1.1 Creación de base de datos

Las bases de datos se crean con el comando CREATE DATABASE, seguido del nombre de la base de datos.

```
CREATE DATABASE <nombre>;
```

1.2 Manejo de tablas

1.2.1 Crear tablas: CREATE

El comando CREATE TABLE se encarga de crear una nueva tabla, para ello necesitamos pasarle todas las columnas, con sus respectivos tipos de datos separadas por comas.

```
CREATE TABLE <nombre> (
    <nombre_de_col_1> <tipo_de_dato> <otras_opciones>,
    <nombre_de_col_2> INT PRIMARY KEY,
    <nombre_de_col_3> VARCHAR NOT NULL,
    <nombre_de_col_4> INT DEFAULT 0
);
```

1.2.2 Eliminar tablas: DROP

Las tablas se borran con el comando DROP

```
DROP TABLE <nombre_de_tabla>;
```

1.2.3 Modificar tablas: ALTER

Alter nos permite modificar una tabla, ya se para agregar columnas, borrarlas, añadir constraints, removerlos, renombrarlas o borrarlas.

Para añadir una columna usamos ADD.

```
ALTER TABLE <nombre_de_tabla> ADD <column>;
```

Para añadir eliminar una columna usamos DROP COLUMN.

```
ALTER TABLE <nombre_de_tabla> DROP COLUMN <column>;
```

Si queremos añadir un contrain usamos ADD

```
ALTER TABLE <nombre_de_tabla> ADD <constraint>;
```

Si queremos remover un contrain usamos DROP

```
ALTER TABLE <nombre_de_tabla> RENAME TO <nuevo_nombre>;
```

Renombramos las columnas con RENAME también, pero pasándole primero el nombre de la columna antigua y, posteriormente al TO, el nuevo nombre.

```
ALTER TABLE <nombre_de_tabla> RENAME <col_antigua> TO <col_nueva>;
```

Para eliminar toda la información de una tabla usamos la palabra TRUNCATE

```
TRUNCATE TABLE <nombre_de_tabla>;
```

1.3 Consulta de datos

1.3.1 Elegir la tabla: FROM

La palabra FROM nos permite elegir la tabla en la que se realizará la consulta. FROM necesita usarse en conjunto con otra acción.

```
... FROM <table>;
```

1.3.2 Seleccionar columnas: SELECT

Podemos filtrar por columna pasándolas después de la palabra reservada SELECT

```
SELECT <col_1>, <col2>, <col2>, ... FROM <table>;
```

Para indicar que devuelva todas las columnas usamos el wildcard asterísco

```
SELECT * FROM <table>;
```

1.3.3 Limitar resultados: LIMIT

LIMIT limita los resultados de la consulta a un número especificado.

```
SELECT <col_1>, <col2>, <col2>, ... FROM <table> LIMIT <n>;
```

1.3.4 Condicionales: WHERE

WHERE nos sirve para expresar un condicional, ese condicional viene en forma de igualdad (o desigualdad).

Por ejemplo:

```
SELECT * FROM <table> WHERE <col_1> = <algo> and <col2> = <algo>;
```

Aquí tenemos otro con desigualidad

```
SELECT * FROM <table> WHERE <col_1> > <numero>;
```

1.3.5 Ordenar: ORDER BY

Nos permite ordenar los resultados por columna, ya sea de manera ascendente o descendente, siendo ascendente el valor predeterminado.

```
SELECT * FROM <table> ORDER BY <col_1> ASC [DESC];
```

1.3.6 Filtrar datos entre valores: BETWEEN y AND

BETWEEN nos permite especificar rangos al filtrar en una búsqueda.

```
SELECT * FROM <table> WHERE <date> BETWEEN '2000-12-12' AND '1970-12-12';
```

1.3.7 Búsqueda de texto: LIKE

LIKE permite establecer un patrón de búsqueda

```
SELECT * FROM <table> WHERE <col_1> LIKE '<texto>';
```

Este patrón de busqueda usa el simbolo de porcentaje como un comodín. Por ejemplo, para indicar todos los datos de una columa que empiecen con la letra A usariamos:

```
SELECT * FROM <table> WHERE <col_1> LIKE 'A%';
```

1.3.8 Búsqueda en lista: IN

In nos permite revisar si un valor se encuentra dentro de una lista de valores

```
SELECT * FROM <table> WHERE <col_1> IN (1, 2, 3);
```

1.3.9 Conjunciones: AND

Nos sirve para establecer conjunciones, generalmente usada tras la claúsula WHERE.

```
SELECT * FROM <table> WHERE <condicion_1> AND <condicion_2>;
```

1.3.10 Disyunciones: OR

Establece una disyunción, generalmente usada tras la claúsula WHERE.

```
SELECT * FROM <table> WHERE <condicio_1> OR <condicion_2>;
```

1.3.11 HAVING La claúsula HAVING fue agregada para usar funciones de agregación como sumatorias, conteos u otras

```
SELECT <col_1>
FROM <table2> as <t1>
  JOIN <table2> <t2> ON <t1>.<col_id> = <t2>.<col_id>
GROUP BY <col_1>
HAVING
  SUM(<col> = '<value>') > SUM(<col> = '<value>');
```

1.4 Actualizar datos: UPDATE

Update nos permite modificar información existente en una tabla.

Para esto necesitamos pasarle el nombre de la tabla después de la palabra reservada UPDATE y establecer las condiciones.

```
UPDATE <table> SET <col_1> = <data> WHERE <condicion>;
```

1.5 Agregar datos: INSERT

INSERT nos permite añadir un nuevo registro en una tabla de manera manual.

```
INSERT INTO <table> (<col_1>, <col_2>, ...) VALUES (<valor_para_col_1>, <valor_para_col_2>,
```

También podemos insertar directamente los valores en el resultado de una query con SELECT.

```
INSERT INTO <table_a> (<col_a_1>, <col_a_2>, ...) SELECT <col_b_1>, <col_b_2>, ... FROM <tab
```

1.6 Borrar datos: DELETE

DELETE nos permite eliminar información de una tabla

```
DELETE FROM <table>;
```

Es extremadamente importante **siempre especificar una claúsula WHERE en la query**, de otra manera DELETE borrará todos los registros de la tabla.

```
DELETE FROM <table> WHERE <condicion>;
```

1.6.1 Retornar datos: RETURNING

Algunos motores de base de datos permiten retornar los datos de una consulta tras una inserción, actualización o eliminación, con INSERT, UPDATE y DELETE, respectivamente.

```
<INSERT|UPDATE|DELETE> FROM <table> WHERE <condicion> RETURNING <col_1>, <col_2>...;
```

RETURNING permite usar la wildcard asterisco para retornar todo.

```
<INSERT|UPDATE|DELETE> FROM <table> WHERE <condicion> RETURNING *;
```

1.7 Agrupar datos: GROUP BY

GROUP BY agrupa realiza una agrupación de filas en filas promedio, en las que se representará el conjunto de valores. Se usa generalmente con funciones de agregación (ver más adelante), tales como COUNT, AVG, SUM, MIN y MAX.

Por ejemplo:

```
SELECT COUNT(<id>), <col>
FROM <table>
GROUP BY <col>
ORDER BY COUNT(<id>) DESC;
```

Que sería equivalente a algo como: Cuenta la cantidad de veces que se repite un valor en determinada columna y agrupa los datos por ese valor, de manera descendente.

n
n2
n3

1.8 Orden de ejecución

Las palabras reservadas de SQL se ejecutan en un orden predeterminado. Este orden es el siguiente:

1. FROM
2. WHERE
3. GROUP BY
4. HAVING
5. SELECT
6. ORDER BY
7. LIMIT

1.9 Creación de vistas: VIEW

Una vista es una tabla virtual que se genera como resultado de una consulta personalizada.

Puedes pensar en las vistas como funciones que nos devuelven un set de datos y/o columnas, como si se tratara de otra tabla.

Para crear una vista creamos nuestra query, de manera normal y le añadimos la frase CREATE VIEW

```
CREATE VIEW <nombre_de_la_vista> AS
SELECT <columna_1> <columna_2> FROM <tabla>;
```

Ahora podemos acceder al resultado de esa columna como si se tratara de cualquier tabla.

```
SELECT * FROM <nombre_de_la_vista>;
```

Cualquier intento de acceder a columnas que no estén en la vista resultará en un error.

```
SELECT <columna_que_no_esta_en_vista> FROM <nombre_de_la_vista>;
```

1.10 Creación de índices: INDEX

Para acelerar el tiempo de consulta de nuestras queries podemos usar índices.

```
CREATE INDEX <index_name> AS
ON <tabla> (<column>);
```

Tras un proceso largo se habrá creado el índice y todas aquellas búsquedas que incluyan la columna sobre la que se creó el índice veran un incremento en su rendimiento.

Los índices pueden crearse con mayor cantidad de columnas.

```
CREATE INDEX <index_name> AS
ON <tabla> (<column1>, [<column2>...]);
```

Los índices requieren espacio para almacenarse, por lo que es necesario llegar a un equilibrio entre velocidad de ejecución y cantidad de índices creados.

1.11 Unir consultas: JOIN

Un join se encarga de unir el contenido de dos tablas.

Por ejemplo:

Una primera tabla

base_id	Name
1	Brittle Hollow
2	Sun Station
3	Timber Hearth

Unida con una segunda tabla

astronaut_id	Name
1	Gabbro
2	Esker
3	Solanum

Se vería así

base_id	Name	astronaut_id	Name
1	Brittle Hollow	1	Gabbro
2	Sun Station	2	Esker
3	Timber Hearth	3	Solanum

La sintaxis para llevar a cabo la unión anterior sería la siguiente:

```
SELECT <columna_1>, <columna_2>, ... FROM <tabla_izquierda> INNER JOIN <tabla_derecha> ON <t
```

Unirá ambas tablas solamente donde la columna de la tabla izquierda sea igual a la columna elegida de la tabla derecha.

Como puede apreciarse, la información de las tablas se combina en añadiendo nuevas columnas, por lo que no importa si entre ambas tablas coinciden o no el resto de columnas.

La consulta anterior puede afinarse agregando condiciones o cualquier otro filtro que se quiera

```
SELECT <columna_1>, <columna_2>, ... FROM <tabla_izquierda> INNER JOIN <tabla_derecha> ON <t
```

1.11.1 Conflictos en los nombres al usar JOIN

Para evitar conflictos con columnas que tienen el mismo nombre, usamos la palabra AS para declarar un nombre al que hará referencía cada table y, a partir de él, acceder a las columnas.

Por ejemplo.

```
SELECT a.<columna_1>, b.<columna_2>, ... FROM <tabla_izquierda>  as a INNER JOIN <tabla_dere
```

El resultado es una query más corta y mucho más limpia.

1.11.2 Tipos de JOIN

Existen varios tipos de Join, no todas las tablas son compatibles con todos los tipos de join.

```
SELECT <columna_1>, <columna_2>, ... FROM <tabla_izquierda> <INNER|LEFT|RIGHT|FULL> JOIN <ta
```

- INNER, solo retornará las filas si hubo coincidencia
- LEFT, retornará todas las filas de la tabla izquierda, la que sigue de la palabra FROM, sin importar si hubo coincidencia. Si no hay coincidencia se reemplazarán los valores con null.
- RIGHT, retornará todas las filas de la tabla derecha, la que sigue de la palabra JOIN, sin importar si hubo coincidencia. Si no hay coincidencia se reemplazarán los valores con null.
- FULL, también conocida como OUTER, es una mezcla de la LEFT y la RIGHT, retornará ambos y reemplazará los valores de la otra tabla con null si no hubo coincidencia.

1.12 Unión de consultas: UNION

UNION se usa para combinar el resultado de dos, o más, consultas SELECT, donde la unión se realiza a añadiendo filas nuevas.

Es importante que coincidan el número de columnas entre una tabla y la otra, así como sus tipos de datos.

```
SELECT <column_1> FROM <table_1> [WHERE <condicion>]
UNION
SELECT <column_1> FROM <table_2> [WHERE <condicion>];
```

1.13 Excepción de consultas: EXCEPT

EXCEPT devuelve filas de la primera query que no están presentes en la segunda

```
SELECT <column_1> FROM <table_1> [WHERE <condicion>]
EXCEPT
SELECT <column_2> FROM <table_2> [WHERE <condicion>];
```

1.14 Intersecciones de consultas: INTERSECT

Devuelve filas que son retornadas por ambas queries.

```
SELECT <column> FROM <table> [WHERE <condicion>]
INTERSECT
SELECT <column> FROM <table> [WHERE <condicion>];
```

1.15 Otras funciones

1.15.1 TO_DATE y STR_TO_DATE

TO_DATE y STR_TO_DATE, nos permiten convertir un string en una fecha. TO_DATE es usado en Oracle y Postgres, mientras que STR_TO_DATE en MySQL.

1.15.2 CURRENT_TIMESTAMP

Devuelve la fecha actual.

1.15.3 COALESCE

COALESCE devuelve el primer elemento que no es null, es ideal para reemplazar valores nulos por valores por defecto.

```
COALESCE(<columna_nula>, 0) ...;
```

COALESCE puede recibir cualquier número de arugmentos.

```
COALESCE(<columna_nula>, <otra_columna_posiblemente_nula>, 0) ...;
```

1.15.4 Funciones reductoras

Estas funciones se encargan de leer una serie de datos y agregarlos para producir un solo dato, tal como un promedio, una sumatoria, el máximo el mínimo. Cualquiera puede recibir un WHERE para filtrar la consulta.

1.15.5 COUNT

Retorna el número de filas.

```
SELECT COUNT(<columna>) FROM <tabla> [WHERE <condicion>];
```

1.15.6 SUM

Retorna la sumatoria de los valores.

```
SELECT SUM(<columna>) FROM <tabla> [WHERE <condicion>];
```

1.15.7 AVG

Retorna el promedio para el grupo.

```
SELECT AVG(<columna>) FROM <tabla> [WHERE <condicion>];
```

1.15.8 MIN/MAX

Retornan el más pequeño o el más grande valor de una serie, respectivamente.

```
SELECT MIN(<columna>) FROM <tabla> [WHERE <condicion>];
```

1.15.9 TO_DATE y STR_TO_DATE

Convierte un string en una fecha. TO_DATE se usa en postgresql y oracle, mientras que STR_TO_DATE en MySQL.

1.15.10 CURRENT_TIMESTAMP

Retorna la fecha actual

1.16 Views

Una vista es una tabla virtual como resultado de una consulta. Se usan para crear tablas virtuales de consultas complejas.

```
CREATE <nombre_vista> AS <query>;
```

Donde query es una consulta tal como:

```
SELECT * FROM <tabla> WHERE ...;
```

1.17 Window views o vistas de ventanas

Una función de ventana lleva a cabo un cálculo a través de un conjunto de filas de una tabla que están relacionadas a la fila actual.

Puede pensarse en estas como el tipo de cálculo que hace la función aggregate. Sin embargo estas no causan que las filas se agrupen en un único valor de salida, sino que retienen sus identidades por separado.

```
SELECT depname, empno, salary, avg(salary) OVER (PARTITION BY depname) FROM empsalary;
```

La palabra clave OVER es la que se encarga de hacer que se comporte como una función de ventana.

```
  depname   | empno | salary |   avg
------------+-------+--------+--------
 develop    |    11 |   5200 | 5020.0
 develop    |     7 |   4200 | 5020.0
 develop    |     9 |   4500 | 5020.0
 develop    |     8 |   6000 | 5020.0
 develop    |    10 |   5200 | 5020.0
 personnel  |     5 |   3500 | 3700.0
 personnel  |     2 |   3900 | 3700.0
 sales      |     3 |   4800 | 4866.6
 sales      |     1 |   5000 | 4866.6
 sales      |     4 |   4800 | 4866.6
(10 rows)
```

Queries comunes

Encontrar datos duplicados en varias columnas

Para esto usaremos mano de la query

```
SELECT <col_1>, <col_2>, COUNT(*) FROM <table> GROUP BY <col_1>, <col_2> HAVING COUNT(*) > 1
```

Comparar fechas

Para comparar fechas podemos transformar un campo datetime usando funciones como YEAR

```
SELECT COUNT(*) FROM <table> where YEAR(<datetime_column>) IS <number>;
```

Encontrar el máximo o mínimo

Encontrar el máximo es trivial, basta con usar la función MAX o MIN, para valor máximo o mínimo, respectivamente y pasarle la columna como argumento.

```
SELECT <col_1>, <col_2> MAX(<col_numerica>) FROM <table> ;
```

Revisar si una columna es parte de una lista

El truco está en usar el operador IN y pasar la lista entre paréntesis.

```
SELECT * FROM <table> WHERE <col_1> IN (<lista>);
```