Introduction to SQLite

CMPUT 291

Introduction to File and Database Management

UNIVERSITY OF ALBERTA
DEPARTMENT OF COMPUTING SCIENCE

Goals

- Work with SQLite
- Work with SQL statements
  - DDL and DML statements
- Execute SQL statements from file
- Help facilities

SQLite

- Embedded relational database engine
- Cross platform database file
- Supports Structured Query Language (most of the SQL-92 standard).

http://sqlite.org/

Running SQLite

- Foreign key constraints in SQLite should be enabled for each database connection
- To consistently turn it on for connections, run the following command in terminal
  - Unix

```
echo "PRAGMA foreign_keys=ON;" >>~/.sqliterc
```

  - Windows

```
echo PRAGMA foreign_keys=ON; >>%HOME%/.sqliterc
```

HOME environment variable must be defined and point to the current user's home directory path.

Running SQLite (cont'd)

- To run SQLite, issue the command **`sqlite3`**

- Then, open database from file

```
sqlite> .open movie.db
```

Working with SQL statements

- **D**ata **D**efinition **L**anguage (DDL)
  - Build and modify the structure of your tables and other objects in the database
  - CREATE, ALTER, DROP

- **D**ata **M**anipulation **L**anguage (DML)
  - Work with the data
  - SELECT, INSERT, UPDATE, DELETE

DDL: CREATE TABLE syntax

CREATE TABLE *TableName* (

attribute1 type1,

attribute2 type2,

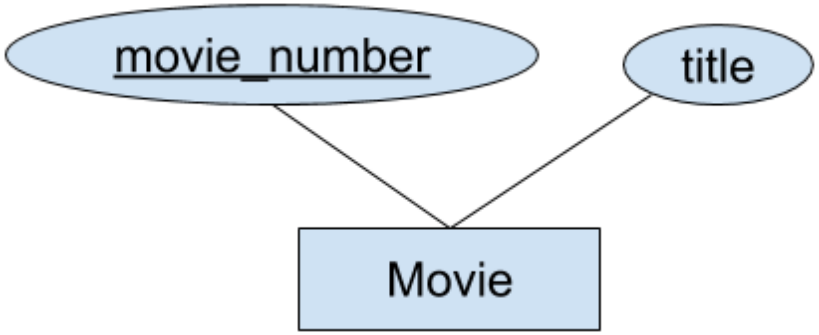
…,

PRIMARY KEY (*attributes*),

FOREIGN KEY (*attributes*) **REFERENCES** *TableName2* (*attributes*)

);

CREATE TABLE Example




```
sqlite> CREATE TABLE movie (
   ...> title TEXT,
   ...> movie_number INTEGER,
   ...> PRIMARY KEY (movie_number));
```

Don't miss the semicolon at the end of the command!

- You can verify that a table is created with `.tables` command which lists all the tables in the database.
- To see the structure of the table, use the following command:

```
sqlite> .schema movie
```

Commands beginning with a dot (a.k.a. “dot-commands”) are considered as shell commands and no semicolon should be used like SQL queries.

- You can also use the above command, to recall the name or type of the attributes of the table.

DDL: ALTER TABLE syntax

- SQLite supports a limited subset of ALTER TABLETo add an attribute to a defined table:

ALTER TABLE *TableName* **ADD** *attributeM typeM*;

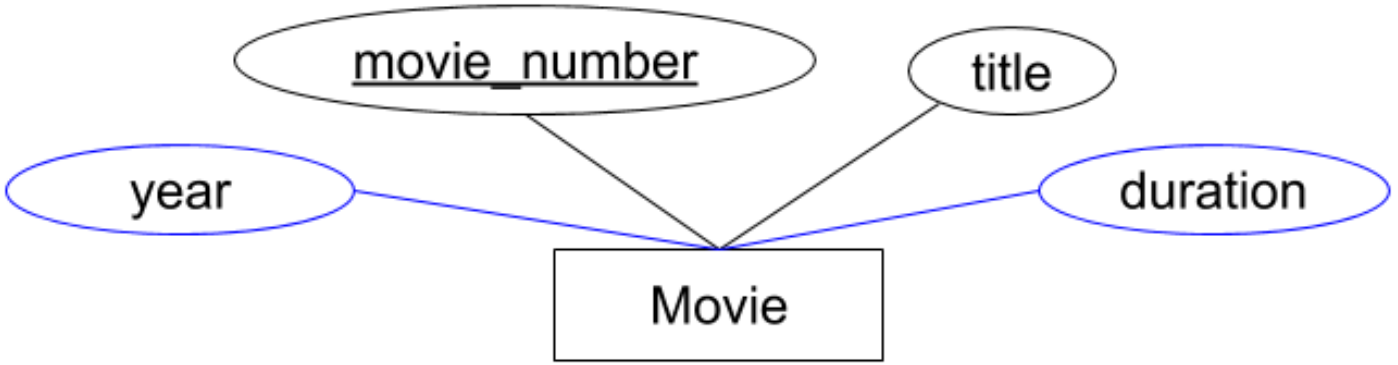
- To rename table:

ALTER TABLE *TableName* **RENAME TO** *NewName*;

- SQLite doesn't support DROP column(s), RENAME column TO new_column, ADD/DROP CONSTRAINT
- We'll see how to tackle this problem

Add Attributes Example

- To add attributes duration and year to movie table:




```
sqlite> ALTER TABLE movie ADD year INTEGER;
sqlite> ALTER TABLE movie ADD duration INTEGER;
```

DML: INSERT records syntax

INSERT INTO *TableName* **VALUES**

(*v1*, …, *vN*), (*w1*, …, w*N*), …, (*z1*, …, *zN*);

The comma-separated list of values must match the table structure exactly in the number of attributes and the data types of each attribute.

Insert Records Example

- Insert 3 rows to movie table:

title	movie_number	year	duration
The Matrix	1	2000	120
La La Land	2	2016	128
Citizen Kane	3	1941	119

```
sqlite> INSERT INTO movie VALUES
   ...> ('The Matrix', 1, 2000, 120),
   ...> ('La La Land', 2, 2016, 128),
   ...> ('Citizen Kane', 3, 1941, 119);
```

ALTER TABLE workaround

- How to perform ALTER TABLEs which are not supported by SQLite
  - Rename the table to be altered to a temporary table
  - Re-create schema with the desired columns or constraints
  - Fill new table with data from temporary table
  - Drop temporary table

Rename Column Example

- Rename ‘duration’ column to ‘runtime’:

```
sqlite> ALTER TABLE movie RENAME TO temp;
sqlite> CREATE TABLE movie (title TEXT,
   ...> movie_number INTEGER,
   ...> runtime INTEGER, year INTEGER,
   ...> PRIMARY KEY (movie_number));
sqlite> INSERT INTO movie SELECT
   ...> title,movie_number,duration,year
   ...> FROM temp;
sqlite> DROP TABLE temp;
```

Rename Column Example

- Rename ‘duration’ column to ‘runtime’:

```
sqlite> ALTER TABLE movie RENAME TO temp;
sqlite> CREATE TABLE movie (title TEXT,
   ...> movie_number INTEGER,
   ...> runtime INTEGER, year INTEGER
```

INSERT INTO *TableName1* **SELECT** *attributes* **FROM** *TableName2*;

```
sqlite> INSERT INTO movie SELECT
   ...> title,movie_number,duration,year
   ...> FROM temp;
```

Another form of insert statement: Records are inserted into the table for each row returned by select.

Type Affinity

- Data Types in SQLite are regarded as the recommended type for data (not required)
- Any column can store any type of data
- Each column is assigned one of TEXT, NUMERIC, INTEGER, REAL, BLOB as preferred type
- The affinity of a column is determined by the declared type

Type Affinity Example

- Try inserting the following record:
  - (2001, 4, ‘1968’, ‘2h 29min’)
- No problem! Added successfully:

```
SELECT movie_number,TYPEOF(title),TYPEOF(year),
TYPEOF(runtime) FROM movie WHERE movie_number >= 3;
```

3|text|integer|integer
4|text|integer|text

SQLite couldn’t convert ‘2h 29min’ to integer, so it is saved in TEXT type

Converted to the specified column types in movie table

DML: UPDATE syntax

UPDATE *TableName*

SET *attributeI=NewValueI, …, attributeK=NewValueK*

[**WHERE** *condition*];

Update Records Example

- Update year and runtime attributes of The Matrix movie:

```
sqlite> UPDATE movie SET
   ...> year=1999, runtime=136
   ...> WHERE title='The Matrix';
```

DML: SELECT syntax

SELECT * **FROM** *TableName* [**WHERE** *condition*];

- Shows records from *TableName* that satisfy the condition. If no condition is stated, shows all records

SELECT *attributes* **FROM** *TableName* [**WHERE** *condition*];

- Shows columns specified in *attributes* of *TableName*, for records satisfying the condition. And all records are displayed, in case no condition is stated

Select Example

- Retrieve all records of movie table:

```
sqlite> SELECT * FROM movie;
```

- Retrieve only title and runtime of movies made after 2010:

```
sqlite> SELECT title, runtime FROM movie
   ...> WHERE year > 2010;
```

DML: DELETE syntax

DELETE FROM *TableName* [**WHERE** *condition*];

- Records from *TableName* that satisfy a condition will be deleted. If no condition is stated, deletes all records

Make sure when you are going to delete records, SQLite doesn't give you '*Are you sure?*' prompt!

Delete Records Example

- Delete movies with number equal to 1

```
sqlite> DELETE FROM movie
   ...> WHERE movie_number=1;
```

- You can check the result using select:

```
sqlite> SELECT * FROM movie
   ...> WHERE movie_number=1;
```

DDL: DROP TABLE syntax

- To remove a defined table:

  DROP TABLE *TableName*;

Be careful!! If you drop a table, you will lose its data!

Drop Table Example

- Remove table movie:

```
sqlite> DROP TABLE movie;
```

- You can check the result using `.schema`:

```
sqlite> .schema movie
```

SQLite Catalogue

- **sqlite_master** table defines the schema for the database
- The table is managed by SQLite
- You can query this table to get information about tables stored in the system

type	TEXT	table, index, view, trigger
name	TEXT	name of the object
tbl_name	TEXT	name of the table or view that the object is related to
sql	TEXT	SQL text that describes the object

sqlite_master Example

- Get a list of tables you defined:

```
sqlite> SELECT name FROM sqlite_master
   ...> WHERE type='table';
```

- You can see the structure of these tables:

```
sqlite> .schema sqlite_master
```

Execute SQL statements from file

- You can store SQL commands in a file and executes them in SQLite
  - Create a file in an editor and store some SQL commands in the file.
  - For example: (my_batch.sql)

```
SELECT * FROM movie WHERE year > 2010;
INSERT INTO movie VALUES ('Logan', 7, 2017, 135);
SELECT * FROM movie WHERE year > 2010;
```

- To run the stored commands:

```
sqlite> .read my_batch.sql
```

`.help` command

- To view the list of available “dot-commands”, you can use **`.help`**:

```
.backup ?DB? FILE       Backup DB (default "main") to FILE
.bail ON|OFF            Stop after hitting an error. Default OFF
.databases              List names and files of attached databases
…
```

Exit SQLite

- To leave SQLite program

```
sqlite> .exit
```

OR

```
sqlite> .quit
```