# MySQL Notes

# MySQL

1. MySQL is a widely used relational database management system (RDBMS).
2. MySQL is free and open-source.
3. MySQL is ideal for both small and large applications.
4. MySQL is very fast, reliable, scalable, and easy to use
5. MySQL is cross-platform
6. MySQL is compliant with the ANSI SQL standard
7. MySQL was first released in 1995
8. MySQL is developed, distributed, and supported by Oracle Corporation
9. MySQL is named after co-founder Monty Widenius's daughter: My

# Uses of MySQL

1. Huge websites like Facebook, Twitter, Airbnb, Booking.com, Uber, GitHub, YouTube, etc.
2. Content Management Systems like WordPress, Drupal, Joomla!, Contao, etc.
3. A very large number of web developers around the world

# RDBMS

1. RDBMS stands for Relational Database Management System.
2. RDBMS is a program used to maintain a relational database.
3. RDBMS is the basis for all modern database systems such as MySQL, Microsoft SQL Server, Oracle, and Microsoft Access.
4. RDBMS uses SQL queries to access the data in the database.

# Relational Database

- A relational database defines database relationships in the form of tables. The tables are related to each other - based on data common to each.

# SQL

1. SQL is the standard language for dealing with Relational Databases.
2. SQL is used to insert, search, update, and delete database records.
3. SQL keywords are NOT case sensitive i.e. select is the same as SELECT

# SQL Commands

- SELECT -> extracts data from a database
- UPDATE -> updates data in a database
- DELETE -> deletes data from a database
- INSERT INTO -> inserts new data into a database
- CREATE DATABASE -> creates a new database
- ALTER DATABASE -> modifies a database
- CREATE TABLE -> creates a new table
- ALTER TABLE -> modifies a table
- DROP TABLE -> deletes a table
- CREATE INDEX -> creates an index (search key)
- DROP INDEX -> deletes an index

# Create and Drop Database

# CREATE DATABASE: "To create a new SQL database"

```
    CREATE DATABASE databasename;
```

# DROP DATABASE: "To drop an existing SQL database"

```
    DROP DATABASE databasename;
```

# Create and Drop Table

# Create table Syntax: "To create a new table in a database"

```
    CREATE TABLE table_name (
    column1 datatype,
    column2 datatype,
    column3 datatype,
```

```
    ....
    );
```

# To create a new table using an existing table:

```
    CREATE TABLE new_table_name AS
    SELECT column1, column2,...
    FROM existing_table_name
    WHERE ....; // Optional
```

# Drop table Syntax: "To drop an existing table in a database"

```
    DROP TABLE table_name;
```

# To delete the data inside a table, but not the table itself:

```
    TRUNCATE TABLE table_name;
```

# INSERT INTO

- The INSERT INTO statement is used to insert new records in a table.

# 1. When Specify both the column names and the values:

```
    INSERT INTO table_name (column1,column2,column3, ...)
    VALUES (value1, value2, value3, ...);
```

# 2. When adding the values into all columns:

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

# Fetch Data From Database

# SELECT

- To select data from a database

# 1. When select data from any Column

```
SELECT column1, column2, ...FROM table_name;
```

# 2. When select all the Columns

```
SELECT * FROM table_name;
```

# 3. To return only distinct (different) values

```
SELECT DISTINCT columnname FROM table_name
```

# 4. When we want to count the columndata

```
SELECT COUNT(DISTINCT columnname) FROM table_name
```

# WHERE

- used to filter records

```
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

# Operators in Where

- '=' -> Equal
- '>' -> Greater than
- '<' -> Less than
- '>=' -> Greater than or equal
- '<=' -> Less than or equal
- '<>' -> Not equal. Note: In some versions of SQL this operator may be written as '!='

- 'BETWEEN' -> Between a certain range
- 'LIKE' -> Search for a pattern
- 'IN' -> To specify multiple possible values for a column

# Where with AND,OR, and NOT operators:

# AND Syntax: "If all the conditions separated by AND are TRUE"

```
    SELECT column1, column2, ...
    FROM table_name
    WHERE condition1 AND condition2 AND condition3 ...;
```

# OR Syntax: "If any of the conditions separated by OR is TRUE"

```
    SELECT column1, column2, ...
    FROM table_name
    WHERE condition1 OR condition2 Ocondition3 ...;
```

# NOT Syntax: "If the condition(s) is NOT TRUE"

```
    SELECT column1, column2, ...
    FROM table_name
    WHERE NOT condition;
```

# ORDER BY

- used to sort the result-set in ascending or descending order

# Order By

```
    SELECT column1, column2, ...
    FROM table_name
    ORDER BY column1, column2, ... ASC|DESC;
```

# Order By several columns

```
SELECT * FROM table_name
ORDER BY column1 ASC|DESC, column2 ASC|DESC, ...;
```

# NULL Values

- NULL value is a field with no value
- A NULL value is different from a zero value or a field that contains spaces. A field with a NULL value is one that has been left blank during record creation!

# Is Null

```
SELECT column_names
FROM table_name
WHERE column_name IS NULL;
```

# Is Not Null

```
SELECT column_names
FROM table_name
WHERE column_name IS NOT NULL;
```

# UPDATE

- used to modify the existing records in a table

# Update record

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

# Update Multiple record

```
UPDATE table_name
SET columnname = condition
WHERE columnname = condition;
```

# Limit, Min(), Max(), Count(), Avg(), and Sum()

## Limit: "To specify the number of records to return"

```
    SELECT column_name(s)
    FROM table_name
    WHERE condition
    LIMIT number;
```

## Min(): "It returns the smallest value of the selected column"

```
    SELECT MIN(column_name)
    FROM table_name
    WHERE condition;
```

## Max(): "It returns the largest value of the selected column"

```
  SELECT MAX(column_name)
  FROM table_name
  WHERE condition;
```

## Count(): "It returns the number of rows that matches a specified criterion"

```
  SELECT COUNT(column_name)
  FROM table_name
  WHERE condition;
```

# Avg(): "It returns the average value of a numeric column"

```
SELECT AVG(column_name)
FROM table_name
WHERE condition;
```

# Sum(): "It returns the total sum of a numeric column"

```
SELECT SUM(column_name)
FROM table_name
WHERE condition;
```

# LIKE

- used in a WHERE clause to search for a specified pattern in a column

There are two wildcards often used in conjunction with the LIKE operator:

- The percent sign (%) represents zero, one, or multiple characters
- The underscore sign (_) represents one, single character

# Like

```
SELECT column1, column2, ...
FROM table_name
WHERE columnN LIKE pattern;
```

# Not Like

```
SELECT column1, column2, ...
FROM table_name
WHERE columnN NOT LIKE pattern;
```

# IN

- To specify multiple values in a WHERE clause
- It's a shorthand for multiple OR conditions

# In:

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

# Not In:

```
    SELECT column_name(s)
    FROM table_name
    WHERE column_name NOT IN (value1, value2, ...);
```

OR

# In:

```
    SELECT column_name(s)
    FROM table_name
    WHERE column_name IN (SELECT STATEMENT);
```

# Not In:

```
    SELECT column_name(s)
    FROM table_name
    WHERE column_name NOT IN (SELECT STATEMENT);
```

# BETWEEN

- It selects values within a given range
- The values can be numbers, text, or dates
- begin and end values are included

# Between

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

# Not Between

```
    SELECT column_name(s)
    FROM table_name
    WHERE column_name NOT BETWEEN value1 AND value2;
```

# Between with IN

```
    SELECT column_name(s)
    FROM table_name
    WHERE column_name BETWEEN value1 AND value2 And column_name NOT IN
(value3,value4,...);
```

# Between Text

```
    SELECT column_name(s)
    FROM table_name
    WHERE column_name BETWEEN textvalue1 AND textvalue2;
```

# Not Between Text

```
    SELECT column_name(s)
    FROM table_name
    WHERE column_name NOT BETWEEN textvalue1 AND textvalue2;
```

# Between Dates

```
    SELECT column_name(s)
    FROM table_name
    WHERE column_name BETWEEN date1 AND date2;
```

# Aliases

- used to give a table, or a column in a table, a temporary name
- used to make column names more readable
- only exists for the duration of that query
- created with the AS keyword

# Alisas Column

```
    SELECT column_name AS alias_name
    FROM table_name;
```

# Alisas Table

```
    SELECT column_name(s)
    FROM table_name AS alias_name;
```

# Joins

- used to combine rows from two or more tables, based on a related column between them

# Types of Joins

- INNER JOIN: Returns records that have matching values in both tables
- LEFT JOIN: Returns all records from the left table, and the matched records from the right table
- RIGHT JOIN: Returns all records from the right table, and the matched records from the left table
- CROSS JOIN: Returns all records from both tables

# Inner Join: "Selects records that have matching values in both tables"

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

# LEFT JOIN: "Returns all records from the left table1, and the matching records (if any) from the right table2"

```
    SELECT column_name(s)
    FROM table1
```

```
    LEFT JOIN table2
    ON table1.column_name = table2.column_name;
```

# RIGHT JOIN: "Returns all records from the right table2, and the matching records (if any) from the left table1"

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

# CROSS JOIN: "Returns all records from both tables"

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

# Self Join: "Table is joined with itself"

```
    SELECT column_name(s)
    FROM table1 T1, table1 T2
    WHERE condition;
```

# UNION

- used to combine the result-set of two or more SELECT statements
- Every SELECT statement within UNION must have the same number of columns
- The columns must also have similar data types
- The columns in every SELECT statement must also be in the same order

# Union:

```
    SELECT column_name(s) FROM table1
    UNION
```

```
    SELECT column_name(s) FROM table2;
    ORDER BY columan_name; //Optional
```

# Union All:

```
    SELECT column_name(s) FROM table1
    UNION ALL
    SELECT column_name(s) FROM table2
    ORDER BY columan_name; //Optional
```

# Union With Where:

```
    SELECT column_name(s) FROM table1
    WHERE column_name = 'value1'
    UNION
    SELECT column_name(s) FROM table2
    WHERE column_name = 'value1'
    ORDER BY columan_name; //Optional
```

# Union All With Where:

```
    SELECT column_name(s) FROM table1
    WHERE column_name = 'value1'
    UNION All
    SELECT column_name(s) FROM table2
    WHERE column_name = 'value1'
    ORDER BY columan_name; //Optional
```

# GROUP BY

- The GROUP BY statement is often used with aggregate functions (COUNT(), MAX(), MIN(), SUM(), AVG()) to group the result-set by one or more columns.

```
    SELECT column_name(s)
    FROM table_name
    WHERE condition
    GROUP BY column_name(s)
    ORDER BY column_name(s);
```

# HAVING

- The HAVING clause was added to SQL because the WHERE keyword cannot be used with aggregate functions

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
HAVING condition
ORDER BY column_name(s);
```

# EXISTS

- Used to test for the existence of any record in a subquery
- Returns TRUE if the subquery returns one or more records

```
SELECT column_name(s)
FROM table_name
WHERE EXISTS (SELECT column_name FROM table_name WHERE condition);
```

# ANY and ALL

- The ANY and ALL operators allow you to perform a comparison between a single column value and a range of other values

# Any:

- Returns a boolean value as a result
- ANY means that the condition will be true if the operation is true for any of the values in the range.

```
SELECT column_name(s)
FROM table_name
WHERE column_name operator ANY (SELECT column_name FROM table_name
WHERE condition);
```

# All:

- Returns a boolean value as a result
- Returns TRUE if ALL of the subquery values meet the condition

# ALL Syntax With SELECT:

```
    SELECT ALL column_name(s)
    FROM table_name
    WHERE condition;
```

# ALL Syntax With WHERE or HAVING:

```
    SELECT column_name(s)
    FROM table_name
    WHERE column_name operator ALL (SELECT column_name FROM table_name
WHERE condition);
```

# CASE

- The CASE statement goes through conditions and returns a value when the first condition is met (like an if-then-else statement). So, once a condition is true, it will stop reading and return the result. If no conditions are true, it returns the value in the ELSE clause.
- If there is no ELSE part and no conditions are true, it returns NULL.

```
    SELECT column1, columnn2,
    CASE
        WHEN condition1 THEN result1
        WHEN condition2 THEN result2
        WHEN conditionN THEN resultN
        ELSE result
    END;
    FROM Database_name;
```

# MySQL Operators

# Arithmetic Operators

- '+' -> Add
- '-' -> Subtract
- '*' -> Multiply
- '/' -> Divide
- '%' -> Modulo

# Bitwise Operators

- '&' Bitwise AND
- '|' Bitwise OR
- '^' Bitwise exclusive OR

# Comparison Operators

- '=' -> Equal to
- '>' -> Greater than
- '<' -> Less than
- '>=' -> Greater than or equal to
- '<=' -> Less than or equal to
- '<>' -> Not equal to

# Compound Operators

- '+=' -> Add equals
- '-=' -> Subtract equals
- '*=' -> Multiply equals
- '/=' -> Divide equals
- '%=' -> Modulo equals
- '&=' -> Bitwise AND equals
- '^-=' -> Bitwise exclusive equals
- '|*=' -> Bitwise OR equals

# Logical Operators

- 'ALL' -> TRUE if all of the subquery values meet the condition
- 'AND' -> TRUE if all the conditions separated by AND is TRUE
- 'ANY' -> TRUE if any of the subquery values meet the condition
- 'BETWEEN' -> TRUE if the operand is within the range of comparisons
- 'EXISTS' -> TRUE if the subquery returns one or more records
- 'IN' -> TRUE if the operand is equal to one of a list of expressions
- 'LIKE' -> TRUE if the operand matches a pattern
- 'NOT' -> Displays a record if the condition(s) is NOT TRUE
- 'OR' -> TRUE if any of the conditions separated by OR is TRUE
- 'SOME' -> TRUE if any of the subquery values meet the condition

# ALTER TABLE

- used to add, delete, or modify columns in an existing table

- And used to add and drop various constraints on an existing table

# 1. ADD Column: "To add a column in a table"

```
    ALTER TABLE table_name
    ADD column_name datatype;
```

# 2. DROP COLUMN: "To delete a column in a table"

```
    ALTER TABLE table_name
    DROP COLUMN column_name;
```

# 3. MODIFY COLUMN: "To change the data type of a column in a table"

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

# Constraints

- used to specify rules for data in a table
- used to limit the type of data that can go into a table

The following constraints are commonly used in SQL:

- NOT NULL - Ensures that a column cannot have a NULL value
- UNIQUE - Ensures that all values in a column are different
- PRIMARY KEY - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table
- FOREIGN KEY - Prevents actions that would destroy links between tables
- CHECK - Ensures that the values in a column satisfies a specific condition
- DEFAULT - Sets a default value for a column if no value is specified
- CREATE INDEX - Used to create and retrieve data from the database very quickly

Syntax:

```
    CREATE TABLE table_name (
        column1 datatype constraint,
        column2 datatype constraint,
        column3 datatype constraint,
```

```
    ....
);
```

# NOT NULL

# NOT NULL on CREATE TABLE

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255) NOT NULL,
    Age int
);
```

# NOT NULL on ALTER TABLE

```
ALTER TABLE Persons
MODIFY Age int NOT NULL;
```

# UNIQUE

# UNIQUE Constraint on CREATE TABLE

```
CREATE TABLE Persons (
ID int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Age int,
UNIQUE (ID)
);
```

# UNIQUE Constraint on ALTER TABLE

```
ALTER TABLE Persons
ADD UNIQUE (ID);
```

## DROP a UNIQUE Constraint

```
ALTER TABLE Persons
DROP INDEX UC_Person;
```

## PRIMARY KEY

## PRIMARY KEY on CREATE TABLE

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    PRIMARY KEY (ID)
);
```

## PRIMARY KEY on ALTER TABLE

```
ALTER TABLE Persons
ADD PRIMARY KEY (ID);
```

## DROP a PRIMARY KEY Constraint

```
ALTER TABLE Persons
DROP PRIMARY KEY;
```

## FOREIGN KEY

## FOREIGN KEY on CREATE TABLE

```
CREATE TABLE Orders (
    OrderID int NOT NULL,
    OrderNumber int NOT NULL,
    PersonID int,
    PRIMARY KEY (OrderID),
    FOREIGN KEY (PersonID) REFERENCES Persons(PersonID)
);
```

# FOREIGN KEY on ALTER TABLE

```
    ALTER TABLE Orders
    ADD FOREIGN KEY (PersonID) REFERENCES Persons(PersonID);
```

# DROP a FOREIGN KEY Constraint

```
    ALTER TABLE Orders
    DROP FOREIGN KEY FK_PersonOrder;
```

# CHECK

# CHECK on CREATE TABLE

```
    CREATE TABLE Persons (
        ID int NOT NULL,
        LastName varchar(255) NOT NULL,
        FirstName varchar(255),
        Age int,
        CHECK (Age>=18)
    );
```

# CHECK on ALTER TABLE

```
    ALTER TABLE Persons
    ADD CHECK (Age>=18);
```

# DROP a CHECK Constraint

```
    ALTER TABLE Persons
    DROP CHECK CHK_PersonAge;
```

# DEFAULT

## DEFAULT on CREATE TABLE

```
    CREATE TABLE Persons (
        ID int NOT NULL,
        LastName varchar(255) NOT NULL,
        FirstName varchar(255),
        Age int,
        City varchar(255) DEFAULT 'Sandnes'
    );
```

## DEFAULT on ALTER TABLE

```
    ALTER TABLE Persons
    ALTER City SET DEFAULT 'Sandnes';
```

## DROP a DEFAULT Constraint

```
    ALTER TABLE Persons
    ALTER City DROP DEFAULT;
```

# CREATE INDEX

## CREATE INDEX

```
    CREATE INDEX index_name
    ON table_name (column1, column2, ...);
```

## CREATE UNIQUE INDEX

```
    CREATE UNIQUE INDEX index_name
    ON table_name (column1, column2, ...);
```

# DROP INDEX

```
ALTER TABLE table_name
DROP INDEX index_name;
```

# AUTO INCREMENT

- Auto-increment allows a unique number to be generated automatically when a new record is inserted into a table.
- Often this is the primary key field that we would like to be created automatically every time a new record is inserted.

```
CREATE TABLE Persons (
    Personid int NOT NULL AUTO_INCREMENT,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    PRIMARY KEY (Personid)
);
```

# Working With Dates

# MySQL Date Data Types

MySQL comes with the following data types for storing a date or a date/time value in the database:

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

```
SELECT * FROM Orders WHERE OrderDate='2008-11-11'
```

# CREATE VIEW

- A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.

## Creating a View

```
    CREATE VIEW view_name AS
    SELECT column1, column2, ...
    FROM table_name
    WHERE condition;
```

## Updating a View

```
    CREATE OR REPLACE VIEW view_name AS
    SELECT column1, column2, ...
    FROM table_name
    WHERE condition;
```

## Dropping a View

```
    DROP VIEW view_name;
```