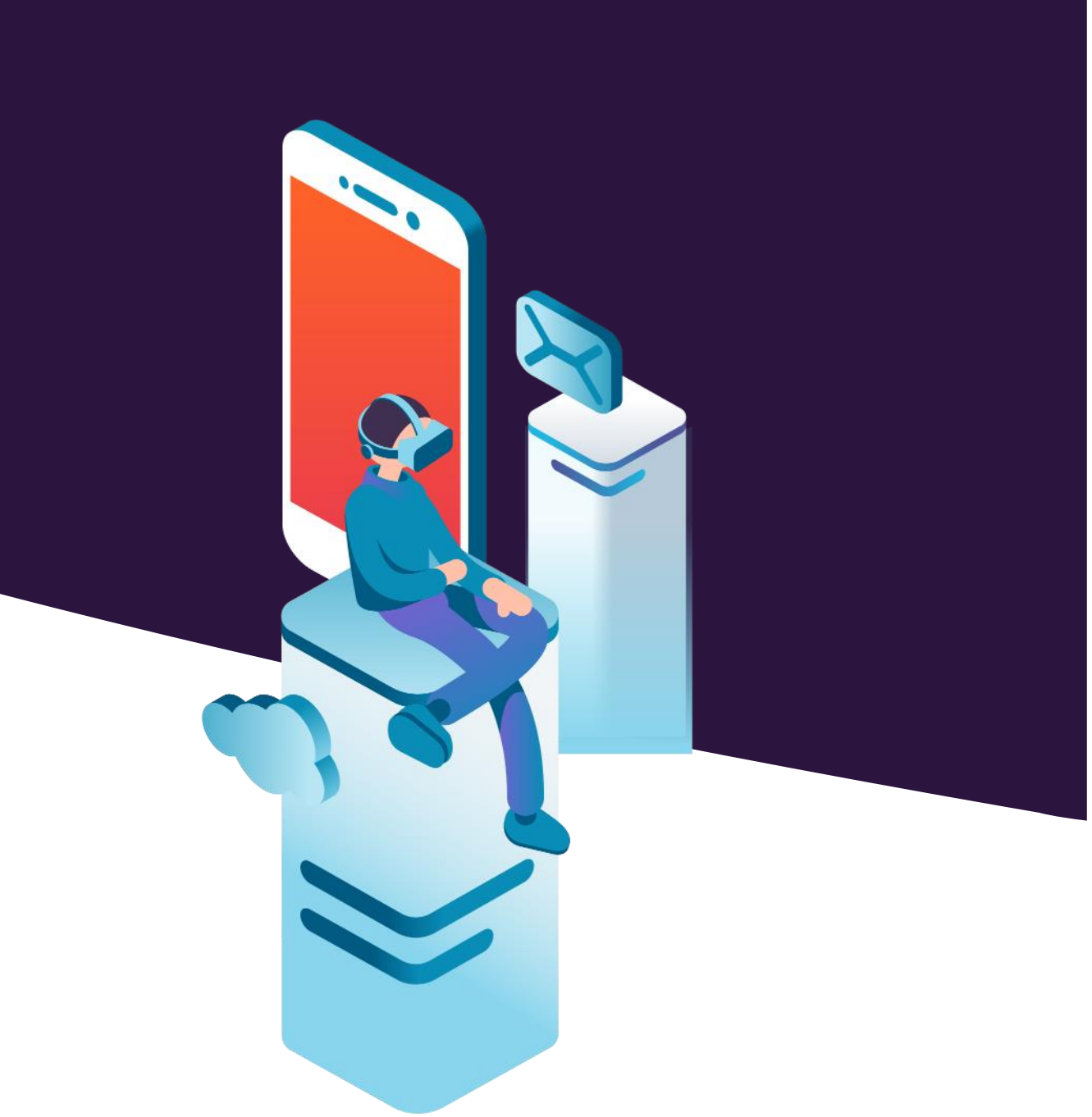
MySQL Fundamentals

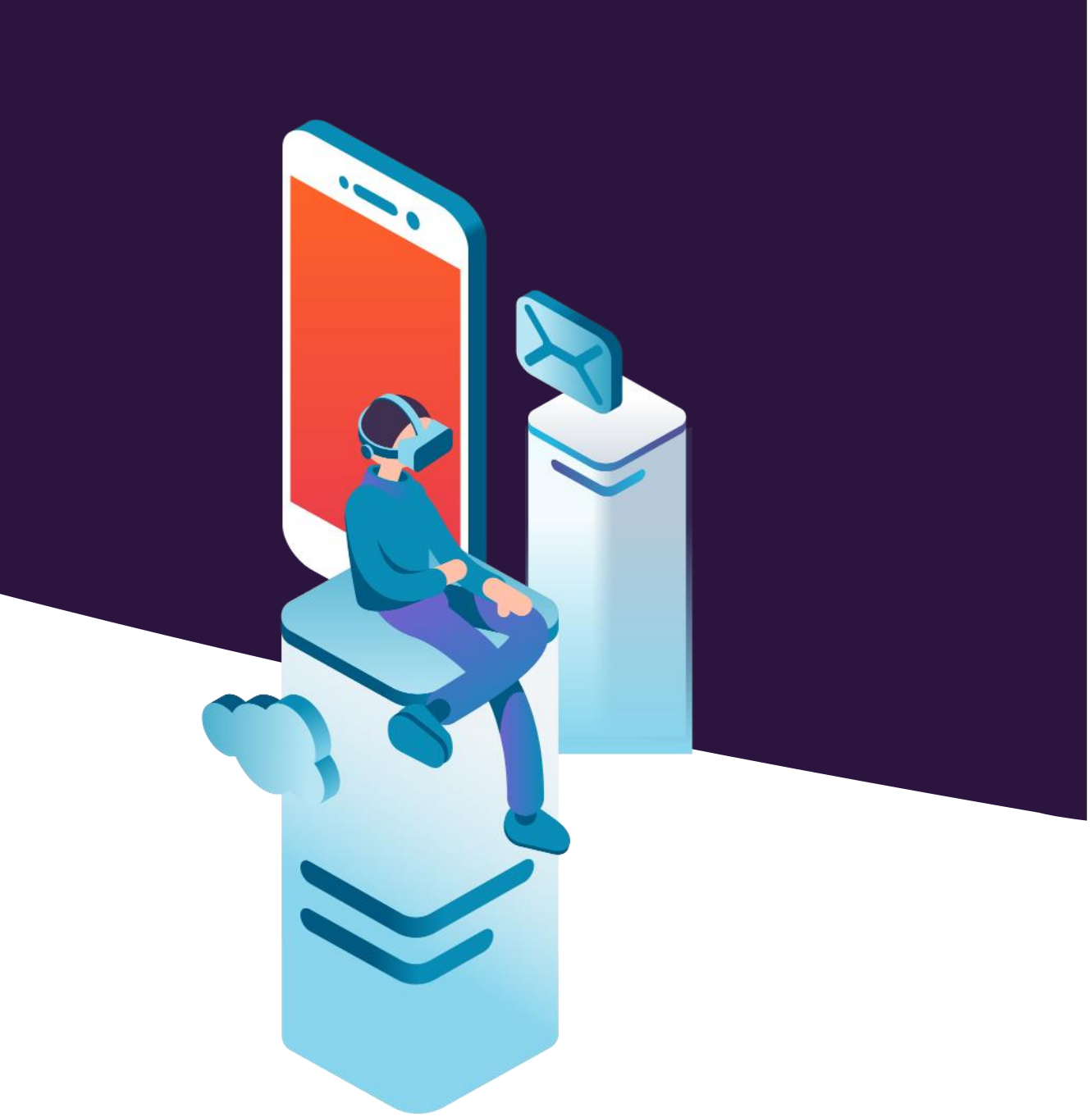
Introducción a las bases de datos MySQL

Toni Miquel Llull Amengual
Computer Science (IT) & Testing Engineer
+3 años en Sogeti | SogetiLabs
antoni-miquel.llull-amengual@sogeti.com



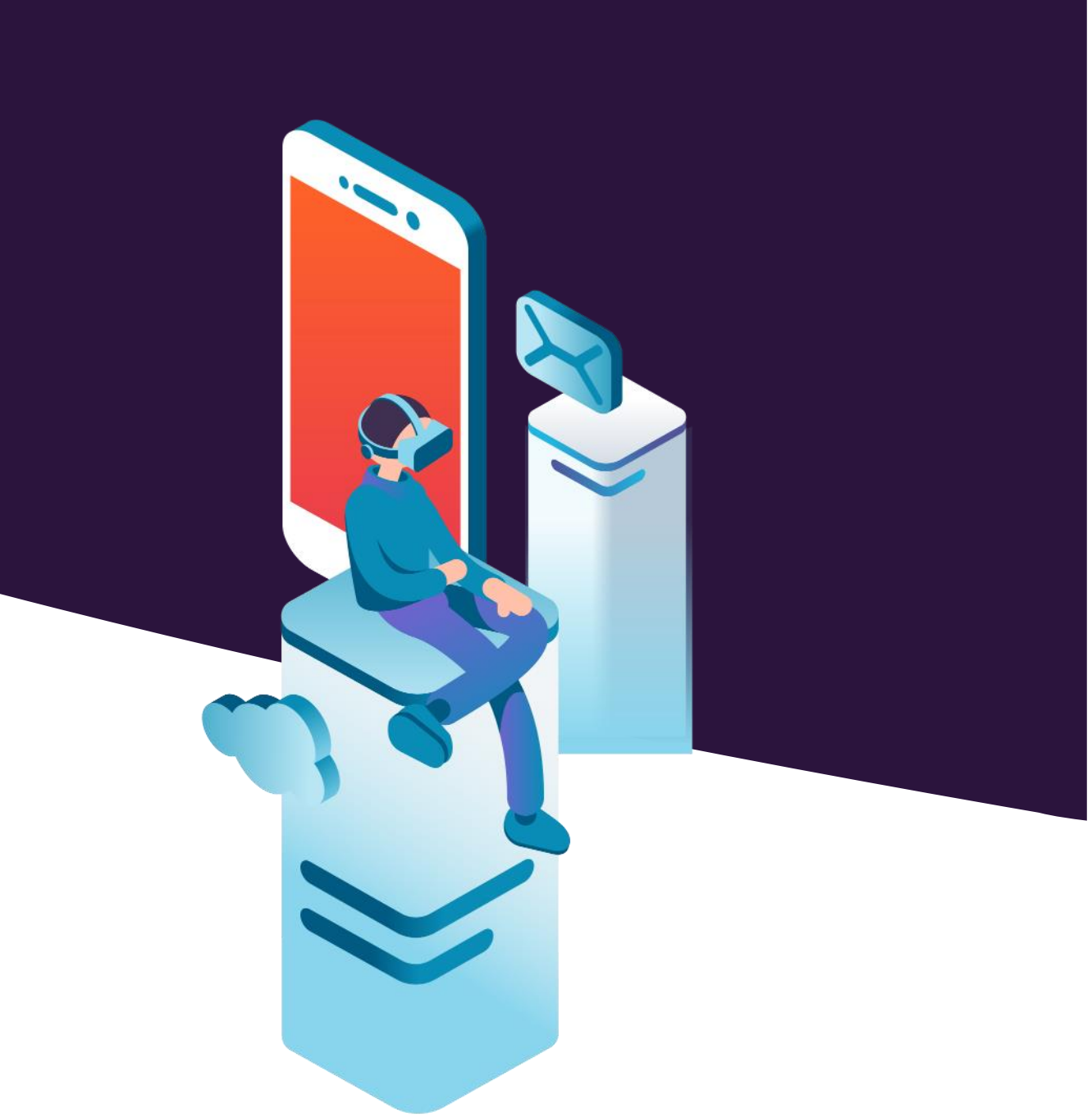
Índice

- ¿Qué es una base de datos?
- Qué es SQL
- Bases de datos relacionales y no relacionales
- Diseño básico de una base de datos relacional
  - Tablas
  - Columnas
  - Filas
  - Tipos de datos
- Normalización de la base de datos
  - Primay Key
  - Foreign Key
- CRUD



Índice

- Sentencias e instrucciones básicas
  - CREATE
  - INSERT
  - SELECT
  - WHERE
  - LIMIT
  - UPDATE
  - DELETE
  - DROP/TRUNCATE
  - FOREIGNKEY



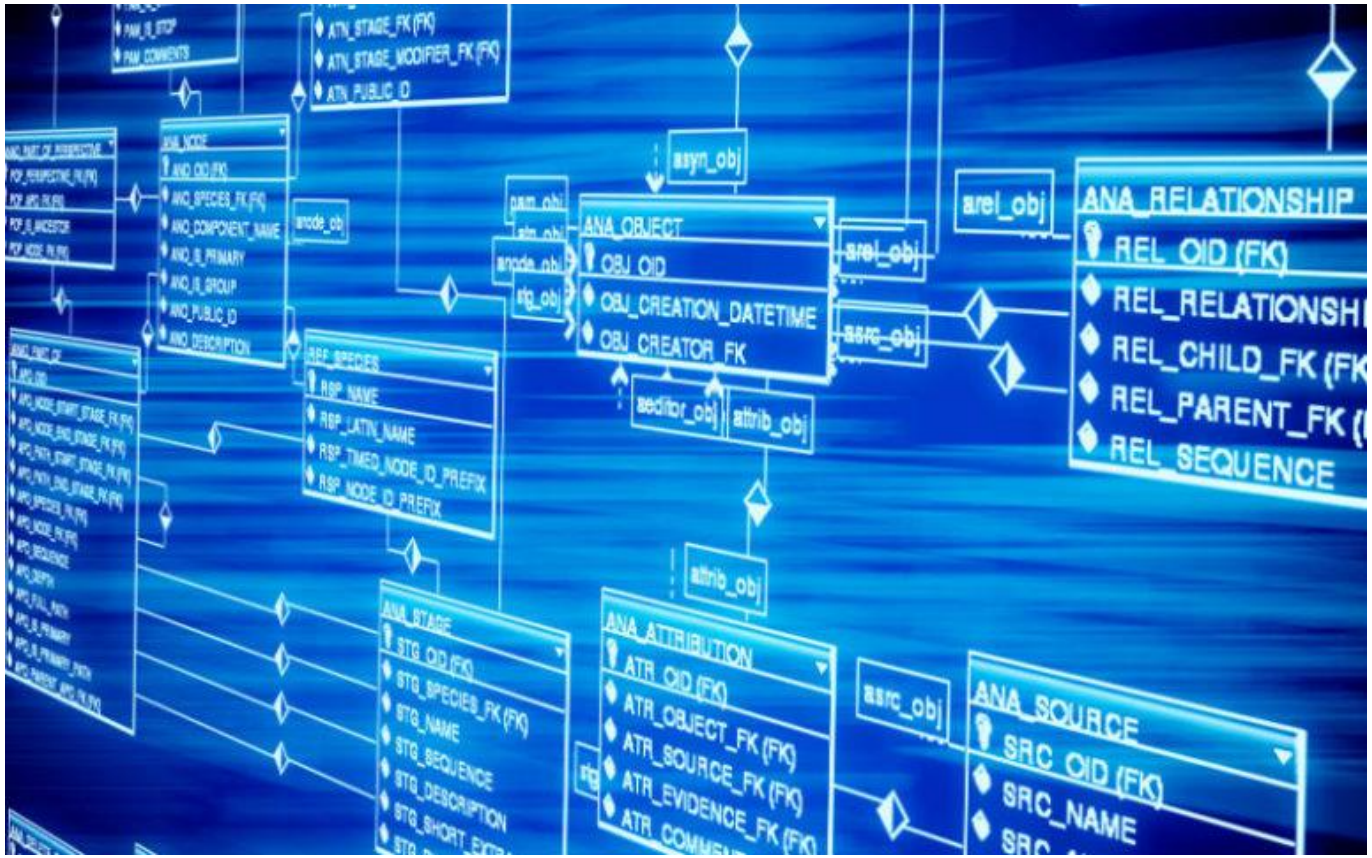
Índice

- Sentencias e instrucciones complementarias
  - ORDER BY
  - AND/OR
  - IN/NOT IN
  - BETWEEN
  - LIKE
  - Alias/AS
  - JOINS
  - DISTINCT
  - COUNT
  - GROUP BY
  - MAX/MIN
  - SUM/AVG
  - SELECT anidado
  - HAVING

¿Qué es una base de datos?

¿Qué es una base de datos?

Podemos definir una base de datos como una **colección de datos estructurada** que facilita su manipulación y manejo



Pregunta: ¿Un Excel puede considerarse una base de datos?

¿Qué es SQL?

¿Qué es SQL?

- SQL es el acrónimo de "**S**tructured **Q**uery **L**anguage"

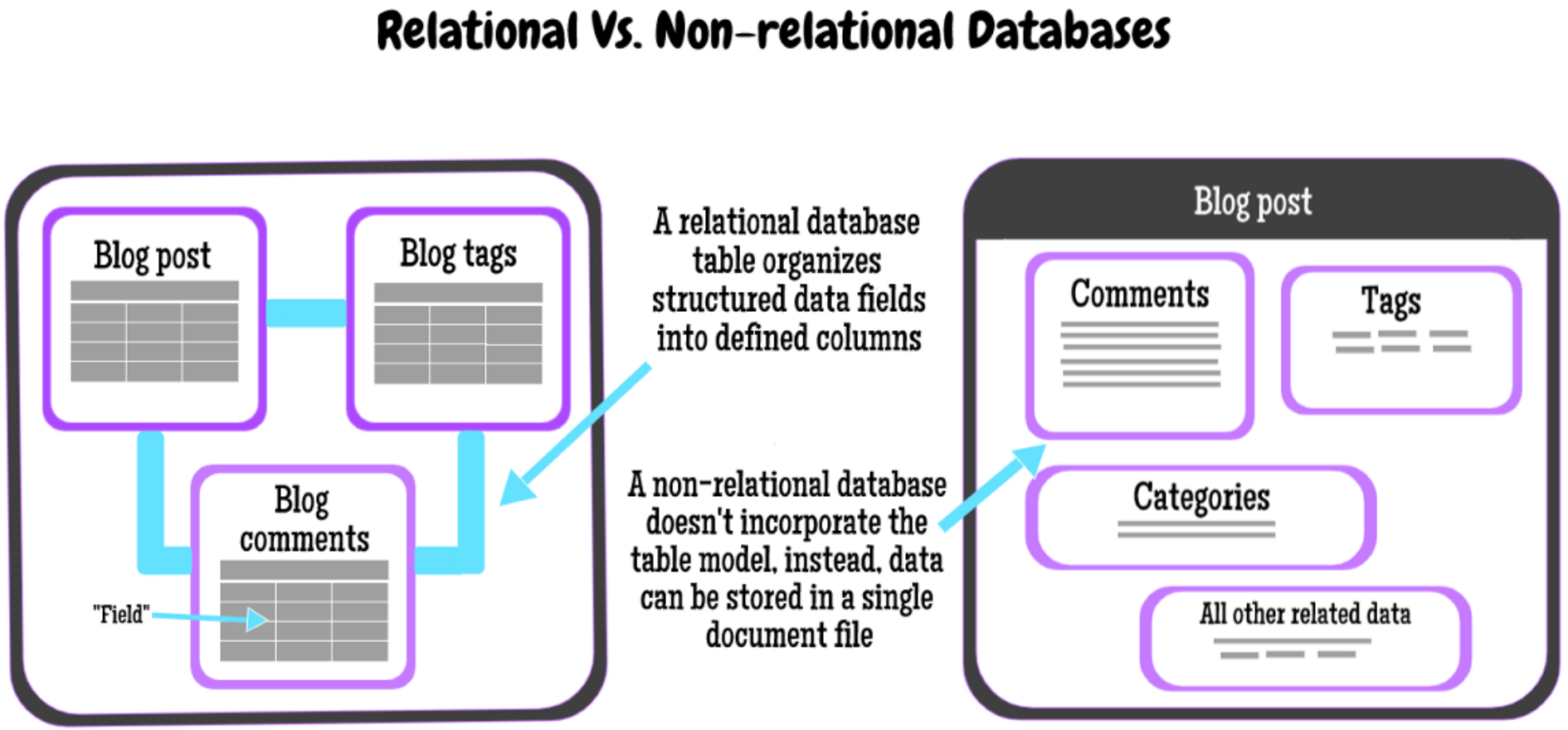
- Suele pronunciarse directamente diciendo el nombre de cada letra (S-Q-L) o, como los más puristas, "See-Quel"

- Es el lenguaje usado en **bases de datos relacionales** y permite realizar las sentencias y consultas necesarias para poder trabajar directamente con ellas

- Herramientas como Jira usan un lenguaje propio llamado JQL, derivado de SQL

Bases de datos relacionales y no relacionales

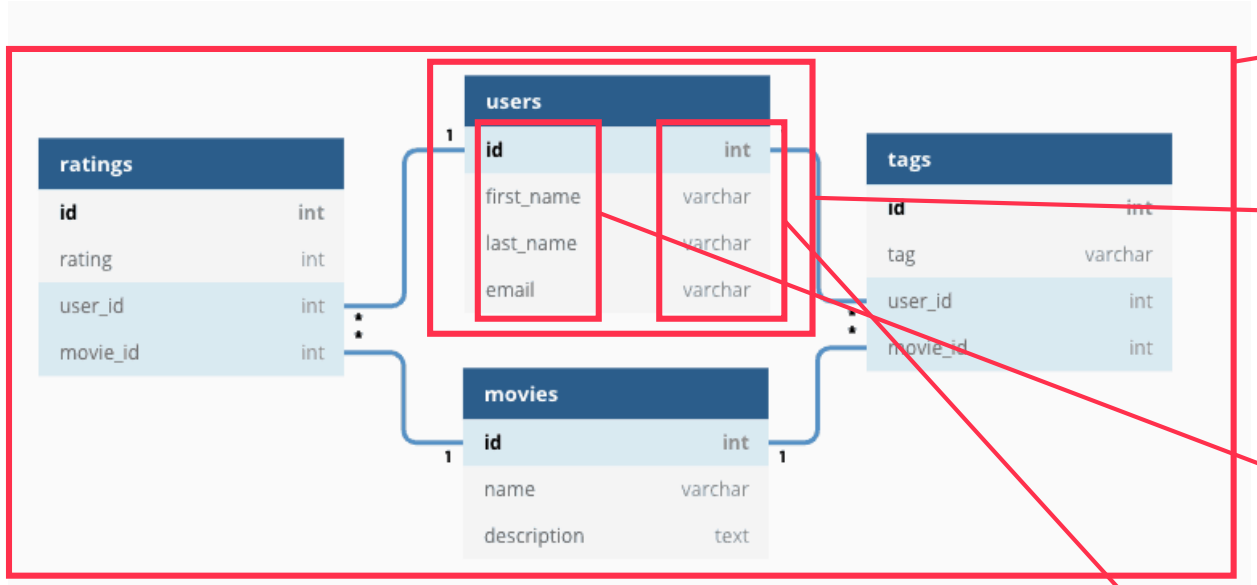
Bases de datos relacionales y no relacionales



Bases de datos relacionales y no relacionales

Diseño básico de una base de datos relacional

Diseño básico de una base de datos relacional



id	First_name	Last_name	email
1	Toni Miquel	Llull	a@a.com
2	Juan	Álvarez	b@b.com
3	Pepe	Viyuela	c@c.com
…	…	…	…

- **Base de datos**: todo el conjunto de nuestros datos
- **Tablas**: subconjunto de nuestra base de datos. Las entradas de cada una de las tablas deben tener todas los mismos atributos
- **Columnas**: los atributos de cada una de las tablas
- **Filas**: cada una de las entradas en cada una de nuestras tablas
- **Tipos de datos**: tipo de dato de cada uno de los atributos

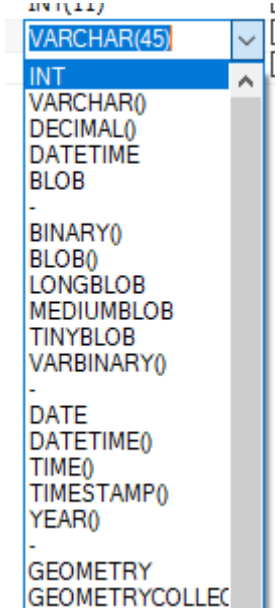
Diseño básico de una base de datos relacional

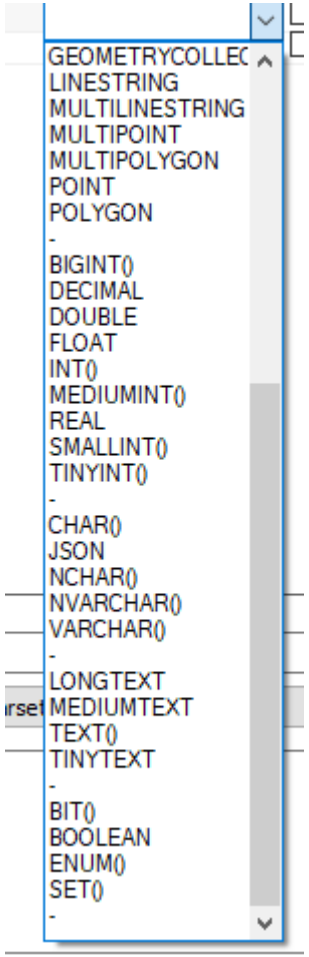
Tipos de datos

Existen una gran cantidad de tipos de datos que podemos guardar en nuestra base de datos, por lo que no los explicaremos todos.

En la mayoría se suele indicar el valor máximo que pueden contener (tamaño máximo)

Por ejemplo, un tipo “int(11)” quiere decir que podremos guardar un entero de 11 dígitos, mientras que en un “varchar(255)” nos permitirá guardar una cadena de hasta 255 caracteres





Normalización de una base de datos

Normalización de una base de datos relacional

Podemos definir “normalización” como el proceso de estructuración de nuestra base de datos para evitar redundancia en nuestros datos y poder mantener de una forma más sencilla su integridad.

Esta normalización se compone de varias fases, que veremos de forma simplificada con un ejemplo:

REGISTER					
id	**first_name**	**last_name**	**email**	**movie**	**description**
1	Toni Miquel	Llull	a@a.com	ESDLA	Un elfo en un marrón
2	Toni Miquel	Llull	a@a.com	ESDLA2	Un elfo en un marrón más grande
…	…	…	…	…	

Imaginemos que queremos guardar la información de un videoclub sobre quién alquila qué películas

Normalización de una base de datos relacional

REGISTER					
id	**first_name**	**last_name**	**email**	**movie**	**description**
1	Toni Miquel	Llull	a@a.com	ESDLA	Un elfo en un marrón
2	Toni Miquel	Llull	a@a.com	ESDLA2	Un elfo en un marrón más grande
…	…	…	…	…	

Separar en 2 tablas

USERS			
id	**first_name**	**last_name**	**email**
1	Toni Miquel	Llull	a@a.com
2	Juan	Álvarez	b@b.com
3	Pepe	Viyuela	c@c.com
…	…	…	…

MOVIES		
id	**movie**	**description**
1	ESDLA	Un elfo en un marrón
2	ESDLA2	Un elfo en un marrón más grande
…		

Normalización de una base de datos relacional

REGISTER

id	first_name	last_name	email	movie	description
1	Toni Miquel	Llull	a@a.com	ESDLA	Un elfo en un marrón
2	Toni Miquel	Llull	a@a.com	ESDLA2	Un elfo en un marrón más grande
…	…	…	…	…	

Estas serían PrimaryKeys

USERS

id	first_name	last_name	email
1	Toni Miquel	Llull	a@a.com
2	Juan	Álvarez	b@b.com
3	Pepe	Viyuela	c@c.com
…	…	…	…

MOVIES

id	movie	description
1	ESDLA	Un elfo en un marrón
2	ESDLA2	Un elfo en un marrón más grande
…		

Normalización de una base de datos relacional

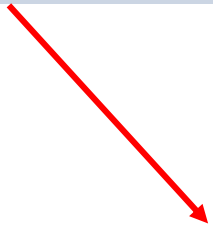
- Llamamos **PrimaryKey** a la columna que nos determina cada una de las final de forma inequívoca.

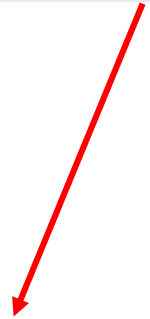
- Esta columna NO puede contener ningún valor repetido, no puede ser nulo y no puede haber más de una PrimaryKey, aunque sí puede haber PrimaryKey combinadas

Normalización de una base de datos relacional

USERS			
id	**first_name**	**last_name**	**email**
1	Toni Miquel	Llull	a@a.com
2	Juan	Álvarez	b@b.com
3	Pepe	Viyuela	c@c.com
…	…	…	…

MOVIES		
id	**movie**	**description**
1	ESDLA	Un elfo en un marrón
2	ESDLA2	Un elfo en un marrón más grande
…		





Crear una tabla intermedia

REGISTER		
id	**user_id**	**movie_id**
1	1	1
2	1	2
…		

Normalización de una base de datos relacional

USERS			
id	**first_name**	**last_name**	**email**
1	Toni Miquel	Llull	a@a.com
2	Juan	Álvarez	b@b.com
3	Pepe	Viyuela	c@c.com
…	…	…	…

MOVIES		
id	**movie**	**description**
1	ESDLA	Un elfo en un marrón
2	ESDLA2	Un elfo en un marrón más grande
…		

Estas serían ForeignKeys

REGISTER		
id	**user_id**	**movie_id**
1	1	1
2	1	2
…		

Normalización de una base de datos relacional

- Llamamos **ForeignKey** a las columnas que referencian a columnas de otras tablas.

- Las columnas referenciadas deben ser o bien PrimaryKey o Unique, y podemos tener más de una ForeignKey

Normalización de una base de datos relacional

- Llamamos **Unique** a las columnas cuyo contenido no puede estar repetido.

- Son similares a las PrimaryKey, pero podemos tener más de una columna Unique.

- La diferencia es que, por norma general, la PrimaryKey no debería cambiar nunca mientras que un campo Unique sí puede hacerlo

CRUD

Llamamos **CRUD** al conjunto de acciones básicas que permiten trabajar con una base de datos y se compone de:

- **C**REATE → Crear
- **R**EAD → Leer o consultar
- **U**PDATE → Actualizar
- **D**ELETE → Eliminar

Sentencias e instrucciones básicas

CREATE

Nos permite crear bases de datos

```
CREATE DATABASE dbname;
```

```
CREATE DATABASE IF NOT EXISTS dbname;
```

Añadir IF NOT EXISTS nos ayuda a “saltar” errores dentro de una secuencia de comandos

CREATE

Nos permite crear tablas

```
CREATE TABLE [IF NOT EXISTS] dbname.tablename (fieldname dataType [optional parameters]);
```

fieldname: nombre del atributo
dataType: tipo de dato del atributo
optional parameters: otros parámetros que pueda tener nuestro campo

Para añadir más de un campo se separan con ,

Los "Optional Parameters" más usados son:

NULL
NOT NULL
AUTO_INCREMENT
DEFAULT

```
USE dbname;

CREATE TABLE [IF NOT EXISTS] tablename (fieldname dataType [optional parameters]);
```


CREATE

Nos permite crear tablas

```
CREATE TABLE [IF NOT EXISTS] dbname.tablename (fieldname dataType [optional parameters]);
```

```
USE dbname;

CREATE TABLE [IF NOT EXISTS] tablename (fieldname dataType [optional parameters]);
```

```
CREATE TABLE mydb.users (
  _id INT NOT NULL AUTO_INCREMENT,
  name VARCHAR(45) NULL,
  lastname VARCHAR(45) NULL,
  email VARCHAR(45) NOT NULL
);
```


PrimaryKeys

Para definir una PrimaryKey al crear una tabla debemos indicarlo al final de la sentencia:

```
CREATE TABLE dbname (
  _id INT NOT NULL AUTO_INCREMENT,
  name VARCHAR(45) NULL,
  lastname VARCHAR(45) NULL,
  email VARCHAR(45) NOT NULL,
  PRIMARY KEY (_id)
);
```

Es muy importante que siempre definamos una PrimaryKey en todas nuestras tablas

CREATE

Ejercicio: vamos a crear una nueva base de datos llamada “mydb” con las siguientes tablas

trabajadores
_id INT nombre VARCHAR apellidos VARCHAR email VARCHAR

PrimaryKey: _id

clientes
_id INT nombre VARCHAR apellidos VARCHAR email VARCHAR

PrimaryKey: _id

INSERT

Nos permite añadir nuevas entradas a la base de datos

```
INSERT INTO `table_name`(column_1,column_2,...) VALUES (value_1,value_2,...);
```

En esta query tenemos 2 partes:

- **INSERT INTO**: indicamos en qué tabla queremos insertar una nueva entrada, seguido de los campos que vamos a incluir.
- **VALUES**: indicamos el valor de los campos que hemos indicado anteriormente

El orden de los campos no es relevante, siempre y cuando los valores coincidan con el tipo de dato que vamos a introducir

INSERT

Ejercicio: añadir las siguientes entradas a nuestras tablas y alguna más a vuestra elección

trabajadores		
Marco	Polo	marco@polo.com
Benito	Martín	benito@martin.com
Benito	Martín	benito@martin.com
…	…	…

clientes		
Pepe	Viyuela	pepe@viyuela.com
Pablo	Ruíz	pablo@ruiz.com
Benito	Martín	benito@martin.com
…	…	…

SELECT

Nos permite consultar el contenido de una tabla de la base de datos

```
SELECT * FROM tablename;
```

En esta query tenemos 2 partes:

- **SELECT ***: indicamos qué columna(s) queremos consultar. En este caso el * indica que queremos consultar todas las columnas
- **FROM tablename**: indicamos la tabla de la cual queremos hacer la consulta

El resultado sería que obtendríamos toda la tabla al completo. Si quisiéramos consultar sólo 2 columnas, las indicaríamos de la siguiente manera:

```
SELECT column1, column2 FROM tablename;
```

Ejercicio: consultar las tablas que acabamos de crear

WHERE

Nos permite ajustar más nuestra búsqueda especificando una condición

```
SELECT * FROM tablename WHERE column1 = 20;
```

El resultado sería que obtendríamos todas las filas y columnas de la tabla cuya columna “column1” sea igual a 20

Ejercicio: hacer alguna consulta con la instrucción “WHERE”

LIMIT

Nos permite limitar el número de filas que queremos obtener

```
SELECT * FROM tablename LIMIT 5;
```

El resultado sería que obtendríamos las 5 primeras filas de nuestra tabla, con todas sus columnas

Ejercicio: hacer alguna consulta con la instrucción “LIMIT”

SELECT, WHERE, LIMIT

Estas instrucciones se pueden combinar entre ellas de la siguiente manera

```
SELECT * FROM tablename WHERE column1 = 20 LIMIT 5;
```

El orden aquí es importante, pero como norma general y para que os acordéis lo básico es “SELECT FROM WHERE”. Las demás instrucciones suelen ir después de éstas.

Ejercicio: consultar las tablas combinando “SELECT”, “WHERE” y “LIMIT”

UPDATE

Nos permite modificar el contenido de una entrada en nuestra base de datos

```
UPDATE table_name SET column_name = new_value [WHERE condition];
```

En esta query tenemos 3 partes:

- **UPDATE**: indicamos en qué tabla se encuentra la fila que queremos modificar
- **SET**: indicamos la(s) columna(s) que se van a modificar indicando el nuevo valor
- **WHERE** (opcional): esta condición nos permite delimitar qué filas van a ser modificadas. Si no especificamos nada se modificará la columna indicada de la tabla entera

Ejercicio: modificar algunas entradas de nuestra base de datos

DELETE

Nos permite eliminar una entrada en nuestra base de datos

```
DELETE FROM table_name;
```

Ejercicio: probar a eliminar algunas entradas de vuestras tablas

DELETE

DELETE

Nos permite eliminar una entrada en nuestra base de datos

```
DELETE FROM table_name [WHERE condition];
```

En esta query tenemos 2 partes:

- **DELETE FROM**: indicamos de qué tabla queremos eliminar una (o varias) entrada(s)

- **WHERE** (opcional, pero no): esta condición nos permite delimitar qué filas van a ser eliminadas. **Si no especificamos nada se eliminará todo el contenido de la tabla entera**

DROP

Nos permite eliminar tanto bases de datos como tablas

```
DROP DATABASE dbname;
```

```
DROP TABLE dbname.tablename;
```

```
USE dbname;
DROP TABLE tablename;
```

TRUNCATE

Nos permite vaciar tablas de nuestra base de datos pero no las elimina

```
TRUNCATE TABLE dbname.tablename;
```

```
USE dbname;
TRUNCATE TABLE tablename;
```

Ejercicio: probad estas 2 instrucciones, una con cada tabla

ForeignKeys

Ejercicio: vamos a crear una nueva base de datos llamada “videoclub” con las siguientes tablas

clientes
_id INT nombre VARCHAR apellidos VARCHAR email VARCHAR

PrimaryKey: _id
Unique: email

peliculas
_id INT nombre VARCHAR

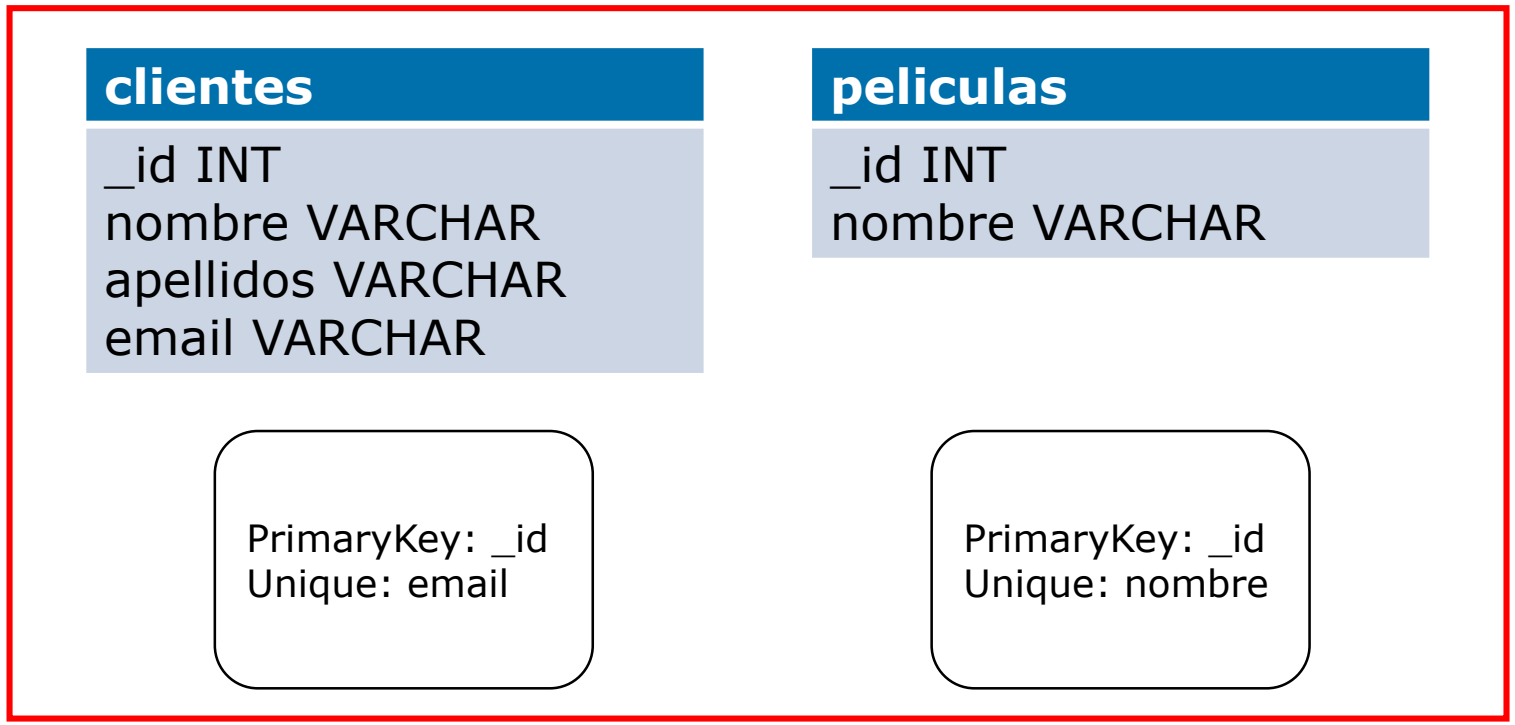
PrimaryKey: _id
Unique: nombre

alquiler
_id INT cliente_id VARCHAR pelicula_id VARCHAR fecha DATE

PrimaryKey: _id
ForeignKey: cliente_id, película_id

ForeignKeys

Ejercicio: vamos a crear una nueva base de datos llamada “videoclub” con las siguientes tablas



Vamos a crear primero estas dos tablas

ForeignKeys

Ejercicio: vamos a crear una nueva base de datos llamada “videoclub” con las siguientes tablas

```
CREATE TABLE `videoclub`.`alquiler` (
  `_id` INT NOT NULL AUTO_INCREMENT,
  `user_id` INT NOT NULL,
  `movie_id` INT NOT NULL,
  `date` DATE NOT NULL,
  PRIMARY KEY (`_id`),
  INDEX `user_id_idx` (`user_id` ASC),
  INDEX `movie_id_idx` (`movie_id` ASC),
  CONSTRAINT `user_id`
    FOREIGN KEY (`user_id`)
    REFERENCES `videoclub`.`users` (`user_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `movie_id`
    FOREIGN KEY (`movie_id`)
    REFERENCES `videoclub`.`movies` (`movie_id`)
    ON DELETE CASCADE
    ON UPDATE NO ACTION);
```

alquiler
_id INT cliente_id VARCHAR pelicula_id VARCHAR fecha DATE

PrimaryKey: _id
ForeignKey:
cliente_id,
película_id

ForeignKeys

Ejercicio: probar a realizar algunas inserciones en nuestras tablas para ver cómo funciona nuestra nueva base de datos.

Sentencias e instrucciones complementarias

Vamos a jugar a Pokémon

Importar la base de datos de Pokémon en Workbench

ORDER BY

Nos permite devolver el resultado ordenado

```
SELECT * FROM tablename WHERE column1 = 20 ORDER BY columnX;
```

Por defecto el orden es ASCENDENTE, pero se puede especificar añadiendo ASC o DESC al final

```
SELECT * FROM tablename WHERE column1 = 20 ORDER BY columnX ASC;
```

```
SELECT * FROM tablename WHERE column1 = 20 ORDER BY columnX DESC;
```

Ejercicios: Selecciona Pokémons aplicando ordenación en diferentes columnas

AND

Nos permite añadir más condiciones dentro de una instrucción WHERE

```
SELECT * FROM table WHERE field = data AND field2 = data2;
```

```
SELECT * FROM table WHERE field = data AND field2 > data2;
```

```
SELECT * FROM table WHERE field = data AND field2 <> data2;
```

Para que la sentencia se ejecute deben cumplirse todas las condiciones

Ejercicios: Selecciona Pokémons con los siguientes criterios:

- Su altura sea mayor a 50 y su experiencia base mayor a 10
- Su altura sea 2 y su peso igual a 20
- Su altura sea 2, su peso igual a 20 y su experiencia base diferente de 64

OR

Nos permite añadir más condiciones dentro de una instrucción WHERE

```
SELECT * FROM table WHERE field = data OR field2 = data2;
```

Para que la sentencia se ejecute basta que se cumpla una de las condiciones

Ejercicios: Selecciona Pokémons con los siguientes criterios:

- Su altura sea mayor a 50 o su experiencia base mayor a 10
- Su altura sea 2 o su peso igual a 20
- Su altura sea 2 o su peso igual a 20 o su experiencia base diferente de 64

IN

Nos permite definir una lista de posibles opciones dentro de la instrucción WHERE

```
SELECT * FROM table WHERE field IN (data, data2, data3);
```

Para que la sentencia se ejecute el valor del campo WHERE debe estar dentro de la lista IN

Ejercicios: Selecciona Pokémons con los siguientes criterios:

- Su altura sea 5, 10 o 20
- Su peso sea 20, 55 o 81
- Su experiencia base sea 64, 2 o 14

NOT IN

Nos permite definir una lista de posibles opciones dentro de la instrucción WHERE

```
SELECT * FROM table WHERE field NOT IN (data, data2, data3);
```

Para que la sentencia se ejecute el valor del campo WHERE no debe estar dentro de la lista NOT IN

Ejercicios: Selecciona Pokémons con los siguientes criterios:

- Su altura no sea 5, 10 o 20
- Su peso no sea 20, 55 o 81
- Su experiencia base no sea 64, 2 o 14

BETWEEN

Nos permite definir un rango de posibles opciones dentro de la instrucción WHERE

```
SELECT * FROM table WHERE field BETWEEN data AND data2;
```

Para que la sentencia se ejecute el valor del campo WHERE debe estar dentro del rango definido

Ejercicios: Selecciona Pokémons con los siguientes criterios:

- Su altura esté entre 5 y 20
- Su peso esté entre 20 u 81
- Su experiencia base esté entre 14 y 64

AND/OR/IN/NOT IN/BETWEEN

Ejercicios: Selecciona Pokémon con los siguientes criterios:

- Su altura sea mayor a 5, su peso menor a 100 y su experiencia base esté entre 50 y 100
- Su altura sea mayor a 5, o su peso menor a 100 y su experiencia base esté entre 50 y 100
- Su altura sea mayor a 5 o su peso menor a 100, y su experiencia base esté entre 50 y 100
- …

LIKE

Nos permite buscar texto “inexacto” dentro de la instrucción WHERE

```
SELECT * FROM table WHERE field LIKE pattern;
```

Para que la sentencia se ejecute el valor del campo WHERE debe coincidir con el patrón indicado. Estos son algunos ejemplos:

- ‘abc%’ → El texto empieza por ‘abc’
- ‘%abc’ → El texto termina con ‘abc’
- ‘%abc%’ → El texto incluye ‘abc’
- ‘_abc%’ → El texto contiene ‘abc’ en la segunda posición
- ‘abc_%’ → El texto empieza por ‘abc’ y tiene al menos 1 carácter más
- ‘abc__%’ → El texto empieza por ‘abc’ y tiene al menos 2 carácteres más
- ‘a%o’ → El texto empieza por ‘a’ y acaba por ‘o’
- …

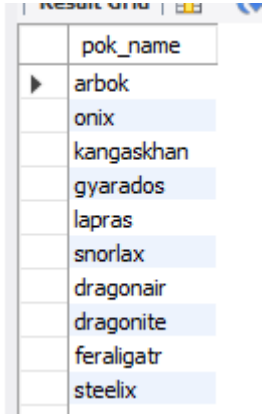
Ejercicios: Selecciona Pokémons aplicando algunos de los patrones anteriores

Alias/AS

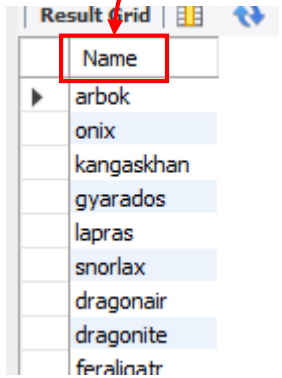
SQL permite asignar alias a tablas o campos para facilitar su lectura en sentencias más largas

```
SELECT pok_name
FROM pokemon
WHERE pok_height > 20
```

```
SELECT P.pok_name AS name
FROM pokemon P
WHERE P.pok_height > 20
```



pok_name
arbok
onix
kangaskhan
gyarados
lapras
snorlax
dragonair
dragonite
feraligatr
steelix

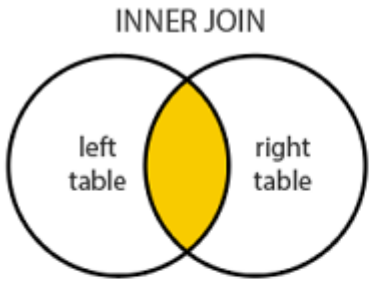


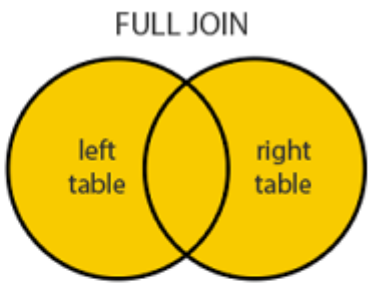
Result Grid

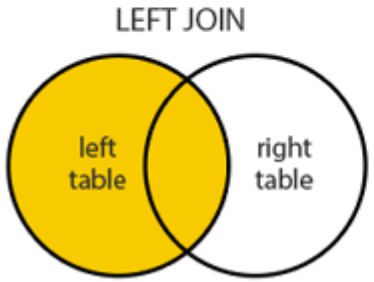
Name
arbok
onix
kangaskhan
gyarados
lapras
snorlax
dragonair
dragonite
feraligatr

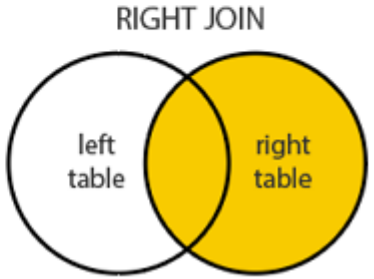
JOINS

Los JOIN nos permiten combinar 2 tablas que tengan algún campo en común. Existen varios tipos de Join:





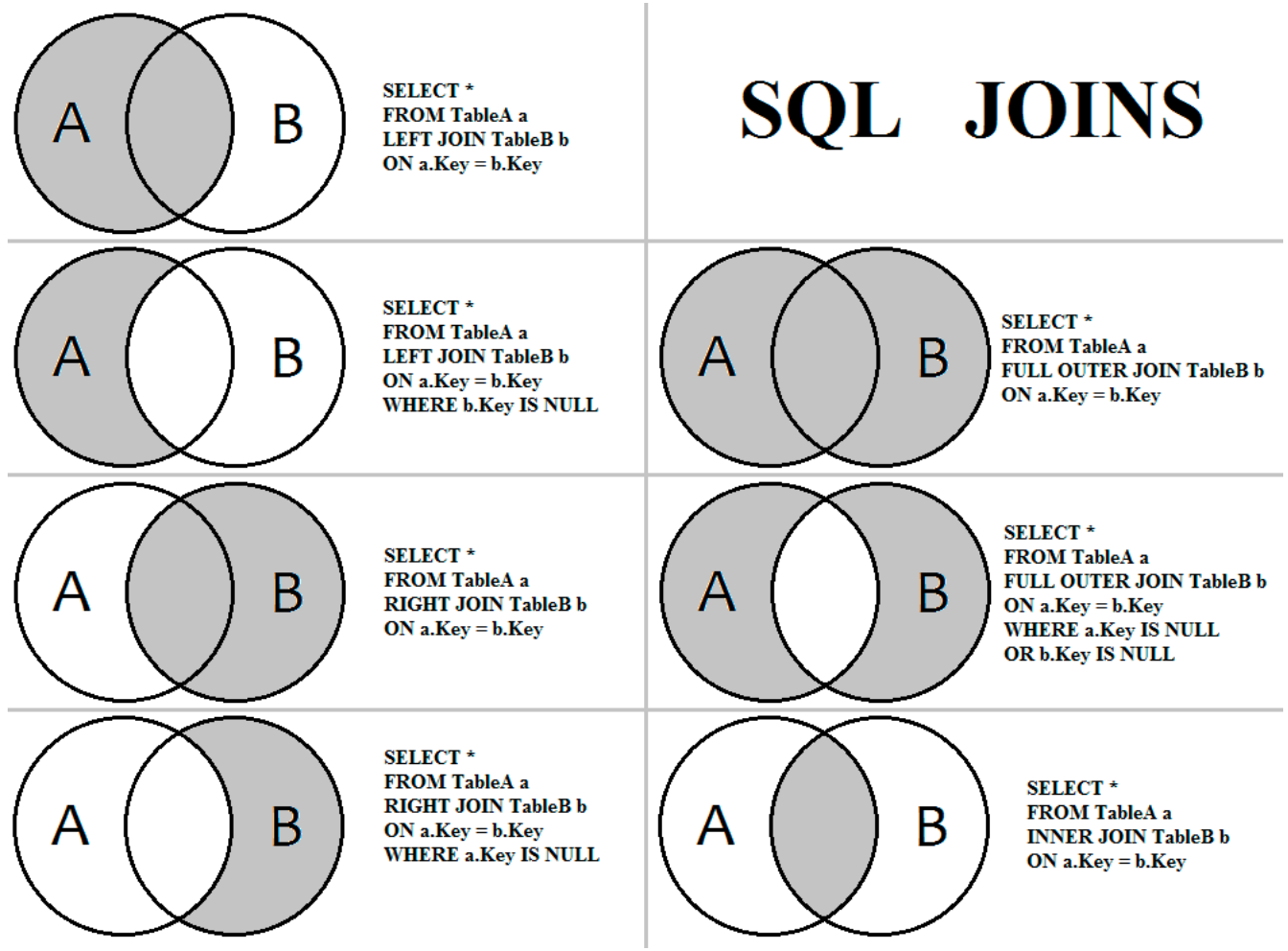




- **(INNER) JOIN**: Devuelve registros que tienen valores coincidentes en ambas tablas
- **LEFT (OUTER) JOIN**: Devuelve todos los registros de la tabla izquierda y los registros coincidentes de la tabla derecha
- **RIGHT (OUTER) JOIN**: Devuelve todos los registros de la tabla derecha y los registros coincidentes de la tabla izquierda
- **FULL (OUTER) JOIN**: Devuelve todos los registros cuando hay una coincidencia en la tabla izquierda o derecha

JOINS

Los JOIN nos permiten combinar 2 tablas que tengan algún campo en común. Existen varios tipos de Join:

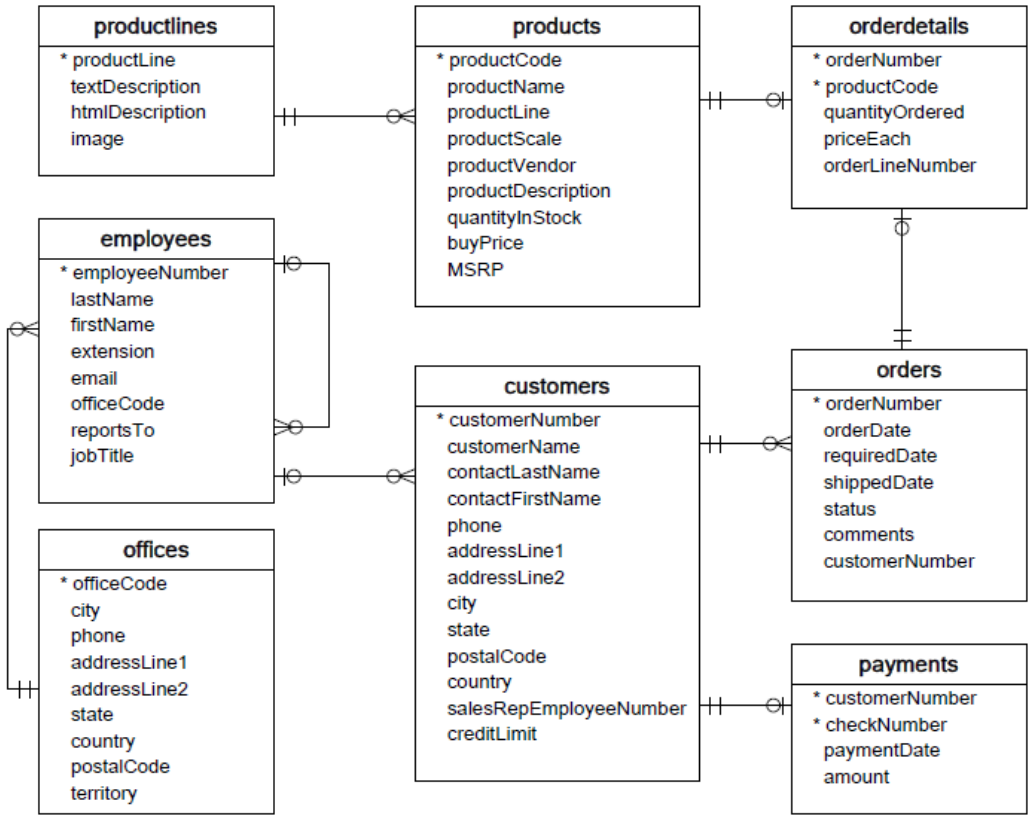


Ejercicios:

- Nombre de Pokémons con la habilidad número 5
- Nombre de Pokémons con un valor de ataque superior a 50
- Nombre de Pokémon y nombre de las habilidades que tiene
- Investigar las tablas y pensar combinaciones para realizar diferentes consultas usando JOIN

DISTINCT

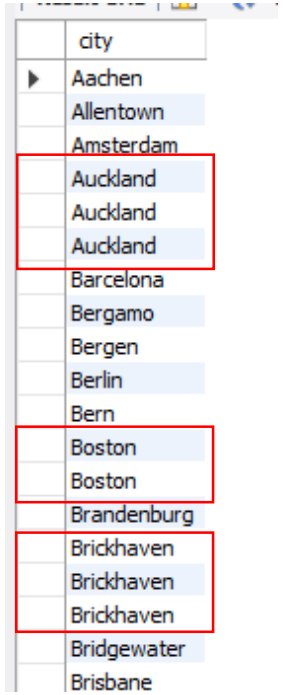
Vamos a importar una nueva base de datos



DISTINCT

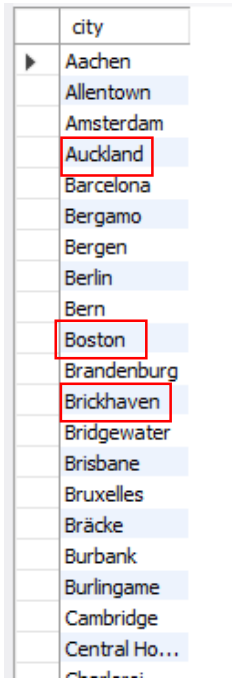
Nos permite devolver únicamente una ocurrencia de cada una de las diferentes filas que encontremos en nuestra base de datos.

```
SELECT DISTINCT column FROM table;
```



city
Aachen
Allentown
Amsterdam
Auckland
Auckland
Auckland
Barcelona
Bergamo
Bergen
Berlin
Bern
Boston
Boston
Brandenburg
Brickhaven
Brickhaven
Brickhaven
Bridgewater
Brisbane

SELECT city from customers



city
Aachen
Allentown
Amsterdam
Auckland
Barcelona
Bergamo
Bergen
Berlin
Bern
Boston
Brandenburg
Brickhaven
Bridgewater
Brisbane
Bruxelles
Bräcke
Burbank
Burlingame
Cambridge
Central Ho...

SELECT DISTINCT city from customers

COUNT

Cuenta el número total de filas de una tabla, y suele usarse junto con la instrucción WHERE para añadir filtros

```
SELECT COUNT(*) FROM table WHERE column = data;
```

Ejercicios:

- Prueba la instrucción COUNT en las diferentes tablas de las bases de datos que tenemos.

- ¿Puedes obtener cuántos clientes hay de cada uno de los diferentes países de la tabla “customers”?

GROUP BY

Es similar al DISTINC, pero la diferencia es que en lugar de devolver 1 única ocurrencia, las agrupa todas para poder realizar varias operaciones

```
SELECT COUNT(*) FROM table WHERE column = data GROUP BY column;
```

Ejercicios:

- ¿Puedes obtener cuántos clientes hay de cada uno de los diferentes países de la tabla “customers”?

MAX/MIN

Permite obtener los valores máximos y mínimos de una determinada columna

```
SELECT MAX(column1) FROM table [WHERE column2 = N];
```

```
SELECT MIN(column1) FROM table [WHERE column2 = N];
```

Ejercicios:

- Prueba las instrucciones MIN y MAX con varias tablas y prueba a concatenar diversas instrucciones

SUM

Suma todos los valores de una columna

```
SELECT SUM(column1) FROM table [WHERE column2 = N];
```

AVG

Devuelve la media de todos los valores de una columna

```
SELECT AVG(column1) FROM table [WHERE column2 = N];
```

Ejercicios:

- Prueba las instrucciones MIN y MAX con varias tablas y prueba a concatenar diversas instrucciones

SELECTS ANIDADOS

Es posible anidar “SELECTS” dentro de otros “SELECTS” para ajustar más nuestras consultas.

Por ejemplo, podemos usar el resultado de un “SELECT” para ver si existe un elemento dentro de otra tabla antes de realizar nuestra sentencia

```
SELECT *
FROM tablename
WHERE field IN(
  SELECT field
  FROM tablename2
  WHERE field2 = XXX
);
```

Ejercicios:

- Consulta los Pokémon que tienen la habilidad 2 usando SELECT anidados

HAVING

La instrucción HAVING es similar al WHERE, pero se usa para cosas muy concretas que WHERE no permite, como tratar agregados.

De esta forma podemos añadir una nueva condición donde poder tratar no sólo una comprobación atómica (como hemos hecho hasta ahora con WHERE), sino poder tratar un conjunto.

```
SELECT customerNumber
FROM orders
GROUP BY customerNumber
HAVING COUNT(customerNumber) > 4;
```

```
SELECT COUNT(customerNumber), customerNumber
FROM orders
GROUP BY customerNumber
HAVING COUNT(customerNumber) > 4;
```

Devuelve los customerNumbers que hayan realizado más de 4 pedidos

PRÁCTICA

#DISEÑAR E IMPLEMENTAR LA SIGUIENTE BASE DE DATOS#
- Diseñar una base de datos para gestionar una biblioteca.
- Deben poderse gestionar los usuarios, los libros, las temáticas de los libros, los autores de los libros y los alquileres
- Los usuarios deben tener información suficiente para que sean identificados como únicos
- Los alquileres deben poder gestionar cuándo una persona alquila un libro y si éste ha sido devuelto o no

#QUERIES PARA VUESTRA NUEVA BASE DE DATOS#
- Consultar la lista de usuarios/libros/autores/temáticas
- Consultar los libros que pertenecen a un autor
- Consultar cuántos libros hay de cada autor
- …

#QUERIES PARA LA BASE DE DATOS CLASSICMODELS#
- Consultar cuántos trabajadores hay en cada oficina, devolviendo el nombre de la oficina y el total de trabajadores
- Consultar los clientes que han realizado más de 3 pagos
- Consultar los clientes cuya media de pagos sea superior a 40000
- Obtener el desglose del pedido de un usuario

About Sogeti

Sogeti is a leading provider of technology and engineering services. Sogeti delivers solutions that enable digital transformation and offers cutting-edge expertise in Cloud, Cybersecurity, Digital Manufacturing, Digital Assurance & Testing, and emerging technologies. Sogeti combines agility and speed of implementation with strong technology supplier partnerships, world class methodologies and its global delivery model, Rightshore®. Sogeti brings together more than 25,000 professionals in 15 countries, based in over 100 locations in Europe, USA and India. Sogeti is a wholly-owned subsidiary of Capgemini SE, listed on the Paris Stock Exchange.

Learn more about us at

www.sogeti.com