Unit-3: Database backup and CSV handling:

3.1 SQLite dump :

- 3.1.1 Dump specific table into file, Dump only table structure
- 3.1.2 Dump entire database into file
- 3.1.3 Dump data of one or more tables into a file

3.2 CSV files handling:

- 3.2.1 Import a CSV file into a table
- 3.2.2 Export a CSV file from table
- 3.3 Python Connectivity with different types of databases

3.1 SQLite dump

- **SQL** stands for **Structured Query Language**, which is the common language used to manipulate relational databases.
- It is used to **create**, **store**, **retrieve**, and **manipulate** databases and tables.
- SQLite is a lightweight version of SQL with some major changes, it doesn't have a separate server, it is not a common language, and it can't connect with databases like **Oracle** or **MySQL** server.

SQLite DUMP

- The **DUMP** command in SQLite is used to **Backup** or **Restore any Database**, **Table**, or **Schema**.
- But unlike other Functions of SQLite, the DUMP command is used with "**.**" (**Dot**) in front of it.
- These types of SQLite commands are called **Dot-Commands**.
- DUMP commands are used to **dump** Tables, Databases, Schemas, etc to some other file.

How To Use The SQLite Dump Command

The **DUMP** command is a **Dot-Command** used in **SQLite3**, so it is used with a **preceding dot**.

Syntax:

```
.dump <table_name> [OPTIONAL]
```

Explanation:

- The <**table_name**> after the **DUMP** command must be provided whenever the user want to just dump a **single Table** and not the **entire Database**.
- If we use the **.dump** command singularly, then all the SQL statements used will be given as output in the same Command Line, **it will not be saved anywhere.**

- To understand the DUMP command in more depth we need a 2 table on which we will perform queries.
- Here we have table called **Employees** and **Students.**
- After inserting data into the table, Our table looks:

Employees Table:

```
sqlite> SELECT * FROM Employees;
empID  FirstName  LastName   Salary  Location
-----  ---------  ---------  ------  --------
1      Sonia      Wong       20000   AL
2      Neel       Lee        25000   FL
3      Melody     Abott      23000   IA
4      Trinity    Kirk       21000   IL
5      Miley      Webster    28000   IN
6      Sydnee     Donaldson  27000   KY
7      Matilda    Roach      35000   MN
8      Chanel     Mcneil     33000   MI
9      Gilberto   Blake      34000   MS
10     Harmony    Serrano    32000   NV
```

Students Table:

```
sqlite> select * from students;
studID  FirstName  LastName    Class  Section
------  ---------  ----------  -----  -------
10      Vivek      Singh       7      B
12      Manish     Roy         8      A
15      Dilip      Mukherjee   10     A
16      Souvik     Sen         10     B
18      Rohit      Das         10     A
21      Mohit      Shetty      9      A
22      Raj        Banerjee    9      B
24      Biswajit   Das         7      B
25      Srijit     Roy         8      A
27      Rakesh     Chatterjee  8      C
```

Now, if we use the dump command now, without mentioning anything else, then all the SQL statements used to create and populate the table will be given as output.

```
.dump
```

```
sqlite> .dump
PRAGMA foreign_keys=OFF;
BEGIN TRANSACTION;
CREATE TABLE Employees (
empID INTEGER,
FirstName TEXT,
LastName TEXT,
Salary INTEGER,
Location TEXT
);
INSERT INTO Employees VALUES(1,'Sonia','Wong',20000,'AL');
INSERT INTO Employees VALUES(2,'Neel','Lee',25000,'FL');
INSERT INTO Employees VALUES(3,'Melody','Abott',23000,'IA');
INSERT INTO Employees VALUES(4,'Trinity','Kirk',21000,'IL');
INSERT INTO Employees VALUES(5,'Miley','Webster',28000,'IN');
INSERT INTO Employees VALUES(6,'Sydnee','Donaldson',27000,'KY');
INSERT INTO Employees VALUES(7,'Matilda','Roach',35000,'MN');
INSERT INTO Employees VALUES(8,'Chanel','Mcneil',33000,'MI');
INSERT INTO Employees VALUES(9,'Gilberto','Blake',34000,'MS');
INSERT INTO Employees VALUES(10,'Harmony','Serrano',32000,'NV');
CREATE TABLE Students (
studID INTEGER,
FirstName TEXT,
LastName TEXT,
Class INTEGER,
Section TEXT
);
INSERT INTO Students VALUES(10,'Vivek','Singh',7,'B');
INSERT INTO Students VALUES(12,'Manish','Roy',8,'A');
INSERT INTO Students VALUES(15,'Dilip','Mukherjee',10,'A');
INSERT INTO Students VALUES(16,'Souvik','Sen',10,'B');
INSERT INTO Students VALUES(18,'Rohit','Das',10,'A');
INSERT INTO Students VALUES(21,'Mohit','Shetty',9,'A');
INSERT INTO Students VALUES(22,'Raj','Banerjee',9,'B');
INSERT INTO Students VALUES(24,'Biswajit','Das',7,'B');
INSERT INTO Students VALUES(25,'Srijit','Roy',8,'A');
INSERT INTO Students VALUES(27,'Rakesh','Chatterjee',8,'C');
COMMIT;
```

Explanation: Using the DUMP dot command, it alone returns all the commands used till now to create and populate the table. It returns the output as a transaction, that's why the BEGIN TRANSACTION and COMMIT is being displayed here which is not required while writing the actual commands.

Dump Entire Database Into file using DUMP Command

- We will dump an entire database into a file using the DUMP command.
- To do that firstly, we need to provide the **name** and **extension of the file** in which we want to store the result with the **.output** command.
- Then we have to use the **.dump** command.

Query:

```
.output <Location of File with Filename.Extension>
.dump
```

- Now, we will store all those commands i.e the output of the **.dump** command to a text file named **Employee_Details.txt.**
- Write the below commands one by one, just replace the file's location with the user's one:

Query:

```
.output C:\Users\user\Downloads\SQLite\Database_Details.txt
.dump
```

Output:

```
sqlite> .output C:\Users\user\Downloads\SQLite\Database_Details.txt
sqlite> .dump
```

Content saved in the Output File:

Database_Details - Notepad

File Edit Format View Help

```
PRAGMA foreign_keys=OFF;
BEGIN TRANSACTION;CREATE TABLE Employees
(empID INTEGER,FirstName TEXT,LastName TEXT,Salary INTEGER,Location TEXT);
INSERT INTO Employees VALUES(1,'Sonia','Wong',20000,'AL');
INSERT INTO Employees VALUES(2,'Neel','Lee',25000,'FL');
INSERT INTO Employees VALUES(3,'Melody','Abott',23000,'IA');
INSERT INTO Employees VALUES(4,'Trinity','Kirk',21000,'IL');
INSERT INTO Employees VALUES(5,'Miley','Webster',28000,'IN');
INSERT INTO Employees VALUES(6,'Sydnee','Donaldson',27000,'KY');
INSERT INTO Employees VALUES(7,'Matilda','Roach',35000,'MN');
INSERT INTO Employees VALUES(8,'Chanel','Mcneil',33000,'MI');
INSERT INTO Employees VALUES(9,'Gilberto','Blake',34000,'MS');
INSERT INTO Employees VALUES(10,'Harmony','Serrano',32000,'NV');
CREATE TABLE Students (
studID INTEGER,
FirstName TEXT,
LastName TEXT,
Class INTEGER,
Section TEXT
);
INSERT INTO Students VALUES(10,'Vivek','Singh',7,'B');
INSERT INTO Students VALUES(12,'Manish','Roy',8,'A');
INSERT INTO Students VALUES(15,'Dilip','Mukherjee',10,'A');
INSERT INTO Students VALUES(16,'Souvik','Sen',10,'B');
INSERT INTO Students VALUES(18,'Rohit','Das',10,'A');
INSERT INTO Students VALUES(21,'Mohit','Shetty',9,'A');
INSERT INTO Students VALUES(22,'Raj','Banerjee',9,'B');
INSERT INTO Students VALUES(24,'Biswajit','Das',7,'B');
INSERT INTO Students VALUES(25,'Srijit','Roy',8,'A');
INSERT INTO Students VALUES(27,'Rakesh','Chatterjee',8,'C');
COMMIT;
```

Dump Specific Table Using Dump Command

- We will see how we can use the dump command to dump a **specific table of the database into a text file**, not the entire database.
- Sometimes, we might need to dump only a specific table into another file, not the entire database.
- For that, the syntax will change a bit.
- We need to provide the output filename as last time, but with the dump command, we need to provide the name of the specific table, which we want to dump.

Syntax:

```
.output <Location of File with Filename.Extension>
.dump <Table_Name>
```

Query:

Here, we will dump just the Employees table into a file named **Employee_Table.txt**

```
.output C:\Users\user\Downloads\SQLite\Employee_Table.txt
.dump Employees
```

```
sqlite> .output C:\Users\user\Downloads\SQLite\Employee_Table.txt
sqlite> .dump Employees
```

Content of the Employee_Table Text File After Dumping:

Employee_Table - Notepad

File Edit Format View Help

```
PRAGMA foreign_keys=OFF;
BEGIN TRANSACTION;
CREATE TABLE Employees (empID INTEGER,FirstName TEXT,LastName TEXT,Salary INTEGER,Location TEXT);
INSERT INTO Employees VALUES(1,'Sonia','Wong',20000,'AL');
INSERT INTO Employees VALUES(2,'Neel','Lee',25000,'FL');
INSERT INTO Employees VALUES(3,'Melody','Abott',23000,'IA');
INSERT INTO Employees VALUES(4,'Trinity','Kirk',21000,'IL');
INSERT INTO Employees VALUES(5,'Miley','Webster',28000,'IN');
INSERT INTO Employees VALUES(6,'Sydnee','Donaldson',27000,'KY');
INSERT INTO Employees VALUES(7,'Matilda','Roach',35000,'MN');
INSERT INTO Employees VALUES(8,'Chanel','Mcneil',33000,'MI');
INSERT INTO Employees VALUES(9,'Gilberto','Blake',34000,'MS');
INSERT INTO Employees VALUES(10,'Harmony','Serrano',32000,'NV');
COMMIT;
```

As we can see, only the details of the **Employees** table has been saved / dumped. Not the entire Database.

Dump Tables Structure Only Using Schema Command

- We will see how we can only dump the **schmea** i.e the **structure of the table**.
- There is a variation of this command, using it **alone will store the schema of all the tables present in database**, but **if we pass the name of a specific table then it will only store that.**
- Now, we will not dump the contents of any table, but we will dump the structures of the table using the **.schema** command.
- **.schema** is a dot-command same as of the **.dump** command, it is used to **dump only the structure** or schema of the tables, just like the .dump command, it is used after the **.output** command.

Syntax:

```
.output <Location of File with Filename.Extension>
.schema
```

Query:

We will store the structure in a file called **Table_Structure.txt**. Write the below command to Dump the tables structure using the schema command:

```
.output C:\Users\user\Downloads\SQLite\Table_Structure.txt
.schema
```

```
sqlite> .output C:\Users\user\Downloads\SQLite\Table_Structure.txt
sqlite> .schema
```

Contents of the Table_Structure.txt:

Table_Structure - Notepad

File Edit Format View Help

```
CREATE TABLE Employees (empID INTEGER,FirstName TEXT,LastName TEXT,Salary INTEGER,Location TEXT);
CREATE TABLE Students (studID INTEGER,FirstName TEXT,LastName TEXT,Class INTEGER,Section TEXT);
```

If we want to store the structure of a specific Table of the database, then we need to provide the table name after the **.schema** command.

Syntax:

```
.output <Location of File with Filename.Extension>
.schema <Table_Name>
```

Query:

Saving the Schema of the **Students** table:

```
.output C:\Users\user\Downloads\SQLite\Student_Structure.txt
.schema Students
```

Contents of the Student_Structure.txt file:

Student_Structure - Notepad

File Edit Format View Help

```
CREATE TABLE Students (studID INTEGER,FirstName TEXT,LastName TEXT,Class INTEGER,Section TEXT);
```

Dump Data of One or More Tables Into File

- We will see how we can dump the data of one or more tables into a file using the dump command, whenever we will use the SELECT statement, in the file we mentioned, all the INSERT commands we have used previously will be stored.
- To dump the data of one or more tables into a file, we first need to change the mode into **insert,** this will ensure that whenever we use the select statement in command line, in the file all the insert statements will be dumped. We also need to provide a filename with an extension with the **.output** dot-command to point out in which file the results will be stored.

Syntax:

```
.mode insert
.output <filename.extension>
```

Query:

Now if we run the select statement to fetch the values of Employees table only, all the insert statement used for the Employees table, will be dumped and saved in the file mentioned.

```
.mode insert
.output mydata.txt
```

Query:

Now using the Select statement:

```
SELECT * FROM Employees;
```

If we now run the select statement again but for the **Students** Table, the content will be updated as below:

Output:

mydata - Notepad

File Edit Format View Help

```
INSERT INTO "table"(empID,FirstName,LastName,Salary,Location) VALUES(1,'Sonia','Wong',20000,'AL');
INSERT INTO "table"(empID,FirstName,LastName,Salary,Location) VALUES(2,'Neel','Lee',25000,'FL');
INSERT INTO "table"(empID,FirstName,LastName,Salary,Location) VALUES(3,'Melody','Abott',23000,'IA');
INSERT INTO "table"(empID,FirstName,LastName,Salary,Location) VALUES(4,'Trinity','Kirk',21000,'IL');
INSERT INTO "table"(empID,FirstName,LastName,Salary,Location) VALUES(5,'Miley','Webster',28000,'IN');
INSERT INTO "table"(empID,FirstName,LastName,Salary,Location) VALUES(6,'Sydnee','Donaldson',27000,'KY');
INSERT INTO "table"(empID,FirstName,LastName,Salary,Location) VALUES(7,'Matilda','Roach',35000,'MN');
INSERT INTO "table"(empID,FirstName,LastName,Salary,Location) VALUES(8,'Chanel','Mcneil',33000,'MI');
INSERT INTO "table"(empID,FirstName,LastName,Salary,Location) VALUES(9,'Gilberto','Blake',34000,'MS');
INSERT INTO "table"(empID,FirstName,LastName,Salary,Location) VALUES(10,'Harmony','Serrano',32000,'NV');
INSERT INTO "table"(studID,FirstName,LastName,Class,Section) VALUES(10,'Vivek','Singh',7,'B');
INSERT INTO "table"(studID,FirstName,LastName,Class,Section) VALUES(12,'Manish','Roy',8,'A');
INSERT INTO "table"(studID,FirstName,LastName,Class,Section) VALUES(15,'Dilip','Mukherjee',10,'A');
INSERT INTO "table"(studID,FirstName,LastName,Class,Section) VALUES(16,'Souvik','Sen',10,'B');
INSERT INTO "table"(studID,FirstName,LastName,Class,Section) VALUES(18,'Rohit','Das',10,'A');
INSERT INTO "table"(studID,FirstName,LastName,Class,Section) VALUES(21,'Mohit','Shetty',9,'A');
INSERT INTO "table"(studID,FirstName,LastName,Class,Section) VALUES(22,'Raj','Banerjee',9,'B');
INSERT INTO "table"(studID,FirstName,LastName,Class,Section) VALUES(24,'Biswajit','Das',7,'B');
INSERT INTO "table"(studID,FirstName,LastName,Class,Section) VALUES(25,'Srijit','Roy',8,'A');
INSERT INTO "table"(studID,FirstName,LastName,Class,Section) VALUES(27,'Rakesh','Chatterjee',8,'C');
```

3.2 CSV files handling:

3.2.1 Import a CSV file into a table

- Open database in cmd. “.open database_name”
- Create Table “Create table table_name(….)”
- Create a CSV file with excel with same Tables header.
- In CMD, “.mode csv”
- “import file_name table_name”
- Check using select query.
- Data from CSV file is imported in table of sqlite3.

```
sqlite> .open company.db
sqlite> .tables
dept      emp_tbl
sqlite> create table items (item_id INT, item_name TEXT(50), price INT);
sqlite> .tables
dept      emp_tbl  items
sqlite> .mode csv
```

```
sqlite> .import item1.csv items
sqlite> select * from items;
101,pencil,10
102,eraser,5
103,scale,10
104,sharpner,5
sqlite>
```

CSV File

101	pencil	10
102	eraser	5
103	scale	10
104	sharpner	5

3.2.2 Export a CSV file from table

- We just adding some rows to our table names ”items”.
- For export table to csv file.
- First, .mode csv
- .header on (for including column title to csv file)
- .output item2.csv (for create a csv file named item2 in default folder where we open a sqlite3)
- Select * from items; (for export or store data into items.csv file)

```
sqlite> insert into items values (105, 'compass', 100);
sqlite> insert into items values (106, 'notebook', 50);
sqlite> insert into items values (107, 'drawingbook', 60);
sqlite> insert into items values (108, 'craftpaper', 10);
sqlite> insert into items values (109, 'crayons', 80);
sqlite> insert into items values (110, 'colors', 70);
sqlite> .mode csv
sqlite> .header on
sqlite> .output item2.csv
sqlite> select * from items;
```

item_id	item_nam	price
101	pencil	10
102	eraser	5
103	scale	10
104	sharpner	5
105	compass	100
106	notebook	50
107	drawingbo	60
108	craftpaper	10
109	crayons	80
110	colors	70

3.3 Python Connectivity with different types of databases

- Connecting Python to different types of databases involves using specific libraries or modules that provide the interface to interact with each type of database.
- It Shows how to connect Python with some common types of databases:
- SQLite, MySQL, PostgreSQL, and MongoDB

Python SQLite

- Python SQLite3 module is used to integrate the SQLite database with Python.
- It is a standardized Python DBI API 2.0 and provides a straightforward and simple-to-use interface for interacting with SQLite databases.
- There is no need to install this module separately as it comes along with Python after the 2.5x version.

```
import sqlite3

# Connect to SQLite database (or create it if it doesn't exist)
conn = sqlite3.connect('example.db')

# Create a cursor object using the cursor() method
cursor = conn.cursor()

# Execute SQL queries using the cursor
cursor.execute("CREATE TABLE IF NOT EXISTS users (id INTEGER PRIMARY
KEY, name TEXT, email TEXT)")
cursor.execute("INSERT INTO users (name, email) VALUES ('John Doe',
'john@example.com')")
```

```
# Commit your changes in the database
conn.commit()

# Close the connection
conn.close()
```

2. MySQL

For MySQL or MariaDB, you can use the `mysql-connector-python` package. Install it via pip:

cmd

```
pip install mysql-connector-python
```

Then, you can connect to your MySQL/MariaDB database like this:

```
import mysql.connector

# Connect to MySQL/MariaDB database
conn = mysql.connector.connect(user='username',
password='password', host='127.0.0.1', database='mydatabase')

# Create a cursor object
cursor = conn.cursor()
```

```
# Execute SQL queries
cursor.execute("CREATE TABLE IF NOT EXISTS users (id INT AUTO_INCREMENT PRIMARY KEY, name VARCHAR(255), email VARCHAR(255))")
cursor.execute("INSERT INTO users (name, email) VALUES ('Jane Doe', 'jane@example.com')")

# Commit and close
conn.commit()
conn.close()
```

3. PostgreSQL

For PostgreSQL, the `psycopg2` package is commonly used. Install it using pip:

```bash
pip install psycopg2-binary
```

Then, connect to PostgreSQL:

```
import psycopg2

# Connect to PostgreSQL database
conn = psycopg2.connect("dbname='mydatabase' user='username'
host='localhost' password='password'")

# Create a cursor object
cursor = conn.cursor()

# Execute SQL queries
cursor.execute("CREATE TABLE IF NOT EXISTS users (id SERIAL PRIMARY
KEY, name VARCHAR(255), email VARCHAR(255))")
cursor.execute("INSERT INTO users (name, email) VALUES ('Bob Doe',
'bob@example.com')")
```

```
# Commit and close
conn.commit()
conn.close()
```

4. MongoDB

MongoDB is a NoSQL database. To connect to MongoDB, use the `pymongo` package. First, install it:

bash
pip install pymongo

Then, connect to MongoDB:

```
from pymongo import MongoClient

# Connect to MongoDB
client = MongoClient('localhost', 27017)

# Select database and collection
db = client['mydatabase']
collection = db['users']

# Insert a document
user_doc = {"name": "Alice Doe", "email": "alice@example.com"}
collection.insert_one(user_doc)
```