



DATABASE MANAGEMENT MADE SIMPLE THROUGH PYTHON

BE A BETTER PROGRAMMER !

-SURYA PRAKASH.S

Database Management made Simple through Python

Be a better programmer!

Suryaprakash S

*This book is Dedicated to my parents,
Smt.Amudha.S & Mr.Sugendra.C*

Tweet This Book!

Please help Suryaprakash S by spreading the word about this book on [Twitter](#)!

The suggested hashtag for this book is [#suryaprakash_s](#).

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

[#suryaprakash_s](#)

Contents

Data, Database, Database management system:	1
Actors on the scene:	2
Workers behind the scene:	5
Categories of Data models:	5
File management system (FMS):	7
Relational database system (RDS):	8
Codd's rules:	9
Advantages of using DBMS approach:	12
Disadvantages of using DBMS:	12
Database Basics:	13
DBMS languages and commands:	14
Data definition language:	15
To connect to mysql:	15
Database:	16
Table:	19
Alter:	20
To increase or decrease the width:	20
To add a new column:	20
To change the data type:	21
To remove a column from table:	21
To truncate the table:	22
To drop a table:	23
Data manipulating language:	24
To insert a Value into the table:	24
To insert NULL values into the table:	24
To insert the limited values into the table:	25
Select statements:	25
To list the all details from the table:	26
To list only one column from the table:	26
To list the columns with aliasing name from the table:	27
Where clause:	28
Using of simple where clause:	28
Using arithmetic operator in where clause:	29
Using of relational operators in where clause:	29
Using logical operators in where clause:	30

Using of between in where clause:	31
To negate the values in the select statement:	31
Distinct clause:.....	32
Using distinct clause in select statement:.....	32
Order by clause:	33
Using of order by clause in select statement:.....	33
Update:.....	33
Using update statement:	34
Delete:.....	34
Using DELETE statement:	34
Transaction control language:	36
Commit:	36
Save point:	37
Rollback:.....	37
Data control language (DCL):	38
Grant:	38
Revoke:	38
Special functions and operators of sql:	40
IN operator:.....	40
Using of IN operator:	40
Like operator:.....	41
Using of LIKE operator:	41
Round function:	41
Using of round function:	42
Dual:	42
System date function:.....	43
Using of sysdate () function:.....	43
System_user ():	43
Using of System_user ():	44
Greatest ():	44
Using of greatest ():	44
Numeric functions:	45
Using of abs ():	45
Using of ceil ():	46
Using of floor:.....	46
Using of trigonometric functions:.....	47
Using of exp ():	48
Using of power ():.....	48
Using of ln ():.....	49
Using of mod ():.....	50
Character function:	50

Soundex:	52
Extract:.....	53
Aggregate functions:	54
Count:	54
Sum:	54
Max:	54
Average:.....	54
Min:.....	54
Group by clause and having clause:	56
Group by clause:	56
Having clause:.....	56
Special functions and operators of sql:	58
IN operator:.....	58
Using of IN operator:	58
Like operator:.....	59
Using of LIKE operator:	59
Round function:	59
Using of round function:	60
Dual:	60
System date function:.....	61
Using of sysdate () function:.....	61
System_user ():	61
Using of System_user ():	62
Greatest ():	62
Using of greatest ():	62
Numeric functions:	63
Using of abs ():	63
Using of ceil ():	64
Using of floor:.....	64
Using of trigonometric functions:.....	65
Using of exp ():	66
Using of power ():.....	66
Using of ln ():.....	67
Using of mod ():.....	68
Character function:	68
Soundex:	70
Extract:.....	71
Constraints:	72
Domain integrity constraints:.....	72
Not null constraints:.....	72
Check constraints:.....	72

Example for creating a table with check constraints:	73
Entity integrity constraints:.....	74
Unique constraints:	74
Primary key constraints:	74
Table level and column level constraints:	74
Example for creating a table with a unique key constraint:	74
Example for creating a table with a primary key constraint:	75
Referential integrity constraint:	75
Example for creating table with referential integrity constraint:	76
Set operators:	77
Example for union operator:	77
Example for union all operators:.....	78
Joins:	79
Simple join:.....	79
Equi-join:	79
Non equi-join:	79
Cross join:.....	79
Examples for simple join (cross join):	79
Example for simple joins (equi join):	80
Example of simple join (non equi join):	80
Self join:.....	81
Examples for self join:	81
Outer join:	81
Example of Right outer join:	82
Example of left outer join:	82
Natural join:	83
Example of natural join;	83
Sub queries and locking:	84
Sub queries:.....	84
Example of sub query:	84
Concept of locking:.....	84
Row level locking:	85
Example of row level locking:	85
Table level lock:.....	86
Share lock:	86
Share update lock:.....	86
Exclusive lock:.....	87
No wait:.....	88
Dead lock:	88
Database objects:	89

Synonyms:	89
Sequences:	90
Views:	91
Example of view with check option:	92
Without check option:	92
DML statements and join views:	93
Key preserved tables:	93
Functions in a view:	94
Partition view:	94
Index:	94
Unique index:	95
Composite index:	96
Reverse key index:	97
Bitmap index:	97
Clusters:	98
Cluster key:	98
Gui development using python and dbms:	99

Data, Database, Database management system:

The person sitting in that enquiry booth is eagerly waiting to respond to the customer's queries. Many times he may direct you to different venues and corners of his organisation to meet person concerned in getting your application or work to be processed. He may supply the useful details about the activities of his organization. In short, his sole responsibility is to cater to the information needs of stakeholders of that organization. He has the collection of discrete data of wide variety about his organization's activities and services offered by them, contact person details, phone number, enquiry form, Address books, Visitor's Diary and so on.... He will do all necessary permutation and combination to supply the necessary data in response to end user's queries, such that it should be mean something useful to customer who raises the query at the enquiry Counter. The very purpose of such Info-data base systems is to fulfil the information needs of the end user's!!!

It's here the term data, database and database systems appear in right sense and catch the reader's attention. **Data** is any fact or figures that can be stored or recorded. Data is discrete and scattered many times. It becomes useful information when it's made concrete and /or a structured one by a way of collection, storage, processing which when get associate meaning. The collection of such inter-related data is called **Database** and finally the collection of inter-related data and a set of program that enables user to create and maintain database and access this data is called the **Database management system**. Defining a database involves specifying the data types, structures, and constraints of the data to be stored in the database in the form of a database catalog or dictionary, it is called **Meta-data**. The primary objective of a DBMS is to provide a convenient environment to create, retrieve, store and manipulate a database. In short, DBMS software package facilitates the defining, constructing and manipulating processes on databases subject to different applications.

Actors on the scene:

For the small personal database, one person typically defines, constructs, and manipulates the database, and there is no sharing. However, in large organizations, many people are involved in the design, use, and maintenance of a large database with hundreds of users. They are,

- Database Administrators (DBA):

In any organization where many people use the same resources, there is a need for a chief administrator to oversee and manage these resources. In a database environment, the primary resource is database itself, and the secondary resource is the DBMS and related software. Administering these resources is the responsibility of the **database administrator (DBA)**.

DBA is responsible for authorizing access to the database, coordinating and monitoring its use and acquiring software and hardware resources as needed. The DBA is accountable for problems such as breach of security or poor system response time. In large organisations, the DBA is assisted by a staff that carries out these functions.

Functions or responsibilities of DBA include:

- Design of the conceptual and physical schemas:

The DBA is responsible for interacting with the users of the system to understand what data is to be stored in the DBMS and how it is likely to be used. The DBA creates the original schema by writing a set of definitions and is permanently stored in the 'Data Dictionary'.

- Security and Authorization:

The DBA is responsible for ensuring the unauthorized data access is not permitted. The granting of different types of authorization allows the DBA to regulate which parts of the database various users can access.

- Storage structure and Access method definition:

The DBA creates appropriate storage structures and access methods by writing a set of definitions, which are translated by the DDL compiler.

- Data Availability and Recovery from Failures:

The DBA must take steps to ensure that if the system fails, users can continue to access as much of the uncorrupted data as possible. The DBA also work to restore the data to consistent state.

- Database Tuning:

The DBA is responsible for modifying the database to ensure adequate Performance as requirements change.

- Integrity Constraint Specification:

The integrity constraints are kept in a special system structure that is consulted by the DBA whenever an update takes place in the system.

- Database Designers:

Database designers are responsible for identifying the data to be stored in the database and for choosing appropriate structures to represent and store this data. These tasks are mostly undertaken before the database is actually implemented and populated with data. It is the responsibility of database designers to communicate with all prospective database users in order to understand their requirements and to create a design that meets these requirements. Database designers typically interact with each potential group of users and develop **views** of database that meet the data and processing requirements of these groups. Each view is then analyzed and integrated with the views of other user groups. The final database design must be capable of supporting the requirements of all user groups.

Function and responsibilities of database designers:

- Create conceptual design of the database (based on requirement analysis like what's the kind of database and dimensions of the database as is needed by the organization)

- Group data items, define tables, identify primary keys, constraints and define the relationship – create now the data model and normalise etc., during the design process.

- Ensure resemblance of the data model towards real-world model.

- End Users:

End users are the people whose jobs require access to the database for querying, updating, and generating reports; the database primarily exists for their use. There are several categories of end users:

- Casual End users:

These people occasionally access the database, but they may need different information each time.

- Naive or Parametric End Users:

Their job function revolves around constantly querying and updating the database using standard types of queries and updates.

- Sophisticated End Users:

These include Engineers, Scientists, Business analyst and others familiarize to implement their applications to meet their complex requirements.

- Stand alone End users:

These people maintain personal databases by using ready-made program packages that provide easy to use menu based interfaces.

- System analysts and application programmers:

System analyst:

System analyst is a person who should understand the requirements of the end user and develop accordingly, the specifications or procedures for transactions on computers.

In other words, the main function of a system analyst is to analyse and develop computer procedures or specifications for transactions to be carried out on computers, that meet the operational and information requirements of an organisation / End user.

Function and responsibilities of system analyst:

- Prepare and understand problem definition.
- Conduct a study of current procedures, specifications.
- Design the proposed system for acceptance.
- Prepare system flow – charts.
- Educate user departments.
- Monitor and review new system.

Application programmers:

Application programmers are software engineers or computer professionals who implement the user requirement specifications as one or more application programs. They can select one or more RAD tools (RAD- Rapid application development) to develop attractive user interfaces in terms of forms and reports.

Function and responsibilities of application programmers:

- Obtain specifications from project leaders / system analyst.
- Write and develop computer programs accordingly.
- Perform testing, debugging, coding of the programs hence developed.
- Facilitate integration and maintenance of various program modules.
- Provide for documentation of the tested program/s.

Workers behind the scene:

Database Designers and Implementers:

The people who design and implement the DBMS modules and interfaces as a software package.

Tool Developers:

Include persons who design and implement tools consisting the packages for design, performance monitoring, and prototyping and test data generation.

Operators and maintenance personnel:

The system administration personnel who are responsible for the actual running and maintenance of the hardware and software environment for the database system.

Categories of Data models:

The Data models describe the availability of services to an interfacing application as how to access a data item when other relevant data items are known. The basic differences among different categories of data models exist in terms of various methods of expressing relationships and constraints among data items. Let's classify the data models based on the concepts they use to describe the structure of a database:

Low level or Physical data model:

They provide concepts about physical storage of data i.e... How to store data in a computer hard disk. They specify about records formats, record ordering and access paths etc. Low level data model details are not meant for typical end users.

Examples: Heaps, ISAM, VSAM, Hash, and Btree.

High level or logical data models:

These models are also referred to as conceptual data models. They provide concepts for typical end users at high level i.e., that are independent of physical storage. Accordingly, high level data models provide concepts such as entities, attributes, relationship and data structures etc. High level data model details can be easily understood by typical end users.

Object – based logical model:

These models can be used in describing the data at the logical and view levels.

These models are having flexible structuring capabilities classified into following types.

- The entity-relationship model.
- The object-oriented model.
- The semantic data model.
- The functional data model.

Record based logical model:

These models can also be used in describing the data at the logical and view levels.

These models can be used for both to specify the overall logical structure of the database and a higher-level description.

These models can be classified into,

- Relational management system.
- Network management system.
- Hierarchal management system.
- File management system.

File management system (FMS):

The FMS was the first method used to store data in a computation database. The data item is stored sequentially in one large file. A particular relationship cannot be drawn between the items other than the sequence in which it is stored if a particular data item has to be located the search start at the beginning and items are checked sequentially till the required item is found.

Limitations / disadvantages of File – oriented approach:

- Data redundancy is increased.
- Data ownership
- Data / program Dependency –(program are tied to data files, since data definitions are part of the application programs themselves)
- Lack of flexibility - (in retrieving information in a desired manner)
- Lack of data integrity-(Difficult to relate as a whole the various data items belonging to different files)
- No centralised control over pool of data and / or data management is scattered or spread over among different application program.

Advantages of File - oriented approach:

- Simple straight forward approach to conventional data processing applications.
- Optimized for a particular application.
- Less expensive.

Hierarchical management system (HMS):

Data storage is in the form of a parent child relationship. The origin of a data tree is root; data located at branch of particular different level from the root is called the node. The last node in the series is called the leaf (child). This node supports one too many relationship each child has pointer to numerous siblings and there is just one pointer to the parent there resulting in one too many relationship .suppose an information is required say some id it is not necessary the dbms to search the whole file to locate the data instead of the follows the dept and branch and fetches the data.

Limitations / disadvantages of Hierarchical – oriented approach:

- Redundancy.
- Complexity.
- Many too many relationships are not possible.

Network database system (NDS):

The main idea behind NDS is to bring about many too many relationships. The relationship between the different data item is called sets. This sequence is also use a pointer to locate a particular record. In the network model the data is also represented by links. Data is organised in the form of graphs i.e. some entities can be accessed through several path, the network data model many parent node and a child can also have many pro node in their network model permits the modelling of many too many relationships in data.

Limitations / disadvantages of Network database system:

- Redundancy can be eliminated.
- But high degree complexity exists in maintaining and managing the links as the data base grows – up.

Relational database system (RDS):

- Any dbms which allows the user to have related multiple tables based

on key fields is called RDBMS.

- Dr E. F. Codd first introduced the relational model. That allows data to be represented in simple row column format each data fields is considered as a column and each record is considered as a rows of a table.
- Consider a set of raw data, which has to be organised the entities and corresponding attributes are recognised and defined then put into the tables and these are then related.
- Different relationships between the various tables are achieved by mathematical set functions namely JOIN and UNION.

Limitations / disadvantages of Relational database system:

- The clear cut interface cannot be determined.
- Reusability of a structure is not possible through unlike the previous model is altering the structure is possible maintenance of the relational structure becomes a continuous issue at times.

Codd's rules:

Dr Edgar F. Codd, after his extensive research on the Relational Model of database systems, came up with twelve rules of his own, which according to him, a database must obey in order to be regarded as a true relational database.

These rules can be applied on any database system that manages stored data using only its

Relational capabilities. This is a foundation rule, which acts as a base for all the other rules.

Rule 1: Information Rule

The data stored in a database, may it be user data or metadata, must be a value of some table

Cell. Everything in a database must be stored in a table format.

Rule 2: Guaranteed Access Rule

Every single data element *value* is guaranteed to be accessible logically with a combination of

Table-name, primary-key row value, and attribute-name column value. No other means, such as pointers, can be used to access data.

Rule 3: Systematic Treatment of NULL Values

The NULL values in a database must be given a systematic and uniform treatment. This is a very Important rule because a NULL can be interpreted as one the following – data is missing, data is

Not known, or data is not applicable.

Rule 4: Active Online Catalog

The structure description of the entire database must be stored in an online catalog, known as

Data dictionary, which can be accessed by authorized users. Users can use the same query Language to access the catalog which they use to access the database itself.

Rule 5: Comprehensive Data Sub-Language Rule

A database can only be accessed using a language having linear syntax that supports data

Definition, data manipulation, and transaction management operations. This language can be used directly or by means of some application. If the database allows access to data without any

Help of this language, then it is considered as a violation.

Rule 6: View Updating Rule

All the views of a database, which can theoretically be updated, must also be updatable by the

System.

Rule 7: High-Level Insert, Update, and Delete Rule

A database must support high-level insertion, updation, and deletion. This must not be limited to a single row, that is, it must also support union, intersection and minus operations to yield sets of Data records.

Rule 8: Physical Data Independence

The data stored in a database must be independent of the applications that access the database.

Any change in the physical structure of a database must not have any impact on how the data is

Being accessed by external applications.

Rule 9: Logical Data Independence

The logical data in a database must be independent of its user's view *application*. Any change in Logical data must not affect the applications using it. For example, if two tables are merged or one is split into two different tables, there should be no impact or change on the user application. This is one of the most difficult rules to apply.

Rule 10: Integrity Independence

A database must be independent of the application that uses it. All its integrity constraints can be independently modified without the need of any change in the application. This rule makes a

Database independent of the front-end application and its interface.

Rule 11: Distribution Independence

The end-user must not be able to see that the data is distributed over various locations. Users

Should always get the impression that the data is located at one site only. This rule has been

Regarded as the foundation of distributed database systems.

Rule 12: Non-Subversion Rule

If a system has an interface that provides access to low-level records, then the interface must not be able to subvert the system and bypass security and integrity constraints.

Advantages of using DBMS approach:

- Controlling redundancy.
- Restricting Unauthorized Access.
- Providing persistent storage for program objects.
- Providing storage structures for efficient query processing.
- Providing backup and recovery.
- Providing multiple user interfaces.
- Representing complex relationships among data.
- Enforcing integrity constraints.
- Permitting inferencing and actions using rules.
- Potential for enforcing the standards.
- Reduced application development time.
- Flexibility.
- Availability of up to date information.
- Economic of scale.

Disadvantages of using DBMS:

- High initial investment in hardware, software, and training.
- The generality that a dbms provides for defining and processing data.
- Overhead for providing security, concurrency control, recovery, and integrity functions.

Database Basics:

Data item:

The data item is also called as field in data processing and is the smallest unit of data

That has meaning to its users.

E.g.: "e101", "sumit"

Entities and attributes:

An **entity** is a thing, object, place, person, or the activities in the real world that is distinguishable from all other Objects which an enterprise record data. A vendor has to maintain details of all items that he sells that could include name, price, and description these are called **entity type** the value that they stored may be periodic or stationary respectively, these values are called **entity instance**.

E.g.: Bank, employee, student

Attributes type are the characteristic properties of an entity type and attributes instance is the property of entity instance.

E.g.: Empcode, ename, rolno, name

Tables:

The data in RDBMS is stored in database objects called **tables**. The table is a collection of related data entries and it consists of columns and rows.

Remember, a table is the most common and simplest form of data storage in a relational database.

Field:

Every table is broken up into smaller entities called fields. For example The fields in the CUSTOMERS table consist of ID, NAME, AGE, ADDRESS and SALARY.

Record or Row:

A record, also called a row of data, is each individual entry that exists in a table.

Column:

A column is a vertical entity in a table that contains all information associated with a specific field in a table.

Null value:

A NULL value in a table is a value in a field that appears to be blank, which means a field with a NULL value is a field with no value.

It is very important to understand that a NULL value is different than a zero value or a field that contains spaces. A field with a NULL value is one that has been left blank during record creation.

DBMS languages and commands:

SQL commands are instructions used to communicate with the database to perform specific task that work with data. SQL commands can be used not only for searching the database but also to perform various other functions like, for example, you can create tables, add data to tables, or modify data, drop the table, set permissions for users. SQL commands are grouped into four major categories depending on their functionality:

- Data Definition Language (DDL):

These SQL commands are used for creating, modifying, and dropping the structure of database objects. The commands are CREATE, ALTER, DROP, RENAME, and TRUNCATE.

- Data Manipulation Language (DML):

These SQL commands are used for storing, retrieving, modifying and deleting data. These commands are SELECT, INSERT, UPDATE, and DELETE.

- Transaction Control Language (TCL) :

These SQL commands are used for managing changes affecting the data. These commands are COMMIT, ROLLBACK, and SAVEPOINT.

- Data Control Language (DCL):

These SQL commands are used for providing security to database objects. These commands are GRANT and REVOKE.

Data definition language:

It is a conceptual schema that provides a link between logical and physical structure of data base. DDL describes the physical characteristic of data. It provides logical data independence and physical data independence.

Commands are: Create, Alter, Drop, and Truncate.

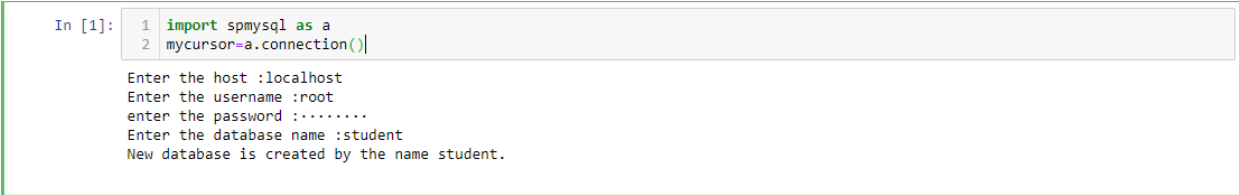
To connect to mysql:

Before executing any query or command first we need to connect with mysql.

Using **spmysql** library:

```
import spmysql as a  
mycursor=a.connection()
```

These 2 lines of code will host, username, password, and database as the input parameter during its execution time.



```
In [1]: 1 import spmysql as a  
        2 mycursor=a.connection()  
  
Enter the host :localhost  
Enter the username :root  
Enter the password :.....  
Enter the database name :student  
New database is created by the name student.
```

fig.1

Using **pymysql** library:

```
import pymysql as m  
mydb=m.connect(host='localhost',user='root',password='your password')
```

These 2 lines of code will host, username, password as the input parameter before its execution time.

```
In [7]: 1 import pymysql as m
        2 mydb=m.connect(host ='localhost',user ='root' ,password='')
```

fig.2

Database:

To create database:

Using **spmysql** library:

```
import smysql as a
mycursor=a.connection()
```

These 2 lines of code will host, username, password, and database as the input parameter during its execution time. If the database is exist then it will use that database, if the database is not existed it will create a new database.

```
In [1]: 1 import smysql as a
        2 mycursor=a.connection()

Enter the host :localhost
Enter the username :root
enter the password :.....
Enter the database name :student
New database is created by the name student.
```

fig.3

Using **pymysql** library:

```
import pymysql as m
mydb=m.connect(host ='localhost',user ='root' ,password='')
mycursor = mydb.cursor()
mycursor.execute("create database student;")
```

These lines of code will host, username, password as the input parameter before its execution time and create a database with a name student.

```
In [7]: 1 import pymysql as m
        2 mydb=m.connect(host ='localhost',user ='root' ,password='')
        3 mycursor = mydb.cursor()
        4 mycursor.execute("create database student;")
```

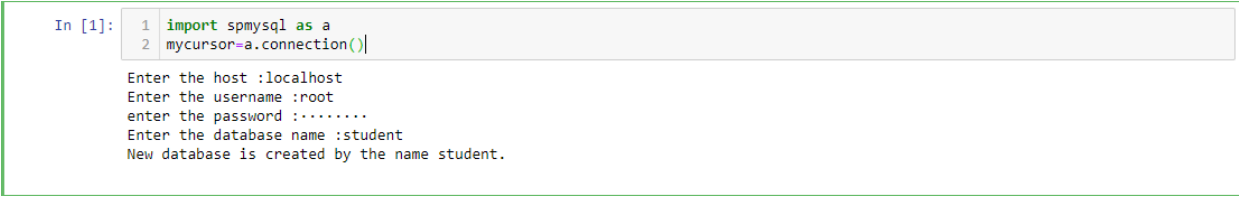
fig.4

To Use Database:

Using **spmysql** library:

```
import spmysql as a
mycursor=a.connection()
```

These 2 lines of code will host, username, password, and database as the input parameter during its execution time. If the database is exist then it will use that database, if the database is not existed it will create a new database and also use the same database.



```
In [1]: 1 import spmysql as a
        2 mycursor=a.connection()

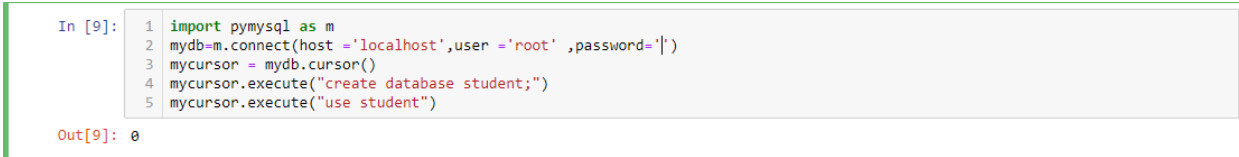
Enter the host :localhost
Enter the username :root
enter the password :.....
Enter the database name :student
New database is created by the name student.
```

fig.5

Using **pymysql** library:

```
import pymysql as m
mydb=m.connect(host='localhost',user='root',password='')
mycursor = mydb.cursor()
mycursor.execute("create database student;")
mycursor.execute("Use student ;")
```

These lines of code will host, username, password as the input parameter before its execution time and create a database with a name student and uses student database for further work.



```
In [9]: 1 import pymysql as m
        2 mydb=m.connect(host='localhost',user='root',password='')
        3 mycursor = mydb.cursor()
        4 mycursor.execute("create database student;")
        5 mycursor.execute("use student")

Out[9]: 0
```

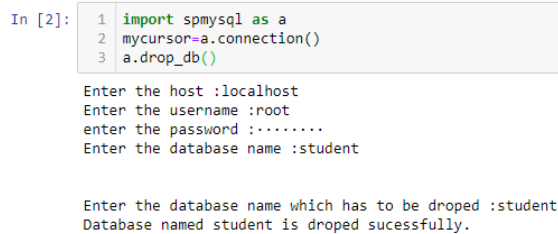
fig.6

To Drop a Database:

Using **spmysql** library:

```
import spmysql as a
mycursor=a.connection()
a.drop_db()
```

These lines of code will host, username, password, and database as the input parameter during its execution time. If the database is exist then it will use that database, if the database is not existed it will create a new database and also use the same database and to drop a database.



```
In [2]: 1 import spmysql as a
        2 mycursor=a.connection()
        3 a.drop_db()

Enter the host :localhost
Enter the username :root
enter the password :.....
Enter the database name :student

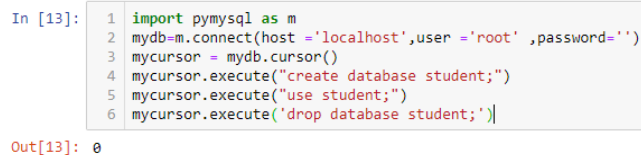
Enter the database name which has to be dropped :student
Database named student is dropped sucessfully.
```

fig.7

Using **pymysql** library:

```
import pymysql as m
mydb=m.connect(host ='localhost',user ='root' ,password='')
mycursor = mydb.cursor()
mycursor.execute("create database student;")
mycursor.execute("use student;")
mycursor.execute('drop database student;')
```

These lines of code will host, username, password as the input parameter before its execution time and create a database with a name student and uses student database for further work also it will drop the student database.



```
In [13]: 1 import pymysql as m
        2 mydb=m.connect(host ='localhost',user ='root' ,password='')
        3 mycursor = mydb.cursor()
        4 mycursor.execute("create database student;")
        5 mycursor.execute("use student;")
        6 mycursor.execute('drop database student;')

Out[13]: 0
```

fig.8

Table:

To Create a Table:

General Sql syntax: create table < table_name>
(*column_name1* datatype (size), *column_name2* datatype
(size),....., *column_namen* datatype (size));

Using **spmysql** library:

```
import  spmysql  as  a
mycursor=a.connection( )
a.cre_table( )
```

These lines will take the required credentials dynamically through its execution time.

```
In [7]: 1 import spmysql as a
        2 mycursor=a.connection()
        3 a.cre_table()

Enter the host :localhost
Enter the username :root
enter the password :.....
Enter the database name :student

Enter the table name that has to be created :STUDENTINFO
Enter the number of columns :5
Enter the details of the column :stno int
Enter the details of the column :stname varchar(25)
Enter the details of the column :course varchar(25)
Enter the details of the column :addate date
Enter the details of the column :fees int
New table is created with the name STUDENTINFO
```

fig.9

Using **pymysql** library:

```
import pymysql as m
mydb=m.connect(host = 'localhost',user = 'root' ,password='')
mycursor = mydb.cursor()
mycursor.execute("create          database          student;")
mycursor.execute("use student;")
mycursor.execute('create table STUDENTINFO(stno int,stname varchar(25),course varchar(15),addate
date,fees int);')
```

These set of lines will create the table with the name STUDENTINFO by using student database.

```
In [16]: 1 import pymysql as m
2 mydb=m.connect(host='localhost',user='root',password='')
3 mycursor = mydb.cursor()
4 mycursor.execute("create database student;")
5 mycursor.execute("use student;")
6 mycursor.execute('create table STUDENTINFO(stno int,stname varchar(25),course varchar(15),addate date,fees int);')|

Out[16]: 0
```

fig.10

Alter:

To increase or decrease the width:

Using **pymysql** library:

```
import pymysql as m
mydb=m.connect(host='localhost',user='root',password='')
mycursor = mydb.cursor()
mycursor.execute("use student;")
mycursor.execute('alter table studentinfo modify stname varchar(21);')
```

These set of lines will use student database and alter the content of the STUDENTINFO table by decreasing the width of the stname from 25 to 21.

```
In [19]: 1 import pymysql as m
2 mydb=m.connect(host='localhost',user='root',password='')
3 mycursor = mydb.cursor()
4 mycursor.execute("use student;")
5 mycursor.execute('alter table studentinfo modify stname varchar(21);')|

Out[19]: 0
```

fig.11

To add a new column:

Using **pymysql** library:

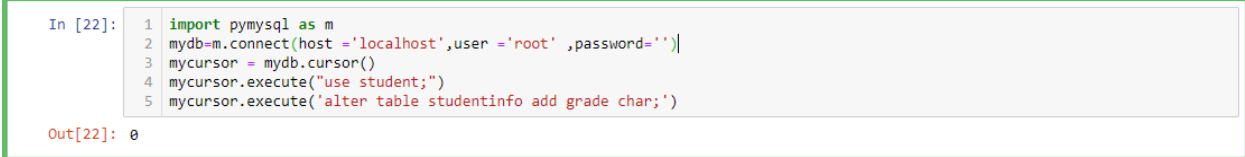
```
import pymysql as m
```

```

mydb=m.connect(host='localhost',user='root',password='')
mycursor = mydb.cursor()
mycursor.execute("use student;")
mycursor.execute('alter table studentinfo add grade char;')

```

These set of lines will add new column that is grade of type character to the STUDENTINFO table.



```

In [22]: 1 import pymysql as m
          2 mydb=m.connect(host='localhost',user='root',password='')
          3 mycursor = mydb.cursor()
          4 mycursor.execute("use student;")
          5 mycursor.execute('alter table studentinfo add grade char;')

Out[22]: 0

```

fig.12

To change the data type:


Using **pymysql** library:

```

import pymysql as m
mydb=m.connect(host='localhost',user='root',password='')
mycursor = mydb.cursor()
mycursor.execute("use student;")
mycursor.execute('alter table studentinfo modify grade varchar(2);')

```

Here the data type of grade is changed from char to varchar of size 2.



```

In [23]: 1 import pymysql as m
          2 mydb=m.connect(host='localhost',user='root',password='')
          3 mycursor = mydb.cursor()
          4 mycursor.execute("use student;")
          5 mycursor.execute('alter table studentinfo modify grade varchar(2);')

Out[23]: 0

```


fig.13

To remove a column from table:

Using **pymysql** library:

```
import pymysql as m
mydb=m.connect(host='localhost',user='root',password='')
mycursor = mydb.cursor()
mycursor.execute("use student;")
mycursor.execute('alter table studentinfo drop grade;')
```

Here the grade column is dropped from the STUDENYINFO table.



```
In [24]: 1 import pymysql as m
2 mydb=m.connect(host='localhost',user='root',password='')
3 mycursor = mydb.cursor()
4 mycursor.execute("use student;")
5 mycursor.execute('alter table studentinfo drop grade;')
```

Out[24]: 0

fig.14

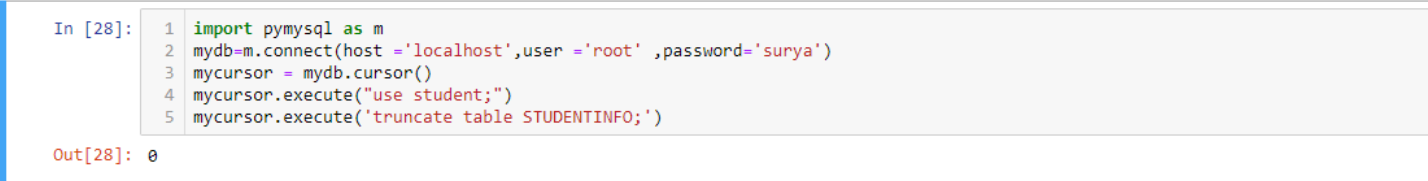
To truncate the table:

Using **pymysql** library:

Method1:

```
import pymysql as m
mydb=m.connect(host='localhost',user='root',password='')
mycursor = mydb.cursor()
mycursor.execute("use student;")
mycursor.execute('truncate table STUDENTINFO;')
```

Here the STUDENTINFO table is truncated.



```
In [28]: 1 import pymysql as m
2 mydb=m.connect(host='localhost',user='root',password='surya')
3 mycursor = mydb.cursor()
4 mycursor.execute("use student;")
5 mycursor.execute('truncate table STUDENTINFO;')
```

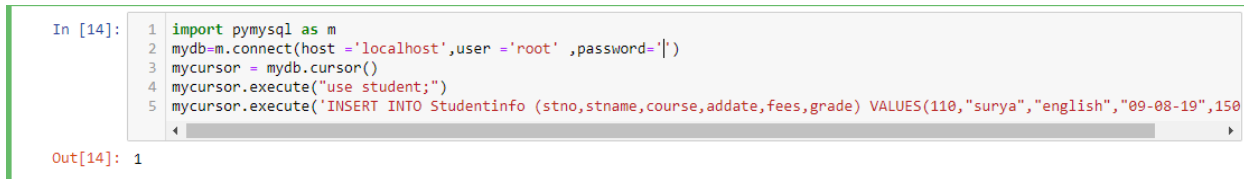
Out[28]: 0

Method2:

```
import pymysql as m
mydb=m.connect(host='localhost',user='root',password='')
mycursor = mydb.cursor()
mycursor.execute("use student;")
```

```
mycursor.execute('truncate table STUDENTINFO reuse storage;')
```

Here the table is truncated and the memory is reused for some other purpose.



```
In [14]: 1 import pymysql as m
2 mydb=m.connect(host='localhost',user='root',password='')
3 mycursor = mydb.cursor()
4 mycursor.execute("use student;")
5 mycursor.execute('INSERT INTO Studentinfo (stno,stname,course,addate,fees,grade) VALUES(110,"surya","english","09-08-19",150

Out[14]: 1
```

fig.16

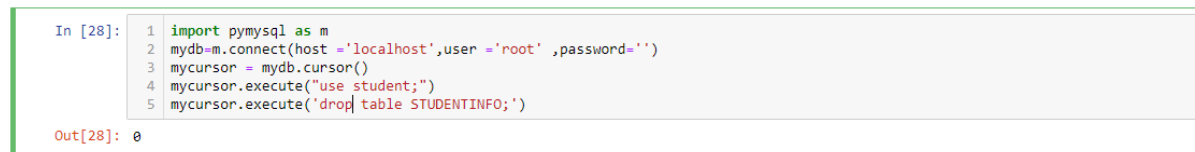
To drop a table:

General syntax: drop table <table_name>;

Using **pymysql** library:

```
import pymysql as m
mydb=m.connect(host='localhost',user='root',password='')
mycursor = mydb.cursor()
mycursor.execute("use student;")
mycursor.execute('drop table STUDENTINFO ;')
```

These set of lines of code is used to dropping a table.



```
In [28]: 1 import pymysql as m
2 mydb=m.connect(host='localhost',user='root',password='')
3 mycursor = mydb.cursor()
4 mycursor.execute("use student;")
5 mycursor.execute('drop table STUDENTINFO;')

Out[28]: 0
```

Data manipulating language:

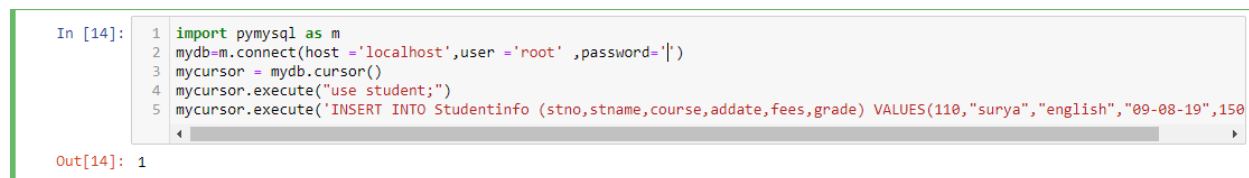
Data manipulating language (DML) statements access and manipulate data stored in the oracle database. You can use them to insert, update, delete, and read data. Control statements are related, since they are used to control how oracle operates when accessing data in the database. There are not as many DML and control statements as there are DDL statements, but many do have a wide range of options and syntax choices.

To insert a Value into the table:

Using **pymysql** library:

```
import pymysql as m
mydb=m.connect(host='localhost',user='root',password='')
mycursor = mydb.cursor()
mycursor.execute("use student;")
mycursor.execute('INSERT INTO Studentinfo (stno,stname, course ,addate, fees, grade) VALUES(110,"surya","english","09-08-19",15008-19",150,"a");')
```

This will insert the values into the table.

The image shows a Jupyter Notebook interface. The input cell (In [14]:) contains five lines of Python code: 1. import pymysql as m, 2. mydb=m.connect(host='localhost',user='root',password=''), 3. mycursor = mydb.cursor(), 4. mycursor.execute("use student;"), and 5. mycursor.execute('INSERT INTO Studentinfo (stno,stname, course ,addate, fees, grade) VALUES(110,"surya","english","09-08-19",15008-19",150,"a");'). The output cell (Out[14]:) shows the number 1, indicating successful execution.

```
In [14]: 1 import pymysql as m
2 mydb=m.connect(host='localhost',user='root',password='')
3 mycursor = mydb.cursor()
4 mycursor.execute("use student;")
5 mycursor.execute('INSERT INTO Studentinfo (stno,stname, course ,addate, fees, grade) VALUES(110,"surya","english","09-08-19",15008-19",150,"a");')

Out[14]: 1
```

fig.1

To insert NULL values into the table:

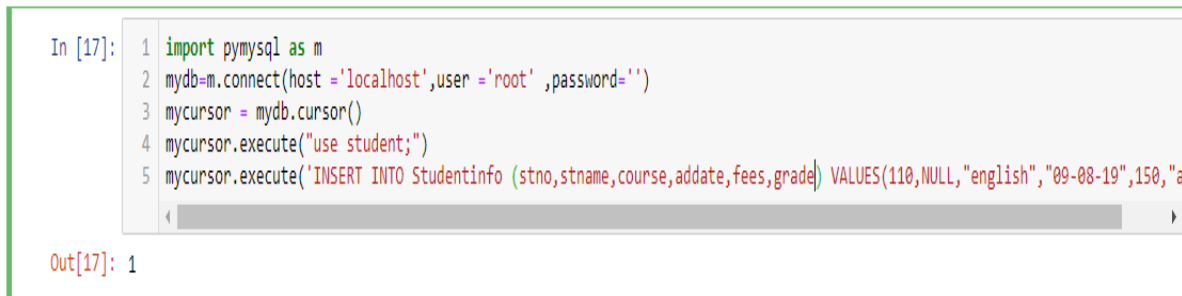
Using **pymysql** library:

```
import pymysql as m
```



```
mydb=m.connect(host='localhost',user='root',password='')
mycursor = mydb.cursor()
mycursor.execute("use student;")
mycursor.execute('INSERT INTO Studentinfo (stno,stname, course ,addate, fees, grade) VALUES(110,NULL,"english",
08-19",150,"a");')
```

This will insert null in place of name.



```
In [17]: 1 import pymysql as m
2 mydb=m.connect(host='localhost',user='root',password='')
3 mycursor = mydb.cursor()
4 mycursor.execute("use student;")
5 mycursor.execute('INSERT INTO Studentinfo (stno,stname, course ,addate, fees, grade) VALUES(110,NULL,"english",
08-19",150,"a");')
```


Out[17]: 1

To insert the limited values into the table:

Using **pymysql** library:

```
import pymysql as m
mydb=m.connect(host='localhost',user='root',password='')
mycursor = mydb.cursor()
mycursor.execute("use student;")
mycursor.execute('INSERT INTO Studentinfo (stno,stname, course ,addate, fees) VALUES(110,"miller","english","09-
08-19",150);')
```

This will insert the limited data which is available and all remaining columns will be filled with NULL values.



```
In [17]: 1 import pymysql as m
2 mydb=m.connect(host='localhost',user='root',password='')
3 mycursor = mydb.cursor()
4 mycursor.execute("use student;")
5 mycursor.execute('INSERT INTO Studentinfo (stno,stname, course ,addate, fees) VALUES(110,"miller","english","09-08-19",150);')
```

Out[17]: 1

fig.3

Select statements:

A select statement is used to fetch the data from database table which results the data in the form of result table. These result tables are called **result set**.

To list the all details from the table:

Using **pymysql** library:

```
import pymysql as m
mydb=m.connect(host='localhost',user='root',password='')
mycursor = mydb.cursor()
mycursor.execute("use student;")
mycursor.execute("select * from studentinfo;")
data=mycursor.fetchall()
for i in data:
    print(i)
```

These lines of code will fetch all the data from the table and each row of the table is stored in the form of tuple we can get each of them by tuple unpacking.

```
In [36]: 1 import pymysql as m
          2 mydb=m.connect(host='localhost',user='root',password='')
          3 mycursor = mydb.cursor()
          4 mycursor.execute("use student;")
          5 mycursor.execute("select * from studentinfo;")
          6 data=mycursor.fetchall()
          7 for i in data:
          8     print(i)
          9
```

```
(113, 'rajini', 'tamil', datetime.date(2028, 8, 19), 200, 'c')
(110, 'miller', 'english', datetime.date(2008, 8, 19), 180, 's')
(111, 'scoot', 'german', datetime.date(2004, 8, 19), 150, 'b')
(120, 'john', 'france', datetime.date(2006, 8, 19), 180, 'b')
(123, 'tiger', 'english', datetime.date(2007, 8, 19), 185, 'c')
(126, 'surya', 'english', datetime.date(2007, 8, 19), 185, 'c')
```

To list only one column from the table:

Using **pymysql** library:

```
import pymysql as m
mydb=m.connect(host='localhost',user='root',password='')
mycursor = mydb.cursor()
mycursor.execute("use student;")
mycursor.execute("select stname from studentinfo;")
data=mycursor.fetchall()
```

Data manipulating language:
for i in data:

print(i)

27

These lines of code will list only the names of the students from the student table.

```
In [38]: 1 import pymysql as m
          2 mydb=m.connect(host='localhost',user='root',password='')
          3 mycursor = mydb.cursor()
          4 mycursor.execute("use student;")
          5 mycursor.execute("select stname from studentinfo;")
          6 data=mycursor.fetchall()
          7 for i in data:
          8     print(i)
          9
          ('rajini',)
          ('miller',)
          ('scoot',)
          ('john',)
          ('tiger',)
          ('surya',)
```

fig.5

To list the columns with aliasing name from the table:

Using **pymysql** library:

```
import pymysql as m
mydb=m.connect(host='localhost',user='root',password='')
mycursor = mydb.cursor()
mycursor.execute("use student;")
mycursor.execute("select stname as name from studentinfo;")
data=mycursor.fetchall()
for i in data:
    print(i)
```

These lines of code will list only names of the students as names instead of stname from the studentinfo table.

```
In [42]: 1 import pymysql as m
          2 mydb=m.connect(host='localhost',user='root',password='surya')
          3 mycursor = mydb.cursor()
          4 mycursor.execute("use student;")
          5 mycursor.execute("select stname as name from studentinfo;")
          6 data=mycursor.fetchall()
          7 for i in data:
          8     print(i)
          9
          ('rajini',)
          ('miller',)
          ('scoot',)
          ('john',)
          ('tiger',)
          ('surya',)
```

fig.5

Where clause:

It is used to specify a condition while fetching a data via single table or joining with multiple tables if they give condition is satisfied then only it returns specific values from the table. It is used to filter the records and fetching only necessary records. Condition retiring of records is done by using where clause.

General syntax: select *columns* from *table_name* where condition;

Condition can have arithmetic operator, relational operation and logical operators.

Arithmetic operators are + , - , / , *

Relational operators are <.>, <=, >=, =, <>, !=

Logical operators are AND, OR, NOT

Using of simple where clause:

Using **pymysql** library:

```
import pymysql as m
mydb=m.connect(host='localhost',user='root',password='')
mycursor = mydb.cursor()
mycursor.execute("use student;")
mycursor.execute("select * from studentinfo where stname='scoot;")
data=mycursor.fetchall()
for i in data:
    print(i)
```

Here only the detail of scoot is fetched from the studentinfo table.

```

In [44]: 1 import pymysql as m
          2 mydb=m.connect(host='localhost',user='root',password='surya')
          3 mycursor = mydb.cursor()
          4 mycursor.execute("use student;")
          5 mycursor.execute("select * from studentinfo where stname='scoot';")
          6 data=mycursor.fetchall()
          7 for i in data:
          8     print(i)
          9
          (111, 'scoot', 'german', datetime.date(2004, 8, 19), 150, 'b')

```

Fig.7

Using arithmetic operator in where clause:

Using **pymysql** library:

```

import pymysql as m
mydb=m.connect(host='localhost',user='root',password='')
mycursor = mydb.cursor()
mycursor.execute("use student;")
mycursor.execute("select * from studentinfo where fees=150+30;")
data=mycursor.fetchall()
for i in data:
print(i)

```

Here only the detail of the students whose fees is equal to 180 is listed.

Using of relational operators in where clause:

Using **pymysql** library:

```

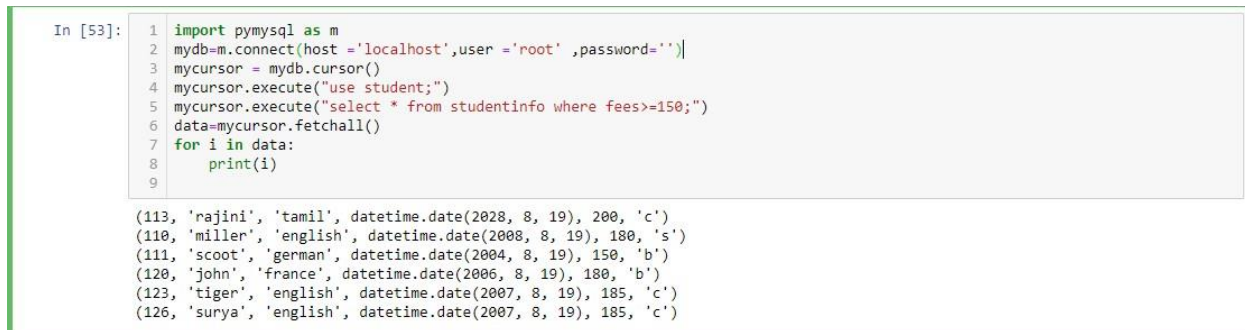
import pymysql as m
mydb=m.connect(host='localhost',user='root',password='')
mycursor = mydb.cursor()
mycursor.execute("use student;")
mycursor.execute("select * from studentinfo where fees>=150;")
data=mycursor.fetchall()

```

for i in data:

print(i)

These lines of code will list all the student details whose fees are greater than or equal to 150.



```
In [53]: 1 import pymysql as m
2 mydb=m.connect(host='localhost',user='root',password='')
3 mycursor = mydb.cursor()
4 mycursor.execute("use student;")
5 mycursor.execute("select * from studentinfo where fees>=150;")
6 data=mycursor.fetchall()
7 for i in data:
8     print(i)
9
(113, 'rajini', 'tamil', datetime.date(2028, 8, 19), 200, 'c')
(110, 'miller', 'english', datetime.date(2008, 8, 19), 180, 's')
(111, 'scoot', 'german', datetime.date(2004, 8, 19), 150, 'b')
(120, 'john', 'france', datetime.date(2006, 8, 19), 180, 'b')
(123, 'tiger', 'english', datetime.date(2007, 8, 19), 185, 'c')
(126, 'surya', 'english', datetime.date(2007, 8, 19), 185, 'c')
```

fig.7

Using logical operators in where clause:

Using **pymysql** library:

```
import pymysql as m
```

```
mydb=m.connect(host='localhost',user='root',password='')
```

```
mycursor = mydb.cursor()
```

```
mycursor.execute("use student;")
```

```
mycursor.execute("select * from studentinfo where stname='miller' or stname='scoot';")
```

```
data=mycursor.fetchall()
```

```
for i in data:
```

```
print(i)
```

These lines of code will list only the details of miller and scoot.

```

In [54]: 1 import pymysql as m
          2 mydb=m.connect(host='localhost',user='root',password='')
          3 mycursor = mydb.cursor()
          4 mycursor.execute("use student;")
          5 mycursor.execute("select * from studentinfo where stname='miller' or stname='scoot';")
          6 data=mycursor.fetchall()
          7 for i in data:
          8     print(i)
          9
          (110, 'miller', 'english', datetime.date(2008, 8, 19), 180, 's')
          (111, 'scoot', 'german', datetime.date(2004, 8, 19), 150, 'b')

```

fig.8

Using of between in where clause:

Using **pymysql** library:

```

import pymysql as m
mydb=m.connect(host='localhost',user='root',password='')
mycursor = mydb.cursor()
mycursor.execute("use student;")
mycursor.execute("select * from studentinfo where fees between 150 and 190;")
data=mycursor.fetchall()
for i in data:
    print(i)

```

This will test inclusion in range of values.

To negate the values in the select statement:

Using **pymysql** library:

```

import pymysql as m
mydb=m.connect(host='localhost',user='root',password='')
mycursor = mydb.cursor()
mycursor.execute("use student;")
mycursor.execute("select * from studentinfo where NOT(fees>=150 and fees<=190);")
data=mycursor.fetchall()

```


for i in data:

print(i)

This will filter the values by negating the fees in between 150 and 190.

```
In [61]: 1 import pymysql as m
          2 mydb=m.connect(host='localhost',user='root',password='')
          3 mycursor = mydb.cursor()
          4 mycursor.execute("use student;")
          5 mycursor.execute("select *| from studentinfo where NOT(fees>=150 and fees<=190);")
          6 data=mycursor.fetchall()
          7 for i in data:
          8     print(i)
          9
          (113, 'rajini', 'tamil', datetime.date(2028, 8, 19), 200, 'c')
```

fig.10

Distinct clause:

Distinct is group function to prevent the selection of duplicate load we include distinct clause in the select command.

Using distinct clause in select statement:

Using **pymysql** library:

```
import pymysql as m
mydb=m.connect(host='localhost',user='root',password='')
mycursor = mydb.cursor()
mycursor.execute("use student;")
mycursor.execute("select distinct fees from studentinfo;")
data=mycursor.fetchall()
for i in data:
print(i)
```

These lines of code will give the distinct fees of students from student table.

Order by clause:

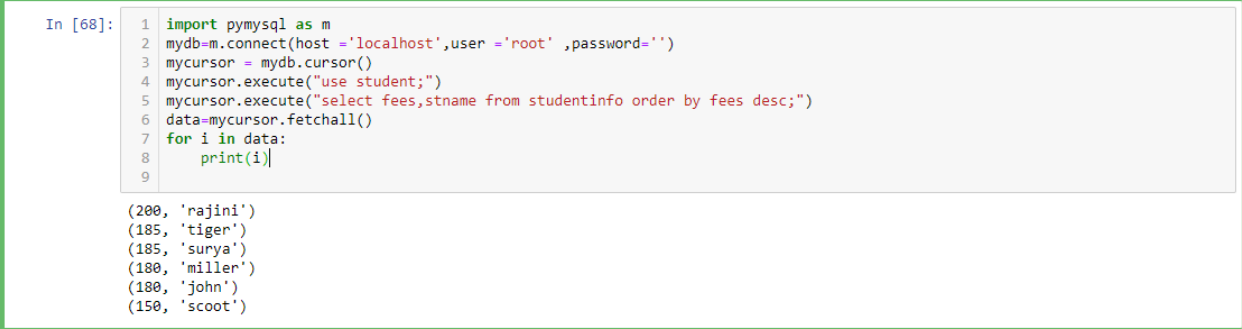
It is used to arrange the records either in ascending or descending order by default it arranges the records in ascending order to arrange in descending order the key word desc should be used.

Using of order by clause in select statement:

Using **pymysql** library:

```
import pymysql as m
mydb=m.connect(host='localhost',user='root',password='')
mycursor = mydb.cursor()
mycursor.execute("use student;")
mycursor.execute("select fees, stname from studentinfo order by fees desc;")
data=mycursor.fetchall()
for i in data:
    print(i)
```

This will arrange the records in descending order.



```
In [68]: 1 import pymysql as m
2 mydb=m.connect(host='localhost',user='root',password='')
3 mycursor = mydb.cursor()
4 mycursor.execute("use student;")
5 mycursor.execute("select fees,stname from studentinfo order by fees desc;")
6 data=mycursor.fetchall()
7 for i in data:
8     print(i)
9
```

(200, 'rajini')
(185, 'tiger')
(185, 'surya')
(180, 'miller')
(180, 'john')
(150, 'scoot')

fig.12

Update:

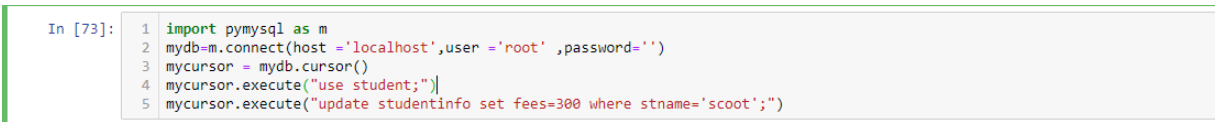
The UPDATE statement modifies data already in a database table or view. The UPDATE statement always has an associated target, and usually has a condition as well. If no condition is specified, all rows of the target table are UPDATED.

Using update statement:

Using **pymysql** library:

```
import pymysql as m
mydb=m.connect(host='localhost',user='root',password='')
mycursor = mydb.cursor()
mycursor.execute("use student;")
mycursor.execute("update studentinfo set fees=300 where stname='scoot';")
```

This will update the fees of the scoot to 300.

A screenshot of a Jupyter Notebook cell. The prompt 'In [73]:' is on the left. The code is as follows:

```
1 import pymysql as m
2 mydb=m.connect(host='localhost',user='root',password='')
3 mycursor = mydb.cursor()
4 mycursor.execute("use student;")
5 mycursor.execute("update studentinfo set fees=300 where stname='scoot';")
```

fig.13

Delete:

The DELETE statement removes rows from a database table or view. The statement will always have an associated target, and usually has a condition as well. If no condition is specified, all rows of the target table or view are removed.

Using DELETE statement:

Using **pymysql** library:

```
import pymysql as m
mydb=m.connect(host='localhost',user='root',password='')
mycursor = mydb.cursor()
mycursor.execute("use student;")
mycursor.execute("delete from studentinfo where stname='scoot';")
```

This will delete the details of the scoot from the studentinfo table.

```
In [75]: 1 import pymysql as m
          2 mydb=m.connect(host='localhost',user='root',password='')
          3 mycursor = mydb.cursor()
          4 mycursor.execute("use student;")
          5 mycursor.execute("delete from studentinfo where stname='scoot';")

Out[75]: 1
```

fig.14

Transaction control language:

Transaction Control language (TCL) commands is used to manage transactions in the database. These are used to manage the changes made to the data in a table by DML statements. It also allows statements to be grouped together into logical transactions.

Commit:

This command is used to end a transaction. Only with the help of commit command, transaction changes can be made permanent to the database, this command also erases all savepoints in the transactions thus releasing the transactions lock.

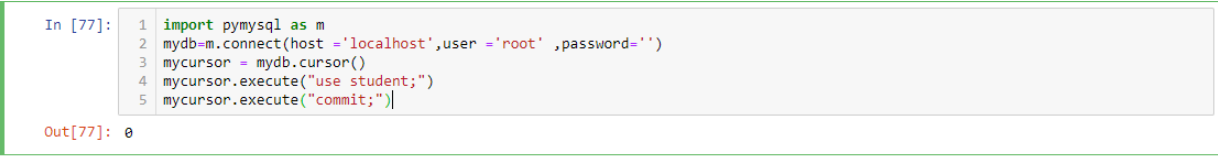
When we use any DML command like INSERT, UPDATE or DELETE, the changes made by these commands are not permanent, until the current session is closed, the changes made by these commands can be rolled back. To avoid that, we use the COMMIT command to mark the changes as permanent.

General syntax: commit;

Using pymysqllibrary:

```
import pymysql as m
mydb=m.connect(host='localhost',user='root',password='')
mycursor = mydb.cursor()
mycursor.execute("use student;")
mycursor.execute("commit;")
```

These lines of code will save all the transactions which has done.



```
In [77]: 1 import pymysql as m
2 mydb=m.connect(host='localhost',user='root',password='')
3 mycursor = mydb.cursor()
4 mycursor.execute("use student;")
5 mycursor.execute("commit;")

Out[77]: 0
```

fig.1

Save point:

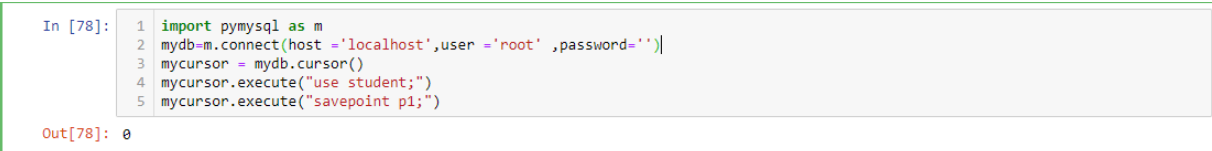
Save point are like markers to divide a very lengthy transaction to smaller ones. Transaction to which we can later rollback. Thus savepoints is used in conjunction with rollback. To rollback portion of the current transaction.

General syntax: `savepoint <any_variable>;`

Using **pymysql** library:

```
import pymysql as m
mydb=m.connect(host='localhost',user='root',password='')
mycursor = mydb.cursor()
mycursor.execute("use student;")
mycursor.execute("savepoint p1;")
```

These lines will create a savepoint p1



```
In [78]: 1 import pymysql as m
2 mydb=m.connect(host='localhost',user='root',password='')
3 mycursor = mydb.cursor()
4 mycursor.execute("use student;")
5 mycursor.execute("savepoint p1;")

Out[78]: 0
```

fig.2

Rollback:

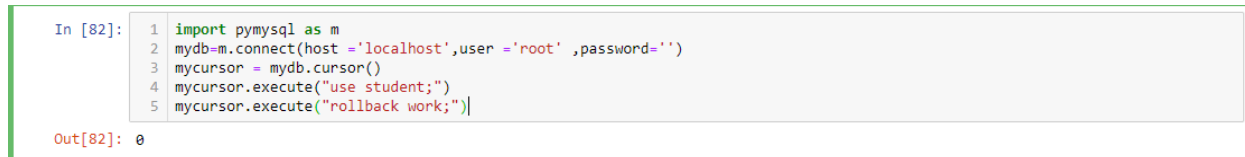
A rollback command is used to undo the work done in the current transactions we can either rollback the entire transaction so that entire transaction so that all changes made by sql transaction to a savepoint so that the sql transaction to a savepoint so that the sql statement after the savepoint are rolled back.

General syntax: `rollback;` or `rollback work;`

```
import pymysql as m
mydb=m.connect(host='localhost',user='root',password='')
mycursor = mydb.cursor()
```

```
mycursor.execute("use student;")
mycursor.execute("rollback work;")
```

These lines will roll back all the transactions.



```
In [82]: 1 import pymysql as m
          2 mydb=m.connect(host='localhost',user='root',password='')
          3 mycursor = mydb.cursor()
          4 mycursor.execute("use student;")
          5 mycursor.execute("rollback work;")

Out[82]: 0
```

fig.3

Data control language (DCL):

Data control in sql provides sql-DCL commands/statements to control the user access to the database (tables). It's concerned with granting privileges/access rights for the authorized users of the database. The GRANT command gives the privileges or access to the users on the database objects. On the other hand, REVOKE command is used for removing the rights or privileges from the users on the database objects.

Grant:

Grant command is used for giving the privileges to the users. The database administrator defines the **GRANT** command in SQL for giving the access or privileges to the users of the database. Three major components which are involved in the authorization are the users, privilege/s (operations) and a database object. The **** user **** is the one who triggers the execution of the application program. **** Operations **** are the component which is embedded in an application program. The **operations** are performed on database objects such as relation or view name.

General syntax: grant<privilege record>on <relation title or view title>to <user/role record>;

Revoke:

The **REVOKE** command in SQL is defined to take away the granted privileges (authorizations) from the user of the database. The one who

has the authority to withdraw the privileges is the database administrator. The command is similar to grant command except for the revoke keyword and 'from'. In given command, the operations included in the privilege are cancelled for the particular user or role list.

Revoking becomes complex when privileges are propagated from one user to other.

General syntax: revoke <privilege list>on <relation name or view name>from <user/role list>;

Special functions and operators of sql:

IN operator:

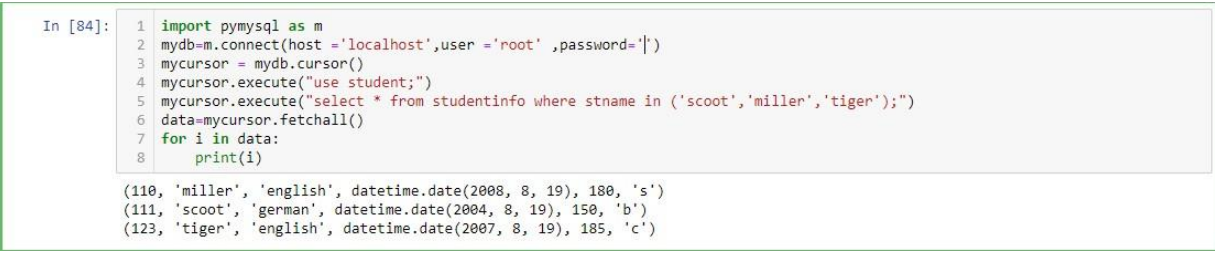
We can be used to select the rows that match one of the values in a list.

Using of IN operator:

Using `pymysql` library:

```
import pymysql as m
mydb=m.connect(host='localhost',user='root',password='')
mycursor = mydb.cursor()
mycursor.execute("use student;")
mycursor.execute("select * from studentinfo where stname in ('scoot', 'miller', 'tiger');")
data=mycursor.fetchall()
for i in data:
    print(i)
```

These lines will print only the details of scoot, miller and tiger.



```
In [84]: 1 import pymysql as m
2 mydb=m.connect(host='localhost',user='root',password='')
3 mycursor = mydb.cursor()
4 mycursor.execute("use student;")
5 mycursor.execute("select * from studentinfo where stname in ('scoot', 'miller', 'tiger');")
6 data=mycursor.fetchall()
7 for i in data:
8     print(i)

(110, 'miller', 'english', datetime.date(2008, 8, 19), 180, 's')
(111, 'scoot', 'german', datetime.date(2004, 8, 19), 150, 'b')
(123, 'tiger', 'english', datetime.date(2007, 8, 19), 185, 'c')
```

fig.1

Like operator:

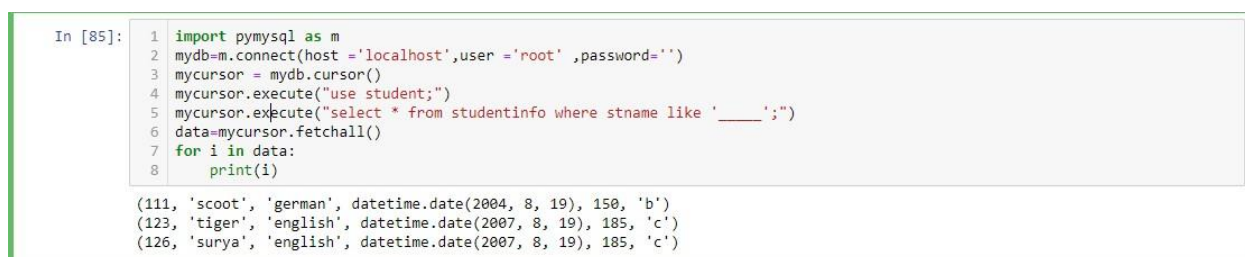
Use to search a character pattern we need not know the exact character value it recognise the special characters like person m_ the format can matches % can be used to match null character .

Using of LIKE operator:

Using **pymysql** library:

```
import pymysql as m
mydb=m.connect(host='localhost',user='root',password='')
mycursor = mydb.cursor()
mycursor.execute("use student;")
mycursor.execute("select * from studentinfo where stname like '____';")
data=mycursor.fetchall()
for i in data:
    print(i)
```

These lines will filter the info of the students whose name is having only 5 letters.



```
In [85]: 1 import pymysql as m
2 mydb=m.connect(host='localhost',user='root',password='')
3 mycursor = mydb.cursor()
4 mycursor.execute("use student;")
5 mycursor.execute("select * from studentinfo where stname like '____';")
6 data=mycursor.fetchall()
7 for i in data:
8     print(i)

(111, 'scoot', 'german', datetime.date(2004, 8, 19), 150, 'b')
(123, 'tiger', 'english', datetime.date(2007, 8, 19), 185, 'c')
(126, 'surya', 'english', datetime.date(2007, 8, 19), 185, 'c')
```

fig.2

Round function:

Date is rounded to the unit specified by the format model.

Round (date, [format])

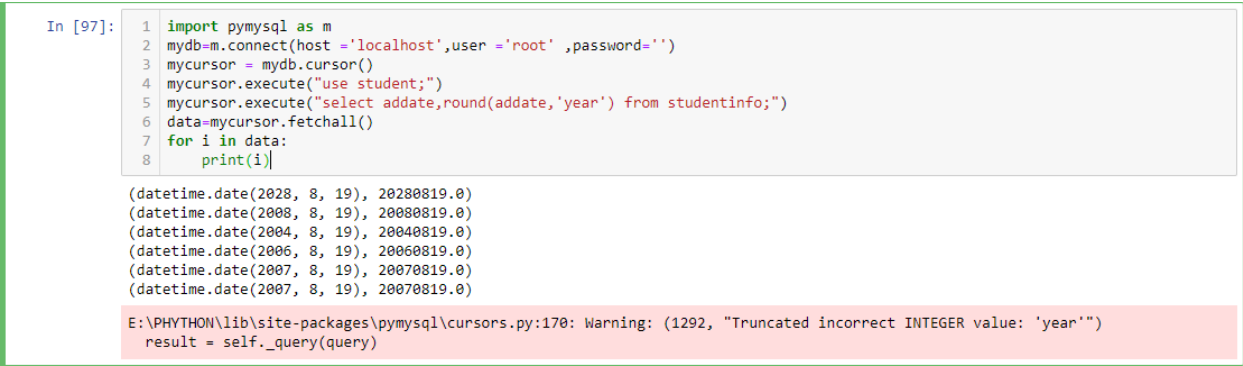
Format can be year, month or day. If year is specified in format it refers to the month. If the month is greater than 6 then the date is rounded to 1-jan-currentyear. If the month specified in format it refers to days. If the days are greater than the date is rounded to 1-nextmonth-currentyear. If days are less than or equal to 15 then the date is rounded to 1-currentmonth-currentyear. If the date is specified in the format then the date is rounded to the nearest Sunday.

Using of round function:

Using **pymysql** library:

```
import pymysql as m
mydb=m.connect(host='localhost',user='root',password='')
mycursor = mydb.cursor()
mycursor.execute("use student;")
mycursor.execute("select adddate, round(adddate,'year') from studentinfo;")
data=mycursor.fetchall()
for i in data:
    print(i)
```

This will round the admission date of the students.



```
In [97]: 1 import pymysql as m
2 mydb=m.connect(host='localhost',user='root',password='')
3 mycursor = mydb.cursor()
4 mycursor.execute("use student;")
5 mycursor.execute("select adddate,round(adddate,'year') from studentinfo;")
6 data=mycursor.fetchall()
7 for i in data:
8     print(i)

(datetime.date(2028, 8, 19), 20280819.0)
(datetime.date(2008, 8, 19), 20080819.0)
(datetime.date(2004, 8, 19), 20040819.0)
(datetime.date(2006, 8, 19), 20060819.0)
(datetime.date(2007, 8, 19), 20070819.0)
(datetime.date(2007, 8, 19), 20070819.0)

E:\PYTHON\lib\site-packages\pymysql\cursors.py:170: Warning: (1292, "Truncated incorrect INTEGER value: 'year'")
  result = self._query(query)
```

fig.3

Dual:

It is a system table automatically created by the oracle along with the dictionary it has one column defined to be varchar datatype and contains

only one row of value 'X'.

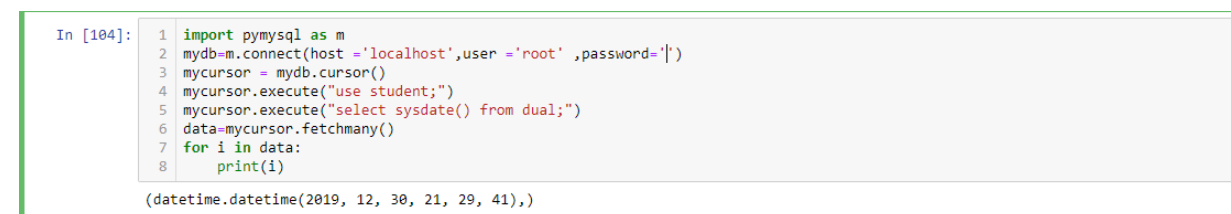
System date function:

The PLSQL SYSDATE function will returns current system date and time on your database. There is no any parameter or argument for the SYSDATE function. The SYSDATE function returns a date value. Note that the SYSDATE function return date and time as "YYYY-MM-DD HH:MM:SS" (string) or as YYYYMMDDHHMMSS (numeric).

Using of sysdate () function:

Using **pymysql** library:

```
import pymysql as m
mydb=m.connect(host = 'localhost',user = 'root' ,password='')
mycursor = mydb.cursor()
mycursor.execute("use student;")
mycursor.execute("select sysdate() from dual;")
data=mycursor.fetchall()
for i in data:
    print(i)
```



```
In [104]: 1 import pymysql as m
2 mydb=m.connect(host = 'localhost',user = 'root' ,password='')
3 mycursor = mydb.cursor()
4 mycursor.execute("use student;")
5 mycursor.execute("select sysdate() from dual;")
6 data=mycursor.fetchall()
7 for i in data:
8     print(i)

(datetime.datetime(2019, 12, 30, 21, 29, 41),)
```

fig.4

System_user ():


The SYSTEM_USER function returns the login name for the current user.

Using of System_user ():

Using **pymysql** library:

```
import pymysql as m
mydb=m.connect(host='localhost',user='root',password='')
mycursor = mydb.cursor()
mycursor.execute("use student;")
mycursor.execute("select System_user() from dual;")
data=mycursor.fetchall()
for i in data:
print(i)
```

This will return the current user.



```
In [108]: 1 import pymysql as m
2 mydb=m.connect(host='localhost',user='root',password='')
3 mycursor = mydb.cursor()
4 mycursor.execute("use student;")
5 mycursor.execute("select system_user() from dual;")
6 data=mycursor.fetchall()
7 for i in data:
8     print(i)

('root@localhost',)
```

fig.5

Greatest ():

The **GREATEST** is an inbuilt function in sql which is used to return the greatest value from a given list of some expressions. These expressions may be numbers, alphabets etc.

Using of greatest ():

Using **pymysql** library:

```
import pymysql as m
mydb=m.connect(host='localhost',user='root',password='')
```

```

mycursor = mydb.cursor()
mycursor.execute("use student;")
mycursor.execute("select greatest(1,2,3,4) from dual;")
data=mycursor.fetchall()
for i in data:
print(i)

```

This will return the greatest among the values.



```

In [110]: 1 import pymysql as m
          2 mydb=m.connect(host ='localhost',user ='root' ,password='')
          3 mycursor = mydb.cursor()
          4 mycursor.execute("use student;")
          5 mycursor.execute("select greatest(1,2,3,4) from dual;")
          6 data=mycursor.fetchall()
          7 for i in data:
          8     print(i)

(4,)

```

fig.6

Numeric functions:

It accepts numeric input and returns numeric value as output.

Using of abs ():

From here after all the code will use **pymysql** library:

```

import pymysql as m
mydb=m.connect(host ='localhost',user ='root' ,password='')
mycursor = mydb.cursor()
mycursor.execute("use student;")
mycursor.execute("select abs(-234) from dual;")
data=mycursor.fetchall()
for i in data:
for a in i:
print(a)

```

Here the absolute value of -234 is the output.

```
In [1]: 1 import pymysql as m
2 mydb=m.connect(host ='localhost',user ='root' ,password='')
3 mycursor = mydb.cursor()
4 mycursor.execute("use student;")
5 mycursor.execute("select abs(-234) from dual;")
6 data=mycursor.fetchall()
7 for i in data:
8     for a in i:
9         print(a)|
```

234

fig.7

Using of ceil ():

```
import pymysql as m
mydb=m.connect(host ='localhost',user ='root' ,password='')
mycursor = mydb.cursor()
mycursor.execute("use student;")
mycursor.execute("select ceil(4.765) from dual;")
data=mycursor.fetchall()
for i in data:
    for a in i:
        print(a)
```

The value is rounded to nearest higher value in this case it is 5.

```
In [3]: 1 import pymysql as m
2 mydb=m.connect(host ='localhost',user ='root' ,password='')
3 mycursor = mydb.cursor()
4 mycursor.execute("use student;")
5 mycursor.execute("select ceil(4.765) from dual;")
6 data=mycursor.fetchall()
7 for i in data:
8     for a in i:
9         print(a)
```

5

fig.8

Using of floor:

```
import pymysql as m
```

```

mydb=m.connect(host ='localhost',user ='root' ,password='')
mycursor = mydb.cursor()
mycursor.execute("use student;")
mycursor.execute("select floor(4.765) from dual;")
data=mycursor.fetchall()
for i in data:
for a in i:
print(a)

```



```

In [4]: 1 import pymysql as m
2 mydb=m.connect(host ='localhost',user ='root' ,password='')
3 mycursor = mydb.cursor()
4 mycursor.execute("use student;")
5 mycursor.execute("select floor(4.765) from dual;")
6 data=mycursor.fetchall()
7 for i in data:
8     for a in i:
9         print(a)

```

4

fig.9

Using of trigonometric functions:

```

import pymysql as m
mydb=m.connect(host ='localhost',user ='root' ,password='')
mycursor = mydb.cursor()
mycursor.execute("use student;")
mycursor.execute("select sin(1.4),cos(3.6),tan(0.6) from dual;")
data=mycursor.fetchall()
for i in data:
for a in i:
print(a)

```

It will take the values in radian.


```

In [12]: 1 import pymysql as m
          2 mydb=m.connect(host='localhost',user='root',password='')
          3 mycursor = mydb.cursor()
          4 mycursor.execute("use student;")
          5 mycursor.execute("select sin(1.4),cos(3.6),tan(0.6) from dual;")
          6 data=mycursor.fetchall()
          7 for i in data:
          8     for a in i:
          9         print(a)

0.9854497299884601
-0.896758416334147
0.6841368083416923

```

fig.10

Using of exp ():

```

import pymysql as m
mydb=m.connect(host='localhost',user='root',password='')
mycursor = mydb.cursor()
mycursor.execute("use student;")
mycursor.execute("select exp(4) from dual;")
data=mycursor.fetchall()
for i in data:
    for a in i:
        print(a)

```

Returns e the base natural logarithm raised to a power.

```

In [13]: 1 import pymysql as m
          2 mydb=m.connect(host='localhost',user='root',password='')
          3 mycursor = mydb.cursor()
          4 mycursor.execute("use student;")
          5 mycursor.execute("select exp(4) from dual;")
          6 data=mycursor.fetchall()
          7 for i in data:
          8     for a in i:
          9         print(a)

54.598150033144236

```

fig.11

Using of power ():

```

import pymysql as m


```

```

mydb=m.connect(host='localhost',user='root',password='')
mycursor = mydb.cursor()
mycursor.execute("use student;")
mycursor.execute("select power(4,2) from dual;")
data=mycursor.fetchall()
for i in data:
    for a in i:
        print(a)

```

Returns the a to the power b.



```

In [14]: 1 import pymysql as m
          2 mydb=m.connect(host='localhost',user='root',password='')
          3 mycursor = mydb.cursor()
          4 mycursor.execute("use student;")
          5 mycursor.execute("select power(4,2) from dual;")
          6 data=mycursor.fetchall()
          7 for i in data:
          8     for a in i:
          9         print(a)

```

16.0

fig.12

Using of ln():

```

import pymysql as m
mydb=m.connect(host='localhost',user='root',password='')
mycursor = mydb.cursor()
mycursor.execute("use student;")
mycursor.execute("select ln(4) from dual;")
data=mycursor.fetchall()
for i in data:
    for a in i:
        print(a)

```

```

In [15]: 1 import pymysql as m
          2 mydb=m.connect(host='localhost',user='root',password='')
          3 mycursor = mydb.cursor()
          4 mycursor.execute("use student;")
          5 mycursor.execute("select ln(4) from dual;")
          6 data=mycursor.fetchall()
          7 for i in data:
          8     for a in i:
          9         print(a)
1.3862943611198906

```

fig.13

Using of mod():

```

import pymysql as m
mydb=m.connect(host='localhost',user='root',password='')
mycursor = mydb.cursor()
mycursor.execute("use student;")
mycursor.execute("select mod(11,2) from dual;")
data=mycursor.fetchall()
for i in data:
    for a in i:
        print(a)

```

It gives the remainder of the numbers.

```

In [16]: 1 import pymysql as m
          2 mydb=m.connect(host='localhost',user='root',password='')
          3 mycursor = mydb.cursor()
          4 mycursor.execute("use student;")
          5 mycursor.execute("select mod(11,2) from dual;")
          6 data=mycursor.fetchall()
          7 for i in data:
          8     for a in i:
          9         print(a)
1

```

fig.14

Character function:

```

import pymysql as m

```

```

mydb=m.connect(host='localhost',user='root',password='')
mycursor = mydb.cursor()
mycursor.execute("use student;")
mycursor.execute("select lower('Jack'),upper('jack'),replace('jack','j','b'),instr('software','f'),substr('software',5),concat('ware','software')
from dual;")
data=mycursor.fetchall()
for i in data:
    for a in i:
        print(a)

```

Lower will convert all the letters into lower case. Upper will convert all the letters into upper case. Replace will replace the letter or word in the given string. Instr will check whether the string is present in the given string or not and returns the index of their position if it is present. Substr will create a sub string from the given string from the starting point till ending point specified. Concat will concatenate the 2 given strings.



The screenshot shows a Jupyter Notebook cell with the following code and output:

```

In [4]: 1 import pymysql as m
        2 mydb=m.connect(host='localhost',user='root',password='')
        3 mycursor = mydb.cursor()
        4 mycursor.execute("use student;")
        5 mycursor.execute("select lower('Jack'),upper('jack'),replace('jack','j','b'),instr('software','f'),substr('software',5),concat('ware','software')
        6 data=mycursor.fetchall()
        7 for i in data:
        8     for a in i:
        9         print(a)

```

The output of the code is:

```

jack
JACK
back
3
ware
software

```

fig.15

```

import pymysql as m
mydb=m.connect(host='localhost',user='root',password='')
mycursor = mydb.cursor()
mycursor.execute("use student;")
mycursor.execute("select char(65),ascii('A'),lpad('function',15,'-'),rpad('function',15,'-'),length('function')
from dual;")
data=mycursor.fetchall()

```

```
for i in data:
```

```
for a in i:
```

```
print(a)
```

Char will convert a number into a ascii equivalent value. ASCII will convert the ASCII equivalent value to the number. Lpad will do left padding. Rpad will do right padding. Length will calculate the length of the string.



```
In [16]: 1 import pymysql as m
2 mydb=m.connect(host='localhost',user='root',password='')
3 mycursor = mydb.cursor()
4 mycursor.execute("use student;")
5 mycursor.execute("select char(65),ascii('A'),lpad('function',15,'-'),rpad('function',15,'-'),length('function') from dual;")
6 data=mycursor.fetchall()
7 for i in data:
8     for a in i:
9         print(a)
```

```
b'A'
65
-----function
function-----
8
```

fig.16

Soundex:

Compare words that are spelled differently but sound alike.

```
import pymysql as m
```

```
mydb=m.connect(host='localhost',user='root',password='')
```

```
mycursor = mydb.cursor()
```

```
mycursor.execute("use student;")
```

```
mycursor.execute("select stname from studentinfo where soundex(stname)=soundex('scout');")
```

```
data=mycursor.fetchall()
```

```
for i in data:
```

```
for a in i:
```

```
print(a)
```

```
In [17]: 1 import pymysql as m
2 mydb=m.connect(host='localhost',user='root',password='')
3 mycursor = mydb.cursor()
4 mycursor.execute("use student;")
5 mycursor.execute("select stname from studentinfo where soundex(stname)=soundex('scout');")
6 data=mycursor.fetchall()
7 for i in data:
8     for a in i:
9         print(a)
```

scoot

fig.17

Extract:

The EXTRACT function extracts a part from a given date.

```
import pymysql as m
mydb=m.connect(host='localhost',user='root',password='')
mycursor = mydb.cursor()
mycursor.execute("use student;")
mycursor.execute("SELECT EXTRACT(YEAR FROM addate) from studentinfo;")
data=mycursor.fetchall()
for i in data:
    for a in i:
        print(a)
```

It will extract the year from the date.

```
In [22]: 1 import pymysql as m
2 mydb=m.connect(host='localhost',user='root',password='')
3 mycursor = mydb.cursor()
4 mycursor.execute("use student;")
5 mycursor.execute("SELECT EXTRACT(YEAR FROM addate) from studentinfo;")
6 data=mycursor.fetchall()
7 for i in data:
8     for a in i:
9         print(a)
```

2028
2008
2004
2006
2007
2007

fig.18

Aggregate functions:

It produce a single value for an entire group or table they are used to produce summarized result they operate onsets of rows they return results based on groups of rows. By default all rows in a table are treated as one groups in all group functions null are ignored.

Count:

Determines the number of rows or non null columns values if * is passed then the total number of rows is returned.

Sum:

Determine the number of rows or non null column values if passed and determines the sum of all selected columns.

Max:

Determines the largest of all the selected column values of a column.

Average:

Determine the average of all selected values of a column.

Min:

Determines the smallest of all selected values of a column.

```
import pymysql as m
mydb=m.connect(host='localhost',user='root',password='')
mycursor = mydb.cursor()
```

```
mycursor.execute("use student;")
mycursor.execute("select count(stno),max(fees),min(fees),sum(fees),avg(fees) from studentinfo;")
data=mycursor.fetchall()
for i in data:
    for a in i:
        print(a)
```

```
In [23]: 1 import pymysql as m
          2 mydb=m.connect(host='localhost',user='root',password='')
          3 mycursor = mydb.cursor()
          4 mycursor.execute("use student;")
          5 mycursor.execute("select count(stno),max(fees),min(fees),sum(fees),avg(fees) from studentinfo;")
          6 data=mycursor.fetchall()
          7 for i in data:
          8     for a in i:
          9         print(a)

6
200
150
1000
180.0000
```

fig.1

Group by clause and having clause:

Group by clause:

The group by clause is used to divide the rows in a table into smaller groups it is used with select to combine a group particular column or expression. Aggregate functions are used to return summary information for each group the aggregate function are applied to the individual groups.

Having clause:

Having clause is used to specify which group are to be displayed, that is restrict the groups that you return on the basic of aggregate function.

Example for group by clause:

```
import pymysql as m
mydb=m.connect(host='localhost',user='root',password='')
mycursor = mydb.cursor()
mycursor.execute("use student;")
mycursor.execute("select count(*),course from studentinfo group by course order by count(*) desc;")
data=mycursor.fetchall()
for i in data:
    print(i)
```

```
In [26]: 1 import pymysql as m
2 mydb=m.connect(host='localhost',user='root',password='')
3 mycursor = mydb.cursor()
4 mycursor.execute("use student;")
5 mycursor.execute("select count(*),course from studentinfo group by course order by count(*) desc;")
6 data=mycursor.fetchall()
7 for i in data:
8     print(i)

(3, 'english')
(1, 'german')
(1, 'france')
(1, 'tamil')
```

fig.1

Example for having clause:

```
import pymysql as m
mydb=m.connect(host='localhost',user='root',password='')
mycursor = mydb.cursor()
mycursor.execute("use student;")
mycursor.execute("select count(*),course,fees from studentinfo group by course having fees<=180 ;")
data=mycursor.fetchall()
for i in data:
    print(i)
```

```
In [31]: 1 import pymysql as m
2 mydb=m.connect(host='localhost',user='root',password='')
3 mycursor = mydb.cursor()
4 mycursor.execute("use student;")
5 mycursor.execute("select count(*),course,fees from studentinfo group by course having fees<=180 ;")
6 data=mycursor.fetchall()
7 for i in data:
8     print(i)

(3, 'english', 180)
(1, 'german', 150)
(1, 'france', 180)
```

fig.2

Special functions and operators of sql:

IN operator:

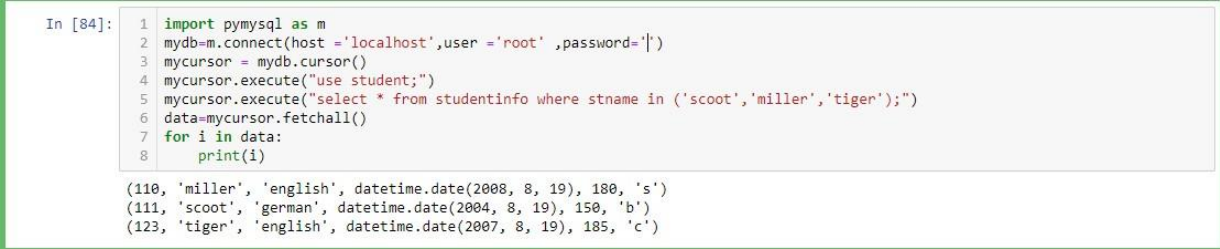
We can be used to select the rows that match one of the values in a list.

Using of IN operator:

Using pymysql library:

```
import pymysql as m
mydb=m.connect(host='localhost',user='root',password='')
mycursor = mydb.cursor()
mycursor.execute("use student;")
mycursor.execute("select * from studentinfo where stname in ('scoot', 'miller', 'tiger');")
data=mycursor.fetchall()
for i in data:
    print(i)
```

These lines will print only the details of scoot, miller and tiger.



```
In [84]: 1 import pymysql as m
2 mydb=m.connect(host='localhost',user='root',password='')
3 mycursor = mydb.cursor()
4 mycursor.execute("use student;")
5 mycursor.execute("select * from studentinfo where stname in ('scoot', 'miller', 'tiger');")
6 data=mycursor.fetchall()
7 for i in data:
8     print(i)

(110, 'miller', 'english', datetime.date(2008, 8, 19), 180, 's')
(111, 'scoot', 'german', datetime.date(2004, 8, 19), 150, 'b')
(123, 'tiger', 'english', datetime.date(2007, 8, 19), 185, 'c')
```

fig.1

Like operator:

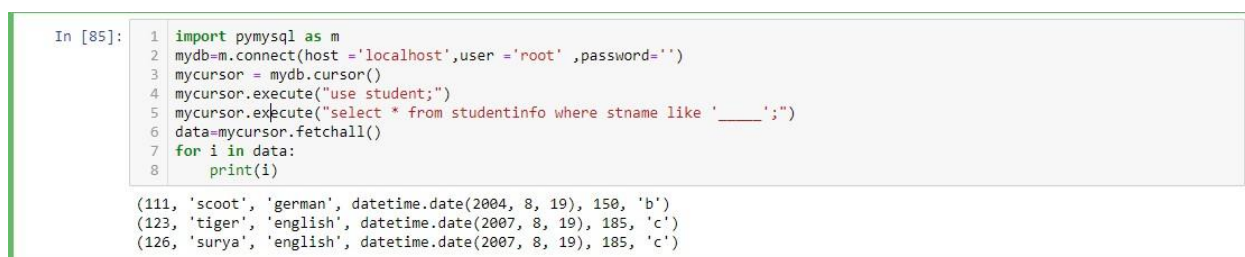
Use to search a character pattern we need not know the exact character value it recognise the special characters like person m_ the format can matches % can be used to match null character .

Using of LIKE operator:

Using **pymysql** library:

```
import pymysql as m
mydb=m.connect(host='localhost',user='root',password='')
mycursor = mydb.cursor()
mycursor.execute("use student;")
mycursor.execute("select * from studentinfo where stname like '____';")
data=mycursor.fetchall()
for i in data:
    print(i)
```

These lines will filter the info of the students whose name is having only 5 letters.



```
In [85]: 1 import pymysql as m
2 mydb=m.connect(host='localhost',user='root',password='')
3 mycursor = mydb.cursor()
4 mycursor.execute("use student;")
5 mycursor.execute("select * from studentinfo where stname like '____';")
6 data=mycursor.fetchall()
7 for i in data:
8     print(i)

(111, 'scoot', 'german', datetime.date(2004, 8, 19), 150, 'b')
(123, 'tiger', 'english', datetime.date(2007, 8, 19), 185, 'c')
(126, 'surya', 'english', datetime.date(2007, 8, 19), 185, 'c')
```

fig.2

Round function:

Date is rounded to the unit specified by the format model.

Round (date, [format])

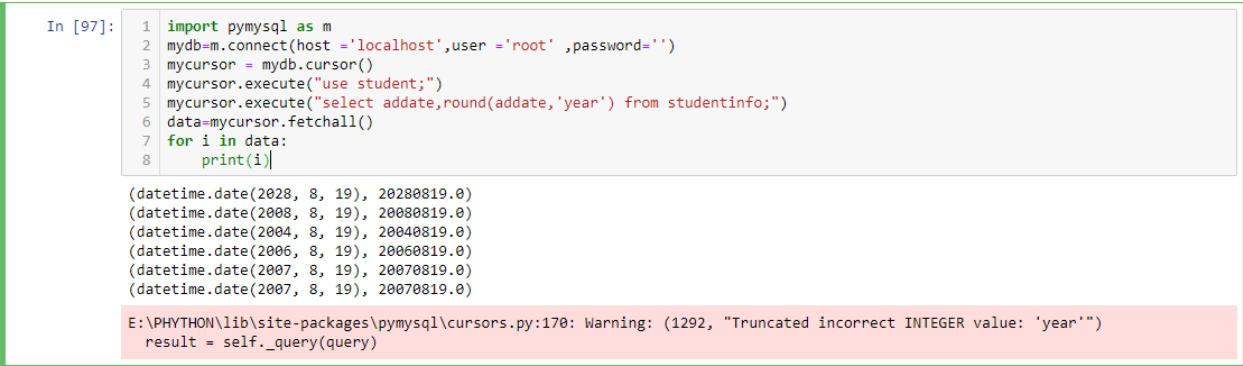
Format can be year, month or day. If year is specified in format it refers to the month. If the month is greater than 6 then the date is rounded to 1-jan-currentyear. If the month specified in format it refers to days. If the days are greater than the date is rounded to 1-nextmonth-currentyear. If days are less than or equal to 15 then the date is rounded to 1-currentmonth-currentyear. If the date is specified in the format then the date is rounded to the nearest Sunday.

Using of round function:

Using **pymysql** library:

```
import pymysql as m
mydb=m.connect(host='localhost',user='root',password='')
mycursor = mydb.cursor()
mycursor.execute("use student;")
mycursor.execute("select adddate, round(adddate,'year') from studentinfo;")
data=mycursor.fetchall()
for i in data:
    print(i)
```

This will round the admission date of the students.



```
In [97]: 1 import pymysql as m
2 mydb=m.connect(host='localhost',user='root',password='')
3 mycursor = mydb.cursor()
4 mycursor.execute("use student;")
5 mycursor.execute("select adddate,round(adddate,'year') from studentinfo;")
6 data=mycursor.fetchall()
7 for i in data:
8     print(i)

(datetime.date(2028, 8, 19), 20280819.0)
(datetime.date(2008, 8, 19), 20080819.0)
(datetime.date(2004, 8, 19), 20040819.0)
(datetime.date(2006, 8, 19), 20060819.0)
(datetime.date(2007, 8, 19), 20070819.0)
(datetime.date(2007, 8, 19), 20070819.0)

E:\PYTHON\lib\site-packages\pymysql\cursors.py:170: Warning: (1292, "Truncated incorrect INTEGER value: 'year'")
  result = self._query(query)
```

fig.3

Dual:

It is a system table automatically created by the oracle along with the dictionary it has one column defined to be varchar datatype and contains

only one row of value 'X'.

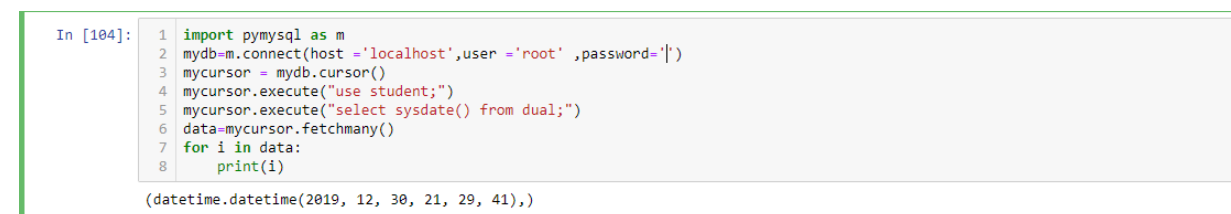
System date function:

The PLSQL SYSDATE function will returns current system date and time on your database. There is no any parameter or argument for the SYSDATE function. The SYSDATE function returns a date value. Note that the SYSDATE function return date and time as "YYYY-MM-DD HH:MM:SS" (string) or as YYYYMMDDHHMMSS (numeric).

Using of sysdate () function:

Using **pymysql** library:

```
import pymysql as m
mydb=m.connect(host='localhost',user='root',password='')
mycursor = mydb.cursor()
mycursor.execute("use student;")
mycursor.execute("select sysdate() from dual;")
data=mycursor.fetchall()
for i in data:
    print(i)
```



```
In [104]: 1 import pymysql as m
2 mydb=m.connect(host='localhost',user='root',password='')
3 mycursor = mydb.cursor()
4 mycursor.execute("use student;")
5 mycursor.execute("select sysdate() from dual;")
6 data=mycursor.fetchall()
7 for i in data:
8     print(i)

(datetime.datetime(2019, 12, 30, 21, 29, 41),)
```

fig.4

System_user ():


The SYSTEM_USER function returns the login name for the current user.

Using of System_user ():

Using **pymysql** library:

```
import pymysql as m
mydb=m.connect(host='localhost',user='root',password='')
mycursor = mydb.cursor()
mycursor.execute("use student;")
mycursor.execute("select System_user() from dual;")
data=mycursor.fetchall()
for i in data:
print(i)
```

This will return the current user.



```
In [108]: 1 import pymysql as m
2 mydb=m.connect(host='localhost',user='root',password='')
3 mycursor = mydb.cursor()
4 mycursor.execute("use student;")
5 mycursor.execute("select system_user() from dual;")
6 data=mycursor.fetchall()
7 for i in data:
8     print(i)

('root@localhost',)
```

fig.5

Greatest ():

The **GREATEST** is an inbuilt function in sql which is used to return the greatest value from a given list of some expressions. These expressions may be numbers, alphabets etc.

Using of greatest ():

Using **pymysql** library:

```
import pymysql as m
mydb=m.connect(host='localhost',user='root',password='')
```

```

mycursor = mydb.cursor()
mycursor.execute("use student;")
mycursor.execute("select greatest(1,2,3,4) from dual;")
data=mycursor.fetchall()
for i in data:
    print(i)

```

This will return the greatest among the values.



```

In [110]: 1 import pymysql as m
          2 mydb=m.connect(host='localhost',user='root',password='')
          3 mycursor = mydb.cursor()
          4 mycursor.execute("use student;")
          5 mycursor.execute("select greatest(1,2,3,4) from dual;")
          6 data=mycursor.fetchall()
          7 for i in data:
          8     print(i)

(4,)

```

fig.6

Numeric functions:

It accepts numeric input and returns numeric value as output.

Using of abs ():

From here after all the code will use **pymysql** library:

```

import pymysql as m
mydb=m.connect(host='localhost',user='root',password='')
mycursor = mydb.cursor()
mycursor.execute("use student;")
mycursor.execute("select abs(-234) from dual;")
data=mycursor.fetchall()
for i in data:
    for a in i:
        print(a)

```

Here the absolute value of -234 is the output.

```
In [1]: 1 import pymysql as m
2 mydb=m.connect(host ='localhost',user ='root' ,password='')
3 mycursor = mydb.cursor()
4 mycursor.execute("use student;")
5 mycursor.execute("select abs(-234) from dual;")
6 data=mycursor.fetchall()
7 for i in data:
8     for a in i:
9         print(a)
```

234

fig.7

Using of ceil ():

```
import pymysql as m
mydb=m.connect(host ='localhost',user ='root' ,password='')
mycursor = mydb.cursor()
mycursor.execute("use student;")
mycursor.execute("select ceil(4.765) from dual;")
data=mycursor.fetchall()
for i in data:
    for a in i:
        print(a)
```

The value is rounded to nearest higher value in this case it is 5.

```
In [3]: 1 import pymysql as m
2 mydb=m.connect(host ='localhost',user ='root' ,password='')
3 mycursor = mydb.cursor()
4 mycursor.execute("use student;")
5 mycursor.execute("select ceil(4.765) from dual;")
6 data=mycursor.fetchall()
7 for i in data:
8     for a in i:
9         print(a)
```

5

fig.8

Using of floor:

```
import pymysql as m
```

```

mydb=m.connect(host ='localhost',user ='root' ,password='')
mycursor = mydb.cursor()
mycursor.execute("use student;")
mycursor.execute("select floor(4.765) from dual;")
data=mycursor.fetchall()
for i in data:
for a in i:
print(a)

```



```

In [4]: 1 import pymysql as m
2 mydb=m.connect(host ='localhost',user ='root' ,password='')
3 mycursor = mydb.cursor()
4 mycursor.execute("use student;")
5 mycursor.execute("select floor(4.765) from dual;")
6 data=mycursor.fetchall()
7 for i in data:
8     for a in i:
9         print(a)

```

4

fig.9

Using of trigonometric functions:

```

import pymysql as m
mydb=m.connect(host ='localhost',user ='root' ,password='')
mycursor = mydb.cursor()
mycursor.execute("use student;")
mycursor.execute("select sin(1.4),cos(3.6),tan(0.6) from dual;")
data=mycursor.fetchall()
for i in data:
for a in i:
print(a)

```

It will take the values in radian.

```

In [12]: 1 import pymysql as m
          2 mydb=m.connect(host='localhost',user='root',password='')
          3 mycursor = mydb.cursor()
          4 mycursor.execute("use student;")
          5 mycursor.execute("select sin(1.4),cos(3.6),tan(0.6) from dual;")
          6 data=mycursor.fetchall()
          7 for i in data:
          8     for a in i:
          9         print(a)

0.9854497299884601
-0.896758416334147
0.6841368083416923

```

fig.10

Using of exp ():

```

import pymysql as m

mydb=m.connect(host='localhost',user='root',password='')
mycursor = mydb.cursor()

mycursor.execute("use student;")
mycursor.execute("select exp(4) from dual;")
data=mycursor.fetchall()

for i in data:
    for a in i:
        print(a)

```

Returns e the base natural logarithm raised to a power.

```

In [13]: 1 import pymysql as m
          2 mydb=m.connect(host='localhost',user='root',password='')
          3 mycursor = mydb.cursor()
          4 mycursor.execute("use student;")
          5 mycursor.execute("select exp(4) from dual;")
          6 data=mycursor.fetchall()
          7 for i in data:
          8     for a in i:
          9         print(a)

54.598150033144236

```

fig.11

Using of power ():

```

import pymysql as m


```

```

mydb=m.connect(host='localhost',user='root',password='')
mycursor = mydb.cursor()
mycursor.execute("use student;")
mycursor.execute("select power(4,2) from dual;")
data=mycursor.fetchall()
for i in data:
    for a in i:
        print(a)

```

Returns the a to the power b.



```

In [14]: 1 import pymysql as m
2 mydb=m.connect(host='localhost',user='root',password='')
3 mycursor = mydb.cursor()
4 mycursor.execute("use student;")
5 mycursor.execute("select power(4,2) from dual;")
6 data=mycursor.fetchall()
7 for i in data:
8     for a in i:
9         print(a)

```

16.0

fig.12

Using of ln():

```

import pymysql as m
mydb=m.connect(host='localhost',user='root',password='')
mycursor = mydb.cursor()
mycursor.execute("use student;")
mycursor.execute("select ln(4) from dual;")
data=mycursor.fetchall()
for i in data:
    for a in i:
        print(a)

```

```

In [15]: 1 import pymysql as m
2 mydb=m.connect(host='localhost',user='root',password='')
3 mycursor = mydb.cursor()
4 mycursor.execute("use student;")
5 mycursor.execute("select ln(4) from dual;")
6 data=mycursor.fetchall()
7 for i in data:
8     for a in i:
9         print(a)

```

1.3862943611198906

fig.13

Using of mod():

```

import pymysql as m
mydb=m.connect(host='localhost',user='root',password='')
mycursor = mydb.cursor()
mycursor.execute("use student;")
mycursor.execute("select mod(11,2) from dual;")
data=mycursor.fetchall()
for i in data:
    for a in i:
        print(a)

```

It gives the remainder of the numbers.

```

In [16]: 1 import pymysql as m
2 mydb=m.connect(host='localhost',user='root',password='')
3 mycursor = mydb.cursor()
4 mycursor.execute("use student;")
5 mycursor.execute("select mod(11,2) from dual;")
6 data=mycursor.fetchall()
7 for i in data:
8     for a in i:
9         print(a)

```

1

fig.14

Character function:

```

import pymysql as m

```

```

mydb=m.connect(host='localhost',user='root',password='')
mycursor = mydb.cursor()
mycursor.execute("use student;")
mycursor.execute("select lower('Jack'),upper('jack'),replace('jack','j','b'),instr('software','f'),substr('software',5),concat('ware','software')
from dual;")
data=mycursor.fetchall()
for i in data:
    for a in i:
        print(a)

```

Lower will convert all the letters into lower case. Upper will convert all the letters into upper case. Replace will replace the letter or word in the given string. Instr will check whether the string is present in the given string or not and returns the index of their position if it is present. Substr will create a sub string from the given string from the starting point till ending point specified. Concat will concatenate the 2 given strings.



```

In [4]: 1 import pymysql as m
2 mydb=m.connect(host='localhost',user='root',password='')
3 mycursor = mydb.cursor()
4 mycursor.execute("use student;")
5 mycursor.execute("select lower('Jack'),upper('jack'),replace('jack','j','b'),instr('software','f'),substr('software',5),concat('ware','software')
6 data=mycursor.fetchall()
7 for i in data:
8     for a in i:
9         print(a)

```

jack
JACK
back
3
ware
software

fig.15

```

import pymysql as m
mydb=m.connect(host='localhost',user='root',password='')
mycursor = mydb.cursor()
mycursor.execute("use student;")
mycursor.execute("select char(65),ascii('A'),lpad('function',15,'-'),rpad('function',15,'-'),length('function')
from dual;")
data=mycursor.fetchall()

```

for i in data:

for a in i:

print(a)

Char will convert a number into a ascii equivalent value. ASCII will convert the ASCII equivalent value to the number. Lpad will do left padding. Rpad will do right padding. Length will calculate the length of the string.



```
In [16]: 1 import pymysql as m
2 mydb=m.connect(host='localhost',user='root',password='')
3 mycursor = mydb.cursor()
4 mycursor.execute("use student;")
5 mycursor.execute("select char(65),ascii('A'),lpad('function',15,'-'),rpad('function',15,'-'),length('function') from dual;")
6 data=mycursor.fetchall()
7 for i in data:
8     for a in i:
9         print(a)

b'A'
65
-----function
function-----
8
```

fig.16

Soundex:

Compare words that are spelled differently but sound alike.

```
import pymysql as m
```

```
mydb=m.connect(host='localhost',user='root',password='')
```

```
mycursor = mydb.cursor()
```

```
mycursor.execute("use student;")
```

```
mycursor.execute("select stname from studentinfo where soundex(stname)=soundex('scout');")
```

```
data=mycursor.fetchall()
```

```
for i in data:
```

```
for a in i:
```

```
print(a)
```

```
In [17]: 1 import pymysql as m
2 mydb=m.connect(host='localhost',user='root',password='')
3 mycursor = mydb.cursor()
4 mycursor.execute("use student;")
5 mycursor.execute("select stname from studentinfo where soundex(stname)=soundex('scout');")
6 data=mycursor.fetchall()
7 for i in data:
8     for a in i:
9         print(a)
```

scoot

fig.17

Extract:

The EXTRACT function extracts a part from a given date.

```
import pymysql as m
mydb=m.connect(host='localhost',user='root',password='')
mycursor = mydb.cursor()
mycursor.execute("use student;")
mycursor.execute("SELECT EXTRACT(YEAR FROM addate) from studentinfo;")
data=mycursor.fetchall()
for i in data:
    for a in i:
        print(a)
```

It will extract the year from the date.

```
In [22]: 1 import pymysql as m
2 mydb=m.connect(host='localhost',user='root',password='')
3 mycursor = mydb.cursor()
4 mycursor.execute("use student;")
5 mycursor.execute("SELECT EXTRACT(YEAR FROM addate) from studentinfo;")
6 data=mycursor.fetchall()
7 for i in data:
8     for a in i:
9         print(a)
```

2028
2008
2004
2006
2007
2007

fig.18

Constraints:

Restriction to the data being entered is called as constraints. Maintaining security and integrity of a database is the most important factor in judging the success of a system. An integrity constraint is a mechanism used by oracle to prevent the invalid data entry into the table in other words constraints are used for enforcing rules that the columns in a table have to conform with.

3 types of constraints:

- Domain integrity constraints.
- Entity integrity constraints.
- Referential integrity constraints.

Domain integrity constraints:

These constraints set a range and any violation that take place will prevent that take place will prevent that caused the breath.

These are basically 2 types:

- Not null constraints.
- Check constraints.

Not null constraints:

By default the table can contain null values the enforcement of not null constraints in a table ensures that the table contains values. Oracle will not validate until this is satisfied.

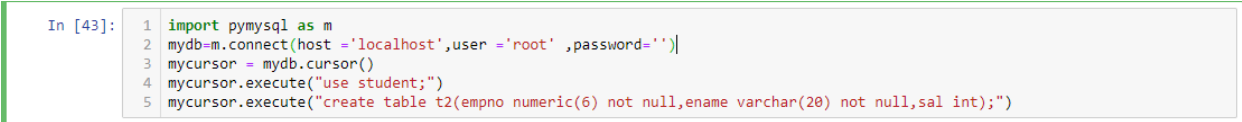
Check constraints:

It allows only a particular range of values. When the value specified in the range is violated oracle will rejects the record. Check constraints specify conditions that each row must satisfied. These are rules governed by logical expression or Boolean expression check condition cannot have sub queries.

Example of creating a table with not null constraints:

```
import pymysql as m
mydb=m.connect(host='localhost',user='root',password='')
mycursor = mydb.cursor()
mycursor.execute("use student;")
mycursor.execute("create table t2(empno numeric(6) not null, ename varchar(20) not null, Sal int);")
```

This will create a table with not null constraint.



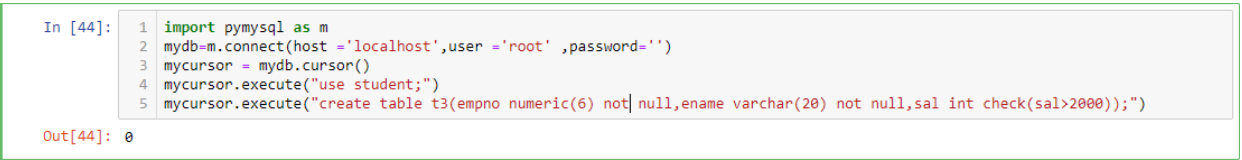
```
In [43]: 1 import pymysql as m
2 mydb=m.connect(host='localhost',user='root',password='')
3 mycursor = mydb.cursor()
4 mycursor.execute("use student;")
5 mycursor.execute("create table t2(empno numeric(6) not null,ename varchar(20) not null,sal int);")
```

fig.1

Example for creating a table with check constraints:

```
import pymysql as m
mydb=m.connect(host='localhost',user='root',password='')
mycursor = mydb.cursor()
mycursor.execute("use student;")
mycursor.execute("create table t3(empno numeric(6) not null, ename varchar(20) not null, sal int
check(sal>2000));")
```

This will create a table with the check constraints.



```
In [44]: 1 import pymysql as m
2 mydb=m.connect(host='localhost',user='root',password='')
3 mycursor = mydb.cursor()
4 mycursor.execute("use student;")
5 mycursor.execute("create table t3(empno numeric(6) not null,ename varchar(20) not null,sal int check(sal>2000));")

Out[44]: 0
```

fig.2

Entity integrity constraints:

An entity is any data recorded in a database. Each entity represented a table and each row of the table represents an instance of that entity. Each row in a table can be uniquely identified using the entity constraints.

Unique constraints:

Usage of the unique key constraints is to prevent the duplication of values within the row of a specified columns or a set of columns in a table. Columns defined within the constraints can allow null values. If unique key constraints are defined in more than one column for example combination of columns, it is said to be composite unique key. Combination of columns cannot be duplicate maximum combination of columns that composite unique key can contain is 16.

Primary key constraints:

The primary key constraint avoids duplication of rows and does not allow null values, when enforced in a column or set of columns. As a result it is used to identify a row. A table can have only one primary key, if a primary key constraint is assigned to a combination of columns it is said to be a composite primary key. This can contain a maximum of 16 columns.

Table level and column level constraints:

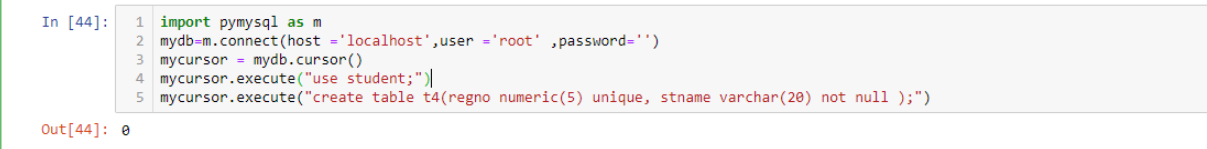
The table level constraints is part of the table definition, an integrity constraints defined at table level can impose rules on any columns in the table where as column level constraints begin a part of the column definition on which it is defined.

Example for creating a table with a unique key constraint:

```
import pymysql as m
```

```
mydb=m.connect(host='localhost',user='root',password='')
mycursor = mydb.cursor()
mycursor.execute("use student;")
mycursor.execute("create table t4(regno numeric(5) unique, stname varchar(20) not null );")
```

This will create a table with unique key constraint.



```
In [44]: 1 import pymysql as m
2 mydb=m.connect(host='localhost',user='root',password='')
3 mycursor = mydb.cursor()
4 mycursor.execute("use student;")
5 mycursor.execute("create table t4(regno numeric(5) unique, stname varchar(20) not null );")

Out[44]: 0
```

fig.3

Example for creating a table with a primary key constraint:

```
import pymysql as m
mydb=m.connect(host='localhost',user='root',password='')
mycursor = mydb.cursor()
mycursor.execute("use student;")
mycursor.execute("create table t7(regno numeric(6) primary key, stname varchar(20) not null);")
```

This will create a table with the primary key constraint.

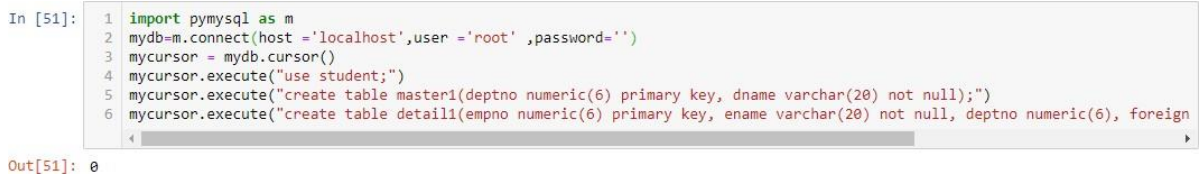
Referential integrity constraint:

The referential integrity constraint enforces relationship between the tables establishing the parent and child or master or slave relationship. A master-detail relationship between two tables having a common column, where we make use of referential integrity, should define the columns in the present table as a primary key and the same column referring to the corresponding parent entry. It designates a column or combination of columns as a primary key and the same column in the child table as a foreign key, establishing a relationship with a specified primary key or unique key in another table called the referenced key. In this relationship, the table

contain the foreign key is called the child table and the table contain the primary key is called master table.

Example for creating table with referential integrity constraint:

```
import pymysql as m
mydb=m.connect(host='localhost',user='root',password='')
mycursor = mydb.cursor()
mycursor.execute("use student;")
mycursor.execute("create table master1(deptno numeric(6) primary key, dname varchar(20) not null);")
mycursor.execute("create table detail1(empno numeric(6) primary key, ename varchar(20) not null, deptno numeric(6), foreign key(deptno) references master1(deptno));")
```



```
In [51]: 1 import pymysql as m
2 mydb=m.connect(host='localhost',user='root',password='')
3 mycursor = mydb.cursor()
4 mycursor.execute("use student;")
5 mycursor.execute("create table master1(deptno numeric(6) primary key, dname varchar(20) not null);")
6 mycursor.execute("create table detail1(empno numeric(6) primary key, ename varchar(20) not null, deptno numeric(6), foreign key(deptno) references master1(deptno));")

Out[51]: 0
```

fig.4

Set operators:

Set operators combine the results of two queries into a single row. Both the queries should have same number of columns and corresponding columns must have the same type. no long data type is allowed.

If order by clause is used it should be used at the end of the last query.

They are 4 set operators:

- Union (all unique values without any redundancy.)
- Union all (includes duplicate values from both the queries.)
- Intersect (common to both the query.)
- Minus (returns all distinct rows selected only by first query and not by the second.)

Example for union operator:

```
import pymysql as m
mydb=m.connect(host='localhost',user='root',password='')
mycursor = mydb.cursor()
mycursor.execute("use student;")
mycursor.execute("select course from studentinfo where fees<=150 union select course from studentinfo where fees<190 ;")
data=mycursor.fetchall()
for i in data:
    for a in i:
        print(a)
```

This will combine the results of two queries and duplicate values are removed.

```
In [36]: 1 import pymysql as m
2 mydb=m.connect(host='localhost',user='root',password='')
3 mycursor = mydb.cursor()
4 mycursor.execute("use student;")
5 mycursor.execute("select course from studentinfo where fees<=150 union select course from studentinfo where fees<190 ;")
6 data=mycursor.fetchall()
7 for i in data:
8     for a in i:
9         print(a)

german
english
france
```

fig.1

Example for union all operators:

```
import pymysql as m

mydb=m.connect(host='localhost',user='root',password='')

mycursor = mydb.cursor()

mycursor.execute("use student;")

mycursor.execute("select course from studentinfo where fees<=150 union all select course from
studentinfo where fees<190 ;")

data=mycursor.fetchall()

for i in data:
    for a in i:
        print(a)
```

This will combine the results of two queries and the duplicate values are not removed in this case.

```
In [37]: 1 import pymysql as m
2 mydb=m.connect(host='localhost',user='root',password='')
3 mycursor = mydb.cursor()
4 mycursor.execute("use student;")
5 mycursor.execute("select course from studentinfo where fees<=150 union all select course from studentinfo where fees<190 ;")
6 data=mycursor.fetchall()
7 for i in data:
8     for a in i:
9         print(a)

german
english
german
france
english
english
```

fig.2

Joins:

Joins is used to combine the data spread across tables.

- Simple join.
- Self join.
- Outer join.

Simple join:

Common type of join, returns rows from 2 tables having a common column. It is categorised into 2 types that are equi join and non-equi join.

Equi-join:

A join which is based on equalities it combines the rows that have equivalent values for specified columns.

Non equi-join:

It specifies the relationship between columns belonging to different tables by making use of relational operator.

Cross join:

The cross product of two tables same as Cartesian product.

Examples for simple join (cross join):


```
import pymysql as m
mydb=m.connect(host='localhost',user='root',password='')
mycursor = mydb.cursor()
```



```
mycursor.execute("use student;")
```

```
mycursor.execute("select ename, sal, dname, loc from emp cross join dept;")
```

This will join 2 tables to form a single table.



```
In [51]: 1 import pymysql as m
2 mydb=m.connect(host='localhost',user='root',password='')
3 mycursor = mydb.cursor()
4 mycursor.execute("use student;")
5 mycursor.execute("select ename, sal, dname, loc from emp cross join dept;")

Out[51]: 0
```

fig.1

Example for simple joins (equi join):

```
import pymysql as m
```

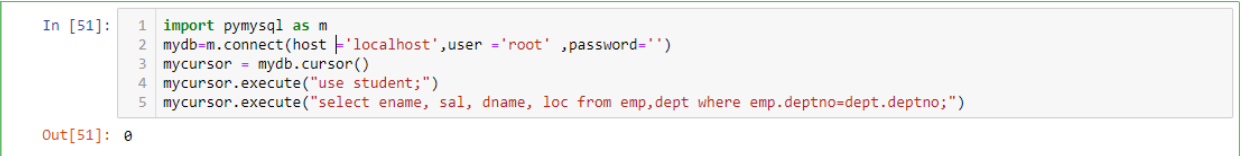
```
mydb=m.connect(host='localhost',user='root',password='')
```

```
mycursor = mydb.cursor()
```

```
mycursor.execute("use student;")
```

```
mycursor.execute("select ename, sal, dname, loc from emp, dept where emp.deptno=dept.deptno;")
```

This join 2 tables having same deptno.



```
In [51]: 1 import pymysql as m
2 mydb=m.connect(host='localhost',user='root',password='')
3 mycursor = mydb.cursor()
4 mycursor.execute("use student;")
5 mycursor.execute("select ename, sal, dname, loc from emp,dept where emp.deptno=dept.deptno;")

Out[51]: 0
```

fig.2

Example of simple join (non equi join):

```
import pymysql as m
```

```
mydb=m.connect(host='localhost',user='root',password='')
```

```
mycursor = mydb.cursor()
```

```
mycursor.execute("use student;")
```

```
mycursor.execute("select ename, sal, dname, loc from emp, dept where emp.sal<=3000;")
```

This will join two tables contains the sal lesser than equal to 3000.

```
In [51]: 1 import pymysql as m
2 mydb=m.connect(host='localhost',user='root',password='')
3 mycursor = mydb.cursor()
4 mycursor.execute("use student;")
5 mycursor.execute("select ename, sal, dname, loc from emp, dept where emp.sal<=3000;")

Out[51]: 0
```

fig.3

Self join:

To join a table to itself means that each row of the table is combined with itself and with every other row of the table. The self join can be viewed as a join of two copies of the sample table.

Examples for self join:

```
import pymysql as m
mydb=m.connect(host='localhost',user='root',password='')
mycursor = mydb.cursor()
mycursor.execute("use student;")
mycursor.execute("select w.ename, m.ename from emp 'w', emp 'm' dept where w.mgr=m.empno;")
```

This will join the table to itself.

```
In [51]: 1 import pymysql as m
2 mydb=m.connect(host='localhost',user='root',password='')
3 mycursor = mydb.cursor()
4 mycursor.execute("use student;")
5 mycursor.execute("select w.ename, m.ename from emp 'w', emp 'm' dept where w.mgr=m.empno;")

Out[51]: 0
```

fig.4

Outer join:

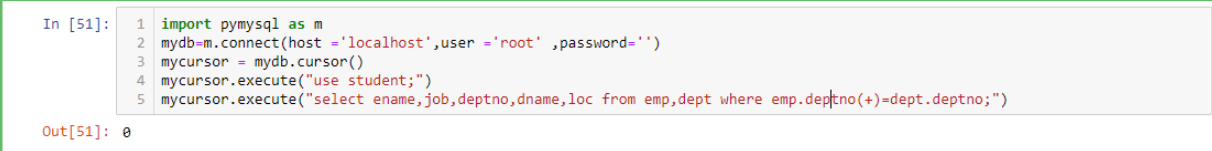
If there are any values in one table that do not have corresponding values in the other, in a equi join that row will not selected by using

the outer join symbol (+). The corresponding columns for that row will have nulls. If the symbol (+) is placed on the other side of the equation then all the employee details with no corresponding department name and location , will be displayed with null values in dname and loc column. The outer join cannot be on both sides.

Example of Right outer join:

```
import pymysql as m
mydb=m.connect(host='localhost',user='root',password='')
mycursor = mydb.cursor()
mycursor.execute("use student;")
mycursor.execute("select ename, job, deptno, dname, loc from emp, dept where emp.deptno(+) = dept.deptno;")
```

This will join 2 tables using right outer join.



```
In [51]: 1 import pymysql as m
2 mydb=m.connect(host='localhost',user='root',password='')
3 mycursor = mydb.cursor()
4 mycursor.execute("use student;")
5 mycursor.execute("select ename,job,deptno,dname,loc from emp,dept where emp.deptno(+) = dept.deptno;")

Out[51]: 0
```

fig.5

Example of left outer join:

```
import pymysql as m
mydb=m.connect(host='localhost',user='root',password='')
mycursor = mydb.cursor()
mycursor.execute("use student;")
mycursor.execute("select ename, job, deptno, dname, loc from emp, dept where emp.deptno = dept.deptno(+);")
```

This will join two tables by using left outer join.

```

In [51]: 1 import pymysql as m
          2 mydb=m.connect(host='localhost',user='root',password='')
          3 mycursor = mydb.cursor()
          4 mycursor.execute("use student;")
          5 mycursor.execute("select ename,job,deptno,dname,loc from emp,dept where emp.deptno=dept.deptno(+);")

Out[51]: 0

```

fig.6

Natural join:

Natural join is based on all columns in the two tables that have the same name it select rows from the 2 tables that have equal values in all matched columns. If the columns having the same names but having different data type an error is returned with natural join the (.) qualifier is not used.

Example of natural join;

```

import pymysql as m
mydb=m.connect(host='localhost',user='root',password='')
mycursor = mydb.cursor()
mycursor.execute("use student;")
mycursor.execute("select ename, job, deptno, dname, loc from emp natural join dept;")

```

This will join 2 tables using natural join.

```

In [51]: 1 import pymysql as m
          2 mydb=m.connect(host='localhost',user='root',password='')
          3 mycursor = mydb.cursor()
          4 mycursor.execute("use student;")
          5 mycursor.execute("select ename,job,deptno,dname,loc from emp natural join dept;")

Out[51]: 0

```

fig.7

Sub queries and locking:

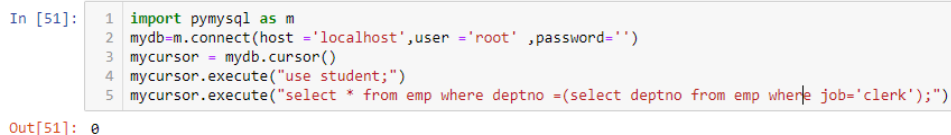
Sub queries:

Query within in the query is call sub query. The inner most query will be evaluated first and output of the inner query is given as an input to the outer query and the outer query is evaluated if the order by clause has to be used it should be used at the end of the inner query. The inner query should return only value as the output. That has to be given as the input of the outer query and if the inner query returns more than one row we need multi row comparison operations.

Example of sub query:

```
import pymysql as m
mydb=m.connect(host='localhost',user='root',password='')
mycursor = mydb.cursor()
mycursor.execute("use student;")
mycursor.execute("select * from emp where deptno =(select deptno from emp where job='clerk');")
```

This will select all records of the clerk.



```
In [51]: 1 import pymysql as m
2 mydb=m.connect(host='localhost',user='root',password='')
3 mycursor = mydb.cursor()
4 mycursor.execute("use student;")
5 mycursor.execute("select * from emp where deptno =(select deptno from emp where job='clerk');")

Out[51]: 0
```

fig.1

Concept of locking:

Oracle automatically uses different types of lock to control concurrent access to data and to prevent destructive interaction between users. Oracle automatically locks a resources on behalf of a traction this is

done to prevent other truncation from doing something also requiring the exclusive access to the same automatically acquires different types of locks at different levels of restrictiveness depending on the resources begin locked and the operation begin performed the lock is released automatically where an event occurs so that the transaction no longer requires the resources.

Types of locking:

- Row level locking.
- Table level locking.


Row level locking:

In the row level locking row is locked exclusive so that other users cannot modify the row until the transaction holding the lock is committed or rolled back. Row locks are acquired automatically by oracle as a result of insert, update, delete and select for update clause statement.

Select... for update clause: The select command when used with for update of clause places an exclusive lock on one or more rows of a table this command can be used to lock the rows that would be updated later.

Example of row level locking:

```
import pymysql as m
mydb=m.connect(host='localhost',user='root',password='')
mycursor = mydb.cursor()
mycursor.execute("use student;")
mycursor.execute("select * from emp where empno in (100,200) for update of ename, sal;")
```



```
In [51]: 1 import pymysql as m
2 mydb=m.connect(host='localhost',user='root',password='')
3 mycursor = mydb.cursor()
4 mycursor.execute("use student;")
5 mycursor.execute("select * from emp where empno in (100,200) for update of ename, sal;")

Out[51]: 0
```

fig.2

Table level lock:

- Share lock
- Share update lock
- Exclusive lock

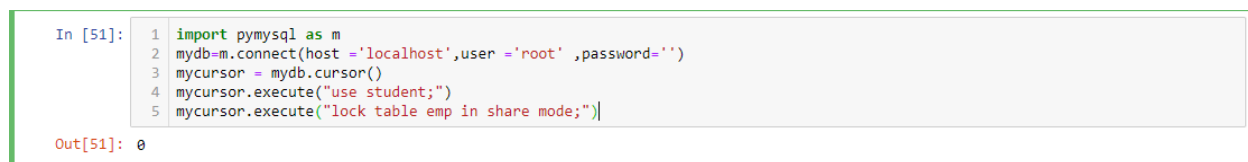
Share lock:

A share lock locks the table allowing other users to only query but not insert, update or delete rows in a table. Multiple users can place share lock at the same time it allows resources to be shared and hence the name shared lock.

Example of share lock:

```
import pymysql as m
mydb=m.connect(host='localhost',user='root',password='')
mycursor = mydb.cursor()
mycursor.execute("use student;")
mycursor.execute("lock table emp in share mode;")
```

This will lock the table in shared lock mode.



```
In [51]: 1 import pymysql as m
2 mydb=m.connect(host='localhost',user='root',password='')
3 mycursor = mydb.cursor()
4 mycursor.execute("use student;")
5 mycursor.execute("lock table emp in share mode;")

Out[51]: 0
```

fig.3

Share update lock:

It locks rows that are to be updated in a table it permits other users to concurrently query insert, update or even lock other rows in the same table it prevents the other users from updating the row that has been locked. We can enforce a share update lock by using the share update clause in the selected statements.

Examples of share update lock:

```
import pymysql as m
mydb=m.connect(host='localhost',user='root',password='')
mycursor = mydb.cursor()
mycursor.execute("use student;")
mycursor.execute("lock table emp in share update mode;")
```

This will lock the table in share lock mode.



fig.4

Exclusive lock:

Exclusive lock is the most restrictive of table locks. When issued by one user to only query but not insert, delete or update rows in a table. It is almost similar to a share lock but only one user can place an exclusive lock on a table at a time whereas many users can place a share lock on the same table at the same time.

Example of exclusive lock:

```
import pymysql as m
mydb=m.connect(host='localhost',user='root',password='')
mycursor = mydb.cursor()
mycursor.execute("use student;")
mycursor.execute("lock table emp in exclusive mode;")
```

This will lock the table in exclusive manner.


```

In [51]: 1 import pymysql as m
          2 mydb=m.connect(host='localhost',user='root',password='')
          3 mycursor = mydb.cursor()
          4 mycursor.execute("use student;")
          5 mycursor.execute("lock table emp in exclusive mode;")

Out[51]: 0

```

fig.5

No wait:

Consider a user has locked a table without any wait clause in the locks table for mat. If another user tries to lock the table, then he will make to wait initially locked table issues a commit or rollback statement. The delay could be avoided by appending a no wait clause in the lock table command. (Only for exclusive lock.)

Example of no wait:

```

import pymysql as m

mydb=m.connect(host='localhost',user='root',password='')

mycursor = mydb.cursor()

mycursor.execute("use student;")

mycursor.execute("lock table emp in exclusive mode nowait;")

```

```

In [51]: 1 import pymysql as m
          2 mydb=m.connect(host='localhost',user='root',password='')
          3 mycursor = mydb.cursor()
          4 mycursor.execute("use student;")
          5 mycursor.execute("lock table emp in exclusive mode nowait;")

Out[51]: 0

```

fig.6

Dead lock:

A dead lock occurs when two user have a lock, each other's object when this happens, the first user has to wait for the second user to release the lock but the second user will not release the lock until the lock on the first user's object is freed. At this point, both the user are at an impasse and cannot proceed with their business. In such case Oracle detects the dead lock automatically and solves problem by aborting one of the two transactions.

Database objects:

- Synonyms
- Sequences
- Views
- Indexes

Synonyms:

It is a database object which is used as an alias (alternative name) for a table, view or sequence they are used to,

- Simply sql statements.
- Hide the name and owner of an object.
- Provide location transparency for remote objects of a distributed database.
- Provide public access to an object.

Synonyms can either be private or public. The former is created by normal user, which is available only to that person where as the latter is created by the DBA, which can be availed by any database user.

Syntax:

Create [public] synonym <synonym_name> for <table_name>;

Public synonyms are created by a database administrator to hide the identity complexity of sql statements one such example of public synonym is **TAB**.

Sequences:

It is a database object which can generate unique, sequential integer values. It can be used to automatically generate primary key or unique key values. A sequence can be either in an ascending or a descending order.

Increment by n: n is an integer which specifies the interval between sequence numbers. The default is 1. If n is positive, then the sequence ascends and if it is negative the sequence descends.

Start with n: specifies the first sequence number to be generated.

Min value n: specifies the minimum value of ascending sequence and 10e26-1 for a descending sequence.

Max value n: it specifies the maximum value that the sequence can generate for descending and ascending sequences respectively.

Cycle: specifies that the sequence continues to generate values from the beginning after reaching either it is max or min value.

No cycle: specifies that the sequence cannot generate more values after reaching either it is max value or min value than the default value is no cycle.

Cache: cache option pre-allocates a set of sequences number and retains them in memory so that sequence numbers can be accessed faster. When the last of the sequence number in the cache has been used. Oracle reads another set of number into the cache.

No cache: the default value no cache does not pre allocate sequence number for faster access.

After creating sequence we can access its values with the help of pseudo columns like curval and nextval. A pseudo column behaves like a table column but it select values from pseudo columns but cannot perform manipulations on their values.

Next Val: nextval returns initial values of the sequence, when referred to the first time later reference to the next Val will increment by clause and return the new value.

Curval: curval returns the current value of the sequence which is the value returned by the last reference to the nextval.

Views:

A view is a tailored presentation of the data contained in one or more tables (or other views). A view takes the output of a query and treats it as a table therefore a view can be thought of as a stored query or virtual table. We can be used the table upon which a view is based are called base tables.

Advantages of views:

They provide an additional level of table security by restring access to and predetermine set of rows and or column of a table. They hide data complexity for example a single view might be defined with a join, which is a collection of related columns or rows in multiple tables. They simplify commands for the user because they allow them to select information from multiple tables without actually knowing how to perform a join. They isolate applications from changes in definitions of base tables. For example if a views defining query references three columns of a four columns table. The view definition is not affected and all application using the view is not affected. Using the view is not affected. They provide data in a different perspective than that of a base table by renaming columns without affecting the base tables.

Syntax:

```
Create [[or replace] [no] [force]] view <view_name>
[column alias name] as <query> [with [check option] [read
only] [constraint]];
```

Example of creating view:

```
import pymysql as m
mydb=m.connect(host = 'localhost',user = 'root' ,password='')
mycursor = mydb.cursor()
mycursor.execute("use student;")
mycursor.execute("create or replace view v1 as select stname, fees from studentinfo;")
```

This will create a view v1 with 2 columns.

```

In [53]: 1 import pymysql as m
          2 mydb=m.connect(host='localhost',user='root',password='')
          3 mycursor = mydb.cursor()
          4 mycursor.execute("use student;")
          5 mycursor.execute("create or replace view v1 as select stname,fees from studentinfo;")

Out[53]: 0

```

fig.1

Example of view with check option:

```

import pymysql as m
mydb=m.connect(host='localhost',user='root',password='')
mycursor = mydb.cursor()
mycursor.execute("use student;")
mycursor.execute("create or replace view v2 as select stname,fees from studentinfo where fees<=180
with check option ;")

```

```

In [54]: 1 import pymysql as m
          2 mydb=m.connect(host='localhost',user='root',password='')
          3 mycursor = mydb.cursor()
          4 mycursor.execute("use student;")
          5 mycursor.execute("create or replace view v2 as select stname,fees from studentinfo where fees<=180 with check option ;")

Out[54]: 0

```

fig.2

Without check option:

This will produce rows which are not viewable through the view. Creating the same view with check option can prevent this it will allow us to update only those rows which are viewable through view and give an error when we try to override it.

Example of order by clause in view:

```

import pymysql as m
mydb=m.connect(host='localhost',user='root',password='')
mycursor = mydb.cursor()

```

```
mycursor.execute("use student;")
```

```
mycursor.execute("create or replace view v2 as select sname, fees from studentinfo order by fees desc;")
```

```
In [55]: 1 import pymysql as m
          2 mydb=m.connect(host='localhost',user='root',password='')
          3 mycursor = mydb.cursor()
          4 mycursor.execute("use student;")
          5 mycursor.execute("create or replace view v2 as select sname,fees from studentinfo order by fees desc;")

Out[55]: 0
```

fig.3

DML statements and join views:

Joining of tables is also possible in a view. Any update, insert or delete statement on a join view can modify only one underlying base table. But it is possible through instead of trigger.

Key preserved tables:

A table is key preserved if every key of the table can also be a key of the table can also be a key of the result of the join.

Example of using joins in view:

```
import pymysql as m
```

```
mydb=m.connect(host='localhost',user='root',password='')
```

```
mycursor = mydb.cursor()
```

```
mycursor.execute("use student;")
```

```
mycursor.execute("create or replace view v2 as select ename, job, sal from emp, dept where emp.deptno=dept.deptno;")
```

```
In [55]: 1 import pymysql as m
          2 mydb=m.connect(host='localhost',user='root',password='')
          3 mycursor = mydb.cursor()
          4 mycursor.execute("use student;")
          5 mycursor.execute("create or replace view v2 as select ename, job, sal from emp, dept where emp.deptno=dept.deptno;")

Out[55]: 0
```

fig.4

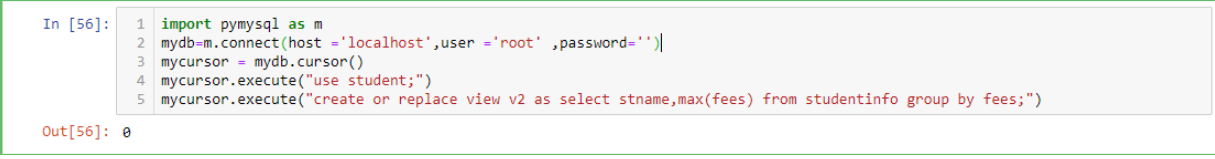
Functions in a view:

Views query cannot select pseudo columns like `curval` and `nextval` if a view query functions and set operators, group function and distinct clause then deletion, updation, and insertion cannot be performed in a view with affect base table and vice versa.

Group function and group by clause can also be included in view while using the function the columns should be given alias name.

Example of using the functions in view:

```
import pymysql as m
mydb=m.connect(host='localhost',user='root',password='')
mycursor = mydb.cursor()
mycursor.execute("use student;")
mycursor.execute("create or replace view v2 as select sname, max(fees) from studentinfo group by fees;")
```



```
In [56]: 1 import pymysql as m
2 mydb=m.connect(host='localhost',user='root',password='')
3 mycursor = mydb.cursor()
4 mycursor.execute("use student;")
5 mycursor.execute("create or replace view v2 as select sname,max(fees) from studentinfo group by fees;")

Out[56]: 0
```

fig.5

Partition view:

With partition views data resides in separate tables. These tables are then brought together at runtime using the relational operator union all.

Index:


Indexes are optional structures associated with tables we can create indexes explicitly to speed up sql statement execution on a table. Similar to the indexes in books that helps us to locate information

faster. An oracle index provides a faster access path to table data.

When a index key column is used in the sql statements where clause, then the index points directly clause, then the index point directly to the location of the rows containing those values, indexes are the primary means of reducing disk i/o , index is merely a fast access path to the data.

Example of creating index:

```
import pymysql as m
mydb=m.connect(host='localhost',user='root',password='')
mycursor = mydb.cursor()
mycursor.execute("use student;")
mycursor.execute("create index a1 on studentinfo(fees);")
```



```
In [58]: 1 import pymysql as m
2 mydb=m.connect(host='localhost',user='root',password='')
3 mycursor = mydb.cursor()
4 mycursor.execute("use student;")
5 mycursor.execute("create index a1 on studentinfo(fees);")

Out[58]: 0
```

fig.6

Unique index:

Indexes can be unique or non unique. Unique index guarantee that no two rows of a table have duplicate values in the columns that defined the index. Non unique index do not impose this restriction on the column values. Oracle enforce unique integrity constraints by automatically define a unique index on the unique key. A unique index is automatically created when we create unique or primary key.

Example of unique index:

```
import pymysql as m
mydb=m.connect(host='localhost',user='root',password='')
mycursor = mydb.cursor()
```



```
mycursor.execute("use student;")
```

```
mycursor.execute("create unique index a2 on studentinfo(fees);")
```

```
In [58]: 1 import pymysql as m
          2 mydb=m.connect(host='localhost',user='root',password='')
          3 mycursor = mydb.cursor()
          4 mycursor.execute("use student;")
          5 mycursor.execute("create unique index a2 on studentinfo(fees);")

Out[58]: 0
```

fig.7

Composite index:

A composite index also called concatenated index is an index created on multiple columns of a table. Columns in a composite index can appear in any order and need not be adjacent columns of the table. Composite indexes can enhance the retrieval speed of data for select statements in which the where clause references all or the leading portion of the columns in the composite index generally the most commonly accessed or most selective columns go first in the order of the columnlist.

The index created with 2 column will be used to retrieve data when the where clause has either both of these column or the first column alone but not while second column alone is used.

Example of composite index:

```
import pymysql as m
```

```
mydb=m.connect(host='localhost',user='root',password='')
```

```
mycursor = mydb.cursor()
```

```
mycursor.execute("use student;")
```

```
mycursor.execute("create unique index a2 on studentinfo(fees, stname);")
```

```
In [58]: 1 import pymysql as m
          2 mydb=m.connect(host='localhost',user='root',password='')
          3 mycursor = mydb.cursor()
          4 mycursor.execute("use student;")
          5 mycursor.execute("create unique index a2 on studentinfo(fees,stname);")

Out[58]: 0
```

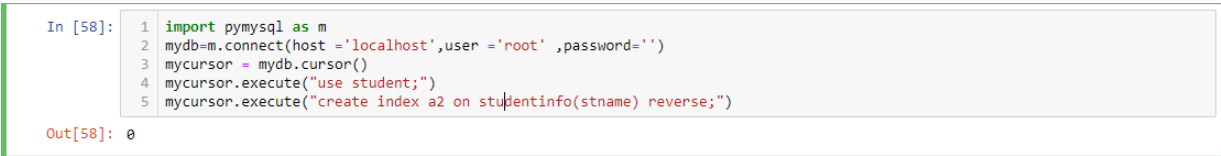
fig.8

Reverse key index:

Creating a reverse key index, when compared to a standard index, reverse each byte of the column begin indexed while keeping the column order such an degradation in indexes where modifications to the indexes are concentrated on a small set of blocks. By reversing the keys of the indexes the instructions become distributed all over the index.

Example of reverse key index:

```
import pymysql as m
mydb=m.connect(host='localhost',user='root',password='')
mycursor = mydb.cursor()
mycursor.execute("use student;")
mycursor.execute("create index a2 on studentinfo(stname) reverse;")
```



```
In [58]: 1 import pymysql as m
2 mydb=m.connect(host='localhost',user='root',password='')
3 mycursor = mydb.cursor()
4 mycursor.execute("use student;")
5 mycursor.execute("create index a2 on studentinfo(stname) reverse;")

Out[58]: 0
```

fig.9

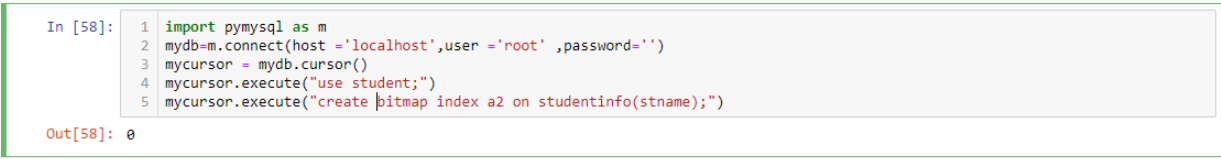
Bitmap index:

The advantages of using bitmap index are greatest for low cardinality columns that are columns in which the number of distinct values is small compared to the number of rows in table. If the values in a column are repeated more than a hundred times, the column is a candidate for a bitmap index. For example on a table with one million rows, a column with 10,000 distinct values is a candidate for a bitmap index in a bitmap index a bitmap for each key value is used instead of a list of row ids each bits in a bit map corresponding to a possible rowed, and if the bit is set it means that the row with the corresponding to a possible row id and if the bit is set, it means that the row with the corresponding rowed contains the key value. A mapping function converts the bit position to an actual rowed , hence the bitmap index provides

the same functionality as a regular index even though it uses a different representation internally if the number of different key values is small, bitmap indexes are very space efficient.

Example of bit map index:

```
import pymysql as m
mydb=m.connect(host='localhost',user='root',password='')
mycursor = mydb.cursor()
mycursor.execute("use student;")
mycursor.execute("create bitmap index a2 on studentinfo(stname);")
```



```
In [58]: 1 import pymysql as m
2 mydb=m.connect(host='localhost',user='root',password='')
3 mycursor = mydb.cursor()
4 mycursor.execute("use student;")
5 mycursor.execute("create bitmap index a2 on studentinfo(stname);")

Out[58]: 0
```

fig.10

Clusters:

Clusters are an optional method of storing table data, a cluster is group of tables that share the same data blocks because they share same data common columns and are often used together cluster are special configurations to use when two or more table are stored in close physical proximity to improve performance on sql join statements using those tables cluster is a method of storing tables that are intimately related and often joined together into the same area on disk.

Cluster key:

The cluster key is the column or group of columns that the cluster tables have in common for example deptno in EMP and dept table you specify the columns of the cluster key when creating the cluster.

Gui development using python and dbms:

```
from tkinter import *
import pymysql as m
import sqlite3

root = Tk()
root.geometry('500x500')
root.title("Registration Form")

Fullname=StringVar()
Email=StringVar()
var = IntVar()
c=StringVar()
var1= IntVar()

def database():
    name1=Fullname.get()
    email=Email.get()
    gender=var.get()
    country=c.get()
    prog=var1.get()
    data=name1,email,str(gender),country,str(prog)
    data=str(data)
    mydb=m.connect(host='localhost',user='root',password='your_password',db='form')
    mycursor = mydb.cursor()
    with mydb:
        cursor = mydb.cursor()
        cursor.execute('CREATE TABLE IF NOT EXISTS Student(Fullname varchar(20),Email varchar(50),Gender
        varchar(2),country varchar(15),Programming varchar(2))')
```

```
cursor.execute('INSERT INTO Student (FullName,Email,Gender,country,Programming) VALUES'+data+';')
mydb.commit()
label_0 = Label(root, text="Registration form",width=20,font=("bold", 20))
label_0.place(x=90,y=53)
label_1 = Label(root, text="FullName",width=20,font=("bold", 10))
label_1.place(x=80,y=130)
entry_1 = Entry(root,textvar=Fullname)
entry_1.place(x=240,y=130)
label_2 = Label(root, text="Email",width=20,font=("bold", 10))
label_2.place(x=68,y=180)
entry_2 = Entry(root,textvar=Email)
entry_2.place(x=240,y=180)
label_3 = Label(root, text="Gender",width=20,font=("bold", 10))
label_3.place(x=70,y=230)
Radiobutton(root, text="Male",padx = 5, variable=var, value=1).place(x=235,y=230)
Radiobutton(root, text="Female",padx = 20, variable=var, value=2).place(x=290,y=230)
label_4 = Label(root, text="country",width=20,font=("bold", 10))
label_4.place(x=70,y=280)
list1 = ['Canada','India','UK','Nepal','Iceland','South Africa'];
droplist=OptionMenu(root,c,*list1)
droplist.config(width=15)
c.set('select your country')
droplist.place(x=240,y=280)
label_4 = Label(root, text="Programming",width=20,font=("bold", 10))
label_4.place(x=85,y=330)
var2= IntVar()
Checkbutton(root, text="java", variable=var1).place(x=235,y=330)
Checkbutton(root, text="python", variable=var2).place(x=290,y=330)
Button(root, text='Submit',width=20,bg='brown',fg='white',command=database).place(x=180,y=380)
root.mainloop()
```
