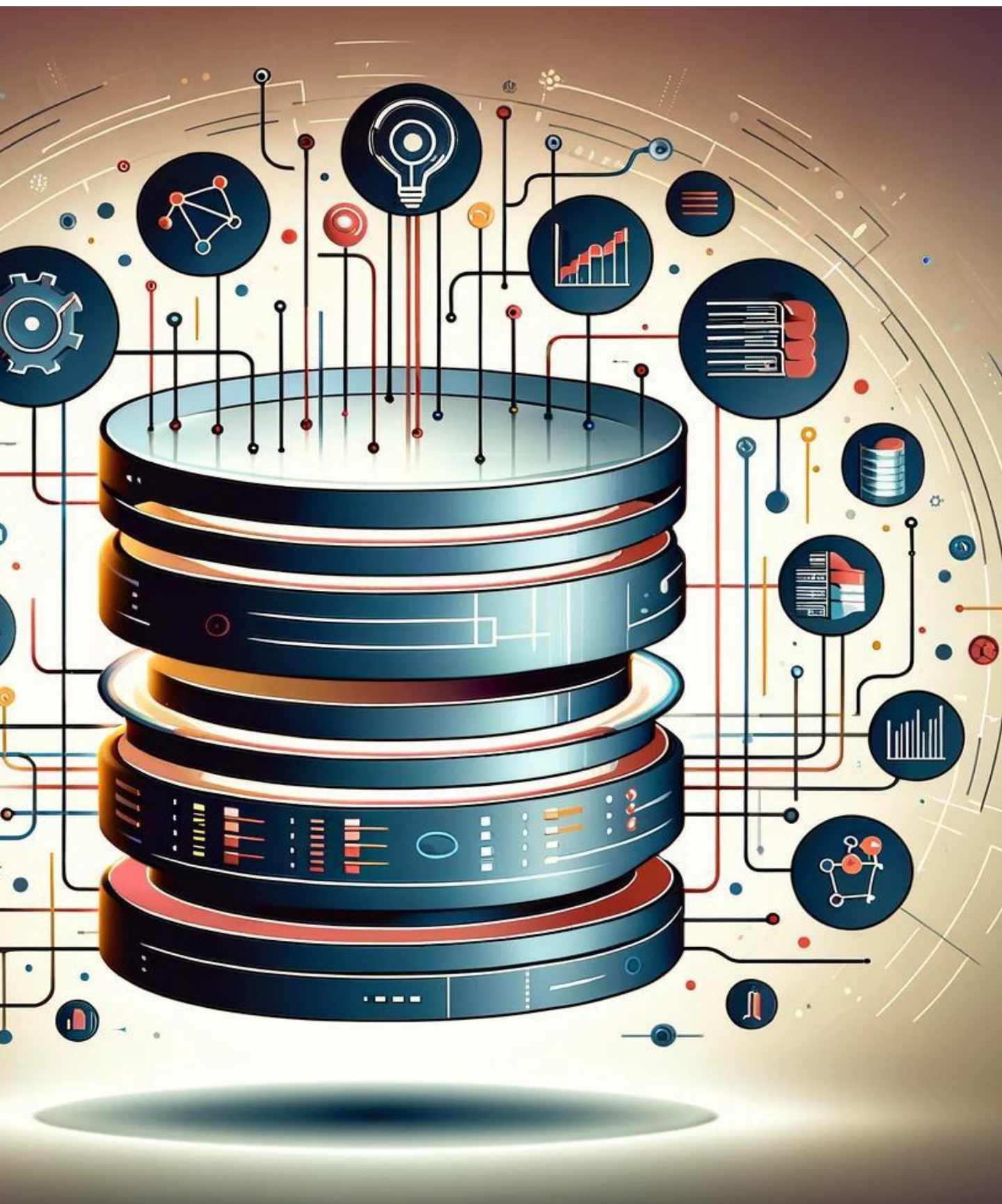




Bases de Datos 1



Alejandra Lliteras
Prof. Titular



Federico Orlando
Prof. Adjunto

TEMAS GENERALES

Bases de Datos 1

1

Modelo de
Datos

2

Teoría de
Diseño de
Bases de
Datos

3

Álgebra
Relacional

4

DBMS
Relacional
MySQL

5

Visualización
de Datos

TEMAS Y SUBTEMAS

Hoy veremos...

1.DBMS

2.Tipos de DBMS

3.Claves

4.Mysql

5.SQL

6.DDL

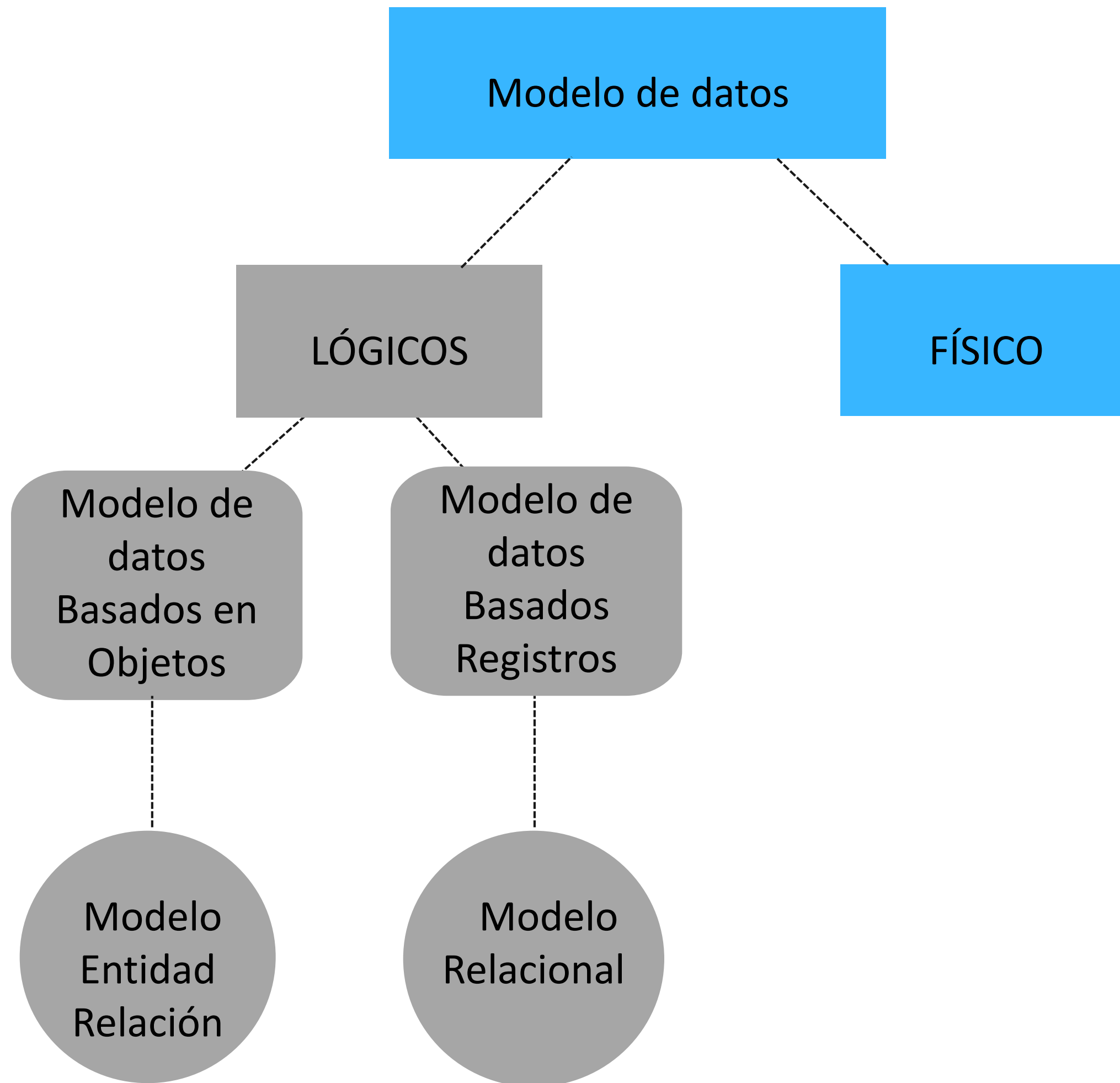
7.DCL

8.DML

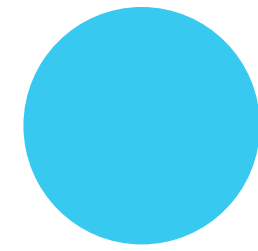
9.Vistas

10.Store Procedure

Tipos de Modelos de Datos



Sistema Gestor de Bases de Datos (DBMS)



Software compuesto por un conjunto de

● aplicaciones y

● herramientas

que permiten la:

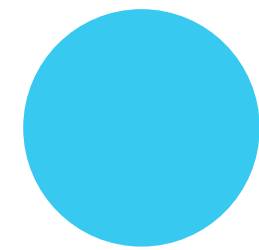
● creación,

● manipulación y

● administración

de bases de datos.

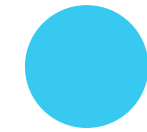
DBMS



Está compuesto por:



DDL (Data Definition Language)



DML (Data Manipulation Language)



DCL (Data Control Language)

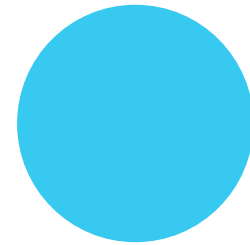


Data Dictionary

DBMS

DDL

(Data Definition Language)

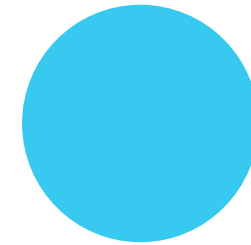


Permite definir las estructuras con la que se almacenarán los datos.

DBMS

DML

(Data Manipulation Language)



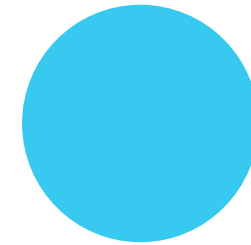
Lenguaje que permite escribir sentencias para:

- **Recuperar información**
- **Agregar información**
- **Eliminar información**
- **Modificar información**

DBMS

DCL

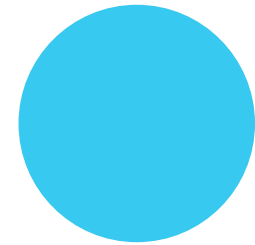
(Data Control Language)



Permite administrar los usuarios de la bases de datos y sus permisos.

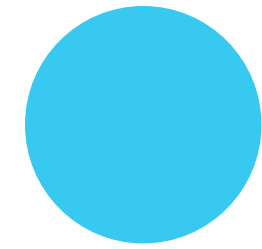
DBMS

Data Dictionary



Posee las definiciones de todas las variables de la base.

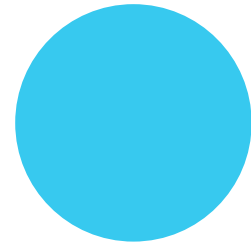
DBMS







Debe proporcionar:

- **Abstracción de la información**
- **Independencia**
- **Redundancia mínima**
- **Consistencia**
- **Seguridad**

DBMS

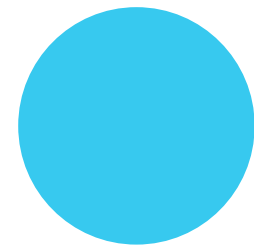


Debe proporcionar (cont):

-  **Integridad**
-  **Respaldo y recuperación**
-  **Control de la concurrencia**
-  **Tiempo de respuesta**

DBMS

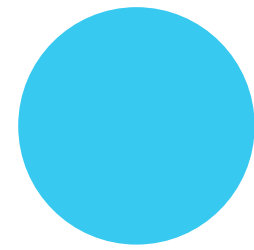
Abstracción de la información



los DBMS esconden a los usuarios detalles acerca del almacenamiento físico de los datos. Da lo mismo si una base de datos ocupa uno o cientos de archivos, este hecho se hace transparente al usuario.

DBMS

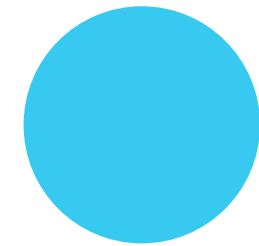
Independencia



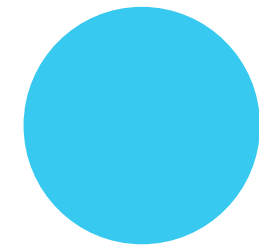
La independencia de los datos consiste en la capacidad de modificar el esquema (físico o lógico) de una base de datos sin tener que realizar cambios en las aplicaciones que se sirven de ella.

DBMS

Redundancia mínima



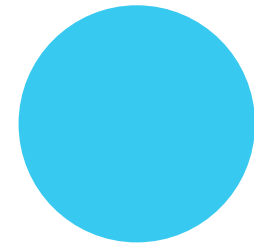
Un buen diseño de una base de datos logrará evitar la aparición de información repetida o redundante.



El objetivo debería ser lograr una redundancia nula; no obstante, en algunos casos la complejidad de los cálculos hace necesaria la aparición de redundancias.

DBMS

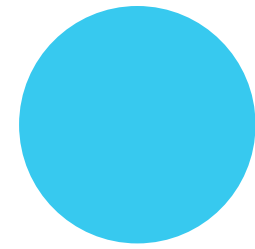
Consistencia



En aquellos casos en los que no se ha logrado esta redundancia nula, será necesario vigilar que aquella información que aparece repetida se actualice de forma coherente, es decir, que todos los datos repetidos se actualicen de forma simultánea.

DBMS

Seguridad

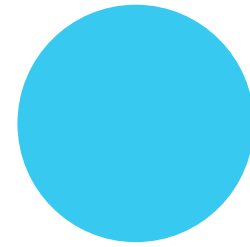


La información almacenada en una base de datos puede llegar a tener un gran valor. Los DBMS deben garantizar que esta información se encuentra asegurada frente a usuarios malintencionados, que intenten leer información privilegiada; frente a ataques que deseen manipular o destruir la información; o simplemente ante las torpezas de algún usuario autorizado pero despistado.

Normalmente, los DBMS disponen de un complejo sistema de permisos a usuarios y grupos de usuarios, que permiten otorgar diversas categorías de permisos.

DBMS

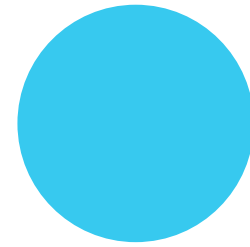
Integridad



Se trata de adoptar las medidas necesarias para garantizar la validez de los datos almacenados. Es decir, se trata de proteger los datos ante fallos de hardware, datos introducidos por usuarios descuidados, o cualquier otra circunstancia capaz de corromper la información almacenada.

DBMS

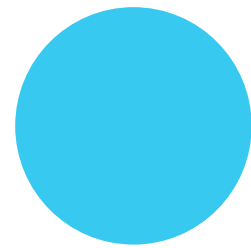
Respaldo y recuperación



Los DBMS deben proporcionar una forma eficiente de realizar copias de seguridad de la información almacenada en ellos, y de restaurar a partir de estas copias los datos que se hayan podido perder.

DBMS

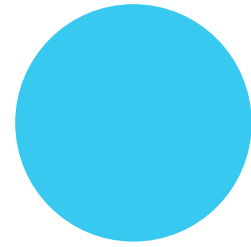
Control de la conurrencia



Habitualmente en la mayoría de los entornos son muchas las personas que acceden a una base de datos, ya sea para recuperar información o para almacenarla. Y es también frecuente que dichos accesos se realicen de forma simultánea. Así pues, un DBMS debe controlar este acceso concurrente a la información, que podría derivar en inconsistencias.

DBMS

Tiempo de respuesta

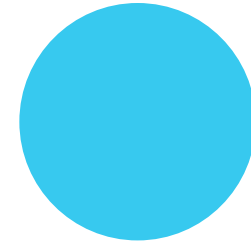


El DBMS debe proveer un acceso eficiente minimizando el tiempo que el DBMS tarda en darnos la información solicitada y en almacenar los cambios realizados en grandes volúmenes y por largos periodos de tiempo.

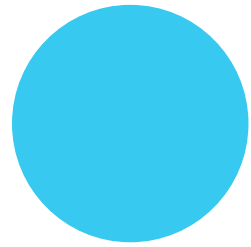
Tipos de DBMS



Tipos de DBMS



Relacional



NoSQL



Documentos



Clave - Valor

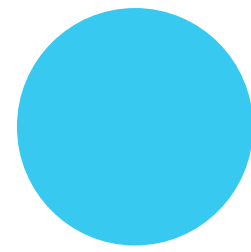


Grafos

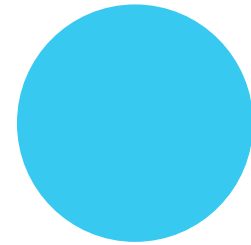


Columnas

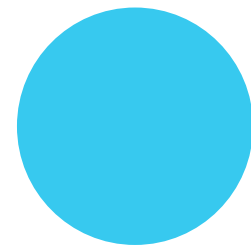
Bases de Datos Relacionales



Son las más antiguas y continúan siendo las más utilizadas hoy en día.



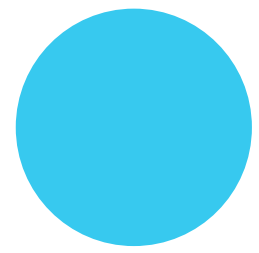
Se basan en el modelo relacional, por lo tanto los datos se almacenan en tablas o relaciones que están compuestas de filas o tuplas y columnas o campos.



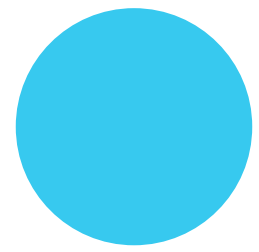
Su fundamento es la teoría de las bases de datos relaciones de Codd, la cual es matemática y establece un conjunto de reglas para la organización, almacenamiento y manipulación de datos. Esta basada en el álgebra relacional y la teoría de conjuntos.

Definición de Claves



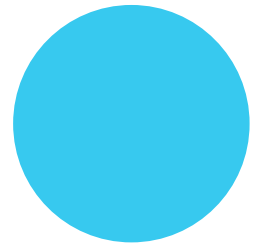


Primaria



Foránea

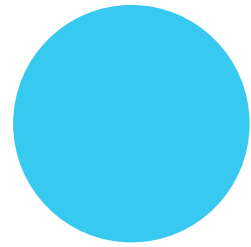
Claves



Es un conjunto de atributos de una tabla que cumple con unicidad y no nulidad.

Claves

Primaria



Es un conjunto de atributos de una tabla que establece una relación con otra tabla, usualmente coincide con la clave primaria de la tabla relacionada.

Estas se utilizan para mantener la integridad referencial que asegura la coherencia de los datos relacionados.

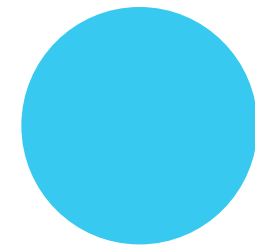
Claves

Foránea

MySQL



MySQL



Es un RDMS de código abierto disponible para múltiples SO.

Actualmente pertenece a ORACLE y existe una versión community como una enterprise.

Se puede operar por consola y a su vez existen distintos IDEs que dan soporte como Workbench o Dbeaver (multi-motor).

Implementa SQL como lenguaje de manipulación de datos.

Al instalar se crea el superusuario "root".

Motores de almacenamiento

- El motor de almacenamiento determina como se organizan, almacenan (formato) y manejan los datos en el sistema.
- MySQL tiene soporte para distintos motores de almacenamiento, como son InnoDB, MyISAM, Memory, entre otros.
- Para visualizar los distintos motores existe el comando **SHOW ENGINES**.

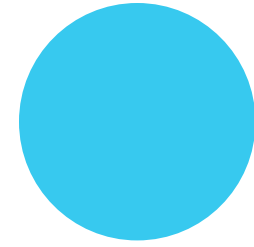
Motores de almacenamiento

- Actualmente InnoDB es el motor por defecto, que brinda soporte para transacciones que cumplen con las propiedades ACID, garantizando la integridad referencial y recuperación ante fallos.
- MyISAM era el motor por defecto en las antiguas versiones, tiene la contra de no ser transaccional y la ventaja de ser muy eficiente en búsquedas de texto.
- Memory mantiene los datos en memoria (HEAP).

SQL



SQL



SQL es un lenguaje diseñado especialmente para manipular y consultar bases de datos relacionales que se convirtió en un estándar.

Es un desarrollo de IBM lanzado en 1974, diseñado por Chamberlin y Boyce, cuyo último dialecto es el 2016.

Esta basado en el álgebra relacional y el cálculo relacional y esta compuesto por un lenguaje de definición de datos (DDL), un lenguaje de manipulación de datos (DML) y un lenguaje de control de datos (DCL).

Data Definition Language (DDL)



DDL

- El DDL permite definir la estructura y características de los elementos de las bases de datos, como son las tablas, columnas, claves primarias y foráneas o restricciones.

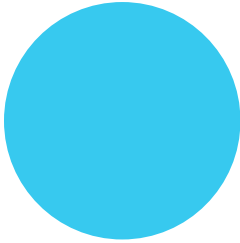
- Lo primero que permite es crear una base de datos

```
CREATE DATABASE bbdd1;
```

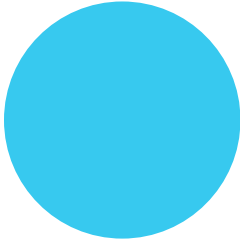
- Para usar una base de datos creada se la debe poner en uso

```
USE bbdd1;
```


DDL



Las bases de datos en MySQL se implementan como directorios que contienen archivos, los cuales se corresponden con las tablas de la base de datos. Como no hay tablas en la base de datos cuando se crean inicialmente, el comando `CREATE DATABASE` en MySQL crea sólo un directorio bajo el directorio de datos de MySQL y el archivo `db.opt`.



Si crea manualmente un directorio bajo el directorio de datos (por ejemplo, con `mkdir`), el servidor lo considera como un directorio de base de datos y muestra la salida de `SHOW DATABASES`.



`Create database` y `create schema` son sinónimos.

DDL

Tipos de datos

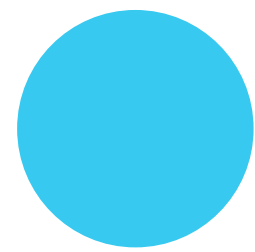
Numéricos:
BIT, TINYINT, SMALLINT,
MEDIUMINT, INT, INTEGER,
BIGINT, REAL, DOUBLE, FLOAT,
DECIMAL, NUMERIC

Fechas:
DATE, TIME, TIMESTAMP,
DATETIME

Texto:
CHAR, VARCHAR, TINYTEXT,
TEXT, MEDIUMTEXT, LONGTEXT

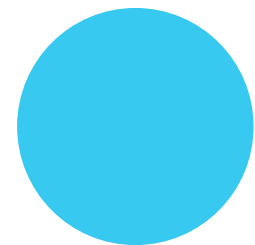
Binarios
TINYBLOB, BLOB,
MEDIUMBLOB, LONGBLOB

Especiales:
ENUM



Para crear tablas existe la sentencia: **CREATE TABLE**, en la cual se define su estructura definiendo cada campo/atributo con su tipo de dato y restricciones.

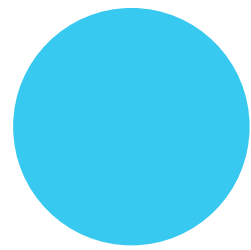
```
CREATE TABLE bbdd1.tabla_ejemplo (  
id_tabla_ejemplo integer auto_increment primary key,  
descripcion varchar(50) not null  
);
```



Una restricción es una regla que se aplica a los valores que se insertan o actualizan en el campo de una tabla para garantizar la integridad.

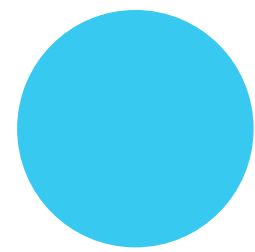
DDL

Create



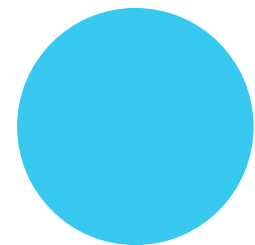
Para crear tablas existe la sentencia: **CREATE TABLE**, en la cual se define su estructura definiendo cada campo/atributo con su tipo de dato y restricciones.

```
CREATE TABLE bbdd1.tabla_ejemplo (  
id_tabla_ejemplo integer auto_increment primary key,  
descripcion varchar(50) not null  
);
```



Para crear tablas existe la sentencia: **CREATE TABLE**, en la cual se define su estructura definiendo cada campo/atributo con su tipo de dato y restricciones.

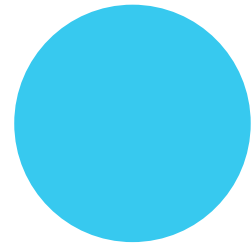
```
CREATE TABLE bbdd1.tabla_ejemplo (  
  id_tabla_ejemplo integer auto_increment primary key,  
  descripcion varchar(50) not null  
);
```



Una restricción es una regla que se aplica a los valores que se insertan o actualizan en el campo de una tabla para garantizar la integridad.

DDL

Create

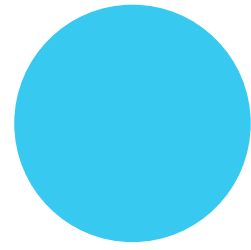


Para modificar una tabla existe la sentencia **ALTER TABLE**.

```
ALTER TABLE bbdd1.tabla_ejemplo add (  
precio float not null  
);
```

DDL

Alter



Para modificar una tabla existe la sentencia **ALTER TABLE**.

```
ALTER TABLE bbdd1.tabla_ejemplo add (  
precio float not null  
);
```

DML

Para eliminar una tabla tenemos la sentencia DROP.

```
DROP TABLE bbdd1.tabla_ejemplo;
```

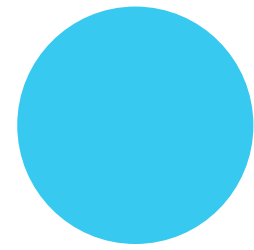
- Todos los datos se eliminan
- La estructura de la tabla se elimina (la información registrada en el diccionario de datos)
- Todas las transacciones pendientes sobre la tabla se commitean
- Todos los índices de la tabla se eliminan
- Esta operación no se puede deshacer

Sólo el creador de la tabla o el usuario root puede ejecutar la sentencia con éxito.

- **Primary Key**
- **Foreing Key**
- **Not Null**
- **Unique**
- **Default**
- **Check**
- **Autoincrement**

DDL

Restricciones

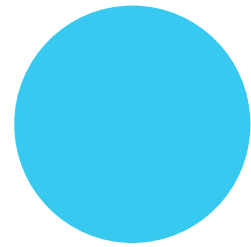


Identifica el campo/atributo que será el identificador de la tabla. Solo puede haber una por tabla.

DDL

Restricciones

Primary Key



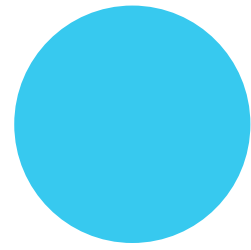
Referencia cruzadas entre tablas para mantener la consistencia.

```
CREATE TABLE bbdd1.otra_tabla (  
  id_otra_tabla integer auto_increment primary key,  
  ejemplo integer not null,  
  constraint fk_otra_tabla_tabla_ejemplo foreign key (ejemplo)  
  references tabla_ejemplo(id_tabla_ejemplo)  
);
```

DDL

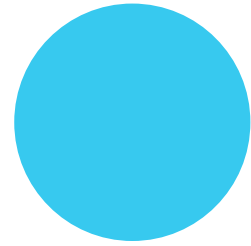
Restricciones

Foreing Key



Referencia cruzadas entre tablas para mantener la consistencia.

```
CREATE TABLE bbdd1.otra_tabla (  
  id_otra_tabla integer auto_increment primary key,  
  ejemplo integer not null,  
  constraint fk_otra_tabla_tabla_ejemplo foreign key (ejemplo)  
  references tabla_ejemplo(id_tabla_ejemplo)  
);
```

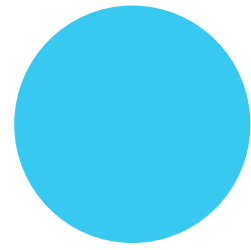


Asegura que el campo/atributo no contenga valores nulos.

DDL

Restricciones

Not Null

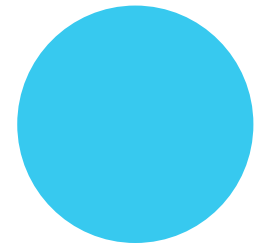


Garantiza que los valores del campo/atributo sean únicos, evita los duplicados.

DDL

Restricciones

Unique

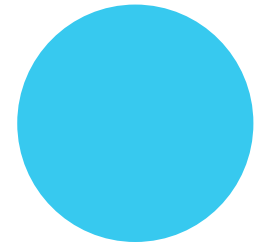


Establece un valor constante por defecto para ese campo/atributo.

DDL

Restricciones

Default

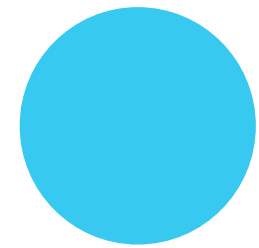


Especifica una condición que los valores del campo/atributo deben cumplir.

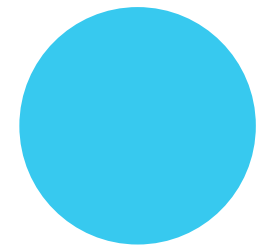
DDL

Restricciones

Check



Asociado a un campo/atributo de tipo entero, al no enviar valor asigna el valor máximo + 1 que posea dicho campo en la tabla, comenzando por 1.



Solo puede haber una columna con esta restricción, por lo general la clave primaria.

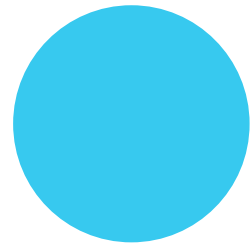
DDL

Restricciones

Autoincrement

Data Control Lenguaje (DCL)





El DCL permite manipular los usuarios de la base de datos y los permisos de cada uno de ellos.

DCL



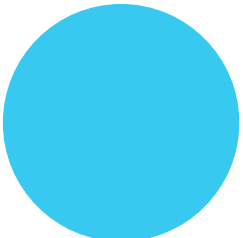
Para crear usuarios tenemos la sentencia CREATE USER.

```
CREATE USER 'fede'@'localhost' IDENTIFIED WITH CACHING_SHA2_PASSWORD BY 'BBdd1';
```

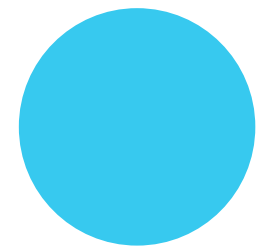


Para eliminar usuarios tenemos la sentencia DROP USER.

```
DROP USER 'fede'@'localhost';
```

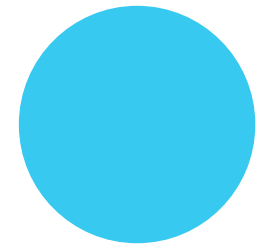


Importante: Un usuario esta formado por un nombre de usuario y el host desde el que va a establecer la conexión. (@localhost en el ej.) Un mismo nombre de usuario con distinto host son considerados distintos usuarios.



Para crear usuarios tenemos la sentencia **CREATE USER**.

```
CREATE USER 'fede'@'localhost' IDENTIFIED WITH CACHING_SHA2_PASSWORD BY 'BBdd1';
```



Para eliminar usuarios tenemos la sentencia **DROP USER**.

```
DROP USER 'fede'@'localhost';
```

DCL

Para crear usuarios tenemos la sentencia CREATE USER.

```
CREATE USER 'fede'@'localhost' IDENTIFIED WITH CACHING_SHA2_PASSWORD BY 'BBdd1';
```

Para eliminar usuarios tenemos la sentencia DROP USER.

```
DROP USER 'fede'@'localhost';
```

Importante: Un usuario esta formado por un nombre de usuario y el host desde el que va a establecer la conexión. (@localhost en el ej.) Un mismo nombre de usuario con distinto host son considerados distintos usuarios.

DCL

● Para otorgar permisos a un usuario tenemos la sentencia **GRANT**.

```
GRANT ALL ON bbdd1.* TO 'fede'@'localhost';
```

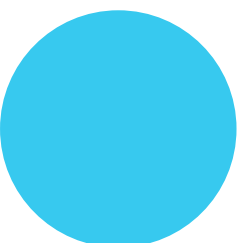
● Para quitar permisos a un usuario tenemos la sentencia **REVOKE**.

```
REVOKE SELECT ON bbdd1.tabla_ejemplo FROM 'fede'@'localhost';
```



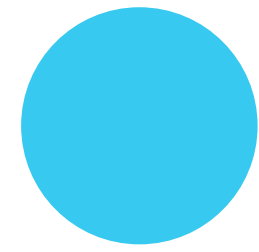
Para otorgar permisos a un usuarios tenemos la sentencia GRANT.

```
GRANT ALL ON bbdd1.* TO 'fede'@'localhost';
```

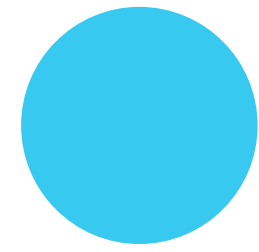


Para quitar permisos a un usuarios tenemos la sentencia REVOKE.

```
REVOKE SELECT ON bbdd1.tabla_ejemplo FROM 'fede'@'localhost';
```

Etapa 1: El servidor comprueba si debe permitirle conectarse.



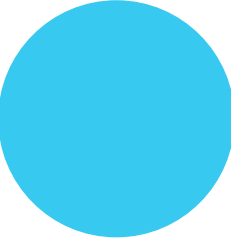
Etapa 2: Asumiendo que se conecta, el servidor comprueba cada comando que ejecuta para ver si tiene suficientes permisos para hacerlo. Por ejemplo, si intenta seleccionar registros de una tabla en una base de datos o eliminar una tabla de la base de datos, el servidor verifica que tenga el permiso SELECT para la tabla o el permiso DROP para la base de datos.

DCL

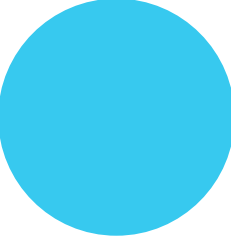
Control de acceso

DCL

Control de acceso



Si sus permisos cambian mientras está conectado, estos cambios no tienen porqué tener efecto inmediatamente para el siguiente comando que ejecute.



El servidor guarda información de privilegios en las tablas de permisos de la base de datos mysql. El servidor MySQL lee el contenido de dichas tablas en memoria cuando arranca y las vuelve a leer bajo ciertas circunstancias.

- **Las decisiones acerca de control de acceso se basan en las copias en memoria de las tablas de permisos.**
- **Puede decirle que las vuelva a leer mediante el comando `FLUSH PRIVILEGES` o ejecutando los comandos `mysqladmin flush-privileges` o `mysqladmin reload` .**

DCL

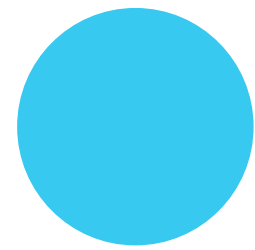
Control de acceso

Descanso



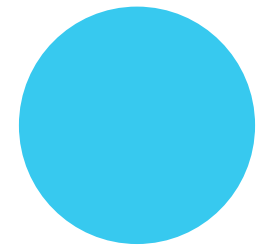
Data Manipulation Languaje (DML)





El DML permite manipular los datos de la base de datos con operaciones como insertar, eliminar, actualizar y seleccionar registros.

DML



Para insertar datos en una tabla tenemos la sentencia INSERT.

```
INSERT INTO bbdd1.tabla_ejemplo (descripcion,precio) VALUES ("prueba bbdd1", 50.95);
```

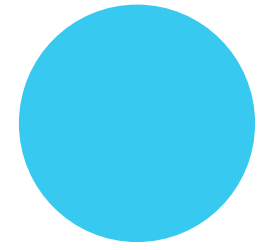
- **Si no se detallan las columnas, la lista de valores debe estar en el mismo orden de las columnas de la tabla**

- **Si se detallan las columnas, pueden ser todas las columnas de la tabla o sólo las que son requeridas. En ambos casos debe existir una concordancia entre columna1 y valor1, y así sucesivamente**

- **Los valores carácter y fecha deben encerrarse entre comillas simples.**

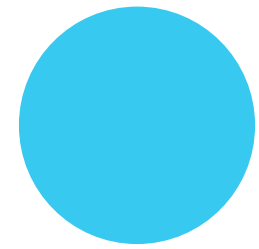
DML

Insert



Para insertar datos en una tabla tenemos la sentencia INSERT.

```
INSERT INTO bbdd1.tabla_ejemplo (descripcion,precio) VALUES ("prueba bbdd1", 50.95);
```

Para insertar datos en una tabla tenemos la sentencia INSERT.

```
INSERT INTO bbdd1.tabla_ejemplo (descripcion,precio) VALUES ("prueba bbdd1", 50.95);
```

- **Si no se detallan las columnas, la lista de valores debe estar en el mismo orden de las columnas de la tabla**

- **Si se detallan las columnas, pueden ser todas las columnas de la tabla o sólo las que son requeridas. En ambos casos debe existir una concordancia entre columna1 y valor1, y así sucesivamente**

- **Los valores carácter y fecha deben encerrarse entre comillas simples.**

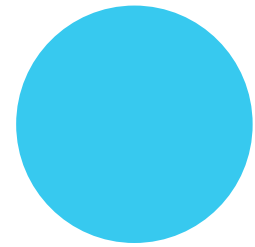
DML

Insert

DML

Errores comunes al insertar

- **Olvidar colocar un valor a una columna que es obligatoria (NOT NULL)**
- **Repetir un valor que no permite valores duplicados**
- **Colocar un valor que depende de otra tabla y en ella no exista (No encontrar una correspondencia con una clave foránea)**
- **Colocar un valor que no cumple con una restricción de entidad (CHECK)**
- **Colocar un valor que no corresponde con el tipo de dato de la columna**
- **Colocar un valor más grande del que se puede almacenar en la columna**



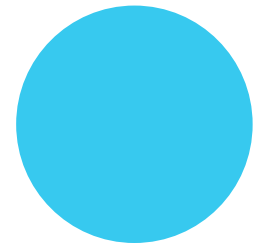
Para actualizar datos en una tabla tenemos la sentencia UPDATE.

```
UPDATE bbdd1.tabla_ejemplo SET precio=99.99 WHERE descripcion="prueba bbdd1";
```

- **Con una sola sentencia se pueden modificar 0, 1 o más filas de la tabla, esto depende de la condición que se establezca en la cláusula WHERE.**
- **Cuidado! si no se coloca ninguna condición se modifican todas las filas de la tabla**
- **Antes de aplicar la sentencia UPDATE, se recomienda usar la sentencia SELECT con la condición que involucra las filas que se van a modificar.**

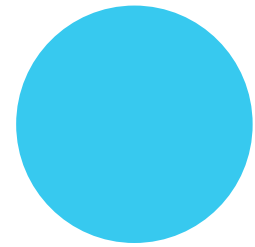
DML

Update



Para actualizar datos en una tabla tenemos la sentencia UPDATE.

```
UPDATE bbdd1.tabla_ejemplo SET precio=99.99 WHERE descripcion="prueba bbdd1";
```



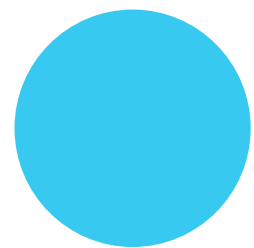
Para actualizar datos en una tabla tenemos la sentencia UPDATE.

```
UPDATE bbdd1.tabla_ejemplo SET precio=99.99 WHERE descripcion="prueba bbdd1";
```

- **Con una sola sentencia se pueden modificar 0, 1 o más filas de la tabla, esto depende de la condición que se establezca en la cláusula WHERE.**
- **Cuidado! si no se coloca ninguna condición se modifican todas las filas de la tabla**
- **Antes de aplicar la sentencia UPDATE, se recomienda usar la sentencia SELECT con la condición que involucra las filas que se van a modificar.**

DML

Update



Para eliminar datos en una tabla tenemos la sentencia DELETE.

```
DELETE FROM bbdd1.tabla_ejemplo WHERE id_tabla_ejemplo=1;
```

● Con una sola sentencia se pueden eliminar 0, 1 o más filas de la tabla, esto depende de la condición que se establezca en la cláusula WHERE.

- Si ninguna fila coincide con la condición, se recibe el mensaje “0 filas borradas”
- Cuidado: si no se coloca ninguna condición se eliminan todas las filas de la tabla
- Antes de aplicar la sentencia DELETE, se recomienda usar la sentencia SELECT con la condición que involucra las filas que se van a eliminar

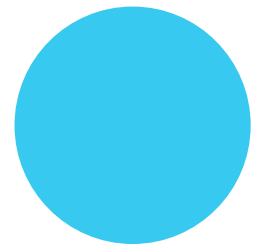
DML

Delete



Para eliminar datos en una tabla tenemos la sentencia DELETE.

```
DELETE FROM bbdd1.tabla_ejemplo WHERE id_tabla_ejemplo=1;
```



Para eliminar datos en una tabla tenemos la sentencia DELETE.

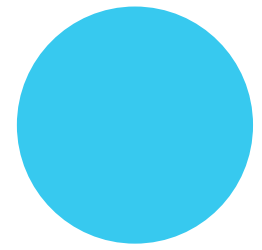
```
DELETE FROM bbdd1.tabla_ejemplo WHERE id_tabla_ejemplo=1;
```

- Con una sola sentencia se pueden eliminar 0, 1 o más filas de la tabla, esto depende de la condición que se establezca en la cláusula WHERE.

- Si ninguna fila coincide con la condición, se recibe el mensaje “0 filas borradas”
- Cuidado: si no se coloca ninguna condición se eliminan todas las filas de la tabla
- Antes de aplicar la sentencia DELETE, se recomienda usar la sentencia SELECT con la condición que involucra las filas que se van a eliminar

DML

Delete

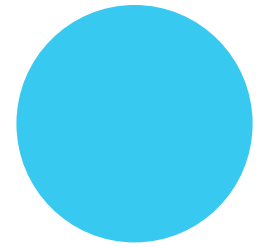


Para eliminar todos los datos en una tabla tenemos la sentencia **TRUNCATE**.

```
TRUNCATE TABLE bbdd1.tabla_ejemplo ;
```

DML

Truncate



Para recuperar datos en una tabla tenemos la sentencia **SELECT**.

```
SELECT precio FROM bbdd1.tabla_ejemplo te WHERE descripcion = "prueba bbdd1";
```

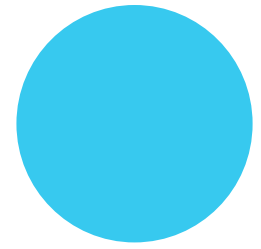
posee tres partes básicas

- **SELECT** donde seleccionamos el listado de columnas que queremos obtener como resultado.
- **FROM** donde se indica las tablas donde obtener los datos.
- **WHERE** (opcional) especifica la condición de los datos a devolver.

En el **SELECT** se puede utilizar el ***** para retornar todos los valores, **DISTINCT** para que no retorne valores repetidos.

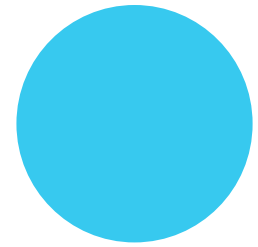
DML

Select



Para recuperar datos en una tabla tenemos la sentencia **SELECT**.

```
SELECT precio FROM bbdd1.tabla_ejemplo WHERE descripcion = "prueba bbdd1";
```



Para recuperar datos en una tabla tenemos la sentencia **SELECT**.

```
SELECT precio FROM bbdd1.tabla_ejemplo te WHERE descripcion = "prueba bbdd1";
```

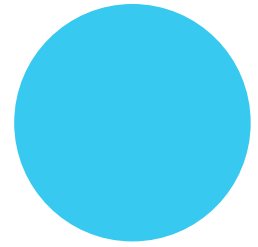
posee tres partes básicas

- **SELECT** donde seleccionamos el listado de columnas que queremos obtener como resultado.
- **FROM** donde se indica las tablas donde obtener los datos.
- **WHERE** (opcional) especifica la condición de los datos a devolver.

En el **SELECT** se puede utilizar el ***** para retornar todos los valores, **DISTINCT** para que no retorne valores repetidos.

DML

Select

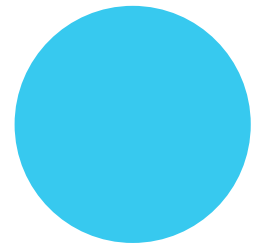


Operadores de comparación:

- = Igual a
- > Mayor que
- >= Mayor o igual que
- < Menor que
- <= Menor o igual que
- <> Diferente a (!=) (^=)
- BETWEEN ... AND ... Entre dos valores los cuales están incluidos en el rango.
- IN (Lista) En la lista de valores dados

DML

**Select
where**



Operadores de comparación (cont):

- **IS NULL** Es un valor nulo
- **LIKE** Concuerda con un patrón

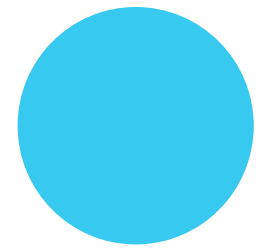
La condición de búsqueda puede contener caracteres, números y/o comodines:

- %** para representar cero o más caracteres
- _** para representar un carácter

Se pueden combinar caracteres y comodines.

DML

**Select
where**

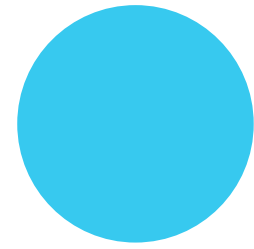


Operadores lógicos:

- **AND**
- **OR**
- **NOT.**

DML

**Select
where**



Para obtener datos del cruce de 2 tablas existe la sentencia JOIN, que presenta varias opciones:

- JOIN
- INNER JOIN
- LEFT JOIN
- RIGHT JOIN

DML

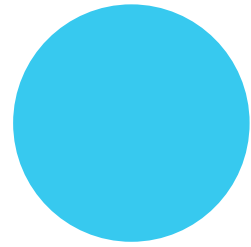
JOINS


```
SELECT *
FROM bbdd1.tabla_ejemplo te;
```

id_tabla_ejemplo	descripcion	precio
3	ejemplo 1	9.99
4	ejemplo 2	19.99
5	ejemplo 3	29.99
6	ejemplo 4	39.99

```
CREATE TABLE bbdd1.tabla_para_join (
id_tabla_para_join integer auto_increment primary key,
id_ejemplo integer not null,
descripcion varchar(100) not null,
constraint fk_tabla_para_join_tabla_ejemplo foreign key (id_ejemplo)
references bbdd1.tabla_ejemplo(id_tabla_ejemplo)
);
```

id_tabla_para_join	id_ejemplo	descripcion
2	3	join 1
3	6	join 2
4	6	join 3



JOIN: realizará un producto cartesiano entre las 2 tablas.

También se lo puede escribir como una "," (coma) entre las 2 tablas en el FROM.

```
SELECT *  
FROM bbdd1.tabla_para_join tpj JOIN bbdd1.tabla_ejemplo te;
```

```
SELECT *  
FROM bbdd1.tabla_para_join tpj, bbdd1.tabla_ejemplo te;
```

id_tabla_para_join	id_ejemplo	descripcion	id_tabla_ejemplo	descripcion	precio
4	6	join 3	3	ejemplo 1	9.99
3	6	join 2	3	ejemplo 1	9.99
2	3	join 1	3	ejemplo 1	9.99
4	6	join 3	4	ejemplo 2	19.99
3	6	join 2	4	ejemplo 2	19.99
2	3	join 1	4	ejemplo 2	19.99
4	6	join 3	5	ejemplo 3	29.99
3	6	join 2	5	ejemplo 3	29.99
2	3	join 1	5	ejemplo 3	29.99
4	6	join 3	6	ejemplo 4	39.99
3	6	join 2	6	ejemplo 4	39.99
2	3	join 1	6	ejemplo 4	39.99

DML

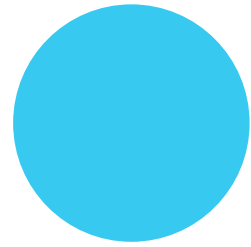
JOINS

JOIN (CARTESIANO)

```
SELECT *
FROM bbdd1.tabla_para_join tpj JOIN bbdd1.tabla_ejemplo te;
```

```
SELECT *
FROM bbdd1.tabla_para_join tpj, bbdd1.tabla_ejemplo te;
```

id_tabla_para_join	id_ejemplo	descripcion	id_tabla_ejemplo	descripcion	precio
4	6	join 3	3	ejemplo 1	9.99
3	6	join 2	3	ejemplo 1	9.99
2	3	join 1	3	ejemplo 1	9.99
4	6	join 3	4	ejemplo 2	19.99
3	6	join 2	4	ejemplo 2	19.99
2	3	join 1	4	ejemplo 2	19.99
4	6	join 3	5	ejemplo 3	29.99
3	6	join 2	5	ejemplo 3	29.99
2	3	join 1	5	ejemplo 3	29.99
4	6	join 3	6	ejemplo 4	39.99
3	6	join 2	6	ejemplo 4	39.99
2	3	join 1	6	ejemplo 4	39.99



INNER JOIN: realizará un producto natural entre los campos indicados.

También se lo puede escribir como una "," (coma) entre las 2 tablas en el FROM colocando la condición en el WHERE.

```
SELECT *  
FROM bbdd1.tabla_para_join tpj INNER JOIN bbdd1.tabla_ejemplo te  
ON tpj.id_ejemplo = te.id_tabla_ejemplo;
```

```
SELECT *  
FROM bbdd1.tabla_para_join tpj, bbdd1.tabla_ejemplo te  
WHERE tpj.id_ejemplo = te.id_tabla_ejemplo;
```

id_tabla_para_join	id_ejemplo	descripcion	id_tabla_ejemplo	descripcion	precio
2	3	join 1	3	ejemplo 1	9.99
3	6	join 2	6	ejemplo 4	39.99
4	6	join 3	6	ejemplo 4	39.99

DML

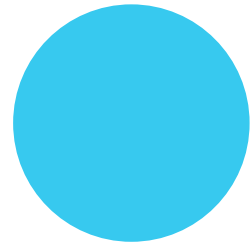
JOINS

INNER JOIN (NATURAL)

```
SELECT *
FROM bbdd1.tabla_para_join tpj INNER JOIN bbdd1.tabla_ejemplo te
ON tpj.id_ejemplo = te.id_tabla_ejemplo;
```

```
SELECT *
FROM bbdd1.tabla_para_join tpj, bbdd1.tabla_ejemplo te
WHERE tpj.id_ejemplo = te.id_tabla_ejemplo;
```

id_tabla_para_join	id_ejemplo	descripcion	id_tabla_ejemplo	descripcion	precio
2	3	join 1	3	ejemplo 1	9.99
3	6	join 2	6	ejemplo 4	39.99
4	6	join 3	6	ejemplo 4	39.99



LEFT JOIN: luego de calcular el **INNER JOIN** se agregan las tuplas de la relación de la izquierda que no tienen relación con tuplas de la relación de la derecha en el caso de existir.

```
SELECT *  
FROM bdd1.tabla_ejemplo te LEFT JOIN bdd1.tabla_para_join tpj  
ON tpj.id_ejemplo = te.id_tabla_ejemplo;
```

id_tabla_ejemplo	descripcion	precio	id_tabla_para_join	id_ejemplo	descripcion
3	ejemplo 1	9.99	2	3	join 1
4	ejemplo 2	19.99			
5	ejemplo 3	29.99			
6	ejemplo 4	39.99	4	6	join 3
6	ejemplo 4	39.99	3	6	join 2

DML

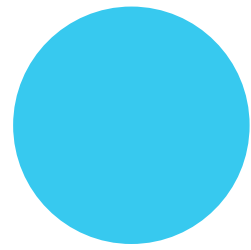
JOINS
LEFT JOIN

```

SELECT *
FROM  bbdd1.tabla_ejemplo te LEFT JOIN  bbdd1.tabla_para_join tpj
ON tpj.id_ejemplo = te.id_tabla_ejemplo;

```

id_tabla_ejemplo	descripcion	precio	id_tabla_para_join	id_ejemplo	descripcion
3	ejemplo 1	9.99	2	3	join 1
4	ejemplo 2	19.99			
5	ejemplo 3	29.99			
6	ejemplo 4	39.99	4	6	join 3
6	ejemplo 4	39.99	3	6	join 2



RIGHT JOIN: luego de calcular el INNER JOIN se agregan las tuplas de la relación de la derecha que no tienen relación con tuplas de la relación de la izquierda en el caso de existir.

```
SELECT *  
FROM bbdd1.tabla_para_join tpj RIGHT JOIN bbdd1.tabla_ejemplo te  
ON tpj.id_ejemplo = te.id_tabla_ejemplo;
```

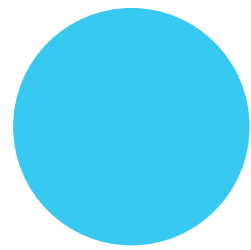
id_tabla_para_join	id_ejemplo	descripcion	id_tabla_ejemplo	descripcion	precio
2	3	join 1	3	ejemplo 1	9.99
			4	ejemplo 2	19.99
			5	ejemplo 3	29.99
4	6	join 3	6	ejemplo 4	39.99
3	6	join 2	6	ejemplo 4	39.99

DML

JOINS RIGHT JOIN


```
SELECT *
FROM bbdd1.tabla_para_join tpj RIGHT JOIN bbdd1.tabla_ejemplo te
ON tpj.id_ejemplo = te.id_tabla_ejemplo;
```

id_tabla_para_join	id_ejemplo	descripcion	id_tabla_ejemplo	descripcion	precio
2	3	join 1	3	ejemplo 1	9.99
			4	ejemplo 2	19.99
			5	ejemplo 3	29.99
4	6	join 3	6	ejemplo 4	39.99
3	6	join 2	6	ejemplo 4	39.99



Para ordenar los resultados de una consulta, puede ser por uno o más campos en orden ascendente (ASC) o descendente.(DESC)

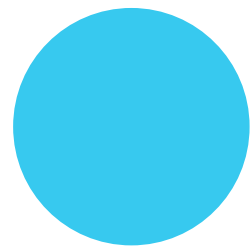
```
SELECT te.id_tabla_ejemplo, te.descripcion, te.precio
FROM bbdd1.tabla_ejemplo te
ORDER BY te.precio DESC;
```

Debe ser la última cláusula de la sentencia SELECT.

id_tabla_ejemplo	descripcion	precio
6	ejemplo 4	39.99
5	ejemplo 3	29.99
4	ejemplo 2	19.99
3	ejemplo 1	9.99

DML

ORDER BY

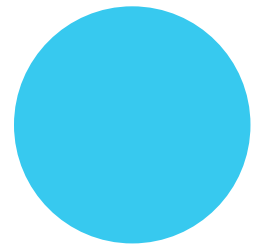


Para ordenar los resultados de una consulta, puede ser por uno o más campos en orden ascendente (ASC) o descendente.(DESC)

```
SELECT te.id_tabla_ejemplo, te.descripcion, te.precio
FROM bbdd1.tabla_ejemplo te
ORDER BY te.precio DESC;
```

Debe ser la última cláusula de la sentencia SELECT.

id_tabla_ejemplo	descripcion	precio
6	ejemplo 4	39.99
5	ejemplo 3	29.99
4	ejemplo 2	19.99
3	ejemplo 1	9.99



Se utiliza para agrupar los resultados de una consulta bajo un criterio, puede ser por uno o más campos.

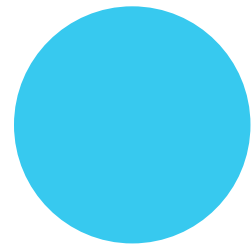
Permite el uso de funciones de agregación.

```
SELECT tpj.id_ejemplo, count(*)  
FROM bbdd1.tabla_para_join tpj  
GROUP BY tpj.id_ejemplo;
```

id_ejemplo	count(*)
3	1
6	2

DML

GROUP BY



Se utiliza para agrupar los resultados de una consulta bajo un criterio, puede ser por uno o más campos.

Permite el uso de funciones de agregación.

```
SELECT tpj.id_ejemplo, count(*)  
FROM bbdd1.tabla_para_join tpj  
GROUP BY tpj.id_ejemplo;
```

id_ejemplo	count (*)
3	1
6	2

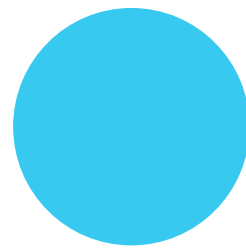


Errores comunes con GROUP BY

- Columnas en la cláusula SELECT deben aparecer en la cláusula GROUP BY (no a la inversa)
- La cláusula WHERE no se usa para excluir grupos, ese trabajo lo hace la cláusula HAVING
- No se pueden usar funciones de agregación en la cláusula WHERE

DML

GROUP BY



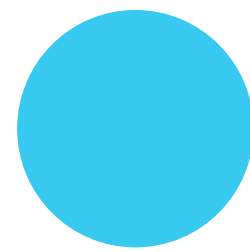
Se utiliza para filtrar el resultado de una consulta por los grupos realizados utilizando la cláusula GROUP BY.

```
SELECT tpj.id_ejemplo, count(*)  
FROM bdd1.tabla_para_join tpj  
GROUP BY tpj.id_ejemplo  
HAVING COUNT(id_ejemplo) > 1 ;
```

id_ejemplo	count(*)
6	2

DML

HAVING



Se utiliza para filtrar el resultado de una consulta por los grupos realizados utilizando la cláusula GROUP BY.

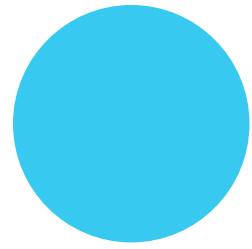
```
SELECT tpj.id_ejemplo, count(*)  
FROM bbdd1.tabla_para_join tpj  
GROUP BY tpj.id_ejemplo  
HAVING COUNT(id_ejemplo) > 1 ;
```

id_ejemplo	count (*)
6	2

DML

Funciones de agregación

- **Promedio (avg):** se aplica sobre atributos numéricos. Retorna el promedio de la cuenta.
- **Mínimo (min):** retorna el elemento mas chico dentro de las tuplas para ese atributo.
- **Máximo (max)** retorna el elemento mas grande dentro de las tuplas para ese atributo.
- **Total (sum):** se aplica sobre atributos numéricos. Realiza la suma matemática.
- **Cuenta (count):** cuenta la cantidad de tuplas resultantes.

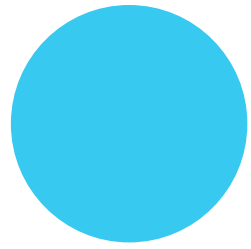


```
SELECT[Columna,] FuncionDeGrupo  
(Expresión), ...  
FROM Tabla  
[WHERE Condición]  
[GROUP BY Columna]  
[HAVING CondiciónDeGrupo]  
[ORDER BY Columna];
```

1. Se toma la fuente de datos con la cláusula **FROM**
2. Se excluyen las filas que no cumplen con la condición de la cláusula **WHERE**
3. Se arman los grupos de acuerdo a las columnas de la cláusula **GROUP BY**
4. Se aplican las Funciones de Grupo que están en la cláusula **SELECT** a los grupos previamente formados
5. Se excluyen los grupos que no cumplen con la condición de la cláusula **HAVING**
6. Se ordenan los resultados conforme se establece en la cláusula **ORDER BY**

DML

Orden de ejecución

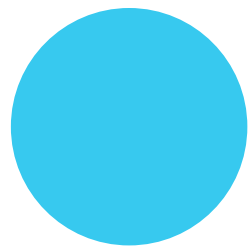


Una subconsulta es una consulta completa que aparece en la cláusula FROM, WHERE o HAVING de una sentencia SQL.

```
SELECT sum(te.precio)
FROM bbdd1.tabla_ejemplo te
WHERE EXISTS (SELECT ot.ejemplo, COUNT(id_otra_tabla)
              FROM bbdd1.otra_tabla ot
              GROUP BY ot.ejemplo
              HAVING COUNT(id_otra_tabla) > 2);
```

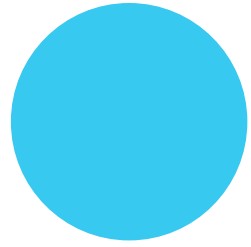
DML

Subconsultas



Una subconsulta es una consulta completa que aparece en la cláusula FROM, WHERE o HAVING de una sentencia SQL.

```
SELECT sum(te.precio)
FROM bbdd1.tabla_ejemplo te
WHERE EXISTS (SELECT ot.ejemplo, COUNT(id_otra_tabla)
              FROM bbdd1.otra_tabla ot
              GROUP BY ot.ejemplo
              HAVING COUNT(id_otra_tabla) > 2);
```



Una subconsulta es una consulta completa que aparece en la cláusula FROM, WHERE o HAVING de una sentencia SQL.

```
SELECT sum(te.precio)
FROM bbdd1.tabla_ejemplo te
WHERE EXISTS (SELECT ot.ejemplo, COUNT(id_otra_tabla)
              FROM bbdd1.otra_tabla ot
              GROUP BY ot.ejemplo
              HAVING COUNT(id_otra_tabla) > 2);
```

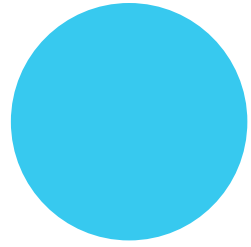
En el caso de estar presente en el WHERE o HAVING se utiliza con un conector como "=", IN, NOT IN, EXISTS, NOT EXISTS.

La subconsulta (consulta interna o anidada) se ejecuta antes de la consulta principal y su resultado se usa para ejecutar la consulta principal.

Las subconsultas se ubican en el lado derecho de la condición de comparación.

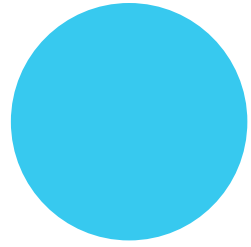
DML

Subconsultas



Una subconsulta es una consulta completa que aparece en la cláusula FROM, WHERE o HAVING de una sentencia SQL.

```
SELECT sum(te.precio)
FROM bbdd1.tabla_ejemplo te
WHERE EXISTS (SELECT ot.ejemplo, COUNT(id_otra_tabla)
              FROM bbdd1.otra_tabla ot
              GROUP BY ot.ejemplo
              HAVING COUNT(id_otra_tabla) > 2);
```



Una subconsulta es una consulta completa que aparece en la cláusula FROM, WHERE o HAVING de una sentencia SQL.

```
SELECT sum(te.precio)
FROM bbdd1.tabla_ejemplo te
WHERE EXISTS (SELECT ot.ejemplo, COUNT(id_otra_tabla)
              FROM bbdd1.otra_tabla ot
              GROUP BY ot.ejemplo
              HAVING COUNT(id_otra_tabla) > 2);
```

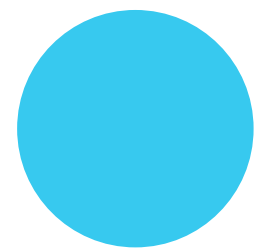
En el caso de estar presente en el WHERE o HAVING se utiliza con un conector como "=", IN, NOT IN, EXISTS, NOT EXISTS.

La subconsulta (consulta interna o anidada) se ejecuta antes de la consulta principal y su resultado se usa para ejecutar la consulta principal.

Las subconsultas se ubican en el lado derecho de la condición de comparación.

DML

Subconsultas

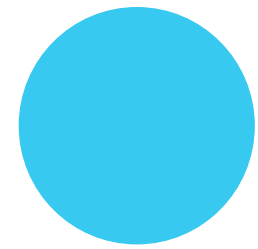


En el SELECT se pueden realizar operaciones matemáticas

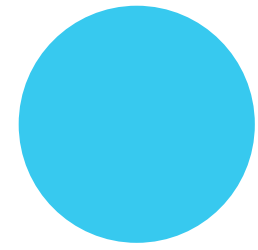
- **Operadores: Suma (+), Resta (-), Multiplicación (*), División (/)**
- **Multiplicación (*) y División (/) toman prioridad sobre la Suma (+) y la Resta (-)**
- **Los operadores de la misma prioridad se evalúan de izquierda a derecha**
- **Los paréntesis se usan para forzar un orden de evaluación y para dar claridad a las expresiones.**

DML

Operaciones matemáticas



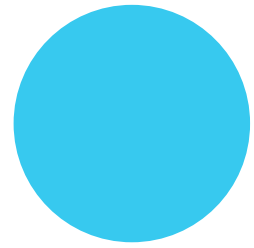
Un valor NULL es un valor NO disponible, NO asignado, NO conocido o NO aplicable.



Un NULL NO es lo mismo que CERO o un espacio en blanco

DML

NULL



Un alias de columna:

- Renombra el encabezado de la columna
- Es útil con expresiones (o cálculos)
- Se coloca después de la columna o la expresión. Opcionalmente se usa la palabra clave AS antes del alias.
- Se debe colocar entre comillas dobles (") si tiene espacios, caracteres especiales o diferencia entre mayúsculas y minúsculas.

DML

ALIAS
Columna



Un alias de tabla:

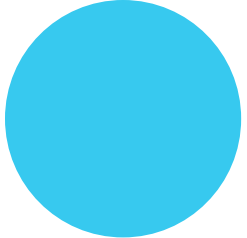
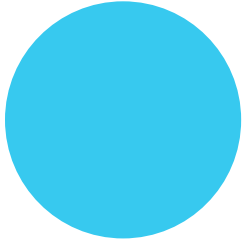
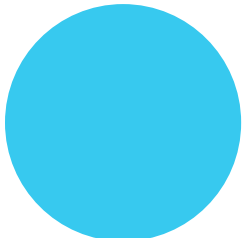
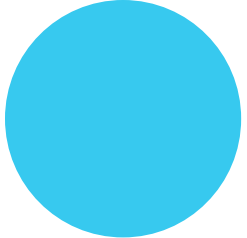
- Son cortos.
- Al asignarse en la cláusula **FROM** se debe usar en toda la sentencia **SELECT**.
- Deben tener un significado.
- Son validos en la sentencia **SELECT** actual.

DML

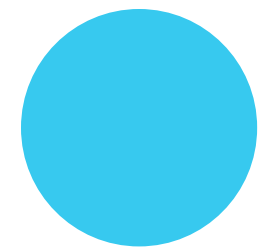
ALIAS
Tabla

Pautas para escribir sentencias SQL

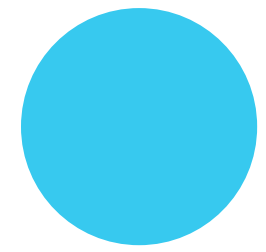


-  **NO son sensibles a mayúsculas y/o minúsculas**
-  **Pueden ser escritas en una o más líneas**
-  **Las palabras claves no pueden ser abreviadas o divididas entre líneas separadas**
-  **Usualmente se escriben con una cláusula por línea para mejorar su lectura**

PAUTAS PARA ESCRIBIR SENTENCIAS SQL



Usualmente se “identita” para mejorar su lectura

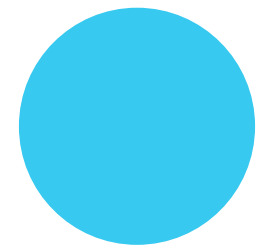


Usualmente las palabras claves se escriben en mayúscula y todas las otras (Tablas, Columnas) en minúsculas para mejorar su lectura

PAUTAS PARA ESCRIBIR SENTENCIAS SQL

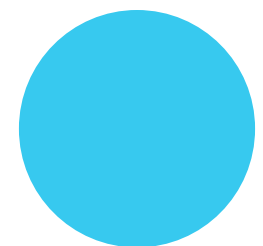
Vistas





Representación lógica de un subconjunto de datos de una o más tablas (No es una copia de los datos). Es una tabla lógica que se basa en otra tabla o conjunto de tablas (tablas base). La vista se almacena como un SELECT en el diccionario de datos.

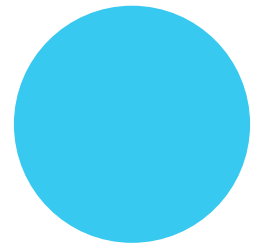
```
CREATE VIEW bbdd1.vista_ejemplo AS (SELECT te.descripcion, te.precio  
FROM bbdd1.tabla_ejemplo te);
```



Las tablas y las vistas comparten el mismo espacio de nombres en la base de datos, por eso una base de datos no puede contener una tabla y una vista con el mismo nombre.

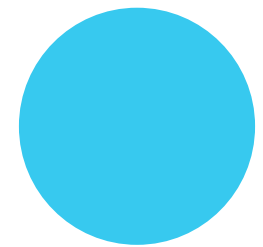
DML

VISTAS



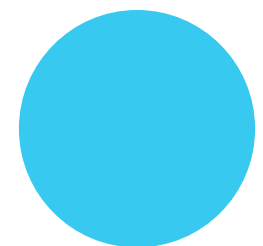
Representación lógica de un subconjunto de datos de una o más tablas (No es una copia de los datos). Es una tabla lógica que se basa en otra tabla o conjunto de tablas (tablas base). La vista se almacena como un SELECT en el diccionario de datos.

```
CREATE VIEW bbdd1.vista_ejemplo AS (SELECT te.descripcion, te.precio  
FROM bbdd1.tabla_ejemplo te);
```



Representación lógica de un subconjunto de datos de una o más tablas (No es una copia de los datos). Es una tabla lógica que se basa en otra tabla o conjunto de tablas (tablas base). La vista se almacena como un SELECT en el diccionario de datos.

```
CREATE VIEW bbdd1.vista_ejemplo AS (SELECT te.descripcion, te.precio  
FROM bbdd1.tabla_ejemplo te);
```



Las tablas y las vistas comparten el mismo espacio de nombres en la base de datos, por eso una base de datos no puede contener una tabla y una vista con el mismo nombre.

DML

VISTAS

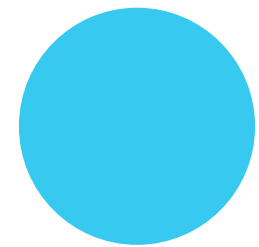


Las vistas sirven para:

- **Restringir el acceso a los datos**
- **Mostrar sólo algunas columnas o filas de una tabla a determinados usuarios**
- **Hacer las consultas complejas más fáciles de usar para los usuarios**
- **Hacer una vista que oculte una consulta que reúne (join) a varias tablas**
- **Presentar diferentes vistas de los mismos datos**

DML

VISTAS

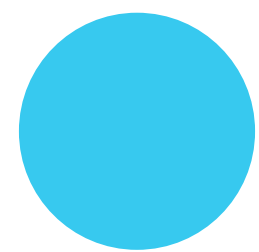


La definición de una vista está sujeta a las siguientes limitaciones:

- **La sentencia SELECT no puede contener una subconsulta en su cláusula FROM.**
- **La sentencia SELECT no puede hacer referencia a variables del sistema o del usuario.**
- **La sentencia SELECT no puede hacer referencia a parámetros de procedimientos.**

DML

VISTAS

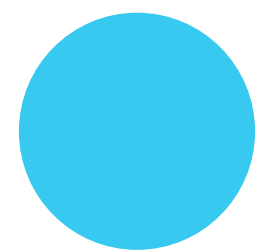


La definición de una vista está sujeta a las siguientes limitaciones (cont):

- Dentro de un procedimiento, la definición no puede hacer referencia a parámetros de la rutina o a variables locales.**
- Cualquier tabla o vista referenciada por la definición debe existir. Sin embargo, es posible que después de crear una vista, se elimine alguna tabla o vista a la que se hace referencia. Para comprobar la definición de una vista en busca de problemas de este tipo, utilice la sentencia CHECK TABLE.**

DML

VISTAS



La definición de una vista está sujeta a las siguientes limitaciones (cont):

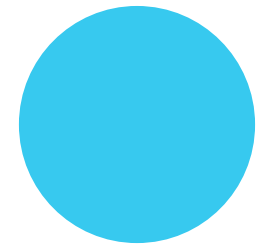
- **La definición no puede hacer referencia a una tabla TEMPORARY, y tampoco se puede crear una vista TEMPORARY.**
- **No se puede asociar un trigger con una vista.**
- **En la definición de una vista está permitido ORDER BY, pero es ignorado si se seleccionan columnas de una vista que tiene su propio ORDER BY.**

DML

VISTAS

Store Procedures



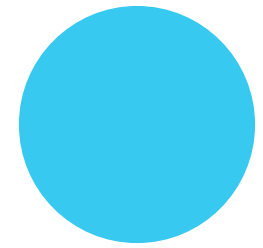


Es un bloque de código SQL que se almacena en la base de datos y puede ser ejecutado repetidamente.

- **Se utilizan para realizar tareas específicas, como consultas complejas, operaciones tupla a tupla, etc.**
- **Se almacenan en el servidor, de manera pre-compilada, y se pueden invocar desde aplicaciones externas o directamente desde el servidor.**

DML

STORE PROCEDURE

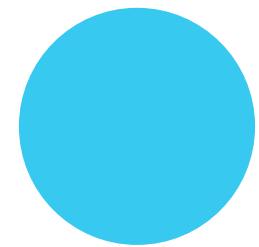


El uso de SP nos proporciona

- **Acceso homogéneo: Cuando es necesario realizar las mismas operaciones desde diferentes aplicaciones, escritas en diferentes lenguajes y hasta en diferentes plataformas.**
- **Consistencia en las operaciones: para cada ejecución, la secuencia de instrucciones retorna de manera consistente los resultados.**

DML

STORE PROCEDURE

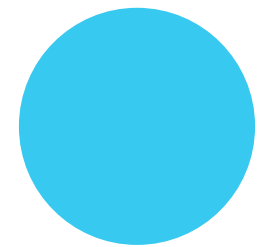


El uso de SP nos proporciona(cont)

- **Seguridad: Permite no acceder a tablas directamente, sino que solo por medio del SP**
- **Mejora la performance: se disminuye el tráfico entre el servidor y el cliente**
- **Consultas complejas: tenemos estructuras y funciones adicionales para resolver consultas complejas**

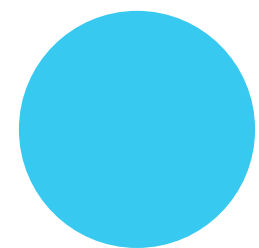
DML

STORE PROCEDURE

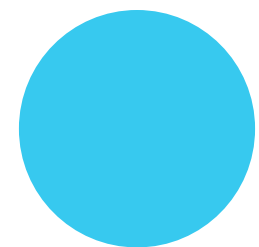


Un SP tiene un nombre, parámetros y un cuerpo.

```
DELIMITER //  
CREATE PROCEDURE actualizar_precio (IN p_id INTEGER, IN nuevo_precio FLOAT)  
BEGIN  
    UPDATE bbdd1.tabla_ejemplo  
    SET precio=nuevo_precio  
    WHERE id_tabla_ejemplo = p_id;  
END //  
DELIMITER ;
```



Es necesario redefinir el delimitador para evitar confusiones entre los delimitadores del SP y el estándar.

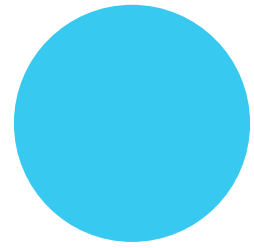


Al SP se lo invoca con la sentencia CALL().

```
CALL bbdd1.actualizar_precio(2,100.55);
```

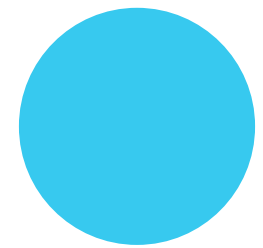
DML

STORE PROCEDURE



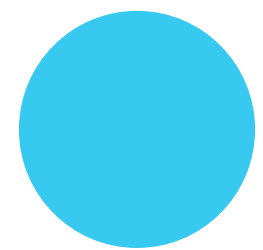
Un SP tiene un nombre, parámetros y un cuerpo.

```
DELIMITER //  
CREATE PROCEDURE actualizar_precio (IN p_id INTEGER, IN nuevo_precio FLOAT)  
BEGIN  
    UPDATE bbdd1.tabla_ejemplo  
    SET precio=nuevo_precio  
    WHERE id_tabla_ejemplo = p_id;  
END //  
DELIMITER ;
```

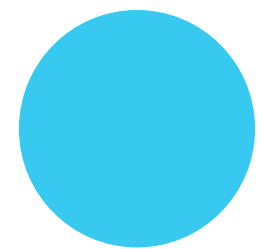


Un SP tiene un nombre, parámetros y un cuerpo.

```
DELIMITER //  
CREATE PROCEDURE actualizar_precio (IN p_id INTEGER, IN nuevo_precio FLOAT)  
BEGIN  
    UPDATE bbdd1.tabla_ejemplo  
    SET precio=nuevo_precio  
    WHERE id_tabla_ejemplo = p_id;  
END //  
DELIMITER ;
```



Es necesario redefinir el delimitador para evitar confusiones entre los delimitadores del SP y el estándar.



Al SP se lo invoca con la sentencia CALL().

```
CALL bbdd1.actualizar_precio(2,100.55);
```

DML

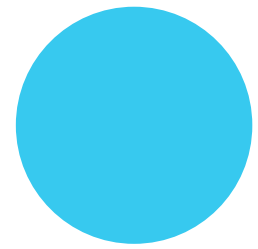
STORE PROCEDURE

 El contenido de un SP puede incluir

- Sentencias SQL
- Declaración de variables locales
- Uso de estructuras de control:
condicionales,
repetitivas,
manejo de excepciones
- Llamadas a otros procedimientos
- Manejo de transacciones.

DML

STORE PROCEDURE

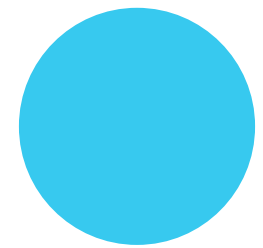


Para declarar las variables locales se utiliza la cláusula **DECLARE**, con el nombre, tipo y opcionalmente un valor por defecto.

```
DECLARE precio_minimo FLOAT 1.99;
```

DML

STORE PROCEDURE



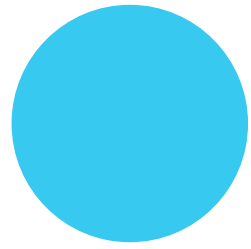
Los SP pueden recibir 3 tipos de parámetros

- **Entrada (IN): se puede modificar su valor pero no será reflejado fuera del SP.**
- **Salida (OUT): dentro del cuerpo del SP se le asigna un valor y será reflejado fuera del mismo.**
- **Entrada/Salida (IN/OUT): se puede usar y modificar su valor, que será reflejado fuera del SP.**

Todo parámetro será una variable local en el cuerpo del SP.

DML

STORE PROCEDURE



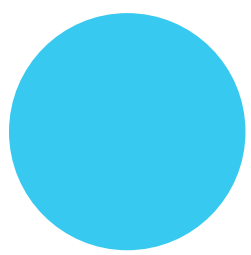
Se pueden usar estructura de control condicionales como IF y CASE.

Se pueden usar estructuras de control repetitivas como LOOP, WHILE y REPEAT UNTIL.

```
DELIMITER //
CREATE PROCEDURE pruebaestructurascontrol()
BEGIN
    DECLARE cant_iteraciones INT;
    SET cant_iteraciones = 0;
loop_label : LOOP
    SET cant_iteraciones = cant_iteraciones + 1;
    IF cant_iteraciones = 3 THEN
        ITERATE loop_label;
    END IF;
    IF cant_iteraciones >= 4 THEN
        LEAVE loop_label;
    END IF;
END LOOP;
END//
DELIMITER ;
```

DML

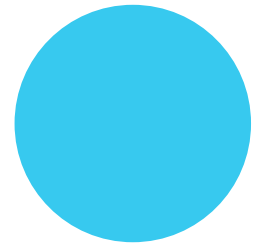
STORE PROCEDURE



Se pueden usar estructura de control condicionales como IF y CASE.

Se pueden usar estructuras de control repetitivas como LOOP, WHILE y REPEAT UNTIL.

```
DELIMITER //
CREATE PROCEDURE pruebaestructurascontrol()
BEGIN
    DECLARE cant_iteraciones INT;
    SET cant_iteraciones = 0;
    loop_label : LOOP
        SET cant_iteraciones = cant_iteraciones + 1;
        IF cant_iteraciones = 3 THEN
            ITERATE loop_label;
        END IF;
        IF cant_iteraciones >= 4 THEN
            LEAVE loop_label;
        END IF;
    END LOOP;
END//
DELIMITER ;
```

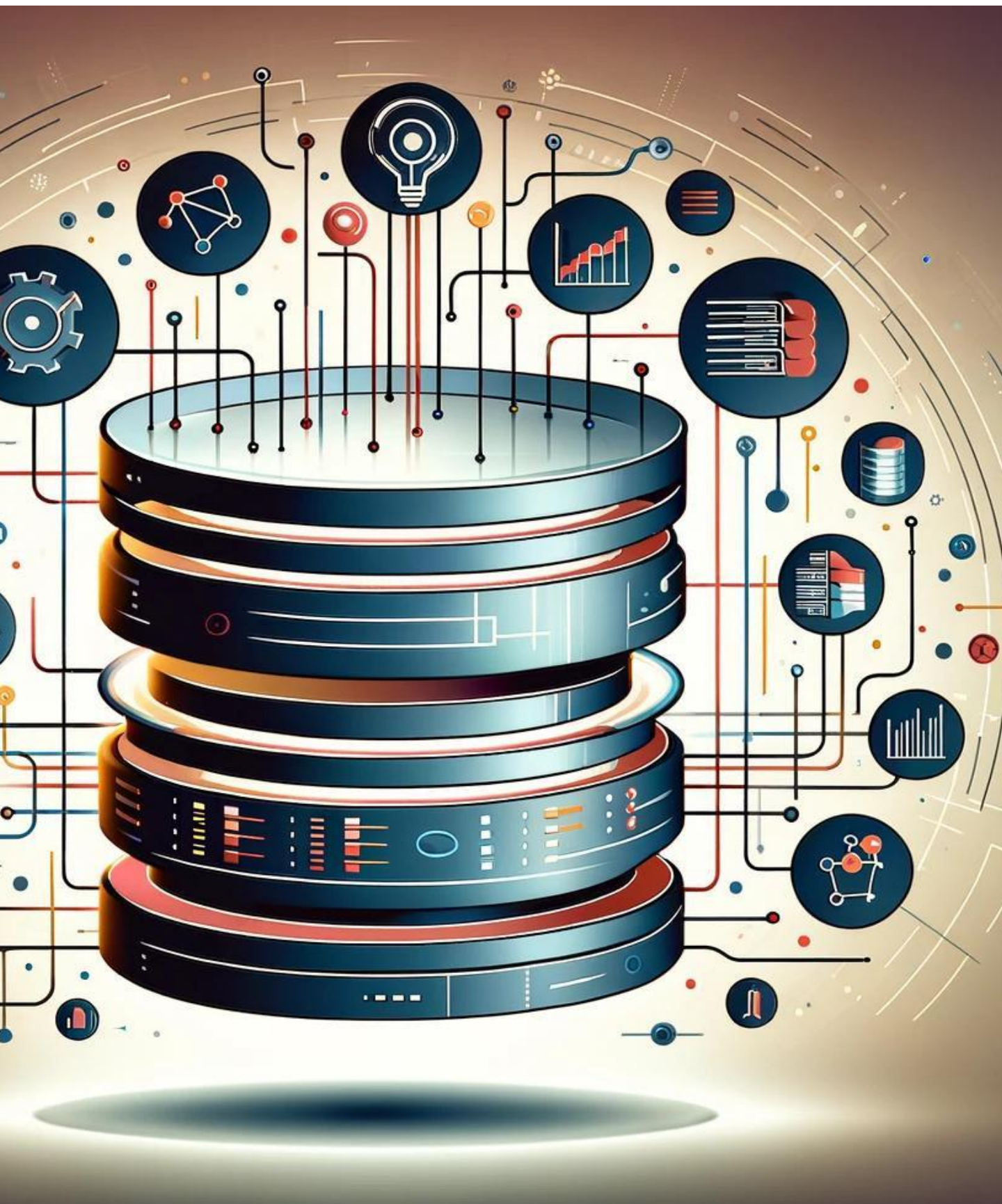


Privilegios que tiene que tener el usuario para:

- **Crearlo: CREATE ROUTINE**
- **Modificarlo: ALTER ROUTINE**
- **Ejecutarlo: EXECUTE**

DML

STORE PROCEDURE



Bibliografía de la clase

Bibliografía

- <https://dev.mysql.com/doc/refman/9.4/en/>

Importante!



**Los slides usados en las clases
teóricas de esta materia, no son
material de estudio por sí solos en
ningun caso.**