

# SQL: Programming

CPS 116  
Introduction to Database Systems

## Announcements (September 25)

- ❖ Homework #2 due this Thursday
  - Submit to Yi—not through Jun’s office door
  - Solution available this weekend
- ❖ No class this Thursday
- ❖ Midterm in class next Thursday (October 4)
  - Open book, open notes
  - Format similar to the sample midterm
    - Solution available this weekend
  - Covers everything up to next Tuesday’s lecture
  - Emphasizes materials exercised in homeworks
- ❖ Project milestone #1 due in 2½ weeks

## Motivation

- ❖ Pros and cons of SQL
  - Very high-level, possible to optimize
  - Not intended for general-purpose computation
- ❖ Solutions
  - Augment SQL with constructs from general-purpose programming languages (SQL/PSM)
  - Use SQL together with general-purpose programming languages (JDBC, embedded SQL, etc.)

## Impedance mismatch and a solution

- ❖ SQL operates on a set of records at a time
- ❖ Typical low-level general-purpose programming languages operates on one record at a time
- ❖ Solution: cursor
  - Open (a result table): position the cursor before the first row
  - Get next: move the cursor to the next row and return that row; raise a flag if there is no such row
  - Close: clean up and release DBMS resources
  - ❖ Found in virtually every database language/API
    - With slightly different syntaxes
  - ❖ Some support more positioning and movement options, modification at the current position (analogous to view update), etc.

## Augmenting SQL: SQL/PSM

- ❖ PSM = Persistent Stored Modules
- ❖ CREATE PROCEDURE *proc\_name* ( *parameter\_declarations* )  
*local\_declarations*  
*procedure\_body*;
- ❖ CREATE FUNCTION *func\_name* ( *parameter\_declarations* )  
RETURNS *return\_type*  
*local\_declarations*  
*procedure\_body*;
- ❖ CALL *proc\_name* ( *parameters* );
- ❖ Inside procedure body:  
SET *variable* = CALL *func\_name* ( *parameters* );

## SQL/PSM example

```
CREATE FUNCTION SetMaxGPA(IN newMaxGPA FLOAT)
RETURNS INT
-- Enforce newMaxGPA; return number of rows modified.
BEGIN
  DECLARE rowsUpdated INT DEFAULT 0;
  DECLARE thisGPA FLOAT;
  -- A cursor to range over all students:
  DECLARE studentCursor CURSOR FOR
    SELECT GPA FROM Student
  FOR UPDATE;
  -- Set a flag whenever there is a "not found" exception:
  DECLARE noMoreRows INT DEFAULT 0;
  DECLARE CONTINUE HANDLER FOR NOT FOUND
    SET noMoreRows = 1;
  ... (see next slide) ...
  RETURN rowsUpdated;
END
```

## SQL/PSM example continued

```
-- Fetch the first result row:
OPEN studentCursor;
FETCH FROM studentCursor INTO thisGPA;
-- Loop over all result rows:
WHILE noMoreRows => 1 DO
  IF thisGPA > newMaxGPA THEN
    -- Enforce newMaxGPA:
    UPDATE Student SET Student.GPA = newMaxGPA
    WHERE CURRENT OF studentCursor;
    -- Update count:
    SET rowsUpdated = rowsUpdated + 1;
  END IF;
  -- Fetch the next result row:
  FETCH FROM studentCursor INTO thisGPA;
END WHILE;
CLOSE studentCursor;
```

## Other SQL/PSM features

- ❖ Assignment using scalar query results
  - SELECT INTO
- ❖ Other loop constructs
  - FOR, REPEAT UNTIL, LOOP
- ❖ Flow control
  - GOTO
- ❖ Exceptions
  - SIGNAL, RESIGNAL
- ...
- ❖ For more DB2-specific information, check out *Developing SQL and External Routines*
  - Available as part of DB2 v9 manual collection, or directly as [http://fp.software.ibm.com/pt/products/db2/info/v90/pdf/ctoc/en\\_us/db2atv90.pdf](http://fp.software.ibm.com/pt/products/db2/info/v90/pdf/ctoc/en_us/db2atv90.pdf)

## Interfacing SQL with another language

- ❖ API approach
  - SQL commands are sent to the DBMS at runtime
  - Examples: JDBC, ODBC (for C/C++/VB), Perl DBI
  - These APIs are all based on the SQL/CLI (Call-Level Interface) standard
- ❖ Embedded SQL approach
  - SQL commands are embedded in application code
  - A precompiler checks these commands at compile-time and converts them into DBMS-specific API calls
  - Examples: embedded SQL for C/C++, SQLJ (for Java)

## Example API: JDBC

- ❖ JDBC (Java DataBase Connectivity) is an API that allows a Java program to access databases

```
// Use the JDBC package:
import java.sql.*;
...
public class ... {
  ...
  static {
    // Load the JDBC driver:
    try {
      Class.forName("com.ibm.db2.jcc.DB2Driver");
    } catch (ClassNotFoundException e) {
      ...
    }
  }
  ...
}
```

## Connections

```
// Connection URL is a DBMS-specific string:
String url =
  "jdbc:db2://localhost:50000/dbcourse";
// Making a connection:
Connection con =
  DriverManager.getConnection(url, user, password);
...
// Closing a connection:
con.close();
```

For clarity we are ignoring exception handling for now

## Statements

```
// Create an object for sending SQL statements:
Statement stmt = con.createStatement();
// Execute a query and get its results:
ResultSet rs =
  stmt.executeQuery("SELECT SID, name FROM Student");
// Work on the results:
...
// Execute a modification (returns the number of rows affected):
int rowsUpdated =
  stmt.executeUpdate
    ("UPDATE Student SET name = 'Barney' WHERE SID = 142");
// Close the statement:
stmt.close();
```

## Query results

13

```
// Execute a query and get its results:
ResultSet rs =
    stmt.executeQuery("SELECT SID, name FROM Student");
// Loop through all result rows:
while (rs.next()) {
    // Get column values:
    int sid = rs.getInt(1);
    String name = rs.getString(2);
    // Work on sid and name:
    ...
}
// Close the ResultSet:
rs.close();
```

## Other ResultSet features

14

- ❖ Move the cursor (pointing to the current row) backwards and forwards, or position it anywhere within the `ResultSet`
- ❖ Update/delete the database row corresponding to the current result row
  - Analogous to the view update problem
- ❖ Insert a row into the database
  - Analogous to the view update problem
- ❖ Obtain metadata: `rs.getMetaData()` returns a `ResultSetMetaData` object describing the output table schema (number, order, names, types of columns, etc.)

## Prepared statements: motivation

15

```
Statement stmt = con.createStatement();
for (int age=0; age<100; age+=10) {
    ResultSet rs = stmt.executeQuery
        ("SELECT AVG(SPA) FROM Student" +
         " WHERE age >= " + age + " AND age < " + (age+10));
    // Work on the results:
    ...
}
```

- ❖ Every time an SQL string is sent to the DBMS, the DBMS must perform parsing, semantic analysis, optimization, compilation, and then finally execution
- ❖ These costs are incurred 10 times in the above example
- ❖ A typical application issues many queries with a small number of patterns (with different parameter values)

## Prepared statements: syntax

16

```
// Prepare the statement, using ? as placeholders for actual parameters:
PreparedStatement stmt = con.prepareStatement
    ("SELECT AVG(SPA) FROM Student WHERE age >= ? AND age < ?");
for (int age=0; age<100; age+=10) {
    // Set actual parameter values:
    stmt.setInt(1, age);
    stmt.setInt(2, age+10);
    ResultSet rs = stmt.executeQuery();
    // Work on the results:
    ...
}
```

- ❖ The DBMS performs parsing, semantic analysis, optimization, and compilation only once, when it prepares the statement
- ❖ At execution time, the DBMS only needs to check parameter types and validate the compiled execution plan

## Odds and ends of JDBC

17

- ❖ Most methods can throw `SQLException`
  - Make sure your code catches them
  - Remember to close `Statement`, `ResultSet`, etc., in finally block
  - `getSQLState()` returns the standard SQL error code
  - `getMessage()` returns the error message
- ❖ `DataSource` interface for establishing connections
  - Better than through `DriverManager`
- ❖ Methods for examining metadata in databases
- ❖ Methods to retrieve the value of a column for all result rows into an array without calling `ResultSet.next()` in a loop
- ❖ Methods to construct/execute a batch of SQL statements
- ...

## JDBC drivers – Types I, II

18

- ❖ Type I (bridge): translate JDBC calls to a standard API not native to the DBMS (e.g., JDBC-ODBC bridge)
  - Driver is easy to build using existing standard API's
  - Extra layer of API adds overhead
- ❖ Type II (native API, partly Java): translates JDBC calls to DBMS-specific client API calls
  - DBMS-specific non-Java client library needs to be installed on each client
  - Good performance