MySQL 25 Me Drive

>> A relational database management System (RDBMS). Open source + free

>> Cross-platform + ANSI-Compliant.

>> Found in 1995 by "Oracle" Corporation

>> Database Table?

• each row is a record / data entry

• each Coloumn has information about every record in the table.

>> Relational Database?

• Database defines the relationships between the tables.

>> SQL: Structured Query Language

>> SemiColon (;) is the standard way to separate each SQL statement.

```
SELECT Col1, Col2, ... FROM tablename
```

→ select specific columns from a table to retrieve.

```
SELECT * FROM table_name;
```

→ select all fields available in the table to retrieve.

```
SELECT DISTINCT Col1, Col2, ...
FROM table_name;
```

→ remove duplicate values.

```
SELECT Col1, Col2, ... FROM table_name
WHERE Condition;
```

→ filter records that fulfill a specified Condition.

```
SELECT Col1, Col2, ... FROM table_name
WHERE NOT Condition;
```

→ filter records that does not fulfill a specified condition.

(1)
```
SELECT Col1, Col2, ...
FROM table_name
ORDER BY Col1, Col2, ... ASC|DESC;
```

→ Sorts the records in ascending order by default.

```
INSERT INTO table_name (Col1, Col2,
Col3, ...) VALUES (value1, value2,
value3, ...);
```

→ Add a new record to the table

```
WHERE Column_name IS NULL;
```

→ A NULL value: not Zero value or a field that contains spaces.

```
WHERE Column_name IS NOT NULL;
```

→ NULL values are created when no value is added to the field at the time of inserting a new record or update a record.

```
UPDATE table_name
SET Column1 = Value1, Column2 = value2, ...
WHERE Condition;
```

→ Without "WHERE" clause, all records in the table will be updated.

```
DELETE FROM table_name WHERE Condition;
```

→ without "WHERE" clause, all records in the table will be deleted.

```
SELECT Column_name(s)
FROM table_name WHERE Condition
LIMIT n OFFSET m;
```

→ returns "n" records starting from m+1 to m+n+1

```
SELECT MIN/MAX/AVG/COUNT/SUM (Col_name) AS alias_name
FROM table_name
WHERE Condition;
```

→ Aggregate Functions: MIN - MAX - AVG - COUNT - SUM

```
SELECT Column1, Column2, ...
FROM table_name
WHERE Column LIKE Pattern
```

→ patterns: 'a%', '%a', '%or%', '_r%', 'a_%', 'a__%', 'a%o' [percent sign (%) → zero or more characters, underscore sign (_) → one single character]

25 Me Drive

```
SELECT Column_name(s)
FROM table_name
WHERE Column_name IN (value1, value2, ...)
```

↳ IN : a Shorthand for multiple OR Conditions.

```
SELECT Column_name(s)
FROM table_name
WHERE Column_name IN (SELECT STATEMENT)
```

↳ "NOT IN" is also valid

```
SELECT Column_name(s)
FROM table_name
WHERE column_name BETWEEN value1 AND value2;
```

↳ BETWEEN operator is inclusive & begin and end values are included.

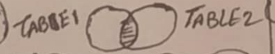
↳ "NOT BETWEEN"

```
• SELECT column_name AS alias_name
  FROM table_name;
• SELECT Column_name(s)
  FROM table_name AS alias_name;
• SELECT MIN/MAX/COUNT/AVG/SUM AS alias_name
  FROM table_name
```

⇓

Aliases for Columns / Tables / Aggregation Functions

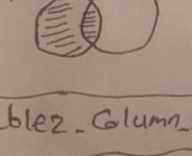
Supported Types of Joins



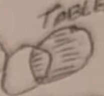
```
SELECT column_name(s) FROM table1
INNER JOIN table2 ON table1.Column_name
                   = table2.Column_name
```

↳ Selects all rows from both tables as long as there is a match between the Columns.

```
SELECT Column_name(s)
FROM table1
LEFT JOIN table2
ON table1.Column_name = table2.Column_name
```



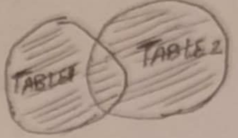
↳ The LEFT JOIN Keyword returns all records from the left table, and the matching records (if any) from the right table.



```
SELECT Column_name(s) FROM table1
RIGHT JOIN table2
ON table1.Column_name = table2.Column_name
```

↳ The RIGHT JOIN Keyword returns all records from the right table, even if there are no matches in the left table.

```
SELECT Column_name(s)
FROM table1
CROSS JOIN table2;
```



↳ CROSS JOIN can potentially return very large result-set! ex: TABLE1 → 10 rows, TABLE2 → 20 rows → 200

If there is no match → returned no records.

```
SELECT Column_name(s)
FROM table1
CROSS JOIN table2
WHERE table1.Column_name = table2.Column_name;
```

↳ This is equivalent to "INNER JOIN"

MySQL SELF Join → table is joined with itself.

```
SELECT Column_name(s)
FROM table1 T1, table2 T2
WHERE Condition;
```

MySQL UNION Operator → Same no of Columns & similar data types

```
SELECT Column_name(s) FROM table1
UNION
SELECT Column_name(s) FROM table2;
```

↳ Combine the result-set of two or more SELECT Statements & removes duplicate rows from the result-set.

→ UNION ALL → retrieves duplicates

```
→ SELECT 'Customer' AS Type, ContactName, City, Country
  FROM Customers
  UNION
  SELECT 'Supplier', ContactName, City, Country
  FROM Supplier;
```

25 Me rive

```
SELECT Column_name(s)
FROM table_name
WHERE Condition
GROUP BY Column_name(s)
ORDER BY Column_name(s);
```

GROUP BY = often [COUNT(), MAX(), MIN(), SUM(), AVG()]

```
SELECT Column_name(s)
FROM table_name
WHERE Condition
GROUP BY Column_name(s)
HAVING Condition
ORDER BY Column_name(s);
```

"HAVING" Clause was added to SQL because "WHERE" Keyword Cannot be used with aggregate functions.

```
SELECT Column_name(s)
FROM table_name
WHERE EXISTS
(SELECT Column_name FROM table_name
  WHERE Condition);
```

test for the existence of any record in a subquery.

```
SELECT Column_name(S)
FROM table_name              =, <>, >, <
WHERE Column_name Operator   ALL / ANY
(SELECT Column_name FROM table_name
  WHERE condition);
```

returns TRUE if ANY of the Subquery values meet the Condition.

returns TRUE if ALL of the Subquery values meet the Condition.

3 (INSERT INTO SELECT Syntax)

```
INSERT INTO table2 (col1, col2, ...)
SELECT column1, column2, column3, ...
FROM table1
WHERE Condition;
```

Source & target tables should have the same data types

Copies data from one table and inserts it into another table.

```
CASE
    WHEN Condition1 THEN result1
    WHEN Condition2 THEN result2
    WHEN Condition3 THEN result3
    ELSE result
END AS alias_name
```

Statement goes through Conditions and returns a value when the 1st Condition is met.

ELSE result : default value when there is not condition met

```
SELECT ProductName, Unitprice*(X+IFNULL(Y,0))
FROM Products;
SELECT productName, Unitprice*(X+COALESCE(Y,0))
FROM Products;
```

IFNULL/COALESCE ⇒ returns 0 if the value is NULL

```
-- line of Comment
/* ≡≡≡ */ ⇒ multi-line Comment
```

explain sections or prevent execution of SQL Statements

SQL Operators

Arithmetic

+, -, *, /, %

Bitwise

&, |, ^ XOR

MySQL Logical

ALL - AND - ANY - NOT
BETWEEN - EXISTS - IN
LIKE - NOT - OR - SOME

Comparison Operators

= > < >= <= <>

<> : not equal

Compound Operators

+= -= *= /= %=

&= ^-= |*=

^-= : XOR equals

|*= : Bitwise OR equals

MySQL DATABASE)

```
CREATE DATABASE databasename;
```

Make Sure you have admin privilege before Creating any database.

```
SHOW DATABASES
```

to display all databases Created

```
DROP DATABASE database_name;
```

make sure you have admin privilege before dropping a database.

```
CREATE TABLE new_table_name AS
    SELECT Col1, Col2, ...
    FROM existing_table_name
    WHERE ... ;
```

If you Create a new table using an existing table, the new table will be filled with the existing values from the old table.

```
DROP TABLE table_name
```

drop an existing table

```
TRUNCATE TABLE table_name
```

delete the data inside a table, but not the table itself.

```
ALTER TABLE table_name
ADD Column_name data type
```

add ~~a~~ column to the table

```
ALTER TABLE table_name
DROP COLUMN Column_name;
```

drops a column from a table

```
ALTER TABLE table_name
MODIFY COLUMN Column_name datatype;
```

modifies the datatype of a Column

4

```
CREATE TABLE table_name(
    Column1 datatype Constraint
    Column2 datatype Constraint,
    Column3 datatype Constraint,
    ...
);
```

NOT NULL - UNIQUE - PRIMARY KEY - FOREIGN KEY
CHECK - DEFAULT - CREATE, INDEX

```
CREATE TABLE table_name(
    ID int NOT NULL,
    Column2 datatype NOT NULL,
    ----
);
```

enforces a Column not to accept NULL values.

```
ALTER TABLE Persons
MODIFY Column_name datatype NOT NULL;
```

add this Constraint to a Column_name

```
CREATE TABLE table_name (
    Column_name datatype NOT NULL,
    Column_name datatype,
    UNIQUE(ID)
);
```

gurantee for uniqueness for a Column or set of columns

```
CREATE TABLE tablename(
    Colmn datatype NOT NULL,
    CONSTRAINT constraint_name UNIQUE(Column)
);
```

naming a UNIQUE Constraint

```
ALTER TABLE table_name
ADD UNIQUE (Column)
ADD CONSTRAINT UC_person UNIQUE(Column)
```

~~Drop~~ add a 'UNIQUE' Constraint to an existing table

```
ALTER TABLE table_name
DROP INDEX Constraint_name
```

```
CREATE TABLE table_name(
    Column1 datatype NOT NULL,
    ______
    PRIMARY KEY ( Column3)
);
```

↳ uniquely identifies each record in a table. A table can have only ONE primary Key. Primary keys cannot contain NULL values.

```
CREATE TABLE table_name(
    Column1 datatype NOT NULL,
    COSTRAINT constraint name PRIMARY KEY(ID, col2)
);
```

→ Naming the PRIMARY KEY Constraint

```
ALTER TABLE table_name
ADD PRIMARY KEY( Column)
ADD CONSTRAINT constraint name PRIMARY KEY(columns);
```

↳ add a 'Primary Key' Constraint to an existing table.

```
ALTER TABLE table_name
DROP PRIMARY KEY;
```

↳ removes the PRIMARY KEY

```
CREATE TABLE table_name(
    Column1 datatype NOT NULL,
    PRIMARY KEY ( Colum3),
    FOREIGN KEY (Column) REFRENCES
    table_name2 ( Column)
);
```

→ FOREIGN KEY Constraint prevents invalid data from being inserted into the foreign Key column.

→ FOREIGN KEY Constraint has to hold a value = one of the other table's values.

→ table_name = child table
table_name2 = Parent / Referenced table

(5)

```
CREATE TABLE table_name(
    Column1 datatype NOT NULL,
    ______
    PRIMARY KEY (—),
    CONSTRAINT name FOREIGN KEY
    (—) REFRENCES table_name2(—)
```

↳ naming the foreign Key Constraint.

```
ALTER TABLE table_name
ADD FOREIGN KEY (—) REFRENCES
table_name2 (—);
ADD CONSTRAINT name FOREIGN
KEY (—) REFRENCES table_name2
(—);
```

↳ adding a FOREIGN KEY Constraint to an existing table

```
ALTER TABLE table_name
DROP FOREIGN KEY name;
```

↓ to remove the foreign Key constraint

```
CREATE TABLE table_name(
    Column1 datatype NOT NULL,
    ----
    CHECK ( Condition)
);
```

⇒ limits the value range that can be placed in a column.

```
CREATE TABLE table_name(
    Column1 datatype NOT NULL,
    ----
    CONSTRAINT name CHECK(condition)
);
```

⇒ limits the values in certain columns based on values in other columns in the row.

```
ALTER TABLE table_name
ADD CHECK ( Condition)
```

⇒ add a CHECK constraint to an existing table

```
ALTER TABLE table_name
ADD CONSTRAINT name CHECK(condition);
```
naming the "CHECK" constraint added to the existing table.

```
ALTER TABLE table_name
DROP CHECK name;
```
removes "CHECK" constraint

```
CREATE TABLE table_name(
    Column1 datatype NOT NULL,
    Column datatype DEFAULT('txt')
                            OR
                    MySQL
                    Function  CURRENT_DATE()
);
```
"DEFAULT" is used to set a default value for a column. The default value will be added to all new records, if no other value is specified.

```
ALTER TABLE table_name
ALTER Column SET DEFAULT 'txt'
```
Adding a "DEFAULT" constraint to an existing table.

```
ALTER TABLE table_name
ALTER Column DROP DEFAULT;
```
Removes the "DEFAULT" constraint from an existing table

```
CREATE INDEX index_name
ON table_name( Column1, Col2, ---);
```
Create indexes in tables. They are used to retrieve data from the database more quickly than otherwise. The users cannot see the indexes, they are just used to speed up searches/queries.

```
CREATE UNIQUE INDEX index_name
ON table_name (Col1, Col2, --);
```
Duplicate values are not allowed

(6) Updating a table with indexes takes more time than updating a table without (because the indexes also need an update).

```
CREATE INDEX idx_name
ON table_name (Column);
```
index is created for an existing table

```
ALTER TABLE table_name
DROP INDEX index_name;
```
Removes the index from the table

```
CREATE TABLE table_name(
    Column1 datatype Not NULL AUTO_INCREMENT,
    ------
    PRIMARY KEY (Column1)    The same Column
);
```
By default the starting value for AUTO_INCREMENT is 1, and it will increment by 1 for each new record.

```
ALTER TABLE table_name AUTO_INCREMENT=100;
```
make the starting value for AUTO_INCREMENT is 100

~~ALTER TABLE table_name OFFSET = 3;~~

often the PRIMARY KEY field that we would like to be created automatically every time a new record is inserted.

MySQL Datatypes → Date Data type

DATE	DATETIME	TIMESTAMP	YEAR
YYYY-MM-DD	YYYY-MM-DD HH:MI:SS	YYYY-MM-DD HH:MI:SS	YYYY or YY

Tip To Keep your queries simple and easy to maintain, do not use time-components in your dates, unless you have to!

```
CREATE VIEW view_name AS
SELECT Column1, Column2, --
FROM table_name
WHERE Condition;
```
A view always shows up to-date! The database engine recreates the view, every time a user queries it.

```
CREATE VIEW [Brazil Customers] AS
SELECT CustomerName, ContactName
FROM Customers
WHERE Country = 'Brazil';
```

use "[ ]" when the name of your "VIEW" has white space characters.

```
SELECT * FROM [Brazil Customers];
```

We can query the view above

```
                          view_name
CREATE OR REPLACE VIEW AS
SELECT Column1, Column2, ...
FROM table_name
WHERE condition;
```

View_name must be inside two square parentheses [ ] if it has white space characters.

```
DROP VIEW View_name;
```

A view is deleted with DROP VIEW statement

```
SELECT Column(s)
FROM (SELECT Column(s) FROM table_name)
AS table_alias_name
```

⇓

use the query Statement as a table using a table_alias_name.

```
SELECT *, SUM(TOTAL_LAID_OFFS) OVER (ORDER BY 'MONTH')
AS ACCumulative_SUM
FROM( SELECT DATE_FORMAT('date', '%Y-%m') AS 'MONTH',
        SUM( total_laid_off) AS TOTAL_LAID_OFFS
FROM layoffs_stagging2
GROUP BY 'MONTH'
HAVING 'MONTH' IS NOT NULL
ORDER BY 1) AS T1;
```

⇓

Accumulative Summation of total_laid_off

Reminders

UPDATE With JOIN

```
UPDATE layoffs_stagging2 T1
JOIN layoffs_stagging2 T2
ON T1.Company = T2.Company
SET T2.industry = T1.industry
WHERE (T2.industry IS NULL OR T2.industry ='')
AND NOT (T1.industry IS NULL OR T2.industry ='');
```

"JOIN" can be used with "UPDATE" like "SELECT"

```
SELECT Columns FROM table_name
GROUP By non-aggregated Column(S)
```

Only Columns in "GRoup By" clause are non-aggregated, but other Columns in SELECT list should be aggregated. (SUM-MAX_MIN_AVG_COUNT)

(EX)

```
SELECT Continent, Country, MAX(TotalDeath)
GRoup BY Continent, Country
ORDER BY MAX(Total Death)
```

```
DRop TABLE IF EXISTs table_name
```

Create a new table with the Same Name

(CTE)

CTE Name   Arguments

```
WITH ACCUMULATED_NEW_VAC_PER_LOCATION (population,
    continent, location, ACC_NEW_VAC) AS
( SELECT continent, location, population,
    SUM(new_vaccinations) OVER (partition By
    location) FROM Table_name)
SELECT *, ACC_NEW_VAC/population *100.0
FROM ACCUMULATED_NEW_VAC_PER_LOCATION;
```

```
WITH CTE_name ( Parameters) As
( SELECT statement)
SELECT Column(s) FROM CTE_name
```