# Modelo de Entidad-Relación

**Modelado de Datos**: organizar los datos para que representen una situación del mundo real con la mayor fidelidad posible, con el objetivo de poder manejarlos computacionalmente

• No es posible tener un conocimiento completo del mundo real: centrar la atención en la información relevante, ocultando o ignorando otra que NO sea relevante o que resulte inadecuada para el propósito requerido

**¿Cómo Construir el Modelo de Datos?**

• Identificación de los **datos** de la realidad: actores, recursos, objetos, etc. del mundo real de los cuales interesa guardar información

• Identificación de **relaciones** entre datos: detección de los vínculos significativos que se dan entre los elementos de las etapas anteriores

• **Abstracción** de datos y relaciones: representación simbólica sólo de los elementos detectados en las etapas anteriores

**Entidad**

Cualquier objeto (real o abstracto) que existe en la realidad y acerca del cual se desea almacenar información en la base de datos

➢ Cosas tangibles: Artículo, Repuesto, Rodado ➢ Roles desempeñados por personas u organizaciones: Cliente, Proveedor, Personal

➢ Incidentes: Usado para representar la ocurrencia de un hecho (en un sistema de una compañía de seguros: Siniestros; en una empresa de transporte: Viajes)

➢ Interacciones: Representan alguna transacción (Compra, Pedido, Venta, Pago)

➢ Es importante **una buena elección del nombre** dado a una entidad para la legibilidad y el entendimiento del modelo de datos

**Atributo**

Un atributo es una abstracción que identifica características, propiedades que posee una entidad. Los atributos de una entidad deben ser:

✓ Completos: capturar toda la información que interesa del objeto, desde el punto de vista del sistema

✓ Plenamente elaborados: cada atributo captura un aspecto separado de la entidad

✓ Mutuamente independientes: cada atributo debe tomar un valor independientemente de los valores asumidos por otros atributos

**Clasificación de Atributos**

• Atributos identificadores: el o los atributos que permiten identificar unívocamente a una instancia de una entidad. Constituyen la "clave primaria"

• Atributos descriptivos: son las características intrínsecas de cada instancia de la entidad; como lo dice su nombre, describen a la entidad, representan sus propiedades

• Atributos referenciales: son atributos que sirven para relacionar entidades entre sí. Se denominan REFERENCIALES ya que hacen referencia al ATRIBUTO IDENTIFICADOR de la entidad con que se relacionan

**Atributo Identificador Único**

• Se denomina identificador a uno o más atributos que identifican unívocamente cada instancia de una entidad; es conocido también como "clave candidata“, “clave primaria”..

• Para elegir el atributo identificador debemos tener en cuenta estas reglas:

✓ Que la clave sea mínima: Es decir elegir la alternativa en la que se necesiten menos atributos para conformar la clave

✓ Elegir el atributo más significativo dentro del dominio del problema que se está modelando

• Toda entidad tiene por lo menos un atributo como posible atributo identificador. El o los atributos identificadores se señalan con el símbolo "@"(arroba), o de lo contrario con la sigla PK (clave primaria)

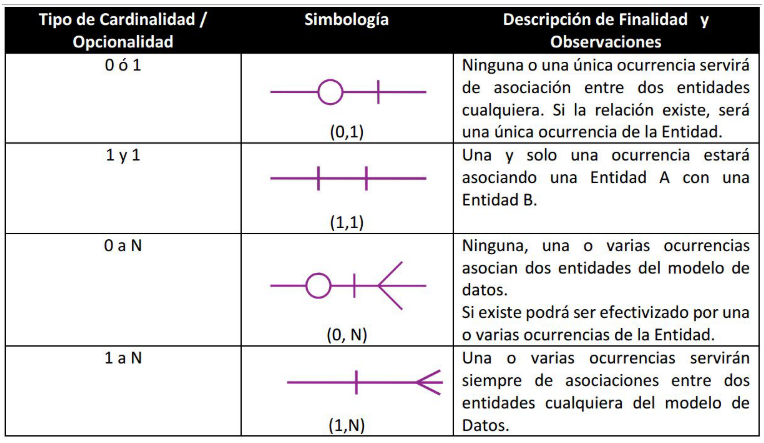
• Si no hay un atributo identificador, se crea uno ficticio

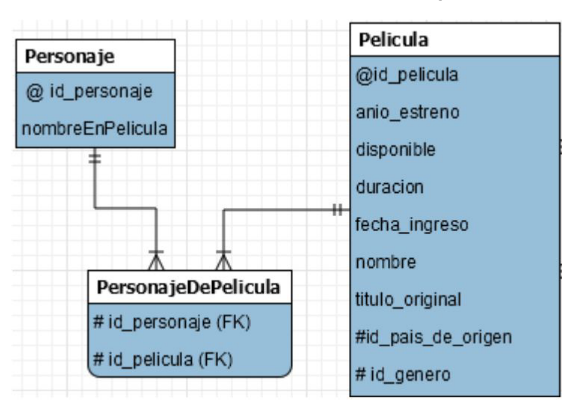
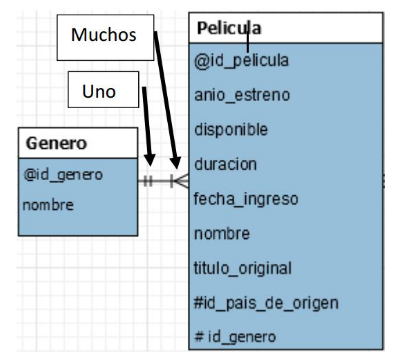
**Relación**

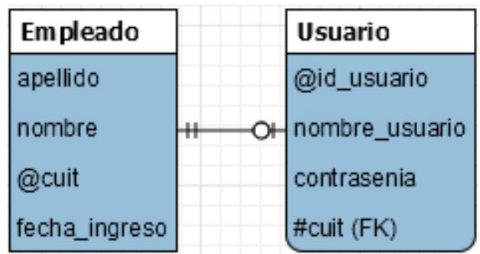
• Una relación es la abstracción de un conjunto de asociaciones que existen entre las instancias de dos entidades, por ejemplo, existe una relación entre Película y Genero

• Cardinalidad: Indica para una instancia de una entidad A con cuántas instancias de la entidad B se relaciona. Las posibilidades son: 0, 1 o muchos

• Opcionalidad: Indica para una instancia de una entidad A, si la relación con instancias de la entidad B, es opcional u obligatoria. Las posibilidades son: 0 o 1



Ejemplo 0 a 1: Ejemplo 1 a muchos: Casos Muchos a Muchos:



**Atributo Referencial**

• Un atributo referencial/clave extranjera/clave derivada se utiliza para establecer relaciones entre diferentes entidades de un modelo entidad-relación

• Se dice que un atributo j, ó un conjunto de atributos, de una entidad B es un atributo referencial si y sólo si satisface dos propiedades:

• Cada valor j es nulo del todo (es decir, no existe) o no nulo del todo. En caso de ser un atributo compuesto, formado por más de un atributo

• Existe una entidad A con atributo identificador j tal que cada valor no nulo de j es en la entidad B idéntico al valor j en alguna instancia de la entidad A. Es decir que si en B el atributo j tiene valor es porque existe ese mismo valor de j en la entidad A

Ejemplo: PELICULA y GENERO. Se utiliza el símbolo “#” para señalar que un atributo es referencial, o de lo contrario con la sigla FK (clave foránea)

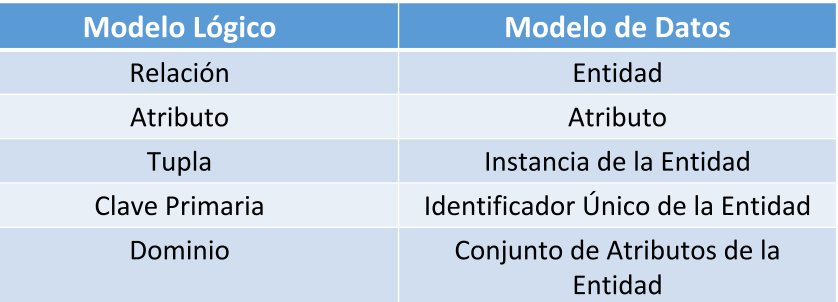
# Modelo Lógico Relacional

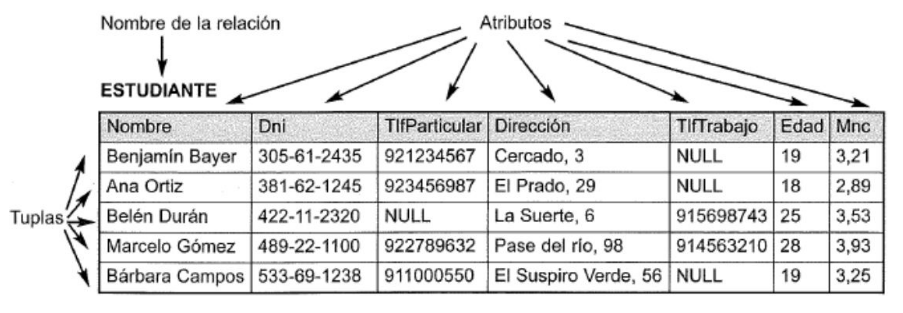
• El Modelo Lógico Relacional es un modelo de datos conceptual de alto nivel

• Este modelo y sus variaciones se utilizan con frecuencia para el diseño conceptual de las aplicaciones de base de datos

• Este modelo es una forma de representar los datos (mediante tablas), y la manera para manipular esa representación (utilizando operadores)

• Se ocupa de tres aspectos de los datos: su estructura, su integridad y su manipulación





• Una **relación** corresponde a lo que conocemos como **tabla**, que se utiliza para representar los datos que queremos almacenar en nuestra base de datos. En el modelado de entidad relación es la **Entidad**

• Un **atributo** corresponde a una **columna o campo**. El número de atributos se denomina **grado**

• Una **tupla** corresponde a una **fila o registro** de esa tabla. En el modelado de entidad relación es lo que denominamos **instancia** de la Entidad. El número de tuplas de una tabla se denomina **cardinalidad**

• La **clave primaria** es un identificador único para la tabla. Nunca existen dos filas de la tabla con el mismo valor en esa columna o combinación de columnas

• Un **dominio** es una colección de valores, de los cuales uno o más atributo (columnas) obtienen sus valores reales

# Integridad en las Bases de Datos Relacionales

Se debe vigilar las operaciones de inserción y modificación y rechazar cualquier entrada que no cumpla con las especificaciones

• El Modelo Relacional incluye dos reglas generales de integridad:

**Reglas de Integridad para Clave Primaria**

➢ Unicidad: no existen dos tuplas de R con el mismo valor de a, en un momento dado

➢ Minimalidad: si a es un atributo compuesto, no puedo eliminar un componente de a sin destruir la propiedad de unicidad

• En el caso de que existan varias claves candidatas, debemos escoger una y las demás serán claves Alternativas (claves candidata que no son clave primaria)

• Ningún componente de la clave primaria de una relación base puede aceptar nulos

**Reglas de Integridad para Clave Foránea**

Todos los valores que toma una clave foránea deben ser valores que existen en la clave primaria que referencia. Si un valor de una clave foránea no estuviese presente en la clave primaria correspondiente, representaría una referencia o una conexión incorrecta

• Ningún componente del atributo identificador en una entidad aceptará NULOS (nulo se considera que es inexistente, es decir, ausencia de valor)

Esta política se puede aplicar en las siguientes **operaciones de actualización** que violarían la regla de integridad:

a) Borrado de una tupla que tiene una clave primaria referenciada

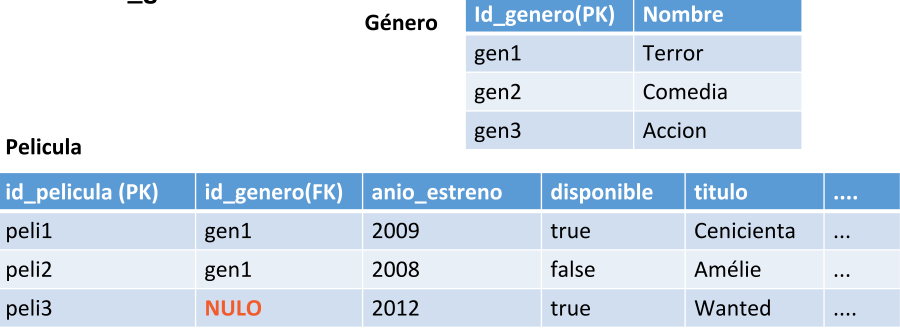
b) Modificación de los valores de los atributos de la clave primaria de una tupla que tiene una clave primaria referenciada

En los casos anteriores, algunas de las **políticas** que se podrán aplicar serán las siguientes:

• Restricción: ejemplo, queremos borrar al género gen1, no podremos hacerlo porque hay películas que lo referencian.

• Actualización en cascada: ejemplo, si queremos borrar el género gen3, se borrarán también la película peli3 que hay en la tabla película

• Anulación: ejemplo, si queremos borrar al género gen3, se modificarán todos las películas que lo tenían asignado, y pasarán a tener un valor nulo en id\_genero.



# Introducción a SQL

* Para acceder y manipular los datos dentro de la base de datos se necesita un lenguaje declarativo
* **SQL** significa Lenguaje de consulta estructurado (Structured Query Language)
* **SQL** es un lenguaje de bases de datos global: cuenta con sentencias para definir datos, consultas y actualizaciones
* Sintaxis para:
  + Lenguaje de definición de datos: **DDL** (Data Definition Language)
  + Lenguaje de manipulación de datos: **DML** (Data Manipulation Language)

Algunas funciones del estándar SQL son:

* DDL
  + **Definición de datos:**
    - **Creación de tablas (CREATE)**
    - **Modificación de tablas (ALTER)**
    - **Eliminación de tablas (DROP)**
* DML
  + Consulta de datos
    - Selección (SELECT)
  + Actualización de los datos
    - Inserción (INSERT)
    - Actualización (UPDATE)
    - Eliminación (DELETE)

**Sintaxis**

* ***{Alternativas****}*, entre llaves se colocarán las palabras que tienen opciones o alternativas en la sentencia a la que pertenecen
* ***[Opcional]*,** entre corchetes se colocarán las palabras que son opcionales en la sentencia, es decir que pueden colocarse o pueden obviarse

# Lenguaje de definición de datos – DDL

* Permite crear y definir nuevas bases de datos, campos e índices:
  + **CREATE:** Crea nuevas tablas, campos e índices
  + **DROP:** Elimina tablas e índices
  + **ALTER:** Modifica las tablas agregando campos o cambiando la definición de los campos

**Creación de Base de Datos**

CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] ***nombre\_base\_datos***

**Creación de Tablas**

* CREATE [TEMPORARY] TABLE [IF NOT EXISTS] nombre\_de\_tabla (
* nombre\_de\_columna **tipo\_de\_dato** [NOT NULL | NULL] [DEFAULT valor\_por\_defecto] [AUTO\_INCREMENT] [UNIQUE | PRIMARY KEY] )
* [CONSTRAINT [nombre\_relación] FOREIGN KEY (nombre\_columna) REFERENCES nombre\_de\_tabla (nombre\_columna)]
* [ON DELETE **opciones\_de\_referencia**]
* [ON UPDATE **opciones\_de\_referencia**]

*Ejemplo:*

**CREATE TABLE IF NOT EXISTS** `PELICULA` (

 `id\_pelicula` **INT NOT NULL**,

 `anio\_estreno` **BIGINT**,

 `disponible` **BIT**,

 `duracion` **VARCHAR(45)**,

 `fecha\_ingreso` **DATETIME**,

**PRIMARY KEY** (`id\_pelicula`))

**Tipo de Datos Más Comunes**

* BIT[(longitud)] → indica un valor booleano
* INT[(longitud)] → indica un número entero
* BIGINT[(longitud)]→ indica un número entero grande
* DOUBLE[(longitud,decimales)] → indica un número en punto flotante con precisión
* DATE → almacenar solo el día, mes y año
* TIME → almacena una hora con minutos y segundos
* DATETIME→ almacenar día, mes, año, hora, minuto y segundo
* CHAR[(longitud)] → indica un caracter
* VARCHAR(longitud)  → indica una cadena de caracteres (string)

**Opciones de Referencias**

* Las opciones de referencia sirven para establecer que se hará en casos de que se elimine o se actualice una fila de la tabla primaria que está siendo referenciada por una fila de la tabla secundaria
* **CASCADE**: Eliminar o actualizar la fila de la tabla primaria, y automáticamente eliminar o actualizar las filas coincidentes en la tabla secundaria
* **SET NULL**: Eliminar o actualizar la fila de la tabla primaria, y establecer la columna de clave externa (Foreign key) de la tabla secundaria a NULL. Si se especifica una SET NULL, hay que asegurarse que no se haya declarado la columna de la tabla secundaria como NOT NULL
* **RESTRICT:** Rechaza la operación de eliminación o actualización en la tabla primaria
* **SET DEFAULT:** Para una operación de eliminación o actualización en la tabla primaria se establecerá un valor por defecto para la tabla secundaria

**Definición de Opcionalidades**

* Las opciones **NOT NULL | NULL**, sirven para especificar si dicha columna puede aceptar valores nulos: NULL, o si no puede guardar valores nulos: NOT NULL
* También se puede de manera opcional, indicar un valor por defecto **DEFAULT**  para una columna
* **AUTO\_INCREMENT:** se refiere a un valor AUTO INCREMENTAL; sirve para aquellas columnas con valores que numéricos enteros donde se necesita que dicho valor se incremente en uno por cada fila insertada en la tabla. Se utiliza muy frecuentemente en las claves primarias
* **UNIQUE:** sirve para indicar que una columna en la tabla no puede tener valores repetidos, debe ser UNICA. No pueden existir dos filas en la tabla que tengan el mismo valor para un atributo definido como UNIQUE
* **PRIMARY KEY:** sirve para especificar que una columna es clave primaria de una tabla. Si una columna es clave primaria implica que sus valores no deben repetirse (UNIQUE)
* **CONSTRAINT:** significa RESTRICCIÓN y se le asigna un nombre para identificarla; dicho nombre funciona como clave que identifica unívocamente a cada CONSTRAINT que exista en la base de datos, por lo que toda CONSTRAINT debe tener un nombre no repetido; luego se indica con la palabra FOREIGN KEY, cuál es la columna de la tabla que funciona como clave foránea, indicando a continuación a cuál tabla y a cuál columna de dicha tabla hace referencia la clave foránea con la palabra REFERENCES

**Alteración de Tabla**

* Las acciones que se pueden hacer sobre una **Tabla** mediante la alteración son:
* **ADD:** COLUMN, INDEX, KEY, CONSTRAINT, FULLTEXT
* **CHANGE:** COLUMN
* **MODIFY:** COLUMN
* **DROP:** COLUMN. PRIMARY KEY, INDEX, KEY, FOREING KEY
* **DISABLE:** KEYS
* **ENABLE:** KEYS
* **RENAME:** INDEX, KEY, TO, AS
* **ORDER BY**

*Ejemplo:*

* Para cambiar el nombre de una tabla de t1 a t2

**ALTER TABLE t1 RENAME t2;**

* Para cambiar los tipos de datos de sus atributos, suponiendo que tenemos dos atributos **a** es INTEGER y **b** es CHAR con longitud 10: CHAR (10), cambiaremos a como TINYINT sin aceptar nulos y cambiaremos b por el nombre **c** y con longitud 20:

**ALTER TABLE t2 MODIFY a TINYINT NOT NULL, CHANGE b c CHAR(20);**

* Para agregar una nueva columna d, de tipo DATETIME:

**ALTER TABLE t2 ADD d DATETIME;**

* Para agregar un nuevo índice en una columna d

**ALTER TABLE t2 ADD INDEX (d);**

* Para eliminar la columna c:

**ALTER TABLE t2 DROP COLUMN c;**

* Para agregar una nueva columna auto incremental y clave primaria, de tipo entera llamada e:

**ALTER TABLE t2 ADD e INT NOT NULL AUTO\_INCREMENT, ADD PRIMARY KEY (e);**

**Borrar Tabla**

* DROP TABLE remueve una o más tablas. Para poder eliminar tablas, el usuario que ejecuta la sentencia debe tener privilegios de DROP para cada tabla que quiera eliminar
* Con este comando se elimina todos los datos de la tabla, tanto la definición como las filas de datos de la misma. Si no existe la tabla que se desea borrar MySQL devuelve error, por eso es útil utilizar la opción IF EXISTS en caso de que exista la tabla la elimina

**DROP [TEMPORARY] TABLE [IF EXISTS] nombre\_tabla [,nombre\_tabla] ...**

**Creación de Índices**

Un índice es una estructura de disco asociada con una tabla o una vista que acelera la recuperación de filas de la tabla

Un índice contiene claves generadas a partir de una o varias columnas de la tabla. Dichas claves están almacenadas en una estructura que permite que SQL Server busque de forma rápida y eficiente la fila o filas asociadas a los valores de cada clave

**CREATE INDEX *nombre\_indice***

**ON *nombre\_tabla* (*col\_name* [(*length*)] [ASC | DESC])**

**[*opcion\_algoritmo* | *opcion\_bloqueo*] ...**

**Tipos de Índices**

* Una tabla puede contener los siguientes tipos de índices:
* Agrupado
  + Los índices agrupados ordenan y almacenan las filas de los datos de la tabla de acuerdo con los valores de la clave del índice. Son columnas incluidas en la definición del índice
  + La única ocasión en la que las filas de datos de una tabla están ordenadas es cuando la tabla contiene un índice clúster. Cuando una tabla tiene un índice clúster, la tabla se denomina tabla agrupada. Si una tabla no tiene un índice clúster, sus filas de datos están almacenadas en una estructura sin ordenar denominada montón
* No agrupado
  + Tienen una estructura separada de las filas de datos. Contienes los valores de clave de índice no agrupado y cada entrada de valor de clave tiene un puntero a la fila de datos que contiene el valor clave
  + El puntero de una fila hacia una fila de datos se denomina localizador de fila. La estructura del localizador de filas depende de si las páginas de datos están almacenadas en un montón o en una tabla agrupada
  + Se puede agregar columnas sin clave al nivel hoja de un índice no agrupado con el fin de eludir los límites existentes para las claves de índice, 900 bytes y columnas de 16 claves, así como para ejecutar consultas indizadas y totalmente cubiertas

# Lenguaje de manipulación de datos - DML

* Permite generar consultas para ordenar, filtrar y extraer datos de la base de datos:
  + **SELECT:** Consulta registros de la base de datos que satisfagan un criterio determinado
  + **INSERT:** Carga lotes de datos en la base de datos en una única operación
  + **UPDATE:** Modifica los valores de los campos y registros especificados
  + **DELETE:** Elimina registros de una tabla de una base de datos

**SELECT sin condición**

**SELECT** <lista **de** columnas separadas por coma> **FROM** <lista de tablas>;

**SELECT \* FROM** <**nombre**\_tabla>;

**SELECT DISTINCT** <nombre\_**columna**>, <nombre\_columna> **FROM** <nombre\_tabla>**;**

* SELECT identifica las columnas a recuperar (el ***qué***)
* FROM identifica la tabla (de ***dónde*** obtener los datos)
* El resultado de una consulta es una **tabla** (si es un número, se considera como una tabla con una fila y una columna)
* <lista de atributos> es una lista de los atributos cuyos valores serán recuperados por la consulta
* <lista de tablas> es una lista de las tablas necesarias para procesar la consulta

**SELECT con condición**

**SELECT** <lista de atributos> **FROM** <nombre\_tabla> **WHERE** <condición>;

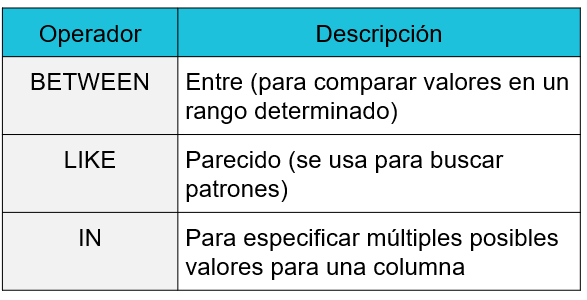
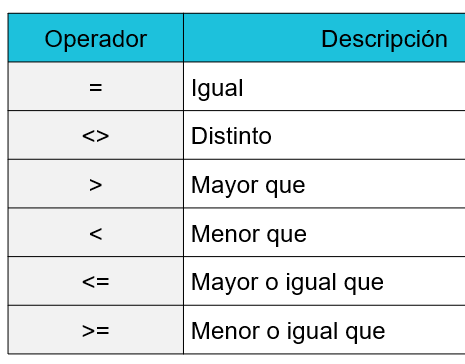
**SELECT** <lista de atributos> **FROM** <nombre\_tabla> **WHERE** <nombre\_columna> <operador> <valor>;

**SELECT** <lista de atributos> **FROM** <nombre\_tabla> **WHERE** <nombre\_columna> **BETWEEN** <x> **AND** <y>;

**SELECT** <lista de atributos> **FROM** <nombre\_tabla> **WHERE** <nombre\_columna> **LIKE** <er>;

**SELECT** <lista de atributos> **FROM** <nombre\_tabla> **WHERE** <condicion> **AND** <condicion>;

**SELECT** <lista de atributos> **FROM** <nombre\_tabla> **WHERE** <condicion> **OR** <condicion>;

* <condición> es una expresión condicional (booleana) que se evalúa en las filas de la tabla
* Una cláusula WHERE contiene una condición lógica, la cual usa **operadores de comparación** u **operadores lógicos** (AND, OR, NOT)
* El resultado de la comparación puede ser:
  + Verdadero (T)
  + Falso (F)
  + Desconocido (U)
* BETWEEN x AND y es equivalente a a >= x AND a <= y
* NOT BETWEEN x AND y es equivalente a a < x OR a > y
* El operador ***LIKE*** busca en una columna según un patrón especifico. Por ejemplo todos los nombres que empiezan con determinada letra. También se emplea en la forma negativa ***NOT LIKE****.* Comodines

%: cualquier secuencia de cero o más caracteres

\_ : denota un solo carácter

*Ejemplos*:

* Seleccionar los nombres y las marcas de los productos cuyo precio sea superior a 900$

**SELECT** nombre, **marca**, precio **FROM** e01\_producto **WHERE** precio **>** 900;

* Obtener la marca del producto cuyo código es 10.

**SELECT** marca, codigo\_producto **FROM** e01\_producto **WHERE** codigo\_producto**=**10;

* Seleccionar código y nombre de los productos que tengan un stock de entre 60 y 100 unidades

**SELECT** codigo\_producto, nombre, stock **FROM** e01\_producto **WHERE** stock **BETWEEN** 60 **AND** 100;

* Seleccionar apellido y código del cliente con nombre “Jescie”

**SELECT** nro\_cliente, apellido, nombre **FROM** e01\_cliente **WHERE** nombre **LIKE** 'Jescie';

* Seleccionar código, nombre y apellido de clientes cuyo nombre empiece con la letra “F”

**SELECT** nro\_cliente,nombre,apellido **FROM** e01\_cliente **WHERE** nombre **LIKE** "F%";

* Obtener los productos cuyo nombre sea “fish” o tengan stock de más de 26 u.

**SELECT** codigo\_producto, nombre **FROM** e01\_producto **WHERE** (nombre **LIKE** 'fish') **OR** (stock **<=** 26);

**Actualización de datos:**

**INSERT**

**INSERT INTO** <nombre\_tabla> (<columna1>, <columna2>,...) **VALUES** (<valor1>, <valor2>,...);

* Se utiliza para agregar nuevas filas de datos a una tabla determinada.
* Debemos especificar el nombre de la tabla y una lista de valores para la fila.
* Los valores deben suministrarse en el mismo orden en el que se especificaron los atributos correspondientes en el comando CREATE TABLE.
* Ejemplo 1: Insertar el teléfono móvil 229 - 4639675 para el cliente cuyo número (nro\_cliente) es 50.
  + INSERT INTO E01\_TELEFONO VALUES (229,4639675,'M',50);
* Ejemplo 2: Una segunda forma de la sentencia INSERT permite especificar explícitamente los nombres de los atributos que se corresponden con los valores suministrados en el comando INSERT.
  + INSERT INTO E01\_TELEFONO (nro\_telefono, tipo, codigo\_area) VALUES (4547894,'M',249);

**UPDATE**

**UPDATE** <nombre\_tabla> **SET** <col\_nombre\_1>={<valor1>|DEFAULT} [, <col\_nombre\_2>={<valor2>|DEFAULT}] [**WHERE** <condicion>] [**ORDER BY** ...] [**LIMIT** <cantidad\_filas>];

* Se utiliza para modificar los valores de atributo de una o más filas seleccionadas.
* La cláusula WHERE selecciona las filas que se van a modificar.
* La cláusula SET especifica los atributos que se modificarán y sus nuevos valores.
* La actualización del valor de una clave puede propagarse a los valores de la foreign key de las filas de otras relaciones en caso de haberse especificado una opción de acción referencial en las restricciones de integridad referencial del DDL.
* Ejemplo: Cambiar el nombre del cliente número “15” por "Juan".
  + UPDATE e01\_cliente SET nombre = 'Juan' WHERE nro\_cliente = 15;

**DELETE**

**DELETE FROM** <nombre\_tabla> [**WHERE** <condicion>] [**ORDER BY** ...] [**LIMIT** <cantidad\_filas>];

* El comando DELETE elimina filas de una tabla.
* Con la cláusula WHERE se seleccionan las filas que se van a eliminar.
* La eliminación se puede propagar a filas de otras tablas si se han especificado opciones de acciones referenciales en las restricciones de integridad referencial del DDL.
* Ejemplo: Borrar todos los teléfonos del cliente número “20”.
  + DELETE FROM e01\_telefono WHERE nro\_cliente = 20;

# Subconsultas

**SELECT** \* **FROM** <tabla> **WHERE** <columna> **IN** ( [ **SUBCONSULTA** ] ) ;

**SELECT** \* **FROM** <tabla> **WHERE** <columna> **NOT** **IN** ( [ **SUBCONSULTA** ] ) ;

**SELECT** \* **FROM** <table> **WHERE** **EXISTS** ( [ **SUBCONSULTA** ] ) ;

**SELECT** \* **FROM** <table> **WHERE** **NOT** **EXISTS** ( [ **SUBCONSULTA** ] ) ;

**SELECT** \* **FROM** <tabla> **WHERE** <columna> <operador> **ANY** ( [ **SUBCONSULTA** ] );

**SELECT** \* **FROM** <tabla> **WHERE** <columna> <operador> **ALL** ( [ SUBCONSULTA ] );

* Una subconsulta es una consulta anidada en una instrucción SELECT, INSERT, UPDATE o DELETE, o bien en otra subconsulta.
  + Primero obtenemos los valores de la subconsulta interna.
  + Luego se ejecuta la condición externa.
* Una subconsulta anidada en la instrucción externa SELECT tiene los componentes siguientes:
  + Una consulta SELECT normal.
  + Una cláusula normal FROM que incluye uno o varios nombres de tablas o vistas.
  + Una cláusula opcional WHERE.
  + Una cláusula opcional GROUP BY.
  + Una cláusula opcional HAVING.
  + La cláusula WHERE puede contener un SELECT anidado, como una consulta conjunta en 2 pasos.
* El predicado EXISTS (con la palabra reservada NOT opcional) se utiliza en comparaciones de verdad/falso para determinar si la subconsulta devuelve algún registro.
* Se puede utilizar el predicado ANY o SOME, los cuales son sinónimos, para recuperar registros de la consulta principal, que satisfagan la comparación con cualquier otro registro recuperado en la subconsulta.
* El predicado ALL se utiliza para recuperar únicamente aquellos registros de la consulta principal que satisfacen la comparación con todos los registros recuperados en la subconsulta.

Ejemplos:

* Obtener el teléfono y el número de cliente del cliente con nombre “Wanda” y apellido “Baker”.

**SELECT** \* **FROM** e01\_telefono **WHERE** nro\_cliente **IN** (

**SELECT** nro\_cliente **FROM** e01\_cliente **WHERE** nombre **LIKE** 'Wanda' **AND** apellido **LIKE** 'Baker'

);

* Seleccionar todos los clientes que tengan registrada al menos una factura.

**SELECT** nombre, apellido **FROM** e01\_cliente **WHERE** **EXISTS** (

**SELECT** \* **FROM** e01\_factura **WHERE** nro\_cliente = e01\_cliente.nro\_cliente

);

* Seleccionar todos los datos de los clientes a los cuales se les ha realizado al menos una factura.

**SELECT** \* **FROM** e01\_cliente **WHERE** nro\_cliente = **ANY** (

**SELECT** nro\_cliente **FROM** e01\_factura

);

* Seleccionar las facturas que se hayan realizado sobre todos los productos.

**SELECT** \* **FROM** e01\_detalle\_factura **WHERE** nro\_item = **ALL** (

**SELECT** codigo\_producto **FROM** e01\_producto

);

# Consultas Multi-Tabla

* Por medio del operador **JOIN** podemos combinar dos tablas según una condición para obtener tuplas compuestas por atributos de las dos relaciones combinadas.
* Además de **INNER JOIN** que devuelve todas las filas que cumplen con la condición de igualdad de atributos expresada en la cláusula **ON**.
* **LEFT JOIN:** Devuelve todas las filas de la tabla izquierda y las filas coincidentes de la tabla derecha.
  + Si existe alguna fila de la tabla izquierda que no tiene coincidencia con ninguna fila de la tabla derecha, dicha fila es mostrada en el resultado, pero aparecerá el valor NULL en las columnas de la tabla derecha ya que no existe coincidencia.
* **RIGHT JOIN:** Devuelve todas las filas de la tabla derecha y las filas coincidentes de la tabla izquierda.
  + Si existe alguna fila de la tabla derecha que no tiene coincidencia con ninguna fila de la tabla izquierda, dicha fila es mostrada en el resultado, pero aparecerá el valor NULL en las columnas de la tabla izquierda ya que no existe coincidencia.
* En el caso de las consultas multi-tablas es útil utilizar abreviaciones de nombres o **alias** de tablas para llamarlas de forma más simple y breve. También puede darse apodos a los nombres de columnas.

Ejemplos

**SELECT** c.nombre, c.apellido, t.nro\_telefono **FROM** e01\_cliente c

**INNER JOIN** e01\_telefono t **ON** c.nro\_cliente = t.nro\_cliente;

**SELECT** c.\*, COUNT(f.nro\_factura) **FROM** e01\_cliente c

**LEFT JOIN** e01\_factura f **ON** (c.nro\_cliente = f.nro\_cliente)

# Funciones de Agrupamiento

Estas funciones operan sobre conjuntos de filas para proporcionar un resultado por grupo

* La función **COUNT** devuelve el número de filas o valores especificados en una consulta
* Las funciones **SUM, MAX, MIN y AVG** se aplican a un conjunto o multiconjunto de valores numéricos y devuelven, respectivamente, la suma, el valor máximo, el valor mínimo y el promedio de esos valores

Ejemplos

**SELECT COUNT**(\*) **FROM** e01\_producto **WHERE** nombre **LIKE** 'fish';

**SELECT AVG**(precio) **FROM** e01\_producto **WHERE** nombre **LIKE** 'fish';

**SELECT MAX**(precio) **FROM** e01\_producto;

**SELECT MIN**(precio) **FROM** e01\_producto;

**Group By**

o A veces queremos aplicar funciones a subgrupos de filas de una tabla, estando los subgrupos divididos en base a algunos valores de uno o más atributos

o Cada grupo estará compuesto por las filas que tienen el mismo valor para algún(os) atributo(s), denominado(s) atributo(s) de agrupamiento

o Después podemos aplicar la función independientemente a cada grupo

La cláusula **GROUP BY** especifica los atributos de agrupamiento, que también deben aparecer en la cláusula SELECT o Si se usa la cláusula GROUP BY en una sentencia SELECT, se dividen las filas de la tabla consultada en grupos

**SELECT** <atributos>, <fun de grupo> (<nombre\_columna>)

**FROM** <nombre\_tabla> [WHERE condicion]

**GROUP BY** expresión de grupo;

Ejemplo:

**SELECT** nombre, **COUNT**(nombre) **FROM** e01\_producto **GROUP BY** nombre;

**Having**

Se puede anexar la cláusula **HAVING** para restringir grupos o Las filas se agrupan por la/s columnas especificada/s

o Se aplica la función de grupo o Se muestran los grupos que satisfacen la cláusula

o NO se puede utilizar la cláusula WHERE para restringir grupos

o Se debe utilizar la cláusula HAVING para restringir grupos

**SELECT** <lista\_atributos>

**FROM** <nombre\_tabla> [WHERE condición]

**GROUP BY** <nombre\_columna>

**HAVING** where\_condicion;

Ejemplo

**SELECT** marca, **SUM**(stock) **FROM** e01\_producto **GROUP BY** marca **HAVING** SUM(stock)<50;

**Order By**

SQL permite ordenar las filas del resultado de una consulta por los valores de uno o más atributos, utilizando la cláusula ORDER BY. El orden predeterminado es el ascendente

**SELECT** <lista\_atributos> **FROM** <nombre\_tabla> ...

**ORDER BY** <nombre\_columna> **[ASC | DESC];**

Ejemplo

**SELECT** \* **FROM** e01\_cliente **ORDER BY** apellido, nombre **ASC**;

**Limit**

Se puede utilizar para restringir el número de filas que puede retornar una sentencia

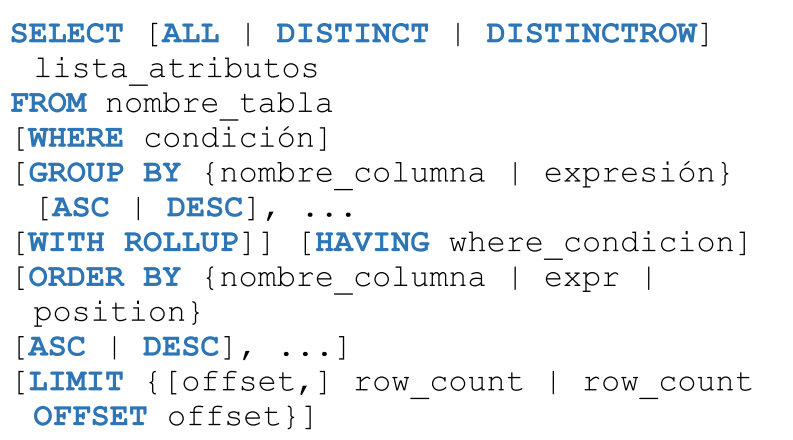
**SELECT** <lista\_atributos> **FROM** <nombre\_tabla> ...

**LIMIT** <numero>;

Ejemplo

**SELECT** \* **FROM** e01\_producto **ORDER BY** precio **DESC LIMIT** 3;

**Estructura completa de la sentencia SELECT**



# Seguridad y Transacciones en Bases de Datos

Tipos de Amenazas

• Pérdida de integridad. La integridad se pierde si se realizan cambios no autorizados en los datos mediante acciones intencionadas o accidentales

• Pérdida de disponibilidad. Se refiere a que los objetos estén disponibles para un usuario humano o para un programa que tenga los derechos correspondientes

• Pérdida de confidencialidad. La confidencialidad de la base de datos tiene relación con la protección de los datos frente al acceso no autorizado

**Niveles de Seguridad**

• Sistema de bases de datos. Autorización al acceso a una parte limitada de la base de datos

• Sistema operativo. La debilidad de la seguridad del sistema operativo puede servir como medio para el acceso no autorizado

• Red. La seguridad en el nivel del software de la red es tan importante en Internet y en las redes privadas de las empresas

• Físico. Los sitios que contienen los sistemas informáticos deben estar protegidos de intrusos

• Humano. Los usuarios deben ser autorizados cuidadosamente para reducir la posibilidad de intrusión

**Autenticación y Autorización**

• La autenticación es el proceso por el cual se identifica un usuario como válida para posteriormente acceder a ciertos recursos definidos

• Relacionado con la gestión de usuarios y control de acceso al DBMS

• La autorización es el proceso sobre el cual se establecen qué tipos de recursos están permitidos o denegados para cierto usuario o grupo de usuarios concreto

• Relacionado con permisos (lectura, escritura) para cada usuario autentificado

**Creación de Usuarios**

**CREATE USER 'nombre\_usuario'@'host' IDENTIFIED BY ‘tu\_contrasena';**

• Seguido al nombre de usuario, se debe especificar la IP desde donde podrá realizar conexiones a la base de datos el usuario creado. Puede ser:

• 'localhost' o '127.0.0.1', desde la misma PC en la que se encuentre instalado MySQL, es decir el host local

• '192.168.1.100', sólo permite conexiones desde dicha IP (utilizada para identificar a un PC en un LAN)

• '%' es un comodín que permite conexiones desde cualquier IP

**Concesión de Privilegios a Usuarios**

**GRANT** [permiso] **ON** [nombre de bases de datos]. [nombre de tabla]

**TO** ‘[usuarioX]’@'host’ **[WITH GRANT OPTION];**

**REVOKE** [permisos] **ON** [nombre de base de datos] [nombre de tabla]

**FROM** ‘[nombre de usuario]’ @‘host’;

**FLUSH PRIVILEGES;**

(Una vez que se haya finalizado con la configuración de privilegios (GRANT o REVOKE) se deben refrescar todos los con este comando)

Tipos de Permisos

• ALL : esta opción otorga todos los permisos

• CREATE: permite crear nuevas tablas o bases de datos

• DROP: permite eliminar tablas o bases de datos

• DELETE: permite eliminar registros de tablas

• INSERT: permite insertar registros en tablas

• SELECT: permite leer registros en las tablas

• UPDATE: permite actualizar registros seleccionados en tablas

• WITH GRANT OPTION: permite que usuarioX maneje privilegios de otros usuarios

Ejemplos

• GRANT ALL ON midb.usuarios TO ‘juan’@'%’ WITH GRANT OPTION;

• GRANT ALL ON midb.\* TO ‘admin’@'%’; • GRANT SELECT ON midb.log TO ‘auditor’@'192.168.1.100’;

• REVOKE ALL ON midb.\* TO ‘admin’@'%’; • FLUSH PRIVILEGES;

# Transacciones en Base de Datos

Conceptos

• Una transacción es un mecanismo para definir las unidades lógicas del procesamiento de una base de datos

• Una transacción se inicia por la ejecución de un programa escrito en un lenguaje en un lenguaje de programación

• Una transacción está delimitada por instrucciones de la forma inicio transacción y fin transacción

Fallos

• Fallo de la computadora (caída del sistema)

• Un error de la transacción o del sistema

• Errores locales o condiciones de excepción detectados por la transacción

• Control de la concurrencia

• Fallo del disco rígido

• Problemas físicos y catástrofes

ACID

**Atomicidad** • Requiere que cada transacción sea "todo o nada": si una parte de la transacción falla, todas las operaciones de la transacción fallan, y por lo tanto la base de datos no sufre cambios

**Consistencia** • Asegura que cualquier transacción llevará a la base de datos de un estado válido a otro estado válido. Cualquier dato que se escriba en la base de datos tiene que ser válido de acuerdo a todas las reglas definidas

**Aislamiento**(Isolation) • Asegura que la ejecución concurrente de las transacciones resulte en un estado del sistema que se obtendría si estas transacciones fueran ejecutadas una detrás de otra

**Durabilidad** • Significa que una vez que se confirmó una transacción quedará persistida, incluso ante eventos como pérdida de alimentación eléctrica, errores y caídas del sistema

Estados

• **Activa**, la transacción permanece en este estado durante su ejecución

• **Parcialmente confirmada**, después de ejecutarse la última instrucción

• **Fallida**, tras descubrir que no puede continuar la ejecución normal

• **Abortada**, después de haber retrocedido la transacción y restablecido la base de datos a su estado anterior al comienzo de la transacción

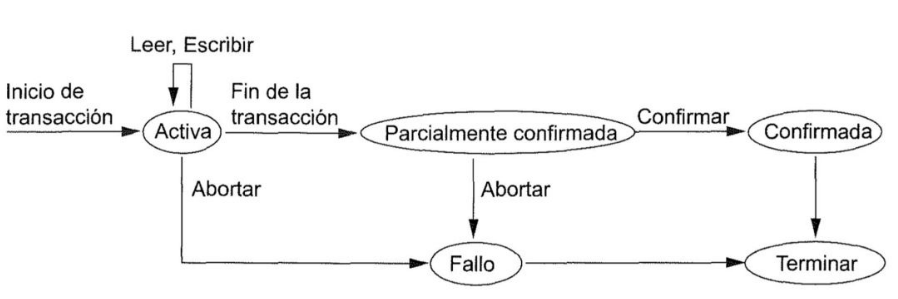
• **Confirmada**, tras completarse con éxito

Operaciones

• **INICIO DE TRANSACION**. Marca el inicio de la ejecución de una transacción • LEER o ESCRIBIR. Especifican operaciones de lectura o escritura en los elementos de la base de datos que se ejecutan como parte de una transacción

• **FIN DE LA TRANSACION**. Especifica que las operaciones LEER y ESCRIBIR de la transacción han terminado y marca el final de la ejecución de la transacción. En este punto se comprueba si los cambios introducidos pueden confirmarse

• **CONFIRMAR (Commit).** Señala una finalización satisfactoria de la transacción, por lo que los cambios (actualizaciones) ejecutados se pueden enviar con seguridad a la base de datos

• **ABORTAR (Rollback).** Señala que la transacción no ha terminado satisfactoriamente, por lo que deben deshacerse los cambios

Concurrencia de transacciones

• **Enfoque de ejecución secuencial** de transacciones es más sencillo de implementar, pero menos eficiente. Para comenzar una transacción es necesario finalizar la anterior

• **Enfoque de ejecución concurrente** permite varias transacciones que actualizan concurrentemente los datos y puede provocar complicaciones en la consistencia de los mismos

El **esquema de control de concurrencia de un DBMS** controla la interacción entre las transacciones concurrentes para evitar que se destruya la consistencia de la base de datos

Ventajas:

• **Productividad y utilización de recursos mejorados** • El número de transacciones que puede ejecutar en un tiempo dado aumenta cuando varias transacciones se ejecutan en paralelo. Por ej., las operaciones de E/S, como uso de CPU y discos pueden trabajar en paralelo en una computadora

• **Tiempo de espera reducido** • Frente a transacciones cortas y largas, la ejecución concurrente reduce los retardos impredecibles en la ejecución de las transacciones • Se reduce también el tiempo medio de respuesta

# Índices

• Estructuras complementarias en DBMS que se asocian con los atributos de las tablas y agilizan la operaciones de búsqueda

• Juegan el mismo papel que los índices de los libros o los catálogos de fichas de las bibliotecas

• Por ejemplo, para recuperar un registro de cuenta dado su número de cuenta, el sistema de bases de datos buscaría en un índice para encontrar el bloque de disco en que se encuentra el registro correspondiente, y entonces extraería ese bloque de disco para obtener el registro cuenta

Tipos de Índices

• **Índices ordenados**. Estos índices están basados en una disposición ordenada de los valores

• **Índice primario o índices con agrupación**. La clave de búsqueda especifica el orden secuencial del archivo (archivos ordenados secuencialmente). La clave de búsqueda de un índice primario es normalmente la clave primaria (pero no necesariamente)

• **Índices secundarios o índices sin agrupación**. Las claves de búsqueda especifican un orden diferente del orden secuencial del archivo. Un archivo puede tener varios índices secundarios además de su método de acceso principal

• **Índices asociativos**. Estos índices están basados en una distribución uniforme de los valores a través de una serie de cajones (buckets). El valor asignado a cada cajón está determinado por una función, llamada función de asociación (hash function)

Índices Multinivel

• Solución para índices muy grandes • El índice se trata como si fuese un archivo secuencial y se construye otro índice sobre el índice con agrupación

• Para localizar un registro se usa en primer lugar una búsqueda binaria sobre el índice más externo para buscar el registro con el mayor valor de la clave de búsqueda que sea menor o igual al valor deseado

• Los índices multinivel están estrechamente relacionados con la estructura de árbol, tales como los árboles binarios usados para la indexación en memoria

Índices en SQL

**create index** <nombre-índice> **on** <nombre-tabla> (<lista-atributos>);

**drop index** <nombre-índice>;

lista-atributos es la lista de atributos de la tabla que constituye la clave de búsqueda del índice

Ejemplo:

create index índice-s on sucursal (nombre-sucursal)

# Stored Procedures y Funciones

• Comúnmente llamados procedimientos almacenados aunque pueden ser funciones o procedimientos

• Son módulos de programa de bases de datos que el DBMS almacena y ejecuta en el servidor de bases de datos

• Ayudan a mantener integridad de los datos y su consistencia. Incrementan la seguridad de las operaciones en el DBMS

• Si varias aplicaciones necesitan un mismo programa de base de datos, este último se puede almacenar en el servidor e invocarlo desde esas aplicaciones

**CREATE PROCEDURE** <nombre del procedimiento> «parámetros»

<declaraciones locales> <cuerpo del procedimiento>;

**CREATE FUNCTION** <nombre de la función> «parámetros»

**RETURNS** <tipo de devolución> <declaraciones locales> <cuerpo de la función>;

Ejemplo

CREATE PROCEDURE saldo\_cuenta (IN dni INT, OUT saldo INT, OUT total\_de\_operaciones INT, OUT fecha\_ultima\_operacion INT)

BEGIN

cuerpo del procedimiento

END

**Variables**

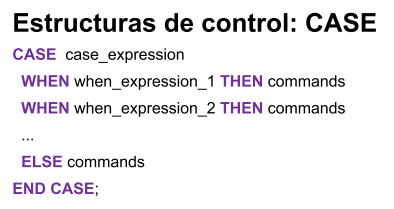
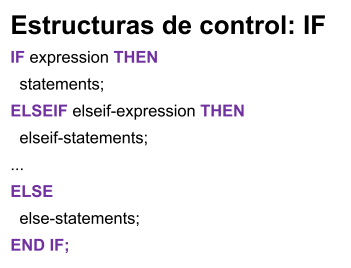
**DECLARE** <nombre\_variable> tipo\_de\_dato (tamaño) **DEFAULT** valor\_defecto.

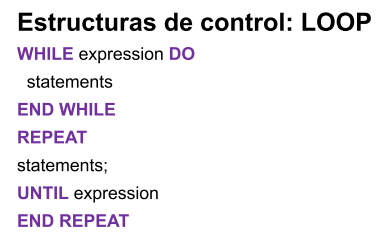
Ejemplos:

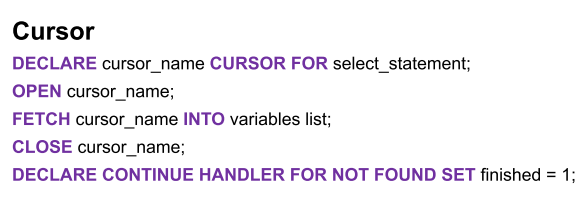
DECLARE x, y INT DEFAULT 0;

DECLARE total INT DEFAULT 0;

SET total= 10;

**Estructuras de control: IF Estructuras de control: CASE**

**Estructuras de control: LOOP Cursor**



# Vistas en SQL

**create view** <nombre-vista> **as** <expresión de consulta>

La <expresión de consulta> puede ser cualquier consulta válida

• Puede conceptualizarse como una tabla de sólo lectura o tabla virtual que deriva de otras tablas

• Las tablas de las que se deriva **pueden ser tablas base** (tablas propias de la base de datos, almacenadas físicamente en la base de datos) **o vistas definidas anteriormente**

• Se restringen las operaciones de actualización, **sólo puede consultarse**

• Permiten **encapsular la lógica compleja** de una consulta y simplificar la consulta desde una aplicación

• Frente a múltiples usuarios, permite **esconder detalles** de implementación de la DB y exponer sólo una parte de dichos datos

• En el caso de vistas indexadas, puede **mejorarse el rendimiento** general del sistema, sobre todo con consultas recurrentes

Ejemplo

**CREATE VIEW** detalle\_ultimos\_10\_movimientos **AS**

**SELECT** CL.dni, MO.\* **FROM** Clientes CL

**JOIN** Cuentas CU **ON** (CL.Id = CU.Id\_Cliente)

**JOIN** Movimientos MO **ON** (CU.Id = MO.Id\_Cuenta)

**ORDER BY** MO.Fecha **DESC LIMIT** 10

Luego se hace consulta:

1. SELECT \* FROM detalle\_ultimos\_10\_movimientos;

2. SELECT SUM(monto) FROM (SELECT \* FROM detalle\_ultimos\_10\_movimientos WHERE dni=’12345678’);

Actualización

Una vista es actualizable si:

• Está definida a partir de una única tabla

• Conserva la clave (primaria o alternativa)

• No está definida mediante agrupación o funciones de agregación

• No incluye la cláusula

• No incluye sub-consultas

• No se define mediante unión, intersección, diferencia

• Si la vista es actualizable y está definida con:

• **WITH CHECK OPTION**: todos los INSERT y UPDATE sobre la vista son chequeados para asegurar que los datos satisfacen la condición de definición de la vista

• **WITH LOCAL CHECK OPTION**: chequea la integridad sobre la vista

• **WITH CASCADE CHECK OPTION**: chequea la integridad sobre la vista y cualquier vista dependiente

Ejemplos

1. CREATE TABLE t1 (a INT); 2. CREATE VIEW v1 AS SELECT \* FROM t1 WHERE a < 2 WITH CHECK OPTION;

3. CREATE VIEW v2 AS SELECT \* FROM v1 WHERE a > 0 WITH LOCAL CHECK OPTION;

4. CREATE VIEW v3 AS SELECT \* FROM v1 WHERE a > 0 WITH CASCADED CHECK OPTION;

•mysql> INSERT INTO v2 VALUES (2);

Query OK, 1 row affected (0.00 sec)

•mysql> INSERT INTO v3 VALUES (2);

ERROR 1369 (HY000): CHECK OPTION failed 'test.v3'

# Triggers en SQL

**CREATE TRIGGER** nombre del trigger

**{BEFORE|AFTER}**

**{INSERT|UPDATE|DELETE}**

**ON** nombre de la tabla

**FOR EACH ROW**

**BEGIN**

sentencia SQL

**END**;

• Un trigger es un objeto de base de datos con nombre que se asocia a una tabla, y se activa cuando ocurre un evento en particular para la tabla

• Un uso puede ser verificar valores a ser insertados o llevar a cabo cálculos sobre valores involucrados en una actualización

• Se almacenan en la BD, por lo que son persistentes y accesibles para todas las operaciones de la base de datos

• Los triggers tienen dos palabras clave, **OLD** y **NEW**. Son los valores que tienen las columnas antes y después de la modificación

• Los **INSERT** permiten NEW, los **DELETE** sólo OLD y los **UPDATE** ambas

• **Triggers de nivel de fila**: se ejecutan una vez para cada fila afectada. Se crean utilizando la cláusula **for each row** en el comando create trigger

• **Triggers de nivel de instrucción**: se ejecutan una vez para cada instrucción. Se crean utilizando la cláusula **for each statement** en el comando create trigger

• **Triggers Before y After**: puesto que los triggers son ejecutados por sucesos, puede establecerse que se produzcan inmediatamente antes (before) o después (after) de dichos sucesos

• **Triggers Instead Of**: Indican lo que se tiene que hacer en lugar de realizar las acciones que invoca el trigger

• **Triggers de esquema**: Son triggers sobre operaciones en el nivel de esquema tales como create table, alter table, drop table, audit, rename, truncate y revoke

• **Triggers en nivel de base de datos**: puede crear triggers que se activen al producirse sucesos de la base de datos, incluyendo errores, inicios de sesión, conexiones y desconexiones

Ejemplo

Cuando un usuario inserta un registro en la tabla Prueba cuyo Atributo2 tenga un valor superior a 1000 se insertará automáticamente una fila en la tabla ResultadoDisparador con:

• la fecha de la inserción (sysdate) → Fecha actual del Sistema

• el aviso “Registro con Atributo2 superior a 1000”

• el nombre de la tabla Prueba

