**Experiment No: 01**

**Experiment Name:** Study and implementation of DML commands of sql with suitable example: (i) Insert (ii) Delete (iii) Update

**Objectives:**

- Understand the fundamental concepts of Data Manipulation Language (DML) commands in SQL.
- Learn how to use the SQL INSERT statement to add new records to a database.
- Explore the SQL DELETE statement to remove data from a database.
- Gain proficiency in using the SQL UPDATE statement to modify existing data in a database.

**Theory**

**Data Manipulation Language (DML)**

Data Manipulation Language (DML) is a subset of SQL used for managing and manipulating data in a database. DML commands include INSERT, DELETE, and UPDATE, which allow users to perform the following actions:

**INSERT:** The INSERT statement is used to add new records into a table. It typically follows the format:

**INSERT INTO table_name (column1, column2, ...) VALUES (value1, value2, ...);**

**DELETE:** The DELETE statement is used to remove one or more rows from a table based on a specified condition. It typically follows the format:

DELETE FROM table_name WHERE condition;

**UPDATE:** The UPDATE statement is used to modify existing data in a table. It typically follows the format:

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

**Source Code:**

```
use DBMS_LAB;
create table students(
first_name varchar(20),
last_name varchar(20),
age varchar(10),
major_sub varchar(40)
```

```
)
go


-- Insert a new record into the 'students' table
INSERT INTO students (first_name, last_name, age, major_sub) VALUES ('Rashedul', 'islam', 23,
'Computer Science');
INSERT INTO students (first_name, last_name, age, major_sub) VALUES ('Tanzil', 'Ahmed', 23, 'Math');
INSERT INTO students (first_name, last_name, age, major_sub) VALUES ('Kaneez', 'Fateema', 25,
'Biology');
select * from students;

-- Delete all records of students with the age of 25
DELETE FROM students WHERE age = 25;
select * from students;

-- Update the major of all students named 'Alice' to 'Biology'
UPDATE students
SET major_sub = 'ICE'
WHERE first_name = 'Rashedul';
select * from students;
```

# Output:

| | first_name | last_name | age | major_sub |
|---|---|---|---|---|
| 1 | Rashedul | islam | 23 | Biology |
| 2 | Rashedul | islam | 23 | Computer Science |
| 3 | Tanzil | Ahmed | 23 | Math |
| 4 | Kaneez | Fateema | 23 | Biology |
| 5 | Rashedul | islam | 23 | Computer Science |
| 6 | Tanzil | Ahmed | 23 | Math |
| 7 | Kaneez | Fateema | 25 | Biology |
| 8 | Rashedul | islam | 23 | Computer Science |

| | first_name | last_name | age | major_sub |
|---|---|---|---|---|
| 1 | Rashedul | islam | 23 | ICE |
| 2 | Rashedul | islam | 23 | ICE |
| 3 | Tanzil | Ahmed | 23 | Math |
| 4 | Kaneez | Fateema | 23 | Biology |
| 5 | Rashedul | islam | 23 | ICE |
| 6 | Tanzil | Ahmed | 23 | Math |
| 7 | Rashedul | islam | 23 | ICE |
| 8 | Tanzil | Ahmed | 23 | Math |

| | first_name | last_name | age | major_sub |
|---|---|---|---|---|
| 1 | Rashedul | islam | 23 | ICE |
| 2 | Rashedul | islam | 23 | ICE |
| 3 | Tanzil | Ahmed | 23 | Math |
| 4 | Kaneez | Fateema | 23 | Biology |
| 5 | Rashedul | islam | 23 | ICE |
| 6 | Tanzil | Ahmed | 23 | Math |
| 7 | Rashedul | islam | 23 | ICE |
| 8 | Tanzil | Ahmed | 23 | Math |

**Experiment No: 02**

**Experiment Name:** Study and implementation of DML commands of sql with suitable example: (i) Create (ii) Alter (iii) Drop

**Objectives:**

- Understand the fundamental concepts of Data Manipulation Language (DML) commands in SQL, including CREATE, ALTER, and DROP.
- Learn how to create new database objects like tables, views, and indexes using the CREATE statement.
- Explore the ALTER statement for modifying existing database objects.
- Gain proficiency in using the DROP statement to remove database objects.

**Theory:**

Data Manipulation Language (DML)

Data Manipulation Language (DML) in SQL comprises commands for managing and manipulating data and database objects. This lab will focus on three key DML commands:

**CREATE:** The CREATE statement is used to create new database objects, such as tables, views, indexes, and more. The syntax for creating a table is as follows:

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

**ALTER:** The ALTER statement is used to modify the structure of existing database objects. It can be used to add, modify, or drop columns, constraints, or indexes.

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

**DROP:** The DROP statement is used to remove existing database objects. It can be used to delete tables, views, indexes, or even the entire database.

```
DROP TABLE table_name;
```

## Source Code:

```
-- Create a new table named 'employees'
CREATE TABLE employees (
    employee_id INT PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    hire_date DATE
);
go

INSERT INTO employees (employee_id,first_name, last_name,hire_date) VALUES (1, 'Rashedul', 'Islam',
'2023-10-20');
INSERT INTO employees (employee_id,first_name, last_name,hire_date) VALUES (2001,
'Kaneez','Fateema',2023-10-25);
select * from employees;

-- Insert a new employee record into the 'employees' table
INSERT INTO employees (employee_id, first_name, last_name, hire_date)
VALUES (1, 'John', 'Doe', '2023-10-20');
INSERT INTO employees (employee_id, first_name, last_name, hire_date)
VALUES (2, 'John', 'Doe', '2023-10-20');
INSERT INTO employees (employee_id, first_name, last_name, hire_date)
VALUES (3, 'John', 'Doe', '2023-10-20');
INSERT INTO employees (employee_id, first_name, last_name, hire_date)
VALUES (4, 'John', 'Doe', '2023-10-20');
select * from employees;


-- Add a 'salary' column to the 'employees' table
ALTER TABLE employees
ADD family_member DECIMAL(20);
select * from employees;

-- Delete the 'employees' table
DROP TABLE employees;
```

## Output:

| | employee_id | first_name | last_name | hire_date | salary | age |
|---|---|---|---|---|---|---|
| 1 | 1 | John | Doe | 2023-10-20 | NULL | NULL |
| 2 | 2 | John | Doe | 2023-10-20 | NULL | NULL |
| 3 | 3 | John | Doe | 2023-10-20 | NULL | NULL |
| 4 | 4 | John | Doe | 2023-10-20 | NULL | NULL |

| | employee_id | first_name | last_name | hire_date | salary | age | family_member |
|---|---|---|---|---|---|---|---|
| 1 | 1 | John | Doe | 2023-10-20 | NULL | NULL | NULL |
| 2 | 2 | John | Doe | 2023-10-20 | NULL | NULL | NULL |
| 3 | 3 | John | Doe | 2023-10-20 | NULL | NULL | NULL |
| 4 | 4 | John | Doe | 2023-10-20 | NULL | NULL | NULL |

**Experiment No: 03**

**Experiment Name:** Study and implementation of DML commands of (i)Select clause (ii) From clause (iii) Where clause.

## Objectives

Understand the fundamental concepts of SQL (Structured Query Language) for data retrieval.

Learn how to use the SELECT clause to specify the columns you want to retrieve from a database table.

Explore the FROM clause to specify the table or tables from which you want to retrieve data.

Gain proficiency in using the WHERE clause to filter data based on specific conditions.

## Theory

### SELECT Clause

The SELECT clause is used to retrieve data from one or more tables. It allows you to specify the columns you want to retrieve. The basic syntax is as follows:

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

### FROM Clause

The FROM clause specifies the table or tables from which you want to retrieve data. You can select data from a single table or combine data from multiple tables using JOIN operations. The basic syntax is as follows:

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

### WHERE Clause

The WHERE clause is used to filter data based on specific conditions. It allows you to retrieve only the rows that meet the specified criteria. The basic syntax is as follows:

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

## Source Code:

```
use University;
select all department_name
from instructor;
select name from instructor
where department_name=' Comp. Sci.' and salary > 70000;
```

## Output:

| | department_name |
|---|---|
| 1 | Comp. Sci. |
| 2 | Finance |
| 3 | Music |
| 4 | Physics |
| 5 | History |
| 6 | Physics |
| 7 | Comp. Sci. |
| 8 | History |
| 9 | Finance |
| 10 | Biology |
| 11 | Comp. Sci. |
| 12 | Elec. Eng. |

After Using Where clause

| | name |
|---|---|
| 1 | Srinivasan |
| 2 | Katz |
| 3 | Brandt |

**Experiment No: 04**

**Experiment Name:** Study and implementation of DML commands of (i)Group by & Having clause (ii) Order by clause (iii) Create view,Indexing & Procedure clause.

**Objectives**

- Understand the advanced data retrieval and manipulation techniques using SQL DML commands.
- Learn how to use the GROUP BY and HAVING clauses for data aggregation and filtering.
- Explore the ORDER BY clause for sorting query results.
- Gain proficiency in creating views to simplify complex queries, indexing for optimizing data access, and creating stored procedures for code reusability and enhanced security.

**Theory**

**GROUP BY & HAVING Clause**

The GROUP BY clause is used for grouping rows that have the same values in specified columns. It is typically used with aggregate functions like SUM, COUNT, AVG, etc., to perform calculations on grouped data.

```
SELECT column1, aggregate_function(column2)
FROM table_name
GROUP BY column1
HAVING condition;
```

**ORDER BY Clause**

The ORDER BY clause is used to sort the result set in ascending (ASC) or descending (DESC) order based on one or more columns.

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

**CREATE VIEW**

A view is a virtual table based on the result of a SELECT statement. It simplifies complex queries and allows users to access specific data without directly interacting with the underlying tables.

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

## Indexing

Indexes are used to optimize data retrieval by creating a data structure that allows for faster lookups. You can create indexes on columns to speed up search operations.

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

## Procedure

Stored procedures are a set of SQL statements that can be executed as a single unit. They are used for code reusability, enhanced security, and transaction management.

```
CREATE PROCEDURE procedure_name
AS
BEGIN
-- SQL statements here
END;
```

## Source Code:

```sql
select department_name ,avg(salary) as average_salary
 from instructor
 group by department_name
 having avg(salary)>42000;

-- Retrieve all employees and order them by hire date in descending order
SELECT first_name, last_name, hire_date
FROM employees
ORDER BY hire_date DESC;

-- Create a view to retrieve information about high-value orders
CREATE VIEW high_salary AS
SELECT department_name, salary
FROM instructor
WHERE salary > 1000;
select * from high_salary;


-- Create an index on the 'email' column of the 'customers' table
CREATE INDEX slry
ON instructor (salary);

-- Display indexing information for a table in MySQL
EXEC sp_helpindex 'instructor';
```

## Output:

| | department_name | salary |
|---|---|---|
| 1 | Comp. Sci. | 75245 |
| 2 | Finance | 94500 |
| 3 | Music | 46305 |
| 4 | Physics | 99750 |
| 5 | History | 69457 |
| 6 | Physics | 91350 |
| 7 | Comp. Sci. | 82687 |
| 8 | History | 71772 |
| 9 | Finance | 84000 |
| 10 | Biology | 79380 |
| 11 | Comp. Sci. | 96600 |
| 12 | Elec. Eng. | 84000 |

| | index_name | index_description | index_keys |
|---|---|---|---|
| 1 | PK__instruct__3213E83F718E5365 | clustered, unique, primary key located on PRIMARY | id |
| 2 | salary | nonclustered located on PRIMARY | salary |
| 3 | slry | nonclustered located on PRIMARY | salary |

| | first_name | last_name | hire_date |
|---|---|---|---|
| 1 | John | Doe | 2023-10-28 |
| 2 | John | Doe | 2023-10-26 |
| 3 | John | Doe | 2023-10-24 |
| 4 | John | Doe | 2023-10-20 |
| 5 | John | Doe | 2023-10-20 |
| 6 | John | Doe | 2023-10-20 |
| 7 | John | Doe | 2023-10-20 |
| 8 | John | Doe | 2023-10-20 |

**Experiment No: 05**

**Experiment Name:** Study and implementation of DML commands of Join operations with example

(i)Cartesian product (ii) Natural join (iii)Left outer join (iv) Right outer join (v) Full outer join.

**Objectives:**

Understand the fundamental concepts of SQL join operations.

Learn how to use different types of join operations, including Cartesian product, Natural join, Left outer join, Right outer join, and Full outer join.

Gain hands-on experience in implementing these join operations with practical examples.

**Theory**

**Join Operations**

Join operations in SQL are used to combine data from multiple tables based on a related column. The common types of join operations include:

**Cartesian Product (Cross Join):** A Cartesian product combines each row from the first table with every row from the second table. It results in a large result set with all possible combinations.

**Natural Join:** A natural join combines rows from two tables where column names match. It automatically joins on columns with the same names.

**Left Outer Join (or Left Join):** A left outer join returns all rows from the left table (first table) and the matched rows from the right table (second table). Unmatched rows from the right table will contain NULL values.

**Right Outer Join (or Right Join):** A right outer join is similar to a left outer join but returns all rows from the right table and the matched rows from the left table. Unmatched rows from the left table will contain NULL values.

**Full Outer Join (or Full Join):** A full outer join combines all rows from both tables and returns NULL where there is no match. It includes all rows from both tables.

**Source Code:**

```
-- Cartesian product of 'employees' and 'departments' tables
SELECT *
FROM instructor
CROSS JOIN takes;

select * from takes;
```

```sql
--For all students in the university who have taken some
--course, find their names and the course ID of all courses they took” as:
select name, id
from student natural join takes;   --this command does not support in sql
--alternate code
 select * from student join takes on student.id=takes.id

 --“Find all students who have not taken a course” as:
 --left outer join
select *
from student natural left outer join takes;

--right outer join
select student.id
from student  right outer join takes on student.id=takes.id;

--full outer join
select *
from (select *
from student
where dept_name = ' Comp. Sci.')
natural full outer join
(select * from takes
where semester = ' spring' and year1 = 2017)
```

# Output:

| | id | name | department_name | salary | id | course_id | sec_id | semester | year1 | grade |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 10101 | Srinivasan | Comp. Sci. | 75245 | 10101 | CS-101 | 1 | Fall | 2017 | A+ |
| 2 | 12121 | Wu | Finance | 94500 | 10101 | CS-101 | 1 | Fall | 2017 | A+ |
| 3 | 15151 | Mozart | Music | 46305 | 10101 | CS-101 | 1 | Fall | 2017 | A+ |
| 4 | 22222 | Karim | Physics | 99750 | 10101 | CS-101 | 1 | Fall | 2017 | A+ |
| 5 | 32343 | ElSaid | History | 69457 | 10101 | CS-101 | 1 | Fall | 2017 | A+ |
| 6 | 33456 | Gold | Physics | 91350 | 10101 | CS-101 | 1 | Fall | 2017 | A+ |
| 7 | 45565 | Katz | Comp. Sci. | 82687 | 10101 | CS-101 | 1 | Fall | 2017 | A+ |
| 8 | 58583 | Califieri | History | 71772 | 10101 | CS-101 | 1 | Fall | 2017 | A+ |
| 9 | 76543 | Singh | Finance | 84000 | 10101 | CS-101 | 1 | Fall | 2017 | A+ |
| 10 | 76766 | Crick | Biology | 79380 | 10101 | CS-101 | 1 | Fall | 2017 | A+ |
| 11 | 83821 | Brandt | Comp. Sci. | 96600 | 10101 | CS-101 | 1 | Fall | 2017 | A+ |
| 12 | 98345 | Kim | Elec. Eng. | 84000 | 10101 | CS-101 | 1 | Fall | 2017 | A+ |
| 13 | 10101 | Srinivasan | Comp. Sci. | 75245 | 10102 | CS-315 | 1 | spring | 2018 | A+ |
| 14 | 12121 | Wu | Finance | 94500 | 10102 | CS-315 | 1 | spring | 2018 | A+ |
| 15 | 15151 | Mozart | Music | 46305 | 10102 | CS-315 | 1 | spring | 2018 | A+ |
| 16 | 22222 | Karim | Physics | 99750 | 10102 | CS-315 | 1 | spring | 2018 | A+ |
| 17 | 32343 | ElSaid | History | 69457 | 10102 | CS-315 | 1 | spring | 2018 | A+ |
| 18 | 33456 | Gold | Physics | 91350 | 10102 | CS-315 | 1 | spring | 2018 | A+ |
| 19 | 45565 | Katz | Comp. Sci. | 82687 | 10102 | CS-315 | 1 | spring | 2018 | A+ |
| 20 | 58583 | Califieri | History | 71772 | 10102 | CS-315 | 1 | spring | 2018 | A+ |
| 21 | 76543 | Singh | Finance | 84000 | 10102 | CS-315 | 1 | spring | 2018 | A+ |
| 22 | 76766 | Crick | Biology | 79380 | 10102 | CS-315 | 1 | spring | 2018 | A+ |
| 23 | 83821 | Brandt | Comp. Sci. | 96600 | 10102 | CS-315 | 1 | spring | 2018 | A+ |
| 24 | 98345 | Kim | Elec. Eng. | 84000 | 10102 | CS-315 | 1 | spring | 2018 | A+ |
| 25 | 10101 | Srinivasan | Comp. Sci. | 75245 | 10103 | CS-347 | 1 | Fall | 2018 | A+ |

| | id |
|---|---|
| 1 | NULL |
| 2 | 10102 |
| 3 | 10103 |
| 4 | NULL |
| 5 | 10109 |
| 6 | NULL |
| 7 | NULL |
| 8 | NULL |
| 9 | NULL |
| 10 | NULL |

| id | name | dept_name | tot_cred | id | course_id | sec_id | semester | year1 | grade |
|---|---|---|---|---|---|---|---|---|---|
| 10102 | farim | BIO | 130 | 10102 | CS-315 | 1 | spring | 2018 | A+ |
| 10103 | rarim | CHEM | 135 | 10103 | CS-347 | 1 | Fall | 2018 | A+ |
| 10109 | karim | PHY | 130 | 10109 | CS-101 | 1 | Fall | 2017 | A+ |

**Experiment No: 06**

**Experiment Name:** Study and implementation of Aggregate function with example

(i)Count Function (ii) Max Function (iii) Min Function (iv) Avg Function

## Objectives

- To understand the concept of aggregate functions in databases.
- To study and implement the Count, Max, Min, and Avg functions.
- To learn how these functions are used to retrieve useful information from a database.

## Theory

Aggregate functions are essential in database management systems for performing calculations on sets of values within a table. The four aggregate functions covered in this experiment are:

### Count Function

The COUNT function is used to count the number of rows in a table that meet a specific condition. It is often used to determine the size of a result set.

### Max Function

The MAX function is used to find the maximum value within a set of values in a specific column of a table.

### Min Function

The MIN function is used to find the minimum value within a set of values in a specific column of a table.

### Avg Function

The AVG function is used to calculate the average value of a set of numeric values within a column.

Write SQL queries to demonstrate each of the aggregate functions:

**Count Function:** SELECT COUNT(column_name) FROM table_name WHERE condition;

**Max Function:** SELECT MAX(column_name) FROM table_name WHERE condition;

**Min Function:** SELECT MIN(column_name) FROM table_name WHERE condition;

**Avg Function:** SELECT AVG(column_name) FROM table_name WHERE condition;

## Source Code:

```
--“Find the average salary of instructors in the Computer Science
department.” We write this query as follows:
```

```
 select avg (salary) as average_salary
 from instructor
 where department_name = ' Comp. Sci.';

  --“Find the total number of instructors who teach a course in the Spring
2018 semester.”
 select count (distinct id) as Number_of_instructor
from teaches
where semester = ' spring' and year1 = 2018;

select min (salary) as Min_salary  from instructor
select Max (salary) as Max_salary  from instructor
```

**Output:**

| | Number_of_instructor |
|---|---|
| 1 | 3 |

| | average_salary |
|---|---|
| 1 | 84844 |

| | Max_salary |
|---|---|
| 1 | 99750 |

| | Min_salary |
|---|---|
| 1 | 46305 |

**Experiment No: 07**

**Experiment Name:** Study and implementation of Triggering system on database table using SQL commands with example.

## Objectives

- To understand the concept of database triggers.
- To study the types of triggers in a database.
- To implement database triggers using SQL commands.
- To demonstrate the practical use of triggers with a relevant example.

## Theory

Database triggers are stored programs in the database that are automatically executed in response to specific events or conditions. There are two main types of triggers:

### DML Triggers (Data Manipulation Language)

These triggers are fired in response to data manipulation language (DML) events like INSERT, UPDATE, or DELETE operations on a table.

### DDL Triggers (Data Definition Language)

DDL triggers are fired in response to data definition language (DDL) events, such as CREATE, ALTER, or DROP operations on database objects.

## Source Code:

```sql
-- Create a new database
CREATE DATABASE bank_database;

-- Switch to the newly created database
USE bank_database;

-- Create a table to store bank account information
CREATE TABLE bank_accounts (
    account_id INT PRIMARY KEY,
    account_holder VARCHAR(50),
    balance DECIMAL(10, 2)
);

-- Insert sample data into the table
INSERT INTO bank_accounts (account_id, account_holder, balance)
VALUES
    (1, 'John Doe', 1500.00),
    (2, 'Jane Smith', 800.00),
```

```
    (3, 'Alice Johnson', 1200.00);

-- Display the initial state of the bank_accounts table
SELECT * FROM bank_accounts;

-- Create a trigger to enforce a minimum balance requirement

-- Create a trigger in SQL Server
-- Create an AFTER UPDATE trigger in SQL Server

CREATE TRIGGER check_min_balance
ON bank_accounts
AFTER UPDATE
AS
BEGIN
    IF UPDATE(balance)  -- Check if the 'balance' column was updated
    BEGIN
        IF (SELECT MIN(balance) FROM deleted) < 1000.00
        BEGIN
            THROW 50000, 'Minimum balance requirement not met', 1;
        END
    END
END;

-- Attempt to update an account balance that violates the minimum balance requirement
-- This update should be blocked by the trigger
UPDATE bank_accounts
SET balance = 800.00
WHERE account_id = 2;

-- Attempt to update an account balance that meets the minimum balance requirement
-- This update should be allowed by the trigger
UPDATE bank_accounts
SET balance = 1700.00
WHERE account_id = 3;

-- Display the updated state of the bank_accounts table
SELECT * FROM bank_accounts;

-- Clean up by dropping the database (be cautious in a real environment)
-- DROP DATABASE bank_database;
```

## Output:

| | account_id | account_holder | balance |
|---|---|---|---|
| 1 | 1 | John Doe | 1500.00 |
| 2 | 2 | Jane Smith | 800.00 |
| 3 | 3 | Alice Johnson | 1600.00 |

| | account_id | account_holder | balance |
|---|---|---|---|
| 1 | 1 | John Doe | 1500.00 |
| 2 | 2 | Jane Smith | 800.00 |
| 3 | 3 | Alice Johnson | 1700.00 |

**Experiment No: 08**

**Experiment Name:** Study and implementation of SQL commands with to connect MySQL database with Java or PHP.

**Objectives**

- To understand the principles of connecting to a MySQL database using Java.
- To comprehend the basics of connecting to a MySQL database using PHP.
- To implement practical examples of database connectivity with both Java and PHP.
- To perform basic SQL operations on the connected database.

**Theory**

**Connecting to a MySQL Database with Java**

To connect a MySQL database with Java, we utilize the JDBC (Java Database Connectivity) API. JDBC allows Java applications to interact with databases through a standard interface. The essential steps include:

**Import JDBC libraries.**

Load the MySQL JDBC driver.

Establish a connection to the MySQL database.

Create a statement for executing SQL queries.

Execute SQL queries, retrieve data, and handle exceptions.

Close the connection when finished.

Connecting to a MySQL Database with PHP

In PHP, we can connect to a MySQL database using MySQLi (MySQL Improved) or PDO (PHP Data Objects) extensions. The basic steps involve:

- Initialize a database connection using MySQL or PDO.
- Create SQL queries using prepared statements to prevent SQL injection.
- Execute SQL queries and fetch data.
- Close the database connection.

**Source code:**

```php
<?php
$serverName = "DESKTOP-SNLQ279"; // Use "localhost" if SQL Server is on the same machine
$connectionOptions = array(
    "Database" => "University",
    "Uid" => "your_username",
    "PWD" => "your_password"
);
```

```php
// Establish the connection
$conn = sqlsrv_connect($serverName, $connectionOptions);

if (!$conn) {
    die(print_r(sqlsrv_errors(), true));
}

// Your SQL queries and operations go here

// Close the connection when done
sqlsrv_close($conn);
?>
```