Embedded SQL & JDBC

Kathleen Durant

CS 3200

Lesson 8

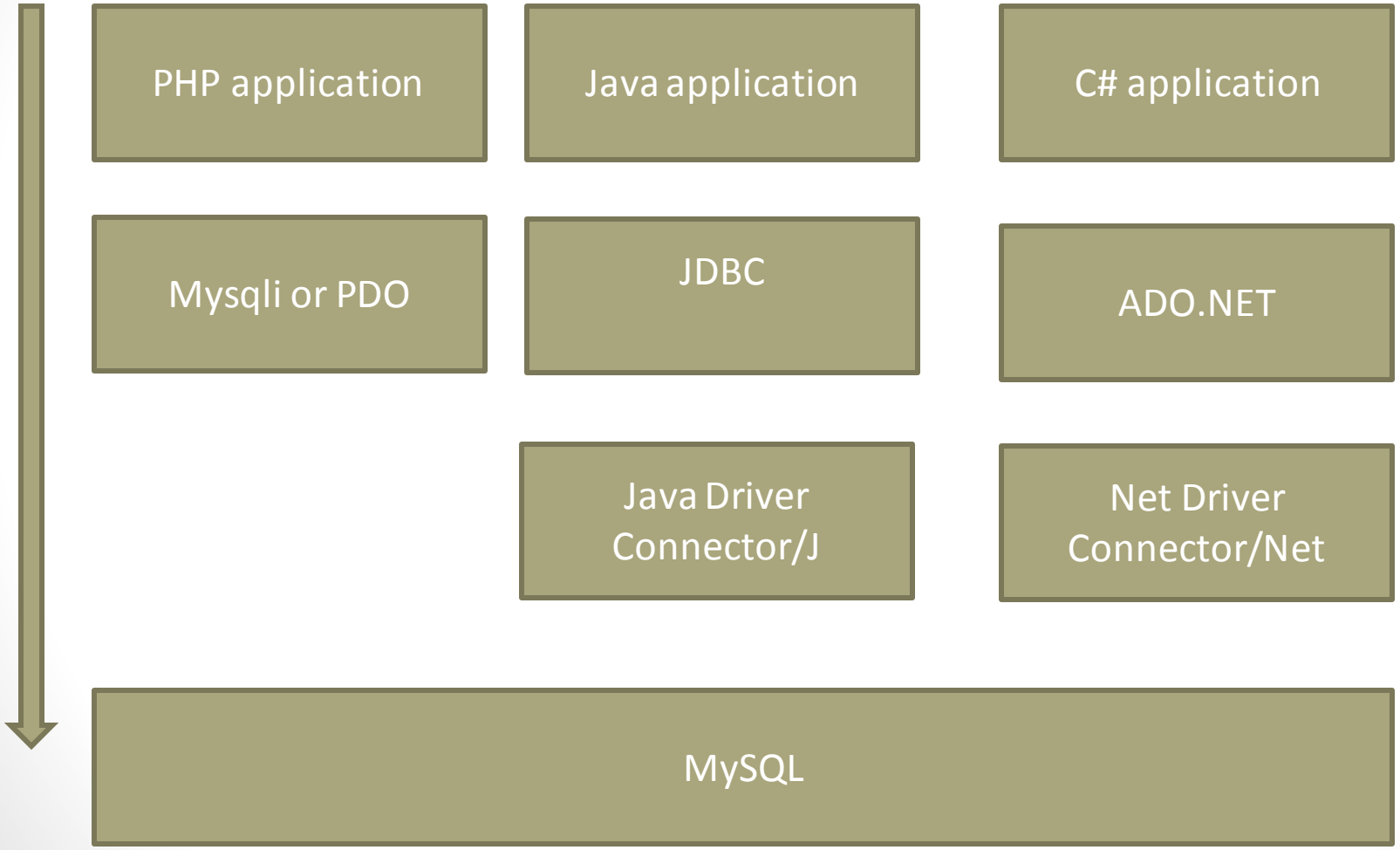
Outline for today

- SQL in application code such as C++ and Java
- Embedded SQL
- Cursors
- Dynamic SQL
- JDBC
- SQLJ
- Stored procedures

Database API's

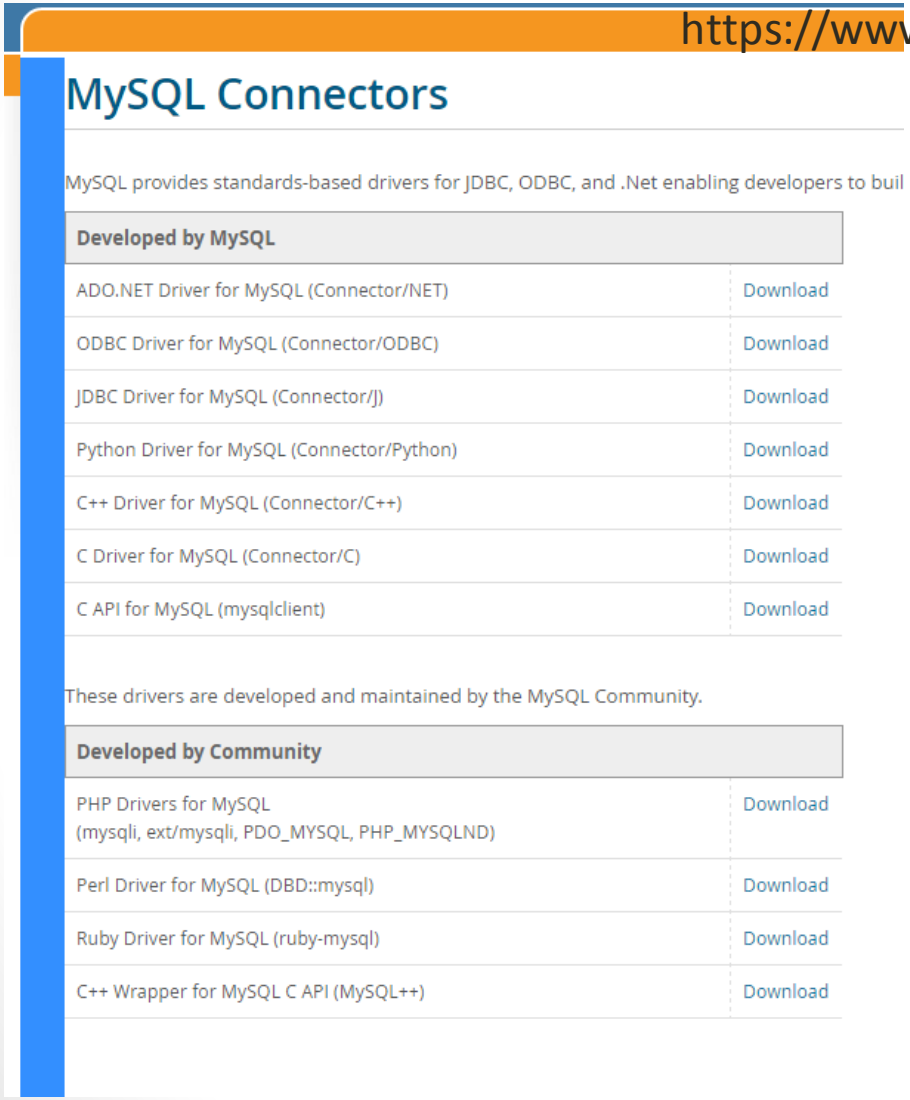
- Rather than modify the compiler, add a library with database calls (API)
  - Special standardized interface: procedures/objects
  - Pass SQL strings from language, presents result sets in a language-friendly way
    - Sun's *JDBC:* Java API
    - Supposedly DBMS-neutral
- A "driver" traps the calls and translates them into DBMS specific code
  - database can be across a network

Embedded SQL



Download the driver you want

https://www.mysql.com/products/connector/



MySQL Connectors

MySQL provides standards-based drivers for JDBC, ODBC, and .Net enabling developers to buil

Developed by MySQL	
ADO.NET Driver for MySQL (Connector/NET)	Download
ODBC Driver for MySQL (Connector/ODBC)	Download
JDBC Driver for MySQL (Connector/J)	Download
Python Driver for MySQL (Connector/Python)	Download
C++ Driver for MySQL (Connector/C++)	Download
C Driver for MySQL (Connector/C)	Download
C API for MySQL (mysqlclient)	Download

These drivers are developed and maintained by the MySQL Community.

Developed by Community	
PHP Drivers for MySQL (mysqli, ext/mysqli, PDO_MYSQL, PHP_MYSQLND)	Download
Perl Driver for MySQL (DBD::mysql)	Download
Ruby Driver for MySQL (ruby-mysql)	Download
C++ Wrapper for MySQL C API (MySQL++)	Download

SQL code in other programming languages

- SQL commands can be called from within a host language (e.g., C++ or Java) program.
  - SQL statements can refer to host variables (including special variables used to return status).
  - Must include a statement to *connect* to the right database.
- Two main integration approaches:
  - Embed SQL in the host language (Embedded SQL, SQLJ)
  - Create special API to call SQL commands (JDBC)

Issues with Embedded SQL

- SQL relations are (multi-) sets of records, with no *a priori* bound on the number of records.
- No such data structure exist traditionally in procedural programming languages

SQL supports a mechanism called a *cursor* to handle this.

Embed SQL in the host language

- How does it work?
  - A preprocessor converts the SQL statements into specific API calls.
  - Compiler takes preprocessed file as input
- Supporting Language constructs:
  - Connecting to a database:
    - EXEC SQL CONNECT
  - Declaring variables:
    - EXEC SQL BEGIN SECTION
      <DECLARATIONS>
    - END DECLARE SECTION
  - Statements:
    - EXEC SQL Statement;

Example: Embedded SQL Declare

- Special variables at least one needs to be declared by your code for error handling
  - SQLCODE (long, is negative if an error has occurred)
  - SQLSTATE (char[6], predefined codes for common errors)

- User Variables
- EXEC SQL BEGIN DECLARE SECTION

```
char c_sname[20];
long c_sid;
short c_rating;
float c_age;
```

- EXEC SQL END DECLARE SECTION

Handling multisets in your Application

- Can *DECLARE* a cursor on a relation or query statement
  - *OPEN* a cursor
  - repeatedly *FETCH* a tuple
  - then *MOVE* the cursor until all tuples have been retrieved.
  - Can use a special clause, called ORDER BY, in queries that are accessed through a cursor, to control the order in which tuples are returned.
    - Fields in ORDER BY clause must also appear in SELECT clause
    - Restriction typically not true when ORDER BY used without cursor
- Can also modify/delete tuple pointed to by a cursor.

ANSWER: CURSOR

Declare a cursor: state what multiset you are perusing

Peruse the sailor's names that have reserved a red boat

```
EXEC SQL DECLARE sinfo CURSOR FOR
        SELECT S.sname
       FROM Sailors S, Boats B, Reserves R
      WHERE S.sid=R.sid AND R.bid=B.bid
      AND  B.color='red' ORDER BY S.sname
```

General declaration of a cursor

DECLARE cursorname [INSENSITIVE|SCROLL|CURSOR] [WITH HOLD]
FOR some query
[ORDER BY order-item-list]
[FOR READ ONLY | FOR UPDATE]

Define the scope of the cursors operation with:
[FOR READ ONLY | FOR **UPDATE**] **DEFAULT VALUE**
Determine the order of the return values with:
[ORDER BY order-item-list]
Cannot update a field that you use to order your return set
SCROLL provides more powerful cursor positioning than just the FETCH operation
Seek functionality within the cursor
MY SQL seems to have problems with this functionality
INSENSITIVE -Grabs a local copy of the multiset (INSENSITIVE TO CHANGES)
- Do NOT UPDATE my multiset while I am working on it
- Emulated in MY SQL via a temporary file
WITH HOLD – Do not release the cursor at the end of the transaction

Example: Print all sailor's age and name with a rating greater than 5

- **EXEC SQL BEGIN DECLARE SECTION**
  char c_sname[20];
  short c_minrating;
  float c_age;
  EXEC SQL END DECLARE SECTION

Variable Declarations: Variables used in the SQL statement

c_minrating = 5;
EXEC SQL DECLARE sinfo **CURSOR FOR**
SELECT S.sname, S.age FROM Sailors S
WHERE S.rating > :c_minrating ORDER BY S.sname;

Cursor Declaration

do {
EXEC SQL FETCH sinfo **INTO** :c_sname, :c_age;
printf("%s is %d years old\n", c_sname, c_age);
} while (SQLSTATE < '02000');

Do some work with the declared variables

Until Last record

EXEC SQL CLOSE sinfo;

Close the cursor

SQL Errors: SQLSTATE return values

Success	"00000"
Success, but no rows found	"02000"
Success, but warnings generated	"01"
Failure, Runtime Error Generated	> “02”

- SQLSTATE return values are standardized and determined by SQL standard 92
- SQLSTATE values are comprised of a two-character class code value, followed by a three-character subclass code value
- Complete table at:
- http://publib.boulder.ibm.com/infocenter/dzichelp/v2r2/index.jsp?topic=%2Fcom.ibm.db2z10.doc.codes%2Fsrc%2Ftpc%2Fdb2z_sqlstatevalues.htm

- https://dev.mysql.com/doc/refman/5.6/en/mysql-sqlstate.html

SQLSTATE in My SQL

- Use **mysql_sqlstate()**

  const char *mysql_sqlstate(MYSQL *mysql)

To get the result for the last executed SQL statement

https://dev.mysql.com/doc/refman/5.6/en/mysql-sqlstate.html

Dynamic SQL

- What if you do not know what data you are looking for until your program is running
- The data you want is determined by user input or some other event that occurs after compilation time
- Can construct the queries on the fly via a well defined algorithm

ANSWER: Dynamic SQL

Dynamic SQL example

- char c_sqlstring[]=
- {“DELETE FROM Sailors WHERE rating>5”};
- **EXEC SQL PREPARE** readytogo FROM :c_sqlstring;
- **EXEC SQL EXECUTE** readytogo;

MY SQL Dynamic SQL stored procedure

```
DECLARE TableName VARCHAR(32);
 SET @SelectedId = NULL;
SET @s := CONCAT("SELECT Id INTO @SelectedId FROM ", TableName,
                                        " WHERE param=val LIMIT 1");
PREPARE stmt FROM @s;
EXECUTE stmt;
DEALLOCATE PREPARE stmt;
```

PREPARE command:

- Parses the query,
- Determine the execution plan
- Associates an identifier (handle) with the statement (stmt)
- This statement can be executed again, and again, and again
- Execute statement can actually take parameters

PREPARE: passing parameters

PREPARE stmt FROM ‘SELECT COUNT(*) FROM information_schema.schemata where schema_name = **?**‘

SET @schema := 'test';

EXECUTE stmt USING @schema;

- **Parameter binding occurs in a positional manner**
- PREPARE stmt FROM 'select count(*) from information_schema.schemata WHERE schema_name = **?** OR schema_name = **?**‘

- EXECUTE stmt USING @schema, @schema1

JDBC Processing

- Steps to submit a database query:
  - Load the JDBC driver
  - Connect to the data source
  - Execute SQL statements

JDBC Architecture: 4 components

- **Application** (initiates and terminates connections, submits SQL statements)
- **Driver manager** (load JDBC driver)
- **Driver** (connects to data source, transmits requests and returns/translates results and error codes)
- **Data source** (processes SQL statements)

JDBC: Driver Manager

- All drivers are managed by the DriverManager class
  - Loading a JDBC driver:

    In the Java code:
  - Class.forName(“oracle/jdbc.driver.Oracledriver”); /Oracle
  - Class.forName("com.mysql.jdbc.Driver"); /My SQL
- When starting the Java application:
  - -Djdbc.drivers=oracle/jdbc.driver
- Or provide the driver in the CLASSPATH directory

Connecting to DB via JDBC

Interact with a data source through sessions. Each connection identifies a logical session.

- JDBC URL:
- jdbc:<subprotocol>:<otherParameters>

Example:

```
//Define URL of database server for
 // database named mysql on the localhost
 // with the default port number 3306.
 String url =
    "jdbc:mysql://localhost:3306/mysql";

//Get a connection to the database for a user named root with a root password.
// This user is the default administrator having full privileges to do anything.
Connection con = DriverManager.getConnection( url,"root", "root");

//Display URL and connection information
 System.out.println("URL: " + url);
 System.out.println("Connection: " + con);
```

Connection Class Interface

- public int getTransactionIsolation() and

  void setTransactionIsolation(int level)
  - Sets isolation level for the current connection.
- public boolean getReadOnly() and void setReadOnly(boolean b)
  - Specifies whether transactions in this connection are readonly
- public boolean getAutoCommit()

  and void setAutoCommit(boolean b)
  - If autocommit is set, then each SQL statement is considered its own transaction. Otherwise, a transaction is committed using commit(), or aborted using rollback().
- public boolean isClosed()
  - Checks whether connection is still open.

Executing SQL Statements

- Three different methods to execute SQL statements:
  - **Statement** (both static and dynamic SQL statements)
  - **PreparedStatement** (semi-static SQL statements)
  - **CallableStatment** (stored procedures)
- PreparedStatement class: Precompiled, parametrized SQL statements:
  - Structure of the SQL statement is fixed
  - Values of parameters are determined at run-time

PreparedStatement: Passing and defining Parameters

```
String sql="INSERT INTO Sailors VALUES(?,?,?,?)";
PreparedStatment pstmt=con.prepareStatement(sql);
pstmt.clearParameters();
pstmt.setInt(1,sid);
pstmt.setString(2,sname);
pstmt.setInt(3, rating);
pstmt.setFloat(4,age);
// No return rows use executeUpdate()
int numRows = pstmt.executeUpdate();
```

Parameters are positional

Result Sets

- PreparedStatement.executeUpdate only returns the number of affected records
- PreparedStatement.executeQuery returns data, encapsulated in a ResultSet object (a cursor)
- ResultSet rs=pstmt.executeQuery(sql);
- // rs is now a cursor
- While (rs.next()) {
- // process the data
- }

ResultSet: Cursor with seek functionality

- A ResultSet is a very powerful cursor:
  - **previous**(): moves one row back
  - **absolute**(int num): moves to the row with the specified number
  - **relative** (int num): moves forward or backward
  - **first**() and **last**()

Java to SQL Data Types and Result methods

SQL Type	Java class	Result Set get method
BIT	Boolean	getBoolean()
CHAR	String	getString()
VARCHAR	String	getString()
DOUBLE	Double	getDouble()
FLOAT	Double	getDouble()
INTEGER	Integer	getInt()
REAL	Double	getFloat()
DATE	Java.sql.Date	getDate()
TIME	Java.sql.Time	getTime()
TIMESTAMP	Java.sql.Timestamp	getTimestamp()

JDBC: Processing exceptions and warnings

- Most of java.sql can throw an error and set SQLException when an error occurs
- SQLWarning is a subclass of SQLException
  - Not as severe as an error
  - They are not thrown
  - Code has to explicitly test for a warning

Example of catching and error

```
try {
        stmt=con.createStatement();
        warning=con.getWarnings();
        while(warning != null) {
                // handle SQLWarnings;
                warning = warning.getNextWarning():
         }
     con.clearWarnings();
    stmt.executeUpdate(queryString);
     warning = con.getWarnings();
          …
  } //end try
  catch( SQLException SQLe) {
   // handle the exception
```

Examining Metadata on the DB

DatabaseMetaData object gives information about the database system and the catalog.

```
DatabaseMetaData md = con.getMetaData();
// print information about the driver:
System.out.println(
“Name:” + md.getDriverName() +
“version: ” + md.getDriverVersion());
```

Metadata: Print out table and its columns

```
DatabaseMetaData md=con.getMetaData();
ResultSet trs=md.getTables(null,null,null,null);
String tableName;
While(trs.next()) {
        tableName = trs.getString("TABLE_NAME");
        System.out.println("Table: " + tableName);
        //print all attributes
        ResultSet crs = md.getColumns(null,null,tableName, null);
        while (crs.next()) {
            System.out.println(crs.getString("COLUMN_NAME" + ", ");
         }
  }
```

http://docs.oracle.com/javase/7/docs/api/java/sql/DatabaseMetaData.html

Connect, Process, Check errors

```
Connection con = // connect
    DriverManager.getConnection(url,”login",”pass");
Statement stmt = con.createStatement(); // set up stmt
String query = "SELECT name, rating FROM Sailors";
ResultSet rs = stmt.executeQuery(query);
try { // handle exceptions
      // loop through result tuples
       while (rs.next()) {
            String s = rs.getString(“name");
             Int n = rs.getFloat(“rating");
             System.out.println(s + " " + n);
        }
   } catch(SQLException ex) {
       System.out.println(ex.getMessage () +
         ex.getSQLState () + ex.getErrorCode ());
}
```

Connect

Get multiset

Process with cursor

Catch Errors

SQLJ

- Complements JDBC with a (semi-)static query model: Compiler can perform syntax checks, strong type checks, consistency of the query with the schema
  - All arguments always bound to the same variable:
  - #sql = { SELECT name, rating INTO :name, :rating FROM Books WHERE sid = :sid;
- Compare to JDBC:

```
sid=rs.getInt(1);
if (sid==1) {sname=rs.getString(2);}
else { sname2=rs.getString(2);}
```

SQLJ (part of the SQL standard) versus embedded SQL (vendor-specific)

SQLJ Code

```
Int sid; String name; Int rating;
// named iterator
#sql iterator Sailors(Int sid, String name, Int rating);
Sailors sailors;
// assume that the application sets rating
#sailors = {
        SELECT sid, sname INTO :sid, :name
         FROM Sailors WHERE rating = :rating
};
// retrieve results
while (sailors.next()) {
            System.out.println(sailors.sid + " " + sailors.sname));
  }
sailors.close();
```

SQLJ Iterators

- Two types of iterators ("cursors"):
- Named iterator
  - Need both variable type and name, and then allows retrieval of columns by name.
  - See example on previous slide.
- Positional iterator
  - Need only variable type, and then uses FETCH .. INTO construct:

```
#sql iterator Sailors(Int, String, Int);
  Sailors sailors;
  #sailors = …
  while (true) {
        #sql {FETCH :sailors INTO :sid, :name} ;
        if (sailors.endFetch()) { break; }
        // process the sailor
    }
```

Stored procedures

- Program executed through a single SQL statement
- Executed in the process space of the server
- Advantages:
  - Can encapsulate application logic while staying "close" to the data
  - Reuse of application logic by different users
  - Avoid tuple-at-a-time return of records through cursors

Stored Procedure: Examples

- CREATE PROCEDURE ShowNumReservations

SELECT S.sid, S.sname, COUNT(*) FROM Sailors S, Reserves R
WHERE S.sid = R.sid GROUP BY S.sid, S.sname

- **Create procedure can have parameters**
  - **Three different modes: IN, OUT, INOUT**
- CREATE PROCEDURE IncreaseRating(

    IN sailor_sid INTEGER, IN increase INTEGER)

  UPDATE Sailors

    SET rating = rating + increase

    WHERE sid = sailor_sid

Calling stored procedures

```
EXEC SQL BEGIN DECLARE SECTION
    Int sid;
    Int rating;
    EXEC SQL END DECLARE SECTION

// now increase the rating of this sailor
 EXEC CALL IncreaseRating(:sid,:rating);
```

Stored Procedures can be written in other languages

- CREATE PROCEDURE TopSailors(
  IN num INTEGER)
  LANGUAGE JAVA
  EXTERNAL NAME
  “file:///c:/storedProcs/rank.jar”

Calling procedures from JDBC, SQLJ

JDBC:

```
CallableStatement cstmt=
    con.prepareCall("{call
    ShowSailors}");
ResultSet rs =
   cstmt.executeQuery();
    while (rs.next()) {
        //process data …
     }
```

SQLJ

```
#sql iterator
    ShowSailors(…);
ShowSailors showsailors;
#sql showsailors={CALL
        ShowSailors};
while (showsailors.next())
{
      /process data …
}
```

SQL/PSM

- Most DBMSs allow users to write stored procedures in a simple, general-purpose language (close to SQL) SQL/PSM standard is a representative
- **Declare a stored procedure:**
  - CREATE PROCEDURE name(p1, p2, …, pn)
  - local variable declarations
  - procedure code;
- **Declare a function:**
  - CREATE FUNCTION name (p1, …, pn) RETURNS sqlDataType
    local variable declarations
    function code

CREATE A FUNCTION

```
CREATE FUNCTION rate Sailor
        IN sailorId INTEGER)
        RETURNS INTEGER
DECLARE rating INTEGER
DECLARE numRes INTEGER
SET numRes = (SELECT COUNT(*) FROM Reserves R
                        WHERE R.sid = sailorId)
IF (numRes > 10) THEN rating =1;
 ELSE rating = 0;
END IF;
RETURN rating;
```

Function constructs

- Local variables (DECLARE)
- RETURN values for FUNCTION
- Assign variables with SET
- Branches and loops:
  - IF (condition) THEN statements;
  - ELSEIF (condition) statements;
  - … ELSE statements; END IF;
  - LOOP statements; END LOOP
- Queries can be parts of expressions
- Can use cursors naturally without “EXEC SQL”

Summary

- Embedded SQL allows execution of parameterized static queries within a host language
- Dynamic SQL allows execution of completely ad hoc queries within a host language
- Cursor mechanism allows retrieval of one record at a time and bridges impedance mismatch between host language and SQL
- APIs such as JDBC introduce a layer of abstraction between application and DBMS
- SQLJ: Static model, queries checked at compile-time.
- Stored procedures execute application logic directly at the server
- SQL/PSM standard for writing stored procedures

MySQL Connectors

- ODBC (MySQL Connector/ODBC (sometimes called just Connector ODBC or MyODBC)
  - A driver for connecting to a MySQL database server through the Open Database Connectivity (ODBC) application program interface (API)
  - Standard means of connecting to any database.
  - Supports .NET
- JDBC Connector