**IS 456 IT Database Systems Management**

**HOP01B Basic SQLite Queries**

4/13/2021 Developed by Farzin Bhadori

5/13/2021 Developed by Smita Dutta

School of Technology & Computing @ City University of Seattle (CityU)



**Before You Start**

* The directory path shown in screenshots may be different from yours.
* Some steps are not explained in the tutorial**.** If you are not sure what to do:
  1. Consult the resources listed below.
  2. If you cannot solve the problem after a few tries, ask a TA for help.

**Learning Outcomes**

Students will be able to:

Students will be able to:

* Understand the SQLite queries.
* Run queries in SQLite.
* Create tables.

-- 01 CREATE TABLE

-- test.db

CREATE TABLE test (

a INTEGER,

b TEXT

);

INSERT INTO test VALUES ( 1, 'a' );

INSERT INTO test VALUES ( 2, 'b' );

INSERT INTO test VALUES ( 3, 'c' );

SELECT \* FROM test;

-- 02 DROP TABLE

-- test.db

CREATE TABLE test ( a TEXT, b TEXT );

INSERT INTO test VALUES ( 'one', 'two' );

SELECT \* FROM test;

DROP TABLE test;

DROP TABLE IF EXISTS test;

-- 03 INSERT INTO

-- test.db

CREATE TABLE test ( a INTEGER, b TEXT, c TEXT );

INSERT INTO test VALUES ( 1, 'This', 'Right here!' );

INSERT INTO test ( b, c ) VALUES ( 'That', 'Over there!' );

INSERT INTO test DEFAULT VALUES;

INSERT INTO test ( a, b, c ) SELECT id, name, description from item;

SELECT \* FROM test;

-- 04 DELETE FROM

-- test.db

SELECT \* FROM test;

DELETE FROM test WHERE a = 3;

SELECT \* FROM test WHERE a = 1;

DELETE FROM test WHERE a = 1;

-- 05 NULL

-- test.db

SELECT \* FROM test;

SELECT \* FROM test WHERE a = NULL;

SELECT \* FROM test WHERE a IS NULL;

SELECT \* FROM test WHERE a IS NOT NULL;

INSERT INTO test ( a, b, c ) VALUES ( 0, NULL, '' );

SELECT \* FROM test WHERE b IS NULL;

SELECT \* FROM test WHERE b = '';

SELECT \* FROM test WHERE c = '';

SELECT \* FROM test WHERE c IS NULL;

DROP TABLE IF EXISTS test;

CREATE TABLE test (

a INTEGER NOT NULL,

b TEXT NOT NULL,

c TEXT

);

INSERT INTO test VALUES ( 1, 'this', 'that' );

SELECT \* FROM test;

INSERT INTO test ( b, c ) VALUES ( 'one', 'two' );

INSERT INTO test ( a, c ) VALUES ( 1, 'two' );

INSERT INTO test ( a, b ) VALUES ( 1, 'two' );

DROP TABLE IF EXISTS test;

-- 06 Constraints

-- test.db

DROP TABLE IF EXISTS test;

CREATE TABLE test ( a TEXT, b TEXT, c TEXT );

INSERT INTO test ( a, b ) VALUES ( 'one', 'two' );

SELECT \* FROM test;

CREATE TABLE test ( a TEXT, b TEXT, c TEXT NOT NULL );

CREATE TABLE test ( a TEXT, b TEXT, c TEXT DEFAULT 'panda' );

CREATE TABLE test ( a TEXT UNIQUE, b TEXT, c TEXT DEFAULT 'panda' );

CREATE TABLE test ( a TEXT UNIQUE NOT NULL, b TEXT, c TEXT DEFAULT 'panda' );

-- 07 ALTER TABLE

-- test.db

DROP TABLE IF EXISTS test;

CREATE TABLE test ( a TEXT, b TEXT, c TEXT );

INSERT INTO test VALUES ( 'one', 'two', 'three');

INSERT INTO test VALUES ( 'two', 'three', 'four');

INSERT INTO test VALUES ( 'three', 'four', 'five');

SELECT \* FROM test;

ALTER TABLE test ADD d TEXT;

ALTER TABLE test ADD e TEXT DEFAULT 'panda';

DROP TABLE IF EXISTS test;

-- 08 ID

-- test.db

CREATE TABLE test (

id INTEGER PRIMARY KEY,

a INTEGER,

b TEXT

);

INSERT INTO test (a, b) VALUES ( 10, 'a' );

INSERT INTO test (a, b) VALUES ( 11, 'b' );

INSERT INTO test (a, b) VALUES ( 12, 'c' );

SELECT \* FROM test;

DROP TABLE IF EXISTS test;

-- 09 WHERE, LIKE, and IN

-- world.db

SELECT \* FROM Country;

SELECT Name, Continent, Population FROM Country

WHERE Population < 100000 ORDER BY Population DESC;

SELECT Name, Continent, Population FROM Country

WHERE Population < 100000 OR Population IS NULL ORDER BY Population DESC;

SELECT Name, Continent, Population FROM Country

WHERE Population < 100000 AND Continent = 'Oceania' ORDER BY Population DESC;

SELECT Name, Continent, Population FROM Country

WHERE Name LIKE '%island%' ORDER BY Name;

SELECT Name, Continent, Population FROM Country

WHERE Continent IN ( 'Europe', 'Asia' ) ORDER BY Name;

-- 10 SELECT DISTINCT

-- world.db

SELECT Continent FROM Country;

SELECT DISTINCT Continent FROM Country;

Using the SELECT DISTINCT statement, you will get only unique results.

-- test.db

DROP TABLE IF EXISTS test;

CREATE TABLE test ( a int, b int );

INSERT INTO test VALUES ( 1, 1 );

INSERT INTO test VALUES ( 2, 1 );

INSERT INTO test VALUES ( 3, 1 );

INSERT INTO test VALUES ( 4, 1 );

INSERT INTO test VALUES ( 5, 1 );

INSERT INTO test VALUES ( 1, 2 );

INSERT INTO test VALUES ( 1, 2 );

INSERT INTO test VALUES ( 1, 2 );

INSERT INTO test VALUES ( 1, 2 );

INSERT INTO test VALUES ( 1, 2 );

SELECT \* FROM test;

SELECT DISTINCT a FROM test;

SELECT DISTINCT b FROM test;

SELECT DISTINCT a, b FROM test;

DROP TABLE IF EXISTS test;

-- 11 ORDER BY

-- world.db

SELECT Name FROM Country;

SELECT Name FROM Country ORDER BY Name;

SELECT Name FROM Country ORDER BY Name DESC;

SELECT Name FROM Country ORDER BY Name ASC;

SELECT Name, Continent FROM Country ORDER BY Continent, Name;

SELECT Name, Continent, Region FROM Country ORDER BY Continent DESC, Region, Name;

-- 12 CASE

-- test.db

DROP TABLE IF EXISTS booltest;

CREATE TABLE booltest (a INTEGER, b INTEGER);

INSERT INTO booltest VALUES (1, 0);

SELECT \* FROM booltest;

SELECT

CASE WHEN a THEN 'true' ELSE 'false' END as boolA,

CASE WHEN b THEN 'true' ELSE 'false' END as boolB

FROM booltest

;

SELECT

CASE a WHEN 1 THEN 'true' ELSE 'false' END AS boolA,

CASE b WHEN 1 THEN 'true' ELSE 'false' END AS boolB

FROM booltest

;

DROP TABLE IF EXISTS booltest;

# Screenshots

Provide at least 3 screenshots as part of HOP submission.



# Summary

Write a 150-word summary to explain your understandings and findings from this lab assignment.

I found that much like the previous queries The number of screenshots is laborious and asking me to choose three out of possibly 100 is just not very focused instructions. Also, this is all review for me as I just took a SQL class. Its also confusing because TSQL has stricter rules and different naming conventions than SQLite.

I am shocked that those case statements worked though. Personally, I like the complicated queries when theres a ton of data on the table. This way we can see all the different ways to configure large tables for more complicated queries. The main thing that can be learned from this is how the data gets inserted into the tables in this type of database.