SQL and MySQL

SQL which stands for Structured Query Language, is a standard language for accessing and manipulating databases. As per ANSI (American National Standards Institute), it is the standard language for relational database management systems. All relational database management systems like MySQL, MS Access, Oracle, postgres and SQL Server use SQL as standard database language. SQL consists of data definition language, data manipulation language and data control language. The scope of SQL includes data insert, query, update and delete, schema creation and modification, and data access control. When PHP builds web pages, it uses SQL to retrieve data to display on the resulting page. Hence, SQL is also used widely in collaboration with PHP.

What is a relational database management system?

A relational database management system (RDBMS) is a database management system based on the relational model, i.e. it handles the way data is stored, maintained and retrieved. A relational database is a collection of data items organized as a set of tables from which data can be accessed. Relational databases establish well-defined relationship between database tables.

Table basics:

A database system consists of one or more objects called tables. The information in the databases are stored in these tables. Each table is uniquely identified by a name (e.g. "Books", "Students"). Tables are comprised of rows and columns.

The following is an example of a table "Students":

student_id	**student_name**	**age**
1	John Smith	22
2	Mark Wang	22
3	Mary Kom	24
4	Annie Vincent	25
5	Jason Holder	29
6	Suzie Bates	27

Table 1

The table consists of 3 columns (student_id, student_name, age), and 6 rows, one for each student.

(I.) SQL Statements:

The SQL keywords are case insensitive but are usually written in all caps. The SQL table names and column names are however case sensitive.

(a) SELECT Statement

The SQL SELECT statement is the most basic statement used in SQL. It is used to select data from a database and display it on the screen.

Syntax of SELECT statement is:

1. SELECT * FROM *table_name*;

This statement selects all the columns from the table.

2. SELECT *column_1, column_2* FROM *table_name*;

This statement displays the specified columns (*column_1, column_2* in the above example) from the table.

Example:

SELECT student_id, student_name FROM Students;

The above statement generates the following table:

student_id	**student_name**
1	John Smith
2	Mark Wang
3	Mary Kom
4	Annie Vincent
5	Jason Holder
6	Suzie Bates

(b) SELECT DISTINCT Statement

The SELECT statement selects information from the table which may contain redundancies. To avoid such redundant data, we use the keyword "DISTINCT".

The syntax is as follows:

SELECT DISTINCT *column_1, column_2,...,column_n* FROM *table_name*;

Example:

SELECT DISTINCT age FROM Students;

The above statement displays the following result.

age
22
24
25
29
27

We can observe that "22" has not been repeated twice, due to the distinct clause.

(c) WHERE clause

The WHERE clause is used to select only those rows that satisfy a specific condition.

The syntax is as follows:

SELECT *column_1, column_2,...,column_n* FROM *table_name* WHERE *condition*;

Example:

SELECT * FROM Students WHERE age='22';

This statement displays the following table,

student_id	student_name	age
1	John Smith	22
2	Mark Wang	22

(d) SQL AND & OR operators:

The **AND** operator displays the tuples which satisfy both the first condition and the second condition.

The syntax is as follows:

SELECT *column_1, column_2,....,column_n* WHERE *condition1* AND *condition2*;

Example:

SELECT * FROM Students WHERE age=’22’ AND student_name='John Smith';

student_id	student_name	age
1	John Smith	22

The **OR** operator displays the tuples if the first condition or the second condition is true.

The syntax is as follows:

SELECT *column_1, column_2,....,column_n* WHERE *condition1* OR *condition2*;

Example:

SELECT * FROM Students WHERE age=’22’ OR student_name='Mary Kom';

student_id	student_name	age
1	John Smith	22
2	Mark Wang	22
3	Mary Kom	24

(e) SQL IN clause:

The SQL in clause allows us to specify multiple values in the where clause. The values can be numbers, text etc.

The syntax is as follows:

SELECT *column_1,...,column_n* FROM *table_name WHERE column_name* IN (*value1, value2,...*);

Example:
SELECT * FROM Students WHERE student_id IN (1,2,3,4);

student_id	**student_name**	**age**
1	John Smith	22
2	Mark Wang	22
3	Mary Kom	24
4	Annie Vincent	25

(f) SQL BETWEEN clause:

The SQL between clause selects values between a specified range. The values can be numbers, text or dates.

The syntax is as follows:

SELECT *column_1,...,column_n* FROM *table_name* WHERE *column_name* BETWEEN *value1* AND *value2*;

Example:
SELECT * FROM Students WHERE student_id BETWEEN 2 AND 4;

student_id	**student_name**	**age**
2	Mark Wang	22
3	Mary Kom	24
4	Annie Vincent	25

SELECT * FROM Students WHERE student_id NOT BETWEEN 2 AND 4;

student_id	**student_name**	**age**
1	John Smith	22
5	Jason Holder	29
6	Suzie Bates	27

The NOT BETWEEN clause displays the tuples outside the specified range.

(g) SQL LIKE operator:

The LIKE operator in SQL is used to specify certain pattern.

The syntax is as follows:

SELECT *column_1,...,column_n* FROM *table_name* WHERE *column_name* LIKE *pattern*;

Example:

SELECT * FROM Students WHERE student_name LIKE '%m';

student_id	**student_name**	**age**
3	Mary Kom	24

The above selection from the Students table consists of tuples with student names that end with the alphabet "m".

The % sign is used to define wildcards (missing letters) in the pattern.

The wildcards can be of the following forms:

% = Substitute for 0 or more characters

_= Substitute for 1 character

Example: SELECT * FROM Students WHERE student_name LIKE '_uzie Bates';

student_id	student_name	age
6	Suzie Bates	27

[charlist]=ranges of characters to match

Example: SELECT * FROM Students WHERE student_name LIKE '[AMS]%';

Student_id	student_name	age
2	Mark Wang	22
3	Mary Kom	24
4	Annie Vincent	25
6	Suzie Bates	27

The above table displays tuples with students' names that start with 'A' or 'M' or 'S'.

[^charlist]=match only a character not specified in the list

(h) SQL ORDER BY clause:

The order by clause is used in a SELECT statement to sort results either in ascending or descending order by one or more columns. The order by clause sorts the rows in ascending order by default. The rows can be ordered in descending order by using the keyword desc.

The syntax is as follows:

SELECT *column_1,...,column_n* FROM *table_name* ORDER BY *column_name asc/desc*, *column_name asc/desc;*

Example:

SELECT * FROM Students ORDER BY student_id;

student_id	student_name	age
1	John Smith	22
2	Mark Wang	22
3	Mary Kom	24
4	Annie Vincent	25
5	Jason Holder	29
6	Suzie Bates	27

The student_id is sorted in ascending order by default because we did not mention desc explicitly.

(i) SQL COUNT clause:

The SQL count clause returns the number of rows that match a specified criterion.

The syntax is as follows:

SELECT COUNT(column_1) FROM table_name;

This returns the number of rows in column 1.

SELECT COUNT(distinct column_1) FROM table_name;

This returns the number of distinct valued rows in column 1.

SELECT COUNT(*) FROM table_name;

This returns the number of rows in the table.

Example:

SELECT COUNT(DISTINCT age) AS age_num FROM Students;

Output:

age_num
5

(j) SQL aliases:

The SQL aliases are used to rename a table or a column in the table temporarily. The actual table name or column name does not change. The SQL aliases are used to make the tables more readable or for a particular SQL query.

The syntax is as follows:

SELECT *column_name* AS *alias_name* FROM *table_name*;

The output column heading will be the new temporary name given to the column.

Example:

SELECT student_name AS name FROM Students;

name
John Smith
Mark Wang
Mary Kom
Annie Vincent
Jason Holder
Suzie Bates

SELECT column_name FROM table_name AS alias_name;

Consider the following two tables that will be used to demonstrate examples in the following concepts.

student_id	student_name	instructor_id
101	Alex	501
102	Roger	504
103	Haley	502
104	Robin	506
105	Priyanka	503

Table 1: Students Table

instructor_id	instructor_name
501	Michelle Tyler
502	Francine Smith
503	John Smith
504	Gaurav Patel
505	Adam Curtis

Table 2: Instructors table

SELECT *column_name* FROM *table_name* AS *alias_name*;

Example:

SELECT S.student_name, I.instructor_name FROM Students AS S, Instructors AS I WHERE S.instructor_id=I.instructor_id ORDER BY S.student_name;

student_name	instructor_name
Alex	Michelle Tyler
Haley	Francine Smith
Priyanka	John Smith
Roger	Gaurav Patel

(k) SQL JOINS:

SQL JOINS are used to relate information in different tables. It is used to combine rows from two or more tables, based on a common field between them. The SQL JOIN condition is used in the SQL where clause.

There are 4 kinds of joins:

INNER JOIN

LEFT JOIN

RIGHT JOIN

OUTER JOIN

We'll discuss each of these joins in detail below.

(l) SQL INNER JOIN:

The inner join selects all rows from both tables when there is a match between the columns in both tables.

Syntax for INNER JOIN:

SELECT *column_1,...,column_n* FROM *table1* INNER JOIN *table2* ON *table1.column_name=table2.column_name*;

product_ID	product_Name	price
501	Mobile	80
502	Refrigerator	200
503	Television	150
504	Oven	100
505	Laptop	600

Table 2: Product table

order_ID	product_ID	total_units
101	501	40
102	503	25
103	504	10
104	505	60

Table 3: Order table

Consider the two tables Product and Order.

Example of an INNER JOIN:

```
SELECT product_Name, price, total_units
FROM Product p
INNER JOIN Order o
ON p.product_ID=o.product_ID;
```

Result:

product_Name	price	total_units
Mobile	80	40
Television	150	25
Oven	100	10
Laptop	600	60

(m) LEFT JOIN:

The LEFT JOIN selects all rows from the left table and the matching rows in the right table. If there is no match for a particular tuple on the right table, then it simply displays NULL in the corresponding column of the right table.

Syntax of LEFT JOIN is:

SELECT *column_1,...,column_n* FROM table1 LEFT JOIN table2 ON *table1.column_name=table2.column_name*;

Example:

SELECT product_Name, price, total_units

FROM Product p

LEFT JOIN Order o

ON p.product_ID=o.product_ID;

product_Name	price	total_units
Mobile	80	40
Refrigerator	200	NULL
Television	150	25
Oven	100	10
Laptop	600	60

(n) RIGHT JOIN:

The RIGHT JOIN selects all rows from the right table and the matching rows in the left table. If there is no match for a tuple on the left table, then it simply displays NULL in the corresponding column of the left table.

Syntax:

SELECT *column_1,...,column_n* FROM table1 RIGHT JOIN table2 ON *table1.column_name=table2.column_name*;

Example:

SELECT product_Name, price, total_units

FROM Product p

RIGHT JOIN Order o

ON p.product_ID=o.product_ID;

product_Name	**price**	**total_units**
Mobile	80	40
Television	150	25
Oven	100	10
Laptop	600	60

(n) FULL JOIN:

A FULL JOIN is used to select list of all records from both the tables, i.e. it combines the result of both left and right tables.

Syntax:

SELECT *column_1,...,column_n* FROM table1 FULL JOIN table2 ON *table1.column_name=table2.column_name*;

Example:

SELECT product_Name, price, total_units

FROM Product p

FULL JOIN Order o

ON p.product_ID=o.product_ID;

product_Name	**price**	**total_units**
Mobile	80	40
Refrigerator	200	NULL
Television	150	25
Oven	100	10
Laptop	600	60

(o) SQL UNION clause:

The union operator is used to combine the results of two or more SELECT statements. The union operator does not return duplicate values.

The conditions to be followed to use the union operator are,

1. Each of the SELECT statements must have the same number of columns.

2. All the columns must have the same data type.

3. All the columns in each of the SELECT statements should be in the same order.

Syntax:

SELECT *column_1,..,column_n* FROM *table1*

UNION

SELECT *column_1,..,column_n* FROM *table2*;

Example:

```
SELECT product_ID FROM Product

UNION

SELECT product_ID FROM Order

ORDER BY product_ID;
```

Prdouct_ID
501
502
503
504
505

(II.) Creating SQL tables:

The create statement is used to create table in a database. Each table has a unique name and is organized into rows and columns.

Syntax:

```
CREATE TABLE table_name

(

column_1 data_type(size),

column_2 data_type(size),

column_3 data_type(size),

.....

);
```

The table_name gives the name to the table, the column_n parameters specify the names of the columns, the data_type parameter specifies the type of data in the columns, the size parameter specifies the size of the data the column can hold.

The most common datatypes used in SQL are:

char(size)	Fixed-length character string. Size is specified in parenthesis. Max 255 bytes.
varchar(size)	Variable-length character string. Max size is specified in parenthesis.
float(size)	Approximate numeric values with a precision up to 64
int	Integer numerical
date	Date value
number(size,d)	Number value with a maximum number of digits of "size" total, with a maximum number of "d" digits to the right of the decimal.

Example:

```
CREATE TABLE Students
(
student_id int,
student_name varchar(30),
age int
);
```

SQL Constraints:

SQL constraints are used to specify rules on data columns in a table. The constraint must be met in order to enter data into a table. Constraints improve the accuracy of the database.

Constraints can be specified when a table is created with the CREATE TABLE statement or we can use ALTER TABLE statement to create constraints even after the table is created.

Following are the commonly used constraints in SQL:

NOT NULL: Specifies that a column cannot accept null values. NOT NULL constraint requires that each field contains a value.

CREATE TABLE

Example:

```
CREATE TABLE Students
(
student_id int NOT NULL,
student_name varchar(30),
age int NOT NULL
);
```

The above NOT NULL constraint against the student_id and age ensures that student_id and age never take NULL value as input.

UNIQUE: Ensures that each row for a column has a unique value.

Example:

```
CREATE TABLE Students
(
student_id int NOT NULL,
student_name varchar(30),
age int UNIQUE
);
```

The Students table in Table 1 does not satisfy this constraint because the Age column has two records with the same value 22.

PRIMARY KEY: A primary key uniquely identifies each row in a table. Primary key implies NOT NULL and UNIQUE constraints implicitly.

```
CREATE TABLE Students

(

student_id int,

student_name varchar(30),

age int,

PRIMARY KEY (Student_id)

);
```

The student_id is the primary key, hence it implicitly means that NOT NULL and UNIQUE constraints apply on it. student_id uniquely identifies each row in the Students table.

FOREIGN KEY: A foreign key is used to enforce link between two tables. A foreign key in one table refers to a primary key in another table.

Example:

```
CREATE TABLE Order

(

order_ID int,

product_ID int,

total_units int,

PRIMARY KEY (order_ID) ,

FOREIGN KEY (product_ID) REFERENCES Product(product_ID)

)
```

Hence, the foreign key product_ID in Order table refers to the primary key product_ID in the Product table.

DEFAULT: Specifies a default value for a column when no value is provided.

Example:

```
CREATE TABLE Students
(
student_id int,
student_name varchar(30),
age int DEFAULT '20',
PRIMARY KEY (student_id)
);
```

The default constraint in the above example inserts value 20 into the age column when no value is specified.

CHECK: Check constraint is used to check whether all values in column satisfy certain conditions.

Constraints in create table statement:

```
CREATE TABLE Students
(
student_id int,
student_name varchar(30),
age int,
CHECK (student_id<10)
);
```

In the above example, the check constraint sees whether the value of student_id is less than 10.

DROP table:

The sql drop table statement is used to delete a table.

The syntax for drop table statement is as follows:

DROP TABLE table_name;

Example:

DROP TABLE Students;

This ensures that the table Students is removed and so is all its related data, constraints etc.

SQL INSERT INTO STATEMENT:

The SQL INSERT INTO statement is used to add rows of data into a table in a database.

Syntax for INSERT INTO is as follows:

INSERT INTO table_name (column_1, column_2, ..., column_n)

VALUES (value1, value2,....., valuen);

column_1, column_2... are the names of the columns.

value1, value2,.... are the values inserted into the respective columns.

Example:

INSERT INTO Students(student_id, student_name, age)

VALUES (1, 'John Smith', '22');

INSERT INTO Students(student_id, student_name, age)

VALUES (2, 'Mark Wang', '22');

INSERT INTO Students(student_id, student_name, age)

VALUES (3, 'Mary Kom', '24');

INSERT INTO Students(student_id, student_name, age)

VALUES (4, 'Annie Vincent', '25');

INSERT INTO Students(student_id, student_name, age)

VALUES (5, 'Jason Holder', '29');

INSERT INTO Students(student_id, student_name, age)

VALUES (6, 'Suzie Bates', '27');

SQL UPDATE STATEMENT:

The SQL update statement is used to update existing records in a table.

The syntax of the update statement is as follows:

UPDATE table_name

SET column_1=value1, column_2=value2,.....column_n=valuen

WHERE condition;

example:

UPDATE Students

SET age='21'

WHERE student_id=2;

Result:

SELECT * FROM Students;

student_id	**student_name**	**age**
1	John Smith	22
2	Mark Wang	21
3	Mary Kom	24
4	Annie Vincent	25
5	Jason Holder	29
6	Suzie Bates	27

SQL DELETE STATEMENT:

The SQL delete statement is used to delete existing records from a table.

The syntax for delete statement is as follows:

DELETE FROM table_name

WHERE condition;

Example:

DELETE FROM Students

WHERE student_id=6;

Result:

SELECT * FROM Students;

student_id	**student_name**	**age**
1	John Smith	22
2	Mark Wang	22
3	Mary Kom	24
4	Annie Vincent	25
5	Jason Holder	29

MYSQL

MySQL is an open source relational SQL database management system. It is very commonly used in conjunction with PHP scripts to create powerful and dynamic server side applications.

MySQL and PHP

Now we will see how PHP and MYSQL work together to create dynamic server side applications.

Opening database connection:

Before accessing data in a database, we need to be able to connect to the server. MySQL provides mysqli_connect function to open a database connection. The i in mysqli stands for improved.

Syntax:

mysqli_connect(host*, username, password, dbname, port, socket*);

host: host specifies a host name or an IP address.

username: Specifies MySQL username.

password: Specifies MySQL password.

dbname: Specifies the name of the database where connection is to be established. (Optional)

port: Specifies port number to attempt to connect to MySQL server. (Optional)

socket: Specifies the socket or named pipe to use. (Optional)

Example:

```
$conn=mysqli_connect("localhost", "usern", "root", "db");
```

Closing database connection:

A database connection can be closed in any of the following two ways.

1. Using *mysqli_close()* function which closes a previously opened connection.

Syntax: mysqli_close(*connection*);

Ex:

```
<? php
$conn=mysqli_connect("localhost", "usern", "root", "db");
// PHP code
mysqli_close($conn);
?>
```

2. Using *connection*->close();

Example:

```
<? php
$conn=mysqli_connect("localhost", "usern", "root", "db");
// PHP code
$conn->close();
?>
```

Checking database connection:

We can check if a database connection has been established. If there is a failure, *mysqli_connect_errno()* returns an error code from the last connection error.

Example:

```
<? php

$conn=mysqli_connect("localhost", "usern", "root", "db");

//Check connection

if(!$con)

{

echo "Connection failed: ".mysqli_connect_errno();

}

?>
```

We can alternatively use *mysqli_connect_error()* function instead of *mysqli_connect_errno(). mysqli_connect_error()* function returns an error description from the last connection error, if any.

Creating MySQL Database using PHP

A database can be created using CREATE DATABASE statement.

Syntax:

```
$variable="CREATE DATABASE db";
```

Example:

```
<? php

$conn=mysqli_connect("localhost", "usern", "root");

$sql="CREATE DATABASE db";

mysqli_query($conn, $sql);
```

```
mysqli_query($conn, "SELECT * FROM Students");

mysqli_close($conn);

?>
```

The mysqli_query performs queries against the database.

Creating tables

Tables can be created like the database. First we create the sql query to create the table and then execute the query using mysqli_query() function.

Example:

```
<? php

$conn=mysqli_connect("localhost", "usern", "root", "db");

$sql="CREATE TABLE Students (

student_id int,

student_name VARCHAR(20) NOT NULL,

age int NOT NULL,

PRIMARY KEY (student_id)

)";

mysqli_query($conn, $sql);

mysqli_query($conn, "SELECT * FROM Students");

mysqli_close($conn);

?>
```

Inserting data into table

After creating a database and a table, data can be inserted using the SQL INSERT INTO statements, as follows,

Example:

```
<? php

$conn=mysqli_connect("localhost", "usern", "root", "db");

$sql="CREATE TABLE Students (

student_id int,

student_name VARCHAR(20) NOT NULL,

age int NOT NULL,

PRIMARY KEY (student_id)

)";

$sql1="INSERT INTO Students (student_id, student_name, age)

VALUES (1, 'John Smith', '22')";

mysqli_query($conn, $sql);

mysqli_query($conn, $sql1);

mysqli_query($conn, "SELECT * FROM Students");

mysqli_close($conn);

?>
```

Deleting a database

A database can be deleted using *drop database* statement and issuing a mysqli_query() function to delete the database.

Syntax:

```
variable="DROP DATABASE database_name";
```

Example:

```
<? php
$conn=mysqli_connect("localhost", "usern", "root");
$sql="DROP DATABASE db";
mysqli_query($conn, $sql);
mysqli_close($conn);
?>
```

Deleting a table

Again, deleting a table can be done similar to deleting a database.

Example:

```
<? php
$conn=mysqli_connect("localhost", "usern", "root", "db");
$sql="DROP TABLE Students";
mysqli_query($conn, $sql);
mysqli_close($conn);
?>
```

Retrieving data from database

Data can be fetched from MySQL tables by executing SQL SELECT statement through PHP function mysqli_query.

There are many ways in which data can be retrieved. Some of them are defined below.

mysqli_fetch_array()

The mysqli_fetch_array() function returns a result row as an associative array, a numeric array or both.

Syntax:

mysqli_fetch_array(*result, resulttype*);

result specifies an identifier returned by mysqli_query().

resulttype returns the type of array that should be returned. It can be MYSQL_ASSOC, MYSQL_NUM, or MYSQL_BOTH. (Optional)

Example:

```
<? php
$conn=mysqli_connect("localhost", "usern", "root", "db");
$sql="SELECT student_id, student_name FROM Students";
$result=mysqli_query($conn, $sql);
//Associative array
while($row=mysqli_fetch_array($result, MYSQLI_ASSOC))
{
echo "ID : " . $row["student_id"] . "<br>". "Name : ". $row["student_name"]."<br>";
}
//Numeric array
while($row=mysqli_fetch_array($result, MYSQLI_NUM))
{
echo "ID : $row[0] <br>".
      "Name : $row[1] <br>";
}
echo "Data retrieved";
mysqli_close($conn);
?>
```

The MYSQL_ASSOC is used as second argument in the mysqli_fetch_array() when we want to return a row as an associative array. Using an associative array, we can access the field using name instead of index. Using MYSQL_ARRAY, we access the field using index.

mysqli_fetch_assoc()

The mysqli_fetch_assoc() function returns result rows as an associative array.

Syntax:

mysqli_fetch_assoc(*result*);

result specifies an identifier returned by mysqli_query().

Example:

```
<?php
$conn=mysqli_connect("localhost", "usern", "root", "db");
$sql="SELECT student_id, student_name FROM Students";
$result=mysqli_query($conn, $sql);
while($row=mysqli_fetch_assoc($result))
{
echo "ID : " . $row["student_id"] . "<br>". "Name : ". $row["student_name"]."<br>";
}
echo "Data retrieved";
mysqli_close($conn);
?>
```

Prepared statements and Bind parameters

SQL injection is a technique where malicious users can inject SQL commands into an SQL statement, via web page input. Injected SQL commands can alter SQL statement and compromise the security of a web application. Prepared statements are very useful against SQL injections. *Prepared statements* are SQL statements that are sent to and parsed by the database server separately from any parameters.

Prepared statements work as follows,

(i) An SQL statement template is created and sent to the database. Parameters are left unspecified (labeled "?")

(ii) The database parses, compiles, and performs query optimization on the SQL statement template, and stores the result without executing it.

(iii) At a later time, the application binds the values to the parameters using bind_param() function, and the database executes the statement. The application may execute the statement as many times as it wants with different values.

Example:

```
<?php

$conn=mysqli_connect("localhost", "usern", "root", "db");

// prepare and bind

$stmt=$conn->prepare("INSERT INTO Students (student_id, student_name, age) VALUES (?, ?, ?)");

$stmt->bind_param("iss", $id, $name, $age);

$id=7;

$name="Janice";

$age=34;

$stmt->execute();

echo "New record created successfully";

$stmt->close();
```

```
$conn->close();

?>
```

bind_param() function binds the parameter to the SQL query and tells the database what the parameters are. The "iss" argument lists the type of data that the parameters are. The parameters listed can be of the following types:

i - integer

d - double

s - string

b - BLOB

By telling MySQL what type of data to expect, we minimize the risk of SQL injections.

Selecting data with MySQL

The SELECT statement is used to select data from tables.

Example:

```
<? php

$conn=mysqli_connect("localhost", "usern", "root", "db");


$sql="SELECT student_id, student_name FROM Students";

$result=mysqli_query($conn, $sql);

// Output data of each row

while($row=mysqli_fetch_assoc($result))

{

echo "ID : " . $row["student_id"] . "<br>". "Name : ". $row["student_name"]."<br>";
```

```
}
echo "Data retrieved";
mysqli_close($conn);
?>
```

Deleting data from database

Delete statement is used to delete records from table.

Example:

```
<? php
$conn=mysqli_connect("localhost", "usern", "root", "db");

$sql="DELETE FROM Students WHERE student_id=4";
if(mysqli_query($conn, $sql))
{
 echo "Row deleted successfully!";
}
else
{
 echo "Error in deleting the record";
}

mysqli_close($conn);
?>
```

Updating data in database

The update statement is used to update database existing record in table.

Example:

```
<? php
$conn=mysqli_connect("localhost", "usern", "root", "db");

$sql="UPDATE Students SET age="25" WHERE student_id=2";
if(mysqli_query($conn, $sql))
{
 echo "Record updated successfully!";
}
else
{
 echo "Error in updating the record";
}

mysqli_close($conn);
?>
```