MYSQL

MySQL is a very popular open-source relational database management system (RDBMS).

What is MySQL?

- MySQL is a relational database management system
- MySQL is open-source
- MySQL is free
- MySQL is ideal for both small and large applications
- MySQL is very fast, reliable, scalable, and easy to use
- MySQL is cross-platform
- MySQL is compliant with the ANSI SQL standard
- MySQL was first released in 1995
- MySQL is developed, distributed, and supported by Oracle Corporation

Who Uses MySQL?

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

What is RDBMS?

RDBMS stands for Relational Database Management System.

RDBMS is a program used to maintain a relational database.

RDBMS is the basis for all modern database systems such as MySQL, Microsoft SQL Server, Oracle, and Microsoft Access.

What is a Database?

A **database** is a digital system designed for the **storage** and **arrangement** of data.

What is a Database Table?

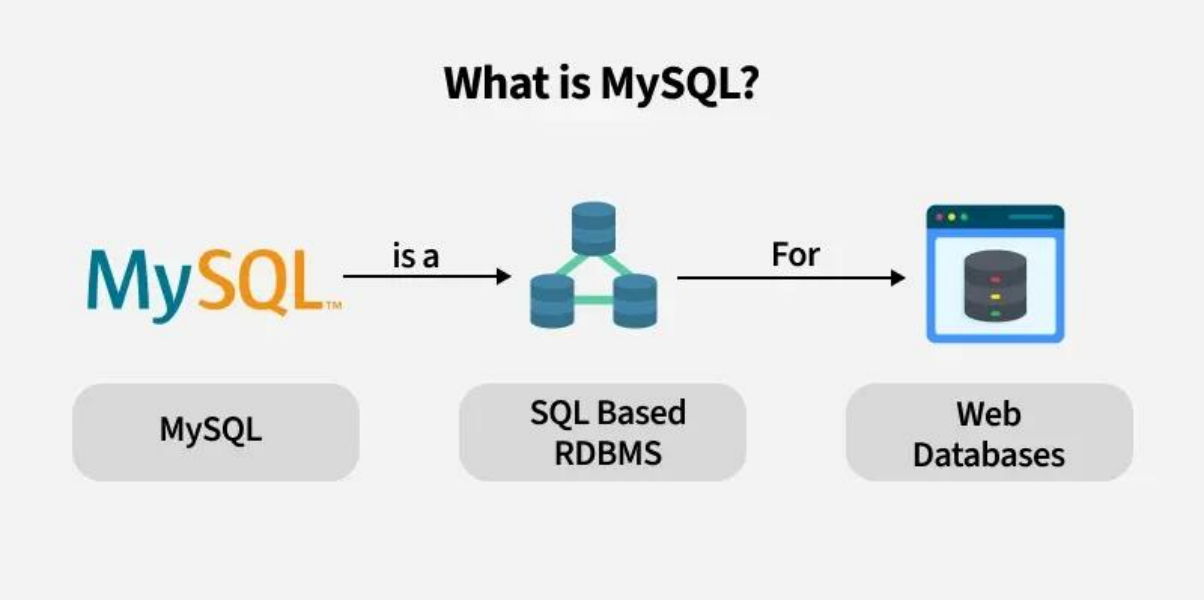
A table is a collection of related data entries, and it consists of columns and rows.

A column holds specific information about every record in the table.

A record (or row) is each individual entry that exists in a table.

What is a 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.



Key Features in MySQL

- **Open-Source**: MySQL is free and open-source, allowing modification and distribution.
- **High Performance**: It offers fast data retrieval and processing for large datasets.

- **ACID Compliance**: Ensures data integrity and reliability, especially with InnoDB storage.
- **Scalability**: Supports large databases and high traffic with features like partitioning and clustering.
- **Multiple Storage Engines**: Offers different storage engines (e.g., InnoDB, MyISAM) for flexible use.
- **Replication**: Supports master-slave replication for data redundancy and high availability.
- **Security Features**: Provides user authentication, SSL encryption, and secure data storage options.

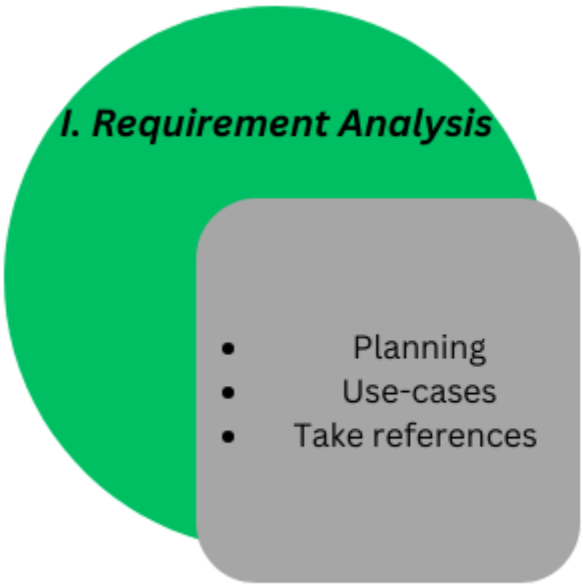
Applications of MySQL

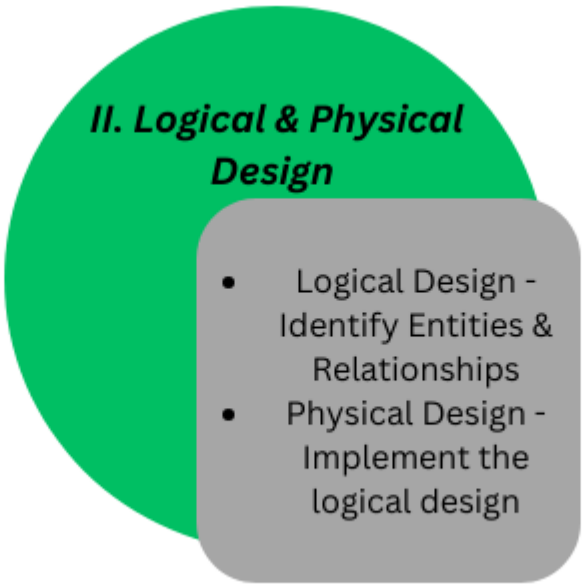
- **E-commerce:** MySQL is extensively used in e-commerce platforms for managing **product catalogs**, **customer data**, orders, and transactions.
- **Content Management Systems (CMS):** Many popular CMS platforms rely on MySQL as their backend database to store **website content**, **user profiles**, comments, and configuration settings.
- **Financial Services:** MySQL is employed in **financial applications**, including banking systems, payment processing platforms, and accounting software, to **manage transactional data**, customer accounts, and financial records.
- **Healthcare:** MySQL is used in **healthcare applications** for storing and managing **patient records**, medical histories, treatment plans, and diagnostic information.
- **Social Media:** MySQL powers the backend databases of many social media platforms, including **user profiles**, posts, comments, likes, and connections.

MySQL	S
MySQL is a Relational Database Management System (**RDBMS**) that uses **SQL** (Structured Query Language).	SQL (Structured C standard langu communicating wit
It is open source and accessible to any and everyone for free.	It is not an **open-s**
It supports basic programming languages like C, C++, Python, Ruby, etc.	It is in itself a Qu database systems.
It available only in the **English language**.	It is available in **di**
It doesn't support user-defined functions and XML.	It supports user-d XML.

How to Install MySQL on Windows?
WORKBENCH, xampp, wampp

Database design





Introduction of ER Model

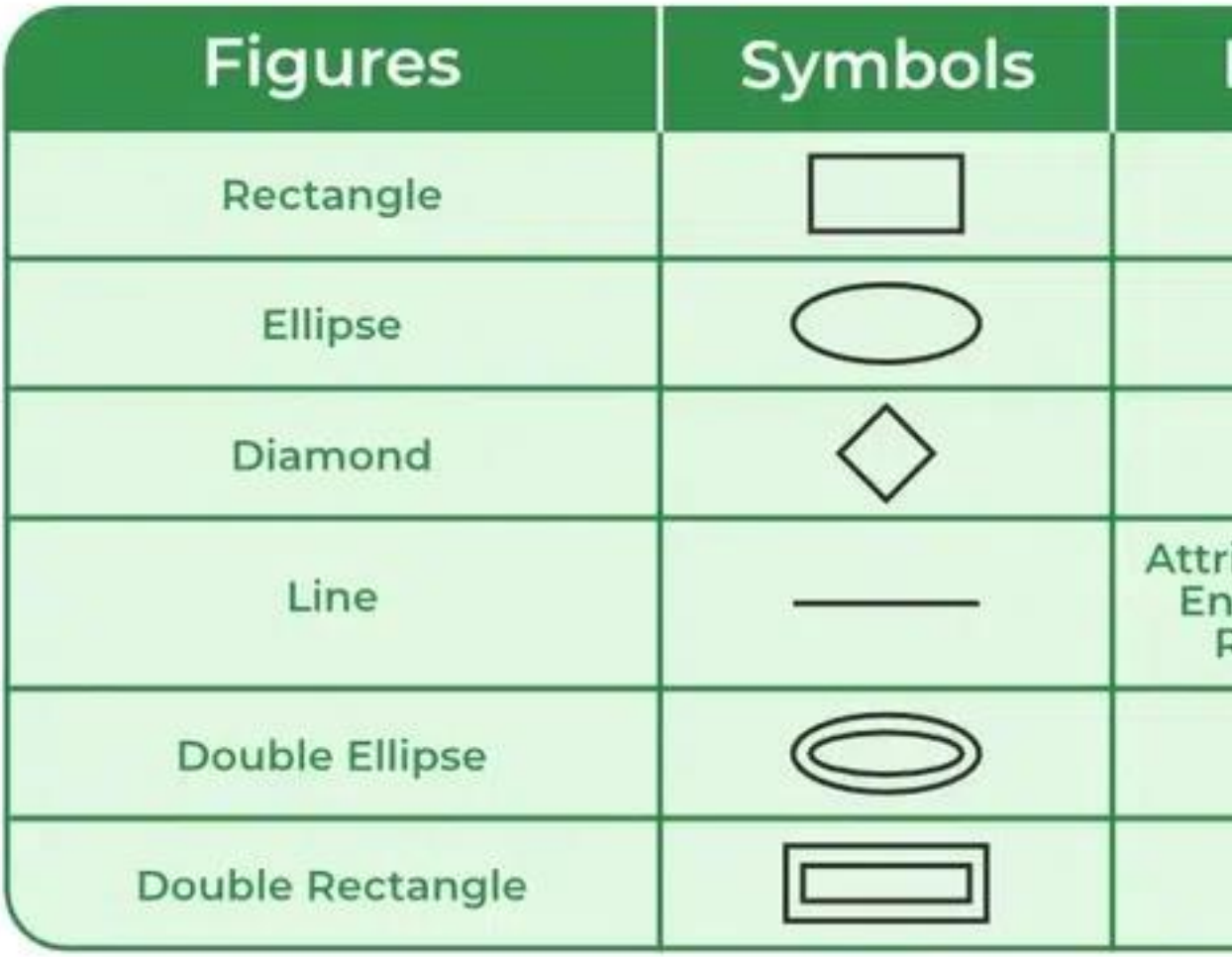
The Entity-Relationship Model (ER Model) is a conceptual model for designing a databases. This model represents the logical structure of a database, including entities, their attributes and relationships between them.

Symbols Used in ER Model

ER Model is used to model the logical view of the system from a data perspective which consists of these symbols:

- **Rectangles:** Rectangles represent entities in the ER Model.
- **Ellipses:** Ellipses represent attributes in the ER Model.
- **Diamond:** Diamonds represent relationships among Entities.

- **Lines:** Lines represent attributes to entities and entity sets with other relationship types.
- **Double Ellipse:** Double ellipses represent multi-valued Attributes, such as a student's multiple phone numbers
- **Double Rectangle:** Represents weak entities, which depend on other entities for identification.



What is an Entity?

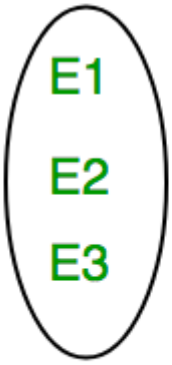
An Entity represents a real-world object, concept or thing about which data is stored in a database. It act as a building block of a database.

- **Real-World Objects:** Person, Car, Employee etc.
- **Concepts:** Course, Event, Reservation etc.
- **Things:** Product, Document, Device etc.

What is an Entity Set?

An entity refers to an individual object of an entity type, and the collection of all entities of a particular type is called an entity set

Entity Type



Entity Set

.

Types of Entity

1. Strong Entity
2. Weak Entity

- **Entity Set:**

  A collection of instances of the same entity type at a specific point in time.

  Attribute

  attributes are properties or characteristics that define an entity

  Types of Relationships:

- **One-to-one:**

  Each entity in one table is associated with only one entity in another table.

- **One-to-many:**

  Each entity in one table can be associated with multiple entities in another table.
- **Many-to-one:**

  Each entity in one table can be associated with multiple entities in another table, similar to one-to-many but from the perspective of the second table.
- **Many-to-many:**

  Each entity in one table can be associated with multiple entities in another table, and vice-versa

  Data integrity in MySQL refers to ensuring the accuracy, consistency, and reliability of data stored in a database

  Types of Data Integrity:
- **Entity Integrity:**

  Ensures that each record in a table **can be uniquely identified by its primary key.**
- **Referential Integrity:**

  Ensures that **relationships between tables are valid and consistent, often through foreign key constraints.**
- **Domain Integrity:**

  Ensures that **data values in a column are within a defined range or type.**
- **User-defined Integrity:**

  Allows **defining custom rules and constraints to enforce specific business requirements**.

  Key Mechanisms for Data Integrity in MySQL:
- **Constraints:**

MySQL employs various constraints to enforce data integrity rules. These include:

- **Not NULL:** Ensures that a column does not allow null values.

```
create table staff (staff_id int not null, staff_name varchar(255) not null,
phone varchar(15) not null,address text not null,salary double not null);


alter table table_name modify column phone varchar(15) not null
```

- **Default:** Specifies a default value for a column if no value is explicitly provided during insertion.

```
# set the default values in the columns
 create table buses(bus_no int NOT NULL,bus_route varchar(255),
 entry_datetime timestamp default current_timestamp());
 describe buses;


 alter table buses add district varchar(100) NOT NULL default 'TRICHY';


 # alter table table_name alter column_name set default 'content'
 alter table buses alter bus_route set default 'trichy - karaikudi';
```

- **Primary keys:** Ensure unique identification of each record in a table.

Primary keys must contain Unique values , and cannot contain NULL values (auto_increment)

A table can have only one primary key

Create table table_name(empid int primary key ,empname varchar(100) not null);

Create table table_name(empid int , empname varchar(100),primary key(empid));

Create table table_name(empid int , empname varchar(100), constraint pk_person primary key(id))

Alter:

Alter table table_name add primary key(id)

Alter table table_name add constraint pk_id primary key(id)

Drop:

Alter table table_name drop primary key;

MySQL uses the AUTO_INCREMENT keyword to perform an auto-increment feature.

By default, the starting value for AUTO_INCREMENT is 1, and it will increment by 1 for each new record.

ALTER TABLE Persons AUTO_INCREMENT=100;

- **Unique constraints:** Enforce uniqueness for one or more columns within a table.

  The unique constraint ensure that all values in a column are different

  Both unique and primary key

  But unique key accept the null values

  You can add many unique constraints per table

Create table table_name(empid int unique key ,empname varchar(100) not null);

Create table table_name(empid int , empname varchar(100), unique key(empid));

Create table table_name(empid int , empname varchar(100), constraint uq_person unique key(id))

Alter:

Alter table table_name add unique key(id)

Alter table table_name add constraint uq_id unique key(id)

Drop:

Alter table table_name drop index uq_person;

```
create table books(
 book_id int ,
 book_name varchar(255),
 author_phone varchar(20),
 email varchar(25),
 constraint uq_phone unique key(author_phone),
 constraint uq_email unique key (email)
 );
```

PRIMARY KEY	
Used to serve as a unique identifier for each row in a table.	Uni key.
Creates clustered index	Crea
Cannot accept NULL values.	Can
A Primary key supports auto increment value.	A un valu
We cannot change or delete values stored in primary keys.	We
Only one primary key	Mo

- **Check constraints:** Allow defining specific rules for the values that can be stored in a column.
    - The CHECK constraint is used to limit the value range that can be placed in a column.
    - If you define a CHECK constraint on a column it will allow only certain values for this column.
    - Syntax:

Create table table_name(column1 datatype, column2 datatype, check (condition))

Create table table_name(column1 datatype, column2 datatype, check ck_name (condition))

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

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    City varchar(255),
    CONSTRAINT CHK_Person CHECK (Age>=18 AND C
ity='Trichy')
);
```

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    City varchar(255),
    CONSTRAINT CHK_Person CHECK (Age>=18 ),
    CONSTRAINT CHK_City CHECK (City = 'Trichy'),


);
```

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

```
ALTER TABLE Persons
ADD CONSTRAINT CHK_PersonAge CHECK (Age>=18
AND City='Sandnes');
```

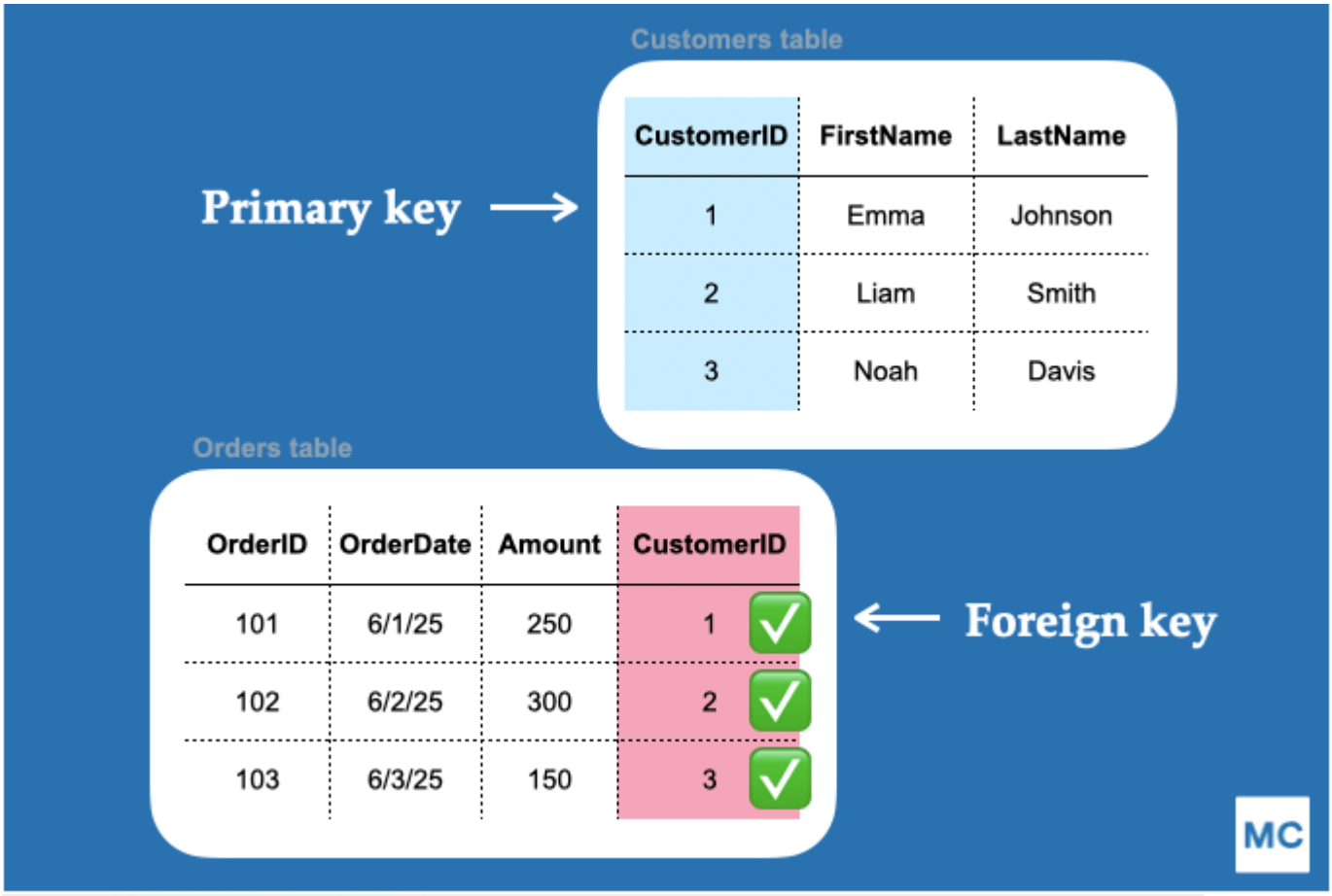
```
ALTER TABLE Persons
DROP CHECK CHK_PersonAge;
```

- **Foreign keys:** Maintain **relationships between tables by enforcing referential integrity**, meaning that data in a referencing table must be valid in the referenced table.

  The FOREIGN KEY is a field (or a set of fields) in one table that references the primary key in another table. This creates a relationship between the two tables, ensuring referential integrity.

  A FOREIGN KEY is a field (or collection of fields) in one table, that refers to the PRIMARY KEY in another table.

  The table with the foreign key is called the child table, and the table with the primary key is called the referenced or parent table.



Customer	Street	City
Firm 1 SE	Main Street 1	Las Vegas
Firm 2 SE	Fifth Avenue 14	New York
Firm 3 SE	Second Street 3	Houston

Primary Key

Primary Key — Foreign Key

Order ID	Customer	Sales
90001	Firm 1 SE	1,394.49 €
90002	Firm 2 SE	532.20 €
90003	Firm 1 SE	19,244.32 €
90004	Firm 3 SE	23.39 €

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

CREATE TABLE Orders (
    OrderID int NOT NULL,
    OrderNumber int NOT NULL,
    PersonID int,
    PRIMARY KEY (OrderID),
    CONSTRAINT FK_PersonOrder FOREIGN KEY (PersonID)
    REFERENCES Persons(PersonID)
);
-----------------------------------------------------------------
-----
ALTER:
ALTER TABLE Orders
ADD FOREIGN KEY (PersonID) REFERENCES Persons(Perso
nID);

ALTER TABLE Orders
ADD CONSTRAINT FK_PersonOrder
FOREIGN KEY (PersonID) REFERENCES Persons(PersonID);
-----------------------------------------------------------------
----
DROP:
ALTER TABLE Orders
DROP FOREIGN KEY FK_PersonOrder;
```

The `CREATE INDEX` statement is used to create indexes in tables.

```
CREATE INDEX index_name
ON table_name (column1, column2, ...);
CREATE INDEX idx_lastname
ON Persons (LastName);
CREATE INDEX idx_pname
ON Persons (LastName, FirstName);
ALTER TABLE table_name
DROP INDEX index_name;
```

scalability refers to its ability to **handle increasing amounts of data and user requests without compromising performance or availability**
data redundancy refers to **storing the same information in multiple places within a database**

- **Normalization:**
  This database design technique helps reduce data redundancy and dependency, which can improve data integrity and performance.

  particularly from **Unnormalized Form (UNF) → 1NF → 2NF → 3NF**.

  Unnormalized Form (UNF)

StudentID	Name	Courses
1	Alice	Math, English
2	Bob	Science, Math

  Problem: Multiple values in one field (Courses). This violates 1NF.

  First Normal Form (1NF)

  Break multi-valued fields into separate rows:

StudentID	Name	Course
1	Alice	Math

StudentID	Name	Course
1	Alice	English
2	Bob	Science
2	Bob	Math

- **Fixed**: Each cell has atomic (single) values.

Second Normal Form (2NF)

Remove **partial dependencies** by splitting into two tables:

Students Table:

StudentID	Name
1	Alice
2	Bob

Courses Table:

StudentID	Course
1	Math
1	English
2	Science
2	Math

- **Fixed**: Student name only depends on StudentID, not on Course.

Third Normal Form (3NF)

Create a separate **Courses** table with Course IDs to remove transitive dependencies.

🎓 Students Table:

StudentID	Name
1	Alice
2	Bob

📝 Enrollments Table:

StudentID	CourseID
1	101
1	102
2	103
2	101

☐ Courses Table:

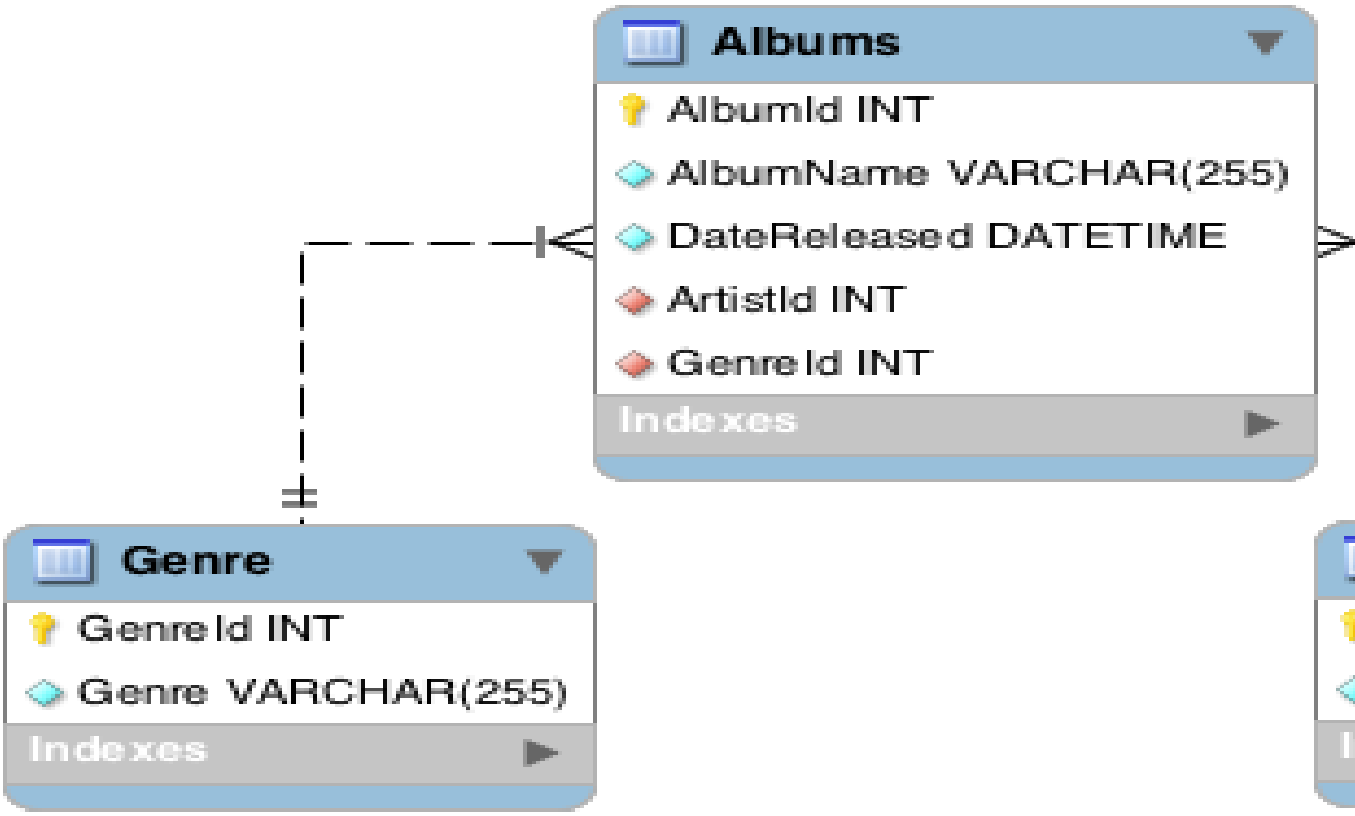
CourseID	CourseName
101	Math
102	English
103	Science

🩹 **Fixed**: All non-key columns depend **only on the key**.

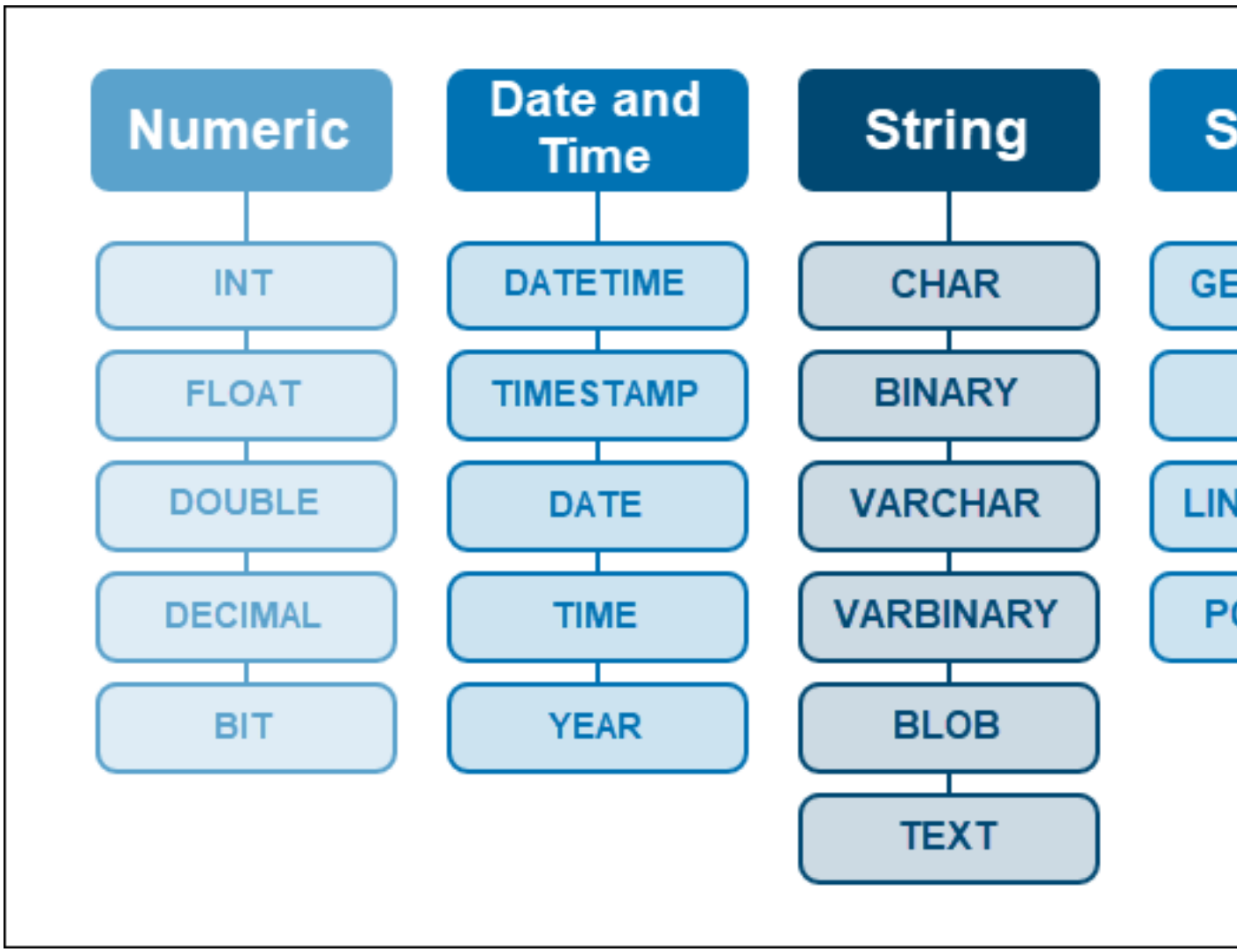
✅ Benefits of Normalization:

- ✅ Removes redundancy
- ✅ Improves data consistency

- ✓ Makes queries easier and more accurate

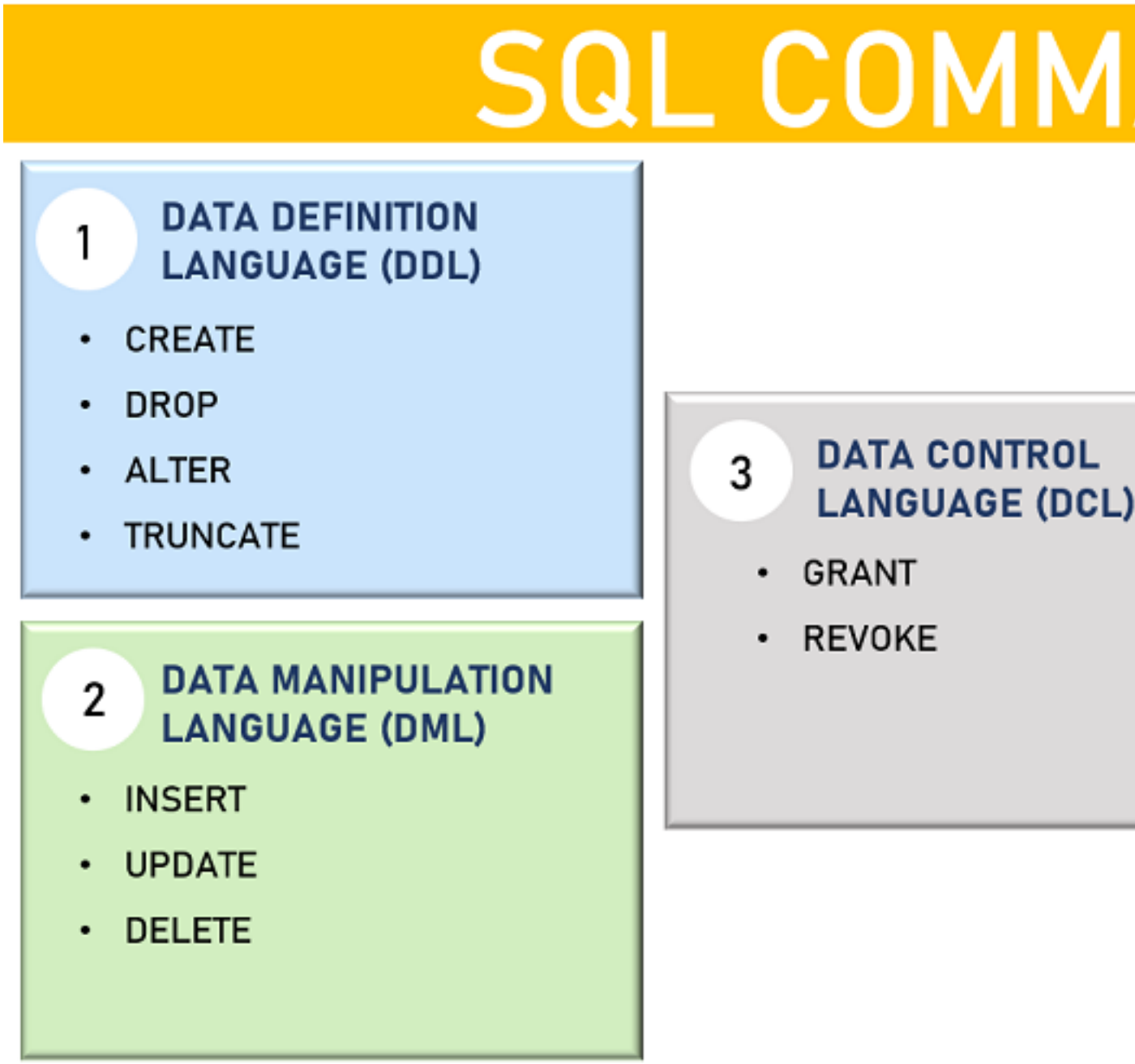


==========
==========
==========.


Numeric
INT
FLOAT
DOUBLE
DECIMAL
BIT
Date and Time
DATETIME
TIMESTAMP
DATE
TIME
YEAR
String
CHAR
BINARY
VARCHAR
VARBINARY
BLOB
TEXT

MySQL DATA TYPES

DATE TYPE	SPEC	DATA TYPE
CHAR	String (0 - 255)	INT
VARCHAR	String (0 - 255)	BIGINT
TINYTEXT	String (0 - 255)	FLOAT
TEXT	String (0 - 65535)	DOUBLE
BLOB	String (0 - 65535)	DECIMAL
MEDIUMTEXT	String (0 - 16777215)	DATE
MEDIUMBLOB	String (0 - 16777215)	DATETIME
LONGTEXT	String (0 - 4294967295)	TIMESTAMP
LONGBLOB	String (0 - 4294967295)	TIME
TINYINT	Integer (-128 to 127)	ENUM
SMALLINT	Integer (-32768 to 32767)	SET
MEDIUMINT	Integer (-8388608 to 8388607)	BOOLEAN



Single line comments start with --.

Multi-line comments start with /* and end with */.

arithmetic, comparison, logical, and others.

1. Arithmetic Operators:

- **+ (Addition):** Adds two values.
- **- (Subtraction):** Subtracts one value from another.

- *** (Multiplication):** Multiplies two values.
- **\ (Division):** Divides one value by another.
- **% or MOD (Modulo):** Returns the remainder of the division.

2. Comparison Operators:

- **= (Equal):** Checks if two values are equal.
- **!= or <> (Not Equal):** Checks if two values are not equal.
- **< (Less Than):** Checks if one value is less than another.
- >** (Greater Than):** Checks if one value is greater than another.
- **<= (Less Than or Equal To):** Checks if one value is less than or equal to another.
- >> (Greater Than or Equal To): Checks if one value is greater than or equal to another.
- **LIKE:** Checks if a string matches a pattern.
- **IN:** Checks if a value is within a list of values.
- **BETWEEN ... AND ...:** Checks if a value falls within a range.

3. Logical Operators:

- **AND:** Returns TRUE if both operands are TRUE.
- **OR:** Returns TRUE if at least one operand is TRUE.
- **NOT:** Reverses the value of an operand.

4. Other Operators:

- **EXISTS:** Checks if a subquery returns any records.
- **ANY:** Used in conjunction with comparison operators to check if the condition is true for at least one value in a subquery.

- **ALL:** Used in conjunction with comparison operators to check if the condition is true for all values in a subquery.

DDL

The CREATE DATABASE statement is used to create a new SQL database.

Syntax:

CREATE DATABASE *databasename*;

CREATE DATABASE testDB;

SHOW DATABASES;

The DROP DATABASE statement is used to drop an existing SQL database. Ex: DROP DATABASE *databasename*;

Note: Be careful before dropping a database. Deleting a database will result in loss of complete information stored in the database!

==========

The CREATE TABLE statement is used to create a new table in a database.

```
CREATE TABLE table_name (
    column1 datatype,
    column2 datatype,
    column3 datatype,
   ....
);
```

The column parameters specify the names of the columns of the table.

```
CREATE TABLE Persons (
    PersonID int,
    LastName varchar(255),
    FirstName varchar(255),
    Address varchar(255),
    City varchar(255)
);
```

DESCRIBE Tables;

The DROP TABLE statement is used to drop an existing table in a database.

DROP TABLE *table_name*;

DROP TABLE Shippers;

The TRUNCATE TABLE statement is used to **delete the data inside a table, but not the table itself**.

TRUNCATE TABLE *table_name*;

truncate table students;

The ALTER TABLE statement is used to add, delete, or modify columns in an **existing table.**

The ALTER TABLE statement is also used to add and drop various constraints on an existing table.

To add a column in a table

ALTER TABLE *table_name* ADD *column_name datatype*;

Ex:

ALTER TABLE Customers ADD Email varchar(255);

ALTER TABLE - DROP COLUMN

ALTER TABLE *table_name*
DROP COLUMN *column_name*;

Ex:

ALTER TABLE Customers DROP COLUMN Email;

ALTER TABLE - MODIFY COLUMN

To change the **data type of a column** in a table

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

Ex:

ALTER TABLE Persons MODIFY COLUMN DateOfBirth year;

Change Data Type

ALTER TABLE Persons MODIFY COLUMN DateOfBirth year;

==========

Rename the columns

Syntax:

Alter table table_name change column old_name new_name datatype;

Ex:

Alter table students change column depart department varchar(20);

Aspect	TRUNCATE	DELETE
Description	Removes all data from a table	Removes specific row from a table
Rollback	Cannot be rolled back	Can be rolled back
WHERE Clause	Not applicable	Optional
Table Structure	Preserves table structure	Preserves table structure

DML

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

It is possible to write the INSERT INTO statement in two ways:

1. Specify both the column names and the values to be inserted:

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

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

```
INSERT INTO Customers (CustomerName, ContactName, Address, City, PostalCode, Country)
VALUES ('Cardinal', 'Tom B. Erichsen', 'Skagen 21', 'Stavanger', '4006', 'Norway');
```

```
INSERT INTO Employees (EmployeeID, EmployeeName, Age, Department)
```

```
VALUES
    (1, 'John Doe', 30, 'Engineering'),
    (2, 'Jane Smith', 28, 'Marketing'),
    (3, 'Sam Brown', 35, 'Sales'),
    (4, 'Lucy Green', 25, 'Human Resources');
```

The UPDATE statement is used to modify the existing records in a table.

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

Note: Be careful when updating records in a table! Notice the WHERE clause in the UPDATE statement.
The WHERE clause specifies which record(s) that should be updated. If you omit the WHERE clause, all records in the table will be updated!

```
UPDATE Customers
SET ContactName = 'Alfred Schmidt', City = 'Frankfurt'
WHERE CustomerID = 1;
```

The DELETE statement is used to delete existing records in a table.

```
DELETE FROM table_name WHERE condition;
```

Note: Be careful when deleting records in a table! Notice the WHERE clause in the DELETE statement.
The WHERE clause specifies which record(s) should be deleted. If you omit the WHERE clause, all records in the table will be deleted!

```
DELETE FROM Customers WHERE CustomerName='Alfreds Futterkiste';
```

What is a NULL Value?

A field with a NULL value is a field with no value.

How to Test for NULL Values?

It is not possible to test for NULL values with comparison operators, such as =, <, or <>.

We will have to use the IS NULL and IS NOT NULL operators instead.

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

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

Transaction Control Language (TCL)

Transaction control is essential for managing changes in a **database** effectively.

COMMIT and **ROLLBACK** are two crucial **Transaction Control Language** (TCL) commands that help maintain **data integrity** and **consistency**.

COMMIT ensures that all changes in a transaction are permanently saved,

After executing a COMMIT statement, the changes are **irreversible**, and the database **cannot revert** to its **previous state**.

ROLLBACK provides a mechanism to undo changes when something goes wrong.

```
SET autocommit=0;
START TRANSACTION;
```

#code

COMMIT;

ROLLBACK;

BEGIN;

SAVEPOINT samplesavepoint;

-- Do some queries here, e.g.:

UPDATE users SET name = 'John' WHERE id = 1;

ROLLBACK TO SAVEPOINT samplesavepoint;

-- Continue doing other operations or COMMIT;

COMMIT;

DQL

The SELECT statement is used to select data from a database.

The data returned is stored in a result table, called the result-set.

Syntax:

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

SELECT * FROM *table_name*;

* selects ALL the columns from the table

Select Particular columns from the table

SELECT CustomerName, City,
Country FROM Customers;

SELECT DISTINCT Statement

The SELECT DISTINCT statement is used to return only distinct (different) values.

Inside a table, a column often **contains many duplicate values**; and sometimes you only want to list the different (distinct) values.

SELECT DISTINCT *column1, column2, ...*
FROM *table_name*;

SELECT DISTINCT Country FROM Customers;

The MySQL WHERE Clause

WHERE Syntax

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

The WHERE clause is not only used
in SELECT statements, it is also used
in UPDATE, DELETE, etc.!

SELECT * FROM Customers
WHERE Country = 'Mexico'; Text fields

SELECT * FROM Customers
WHERE CustomerID = 1; numeric fields

The MySQL AND, OR and NOT Operators

The AND and OR operators are used to filter records based on more than one condition:

- The AND operator displays a record if all the conditions separated by AND are TRUE.
- The OR operator displays a record if any of the conditions separated by OR is TRUE.

The NOT operator displays a record if the condition(s) is NOT TRUE.

AND Syntax

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

```
SELECT * FROM Customers WHERE Country
= 'Germany' AND City = 'Berlin';
```

OR Syntax

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

```
SELECT * FROM Customers WHERE City = 'Berlin' OR City
= 'Stuttgart';
```

NOT Syntax

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

```
SELECT * FROM Customers WHERE NOT Country
= 'Germany';
```

```
SELECT * FROM Customers
WHERE Country = 'Germany' AND (City
= 'Berlin' OR City = 'Stuttgart');
```

```
SELECT * FROM Customers
WHERE NOT Country = 'Germany' AND NOT Country
= 'USA';
```

The MySQL IN Operator

The IN operator allows you to specify multiple values in a WHERE clause.

The IN operator is a shorthand for multiple OR conditions.

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

```
SELECT * FROM Customers
WHERE Country IN ('Germany', 'France', 'UK');
```

```
SELECT * FROM Customers
WHERE Country NOT IN ('Germany', 'France', 'UK');
```

==========

The MySQL ORDER BY Keyword

The ORDER BY keyword is used to sort the result-set in ascending or descending order.

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

```
SELECT * FROM Customers
ORDER BY Country DESC;
```

```
SELECT * FROM Customers
ORDER BY Country ASC, CustomerName DESC;
```

The MySQL LIMIT Clause

The LIMIT clause is used to specify the number of records to return.

The LIMIT clause is useful on large tables with thousands of records. Returning a large number of records can impact performance.

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

```
SELECT * FROM Customers
LIMIT 3;
```

SELECT * FROM Customers LIMIT 3 OFFSET 3;

return only 3 records, start on record 4 (OFFSET 3)":

```
SELECT * FROM Customers
WHERE Country='Germany'
LIMIT 3;
```

```
SELECT * FROM Customers
ORDER BY Country
LIMIT 3;
```

What is a NULL Value?

A field with a NULL value is a field with no value.

How to Test for NULL Values?

It is not possible to test for NULL values with comparison operators, such as =, <, or <>.

We will have to use the IS NULL and IS NOT NULL operators instead.

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

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

The MySQL BETWEEN Operator

The BETWEEN operator selects values within a given range. The values can be numbers, text, or dates.

The BETWEEN operator is inclusive: begin and end values are included.

BETWEEN Syntax

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

```
SELECT * FROM Products
WHERE Price BETWEEN 10 AND 20;
```

```
SELECT * FROM Orders
WHERE OrderDate BETWEEN '1996-07-01' AND '1996-07-31';
```

=	Equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
<>	Not equal. **Note:** In some versions of SQL this operator
BETWEEN	Between a certain range SELECT * FROM Products WHE
LIKE	Search for a pattern

IN	To specify multiple possible values for a column

The MySQL LIKE Operator

The LIKE operator is 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**

R_ja

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

WHERE CustomerName LIKE 'a%'	Finds any values that start with
WHERE CustomerName LIKE '%a'	Finds any values that end with '
WHERE CustomerName LIKE '%or%'	Finds any values that have "or"
WHERE CustomerName LIKE '_r%'	Finds any values that have "r" i
WHERE CustomerName LIKE 'a_%'	Finds any values that start with in length

WHERE CustomerName LIKE 'a__%'	Finds any values that start with in length
WHERE ContactName LIKE 'a%o'	Finds any values that start with

```
SELECT * FROM Customers
WHERE CustomerName LIKE 'a%';
```

```
SELECT * FROM Customers
WHERE CustomerName LIKE '%a';
```

```
SELECT * FROM Customers
WHERE CustomerName LIKE '%or%';
```

selects all customers with a CustomerName that have "r" in the second position:

```
SELECT * FROM Customers
WHERE CustomerName LIKE '_r%';
```

```
SELECT * FROM Customers
WHERE CustomerName LIKE 'a__%';
```

```
SELECT * FROM Customers
WHERE ContactName LIKE 'a%o';
```

```
SELECT * FROM Customers
WHERE CustomerName NOT LIKE 'a%';
```

MySQL Aliases

Aliases are used to give a t**able, or a column in a table, a temporary name.**

An alias only **exists for the duration of that query.**

An alias is created with **the AS keyword.**

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

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

SELECT CustomerID AS ID,
CustomerName AS Customer
FROM Customers;

SQL Aggregate Functions

An aggregate function is a function that performs a calculation on a set of values, and returns a single value.

Aggregate functions are often used with the GROUP BY clause of the SELECT statement. The GROUP BY clause splits the result-set into groups of values and the aggregate function can be used to return a single value for each group.

The most commonly used SQL aggregate functions are:

- MIN() - returns the smallest value within the selected column
- MAX() - returns the largest value within the selected column
- COUNT() - returns the number of rows in a set
- SUM() - returns the total sum of a numerical column
- AVG() - returns the average value of a numerical column

Syntax:

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

SELECT MIN(Price) AS SmallestPrice FROM Products;

Syntax:

SELECT MAX(column_name) FROM table_name WHERE condition;

SELECT MAX(Price) AS LargestPrice FROM Products;
```

COUNT() Syntax

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

AVG() Syntax

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

SUM() Syntax

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

The MySQL GROUP BY Statement

The GROUP BY statement groups rows that have the same values into summary rows, like "find the number of customers in each country".

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);
```

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country;
```

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country
ORDER BY COUNT(CustomerID) DESC;
```

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

HAVING Syntax

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

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country
HAVING COUNT(CustomerID) > 5;
```

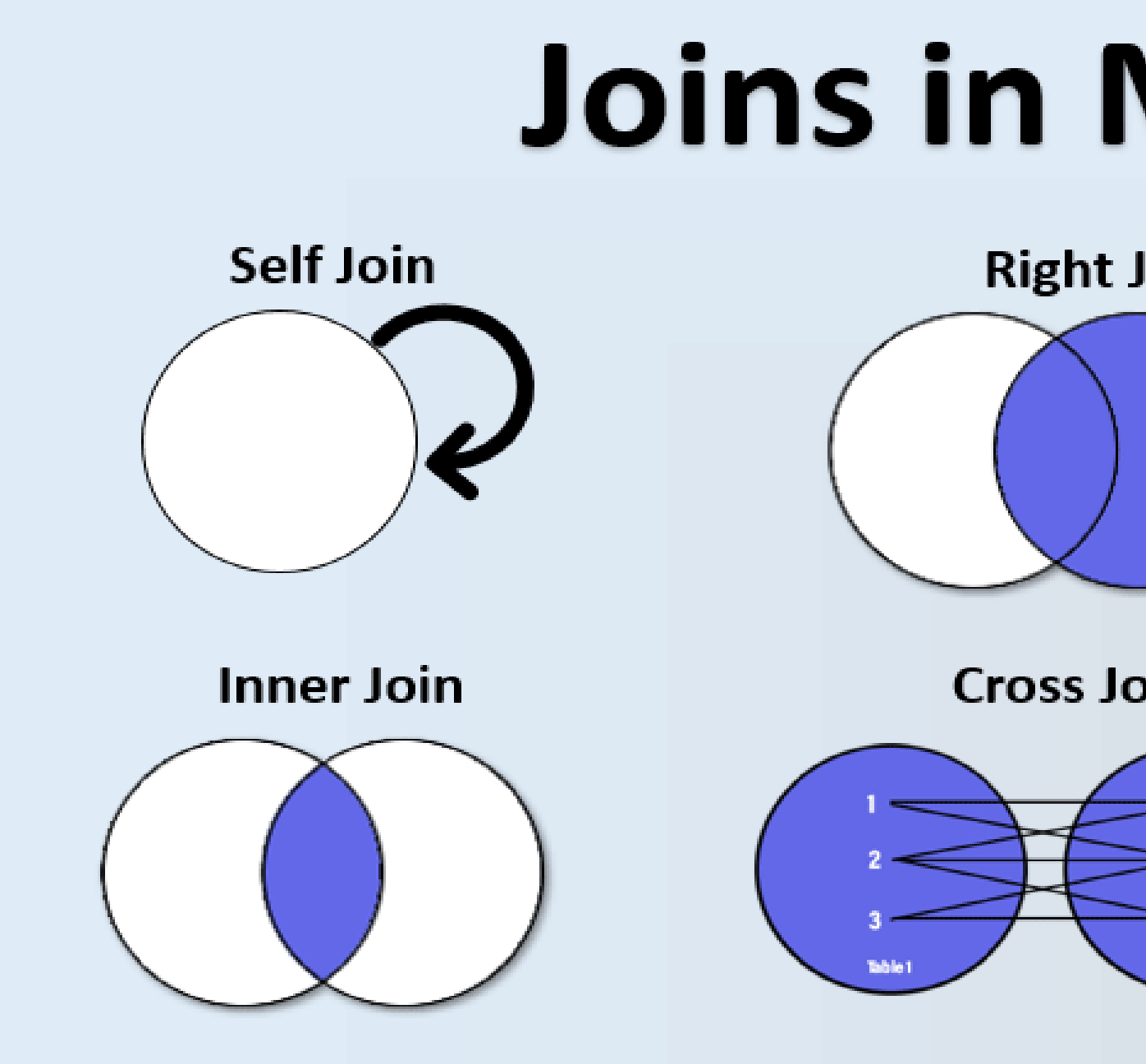
```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country
HAVING COUNT(CustomerID) > 5
ORDER BY COUNT(CustomerID) DESC;
```

MySQL Joins

A `JOIN` clause is used to combine rows from two or more tables, based on a related column between them.

It allows for efficient **data retrieval** by enabling the extraction of related information from multiple tables in a single query

- JOINs help maintain **referential integrity** by ensuring that relationships between tables are respected and **data consistency** is preserved.



Syntax:

SELECT column_names
FROM table1
INNER JOIN table2
ON table1.common_column = table2.common_column;

select t1.col1,t1.col2,t2.col4,t2.col5
 from table1 as t1 join table2 as t2 on t1.column = t2.column

https://dataschool.com/how-to-teach-people-sql/left-right-join-animated/

1. INNER JOIN

It Returns records that have matching values in both tables.

```
SELECT employees.name, departments.department_name
FROM employees
INNER JOIN departments
ON employees.department_id = departments.department_id;
```

2. LEFT JOIN

It returns all records from the Left table and matched records from the Right table. If there is no match, then NULL values are returned for Right table columns.

```
SELECT employees.name, departments.department_name
FROM employees
LEFT JOIN departments
ON employees.department_id = departments.department_id;
```

t-shirt => s
‘’ => L
‘’ => M
“” => XL

select
e.employee_id,e.name,e.phone,e.address,s.salary_amount,s.created_at

from employees as e left join salaries as s

on e.employee_id = s.emp_id;

3. RIGHT JOIN

It Returns all the rows from the right table and the matched rows from the left table. NULL values will be returned for columns from the left table when there are no matches.

```
SELECT employees.name, departments.department_name
FROM employees
RIGHT JOIN departments
ON employees.department_id = departments.department_id;
```

select
e.employee_id,e.name,e.phone,e.address,s.salary_amount,s.created_at

from employees as e right join salaries as s

on e.employee_id = s.emp_id;

4. FULL JOIN

It Returns all records when there is a match in either the left or the right table. In case of no match, NULL values are returned for columns that have no match in either table.

5. CROSS JOIN

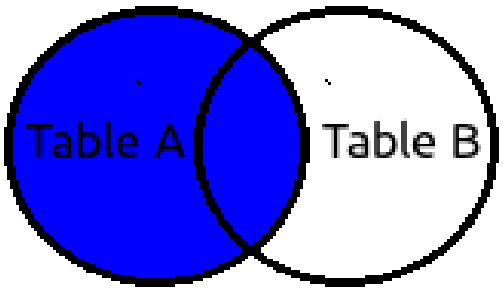
It Returns the Cartesian product of two tables. Matches every row of one table with every row of another table.

```
SELECT employees.name, departments.department_name
FROM employees
CROSS JOIN departments;
```

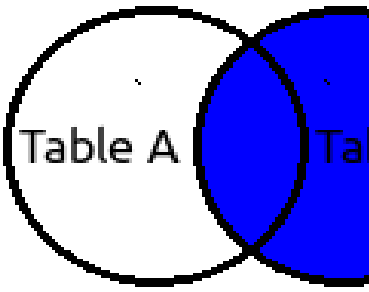
6. SELF JOIN

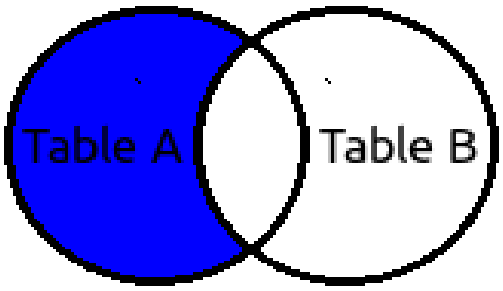
A **self join** is a type of join in which a table is joined to itself. This is useful when you need to compare rows within the same table, such as relating employees to their managers in an organizational hierarchy.

```
SELECT a.name AS employee, b.name AS manager
FROM employees a, employees b
WHERE a.manager_id = b.employee_id;
```

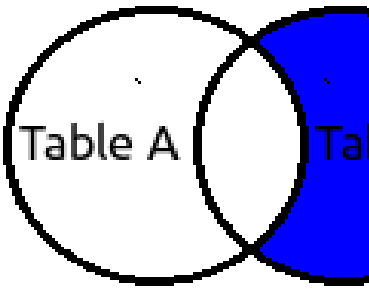


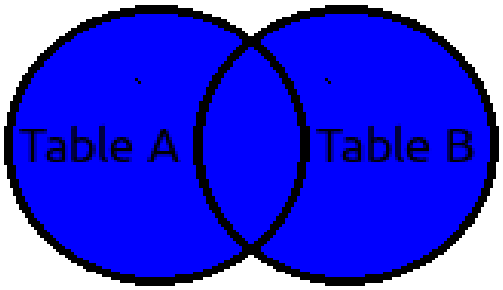

```
SELECT [list] FROM
    [Table A] A
LEFT JOIN
    [Table B] B
ON A.Value = B.Value
```



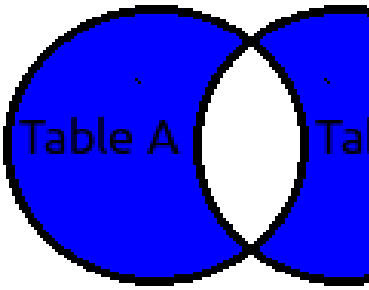


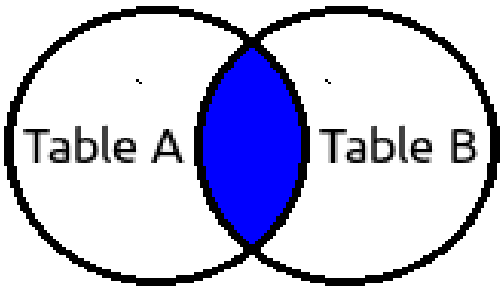

```
SELECT [list] FROM
    [Table A] A
LEFT JOIN
    [Table B] B
ON A.Value = B.Value
WHERE B.Value IS NULL
```






```
SELECT [list] FROM
    [Table A] A
FULL OUTER JOIN
    [Table B] B
ON A.Value = B.Value
```






```
SELECT [list] FROM
    [Table A] A
INNER JOIN
    [Table B] B
ON A.Value = B.Value
```

The `UNION` operator is 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 Syntax

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

The `UNION` operator selects only distinct values by default. To allow duplicate values, use `UNION ALL`:

```
SELECT column_name(s) FROM table1
UNION ALL
SELECT column_name(s) FROM table2;
```

```
select
e.employee_id,e.name,e.phone,e.address,s.salary_amount,s.created_at

from  employees as e left join salaries  as s

 on e.employee_id = s.emp_id

 union

 select
e.employee_id,e.name,e.phone,e.address,s.salary_amount,s.created_at

from  employees as e right join salaries  as s

 on e.employee_id = s.emp_id;
```

ANY and ALL

- ANY and ALL used with subq produce single column of numbers
- ALL
  - Condition only true if satisfied b produced by subquery
- ANY
  - Condition true if satisfied by produced by subquery
- If subquery empty
  - ALL returns true
  - ANY returns false
- SOME may be used in place of ANY

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

```
SELECT ProductName
FROM Products
```

```
WHERE ProductID = ANY
  (SELECT ProductID
  FROM OrderDetails
  WHERE Quantity = 10);
```

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

```
SELECT ProductName
FROM Products
WHERE ProductID = ALL
  (SELECT ProductID
  FROM OrderDetails
  WHERE Quantity = 10);
```

https://ncertlibrary.com/class-12-informatics-practices-notes-chapter-11/

MySQL String Functions

Function	Description
ASCII	Returns the ASCII value for the specific character
CHAR_LENGTH	Returns the length of a string (in characters)
CHARACTER_LENGTH	Returns the length of a string (in characters)
CONCAT	Adds two or more expressions together

CONCAT_WS	Adds two or more expressions together with a separato
FIELD	Returns the index position of a value in a list of values
FIND_IN_SET	Returns the position of a string within a list of strings
FORMAT	Formats a number to a format like "#,###,###.##", places
INSERT	Inserts a string within a string at the specified position
INSTR	Returns the position of the first occurrence of a string i
LCASE	Converts a string to lower-case
LEFT	Extracts a number of characters from a string (starting
LENGTH	Returns the length of a string (in bytes)
LOCATE	Returns the position of the first occurrence of a substri
LOWER	Converts a string to lower-case

LPAD	Left-pads a string with another string, to a certain leng
LTRIM	Removes leading spaces from a string
MID	Extracts a substring from a string (starting at any posit
POSITION	Returns the position of the first occurrence of a substri
REPEAT	Repeats a string as many times as specified
REPLACE	Replaces all occurrences of a substring within a string,
REVERSE	Reverses a string and returns the result
RIGHT	Extracts a number of characters from a string (starting
RPAD	Right-pads a string with another string, to a certain len
RTRIM	Removes trailing spaces from a string
SPACE	Returns a string of the specified number of space chara
STRCMP	Compares two strings

SUBSTR	Extracts a substring from a string (starting at any posit
SUBSTRING	Extracts a substring from a string (starting at any posit
SUBSTRING_INDEX	Returns a substring of a string before a specified numb
TRIM	Removes leading and trailing spaces from a string
UCASE	Converts a string to upper-case
UPPER	Converts a string to upper-case

MySQL Numeric Functions

Function	Description
ABS	Returns the absolute value of a number
ACOS	Returns the arc cosine of a number
ASIN	Returns the arc sine of a number
ATAN	Returns the arc tangent of one or two numbers

ATAN2	Returns the arc tangent of two numbers
AVG	Returns the average value of an expression
CEIL	Returns the smallest integer value that is >= to a num
CEILING	Returns the smallest integer value that is >= to a num
COS	Returns the cosine of a number
COT	Returns the cotangent of a number
COUNT	Returns the number of records returned by a select que
DEGREES	Converts a value in radians to degrees
DIV	Used for integer division
EXP	Returns e raised to the power of a specified number
FLOOR	Returns the largest integer value that is <= to a numbe
GREATEST	Returns the greatest value of the list of arguments

LEAST	Returns the smallest value of the list of arguments
LN	Returns the natural logarithm of a number
LOG	Returns the natural logarithm of a number, or the logar
LOG10	Returns the natural logarithm of a number to base 10
LOG2	Returns the natural logarithm of a number to base 2
MAX	Returns the maximum value in a set of values
MIN	Returns the minimum value in a set of values
MOD	Returns the remainder of a number divided by another
PI	Returns the value of PI
POW	Returns the value of a number raised to the power of a
POWER	Returns the value of a number raised to the power of a
RADIANS	Converts a degree value into radians

RAND	Returns a random number
ROUND	Rounds a number to a specified number of decimal plac
SIGN	Returns the sign of a number
SIN	Returns the sine of a number
SQRT	Returns the square root of a number
SUM	Calculates the sum of a set of values
TAN	Returns the tangent of a number
TRUNCATE	Truncates a number to the specified number of decimal

MySQL Date Functions

Function	Description
ADDDATE	Adds a time/date interval to a date and then returns th

ADDTIME	Adds a time interval to a time/datetime and then retur
CURDATE	Returns the current date
CURRENT_DATE	Returns the current date
CURRENT_TIME	Returns the current time
CURRENT_TIMESTAMP	Returns the current date and time
CURTIME	Returns the current time
DATE	Extracts the date part from a datetime expression
DATEDIFF	Returns the number of days between two date values
DATE_ADD	Adds a time/date interval to a date and then returns th
DATE_FORMAT	Formats a date
DATE_SUB	Subtracts a time/date interval from a date and then ret
DAY	Returns the day of the month for a given date

DAYNAME	Returns the weekday name for a given date
DAYOFMONTH	Returns the day of the month for a given date
DAYOFWEEK	Returns the weekday index for a given date
DAYOFYEAR	Returns the day of the year for a given date
EXTRACT	Extracts a part from a given date
FROM_DAYS	Returns a date from a numeric datevalue
HOUR	Returns the hour part for a given date
LAST_DAY	Extracts the last day of the month for a given date
LOCALTIME	Returns the current date and time
LOCALTIMESTAMP	Returns the current date and time
MAKEDATE	Creates and returns a date based on a year and a num
MAKETIME	Creates and returns a time based on an hour, minute,

MICROSECOND	Returns the microsecond part of a time/datetime
MINUTE	Returns the minute part of a time/datetime
MONTH	Returns the month part for a given date
MONTHNAME	Returns the name of the month for a given date
NOW	Returns the current date and time
PERIOD_ADD	Adds a specified number of months to a period
PERIOD_DIFF	Returns the difference between two periods
QUARTER	Returns the quarter of the year for a given date value
SECOND	Returns the seconds part of a time/datetime
SEC_TO_TIME	Returns a time value based on the specified seconds
STR_TO_DATE	Returns a date based on a string and a format
SUBDATE	Subtracts a time/date interval from a date and then ret

SUBTIME	Subtracts a time interval from a datetime and then retu
SYSDATE	Returns the current date and time
TIME	Extracts the time part from a given time/datetime
TIME_FORMAT	Formats a time by a specified format
TIME_TO_SEC	Converts a time value into seconds
TIMEDIFF	Returns the difference between two time/datetime expr
TIMESTAMP	Returns a datetime value based on a date or datetime v
TO_DAYS	Returns the number of days between a date and date "
WEEK	Returns the week number for a given date
WEEKDAY	Returns the weekday number for a given date
WEEKOFYEAR	Returns the week number for a given date
YEAR	Returns the year part for a given date

YEARWEEK	Returns the year and week number for a given date

MySQL Advanced Functions

Function	Description
BIN	Returns a binary representation of a number
BINARY	Converts a value to a binary string
CASE	Goes through conditions and return a value when the fi
CAST	Converts a value (of any type) into a specified datatype
COALESCE	Returns the first non-null value in a list
CONNECTION_ID	Returns the unique connection ID for the current conne
CONV	Converts a number from one numeric base system to a
CONVERT	Converts a value into the specified datatype or characte

CURRENT_USER	Returns the user name and host name for the MySQL a the current client
DATABASE	Returns the name of the current database
IF	Returns a value if a condition is TRUE, or another value
IFNULL	Return a specified value if the expression is NULL, othe
ISNULL	Returns 1 or 0 depending on whether an expression is
LAST_INSERT_ID	Returns the AUTO_INCREMENT id of the last row that h
NULLIF	Compares two expressions and returns NULL if they are returned
SESSION_USER	Returns the current MySQL user name and host name
SYSTEM_USER	Returns the current MySQL user name and host name
USER	Returns the current MySQL user name and host name
VERSION	Returns the current version of the MySQL database