

SQL



Pre work – SQL

Índice

1. Manipulación
2. Queries (consultas)
3. Funciones de agregación
4. Relación entre tablas
5. Usar una base de datos
6. Ejercicios



1. Manipulación

Introducción a SQL

SQL, **S**tructured **Q**uery **L**anguage, es un lenguaje de programación que nos permite hacer consultas en una base de datos **relacional**. Trabaja a través de sentencias simples y declarativas. Mantiene la información segura y concisa, además de mantener su integridad sin importar su tamaño.

Es un lenguaje ampliamente usado. Su conocimiento te da la libertad para explorar la información de la data y así poder tomar decisiones. Aprendiendo SQL también aprenderás conceptos aplicables a cualquier otro sistema de almacenaje de datos.

Base de datos relacional

Es un tipo de base de datos que almacena la información en tablas. Cada tabla se organiza en filas y columnas y entre ellas hay relaciones.

- Columna: es todo el conjunto de datos de un tipo particular (id, nombre, fecha...)
- Fila: es cada registro individual.

Toda la información de una base de datos tiene algún tipo de dato (integer, text, date...)



1. Manipulación

Sentencias

Una sentencia es texto que da una serie de instrucciones que SQL entiende como un comando válido. Siempre va a finalizar con un punto y coma **;**.

En el ejemplo de la derecha puedes observar la sentencia para crear una tabla en una base de datos.

- **CREATE TABLE:** esto es cómo inicia la sentencia y le da un orden específica a SQL. Por convención se escriben en mayúsculas.
- **nombre_tabla:** la tabla que se va a crear va a recibir este nombre.
- **(columna_1 tipo_de_dato, columna_2 tipo_de_dato...):** son parámetros. Un parámetro es una lista de columnas, de tipo de dato que albergará cada columna, los valores que se añadirán a las filas de la tabla... En este ejemplo columna_x es el nombre que va a recibir la columna, tipo_de_dato el tipo de dato que tendrá cada registro.

La estructura puede variar y el número de líneas que se use no es problema.

```
CREATE TABLE nombre_tabla (  
    columna_1 tipo_de_dato,  
    columna_2 tipo_de_dato,  
    columna_3 tipo_de_dato  
);
```



1. Manipulación

CREATE

Esta sentencia permite crear una nueva tabla en la base de datos. Cada tabla que quieras crear empezará con **CREATE TABLE** y a continuación el nombre que le vas a dar a la tabla.

INSERT

Esta sentencia da la instrucción de añadir información, en filas, a una tabla. A continuación se le indica la tabla y entre paréntesis las columnas de la misma.

Se añade la cláusula **VALUES** para indicarle entre paréntesis los registros en el mismo orden que las columnas separados por comas,

```
INSERT INTO tabla (columna1, columna2, columna3 )  
VALUES (registro1, registro2, registro3 );
```

SELECT

Esta sentencia es especial. Es la que da la instrucción de hacer una consulta a la base de datos. Recibe un nombre especial: query (consulta).

Cada query empezará con **SELECT**. A continuación van las columnas, separadas por comas. Lo siguiente es indicar la tabla desde dónde debe obtener la información. Esta instrucción se hace con **FROM** y el nombre de la tabla.

En las consultas hay un caracter especial: *, el asterisco. Con él le estamos indicando a SQL que seleccionamos toda la información de la tabla, todas sus columnas.

```
SELECT columna1, columna2  
  
FROM tabla;  
  
SELECT *  
  
FROM tabla;
```



1. Manipulación

ALTER

La sentencia empieza con **ALTER TABLE** y sirve para alterar una tabla, como por ejemplo añadir columnas. Para añadir una columna, igual que se hace al crear la tabla con sus columnas, hay que especificar el tipo de dato.

Al alterar la estructura de la tabla aparecerá sin registros, hasta que se añada la información. Al no contener información aparecerán los registros **NULL**, un valor especial que representa información inexistente o perdida.

```
ALTER TABLE tabla  
  
ADD COLUMN nombre_columna tipo_de_dato;
```

UPDATE

La sentencia UPDATE sirve para actualizar la información de una tabla, como introducir información en una columna nueva creada, cambiar información...

Se tiene que indicar en primer lugar la tabla sobre la que se va a trabajar.

A continuación, con **SET**, se le indica la columna y la información que se va a modificar, añadir...

Si hay que hacerlo sobre los registros que cumplan una condición concreta esa instrucción se expresará a través de la cláusula **WHERE**, que veremos más adelante.

```
UPDATE tabla  
  
SET nombre_columna = registro  
  
WHERE condición;
```



1. Manipulación

DELETE

Puede haber la necesidad de eliminar registros. Con esta sentencia se pueden eliminar una o varias filas (cuidado, incluso todas).

Hay que indicar sobre qué tabla se trabaja y la condición que debe cumplir la información que se quiere eliminar.

```
DELETE FROM tabla  
  
WHERE condición;
```

Constraint

Es importante no sólo indicar el tipo de dato cuando se crea una tabla, también si tiene alguna limitación.

En el momento de crear la tabla, indicar las columnas y su tipo de dato, se pueden añadir restricciones como primary key, unique, not null, un registro concreto...

Una **primary key** es un identificador único que permite identificar filas concretas. Cada identificador, al ser único, permite que si hay que alterar la tabla se puedan identificar las filas cómodamente y no se pueda introducir un registro en una fila diferente con un mismo identificador, pues violaría la restricción de que debe ser único y solamente puede haber una por fila.

```
CREATE TABLE tabla (  
    id tipo_de_dato PRIMARY KEY,  
    columnaX tipo_de_dato NOT NULL,  
    columnaY tipo_de_dato DEFAULT 'Grammy'  
);
```



2. Query

Introducción a las consultas

Para poder realizar una query hay que tener en cuenta una serie de cláusulas. Con ellas se le va a dar a SQL las instrucciones necesarias para localizar la información dentro de la base de datos y que nos devuelva el resultado con la información organizada y estructurada de manera correcta.

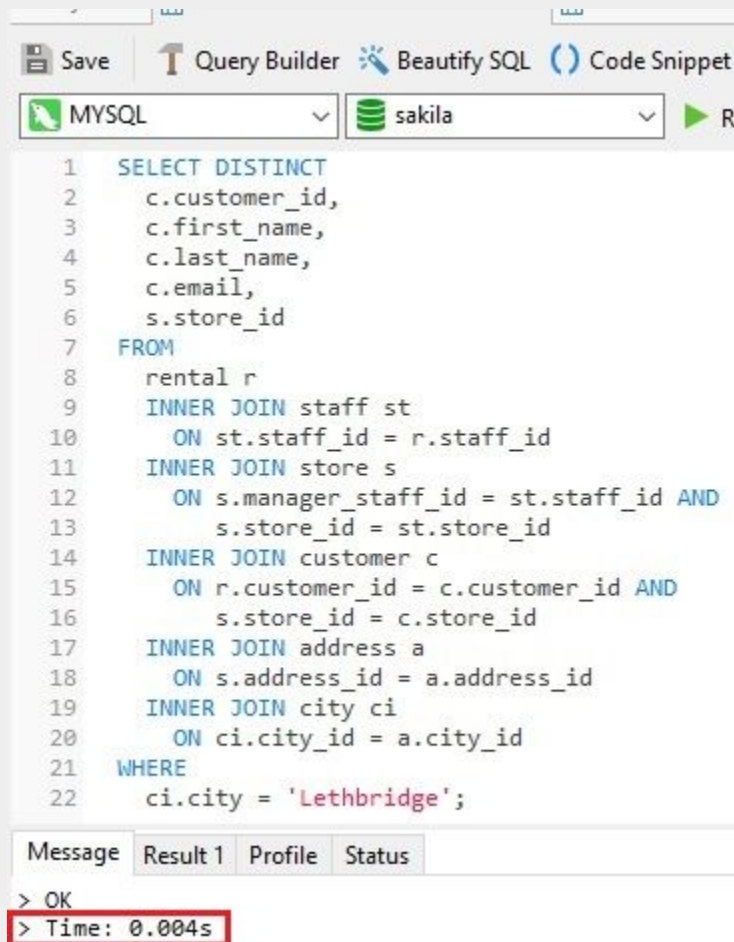
Simplemente, a través de la query, nos comunicamos con la base de datos haciendo preguntas.

SELECT

Ya conoces SELECT y el uso del asterisco para hacer una selección íntegra de la información contenida en la tabla.

FROM

En la siguiente línea de código con el FROM se le indica a SQL la tabla en la que debe fijarse.



```
1  SELECT DISTINCT
2      c.customer_id,
3      c.first_name,
4      c.last_name,
5      c.email,
6      s.store_id
7  FROM
8      rental r
9      INNER JOIN staff st
10         ON st.staff_id = r.staff_id
11      INNER JOIN store s
12         ON s.manager_staff_id = st.staff_id AND
13            s.store_id = st.store_id
14      INNER JOIN customer c
15         ON r.customer_id = c.customer_id AND
16            s.store_id = c.store_id
17      INNER JOIN address a
18         ON s.address_id = a.address_id
19      INNER JOIN city ci
20         ON ci.city_id = a.city_id
21  WHERE
22      ci.city = 'Lethbridge';
```

Message Result 1 Profile Status

> OK
> Time: 0.004s



2. Query

AS (alias)

Quizá la nomenclatura de las columnas de las tablas de la base de datos no son del todo claras. Se pueden renombrar para que la salida de nuestra consulta sea lo más clara posible. Además se puede otorgar un alias a las tablas, esto será muy útil para hacer uniones y escribir menos.

Esta práctica no altera la base de datos, únicamente renombra la columna o la tabla para nuestra consulta.

```
SELECT columna_1 AS uno, columna_2 AS dos
FROM tabla_1 AS primera;
```

DISTINCT

Mientras examinamos nuestros datos una cosa interesante de conocer es cuántos registros distintos hay, o cuáles son. Por ejemplo, tienes un videoclub y en una tabla tienes el registro de las películas y su género. Con distinct vas a poder seleccionar la columna de los géneros y darle la instrucción que te dé los registros diferentes.

Es importante conocer bien los datos para poder filtrar correctamente.

```
SELECT DISTINCT columna
FROM tabla;
```



2. Query (WHERE y condiciones)

WHERE

Esta cláusula se usa para filtrar los datos para que el resultado sea el correcto. A continuación del WHERE se introducen condiciones que permiten restringir los datos para acotar la respuesta a que contenga la información.

Imagina que estás consultando la base de datos de una universidad y quieres consultar únicamente aquellos estudiantes entre 1985 y 1987 que han sacado 'cum laude' en su tesis doctoral. Seleccionarías la columna que albergue el nombre, de la tabla con la información de los alumnos, que cumpla la condición en los años entre 1985 y 1987 y que en calificaciones tengan un 'cum laude'.

Para poder hacer este tipo de filtrados es necesario que conozcas las diferentes herramientas de escritura para indicarle a SQL estas restricciones.

Cuando se trabaja con cadenas de texto tienes que referenciar el texto entre comillas → 'texto'.

Comparadores

=	igual
!=	no igual
>	mayor que
<	menor que
>=	mayor o igual
<=	menor o igual

```
SELECT columna
FROM tabla
WHERE columna < 5;
```



2. Query (WHERE, like)

LIKE

Se utiliza dentro del WHERE cuando la condición es que cumpla el requisito de ser parecido o que contenga lo que se le indicará a continuación.

Puede ser que necesitemos buscar, siguiendo el ejemplo anterior, estudiantes cuyo apellido empiece por A, que su nombre tenga únicamente tres caracteres...

Para poder hacerlo hay varios caracteres comodín.

```
SELECT columna
FROM tabla
WHERE columna LIKE '%';
```

```
-----

SELECT columna
FROM tabla
WHERE columna LIKE '_';
```

Comparadores

%	cualquier caracter y cantidad de caracteres
_ (guion bajo)	un único caracter, sea el que sea

Su uso es muy sencillo. Por ejemplo, si queremos encontrar el resultado con la información de nombres acabados en X, indistintamente de la cantidad de caracteres que haya

- WHERE nombres LIKE '%X'

Nombres que no conocemos la última letra pero sí el resto:

- WHERE nombres LIKE 'CARL_'

Aplicando la lógica se puede traducir la necesidad para filtrar cualquier cadena.



2. Query (WHERE, NULL, BETWEEN)

IS NULL

Puede haber en la base de datos registros NULL. Cuidado, pues son registros vacíos, que no contienen ningún tipo de información.

No es posible usar operadores comparativos con los registros NULL. Si quiero buscar jugadores con el dorsal 5 la condición es: WHERE dorsal = 5; no obstante, no es posible hacer eso si quiero buscar registros NULL en dorsal: WHERE dorsal = NULL no lo va a entender.

NULL es una ausencia de registro, por ende tiene un uso particular:

- **IS NULL**: para aquellos registros NULL.
- **IS NOT NULL**: para aquellos registros **no** NULL.

```
SELECT columna
FROM tabla
WHERE columna IS NULL;
```

BETWEEN

El operador BETWEEN se usa en la cláusula WHERE para filtrar registros que se encuentren en un determinado rango. Necesita dos valores, desde inicio hasta final del rango. Pueden ser números, textos (o letras), fechas.

Cuando el rango es numérico se incluye el inicio y el fin. Cuando el rango es alfabético no se va a incluir el fin.

```
SELECT columna
FROM tabla
WHERE columna BETWEEN n_inicial AND n_final;
-----
SELECT columna
FROM tabla
WHERE columna BETWEEN letra_ini AND letra_fin;
```



2. Query (WHERE, operadores)

AND, OR, NOT, IN

Los dos primeros (AND, OR) permiten anidar condiciones, escribir más de una condición y si se cumplen todas o alguna filtrará la información.

- **AND:** sirve para añadir condiciones. Todas deben cumplirse. Ver los alumnos que se llaman Pepe nacidos en 1995.
 - `WHERE nombre = 'Pepe' AND nacido_en = 1995`
- **OR:** mostrará los datos que cumplan alguna de las condiciones. Ver los alumnos que se llaman Pepe o los que hayan nacido en 1995. Mostrará todos los alumnos cuyo nombre sea Pepe y todos los que hayan nacido en 1995, se llamen Pepe o no.
 - `WHERE nombre = 'Pepe' OR nacido_en = 1995`
- **NOT:** sirve para excluir esos registros. Ver todos los alumnos que no se llamen Pepe (también puede usarse el operador distinto a, `!=`, excepto con los registros NULL).
 - `WHERE NOT nombre = 'Pepe'`
 - `WHERE nombre != 'Pepe'`
- **IN:** sirve para especificar diferentes valores. Ver los alumnos que se llaman Pepe o Carla. También se puede hacer con el operador OR.
 - `WHERE nombre IN ('Pepe', 'CARLA')`
 - `WHERE nombre = 'Pepe' OR nombre = 'Carla'`

Operadores

AND	y
OR	o
NOT	no
IN	en

```
SELECT columna
FROM tabla
WHERE cualquier_ejemplo_amarillo;
```



2. Query (ORDER BY, LIMIT)

ORDER BY

Es muy habitual la necesidad de que la salida de nuestra consulta esté ordenada de una determinada manera. Ordenarla va a facilitar su análisis y utilidad, en muchas ocasiones.

Se indica después de la cláusula la columna por la que se deben ordenar los datos, que por defecto se hará ascendentemente. Si necesitas ordenarlo de mayor a menor, descendientemente, hay que especificarlo añadiendo **DESC** al final de la instrucción.

```
SELECT columna1, columna2
FROM tabla
WHERE condición_(si_es_necesaria)
ORDER BY columna DESC -> DESC si_es_necesario;
```

LIMIT

Puedes tener la necesidad de limitar la cantidad de registros que se muestran en la salida.

Esta cláusula te permite especificar la cantidad máxima de filas que se van a mostrar.

Imagina que quieres ver los tres alumnos con la nota más alta. Seleccionas la información de la tabla y lo ordenas de manera descendente. Lo limitas a tres registros y ya lo tienes.

```
SELECT nombre, nota
FROM alumnos
ORDER BY nota DESC
LIMIT 3;
```



2. Query (CASE)

CASE

La sentencia CASE se ubica, generalmente, en el SELECT de la query. Es la lógica conocida como **if-then**.

Se abre CASE y se van añadiendo condiciones precedidas por WHEN. Después de la condición THEN, si se cumple, haz lo que se indique a continuación.

Cada línea incluirá una condición.

Una vez se han añadido las condiciones necesarias se sigue con ELSE y a continuación la instrucción que hará con todos aquellos casos que no cumplan ninguna condición anterior.

Para finalizar y cerrar la sentencia se le indica END AS y el nombre que va a recibir.

Esta instrucción es como clasificar la información según las condiciones especificadas en cada WHEN y añadirá una columna a la salida de la query con el nombre que se le haya indicado en el END AS.

Esto no modifica la base de datos, únicamente se verá reflejado en la salida de la query.

```
SELECT columna1, columna2

CASE

    WHEN condición THEN haz_esto

    WHEN condición THEN haz_esto_otro

    ELSE haz_esto_último

END AS clasificación

FROM tabla;
```

Quiero ver los alumnos, con su nota y el tipo de calificación que supone esa nota (excelente, notable, suficiente, suspenso).

```
SELECT nombre, nota

CASE

    WHEN nota BETWEEN 10 AND 9 THEN 'excelente'

    WHEN nota BETWEEN 8.99 AND 7 THEN 'notable'

    WHEN nota BETWEEN 6.99 AND 5 THEN 'suficiente'

    ELSE 'suspenso'

END AS calificación

FROM alumnos;
```



3. Funciones de agregación

Introducción

Una query en SQL no va a acceder a la información en sin procesar sin más. Tiene la capacidad de hacer operaciones sobre esa información para poder responder a cuestiones específicas.

Estas operaciones aplicadas sobre múltiples filas de una tabla se llaman agregaciones, o funciones de agregación.

Una función va seguida de paréntesis. En el interior de los paréntesis se añade sobre qué se debe aplicar esa función, generalmente una columna.

COUNT()	Cuenta la cantidad de filas, no cuenta las filas que sean NULL
SUM()	Hace la suma de los valores de los registros
MAX()	Devuelve el resultado de valor máximo
MIN()	Devuelve el resultado de valor mínimo
AVG()	Calcula la media de los registros

SQL hará todos estos cálculos teniendo en cuenta la información completa trabajando con todos los decimales, sean los que sean.

La función **ROUND()** sirve para redondear el resultado y recibe dos argumentos:

- el primero sobre qué se va a aplicar
- el segundo la cantidad de decimales que va a tener

Se puede anidar, por ejemplo: redondear la media de la columna de los precios a dos decimales:

```
SELECT ROUND (AVG (precio), 2)
FROM tabla;
```



3. Funciones de agregación (GROUP BY)

GROUP BY

Cuando trabajamos con funciones de agregación de todos los registros que hay en una columna va a hacer el cálculo pertinente y va a devolver un único registro. Si en el SELECT hay más columnas SQL va a necesitar que las columnas tengan la misma cantidad de registros para poder mostrar la salida de la query.

Para ello existe GROUP BY. Esta cláusula agrupará los registros idénticos de una columna y aplicará la función de agregación sobre la columna que se le indique haciendo las agrupaciones según la información que se le indique en GROUP BY.

Imagina que en una misma tabla con información de alumnos hay las columnas de **nombre**, **nota**, **curso** y **grupo**. Quiero saber la nota media de cada grupo de cada curso, es decir, para la clase '1º A' qué nota media hay, y así con todos los grupos.

En el ejemplo de la izquierda tienes una query estándar y en el de la derecha la query para este ejemplo.

```
SELECT columna1, columna2, AGREGACION(columna)
FROM tabla
WHERE condición_si_procede
GROUP BY columna1, columna2;
```

```
SELECT curso, grupo, AVG(nota)
FROM alumnos
GROUP BY curso, grupo;
```



3. Funciones de agregación (HAVING, ORDER BY)

GROUP BY, HAVING, ORDER BY

Cuando agregamos datos y los partimos en diferentes grupos con la cláusula GROUP BY podemos añadir filtros para seleccionar qué información va a formar parte de las agrupaciones generadas.

La cláusula **HAVING** es muy similar a WHERE. La diferencia es que WHERE filtra los datos generales de la consulta y HAVING filtra los datos agregados. El funcionamiento de ambas es el mismo.

Para ello existe GROUP BY. Esta cláusula agrupará los registros idénticos de una columna y aplicará la función de agregación sobre la columna que se le indique haciendo las agrupaciones según la información que se le indique en GROUP BY.

Además, puedes necesitar ordenar el resultado de tu query según la información de alguna de las columnas. Esto se hace con la cláusula **ORDER BY** y a continuación se añaden las columnas que contienen los registros por los que queremos ordenar el resultado.

Imagina que en una misma tabla con información de alumnos hay las columnas de **nombre**, **nota**, **curso** y **grupo**. Quiero saber cuántos alumnos hay de cada grupo de cada curso, es decir, para la clase '1º A' número de alumnos que hay, y así con todos los grupos. Además solo queremos ver aquellos grupos que tengan más de 20 alumnos y lo queremos ordenar por curso y grupo (que el resultado sea 1º A, 1º B, 1º C, 2º A, 2º B, 2º C y así sucesivamente).

```
SELECT columna1, columna2, AGREGACION(columna)
FROM tabla
WHERE condición_si_procede
GROUP BY columna1, columna2;
HAVING restricción
ORDER BY columna(s);
```

```
SELECT curso, grupo, COUNT(*)
FROM alumnos
GROUP BY curso, grupo
HAVING COUNT(*) > 20
ORDER BY curso, grupo;
```



4. Relación entre tablas

Introducción

Para preservar la eficiencia en el almacenamiento de la información de la base de datos se reparte agrupándola en diferentes tablas.

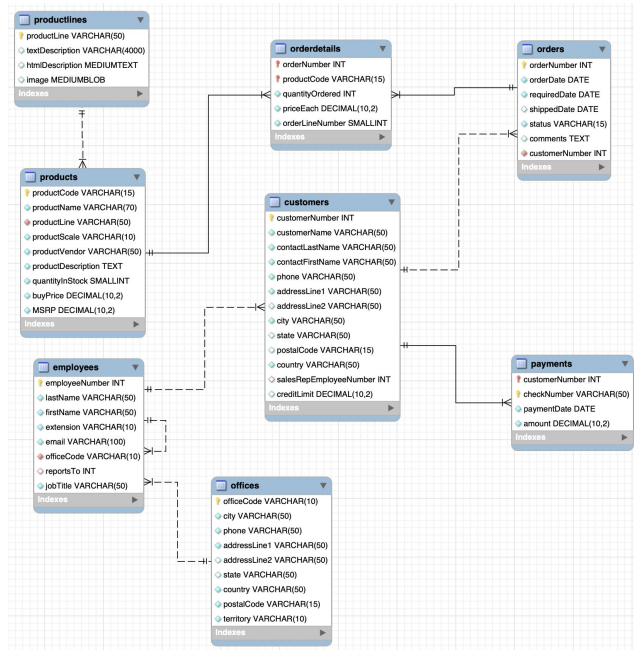
Imagina una plataforma de suscripción para ver contenido audiovisual en streaming. En la base de datos está la información de todos los usuarios registrados, el tipo de suscripción que tienen y el contenido que ven.

Si todo está contenido en la misma tabla todos los usuarios que tengan el mismo plan de suscripción harán que esa información se repita a lo largo de la tabla, además cada usuario va a ver diverso contenido y eso hará que a su vez se multipliquen los registros del mismo usuario.

Para evitar eso se puede distribuir teniendo la tabla de los usuarios, la tabla de los planes de suscripción y la tabla de los títulos que hay en el catálogo.

Estas tablas pueden relacionarse entre ellas si queremos ver por ejemplo el tipo de suscripción que tiene cada usuario juntando la información de las tablas donde está la información de los usuarios y la información de las suscripciones.

En la siguiente imagen puedes ver el diagrama de las relaciones de una base de datos de un concesionario. Hay tablas para las diferentes líneas de producto, todos los productos que se venden, todos los pedidos, los detalles de los pedidos, los clientes, los pagos de los clientes, los empleados y las diferentes oficinas que hay. Las flechas de unión indican las tablas que están relacionadas entre sí.



4. Relación entre tablas (JOIN)

JOIN

JOIN es la cláusula que nos permite unir tablas. A lo largo de la query después de indicar qué columnas necesitas indicas de qué tabla debe seleccionar esa información en el FROM. Ahora hay que dar un paso más y unirlo con otra tabla. En la siguiente línea se indica el tipo de JOIN y la tabla con la que hay que hacer la unión.

Al unir tablas hay que indicarle cuál es el punto de unión. **ON** la columna de una tabla sea igual **=** a la columna de la otra tabla.

Cuando se trabaja con diferentes tablas hay que tener en cuenta que la sintaxis varía. Siempre que nos refiramos a cualquier columna hay que decirle de qué tabla es esa información.

tabla.columna

En el ejemplo de la derecha podrás ver una query de ejemplo estándar para comprender la estructura.

```
SELECT

    tabla1.columna1

    ,  tabla2.columna_1

    ,  tabla2.columna_2

FROM  tabla1

JOIN  tabla2

    ON  tabla1.columna_común = tabla2.columna_común

WHERE (condición_si_es_necesario_filtrar)

ORDER BY (columna_por_la_que_ordenar_la_información);
```



4. Relación entre tablas (JOIN)

Tipos de JOIN

Hay diferentes maneras de hacer la unión entre las tablas. Puede hacerse:

- uniendo toda la información de una y otra tabla.
- uniendo únicamente la información que es coincidente en ambas tablas.
- uniendo toda la información de una tabla con aquella información coincidente de otra tabla.

INNER JOIN	muestra únicamente las coincidencias en ambas tablas
LEFT JOIN	muestra todos los datos de la tabla de la izquierda y únicamente los coincidentes en la tabla de la derecha. Aquellas filas en las que no haya coincidencia se rellenan con NULL
RIGHT JOIN	muestra todos los datos de la tabla de la derecha y únicamente los coincidentes en la tabla de la izquierda. Aquellas filas en las que no haya coincidencia se rellenan con NULL
FULL OUTER JOIN	muestra todos los datos de ambas tablas, uniendo las filas que coinciden, y mostrando el resto de la información de ambas tablas aunque no haya coincidencias. Todos aquellos datos sin coincidencias serán rellenos con valores NULL



4. Relación entre tablas (PK vs FK)

PRIMARY KEY (PK)

Toda tabla de una base de datos va a tener una columna que será el identificador de esa fila. Debe cumplir una serie de características:

- Ningún valor puede ser NULL.
- Cada registro debe ser único (en una tabla de alumnos no puede haber dos alumnos con el mismo identificador).
- Una tabla no puede contener más de una columna como PRIMARY KEY.

FOREIGN KEY (FK)

Cuando una clave primaria de una tabla aparece en otra tabla recibe el nombre de FOREIGN KEY, por ser el identificador único de las filas de otra tabla.

Así se puede relacionar una tabla con otra.

Esto es muy importante para poder unir unas tablas con otras. En la mayoría de los casos el nexo de unión entre dos tablas es la clave primaria de una tabla con la clave foránea de la otra tabla, siendo clave primaria de la primera tabla.



4. Relación entre tablas (UNION, WITH)

UNION

Cuando sea necesario unir verticalmente dos, o más, tablas se usa la cláusula UNION. Para poder realizar la unión entre las tablas:

- debe haber la misma cantidad de columnas en ambas tablas.
- los tipos de dato de esas tablas deben ser el mismo.

Al ser una unión VERTICAL se realiza una query que genera una salida en forma de tabla, seguidamente va la cláusula UNION y se sigue con la siguiente query para preparar la tabla que se añadirá en sentido vertical a la primera.

WITH

Muchas veces es necesario unir tablas que son el resultado de una query hecha anteriormente, para ello existe la sentencia WITH.

- permite separar una query dándole un nombre que funcionará como si fuera una tabla existente en nuestros datos.
- el nombre es el alias de esa tabla que sirve para referenciarla.
- podemos hacer cualquier tipo de consulta con esta tabla temporal, incluso JOINS, es como si fuera una tabla existente.

```
WITH alias_tabla AS (  
    SELECT...  
    FROM...)  
SELECT columnas  
FROM alias_tabla  
JOIN otra_tabla  
    ON alias_tabla.columna = otra_tabla.columna
```



5. Usar una base de datos

Ponte a practicar

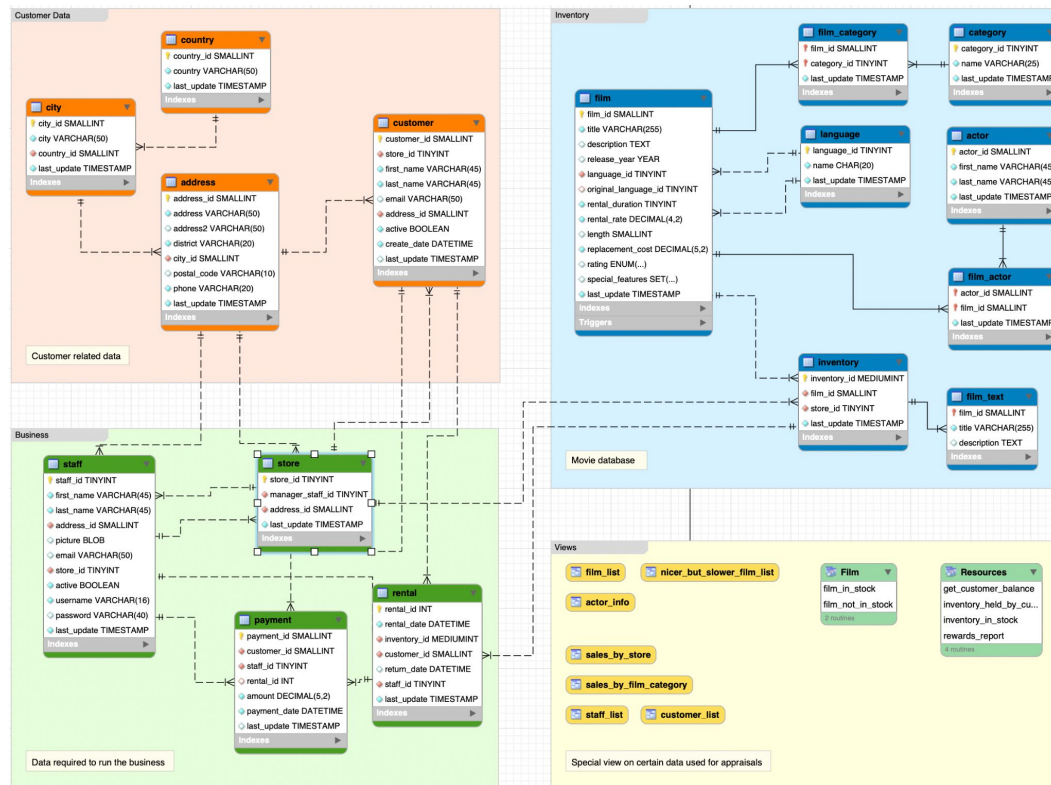
Ahora que ya tienes **MySQL Workbench** funcionando con bases de datos de muestra instaladas, como Sakila, te proponemos unos ejercicios para que practiques.

¡Vamos a hacerlo más fácil!

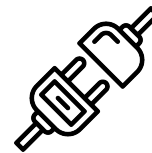
A la derecha verás el **diagrama entidad-relación** de la base de datos, te va a ayudar a familiarizarte con la información que contiene.

Es una base de datos sobre un **videoclub**, con la siguiente información:

- Clientes: dirección, país, ciudad...
- Inventario: películas, categoría, actores, inventario...
- Negocio: tiendas, personal, alquileres, pagos...

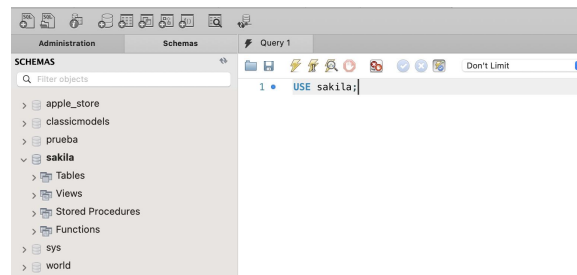
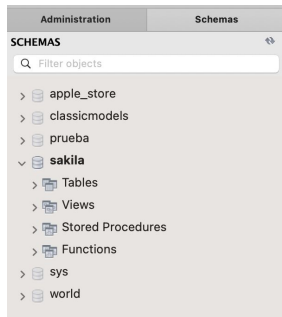
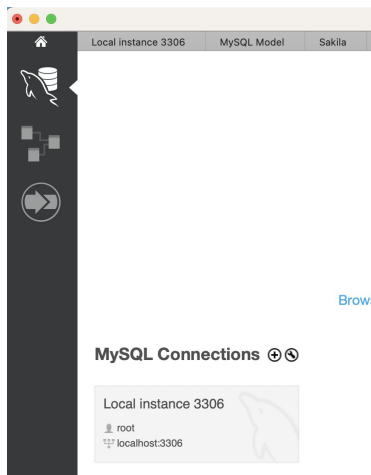


5. Usar una base de datos



Antes de empezar a utilizar la base de datos **sakila** tendrás que hacer unos pasos previos, en Workbench, para conectarte y poderla utilizar. Aquí tienes una guía:

1. Esto es lo primero que verás al abrir tu workbench. Debes hacer doble clic en “Local instance 3306”
2. En la siguiente ventana, en el menú lateral izquierdo, debes situarte en la pestaña “Schemas”. Ahí verás el listado de bases de datos que tienes en este momento. Si sakila no aparece en negrita vas a tener que lanzar tu primera línea de código para acceder a ella.
3. En el espacio del centro vas a escribir tu código. Para usar “sakila” escribe: **USE sakila;** Puedes hacer clic en el botón del rayo, o a la combinación de teclas **ctrl + enter** (windows) o **⌘ + enter**



6. Ejercicios

SELECT/FROM/DISTINCT

1. Muestra toda la información de la tabla **actor**.
2. Muestra únicamente los nombres de los diferentes idiomas, seleccionando la columna **name** de la tabla **language**.
3. Muestra el título, la descripción, la puntuación y la duración de las películas, seleccionando las columnas **title**, **description**, **rating** y **length** de la tabla **film**.
4. Con la misma query del ejercicio anterior da un alias a las columnas para mostrar su nombre en español (Título, Descripción, Puntuación, Duración).
5. Fíjate en la tabla **address**. Tienes direcciones completas. Puede haber muchas direcciones en un mismo distrito. Muestra únicamente la columna **district**, pero mostrándolos sin repeticiones, usando **DISTINCT**.

Te facilitamos una captura de cómo deberías ver el resultado de cada ejercicio.

Ejercicio 1

*200 filas

actor_id	first_name	last_name	last_update
1	PENELOPE	GUINNESS	2006-02-15 04:34:33
2	NICK	WAHLBERG	2006-02-15 04:34:33
3	ED	CHASE	2006-02-15 04:34:33
4	JENNIFER	DAVIS	2006-02-15 04:34:33
5	JOHNNY	LOI ORRIGIDA	2006-02-15 04:34:33

Ejercicio 2

*6 filas

name
English
Italian
Japanese
Mandarin
French

Ejercicio 3

*1000 filas

title	description	rating	length
ACADEMY DINOSAUR	A Epic Drama of a Feminist And...	PG	86
ACE GOLDFINGER	A Astounding Epistle of a Datab...	G	48
ADAPTATION HOLES	A Astounding Reflection of a Lu...	NC-17	50
AFFAIR PREJUDICE	A Fanciful Documentary of a Fri...	G	117
AFRICAN FGG	A Fast-Paced Documentary of a...	G	130

Ejercicio 4

*1000 filas

Título	Descripción	Puntuación	Duración
ACADEMY DINOSAUR	A Epic Drama of a Feminist An...	PG	86
ACE GOLDFINGER	A Astounding Epistle of a Data...	G	48
ADAPTATION HOLES	A Astounding Reflection of a L...	NC-17	50
AFFAIR PREJUDICE	A Fanciful Documentary of a F...	G	117
AFRICAN FGG	A Fast-Paced Documentary of...	G	130

Ejercicio 5

*378 filas

district
Alberta
QLD
Nagasaki
California
Atika



6. Ejercicios

WHERE/LIKE

Te facilitamos una captura de cómo deberías ver el resultado de cada ejercicio.

1. Muestra el **category_id** y el **name** de aquella categoría que su **id** sea **5** (tabla **category**).
2. Muestra toda la información de los pagos (**amount**) superiores a **7** (tabla **payment**).
3. Muestra toda la información de los idiomas que su **name** no sea **Italian** (tabla **language**).
4. Muestra los títulos de las películas que empiezan por la letra C (tabla **film**).
5. Muestra nombre y apellido de los actores que su apellido (**last_name**) tenga cinco letras en total y la última sea una H.

Ejercicio 1

*1 fila

category_id	name
5	Comedy
NULL	NULL

Ejercicio 2

*1532 filas

payment_id	customer_id	staff_id	rental_id	amount	payment_date	last_u
5	1	2	1476	9.99	2005-06-15 21:08:46	2006-1
14	1	1	6163	7.99	2005-07-11 10:13:46	2006-1
44	2	2	9236	10.99	2005-07-30 13:47:43	2006-1
62	3	1	1546	8.99	2005-06-16 01:34:05	2006-1
69	3	2	7503	10.99	2005-07-27 20:23:12	2006-1

Ejercicio 3

*5 filas

language_id	name	last_update
1	English	2006-02-15 05:02:19
3	Japanese	2006-02-15 05:02:19
4	Mandarin	2006-02-15 05:02:19
5	French	2006-02-15 05:02:19
6	German	2006-02-15 05:02:19

Ejercicio 4

*92 filas

title
CABIN FLASH
CADDYSHACK JEDI
CALENDAR GUNFIGHT
CALIFORNIA BIRDS
CAMFI OT VACATION

Ejercicio 5

*4 filas

first_name	last_name
CHARLIZE	DENCH
MATTHEW	LEIGH
JULIANNE	DENCH
CUBA	BIRCH



6. Ejercicios

WHERE/NULL/BETWEEN

Te facilitamos una captura de cómo deberías ver el resultado de cada ejercicio.

1. Muestra toda la información de la tabla **address** que en la columna **address2** haya **NULL**.
2. Muestra toda la información de los pagos (**amount**) superiores a **7** (tabla **payment**).
3. Muestra toda la información de los idiomas que su **name** no sea **Italian** (tabla **language**).
4. Muestra todas las columnas de la tabla **film** que su **rental_duration** sea entre **5** y **7**.
5. Modifica la query anterior para que además las películas que su **rental_duration** sea entre 5 y 7 cumplan que su **rental_rate** sea superior a **3** y que su **rating** sea **G**.

Ejercicio 1

*4 filas

address_id	address	address2	district	city_id
1	47 MySakila Drive	NULL	Alberta	300
2	28 MySQL Boulevard	NULL	QLD	576
3	23 Workhaven Lane	NULL	Alberta	300
4	1411 Lillydale Drive	NULL	QLD	576

Ejercicio 2

*1532 filas

payment_id	customer_id	staff_id	rental_id	amount	payment_date	last_u
5	1	2	1476	9.99	2005-06-15 21:08:46	2006-1
14	1	1	6163	7.99	2005-07-11 10:13:46	2006-1
44	2	2	9236	10.99	2005-07-30 13:47:43	2006-1
62	3	1	1546	8.99	2005-06-16 01:34:05	2006-1
69	3	2	7503	10.99	2005-07-27 20:23:12	2006-1

Ejercicio 3

*5 filas

language_id	name	last_update
1	English	2006-02-15 05:02:19
3	Japanese	2006-02-15 05:02:19
4	Mandarin	2006-02-15 05:02:19
5	French	2006-02-15 05:02:19
6	German	2006-02-15 05:02:19

Ejercicio 4

*594 filas

film_id	title	description	release...	language
1	ACADEMY DINOSAUR	A Epic Drama of a Feminist And...	2006	1
3	ADAPTATION HOLES	A Astounding Reflection of a Lu...	2006	1
4	AFFAIR PREJUDICE	A Fanciful Documentary of a Fri...	2006	1
5	AFRICAN EGG	A Fast-Paced Documentary of a...	2006	1
7	AIRPI ANF SIERRA	A Touching Saga of a Hunter An...	2006	1

Ejercicio 5

*30 filas

film_id	title	description	release...	lang
61	BEAUTY GREASE	A Fast-Paced Display of a Com...	2006	1
75	BIRD INDEPENDENCE	A Thrilling Documentary of a Ca...	2006	1
77	BIRDS PERDITION	A Boring Story of a Womanizer...	2006	1
95	BREAKFAST GOLDFINGER	A Beautiful Reflection of a Stud...	2006	1
123	CASARI ANCA SUPFR	A Amazing Panorama of a Croc...	2006	1



6. Ejercicios

ORDER BY/LIMIT/CASE

Te facilitamos una captura de cómo deberías ver el resultado de cada ejercicio.

1. Muestra toda la información de la tabla **payment** ordenando **amount** de menor a mayor.
2. Muestra toda la información de la tabla **payment** ordenando **amount** de mayor a menor.
3. Muestra toda la información las películas (tabla **film**) cuyo **title** empiece por **C** y muestra únicamente **3** resultados.
4. Muestra las columnas **title** y **length** de la tabla **film**. Añade otra columna que se llame **duración** que las clasifique por duración: si es **menor o igual a 120** → **'normal'** y si es **mayor de 120** → **'larga'**.
5. Selecciona los distintos valores de la columna **rating** de la tabla **film** y añade a tu query una columna que haga una clasificación de la siguiente manera, dale el alias **clasificación**:
 - a. PG → 'Parental Guidance Suggested'
 - b. G → 'General Audiences'
 - c. NC-17 → 'Adults Only'
 - d. PG-13 → 'Parents Strongly Cautioned'
 - e. R → 'Restricted'

Ejercicio 1

*16044 filas

payment_id	customer_id	staff_id	rental_id	amount	payment_date	last_u
9586	354	1	11782	0.00	2006-02-14 15:16:03	2006-
9773	361	1	14769	0.00	2006-02-14 15:16:03	2006-
12113	448	1	13577	0.00	2006-02-14 15:16:03	2006-
12357	457	2	14516	0.00	2006-02-14 15:16:03	2006-
12013	516	1	12015	0.00	2006-02-14 15:16:03	2006-

Ejercicio 2

*16044 filas

payment_id	customer_id	staff_id	rental_id	amount	payment_date	last_u
15821	591	2	4383	11.99	2005-07-07 20:45:51	2006-
15850	592	1	3973	11.99	2005-07-06 22:58:31	2006-
342	13	2	8831	11.99	2005-07-29 22:37:41	2006-
3146	116	2	14763	11.99	2005-08-21 23:34:00	2006-
5280	105	2	16040	11.99	2005-08-23 22:10:33	2006-

Ejercicio 3

*3 filas

film_id	title	description	relea...	languag
110	CABIN FLASH	A Stunning Epistle of a Boat An...	2006	1
111	CADDYSHACK JEDI	A Awe-Inspiring Epistle of a Wo...	2006	1
112	CALENDAR GUNFIGHT	A Thrilling Drama of a Frisbee A...	2006	1
	HULL	HULL	HULL	HULL

Ejercicio 4

*1000 filas

title	length	duración
ACADEMY DINOSAUR	86	normal
ACE GOLDFINGER	48	normal
ADAPTATION HOLES	50	normal
AFFAIR PREJUDICE	117	normal
AFRICAN EGG	120	larga

Ejercicio 5

*5 filas

rating	clasificación
PG	Parental Guidance Suggested
G	General Audiences
NC-17	Adults ONLY
PG-13	Parents Strongly Cautioned
R	Restricted



6. Ejercicios

AGREGACIONES/GROUP BY/HAVING

Te facilitamos una captura de cómo deberías ver el resultado de cada ejercicio.

1. Cuenta todas las filas que hay en la tabla **film**.
2. ¿Cuál es el **amount** total de la tabla **payment**? Haz una suma.
3. Muestra la cantidad de películas por cada **rental_rate**, desde la tabla **film**. Para poder hacerlo tendrás que **agrupar** por **rental_rate**.
4. Selecciona las columnas **customer_id** y **amount** de la tabla **payment**. Haz la **suma** de amount y **agrupa** por **customer_id** para saber el dinero total que ha gastado cada cliente.
5. Utiliza la query anterior, que nos dice cada customer el gasto total que ha hecho. Cada customer puede haber venido al videoclub infinidad de veces. Puedes filtrar usando **HAVING** después del **GROUP BY**. Haz que muestre únicamente aquellos **customer_id**, con su gasto total, que hayan hecho **40 pagos o más**.

Ejercicio 1

*1 fila

COUNT(*)	
1000	

Ejercicio 2

*1 fila

SUM(amount)	
67406.66	

Ejercicio 3

*3 filas

rental_rate	COUNT(*)	
0.99	341	
4.99	336	
2.99	323	

Ejercicio 4

*599 filas

customer_id	SUM(amount)	
1	118.68	
2	128.73	
3	135.74	
4	81.78	
5	144.62	

Ejercicio 5

*7 filas

customer_id	SUM(amount)	
75	155.59	
144	195.58	
148	216.54	
197	154.60	
236	175.58	



6. Ejercicios

JOIN/UNION/WITH

Te facilitamos una captura de cómo deberías ver el resultado de cada ejercicio.

1. Muestra por cada inventory_id el título de la película que le corresponde. Selecciona las columnas **inventory_id** y **title** (una de la tabla **inventory** y la otra de la tabla **film**). Parte de la tabla inventory en el FROM y haz un JOIN con la tabla film. El punto de unión entre ambas tablas es la columna **film_id**.

- a. ¿Te sale muchas veces el mismo título? No te preocupes, recuerda que es un videoclub. Eso es porque en el inventario hay varias copias de la misma película y cada copia tiene su identificador.

Ejercicio 1

*4581 filas

inventory_id	title
1	ACADEMY DINOSAUR
2	ACADEMY DINOSAUR
3	ACADEMY DINOSAUR
4	ACADEMY DINOSAUR
5	ACADEMY DINOSAUR

Ejercicio 2

*2 filas

id	name
1	ACADEMY DINOSAUR
1	PENELOPE

Ejercicio 3

*200 filas

actor_id	first_name	last_name	conteo
1	PENELOPE	GUINNESS	19
2	NICK	WAHLBERG	25
3	ED	CHASE	22
4	JENNIFER	DAVIS	22
5	JOHNNY	LOUORRIGANA	20

2. Quieres ver en la misma salida el título de la película con el id 1 y el nombre del actor con el id 1. Selecciona **film_id** y **title** de la tabla **film** (usa WHERE para que solo te dé el resultado con film_id = 1). Usa UNION para conectarlo con la siguiente query: selecciona **actor_id** y **first_name** de la tabla **actor** (usa WHERE para que solo te dé el resultado con actor_id = 1).
3. Quieres saber cuántas películas ha hecho cada actor. Tienes una tabla, **film_actor**, que tiene el **id de cada actor** y el **id de las películas** relacionadas. Selecciona el id del actor y haz un count agrupando por el id para ver cuántas películas ha hecho, dale el alias **conteo**. Utiliza WITH para usar esto como una tabla temporal. Dale el nombre a esta tabla de **conteo_peliculas**. Ahora que tienes esta tabla selecciona las columnas **actor_id**, **first_name**, **last_name** de la tabla **actor**, y **conteo** de la tabla temporal que acabas de hacer **conteo_peliculas**. Tendrás que hacer un JOIN, ambas tablas tienen el id de los actores.

