Cherry blossoms of the Japanese Yoshino variety bloom along the Tidal Basin, March 19, 2012, in Washington, DC, with the Jefferson Memoria to the rear. This season celebrates the 100-year anniversary of the gift of the cherry trees from Japan to Washington, DC. (Paul J. Richards/AFF Getty Images)

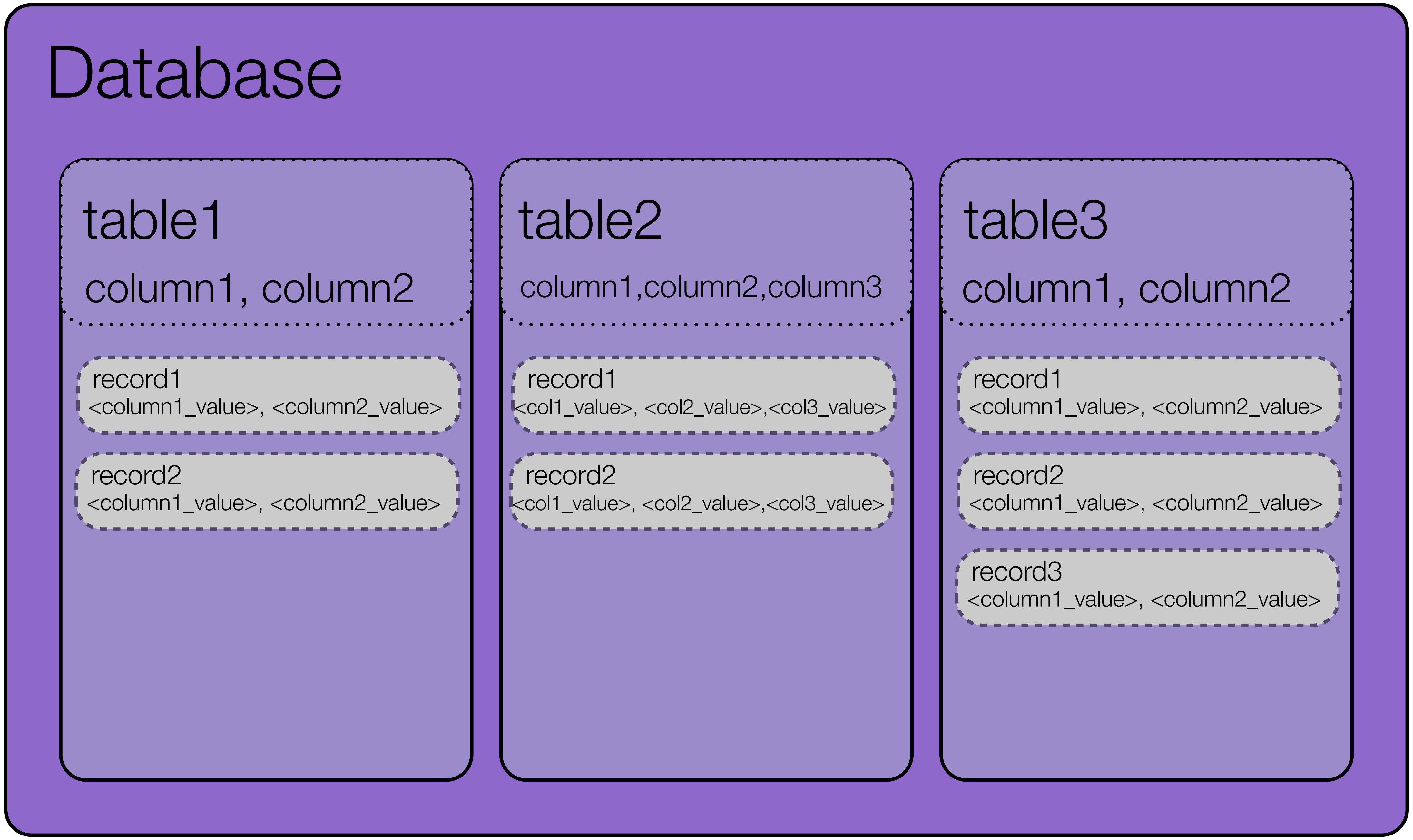
COMP 364 - Lecture 25
March 26th, 2012
Mathieu Perreault

Python and SQLite

What we know so far

- **Database** is the main structure (only one database at a time)
- A database contains many **tables**, each with a **column structure**.
  e.g. gene_id, gene_name, gene_length for table *genes*
- A table can contain multiple **records**, each represented by a **row**.
- We can access the different **columns** of a record to get/set the data.

Database structure


Database
table1
column1, column2
record1
<column1_value>, <column2_value>
record2
<column1_value>, <column2_value>
table2
column1,column2,column3
record1
<col1_value>, <col2_value>,<col3_value>
record2
<col1_value>, <col2_value>,<col3_value>
table3
column1, column2
record1
<column1_value>, <column2_value>
record2
<column1_value>, <column2_value>
record3
<column1_value>, <column2_value>

Getting the content from a database

- Opening the database
- View the tables in a database
- View the fields (columns) in a table
- View all the records (rows) in a table
- View *some* of the records according to some filtering rule
- Full SQLite Syntax reference: http://www.sqlite.org/lang.html

Creating new content in a database

- Opening the database
- Create a table in the database
- Insert records in the database
- Update records in the database
- Delete records according to a specific condition
- Delete tables
- Full SQLite Syntax reference: http://www.sqlite.org/lang.html

A word about PRIMARY KEY

- A primary key in a table ensures that the specified column will have a distinct value for each record
- Only one primary key is allowed **per table**.
- It could be an INTEGER, TEXT, etc.
- When specifying a column definition (such as when creating a table), just add PRIMARY KEY:

```
CREATE TABLE grades (student_id INTEGER PRIMARY KEY, grade REAL);
```

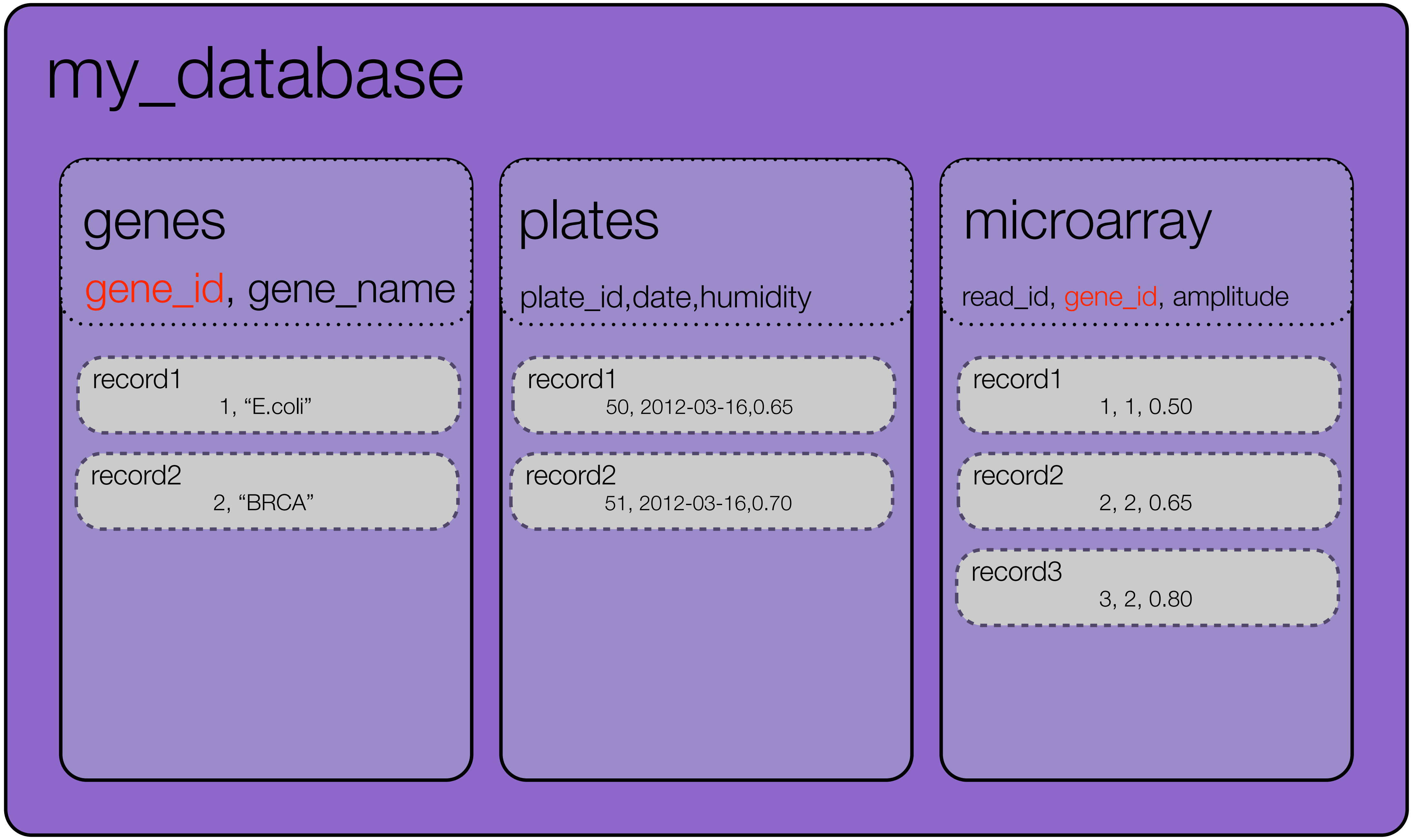
- `PRIMARY KEY AUTOINCREMENT` will increase the key for you, no need to specifically insert it.

Default value

- You can also specify default values if you have special needs
- You simply need to add it to the column definition (for example, when creating a table)
- For example, specifying a default value of 100.0

```
CREATE TABLE grades (student_id INTEGER PRIMARY KEY, grade REAL DEFAULT
100.0);
```

Database structure



Joined at the hip (or: JOIN clauses)

- Sometimes you will have tables that complement other tables
- It’s one advantage of SQLite, no need to store redudant information
    - No need to store gene_name every time you make a measurement on the gene!
- To **query** for this information, we will need to tell SQLite that two fields from different tables are really talking about the same thing!
    - Not sufficient to name them the same (e.g. gene_id)

Joined at the hip (or: JOIN clauses)

- The JOIN clause can be added to the source:
  - `SELECT [columns] FROM table1 JOIN table2 ON table1.field_foo=table2.field_bar WHERE [condition];`
  - This says: "I'm selecting records in table 1, but bring in extra information about those records from table 2"
- The extra information can be used in the WHERE clause:

```
SELECT * FROM genes JOIN microarray ON genes.gene_id=microarray.gene_id WHERE
microarray.amplitude>0.6;
```

- Can augment the results:

```
SELECT microarray.amplitude, genes.gene_name FROM genes JOIN microarray ON
genes.gene_id=microarray.gene_id;
```

Today: SQLite in Python

- SQLite is included in Python (how convenient!)
- You can write code that interacts with SQLite.
- The power of Python, combined with SQLite, makes for a great program.

SQLite in Python

- Four things need to happen for Python to interact with SQLite
- Import the sqlite module

```
import sqlite3
```

- Connect to the database (or create one) with a given name

```
connection = sqlite3.connect('database.sql')
```

- Get a cursor to the database:

```
cursor = connection.cursor()
```

- Execute queries on the cursor:

```
cursor.execute('SELECT * FROM ...')
```

SQLite in Python

- Once a query has been executed on the cursor, some data might be available, e.g. if you made a SELECT query.
- `cursor.fetchone()` will return one record at a time.
- `cursor.fetchall()` will return all matching records at once in a Python list.
- You have to test if the result is `None` before using it.
- **Don't forget to call `connection.commit()` to commit the changes!**