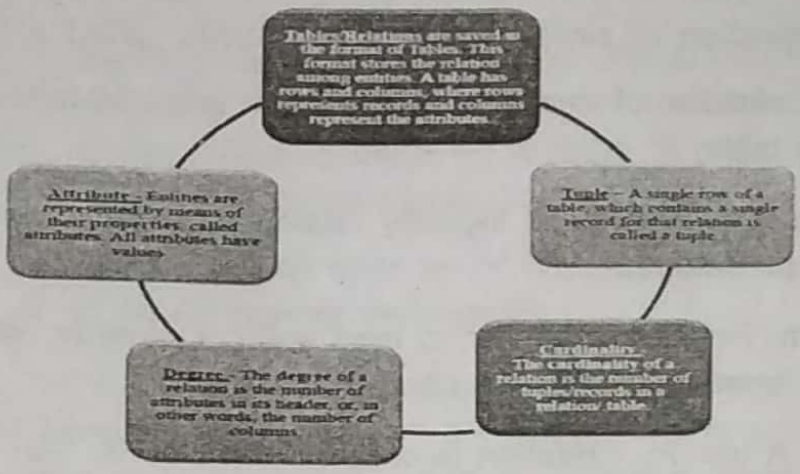
DATABASE MANAGEMENT SYSTEM

SQL COMMANDS AND AGGREGATE FUNCTIONS



Database:

A database system is basically a computer based record keeping system. This information may be necessary to the decision making process involved in the management of that organization. This information is independent of programs using this information and can be serve to multiple applications.

DBMS (Data base management system) : — It refers to a software that is responsible for storing, maintain and utilizing databases. A database along with DBMS is referred to as a database system. e.g., MS Access, Oracle, MySQl, DB2, SQL Server, Mongo DB, Sqlite, PostgreSQL, etc.

Advantage of DBMS (Data Base Management System) over simple file processing system

In typical file processing system, records are stored in various files. A number of different application programs are written for accessing record, adding the record. In typical file processing system there are many limitation and disadvantage like Date redundancy, Data Inconsistency, un-sharable data, unstandardized data, insecure data.

Data Redundancy: Redundancy means same data are duplicated in different places (files).

Data Inconsistency: Data inconsistency occurs when same data maintained in different places do not match.

RDBMS TERMINOLOGY

Field: Set of characters that represents specific data element.

Record: Collection of fields is called a record. A record can have fields of different data types.

File: Collection of similar types of records is called a file.

Table: Collection of rows and columns that contains useful data/information is called a table.

Database: Collection of logically related data along with its description is termed as database.

Relation: Relation (collection of rows and columns) on which we can perform various operations. It is also known as table.

Tuple: A row in a relation is called a tuple. It is also known as record.

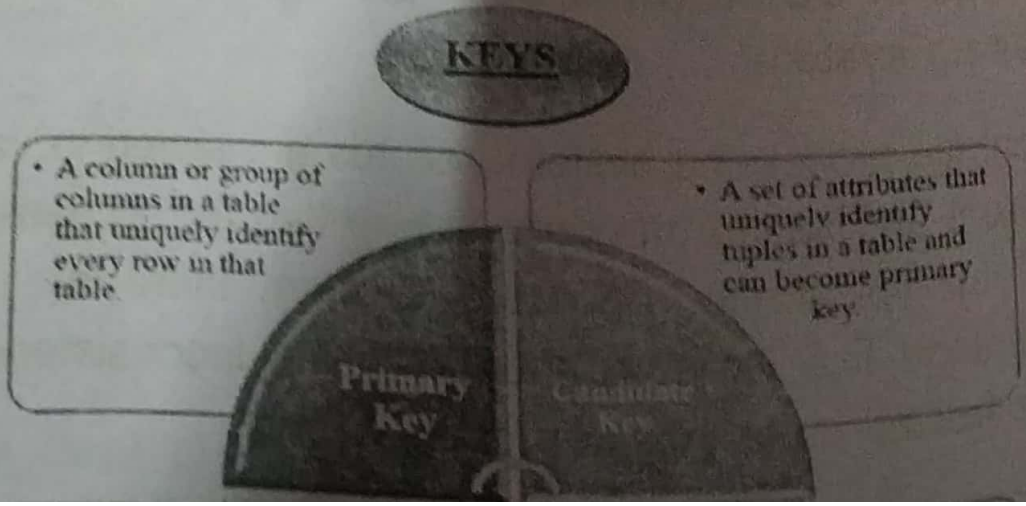
Attribute: A column in a relation is called an attribute. It is also termed as field or data item.

Degree: Number of attributes in a relation is called degree of a relation.

Cardinality: Number of tuples in a relation is called cardinality of a relation.

DOMAIN: It is a set of values from which an attribute can take a value in each row.

Cartesian Product: No of Rows /Tuple are multiply and no. of Columns/ attributes are added when we apply Cartesian Product on two tables.



Candidate Key: A relation can have one or more attributes that takes distinct values. Any of these attributes can be used to uniquely identify the tuples in the relation. Such attributes are called candidate keys as each of them are candidates for the primary key. Attributes (or group of attributes) of a table which can serve as a primary key are called candidate key.

Primary Key: Out of one or more candidate keys, the attribute chosen by the database designer to uniquely identify the tuples in a relation is called the primary key of that relation. Primary key is an attribute or group of attributes that can uniquely identifies the records/tuples in that relation.

Alternate Key: All the candidate keys other than the primary keys of a relation are known alternate keys of that relation.

Foreign Key: Non key attribute of a table acting as primary key in some other table or whose value is derived from the primary key of another table.is known as Foreign Key in its current table. This key is used to enforce referential integrity in RDBMS.

Structured Query Language

SQL is a non procedural language that is used to create, manipulate and process the databases(relations).

Processing Capabilities of SQL

1. **Data Definition Language (DDL)**

   DDL contains commands that are used to create the tables, databases, indexes, views, sequences and synonyms etc.

   e.g: Create table, create database, alter table, drop table etc.

2. **Data Manipulation Language (DML)**

   DML contains commands that can be used to manipulate the data base objects and to query the databases for information retrieval.

   e.g: Select, Insert, Delete, Update etc.

3. **Transaction Control Language (TCL)**

   TCL include commands to control the transactions in a data base system. The commonly used commands in TCL are COMMIT, ROLLBACK and SAVEPOINT.

DDL	DML
It is Data Definition Language	It is Data Manipulation Language
These are used to define data structure	It is used to manipulate the existing databases
It is used to define database structure or scheme	it is used for managing data within scheme objects
Commands are CREATE, ALTER, DROP, TRUNCATE, RENAME	Commands are SELECT, INSERT DELETE, UPDATE, MERGE, CALL
It works on whole table	It works on one or more rows
It do not have a where clause to filter	It have where clause to filter records
Changes done by DDL commands cannot be rolled back	Changes can be rolled back
It is not further classified	It is further classified as procedural and non procedural DML's
Example: droop table tablename:	Select * from employee

Data types of SQL: Just like any other programming language, the facility of defining data of various types is available in SQL also. Following are the most common data types of SQL.

1)	NUMBER	Number(n,d), Number (5,2)
2)	CHAR	CHAR(SIZE)
3)	VARCHAR / VARCHAR2	VARCHAR2(SIZE)
4)	DATE	DATE

Operators in SQL: The following are the commonly used operators in SQL

1.	Arithmetic Operators	+, -, *, /
2.	Relational Operators	=, <, >, <=, >=, <>
3.	Logical Operators	OR, AND, NOT

Constraints: Constraints are the conditions that can be enforced on the attributes of a relation. The constraints come in play whenever we are trying to insert, delete or update a record in a relation.

Not null ensures that we cannot leave a column as null. That is a value has to be supplied for that column.

e.g. name varchar(25) not null

Unique constraint means that the values under that column are always unique.

e.g. Roll_no number(3) unique

Primary key constraint means that a column can not have duplicate values and not even a null value.

e.g. Roll_no number(3) primary key

The main difference between unique and primary key constraint is that a column specified as unique may have null value but primary key constraint does not allow null values in the column.

SQL COMMANDS :

1. Create Database command is used to create a Database. The syntax of this Command is

   CREATE DATABASE <Database Name>

2. To use a database, we can use the USE command: Syntax of this command is-

   USE <database Name>;

3. **Create Table command** is used to create a table . The syntax of this Command is:

   CREATE TABLE <Table_name>
   (column_name 1 data_type [(size) column_constraints],
   column_name 2 data_type [(size) column_constraints],
   :
   :
   [<table_constraint> (column_names)]);

4. **Describe Table Command:** We can view the structure of an already created table using the DESCRIBE statement or DESC statement.

Syntax:

DESCRIBE tablename;

5. **Show Tables Command:** We can use the SHOW TABLES statement to see the tables in a database.

Syntax:

SHOW TABLES;

6. **The ALTER Table command** is used to change the definition (structure) of existing table.

For Add or modify column:-

ALTER TABLE <Table_name> ADD/MODIFY <Column_defnition>;

For Deleting a column

ALTER TABLE <Table_name> DROP COLUMN <Column_name>;

7. **The INSERT Command:** The rows (tuples) are added to a table by using INSERT command. The syntax of Insert command is:

INSERT INTO <table_name> [(<column_list>)] VALUES (<value_list>);

e.g.,

INSERT INTO EMP (empno, ename, sex, sal, deptno) VALUES(1001, 'Ravi', 'M', 4500.00, 10);

If the order of values matches the actual order of columns in table then it is not required to give the column_list in INSERT command. e.g.

INSERT INTO EMP VALUES(1001, 'Ravi', 'M', 4500.00, 10);

8. **The Update command** is used to change the value in a table. The syntax of this command is:

UPDATE <table_name>

SET column_name1=newvalue1/expression

[,column_name2=newvalue2/expression,.....]

WHERE <condition>;

e.g., to increase the salary of all the employees of department No 10 by 10% , then command will be:

UPDATE emp

SET sal=sal*1.1

WHERE Deptno=10;

9. **The DELETE command** removes rows from a table. This removes the entire rows, not individual field values. The syntax of this command is

DELETE FROM <table_name>

[WHERE <condition>];

e.g., to delete the tuples from EMP that have salary less than 2000, the following command is used:

DELETE FROM emp WHERE sal<2000;

To delete all tuples from emp table:

DELETE FROM emp;

10. **The SELECT command** is used to make queries on database. A query is a command that is given to produce certain specified information from the database table(s). The SELECT command can be used to retrieve a subset of rows or columns from one or more tables. The syntax of Select Command is:

SELECT <Column-list>

FROM <table_name>

[WHERE <condition>]

[GROUP BY <column_list>]

[HAVING <condition>]

[ORDER BY <column_list [ASC|DESC]>]

The **select** clause list the attributes desired in the result of a query.

e.g.,To display the names of all Employees in the *emp* relation:

select *ename* **from** *emp;*

To force the elimination of duplicates, insert the keyword **distinct** after select.

Find the number of all departments in the *emp* relations, and remove duplicates

select **distinct** *deptno* from *emp;*

An asterisk (*) in the select clause denotes "all attributes"

SELECT * FROM *emp;*

The **select** clause can contain arithmetic expressions involving the operation, +, –, *, and /, and operating on constants or attributes of tuples. The query:

SELECT *empno, ename, sal* * 12 **FROM** ***emp;***

would display all values same as in the *emp* relation, except that the value of the attribute *sal* is multiplied by 12.

The WHERE clause in SELECT statement specifies the criteria for selection of rows to be returned.

- **Conditions based on a range (BETWEEN Operator):** The Between operator defines a range of values that the column values must fall in to make condition true . The range includes both lower value and upper value.

  e.g., Find the empno of those employees whose salary between 90,000 and 100,000 (that is, 90,000 and 100,000)

  SELECT *empno* **FROM** *emp* **WHERE** *sal* **BETWEEN** 90000 **AND** 100000;

- **Conditions based on a list (IN operator):** To specify a list of values , IN operator is used. IN operator selects values that match any value in a given list of values.

  For example , to display a list of members from 'DELHI', 'MUMBAI', 'CHENNAI' or 'BANGALORE' cities :

  SELECT * FROM members WHERE city IN ('DELHI', 'MUMBAI', 'CHENNAI' , 'BANGALORE') ;

  The **NOT IN** operator finds rows that do not match in the list. So if you write

  SELECT * FROM members WHERE city NOT IN ('DELHI', 'MUMBAI', 'CHENNAI' , 'BANGALORE') ;

  It will list members not from the cities mentioned in the list.

- **Conditions based on Pattern:** SQL also includes a string-matching operator, LIKE, for comparison on character string using patterns. Patterns are described using two special wildcard characters:

→ Percent (%) – '%' matches any substring(one ,more than one or no character).

→ Underscore (_) – '_' character matches exactly one character.

•• Patterns are case-senstive.

•• **Like** keyword is used to select row contaning columns that match a wildcard pattern.

- The keyword **not like** is used to select the row that do not match the specified patterns of characters.

- **Searching for NULL:** The NULL value in a column is searched for in a table using *IS NULL* in the WHERE clause (Relational Operators like =,<> etc can not be used with NULL).

  For example, to list details of all employees whose departments contain NULL (i.e., novalue), you use the command:

  SELECT empno, ename FROM emp Where Deptno **IS NULL**;

ORDER BY Clause: You can sort the result of a query in a specific order using ORDER BY clause. The ORDER BY clause allow sorting of query result by one or more columns. The sorting can be done either in ascending or descending order.

Note: - If order is not specified then by default the sorting will be performed in ascending order.

e.g., Select * from emp order by deptno;

Three methods of ordering data are:

i. Ordering data on single column.
ii. Ordering data on multiple column.
iii. Ordering data on the basis of an expression

- **GROUP BY Clause:** The GROUP BY clause groups the rows in the result by columns that have the same values. Grouping is done on column name. It can also be performed using aggregate functions in which case the aggregate function produces single value for each group.

  HAVING Clause: The HAVING clause place conditions on groups in contrast to WHERE clause that place conditions on individual rows. While WHERE condition cannot include aggregate functions, HAVING conditions can do so.

  e.g.,

  Select avg(sal), sum(sal) from emp group by deptno having deptno=10;

  Select job, count(*) from emp group by job having count(*)<3;

11. **The DROP Command :** The DROP TABLE command is used to drop (delete) a table from database. But there is a condition for droping a table ; it must be an empty table i.e. a table with rows in it cannot be dropped.The syntax of this command is :

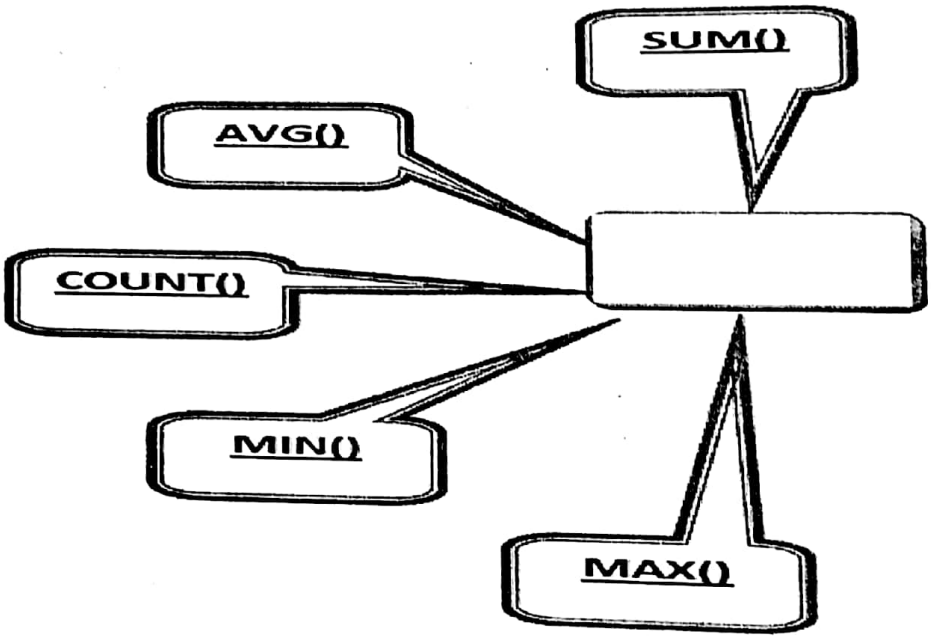
DROP TABLE <Table_name>;

e.g.,

DROP TABLE EMP;

12. **Multiple Row or Group or Aggregate function**

These functions are called aggregate functions because they operate on aggregates of tuples. *These functions return only single value for a group and therefore, they are called* aggregate or group functions.



(i). SUM() :

It returns the sum of values of numeric type of a column.

e.g. Select sum(salary) from employee;

(ii). AVG() :

It returns the average of values of numeric type of a column.

e.g. Select avg(salary) from employee;

(iii). MIN() :

It returns the minimum of the values of a column of a given relation.

e.g. Select min(salary) from employee;

(iv). MAX() :

It returns the maximum of the values of a column of a given relation.

e.g. Select max(salary) from employee;

(v). COUNT():

It returns the number of rows in a relation.

e.g., Select count(*) from employee;

GROUP BY Clause: The GROUP BY clause groups the rows in the result by columns that have the same values. Grouping is done on column name. It can also be performed using aggregate functions in which case the aggregate function produces single value for each group.

e.g., Select deptno, avg (sal), sum (sal) from emp group by deptno

HAVING Clause: The HAVING clause place conditions on groups in contrast to WHERE clause that place conditions on individual rows. While WHERE condition cannot include aggregate functions, HAVING conditions can do so.

e.g., Select deptno, avg (sal), sum (sal) from emp group by deptno having deptno=10;
Select job, count (*) from emp group by job having count (*) 3;

JOIN OPERATION :

MySQL JOINS are used to retrieve data from multiple tables.

A MySQL JOIN is performed whenever two or more tables are joined in a SQL statement.

We have a table called suppliers with two fields (supplier_id and supplier_name). It contains the following data:

Supplier_id	supplier_name
10000	IBM
10001	Hewlett Packard
10002	Microsoft
10003	NVIDIA

We have another table called orders with three fields (order_id, supplier_id, and order_date). It contains the following data:

order_id	supplier_id	order_date
500125	10000	2013/05/12
500126	10001	2013/05/13
500127	10004	2013/05/14

If we run the MySQL SELECT statement (that contains an INNER JOIN) below:

SELECT suppliers.supplier_id, suppliers.supplier_name, orders.order_date FROM

suppliers, orders ON suppliers.supplier_id = orders.supplier_id;

Our result set would look like this:

supplier_id	name	order_date
10000	IBM	2013/05/12
10001	Hewlett Packard	2013/05/13

Note-The rows for Microsoft and NVIDIA from the supplier table would be omitted, since the supplier_id's 10002 and 10003 do not exist in both tables. The row for 500127 (order_id) from the orders table would be omitted, since the supplier_id 10004 does not exist in the suppliers table.

Types of Join :

(i) **Equi-Join**-The join in which column are compared for equality.

Syntax-

```
Select * from tablename1, tablename2
    where tablename1. columnname = tablename2.column ;
```

e.g. .

```
Select * from Student, Teacher where Student.ID=Teacher.ID;
```

(ii) **Natural Join:** The Join in which only one of the identical columns (coming from joined tables)

e.g., Select student.* , phy,chem from student,marks;

9. **Query Based on Two table (Join):**

```
SELECT <Column-list>
FROM <table_name1>,<table_name2>
WHERE <Join_condition>[AND condition];
```

10. **Qualified Names :**

```
<tablename>.<fieldname>
```

This type of field names are called qualified field names and are used to identifying a field if the two joining tables have fields with same name.

SQL (Question and Answers)(2-Marks)

Q.1 State two advantages of using Databases.

Ans: Databases help in reducing Data Duplication i.e. Data Redundancy and controls Data Inconsistency.

Q2. *Name some popular relational database management systems.*

Ans: Oracle, MYSQL, Sqlite, MS-Access etc

Q.3 *Define – Relation, Tuple, Degree, Cardinality*

Ans: A Relation is logically related data organized in the form of tables.

Tuple indicates a row in a relation. Degree indicates the number of Columns. Cardinality indicates the number of Rows.

Q4. Differentiate between Char and Varchar.

Ans: Char means fixed length string and Varchar means variable length character string. e.g., For the data "Computer" char (30) reserves constant space for 30 characters whereas Varchar (30) reserves space for only 8 characters.

Q.5 What is a Primary Key?

Ans: A Primary Key is a set of one or more attributes (columns) of a relation used to uniquely identify the records in relation.

Q.7 *What is a Foreign Key? What is its use?*

Ans: A Foreign key is a non-key attribute of one relation whose values are derive from the primary key of some other relation. It is used to join two / more relations and extract data from them.

Q.8 Write SQL statements to do the following

(a) Create a table Result with two columns Roll and Name with Roll as primary key

```
CREATE TABLE Result
(Roll INT PRIMARY KEY, Name Varchar (30)) ;
```

(b) Add a column Marks to Result table

```
ALTER TABLE Result ADD (Marks DECIMAL (10,2));
```

(c) Update marks of,"Raj" to 80

```
UPDATE Result
SET Marks = 80 WHERE
Name="Raj";
```

(Query -Based Questions 3/4 Marks)

Q.1. Write SQL queries for (i) to (v), which are based on the table: **SCHOOL and ADMIN**

TABLE: SCHOOL

CODE	TEACHERNAME	SUBJECT	DOJ	PERIODS	EXPERIECE
1001	RAVI SHANKAR	ENGLISH	12/03/2000	24	10
1009	PRIYA RAI	PHYSICS	03/09/1998	26	12
1203	LISA ANAND	ENGLISH	09/04/2000	27	5
1045	YASHRAJ	MATHS	24/08/2000	24	15
1123	GANAN	PHYSICS	16/07/1999	28	3
1167	HARISH B	CHEMISTRY	19/10/1999	27	5
1215	UMESH	PHYSICS	11/05/1998	22	16

TABLE: ADMIN

CODE	GENDER	DESIGNATION
1001	MALE	VICE PRINCIPAL
1009	FEMALE	COORDINATOR
1203	FEMALE	COORDINATOR
1045	MALE	HOD
1123	MALE	SENIOR TEACHER
1167	MALE	SENIOR TEACHER
1215	MALE	HOD

i) To decrease period by 10% of the teachers of English subject.

ii) To display TEACHERNAME, CODE and DESIGNATION from tables SCHOOL and ADMIN whose gender is male.

iii) To Display number of teachers in each subject.

iv) To display details of all teachers who have joined the school after 01/01/1999 in descending order of experience.

v) Delete all the entries of those teachers whose experience is less than 10 years in SCHOOL table.

Answer:

i) update SCHOOL set PERIODS=0.9*PERIODS;

ii) select SCHOOL.TEACHERNAME, SCHOOL.CODE, ADMIN.DESIGNATION from SCHOOL, ADMIN where gender='MALE'.

iii) select SUBJECT, count(*) from SCHOOL group by SUBJECT;

iv) select * from SCHOOL where DOJ>' 01/01/1999' order by EXPERIENCE desc;

v) delete from SCHOOL where EXPERIENCE<10;

Q.2. A Watch Store is considering maintaining their inventory using SQL to store the data. One table is given below with its structure:

TABLE: WATCHES

Watchid	Watch_Name	Type	QTY_State
W001	High Time	Unisex	100
W002	Life Time	Ladies	150
W003	Wave	Gents	200
W004	High Fashion	Unisex	250
W005	Golden Time	Gents	100

(i) Identify the attribute best suitable to be declared as a primary key.

(ii) Write the degree and cardinality of the table Watches.

(iii) Write a SQL command to display all the details of those watches whose type is Unisex.

(iv) Write a SQL command to display name of watches whose quantity is greater than 150.

(v) Write a SQL command to display details of all Gents.

Answer:

(i) Watchid

(ii) Degree = 4, Cardinality = 5

(iii) select * from Watches where type='Unisex';

(iv) select Watch_Name from Watches where Qty_Store > 150;

(v) select * from Watches where type='Gents';

Case Study based Questions:

Rohit Confused in the following two table / relation

1. Book_Information and 2. Sales.

Please assist him find the correct answers-

Table: Book_Information	Table: Sales
Column Name	Column Name
Book_ID	Store_ID
Book_Title	Sales_Date
Book_Table	Sales_Amount
Price	

(i) Which SQL statement allows you to find the highest price from the table Book_Information?

(a) SELECT Book_ID,Book_Title,MAX(Price) FROM Book_Information;

(b) SELECT MAX(Price) FROM Book_Information;

(c) SELECT MAXIMUM(Price) FROM Book_Information;

(d) SELECT Price FROM Book_Information ORDER BY Price DESC ;

Ans. (*b*) SELECT MAX(Price) FROM Book_Information;

(ii) Which SQL statement allows you to find sales amount for each store?

(a) SELECT Store_ID, SUM(Sales_Amount) FROM Sales;

(b) SELECT Store_ID, SUM(Sales_Amount) FROM Sales ORDER BY Store_ID;

(c) SELECT Store_ID, SUM(Sales_Amount) FROM Sales GROUP BY Store_ID;

(d) SELECT Store_ID, SUM(Sales_Amount) FROM Sales HAVING UNIQUE Store_ID;

Ans. (*c*) SELECT Store_ID, SUM(Sales_Amount) FROM Sales GROUP BY Store_ID;

(iii) Which SQL statement lets you to list all store name whose total sales amount is over 5000 ?

(a) SELECT Store_ID, SUM(Sales_Amount) FROM Sales GROUP BY Store_ID HAVING SUM(Sales_Amount) > 5000;

(b) SELECT Store_ID, SUM(Sales_Amount) FROM Sales GROUP BY Store_ID HAVING Sales_Amount > 5000;

(c) SELECT Store_ID, SUM(Sales_Amount) FROM Sales WHERE SUM(Sales_Amount) > 5000 GROUP BY Store_ID;

(d) SELECT Store_ID, SUM(Sales_Amount) FROM Sales WHERE Sales_Amount > 5000 GROUP BY Store_ID;

Ans. (*a*) SELECT Store_ID, SUM(Sales_Amount) FROM Sales GROUP BY Store_ID HAVING SUM(Sales_Amount) > 5000;

(iv) Which SQL statement lets you find the total number of stores in the SALES table?

(a) SELECT COUNT(Store_ID) FROM Sales;

(b) SELECT COUNT(DISTINCT Store_ID) FROM Sales;

(c) SELECT DISTINCT Store_ID FROM Sales;

(d) SELECT COUNT(Store_ID) FROM Sales GROUP BY Store_ID;

Ans. (*d*) SELECT COUNT(Store_ID) FROM Sales GROUP BY Store_ID;

(v) Which SQL statement allows you to find the total sales amount for Store_ID 25 and the total sales amount for Store_ID 45?

(a) SELECT Store_ID, SUM(Sales_Amount) FROM Sales WHERE Store_ID IN (25, 45) GROUP BY Store_ID;

(b) SELECT Store_ID, SUM(Sales_Amount) FROM Sales GROUP BY Store_ID HAVING Store_ID IN (25, 45);

(c) SELECT Store_ID, SUM(Sales_Amount) FROM Sales WHERE Store_ID IN (25,45);

(d) SELECT Store_ID, SUM(Sales_Amount) FROM Sales WHERE Store_ID = 25 AND Store_ID =45 GROUP BY Store_ID;

Ans. (*b*) SELECT Store_ID, SUM(Sales_Amount) FROM Sales GROUP BY Store_ID HAVING Store_ID IN (25, 45);

INTERFACE PYTHON WITH MYSQL

- **Database connectivity**-Database connectivity refers to connection and communication between an application and a database system.
- **Mysql.connector**-Library or package to connect from python to MySQL.
- Command to install connectivity package:- pip install mysql-connector-python
- Command to import connector:- import mysql.connector

Steps for writing the python MySQL connectivity program:

1. Import the packages required (i.e., mysql.connector):

   import mysql.connector

   or

   import mysql.connector as sc

2. Open a connection to MySQL database:

   <connection_object> =mysql.connector.connect

   (host="localhost",user=<username>, passwd=<password>

   [,database=<database>])

   Where my<connection_object> is the name of connection object it may be of your choice but a valid identifiers name. where username is the username of MySQL and password is the password of the user in MySQL and database is the name of database in MySQL.

3. Create a Cursor Instance:

<cursor_object>=<connection-object>.cursor()

4. Execute the SQL Query:

<cursor_object>.execute(<SQL query String>)

5. Extract Data from Resultset:

You can extract data from result set using any of the following fetch....() functions.

fetchall()- The fetchall() method will return all the rows from the resultset in the formof tuple.

e.g., data = cursor.fetchall()

fetchone()- The fatchone() method will return only one row from the resultset in the form of a tuple.

e.g., data =cursor.fetchone()

6. Clean up the Environment:

After you are completed all processing , in the final step you have to close the connection established.

<connection_object>.close()

rowcount - The rowcount is a property of cursor object that returns the number of rows retrieved from the cursor so far.

,variable> = <cursor_object>.rowcount

is_connected() – You can also check for successful connection using function is_connected() with connection object (which returns True , if connection is successful).

<connection_object>.is_connected()

#CREATE DATABASE

```
import mysql.connector
mydb=mysql.connector.connect(host="localhost",user="root",passwd="12345")
mycursor=mydb.cursor()
mycursor.execute("CREATE DATABASE SCHOOL")
```

 # SHOW DATABASE

```
import mysql.connector
mydb=mysql.connector.connect(host="localhost",user="root",passwd="12345")
mycursor=mydb.cursor()
```

```
mycursor.execute("SHOW DATABASES")
for x in mycursor:
print (x)
# CREATE TABLE
import mysql.connector
mydb=mysql.connector.connect(host="localhost",user="root",passwd="12345",
database="student")
mycursor=mydb.cursor()
mycursor.execute("CREATE TABLE FEES (ROLLNO INTEGER(3),NAME
VARCHAR(20),AMOUNT INTEGER(10));")
# SHOW TABLES
import mysql.connector
mydb=mysql.connector.connect(host="localhost",user="root",passwd="12345",
database="student")
mycursor=mydb.cursor()
mycursor.execute("SHOW TABLES")
for x in mycursor:
print(x)
# DESCRIBE TABLE
import mysql.connector
mydb=mysql.connector.connect(host="localhost",user="root",passwd="12345",
database="student")
mycursor=mydb.cursor()
mycursor.execute("DESC STUDENT")
for x in mycursor:
    print(x)
# SELECT QUERY
import mysql.connector
conn=mysql.connector.connect(host="localhost",user="root",passwd="12345",
database="student")
c=conn.cursor()
c.execute("select * from student")
r=c.fetchone()
```

```
while r is not None:
        print(r)
        r=c.fetchone()
#WHERE CLAUSE
import mysql.connector
conn=mysql.connector.connect(host="localhost",user="root",passwd="12345",
database="student")
if conn.is_connected():
        c=conn.cursor()
c.execute("select * from student where marks>90")
        r=c.fetchall()
count=c.rowcount()
print("total no rows:",count)
for row in r:
        print(row)
else:
        print("Error connecting to MYSQL DATABASE")
# INSERTION
import mysql.connector
mydb=mysql.connector.connect(host="localhost",user="root",passwd="12345",
database="student")
mycursor=mydb.cursor()
mycursor.execute ("insert into student values (%s, %s, %s)",
(10,"ABHAY",97.8))
mydb.commit()
print(mycursor.rowcount,"RECORD INSERTED")
# UPDATE COMMAND
import mysql.connector
mydb=mysql.connector.connect(host="localhost",user="root",passwd="12345",
database="student")
mycursor=mydb.cursor()
mycursor.execute("UPDATE STUDENT SET MARKS=100 WHERE MARKS=40")
```

```
mydb.commit()
print(mycursor.rowcount,"RECORD UPDATED")
# DELETE COMMAND
import mysql.connector
mydb=mysql.connector.connect(host="localhost",user="root",passwd="12345",
database="student")
mycursor=mydb.cursor()
mycursor.execute("DELETE FROM STUDENT WHERE MARKS<50")
mydb.commit()
print(mycursor.rowcount,"RECORD DELETED")
```

Note: With INSERT, UPDATE and DELETE queries (which make changes to the database unlike SELECT) you must run commit( ) method of connection in the end i.e.,

<connection_object>.commit( )

This is necessary to reflect the changes in database.

Q. A resultset is extracted from the database using the cursor object (that has been already created) by giving the following statement.

Mydata=cursor.fetchone()

(a) How many records will be returned by fetchone() method?

(b) What will be the datatype of Mydata object after the given command is executed?

Answer:

(a) One record

(b) tuple

Q. The code given below inserts the following record in the table Student:

RollNo – integer

Name – string

Clas – integer

Marks – integer

Note the following to establish connectivity between Python and MYSQL:

- Username is root
- Password is tiger

- The table exists in a MYSQL database named school.
- The details (Roll no, Name,Clas and Marks) are to be accepted from the user.

* Write the following missing statements to complete the code:

Statement 1 – to create the cursor object

* Statement 2 – to execute the command that inserts the record in the table student.

* Statement 3- to add the record permanently in the database.

```
import mysql.connector as mysql
def sql_data():
con1=mysql.connect(host="localhost",user="root", password="tiger",
database="school")
mycursor= #Statement 1
rno=int(input("Enter Roll Number::"))
name=int(input("Enter Name::"))
clas=int(input("Enter Marks::"))
marks=int(input("Enter Marks::"))
query="insert into student values({},'{}',{},{})".format(rno,name,clas,
marks)
_____ #Statement 2
_____ # Statement 3
print("Data Added Suceesfully")
```

Answer: -

Statement 1:

con1.cursor()

Statement 2:

mycursor.execute(querry)

Statement 3:

con1.commit()