SQL

...

CRUD Operations tutorial
Colab [Link]
Credit: TA.Theerapat

Outline

1. Install Library
2. Connect to Server and Create Database
3. Create tables
4. Insert records
5. Read reocords
6. Update records
7. Delete records

1. Import libraries

```
import sqlite3
```

Sqlite3 allows us to work with SQL <u>without</u> any installments.

2.1 Connect to Database ‘School’

```python
def create_db_connection(db_name):
    connection = None
    try:
        connection = sqlite3.connect(f"{db_name}.db")  # Connect or create the database file
        print(f"Connection to SQLite DB '{db_name}' successful.")
    except sqlite3.Error as e:
        print(f"The error '{e}' occurred.")
    return connection
```

```python
db_name = "school"
connection = create_db_connection(db_name)
```

2.2 Query Execution Function

```python
def execute_query(connection, query):
    cursor = connection.cursor()
    try:
        cursor.execute(query)
        connection.commit()
        print("Query successful")
    except Error as err:
        print(f"Error: '{err}'")
```

cursor is the interface between your Python program and the database, used to execute SQL commands and manage query results.

3.1 Create Table

```
create_teacher_table = """
CREATE TABLE IF NOT EXISTS teacher (
  teacher_id INTEGER PRIMARY KEY,
  first_name TEXT NOT NULL,
  last_name TEXT NOT NULL,
  language_1 TEXT NOT NULL,
  language_2 TEXT,
  dob DATE,
  tax_id INTEGER UNIQUE,
  phone_no TEXT
);
"""

execute_query(connection, create_teacher_table)
```

Create a teacher with **teacher_id** as the primary key and other fields.

4.1 Insert Into Table

```
teachers = """
INSERT INTO teacher VALUES
(1,  'James', 'Smith', 'ENG', NULL, '1985-04-20', 12345, '+491774553676'),
(2, 'Stefanie',  'Martin',  'FRA', NULL,  '1970-02-17', 23456, '+491234567890'),
(3, 'Steve', 'Wang',  'MAN', 'ENG', '1990-11-12', 34567, '+447840921333'),
(4, 'Friederike',  'Müller-Rossi', 'DEU', 'ITA', '1987-07-07',  45678, '+492345678901'),
(5, 'Isobel', 'Ivanova', 'RUS', 'ENG', '1963-05-30',  56789, '+491772635467'),
(6, 'Niamh', 'Murphy', 'ENG', 'IRI', '1995-09-08',  67890, '+491231231232');
"""

execute_query(connection, teachers)
```

Insert rows into the table, specifying the **primary key** and other fields, respectively.

primary key

5.1 Data Reading Function

```python
def read_query(connection, query):
    cursor = connection.cursor()
    result = None
    try:
        cursor.execute(query)
        result = cursor.fetchall()
        return result
    except Error as err:
        print(f"Error: '{err}'")
```

Read the result from the query execution.

5.2 Read Data from the Database

```
q1 = """
SELECT *
FROM teacher;
"""
results = read_query(connection, q1)

for result in results:
    print(result)
```

SELECT * retrieves all columns that exist in the table."

```
(1, 'James', 'Smith', 'ENG', None, datetime.date(1985, 4, 20), 12345, '+491774553676')
(2, 'Stefanie', 'Martin', 'FRA', None, datetime.date(1970, 2, 17), 23456, '+491234567890')
(3, 'Steve', 'Wang', 'MAN', 'ENG', datetime.date(1990, 11, 12), 34567, '+447840921333')
(4, 'Friederike', 'Müller-Rossi', 'DEU', 'ITA', datetime.date(1987, 7, 7), 45678, '+492345678901')
(5, 'Isobel', 'Ivanova', 'RUS', 'ENG', datetime.date(1963, 5, 30), 56789, '+491772635467')
(6, 'Niamh', 'Murphy', 'ENG', 'IRI', datetime.date(1995, 9, 8), 67890, '+491231231232')
```

5.2 Conditional Queries

selected columns

```
SELECT course.course_id, course.course_name, course.language, teacher.first_name, teacher.last_name
FROM course
JOIN teacher
ON course.teacher = teacher.teacher_id
WHERE course.in_school = FALSE;

(13, 'Beginner English', 'ENG', 'Niamh', 'Murphy')
(14, 'Intermediate English', 'ENG', 'Niamh', 'Murphy')
(15, 'Advanced English', 'ENG', 'Niamh', 'Murphy')
(17, 'Français intermédiaire', 'FRA', 'Stefanie', 'Martin')
(19, 'Intermediate English', 'ENG', 'James', 'Smith')
(20, 'Fortgeschrittenes Russisch', 'RUS', 'Isobel', 'Ivanova')
```

5.3 Read Result into DataFrame

```
from_db = []

results = read_query(connection, q5)
for result in results:
    result = list(result)
    from_db.append(result)


columns = ["course_id", "course_name", "language",
           "teacher_first_name", "teacher_last_name"]
df = pd.DataFrame(from_db, columns=columns)

display(df)
```

✓ 0.1s

	course_id	course_name	language	teacher_first_name	teacher_last_name
0	13	Beginner English	ENG	Niamh	Murphy
1	14	Intermediate English	ENG	Niamh	Murphy
2	15	Advanced English	ENG	Niamh	Murphy
3	17	Français intermédiaire	FRA	Stefanie	Martin
4	19	Intermediate English	ENG	James	Smith
5	20	Fortgeschrittenes Russisch	RUS	Isobel	Ivanova

6.1 Updating Record

```
update = """
UPDATE teacher
SET first_name = 'Jim', last_name = 'Halpert'
WHERE teacher_id = 1;
"""

execute_query(connection, update)
```

then check updated result

```
q1 = """
SELECT *
FROM teacher
WHERE teacher_id = 1;
"""

results = read_query(connection, q1)

for result in results:
    print(result)
```

```
(1, 'Jim', 'Halpert', 'ENG', None, datetime.date(1985, 4, 20), 12345, '+491774553676')
```

7.1 Delete a Record

```
delete_course = """
DELETE FROM course WHERE course_id = 20;
"""

connection = create_db_connection("localhost", "root", pw, db)
execute_query(connection, delete_course)
```

SQL AWS Service: Amazon RDS [link]

aws

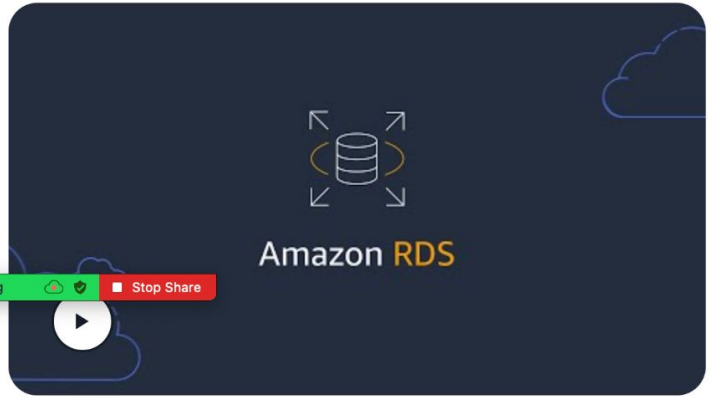
About AWS | Contact Us | Support | English | My Account | Sign In | Create an AWS Account

Amazon Q | Products | Solutions | Pricing | Documentation | Learn | Partner Network | AWS Marketplace | Customer Enablement | Events | Explore More

Amazon RDS | Overview | Features | DB Engines | Pricing | Resources | Solutions | FAQs | Customers | Partners | Migrate with AWS DMS

Why Amazon RDS?

Amazon Relational Database Service (Amazon RDS) is an easy-to-manage relational database service optimized for total cost of ownership. It is simple to set up, operate, and scale with demand. Amazon RDS automates undifferentiated database management tasks, such as provisioning, configuring, backing up, and patching. Amazon RDS allows customers to create a new database in minutes and offers flexibility to customize databases to me needs across eight engines and two deployment options. Customers can optimize performance with features like Multi-AZ with two readable standbys, optimized writes and reads, and AWS Graviton3-based instances, and they can choose from multiple pricing options to effectively manage costs.



You are screen sharing | Stop Share