

Topic 9:

SQL in Applications

SQL in Applications – CSc 460 v1.1 (McCann) – p. 1/14

Classic Approaches

1. Use a preprocessor
 - Usually for older languages (e.g., C and C++)
 - I'll show an example or two of this, just for context
2. Use a library (API)
 - Usually the only option for languages with APIs
 - Often several options per language

The Preprocessor Approach (1 / 2)

A common C program line: `#include <stdio.h>`

But that is not C; rather, it's a _____.

A preprocessor can be used to expand DBMS commands, thus saving us coding:

1. Insert preprocessor statements into program code
2. Execute the DBMS' preprocessor
3. Compile & link the program
4. Execute the application

SQL in Applications – CSc 460 v1.1 (McCann) – p. 3/14

The Preprocessor Approach (2 / 2)

Two varieties of preprocessed SQL statements:

1. Embedded SQL
 - SQL statements are hard-coded (static)
2. Dynamic SQL
 - Arguments added to an SQL statement shell

SQL in Applications – CSc 460 v1.1 (McCann) – p. 4/14

Cursors

A Problem:

How large will your query's result be?

(That is, how much memory do we need to hold what the DBMS is going to return to us?)

The Solution:

SQL in Applications – CSc 460 v1.1 (McCann) – p. 5/14

Preprocessor Examples

See the C & Postgres example, `ecpg.pdf`, on the class web page: “[Embedded and Dynamic SQL APIs in Postgres](#)”

Advantage:

- Can be adapted to any programming language

Disadvantages:

- Several preprocessor directives to learn.
- Very little abstraction (e.g., cursors are explicit)

SQL in Applications – CSc 460 v1.1 (McCann) – p. 6/14

The Library Approach

See the C & Postgres example: “Postgres’ libpq API”

Advantage:

- Just another API; use it like any other API

Disadvantage:

- Might be a 3rd party add-on; needs to be installed

SQL in Applications – CSc 460 v1.1 (McCann) – p. 7/14

ODBC vs. JDBC

ODBC:

- An early 1990s Microsoft API to connect C programs to DBMSes
- ODBC stands for “Open Database Connectivity”
- V4.0 (2018) supports hierarchical and semistructured data

JDBC:

- Sun Microsystem’s (now Oracle’s) 1997 Java API based on ODBC
- JDBC stands for ... _____

SQL in Applications – CSc 460 v1.1 (McCann) – p. 8/14

JDBC

Core capabilities:

Some related technologies:

- SQLJ — a preprocessor-based Java language extension
- Java Persistence API (JPA) — supplies object persistence
- Java Data Objects (JDO) — ditto

SQL in Applications – CSc 460 v1.1 (McCann) – p. 9/14

Using JDBC (1 / 4)

1. Establish connection to a data source

(a) `import java.sql.*`

(b) Load the driver (names vary by DBMS)

- Add your DBMS' JAR file to your classpath:
 - Oracle 11 via lectura: `ojdbc8.jar`
- Call `Class.forName()` to initialize the driver class:
 - Oracle 11: `oracle.jdbc.OracleDriver`

Using JDBC (2 / 4)

(c) Connect to the DBMS

```
Connection dbConnect = DriverManager.getConnection (
    "jdbc:oracle:thin:@host.foo.bar.com:1234:oracle",
    "username", "password" );
```

where:

- The first argument is the DB URL. Parts:
 - `thin` is the type of driver
 - `host.foo.bar.com` is the DBMS server
 - `1234` is the port number
 - `oracle` is the sid (system ID)
- `username` is the user's DBMS login
- `password` is the user's DBMS password

SQL in Applications – CSc 460 v1.1 (McCann) – p. 11/14

Using JDBC (3 / 4)

2. Send SQL statements to that source

Create a Statement object:

```
Statement stmt = dbConnect.createStatement();
```

Ask it to execute the SQL query:

```
ResultSet answer = stmt.executeQuery (
    "SELECT sno, status FROM s" );
```



NOTE: No semicolon after the query!

SQL in Applications – CSc 460 v1.1 (McCann) – p. 12/14

Using JDBC (4 / 4)

3. Process returned results and messages

JDBC uses cursors, too, but the details are implicit.

Before the first read, test `answer.next()`:

If true, a tuple is available

Then, fetch field values by type. E.g.:

```
answer.getString("sno")
```

```
answer.getInt("status")
```

SQL in Applications – CSc 460 v1.1 (McCann) – p. 13/14

Accessing MetaData with JDBC

First, get a `ResultSetMetaData` object by calling:

```
rsmd = answer.getMetaData()
```

Then, fetch the metadata you want to see. E.g.:

<code>rsmd.getColumnCount()</code>	returns degree
<code>rsmd.getColumnName()</code>	returns attr. name
<code>rsmd.getColumnDisplaySize()</code>	returns width

A final ‘FYI’: To get a result’s cardinality, call in sequence:

<code>answer.last()</code>	moves to last tuple
<code>answer.getRow()</code>	to get current row number

SQL in Applications – CSc 460 v1.1 (McCann) – p. 14/14