# UNIT 5 DATABASE PROGRAMMING

## PART-A

## SHORT QUESTIONS WITH SOLUTIONS

**Q1. Define database, user interface and components.**

**Answer :**

Model Paper-I, Q1(i)

**Database**

The RDBMS has the ability to manage various databases such as customer support, sales, marketing etc at a time on one server. The MYSQL is an example of server based RDBMS because it consists of server based RDBMS because it consists of server process that is running continuously.

**User Interface**

A command line tool is used for SQL commands and queries. This tool is provided by the database systems. Even the GUI tools make use of command line clients or database client library so that the users are provided with an easy interface.

**Components**

Databases consists of tables as storage abstractions. And every table consists of rows and columns. A data base schema can be defined as set of table definitions of columns and data types together. The databases and tables can be created and deleted as well. It is even possible to perform various operations such as insertion, updation, deletion etc., on them.

**Q2. List out various commands of SQL.**

Model Paper-III, Q1(i)

**Answer :**

SQL provides database commands and queries to the database. Various database commands in SQL are as follows.

**Create Database**

```
CREATE DATABASE SCHOOL;
GRANT ALL ON SCHOOL. * to user(s);
```

It creates a database with name SCHOOL and grants permission to specific users in order to perform database operations.

**Use a Database**

```
USE SCHOOL;
```

It specifies the database that is to be used for performing operations.

**Drop the Database**

```
DROP database SCHOOL;
```

It deletes the database and its tables from the system.

**Create a Table**

```
CREATE TABLE SCHOOL (ID INT, NAME VARCHAR(20));
```

It creates a new table with columns ID and NAME.

**Drop a Table**

```
DROP TABLE SCHOOL;
```

It deletes the table completely along with the data.

**Insert a Row**

```
INSERT INTO SCHOOL VALUES('21', 'JAMES');
```

It inserts a row in the table.

**Update a Row**

```
UPDATE SCHOOL SET NAME = 'HARRY'
        WHERE id = 4;
```

It modifies and updates the values of specified row.

**Delete a Row**

```
DELETE FROM SCHOOL WHERE id = 2;
```

It deletes the data of specified row.

### Q3. Define ORM.

**Answer :** Model Paper-II, Q1(i)

ORM stands for Object-Relational Managers. There are various types of database systems that enable users to maintain Python interfaces to harness their power. But they need atleast some knowledge of SQL. If the programmer can manipulate Python objects rather than SQL queries and need relational database then ORM would be a best option.

These systems are designed in such a way that users can manipulate the objects and perform the tasks. The tables are converted to classes containing features and columns as attributes and methods as database operations. The application can be set to an ORM. Certain operations might be complex and even need more lines of code than using an adapter directly because ORM's perform a lot of work on behalf of users. There are two ORM's that are used more frequently, they are,

1. SQL object
2. SQL Alchemy.

### Q4. List out the commonly used related modules of database programming.

**Answer :** Model Paper-III, Q1(j)

The most commonly used related modules and databases that work with them are depicted as follows,

| | |
|---|---|
| MYSQL | PypgSQL |
| Gadfly | Psycopg |
| Popy | PypgSQL |
| Psycopg 2 | PostgreSQL |
| PyGreSQL | MySQLdba.k.a MYSQL.python |
| SQLite | adodbapi |
| APSW | sybase |
| sqlite3 | sybase |
| pysqlite | oracle |
| sapdb | Dcoracle2 |
| sdb | ex_oracle |
| MaxDB(SAD) | ingres DBI |
| KInterbasDBIngres | ORMS |
| pymssql | ingmod |
| SQLserver | SQLAlchemy |
| Fire bird(Interbase) | PyDO/PyDO2 SQL object |

List out the various connection objects and relational databases.

wer :

Model Paper-II, Q1(j)

### ection Objects

Connections usually depict the communication between the users and the database. They are the communication mechanisms send commands to the server. Initially connections are established and then cursors are created to send and receive requests responses respectively. Various methods of connection objects are as follows,

| Method Name | Description |
|---|---|
| Cursor () | It creates a cursor |
| Commit () | It commits the transaction |
| Close () | It closes the database connection |
| Rollback () | It cancels the current transaction |
| Errorhandler (cxn, errcls, cur errval) | It server as handler for specified connection cursor. |

### lational Databases

The database systems that are accessible to interfaces in Python are as follows,

| Commercial RDBMSs | Open Source RDBMSs | Database APIs |
|---|---|---|
| SAP | SQlite | ODBC |
| Sybase | Gadifly | JDBC |
| Informix | MySQL | |
| Oracle | Postgre SQL | |
| DB12 | | |
| Interbase | | |
| MSSQL server | | |
| Ingres | | |

# PART-B

## ESSAY QUESTIONS WITH SOLUTIONS

### 5.1 INTRODUCTION

**Q6. Give a detail description about database programming.**

**Answer :** Model Paper-II, Q10

Data bases can be defined as the collections of tables that are connected with each other through columns. The basic concepts of database programming are as follows,

**1. Persistent Storage**

All the applications need persistent storage. The basic types of storage mechanisms are files, relational database system (RDBMS) and a type of hybrid ie., an API that is root of existing systems, file manager, spreadsheet, object relational mapper (ORM), configuration file etc.

**2. Basic Database Operations and SQL**

Some of the elementary database concepts and SQL are illustrated as follows,

**(i) Underlying Storage**

All the databases have fundamental persistent storage by using the file system. Examples of it are special operating system files, normal operating system files and raw disk partitions.

**(ii) User Interface**

A command line tool is used for SQL commands and queries. This tool is provided by the database systems. Even the GUI tools make use of command line clients or database client library so that the users are provided with an easy interface.

**(iii) Databases**

The RDBMS has the ability to manage various databases such as customer support, sales, marketing etc at a time on one server. The MYSQL is an example of server based RDBMS because it consists of server process that is running continuously.

**(iv) Components**

Databases consists of tables as storage abstractions. And every table consists of rows and columns. A data base schema can be defined as set of table definitions of columns and data types together. The databases and tables can be created and deleted as well. It is even possible to perform various operations such as insertion, updation, deletion etc on them.

**(v) SQL**

SQL provides database commands and queries to the database. Various database commands in SQL are as follows,

**Create Database**

```
CREATE DATABASE SCHOOL;
GRANT ALL ON SCHOOL. * to user(s);
```

It creates a database with name SCHOOL and grants permission to specific users in order to perform database operations.

**Use a Database**

```
USE SCHOOL;
```

It specifies the database that is to be used for performing operations.

**Drop the Database**

```
DROP database SCHOOL;
```

It deletes the database and its tables from the system.

**Create a Table**

```
CREATE TABLE SCHOOL (ID INT, NAME VARCHAR(20));
```

It creates a new table with columns ID and NAME.

**Drop a Table**

```
DROP TABLE SCHOOL;
```

It deletes the table completely along with the data.

**Insert a Row**

```
INSERT INTO SCHOOL VALUES('21', 'JAMES');
```

It inserts a row in the table.

**Update a Row**

```
UPDATE SCHOOL SET NAME = 'HARRY'
        WHERE id = 4;
```

It modifies and updates the values of specified row.

**Delete a Row**

```
DELETE FROM SCHOOL WHERE id = 2;
```

It deletes the data of specified row.

### 3. Databases and Python

The users are allowed to access RDBMS from Python directly by using the database interface or through an ORM. The Python applications can be integrated with some type of database system. The data can be stored and retrieved to/from RDBM's while working in Python frame work. An adapter is used to access the database from Python. It is a Python module that enables users to interface with relational database's client library.

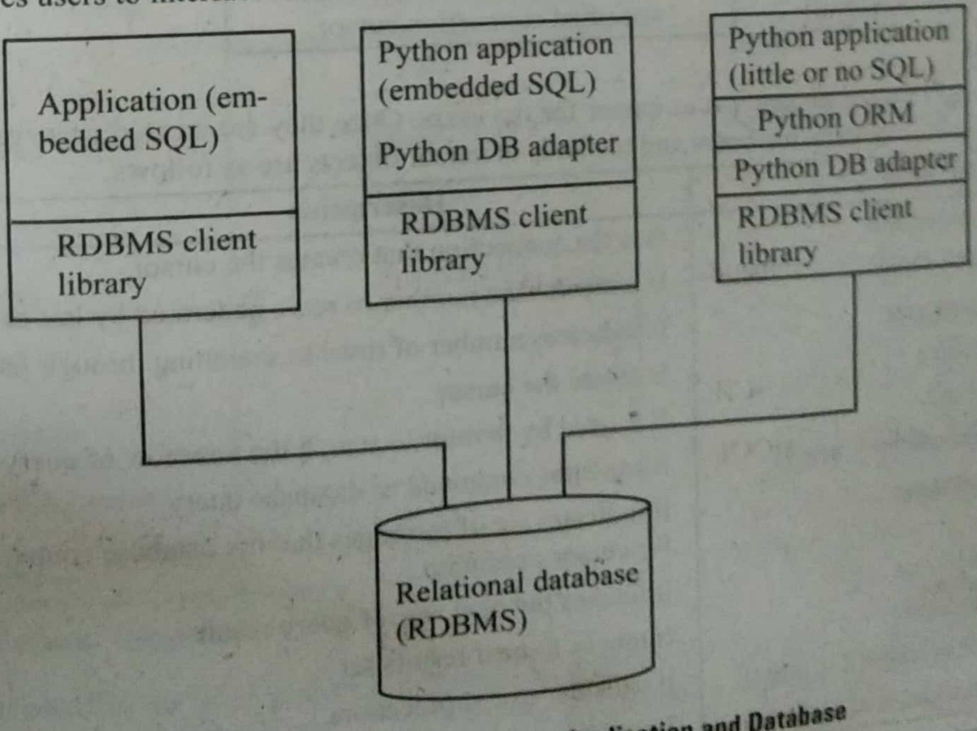
Application (embedded SQL)
RDBMS client library
Python application (embedded SQL)
Python DB adapter
RDBMS client library
Python application (little or no SQL)
Python ORM
Python DB adapter
RDBMS client library
Relational database (RDBMS)

**Figure: Communication between Application and Database**

## 5.2 PYTHON DATABASE APPLICATION PROGRAMMER'S INTERFACE (DB – API)

**Q7. Explain in detail about database application programmers interface.**

Model Paper-I, Q10

**Answer :**

**Database Application Programmer's Interface (DB-API)**

The API is a specification which defines a set of objects as well as database access mechanisms for providing access to different database adapters and database systems. The API is developed by SIG (a special interest group). This API is responsible for providing the consistent interface to different relational databases. The code porting among the databases is simple and needs very less code.

**1. Module Attributes**

The DB-API complaint module defines the below define global attributes.

| Attributes | Description |
|---|---|
| Apilevel | It indicates the version of DB-API with which the module has complaint. |
| Paramstyle | It represents the SQL statement parameter style of this module. Examples of various paramstyles are qmark, format, numeric pyformat, named etc. |
| Connect | It is a function that is used to access the database with connection objects. |
| Threadsafety | It indicates the level of thread safety of the module. |
| Exception | The exceptions that are included in complaint module as globals are error, warning, database error etc. |

**2. Connection Objects**

Connections usually depict the communication between the users and the database. They are the communication mechanisms that send commands to the server. Initially connections are established and then cursors are created to send and receive requests and responses respectively. Various methods of connection objects are as follows,

| Method Name | Description |
|---|---|
| Cursor() | It creates a cursor |
| Commit() | It commits the transaction |
| Close() | It closes the database connection |
| Rollback() | It cancels the current transaction |
| Errorhandler(cxn, errcls, curerrval) | It serves as handler for specified specified connection cursor. |

**3. Cursor Objects**

The Python DB-API cursor object acts as cursor for the users. Once they are created users can execute the queries and retrieve data from result set. The data attributes and methods of cursor objects are as follows,

| Attribute | Description |
|---|---|
| Connection | It is the connection that creates the cursor. |
| Row count | It indicates the number of rows generated by last execute () |
| Array size | It indicates number of rows to searching through fetch many () |
| Close() | It closes the cursor |
| Next() | It is used by iterator to search the next row of query result. |
| Execute(op [, args]) | It executes command or database query |
| Messages | It indicates set of messages that the database contains for for cursor execution |
| Fetch one | It fetches the next row of query result |
| Next set() | It moves to next results set |
| Callproc(fun C [, args]) | It calls the stored procedure |
| Set input-sizes (sizes) | It sets the maximum input size that is allowed |

### 4. Type Objects and Constructors

The Python objects and native data objects are related with each other. When the programmer sends any parameters to database, they will reach in the form of strings. They need to be converted into some other data types compatible with the query. The DBAPI also needs to create the constructors that in turn create special objects. These objects can be easily converted to database objects. The type objects that are used for this purpose are as follows,

| Type Object | Description |
|---|---|
| Date from Ticks (ticks) | It indicates the date object as per the given number of seconds from epoch. |
| Date(yr, mo, dy) | It indicates object for date value. |
| Time(hr, min, sec) | It indicates the object from time value. |
| Timestamp(yr, mo, dy, hr, min, sec) | It indicates the object for timestamp value. |
| Time stamp from ticks(ticks) | It indicates object as per number of seconds from epoch. |
| Time from ticks(ticks) | It indicates the time object as per number of seconds from epoch. |
| NUMBER | It indicates the object that represents numeric columns. |
| STRING | It indicates the object that represents the string based columns. |
| ROWID | It indicates the object that represents 'row ID' columns. |
| BINARY | It indicates the object that represents binary columns. |
| DATETIME | It indicates the object that represents data/time columns. |

### 5. Relational Databases

The database systems that are accessible to interfaces in Python are as follows,

| Commercial RDBMSs | Open Source RDBMSs | Database APIs |
|---|---|---|
| SAP | SQlite | ODBC |
| Sybase | Gadifly | JDBC |
| Informix | MySQL | |
| Oracle | Postgre SQL . | |
| DBI2 | | |
| Interbase | | |
| MSSQL server | | |
| Ingres | | |

### 6. Databases and Python: Adapters

The adapter is responsible for the connection between the database system and python. The databases like SAP, SQL server, Sybase, Oracle etc., have multiple adapters. If there are multiple adapters provided for any database system, then the best among them must be selected based on their features.

## 5.3 OBJECT RELATION MANAGERS (ORMs)

**Q8. Discuss about object relational managers.**

**Answer :**

Model Paper-III, Q10

**ORM**

ORM stands for object-relational managers. There are various types of database systems that enable users to maintain Python interfaces to harness their power. But they need atleast some knowledge of SQL. If the programmer can manipulate Python objects rather than SQL queries and need relational database then ORM would be a best option.

These systems are designed in such a way that users can manipulate the objects and perform the tasks. The tables are converted to classes containing features and columns as attributes and methods as database operations. The application can be set to an ORM. Certain operations might be complex and even need more lines of code than using an adapter directly because ORM's perform a lot of work on behalf of users. There are two ORM's that are used more frequently, they are,

1. SQL object
2. SQL Alchemy

Other Python ORM's are Durus, PDO, Dejavu, PyDO/PyDO2, QLime, Forget SQL etc. Certain web based systems can maintain their own ORM component i.e., webware middle kit and Django's database API.

**Example**

**SQLAlchemy ORM**

```
import os
from random import randrange as rrange
from sqlalchemy import *
from ushuffle_db import NAMES, randName
FIELDS = ('login', 'uid', 'prid')
DBNAME = 'test'
COLSIZ = 10
class MySQLAlchemy(object):
def__init__(self, db, dbName):
import MySQLdb
import _mysql_exceptions
MySQLdb = pool.manage(MySQLdb)
url = 'mysql://db=%s' % DBNAME
eng = create_engine(url)
try:
cxn = eng.connection()
except _mysql_exceptions.OperationalError, e:
eng1 = create_engine('mysql://user=root')
try:
```

```
eng1.execute('DROP DATABASE %s' % DBNAME)
except _mysql_exceptions.OperationalError, e:
pass
eng1.execute('CREATE DATABASE %s' % DBNAME)
eng1.execute( "GRANT ALL ON %s.* TO "@'localhost'" % DBNAME)
eng1.commit()
cxn = eng.connection()
try:
users = Table('users', eng, autoload=True)
except exceptions.SQLError, e:
users = Table('users', eng, Column('login', String(8)), Column('uid', Integer), Column('prid', Integer), redefine=True)
self.eng = eng
self.cxn = cxn
self.users = users
def create(self):
users = self.users
try:
users.drop()
except exceptions.SQLError, e:
pass
users.create()
def insert(self):
d = [dict(zip(FIELDS,[who, uid, rrange(1,5)])) for who, uid in randName()]
return self.users.insert().execute(*d).rowcount
def update(self):
users = self.users
fr = rrange(1,5)
to = rrange(1,5)
return fr, to, \
users.update(users.c.prid==fr).execute(prid=to).rowcount
```

```
def delete(self):
users = self.users
rm = rrange(1,5)
return rm, \
users.delete(users.c.prid==rm).execute().rowcount
def dbDump(self):
res = self.users.select().execute()
print '\n%s%s%s' % ('LOGIN'.ljust(COLSIZ), 'USERID'.ljust(COLSIZ), 'PROJ#'.ljust(COLSIZ))
for data in res.fetchall():
print '%s%s%s' % tuple([str(s).title().ljust (COLSIZ) for s in data])
def-getattr-(self, attr):
return getattr(self.users, attr)
def finish(self):
self.cxn.commit()
self.eng.commit()
def main():
print '*** Connecting to %r database' % DBNAME
orm = MySQLAlchemy('mysql', DBNAME)
print '\n*** Creating users table' orm.create()
print '\n*** Inserting names into table'
orm.insert()
orm.dbDump()
print '\n*** Randomly moving folks',fr, to, num = orm.update()
print 'from one group (%d) to another (%d)' % (fr, to)
print '\t(%d users moved)' % num
orm.dbDump()
print '\n*** Randomly choosing group', rm, num = orm.delete()
print '(%d) to delete' % rm
print '\t(%d users removed)' % num
orm.dbDump()
```

```
print '\n*** Dropping users table'
orm.drop()
orm.finish()
```

## SQLObject ORM

```
import os
from random import randrange as rrange
from sqlobject import *
from ushuffle_db import NAMES, randName
DBNAME = 'test'
COLSIZ = 10
FIELDS = ('login', 'uid', 'prid')
class MySQLObject(object):
def __init__(self, db, dbName):
import MySQLdb
import _mysql_exceptions
url = 'mysql://localhost/%s' % DBNAME
while True:
cxn = connectionForURI(url)
sqlhub.processConnection = cxn
#cxn.debug = True
try:
class Users(SQLObject):
class sqlmeta:
fromDatabase = True
login = StringCol(length=8)
uid = IntCol()
prid = IntCol()
break
except _mysql_exceptions.ProgrammingError, e:
class Users(SQLObject):
```

```
login = StringCol(length=8)
uid = IntCol()
prid = IntCol()
break
except _mysql_exceptions.OperationalError, e:
cxn1 = sqlhub.processConnection=
connectionForURI('mysql://root@localhost')
cxn1.query("CREATE DATABASE %s" % DBNAME)
cxn1.query("GRANT ALL ON %s.* TO "@'localhost'" % DBNAME)
cxn1.close()
self.users = Users
self.cxn = cxn
def create(self):
Users = self.users
Users.dropTable(True)
Users.createTable()
def insert(self):
for who, uid in randName():
self.users(**dict(zip(FIELDS,[who, uid, rrange(1,5)])))
def update(self):
fr = rrange(1,5)
to = rrange(1,5)
users = self.users.selectBy(prid=fr)
for i, user in enumerate(users):
user.prid = to
return fr, to, i+1
def delete(self):
rm = rrange(1,5)
users = self.users.selectBy(prid=rm)
for i, user in enumerate(users):
```

```
user.destroySelf()
return rm, i+1
def dbDump(self):
print '\n%s%s%s' % ('LOGIN'.ljust(COLSIZ),'USERID'.ljust(COLSIZ), 'PROJ#'.ljust(COLSIZ))
for usr in self.users.select():
print '%s%s%s' % (tuple([str(getattr(usr,field)).title().ljust(COLSIZ) \for field in FIELDS]))
drop = lambda self: self.users.dropTable()
finish = lambda self: self.cxn.close()
def main():
print '*** Connecting to %r database' % DBNAME
orm = MySQLObject('mysql', DBNAME)
print '\n*** Creating users table'
orm.create()
print '\n*** Inserting names into table'
orm.insert()
orm.dbDump()
print '\n*** Randomly moving folks',fr, to, num = orm.update()
print 'from one group (%d) to another (%d)' % (fr, to)
print '\t(%d users moved)' % num
orm.dbDump()
print '\n*** Randomly choosing group',rm, num = orm.delete()
print '(%d) to delete' % rm
print '\t(%d users removed)' % num
orm.dbDump()
print '\n*** Dropping users table'
orm.drop()
orm.finish()
if_name__ =='__main__':
main()
```

## 5.4 RELATED MODULES

**Q9. List out various related modules of database programming.**

[Model Paper-II, Q11 | Model Paper-III, Q11]

**Answer :**

The most commonly used related modules and databases that work in database programming are as follows.

MYSQL
Gadfly
Popy
Psycopg 2
PyGreSQL
SQLite
APSW
sqlite3
pysqlite
sapdb
sdb
MaxDB(SAD)
KInterbasDB
pymssql
SQLserver
Fire bird(Interbase)

PypgSQL
Psycopg
PypgSQL
PostgreSQL
MySQLdba.k.a MYSQL.python
adodbapi
sybase
sybase
oracle
Dcoracle2
ex_oracle
ingres DBI
ORMS
ingmod
SQLAlchemy
PyDO/PyDO2 SQL object