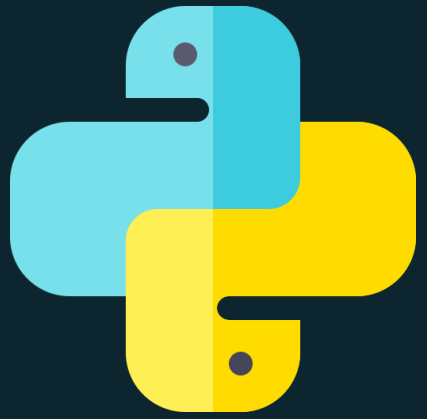
Brief Python

Python Course for Programmers

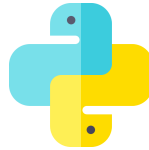


Learn Python Language for Data Science

CHAPTER 12A: DATABASE (PYTHON)

DR. ERIC CHOU

IEEE SENIOR MEMBER

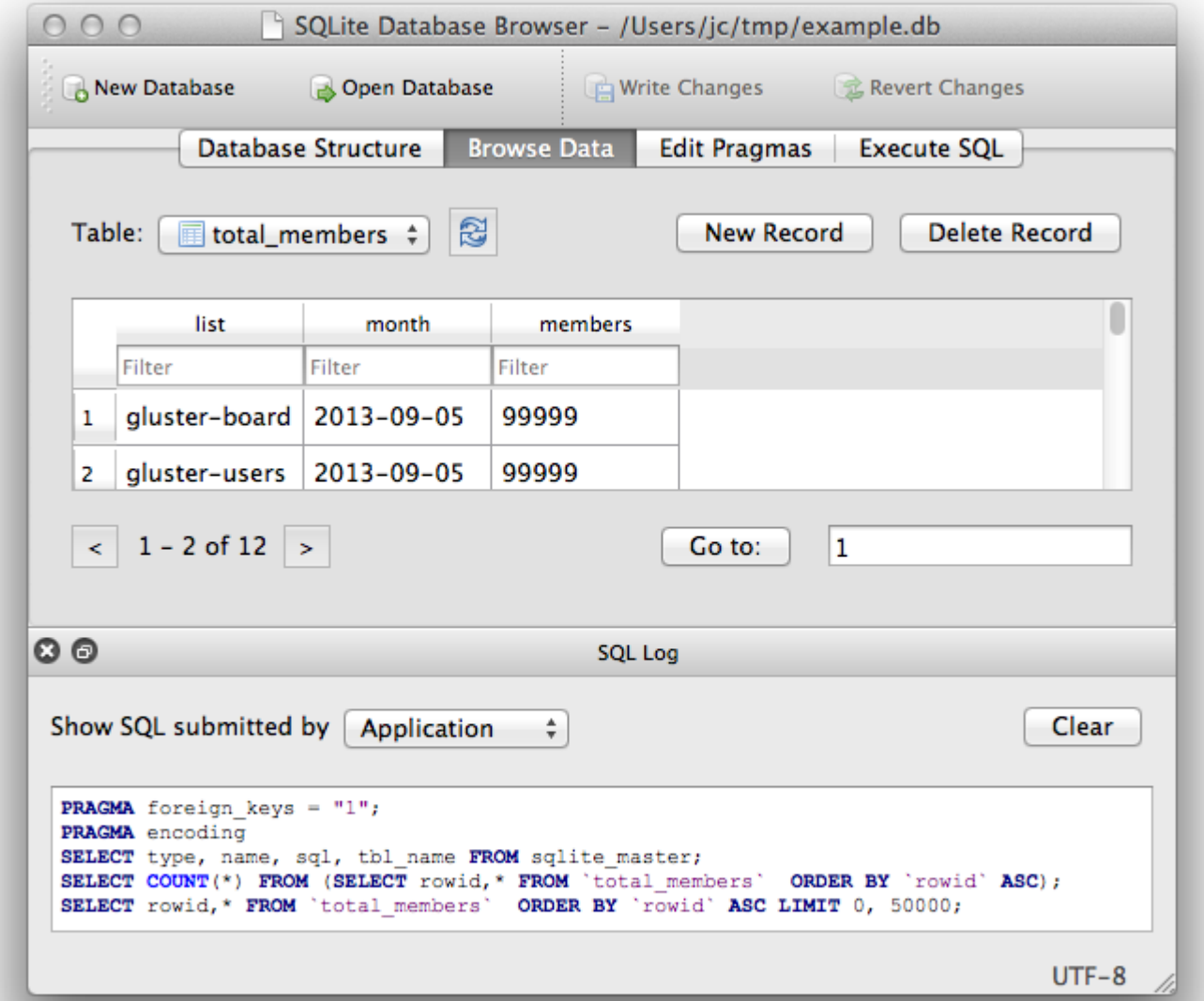


Objectives

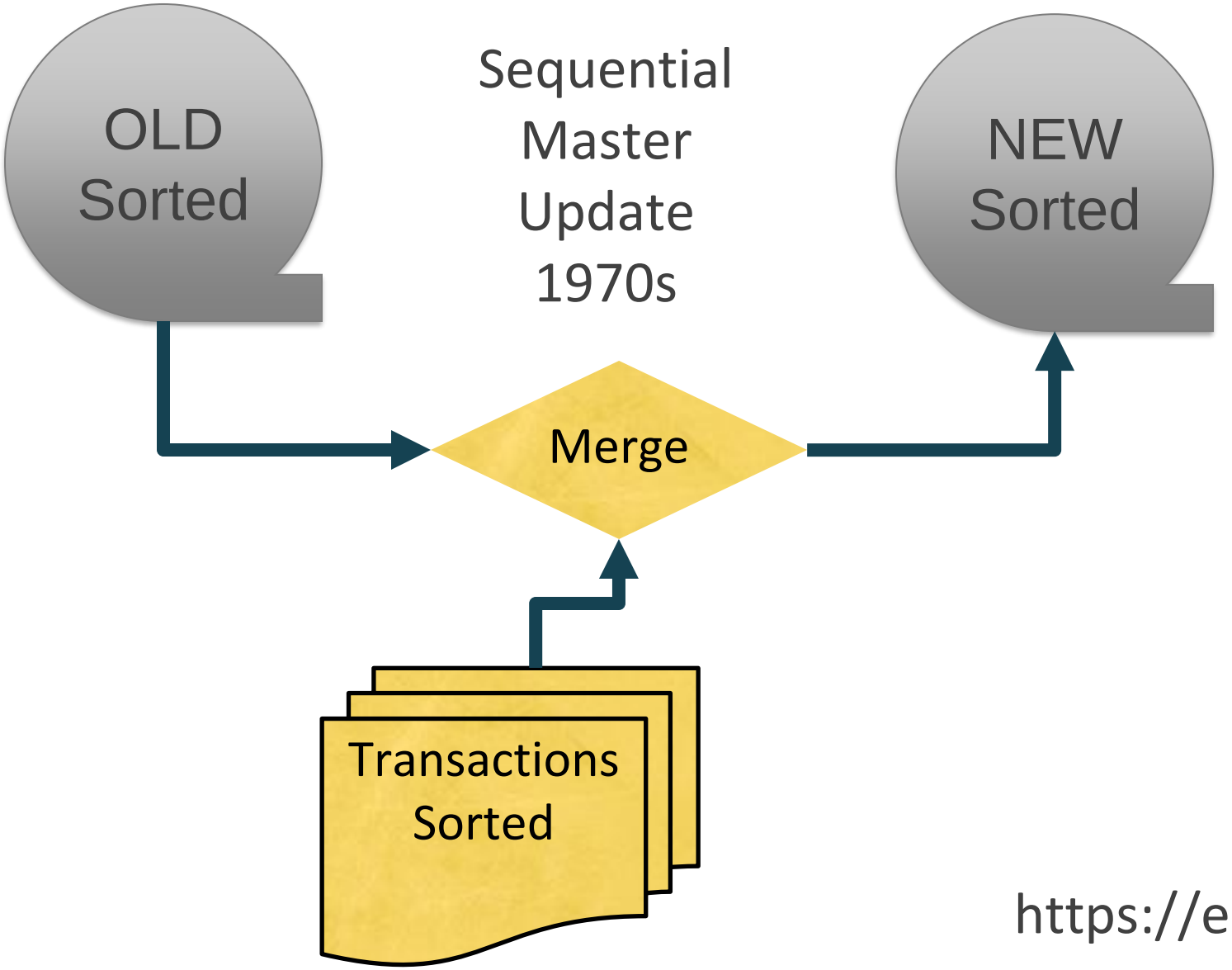
- DB Browser for SQLite
- SQL Database
- SQL Table Design
- Data Models
- Relational Database

DB Browser for SQLite

LECTURE 1



http://sqlitebrowser.org/



https://en.wikipedia.org/wiki/IBM_729

Random Access

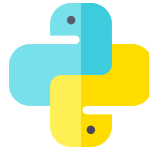
- When you can randomly access data...
- How can you layout data to be most efficient?
- Sorting might not be the best idea

https://en.wikipedia.org/wiki/Hard_disk_drive_platter

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



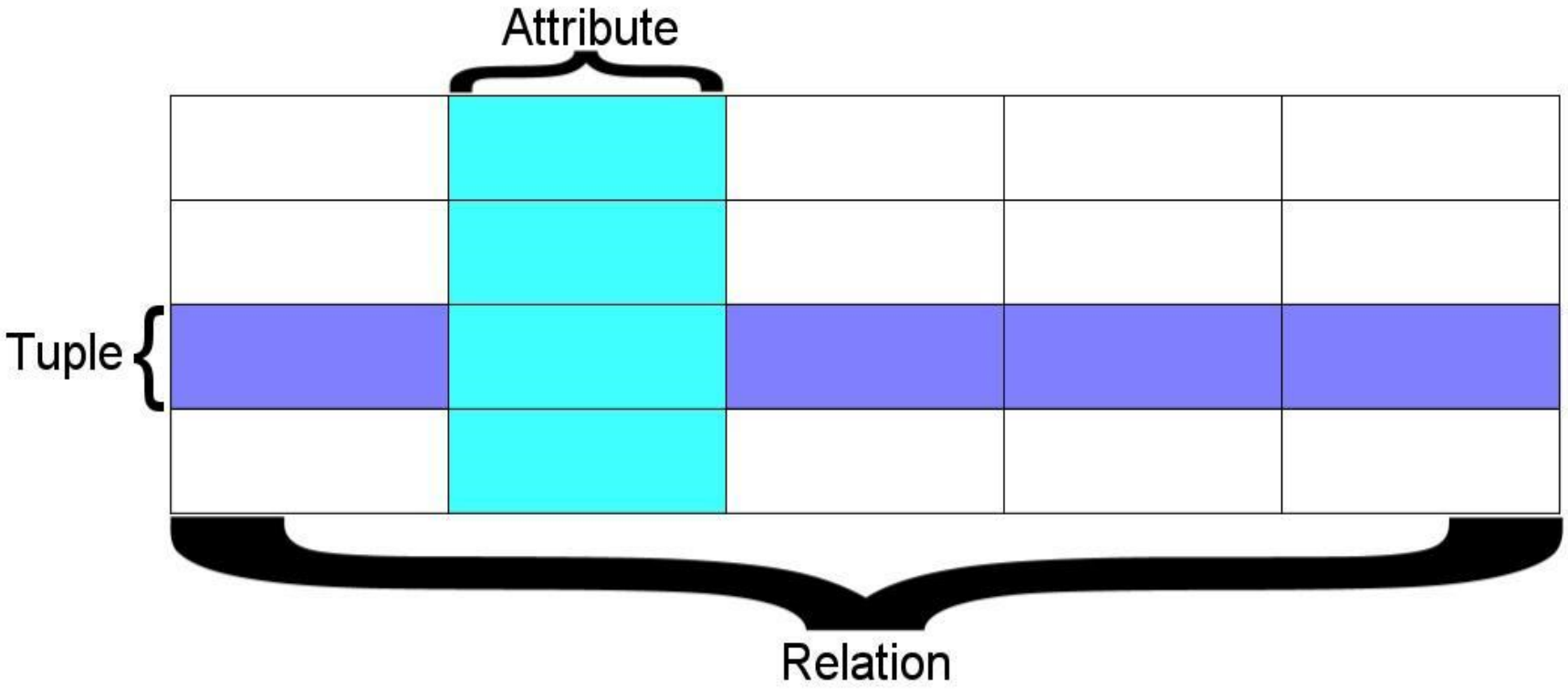
Terminology

Database - contains many tables

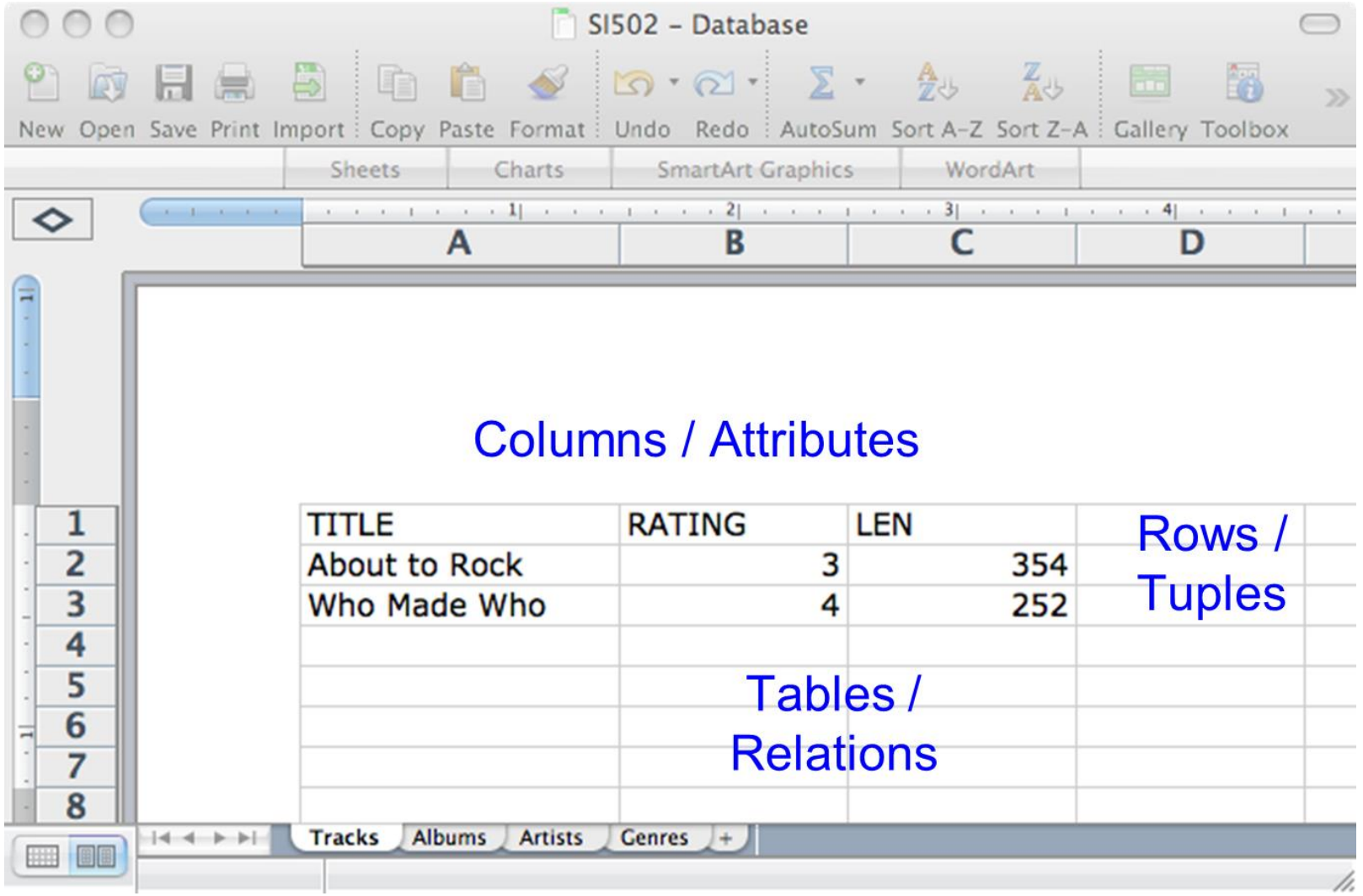
Relation (or table) - contains tuples and attributes

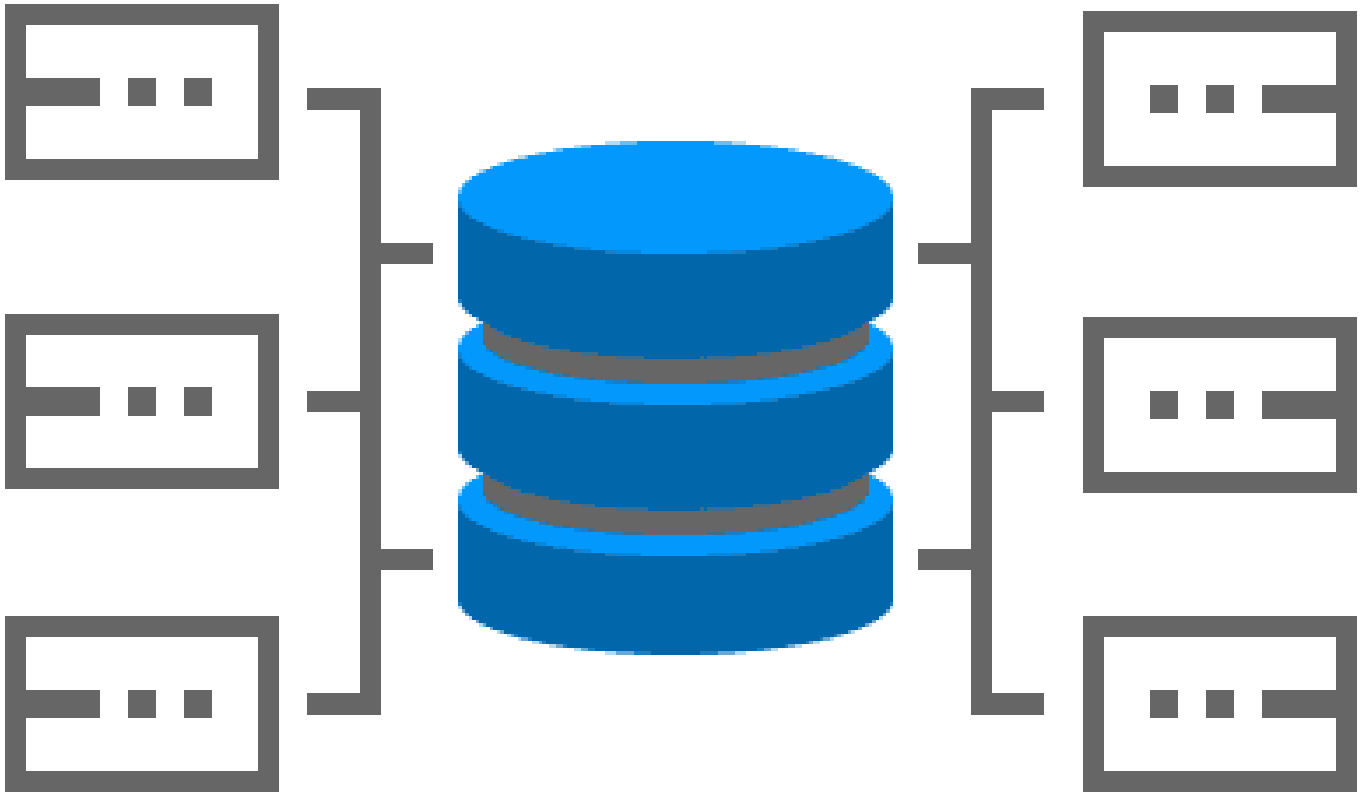
Tuple (or row) - a set of fields that generally represents an "object" like a person or a music track

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)


SI502 – Database
New Open Save Print Import Copy Paste Format Undo Redo AutoSum Sort A–Z Sort Z–A Gallery Toolbox
Sheets Charts SmartArt Graphics WordArt
A B C D
1 2 3 4 5 6 7 8
Columns / Attributes
TITLE RATING LEN
About to Rock 3 354
Who Made Who 4 252
Rows / Tuples
Tables / Relations
Tracks Albums Artists Genres +

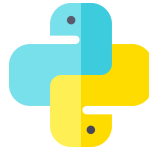

Student.sql
Student.csv
Student.json
Student.db

Demonstration Program

STUDENT.DB + STUDENT.SQL + STUDENT.CSV

SQL Database

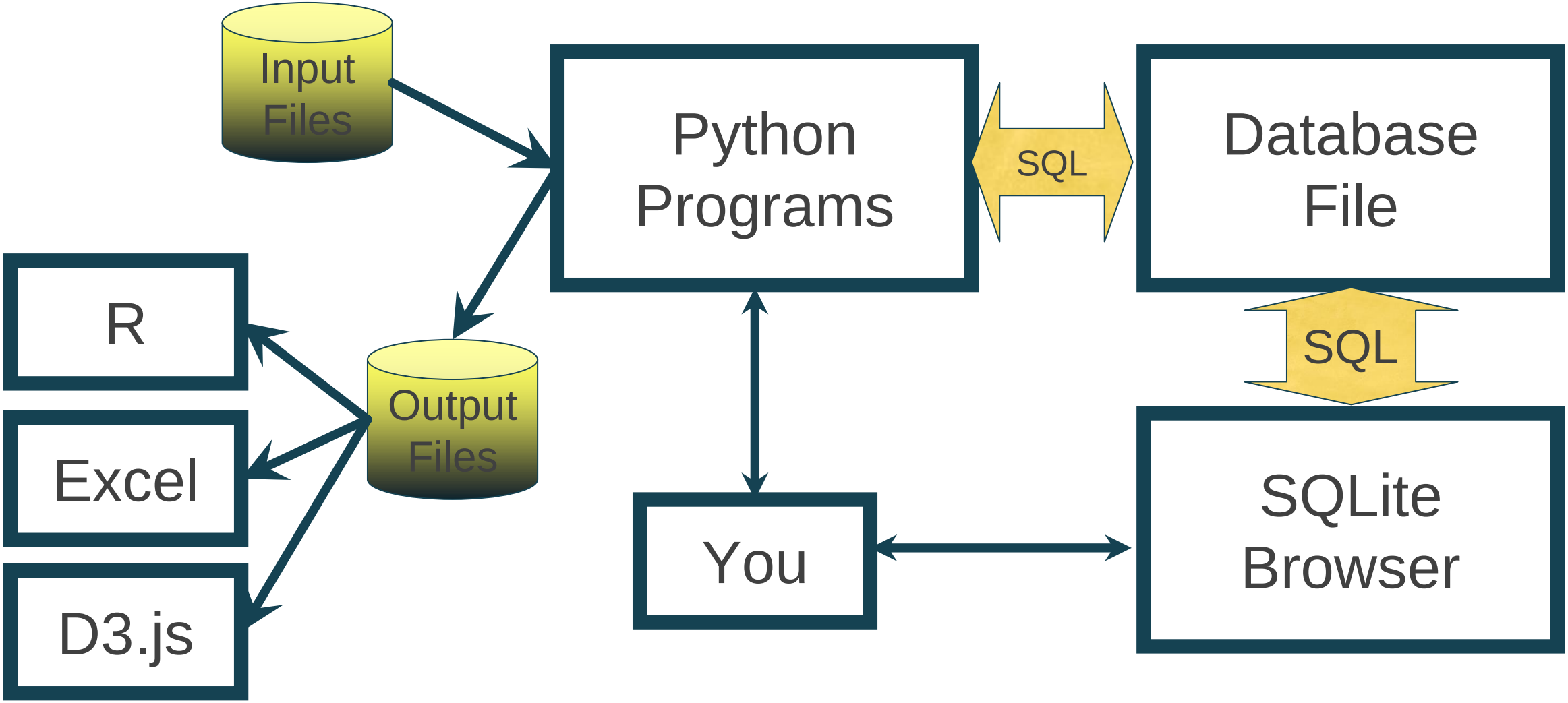
LECTURE 2

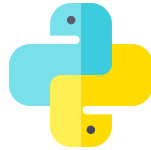


SQL

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

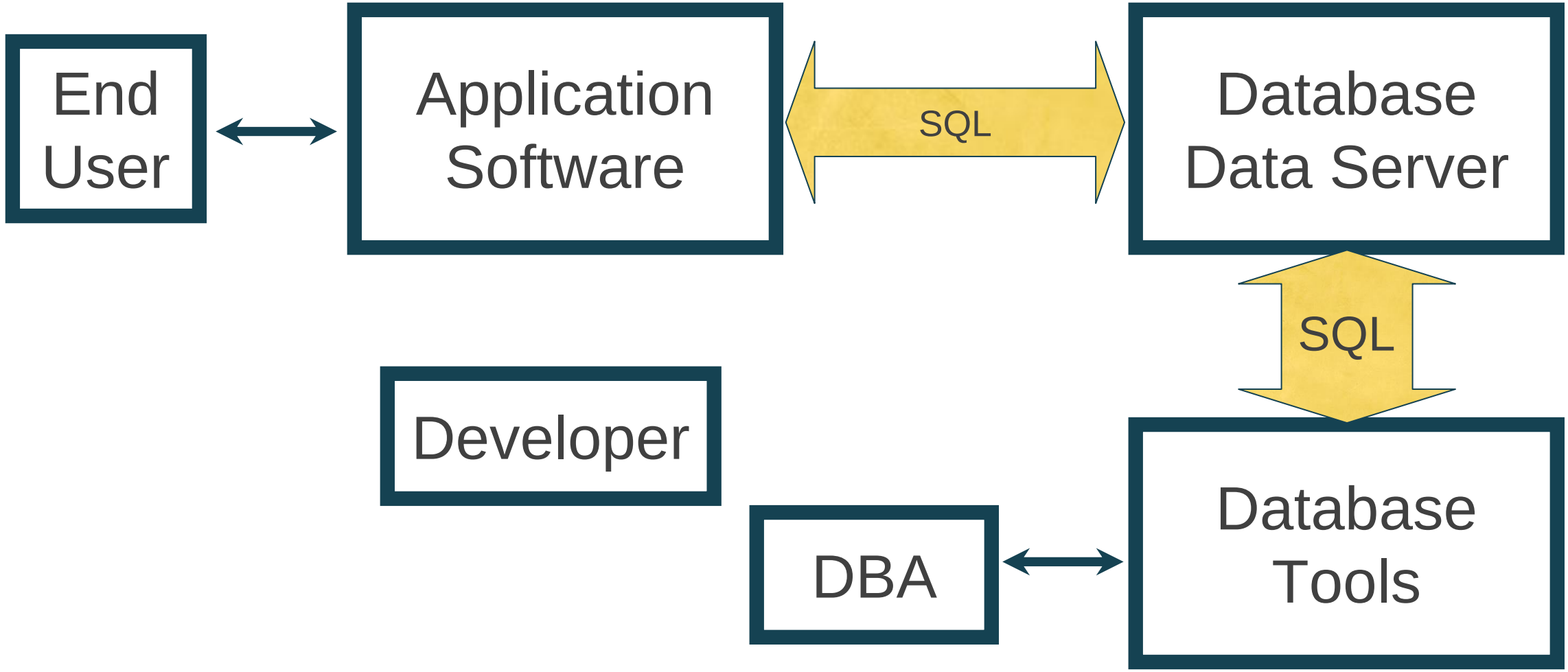
- Create data (a.k.a Insert)
- Retrieve data
- Update data
- Delete data


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



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”

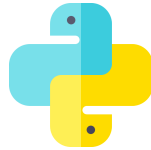

End User
Application Software
SQL
Database Data Server
SQL
Developer
DBA
Database Tools

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.

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.



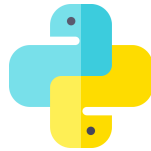
Common Database Systems

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™

McAfee®

skype™

php

Microsoft®

Adobe®

Sun microsystems

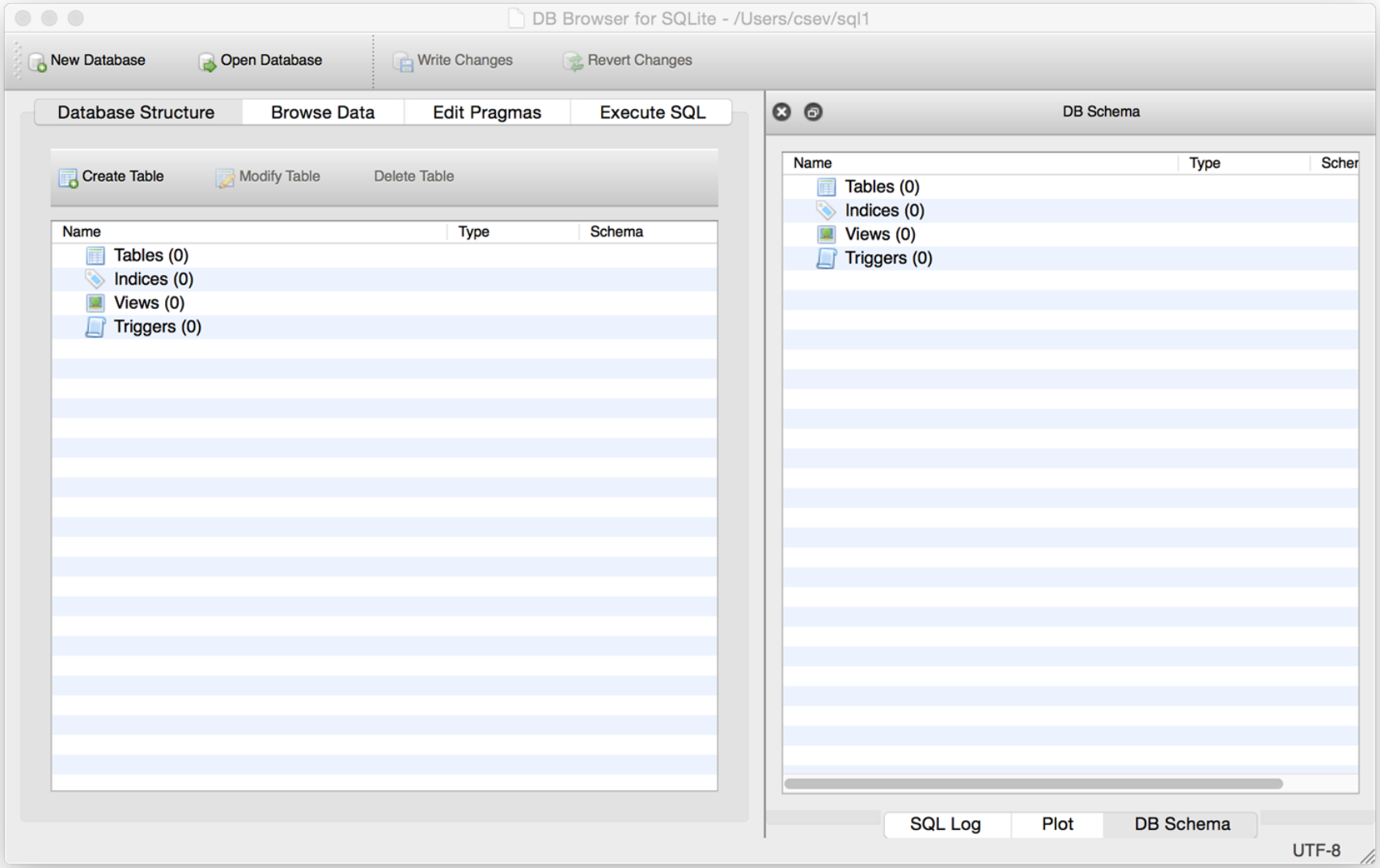
Google™

TOSHIBA

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



http://sqlitebrowser.org/

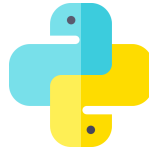
In-Class Demonstration Program

LET'S MAKE A DATABASE

https://www.py4e.com/lectures3/Pythonlearn-15-Database-Handout.txt

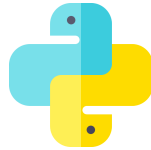
Python Connector

LECTURE 3



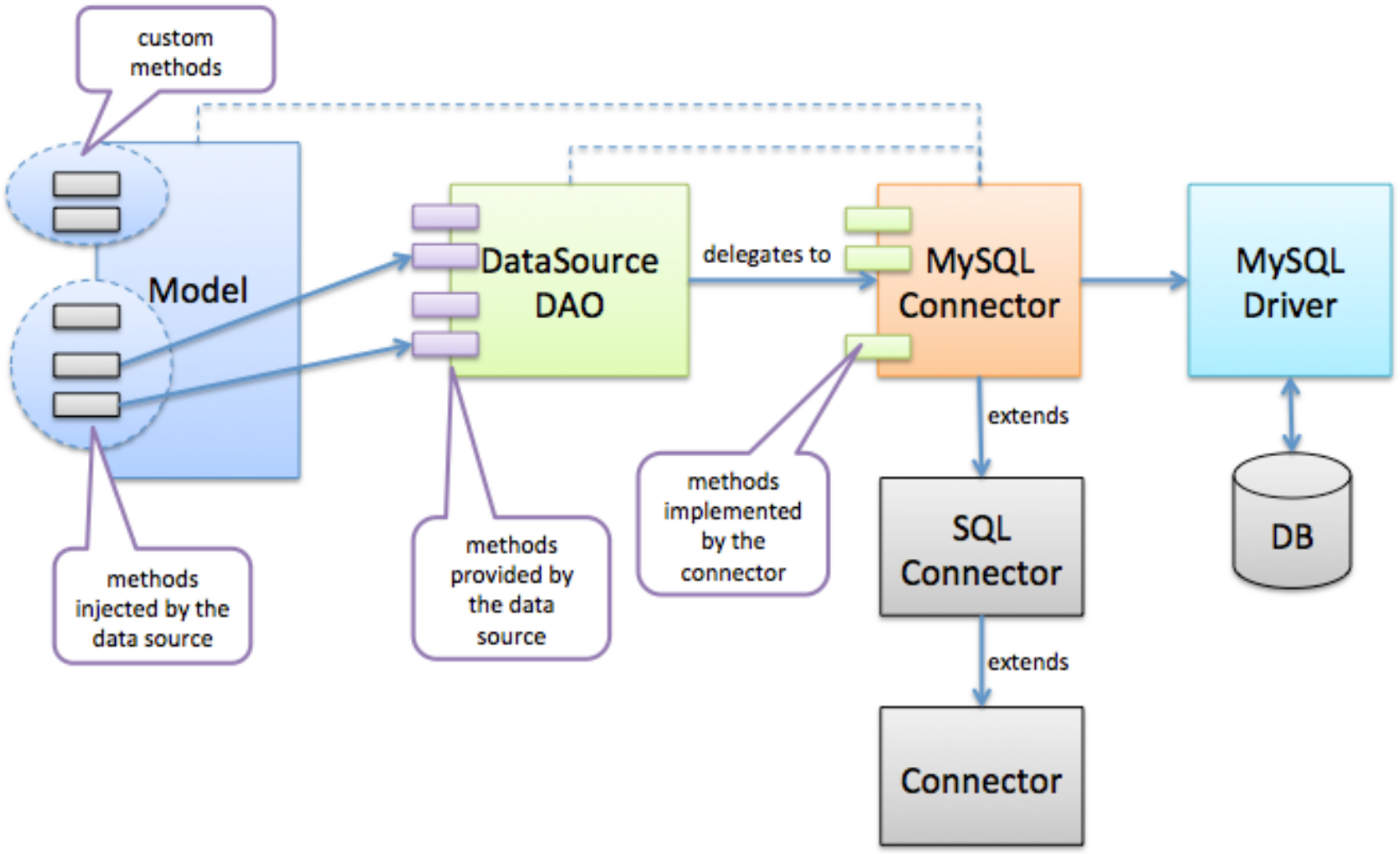
PySQLite

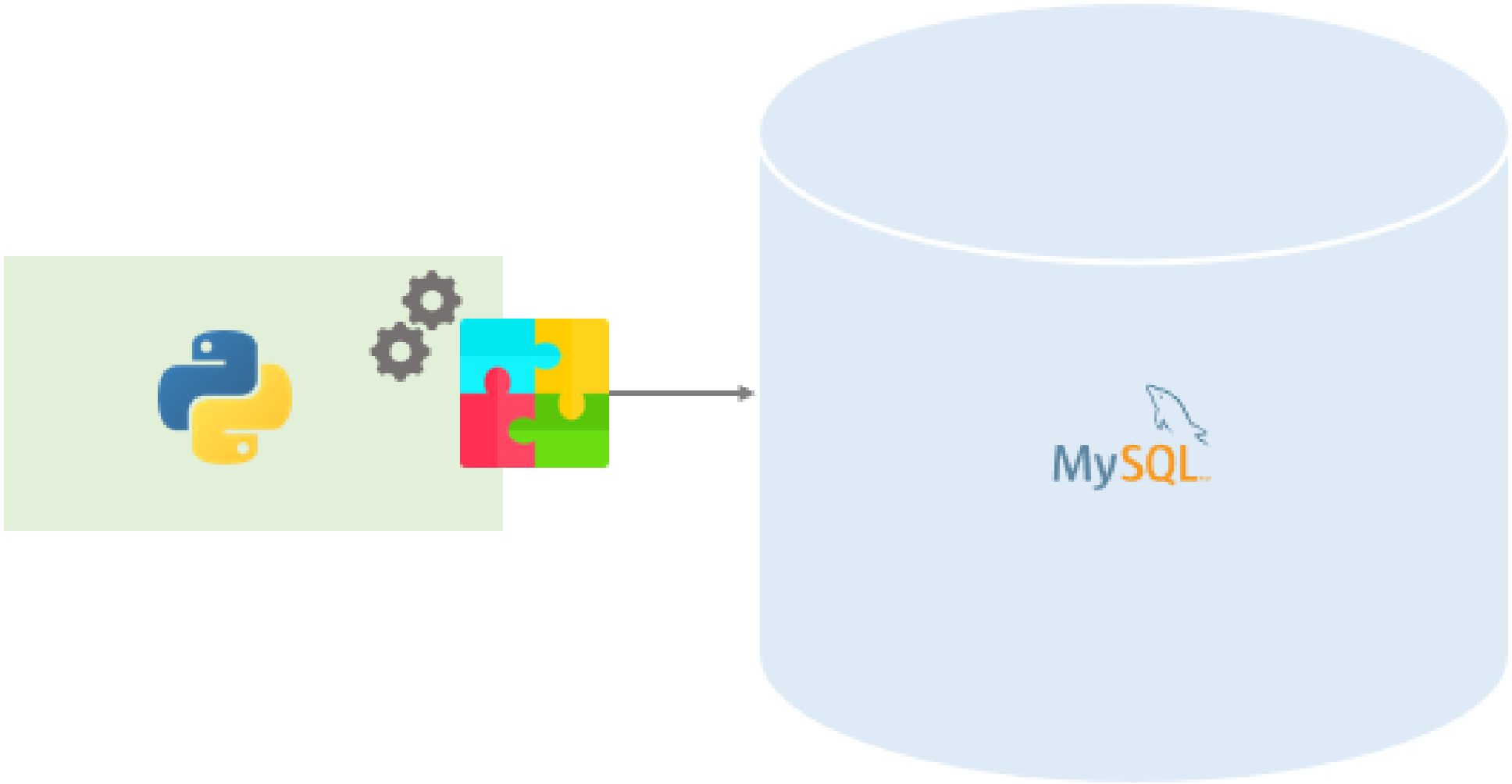
- The PySQLite provides a standardized Python DBI API 2.0 compliant interface to the SQLite database. If your application needs to support not only the SQLite database but also other databases such as MySQL, PostgreSQL, and Oracle, the PySQLite is a good choice.
- PySQLite is a part of the Python Standard library since Python version 2.5



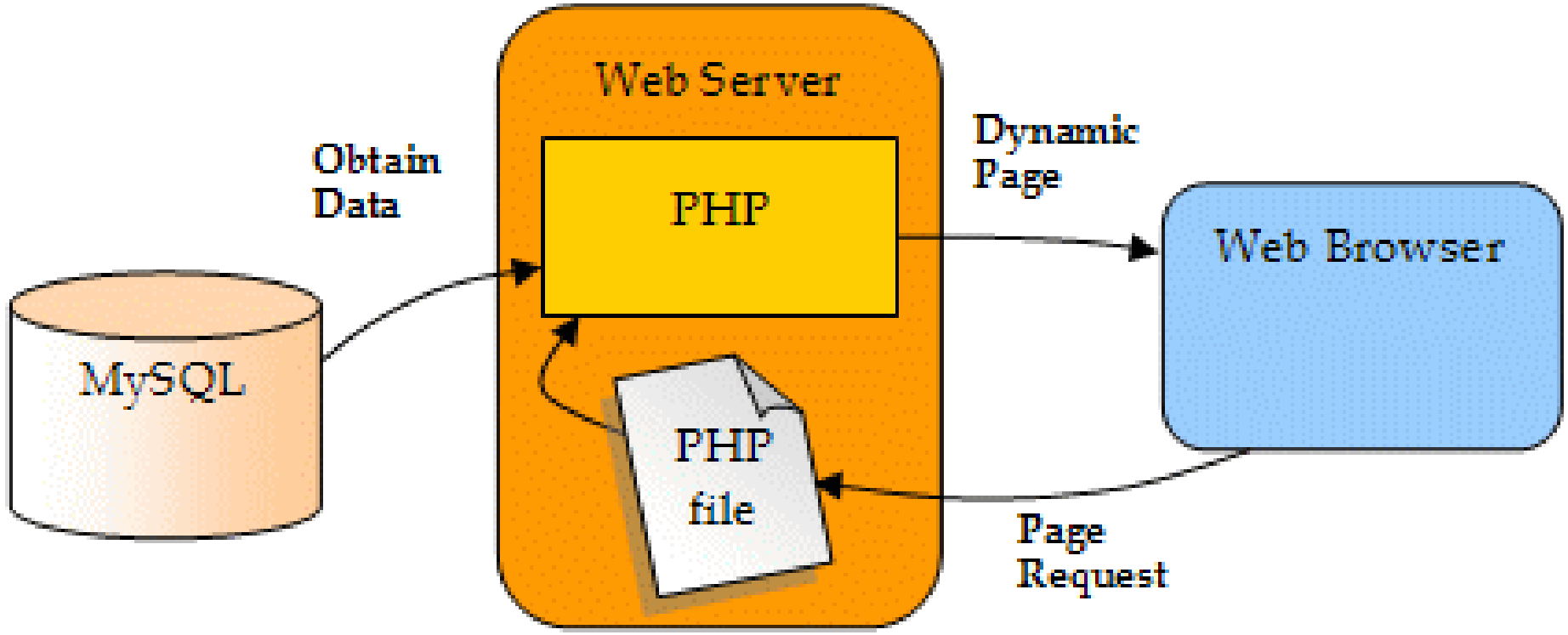
APSW

- If your application needs to support only SQLite database, you should use the APSW module, which is known as Another Python SQLite Wrapper.
- The APSW provides the thinnest layer over the SQLite database library. The APSW is designed to mimic the native SQLite C, therefore, whatever you can do in SQLite C API, you can do it also from Python.
- Besides covering the SQLite library, the APSW provides many low-level features including the ability to create user-defined aggregate, function, and collations from Python. It even allows you to write a virtual table implementation using Python.
- We will use the **PySQLite** wrapper to demonstrate how to work with SQLite database library using Python.


custom methods
Model
DataSource DAO
delegates to
MySQL Connector
MySQL Driver
extends
SQL Connector
extends
Connector
DB
methods injected by the data source
methods provided by the data source
methods implemented by the connector


MySQL

PHP does not need Connector



Creating Database

LECTURE 4

Creating a New Database

- When you connect to an SQLite database file that does not exist, SQLite automatically creates the new database for you.
- To create a database, first, you have to create a Connection object that represents the database using the connect() function of the sqlite3 module.
- For example, the following Python program creates a new database file Student.db in the folder.

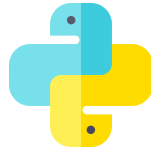
Demonstration Program

STUDENT.PY

Creating Database

Demo Program: Student.py

```
import sqlite3
from sqlite3 import Error
def create_connection(db_file):
    """ create a database connection to a SQLite database """
    conn = None
    try:
        conn = sqlite3.connect(db_file)
        print(sqlite3.version)
    except Error as e:
        print(e)
    finally:
        if conn:
            conn.close()

if __name__ == '__main__':
    create_connection(r"Student.db")
```

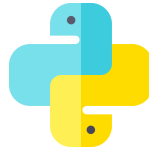


Creating a New Database

- First, we define a function called create_connection() that connects to an SQLite database specified by the database file db_file. Inside the function, we call the connect() function of the sqlite3 module.
- The connect() function opens a connection to an SQLite database. It returns a Connection object that represents the database. By using the Connection object, you can perform various database operations.
- In case an error occurs, we catch it within the try except block and display the error message. If everything is fine, we display the SQLite database version.
- It is a good programming practice that you should always close the database connection when you complete with it.

Name of the Database

- Second, we pass the path of the database file to the create_connection() function to create the database. Note that the prefix r in the r"C:\sqlite\db\dbname.db" instructs Python that we are passing a raw string.
- Let’s run the program and check the c:\sqlite\db folder.



Creating a New Database in Memory

- If you pass the file name as :memory: to the connect() function of the sqlite3 module, it will create a new database that resides in the memory (RAM) instead of a database file on disk.
- The following program creates an SQLite database in the memory.

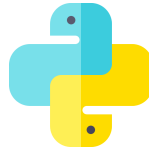
Creating Database

Demo Program: StudentMem.py

```
import sqlite3
from sqlite3 import Error
def create_connection():
    """ create a connection to a database that resides in memory """
    conn = None;
    try:
        conn = sqlite3.connect(':memory:')
        print(sqlite3.version)
    except Error as e:
        print(e)
    finally:
        if conn:
            print("DB in memory Successfully connected. ")

if __name__ == '__main__':
    create_connection()
```

Creating Tables

LECTURE 5



Creating a Table

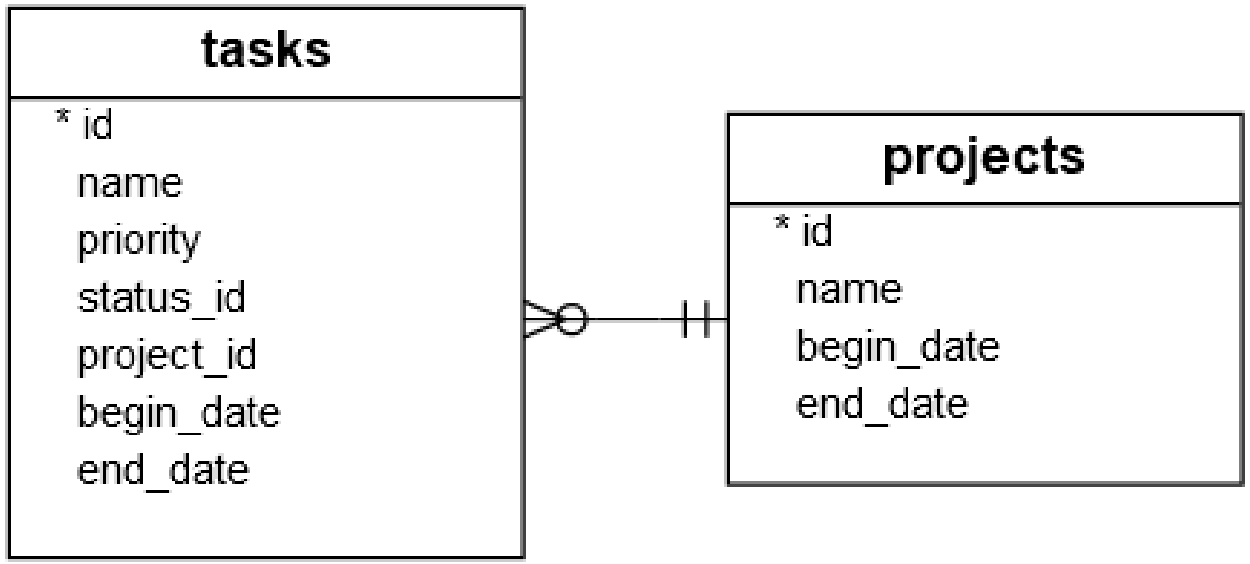
To create a new table in an SQLite database from a Python program, you use the following steps:

1. First, create a Connection object using the connect() function of the sqlite3 module.
2. Second, create a Cursor object by calling the cursor() method of the Connection object.
3. Third, pass the CREATE TABLE statement to the execute() method of the Cursor object and execute this method.

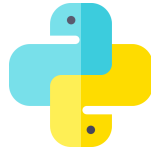
Creating a Table

Demo Program: project.py

- For the demonstration, we will create two tables: projects and tasks as shown in the following database diagram:

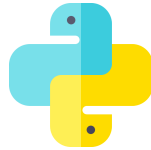


```
-- projects table
CREATE TABLE IF NOT EXISTS projects (
   id integer PRIMARY KEY,
   name text NOT NULL,
   begin_date text,
   end_date text
);
-- tasks table
CREATE TABLE IF NOT EXISTS tasks (
   id integer PRIMARY KEY,
   name text NOT NULL, priority integer,
   project_id integer NOT NULL,
   status_id integer NOT NULL,
   begin_date text NOT NULL,
   end_date text NOT NULL, FOREIGN KEY (project_id) REFERENCES
   projects (id)
);
```



Make Connection

```
def create_connection(db_file):
    """ create a database connection to the SQLite database
        specified by db_file
    :param db_file: database file
    :return: Connection object or None
    """
    conn = None
    try:
        conn = sqlite3.connect(db_file)
        return conn
    except Error as e:
        print(e)

    return conn
```



Feed SQL Code to Create Table

- Second, develop a function named create_table() that accepts a Connection object and an SQL statement. Inside the function, we call the execute() method of the Cursor object to execute the CREATE TABLE statement.

```
def create_table(conn, create_table_sql):
    """ create a table from the create_table_sql statement
    :param conn: Connection object
    :param create_table_sql: a CREATE TABLE statement
    :return:
    """
    try:
        c = conn.cursor()
        c.execute(create_table_sql)
    except Error as e:
        print(e)
```

```
# main program
def main():
    database = r"C:\sqlite\db\pythonsqlite.db"

    sql_create_projects_table = """ CREATE TABLE IF NOT EXISTS projects (
                                        id integer PRIMARY KEY,
                                        name text NOT NULL,
                                        begin_date text,
                                        end_date text
                                    ); """

    sql_create_tasks_table = """CREATE TABLE IF NOT EXISTS tasks (
                                    id integer PRIMARY KEY,
                                    name text NOT NULL,
                                    priority integer,
                                    status_id integer NOT NULL,
                                    project_id integer NOT NULL,
                                    begin_date text NOT NULL,
                                    end_date text NOT NULL,
                                    FOREIGN KEY (project_id) REFERENCES projects (id)
                                );"""
```

```
# create a database connection
conn = create_connection(database)

# create tables
if conn is not None:
    # create projects table
    create_table(conn, sql_create_projects_table)

    # create tasks table
    create_table(conn, sql_create_tasks_table)
else:
    print("Error! cannot create the database connection.")
```

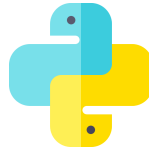
Inserting Data

LECTURE 6

Inserting Data

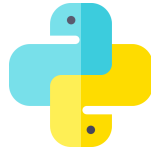
To insert rows into a table in SQLite database, you use the following steps:

1. First, connect to the SQLite database by creating a Connection object.
2. Second, create a Cursor object by calling the cursor method of the Connection object.
3. Third, execute an INSERT statement. If you want to pass arguments to the INSERT statement, you use the question mark (?) as the placeholder for each argument.



Make Connection

```
def create_connection(db_file):
    """ create a database connection to the SQLite database
        specified by db_file
    :param db_file: database file
    :return: Connection object or None
    """
    conn = None
    try:
        conn = sqlite3.connect(db_file)
    except Error as e:
        print(e)

    return conn
```

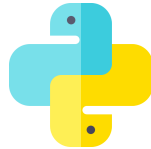


Inserting Data

Assumption: project.db exists

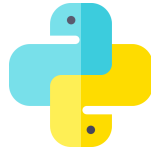
```
def create_project(conn, project):
    """
    Create a new project into the projects table
    :param conn:
    :param project:
    :return: project id
    """
    sql = ''' INSERT INTO projects(name,begin_date,end_date)
              VALUES(?,?,?) '''
    cur = conn.cursor()
    cur.execute(sql, project)
    conn.commit()
    return cur.lastrowid
```

In this function, we used the lastrowid attribute of the Cursor object to get back the generated id.



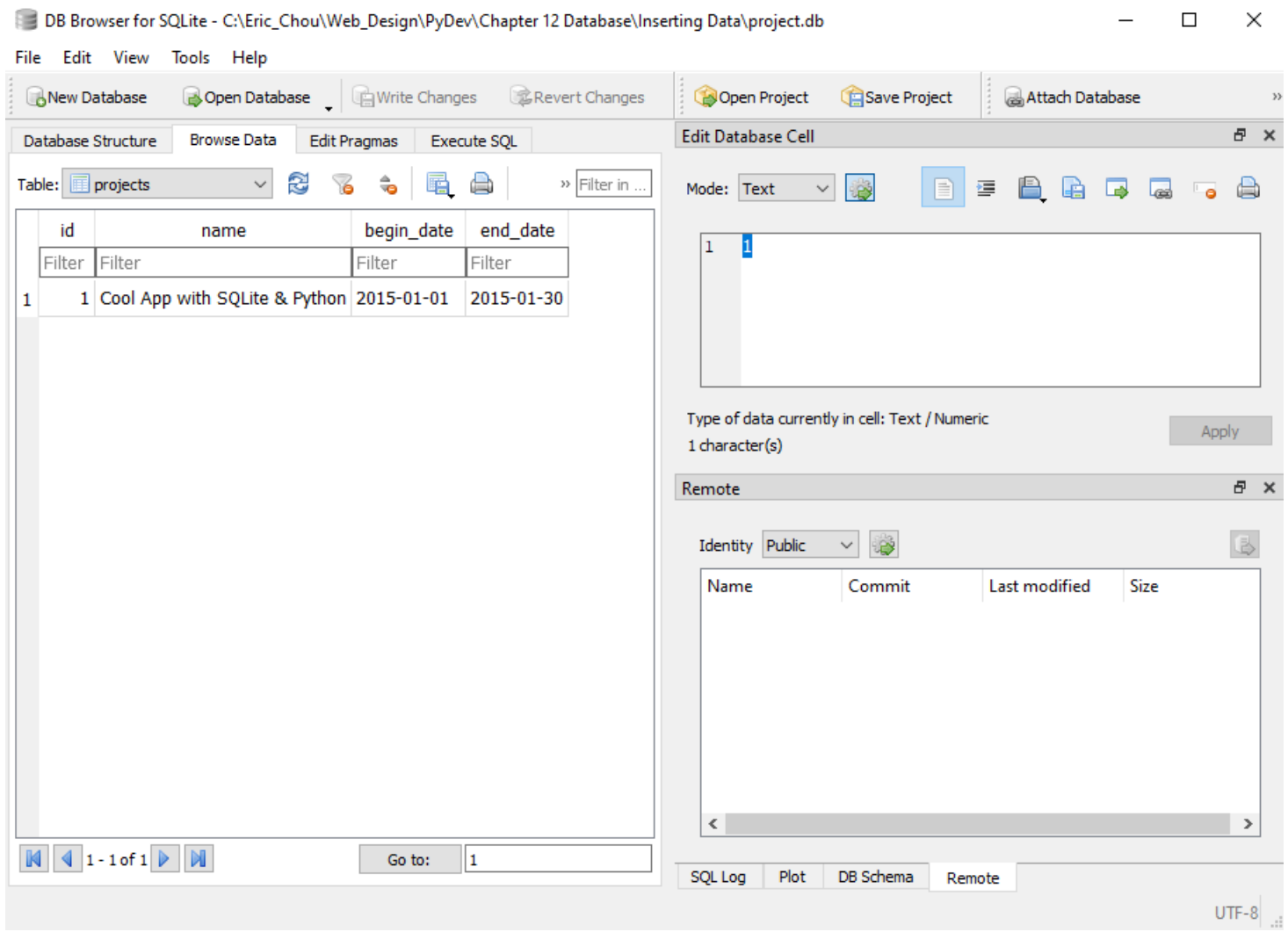
Creating Task

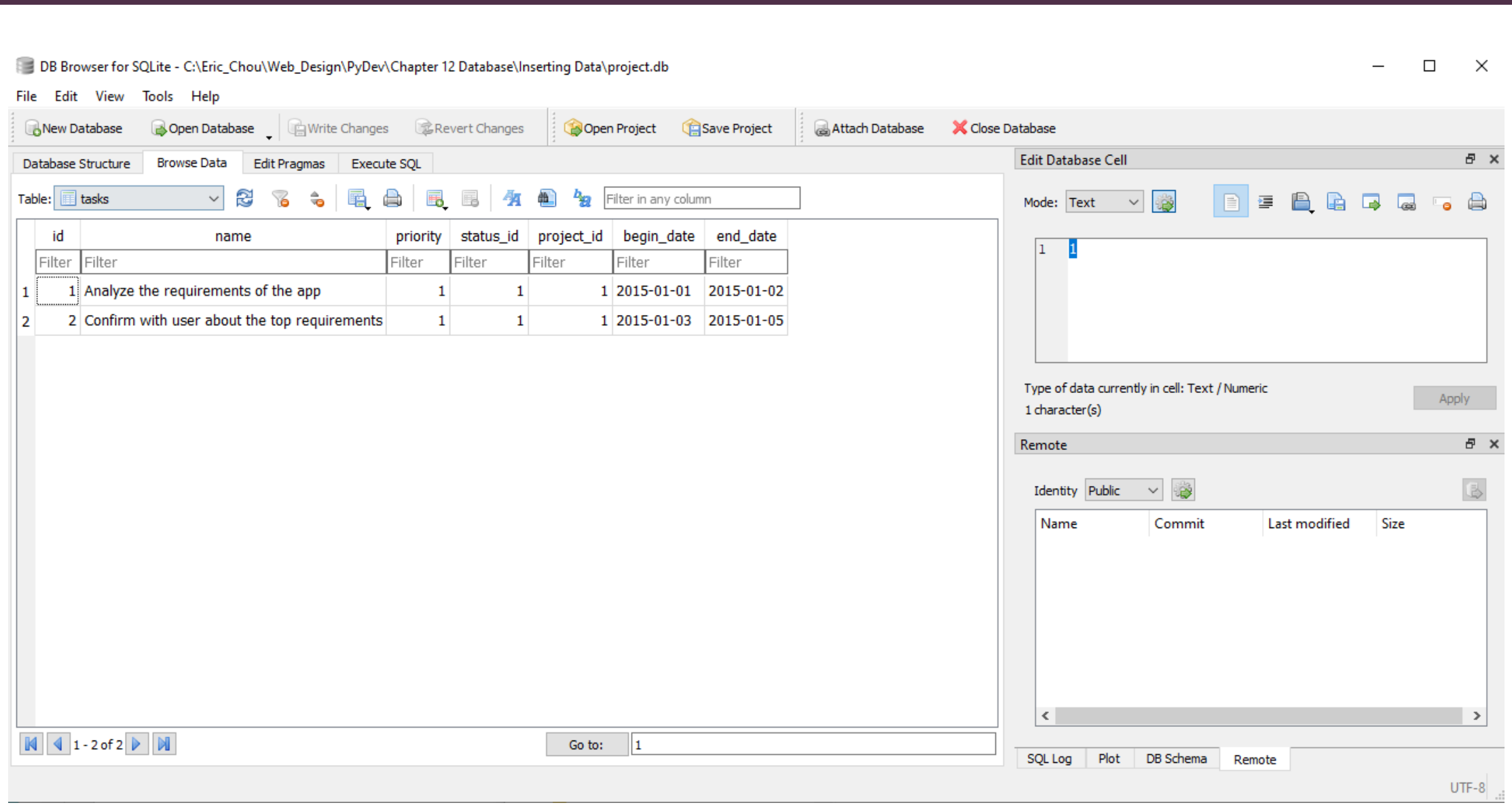
```
def create_task(conn, task):
    """
    Create a new task
    :param conn:
    :param task:
    :return:
    """
    sql = ''' INSERT INTO tasks(name,priority,status_id,project_id,begin_date,end_date)
              VALUES(?,?,?,?,?,?) '''
    cur = conn.cursor()
    cur.execute(sql, task)
    conn.commit()
    return cur.lastrowid
```



Main function

```
def main():
    database = r"project.db"

    # create a database connection
    conn = create_connection(database)
    with conn:
        # create a new project
        project = ('Cool App with SQLite & Python', '2015-01-01', '2015-01-30');
        project_id = create_project(conn, project)

        # tasks
        task_1 = ('Analyze the requirements of the app', 1, 1, project_id, '2015-01-01', '2015-01-02')
        task_2 = ('Confirm with user about the top requirements', 1, 1, project_id, '2015-01-03', '2015-01-05')

        # create tasks
        create_task(conn, task_1)
        create_task(conn, task_2)
```


DB Browser for SQLite - C:\Eric_Chou\Web_Design\PyDev\Chapter 12 Database\Inserting Data\project.db
File
Edit
View
Tools
Help
New Database
Open Database
Write Changes
Revert Changes
Open Project
Save Project
Attach Database
Database Structure
Browse Data
Edit Pragmas
Execute SQL
Table: projects
Filter in ...
id
name
begin_date
end_date
Filter
Filter
Filter
Filter
1
1
Cool App with SQLite & Python
2015-01-01
2015-01-30
1 - 1 of 1
Go to:
1
Edit Database Cell
Mode: Text
1
1
Type of data currently in cell: Text / Numeric
1 character(s)
Apply
Remote
Identity
Public
Name
Commit
Last modified
Size
SQL Log
Plot
DB Schema
Remote
UTF-8


DB Browser for SQLite - C:\Eric_Chou\Web_Design\PyDev\Chapter 12 Database\Inserting Data\project.db
File
Edit
View
Tools
Help
New Database
Open Database
Write Changes
Revert Changes
Open Project
Save Project
Attach Database
Close Database
Database Structure
Browse Data
Edit Pragmas
Execute SQL
Table:
tasks
Filter in any column
id
name
priority
status_id
project_id
begin_date
end_date
1
Analyze the requirements of the app
1
1
1
2015-01-01
2015-01-02
2
Confirm with user about the top requirements
1
1
1
2015-01-03
2015-01-05
1 - 2 of 2
Go to:
1
Edit Database Cell
Mode:
Text
Type of data currently in cell: Text / Numeric
1 character(s)
Apply
Remote
Identity
Public
Name
Commit
Last modified
Size
SQL Log
Plot
DB Schema
Remote
UTF-8

Updating Data

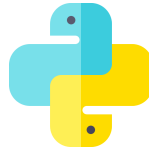
LECTURE 7

Updating Data

To update data in a table from a Python program, you follow these steps:

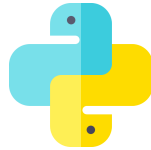
1. First, create a database connection to the SQLite database using the connect() function. Once the database connection created, you can access the database using the Connection object.
2. Second, create a Cursor object by calling the cursor() method of the Connection object.
3. Third, execute the UPDATE statement by calling the execute() method of the Cursor object.

In this example we will update the priority, begin date, and end date of a specific task in the tasks table.



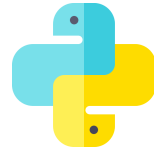
Make Connection

```
def create_connection(db_file):
    """ create a database connection to the SQLite database
        specified by the db_file
    :param db_file: database file
    :return: Connection object or None
    """
    conn = None
    try:
        conn = sqlite3.connect(db_file)
    except Error as e:
        print(e)

    return conn
```



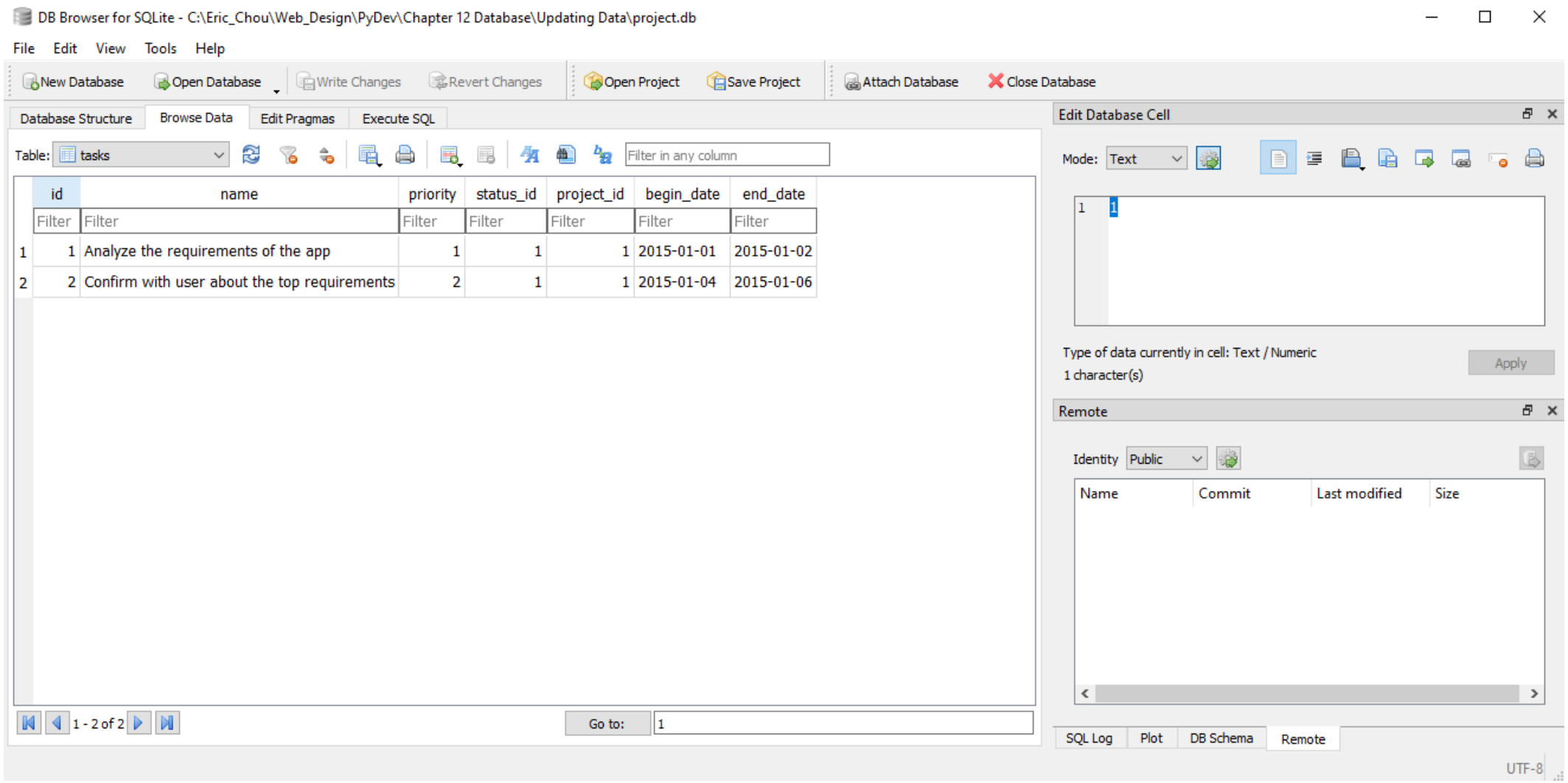
Update Task

```
def update_task(conn, task):
    """
    update priority, begin_date, and end date of a task
    :param conn:
    :param task:
    :return: project id
    """
    sql = ''' UPDATE tasks
              SET priority = ? ,
                  begin_date = ? ,
                  end_date = ?
              WHERE id = ?'''
    cur = conn.cursor()
    cur.execute(sql, task)
    conn.commit()
```



Main Function

```
def main():
    database = r"project.db"

    # create a database connection
    conn = create_connection(database)
    with conn:
        update_task(conn, (2, '2015-01-04', '2015-01-06', 2))
```


DB Browser for SQLite - C:\Eric_Chou\Web_Design\PyDev\Chapter 12 Database\Updating Data\project.db
File Edit View Tools Help
New Database
Open Database
Write Changes
Revert Changes
Open Project
Save Project
Attach Database
Close Database
Database Structure
Browse Data
Edit Pragmas
Execute SQL
Table: tasks
Filter in any column
id name priority status_id project_id begin_date end_date
1 1 Analyze the requirements of the app 1 1 1 2015-01-01 2015-01-02
2 2 Confirm with user about the top requirements 2 1 1 2015-01-04 2015-01-06
1 - 2 of 2
Go to: 1
Edit Database Cell
Mode: Text
Type of data currently in cell: Text / Numeric
1 character(s)
Apply
Remote
Identity Public
Name Commit Last modified Size
SQL Log Plot DB Schema Remote
UTF-8

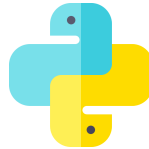
Query

LECTURE 8

Query

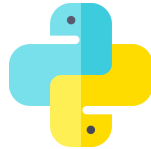
To query data in an SQLite database from Python, you use these steps:

1. First, establish a connection to the SQLite database by creating a Connection object.
2. Next, create a Cursor object using the cursor method of the Connection object.
3. Then, execute a SELECT statement.
4. After that, call the fetchall() method of the cursor object to fetch the data.
5. Finally, loop the cursor and process each row individually.



Make Connection

```
def create_connection(db_file):
    """ create a database connection to the SQLite database
        specified by the db_file
    :param db_file: database file
    :return: Connection object or None
    """
    conn = None
    try:
        conn = sqlite3.connect(db_file)
    except Error as e:
        print(e)

    return conn
```

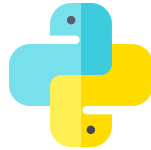


Select All Tasks

```
def select_all_tasks(conn):
    """
    Query all rows in the tasks table
    :param conn: the Connection object
    :return:
    """
    cur = conn.cursor()
    cur.execute("SELECT * FROM tasks")

    rows = cur.fetchall()

    for row in rows:
        print(row)
```

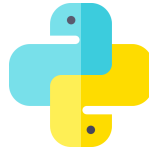
Select Tasks with Priority

- In the select_task_by_priority() function, we selected the tasks based on a particular priority. The question mark (?) in the query is the placeholder.
- When the cursor executed the SELECT statement, it substituted the question mark (?) by the priority argument. The fetchall() method fetched all matching tasks by the priority.

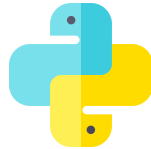


Select Task with Certain Priority

```
def select_task_by_priority(conn, priority):
    """
    Query tasks by priority
    :param conn: the Connection object
    :param priority:
    :return:
    """
    cur = conn.cursor()
    cur.execute("SELECT * FROM tasks WHERE priority=?", (priority,))

    rows = cur.fetchall()

    for row in rows:
        print(row)
```



```python
def main():
    database = r"project.db"

    # create a database connection
    conn = create_connection(database)
    with conn:
        print("1. Query task by priority:")
        select_task_by_priority(conn, 1)

        print("2. Query all tasks")
        select_all_tasks(conn)
```



Query Results

```
1. Query task by priority:
(1, 'Analyze the requirements of the app', 1, 1, 1, '2015-01-01', '2015-01-02')

2. Query all tasks
(1, 'Analyze the requirements of the app', 1, 1, 1, '2015-01-01', '2015-01-02')

(2, 'Confirm with user about the top requirements', 2, 1, 1, '2015-01-04', '2015-01-
06')
```

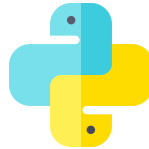
Deleting Data

LECTURE 9

Deleting Data

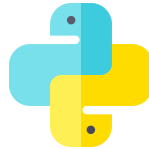
In order to delete data in the SQLite database from a Python program, you use the following steps:

1. First, establish a connection the SQLite database by creating a `Connection` object using the `connect()` function.
2. Second, to execute a `DELETE` statement, you need to create a `Cursor` object using the `cursor()` method of the `Connection` object.
3. Third, execute the `DELETE` statement using the `execute()` method of the `Cursor` object. In case you want to pass the arguments to the statement, you use a question mark (`?`) for each argument.



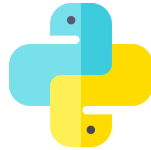
Make Connection

```
def create_connection(db_file):
    """ create a database connection to the SQLite database
        specified by the db_file
    :param db_file: database file
    :return: Connection object or None
    """
    conn = None
    try:
        conn = sqlite3.connect(db_file)
    except Error as e:
        print(e)

    return conn
```



Delete a Task by ID

```
def delete_task(conn, id):
    """
    Delete a task by task id
    :param conn:  Connection to the SQLite database
    :param id: id of the task
    :return:
    """
    sql = 'DELETE FROM tasks WHERE id=?'
    cur = conn.cursor()
    cur.execute(sql, (id,))
    conn.commit()
```

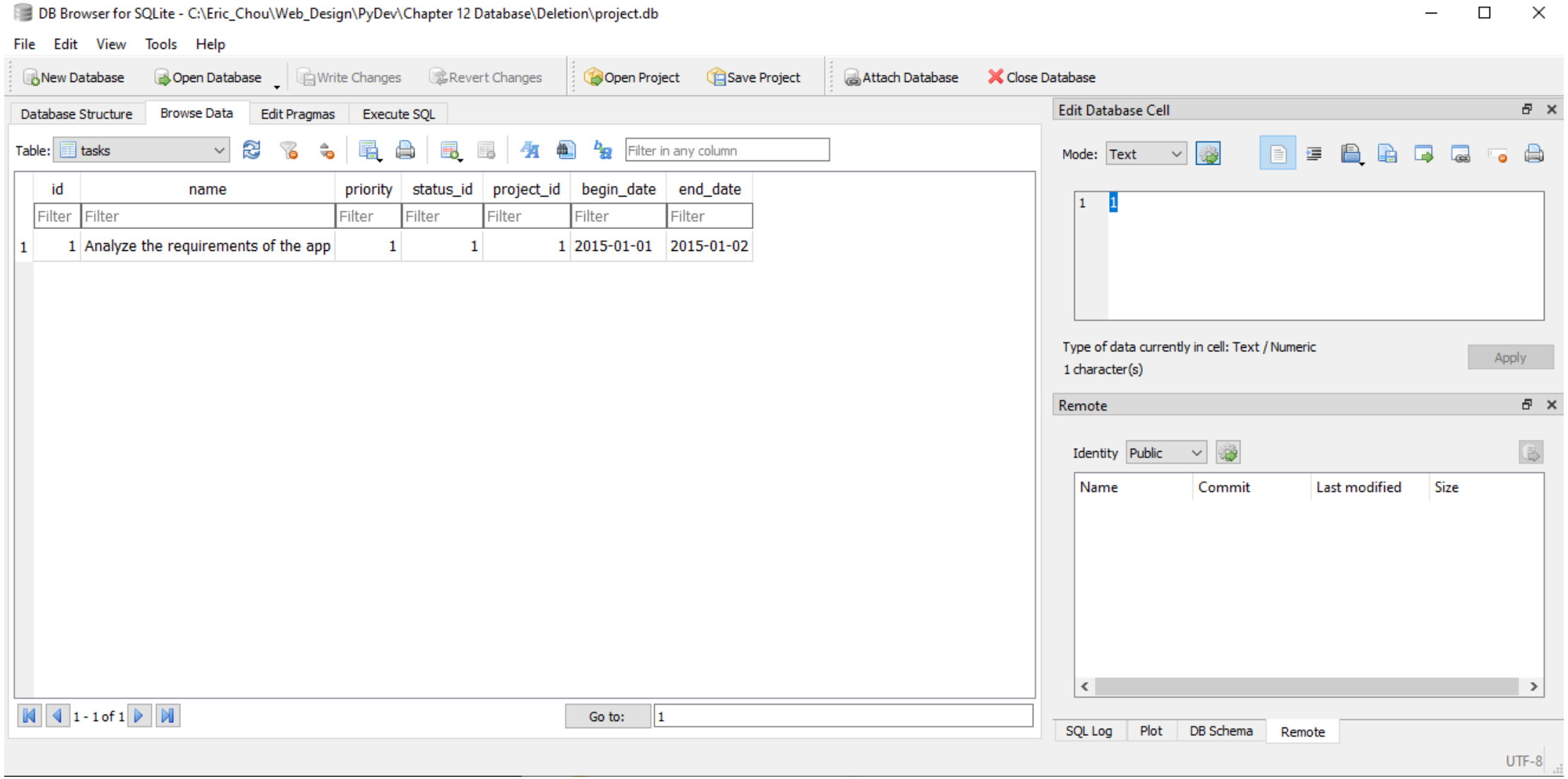


Delete All Tasks

```
def delete_all_tasks(conn):
    """
    Delete all rows in the tasks table
    :param conn: Connection to the SQLite database
    :return:
    """
    sql = 'DELETE FROM tasks'
    cur = conn.cursor()
    cur.execute(sql)
    conn.commit()
```

Main function

```
def main():
    database = r"C:\sqlite\db\pythonsqlite.db"

    # create a database connection
    conn = create_connection(database)
    with conn:
        delete_task(conn, 2);
        # delete_all_tasks(conn);
```


DB Browser for SQLite - C:\Eric_Chou\Web_Design\PyDev\Chapter 12 Database\Deletion\project.db
File Edit View Tools Help
New Database
Open Database
Write Changes
Revert Changes
Open Project
Save Project
Attach Database
Close Database
Database Structure
Browse Data
Edit Pragmas
Execute SQL
Table: tasks
Filter in any column
id name priority status_id project_id begin_date end_date
Filter Filter Filter Filter Filter Filter Filter
1 1 Analyze the requirements of the app 1 1 1 2015-01-01 2015-01-02
1 - 1 of 1
Go to: 1
Edit Database Cell
Mode: Text
Type of data currently in cell: Text / Numeric
1 character(s)
Apply
Remote
Identity Public
Name Commit Last modified Size
SQL Log Plot DB Schema Remote
UTF-8