Database Basics

Dachuri Chaitanya

M.Tech (IIT Roorkee), AI Expert, 20 yrs Exp.

Faculty Introduction

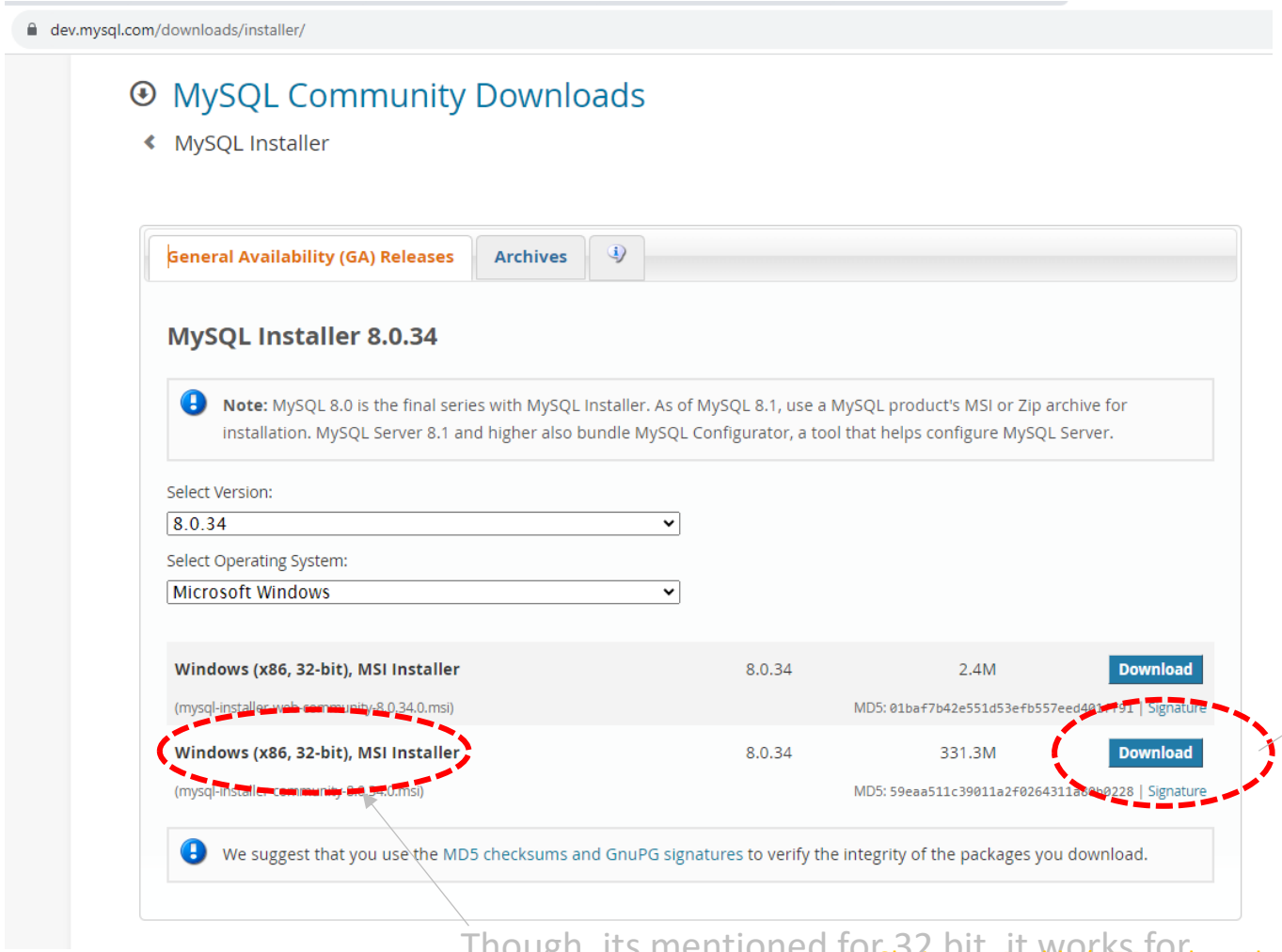
- Post graduate from IIT Roorkee
- 20 yrs Industry Experience
- 10+ yrs Teaching Experience

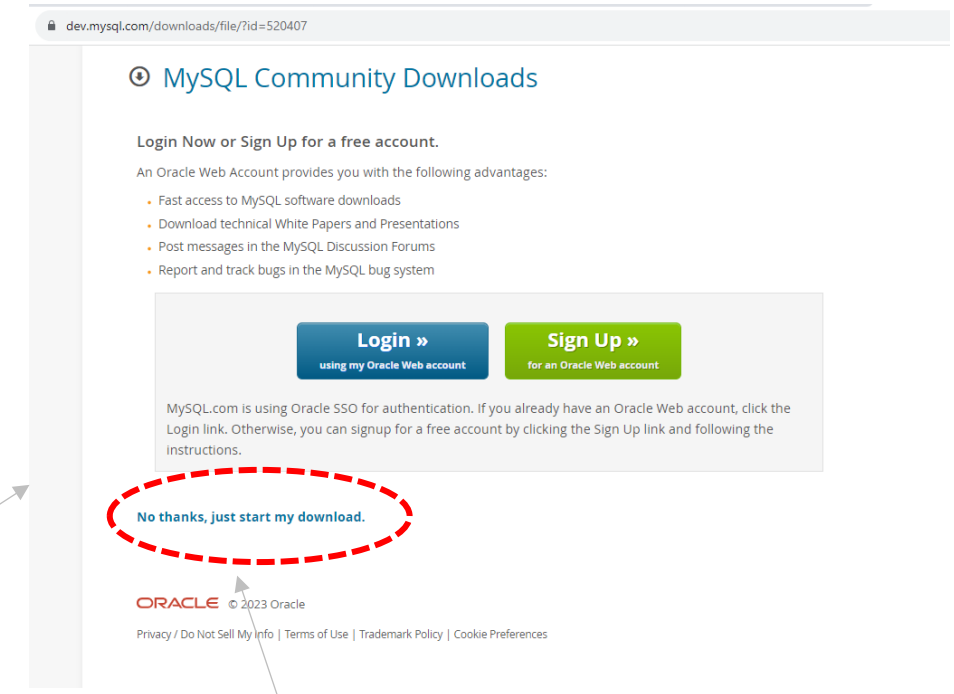
Course Specialties

- Session recording
- Code notebook
- Course Material
- Assignments
- Highlight IMP
- Question hour
- Essence from experience
- Accessible Trainer
- 2.5 months
- Affordable price

Download **MySQL**

- *https://dev.mysql.com/downloads/installer/*

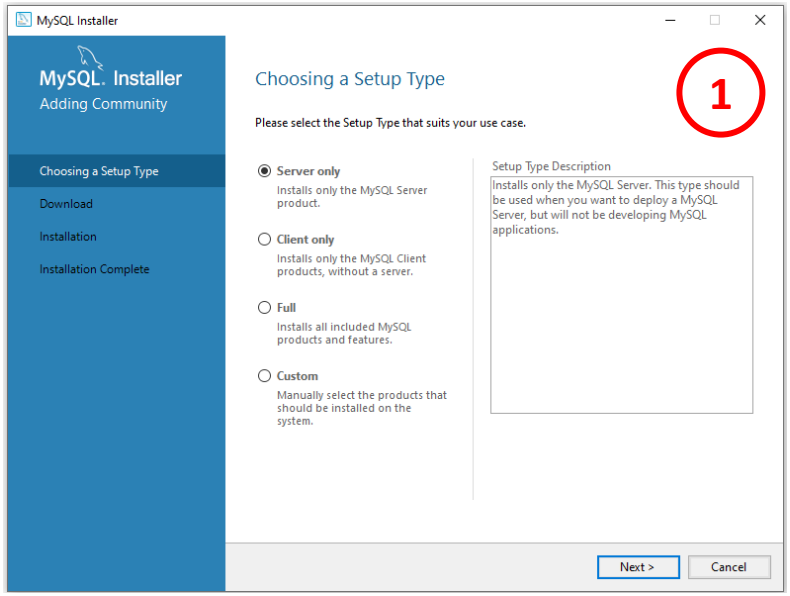


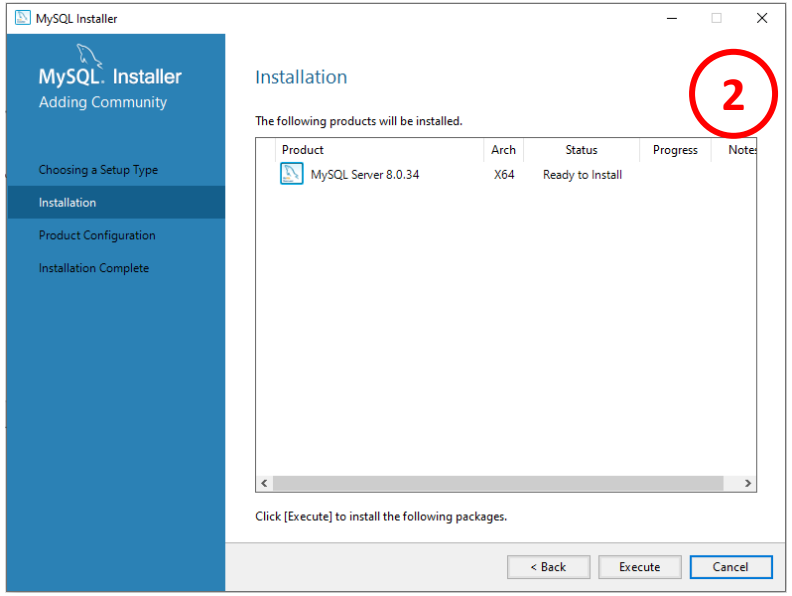


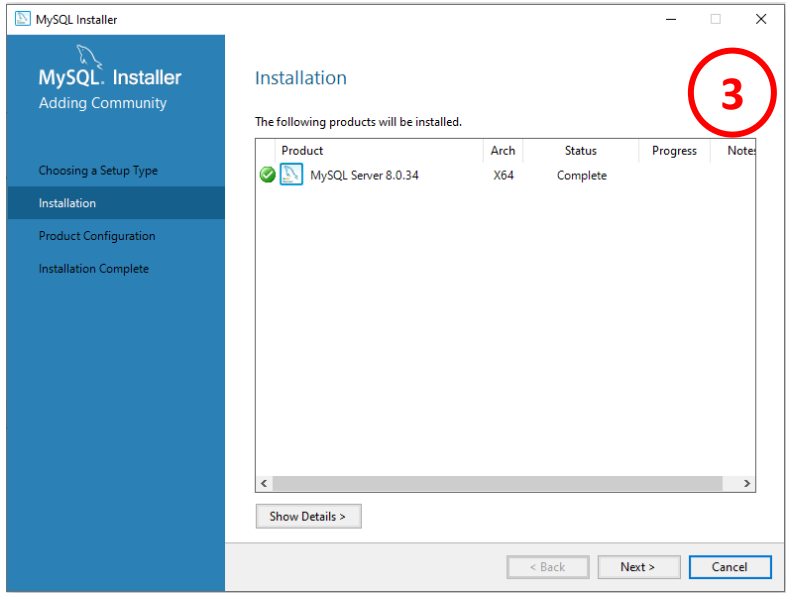
Though, its mentioned for 32 bit, it works for 64-bit OS also.

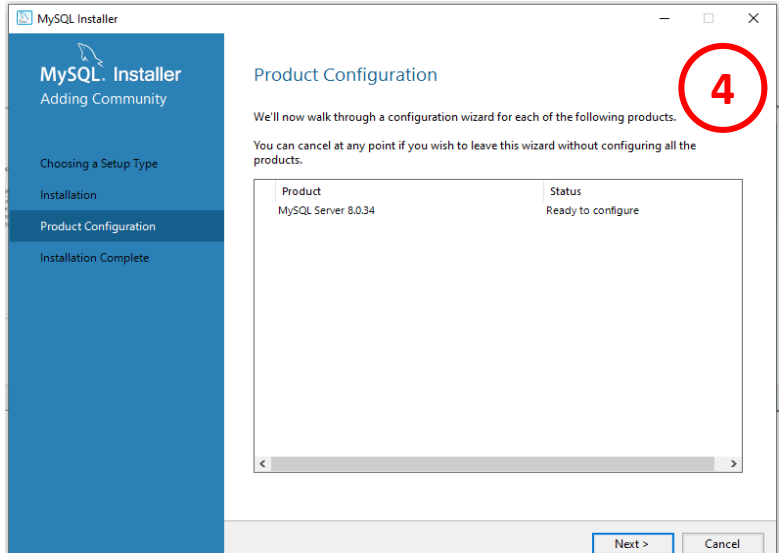
Click on this link to download MySQL

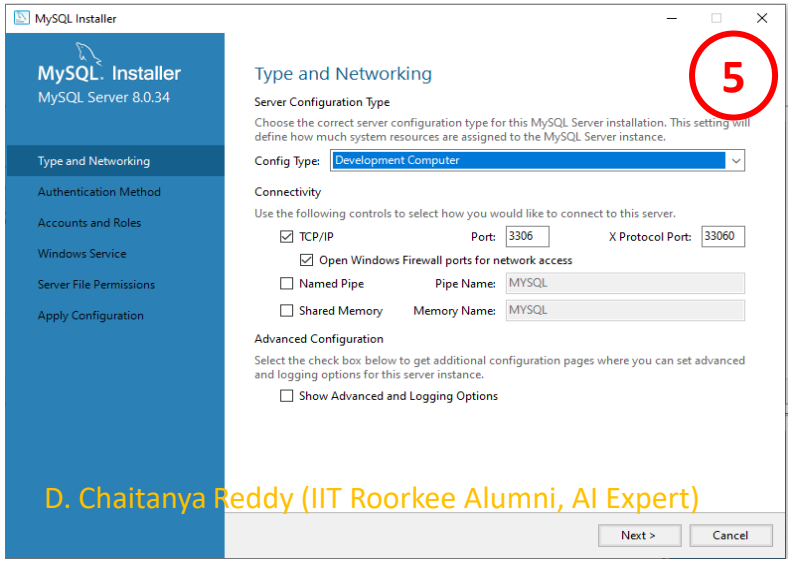
Install **MySQL**

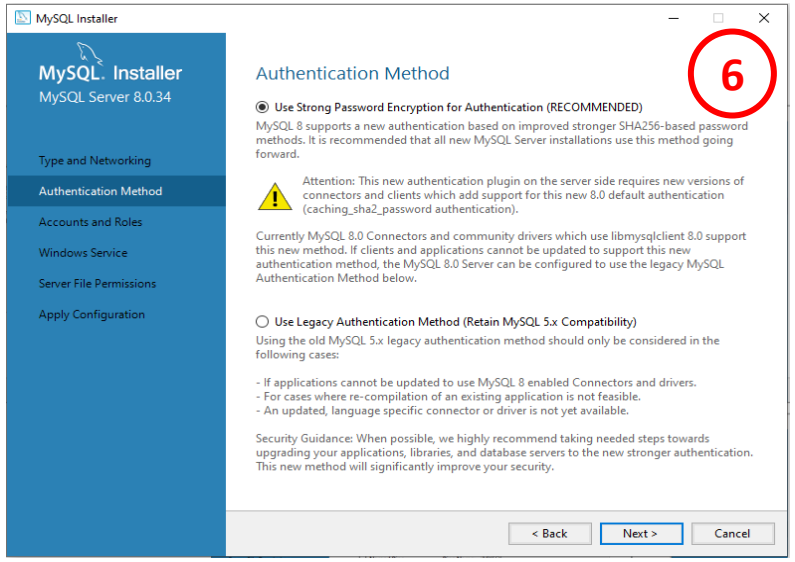






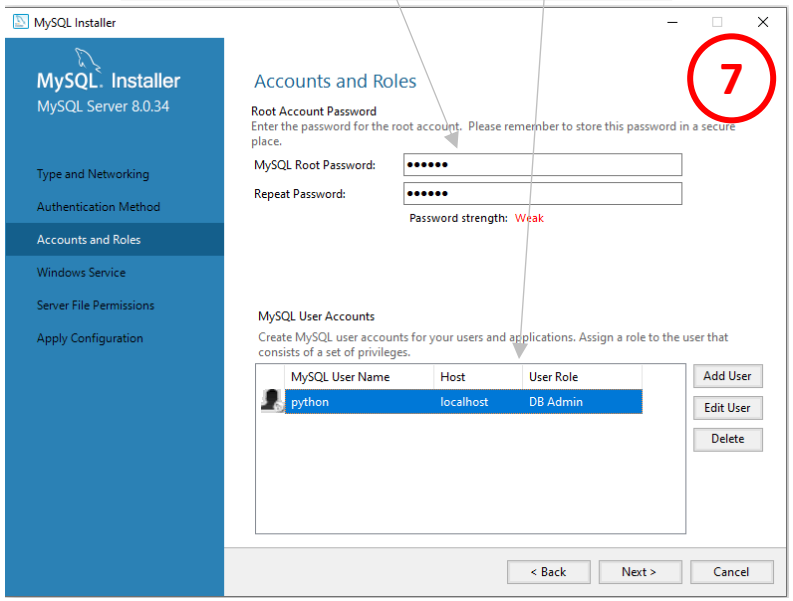


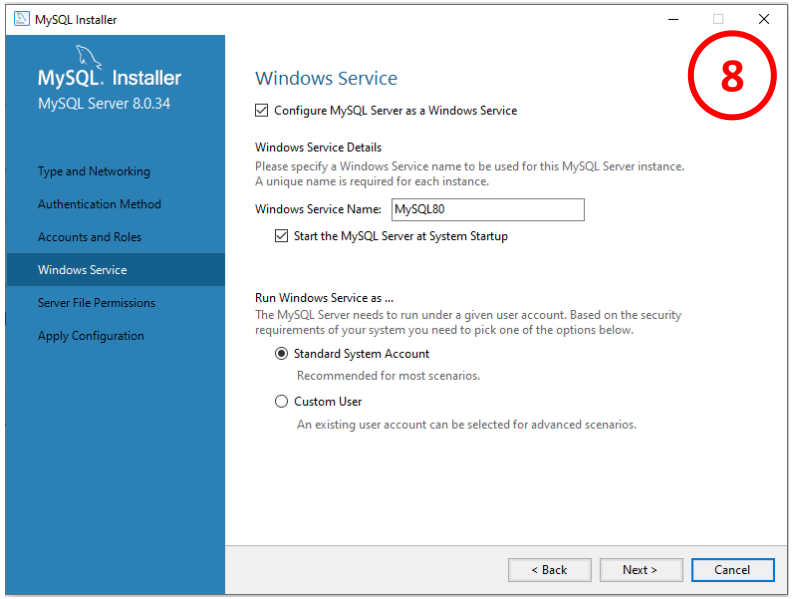


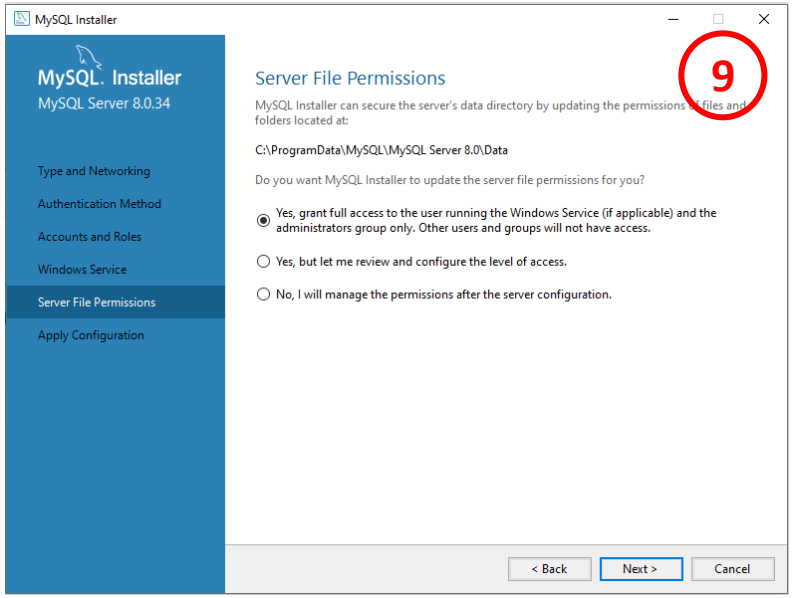


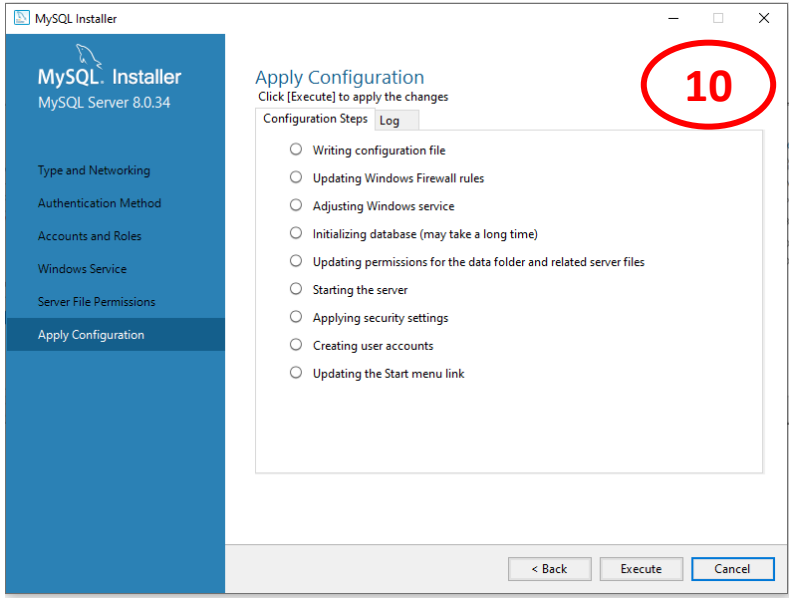
Root Password: python
Admin Username: python
Admin Password: python

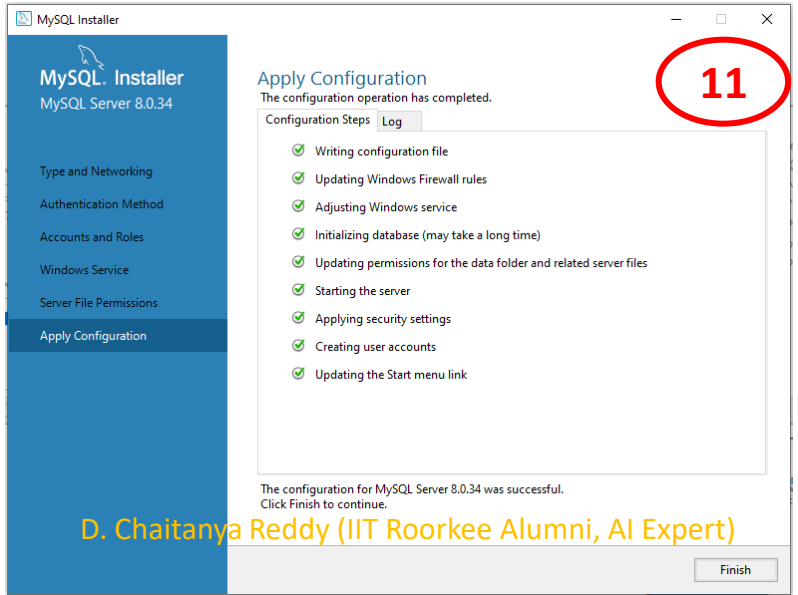
Install MySQL

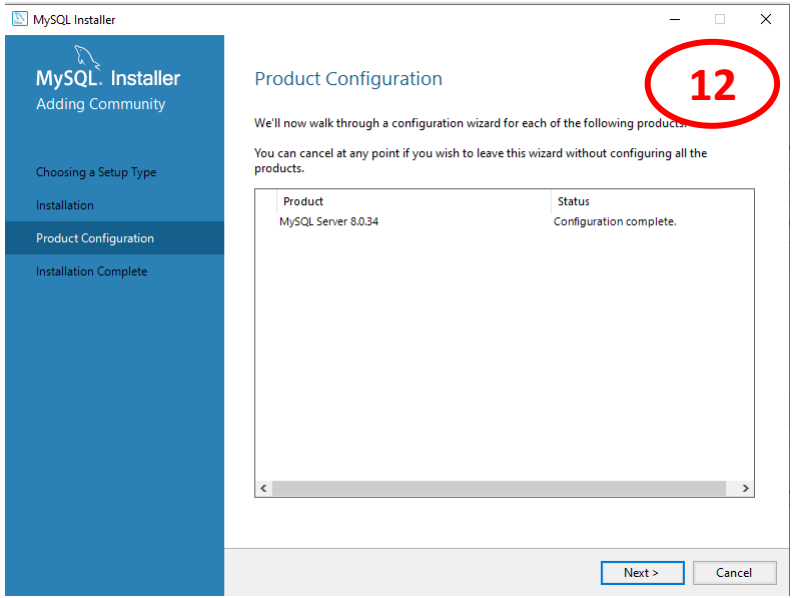




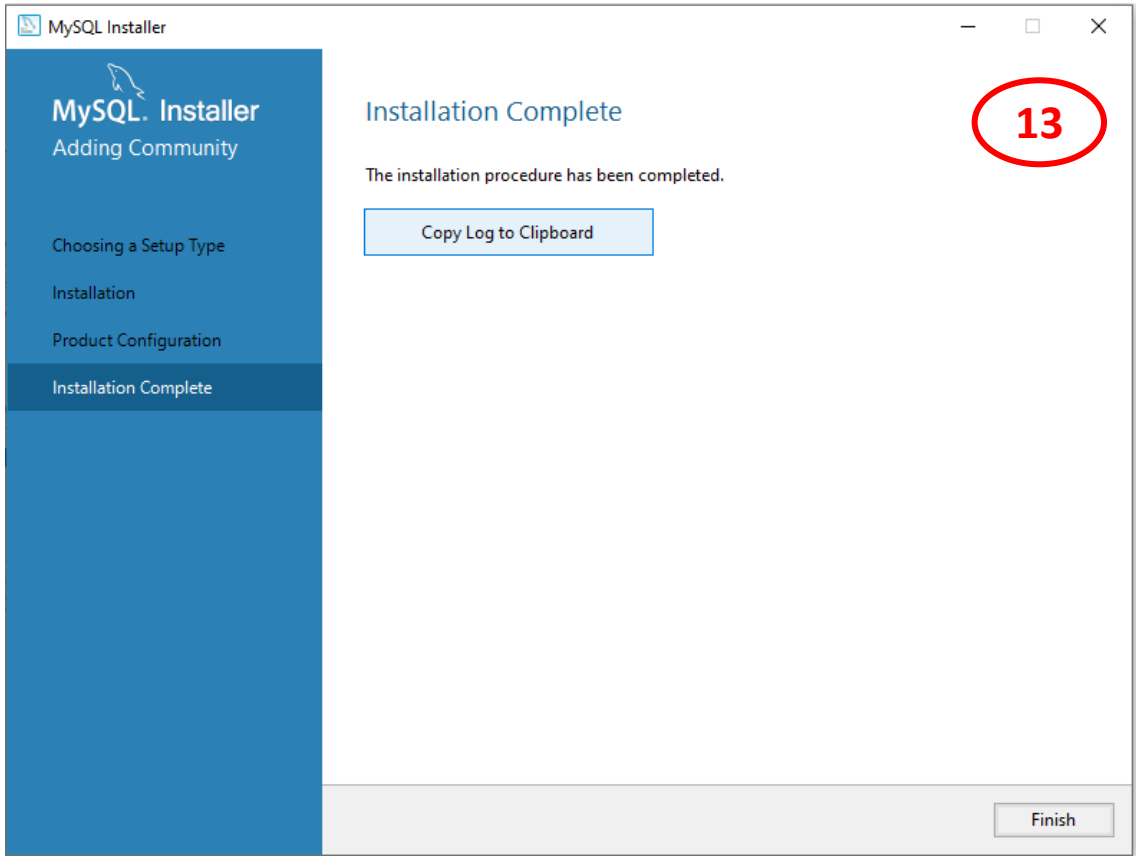






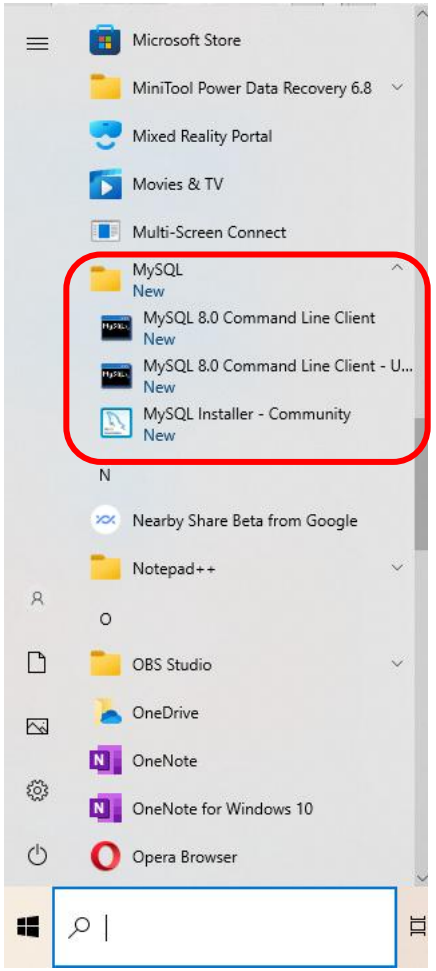


Install **MySQL**



Verify **MySQL**

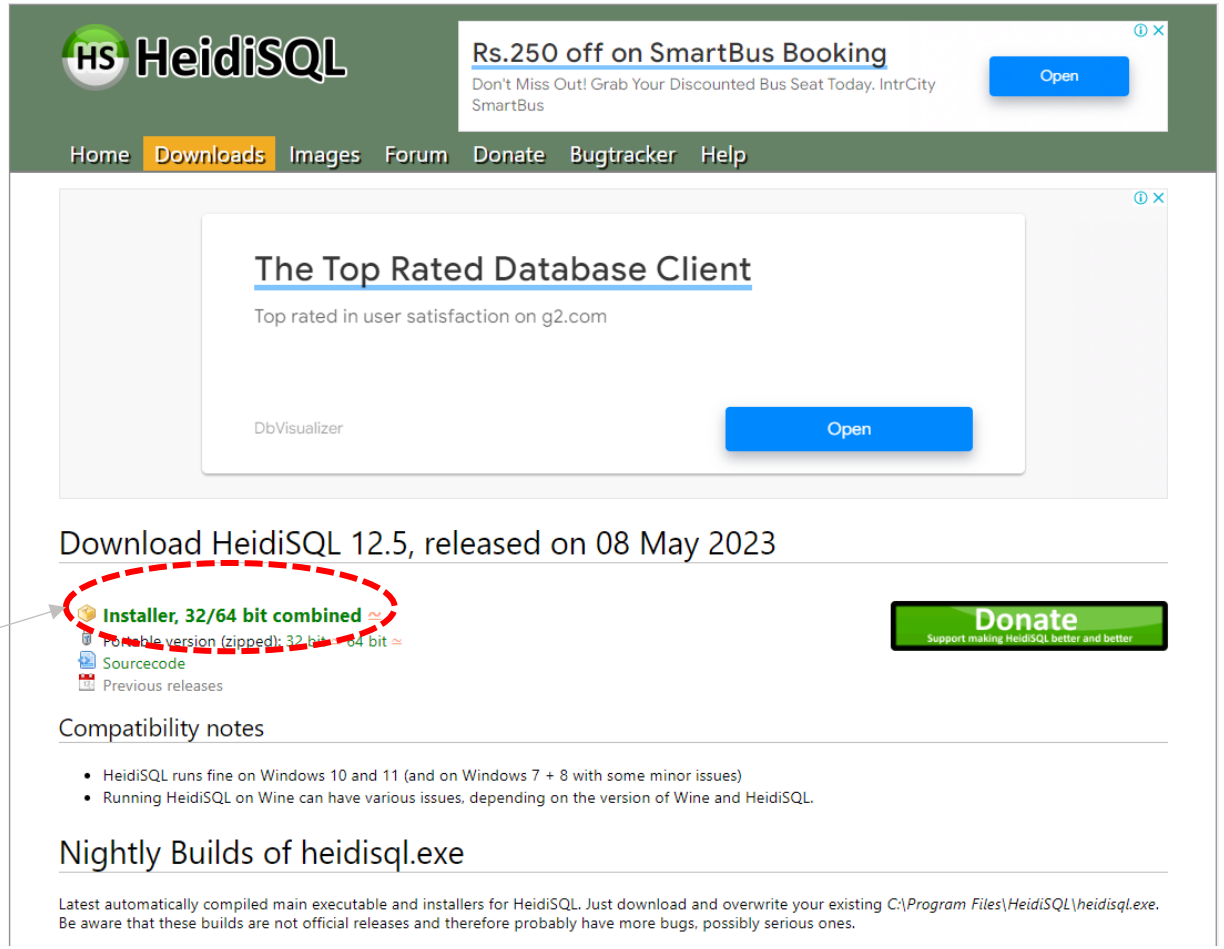
In Start Menu:



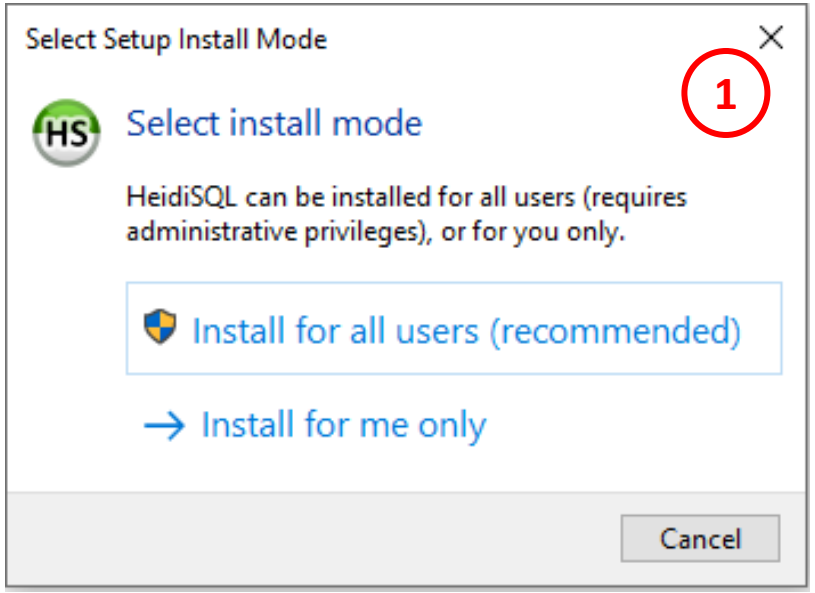
Download **Heidi SQL**

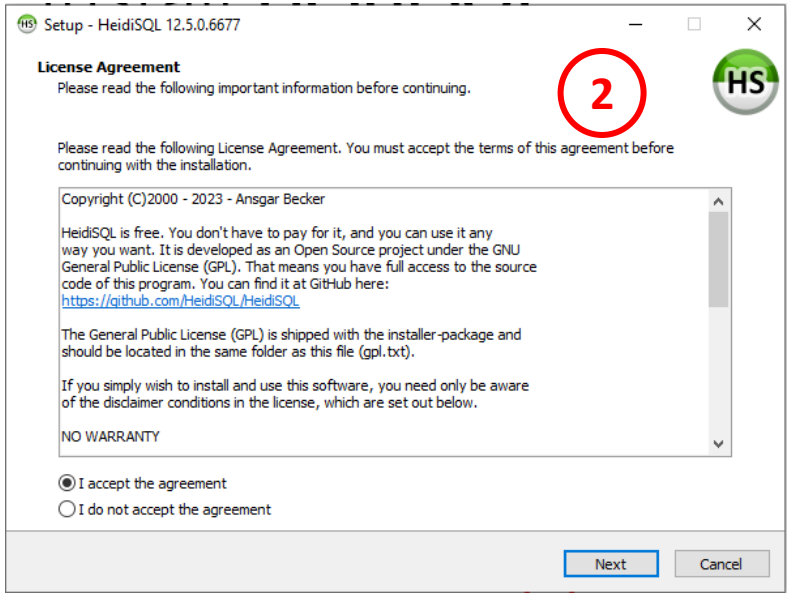
- *https://www.heidisql.com/download.php*

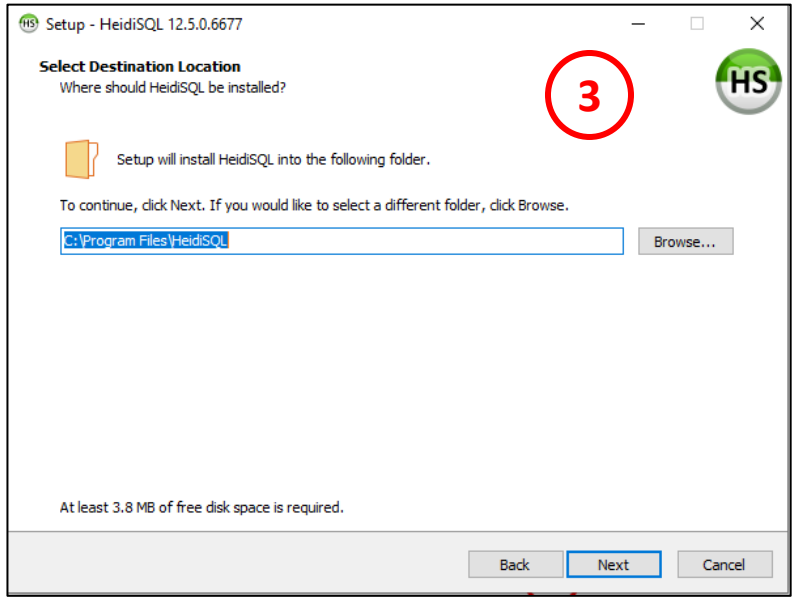
Click on this link to download Heidi SQL

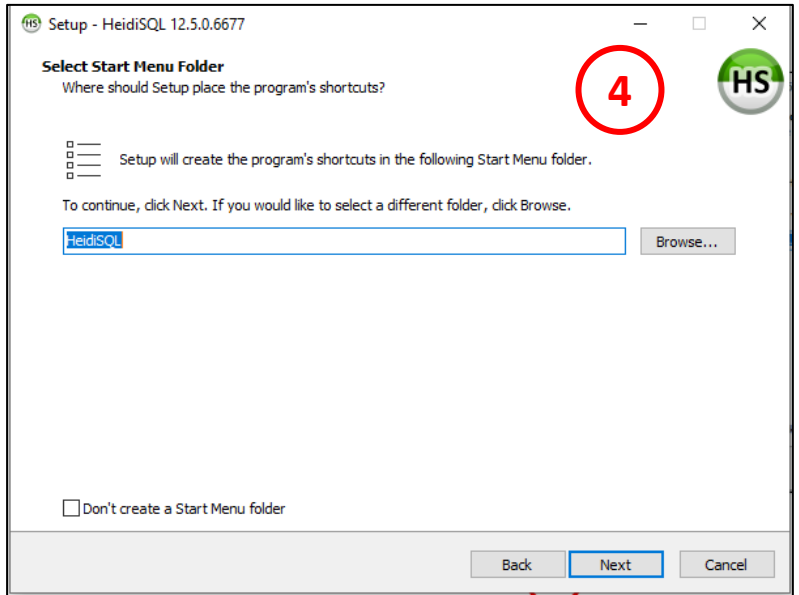


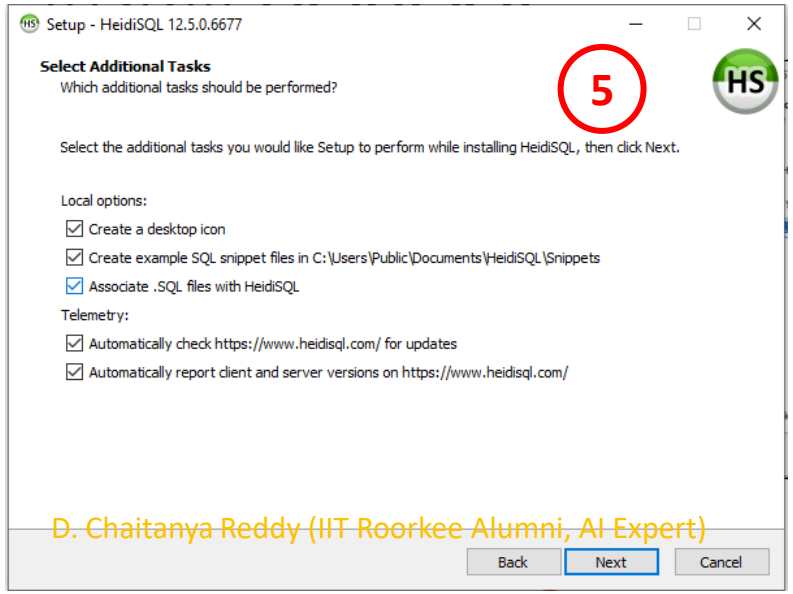
Install **Heidi SQL**

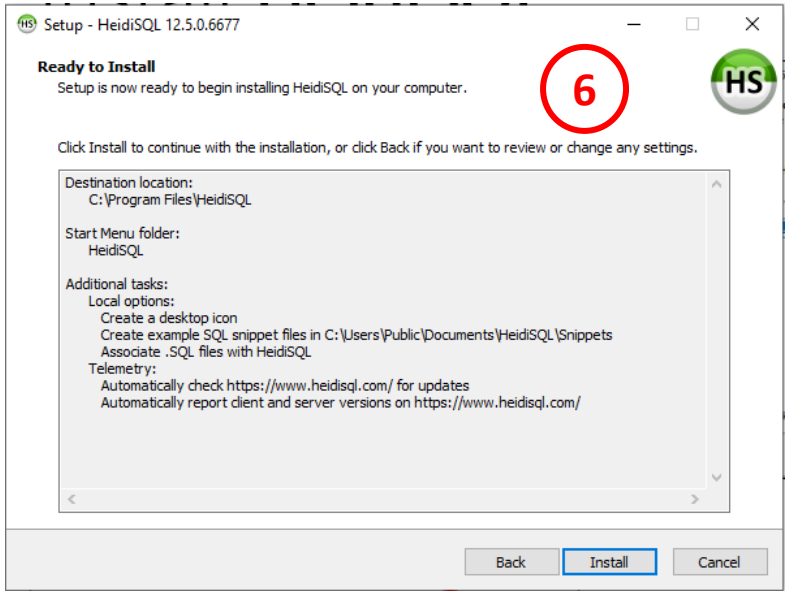




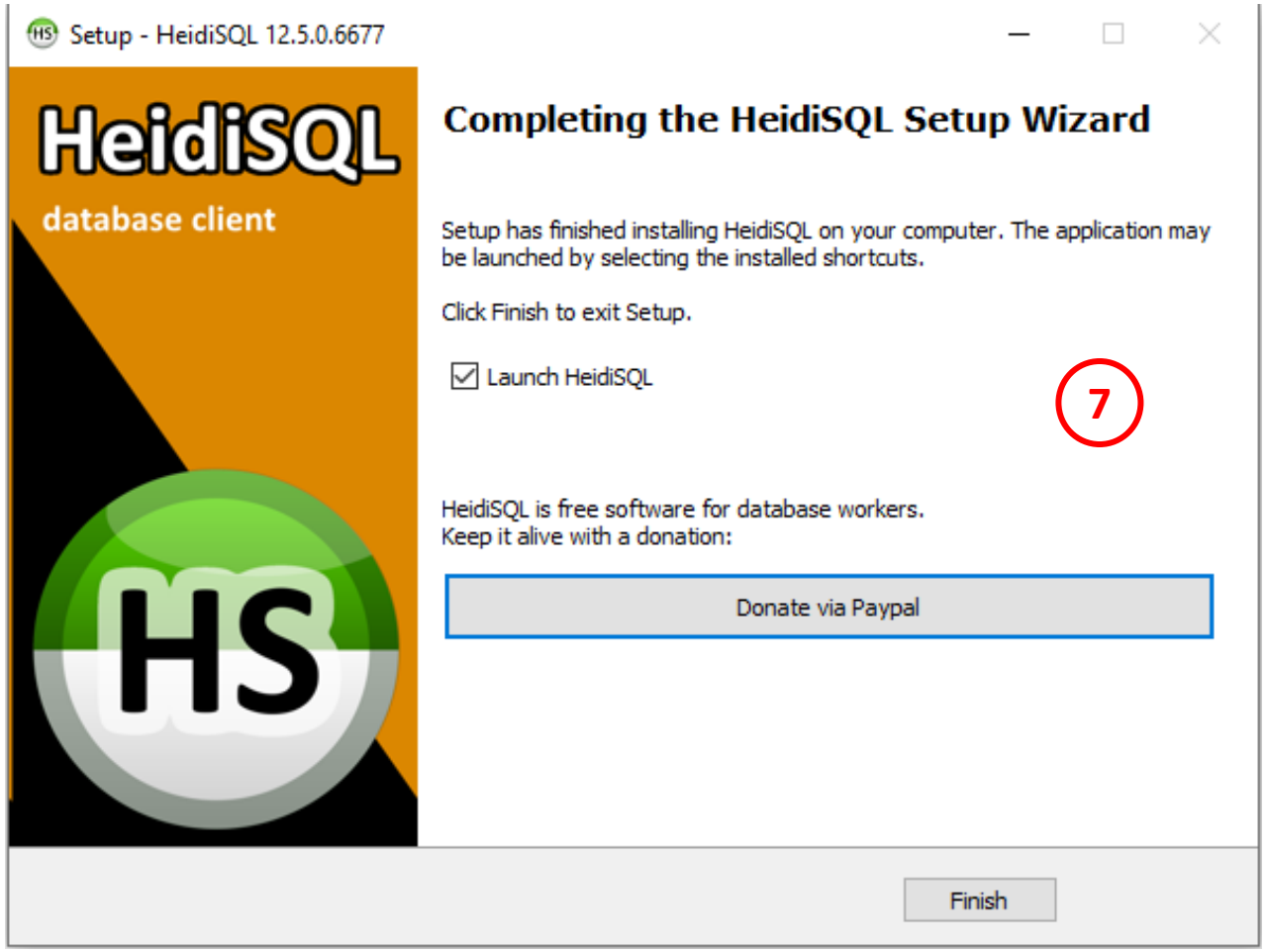






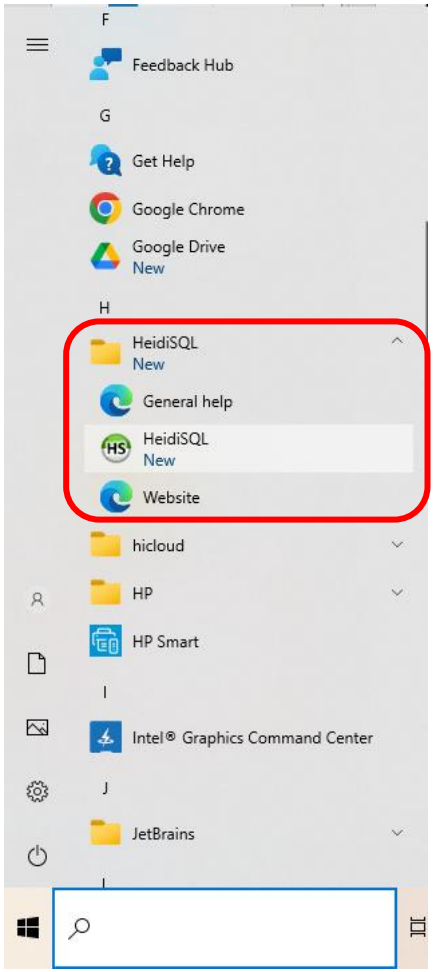


Install **Heidi SQL**

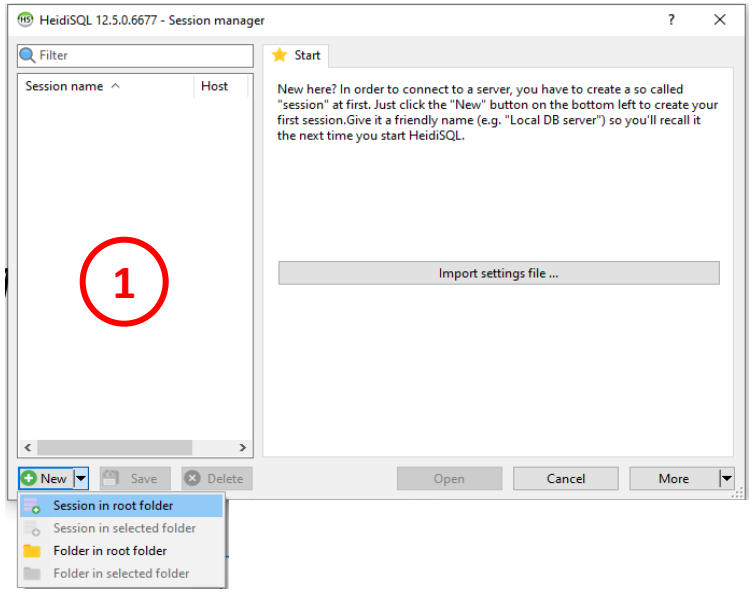


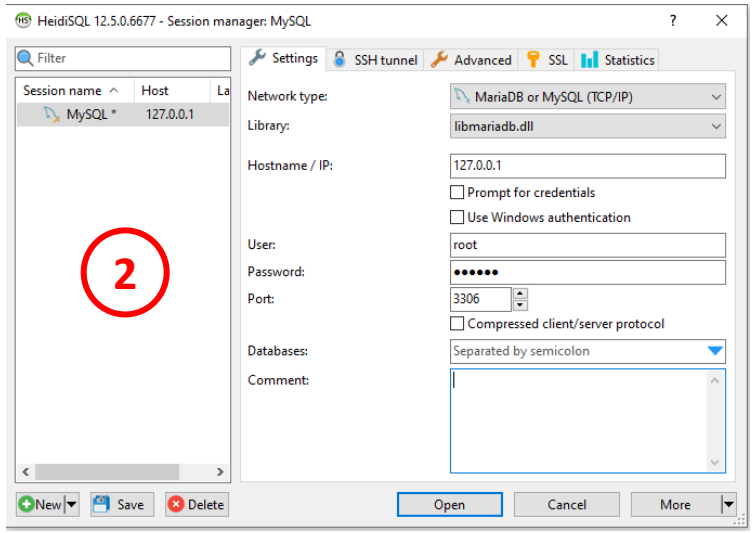
Verify **Heidi SQL**

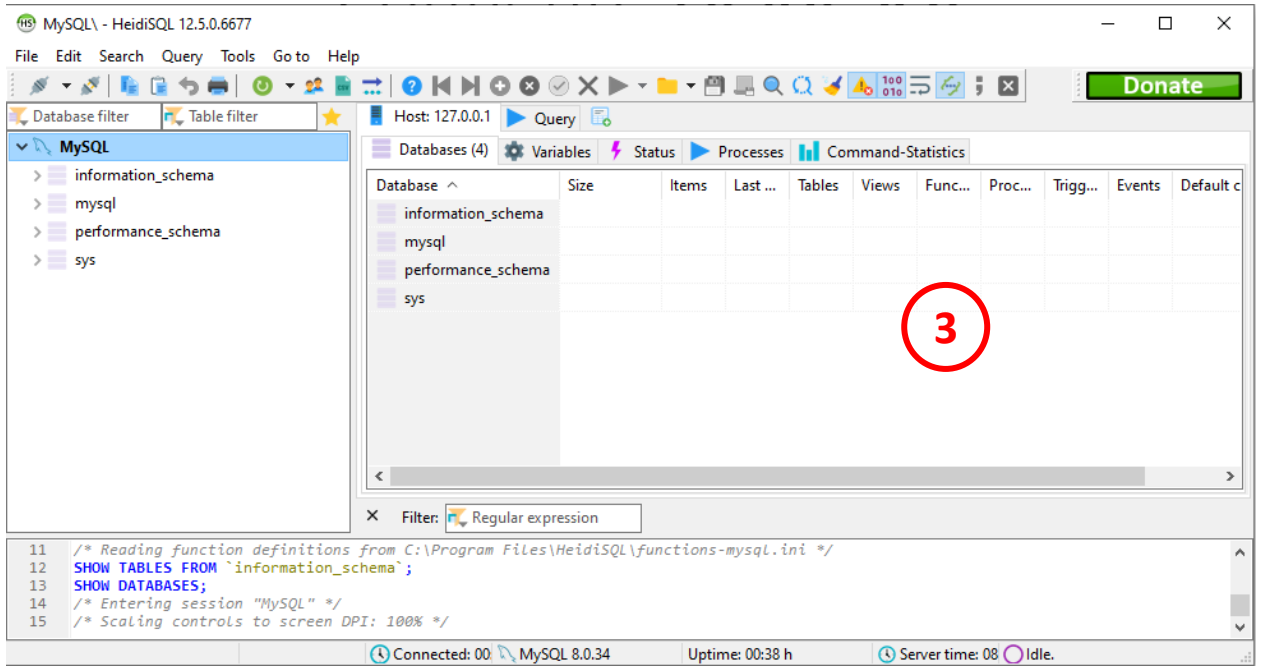
In Start Menu:



Configure **Heidi SQL**







D. Chaitanya Reddy (IIT Roorkee Alumni, AI Expert)

RDBMS

What is **RDBMS** ?

- Relational Database Management System
- It’s a type of database that stores and allows access to data.
- It’s relational, because tables in database have pre-determined relationships with one another.
- Data stored in tables.
- Supports a database language (Ex: SQL) to create, update, delete, fetch DBs, tables, data, etc…

RDBMS Terminology

- **Database**: A repository to store data in various tables in it.
- **Table**: Part of DB that stores data. A table has columns/attributes and data stored in rows.
- **Attributes/Columns**: nothing but columns in a table.
- **Rows/Records/Data**: data entries in a table. Row contains values for each attribute.

RDBMS Terminology

- **Relational model**: A model that uses tables to store data and manages the relationship between tables.

- **RDBMS**: A system that manages data in a database and is based on the relational model.

- **SQL**: A Structured Query Language that interacts with a DBMS.

- **Constraints**: Restrictions/limitations on tables and attributes.

For ex: a wine can be produced only by one winery. An order for wine can't exist if it is not associated with a customer, having a name attribute could be mandatory for a customer

RDBMS Terminology

- **Index:** Used to retrieve data from the database more quickly (10x). The users cannot see the indexes, they are just used to speed up searches/queries.

  - Can't see Indexes. Used to speed up searches/queries
  - Updating a table with indexes takes more time than updating a table without (because the indexes also need an update)
  - only create indexes on columns that will be frequently searched against

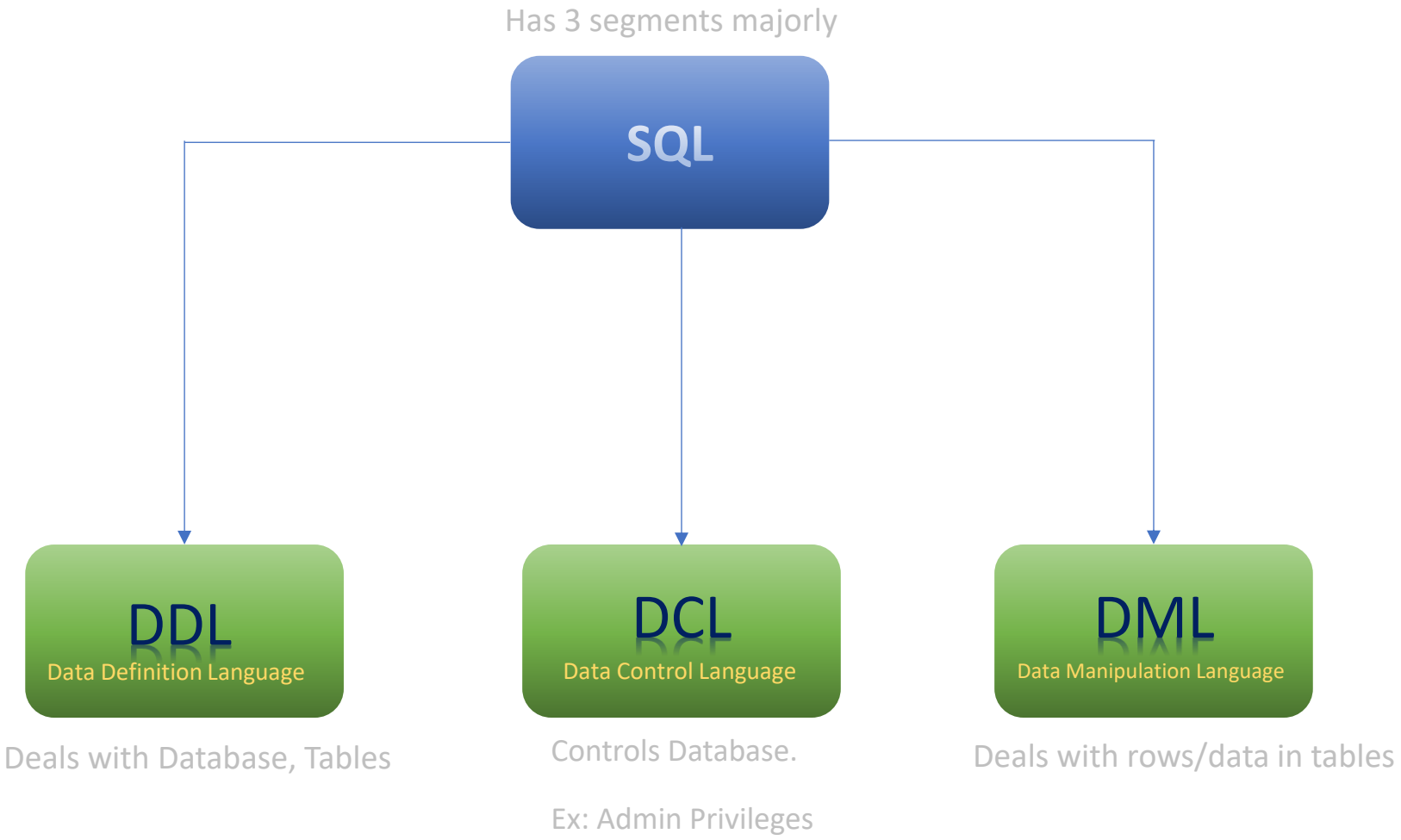
- **Primary key**: one or more columns that contain values that uniquely identify each row.

- **Foreign key**: one or more columns in a table, that refers to the PRIMARY KEY in another table. The table with the foreign key is called the child table, and the table with the primary key is called the referenced or parent table.

- **Normalized Database**: A correctly designed database that is created from an ER model. We have different types/levels of normalization

SQL
Structured Query Language

SQL Overview

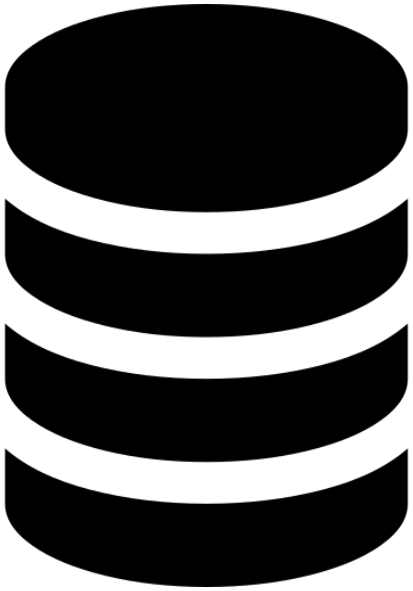


- SQL is not a programming language, it's a query language.
- SQL commands are interpreted by the DBMS engine.
- SQL commands can be embedded within programming languages.
- SQL commands can be used interactively.

DDL
Data Definition Language

DDL

- It is the set of SQL statements used to managed a database.



Database

Create

Alter

Drop

Show

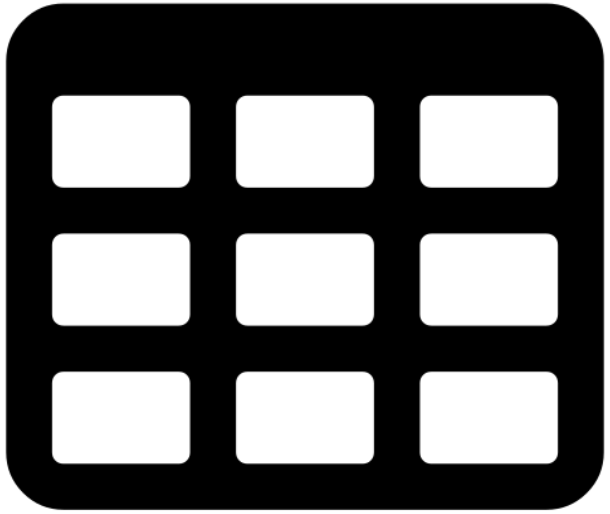


Table in DB

DDL statements

Database:

```
CREATE DATABASE DB_MYAPTZ;

USE DB_MYAPTZ;

SHOW DATABASES;

DROP database DB_MYAPTZ;
```

Tables:

```
create table DB_MYAPTZ.FLAT_INFO(
BLOCK_NAME varchar(5) NOT NULL,
FLAT_NUM varchar(8) NOT NULL,
FLAT_TYPE varchar(5),
FLAT_AREA numeric(10,2),
FLAT_AREA_UNITS varchar(10),
FLAT_FACING varchar(15),
FLAT_DESCRIPTION varchar(3000),
FLAT_STATUS varchar(20) NOT NULL,
Primary key (BLOCK_NAME, FLAT_NUM)
);

SELECT * FROM information_schema.tables; #show tables.
TRUNCATE TABLE flat_info;       #remove all DATA in table.
DROP TABLE flat_info;           #delete entire TABLE itself.
```

DDL statements

Tables:

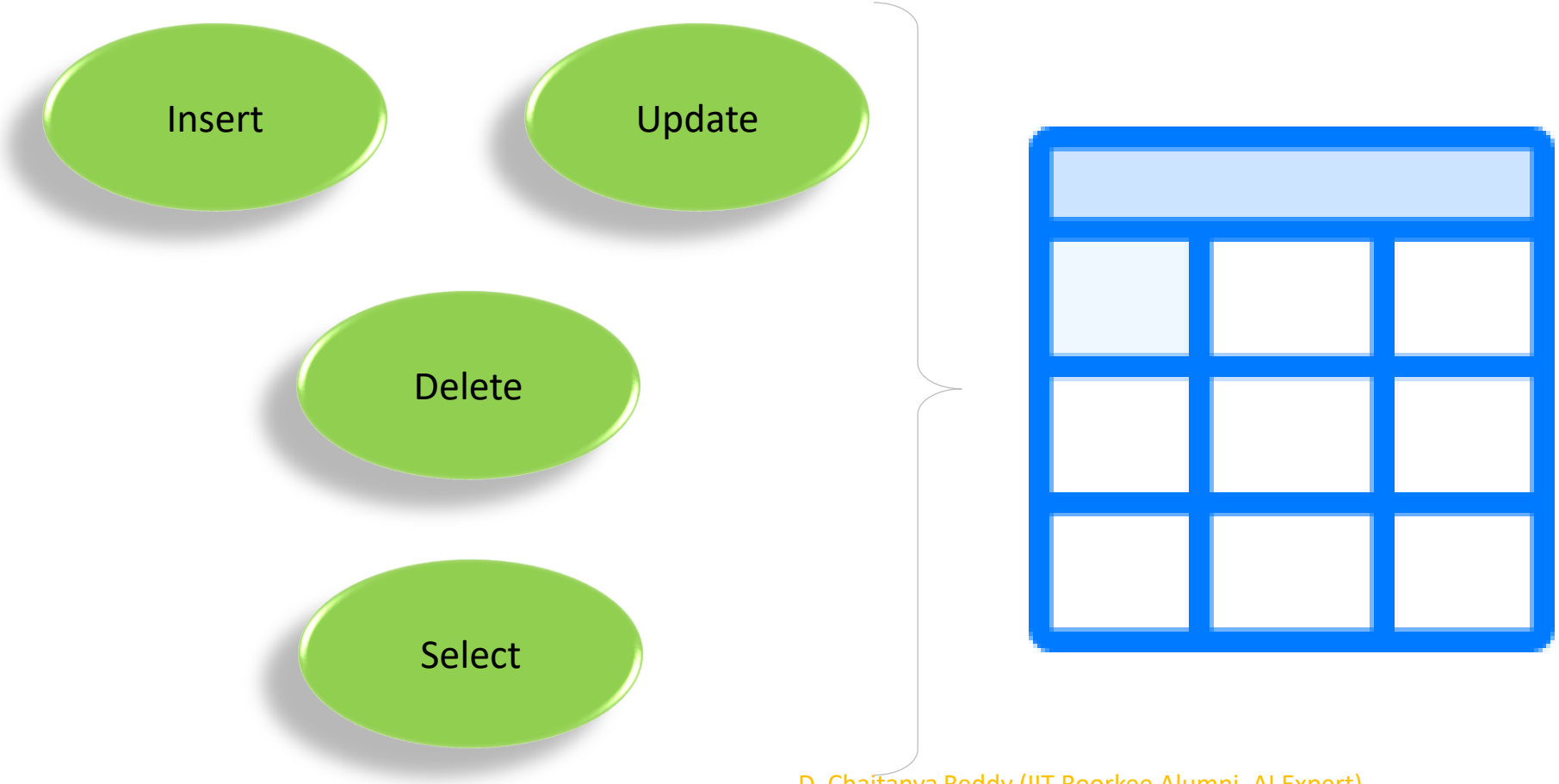
Syntax: ALTER TABLE table_name [alter_option ...];

```
ALTER TABLE flat_info ADD COLUMN Reg_date DATE; # add the new column as a last column.
ALTER TABLE flat_info ADD COLUMN STATUS VARCHAR(10) AFTER flat_type; # Add new column after a specified column.
ALTER TABLE flat_info ADD COLUMN ID VARCHAR(8) FIRST;  # Add new column at the starting
ALTER TABLE flat_info DROP COLUMN reg_date; # removes the specified column from table.

ALTER TABLE flat_info ADD INDEX demo_index (block_name, flat_num); # adding the index to table.
ALTER TABLE flat_info DROP INDEX demo_index; # drop index from a table.

ALTER TABLE flat_info DROP PRIMARY KEY; # drop primay key of a table.
ALTER TABLE flat_info ADD PRIMARY KEY (ID); # adding primary key to a table.

CREATE TABLE flats(FLAT_ID VARCHAR(8) PRIMARY KEY);
ALTER TABLE flat_info ADD FOREIGN KEY (ID) REFERENCES FLATS(FLAT_ID); # adding FK to table. SQL provides default FK name.
ALTER TABLE flat_info DROP FOREIGN KEY flat_info_ibfk_1; # drop FK from table. should use the SQL provided default FK name.

ALTER TABLE flat_info ADD CONSTRAINT own_fk FOREIGN KEY(ID) REFERENCES flats(FLAT_ID); # adding FK with given name.
ALTER TABLE flat_info DROP FOREIGN KEY own_fk; # drop foreign key from a table.# drop FK. should use the given FK name.
```

DML
Data Manipulation Language

DML

- It is the set of SQL statements used to manage data in table.



DML statements

INSERT:

```
INSERT INTO FLAT_INFO (BLOCK_NAME, FLAT_NUM, FLAT_TYPE, FLAT_AREA, FLAT_AREA_UNITS, FLAT_FACING, FLAT_DESCRIPTION, FLAT_STATUS)
VALUES ('Z', '101', '3BHK', '1590.00', 'Sq.ft', '', 'GOOD flat', 'Tenant Occupied');

INSERT INTO FLAT_INFO(BLOCK_NAME, FLAT_NUM, FLAT_STATUS) VALUES ('Z', '102', 'empty')

INSERT INTO FLAT_INFO VALUES ('Z', '103', '2BHK', '1210.00', 'Sq.ft', 'East', 'Average model', 'Owner Occupied');
```

Add_More_Table_Data.txt

SELECT:

```
SELECT * FROM flat_info;        #Fetch entire data (rows & columns) from table.

SELECT BLOCK_NAME, FLAT_NUM, FLAT_STATUS FROM flat_info;    #Fetch only specified columns information of all rows in a table.

SELECT CONCAT(BLOCK_NAME, FLAT_NUM), FLAT_STATUS FROM flat_info;      #concatenating multiple columns as a single column.

SELECT CONCAT(BLOCK_NAME, FLAT_NUM) AS FLAT_ID, FLAT_STATUS AS Status FROM flat_info;     #giving alias names for columns.
```

DML statements

UPDATE:

```
UPDATE flat_info SET flat_area_units = 'Sq.ft'`;  #update the entire column value.
UPDATE flat_info SET FLAT_AREA = 1222.0 WHERE FLAT_AREA IS NULL;      #update the given column value with where condition.
UPDATE flat_info SET FLAT_TYPE = '2BHK' WHERE FLAT_AREA=1222.0;
UPDATE flat_info SET FLAT_FACING = 'West', FLAT_DESCRIPTION='Not Available', FLAT_STATUS='Vacant' WHERE BLOCK_NAME='Z' AND
FLAT_NUM=102;
```

DELETE:

```
DELETE FROM flat_info;        #deletes all DATA from the table. same as TRUNCATE table.
```

Load_Entire_Table_Data.txt

```
DELETE FROM flat_info WHERE FLAT_STATUS='EMPTY';  #delete data from table with where condition.
```

DML statements

ORDER BY:

```
Syntax: SELECT column-list FROM table_name [WHERE condition] [ORDER BY column1, column2, .. columnN] [ASC | DESC];
```

```
SELECT * FROM flat_info WHERE FLAT_STATUS='Vacant' ORDER BY FLAT_AREA, FLAT_NUM DESC;
```

DISTINCT:

```
Syntax: SELECT DISTINCT column1, column2,.....columnN FROM table_name
```

```
SELECT DISTINCT FLAT_STATUS FROM flat_info;
SELECT DISTINCT FLAT_AREA, FLAT_TYPE, FLAT_STATUS FROM flat_info ORDER BY flat_area;
```

GROUP BY:

```
Syntax: SELECT column_name(s) FROM table_name GROUP BY column_name(s);
```

```
SELECT flat_facing, AVG(flat_area) as avg_flat_area FROM flat_info GROUP BY flat_facing;
```

DML statements

LIKE:

Wild character in SQL

```
SELECT * FROM flat_info WHERE flat_type LIKE '2%'

SELECT * FROM flat_info WHERE flat_status LIKE '%Occ%'
```

IN:

```
Syntax: WHERE column_name IN (value1, value2, value3, ...);
```

Load_Flat_Owner_Info_Table.txt

Nested **select** query

```
SELECT * FROM flat_info WHERE FLAT_FACING IN ('East', 'West');

SELECT * FROM flat_owner_info WHERE ID IN (SELECT CONCAT(block_name, flat_num) AS flat_id FROM flat_info WHERE flat_status = 'Not Occupied');
```

EXISTS:

```
Syntax: WHERE EXISTS (subquery);
```

First, the records from table in subquery will be filtered and the corresponding records from main table will be resulted based on sub_table.colum1 = main_table.column2

```
SELECT * FROM flat_owner_info WHERE EXISTS (SELECT * FROM flat_info WHERE flat_status='Vacant' AND flat_owner_info.ID = flat_info.FLAT_ID);
```

This condition is incorporated in nested select statement to check the same column in both tables.

DML statements

ANY, ALL: Used in nested `select` statements with `where` clause

```
SELECT column FROM  table1 WHERE column OPERATOR ANY (SELECT column FROM
table2);                              Operators are =, != <>, >, >=, <, <=
```

SQL **ANY** compares a value of the first table with all values of the second table and returns the row if there is a match with **any** value.

```
SELECT * FROM flat_owner_info WHERE flat_num > ANY (SELECT
flat_num FROM flat_info WHERE flat_status = 'Vacant' and
flat_facing = 'East')
```

```
SELECT column FROM  table1 WHERE column OPERATOR ALL (SELECT column FROM
table2);                                 Operators are =, != <>, >, >=, <, <=
```

SQL **ALL** compares a value of the first table with all values of the second table and returns the row if there is a match with **ALL** values.

```
SELECT * FROM flat_owner_info WHERE flat_num > ALL (SELECT
flat_num FROM flat_info WHERE flat_status = 'Vacant' and
flat_facing = 'East')
```

AND, OR: Used to implement multiple conditions in `where` clause

NOT:

```
SELECT * FROM flat_info WHERE NOT (FLAT_TYPE LIKE '2%');
```

```
UPDATE flat_info SET FLAT_FACING = NULL WHERE FLAT_NUM = 101;
```

IS NULL:

```
SELECT * FROM flat_info WHERE FLAT_FACING IS NULL;
```

NOT EQUAL:

```
SELECT * FROM flat_info WHERE FLAT_FACING <> 'North';
```

IS NOT NULL:

```
SELECT * FROM flat_info WHERE FLAT_FACING IS NOT NULL;
```

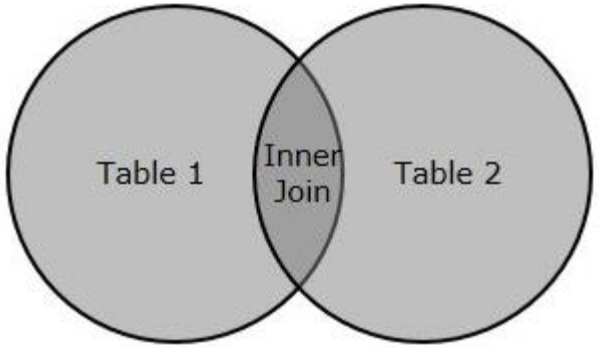
BETWEEN:

```
SELECT * FROM flat_info WHERE FLAT_AREA BETWEEN 2000 AND 3000;
SELECT * FROM flat_owner_info WHERE name BETWEEN 'C' AND 'R';
```

Load_Tables_For_Join.txt

DML statements

INNER JOIN:



Default is always Inner join in SQL

customers

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2,000.0
2	Khilan	25	Delhi	1,500.0
3	kaushik	23	Kota	2,000.0
4	Chaitali	25	Mumbai	6,500.0
5	Hardik	27	Bhopal	8,500.0
6	Komal	22	MP	4,500.0
7	Muffy	24	Indore	10,000.0

orders

OID	DATE	CUSTOMER_ID	AMOUNT
102	2009-10-08 ...	3	3,000.0
100	2009-10-08 ...	3	1,500.0
101	2009-11-20 ...	2	1,560.0
103	2008-05-20 ...	4	2,060.0
104	2008-05-21 ...	8	1,010.0

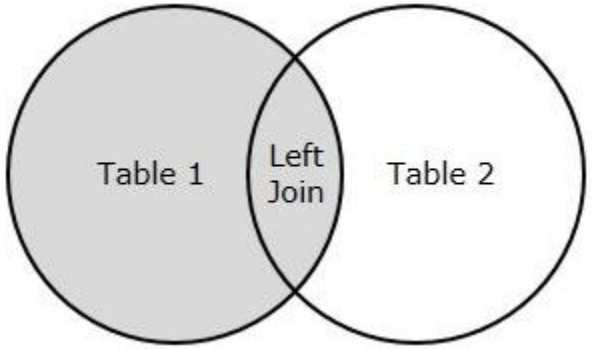
result

ID	NAME	AGE	ADDRESS	SALARY	OID	DATE	CUSTOMER_ID	AMOUNT
3	kaushik	23	Kota	2,000.0	102	2009-10-08 ...	3	3,000.0
3	kaushik	23	Kota	2,000.0	100	2009-10-08 ...	3	1,500.0
2	Khilan	25	Delhi	1,500.0	101	2009-11-20 ...	2	1,560.0
4	Chaitali	25	Mumbai	6,500.0	103	2008-05-20 ...	4	2,060.0

```
SELECT * FROM customers INNER JOIN orders ON customers.id = orders.customer_id;

SELECT * FROM customers JOIN orders ON customers.id = orders.customer_id;
```

DML statements

LEFT JOIN:



Left join is one of the outer join.

customers

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2,000.0
2	Khilan	25	Delhi	1,500.0
3	kaushik	23	Kota	2,000.0
4	Chaitali	25	Mumbai	6,500.0
5	Hardik	27	Bhopal	8,500.0
6	Komal	22	MP	4,500.0
7	Muffy	24	Indore	10,000.0

orders

OID	DATE	CUSTOMER_ID	AMOUNT
102	2009-10-08 ...	3	3,000.0
100	2009-10-08 ...	3	1,500.0
101	2009-11-20 ...	2	1,560.0
103	2008-05-20 ...	4	2,060.0
104	2008-05-21 ...	8	1,010.0

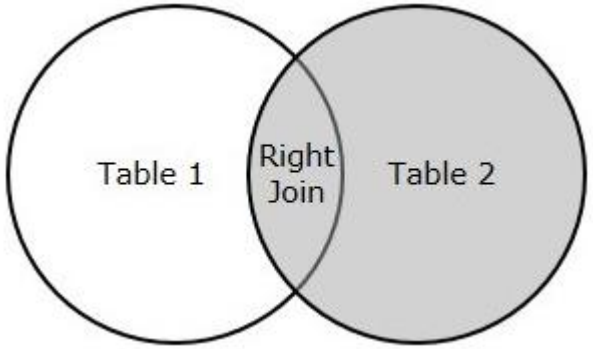
result

ID	NAME	AGE	ADDRESS	SALARY	OID	DATE	CUSTOMER_ID	AMOUNT
1	Ramesh	32	Ahmedabad	2,000.0	(NULL)	(NULL)	(NULL)	(NULL)
2	Khilan	25	Delhi	1,500.0	101	2009-11-20 00:00:00	2	1,560.0
3	kaushik	23	Kota	2,000.0	100	2009-10-08 00:00:00	3	1,500.0
3	kaushik	23	Kota	2,000.0	102	2009-10-08 00:00:00	3	3,000.0
4	Chaitali	25	Mumbai	6,500.0	103	2008-05-20 00:00:00	4	2,060.0
5	Hardik	27	Bhopal	8,500.0	(NULL)	(NULL)	(NULL)	(NULL)
6	Komal	22	MP	4,500.0	(NULL)	(NULL)	(NULL)	(NULL)
7	Muffy	24	Indore	10,000.0	(NULL)	(NULL)	(NULL)	(NULL)

```
SELECT * FROM customers LEFT JOIN orders ON customers.id = orders.customer_id;
```

DML statements

RIGHT JOIN:



Right join is another outer join.

customers

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2,000.0
2	Khilan	25	Delhi	1,500.0
3	kaushik	23	Kota	2,000.0
4	Chaitali	25	Mumbai	6,500.0
5	Hardik	27	Bhopal	8,500.0
6	Komal	22	MP	4,500.0
7	Muffy	24	Indore	10,000.0

orders

OID	DATE	CUSTOMER_ID	AMOUNT
102	2009-10-08 ...	3	3,000.0
100	2009-10-08 ...	3	1,500.0
101	2009-11-20 ...	2	1,560.0
103	2008-05-20 ...	4	2,060.0
104	2008-05-21 ...	8	1,010.0

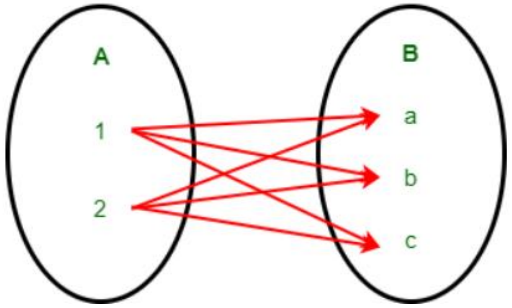
result

ID	NAME	AGE	ADDRESS	SALARY	OID	DATE	CUSTOMER_ID	AMOUNT
3	kaushik	23	Kota	2,000.0	102	2009-10-08 00:00:00	3	3,000.0
3	kaushik	23	Kota	2,000.0	100	2009-10-08 00:00:00	3	1,500.0
2	Khilan	25	Delhi	1,500.0	101	2009-11-20 00:00:00	2	1,560.0
4	Chaitali	25	Mumbai	6,500.0	103	2008-05-20 00:00:00	4	2,060.0
(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	104	2008-05-21 00:00:00	8	1,010.0

```
SELECT * FROM customers RIGHT JOIN orders ON customers.id = orders.customer_id;
```

DML statements

CROSS JOIN:



Cross join is cartesian product of all rows in both tables.

customers

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2,000.0
2	Khilan	25	Delhi	1,500.0
3	kaushik	23	Kota	2,000.0
4	Chaitali	25	Mumbai	6,500.0
5	Hardik	27	Bhopal	8,500.0
6	Komal	22	MP	4,500.0
7	Muffy	24	Indore	10,000.0

orders

OID	DATE	CUSTOMER_ID	AMOUNT
102	2009-10-08 ...	3	3,000.0
100	2009-10-08 ...	3	1,500.0
101	2009-11-20 ...	2	1,560.0
103	2008-05-20 ...	4	2,060.0
104	2008-05-21 ...	8	1,010.0

result

See the result in Heidi SQL

```
SELECT * FROM customers CROSS JOIN orders;
```

Python DB Libraries

Interacting with DB using PYTHON

mysql-connector-python

In Command Prompt

pip install **mysql-connector-python**

Connect to DB

Create Cursor

Execute Query

Close cursor, connection

1

2

3

4

Need the below details

- Username
- Password
- Hostname (IP/localhost)
- Database name

To execute SQL Queries in Python

Database vs Python Connector Library:

Oracle: pip install **oracledb**

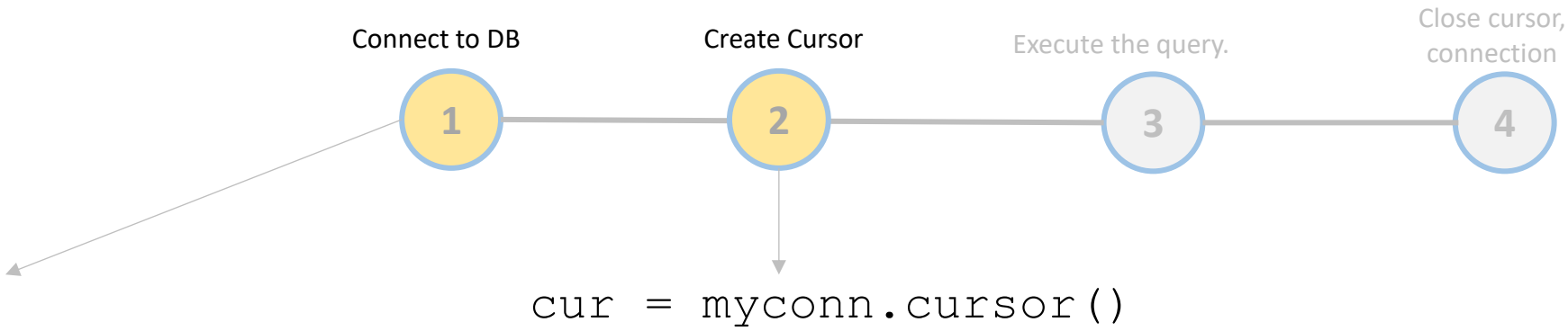
MYSQL: pip install **mysql-connector-python**

MSSQL: pip install **pymssql**

MongoDB: pip install **mongo-connector**

Postgres: pip install **psycopg2**

Interacting with DB using PYTHON




```
import mysql.connector


username = 'root'
password = 'python'
host = 'localhost'
ip = '127.0.0.1'
port = 3306
database = 'db_myaptz'
```

Method-1:

```
myconn = mysql.connector.connect(host = host, user = username, password = password, database = database)
```

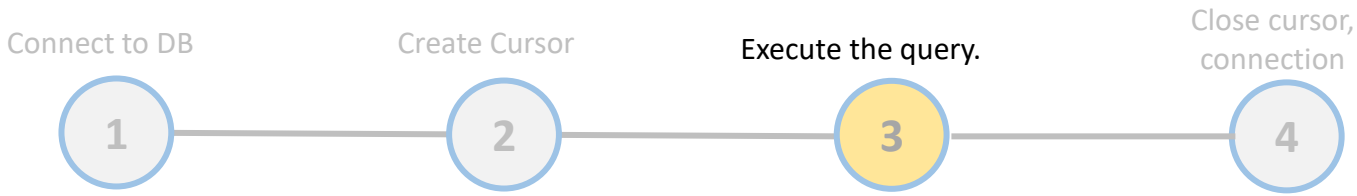
Method-2:

```
config = {'user': username, 'password': password, 'host': ip, 'database': database}
myconn = mysql.connector.connect(**config)
```

Method-3:

```
from mysql.connector import connection
myconn = connection.MySQLConnection(user = username, password = password, host=ip, database = database)
```

Interacting with DB using PYTHON



Method-1: For Fetch Queries

```
cur.execute('SHOW DATABASES')
for res in cur:
   print(res)
```

Method-2: For Fetch Queries

```
cur.execute('SELECT * FROM FLAT_INFO')
res = cur.fetchall()

for el in res:
   print(el)
```

Method-3: For Update Queries

```
cur.execute('UPDATE FLAT_INFO SET FLAT_AREA=1410.0 WHERE FLAT_NUM=\'103\'')
myconn.commit()
```

Method-4: For Update Queries

```
cur.execute('UPDATE FLAT_INFO SET FLAT_AREA=1310.0 WHERE FLAT_NUM="115"')
cur.execute('UPDATE FLAT_INFO SET FLAT_FACING="West" WHERE FLAT_FACING IS NULL')
cur.execute('UPDATE FLAT_INFO SET FLAT_DESCRIPTION="Perfect for 6 members family." WHERE FLAT_NUM="108"')
myconn.commit()
```

Interacting with DB using PYTHON

Closing Cursor:

```
cur.close()
```

Closing Connection :

```
myconn.close()
```

Interacting with DB using PYTHON

Few important methods in **Connection** Class:

`close()`:
Close the connection now

`commit()`:
Commit any pending transaction to the database

`cursor()`:
return a new Cursor Object using the connection.

`rollback()`:
Database to roll back to the start of any pending transaction.

Prerequisite:
Database should provide transaction support.

Caution:
Closing a connection without committing the changes first will cause an implicit rollback to be performed

Few important methods in **Cursor** Class:

`close()`:
Close the cursor now.

`execute()`:
Prepare and execute a database operation.

`fetchall()`:
Fetch all (remaining) rows of a query result (list of tuples).

`fetchone()`:
Fetch the next row of a query result set (tuple).

`fetchmany()`:
Fetch the next set of rows of a query result (list of tuples).

E N D