LECTURE 0x20

SQL AND SQLITE II

MCS 275 Spring 2021

Emily Dumas

SELECT * FROM bulletins WHERE lecture=32;

- Quiz 11 coming on Sunday (due Tuesday), will involve matplotlib. Use of VanderPlas will be allowed.
- I decided not to remove the unused Lecture 31 slides. Some of those will be used again today.

USING SQLITE

Method 1: From a Python script

```
import sqlite3
con = sqlite3.connect("mydbfile")
res = con.execute("SELECT * FROM evil_plans WHERE year=2021;")
print(res.fetchall())
con.close()
```

Method 2: Run sqlite command line shell and type

```
.open "mydbfile"
SELECT * FROM evil_plans WHERE year=2021;
```

SAMPLE DATABASE

Most of my work today will use a sample database containing information on ≈100,000 stars:

hyg_data.zip

SELECT

Find and return rows. The most common query.

```
SELECT * FROM table_name; -- give me everything
SELECT * FROM table_name WHERE condition; -- some rows
SELECT col3, col1 FROM table_name; -- some columns
SELECT * FROM table_name LIMIT 10; -- at most 10 rows

SELECT * FROM table_name
ORDER BY col2; -- sort by col2, smallest first

SELECT * FROM table_name
ORDER BY col2 DESC; -- sort by col2, biggest first
```

Conditions can be e.g. equalities and inequalities.

WHERE, ORDER BY, LIMIT can be used together, but must appear in that "WOBL" order. (Details.)

GETTING DATA FROM SQLITE

After `SELECT`, where are the data?

`execute()` doesn't return the rows directly. It returns a **Cursor** object which is ready to give them to you.

To request rows from a Cursor c, several options:

- Use it as an iterable (it yields one tuple per row).
- `c.fetchone()` returns next row as a tuple.
- `c.fetchall()` returns a list of tuples.

SQL CONDITIONS

Examples of things that can appear after WHERE:

```
col = value  -- Also supports >, >=, <, <=, !=
col IN (val1, val2, val3)
col BETWEEN lowval AND highval
col IS NULL
col IS NOT NULL
stringcol LIKE pattern  -- string pattern matching
condition1 AND condition2
condition1 OR condition2
```

LIKE

```
coursetitle LIKE "Introduction to %"
itemtype LIKE "electrical adapt_r"
```

In a pattern string:

- `%` matches any number of characters (including 0)
- `_` matches any single character

e.g. `"%d_g"` matches `"fossil dig"` and `"dog"` but does not match `"hypersonic drag"`,`"dog toy"`, or `"dg"`.

CREATE TABLE

Creates a table. The set of tables doesn't change very often in most databases, and this setup step is often performed manually or by a separate program.

```
CREATE TABLE [IF NOT EXISTS] table_name (
    col1 TYPE1 [MODIFIERS],
    col2 TYPE2 [MODIFIERS], ...
);  -- or you could write it all on one line!
```

Types include: `TEXT`, `REAL`, `INTEGER`

Modifiers include: `UNIQUE`, `NOT NULL`, `PRIMARY KEY`

Creating a table twice generates an error unless `IF NOT EXISTS` is given.

INSERT INTO ... VALUES

Add one row to an existing table.

```
-- Set every column (need to know column order!)
INSERT INTO table_name
VALUES ( val1, val2, val3, val4, val5, val6, val7 );

-- Set some columns, in an order I specify
INSERT INTO table_name ( col1, col7, col3 )
VALUES ( val1, val7, val3 );
```

Missing columns are set to default values (often null).

Exceptions indicate constraint violations (e.g. typing).

There is also a way to insert many rows at once, taken from the result of another query.

GIVING DATA TO SQLITE

Don't use string formatting to embed data in a call to `execute()`. Instead, use `?` characters as placeholders and then give a tuple of values in the second argument.

```
# works, but bad practice; code and data mixed
# and everything forced into a string
con.execute("INSERT INTO planets VALUES (\"Earth\", 1.0, null);")

# do this instead; it keeps data in native types
# separate from the SQL code
con.execute(
    "INSERT INTO planets VALUES (?,?,?);",
    ("Earth", 1.0, None)
)
```

UPDATE

Change values in a row (or rows).

```
UPDATE table_name SET col1=val1, col5=val5 WHERE condition;
```

Warning: Every row meeting the condition is changed!

Also supports ORDER BY and LIMIT.

DELETE

Remove rows matching a condition.

```
DELETE FROM table_name WHERE condition;
```

Also supports ORDER BY and LIMIT (e.g. to remove *n* rows with largest values in a given column).

Immediate, irreversible.

Omit `WHERE` clause to delete all rows.

DROP TABLE

Deletes an entire table.

```
DROP TABLE table_name;              -- no such table = ERROR
DROP TABLE IF EXISTS table_name; -- no such table = ok
```

Immediate, irreversible. Think of it as "throw the only copy of this table into a pool of lava". Use caution.

TRANSACTION CONTEXT MANAGER

You can use a sqlite3 Connection object as a context manager (i.e. in `with`) to create a **transaction**.

```
with con:
    # Make all the changes necessary to reflect the closing
    # of the Scranton office.
    con.execute("UPDATE...")
    con.execute("UPDATE...")
```

Another connection to the same database will never see it in a state other than "everything in the transaction happened" (if no exceptions) or "nothing in the transaction happened" (if an exception occurs).

REFERENCES

- SQLite home page
- sqlitetutorial.net has a nice tutorial where you can run SQL command directly in your browser. Their SQLite install instructions are detailed and easy to follow, too.
- *Intro to Python for Computer Science and Data Science* by Deitel and Deitel, Section 17.2. (This is an O'Reilly book, free for anyone with a UIC email; see course page for login details.)
- *Computer Science: An Overview* by Brookshear and Brylow, Chapter 9.

REVISION HISTORY

- 2021-04-24 Revise DELETE slide to mention how to delete all rows
- 2021-04-02 Initial publication