**13. Using Databases in Python**

Week 4

1 April 2019

1. Introduction to Databases

<https://sqlite.org/index.html>

2. Database Terminology

Field : basic unit of storage - has type value - called blob Binary large Object

3. 4. 5. Sqlite3 Install on Windows

<https://sqlite.org/download.html>

**Precompiled Binaries for Windows**

[**Sqlite-tools-win32-x86-3270200.zip**](https://sqlite.org/2019/sqlite-tools-win32-x86-3270200.zip)

Environment variable -> user variable -> path -> set the directory.

6. Introduction to SQLite

sqlite3 test.db

.header on

create table contacts (name text, phone integer, email text);

insert into contacts (name, phone, email) values ("Puneeth", 9066339217, "[puneeth1996p@gmail.com](mailto:puneeth1996p@gmail.com)");

SELECT \* FROM contacts;

SELECT email FROM contacts;

INSERT INTO contacts(name, phone) values (“steve”,8214);

7. More with SQL using SQLite

INSERT INTO contacts (name, phone, email) values ('aviril', '+91 (90) 9066339217', '[aviril@gmail.com](mailto:aviril@gmail.com)');

The phone number here not of proper type , even then it gets inserted into the DB

But when transferring data from one sql to another sql server the DB actually checks the type for each , this crashes the program

Also its important to maintain the type correctly as specified because when importing it and using in actual program , make sure that it doesnot crash.

Sql lite lacks at changing the data type of the column

.backup -> to backup the DB

UPDATE contacts Set email=’[newthings@update.com](mailto:newthings@update.com)’;

The above line of command is bad as it updates all the values of the table with email field , there the particular of field should be entered.

But we restore using .restore testback

Which holds the original db before the mistakes

To update one particluar field , we should use WHERE clause

UPDATE contacts SET email='puneet@newemail.com' WHERE name="Puneeth";

SELECT \* FROM contacts;

SELECT \* FROM contacts WHERE name="Puneeth";

SELECT phone, email FROM contacts WHERE name="Puneeth";

DELETE FROM contacts WHERE name="Puneeth";

SELECT \* FROM contacts;

.tables

.schema

.dump

8. Querying data with Sqlite

Auto increment

<https://www.sqlite.org/autoinc.html>

.schema or .dump to see the structure or values in the DB

SELECT \* FROM artists;

SELECT \* FROM artists WHERE \_id = 123;

Key is index for faster searched ,ordered based on that

There would be only one PRIMARY KEY( ITS NEVER NULL)

Relation db ordering row is undefined , its SET THEORY.

INSERT INTO artists VALUES(201,"Beyonce");

This throws an error as the primary key already exists

INSERT INTO artists (name) VALUES("Beyonce");

Please look into the page for Auto increment

<https://www.sqlite.org/autoinc.html>

Which helps understand the DB administration more clearly

9. Order by and Joins

Ordering is done default by primary key

SELECT \* FROM albums ORDER BY name;

SELECT \* FROM albums ORDER BY name COLLATE NOCASE ;

SELECT \* FROM albums ORDER BY name COLLATE NOCASE DESC ;

SELECT \* FROM albums ORDER BY artist,name COLLATE NOCASE DESC;

SELECT \* FROM songs ORDER BY album,track COLLATE NOCASE DESC;

SELECT \* FROM albums WHERE \_id=234;

Relate the tables together

Join clause

SELECT songs.track,songs.title,albums.name FROM songs JOIN albums ON songs.album = albums.\_id;

SELECT track, title, name FROM songs JOIN albums ON song.album = albums.\_id;

SELECT songs.track,songs.title,albums.name FROM songs INNER JOIN albums ON songs.album = albums.\_id;

SELECT songs.track,songs.title,albums.name FROM songs INNER JOIN albums ON songs.album = albums.\_id ORDER BY albums.name;

2 April 2019

10. More complex Joins

Challenge

SELECT artists.name, albums.name FROM albums INNER JOIN artists ON albums.artist = artists.\_id ORDER BY artists.name;

SELECT artists.name, albums.name, songs.track, songs.title FROM songs

INNER JOIN albums ON songs.album = albums.\_id

INNER JOIN artists ON albums.artist = artists.\_id

ORDER BY artists.name, albums.name, songs.track;

SELECT artists.name, albums.name, songs.track, songs.title FROM songs

INNER JOIN albums ON songs.album = albums.\_id

INNER JOIN artists ON albums.artist = artists.\_id

WHERE albums.\_id = 18

ORDER BY artists.name, albums.name, songs.track;

SELECT artists.name, albums.name, songs.track, songs.title FROM songs

INNER JOIN albums ON songs.album = albums.\_id

INNER JOIN artists ON albums.artist = artists.\_id

WHERE albums.name = “Doolittle”

ORDER BY artists.name, albums.name, songs.track;

SELECT artists.name, albums.name, songs.track, songs.title FROM songs

INNER JOIN albums ON songs.album = albums.\_id

INNER JOIN artists ON albums.artist = artists.\_id

WHERE albums.name = "Doolittle"

ORDER BY artists.name, albums.name, songs.track;

11. Wildcards and Views

SELECT artists.name, albums.name, songs.track, songs.title FROM songs

INNER JOIN albums ON songs.album = albums.\_id

INNER JOIN artists ON albums.artist = artists.\_id

WHERE albums.name LIKE "%Doo%"

ORDER BY artists.name, albums.name, songs.track;

LIKE is not case-sensitive

SELECT artists.name, albums.name, songs.track, songs.title FROM songs

INNER JOIN albums ON songs.album = albums.\_id

INNER JOIN artists ON albums.artist = artists.\_id

WHERE albums.name = "%Doolittle%"

ORDER BY artists.name, albums.name, songs.track;

CREATE VIEW artist\_list1 AS

SELECT artists.name, albums.name, songs.track, song.title FROM songs

INNER JOIN albums ON songs.album = albums.\_id

INNER JOIN albums ON albums.artist = artists.\_id

ORDER BY artists.name, albums.name, songs.track;

.schema

You can just the same operation as similar to table

SELECT \* FROM artist\_list;

SELECT \* FROM artist\_list WHERE name LIKE "%Doo%";

CREATE VIEW albums\_list AS

SELECT name FROM albums

ORDER BY name;

SELECT \* FROM albums\_list;

Command to delete the VIEW is

DROP VIEW albums\_list;

( you can also delete the table similarly)

12. Housekeeping and the Challenge

DELETE FROM songs WHERE track < 50;

SELECT \* FROM songs;

SELECT \* FROM artist\_list;

DELETE FROM songs WHERE track <> 50;

We can also use function in the select statements

SELECT COUNT(\*) FROM songs;

SELECT COUNT(\*) FROM albums;

SELECT COUNT(\*) FROM artists;

.restore music-backup2

SELECT COUNT(\*) FROM songs;

Challenge:

All the titles from album Forbidden

SELECT songs.title FROM songs INNER JOIN albums ON songs.album = albums.\_id WHERE albums.name = "Forbidden";

Sorted in track order.

SELECT songs.title FROM songs INNER JOIN albums ON songs.album = albums.\_id WHERE albums.name = "Forbidden" ORDER BY songs.track;

SELECT songs.title FROM songs

INNER JOIN albums ON songs.albums = artists.\_id

INNER JOIN artists ON albums.artists = artists.\_id

WHERE artists.name = "Deep Purple";

All the titles from album Forbidden

SELECT songs.title FROM songs INNER JOIN albums ON songs.album = albums.\_id WHERE albums.name = "Forbidden";

All songs band deep purple

SELECT songs.title FROM songs

INNER JOIN albums ON songs.albums = artists.\_id

INNER JOIN artists ON albums.artists = artists.\_id

WHERE artists.name = "Deep Purple";

SELECT \* FROM artist\_list WHERE artist = "Deep Purple";

Rename Mehatable to one kitten

UPDATE artists SET name = "One Kitten" WHERE artists.name = "Mehitabel";

SELECT \* FROM artists WHERE artists.name = "One Kitten";

Select all the songs by Aerosmith by alphabetical order with only title in the output

SELECT title FROM artist\_list WHERE artist = "Aerosmith" ORDER BY title;

SELECT count(title) FROM artist\_list WHERE artist = "Aerosmith";

SELECT DISTINCT title FROM artist\_list wHERE artist = "Aerosmith" ORDER BY title;

SELECT count (DISTINCT artist) FROM artist\_list WHERE artist = "Aerosmith";

SELECT count (DISTINCT title) FROM artist\_list WHERE artist = "Aerosmith";

13. SQL in Python

14. Connections, Cursors and Transactions

Committing the code after sql statement are important to push the changes to table.

15. SQL Injection Attacks

Be careful malicious user that use sql statements to inject sql commands that harm the database.

Please sanitize the input

16. Placeholders and Parameter Substitution

17. Exceptions

18. Exceptions Challenge

19. Exceptions Continued

Please note: It's important to in catching the error in the right order too.

Cmd + D , is to tested as exception is used to make the the string termination.

20. Raising Exceptions

21. More on Exceptions

22. Custom Exceptions

23. Rolling back Transactions

24. Adding Database code to the Account Class

05 April 2019

25. GUI Database Editing Overview

27. Community Edition Database Plugin

Configure -> Plugin ins -> Browse Repostitory -> search for

Database navigator -> restart

28. Update Deposit and Withdrawal Methods

29. Displaying Time in Different Timezones

30. SQLite3 strftime Function

31. Challenge

32. Problems Storing Timezones

33. Rolling Back Transactions

34. Simple Database Browser

35. Scrollbars

36. Star Args

37. Kwargs

38. More on KWArgs

39. Scrollable Listbox

6 April 2019

40. Populating a Listbox from a Database

41. Show Songs from Album

42. The DataListbox Class Code

43. Linking our DataListBoxes

44. Linking our DataListBoxes Continued

45. DataListbox Challenge