Database Examples:

Q. How to create the MySQL connection object using Python script?

```
import pymysql
connection_obj = pymysql.connect(host='localhost',user='root',password='root')
print(connection_obj )
```

Q. How to create the MySQL cursor object using Python script?

```
import pymysql
connection_obj  = pymysql.connect(host='localhost',user='root',password='root')
cursor_obj = connection_obj.cursor()
print(cursor_obj)
```

Q. How to show all available databases from MySQL DB using Python script?

```
import pymysql
connection_obj = pymysql.connect(host='localhost',user='root',password='root')
cursor_obj = connection_obj.cursor()
sql = cursor_obj.execute('show databases')
print(sql)  #  returns count of total dbs
# displays all dbs from cursor_obj
for db in cursor_obj:
    print(db)
```

Q. How to create a new database using Python script?

```
import pymysql
connection_obj = pymysql.connect(host='localhost',user='root',password='root')
cursor_obj = connection_obj.cursor()
sql = cursor_obj.execute('create database python_db2')
print(sql) # 1  returns
print('database created successfully.')
```

Note : If database already created then we will get error like bellow.

```
pymysql.err.ProgrammingError: (1007, "Can't create database 'python_db2';
database exists")
```

Q. How to use existing database using Python script?

```
import pymysql
mydb = pymysql.connect(
  host="localhost",
  user="myusername",
  password="mypassword",
  database="mydatabase"
)
```

Q. How to show tables from current db?

```
import pymysql
mydb = pymysql.connect(
  host="localhost",
  user="myusername",
  password="mypassword",
  database="mydatabase"
)
cur_obj = mydb.cursor()
sql = 'show tables;'
cur_obj.execute(sql)
for table in cur_obj:
    print(table)
```

Q. How to select data from a table?

```
#step1 -->> Import the required database connector module
import pymysql
#step2  --->> creating database conncection object
connection_object = pymysql.connect(
  host='localhost',
  user='root',
  password='root',
  database='mypython_db'
  )
print(connection_object)
```

```
#step3 --->> create the cursor object
cursor_object = connection_object.cursor()
print(cursor_object)
#step4 --->> create the required sql  queries
sql = 'select * from customers2;'
#step5 --->> execute the sql queries using execute() of cursor_object
cursor_object.execute(sql)
#step6 --->>  access the data from cursor object and display it.
for row in cursor_object:
  for col in row:
    print(col,end=' ')
  print()
```

Creating a Table :

--->> To create a table in MySQL, use the "CREATE TABLE" statement.

```
mysql> CREATE TABLE customers (name VARCHAR(255), address VARCHAR(255))
```

Q. How to create a table using Python Script?

```
# import the MYSQL connector|driver module
import pymysql
# creating the database connection object
connecton_object = pymysql.connect(
    host='localhost',
    user='root',
    password='root',
    database='mypython_db'
)
print(connecton_object)
# How to create the Cursor object
cursor_object = connecton_object.cursor()
print(cursor_object)
# create the required sql query for executing
sql = 'CREATE TABLE customers (name VARCHAR(255), address VARCHAR(255))'
# how to execute the sql queries.
```

```
cursor_object.execute(sql)
print('table created successfully.')
```

Q. How to show the available tables from current db ?

```
# import the MYSQL connector|driver module
import pymysql
# creating the database connection object

connecton_object = pymysql.connect(
   host='localhost',
   user='root',
   password='root',
   database='mypython_db'
   )
print(connecton_object)
# How to create the Cursor object
cursor_object = connecton_object.cursor()
print(cursor_object)
# create the required sql query for executing
sql = 'show tables;'
# how to execute the sql queries.
cursor_object.execute(sql)
# display the tables from cursor object.
for db  in cursor_object:
   print(db)
```

Q. How to create a primary key value for our table?

Primary Key:

When creating a table, you should also create a column with a unique key for each record.

This can be done by defining a **PRIMARY KEY**.

We use the statement "INT AUTO_INCREMENT PRIMARY KEY" which will insert a unique number for each record. Starting at 1, and increased by one for each record.

code:

```
sql = "CREATE TABLE customers2 (id INT AUTO_INCREMENT PRIMARY KEY, name VARCHAR(255), address VARCHAR(255))"
# How to execute the sql queries.
cursor_object.execute(sql)
```

Insert Into Table :

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

Insert a record in the "customers" table:

Example:

```
# import the MYSQL connector|driver module
import pymysql
# creating the database connection object
connecton_object = pymysql.connect(
   host='localhost',
   user='root',
   password='root',
   database='mypython_db'
   )
print(connecton_object)
# How to create the Cursor object
cursor_object = connecton_object.cursor()
print(cursor_object)
sql = "INSERT INTO customers2 (name, address) VALUES (%s, %s)"
val = ("John", "Highway 21")
# how to execute the sql queries.
cursor_object.execute(sql,val)
connecton_object.commit() # to save perminatly into database
print(cursor_object.rowcount, "record inserted.")
```

Important!:

Notice the statement: connecton_object.commit(). It is required to make the changes, otherwise no changes are made to the table.

Fill the "customers" table with data:

Select From a Table:

To select from a table in MySQL, use the "SELECT" statement:

Select all records from the "customers" table, and display the result:

Example:

```
import pymysql
mydb = pymysql.connect(
  host="localhost",
  user="root",
  password="root",
  database="mypython_db"
)
mycursor = mydb.cursor()
mycursor.execute("SELECT * FROM customers2")
myresult = mycursor.fetchall()
for x in myresult:
  print(x)
```

Note: We use the **fetchall()** method, which fetches all rows from the last executed statement.

Selecting Columns to select only some of the columns in a table, use the "SELECT" statement followed by the column name(s):

```
mycursor.execute("SELECT name, address FROM customers")
```

Using the fetchone() Method

If you are only interested in one row, you can use the **fetchone()** method.

The fetchone() method will return the **first row** of the result.

Example:

```
mycursor.execute("SELECT * FROM customers")
myresult = mycursor.fetchone()
```

Select With a Filter

When selecting records from a table, you can **filter the selection** by using the "**WHERE**" statement.

```
sql = "SELECT * FROM customers  WHERE address ='Park Lane 38'"
```

Update Table

You can update existing records in a table by using the "UPDATE" statement:

sql = "UPDATE customers SET address = 'Canyon 123' WHERE address = 'Valley 345'"

Important!:

Notice the statement: **mydb.commit**(). It is required to make the changes, otherwise no changes are made to the table.

Notice the WHERE clause in the UPDATE syntax:

The WHERE clause specifies which record or records that should be updated. If you omit the WHERE clause, all records will be updated!

Delete Record

You can delete records from an existing table by using the "DELETE FROM" statement:

```
sql = "DELETE FROM customers WHERE address = 'Mountain 21'"
```

Limit the Result

You can limit the number of records returned from the query, by using the "LIMIT" statement.

Example:

```
mycursor.execute("SELECT * FROM customers LIMIT 5")
myresult = mycursor.fetchall()
for x in myresult:
    print(x)
```

Start From Another Position:

If you want to return five records, starting from the third record, you can use the "**OFFSET**" keyword.

Start from position 3, and return 5 records.

Example:

```
mycursor.execute("SELECT * FROM customers LIMIT 5 OFFSET 2")
myresult = mycursor.fetchall()
for x in myresult:
     print(x)
```

Where Clause:

The `WHERE` clause is used to filter records. It is used to extract only those records that fulfill a specified condition.

WHERE Syntax:

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

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

```
mysql> SELECT * FROM Customers  WHERE Country = 'Mexico';
```

Text Fields vs. Numeric Fields

SQL requires single quotes around text values (most database systems will also allow double quotes).

However, numeric fields should not be enclosed in quotes:

Example

```
mysql> SELECT * FROM Customers  WHERE CustomerID = 1;
```

Operators in The WHERE Clause

The following operators can be used in the `WHERE` clause:

Operator	Description
=	Equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
<>	Not equal. **Note:** In some versions of SQL this operator may be written as !=
BETWEEN	Between a certain range
LIKE	Search for a pattern
IN	To specify multiple possible values for a column

For example, = , > , < , >= , <= , != , between , in , not in , like , etc ….

Examples:

```
mysql> SELECT * FROM  Products WHERE Price = 20;
mysql> SELECT * FROM  Products WHERE Price < 20;
mysql> SELECT * FROM  Products WHERE Price <= 20;
mysql> SELECT * FROM  Products WHERE Price like '2%';
mysql> SELECT * FROM  Products WHERE Price in (18,25);
mysql> SELECT * FROM  Products WHERE Price not in (18,25);
mysql> SELECT * FROM  Products WHERE Price like '2%';
mysql> SELECT * FROM  Products WHERE Price BETWEEN 20 AND 30;
```

The MySQL AND, OR and NOT Operators

The WHERE clause can be combined with 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

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

OR Syntax

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

NOT Syntax

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

The MySQL ORDER BY Keyword

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

The ORDER BY keyword sorts the records in ascending order by default. To sort the records in descending order, use the DESC keyword.

ORDER BY Syntax

```
mysql> SELECT column1, column2, ... FROM table_name
ORDER BY column1, column2, ... ASC|DESC;
Example:
mysql> SELECT * FROM Customers ORDER BY Country;
mysql> SELECT * FROM Customers ORDER BY Country DESC;
```

ORDER BY Several Columns Example

The following SQL statement selects all customers from the "Customers" table, sorted by the "Country" and the "CustomerName" column. This means that it orders by Country, but if some rows have the same Country, it orders them by CustomerName:

Example:

```
mysql> SELECT * FROM Customers ORDER BY Country,
CustomerName;
```

The MySQL INSERT INTO Statement

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

INSERT INTO Syntax

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

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

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

2. If you are adding values for all the columns of the table, you do not need to specify the column names in the SQL query. However, make sure the order of the values is in the same order as the columns in the table. Here, the INSERT INTO syntax would be as follows:

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

Write a SQL statement to inserts a new record in the "Customers" table:

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

The selection from the "Customers" table will now look like this:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
92	Cardinal	Tom B. Erichsen	Skagen 21	Stavanger	4006	Norway

Note: It is also possible to only insert data in specific columns.

The following SQL statement will insert a new record, but only insert data in the "CustomerName", "City", and "Country" columns (CustomerID will be updated automatically):

Example:

```
mysql> INSERT INTO Customers (CustomerName, City, Country)
VALUES ('Cardinal', 'Stavanger', 'Norway');
```

The selection from the "Customers" table will now look like this:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country

92	Cardinal	null	null	Stavanger	null	Norway

The MySQL UPDATE Statement

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

UPDATE Syntax

```
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!

Demo Database

Below is a selection from the "Customers" table in the Northwind sample database:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany

UPDATE Table

The following SQL statement updates the first customer (CustomerID = 1) with a new contact person *and* a new city.

Example:

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

The selection from the "Customers" table will now look like this:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Alfred Schmidt	Obere Str. 57	Frankfurt	12209	Germany

UPDATE Multiple Records

It is the WHERE clause that determines how many records will be updated.

The following SQL statement will update the PostalCode to 00000 for all records where country is "Mexico":

Example:

```
UPDATE Customers SET PostalCode = 00000 WHERE Country
= 'Mexico';
```

The selection from the "Customers" table will now look like this:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Alfred Schmidt	Obere Str. 57	Frankfurt	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	00000	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	00000	Mexico

Update Warning!

Be careful when updating records. If you omit the `WHERE` clause, ALL records will be updated!

Example:

```
UPDATE Customers SET PostalCode = 00000;
```

The selection from the "Customers" table will now look like this:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Alfred Schmidt	Obere Str. 57	Frankfurt	00000	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	00000	Mexico

The MySQL DELETE Statement

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

DELETE Syntax

```
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!

Demo Database

Below is a selection from the "Customers" table in the Northwind sample database:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico

SQL DELETE Example

The following SQL statement deletes the customer "Alfreds Futterkiste" from the "Customers" table:

Example:

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

The "Customers" table will now look like this:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico

Delete All Records

It is possible to delete all rows in a table without deleting the table. This means that the table structure, attributes, and indexes will be intact:

```
DELETE FROM table_name;
```

The following SQL statement deletes all rows in the "Customers" table, without deleting the table:

Example:

```
mysql> DELETE FROM Customers;
```

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.

LIMIT Syntax

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

MySQL LIMIT Examples

The following SQL statement selects the first three records from the "Customers" table:

Example:

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

ADD a WHERE CLAUSE

The following SQL statement selects the first three records from the "Customers" table, where the country is "Germany":

Example:

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