# Curso de Java

## Introducción a las bases de datos relacionales

GOBIERNO DE ESPAÑA
MINISTERIO DE INDUSTRIA, COMERCIO Y TURISMO
EOI Escuela de organización industrial
Unión Europea
Fondo Social Europeo
Iniciativa de Empleo Juvenil
El FSE invierte en tu futuro
SISTEMA NACIONAL DE GARANTÍA JUVENIL

# Base de datos relacional

- **Una base de datos está compuesta por tablas que se relacionan entre sí**
  - Las tablas tienen un nombre que las identifica
  - Estas tablas se componen de columnas (campos) que identifican los datos almacenados en la tabla. Cada columna tiene un nombre y un tipo de datos asignado
  - Los datos están organizados en las tablas en filas o registros
- **Ejemplos de BBDD Relacionales son Oracle, PostgreSQL, MS SQL Server, MariaDB, MySQL, DB2, ...**

# Instalación de MariaDB para pruebas

- **MariaDB tiene raíces comunes con MySQL y un desarrollo más activo. Son compatibles entre sí para casi todo.**
- **La forma más rápida de instalar y ejecutar MariaDB es descargando la aplicación XAMPP**
  - https://www.apachefriends.org/es/download.html
- **Con marcar MySQL (instala MariaDB) y PHPMyAdmin para la instalación es suficiente**

# Tipos de datos (MySQL/MariaDB)

- **Tipos de datos que se le puede asignar a una columna de una tabla**

| Tipo | Denominación | Muestra |
|---|---|---|
| Entero | INT(N) | INT(25) |
| Decimal | DECIMAL(N,D) | DECIMAL(15,3) |
| Booleano | BOOL | BOOL |
| Fecha | DATE | DATE |
| Fecha y hora | DATETIME | DATETIME |
| Fecha y hora automática | TIMESTAMP | TIMESTAMP |
| Hora | TIME | TIME |
| Año | YEAR(D) | YEAR(10) |
| Cadena de longitud fija | CHAR(N) | CHAR(3) |
| Cadena de longitud variable | VARCHAR(N) | VARCHAR(110) |
| Bloque de texto de gran longitud variable | BLOB | BLOB |

# Clave Primaria

- **La clave primaria es un campo de la tabla que identifica cada fila de manera única (sin nulos)**
  - Ejemplo: DNI o email para identificar personas, referencia de un producto, número de habitación en un hotel, etc.
  - Las claves primarias pueden ser compuestas (varios campos). En este caso no se pueden repetir la combinación de los mismos
    - Ejemplo: En un cine, una butaca puede identificarse por número de sala y número de asiento

# Clave Ajena

- **La clave ajena o foránea es una o varias columnas de una tabla que referencian una clave primaria de otra tabla**
  - De esta manera creamos una relación entre las tablas
  - El tipo de datos debe coincidir en clave ajena y primaria
  - Puede admitir valores nulos (depende del diseño de la BBDD)
  - En caso de apuntar a una clave primaria compuesta, la clave ajena debe ser compuesta igualmente (mismos tipos de datos)

# Claves únicas

- **Una clave única o alternativa es aquella que no admite valores repetidos**
  - Por ejemplo, aunque identifiquemos a una persona por su DNI (Clave primaria), podemos establecer el campo email como único para evitar valores repetidos.
  - Se denomina clave alternativa porque podría hacer el papel de clave primaria en la práctica (pero se ha escogido otra)

# Índices

- **Un índice se asocia con una o varias columnas**
  - Se crea una estructura interna ordenada de los valores de dicho índice
  - Permite acelerar las búsquedas cuando se usa el campo en las condiciones
  - Añade coste a la hora de insertar/modificar
  - Recomendado: solo campos que participen mucho en las búsquedas (y cambien poco de valor)

# Valores nulos

- **Las claves primarias y únicas por defecto no admiten valores nulos**
- **Para el resto de columnas se puede establecer si admiten valores nulos (NULL) o no (NOT NULL)**

# Valores por defecto

- **Los campos pueden tener valores por defecto**
  - Al insertar un nuevo registro, si no se le da valor al campo, se le asigna por defecto
  - Se establece con la palabra DEFAULT seguida del valor en la creación de la tabla → DEFAULT 0
  - Para los tipos fecha DATETIME o TIMESTAMP, se puede usar el valor especial por defecto CURRENT_TIMESTAMP, que se traduce por la fecha en el momento de insertar
    - En este caso también se puede indicar que al actualizar (UPDATE) el registro se actualice automáticamente la fecha
    - DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP

# Creación de tablas

- Las tablas se crean con la instrucción CREATE TABLE

```
CREATE TABLE usuario (
   id INT UNSIGNED NOT NULL AUTO_INCREMENT ,
   nombre VARCHAR(150) NOT NULL ,
   email VARCHAR(250) NOT NULL ,
   edad SMALLINT NOT NULL ,
   fecha_registro TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ,
   puntuacion DECIMAL(5,2) NOT NULL DEFAULT 0,
   PRIMARY KEY (`id`),
   UNIQUE (`email`)
)
```

# Creación de tablas

```
CREATE TABLE usuario (
    id INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(150) NOT NULL ,
    email VARCHAR(250) NOT NULL UNIQUE,
    edad SMALLINT NOT NULL ,
    fecha_registro TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ,
    puntuacion DECIMAL(5,2) NOT NULL DEFAULT 0,
)
```

# Borrado de tablas

- **Las tablas se borran con la instrucción DROP TABLE**
  - DROP TABLE usuarios

# Cambiar propiedades campos

- **Se puede cambiar las propiedades de un campo de la tabla a posteriori (siempre que no entre en contradicción con los datos almacenados)**
- **Se utilizar la instrucción ALTER TABLE**
  - Ejemplo: Queremos que la puntuación pueda ser nula

```
ALTER TABLE usuario CHANGE puntuacion puntuacion DECIMAL(5,2) NULL;
```

# Inserción de datos (INSERT)

- **La inserción de datos se hace con la instrucción insert**
  - INSERT INTO tabla (campo1, campo2, campo3) VALUES (valor1, valor2, valor3)
  - Si especificamos todos los valores de los campos (en orden) podemos omitir los nombres de los mismos

```
INSERT INTO usuario VALUES (NULL, 'Pedro', 'pedro@email.com', '35', current_timestamp(), '56,70');
INSERT INTO usuario (nombre, email, edad, puntuacion) VALUES ('Pedro', 'pedro@email.com', '35', '56,70');
```

# Consulta de datos (SELECT)

- **Para consultar los registros de una tabla se utiliza la instrucción SELECT**
  - SELECT campo1, campo2, campo3 FROM tabla
  - SELECT * FROM tabla
- **Se pueden asignar alias a la tablas**
  - SELECT t.campo1, t.campo2, t.campo3 FROM tabla t

# Consultas con condiciones (SELECT … WHERE)

- **Se puede añadir la clausula WHERE con condiciones para la búsqueda**
  - Los valores tipo cadena van entre comillas simples
  - Conectores → and, or
  - Comparadores → = (igual), <> (diferente), != (diferente), > (mayor) , …
  - Comodines en cadenas (%) → nombre LIKE = '%ma%'
  - SELECT * FROM persona WHERE edad < 18 and puntuación > 5

# Ordenar resultados

- **Se puede usar la instrucción ORDER BY para establecer la columna por la cual se ordenan los resultados**
  - SELECT * FROM persona WHERE edad < 18 ORDER BY edad
- **Si usamos varias columnas para ordenar se separan por comas en orden de prioridad**
  - SELECT * FROM persona ORDER BY apellido1,apellido2,nombre
- **Por defecto ordena de menor a mayor. Para ordenar de mayor a menos usamos la palabra DESC al final**
  - SELECT * FROM persona WHERE edad < 18 ORDER BY edad DESC

# Columnas calculadas

- **En el apartado SELECT se pueden crear nuevas columnas calculadas a partir de otras**
  - Los resultados se muestran con la posición de la columna como el nombre de la misma. Se puede establecer un alias con la palabra AS
  - SELECT ref,nombre, precio, precio*0,21 AS precioIVA FROM producto

# Calculos de totales

- **Se pueden establecer consultas donde el resultado sea una función que calcule un total a partir de una columna. Las operaciones que podemos realizar son:**
  - SUM() calcula el total de una columna.
  - AVG() calcula el valor promedio de una columna.
  - MIN() encuentra el valor más pequeño en una columna.
  - MAX() encuentra el valor mayor en una columna.
  - COUNT() cuenta el número de valores en una columna.

# Ejemplos de cálculos

- **Precio total de todos los productos de la categoría 3**
  - SELECT SUM(precio) as total FROM producto WHERE categoria = 3
- **Nota media de los exámenes del alumno 2**
  - SELECT AVG(nota) as media FROM examen WHERE alumno = 2
- **Nota más alta de todos los exámenes**
  - SELECT MAX(nota) as notaMax FROM examen
- **Cuantos exámenes han sido aprobados**
  - SELECT COUNT(*) as numAprobados FROM examen WHERE nota >= 5

# Agrupando columnas calculadas (GROUP BY)

- **Si en lugar del total global, queremos obtener subtotales agrupados por el valor de una columna, podemos combinar las funciones anteriores con GROUP BY columna (esta columna debe aparecer en el SELECT)**
  - Media de notas de examenes por alumno
  - SELECT alumno, AVG(nota) AS media GROUP BY alumno

# Condiciones para agrupar (HAVING)

- **Para establecer condiciones similar al WHERE pero con valores calculados a partir de un GROUP BY (en lugar de registros individuales) usamos la clausula HAVING. Aquí vamos a utilizar generalmente valores calculados**
  - Media de los alumnos que hayan sacado al menos un 5 en todos sus exámenes
  - SELECT alumno, AVG(nota) AS media GROUP BY alumno HAVING MIN(nota) >= 5

# Actualización de datos (UPDATE)

- **Para actualizar los datos de uno o varios registros se usa la instrucción UPDATE**
  - UPDATE tabla SET campo1=valor1, campo2=valor2 WHERE condición
  - Se actualizan los campos de todos los registros que cumplan la condición
  - UPDATE usuarios SET nombre='Nuevo nombre’ WHERE id=1

# Borrado de datos (DELETE)

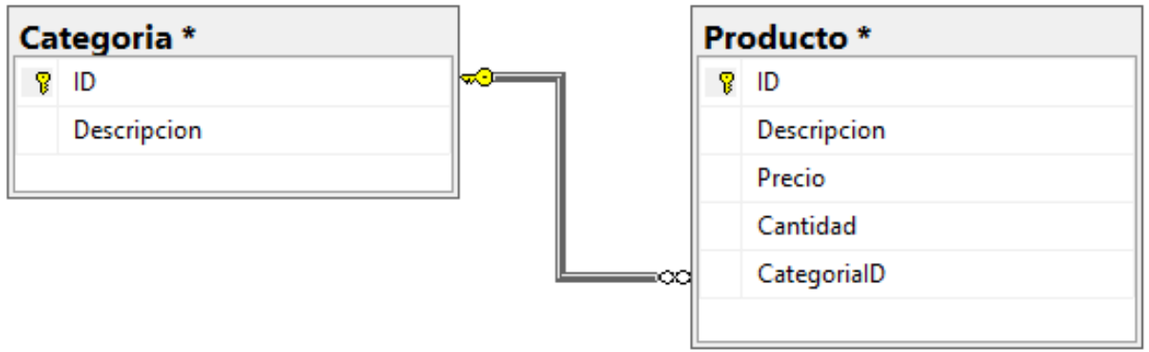
- **Para borrar registros de una tabla se usa la instrucción DELETE**
  - DELETE FROM tabla WHERE condición
  - Se borran todos los registros que cumplan la condición (¡Cuidado!)
  - DELETE FROM usuarios WHERE email = 'nadie@email.com'

# Relaciones

- **Tenemos varios tipos de relaciones entre 2 tablas**
  - Relación Uno a Muchos (1-N)
    - Es la relación más utilizada. La clave ajena de una tabla (muchos → se puede repetir) apunta a otra tabla (uno). Ejemplo: Producto → Categoría
  - Relación Uno a Uno (1-1)
    - Es una variante de la relación 1-N, donde la clave ajena no admite repetidos (clave primaria o única)
  - Relación Muchos a Muchos (N-M)
    - Es la más compleja. Requiere de una tabla intermedia, cuya clave primaria es la composición de 2 claves ajenas y cada una apunta a una tabla.
    - Ejemplo: Concierto ← **Asiste** → Persona

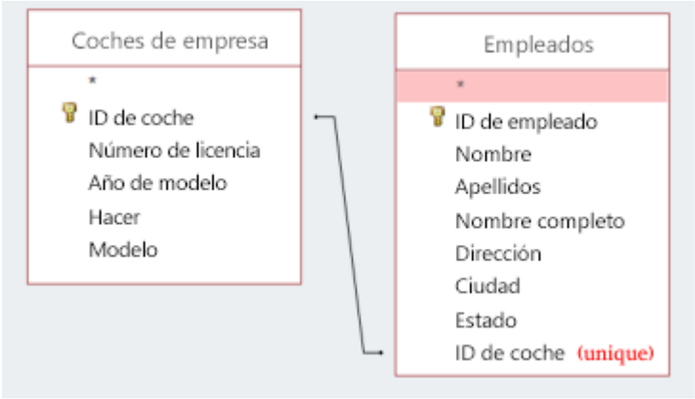
# Relación Uno a Muchos (1-N)

- **La clave ajena está en la tabla Muchos (ej: en jugador apuntando a equipo). La clave ajena siempre es del mismo tipo que la clave primaria de la tabla a la que apunta**
  - Opcionalmente podemos tener claves ajenas que admitan nulos, dejando registros sin relacionar. Ejemplo: Un jugador que en este momento no pertenece a ningún equipo



# Relación Uno a Uno (1-1)

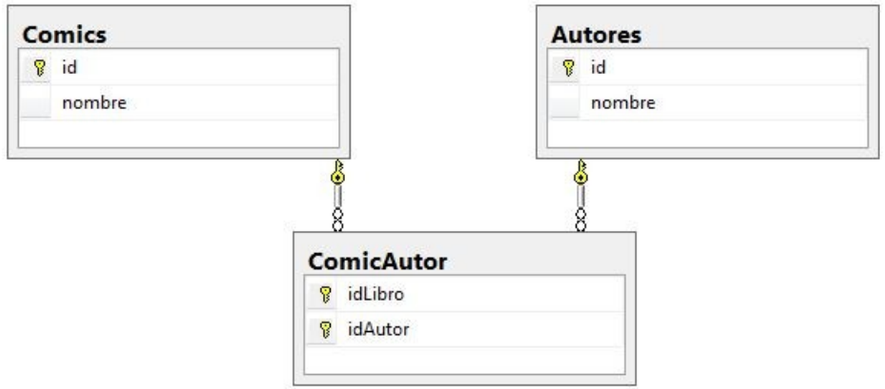
- **Es una versión de la relación 1-N donde la clave ajena no admite valores repetidos (clave única o primaria. Una clave primaria puede ser clave ajena al mismo tiempo)**
  - La clave ajena puede estar en cualquiera de las 2 tablas, siempre que apunte a la clave primaria de la otra tabla
  - Ejemplo: Un equipo solo puede tener un entrenador. Podemos tener un campo en la tabla equipo llamada entrenador que apunte a la tabla entrenador y que no admita repetidos (un entrenador no puede entrenar a 2 equipos). También se puede plantear al revés.



# Relación Muchos a Muchos (N-M)

- **Necesitamos una tabla intermedia para relacionar 2 tablas entre sí**
- **Esta tabla intermedia tendrá una clave primaria compuesta con 2 columnas y cada una es clave ajena a una de las tablas relacionadas**

- La tabla intermedia puede tener más columnas (información de la relación). Ej: Usuario asiste a Concierto, que guardaría también el número de entradas compradas, fecha, etc.
- Se puede interpretar también (sobre todo con columnas adicionales), que tenemos 2 relaciones 1-N entre la tabla intermediaria y las 2 tablas principales.



# Cruce de datos relacionados (JOIN)

- **Se pueden cruzar datos de varias tablas (normalmente relacionadas) utilizando la instrucción JOIN**
  - Esta instrucción tiene muchas variantes
  - https://es.wikipedia.org/wiki/Sentencia_JOIN_en_SQL
  - Muchas veces se utiliza el INNER JOIN implícito igualando la clave ajena de una tabla con la primaria de la otra (WHERE)
    - SELECT c.nombre as cat_nombre, p.nombre as prod_nombre, p.precio FROM categoria c, producto p WHERE p.categoria = categoria.id

# Unión de resultados

- **Se pueden unir los resultados de una consulta SELECT con otra diferente utilizando la instrucción UNION**
  - Ambas consultas deben devolver los mismos campos en el resultado (mismo nombre y tipo de campo aunque sean sobre tablas diferentes)
  - SELECT nombre FROM cliente UNION SELECT nombre FROM proveedor

# SELECT anidadas (subconsultas)

- **Se pueden utilizar los resultados de una consulta SELECT como valores para otra consulta.**
  - Se usa mucho en la condición IN de la clausula WHERE
  - SELECT productos WHERE categoria IN (1,2,3,4)
  - SELECT productos WHERE categoria IN (SELECT id FROM categoria WHERE activa = 1)