**Python & MySQL - Overview**

You must download a separate DB API module for each database you need to access. For example, if you need to access an Oracle database as well as a MySQL database, you must download both the Oracle and the MySQL database modules.

The DB API provides a minimal standard for working with databases using Python structures and syntax wherever possible. This API includes the following −

* Importing the API module.
* Acquiring a connection with the database.
* Issuing SQL statements and stored procedures.
* Closing the connection

MySQLdb

MySQLdb is an interface for connecting to a MySQL database server from Python.

**Install MySQL, Workbench, and mysql connector using Pycharm**

[MySQL :: Download MySQL Community Server](https://dev.mysql.com/downloads/mysql/)

(SELECT WINDOWS SERVICE OTHERWISE MYSQL WILL NOT BE VISIBLE IN SERVICES.MSC)

[MySQL :: Download MySQL Workbench](https://dev.mysql.com/downloads/workbench/)

Go to Pycharm> create project folder>menu button>file>Settings>+ SYBOL>install mysql-connector-python

TO START /STOP SERVER:

WINDOWS SEARCH>SERVICES>MYSQL RIGH CLICK>START OR STOP

YOU CAN ALSO GO TO TASK MANAGER USING CRTL+ALT+DEL

YOU CAN GO TO WIN SEARCH>MYSQL CONFIGURATOR TO ACTIVATE WIN SERVICES so that MySQL is visible in services

Install MySQL Database

* **MySQL DB:** MySQL is an open source database. You can download it from [MySQL Official Site](https://dev.mysql.com/downloads/mysql). We recommend downloading the full Windows installation.

In addition, download and install [MySQL Administrator](https://dev.mysql.com/downloads/gui-tools/) as well as [MySQL Query Browser.](https://dev.mysql.com/downloads/gui-tools/) These are GUI based tools that will make your development much easier.

Finally, download and unzip [MySQL Connector/J](https://dev.mysql.com/downloads/connector/j/3.1.html) (the MySQL JDBC driver) in a convenient directory. For the purpose of this tutorial we will assume that you have installed the driver at C:\Program Files\MySQL\mysql-connector-java-5.1.8.

Accordingly, set CLASSPATH variable to C:\Program Files\MySQL\mysql-connector-java-5.1.8\mysql-connector-java-5.1.8-bin.jar. Your driver version may vary based on your installation.

Set Database Credential

When we install MySQL database, its administrator ID is set to **root** and it gives provision to set a password of your choice.

Using root ID and password you can either create another user ID and password, or you can use root ID and password for your JDBC application.

There are various database operations like database creation and deletion, which would need administrator ID and password.

For rest of the JDBC tutorial, we would use MySQL Database with **guest** as ID and **guest123** as password.

If you do not have sufficient privilege to create new users, then you can ask your Database Administrator (DBA) to create a user ID and password for you.

Create Database

To create the **AMITPATEL** database, use the following steps −

Step 1

Open a **Command Prompt** and change to the installation directory as follows −

C:\>

C:\>cd Program Files\MySQL\bin

C:\Program Files\MySQL\bin>

**Note:** The path to **mysqld.exe** may vary depending on the install location of MySQL on your system. You can also check documentation on how to start and stop your database server.

Step 2

Start the database server by executing the following command, if it is already not running.

C:\Program Files\MySQL\bin>mysqld

C:\Program Files\MySQL\bin>

Step 3

Create the **AMITPATEL** database by executing the following command −

C:\Program Files\MySQL\bin> mysqladmin create AMITPATEL -u guest -p

Enter password: \*\*\*\*\*\*\*\*

C:\Program Files\MySQL\bin>

Create Table

To create the **Employees** table in AMITPATEL database, use the following steps −

Step 1

Open a **Command Prompt** and change to the installation directory as follows −

C:\>

C:\>cd Program Files\MySQL\bin

C:\Program Files\MySQL\bin>

Step 2

Login to the database as follows −

C:\Program Files\MySQL\bin>mysql -u guest -p

Enter password: \*\*\*\*\*\*\*\*

mysql>

Step 3

Create the table **Employees** as follows −

mysql> use AMITPATEL;

mysql> create table Employees

-> (

-> id int not null,

-> age int not null,

-> first varchar (255),

-> last varchar (255)

-> );

Query OK, 0 rows affected (0.08 sec)

mysql>

Create Data Records

Finally you create few records in Employee table as follows −

mysql> INSERT INTO Employees VALUES (100, 18, 'Zara', 'Ali');

Query OK, 1 row affected (0.05 sec)

mysql> INSERT INTO Employees VALUES (101, 25, 'Mahnaz', 'Fatma');

Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO Employees VALUES (102, 30, 'Zaid', 'Khan');

Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO Employees VALUES (103, 28, 'Sumit', 'Mittal');

Query OK, 1 row affected (0.00 sec)

mysql>

For a complete understanding on MySQL database, study the [MySQL Tutorial](https://www.tutorialspoint.com/mysql/index.htm).

MySQLdb

MySQLdb is an interface for connecting to a MySQL database server from Python. It implements the Python Database API v2.0 and is built on top of the MySQL C API.

How do I Install MySQLdb?

Before proceeding, you make sure you have MySQLdb installed on your machine. Just type the following in your Python script and execute it −

#!/usr/bin/python

import MySQLdb

If it produces the following result, then it means MySQLdb module is not installed −

Traceback (most recent call last):

File "test.py", line 3, in <module>

import MySQLdb

ImportError: No module named MySQLdb

To install MySQLdb module, use the following command −

For Ubuntu, use the following command -

$ sudo apt-get install python-pip python-dev libmysqlclient-dev

For Fedora, use the following command -

$ sudo dnf install python python-devel mysql-devel redhat-rpm-config gcc

For Python command prompt, use the following command -

pip install mysqlclient

**Note** − Make sure you have root privilege to install above module.

Now you are ready to start experimenting with Python.

**Python & MySQL - Connect Database Example**

Python MySQLdb provides **MySQLdb.connect()** function to open a database connection. This function takes multiple parameters and returns a connection object to do database operations.

Syntax

db = MySQLdb.connect(host, username, passwd, dbName, port, socket);

|  |  |
| --- | --- |
| **Sr.No.** | **Parameter & Description** |
| 1 | **host**  Optional − The host name running the database server. If not specified, then the default value will be **localhost:3306**. |
| 2 | **username**  Optional − The username accessing the database. If not specified, then the default will be the name of the user that owns the server process. |
| 3 | **passwd**  Optional − The password of the user accessing the database. If not specified, then the default will be an empty password. |
| 4 | **dbName**  Optional − database name on which query is to be performed. |
| 5 | **port**  Optional − the port number to attempt to connect to the MySQL server.. |
| 6 | **socket**  Optional − socket or named pipe that should be used. |

There are several other properties as well. Refer [MySQLdb](https://mysqlclient.readthedocs.io/user_guide.html" \l "functions-and-attributes) for complete reference.

You can disconnect from the MySQL database anytime using another connection object function **close()**.

Syntax

db.close()

Example

Try the following example to connect to a MySQL server −

Copy and paste the following example as mysql\_example.py −

#!/usr/bin/python

import MySQLdb

# Open database connection

db = MySQLdb.connect("localhost","root","root@123")

# prepare a cursor object using cursor() method

cursor = db.cursor()

# execute SQL query using execute() method.

cursor.execute("SELECT VERSION()")

# Fetch a single row using fetchone() method.

data = cursor.fetchone()

if data:

print('Version available: ', data)

else:

print('Version not retrieved.')

# disconnect from server

db.close()

Output

Execute the mysql\_example.py script using python and verify the output.

py mysql\_example.py

Version available: ('8.0.23',)

**Python & MySQL - Create Database Example**

Python uses **c.execute(q)** function to create or delete a MySQL database where c is cursor and q is the query to be executed.

Syntax

# execute SQL query using execute() method.

cursor.execute(sql)

|  |  |
| --- | --- |
| **Sr.No.** | **Parameter & Description** |
| 1 | **$sql**  Required - SQL query to create a MySQL database. |
|  |  |

Example

Try the following example to create a database −

Copy and paste the following example as mysql\_example.ty −

#!/usr/bin/python

import MySQLdb

# Open database connection

db = MySQLdb.connect("localhost","root","root@123")

# prepare a cursor object using cursor() method

cursor = db.cursor()

# execute SQL query using execute() method.

cursor.execute("CREATE DATABASE TUTORIALS")

print('Database created');

# disconnect from server

db.close()

Output

Execute the mysql\_example.py script using python and verify the output.

Database created

**Python & MySQL - Drop Database Example**

Python uses **c.execute(q)** function to create or delete a MySQL database where c is cursor and q is the query to be executed.

Syntax

# execute SQL query using execute() method.

cursor.execute(sql)

|  |  |
| --- | --- |
| **Sr.No.** | **Parameter & Description** |
| 1 | **$sql**  Required - SQL query to drop a MySQL database. |
|  |  |

Example

Try the following example to drop a database −

Copy and paste the following example as mysql\_example.ty −

#!/usr/bin/python

import MySQLdb

# Open database connection

db = MySQLdb.connect("localhost","root","root@123")

# prepare a cursor object using cursor() method

cursor = db.cursor()

# execute SQL query using execute() method.

cursor.execute("DROP DATABASE TUTORIALS")

print('Database dropped');

# disconnect from server

db.close()

Output

Execute the mysql\_example.py script using python and verify the output.

Database dropped

**Python & MySQL - Select Database Example**

Python MySQLdb provides **MySQLdb.connect()** function to select a database. This function takes multiple parameters and returns a connection object to do database operations.

Syntax

db = MySQLdb.connect(host, username, passwd, dbName, port, socket);

|  |  |
| --- | --- |
| **Sr.No.** | **Parameter & Description** |
| 1 | **host**  Optional − The host name running the database server. If not specified, then the default value will be **localhost:3306**. |
| 2 | **username**  Optional − The username accessing the database. If not specified, then the default will be the name of the user that owns the server process. |
| 3 | **passwd**  Optional − The password of the user accessing the database. If not specified, then the default will be an empty password. |
| 4 | **dbName**  Optional − database name on which query is to be performed. |
| 5 | **port**  Optional − the port number to attempt to connect to the MySQL server.. |
| 6 | **socket**  Optional − socket or named pipe that should be used. |

You can disconnect from the MySQL database anytime using another connection object function **close()**.

Syntax

db.close()

Example

Try the following example to connect to a MySQL database −

Copy and paste the following example as mysql\_example.py −

#!/usr/bin/python

import MySQLdb

# Open database connection

db = MySQLdb.connect("localhost","root","root@123", "TUTORIALS")

# prepare a cursor object using cursor() method

cursor = db.cursor()

# execute SQL query using execute() method.

cursor.execute("SELECT VERSION()")

# Fetch a single row using fetchone() method.

data = cursor.fetchone()

if data:

print('Version available: ', data)

else:

print('Version not retrieved.')

# disconnect from server

db.close()

Output

Execute the mysql\_example.py script using python and verify the output.

py mysql\_example.py

Version available: ('8.0.23',)

**Python & MySQL - Create Table**

Python uses **c.execute(q)** function to create a table where c is cursor and q is the query to be executed.

Syntax

# execute SQL query using execute() method.

cursor.execute(sql)

|  |  |
| --- | --- |
| **Sr.No.** | **Parameter & Description** |
| 1 | **$sql**  Required - SQL query to create a table. |
|  |  |

Example

Try the following example to create a table −

Copy and paste the following example as mysql\_example.ty −

#!/usr/bin/python

import MySQLdb

# Open database connection

db = MySQLdb.connect("localhost","root","root@123", "TUTORIALS")

# prepare a cursor object using cursor() method

cursor = db.cursor()

sql = """CREATE TABLE tutorials\_tbl(

tutorial\_id INT NOT NULL AUTO\_INCREMENT,

tutorial\_title VARCHAR(100) NOT NULL,

tutorial\_author VARCHAR(40) NOT NULL,

submission\_date DATE,

PRIMARY KEY ( tutorial\_id )); """;

# execute SQL query using execute() method.

cursor.execute(sql)

print('tutorials\_tbl created')

# disconnect from server

db.close()

Output

Execute the mysql\_example.py script using python and verify the output.

tutorials\_tbl created

**Python & MySQL - Drop Table Example**

Python uses **c.execute(q)** function to drop a table where c is cursor and q is the delete query to be executed.

Syntax

# execute SQL query using execute() method.

cursor.execute(sql)

|  |  |
| --- | --- |
| **Sr.No.** | **Parameter & Description** |
| 1 | **$sql**  Required - SQL query to drop a table. |
|  |  |

Example

Try the following example to drop a table −

Copy and paste the following example as mysql\_example.ty −

#!/usr/bin/python

import MySQLdb

# Open database connection

db = MySQLdb.connect("localhost","root","root@123","TUTORIALS")

# prepare a cursor object using cursor() method

cursor = db.cursor()

# execute SQL query using execute() method.

cursor.execute("Drop Table tutorials\_tbl")

print('Table dropped');

# disconnect from server

db.close()

Output

Execute the mysql\_example.py script using python and verify the output.

Table dropped

**PHP & MySQL - Insert Records Example**

Python uses **c.execute(q)** function to insert a record(s) in a table where c is cursor and q is the insert query to be executed.

Syntax

# execute SQL query using execute() method.

cursor.execute(sql)

# commit the record

db.commit()

# get the row id for inserted record

print("ID:", cursor.lastrowid)

# print the number of records inserted

print(mycursor.rowcount, "records inserted.")

|  |  |
| --- | --- |
| **Sr.No.** | **Parameter & Description** |
| 1 | **$sql**  Required - SQL query to insert record(s) in a table. |
|  |  |

Example

Try the following example to insert records in a table −

Copy and paste the following example as mysql\_example.ty −

#!/usr/bin/python

import MySQLdb

# Open database connection

db = MySQLdb.connect("localhost","root","root@123", "TUTORIALS")

# prepare a cursor object using cursor() method

cursor = db.cursor()

sql = """INSERT INTO tutorials\_tbl

(tutorial\_title,tutorial\_author, submission\_date)

VALUES ('HTML 5', 'Robert', '2010-02-10'),

('Java', 'Julie', '2020-12-10'),

('JQuery', 'Julie', '2020-05-10')

"""

# execute SQL query using execute() method.

cursor.execute(sql)

# commit the record

db.commit()

# get the row id for inserted record

print("ID:", cursor.lastrowid)

# print the number of records inserted

print(cursor.rowcount, "records inserted.")

# disconnect from server

db.close()

Output

Execute the mysql\_example.py script using python and verify the output.

ID: 5

3 records inserted.

**Python & MySQL - Select Records Example**

Select/Read Operation on any database means to fetch some useful information from the database.

Once our database connection is established, you are ready to make a query into this database. You can use either **fetchone()** method to fetch single record or **fetchall()** method to fetch multiple values from a database table.

* **fetchone()** − It fetches the next row of a query result set. A result set is an object that is returned when a cursor object is used to query a table.
* **fetchall()** − It fetches all the rows in a result set. If some rows have already been extracted from the result set, then it retrieves the remaining rows from the result set.
* **rowcount** − This is a read-only attribute and returns the number of rows that were affected by an execute() method.

Syntax

# execute SQL query using execute() method.

cursor.execute(sql)

result = cursor.fetchall()

for record in result:

print(record)

|  |  |
| --- | --- |
| **Sr.No.** | **Parameter & Description** |
| 1 | **$sql**  Required - SQL query to select record(s) from a table. |
|  |  |

Example

Try the following example to select records from a table −

Copy and paste the following example as mysql\_example.ty −

#!/usr/bin/python

import MySQLdb

# Open database connection

db = MySQLdb.connect("localhost","root","root@123", "TUTORIALS")

# prepare a cursor object using cursor() method

cursor = db.cursor()

sql = "Select \* from tutorials\_tbl"

# execute SQL query using execute() method.

cursor.execute(sql)

# fetch all records from cursor

result = cursor.fetchall()

# iterate result and print records

for record in result:

print(record)

# disconnect from server

db.close()

Output

Execute the mysql\_example.py script using python and verify the output.

(1, 'HTML 5', 'Robert', datetime.date(2010, 2, 10))

(2, 'Java', 'Julie', datetime.date(2020, 12, 10))

(3, 'JQuery', 'Julie', datetime.date(2020, 5, 10))

**Python & MySQL - Update Records Example**

Python uses **c.execute(q)** function to update a record from a table where c is cursor and q is the update query to be executed.

Syntax

# execute SQL query using execute() method.

cursor.execute(sql)

cursor.commit()

# get the record count updated

print(mycursor.rowcount, "record(s) affected")

|  |  |
| --- | --- |
| **Sr.No.** | **Parameter & Description** |
| 1 | **$sql**  Required - SQL query to update record(s) in a table. |
|  |  |

Example

Try the following example to update records in a table −

Copy and paste the following example as mysql\_example.ty −

#!/usr/bin/python

import MySQLdb

# Open database connection

db = MySQLdb.connect("localhost","root","root@123", "TUTORIALS")

# prepare a cursor object using cursor() method

cursor = db.cursor()

sql = "UPDATE tutorials\_tbl set tutorial\_title = "Learning Java" where tutorial\_id = 2"

# execute SQL query using execute() method.

cursor.execute(sql)

db.commit()

# get the record count updated

print(cursor.rowcount, " record(s) affected")

# disconnect from server

db.close()

Output

Execute the mysql\_example.py script using python and verify the output.

1 record(s) affected

**Python & MySQL - Delete Records Example**

Python uses **c.execute(q)** function to delete a record from a table where c is cursor and q is the delete query to be executed.

Syntax

# execute SQL query using execute() method.

cursor.execute(sql)

cursor.commit()

# get the record count updated

print(mycursor.rowcount, "record(s) affected")

|  |  |
| --- | --- |
| **Sr.No.** | **Parameter & Description** |
| 1 | **$sql**  Required - SQL query to delete record(s) in a table. |
|  |  |

Example

Try the following example to insert records in a table −

Copy and paste the following example as mysql\_example.ty −

#!/usr/bin/python

import MySQLdb

# Open database connection

db = MySQLdb.connect("localhost","root","root@123", "TUTORIALS")

# prepare a cursor object using cursor() method

cursor = db.cursor()

sql = "Delete from tutorials\_tbl where tutorial\_id = 2"

# execute SQL query using execute() method.

cursor.execute(sql)

db.commit()

# get the record count updated

print(cursor.rowcount, " record(s) affected")

# disconnect from server

db.close()

Output

Execute the mysql\_example.py script using python and verify the output.

1 record(s) affected

**Python & MySQL - Where Clause Example**

Python uses **c.execute(q)** function to select a record(s) conditionally using Where Clause from a table where c is cursor and q is the select query to be executed.

Syntax

# execute SQL query using execute() method.

cursor.execute(sql)

result = cursor.fetchall()

for record in result:

print(record)

|  |  |
| --- | --- |
| **Sr.No.** | **Parameter & Description** |
| 1 | **$sql**  Required - SQL query to select record(s) from a table. |
|  |  |

Example

Try the following example to select records from a table −

Copy and paste the following example as mysql\_example.ty −

#!/usr/bin/python

import MySQLdb

# Open database connection

db = MySQLdb.connect("localhost","root","root@123", "TUTORIALS")

# prepare a cursor object using cursor() method

cursor = db.cursor()

sql = "Select \* from tutorials\_tbl Where tutorial\_id = 3"

# execute SQL query using execute() method.

cursor.execute(sql)

# fetch all records from cursor

result = cursor.fetchall()

# iterate result and print records

for record in result:

print(record)

# disconnect from server

db.close()

Output

Execute the mysql\_example.py script using python and verify the output.

(3, 'JQuery', 'Julie', datetime.date(2020, 5, 10))

**Python & MySQL - Like Clause Example**

Python uses **c.execute(q)** function to select a record(s) conditionally using Like Statement from a table where c is cursor and q is the select query to be executed.

Syntax

# execute SQL query using execute() method.

cursor.execute(sql)

result = cursor.fetchall()

for record in result:

print(record)

|  |  |
| --- | --- |
| **Sr.No.** | **Parameter & Description** |
| 1 | **$sql**  Required - SQL query to select record(s) from a table using Like Statement. |
|  |  |

Example

Try the following example to select records from a table using Like Statement −

Copy and paste the following example as mysql\_example.ty −

#!/usr/bin/python

import MySQLdb

# Open database connection

db = MySQLdb.connect("localhost","root","root@123", "TUTORIALS")

# prepare a cursor object using cursor() method

cursor = db.cursor()

sql = "Select \* from tutorials\_tbl Where tutorial\_title like 'J%'"

# execute SQL query using execute() method.

cursor.execute(sql)

# fetch all records from cursor

result = cursor.fetchall()

# iterate result and print records

for record in result:

print(record)

# disconnect from server

db.close()

Output

Execute the mysql\_example.py script using python and verify the output.

(6, 'Java', 'Julie', datetime.date(2020, 12, 10))

(7, 'JQuery', 'Julie', datetime.date(2020, 5, 10))

**Python & MySQL - Sorting Data Example**

Python uses **c.execute(q)** function to select a record(s) in sorted order using sort by Statement from a table where c is cursor and q is the select query to be executed.

Syntax

# execute SQL query using execute() method.

cursor.execute(sql)

result = cursor.fetchall()

for record in result:

print(record)

|  |  |
| --- | --- |
| **Sr.No.** | **Parameter & Description** |
| 1 | **$sql**  Required - SQL query to select record(s) from a table and sort using sort by Statement. |
|  |  |

Example

Try the following example to select sorted records from a table using sort by Statement −

Copy and paste the following example as mysql\_example.ty −

#!/usr/bin/python

import MySQLdb

# Open database connection

db = MySQLdb.connect("localhost","root","root@123", "TUTORIALS")

# prepare a cursor object using cursor() method

cursor = db.cursor()

sql = "Select \* from tutorials\_tbl order by tutorial\_title asc"

# execute SQL query using execute() method.

cursor.execute(sql)

# fetch all records from cursor

result = cursor.fetchall()

# iterate result and print records

for record in result:

print(record)

# disconnect from server

db.close()

Output

Execute the mysql\_example.py script using python and verify the output.

(1, 'HTML 5', 'Robert', datetime.date(2010, 2, 10))

(2, 'Java', 'Julie', datetime.date(2020, 12, 10))

(3, 'JQuery', 'Julie', datetime.date(2020, 5, 10))

**Python & MySQL - Using Joins Example**

Python uses **c.execute(q)** function to select a record(s) from a table where c is cursor and q is the select query to be executed.

Syntax

# execute SQL query using execute() method.

cursor.execute(sql)

result = cursor.fetchall()

for record in result:

print(record)

|  |  |
| --- | --- |
| **Sr.No.** | **Parameter & Description** |
| 1 | **$sql**  Required - SQL query to select record(s) from a table. |
|  |  |

First create a table in MySQL using following script and insert two records.

create table tcount\_tbl(

tutorial\_author VARCHAR(40) NOT NULL,

tutorial\_count int

);

insert into tcount\_tbl values('Julie', 2);

insert into tcount\_tbl values('Robert', 1);

Example

Try the following example to get records from a two tables using Join. −

Copy and paste the following example as mysql\_example.ty −

#!/usr/bin/python

import MySQLdb

# Open database connection

db = MySQLdb.connect("localhost","root","root@123", "TUTORIALS")

# prepare a cursor object using cursor() method

cursor = db.cursor()

sql = """SELECT a.tutorial\_id, a.tutorial\_author, b.tutorial\_count

FROM tutorials\_tbl a, tcount\_tbl b

WHERE a.tutorial\_author = b.tutorial\_author"""

# execute SQL query using execute() method.

cursor.execute(sql)

# fetch all records from cursor

result = cursor.fetchall()

# iterate result and print records

for record in result:

print(record)

# disconnect from server

db.close()

Output

Execute the mysql\_example.py script using python and verify the output.

(1, 'Robert', 1)

(2, 'Julie', 2)

(3, 'Julie', 2)

**Python & MySQL - Performing Transactions**

Transactions are a mechanism that ensures data consistency. Transactions have the following four properties −

* **Atomicity** − Either a transaction completes or nothing happens at all.
* **Consistency** − A transaction must start in a consistent state and leave the system in a consistent state.
* **Isolation** − Intermediate results of a transaction are not visible outside the current transaction.
* **Durability** − Once a transaction was committed, the effects are persistent, even after a system failure.

The Python DB API 2.0 provides two methods to either *commit* or *rollback* a transaction.

Example

You already know how to implement transactions. Here is again similar example −

# Prepare SQL query to DELETE required records

sql = "Delete from tutorials\_tbl where tutorial\_id = 2"

try:

# Execute the SQL command

cursor.execute(sql)

# Commit your changes in the database

db.commit()

except:

# Rollback in case there is any error

db.rollback()

COMMIT Operation

Commit is the operation, which gives a green signal to database to finalize the changes, and after this operation, no change can be reverted back.

Here is a simple example to call **commit** method.

db.commit()

ROLLBACK Operation

If you are not satisfied with one or more of the changes and you want to revert back those changes completely, then use **rollback()** method.

Here is a simple example to call **rollback()** method.

db.rollback()

Disconnecting Database

To disconnect Database connection, use close() method.

db.close()

If the connection to a database is closed by the user with the close() method, any outstanding transactions are rolled back by the DB. However, instead of depending on any of DB lower level implementation details, your application would be better off calling commit or rollback explicitly.

**Python & MySQL - Handling Errors**

There are many sources of errors. A few examples are a syntax error in an executed SQL statement, a connection failure, or calling the fetch method for an already canceled or finished statement handle.

The DB API defines a number of errors that must exist in each database module. The following table lists these exceptions.

|  |  |
| --- | --- |
| **Sr.No.** | **Exception & Description** |
| 1 | **Warning**  Used for non-fatal issues. Must subclass StandardError. |
| 2 | **Error**  Base class for errors. Must subclass StandardError. |
| 3 | **InterfaceError**  Used for errors in the database module, not the database itself. Must subclass Error. |
| 4 | **DatabaseError**  Used for errors in the database. Must subclass Error. |
| 5 | **DataError**  Subclass of DatabaseError that refers to errors in the data. |
| 6 | **OperationalError**  Subclass of DatabaseError that refers to errors such as the loss of a connection to the database. These errors are generally outside of the control of the Python scripter. |
| 7 | **IntegrityError**  Subclass of DatabaseError for situations that would damage the relational integrity, such as uniqueness constraints or foreign keys. |
| 8 | **InternalError**  Subclass of DatabaseError that refers to errors internal to the database module, such as a cursor no longer being active. |
| 9 | **ProgrammingError**  Subclass of DatabaseError that refers to errors such as a bad table name and other things that can safely be blamed on you. |
| 10 | **NotSupportedError**  Subclass of DatabaseError that refers to trying to call unsupported functionality. |

Your Python scripts should handle these errors, but before using any of the above exceptions, make sure your MySQLdb has support for that exception. You can get more information about them by reading the DB API 2.0 specification.