

---

## UNIT 16 DATA ACCESS USING PYTHON

---

### Structure

- 16.1 Introduction
- 16.2 Database Concepts
- 16.3 Creating Database
- 16.4 Querying Database
- 16.5 Using SQL to get more out of Database
- 16.6 CSV files in Python
- 16.7 Summary
- 16.8 Solutions to Check your Progress

---

### 16.1 INTRODUCTION

---

Python is the most popular high-level programming language that can be used for real-world programming. It is a dynamic and object-oriented programming language that can be used in a vast domain of applications, especially data handling. Python was designed as a very user-friendly general-purpose language with a collection of modules and packages and gained popularity in the recent times. Whenever we think of machine learning, big data handling, data analysis, statistical algorithms, python is the only name which comes first in our mind. It is a dynamically typed language, which makes it highly flexible. Furthermore, creating a database and linking of the database is very fast and secure. It makes the program to compile and run in an extremely flexible environment can support different styles of programming, including structural and object-oriented. Its ability to use modular components that were designed in other programming languages allows a programmer to embed the code written in python for any interface. Due to its enormous features, python has become the first choice of learners in the field of data science and machine learning. This unit is specially designed for beginners to create and connect database in python. You will learn to connect data through SQL as well as CSV files with the help of real dataset (PIMA Indian Dataset), used most commonly by researchers and academia in data analysis.

---

### 16.2 DATABASE CONCEPTS

---

The database is a collection of an interrelated, organized form of persistent data. It has properties of integrity and redundancy to serve numerous applications. It is a storehouse of all important information, not only for large business enterprises rather for anyone who is handling data even at the micro-level. Traditionally, we used to store data using file systems, which eventually taken by advanced Database Management Systems software, which stores, manipulates and secure the data in the most promising manner. To access the data stored in the database, one must know query language like SQL, MySQL etc., which can retrieve the information from the database server and passes it to the client. Thus, client can easily approach the server

for day to day transactions by hitting the database without worrying about the load on the processor as it is very fast and consumes power to run the desired query only.

The nature of serendipity in the Python environment is an example of data analysis in machine learning and datascience. Python is a bucket of packages and libraries which enables us to perform data classification, prediction and analysis in a very efficient way.

Now the question is, the database is stored in different forms in different software's like Oracle, MS-Access etc., so how any programming language like python can access it. For that, we need to learn about database connectivity in python. In this unit you will learn how to create and connect database in python using MySQL along with basic knowledge of database and its features.

### Database Management system DBMS

Database management systems are software that serves the special purpose of storing, managing, extracting and modifying data from a database. It contains a set of programs that allows access to data contained in a database. DBMS also interfaces with application programs so that data contained in the database can be used by multiple application and users. It handles all access to the database and responsible for applying the authorization checks and validation procedures. It acts as a bridge between users and data, and the bridge is crossed using special query language like SQL, MYSQL wrote in high-level languages.

A typical DBMS has users with different rights and permissions who use it for different purposes. Some users retrieve data and some back it up. The users of a DBMS can be broadly categorized as follows –

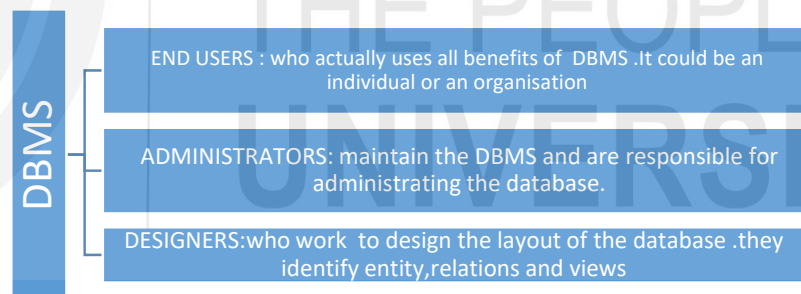


Fig 16.1 (a) Basic structure of users of DBMS

As we discussed no information processing is possible without a database. It is the hub of data in the field of information technology. Some of the major advantages of the database are listed below:

- **Real-world entity** – A modern DBMS is more realistic and uses real-world entities to design its architecture. It uses the behaviour and attributes too. For example, a school database may use students as an entity and their age as an attribute.
- **Reduced data redundancy:** Centralized control of database of avoids unnecessary duplication of data and effectively reduces the total amount of data storage required, unlike the non-database system in which each department is maintaining its own copy of data. Avoiding duplication of data also results in

the elimination of data inconsistencies that tend to be present in duplicate data files.

- **Data consistency:** Centralized control over data, ensures data consistency as there is no duplication of data. In the non-database system, there are multiple copies of the same data with each department, whenever the data is updated, the changes are not reflected in each department which leads to data inconsistency. For example, change in address of the student is reflected in the teacher's record but not in account department, which results in inconsistent data.
- **Shared data:** A database allows the sharing of data by multiple application, users and programs like a database of students can be shared by the teachers for making results as well as admin department to keep track of the fees account.
- **Greater data integrity and independence from applications programs:** Integrity of data means that data stored in the database is both accurate and consistent. Centralized control ensures that adequate checks are incorporated in a database while entering data so that DBMS provide data integrity. DBMS provide sufficient validations to make sure that data fall within a specified range and are of the correct format.
- **Improved data control:** In many organizations, where typically each application has its own private files. This makes the operational data widely dispersed and difficult to control. Central database provide the easy maintenance of the database Access to users through the use of host and query languages
- **Improved data security:** The data is protected in a central system. Data is more secured against unauthorized access control with authentication and encoding mechanism. Different checks and rights can be applied for the access of information from the database
- Reduced data entry, storage, and retrieval costs
- Facilitated development of new applications program

#### Limitations of Database:

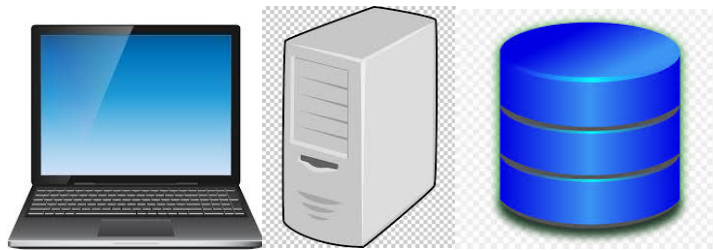
- **Substantial hardware and software start-up costs:** Extensive conversion costs is involved in moving from a file-based system to a database system
- **Database systems are complex,** difficult, and time-consuming to design.
- Damage to database affects virtually all applications programs
- Initial training required for all programmers and users.
- **Cost:** The cost of required hardware, DB development, and DB maintenance is high.
- **Complexity:** Due to its complexity, the user should understand it well enough to use it efficiently and effectively.

#### DBMS Architecture

DBMS is designed on the basis of architecture. This design can be of various forms centralized, client-server systems, parallel systems, distributed and hierarchical. The architecture of a DBMS can be seen as either a single tier or multi-tier. An n-tier

architecture divides the whole system into related but independent n modules, which can be independently modified, altered, changed, or replaced. Generally, three-tier architecture is followed, which consist of three layers:

1. Presentation layer (laptop,PC, Tablet, Mobile, etc.)
2. Application layer (server)
3. Database Server



Client (Presentation layer)    Server(Application layer)    Database

**Fig 16.1(b) Three tier architecture of DBMS**

Multiple-tier database architecture is highly modifiable, as almost all its components are independent and can be changed independently.

### **Data models**

Once the architecture of DBMS is designed, then the next phase is modelling. The data model represents the logical structure of the database. It decides how the data will be stored, connected to each other, processed and stored. Many data models were proposed to store the data like network model, hierarchical model and relational model where each new model was invented with improved features. The first model was based on **Flat File**, where data is stored in a single table in a file, which could be a plain text file or binary file. It is suitable only for a small amount of data. Records follow a uniform format, and there are no structures for indexing or recognizing relationships between records. Later on, the concept of relation is introduced where data is stored in the form of multiple tables and tables are linked using a common field. It is suitable for handling medium to a large amount of data. This is the most promising model ever implemented for DBMS solutions. So, let us see in detail about the structure and features of the relational model.

### **Entity-Relationship Model**

An **Entity-relationship model (ER model)** describes the logical structure of a database with the help of a diagram, which is known as an **Entity Relationship Diagram (ER Diagram)**. It is based on the notion of real-world entities and relationships among them. While formulating real-world scenario into the database model, the ER Model creates an entity set, relationship set, general attributes and constraints. ER Model is best used for the conceptual design of a database. ER Model is based on – Entities and their attributes. Relationships among entities. These concepts are explained below.

- **Entity:** An entity is a real-world object which can be easily identifiable like in an employee database, employee, department, products can be considered as entities.

- **Attributes:** Entities are represented by means of having properties called **attributes**. Every attribute is defined by its set of values called domain. For example, in an employee database, an employee is considered as an entity. An employee has various attributes like empname, age, department, salary etc.
- **Relationship:** describes the logical association among entities. These relationships are mapped with entities in various ways, and the number of association between two entities defines mapping cardinalities. Mapping cardinalities are one to one, one to many, many to many and many to one. For example, a doctor can have many patients in a database which shows one to many relationships.

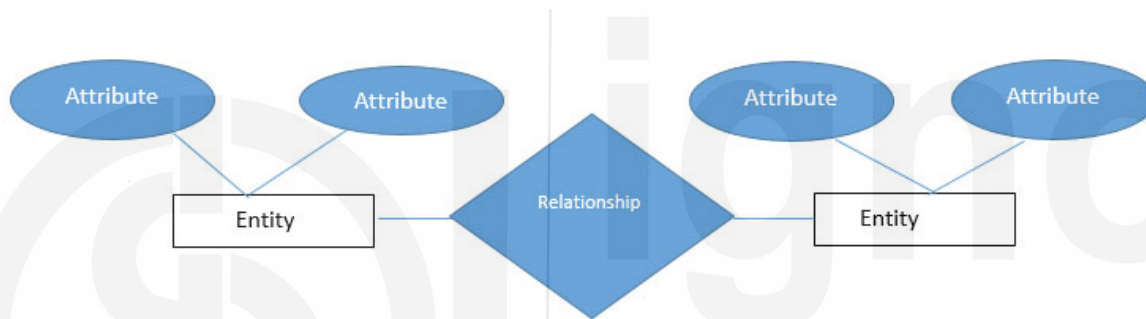
These E-R diagrams are represented by special symbols which are listed below:

**Rectangle:** Represents Entity sets.

**Ellipses:** Attributes

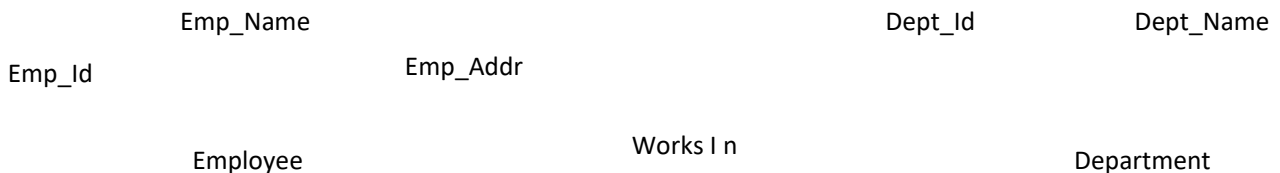
**Diamonds:** Relationship Set

**Lines:** They link attributes to Entity Sets and Entity sets to Relationship Set



**Fig 16.1(c) E-R diagram**

Now let us take an example of an organization where they want to design an entity relationship between various departments and their employees. So, In the following E-R diagram we have taken two entities Employee and Department and showing their relationship. The relationship between Employee and Department is many to one as a department can have many employees however an employee cannot work in multiple departments at the same time. Employee entity has attributes such as Emp\_Id, Emp\_Name & Emp\_Addr and Department entity has attributes such as Dept\_ID & Dept\_Name.



**Fig 16.1(d) Sample E-R Diagram showing employee and department relationship.**

## Structure of Database

A database system is a computer-based system to record and maintain information. The information concerned can be anything of significance to the organization for whose use it is intended. The contents of a database can hold a variety of different things. To make database design more straight-forward, databases contents are divided up into two concepts:

- Schema
- Data

**The Schema** is the structure of data, whereas the Data are the "facts". Schema can be complex to understand to begin with, but really indicates the rules which the Data must obey. Let us consider a real-world scenario where we want to store facts about employees for an organization. Such facts may include Emp\_name, Emp\_address, Date of Birth, and salary. In a database, all the information on all employees would be held in a single storage "container", called a *table*. This table is a tabular object like a spreadsheet page, with different employees as the rows, and the facts (e.g. their names) as columns. Let's call this table EMP, and it could look something like:

Table EMP

Emp_name	Emp_address	Date of Birth	Salary
SmritiKumari	B-15 ,ParkAvenue Delhi	1/3/1986	71000
Samishtha Dutta	D- 45 ,Sector 37,Delhi	7/9/1989	93000
Rahul Sachdeva	C- 6 ,sector 45 Noida	3/2/1982	78000

Fig 16.1(e) Schema of Employee Database

From this information, the *Schema* would define that EMP has four components, "Emp\_name", "Emp\_address", "Date of Birth", "Salary". As database designers, we can call the columns what we like, a meaningful name helps. In addition to the name, we want to try and make sure that people don't accidentally store a name in the DOB column, or some other silly error. We can say things like:

- Emp\_name is a string, holding a minimum of 12 characters.
- Emp\_address is a string, holding a minimum of 12 characters
- Date of Birth is a date. The company can put validation for age must be greater than 18 and less than 60 years.
- A salary is a number. It must be greater than zero.

Such rules can be enforced by a database. The three schema architecture separates the user application and physical database. **The internal level schema or Physical Schema or internal Schema:** determines the physical storage structure of the database. **The conceptual Schema** is the high-level description of the whole database. It hides the detail of the physical storage and focuses on the data types, relationships user operations and constraints. **The external view or is the user view** that describes the view of different user groups.

An eminent scientist E.F.Codd proposed a database model, based on the relational structure of data. In this model data is organized in the form of tables, which is a collection of records and each record in a table consists of fields. A relational database allows the definition of data structures, storage and retrieval operations and integrity constraints. It is proved to be the most effective way of storing data.

**Database Design in Relational Model**

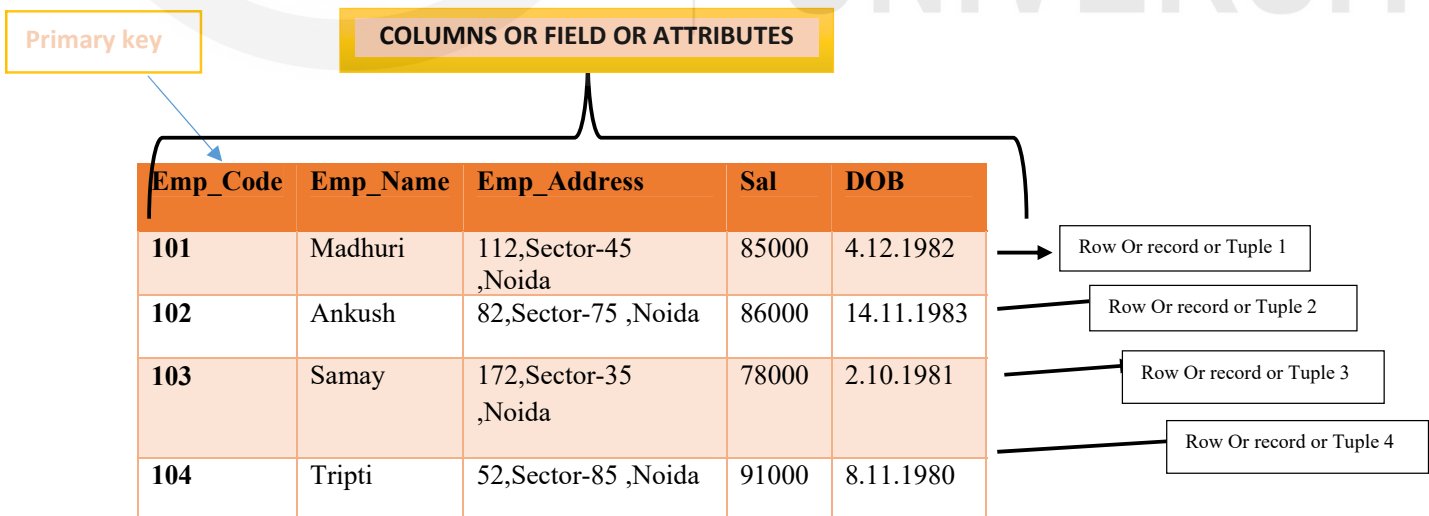
- **Table:** A table is a set of data elements that is organized using a model of vertical columns and horizontal rows. Each row is identified by a unique key index or the key field.
- **Columns/Field/Attributes:** A column is a set of data values of a particular simple type, one for each row of the table. For eg. Emp\_Code, Emp\_Name, Emp\_Address etc.
- **ROWS/ RECORDS /TUPLES:** A row represents a single, data item in a table. Each row in a table represents a set of related data, and every row in the table has the same structure.
- **DATA TYPES:** -Data types are used to identify the type of data we are going to store in the database.

In the relational model, every tuple must have a unique identification or key based on the data. In this figure, an employee code (Emp\_code) is the key that uniquely identifies each tuple in the relation and declares as a primary key. Often, keys are used to join data from two or more relations based on matching identification.

**Primary Key:** A primary key is a unique value that identifies a row in a table. These keys are also indexed in the database, making it faster for the database to search a record. All values in the primary key must be unique and NOT NULL, and there can be only one primary key for a table.

**Foreign Key:** The foreign key identifies a column or set of columns in one (referencing) table that refers to a column or set of columns in another (referenced) table.

**TABLE EMPLOYEE**



**Fig 16.1(f) Components of Relational Database**

The relational model indicates that each row in a table is unique. If you allow duplicate rows in a table, then there's no way to uniquely address a given row via programming. This creates all sorts of ambiguities and problems that are best avoided. You guarantee uniqueness for a table by designating a primary key—a column that contains unique values for a table. Each table can have only one primary key, even though several columns or combination of columns may contain unique values.

All columns (or combination of columns) in a table with unique values are referred to as candidate keys, from which the primary key must be drawn. All other candidate key columns are referred to as alternate keys. Keys can be simple or composite. A simple key is a key made up of one column, whereas a composite key is made up of two or more columns.

The relational model also includes concepts of foreign keys, which are primary keys in one relation, that are kept as non-primary key in another relation, to allow for the joining of data.

Now let us see how to choose the primary key in the table. Consider the Product table given below:

Relating to the employee Table presented in the last section, Now define a second table, order as shown in the figure provided below:

**PRODUCT TABLE**

Product_ID	EmpCode	Order_Date
P101	101	5/1/2020
P201	102	10/4/2020
P301	103	15/5/2020
P401	104	20/6/2020

Foreign Key in Product Table

Fig 16.1(g) Demo table of Foreign Key

Product\_ID is the PrimaryKey in above table Product. Emp\_code is considered a FOREIGN KEY in ProductTable, since it can be used to refer to given Product in the Product table.

### COMMANDS TO DEFINE STRUCTURE AND MANIPULATE THE DATA

In a database, we can define the structure of the data and manipulate the data using some commands. The most common data manipulation language is SQL. SQL commands are instructions used to communicate with the database to perform a specific task that works 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.
- Some of the commonly used datatypes in SQL AND MYSQL are listed below:

Data type	Description
CHAR(n)	Character string, fixed length n ,with a maximum size of 2000 characters . Values of this data type must be enclosed in single quotes " .the storage size of the char value is equal to the maximum size for this column i.e. nColumns , that will not be used for arithmetic operations usually are assigned data types of char.
CHARACTER ,VARYING(n) or VARCHAR(n) ,VARCHAR2(n)	Variable length character string , can store upto 4000 characters . varchar is a variable-length data type, the storage size of the varchar value is the actual length of the data entered, not the maximum size for this column i.e .n
INT	A normal-sized integer that can be signed or unsigned. If signed, the allowable range is from -2147483648 to 2147483647.
DATE	This data type allows to store valid date type data from January 1, 4712 BC to December 31, 4712 AD with standard oracle data format DD-MM-YY.
TIME	Stores the time in a HH:MM:SS format.
TIMESTAMP	A timestamp between midnight, January 1 <sup>st</sup> , 1970 and sometime in 2037. This looks like the previous DATETIME format, only without the hyphens between numbers; 3:30 in the afternoon on December 30 <sup>th</sup> , 1973 would be stored as 19731230153000 (YYYYMMDDHHMMSS ).

## SQL COMMANDS

### DDL COMMANDS[CREATE, ALTER,DROP]

- **CREATE**

It is the mostly used, Data definition command in SQL . It is used to create table index or view. This command describes the layout of the table. The create statement specifies the name of the table , names and types of each column of the table . Each table must have at least one column. The general syntax for creating table is shown below:

**Syntax for CREATE TABLE is:**

```
CREATE TABLE <table name>
    (<attribute name><data type>[<size>]<column constraint>,
    (<attribute name><data type>[<size>]<column constraint>,
    .....))
```

where

**<table name >** : It is the name, of the table to be created

**<attribute name >** : It is the name of the column heading or the field in a table.

**<data Type>** : It defines the type of values that can be stored in the field or the column of the table .

Example:

```
CREATE TABLE Student
(    roll_no number (5),
    name char(20),
    birth_data date );
```

- **ALTER**

Alter is a DDL command in SQL which is used to perform the following operations in table.

1. Add column to the existing table
4. To modify the column of the existing table.

- **Adding column(s) to a table**

To add a column to an existing table, the ALTER TABLE syntax is:

```
ALTER TABLE table_name
(ADD column1_name column-definition);
```

Example :

```
ALTER TABLE PRODUCT ADD (SALES varchar2 (50));
```

This will add a column called *sales* to the *product* table.

- **Modifying column(s) in a table**

To modify a column in an existing table, the ALTER TABLE syntax is:

```
ALTER TABLE table_name
MODIFY (column1_name column_type ,
[column2_name column_type,]
.....);
```

Example:

```
ALTER TABLE PRODUCT
MODIFY Prod_name varchar2(100) not null;
```

This will modify the column called *prod\_name* to be a data type of *varchar2(100)* and force the column to not allow null values.

- **Drop column(s) in a table**

To drop a column in an existing table, the ALTER TABLE syntax is:

```
ALTER TABLE table_name
DROP COLUMN column_name;
```

Example:

```
ALTER TABLE supplier DROP COLUMN Prod_name;
```

This will drop the column called *prod\_name* from the table called *Product*.

#### DML COMMANDS –[INSERT,UPDATE,DELETE]

- INSERT Statement is used to add new rows of data to a table. The value of each field of the record is inserted in the table.

Syntax :

```
INSERT INTO <table _name> [(col1, col2, col3,...colN)]
VALUES (value1, value2, value3,...valueN);
where col1, col2,...colN -- the names of the columns in the table into which you
want to insert data .
```

Example: If you want to insert a row to the employee table, the query would be like,

```
INSERT INTO employee (emp_id, emp_name, emp_dept, age, salary )
```

```
VALUES (105, 'Vaishnavi', 'Medical', 27, 33000);
```

NOTE: When adding a row, only the characters or date values should be enclosed with single quotes.

- **SQL SELECT Statement**

The most commonly used SQL command is SELECT statement. The SQL SELECT statement, the only data retrieval in SQL, used to query or retrieve data from a table in the database.

Syntax of SQL SELECT Statement:

```
SELECT [distinct] column_list FROM table-name
[WHERE Clause]
[GROUP BY clause]
[HAVING clause]
[ORDER BY clause];
```

- *table-name* is the name of the table from which the information is retrieved.
- *column\_list* includes one or more columns from which data is retrieved.
- The code within the brackets are optional.

Example : Write the query in SQL to perform the following in the given Table Student.

R.No	Name	Course	Age	Stream	Sports	Marks	Grade
11	Rajat	BCA	15	Science	Cricket	78	B
12	Anuja	MCA	16	Science	Football	80	B
13	Munil	BCA	15	Science	Cricket	78	B
14	Sonia	MCA	16	Humanities	Badminton	95	A
15	Adya	MCA	15	Commerce	Chess	96	A

- Display the name of all students from the table Student .
- Display the name of students who are in course 'MCA'
- Display the name of students who play 'Chess'
- Display the name of students according to the marks obtained in descending order.

Solution

- SELECT \* FROM student;**  
**SELECT ALL FROM student;**
- SELECT \* FROM student where course='MCA';**
- SELECT \* FROM student where sports='Chess';**
- SELECT \* FROM student ORDERBY Marks Desc;**

Note: SQL is case insensitive

- **UPDATE**

Update statement is used to modify the existing data of the tables .

```
UPDATE table_name
SET column1=value, column2=value2,...
WHERE <condition>
```

**Remember :** The *WHERE* clause in the *UPDATE* syntax specifies which record or records that should be updated. If you remove the *WHERE* clause, all records will be updated by default.

Example :Modify the marks of Anuja from 80 to 87 which was entered wrongly.

```
UPDATE student
SET marks=87
WHERE name='Anuja' and marks='80';
```

- **DELETE Statement**

The DELETE statement allows you to delete a single record or multiple records or all records from the table.

Syntax:

```
DELETE FROM table
WHERE <condition>;
```

**Example :**

```
DELETE FROM Student WHERE course = 'MCA';
```

This would delete all records from the Student table where the course opted by students is MCA

- **DCL (Data Control Language)**

DCL includes commands such as GRANT and REVOKE which mainly deals with the rights, permissions and other controls of the database system.

Examples of DCL commands:

- GRANT-gives user's access privileges to database.
- REVOKE-withdraw user's access privileges given by using the GRANT command.

- **TCL (transaction Control Language)**

TCL commands deals with the transaction within the database.

Examples of TCL commands:

- COMMIT– commits a Transaction.
- ROLLBACK– rollbacks a transaction in case of any error occurs.
- SAVEPOINT–sets a savepoint within a transaction.
- SET TRANSACTION–specify characteristics for the transaction.

**CHECK YOUR PROGRESS:**

1. Define the term database. List its four important features.
2. What do you understand by DDL and DML?
3. How will you explain the term entity and attributes in the table?
4. What is Primary Key?

---

## 16.3 CREATING DATABASE

---

The Python standard for database interfaces is the Python DB-API. Most Python database interfaces follow this standard. Python Database API supports a wide range of database servers such as **MySQL**, PostgreSQL, Microsoft SQL Server 2000, Informix, Interbase, Oracle, Sybase. As a data scientist, you have the freedom to choose the appropriate server for our project. To do so, we need to download a separate DB API module for each database you need to access. For example, if you need to access an Oracle database as well as a MySQL database, you must download both the Oracle and the MySQL database modules.

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

- Importing the API module.
- Acquiring a connection with the database.

- Issuing SQL statements and stored procedures.
- Closing the connection

ANACONDA  
JUPYTER /SPYDER  
NOTEBOOK



MYSQL SERVER  
5.1.33

Fig: shows all applications required to install for setting up database connectivity in python with mysql.

#### Steps for setting python In Anaconda for Database connectivity

1. Install **Anaconda** — <https://www.anaconda.com/distribution/>
2. Select **python version 3.7 (preferably)**
3. MySQL Server (any version: we have used 5.1.33)
4. Mysql.connector or MySQLdb

NOTE: Anaconda is a python and R distribution that aims to provide everything you need:

- core python language,
- python packages,
- IDE/editor — Jupyter and Spyder
- Package manager — Conda, (for managing your packages)

Once it is done, we are ready to have database connectivity in python.

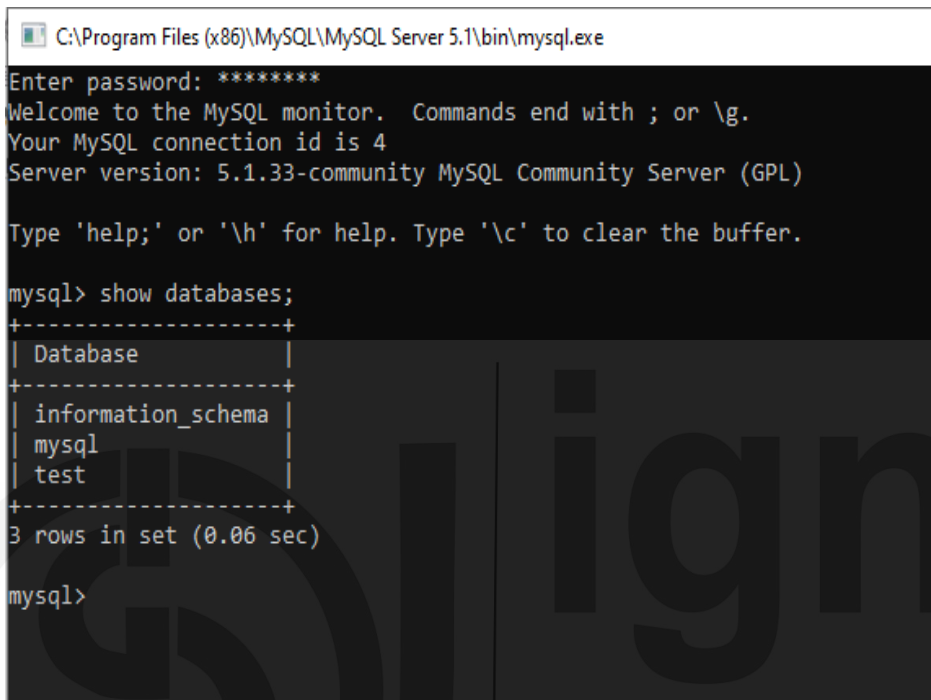
#### Steps to install MySql Setup

1. Download and install **MySql Community server** — <https://downloads.mysql.com/archives/community/>
2. During the installation setup, you will be prompted for a "root" password in the server configuration step.
3. Launch workbench, at the home page, setup a new connection profile with the configuration (Connection method: Standard (TCP/IP), Hostname:

127.0.0.1,Port:3306,Username: root, Password: *yourpassword*) and test your connection.

4. Double click on your local instance and it should bring you the schemas view where you can see all your databases and tables.

The following screenshot gives a glimpse of the screen you may get:



```
C:\Program Files (x86)\MySQL\MySQL Server 5.1\bin\mysql.exe
Enter password: *****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 4
Server version: 5.1.33-community MySQL Community Server (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| mysql      |
| test      |
+-----+
3 rows in set (0.06 sec)

mysql>
```

### **Mysql.connector**

It is an interface for connecting to a MySQL database server from python. It implements the Python Database API v2.0 and is built on top of the MySQL C API. It consist of four methods which are used to establish connection. The following methods are listed below:

- **Steps to install mysql.connector in Anaconda :**

To install mysql.connector navigate to the anaconda prompt and type the following command

1. Using terminal and **conda** to download

```
conda install -c anaconda mysql-connector-python
```

Another way to install mysql.connector is on windows command prompt for using Python d

1. Download Mysql API, exe file and install it.(click here to download)
2. Install mysql-Python Connector (Open command prompt and execute command)

```
>pip install mysql-connector
```

3. Now connect Mysql server using python
4. Write python statement in python shell import mysql.connector

If no error message is shown means mysql connector is properly installed. Mysqlconnector has four important methods which is used to establish and retrieve records from the database.

1. **connect()** : This method is used for creating a connection to our database it have four arguments:
  - a. Host Name
  - b. Database User Name
  - c. Password
  - d. Database Name
2. **cursor()** : This method creates a cursor object that is capable for executing sql query on database.
3. **execute ()**: This method is used for executing SQL query on database. It takes a sql query (as string) as an argument.
4. **fetchone()** : This method retrieves the next row of a query result set and returns a single sequence, or none if no more rows are available.
5. **close ()**: This method close the database connection.

➤ In this chapter, we will be using Anaconda Jupyter notebook for demonstration of all programs

### Connect to MySql

1. Launch Anaconda-Navigator to bring you a list of applications available to install. The application we are interested in is **Jupyter Notebook** which is a web-based python IDE. Install and launch it.
2. In the notebook, create a new python file. In the first cell, write the following code to test the mysql connection.

```
importmysql.connector
```

```
mydb = mysql.connector.connect(  
    host="localhost",  
    user="root",  
    passwd="password",charset='utf8'  
)
```

```
print(mydb)
```

Optional required only when this error comes:  
**ProgrammingError:**  
1115 (42000): Unknown character set: 'utf8mb4'

3. If successful, you should get an object returned with its memory address

```
<mysql.connector.connection_cext.CMySQLConnection object at 0x10b4e1320>
```



**Cursor object:** The `MySQLCursor` class instantiates objects that can execute operations such as SQL statements. Cursor objects interact with the MySQL server using a `MySQLConnection` object.

### How to create a cursor object and use it import `mysql.connector`

Example

Following is the example of connecting with MySQL and display the version of the database.

```
importmysql.connector

# Open database connection

mydb = mysql.connector.connect (
    host="localhost",
    user="root",
    passwd="admin123",charset='utf8'
)

# prepare a cursor object using cursor() method
mycursor=mydb.cursor()

# execute SQL query using execute() method.
mycursor.execute("SELECT VERSION()")

# Fetch a single row using fetchone() method.
data=mycursor.fetchone()

print ("Database version :",data)

# disconnect from server
mydb.close()
```

While running this script, it is producing the following result in my machine. (It could change in your system)

```
Database version : ('5.1.33-community',)
```

In the above code we are creating a cursor to execute the sql query to print database version next line executes the sql query show databases and store result in mycursor as collection ,whose values are being fetched in data. On execution of above program cursor will execute the query and print version of database shown.

### Creating Database

1. Let's create a database called `TEST_DB` IN Python Anaconda .

```
importmysql.connector

mydb = mysql.connector.connect(
    host="localhost",
    user="root",
    passwd="admin123",charset='utf8'
)
mycursor = mydb.cursor()
mycursor.execute("CREATEDATABASETEST_DB")
```

**Next, we will try to connect to this new database.**

```
importmysql.connector

mydb = mysql.connector.connect(
    host="localhost",
    user="root",
    passwd="admin123",charset='utf8',database= TEST_DB
)
```

### CHECK YOUR PROGRESS

1. How will you create a database in mysql .Write the command to create a database named as Start\_db.
2. Name the four arguments required to set connection with mysql.connector.connect.

---

## 16.4 QUERYING DATABASE

---

In this section, you will learn to execute SQL queries at run time of the Anaconda environment. To perform SQL queries you need to create a table in the database and then operations like insert, select, update and delete are shown with examples.

- **Creating Table**

Once a database connection is established, we are ready to create tables or records into the database tables using the **execute** method of the created cursor.

#### Example

To create table EMPLOYEE in database TEST\_DB with following data FIRST\_NAME, LAST\_NAME, AGE, SEX, INCOME.

```

import mysql.connector
mydb = mysql.connector.connect(
    host="localhost",
    user="root",
    passwd="admin123", charset='utf8', database="TEST_DB"
)
# prepare a cursor object using cursor() method
cursor = mydb.cursor()
# Drop table if it already exist using execute() method.
cursor.execute("DROP TABLE IF EXISTS EMPLOYEE")
# Create table as per requirement
sql = """CREATE TABLE EMPLOYEE (
    FIRST_NAME CHAR(20) NOT NULL,
    LAST_NAME CHAR(20),
    AGE INT,
    SEX CHAR(1),
    INCOME FLOAT )"""
cursor.execute(sql)
#Get database table
cursor.execute("SHOW TABLES")
for table in cursor:
    print(table)
# disconnect from server
mydb.close()

```

```
('employee',)
```

- **How to change table structure/(add,edit,remove column of a table) at run time**

To modify the structure of the table, we just have to use alter table query.

Below program will add a column address in the EMPLOYEE table.

```

import
mysql.connector.mydb=mysql.connector.connect(host="localhost",user="root",
passwd ="root",database="school")
mycursor=mydb.cursor()
mycursor.execute("alter table student add (address varchar(2))")
mycursor.execute("desc employee")
for x in mycursor:
print(x)

```

Above program will add a column marks in the table student and will display the structure of the table

- **INSERT Operation**

Insert is used to feed the data in the table created. It is required when you want to create your records into a database table.

## Example

The following example, executes SQL *INSERT* statement to create a record into EMPLOYEE table

```
import mysql.connector

mydb = mysql.connector.connect(
    host="localhost",
    user="root",
    passwd="admin123",charset='utf8',database="TEST_DB"
)

# prepare a cursor object using cursor() method
cursor = mydb.cursor()
# Prepare SQL query to INSERT a record into the database.

sql = "INSERT INTO EMPLOYEE(FIRST_NAME, LAST_NAME, AGE, SEX, INCOME) VALUES('CHINMAY', 'SWAMI', 25, 'M', 50000)"

cursor.execute(sql)
mydb.commit()
print(cursor.rowcount, "Record Inserted")
```

1 Record Inserted

- **UPDATE OPERATION**

UPDATE Operation on any database means to update one or more records, which are already available in the database. Here is an example to show where we run the query to updates all the records having SEX as 'M'. Here, we increase the AGE of all the males by one year.

## Example

```
import mysql.connector
mydb=mysql.connector.connect(host="localhost",user="root",
                             passwd ="admin123",database="TEST_DB",charset='utf8')
mycursor=mydb.cursor()

# Prepare SQL query to UPDATE required records
sql = "UPDATE EMPLOYEE SET AGE = AGE + 1 WHERE SEX =('M')"
try:
    # Execute the SQL command
    mycursor.execute(sql)
    # Commit your changes in the database
    mydb.commit()
except:
    # Rollback in case there is any error
    mydb.rollback()

# disconnect from server
mydb.close()
```

The above code will update the age of all the male employees in the table EMPLOYEE.

- **DELETE Operation**

DELETE operation is required when you want to delete some records from your database. Following example show the procedure to delete all the records from EMPLOYEE where AGE is more than 20.

```

import mysql.connector
mydb=mysql.connector.connect(host="localhost",user="root",
                             passwd="admin123",database="TEST_DB",charset='utf8')
mycursor=mydb.cursor()

# Prepare SQL query to UPDATE required records
sql = "DELETE FROM EMPLOYEE WHERE AGE > 20"
try:
    # Execute the SQL command
    mycursor.execute(sql)
    # Commit your changes in the database
    mydb.commit()
except:
    # Rollback in case there is any error
    mydb.rollback()

# disconnect from server
mydb.close()

```

The above code will delete all the records from the table EMPLOYEE where age is greater than 20.

### Steps to perform SQL queries in the database using Python

Step1. Import mysql.connector

Step 2. Create a variable and assign it to mysql.connector.connect().

Step3. Provide arguments host,user,passwd,database, charset to mysql.connector.connect().

Step4. Call cursor() and assign it to variable.

Step5. Prepare SQL query (Insert or Update or delete)

Step 5. Disconnect from the server using close() function.

Please refer to the above examples with their screenshots which will help in better understanding.

### CHECK YOUR PROGRESS:

1. Write the syntax of the following commands in SQL : insert, update, delete

---

## 16.5 USING SQL TO GET MORE OUT OF DATABASE

---

### READ Operation

READ operation on any database means to fetch some useful information from the database.

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

- **fetchall()** – It fetches all the rows in a result set. If some rows have already been extracted from the result set, then it retrieves the remaining rows from the result set.

- **fetchone()** – It fetches the next row of a query result set. A result set is an object that is returned when a cursor object is used to query a table.
- **rowcount** – This is a read-only attribute and returns the number of rows that were affected by an execute() method.
- **fetchall()**

The method fetches all (or all remaining) rows of a query result set and returns a list of tuples. If no more rows are available, it returns an empty list. The following procedure queries all the records from the EMPLOYEE table

```
import mysql.connector

mydb = mysql.connector.connect(
    host="localhost",
    user="root",
    passwd="admin123", charset='utf8', database="TEST_DB"
)

# prepare a cursor object using cursor() method
cursor = mydb.cursor()
# Prepare SQL query to SELECT ALL records| from the database.
sql = "SELECT * FROM EMPLOYEE"

cursor.execute(sql)
record=cursor.fetchall()
for x in record:

    print(x)

('CHINMAY', 'SWAMI', 25, 'M', 50000.0)
```

- **fetchone()**

To fetch one record from the table fetchone is used in the same manner as fetchall() is shown in above code. This method retrieves the next row of a query result set and returns a single sequence, or None if no more rows are available. By default, the returned tuple consists of data returned by the MySQL server, converted to Python objects.

```
# Fetch a single row using fetchone() method.

record = cursor.fetchone()
```

- **rowcount()**

Rows affected by the query. We can get a number of rows affected by the query by using rowcount. We will use one SELECT query here.

```

import mysql.connector
mydb=mysql.connector.connect(host="localhost",user="root",passwd="admin123",database="TEST_DB
")
mycursor=mydb.cursor()
mycursor = mydb.cursor(buffered=True)
mycursor.execute("select * from employee")
noofrows=mycursor.rowcount
print("No of rows in employee table are",noofrows)

```

In the above code buffered=True. We have used mycursor as buffered cursor which fetches rows and buffers them after getting output from MySQL database. It is used as an iterator, but there is no point in using buffered cursor for the single record as in such case if we don't use a buffered cursor, then we will get -1 as output from rowcount.

- **Manage Database Transaction**

Database transaction represents a single unit of work. Any operation which modifies the state of the MySQL database is a transaction. Python MySQL Connector provides the following method to manage database transactions.

- **Commit** – MySQLConnection.commit() method sends a COMMIT statement to the MySQL server, committing the current transaction.
- **Rollback** – MySQLConnection.rollback revert the changes made by the current transaction.
- **autoCommit** – MySQLConnection.auto-commit value can be assigned as True or False to enable or disable the auto-commit feature of MySQL. By default, its value is False.

**Steps to perform commit,rollback and auto-commit.**

- Step 1. Import the MySQL connector python module
  - Step 2. After a successful MySQL connection, set auto-commit to false, i.e., we need to commit the transaction only when both the transactions complete successfully.
  - Step 3. Call the cursor() function .
  - Step4. Execute the queries using a cursor. Execute method.
  - Step 5 .After successful execution of the queries, commit your changes to the database using  
    - a conn.commit() .In case of an exception or failure of one of the queries, you can revert your changes using a conn.rollback()
- We placed all our code in the "try-except" block to catch the database exceptions that may occur during the process.

The following code shows the role of commit, rollback and auto-commit while performing SQL queries in python.

```

try:
conn = mysql.connector.connect(host='localhost',database='TEST_DB', user='root', passwd='admin123')
conn.autocommit = false
cursor = conn.cursor()
sql_update_query = """Update EMPLOYEE set INCOME = 95000 where FIRST_NAME = raghav"""
cursor.execute(sql_update_query)
print ("Record Updated successfully ")
#Commit your changes
conn.commit()
except mysql.connector.Error as error :
print("Failed to update record to database rollback: {}".format(error))
#reverting changes because of exception
conn.rollback()
finally:
#closing database connection.
if(conn.is_connected()):
cursor.close()
conn.close()
print("connection is closed")

```

In the above code if update query is successfully executed then commit() method will be executed otherwise,anexception error part will be executed.Rollback issued to revert of update query if happened due to error.Finally, we are closing cursor as well as connection. Here the purpose of conn. Auto-commit = false is to activate the role of rollback else rollback will not work.

#### CHECK YOUR PROGRESS:

1. What is the purpose of rollback and commit in SQL.
2. Fill in the blank provided :

```

importmysql.connector
mydb = mysql._____.connect(
host="localhost",
_____="myusername",
passwd="mypassword"
)
mycursor = mydb._____( )
mycursor._____("CREATE DATABASE mydatabase")

```

## 16.6 CSV FILES IN PYTHON

As we have studied various operations on databases, the next thing which comes in our way to implement real dataset for machine learning purpose using python. For the same, we should be aware about all the basic types of database connectivity among which the most common type for connecting real dataset is CSV (Comma Separated Values).

CSV files are ordinarily made by programs that handle a lot of information. Itis just like a text file in a human-readable format which is used to store tabular data in a spreadsheet or database.They are a helpful method to send out information from spreadsheets and data sets just as import or use it in different projects. For instance, you may trade the aftereffects of an information mining project to a CSV document and afterwards import that into a spreadsheet to dissect the information, create charts for an introduction, or set up a report for distribution. The separator character of CSV files is called a delimiter.Default delimiter is comma (,) others are tab (\t), (:), (;) etc.



CSV documents are exceptionally simple to work for database connectivity. Any language that underpins text record info and string control (like python) can work with CSV documents straightforwardly.

Some of the features of CSV are highlighted below, which makes it exceptionally useful in the analysis of the large dataset.

- Simple and easy to use.
- Can store a large amount of data.
- He was preferred to import and export format for data handling.
- Each line of the file is a record.
- Each record consist of fields separated by commas(delimiter)
- They are used for storing tabular data in a spreadsheet or database.

Let us see how to import CSV file in python. In this unit, we will be using pandas module in python to import the CSV file. *Pandas* is a powerful Python package that can be used to perform statistical analysis. In this chapter, you'll also see how to use Pandas to calculate stats from an imported CSV file.

### CASE STUDY OF PIMA INDIAN DATASET

The dataset used here is Pima Indian diabetes data for learning purpose. Pima Indian diabetes dataset describes the medical records for Pima Indians and whether or not each patient will have an onset of diabetes within given years.

Fields description follow:

preg = number of times pregnant

plas = Plasma glucose concentration a 2 hours in an oral glucose tolerance test

pres = Diastolic blood pressure (mm Hg)

skin = Triceps skin fold thickness (mm)

test = 2-Hour serum insulin (mu U/ml)

mass = Body mass index (weight in kg/(height in m)<sup>2</sup>)

pedi = Diabetes pedigree function

age = Age (years)

class = Class variable (1: tested positive for diabetes, 0: tested negative for diabetes)

(This dataset can be downloaded from

<https://www.kaggle.com/kumargh/pima-indians-diabetes-csv?select=pima-indians-diabetes.csv>)

It consists of above mentioned eight features, and one class variable and is very commonly used in the research of diabetes. Given below are the steps and the code to import CSV file in python.

Steps to import a CSV file into Python Using Pandas.

Step 1. Import python module pandas.

Step2. Capture the file path where CSV file is stored (don't forget to include filename and file extension ).

Step3. Read the CSV file using read\_csv

Step4. Print the desired file.

Step5 .Run the code to generate the output.

```
import pandas as pd

df = pd.read_csv ('F:\pimaindiansdiabetescsv\pima-indians-diabetes.csv')

print (df)
```

Note: #read the csv file (put 'r' before the path string to address any special characters in the path, such as '\'). Don't forget to put the file name at the end of the path + ".csv".

This path is taken for reference; it will change as per the location of the file.

### Output

```
6 148 72 35 0 33.6 0.627 50 1
0 1 85 66 29 0 26.6 0.351 31 0
1 8 183 64 0 0 23.3 0.672 32 1
2 1 89 66 23 94 28.1 0.167 21 0
3 0 137 40 35 168 43.1 2.288 33 1
4 5 116 74 0 0 25.6 0.201 30 0
... ..
762 10 101 76 48 180 32.9 0.171 63 0
763 2 122 70 27 0 36.8 0.340 27 0
764 5 121 72 23 112 26.2 0.245 30 0
765 1 126 60 0 0 30.1 0.349 47 1
766 1 93 70 31 0 30.4 0.315 23 0

[767 rows x 9 columns]
```

The above output consists of 769 rows and nine columns. The CSV file is just like excel file which consists of data in tabular form arranged as rows and columns. But the columns are without heading. In the next example, column headings are provided using

```
Colnames=[ 'col1 ', 'col2', 'col3', 'col4 ']
```

Any userdefined name

These are the field names or column names

### • Calculate stats using Pandas from an Imported CSV File.

Finally you will learn to calculate the following statistics using the Pandas package:

- Mean
- Total sum

- Maximum
- Minimum
- Count
- Median
- Standard deviation
- Variance

Simply functions for each of the above mentioned stats is available in pandas package which can be easily applied in the following manner.

```
import pandas as pd
colnames=['pre','pgc','dbp','tsf','2hs','bmi','dpf','age','class']
df=pd.read_csv('F:\pimaindiansdiabetescsv\pima-
indiansdiabetes.csv',names=colnames)
mean1 = df['age'].mean()
sum1 = df['age'].sum()
max1 = df['age'].max()
min1 = df['age'].min()
count1 = df['age'].count()
median1 = df['age'].median()
std1 = df['age'].std()
var1 = df['age'].var()

# print block 1
print ('Mean Age: ' + str(mean1))
print ('Sum of Age: ' + str(sum1))
print ('Max Age: ' + str(max1))
print ('Min Age: ' + str(min1))
print ('Count of Age: ' + str(count1))
print ('Median Age: ' + str(median1))
print ('Std of Age: ' + str(std1))
print ('Var of Age: ' + str(var1))
```

Output:

```
Mean Age: 33.240885416666664
Sum of Age: 25529
Max Age: 81
Min Age: 21
Count of Age: 768
Median Age: 29.0
Std of Age: 11.76023154067868
Var of Age: 138.30304589037365
```

Here is a table that summarizes the operations performed in the code:

Variable	Syntax	Description
mean1	df['age'].mean()	Average of all values under the age column.
sum1	df['age'].sum()	Sum of all values under the age column.
max1	df['age'].max	Maximum of all values under the age column.
min1	df['age'].min()	Minimum of all values under the age column.
count1	df['age'].count()	Count of all values under the age column.
median1	df['age'].median()	Median of all values under the age column.

std1	df['age'].std()	Standard deviation all values under the age column.
var1	df['age'].var()	Variance of all values under the age column.

We have learnt how to calculate simple stats using *Pandas* and import any dataset for machine learning purpose.

---

## 16.7 SUMMARY

---

- A database is an organized, logical collection of data. It is designed to facilitate the access by one or more applications programs for easy access and analysis and to minimize data redundancy.
- DBMS is a software system that enables you to store, modify, and extract information from a database. It has been designed systematically so that it can be used by multiple applications and users
- A relational database is a collection of related tables
- An entity is a person place or things or event.
- Tables in the database are entities.
- An attribute is a property of entity. Attributes are columns in the table.
- A relationship is the association between tables in the database.
- Each columns of the table correspond to an attribute of the relation and is named
- Fields : Columns in the table are called fields or attributes.
- Rows of the relation is referred as tuples to the relation . A tuple or row contains all the data of a single instance of the table such as a employee number 201.
- Records : Rows in a table often are called records . Rows are also known as tuples.
- The values for an attribute or a column are drawn from a set of values known as domain.
- Primary key : An attribute or a set of attributes which can uniquely identify each record (tuple) of a relation (table). All values in the primary key must be unique and not Null, and there can be only one primary key for a table.
- Foreign Key : An attribute which is a regular attribute in one table but a primary key in another table.
- Mysql.connector is an interface for connecting to a MySQL database server from python
- **connect()** is a method used for creating a connection to our database.
- Connect has four arguments:hostname,username,password,databasename .
- **fetchall()** – It fetches all the rows in a result set. If some rows have already been extracted from the result set, then it retrieves the remaining rows from the result set.

- **fetchone()** – It fetches the next row of a query result set. A result set is an object that is returned when a cursor object is used to query a table.
- **rowcount** – This is a read-only attribute and returns the number of rows that were affected by an execute() method.
- Commit,rollback and autocommit are database transactions commands.

## 16.8 SOLUTIONS TO CHECK YOUR PROGRESS

### Section 16.2 Solutions

1. Database is a collection of interrelated, organized form of persistent data. It has properties of integrity and redundancy to serve numerous applications. It is storehouse of all important information, not only for large business enterprises rather for anyone who is handling data even at micro level. Its features are
  - Real world entity.
  - Reduced data redundancy.
  - Data consistency.
  - Sharing of Data.
2. DDL and DML (Refer to section 16.2 SQL Commands)
3. Entity and attributes (Refer to section 16.2 **Entity-Relationship Model**)
4. Primary key: A primary key is a unique value that identifies a row in a table. These keys are also indexed in the database, making it faster for the database to search a record. All values in the primary key must be unique and NOT NULL, and there can be only one primary key for a table.

### Section 16.3 Solutions

1. Create a database called *Start\_db* in *Python Anaconda*.

```
import mysql.connector

mydb = mysql.connector.connect(
    host="localhost",
    user="root",
    passwd="admin123", charset='utf8'
)
mycursor = mydb.cursor()
mycursor.execute("CREATEDATABASE Start_db")
```

Next, we will try to connect to this new database.

```
import mysql.connector
mydb = mysql.connector.connect(
    host="localhost",
    user="root",
    passwd="admin123", charset='utf8', database=Start_db)
```

2. Refer to question 1 second part (host,user,passwd,database)

### Section 16.4 Solutions

#### 1. Syntax of Insert :

```
INSERT INTO TABLE_NAME (column1, column2, column3,...columnN)
```

VALUES (value1, value2, value3,...valueN);

**Syntax of Update:**

UPDATE table\_name  
SET column1 = value1, column2 = value2....., columnN = valueN  
WHERE [condition];

**Syntax of Delete:**

DELETE FROM table\_name WHERE [condition];

**Section 16.5 Solutions**

1. Rollback and commit(Refer to section 16.5 Manage Database transaction)

```
2. importmysql.connector
mydb = mysql.connector.connect(
    host="localhost",
    user="myusername",
    passwd="mypassword"
)
mycursor = mydb.cursor()
mycursor.execute("CREATE DATABASE mydatabase")
```

**MULTIPLE CHOICE QUESTIONS**

1. Which SQL keyword is used to retrieve a maximum value?
  - a. MAX
  - b. MIN
  - c. LARGE
  - d. GREATER
2. Which SQL keyword is used to specify conditional search?
  - a. SEARCH
  - b. WHERE
  - c. FIND
  - d. FROM
3. A relation /table is a
  - a. Collection of fields
  - b. collection of records
  - c. collection of data
  - d. collections of tables
4. \_\_\_\_\_ is a collection of interrelated data and a set of program to access those data .
  - a. table
  - b. record
  - c. Database
  - d. Field
5. A row in a table is called as \_\_\_\_\_
  - a. table
  - b. tuple

- c. Database
  - d. Field
6. Each table in a relational Database must have \_\_\_\_\_.
- a. primary key
  - b. candidate key
  - c. Foreign key
  - d. composite key
7. Attribute combinations that can serve as a primary key in a table are
- a. primary key
  - b. candidate key
  - c. Alternate key
  - d. composite key
8. Duplication of data is known as \_\_\_\_\_
- a. Data redundancy
  - b. Data consistency
  - c. data management
  - d. data schema
9. \_\_\_\_\_ is the total number of rows/tuples in the table .
- a. degree
  - b. cardinality
  - c. records
  - d. domain
10. \_\_\_\_\_ is the total number of columns/fields in the table
- a. degree
  - b. cardinality
  - c. records
  - d. domain
11. A collection of fields that contains data about a single entity is called
- a. table
  - b. record
  - c. database
  - d. Field
12. A set of related characters is called as
- a. table
  - b. record
  - c. database
  - d. Field
13. \_\_\_\_\_ is the independence of application programs from the details of the data representation and data storage.
- a. Data redundancy
  - b. Data consistency
  - c. data independence
  - d. data schema
14. The \_\_\_\_\_ is used for creating a connection to the database in python.
- a. connect()
  - b. cursor()

- c. execute()
- d. close()

15. Which of the following module is provided by python to do several operations on the CSV files?

- a. py
- b. xls
- c. csv
- d. os

Ans 1- a 2- b 3- b 4- c 5- b  
6- a 7- b 8- a 9- b 10- a  
11- b 12- d 13- c 14- a 15- c



ignou  
THE PEOPLE'S  
UNIVERSITY