FULLSTACK COHORT 3

PHP & MySQL

JOAQUIN ANTONIO

Introduction to Databases and SQL

Introduction to relational databases and their key components

1. Relational databases are a type of database management system (DBMS) that stores data in a structured format, using rows and columns in tables to represent and organize data.

2. They are based on the relational model, which was introduced by Edgar F. Codd in the 1970s.

3. Are widely used in various applications and industries due to their flexibility, scalability, and ability to enforce data integrity.

4. Popular relational database management systems (RDBMS) include MySQL, PostgreSQL, Oracle Database, Microsoft SQL Server, and SQLite.

Tables

1. A relational database organises data into tables, with each table representing a specific entity or concept.

2. For example, in a database for a university, you might have tables for students, courses, and instructors.

Employee_ID	First_Name	Last_Name	Dept_ID	Location_ID
1001	John	Jones	10	100
1002	Susan	Smith	20	100
1003	Jackson	Black	10	200
1004	Thom	Thomas	20	300
1005	Robert	Reid	10	400

Rows

1. Rows in a table represent individual records or instances of data. Each row contains data for each column defined in the table.

2. For example, a row in the "Employees" table might contain the information for a single employee, such as their employee ID, name, and department.

Employee_ID	First_Name	Last_Name	Dept_ID	Location_ID
1001	John	Jones	10	100
1002	Susan	Smith	20	100
1003	Jackson	Black	10	200
1004	Thom	Thomas	20	300
1005	Robert	Reid	10	400

Column

1. Each table consists of columns, which define the attributes or properties of the data stored in the table.

2. For example, an "employees" table could have columns for Employee ID, name, and department.

Employee_ID	First_Name	Last_Name	Dept_ID	Location_ID
1001	John	Jones	10	100
1002	Susan	Smith	20	100
1003	Jackson	Black	10	200
1004	Thom	Thomas	20	300
1005	Robert	Reid	10	400

Relationships

1. Employees and Departments

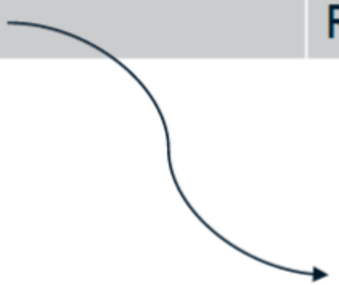
Employee_ID	First_Name	Last_Name	Dept_ID	Location_ID
1001	John	Jones	10	100
1002	Susan	Smith	20	100
1003	Jackson	Black	10	200
1004	Thom	Thomas	20	300
1005	Robert	Reid	10	400

Dept_ID	Name
10	Human Resources
20	Sales

Relationships

1. Employees and Job Histories

Employee_ID	First_Name	Last_Name	Dept_ID	Location_ID
1001	John	Jones	10	100
1002	Susan	Smith	20	100
1003	Jackson	Black	10	200
1004	Thom	Thomas	20	300
1005	Robert	Reid	10	400



Employee_ID	Position_ID	Start_Date	End_Date
1005	2011	20180824	20200105
1005	2015	20200106	NULL

EXERCISE

1. Setting up a local MySQL database

Constraints

1. Constraints in SQL are rules that are enforced on data columns in a table, ensuring the accuracy and reliability of the data.

2. Constraints help maintain the integrity of the database by preventing invalid data from being inserted, updated, or deleted.

NOT NULL

1. Ensures that a column cannot have a NULL value.
2. It forces the column to have a value in every row.

```
CREATE TABLE students (
    student_id INT PRIMARY KEY,
    name VARCHAR(50) NOT NULL
);
```

UNIQUE

1. Ensures that all values in a column are unique and cannot be duplicated in other rows in the same column.

```
CREATE TABLE employees (
    employee_id INT PRIMARY KEY,
    email VARCHAR(50) UNIQUE
);
```

PRIMARY KEY

1. Uniquely identifies each record in a table and ensures that the column cannot have a NULL value.

```
CREATE TABLE orders (
    order_id INT PRIMARY KEY,
    product_name VARCHAR(50),
    quantity INT
);
```

FOREIGN KEY

1. Establishes a relationship between two tables.
2. Ensures that the values in a column match the values in another table's column, usually the primary key of that table.

```
CREATE TABLE orders (
    order_id INT PRIMARY KEY,
    product_id INT,
    FOREIGN KEY (product_id) REFERENCES products(product_id)
);
```

CHECK

1. Ensures that the values in a column meet a specific condition.

```
CREATE TABLE employees (
    employee_id INT PRIMARY KEY,
    age INT CHECK (age >= 18)
);
```

DEFAULT

1. Provides a default value for a column when no value is specified.

```
CREATE TABLE students (
    student_id INT PRIMARY KEY,
    status VARCHAR(10) DEFAULT 'active'
);
```

PRIMARY KEY CONSTRAINT

```
CREATE TABLE Contact(
 ContactID int AUTO_INCREMENT
 PRIMARY KEY,
 LastName varchar(50) NOT NULL,
 FirstName varchar(50),
 Address varchar(50),
 City varchar(50)
);
```

```
CREATE TABLE Contact(
 ContactID int AUTO_INCREMENT NOT
 NULL,
 LastName varchar(50) NOT NULL,
 FirstName varchar(50),
 Address varchar(50),
 City varchar(50),
 CONSTRAINT pk_contact PRIMARY
 KEY (ContactID)
);
```

PRIMARY KEY CONSTRAINT

```
ALTER TABLE Contact
ADD PRIMARY KEY (ContactID);
```

```
ALTER TABLE Contact
ADD CONSTRAINT pk_contact
PRIMARY KEY (ContactID);
```

TO DELETE

```
ALTER TABLE Contact DROP PRIMARY KEY;
```

FOREIGN KEY CONSTRAINT

```
CREATE TABLE Contact(
 ContactID int AUTO_INCREMENT PRIMARY
 KEY,
 LastName varchar(50) NOT NULL,
 FirstName varchar(50),
 Address varchar(50),
 City varchar(50),
 CountryID int,
 FOREIGN KEY (CountryID) REFERENCES
 Country(CountryID)
);
```

```
CREATE TABLE Contact(
 ContactID int AUTO_INCREMENT NOT
 NULL,
 LastName varchar(50) NOT NULL,
 FirstName varchar(50),
 Address varchar(50),
 City varchar(50),
 CountryID int NOT NULL,
 CONSTRAINT fk_contact_country
 FOREIGN KEY(CountryID)
 REFERENCES Country(CountryID)
);
```

FOREIGN KEY CONSTRAINT

```
ALTER TABLE Contact
ADD FOREIGN KEY (CountryID)
REFERENCES Country (CountryID);
```

```
ALTER TABLE Contact
ADD CONSTRAINT
fk_country_contact
FOREIGN KEY (CountryID)
REFERENCES Country
(CountryID);
```

UNIQUE CONSTRAINT

```
CREATE TABLE Contact(
 ContactID int AUTO_INCREMENT
 PRIMARY KEY,
 Name varchar(50) NOT NULL,
 EmailAddress varchar(50) UNIQUE NOT
 NULL,
 City varchar(50),
 CountryID int FOREIGN KEY
 REFERENCES Country(CountryID)
);
```

```
CREATE TABLE Contact(
 ContactID int AUTO_INCREMENT NOT
 NULL,
 Name varchar(50) NOT NULL,
 EmailAddress varchar(50),
 City varchar(50),
 CountryID int NOT NULL,
 CONSTRAINT uq_email
 UNIQUE(EmailAddress)
);
```

UNIQUE CONSTRAINT WITH ALTER TABLE

```
ALTER TABLE Contact
ADD UNIQUE(EmailAddress);
```

```
ALTER TABLE Contact
ADD CONSTRAINT uq_Email
UNIQUE(EmailAddress);
```

EXERCISE

1. Create a database CourseDB
2. In the CourseDB database create a table suppliers:
    1. Primary key supplierid int
    2. Other fields:
        1. `name varchar(40)`
        2. `country varchar(40)`
        3. `city varchar(40)`
3. Create a table products:
    1. Primary key productid int
    2. Other fields:
        1. `name varchar(50)`
        2. `price decimal(6,2)`
        3. `supplierid int  (foreign key references the suppliers table)`

Understanding the role of SQL (Structured Query Language) for interacting with databases

1. SQL (Structured Query Language): SQL is the standard language used to interact with relational databases.

2. It provides commands for:

    1. Creating and modifying database structures (DDL - Data Definition Language)

    2. Querying and retrieving data (DQL - Data Query Language)

    3. Manipulating data (DML - Data Manipulation Language).

Connecting to MySQL with PHP

Establishing a connection to a MySQL database using PHP's built-in functions

```
<?php
$servername = "localhost";
$username = "your_username";
$password = "your_password";
$database = "your_database";

// Create a connection
$conn = new mysqli($servername, $username, $password, $database);

// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

echo "Connected successfully";
?>
```

Writing basic SQL queries using prepared statements

1. Prepared statements in PHP provide a way to execute SQL queries with placeholders for parameters, which can help prevent SQL injection attacks and improve performance by reusing query execution plans.

1. Prepare Statement

1. To use a prepared statement, you first prepare the SQL query with placeholders for parameters.

```
$stmt = $conn->prepare("SELECT id, name FROM users WHERE id = ?");
```

2. Bind Parameters

1. Next, you bind values to the placeholders in the prepared statement. This step also specifies the data types of the parameters.

```
$id = 1;

$stmt->bind_param("i", $id);
```

3. Execute Statement

1. After binding the parameters, you execute the prepared statement. The database server prepares the query execution plan only once, and then you can execute the statement multiple times with different parameter values

```
$stmt->execute();
```

4. Get Result

1. If the prepared statement is a SELECT query, you can fetch the result set.

```
$result = $stmt->get_result();
while ($row = $result->fetch_assoc()) {
    echo "ID: " . $row["id"] . " - Name: " . $row["name"] . "<br>";
}
```

5. Close Statement

1. Finally, you close the prepared statement to free up resources. For example:

```
$stmt->close();
```

Benefits of using prepared statements:

1. Prevention of SQL injection: Prepared statements automatically escape input values, preventing malicious SQL injection attacks.

2. Performance: Prepared statements can improve performance by reducing the overhead of parsing and optimising queries, especially when executing the same query multiple times with different parameters.

3. Code readability and maintainability: Prepared statements make the code more readable and maintainable by separating the SQL query from the data values.

SELECT Part 1

```php
<?php
$servername = "localhost";
$username = "your_username";
$password = "your_password";
$database = "your_database";

// Create a connection
$conn = new mysqli($servername, $username, $password, $database);

// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}
```

SELECT Part 2

```
// Prepare and execute a SELECT statement
$stmt = $conn->prepare("SELECT id, name, email FROM users WHERE id = ?");
$id = 1;
$stmt->bind_param("i", $id);
$stmt->execute();
$result = $stmt->get_result();

// Fetch and display the results
while ($row = $result->fetch_assoc()) {
    echo "ID: " . $row["id"] . " - Name: " . $row["name"] . " - Email: " . $row["email"] . "<br>";
}

$stmt->close();
$conn->close();
?>
```

INSERT

```php
<?php
// Prepare and execute an INSERT statement
$stmt = $conn->prepare("INSERT INTO users (name, email) VALUES (?, ?)");
$name = "John Doe";
$email = "john.doe@example.com";
$stmt->bind_param("ss", $name, $email);
$stmt->execute();

echo "New record inserted successfully";

$stmt->close();
?>
```

UPDATE

```php
<?php
// Prepare and execute an UPDATE statement
$stmt = $conn->prepare("UPDATE users SET email = ? WHERE id = ?");
$email = "john.doe@example.com";
$id = 1;
$stmt->bind_param("si", $email, $id);
$stmt->execute();

echo "Record updated successfully";

$stmt->close();
?>
```

DELETE

```php
<?php
// Prepare and execute a DELETE statement
$stmt = $conn->prepare("DELETE FROM users WHERE id = ?");
$id = 1;
$stmt->bind_param("i", $id);
$stmt->execute();

echo "Record deleted successfully";

$stmt->close();
?>
```

Executing queries and handling results

1. Executing queries and handling results in PHP involves using functions provided by the database extension, such as `mysqli` or `PDO`.

2. When executing queries that return a result set (e.g., `SELECT` queries), you use the query() method to execute the query and get a `mysqli_result` object.

3. You can then use methods like `fetch_assoc()` to fetch rows from the result set.

4. For queries that do not return a result set (e.g., `INSERT`, `UPDATE`, `DELETE` queries), you can use the `query()` method to execute the query directly.

5. If the query is successful, query() will return `TRUE`; otherwise, it will return `FALSE`.

6. It's important to sanitise user input and use prepared statements to prevent SQL injection attacks when constructing SQL queries.

How to execute queries and handle results using mysqli

1. Connect to the database:

```
$servername = "localhost";
$username = "your_username";
$password = "your_password";
$database = "your_database";

$conn = new mysqli($servername, $username, $password, $database);

if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}
```

How to execute queries and handle results using mysqli

2. Executing a SELECT Query:

```
$sql = "SELECT id, name FROM users";
$result = $conn->query($sql);


if ($result->num_rows > 0) {
    while ($row = $result->fetch_assoc()) {
        echo "ID: " . $row["id"] . " - Name: " . $row["name"] .
"<br>";
    }
} else {
    echo "0 results";
}
```

How to execute queries and handle results using mysqli

3. Executing an INSERT, UPDATE, or DELETE Query:

```
$sql = "INSERT INTO users (name, email) VALUES ('John Doe',
'john.doe@example.com')";

if ($conn->query($sql) === TRUE) {

    echo "New record created successfully";

} else {

    echo "Error: " . $sql . "<br>" . $conn->error;

}
```

How to execute queries and handle results using mysqli

4. Closing the connection

```
$conn->close();
```

EXERCISE

1. Create a simple database table to store user information (e.g., name, email).

2. Write PHP scripts to connect to the database, insert a new user record, and retrieve all user information from the table.

Advanced CRUD Operations

Updating and deleting existing records using WHERE clause and prepared statements

1. In this example, we update the email of the user with ID 1 to "newemail@example.com".

```
<?php
// Prepare and execute an UPDATE statement
$stmt = $conn->prepare("UPDATE users SET email = ? WHERE id = ?");
$email = "newemail@example.com";
$id = 1;
$stmt->bind_param("si", $email, $id);
$stmt->execute();


echo "Record updated successfully";
$stmt->close();
?>
```

Updating and deleting existing records using WHERE clause and prepared statements

1. In this example, we delete the user with ID 1 from the users table.

```
<?php

// Prepare and execute a DELETE statement

$stmt = $conn->prepare("DELETE FROM users WHERE id = ?");

$id = 1;

$stmt->bind_param("i", $id);

$stmt->execute();


echo "Record deleted successfully";

$stmt->close();

?>
```

Searching and filtering data using WHERE clauses and operators (e.g., LIKE, IN)

1. In the next examples, we filter users by their IDs using the IN operator with a list of IDs passed as an array.

Search with LIKE Operator

```
<?php
$search_term = "John";
$stmt = $conn->prepare("SELECT id, name, email FROM users WHERE name LIKE ?");
$search_param = "%" . $search_term . "%";
$stmt->bind_param("s", $search_param);
$stmt->execute();
$result = $stmt->get_result();

if ($result->num_rows > 0) {
    while ($row = $result->fetch_assoc()) {
        echo "ID: " . $row["id"] . " - Name: " . $row["name"] . " - Email: " . $row["email"] . "<br>";
    }
} else {
    echo "0 results";
}
$stmt->close();
?>
```

Filtering with IN Operator

```php
<?php
$ids = [1, 2, 3];
$in_list = implode(",", array_fill(0, count($ids), "?"));
$stmt = $conn->prepare("SELECT id, name, email FROM users WHERE id IN ($in_list)");
$stmt->bind_param(str_repeat("i", count($ids)), ...$ids);
$stmt->execute();
$result = $stmt->get_result();

if ($result->num_rows > 0) {
    while ($row = $result->fetch_assoc()) {
        echo "ID: " . $row["id"] . " - Name: " . $row["name"] . " - Email: " . $row["email"] . "<br>";
    }
} else {
    echo "0 results";
}
$stmt->close();
?>
```

Pagination techniques for displaying large datasets in limited chunks

1. Pagination is a common technique used to display large datasets in limited chunks, allowing users to navigate through the dataset more easily.

2. In the following examples, replace your_table with the name of your database table.

3. The code retrieves the total number of records in the table, calculates the total number of pages needed for pagination, retrieves the records for the current page using LIMIT, and displays pagination links to navigate through the pages.

4. Remember to handle SQL injection vulnerabilities by sanitizing user input or using prepared statements when constructing SQL queries.

Retrieve Total Number of Records

```
$sql = "SELECT COUNT(*) as total FROM your_table";

$result = $conn->query($sql);

$row = $result->fetch_assoc();

$total_records = $row['total'];
```

Set Pagination Variables

```
$limit = 10; // Number of records to display per page

$page = isset($_GET['page']) ? $_GET['page'] : 1; // Current page
number

$start = ($page - 1) * $limit; // Starting index for the records on
the current page
```

Retrieve Records for Current Page

```
$sql = "SELECT * FROM your_table LIMIT $start, $limit";

$result = $conn->query($sql);
```

Display Pagination Links

```
$total_pages = ceil($total_records / $limit); // Calculate total pages


// Display pagination links
echo "<ul class='pagination'>";
for ($i = 1; $i <= $total_pages; $i++) {
    echo "<li><a href='?page=$i'>$i</a></li>";
}
echo "</ul>";
```

Display Records

```
if ($result->num_rows > 0) {
    while ($row = $result->fetch_assoc()) {
        // Display record data
    }
} else {
    echo "No records found";
}
```

EXERCISE

1. Create a simple database table to store user information (e.g., name, email).
2. Write PHP scripts to connect to the database, insert a new user record, and retrieve all user information from the table.
3. Enhance this assignment to allow users to edit and delete their information.
4. Implement functionalities to search users based on name and email using appropriate SQL queries