SQL:

1. **conn** = **sqlite3.connect**(gdrivePath+'employee.db')
2. **pd.read_sql_query**(query, conn, index_col =)

3.CREATE TABLE

```
query = '''
CREATE TABLE employee(
    id integer PRIMARY KEY,
    name text,
    salary real
);
'''
```

4. **INSERT**

```
query = '''
INSERT INTO employee (id, name, salary)
VALUES
    (1, ' A', 100.0),
    (2, 'B', 105.5),
    (3, 'C', 110.7);
'''
```

BASIC

5. **SELCT**

```
query = '''
SELECT (DISTINCT) salary, count() AS num
FROM employee
WHERE last_name LIKE 'K_K%'
GROUP BY salary
HAVING num > 100
ORDER BY num
LIMIT 1 OFFSET 2
```

6. **ALTER (ADD,DRO, RENAME)**

```
ALTER TABLE table_name
ADD COLUMN col_name data_type
DROP COLUMN col_name
RENAME COLUMN old_name TO new
```

7. **UPDATE**

```
query = ' UPDATE employee SET genre = 'Fiction' WHERE type = 'science'
```

CREATE 1

8. **DELETE:** query = ' **DELETE FROM** employee **WHERE** lastname = 'Brown' '
9. **DROP:** query = '**DROP TABLE IF EXISTS** table_name'
10. **to_sql**

```
oil_imports.to_sql('oil_imports',conn2)
```

11. **GROUP_CONCAT()**
12. **SUBSTRING(column, where to start(count start 1), how many bit)**
13. **conn.execute("CREATE TABLE leaders AS" + query)**

TRIGGER:

INSERT 1

```
s="""
CREATE TRIGGER update_customer_address UPDATE OF address ON customers
  BEGIN
    UPDATE orders SET customer_address = new.address WHERE customer_name = old.name;
  END;
"""
conn.executescript(s)
```

VIEW

```
t_q = '''
CREATE TRIGGER del_b1 AFTER DELETE ON b1
FOR EACH ROW
  BEGIN
    DELETE FROM enrollment WHERE student_id = OLD.student_id;
  END;
'''
execSQL(conn," DROP TRIGGER IF EXISTS sub_enrol_trigger1 ")
s="""
CREATE TRIGGER sub_enrol_trigger1 Instead of DELETE ON Biology_1
  BEGIN
    DELETE FROM sub_enrol WHERE student_id = old.student_id & subject_id='Bio_1';
  END;
"""
```

```
import sqlite3
import os
import pandas as pd

from google.colab import drive
drive.mount('/content/drive')
path = "/content/drive/My Drive/DataManagement/data files/"

def execSQL(conn, query):
  conn.execute(query)
  conn.commit()

def allrowSelect(conn, query):
  cursor = conn.execute(query)
  for row in cursor:
    print(row)

if os.path.exists(path+'student.db'):
  os.remove(path+'student.db')
conn = sqlite3.connect(path+'student.db')
execSQL(conn,"DROP TABLE if exists students")
q='''
CREATE TABLE students
(
  uni text PRIMARY KEY,
  student_name text NOT NULL,
  year_in_school CHECK( year_in_school IN ('Freshman', 'Sophomore'. 'Junior'. 'Senior'))
)
'''
execSQL(conn,q)

q='''
INSERT INTO students (uni, student_name, year_in_school
SELECT distinct uni, student_name, year_in_school
FROM sample_enrollments
'''
execSQL(conn,q)
pd.read_sql_query("select * from students",conn)
```

FOREIGN KEY

```
q='''PRAGMA foreign_keys=on'''
execSQL(conn,q)
```

TABLES/VIEWS

```
pd.read_sql_query('''select name from sqlite_master where type='table' or type='view' ''',conn)
```

SHOW ALL of attributes: Name: *->name

```
pd.read_sql_query(''' select * from pragma_table_info('course') ''', conn)
```

2

```
query = ''' create table sub_enrol_temp as
select student_name, subject1 from students where subject1 IS NOT NULL
union
select student_name, subject2 from students where subject2 IS NOT NULL
union
select student_name, subject3 from students where subject3 IS NOT NULL
union
select student_name, subject4 from students where subject4 IS NOT NULL
'''
```

3

```
CREATE TABLE Employee
(
ID INT PRIMARY KEY,
NAME TEXT NOT NULL,
deptid INT REFERENCES DEPARTMENT(DEPT_ID),
SALARY INT CHECK(SALARY > 22000 AND SALARY < 50000),
INSURANCE_OPTED TEXT DEFAULT 'YES'
);'''
```

```
NAME TEXT UNIQUE
promoted BOOLEAN DEFAULT True
```

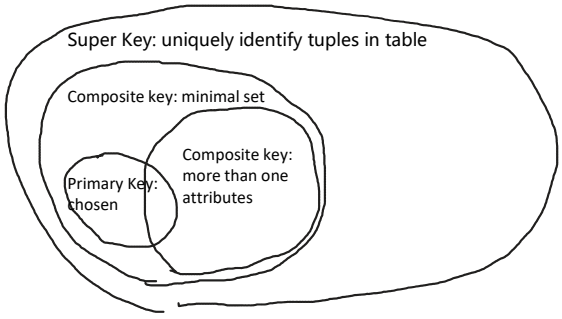
2

```
query='''
insert into subject (subject_id, subject_name)

SELECT 'Bio_1', (select subject_name from subject_temp where subject_id=1)
union
SELECT 'Bio_2', (select subject_name from subject_temp where subject_id=2)
union
SELECT 'Busi_1', (select subject_name from subject_temp where subject_id=3)
union
SELECT 'Eco_1', (select subject_name from subject_temp where subject_id=4)
union
SELECT 'Progra_2', (select subject_name from subject_temp where subject_id=5)
'''
```

```
execSQL(conn,"DROP VIEW if exists Biology_1")
query = """
CREATE VIEW Biology_1 AS

SELECT sub_enrol.subject_id, stu_info.student_name, stu_info.student_id
FROM sub_enrol LEFT JOIN stu_info ON stu_info.student_id = sub_enrol.student_id
WHERE sub_enrol.subject_id = (select subject_id from subject where subject_name like 'Biology1%')
"""
execSQL(conn,query)
```



* **Primary Key**: the candidate key selected by the database administrator to uniquely identify tuples in a table. It was assigned when creating a table, should be unique, and its value should never be NULL.
* **Composite Key**: the candidate key or primary key that consists of more than one attributes that together uniquely identify a tuple in a table. It's a subsets of super keys which have two or more attributes. It can be a candidate key if it is minimal.
* **Foreign Key**: an attribute which is a primary key in its parent table, but is included as an attribute in another table
* **Super Key**: A single key or a group of multiple keys that can uniquely identify tuples in a table. It's a superset of a candidate key (not necessary minimal), and a table can have many super keys
* **Candidate Key**: A single key or a group of multiple keys that uniquely identify rows in a table. Its a minimal super key (a subset of super key), which means a super key with no redundant attributes. And it is not allowed to have NULL values

1NF: Primary Key, unique, not null, no duplicate row
2NF: (1) No partial dependencies: non-prime totally depend on prime key
3NF: (1) No transitive dependencies (one non-prime attribute is dependent on the primary key through another non-prime attribute)
(2) If only one non-key attribute, its already in 3NF
3.5NF (Boyce-Codd): 90% 3NF table is also 3.5NF since 3.5NF is created to reduce redundant data in non-prime attribute
(1) for every non-trivial functional dependency X->B, X is a superkey of R (must not be many to many)
4NF: (always in relation table): eliminate independent **multivalued** factors about entity stored in the same table.

Employee	Skill	Language
Smith	cook	French
Smith	cook	German
Smith	cook	Greek
Smith	type	French
Smith	type	German
Smith	type	Greek

since no relationship between skill and language, and multivalued

Employee	Skill
Smith	cook
Smith	type

Employee	Language
Smith	French
Smith	German
Smith	Greek

if 2 tables have same primary keys, combine them together (shouldn't split)

Table should only have 1 entity to be full dependent (avoid partial dependence) except the table for relation

constrains: primary key, foreign key, value range, unique value