PHP and MySQL

Database

This project is funded by
the European Union

Maharah
Code Your Future

LibyanSpider

What is MySQL?

MySQL is an open-source relational database management system (RDBMS) that allows for efficient storage, management, and retrieval of structured data. It is widely used in web development, powering many popular websites and applications.

Databases are a tool to persist data that you intend to refer to regularly and need to keep for a long time.

Why Use MySQL for Database Management?

There are several reasons why MySQL is a preferred choice for database management:

- **Open-Source:** MySQL is an open-source database system, which means it is freely available and can be customized and modified to meet specific requirements.
- **Reliability**: MySQL is known for its stability and reliability, with a proven track record of handling large-scale databases and high-traffic websites.
- **Scalability**: MySQL is designed to scale efficiently, allowing for the management of growing datasets and handling increased user demand without sacrificing performance.
- **Cross-Platform Compatibility**: MySQL is available for various platforms, including Windows, macOS, Linux, and more, making it versatile and easily deployable in different environments.

MySQL Features and Advantages for Developers

- **SQL Support**: MySQL supports the Structured Query Language (SQL), a standard language for managing relational databases. This enables developers to perform complex queries, retrieve data, manipulate tables, and more.
- **Data Security**: MySQL provides robust security features, including user authentication, access control, and encryption, ensuring that sensitive data is protected.
- **High Performance**: MySQL is optimized for performance, offering features like indexing, caching, and query optimization techniques. This allows developers to retrieve and manipulate data efficiently, resulting in faster response times.

MySQL Features and Advantages for Developers (2)

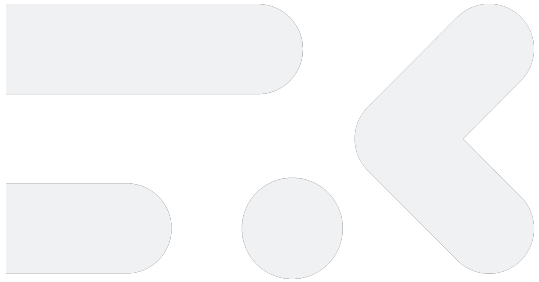
- **Replication and High Availability**: MySQL supports replication, which allows for the replication of data across multiple servers, providing redundancy and high availability. This ensures that even in the event of a server failure, the database remains accessible.

- **Community and Support:** MySQL has a large and active community of developers, providing support, documentation, and resources. This makes it easier for developers to find solutions and receive assistance when facing challenges.

Installing MySQL

There are several ways to get a working MySQL installation:

- Download and install from the official MySQL website: (https://www.mysql.com)
- It is bundled with the development environment, like **XAMPP**.
- Provided by the hosting service in production.



phpMyAdmin is a powerful web-based tool designed for seamless and intuitive management of MySQL (or MariaDB) databases. It serves as the bridge between developers and their databases, offering a user-friendly interface for efficient administration and manipulation of data.

Web-Based Interface

Access your database from anywhere with an internet connection, eliminating the need for a direct server connection.

Multi-Database Support

Manage multiple databases effortlessly, streamlining your workflow when dealing with complex projects.

Query Execution

Execute SQL queries directly through the interface, providing flexibility for developers and database administrators.

Data Import/Export

Easily import and export data in various formats, facilitating data migration and backup processes.

User Privilege Management

Fine-tune access levels for users, ensuring security while collaborating on database projects.

Table and Relationship Management

Create, modify, or delete tables and establish relationships between them with a few clicks.

Database Basics

Let's begin by making sure that some of the concepts of relational databases are clear.

Keys:

Keys impose constraints, such as PRIMARY and UNIQUE. A primary key can be defined on either a single or multiple columns. It guarantees that each row in the database will have a unique value combination for the columns in the key. A row may not have a null value for its primary key.

A table may have only one primary key. A foreign key can also be defined on either a single or multiple columns. It references the primary key on another table. This is a unique reference, so only one row in the referenced table will be linked to the table containing the foreign key.

Database Basics (2)

Indexes:

- Indexes are data structures that are needed to implement the key constraint. Indexes make retrieving records faster.
- The database engine will create a structure on disk or in memory that contains the data from the indexed columns. This structure is optimized for lookups and helps the database find the row in the table faster.
- Whenever you insert a row into a table, the indexes need to be updated. This adds an overhead to writing.
- You cannot have a key without an index, but it is possible to index columns that are not keyed. You would do this in cases where you don't want to enforce uniqueness but do want to speed up SELECT statements that include these columns in their WHERE clauses.
- The binding between keys and indexes is very tight and in MySQL they are considered synonymous.

Database Basics (3)

Relationships:

Relationships are a core feature of relational databases. By declaring how tables are related, you can enforce referential integrity and minimize dirty data.
There are several types of relationships.

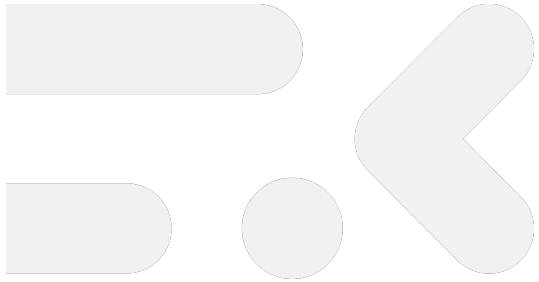
- **One-to-one**: One row in the parent table can reference exactly one row in the child table.
- **One-to-many**: One row in the parent table can be referenced by many rows in the child table.
- **Many-to-many**: Any number of rows in the parent table can be referenced by any number of rows in the child table.

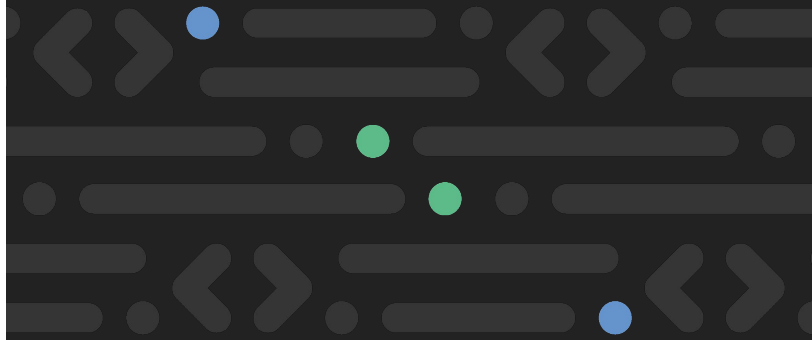
SQL Data Types

Columns in a SQL database table have a data type assigned to them. Just as with PHP variable types, SQL types can each store different formats of data.

Each database manager will implement the SQL data types slightly differently and will have different optimizations between its types.

We discuss a few of the common data types and avoid focusing on any particular implementation of SQL.





SQL Data Types: Numeric Types

The types of integers vary in the amount of bytes that they take to store their value. The following table illustrates the sizes of integers for the MySQL database:

Integer Type	Bytes	(Signed) Values	(Unsigned) Values
BIGINT	8	-9223372036854775808 to +9223372036854775807	0 to 18446744073709551615
INTEGER	4	-2147483648 to +2147483647	0 to 4294967295
MEDIUMINT	3	-8388608 to +8388607	0 to 16777215
SMALLINT	2	-32768 to +32767	0 to 65535
TINYINT	1	-128 to +127	0 to 255

SQL Data Types: Character Types

SQL allows for characters to be stored either in fixed- or variable-length strings. A fixed-length string is always allocated the same number of bytes on disk. This can help speed up read performance in some database implementations. The trade-off is that if a string being stored in a fixed-length data store is shorter than the number of characters allocated, you are storing more characters than you have to.

Variable-length strings can swell up to the limiting size given to them. The database engine allocates storage according to the length of the string.

In general, when storing a string that you know is always going to be of a particular length, such as a hash for example, you should store it in a fixed-length character field. This will improve performance and you won't incur storage waste.

Creating and Using Databases

To create a database, use the CREATE DATABASE statement. For example:

```
CREATE DATABASE maharahdatabase;
```

To switch to a specific database, use the USE statement:

```
USE maharahdatabase;
```

Creating Tables and Defining Columns

Tables are used to store data in a structured manner. To create a table, use the CREATE TABLE statement and define the columns along with their data types and constraints.

When creating a table, you can specify a list of the columns you want to store in it. For each column, you specify the name, data type, and attributes.
For example:

```
CREATE TABLE employees (
    id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(50),
    age INT,
    department VARCHAR(50)
);
```

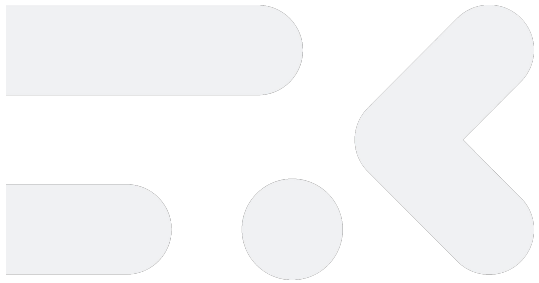
Maharah
Code Your Future

Dropping Database and Tables

The inverse of CREATE is the DROP statement.

```
DROP TABLE employees;
DROP DATABASE maharahdatabase;
```

If you have specified foreign keys, the database will not let you drop a table if this will violate one of the constraints.



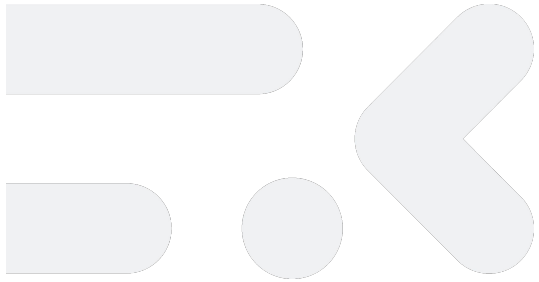
Maharah
Code Your Future

Retrieving Data using SELECT Statements

The SELECT statement is used to retrieve data from a table. For example:

```
SELECT * FROM employees;
```

This will retrieve all the records from the "employees" table.



Maharah
Code Your Future

Retrieving Data Using Aggregate Functions

Aggregate functions perform calculations on a set of values and return a single value. Common aggregate functions include COUNT, SUM, AVG, MIN, and MAX.

To count the number of records in a table, use the COUNT function:

```
SELECT COUNT(column)
FROM table;
```

To Returns the sum of records in a table, use the SUM function:

```
SELECT SUM(column)
FROM table;
```

https://dev.mysql.com/doc/refman/8.0/en/aggregate-functions.html

Filtering Data using WHERE Clause

The WHERE clause allows you to filter data based on specific conditions. For example:

```
SELECT * FROM employees WHERE age > 30;
```

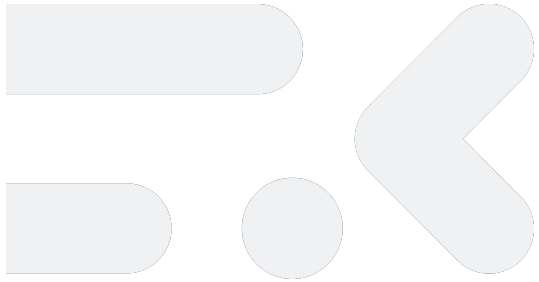
This will retrieve all the records from the "employees" table where the age is greater than 30.

Sorting Data using ORDER BY Clause

The ORDER BY clause is used to sort the retrieved data in ascending or descending order. For example:

```
SELECT * FROM employees ORDER BY age DESC;
```

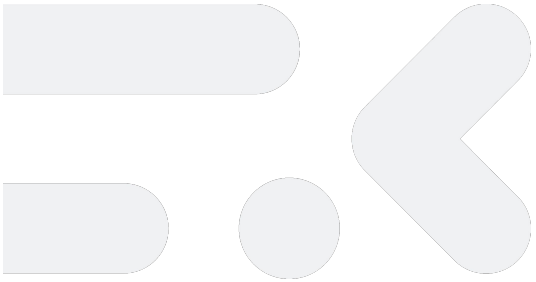
This will retrieve all the records from the "employees" table and sort them in descending order based on the "age" column.



Inserting Data into Tables

The INSERT INTO statement is used to create new rows in the database. You will need to provide a list of the columns and the values to insert to them. Columns that are marked NOT NULL are mandatory and must have a value specified when you create the row.

```
INSERT INTO employees (`id`, `name`, `age`, `department`)
VALUES (1, 'John Doe', 25, 'Sales');
```



Updating Data

The UPDATE statement accepts a list of values similar to the INSERT statement, as well as an optional WHERE clause similar to the SELECT statement.
You must specify what values to update the existing data to, and the criteria for the rows that must be updated.

```
UPDATE employees
SET salary = 5000
WHERE id = 123;
```

Deleting Data

The DELETE statement is used to remove one or more records from a table. It permanently deletes data from the table.

For example, to delete all records from the employees table where the department is 'Marketing', use the following query:

```
DELETE FROM employees
WHERE department = 'Marketing';
```

Grouping Data using GROUP BY Clause

The GROUP BY clause is used to group rows based on specific columns. It is often used in combination with aggregate functions to perform calculations on grouped data.

For example, to calculate the total sales amount for each product category, use the following query:

```
SELECT category, SUM(amount)
FROM sales
GROUP BY category;
```

Maharah
Code Your Future

Subqueries and Nested Queries

Subqueries, also known as nested queries, are queries within another query. They can be used to perform complex operations, such as retrieving data based on the result of another query.

For example, to retrieve employees who have a salary higher than the average salary, use the following query:

```
SELECT name, salary
FROM employees
WHERE salary > (SELECT AVG(salary) FROM employees);
```

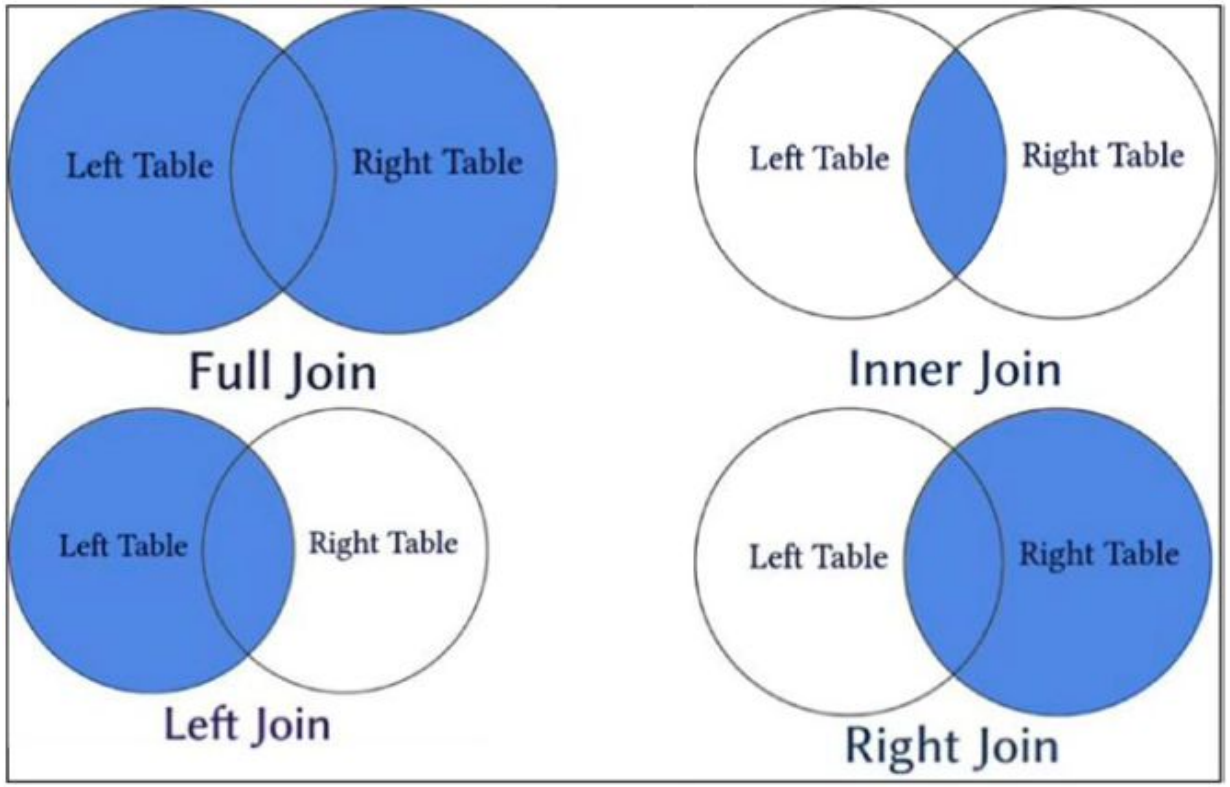
Maharah
Code Your Future

Joins

Joins are used to connect tables based on supplied criteria. This lets you retrieve information from related tables.

In the products and categories database, you can retrieve the category name of products by joining the categories table to the products table:

```
SELECT *
FROM  products
JOIN categories ON categories.id = products.category_id
```



Join Types

- **INNER JOIN**: Selects records that have matching values in both tables, as in the previous example.
- **LEFT JOIN**: Selects tables from the left table that have matching right table records
- **RIGHT JOIN**: Selects records from the right table that have matching left table records
- **FULL JOIN**: Selects all records that match either left or right table records

Primary Keys and Their Importance

Primary keys are unique identifiers for each record in a table. They ensure data integrity by enforcing uniqueness and providing a way to identify and access specific records efficiently.

To create a primary key, use the PRIMARY KEY constraint when defining the table. Only one primary key can be defined per table.

For example, to create a primary key on the "product" table's "id" column, use the following syntax:

```
// To create a PRIMARY KEY constraint on the "ID" column when the table is already created, use the following SQL:
ALTER TABLE product
ADD PRIMARY KEY (id);
// To allow naming of a PRIMARY KEY constraint, and for defining a PRIMARY KEY constraint on multiple columns, use the following SQL syntax:
ALTER TABLE product
ADD CONSTRAINT PK_product PRIMARY KEY (id);
```

Maharah
Code Your Future

What is a Foreign Key?

Foreign keys are used to establish relationships between tables in a relational database. They ensure data integrity and maintain referential integrity by enforcing relationships between tables.

A foreign key is a column or a set of columns in one table that refers to the primary key in another table. It establishes a link between the two tables, representing a parent-child relationship.

What is a Foreign Key?

To create a foreign key, use the FOREIGN KEY constraint when defining the table. The foreign key constraint references the primary key of another table, ensuring that the values in the foreign key column exist in the referenced table.

For example, to create a foreign key constraint on the "orders" table that references the "customers" table's primary key "customer_id," use the following syntax:

```
ALTER TABLE orders
ADD CONSTRAINT fk_customer_id
FOREIGN KEY (customer_id) REFERENCES customers(customer_id);
```

Maharah
Code Your Future

Using Primary and Foreign Keys Together

Primary and foreign keys work together to establish relationships between tables. The primary key in one table becomes the foreign key in another table, creating a link between the two tables.

Importance of Key Constraints

Key constraints, including primary and foreign keys, provide several benefits, including:

- Ensuring data integrity by enforcing relationships and uniqueness.
- Facilitating efficient data retrieval and data manipulation operations.
- Simplifying data management and maintaining consistency.

Indexes

Indexes are data structures that improve the speed of data retrieval operations on database tables. They provide a quick way to locate data based on the values in one or more columns. Indexes play a crucial role in optimizing database performance.

Benefits of Indexes

- Faster data retrieval: Indexes allow the database to quickly find the desired data, reducing the time required for query execution.
- Improved query performance: By utilizing indexes, queries can be executed more efficiently, resulting in faster response times.
- Enhanced data integrity: Indexes can enforce unique constraints and ensure data consistency within the database.

Types of Indexes

MySQL supports various types of indexes, including:

- **B-tree Index**: The most common type of index, suitable for most applications.
- **Hash Index**: Optimal for exact match lookups but not for range queries.
- **Full-Text Index**: Designed for efficient text-based searching.
- **Spatial Index**: Used for spatial data types, enabling fast geographical searches.

Creating Indexes on Tables

To create an index on a table, use the CREATE INDEX statement. Specify the index name, the table name, and the column(s) to be indexed. For example:

```
CREATE INDEX idx_name ON table_name (column1, column2);
```

Removing Index

To remove an existing index, use the DROP INDEX statement. Specify the index name, the table name. For example:

```
DROP INDEX idx_name_department ON Employees;
```