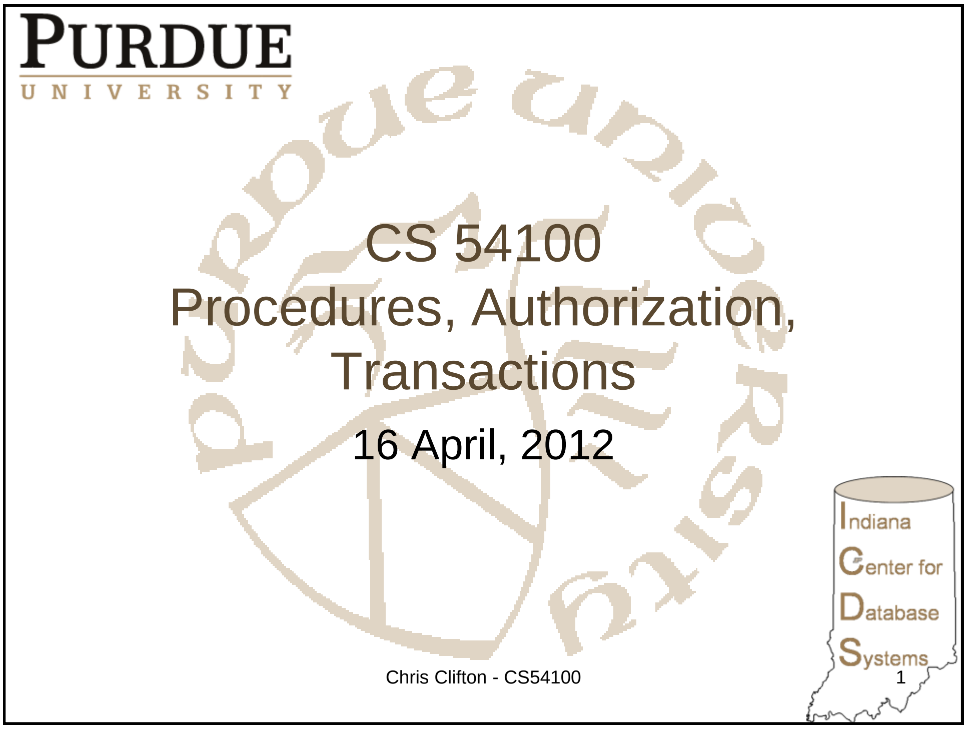
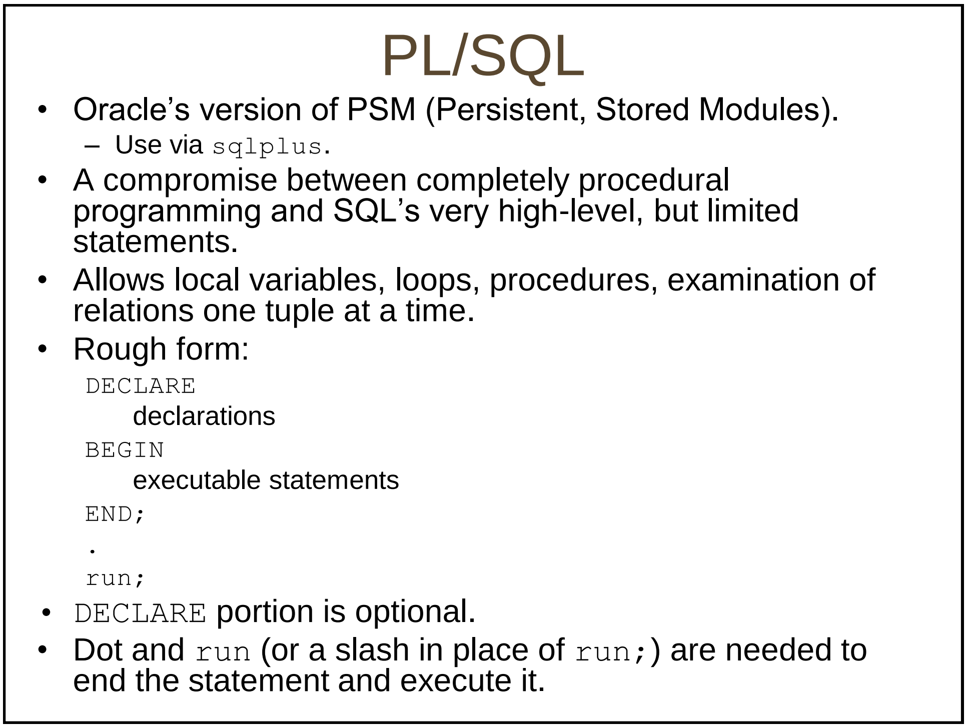


CS 54100
Procedures, Authorization, Transactions

16 April, 2012

PURDUE
UNIVERSITY
Indiana
Center for
Database
Systems

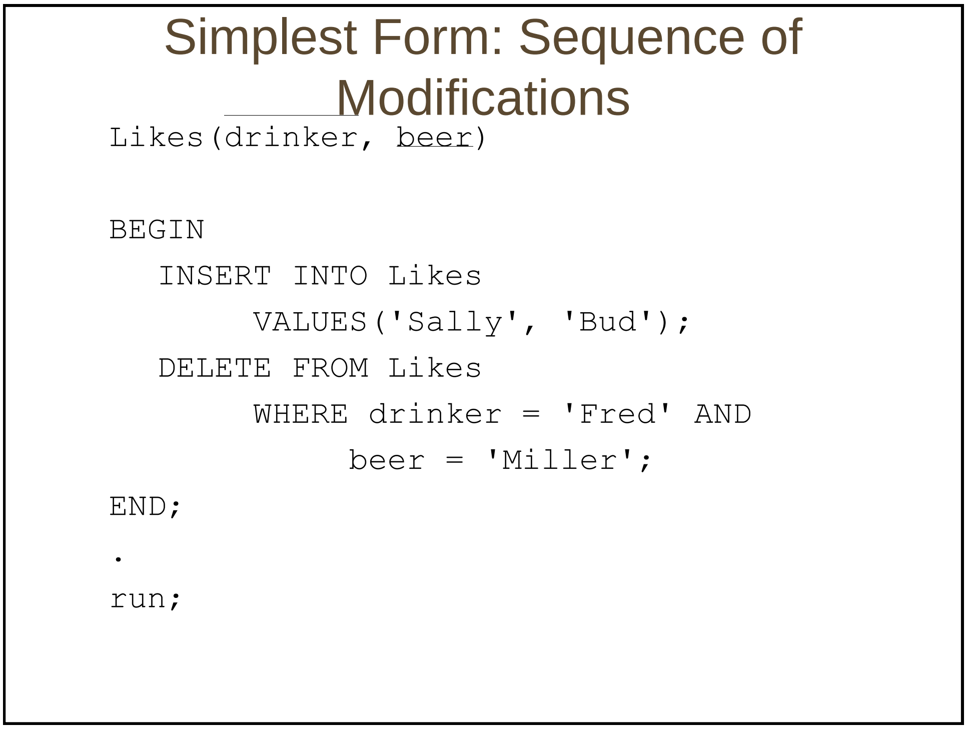


PL/SQL

- Oracle's version of PSM (Persistent, Stored Modules).
  - Use via `sqlplus`.
- A compromise between completely procedural programming and SQL's very high-level, but limited statements.
- Allows local variables, loops, procedures, examination of relations one tuple at a time.
- Rough form:

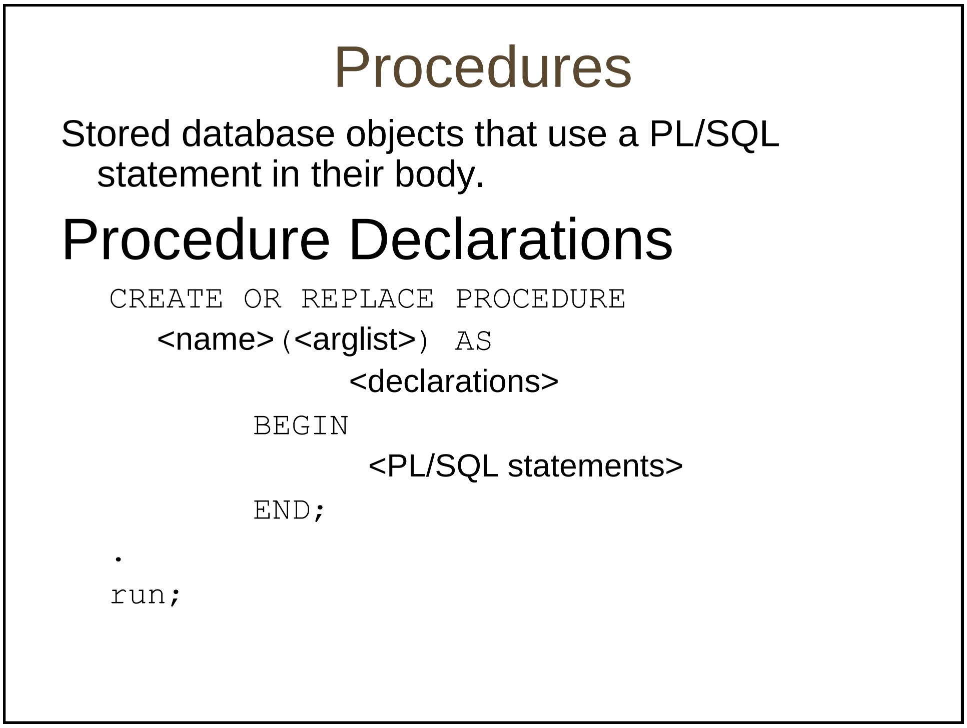
```
DECLARE
     declarations
BEGIN
     executable statements
END;
.
run;
```

- `DECLARE` portion is optional.
- Dot and `run` (or a slash in place of `run;`) are needed to end the statement and execute it.



Simplest Form: Sequence of Modifications

```
Likes(drinker, beer)

BEGIN
   INSERT INTO Likes
         VALUES('Sally', 'Bud');
   DELETE FROM Likes
         WHERE drinker = 'Fred' AND
               beer = 'Miller';
END;
.
run;
```



Procedures

Stored database objects that use a PL/SQL statement in their body.

Procedure Declarations

```
CREATE OR REPLACE PROCEDURE
   <name>(<arglist>) AS
               <declarations>
         BEGIN
                <PL/SQL statements>
         END;
.
run;
```

- Argument list has name-mode-type triples.
  - Mode: IN, OUT, or IN OUT for read-only, write-only, read/write, respectively.
  - Types: standard SQL + generic types like `NUMBER` = any integer or real type.
  - Since types in procedures *must* match their types in the DB schema, you should generally use an expression of the form

    relation.attribute `%TYPE`

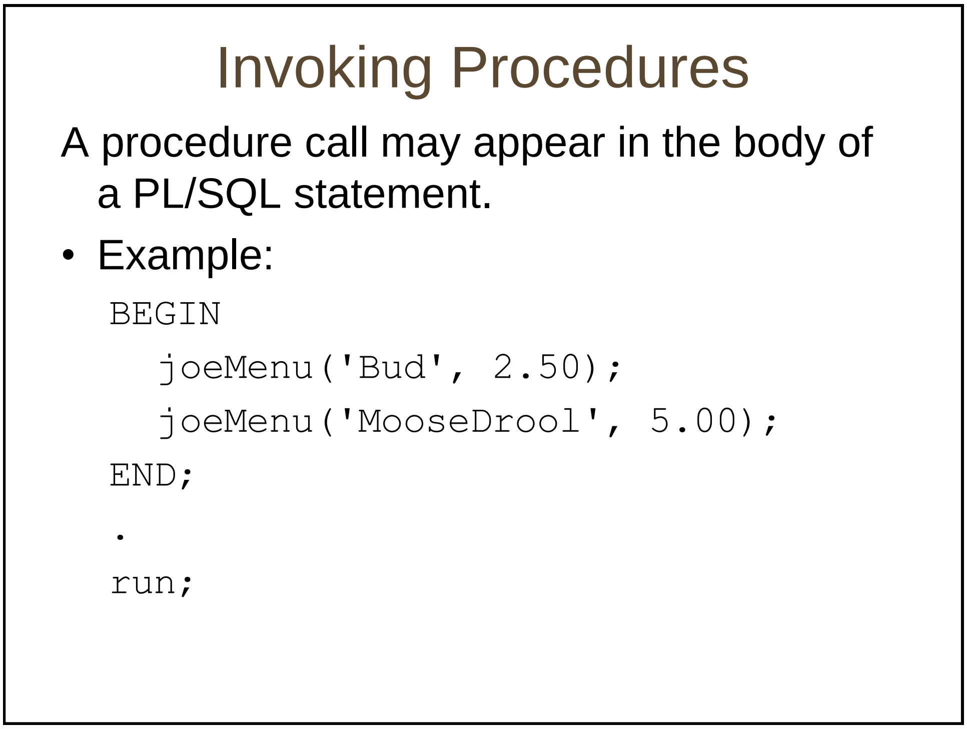
    to capture the type correctly.

Example

A procedure to take a beer and price and add it to Joe's menu.

```
Sells(bar, beer, price)

CREATE PROCEDURE joeMenu(
   b IN Sells.beer %TYPE,
   p IN Sells.price %TYPE
) AS
   BEGIN
         INSERT INTO Sells
         VALUES('Joe''s Bar', b, p);
   END;
.
run;
```

- Note "`run`" only stores the procedure; it doesn't execute the procedure.

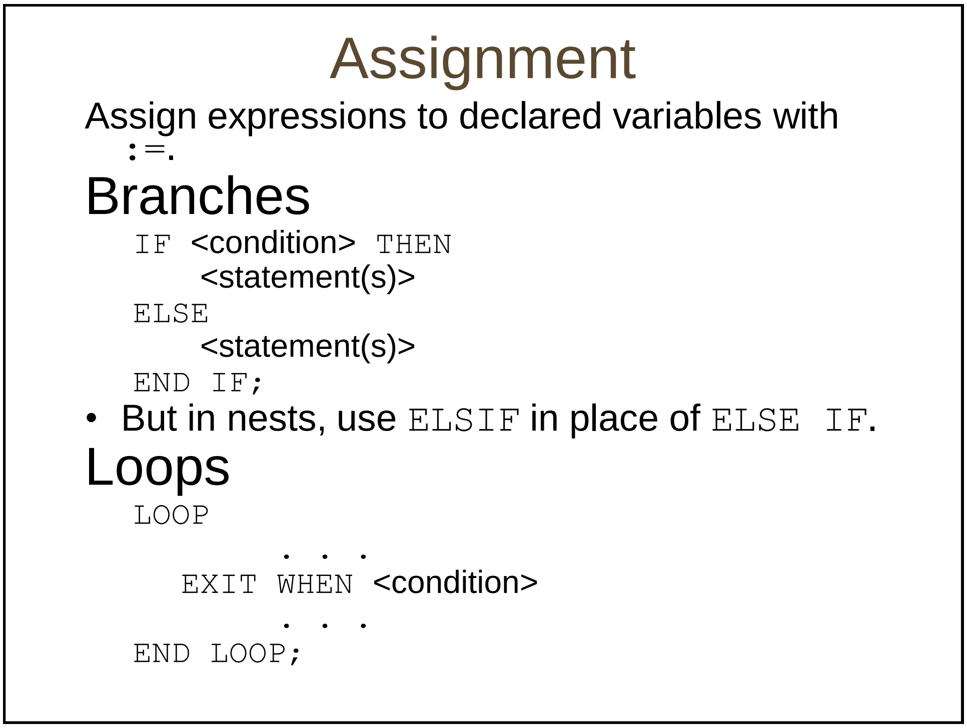


Invoking Procedures

A procedure call may appear in the body of a PL/SQL statement.

- Example:

```
BEGIN
   joeMenu('Bud', 2.50);
   joeMenu('MooseDrool', 5.00);
END;
.
run;
```



Assignment

Assign expressions to declared variables with `:=`.

Branches

```
IF <condition> THEN
    <statement(s)>
ELSE
    <statement(s)>
END IF;
```

- But in nests, use `ELSIF` in place of `ELSE IF`.

Loops

```
LOOP
         . . .
   EXIT WHEN <condition>
         . . .
END LOOP;
```

Queries in PL/SQL

1. *Single-row selects* allow retrieval into a variable of the result of a query that is guaranteed to produce one tuple.
2. *Cursors* allow the retrieval of many tuples, with the cursor and a loop used to process each in turn.

Single-Row Select

- Select-from-where in PL/SQL *must* have an `INTO` clause listing variables into which a tuple can be placed.
- It is an *error* if the select-from-where returns more than one tuple; you should have used a cursor.

Example

- Find the price Joe charges for Bud (and drop it on the floor).

```
Sells(bar, beer, price)

DECLARE
   p Sells.price %TYPE;
BEGIN
   SELECT price
   INTO p
   FROM Sells
   WHERE bar = 'Joe''s Bar' AND beer = 'Bud';
END;
.
run
```

Cursors

Declare by:

```
CURSOR <name> IS
    select-from-where statement
```

- Cursor gets each tuple from the relation produced by the select-from-where, in turn, using a *fetch statement* in a loop.
  - Fetch statement:

```
FETCH <cursor name> INTO
        variable list;
```

- Break the loop by a statement of the form:

```
EXIT WHEN <cursor name> %NOTFOUND;
```

  - True when there are no more tuples to get.
- Open and close the cursor with `OPEN` and `CLOSE`.

Example

A procedure that examines the menu for Joe's Bar and raises by $1.00 all prices that are less than $3.00.

```
Sells(bar, beer, price)
```

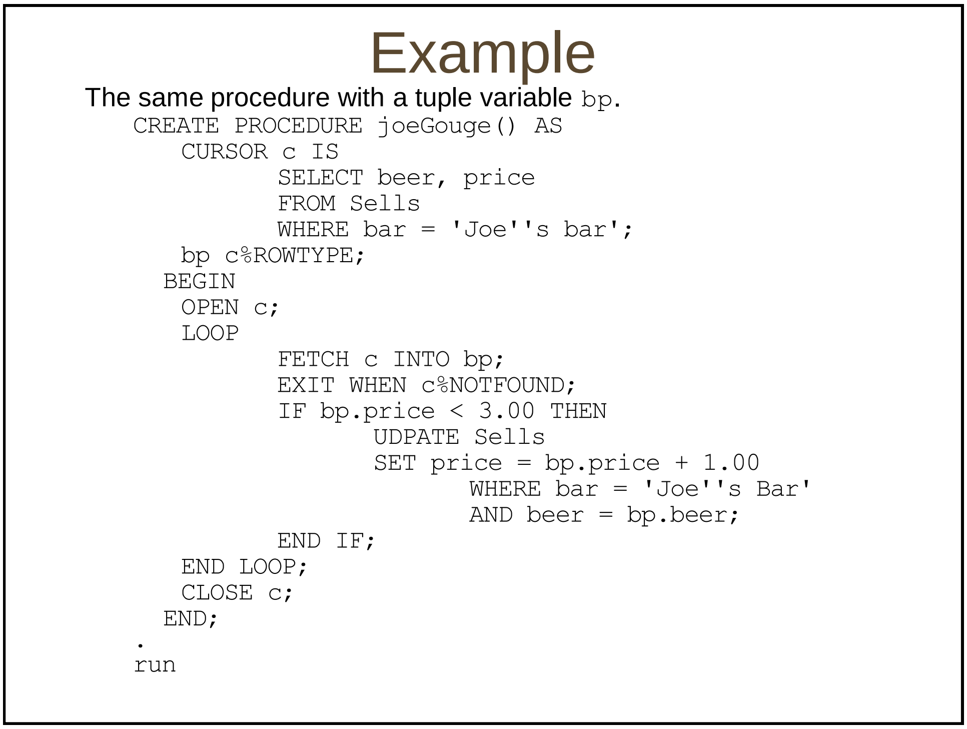
- This simple price-change algorithm can be implemented by a single `UPDATE` statement, but more complicated price changes could not.

```
CREATE PROCEDURE joeGouge() AS
   theBeer Sells.beer%TYPE;
   thePrice Sells.price%TYPE;
   CURSOR c IS
         SELECT beer, price
         FROM Sells
         WHERE bar = 'Joe''s bar';
  BEGIN
   OPEN c;
   LOOP
         FETCH c INTO theBeer, thePrice;
         EXIT WHEN c%NOTFOUND;
         IF thePrice < 3.00 THEN
               UDPATE Sells
               SET price = thePrice + 1.00
               WHERE bar = 'Joe''s Bar'
                     AND beer = theBeer;
         END IF;
   END LOOP;
   CLOSE c;
  END;
.
run
```

Row Types

Anything (*e.g.*, cursors, table names) that has a tuple type can have its type captured with `%ROWTYPE`.

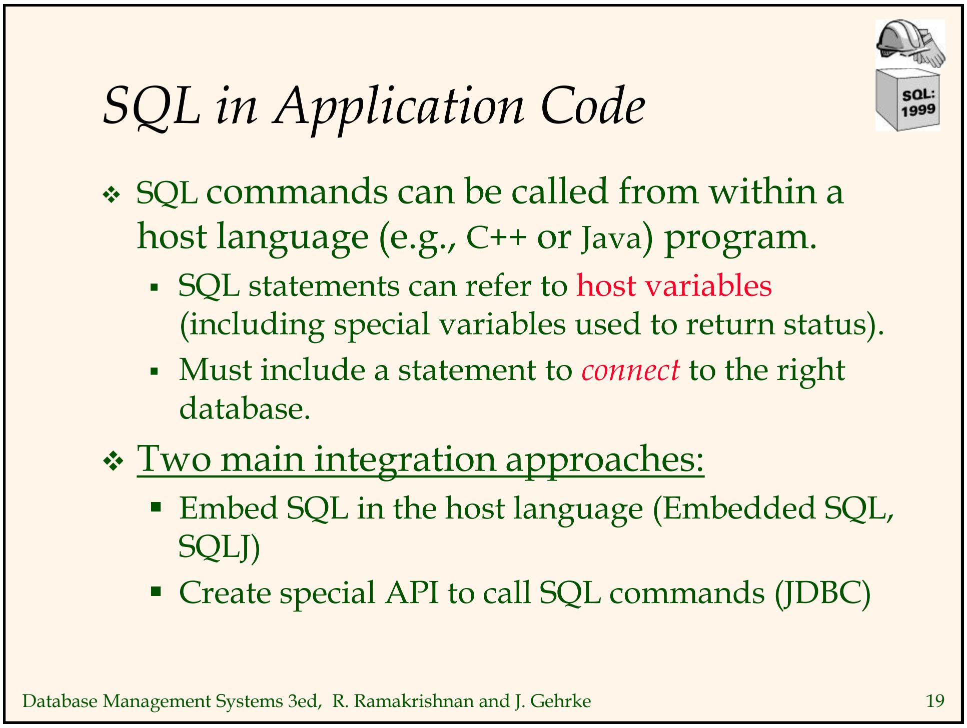
- We can create temporary variables that have tuple types and access their components with dot.
- Handy when we deal with tuples with many attributes.



Example

The same procedure with a tuple variable `bp`.

```
CREATE PROCEDURE joeGouge() AS
   CURSOR c IS
          SELECT beer, price
          FROM Sells
          WHERE bar = 'Joe''s bar';
   bp c%ROWTYPE;
  BEGIN
   OPEN c;
   LOOP
          FETCH c INTO bp;
          EXIT WHEN c%NOTFOUND;
          IF bp.price < 3.00 THEN
                 UDPATE Sells
                 SET price = bp.price + 1.00
                        WHERE bar = 'Joe''s Bar'
                        AND beer = bp.beer;
          END IF;
   END LOOP;
   CLOSE c;
  END;
.
run
```



SQL in Application Code

- 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)

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke 19

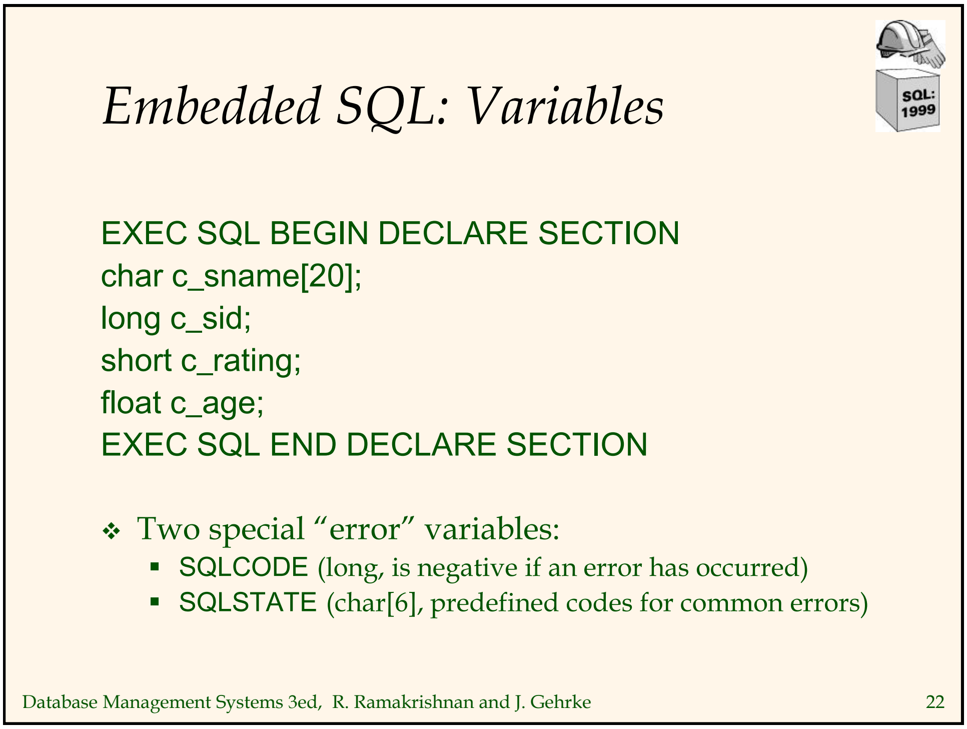
SQL in Application Code (Contd.)

Impedance mismatch:

- 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 such as C++. (Though now: STL)
  - SQL supports a mechanism called a *cursor* to handle this.

Embedded SQL

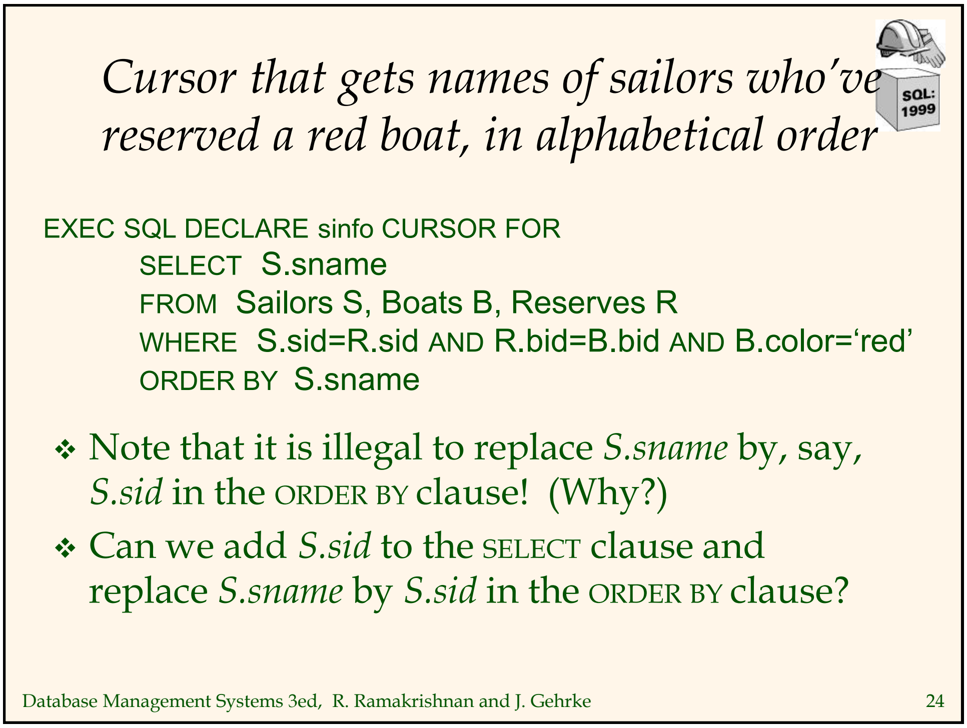
- Approach: Embed SQL in the host language.
  - A preprocessor converts the SQL statements into special API calls.
  - Then a regular compiler is used to compile the code.
- Language constructs:
  - Connecting to a database:
    EXEC SQL CONNECT
  - Declaring variables:
    EXEC SQL BEGIN (END) DECLARE SECTION
  - Statements:
    EXEC SQL Statement;



Embedded SQL: Variables

```
EXEC SQL BEGIN DECLARE SECTION
char c_sname[20];
long c_sid;
short c_rating;
float c_age;
EXEC SQL END DECLARE SECTION
```

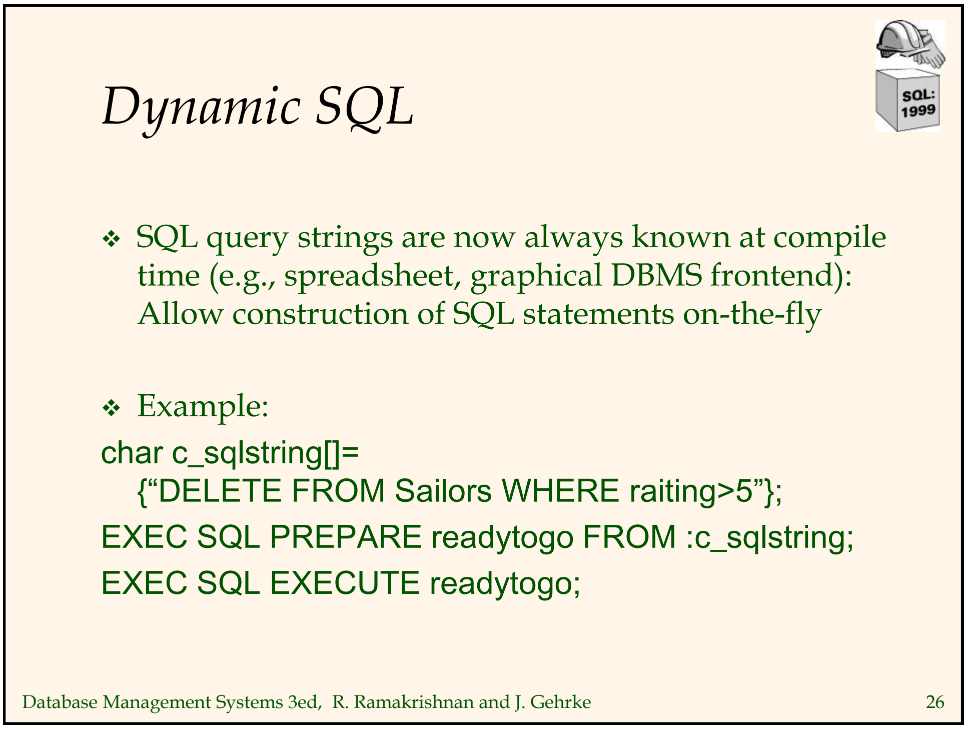
- Two special "error" variables:
  - SQLCODE (long, is negative if an error has occurred)
  - SQLSTATE (char[6], predefined codes for common errors)



Cursor that gets names of sailors who've reserved a red boat, in alphabetical order

```
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
```

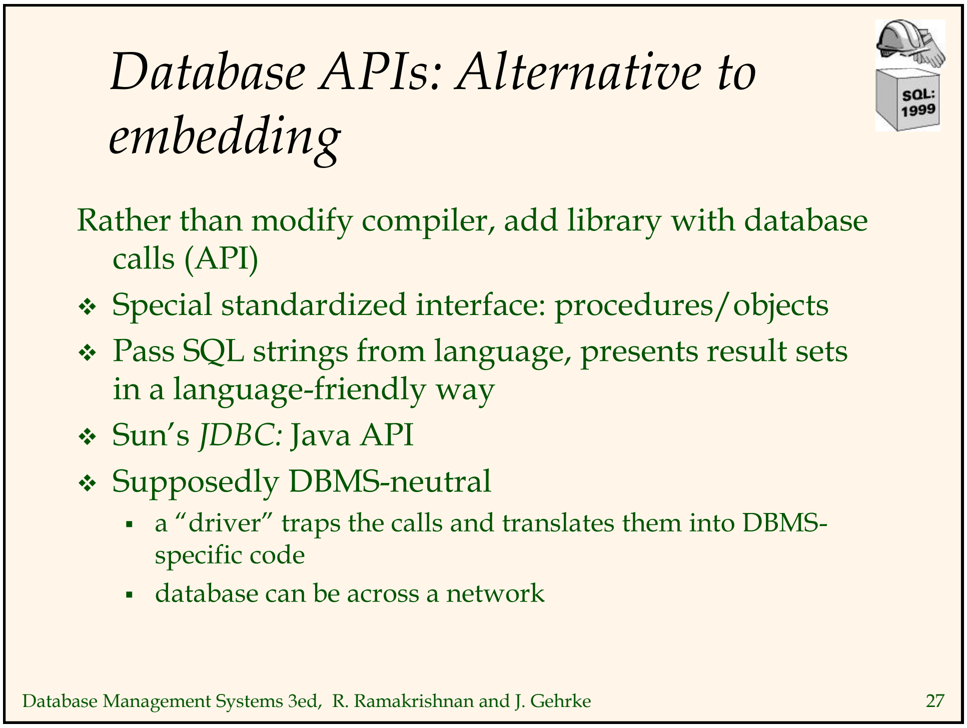
- Note that it is illegal to replace *S.sname* by, say, *S.sid* in the ORDER BY clause! (Why?)
- Can we add *S.sid* to the SELECT clause and replace *S.sname* by *S.sid* in the ORDER BY clause?



Dynamic SQL

- SQL query strings are now always known at compile time (e.g., spreadsheet, graphical DBMS frontend): Allow construction of SQL statements on-the-fly
- Example:

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



Database APIs: Alternative to embedding

Rather than modify compiler, add 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

JDBC: Architecture

- Four architectural 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 Architecture (Contd.)

Four types of drivers:

Bridge:

- Translates SQL commands into non-native API. Example: JDBC-ODBC bridge. Code for ODBC and JDBC driver needs to be available on each client.

Direct translation to native API, non-Java driver:

- Translates SQL commands to native API of data source. Need OS-specific binary on each client.

Network bridge:

- Send commands over the network to a middleware server that talks to the data source. Needs only small JDBC driver at each client.

Direction translation to native API via Java driver:

- Converts JDBC calls directly to network protocol used by DBMS. Needs DBMS-specific Java driver at each client.

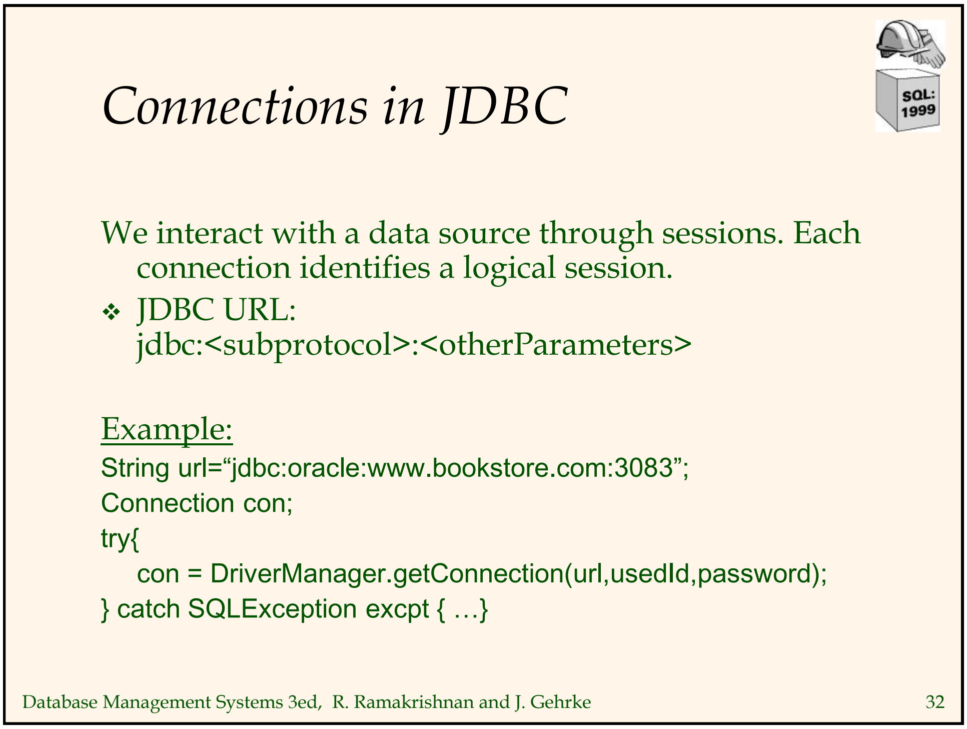
JDBC Classes and Interfaces

Steps to submit a database query:

- Load the JDBC driver
- Connect to the data source
- Execute SQL statements

JDBC Driver Management

- All drivers are managed by the DriverManager class
- Loading a JDBC driver:
  - In the Java code:
    Class.forName("oracle/jdbc.driver.Oracledriver");
  - When starting the Java application:
    -Djdbc.drivers=oracle/jdbc.driver



Connections in JDBC

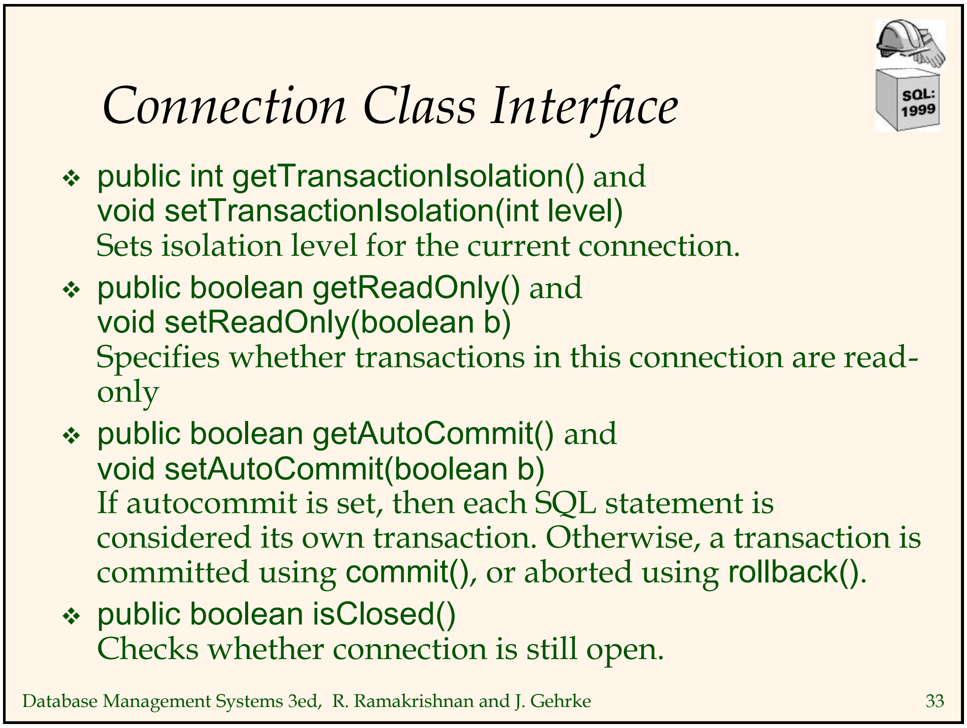
We interact with a data source through sessions. Each connection identifies a logical session.

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

Example:

```
String url="jdbc:oracle:www.bookstore.com:3083";
Connection con;
try{
    con = DriverManager.getConnection(url,usedId,password);
} catch SQLException excpt { …}
```

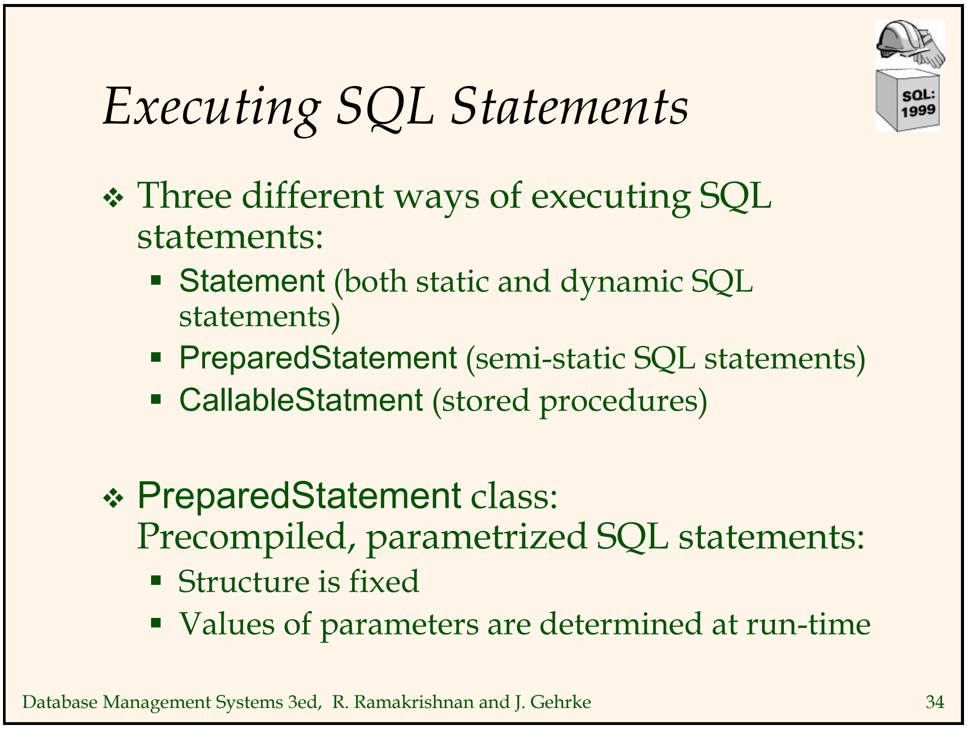
Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke 32



Connection Class Interface

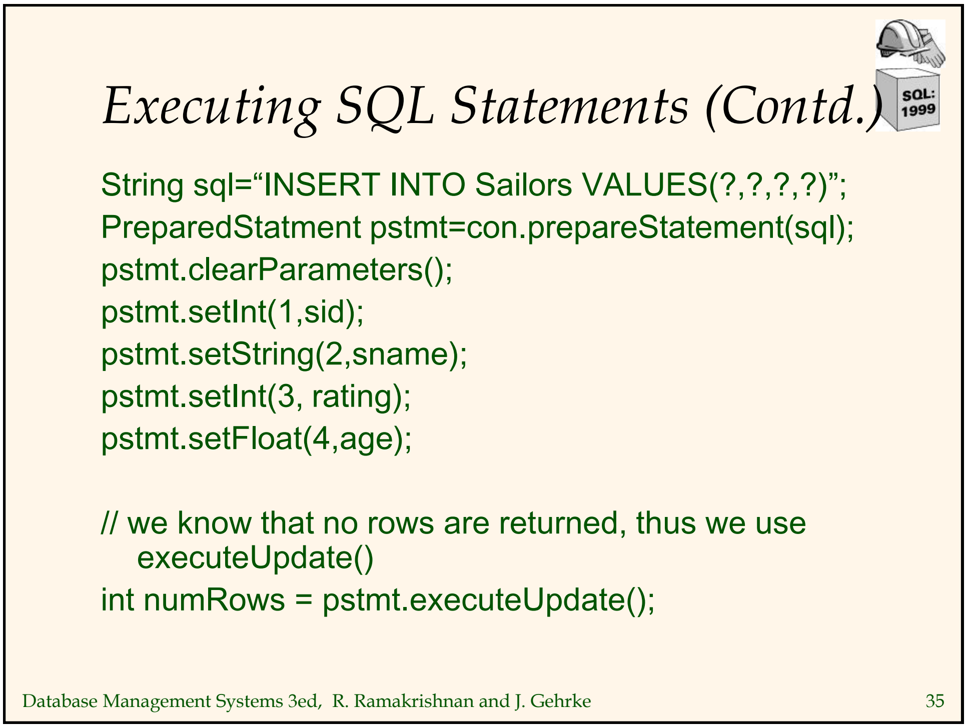
- public int getTransactionIsolation() and
  void setTransactionIsolation(int level)
  Sets isolation level for the current connection.
- public boolean getReadOnly() and
  void setReadOnly(boolean b)
  Specifies whether transactions in this connection are read-only
- public boolean getAutoCommit() and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

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke 33



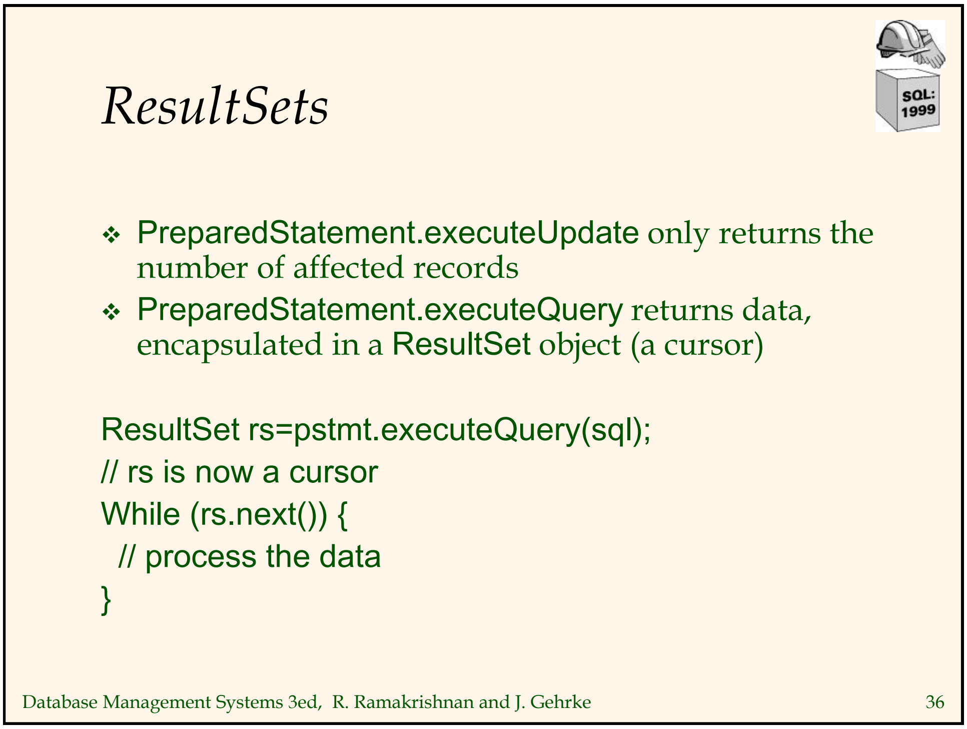
Executing SQL Statements

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



Executing SQL Statements (Contd.)

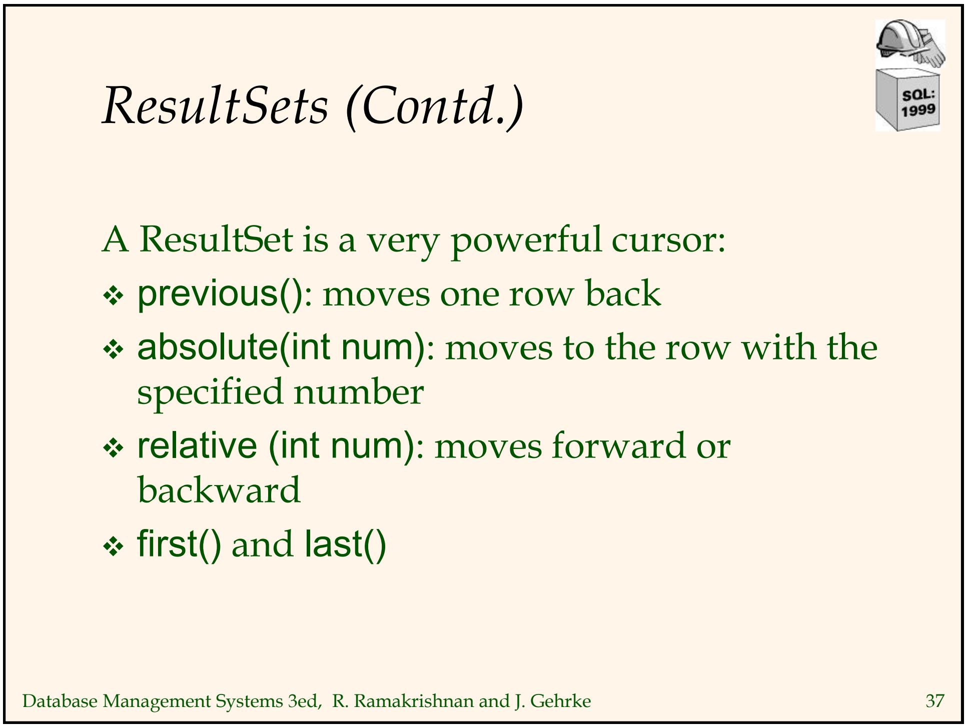
```
String sql="INSERT INTO Sailors VALUES(?,?,?,?)";
PreparedStatment pstmt=con.prepareStatement(sql);
pstmt.clearParameters();
pstmt.setInt(1,sid);
pstmt.setString(2,sname);
pstmt.setInt(3, rating);
pstmt.setFloat(4,age);

// we know that no rows are returned, thus we use
   executeUpdate()
int numRows = pstmt.executeUpdate();
```



ResultSets

- 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
}
```



ResultSets (Contd.)

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()

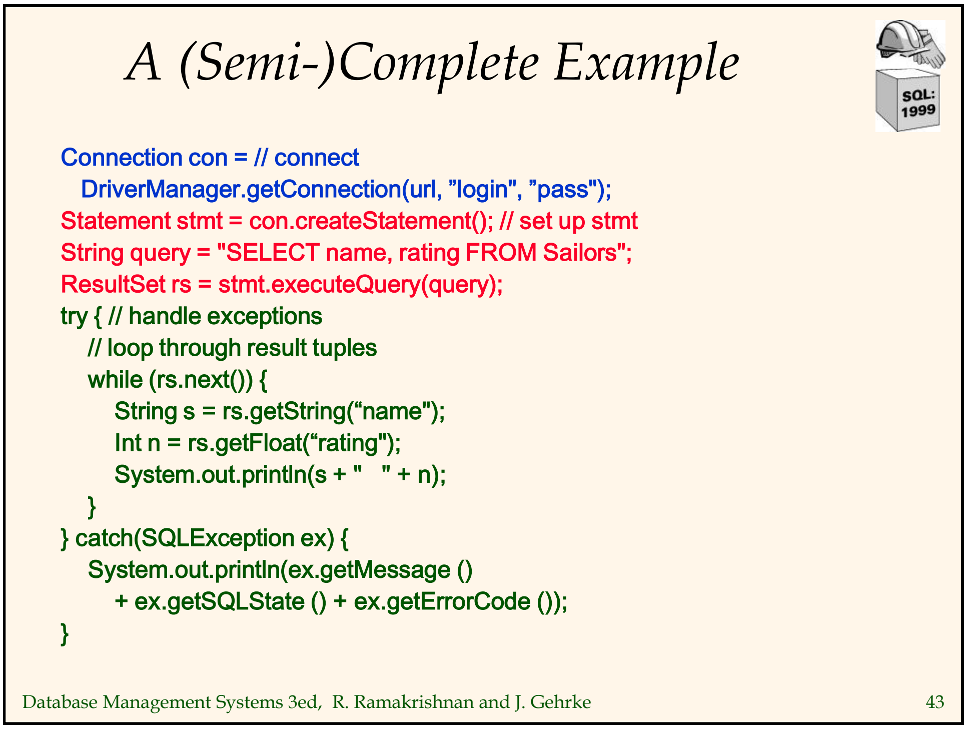
Examining Database Metadata

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());
```

Database Metadata (Contd.)

```
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” + “, “);
   }
}
```



A (Semi-)Complete Example

```
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 ());
}
```