON DELETE CASCADE**.

14/////////////////////////////////////////////////

**14. Operaciones con restricciones**

En Oracle, se pueden realizar varias operaciones con las restricciones que están definidas en las tablas, tales como **añadir**, **eliminar**, **activar**, **desactivar** y **visualizar**. Estas operaciones permiten gestionar las restricciones en tablas ya existentes, permitiendo ajustarlas según sea necesario.

**Operaciones disponibles:**

**1. Añadir una restricción**

Para añadir una restricción a una tabla que ya existe, se utiliza la sentencia **ALTER TABLE** con la cláusula **ADD**. Para añadir una restricción de tipo **NOT NULL**, se debe usar **ALTER TABLE** con la cláusula **MODIFY**, pero esta operación solo es válida si la tabla no tiene filas.

**Sintaxis**:

ALTER TABLE tabla

ADD [CONSTRAINT restricción] tipo\_restricción (columna);

**Ejemplo**:

ALTER TABLE empleados

ADD CONSTRAINT empleados\_mgr\_fk FOREIGN KEY (mgr) REFERENCES empleados(empno);

En este ejemplo, se agrega una clave foránea **mgr** que hace referencia a la columna **empno** en la misma tabla **empleados**.

**2. Eliminar una restricción**

Para eliminar una restricción ya existente, se utiliza **ALTER TABLE** con la cláusula **DROP CONSTRAINT**. Cuando se elimina una restricción de integridad, esta ya no está disponible en el Diccionario de Datos.

**Sintaxis**:

ALTER TABLE tabla DROP CONSTRAINT nombre\_constraint;

**Ejemplos**:

ALTER TABLE emp DROP CONSTRAINT emp\_mgr\_fk;

ALTER TABLE dept DROP PRIMARY KEY CASCADE;

En el segundo ejemplo, no solo se elimina la restricción **PRIMARY KEY** de la tabla **dept**, sino también la **Foreign Key** asociada en la columna **emp.deptno**.

**3. Desactivar restricciones**

Las restricciones pueden desactivarse temporalmente sin eliminarlas, usando **ALTER TABLE** con la cláusula **DISABLE**. Si la restricción tiene otras restricciones dependientes, se pueden desactivar con la opción **CASCADE**.

**Sintaxis**:

ALTER TABLE tabla

DISABLE CONSTRAINT nombre\_restricción [CASCADE];

**Ejemplo**:

ALTER TABLE empleados

DISABLE CONSTRAINT empleados\_empno\_pk CASCADE;

Este ejemplo desactiva la restricción **empno\_pk** en la tabla **empleados** y también desactiva las restricciones dependientes que estén asociadas.

**4. Activar restricciones**

Una vez que una restricción ha sido desactivada, se puede volver a activar utilizando **ALTER TABLE** con la cláusula **ENABLE**. Si se activa una restricción **UNIQUE** o **PRIMARY KEY**, Oracle crea automáticamente un índice para esa columna.

**Sintaxis**:

ALTER TABLE tabla

ENABLE CONSTRAINT nombre\_restricción;

**Ejemplo**:

ALTER TABLE emp ENABLE CONSTRAINT emp\_empno\_pk;

Este ejemplo vuelve a activar la restricción **empno\_pk** en la tabla **emp**.

**Visualización de restricciones en el Diccionario de Datos**

Las restricciones se almacenan en el Diccionario de Datos de Oracle, y se pueden visualizar mediante las vistas correspondientes. Algunas de estas vistas son:

* **USER\_CONSTRAINTS**: Muestra las restricciones definidas por el usuario.
* **ALL\_CONSTRAINTS**: Muestra las restricciones a las que el usuario tiene acceso.
* **DBA\_CONSTRAINTS**: Muestra todas las restricciones del esquema de la base de datos.

Estas vistas permiten consultar información sobre las restricciones activas, las columnas que las componen, y el tipo de restricción que están implementando.

**Resumen de operaciones con restricciones:**

1. **Añadir** una restricción usando ALTER TABLE ... ADD.
2. **Eliminar** una restricción usando ALTER TABLE ... DROP CONSTRAINT.
3. **Desactivar** una restricción con ALTER TABLE ... DISABLE.
4. **Activar** una restricción con ALTER TABLE ... ENABLE.

Estas operaciones proporcionan flexibilidad para gestionar las restricciones a lo largo del ciclo de vida de las tablas en la base de datos.

15////////////////////////////

**15. Consulta de restricciones en el Diccionario de Datos**

Una vez que se han creado las tablas y las restricciones, se pueden consultar para verificar su existencia y detalles en el Diccionario de Datos de Oracle. A continuación, se explican dos vistas útiles que permiten consultar las restricciones de las tablas.

**1. Vista USER\_CONSTRAINTS**

La vista **USER\_CONSTRAINTS** proporciona información sobre las restricciones definidas en las tablas del esquema del usuario actual. Aquí se pueden ver todos los nombres de las restricciones y sus definiciones.

* **constraint\_name**: El nombre de la restricción.
* **constraint\_type**: El tipo de la restricción (por ejemplo, **C** para **CHECK**, **P** para **PRIMARY KEY**, **U** para **UNIQUE**, etc.).
* **search\_condition**: La condición definida para las restricciones de tipo **CHECK**.

**Sintaxis**:

SELECT constraint\_name, constraint\_type, search\_condition

FROM USER\_CONSTRAINTS

WHERE table\_name = 'EMP';

**Explicación**:

* **constraint\_name**: El nombre de la restricción.
* **constraint\_type**: El tipo de restricción (ej. **P** para Primary Key, **F** para Foreign Key, **C** para Check, **U** para Unique).
* **search\_condition**: La condición asociada a las restricciones tipo **CHECK** (si existe).

Este ejemplo muestra las restricciones en la tabla **EMP**.

**2. Vista USER\_CONS\_COLUMNS**

La vista **USER\_CONS\_COLUMNS** permite obtener detalles sobre las columnas que están asociadas con las restricciones definidas en las tablas. Es útil cuando se quiere ver qué columnas están involucradas en las restricciones y cómo están asociadas.

* **constraint\_name**: El nombre de la restricción.
* **column\_name**: El nombre de la columna asociada con la restricción.

**Sintaxis**:

SELECT constraint\_name, column\_name

FROM USER\_CONS\_COLUMNS

WHERE table\_name = 'EMP';

**Explicación**:

* **constraint\_name**: El nombre de la restricción a la que pertenece la columna.
* **column\_name**: El nombre de la columna que está asociada con la restricción.

Este ejemplo muestra las columnas en la tabla **EMP** que están asociadas a las restricciones definidas.

**Ejemplo completo de consultas:**

1. **Ver todas las restricciones de la tabla EMP**:

SELECT constraint\_name, constraint\_type, search\_condition

FROM USER\_CONSTRAINTS

WHERE table\_name = 'EMP';

1. **Ver las columnas asociadas con las restricciones de la tabla EMP**:

SELECT constraint\_name, column\_name

FROM USER\_CONS\_COLUMNS

WHERE table\_name = 'EMP';

Estas vistas son herramientas útiles para explorar y verificar las restricciones de las tablas, especialmente cuando se necesita verificar la integridad de los datos y el cumplimiento de las reglas definidas en la base de datos.

16////////////////////////////////

**16. Vistas**

Una **vista** en bases de datos es una representación lógica de un subconjunto de una o más tablas. No contiene datos propios, sino que actúa como una ventana que permite ver o modificar datos de las tablas base. Es importante entender cómo se utilizan las vistas para gestionar y presentar datos de forma eficiente.

**Usos comunes de las vistas:**

1. **Restringir el acceso a los datos**: Se pueden crear vistas para limitar qué información está disponible para ciertos usuarios o roles.
2. **Simplificar consultas complejas**: Las vistas permiten abstraer consultas complicadas, como uniones de múltiples tablas, para que los usuarios no necesiten escribirlas repetidamente.
3. **Independencia de los datos**: Una vista puede combinar datos de diversas tablas, proporcionando una interfaz estable aunque la estructura de las tablas cambie.
4. **Presentar datos de diferentes formas**: Las vistas pueden ser configuradas para mostrar datos según los criterios específicos de diferentes grupos de usuarios.

**Tipos de vistas:**

* **Vistas Simples**:
  + Utilizan solo una tabla.
  + No contienen funciones agregadas ni cláusulas GROUP BY.
  + Se pueden realizar operaciones DML (INSERT, UPDATE, DELETE) sobre ellas si cumplen ciertos requisitos.
* **Vistas Complejas**:
  + Utilizan más de una tabla.
  + Contienen funciones agregadas o grupos de datos.
  + No siempre permiten operaciones DML.

**Sintaxis para Crear una Vista (CREATE VIEW):**

CREATE [OR REPLACE] [FORCE | NOFORCE] VIEW nombre\_vista

[(columna[, columna]...)]

AS subconsulta

[WITH CHECK OPTION [CONSTRAINT restricción]]

[WITH READ ONLY];

* **OR REPLACE**: Si ya existe una vista con el mismo nombre, se reemplaza.
* **FORCE / NOFORCE**:
  + **FORCE**: Crea la vista sin importar si las tablas base existen.
  + **NOFORCE**: Crea la vista solo si las tablas base existen (valor por defecto).
* **subconsulta**: La consulta SELECT que define los datos visibles en la vista. No puede contener ORDER BY.
* **WITH CHECK OPTION**: Garantiza que solo se puedan insertar o actualizar filas que cumplan con la condición de la vista.
* **WITH READ ONLY**: Evita cualquier operación DML (INSERT, UPDATE, DELETE) sobre la vista.

**Ejemplos de Creación de Vistas:**

* **Vista simple** (empleados del departamento 30):

CREATE VIEW empvu10 AS

SELECT empno, ename, job

FROM emp

WHERE deptno = 10;

* **Vista compleja** (información de salarios por departamento):

CREATE VIEW deptsumvu AS

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

FROM emp e, dept d

WHERE e.deptno = d.deptno

GROUP BY d.dname;

**Recuperación de Datos desde una Vista:**

Se pueden recuperar datos de una vista de la misma manera que de una tabla:

SELECT \* FROM empvu10;

**Modificación de una Vista (OR REPLACE):**

Usando CREATE OR REPLACE, se puede modificar una vista sin tener que eliminarla primero:

CREATE OR REPLACE VIEW empvu10

(num\_empleado, nombre\_empleado, empleo)

AS

SELECT empno, ename, job

FROM emp

WHERE deptno = 10;

**Reglas para realizar operaciones DML sobre vistas:**

Se pueden realizar operaciones DML (INSERT, UPDATE, DELETE) sobre las vistas, pero con ciertas restricciones:

* **No se puede realizar DML** si la vista:
  + Contiene funciones de agregación (SUM, AVG, etc.).
  + Utiliza GROUP BY, DISTINCT, o expresiones complejas en las columnas.
  +  **Si la agregación está solo en una subconsulta (no en la proyección principal), sí se puede hacer DML.**
  +  **Si la vista usa SUM, AVG, GROUP BY, o DISTINCT directamente en la proyección, NO se puede hacer DML.**
* **INSERT**: Se puede insertar datos en la vista siempre que no contenga las condiciones anteriores y las columnas NOT NULL estén correctamente mapeadas.

**Cláusula WITH CHECK OPTION:**

Es posible asegurar que solo se puedan insertar o actualizar filas que cumplen con las condiciones de la vista.

**Ejemplo**: Si se crea una vista con WITH CHECK OPTION, cualquier intento de modificar datos fuera del rango de la vista fallará:

CREATE OR REPLACE VIEW empvu20

AS

SELECT \*

FROM EMP

WHERE deptno = 20

WITH CHECK OPTION CONSTRAINT empvu20\_ck;

UPDATE empvu20

SET deptno = 22

WHERE empno = 7788;

**Resultado**: El intento de cambiar deptno a 22 fallará porque no es válido según la condición de la vista (deptno = 20).

**Opción WITH READ ONLY:**

Esta opción impide que se realicen operaciones DML sobre la vista.

**Ejemplo**:

CREATE OR REPLACE VIEW empvu10

(num\_empleado, nombre\_empleado, empleo)

AS

SELECT empno, ename, job

FROM emp

WHERE deptno = 10

WITH READ ONLY;

Cualquier intento de eliminar o modificar datos a través de esta vista fallará.

**Eliminación de una Vista (DROP VIEW):**

Para eliminar una vista, se usa la sentencia DROP VIEW. Esto elimina la definición de la vista, pero no afecta a las tablas base.

DROP VIEW empvu10;

**Vistas Inline:**

Las **vistas inline** son subconsultas con alias que se pueden usar directamente dentro de una consulta SQL. No son objetos permanentes en la base de datos.

**Ejemplo**:

SELECT a.ename, a.sal, a.deptno, b.maxsal

FROM emp a,

(SELECT deptno, max(sal) maxsal FROM emp GROUP BY deptno) b

WHERE a.deptno = b.deptno

AND a.sal < b.sal;

Aquí, la subconsulta b actúa como una vista inline para obtener el salario máximo por departamento.

Las vistas son una herramienta poderosa para organizar, proteger y simplificar el acceso a los datos, especialmente cuando se manejan grandes volúmenes de información o estructuras de datos complejas.