

SQL Programming

UBCO Master of Data Science – DATA 540



Objectives

- Explain the common steps in querying a database using a programming language.
- Draw and explain the database-program architecture and the key components.
- Write a simple program to query a database in both Python and R.
- Use a variety of databases including MySQL, Microsoft SQL Server, and SQLite.
- Explain the purpose of ODBC.
- List the components of a database connection URL.
- Use dplyr to query a database.
- Be able to debug and resolve database connection and usage issues in Python and R code.



SQL Programming Overview

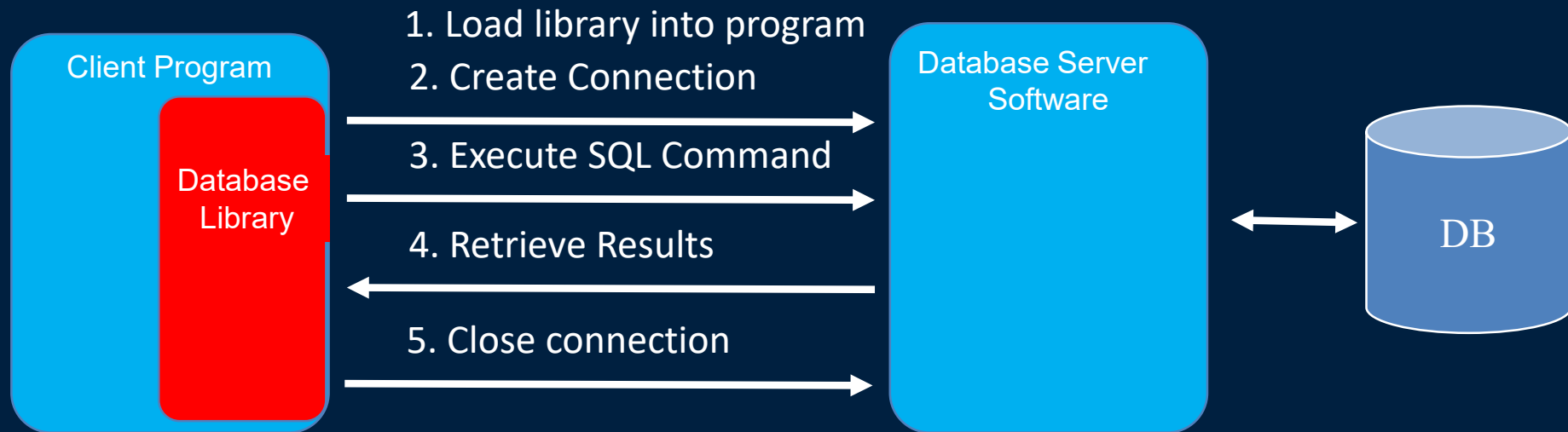
Interaction with a database is often through programs. Programming with a database requires:

- A database server and its connection information
- A programming language for writing the code to query the database
- A library or driver for connecting to the particular database system

General process for programming with a database:

- 1) Load the database access library
- 2) Create a connection to the database
- 3) Execute a SQL command
- 4) Retrieve database results produced by the command
- 5) Close the database connection

SQL Programming Architecture



1) Load Library Error:

- Library not found in path
- Wrong library

2) Create Connection Errors:

- Invalid server URL
- Incorrect user/password
- Network issues
- Wrong library

3) Execute SQL Errors:

- Incorrect SQL
- Wrong database/table
- Improper library use

4) Retrieve Results Errors:

- Wrong column name
- Wrong column index
- Off-by-one
- Improper library use

5) Close Connection:

- FORGET TO DO IT!

SQL Programming Architecture Question

Question: True or False: The user has sent a command to the database and an error is returned. This error is often related to the user providing an incorrect server URL.

A) true

B) false

SQL Programming in Python

Python can connect to many SQL databases by loading the proper library. We will use MySQL, Microsoft SQL Server, and SQLite.

- Python has a standard API called DB-API 2.0.

Step #1 – Install Libraries:

- **MySQL:** `conda install mysql-connector-python`
- **SQL Server:**
 - <https://docs.microsoft.com/en-us/sql/connect/python/python-driver-for-sql-server>
 - Install ODBC driver for SQL Server then: `pip install pyodbc`
- **SQLite:** `pip install pysqlite3`
- **Note:** May have to find `conda.bat` or `conda.sh` or `pip`. Check `Anaconda3/Library/bin`.

Python and MySQL

```
import mysql.connector
```

← Import MySQL API

```
try:
```

```
    cnx = mysql.connector.connect(user='iadaji', password='test',  
                                  host='cosc304.ok.ubc.ca', database='workson')
```

```
    cursor = cnx.cursor()
```

```
    query = "SELECT eno, ename, salary FROM emp WHERE salary < 50000"
```

```
    cursor.execute(query)
```

```
    for (eno, ename, salary) in cursor:  
        print(eno, ename, salary)
```

```
    cursor.close()
```

```
except mysql.connector.Error as err:
```

```
    print(err)
```

```
finally:
```

```
    cnx.close()
```

↑ Execute query and process results

↑ Close connection

Python and MySQL Query with Parameters

```
import mysql.connector
try:
    cnx = mysql.connector.connect(user='iadaji', password='test',
                                   host='cosc304.ok.ubc.ca', database='workson')
    cursor = cnx.cursor()
    query = "SELECT pno, count(*) FROM emp NATURAL JOIN workson "
           "WHERE salary < %s and ename > %s GROUP BY pno"
    cursor.execute(query, (45000, 'L'))
    for (pno, cnt) in cursor:
        print(pno, cnt)
    cursor.close()
except mysql.connector.Error as err:
    print(err)
finally:
    cnx.close()
```


Python and MySQL INSERT Statement

```
try:
    cnx = mysql.connector.connect(user='iadaji', password='test',
                                   host='cosc304.ok.ubc.ca', database='db_ifeoma')
    cursor = cnx.cursor()
    sql = "INSERT INTO emp (eno, ename, salary) VALUES (%s, %s, %s)"
    cursor.execute(sql, ('E10', 'Test Person', 10000))
    cnx.commit() # Update data by committing transaction on connection

    sql = "DELETE FROM emp WHERE eno = 'E10'"
    cursor.execute(sql) # Remove employee just added
    cursor.close()
    cnx.commit()
except mysql.connector.Error as err:
    print(err)
finally:
    cnx.close()
```

Try it: Python and MySQL

Question: Write a program that queries the WorksOn database and returns the employees grouped by title where the employee name is after 'J'. The output should display their title and the average salary for that title. Connection info:

- ```
cnx = mysql.connector.connect(user='iadaji',
password='test', host='cosc304.ok.ubc.ca',
database='workson')
```

Output:

```
EE 30000.000000
ME 40000.000000
PR 20000.000000
SA 50000.000000
```

# ODBC Overview

---

**ODBC** (Open Database Connectivity) is a standard database *application programming interface* (**API**) developed by Microsoft for connecting to databases.

- Java has a similar standard API called JDBC.

The ODBC API contains methods that allow user programs to connect to a database, execute queries, and retrieve results.

For each database system, the vendor writes an **ODBC driver** that implements the API. Application programs can access different systems simply by changing the driver used in their program.

# Python and Microsoft SQL Server

---

**ODBC** is the underlying technology used to connect to Microsoft data sources (Excel, Access, SQL Server, etc.) from Python.

ODBC is language independent (developed first in C/C++) and has been implemented in Python and packaged as library `pyodbc`.

# Python and Microsoft SQL Server

```
import pyodbc
try:
 cnx = pyodbc.connect("""DRIVER={ODBC Driver 17 for SQL Server};
 SERVER=sql04.ok.ubc.ca;
 DATABASE=workson;UID=iadaji;PWD=test""")
 cursor = cnx.cursor()
 cursor.execute("SELECT * FROM emp WHERE salary < ?", [50000])
 for row in cursor:
 print(row[0],row[1], row.salary)
except pyodbc.Error as err:
 print(err)
finally:
 cnx.close()
```

# Database URLs

ODBC and JDBC often specify connection information in a single URL that has the general form:

```
protocol:dbprotocol://server/database?params
```

Example:

`jdbc:mysql://cosc304.ok.ubc.ca/testDB?user=test`

JDBC protocol

subprotocol - used to  
select driver to use

URL of DB  
server

Name of DB  
on server

Parameters

# Python and SQLite

---

**SQLite** is an embedded database designed to be used within other programs.

The amount of data stored in a SQLite database is often considerably smaller than other systems. However, the system is easy to use and transport data between systems.



# Python and SQLite

```
import sqlite3

try:
 cnx = sqlite3.connect("test.db")
 cursor = cnx.cursor()
 cursor.execute("CREATE TABLE test (id int, name text)")
 cursor.execute("INSERT INTO test VALUES(1, 'Joe')")
 cursor.execute("INSERT INTO test VALUES(2, 'Jen')")
 cursor.execute("INSERT INTO test VALUES(3, 'Jeff')")
 cnx.commit()
 cursor.execute("SELECT * FROM test WHERE name > 'Je'")
 for row in cursor:
 print(row)
 cursor.execute("DROP TABLE test")
except sqlite3.Error as err:
 print(err)
finally:
 cnx.close()
```

# SQL Programming in R

---

R can connect to many SQL databases by loading the proper library. We will use MySQL, Microsoft SQL Server, and SQLite.

- R has DBI standard API for querying databases.
  - <https://www.rdocumentation.org/packages/DBI>

## Step #1 – Install Libraries:

- MySQL: `install.packages("RMySQL")`
- MySQL: `install.packages("RMariaDB")`
- SQL Server: `install.packages("odbc")`
- SQLite: `install.packages("RSQLite")`

# R and MySQL Using *RMySQL*

```
library("RMySQL")
con <- dbConnect(RMySQL::MySQL(), user='iadaji', password='test',
dbname='workson', host='cosc304.ok.ubc.ca')

List database tables
dbListTables(con)

Execute a query
res <- dbGetQuery(con, "SELECT * FROM emp")
res

Disconnect
dbDisconnect(con)
```

|   | eno | ename     | bdate      | title | salary | supereno | dno  |
|---|-----|-----------|------------|-------|--------|----------|------|
| 1 | E1  | J. Doe    | 1975-01-05 | EE    | 30000  | E2       | <NA> |
| 2 | E2  | M. Smith  | 1966-06-04 | SA    | 50000  | E5       | D3   |
| 3 | E3  | A. Lee    | 1966-07-05 | ME    | 40000  | E7       | D2   |
| 4 | E4  | J. Miller | 1950-09-01 | PR    | 20000  | E6       | D3   |
| 5 | E5  | B. Casey  | 1971-12-25 | SA    | 50000  | E8       | D3   |
| 6 | E6  | L. Chu    | 1965-11-30 | EE    | 30000  | E7       | D2   |
| 7 | E7  | R. Davis  | 1977-09-08 | ME    | 40000  | E8       | D1   |
| 8 | E8  | J. Jones  | 1972-10-11 | SA    | 50000  | <NA>     | D1   |

# R and MySQL Using *RMySQL*

**Return *project number* and *number of projects* where salary is more than \$45,000 and *ename* is greater than "L"**

```
library(RMySQL)
con <- dbConnect(RMySQL::MySQL(), user='iadaji', password='test',
dbname='workson', host='cosc304.ok.ubc.ca')

Execute a query
salary <- 45000
ename <- "L"
query <- sprintf("SELECT pno, count(*) FROM emp JOIN workson on emp.eno =
workson.eno WHERE salary < '%s' and ename > '%s' GROUP BY pno", salary, ename)

res <- dbGetQuery(con, query)
res

Disconnect
dbDisconnect(con)
```

# R and MySQL

```
library("RMariaDB")
con <- dbConnect(RMariaDB::MariaDB(), user='iadaji', password='test',
dbname='workson', host='cosc304.ok.ubc.ca')

List database tables
dbListTables(con)

Execute a query
res <- dbGetQuery(con, "SELECT * FROM emp")
res

Disconnect
dbDisconnect(con)
```

|   | eno | ename     | bdate      | title | salary | supereno | dno  |
|---|-----|-----------|------------|-------|--------|----------|------|
| 1 | E1  | J. Doe    | 1975-01-05 | EE    | 30000  | E2       | <NA> |
| 2 | E2  | M. Smith  | 1966-06-04 | SA    | 50000  | E5       | D3   |
| 3 | E3  | A. Lee    | 1966-07-05 | ME    | 40000  | E7       | D2   |
| 4 | E4  | J. Miller | 1950-09-01 | PR    | 20000  | E6       | D3   |
| 5 | E5  | B. Casey  | 1971-12-25 | SA    | 50000  | E8       | D3   |
| 6 | E6  | L. Chu    | 1965-11-30 | EE    | 30000  | E7       | D2   |
| 7 | E7  | R. Davis  | 1977-09-08 | ME    | 40000  | E8       | D1   |
| 8 | E8  | J. Jones  | 1972-10-11 | SA    | 50000  | <NA>     | D1   |

# R and Microsoft SQLServer

```
library("DBI")
library("odbc")

db <- dbConnect(odbc::odbc(),
 Driver = 'ODBC Driver 17 for SQL Server',
 Server = 'sql04.ok.ubc.ca', Database = "workson",
 UID='iadaji', PWD='test')

data <- dbGetQuery(db, "SELECT * FROM emp")
data

Disconnect
dbDisconnect(con)
```

# R and SQLite

---

```
library("RSQLite")
library("DBI")

con <- dbConnect(RSQLite::SQLite(), ":memory:")

Write data frame to table
testData <- data.frame(id = 1:5,
 name=c("Abe", "Ben", "Cindy", "Dana", "Emma"))

dbWriteTable(con, "test", testData)
data <- dbGetQuery(con, "SELECT * FROM test")
data

Disconnect
dbDisconnect(con)
```



# R Database Querying with dplyr

```
library(dplyr)
library(dbplyr)

con <- dbConnect(RMariaDB::MariaDB(), user='iadaji', password='test',
dbname='workson', host='cosc304.ok.ubc.ca')

Execute a query
data <- tbl(con, "emp") %>%
 group_by(title) %>%
 summarise(
 totalSalary = sum(salary),
 totalEmp = n())
show_query(data)
data # Executes query

Disconnect
dbDisconnect(con)
```

# SQL Programming: R and Python Question

---

**Question: True or False:** Although the API methods may be slightly different, the general approach to querying a database is very similar in R and Python.

**A)** true

**B)** false

# R SQL Programming Question

---

**Question:** What is the API used to query Microsoft SQL Server with R?

**A)** JDBC

**B)** RMariaDB

**C)** pyodbc

**D)** ODBC

# Try it: R and SQLite

**Question:** Write a program that creates a data frame with 5 rows consisting of id (integer) and message (string). Save the data frame to SQLite. Query SQLite to only return values with id < 12 and message has an "e" in it. Print the results.

Data:

|   | id | message |
|---|----|---------|
| 1 | 10 | Hello   |
| 2 | 11 | Goodbye |
| 3 | 12 | Friend  |
| 4 | 13 | Ant     |
| 5 | 14 | Out     |

# Solution

---

```
SQLite Question

library("RSQLite")
library("DBI")

con <- dbConnect(RSQLite::SQLite(), ":memory:")

Write data frame to table

testData <- data.frame(id = 10:14, message=c("Hello", "Goodbye", "Friend", "Ant",
"Out"))

testData

dbWriteTable(con, "test", testData)

data <- dbGetQuery(con,"SELECT * FROM test WHERE id < 12 AND message LIKE '%e'")

data

Disconnect

dbDisconnect(con)
```

# Conclusion

---

Querying databases can be done from any language with many different APIs. Querying has these steps:

- 1) Load the database access library
- 2) Create a connection to the database
- 3) Execute a SQL command
- 4) Retrieve database results produced by the command
- 5) Close the database connection

We have seen how to query MySQL, Microsoft SQL Server, and SQLite using both Python and R.



THE UNIVERSITY OF BRITISH COLUMBIA

