Lecture 7
Python and SQL

Lecturer: Pieter De Bleser

Bioinformatics Core Facility, IRC

GHENT UNIVERSITY

VIB-UGENT CENTER FOR INFLAMMATION RESEARCH

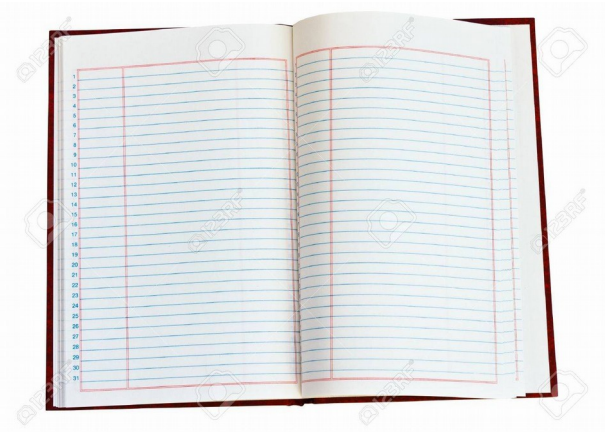
Slides derived from: Andrew C.R. Martin (andrew.martin@ucl.ac.uk); Charles R. Severance (www.dr-chuck.com);www.pynative.com

Analog Databases – example
A Congress database

- Address book and a ledger

- Address book:
  - Speaker name (quick access via lettered tabs)
  - Speaker email address
  - Speaker phone number

- Ledger:
  - Session name
  - Speaker name
  - Session description
  - Session start
  - Session duration



Adding Data

- Address Book:
  - Find the right letter tab
  - Write in the information

- Ledger:
  - Make sure you added speaker to address book first
  - Find an open row
  - Add speaker, session and duration information

Finding Data

- Address Book:
  - Flip to correct letter tab, look down list to find right contact
- Ledger:
  - Scroll down ledger sheet to find correct session name

Cross-Referencing

- Address book
  - Look up a name
  - Look for the name in the ledger to see what sessions they deliver at the conference

- Ledger
  - Look up a session in the ledger
  - Note the speaker’s name
  - Look up the speaker in the address book

Updating/Deleting Data

- Find the data
- Get out an eraser/whiteout
- Fill in new data
- Update the ledger to reflect name changes in the address book if needed

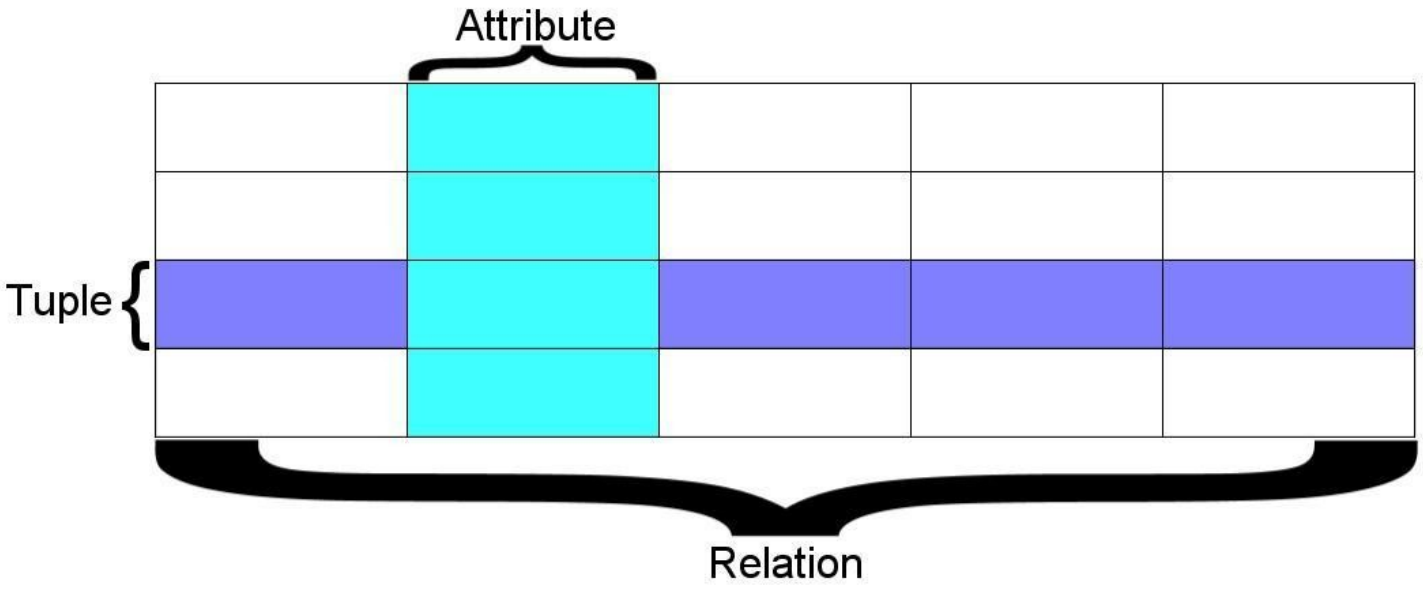
Relational Databases

Relational databases model data by storing rows and columns in tables. The power of the relational database lies in its ability to efficiently retrieve data from those tables and in particular where there are multiple tables and the relationships between those tables involved in the query.

http://en.wikipedia.org/wiki/Relational_database

Terminology

- Database - contains many tables
- Relation (or table) - contains tuples and attributes
- Tuple (or row) - a set of fields that generally represents an “object” like a person or a music track
- Attribute (also column or field) - one of possibly many elements of data corresponding to the object represented by the row



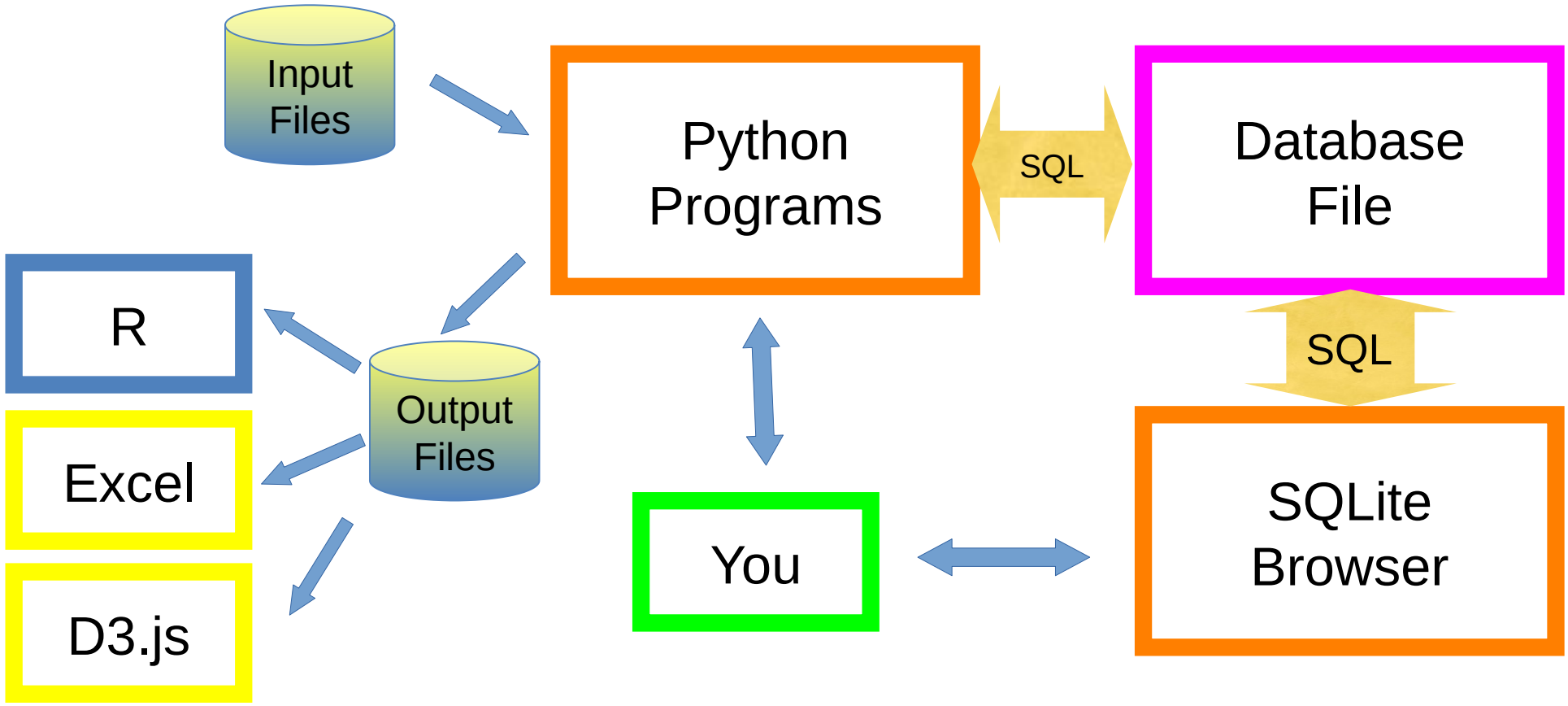
A relation is defined as a set of tuples that have the same attributes. A tuple usually represents an object and information about that object. Objects are typically physical objects or concepts. A relation is usually described as a table, which is organized into rows and columns. All the data referenced by an attribute are in the same domain and conform to the same constraints. (Wikipedia)

SQL

Structured Query Language is the language we use to issue commands to the database

- **C**reate data (a.k.a Insert)
- **R**etrieve data
- **U**pdate data
- **D**elete data

http://en.wikipedia.org/wiki/SQL

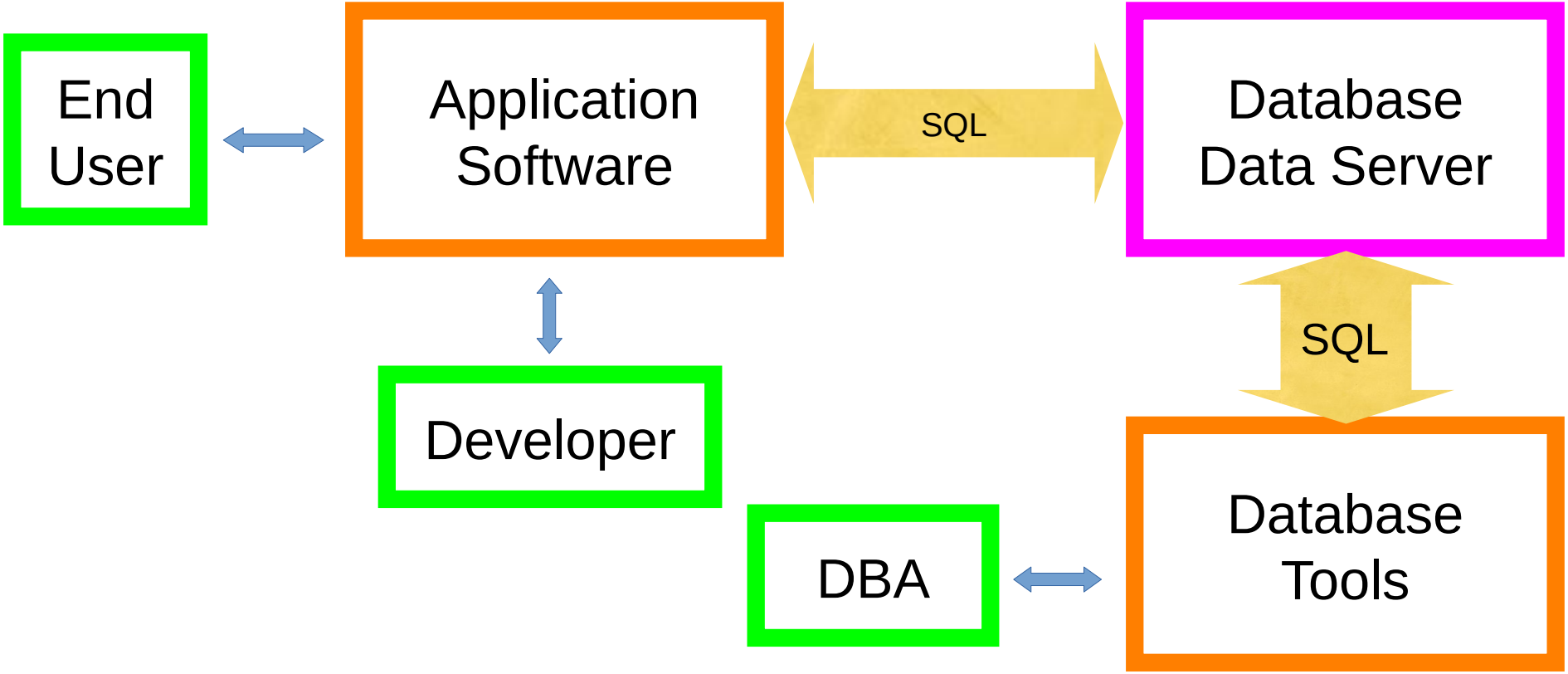

Input Files
Python Programs
SQL
Database File
SQL
R
Output Files
Excel
D3.js
You
SQLite Browser

Web Applications w/ Databases

Application Developer - Builds the logic for the application, the look and feel of the application - monitors the application for problems

Database Administrator - Monitors and adjusts the database as the program runs in production

Often both people participate in the building of the “Data model”


End User
Application Software
SQL
Database Data Server
SQL
Developer
DBA
Database Tools

Database Administrator

A database administrator (DBA) is a person responsible for the design, implementation, maintenance, and repair of an organization’s database. The role includes the development and design of database strategies, monitoring and improving database performance and capacity, and planning for future expansion requirements. They may also plan, coordinate, and implement security measures to safeguard the database.

http://en.wikipedia.org/wiki/Database_administrator

Database Model

A database model or database schema is the structure or format of a database, described in a formal language supported by the database management system. In other words, a “database model” is the application of a data model when used in conjunction with a database management system.

http://en.wikipedia.org/wiki/Database_model

Common Database Systems

Three major Database Management Systems in wide use

- Oracle - Large, commercial, enterprise-scale, very very tweakable
- MySql - Simpler but very fast and scalable - commercial open source
- SqlServer - Very nice - from Microsoft (also Access)

Many other smaller projects, free and open source

HSQL, SQLite, Postgres, ...

SQLite is in Lots of Software...

symbian

python™

PHILIPS

skype™

GE

Microsoft®

McAfee®

Adobe®

php

Google™

TOSHIBA

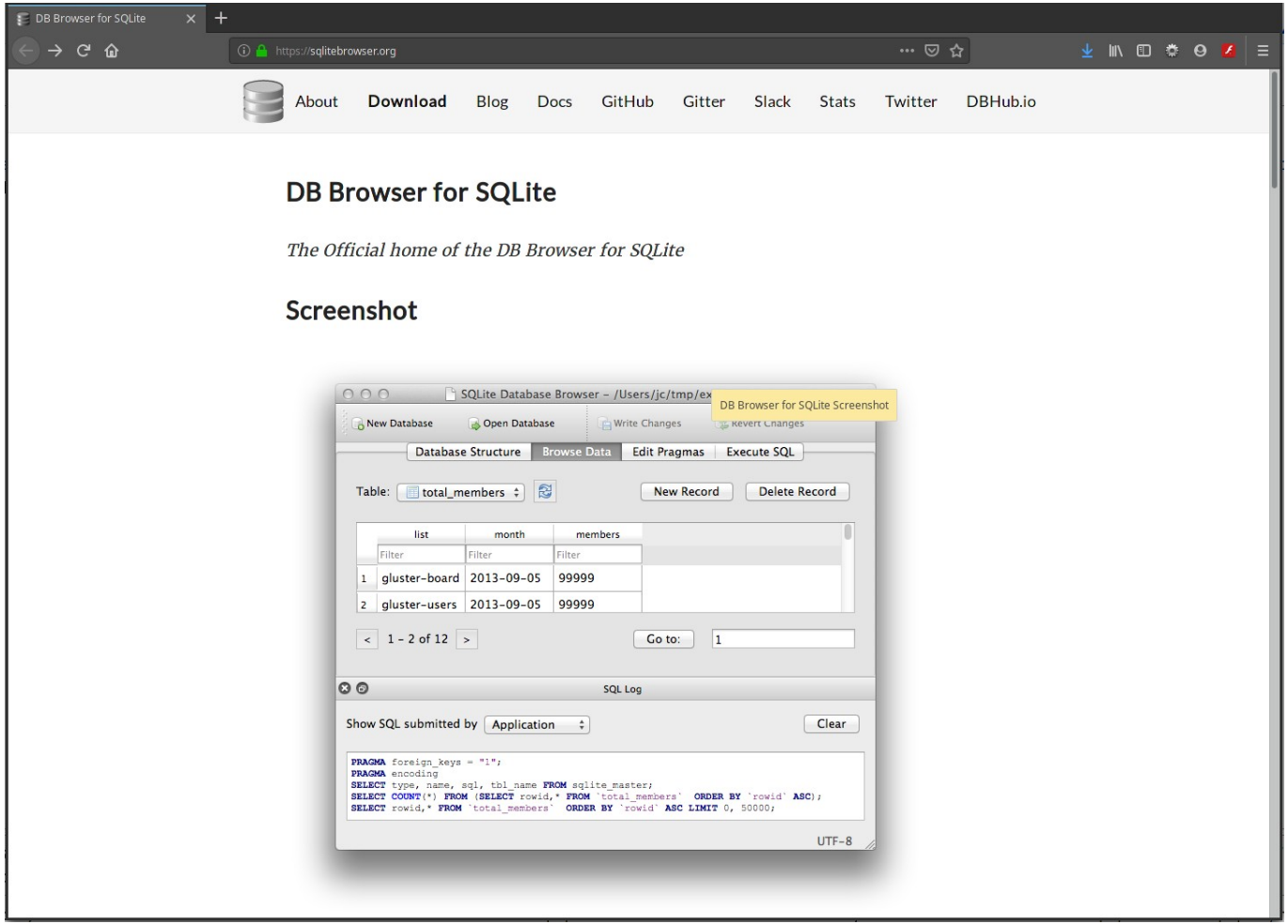
Sun microsystems

http://www.sqlite.org/famous.html

SQLite Browser

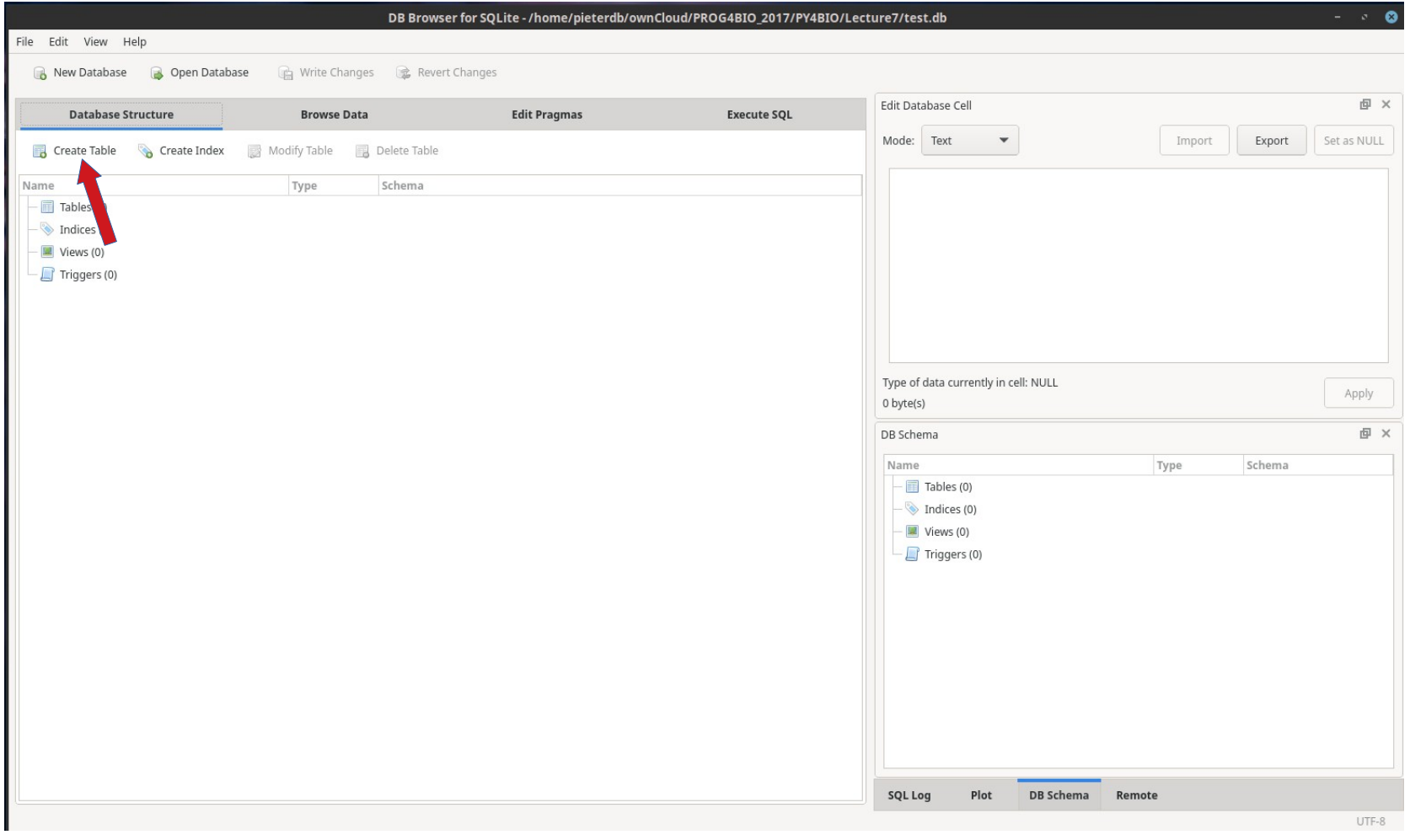
- SQLite is a very popular database - it is free and fast and small
- SQLite Browser allows us to directly manipulate SQLite files
  - http://sqlitebrowser.org/
- SQLite is embedded in Python and a number of other languages

https://sqlitebrowser.org/



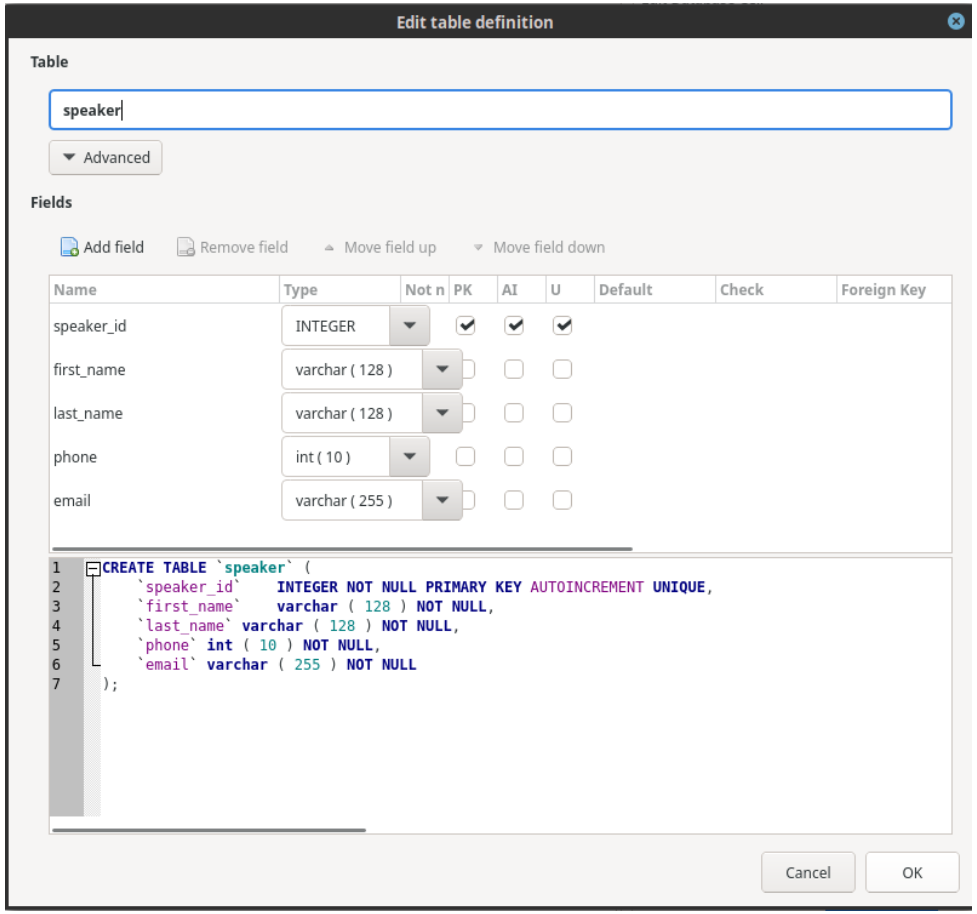
Lets Make a Database!

CREATE a table

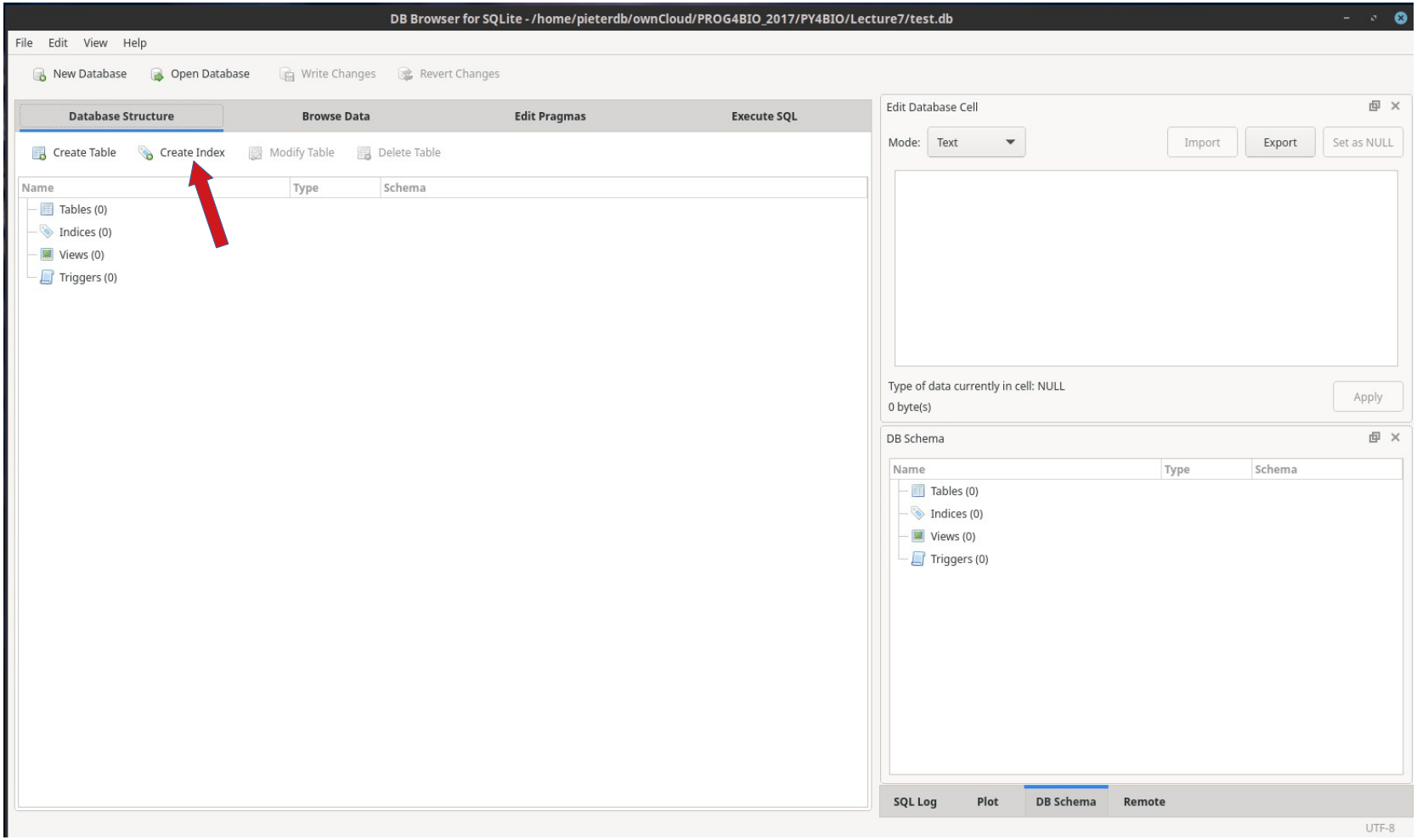


CREATE a table

1. Database data is stored in tables composed of rows and fields of a defined type and size
2. Each field contains one piece of information
3. Use the equivalent of line numbers in a ledger to make it easier to link tables together for querying with an AUTO_INCREMENT field:

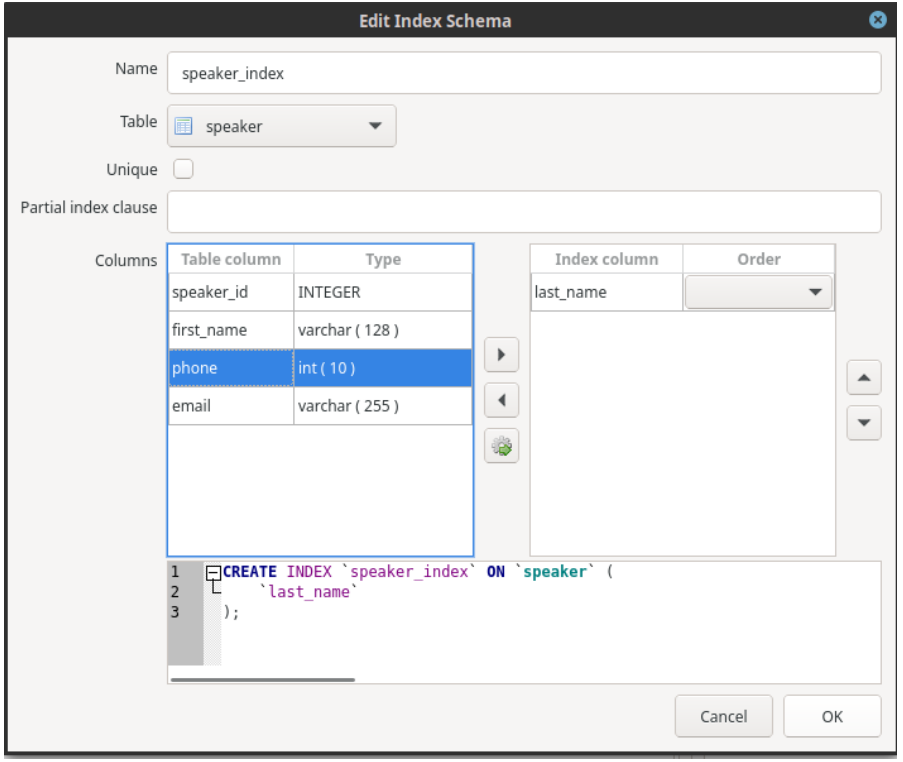


CREATE an INDEX

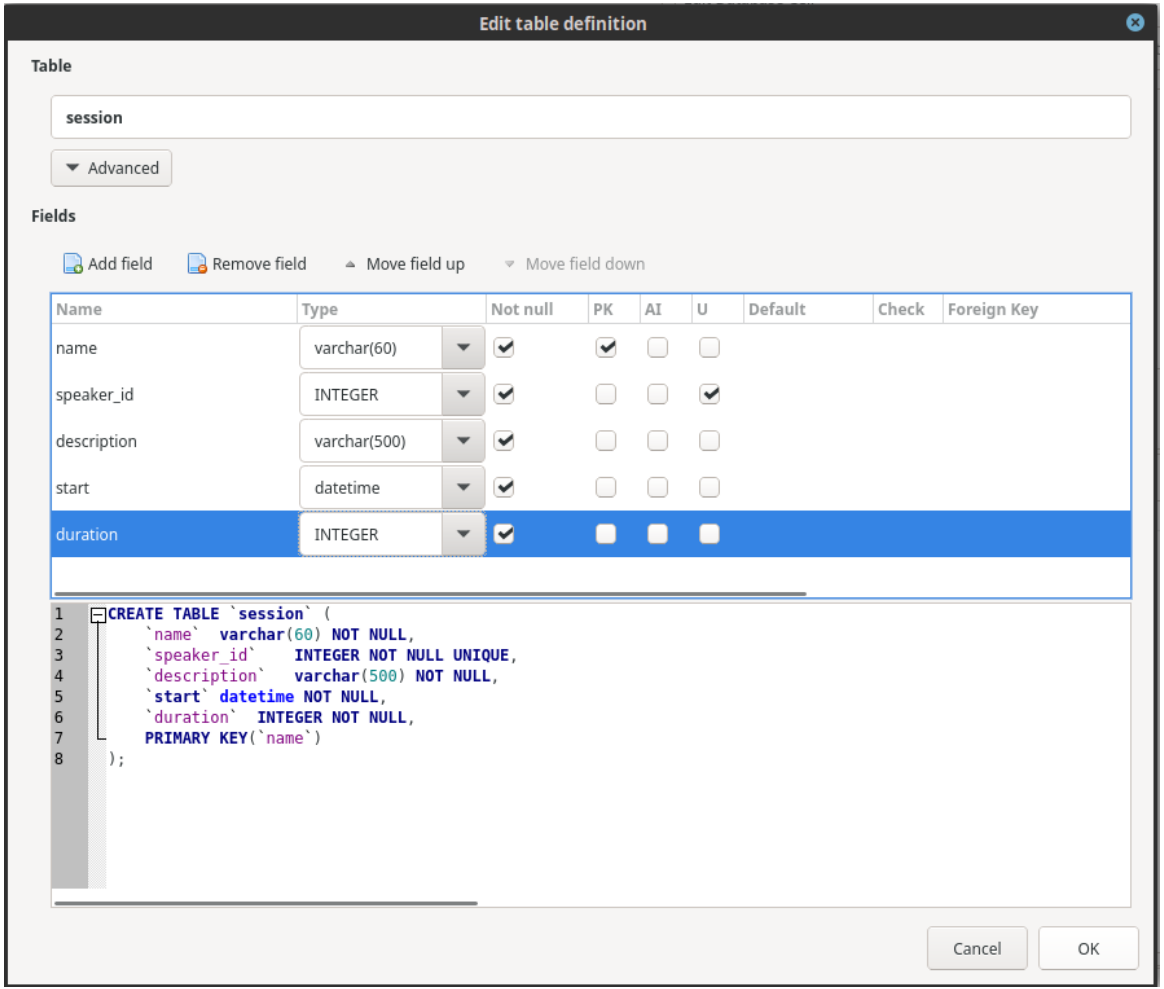


CREATE an INDEX

- Indexes allow the DBMS to skip to the right row (or skip closer to the right row), similar to how the lettered tabs work in an address book

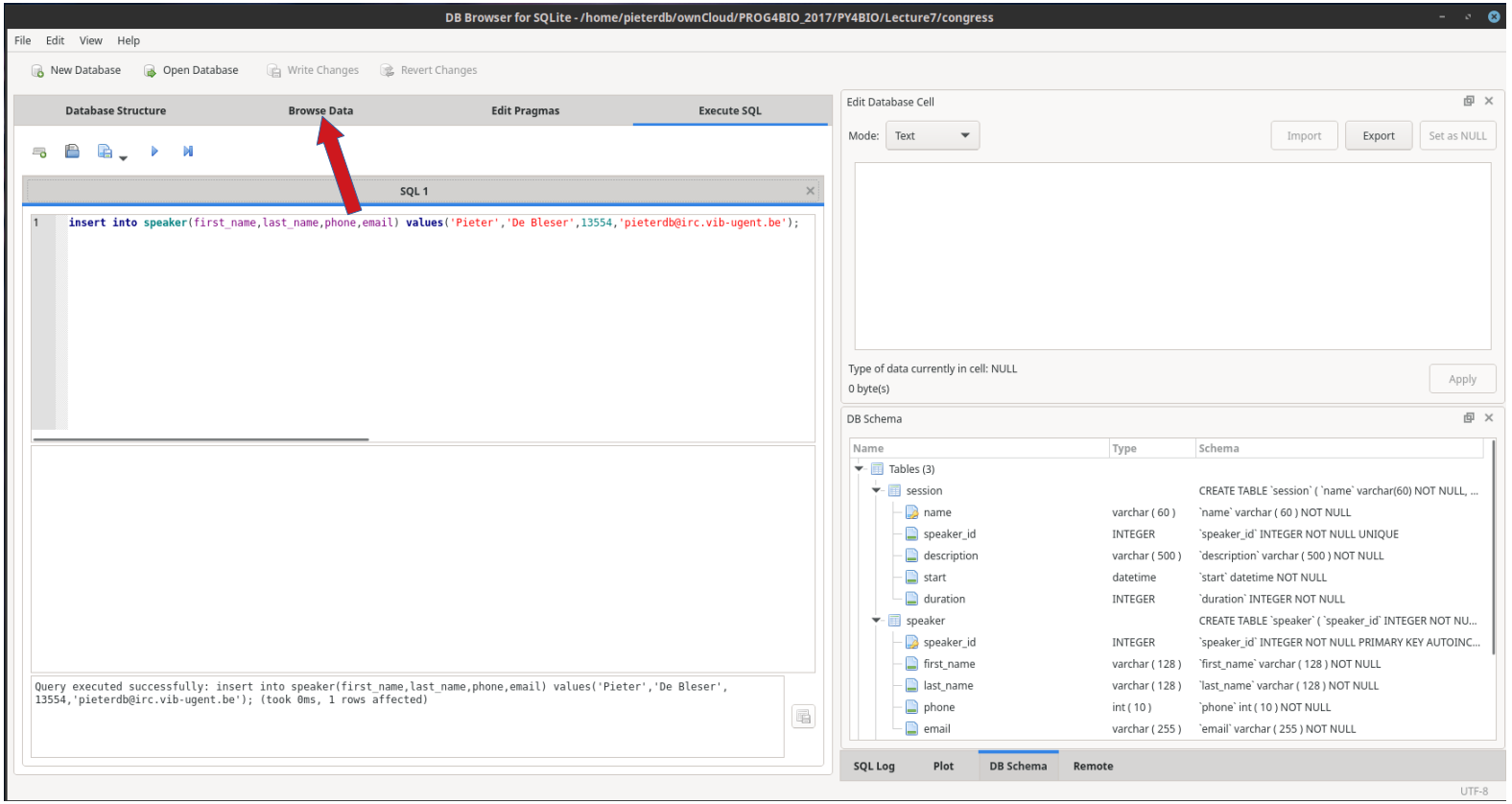


CREATE table ‘session’

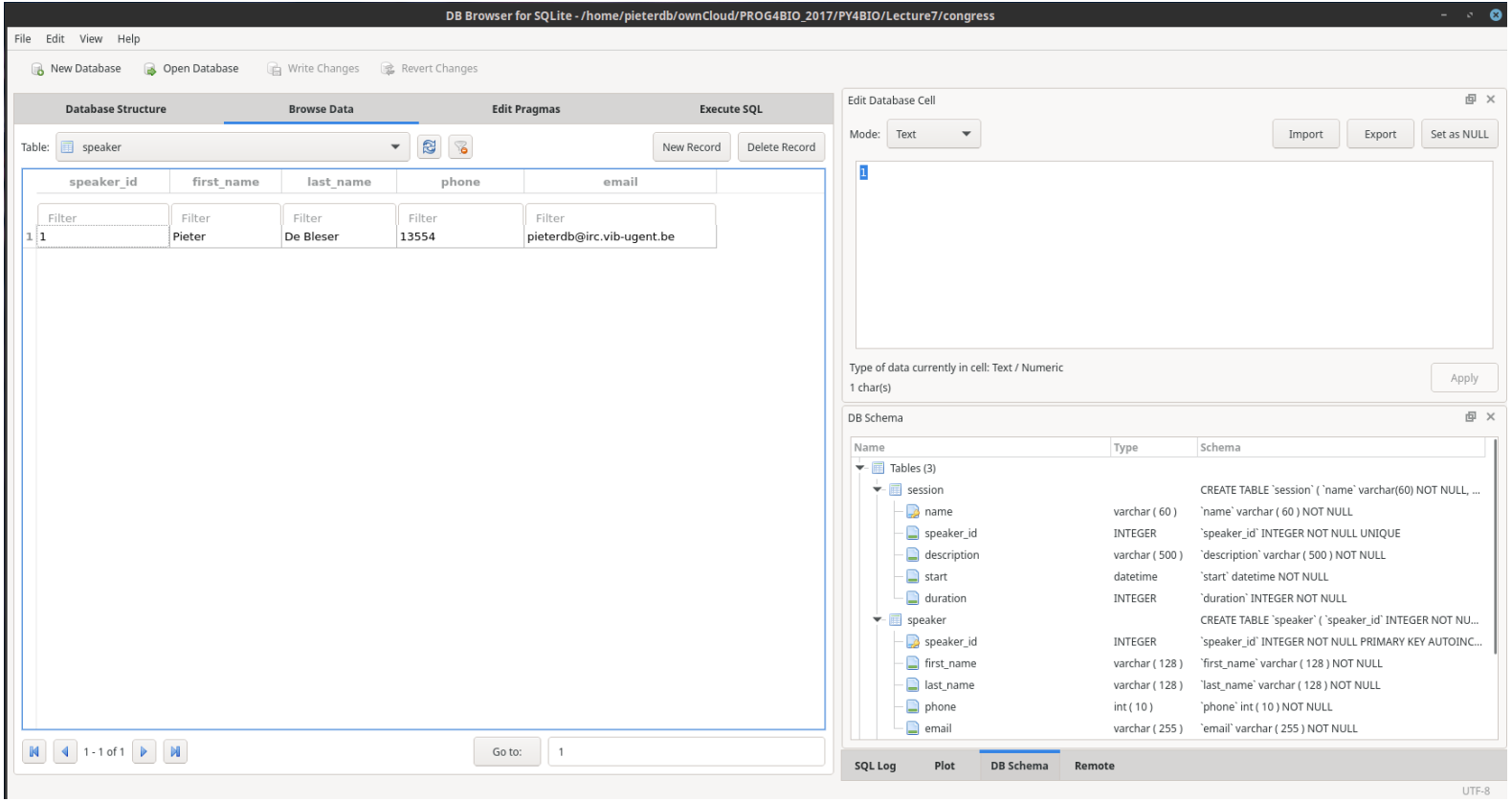


INSERTing Data

Use INSERT statements to add data to a table, specifying which table you are adding data to and what the data is

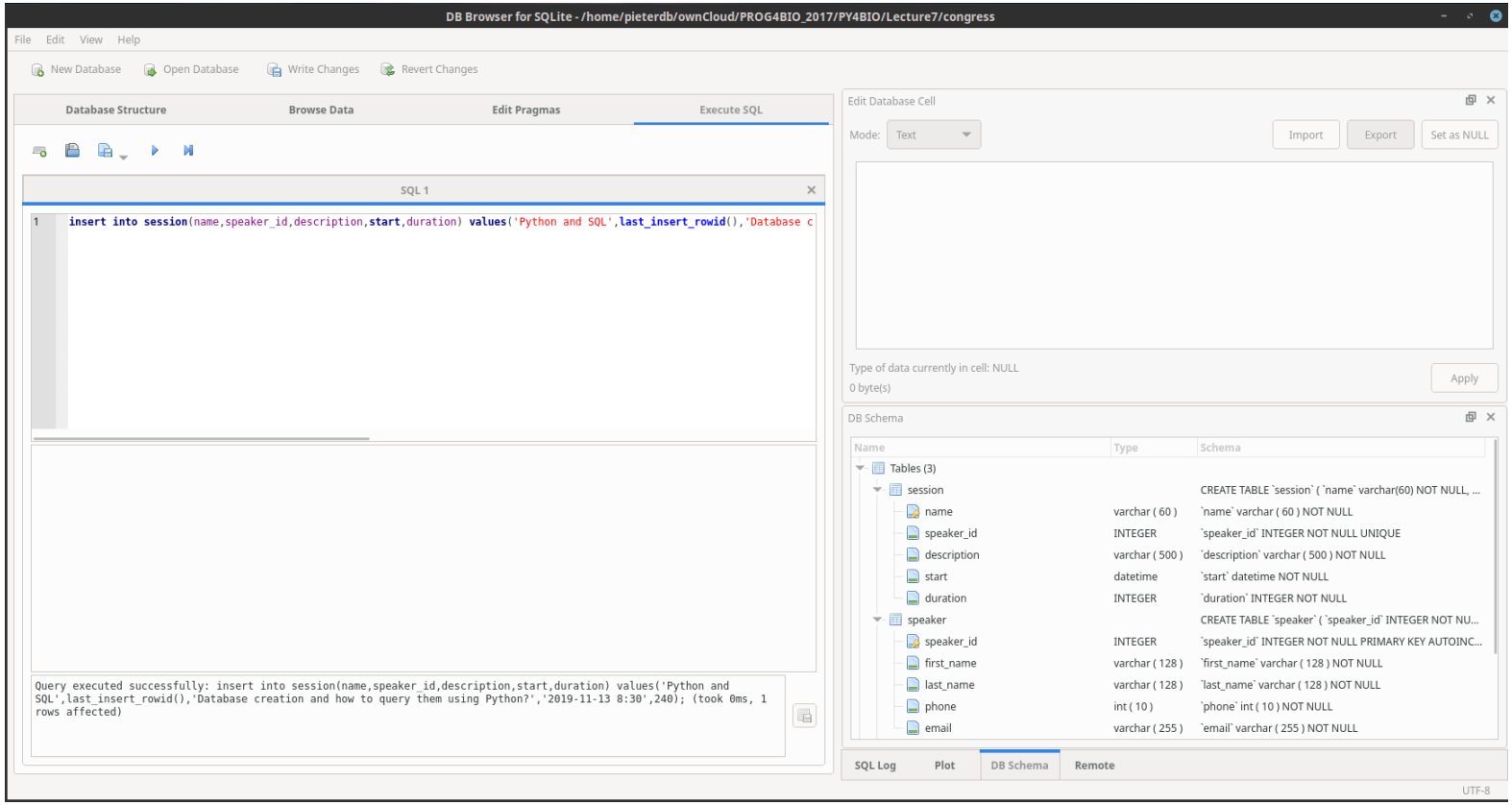


INSERTing Data – Browse Data



INSERTing Data

```
insert into session(name,speaker_id,description,start,duration) values('Python and
SQL',last_insert_rowid(),'Database creation and how to query them using Python?','2019-11-13 8:30',240);
```



Exercise:
Add 3 more records in the ‘speaker’ and ‘session’ tables.

Do not forget to save your changes to the congress database!

Simple SELECT

- Use a SELECT query to retrieve data from one or more tables
- SELECT queries involve a table name and a list of fields to return
- SELECT * FROM speaker;
  - Returns all fields from all rows
- SELECT last_name, first_name, email FROM speaker;
  - Returns just the last and first name and email address

UPDATEing Data

- Use UPDATE statements to change the contents of one or more fields

(Bad)

- `UPDATE speaker SET email = ‘pieterdb@ugent.be’;`

(Good)

- `UPDATE speaker SET email = ‘pieterdb@ugent.be’ WHERE last_name = ‘De Bleser’ AND first_name = ‘Pieter’;`

DELETEing Data

Use the DELETE statement to remove rows from a table:

- DELETE FROM speaker; (Very Bad)
- DELETE FROM speaker WHERE last_name = ‘De Bleser’;
- SELECT FROM speaker WHERE last_name = ‘De Bleser’;
- CREATE TABLE speaker_copy AS SELECT * FROM speaker WHERE 1;

Advanced SELECTs

Concatenating data:

```
SELECT last_name || ', ' || first_name AS full_name FROM speaker;
```

Wildcards:

```
SELECT name, description FROM session WHERE description LIKE '%using%';
```

Advanced SELECTs

Ordering data:

```
SELECT last_name, first_name FROM speaker ORDER BY last_name,
first_name;
```

Using functions:

```
SELECT current_date;
select current_time;
select current_timestamp;
```

Advanced SELECTs

Aliases

```
SELECT last_name || ', ' || first_name AS full_name FROM speaker;
```

Limiting results

```
SELECT * FROM speaker LIMIT 2;
```

Cross-Referencing

- Cartesian Product
  - SELECT * FROM speaker, session;
- Simple JOIN
  - SELECT * FROM speaker, session
    WHERE speaker.speaker_id = session.speaker_id;

Cross-Referencing

- Selecting Columns:

```
SELECT speaker.last_name || ', ' || speaker.first_name AS speaker,
session.name AS session
FROM speaker JOIN session ON
speaker.speaker_id = session.speaker_id;
```

Cross-Referencing

- LEFT JOINS

```
SELECT speaker.last_name || ', ' || speaker.first_name AS speaker,
session.name AS session
FROM speaker LEFT JOIN session ON
speaker.speaker_id = session.speaker_id;
```

- Add `WHERE session.name IS NULL` to get only the speakers without sessions

Python database applications development with the SQLite database.

Topics

- Connecting to the SQLite database from Python and creating a SQLite database and tables.
- SQLite Datatypes and it’s corresponding Python types.
- How to perform SQLite CRUD operation i.e., data insertion, data retrieval, data update, and data deletion from Python.
- How to execute SQLite scripts from Python.
- Insert/Retrieve data in SQLite using Python.
- SQLite error-handling techniques to develop robust python programs.

Steps to connect to SQLite

1. Use the connect() method of a sqlite3 module and pass the database name as an argument to create a connection object.
2. Create a cursor object using the connection object returned by the connect method to execute SQLite queries from Python.
3. Close the cursor object and SQLite database connection object when work is done.
4. Catch database exception if any that may occur during this connection process.

Steps to connect to SQLite - example

sqlite_connect.py

```
import sqlite3

try:
    sqliteConnection = sqlite3.connect('py4bio_meeting.db')
    cursor = sqliteConnection.cursor()
    print("Database created and successfully connected to SQLite")

    sqlite_select_Query = "select sqlite_version();"
    cursor.execute(sqlite_select_Query)
    record = cursor.fetchall()
    print("SQLite database version is: ", record)
    cursor.close()

except sqlite3.Error as error:
    print("Error while connecting to sqlite", error)
finally:
    if (sqliteConnection):
        sqliteConnection.close()
        print("The SQLite connection is closed")
```

```
> python sqlite_connect.py
Database created and successfully connected to SQLite
SQLite database version is:  [('3.29.0',)]
The SQLite connection is closed
```

Notes:

- Using a try-except-finally block: all the code resides in the try-except block to catch the SQLite database exceptions and error that may occur during this process.
- Using the sqlite3.Error class of sqlite3 module allows the handling of any database error and exception that may occur while working with SQLite from Python.

This approach makes the application robust.
The sqlite3.Error class helps to understand the error in detail as it returns an error message and error code.

How to create a SQLite table

Steps for creating a table in SQLite from Python:

1. Connect to SQLite using a sqlite3.connect().
2. Prepare a create table query.
3. Execute the query using a cursor.execute(query)
4. Close the SQLite database connection and cursor object.

How to create a SQLite table - ‘speaker’

```
import sqlite3

try:
    sqliteConnection = sqlite3.connect('py4bio_meeting.db')
    sqlite_create_table_query = '''CREATE TABLE `speaker` (
                                    `speaker_id`        INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
                                    `first_name`        varchar ( 128 ) NOT NULL,
                                    `last_name` varchar ( 128 ) NOT NULL,
                                    `phone`     int ( 10 ) NOT NULL,
                                    `email`     varchar ( 255 ) NOT NULL);'''

    cursor = sqliteConnection.cursor()
    print("Successfully Connected to SQLite")
    cursor.execute(sqlite_create_table_query)
    sqliteConnection.commit()
    print("SQLite table created")

    cursor.close()

except sqlite3.Error as error:
    print("Error while creating a sqlite table", error)
finally:
    if (sqliteConnection):
        sqliteConnection.close()
        print("sqlite connection is closed")
```

```
> python sqlite_create_table.py
Successfully Connected to SQLite
SQLite table created
sqlite connection is closed
```

Exercise: create the SQLite table - ‘session’

Create and execute a new script ‘sqlite_create_table_session.py’ using following table definition:

```
CREATE TABLE `session` (
	`name`	varchar ( 60 ) NOT NULL,
	`speaker_id`	INTEGER NOT NULL UNIQUE,
	`description`	varchar ( 500 ) NOT NULL,
	`start`	datetime NOT NULL,
	`duration`	INTEGER NOT NULL,
	PRIMARY KEY(`name`)
);
```

Check the result in the SQLite DB Browser.
Write a script combining the creation of the ‘speaker’ and ‘session’ tables.
Can you run this script without errors? Why (not)?

SQLite Datatypes and corresponding Python types

SQLite DataTypes:

NULL: – The value is a NULL value.
INTEGER: – To store the numeric value. The integer stored in 1, 2, 3, 4, 6, or 8 bytes depending on the magnitude of the number.
REAL: – The value is a floating-point value, for example, 3.14 value of PI
TEXT: – The value is a text string, TEXT value stored using the UTF-8, UTF-16BE or UTF-16LE encoding.
BLOB: – The value is a blob of data, i.e., binary data. It is used to store images and files.

Compatible Python versus SQLite data types:

Python type	SQLite type
None	NULL
int	INTEGER
float	REAL
str	TEXT
bytes	BLOB

Perform SQLite CRUD Operations from Python

1. Python **Insert** into SQLite Table
2. Python **Select** from SQLite Table
3. Python **Update** SQLite Table
4. Python **Delete** from SQLite Table

Python **Insert** into SQLite Table

How to:

1. Insert single and multiple rows into the SQLite table.
2. Insert Integer, string, float, double, and datetime values into a SQLite table.
3. Use a parameterized query to insert Python variables as dynamic data into a table.

How to insert a single row/record into SQLite table?

Steps to take:

1. Establish a SQLite connection from Python.
2. Create a cursor object using the connection object.
3. Define the SQLite INSERT Query. Here you need to know the table, and it’s column details.
4. Execute the INSERT query using the cursor.execute()
5. Commit your changes to the database.
6. Close the SQLite database connection.
7. Catch SQLite exceptions if any.
8. verify the result by selecting data from SQLite table.

How to insert a single row/record into SQLite table?

sqlite_insert_single_record.py

```
import sqlite3

try:
    sqliteConnection = sqlite3.connect('py4bio_meeting.db')
    cursor = sqliteConnection.cursor()
    print("Successfully Connected to SQLite")

    sqlite_insert_speaker = """insert into speaker(first_name,last_name,phone,email)
                               values('Pieter','De Bleser',13554,'pieterdb@irc.vib-ugent.be');"""

    count = cursor.execute(sqlite_insert_speaker)
    sqliteConnection.commit()
    print("Record inserted successfully into speaker table ", cursor.rowcount)
    cursor.close()

except sqlite3.Error as error:
    print("Failed to insert data into sqlite table", error)
finally:
    if (sqliteConnection):
        sqliteConnection.close()
        print("The SQLite connection is closed")
```

```
> python sqlite_insert_single_record.py
Successfully Connected to SQLite
Record inserted successfully into speaker table  1
The SQLite connection is closed
```

Using Python variables in SQLite INSERT query

We use a parameterized query to insert Python variables into the table.
A parameterized query is a query in which placeholders used for parameters and the parameter values supplied at execution time. That means parameterized query gets compiled only once.

```python
import sqlite3

def insertVariableIntoTable(first_name, last_name, phone, email):
    try:
        sqliteConnection = sqlite3.connect('py4bio_meeting.db')
        cursor = sqliteConnection.cursor()
        print("Connected to SQLite")

        sqlite_insert_with_param = """insert into speaker(first_name,last_name,phone,email)
                                values(?,?,?,?);"""

        data_tuple = (first_name,last_name,phone,email)
        cursor.execute(sqlite_insert_with_param, data_tuple)
        sqliteConnection.commit()
        print("Python Variables inserted successfully into speaker table")

        cursor.close()

    except sqlite3.Error as error:
        print("Failed to insert Python variable into sqlite table", error)
    finally:
        if (sqliteConnection):
            sqliteConnection.close()
            print("The SQLite connection is closed")

insertVariableIntoTable('Joe', 'Bonamassa',23987,'joe@blues.org')
insertVariableIntoTable('Carlos', 'Santana',77755,'carlos@blues.org')
insertVariableIntoTable('Linus', 'Torvalds',88888,'linus@.linux.org')
```

```
> python sqlite_insert_single_record_parameterized.py
Connected to SQLite
Python Variables inserted successfully into speaker table
The SQLite connection is closed
Connected to SQLite
Python Variables inserted successfully into speaker table
The SQLite connection is closed
Connected to SQLite
Python Variables inserted successfully into speaker table
The SQLite connection is closed
```

Python Insert multiple rows into SQLite table

In the previous example, we have used the execute() method of the cursor object to insert a single record but sometimes we need to insert multiple rows into the table in a single insert query.

A bulk insert operation in a single query can be done using the cursor.executemany() method.

cursor.executemany() accepts two arguments: SQL query and a records list.

```python
import sqlite3

def insertMultipleRecords(recordList):
    try:
        sqliteConnection = sqlite3.connect('py4bio_meeting.db')
        cursor = sqliteConnection.cursor()
        print("Connected to SQLite")

        sqlite_insert_query = """insert into speaker(first_name,last_name,phone,email)
                                 values(?,?,?,?);"""

        cursor.executemany(sqlite_insert_query, recordList)
        sqliteConnection.commit()
        print("Total", cursor.rowcount, "Records inserted successfully into speaker table")
        sqliteConnection.commit()
        cursor.close()

    except sqlite3.Error as error:
        print("Failed to insert multiple records into sqlite table", error)
    finally:
        if (sqliteConnection):
            sqliteConnection.close()
            print("The SQLite connection is closed")

recordsToInsert = [('Jos','Vermeulen',55555,'jos@gmail.com'),
                   ('Chris','De Wilde',44444,'chris@gmail.com'),
                   ('Jonny','Winter',22222,'jonny@gmail.com')]

insertMultipleRecords(recordsToInsert)
```

```
> python sqlite_insert_multiple_records.py
Connected to SQLite
Total 3 Records inserted successfully into speaker table
The SQLite connection is closed
```

Python Select from SQLite Table

Goals:
How to use the Python built-in module sqlite3 to fetch rows from a SQLite table to:

1. Fetch all rows using cursor.fetchall()
2. Use cursor.fetchmany(size) to fetch limited rows
3. Fetch only one single row using cursor.fetchone()
4. Use the Python variable in the SQLite Select query to pass dynamic values to the query.

Steps to fetch rows from SQLite table

1. Establish SQLite Connection from Python.
2. Define the SQLite SELECT statement query. Here you need to know the table, and it’s column details.
3. Execute the SELECT query using the cursor.execute() method.
4. Get rows from the cursor object using a cursor.fetchall()
5. Iterate over the rows and get each row and its column value.
6. Close the Cursor and SQLite database connection.
7. Catch any SQLite exceptions that may come up during the process.

sqlite_fetch_all_rows.py

```
import sqlite3

def readSqliteTable():
    try:
        sqliteConnection = sqlite3.connect('py4bio_meeting.db')
        cursor = sqliteConnection.cursor()
        print("Connected to SQLite")

        sqlite_select_query = """SELECT * from speaker"""
        cursor.execute(sqlite_select_query)
        records = cursor.fetchall()
        print("Total rows are:  ", len(records))
        print("Printing each row")
        for row in records:
            print("speaker_id: ", row[0])
            print("first_name: ", row[1])
            print("last_name: ", row[2])
            print("phone: ", row[3])
            print("email: ", row[4])
            print("\n")

        cursor.close()

    except sqlite3.Error as error:
        print("Failed to read data from sqlite table", error)
    finally:
        if (sqliteConnection):
            sqliteConnection.close()
            print("The SQLite connection is closed")

readSqliteTable()
```

```
> python sqlite_fetch_all_rows.py
Connected to SQLite
Total rows are:   7
Printing each row
speaker_id:  1
first_name:  Pieter
last_name:  De Bleser
phone:  13554
email:  pieterdb@irc.vib-ugent.be


speaker_id:  2
first_name:  Joe
last_name:  Bonamassa
phone:  23987
email:  joe@blues.org


speaker_id:  3
first_name:  Carlos
last_name:  Santana
phone:  77755
email:  carlos@blues.org


speaker_id:  4
first_name:  Linus
last_name:  Torvalds
phone:  88888
email:  linus@.linux.org

...
```

Use Python variables as parameters in SQLite Select Query

Many times we need to pass a variable to SQLite select query in the where clause to check some condition.To handle such a requirement, we need to use a parameterized query.
A parameterized query is a query in which placeholders used for parameters and the parameter values supplied at execution time. That means parameterized query gets compiled only once.

```
import sqlite3

def getSpeakerInfo(speaker_id):
    try:
        sqliteConnection = sqlite3.connect('py4bio_meeting.db')
        cursor = sqliteConnection.cursor()
        print("Connected to SQLite")

        sql_select_query = """select * from speaker where speaker_id = ?"""
        cursor.execute(sql_select_query, (speaker_id,))
        records = cursor.fetchall()
        print("Printing ID ", speaker_id)
        for row in records:
            print("speaker_id: ", row[0])
            print("first_name: ", row[1])
            print("last_name: ", row[2])
            print("phone: ", row[3])
            print("email: ", row[4])
            print("\n")
        cursor.close()

    except sqlite3.Error as error:
        print("Failed to read data from sqlite table", error)
    finally:
        if (sqliteConnection):
            sqliteConnection.close()
            print("The SQLite connection is closed")

getSpeakerInfo(2)
```

```
> python sqlite_fetch_records_parameterized.py
Connected to SQLite
Printing ID  2
speaker_id:  2
first_name:  Joe
last_name:  Bonamassa
phone:  23987
email:  joe@blues.org
```

Select limited rows from SQLite table using cursor.fetchmany()

In some circumstances to fetch all the data rows from a table is a time-consuming task if a table contains thousands of rows.

To fetch all rows, we have to use more resources, so we need more space and processing time. To enhance performance it is advisable to use the fetchmany(SIZE) method of the cursor class to fetch fewer rows.

Using cursor.fetchmany(size) method, we can specify how many rows we want to read.

```
import sqlite3

def readLimitedRows(rowSize):
    try:
        sqliteConnection = sqlite3.connect('py4bio_meeting.db')
        cursor = sqliteConnection.cursor()
        print("Connected to SQLite")

        sqlite_select_query = """SELECT * from speaker"""
        cursor.execute(sqlite_select_query)
        print("Reading ", rowSize, " rows")
        records = cursor.fetchmany(rowSize)
        print("Printing each row \n")
        for row in records:
            print("speaker_id: ", row[0])
            print("first_name: ", row[1])
            print("last_name: ", row[2])
            print("phone: ", row[3])
            print("email: ", row[4])
            print("\n")

        cursor.close()

    except sqlite3.Error as error:
        print("Failed to read data from sqlite table", error)
    finally:
        if (sqliteConnection):
            sqliteConnection.close()
            print("The SQLite connection is closed")

readLimitedRows(2)
```

```
> python sqlite_fetch_many.py
Connected to SQLite
Reading  2  rows
Printing each row

speaker_id:  1
first_name:  Pieter
last_name:  De Bleser
phone:  13554
email:  pieterdb@irc.vib-
ugent.be


speaker_id:  2
first_name:  Joe
last_name:  Bonamassa
phone:  23987
email:  joe@blues.org


The SQLite connection is
closed
```

Select a single row from SQLite table using cursor.fetchone()

- When you want to read only one row from the SQLite table, use the fetchone() method of a cursor class.
- This method is used also in the situations when you know the query is going to return only one row.
- The cursor.fetchone() method retrieves the next row from the result set.
  - This method returns a single record or None if no more rows are available.

```
import sqlite3

def readSingleRow(speaker_id):
    try:
        sqliteConnection = sqlite3.connect('py4bio_meeting.db')
        cursor = sqliteConnection.cursor()
        print("Connected to SQLite")

        sqlite_select_query = """SELECT * from speaker where speaker_id = ?"""
        cursor.execute(sqlite_select_query, (speaker_id, ))
        print("Reading single row \n")
        record = cursor.fetchone()
        print("speaker_id: ", record[0])
        print("first_name: ", record[1])
        print("last_name: ", record[2])
        print("phone: ", record[3])
        print("email: ", record[4])
        print("\n")

        cursor.close()

    except sqlite3.Error as error:
        print("Failed to read single row from sqlite table", error)
    finally:
        if (sqliteConnection):
            sqliteConnection.close()
            print("The SQLite connection is closed")

readSingleRow(3)
```

```
> python sqlite_fetch_one.py
Connected to SQLite
Reading single row

speaker_id:  3
first_name:  Carlos
last_name:  Santana
phone:  77755
email:  carlos@blues.org


The SQLite connection is closed
```

Python Update SQLite Table

Topics:

- Update single and multiple columns of a row
- Use a parameterized query to provide value at runtime to the update query.
- Update a column with date-time and timestamp values
- Perform bulk update using a single query.

Steps to update a single record of SQLite table

1. Establish the SQLite connection from Python.
2. Create a cursor object using the connection object.
3. Define the SQLite UPDATE Query. Here you need to know the table, and it’s column name which you want to update.
4. Execute the UPDATE query using the cursor.execute()
5. After the successful execution of a SQLite update query, Don’t forget to commit your changes to the database.
6. Close the SQLite database connection.
7. Catch SQLite exceptions if any.
8. Verify the result by selecting data from a SQLite table from Python.

```python
import sqlite3

def updateSqliteTable():
    try:
        sqliteConnection = sqlite3.connect('py4bio_meeting.db')
        cursor = sqliteConnection.cursor()
        print("Connected to SQLite")

        sql_update_query = """Update speaker set phone = 10000 where speaker_id = 4"""
        cursor.execute(sql_update_query)
        sqliteConnection.commit()
        print("Record Updated successfully ")
        cursor.close()

    except sqlite3.Error as error:
        print("Failed to update sqlite table", error)
    finally:
        if (sqliteConnection):
            sqliteConnection.close()
            print("The SQLite connection is closed")

updateSqliteTable()
```

```
> python sqlite_update_single_record.py
Connected to SQLite
Record Updated successfully
The SQLite connection is closed
```

Using Python variables in SQLite UPDATE query

Most of the time, we need to update a table with some runtime values. For example, when users updating their profile or any other details through User Interface in such cases, we need to update a table with those new values.

In such circumstances, It is always best practice to use a parameterized query. The parameterized query uses placeholders (?) inside SQL statements that contain input from users. It helps us to update runtime values and prevent SQL injection concerns.

```python
import sqlite3

def updateSqliteTable(speaker_id, phone):
    try:
        sqliteConnection = sqlite3.connect('py4bio_meeting.db')
        cursor = sqliteConnection.cursor()
        print("Connected to SQLite")

        sql_update_query = """Update speaker set phone = ? where speaker_id = ?"""
        data = (phone, speaker_id)
        cursor.execute(sql_update_query, data)
        sqliteConnection.commit()
        print("Record Updated successfully")
        cursor.close()

    except sqlite3.Error as error:
        print("Failed to update sqlite table", error)
    finally:
        if (sqliteConnection):
            sqliteConnection.close()
            print("The sqlite connection is closed")

updateSqliteTable(3, 75006)
```

```
> python sqlite_update_parameterized.py
Connected to SQLite
Record Updated successfully
The sqlite connection is closed
```

Update multiple rows of SQLite table using cursor’s executemany()

In the above example, we have used execute() method of cursor object to update a single record, but sometimes in Python application, we need to update multiple rows of the SQLite table. For example, you want to update the phone numbers of many speakers at once.

So instead of executing the UPDATE query every time to update each record, you can perform bulk update operation in a single query. We can modify multiple records of the SQLite table in a single query using the cursor.executemany() method.

The cursor.executemany(query, seq_param) method accepts two arguments:

1. SQL query

2. List of records to be updated.

```python
import sqlite3

def updateMultipleRecords(recordList):
    try:
        sqliteConnection = sqlite3.connect('py4bio_meeting.db')
        cursor = sqliteConnection.cursor()
        print("Connected to SQLite")

        sqlite_update_query = """Update speaker set phone = ? where speaker_id = ?"""
        cursor.executemany(sqlite_update_query, recordList)
        sqliteConnection.commit()
        print("Total", cursor.rowcount, "Records updated successfully")
        sqliteConnection.commit()
        cursor.close()

    except sqlite3.Error as error:
        print("Failed to update multiple records of sqlite table", error)
    finally:
        if (sqliteConnection):
            sqliteConnection.close()
            print("The SQLite connection is closed")

records_to_update = [ (12345, 4), (45678, 5), (98765, 6) ]
updateMultipleRecords(records_to_update)
```

```
> python sqlite_update_multiple_records.py
Connected to SQLite
Total 3 Records updated successfully
The SQLite connection is closed
```

Updating multiple Columns of a SQLite table

```
import sqlite3

def updateMultipleColumns(speaker_id, phone, email):
    try:
        sqliteConnection = sqlite3.connect('py4bio_meeting.db')
        cursor = sqliteConnection.cursor()
        print("Connected to SQLite")

        sqlite_update_query = """Update speaker set phone = ?, email = ? where speaker_id = ?"""
        columnValues = (phone, email, speaker_id)
        cursor.execute(sqlite_update_query, columnValues)
        sqliteConnection.commit()
        print("Multiple columns updated successfully")
        sqliteConnection.commit()
        cursor.close()

    except sqlite3.Error as error:
        print("Failed to update multiple columns of sqlite table", error)
    finally:
        if (sqliteConnection):
            sqliteConnection.close()
            print("sqlite connection is closed")

updateMultipleColumns(3, 66666, 'vlad.dracula@gmail.com')
```

```
> python sqlite_update_multiple_columns.py
Connected to SQLite
Multiple columns updated successfully
sqlite connection is closed
```

Python Delete from SQLite Table

How to:

- Delete a single row, multiple rows, all rows, single column, and multiple columns from SQLite table using Python.
- Use a Python parameterized query to provide value at runtime to the SQLite delete query.
- Commit and rollback the delete operation.
- Perform bulk delete using a single query.

Steps to delete a single row from the SQLite table

1. Connect to SQLite from Python.
2. Create a cursor object using the SQLite connection object.
3. Define the SQLite DELETE Query. Here you need to know the table, and it’s column name on which you want to perform delete operation.
4. Execute the DELETE query using the cursor.execute()
5. After the successful execution of an SQLite delete query, commit your changes to the database.
6. Close the SQLite database connection.
7. Catch SQLite exceptions if any.
8. Verify the result by selecting data from SQLite table from Python.

```python
import sqlite3

def deleteRecord():
    try:
        sqliteConnection = sqlite3.connect('py4bio_meeting.db')
        cursor = sqliteConnection.cursor()
        print("Connected to SQLite")

        # Deleting single record now
        sql_delete_query = """DELETE from speaker where speaker_id = 6"""
        cursor.execute(sql_delete_query)
        sqliteConnection.commit()
        print("Record deleted successfully ")
        cursor.close()

    except sqlite3.Error as error:
        print("Failed to delete record from sqlite table", error)
    finally:
        if (sqliteConnection):
            sqliteConnection.close()
            print("the sqlite connection is closed")

deleteRecord()
```

```
> python sqlite_delete_single_record.py
Connected to SQLite
Record deleted successfully
the sqlite connection is closed
```

Use parameterized query to delete a row from a SQLite table

```
import sqlite3

def deleteSqliteRecord(speaker_id):
    try:
        sqliteConnection = sqlite3.connect('py4bio_meeting.db')
        cursor = sqliteConnection.cursor()
        print("Connected to SQLite")

        sql_update_query = """DELETE from speaker where speaker_id = ?"""
        cursor.execute(sql_update_query, (speaker_id, ))
        sqliteConnection.commit()
        print("Record deleted successfully")

        cursor.close()

    except sqlite3.Error as error:
        print("Failed to delete reocord from a sqlite table", error)
    finally:
        if (sqliteConnection):
            sqliteConnection.close()
            print("sqlite connection is closed")

deleteSqliteRecord(5)
```

```
> python sqlite_delete_single_record_parameterized.py
Connected to SQLite
Record deleted successfully
sqlite connection is closed
```

Delete multiple rows from SQLite table

Instead of executing delete query every time to delete each record, we can perform a bulk delete operation in a single query from Python.

We can delete multiple records of the SQLite table in a single query using the cursor.executemany() method.

The cursor.executemany(query, seq_param) method accepts two arguments SQL query and List of record to delete.

```python
import sqlite3

def deleteMultipleRecords(speaker_idList):
    try:
        sqliteConnection = sqlite3.connect('py4bio_meeting.db')
        cursor = sqliteConnection.cursor()
        print("Connected to SQLite")
        sqlite_update_query = """DELETE from speaker where speaker_id = ?"""

        cursor.executemany(sqlite_update_query, speaker_idList)
        sqliteConnection.commit()
        print("Total", cursor.rowcount, "Records deleted successfully")
        sqliteConnection.commit()
        cursor.close()

    except sqlite3.Error as error:
        print("Failed to delete multiple records from sqlite table", error)
    finally:
        if (sqliteConnection):
            sqliteConnection.close()
            print("sqlite connection is closed")

speaker_idsToDelete = [(4,),(3,)]
deleteMultipleRecords(speaker_idsToDelete)
```

```
> python sqlite_delete_multiple_records.py
Connected to SQLite
Total 2 Records deleted successfully
sqlite connection is closed
```

Exercise:

You are given the ‘refGene_hg19.db’.

Write a Python SQLite script that accepts a human gene symbol e.g. SMAD3 and returns the list of reference sequence Ids (RefSeQ Ids) associated with it...

Acknowledgements / Contributions

Initial Development: Charles Severance, University of Michigan School of Information

… Insert new Contributors here