MODELO ENTIDAD-RELACIÓN

ENTIDAD

Una entidad es un objeto, real o abstracto, acerca del cual se recoge información de interés para la base de datos.
Entidades fuertes: tienen existencia por sí mismas. (alumnos, empleados, departamento.).
Entidades débiles: dependen de otra entidad para su existencia (hijos de empleados).

ATRIBUTOS

describen las características de una entidad. Tipos:
con simple valor
multivalor
derivados
clave
nulos.

CLAVES

1. Clave candidata: Se compone por uno o más atributos cuyos valores identifican unívocamente a cada ocurrencia de la entidad, sin que ningún subconjunto de ellos pueda realizar esta misma función. Una clave candidata es una **posible clave primaria**. Pueden definirse varias claves candidatas para luego seleccionar la más adecuada.
2. Clave primaria: Está compuesta por uno o más atributos cuyos valores i**dentifican unívocamente a cada ocurrencia de la entidad.** No pueden contener valores nulos ni repetidos. Esta clave es una de aquellas que anteriormente se seleccionaron como candidata.
3. Superclave: es el conjunto de uno o más atributos que, tomados colectivamente, permiten identificar de forma unívoca a la ocurrencia de una entidad. Se utiliza generalmente en las tablas de relación, este concepto se desarrollará en las próximas clases.

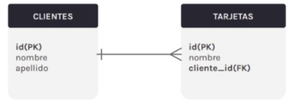
RELACIONES

UNO A UNO 1:1

Un usuario tiene solo una dirección. Una dirección pertenece solo a un usuario. Para establecer la relación colocamos la clave primaria de la dirección en la tabla de usuarios, indicando que esa dirección está asociada a ese usuario (Clave foránea).

UNO A MUCHOS 1:N

Un cliente puede tener muchas tarjetas. Una tarjeta pertenece solo a un cliente. Para establecer la relación colocamos la clave primaria del cliente en la tabla de tarjetas, indicando que esas tarjetas están asociadas a un usuario en particular.



*NOTA: las FK van del lado del "muchos"

MUCHOS A MUCHOS N:M

Un cliente puede comprar muchos productos. Un producto puede ser comprado por muchos clientes. En las relaciones N:M, en la base de datos, la relación en sí pasa a ser una tabla. Esta tabla intermedia —también conocida como tabla pivot— puede tener 3 datos: una clave primaria (PK) y dos claves foráneas (FK), cada una haciendo referencia a cada tabla de la relación.

DATOS

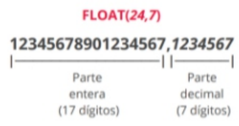
DE TIPO TEXTO

CHAR(n): n = 1 a 255 caracteres. Se recomienda en cadenas de texto de longitud poco variable.
VARCHAR (n) = 1 a 21845 caracteres. Cadenas de texto de longitud muy variable.
TINYTEXT: 0 a 255 caracteres.
MEDIUMTEXT: 0 a 16777215 caracteres.
TEXT: 0 a 4294967295 caracteres.
LONGTEXT: 0 a 18446744073709551615 caracteres.

DE TIPO NUMÉRICOS

DECIMALES

FLOAT(n,d): Almacena numeros de coma flotante PEQUEÑO.
Tiene precisión simple para la parte decimal (máximo 7 digitos)
n = 1 a 24 digitos incluyendo la parte decimal.
d= 0 a 7 digitos segun cuanto se asigne en n.
Ej:



DOUBLE(n,d): Almacena numeros de coma flotante GRANDE.
Tiene precisión doble para la parte decimal (máximo15 digitos)
n = 25 a 53 digitos incluyendo la parte decimal.
d= 0 a 15 digitos segun cuanto se asigne en n.
Ej:

ENTEROS

TINYINT: -128 a 127 (sin signo de 0 a 255)
SMALLINT: -32768 a 32767 (sin signo de 0 a 65535)
MEDIUMINT: -8388608 a 8388607 (sin signo de 0 a 16777215)
INT: -2147483648 a 2147483647 (sin signo de 0 a 4294967295)
BIGINT: -9223372036854775808 a 9223372036854775805 (sin signo de 0 a 18446744073709551615)

DE TIPO FECHA

DATE: solo fecha en formato YYYY-MM-DD. La fecha va entre comillas. "2014-01-14"
TIME: solo la hora en formato HH:MM:SS. La hora va entre comillas."23:59:59"
DATETIME: almacena fecha y hora en formato YYYY-MM-DD HH-MM-SS. "2021-05-15 11:50:55"

BOOLEANOS

MySQL guarda los booleanos como un cero o como un uno. Por cuestiones de performance, este tipo de dato viene desactivado y no se recomienda su uso. En el caso de querer guardar valores "verdaderos" y "falsos", se recomienda utilizar el tipo de dato tinyint(1), donde:
0 para representar el false.
1 para representar el true.

SENTENCIAS SQL Y OTROS

DDL

CREATE:
CREATE DATABASE
nombreDeLaBase

CREATE TABLE nombreDeLaTabla(
nombreColumna1 TipoDeDato
CONSTRAINT,);

DROP.
DROP TABLE IF EXIST nombreTabla;

ALTER:
ADD: para agregar una columna.
MODIFY: para modificar una columna.
DROP. para borrar una columna.

ALTER TABLE nombreTabla
DROP nombreColumna;

DML

INSERT:
Existen dos formas de agregar datos en una tabla:
Insertando datos en todas las Columnas.
Insertando datos en las columnas que especifiquemos.
Todas las columnas:
INSERT INTO nombreTabla (columna1, columna2, columna3...)
VALUES (valor1, valor2, valor3..);

Columnas especificas: Para insertar datos en una columna en específico, aclaramos la tabla y luego escribimos el nombre de la o las columnas entre los paréntesis.
INSERT INTO artistas(nombre)
VALUES ("Calle 13");

UPDATE: modificará los registros existentes de una tabla. Al igual que con DELETE, es importante no olvidar el WHERE.
UPDATE nombreTabla
SET columna1 = valor1, columna2=valor2
WHERE condición;

DELETE:
DELETE FROM nombreTabla
WHERE condicion;

SELECT

Toda consulta a la base de datos va a empezar con la palabra SELECT.

SELECT nombreColumna
FROM nombreaTabla;

SELECT DISTINCT

Al realizar una consulta en una tabla, puede ocurrir que en los resultados existan dos o más filas idénticas. En algunas situaciones, nos pueden solicitar un listado con registros no duplicados, para esto, utilizamos la cláusula DISTINCT que devuelve un listado en donde cada fila es distinta.

SELECT DISTINCT nombreColumna
FROM nombreaTabla;

WHERE

Condiciona y filtra las consultas del SELECT.

Operadores

=	Igual a	IS NULL	Es nulo
>	Mayor que	BETWEEN	Entre dos valores
>=	Mayor o igual que	IN	Lista de valores
<	Menor que	LIKE	Se ajusta a...
<=	Menor o igual que		
<>	Diferente a		
!=	Diferente a		

ORDER BY

Se utiliza para ordenar los resultados de una consulta según el valor de la columna especificada. Por defecto, se ordena de forma ascendente (ASC).

SELECT nombreColumna
FROM nombreTabla
WHERE condicion
ORDER BY nombreColumna1;

BETWEEN

BETWEEN: Cuando necesitamos obtener valores dentro de un rango, usamos el operador BETWEEN. Incluye los extremos. BETWEEN funciona con números, textos y fechas. Se usa como un filtro de un WHERE.

SELECT nombre, edad
FROM alumnos
WHERE edad BETWEEN 6 AND 12;

ORDEN DE ESCRITURA DE UNA QUERY

SELECT — CASE
FROM — JOINS
WHERE
GROUP BY
HAVING
ORDER BY

LIKE

Cuando hacemos un filtro con un WHERE, podemos especificar un patrón de búsqueda que nos permita especificar algo concreto que queremos encontrar en los registros.

COMODÍN % Es un sustituto que representa cero, uno, o varios caracteres.
COMODÍN _ Es un sustituto para un solo carácter.
SELECT nombreColumna
FROM nombreTabla
WHERE condicion LIKE '%A%'

LIMIT

Su funcionalidad es la de limitar el número de filas (registros/resultados) devueltas en las consultas SELECT. También establece el número máximo de registros a eliminar con DELETE.

SELECT nombreColumna1,
nombreColumna2
FROM nombreTabla
LIMIT cantidadDeRegistros;

OFFSET

Nos permite especificar a partir de qué fila comenzar la recuperación de los datos solicitados.

SELECT nombreColumna1,
nombreColumna2
FROM nombreTabla
LIMIT cantidadDeRegistros
OFFSET 20;

ALIAS

Los alias se usan para darle un nombre temporal y más amigable a las tablas, columnas y funciones

SELECT nombreColumna1,
nombreColumna2 AS Nombre
FROM nombreTabla;

SELECT nombreColumna1,
nombreColumna2
FROM nombreTabla AS Tabla;

FUNCIONES Y OTROS

DE AGREGACIÓN

DEVUELVEN **1** SOLO VALOR COMO RESULTADO.
Excepto COUNT, las funciones de agregación ignorarán los valores **NULL.**

COUNT: Devuelve un único resultado indicando la cantidad de filas/registros que cumplen con el criterio.

```
SELECT COUNT(*)
FROM movies;

SELECT COUNT (ID) AS total
FROM movies
WHERE genre_id=3;
```

AVG (average) :Devuelve un único resultado indicando el promedio de una columna cuyo tipo de datos debe ser numérico.

```
SELECT AVG(rating)
FROM movies;
```

SUM (suma): Devuelve un único resultado indicando la suma de una columna cuyo tipo de datos debe ser numérico.

```
SELECT SUM(length)
FROM movies;
```

MIN : Devuelve un único resultado indicando el valor mínimo de una columna cuyo tipo de datos debe ser numérico.

```
SELECT MIN (rating)
FROM movies;
```

MAX: Devuelve un único resultado indicando el valor máximo de una columna cuyo tipo de datos debe ser numérico.

```
SELECT MAX(length)
FROM movies;
```

DE ALTERACIÓN

CONCAT : Usamos CONCAT para concatenar dos o más expresiones:

```
SELECT CONCAT ('La respuesta es: ', 24, '.')
FROM actor;
```

COALESCE: Usamos COALESCE para sustituir el valor NULL en una sucesión de expresiones o campos. Es decir, si la primera expresión es Null, se sustituye con el valor de una segunda expresión, pero si este valor también es Null, se puede sustituir con el valor de una tercera expresión y así sucesivamente.

```
SELECT COALESCE (NULL, 'Digital House')
FROM tabla;
```

DATEDIFF : Usamos DATEDIFF para devolver la diferencia entre dos fechas, tomando como granularidad el intervalo especificado.

```
SELECT DATEDIFF ('2021-01-15', '2021-01-05')
FROM tabla;   //-----> devuelve 10 porque es la cantidad de dias de diferencia entre las fechas indicadas.
```

TIMEDIFF: Usamos TIMEDIFF para devolver la diferencia entre dos horarios, tomando como granularidad el intervalo especificado

```
SELECT TIMEDIFF ('2021-01-15 12:45:00', '2021-01-05 07:00:00')
FROM tabla;  // -----> devuelve 05:45:00
```

EXTRACT: Usamos EXTRACT para extraer partes de una fecha:

```
SELECT EXTRACT(WEEK FROM "2017-06-15");
```

La parte a extraer puede ser: MICROSECOND, SECOND, MINUTE, HOUR DAY, WEEK, MONTH, QUARTER, YEAR, SECOND_MICROSECOND, MINUTE_MICROSECOND, MINUTE_SECOND, HOUR_MICROSECOND HOUR_SECOND, HOUR_MINUTE, DAY_MICROSECOND, DAY_SECOND DAY_MINUTE, DAY_HOUR, YEAR_MONTH.

REPLACE: para reemplazar una cadena de caracteres por otro valor. Cabe aclarar que esta función hace distinción entre minúsculas y mayúsculas.

```
SELECT REPLACE ('Buenas noches', 'noches', 'tardes')
//reemplaza la palabra noches por tardes.
```

DATE_FORMAT :Usamos DATE_FORMAT para cambiar el formato de salida de una fecha según una condición dada.

```
SELECT DATE_FORMAT("2017-06-15", "%Y"); // ---> 2017
```

DATE_ADD: Usamos DATE_ADD para sumar o agregar un período de tiempo a un valor de tipo DATE o DATETIME.

```
SELECT DATE_ADD("2017-06-15", INTERVAL 10 DAY);
```

DATE_SUB: Usamos DATE_SUB para restar o quitar un período de tiempo a un valor de tipo DATE o DATETIME.

```
SELECT DATE_SUB("2017-06-15", INTERVAL 10 MONTH);
```

CASE: Usamos CASE para evaluar condiciones y devolver la primera condición que se cumpla.

```
SELECT id, titulo, rating
CASE
WHEN rating <4 THEN 'Mala'
WHEN rating BETWEEN 4 AND 6 THEN 'Regular'
ELSE 'Excelente'
END AS calificacion
FROM pelicula;
```

GROUP BY

La directriz GROUP BY nos va a permitir agrupar los registros de la tabla resultante de una consulta por una o más columnas, según nos sea necesario.

```
SELECT marca
FROM coche
WHERE anio_fabricacion = 2010
GROUP BY marca;
```

*Se usa para agrupar filas que contienen los mismos valores.
* Opcionalmente, se utiliza junto con las funciones de agregación (SUM, AVG, COUNT, MIN, MAX) con el objetivo de producir reportes resumidos.
*Las consultas que contienen la cláusula GROUP BY se denominan consultas agrupadas y solo devuelven una sola fila para cada elemento agrupado.

Sin contar las funciones de agregación, todo lo que esta en el SELECT debe estar en el GROUP BY

HAVING

la directriz HAVING cumple la misma función que un WHERE, pero —¡ojo!— esta solo se va a poder usar en conjunto con las funciones de agregación para filtrar datos agregados. Es importante tener en cuenta esto porque para cualquier otro escenario la herramienta que tendremos que utilizar es el WHERE.

HAVING permite la implementación de alias y funciones de agregación.

```
SELECT pais, COUNT(clienteID)
FROM clientes
GROUP BY pais
HAVING COUNT(clienteID) > 3;
```

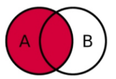
JOINS

INNER JOIN: es la opción predeterminada y nos devuelve todos los registros donde se cruzan dos o más tablas.

```
SELECT facturaid, apellido, nombre
FROM cliente
INNER JOIN factura
ON cliente.id=factura.cliente_id;
```

LEFT JOIN entre dos tablas devuelve todos los registros de la primera tabla (en este caso sería la tabla A), incluso cuando los registros no cumplan la condición indicada en la cláusula ON. Entonces **nos devuelve todos los registros donde se cruzan dos o más tablas. Incluso los registros de una primera tabla (A) que no cumplan con la condición.**

```
SELECT facturaid, apellido, nombre
FROM cliente
LEFT JOIN factura
ON cliente.id=factura.cliente_id;
```

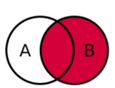


LEFT Excluding JOIN: Este tipo de LEFT JOIN nos devuelve únicamente los registros de una primera tabla (A), excluyendo los registros que cumplan con la condición indicada en la cláusula ON

```
SELECT facturaid, apellido, nombre
FROM cliente
LEFT JOIN factura
ON cliente.id=factura.cliente_id
WHERE factura.id IS NULL;
```

RIGHT JOIN: entre dos tablas devuelve todos los registros de la segunda tabla, incluso cuando los registros no cumplan la condición indicada en la cláusula ON.
Entonces, RIGHT JOIN **nos devuelve todos los registros donde se cruzan dos o más tablas. Incluso los registros de una segunda tabla (B) que no cumplan con la condición indicada en la cláusula ON.**

```
SELECT facturaid, apellido, nombre
FROM cliente
rRIGHT JOIN factura
ON cliente.id=factura.cliente_id;
```



RIGHT Excluding JOIN: Este tipo de RIGTH JOIN nos devuelve únicamente los registros de una segunda tabla (B), excluyendo los registros que cumplan con la condición indicada en la cláusula ON.

```
SELECT facturaid, apellido, nombre
FROM cliente
RIGHT JOIN factura
ON cliente.id=factura.cliente_id
WHERE cliente.id IS NULL;
```

VISTAS

elemento de la base de datos que facilita el acceso a los datos de las tablas. Básicamente, su función es guardar un SELECT.
*Una vista se ejecuta en el momento en que se invoca.
*Los nombres de las vistas deben ser únicos (no se pueden usar nombres de tablas existentes).
*Solamente se pueden incluir sentencias SQL de tipo SELECT.
*Los campos de las vistas heredan los tipos de datos de la tabla.
*El conjunto de resultados que devuelve una vista es inmodificable, a diferencia de lo que sucede con el conjunto de resultados de una tabla.

```
CREATE VIEW nombreVista AS
SELECT canciones.id
FROM canciones
WHERE canciones.id = 2;

Alteración: ALTER VIEW nombreVista AS ...;
Eliminación: DROP VIEW nombreVista;
Invocación: SELECT * FROM nombreVista;
```

INDICES

Un índice dentro de una base de datos es una estructura de datos que mejora la velocidad de las consultas, por medio de un identificador único de cada fila de una tabla, permitiendo un rápido acceso a los registros de una tabla en una base de datos.
VENTAJAS: mejora el rendimmiento de las consultas, ya que los datos existen en el propio indice. Puede mejorar el rendimiento si las consultas tienen agregaciones o joins.
DESVENTAJAS: las tablas donde se almacenan los indices ocupan espacio.
Consumen recursos, cada vez que actualizo, inserto o borro en una tabla indexada, debo actualizar todas las tablas de indice definidas sobre ellas. Hay que evitar crear demasiados índices en tablas que se actualizan con mucha frecuencia y procurar definirlos con el menor número de columnas posible.
Tipos de índices:
Simples :definidos por una sola columna
Compuesto: por varias columnas de la misma tabla)
Agrupado o Clustered: almacena los datos de las filas en orden. Solo se puede crear un único indice agrupado en una tabla de base de datos.
No agrupado: organiza los datos de forma aleatoria, pero especifica internamente un orden lógico.

```
CREATE INDEX “nombreIndice”
ON “nombreTabla” (nombreColumna);

DROP INDEX para eliminar.
ANALIZE INDEX para analizar y almacenar distribución de claves para una tabla.
```

BUENAS PRÁCTICAS

CREATE

- Intentemos utilizar VARCHAR en vez de TEXT.
- Evalúemos cuidadosamente el uso de CHAR y VARCHAR. El uso de CHAR y VARCHAR depende de si el campo en el que se va a usar varía mucho o no de tamaño. Esto para sopesar rendimiento de velocidad sobre rendimiento de almacenamiento. El motor de SQL procesa más rápido las columnas de longitud fija. Usemos CHAR para columnas de poca variación en longitud y VARCHAR para aquellas que no tienen una longitud estable o promedio.
- No usemos columnas con tipos de datos FLOAT, REAL o DATETIME como FOREIGN KEY.
- Usemos CONSTRAINT para mantener la integridad de los datos
- Evitemos claves primarias COMPUESTAS. Tengamos en cuenta que si esperamos que nuestra tabla con una clave primaria compuesta tenga millones de filas, el rendimiento de la operación CRUD está muy degradado. En ese caso, es mucho mejor usar una clave primaria ID simple que tenga un índice lo suficientemente compacto y establezca las restricciones de motor de bases de datos necesarias para mantener la singularidad.

SELECT

- Evitemos usar SELECT * FROM tabla. Aunque resulte fácil y cómodo usar el comodín (*) para traer todos los campos, este debe omitirse y en su lugar especificarse los campos que sean necesario traerse. El uso del comodín impide, además, un uso efectivo de forma eficiente de los índices.
- Anteponer el ALIAS de la tabla a cada columna. Especificar el alias de la tabla delante de cada campo definido en el SELECT ahorra tiempo al motor de tener que buscar a qué tabla pertenece el campo especificado.
- Evitemos en la medida de lo posible el uso de GROUP BY, DISTINCT y ORDER BY.
- Evadamos, siempre que sea posible, el uso de GROUP BY, DISTINCT y ORDER BY, dado que consume una elevada cantidad de recursos. Consideremos si es realmente necesario usarlo o si, por otro lado, se puede dejar el ordenamiento de los resultados a la aplicación que recibirá los datos.
-

WHERE

- Evitemos usar de Wilcards en LIKE como “%valor%” . En el caso de que se use la instrucción LIKE, no usemos el comodín “%” al inicio de la cadena a buscar. Esto debido a que si se aplica, la búsqueda tendría que leer todos los datos de la tabla o tablas involucradas para responder a la consulta. Se recomienda que existan al menos tres caracteres antes del comodín.
- Evitar usar IN en subconsultas, es mejor EXISTS. Promover el uso de EXISTS y NOT EXISTS, en lugar de IN y NOT IN.
- Intente no utilizar funciones dentro de las condiciones del WHERE.SQL no puede buscar eficientemente los registros cuando utiliza funciones, por ejemplo, de conversión, dentro de una columna. En las condiciones intente utilizar el formato de la columna original.

UNION

- Utilice UNION ALL para evitar un distinct implícito. En caso de usar la instrucción UNION y existiera la seguridad de que en los SELECT involucrados no se obtendrán registros duplicados, entonces, lo recomendable en este escenario es sustituir UNION por UNION ALL para evitar que se haga uso implícito de la instrucción DISTINCT ya que esta aumenta el consumo de recursos.

CRUD

- Usar SET NOCOUNT ON con operaciones CRUD. Usar SET NOCOUNT ON con operaciones CRUD para no contar el número de filas afectadas y ganar rendimiento sobre todo en tablas con muchos registros.

ORDEN DE PROCESAMIENTO DE UNA QUERY

1. FROM
2. ON
3. JOIN
4. WHERE
5. GROUP BY
6. HAVING
7. SELECT
8. DISTINCT
9. ORDER BY
10. LIMIT