Database Management Systems
Session 6

Instructor: Vinnie Costa
vcosta@optonline.net

Term Paper

- Due Saturday, Oct 8
- Should be about 3-4 pages (9 or 10 font)
- Most people have submitted topics

Homework

- Read Chapters Four and Five
- Any Questions?

MidTerm Exam - #1

Explain the difference between external, internal, and conceptual schemas. How are these different schema layers related to the concepts of logical and physical data independence? Explain the difficulties around external views, particularly with updateable views. (25 pts)

MidTerm Exam - #1

- External schemas allows data access to be customized (and authorized) at the level of individual users or groups of users. Conceptual (logical) schemas describes all the data that is actually stored in the database. While there are several views for a given database, there is exactly one conceptual schema to *all* users. Internal (physical) schemas summarize how the relations described in the conceptual schema are actually stored on disk (or other physical media). External schemas provide logical data independence, while conceptual schemas offer physical data independence.

MidTerm Exam - #1

- A **view** is just a relation, but we store a **definition**, rather than a set of tuples

```
CREATE  VIEW  GoodStudents (sid, gpa)
        AS  SELECT S.sid, S.gpa
            FROM   Students S
            WHERE  S.gpa > 3.0
```

- SQL-92 standard allows updates to be specified only on views that are defined on a single base table using just selection and projection, with no use of aggregate operations. Such views are called updateable views.
- Update on a view affects the underlying table!
- Section 3.6.2 in the text (p.88)

MidTerm Exam - #2

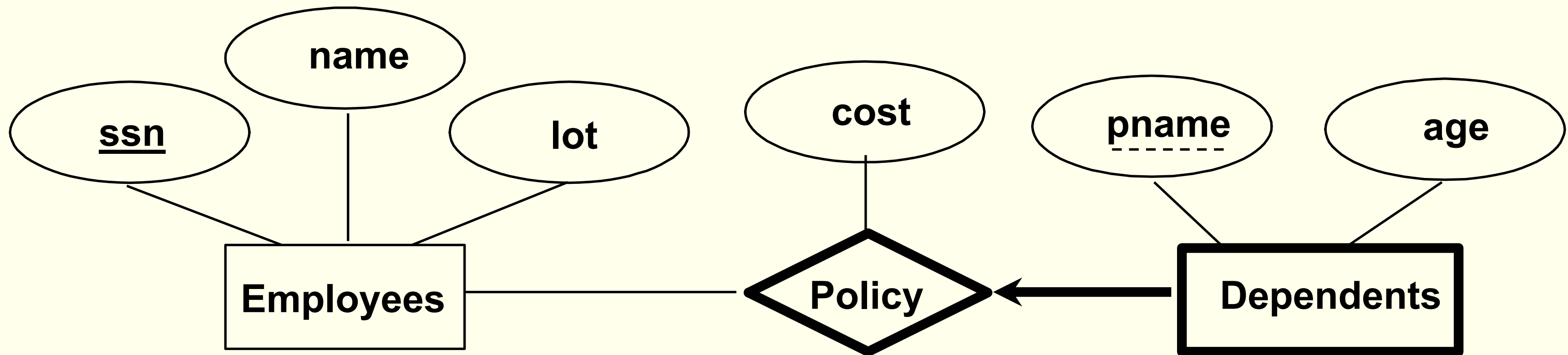
Define the following: *weak entity set, a partial key, participation constraint.*
Draw an ER Diagram that illustrates the use of these constraints on the Employess, Policy, and Dependents entity and realtionship sets discussed in class. (25 pts)

MidTerm Exam - #2

- *Weak entity set* - an entity that cannot be identified uniquely without considering some primary key attributes of another identifying owner entity. An example is including Dependent information for employees for insurance purposes.
- *Partial key* – the set of attributes of a weak entity set that uniquely identify a weak entity for a given owner entity. We indicate a partial key by underlining with a broken line.
- *Participation constraint* - a participation constraint determines whether relationships must involve certain entities. An example is if every department entity has a manager entity. Participation constraints can either be total or partial. A total participation constraint says that every department has a manager. A partial participation constraint says that every employee does not have to be a manager.

MidTerm Exam - #2

- A **weak entity** can be identified uniquely only by considering the primary key of another (**owner**) entity.
- Owner entity set and weak entity set must participate in a one-to-many relationship set (one owner, many weak entities).
- Weak entity set must have total participation in this **identifying relationship** set.
- pname is a **partial key** for the weak entity set
- Dependents is a weak entity and Policy is its identifying relationship. This is indicated by a **thick black line**

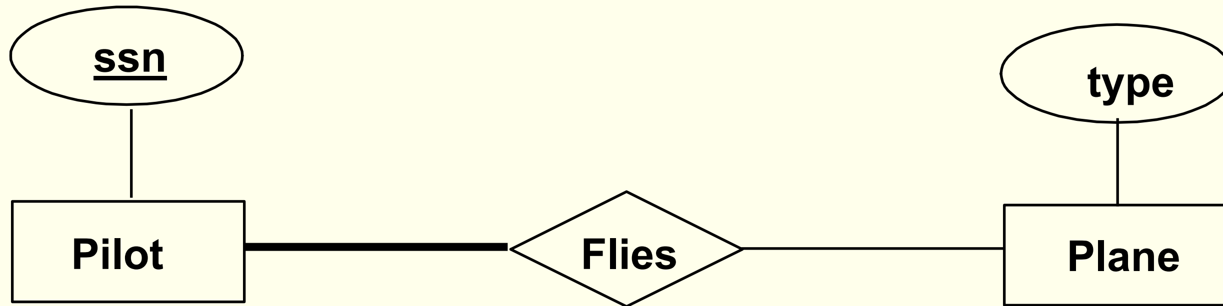


MidTerm Exam - #3

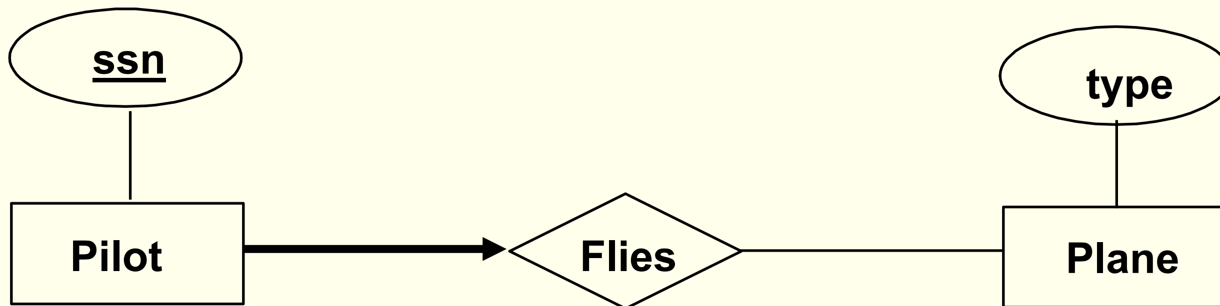
PapaCosta Airlines has a database that contains information about its Pilots (identified by social security number, or SSN) and Planes (identified by type). The plane types can be *single engine*, *multi-engine*, and *jet*. Pilots fly planes; the following situations concern the Flies relationship set. For each situation, draw an ER diagram that describes it (assuming no further constraints hold). (25 pts)

1. Every pilot must fly some plane.
2. Every pilot flies exactly one type plane (no more, no less)

MidTerm Exam - #3



Every pilot must fly some plane



Every pilot flies exactly one type plane (no more, no less)

MidTerm Exam - #4

Consider the SQL query whose answer is shown in Table 1. (25 pts)

1) Modify this query so that only the *name* and *login* columns are included in the answer

2) If the clause `WHERE S.gpa >= 1.9` is added to the original query, what is the set of tuples in the answer?

sid	*name*	*login*	*age*	*gpa*
53831	Madayan	madayan@music	11	1.8
53832	Guldu	guldu@music	12	2

Table 1: Student with *age* < 18 on Instance S

MidTerm Exam - #4

1. Only *name* and *login* are included in the answer:

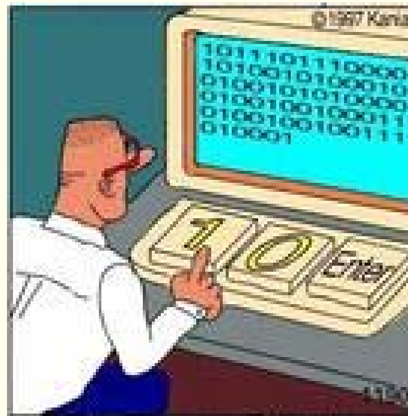
```
SELECT S.name, S.login
FROM Students S
WHERE S.age < 18
```

2. The answer row for Madayan is omitted then.

World Wide Web Consortium

- The World Wide Web Consortium (W3C) is an international consortium where member organizations, a full-time staff, and the public work together to develop Web standards
- Tim Berners-Lee and others created W3C as an industry consortium dedicated to building consensus around Web technologies. Mr. Berners-Lee, who invented the World Wide Web in 1989 while working at the European Organization for Nuclear Research (CERN), has served as the W3C Director since W3C was founded, in 1994
- The place to check for standards!

Database Application Development



Chapter 6

Overview

Concepts covered in this lecture:

- SQL in application code
- Embedded SQL
- Cursors
- Dynamic SQL
- JDBC
- SQLJ
- Stored procedures

Lost In Translation

- SQL is a powerful language but specific to DBMS
- Result of a query is can be a set of rows that come crashing down like a big wave
- Application languages very flexible but no data structure to handle rows and rows of query results
- Mismatch resolved through additional SQL constructs
- Obtain a handle on a collection and iterate over it one record at a time

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.
- <u>Two main integration approaches:</u>
  - Embed SQL in the host language (Embedded SQL, SQLJ)
  - Create special API to call SQL commands (JDBC)

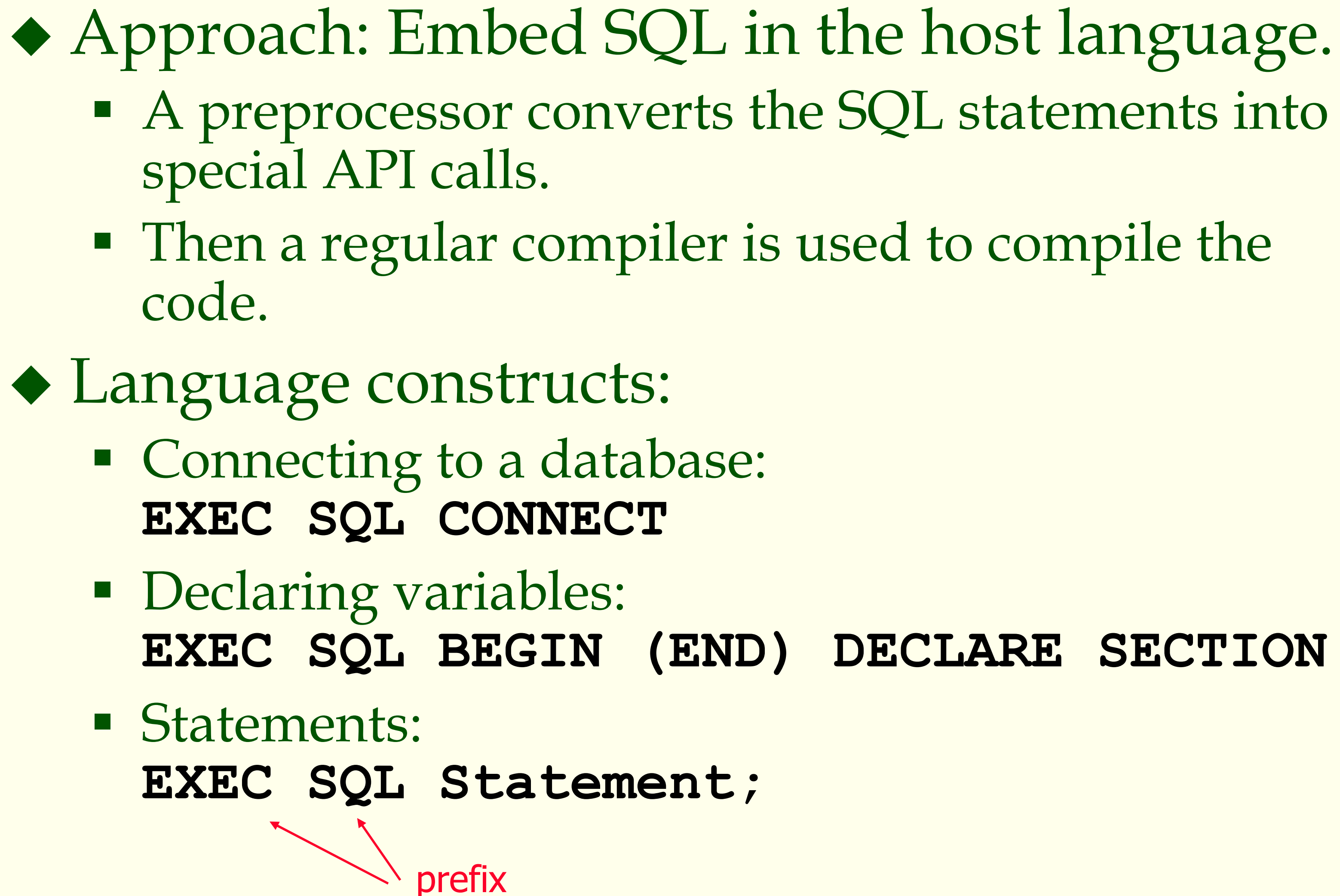
SQL in Application Code (Contd.)

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.

(‡) **Standard Template Library (STL)** is a C++ library of container classes, algorithms, and iterators; it provides many of the basic algorithms and data structures of computer science

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)

Cursors

- Can declare a **cursor** on a relation or query statement (which generates a relation)
- Can **open** a cursor, and repeatedly **fetch** a row then **move** the cursor, until all rows 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 rows are returned
    - Fields in **`ORDER BY`** clause must also appear in **`SELECT`** clause.
  - The **`ORDER BY`** clause, which orders answer rows, is *only* allowed in the context of a cursor.
- Can also modify/delete row pointed to by a cursor.

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
```

must be the same; otherwise, wouldn't know what to sort on

- 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?

Embedding SQL in C: An Example

```
char SQLSTATE[6];
EXEC SQL BEGIN DECLARE SECTION
char c_sname[20]; short c_minrating; float c_age;
EXEC SQL END DECLARE SECTION
c_minrating = random();
EXEC SQL DECLARE sinfo CURSOR FOR
   SELECT S.sname, S.age
   FROM Sailors S
   WHERE S.rating > :c_minrating
   ORDER BY S.sname;
do {
   EXEC SQL FETCH sinfo INTO :c_sname, :c_age;
   printf("%s is %d years old\n", c_sname, c_age);
} while (SQLSTATE != '02000');
EXEC SQL CLOSE sinfo;
```

Dynamic SQL

- SQL query strings are not 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 rating>5”};
EXEC SQL PREPARE readytogo FROM :c_sqlstring;
EXEC SQL EXECUTE readytogo;
```

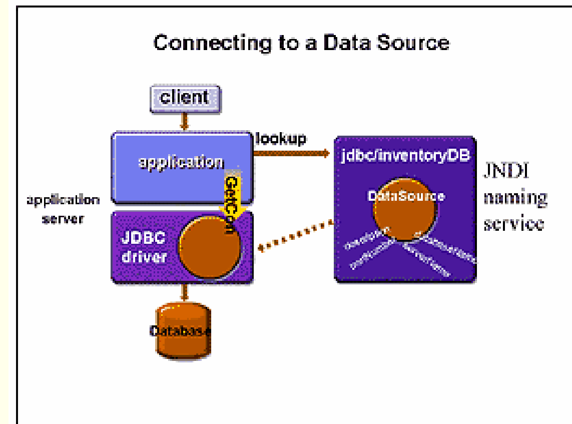
Database APIs: Alternative to embedding

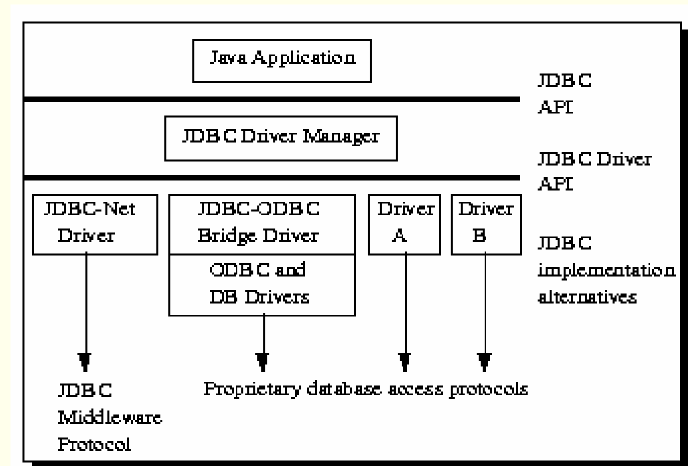
- Rather than modify compiler, add library with database calls (API)
- Special standardized interface: procedures/objects
- Pass SQL strings from language, presents result sets in a language-friendly way
- Sun's **JDBC**: Java API
- Supposedly **DBMS-neutral**
  - a "driver" traps the calls and translates 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 (Type 1)

- 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 (Type 2)

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

Network bridge (Type 3)

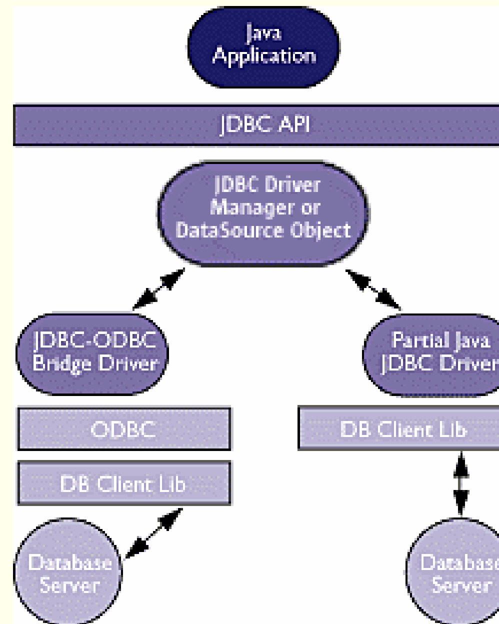
- Send commands over the network to a middleware server that talks to the data source. Needs only small JDBC driver at each client.

Direction translation to native API via Java driver (Type 4)

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

JDBC Architecture

Type 1: JDBC-ODBC Bridge plus ODBC Driver
This combination provides JDBC access via ODBC drivers. ODBC binary code -- and in many cases, database client code -- must be loaded on each client machine that uses a JDBC-ODBC Bridge.

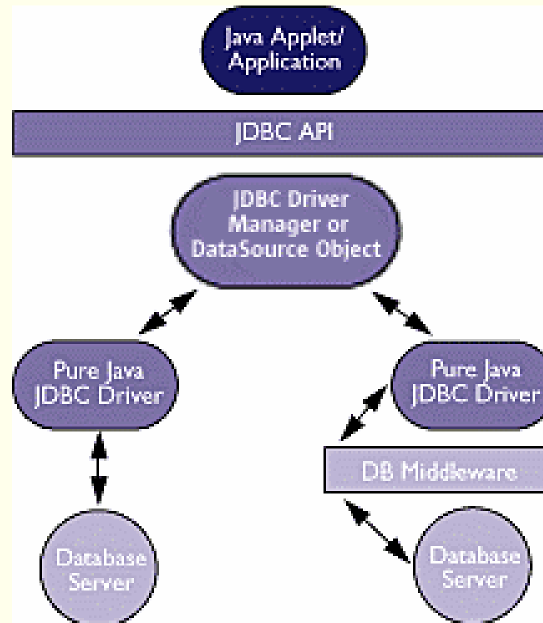


Type 2: A native API partly Java technology-enabled driver
This type of driver converts JDBC calls into calls on the client API for Oracle, Sybase, Informix, DB2, or other DBMS. Note that, like the bridge driver, this style of driver requires that some binary code be loaded on each client machine.

http://java.sun.com/products/jdbc/overview.html

JDBC Architecture

Type 4: Direct-to-Database Pure Java Driver
This style of driver converts JDBC calls into the network protocol used directly by DBMSs, allowing a direct call from the client machine to the DBMS server and providing a practical solution for intranet access.



Type 3: Pure Java Driver for Database Middleware
This style of driver translates JDBC calls into the middleware vendor's protocol, which is then translated to a DBMS protocol by a middleware server. The middleware provides connectivity to many different databases.

http://java.sun.com/products/jdbc/overview.html

JDBC Classes and Interfaces

JDBC is a collection of Java classes and interfaces that enable database access from programs written in the Java language

Basic steps to submit a database query and retrieve results:

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

JDBC Driver Management

- All drivers are managed by the **`DriverManager`** class
- Has methods for dynamic addition and deletion of drivers
- Loading a JDBC driver:
  - In the Java code:
    ```
    Class.forName("oracle/jdbc.driver.Oracledriver");
    ```
  - When starting the Java application (command line):
    ```
    -Djdbc.drivers=oracle/jdbc.driver
    ```

Connections in JDBC

- We interact with a data source through **sessions**. Each connection identifies a logical session with a data source
- Session is started by creating a **`Connection`** object
- Specified through a **JDBC URL**:
  `jdbc:<subprotocol>:<otherParameters>`

Example:

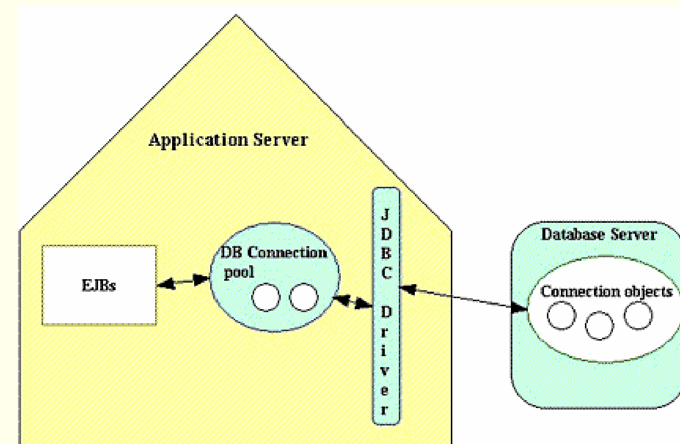
```
String url="jdbc:oracle:www.bookstore.com:3083";
Connection connection;
try{
   connection =
   DriverManager.getConnection(url,usedId,password);
} catch SQLException excpt { …}
```

Connection Class Interface (**Methods**)

- **`public int getTransactionIsolation() and void setTransactionIsolation(int level)`**
  Sets isolation level (5 SQL levels) for the current connection.
- **`public boolean getReadOnly() and void setReadOnly(boolean readOnly)`**
  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.

Connection Pooling

- Connection to a data source is a costly operation – network, authentication, memory
- Many different connections are often **pooled** – web servers open many connections
- **Connection pool** is a set of established connection to a data source
- Handled by optional **`javax.sql`** package
- Define capacity, shrinkage and growth rate
- In most app servers!

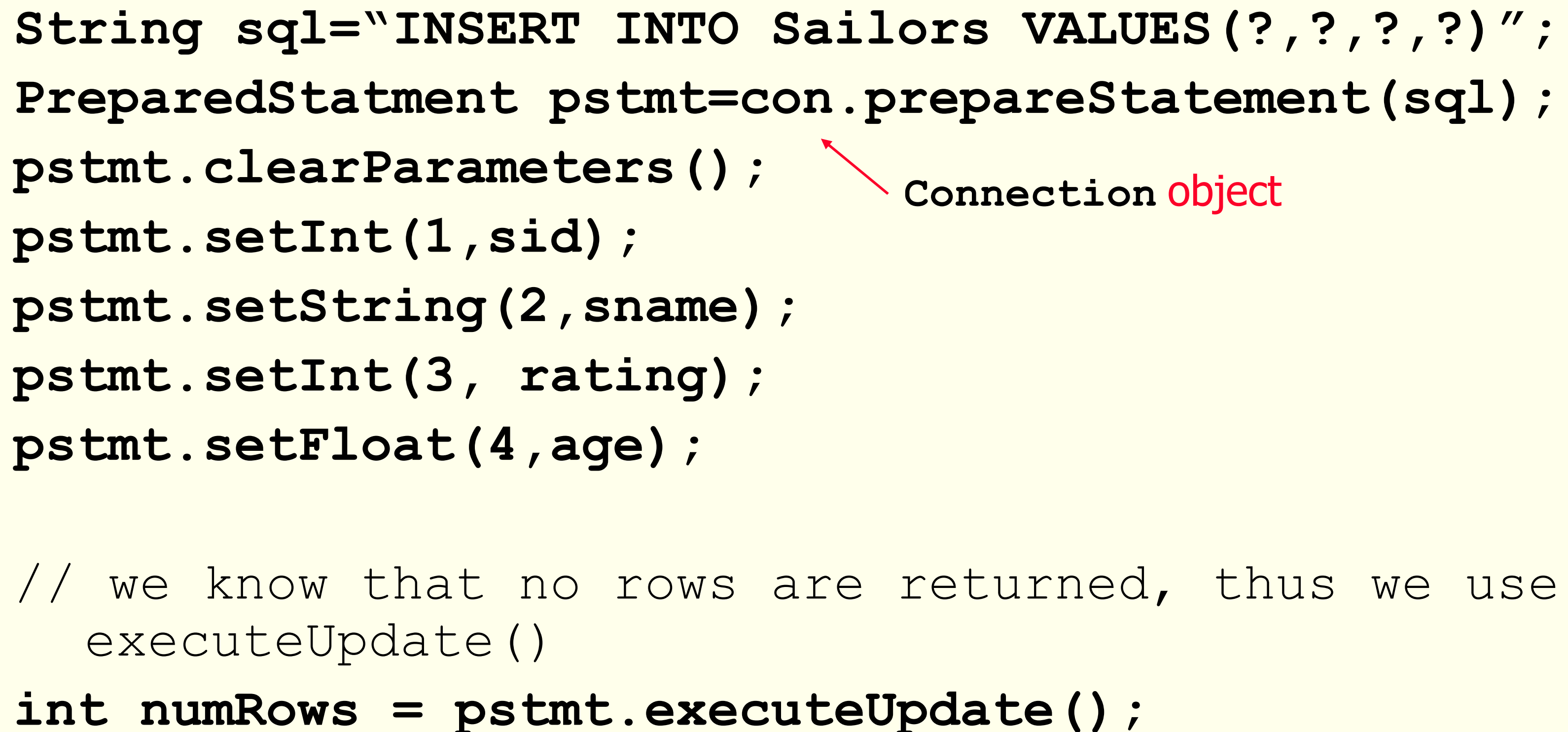


Executing SQL Statements

- 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, parameterized SQL statements:
  - Structure is fixed
  - Values of parameters are determined at run-time

Executing SQL Statements (Contd.)



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
}
```

`next()` method returns **`false`** if there are no more rows in the query answer, **`true`** otherwise

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()`**

Matching Java and SQL Data Types

SQL Type	Java class	ResultSet 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: Exceptions and Warnings

- Most of **`java.sql`** can throw and **`SQLException`** if an error occurs.
- **`SQLWarning`** is a subclass of **`SQLException`**; not as severe (they are not thrown and their existence has to be explicitly tested)

Warning and Exceptions (Contd.)

```
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
}
```

`getWarnings` method of `Connection`

get the next exception

Examining Database Metadata

`DatabaseMetaData` object gives information about the database system and the catalog (over 100 methods)

```
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"
                                 + ", ");
    }
}
```

SQLJ – SQL-Java

- **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 sailors = {
    SELECT sid, sname INTO :sid, :name
    FROM Sailors WHERE rating = :rating
};
```

SQLJ

- Compare to JDBC:

```
//assume we have a ResultSet cursor for rs
sid=rs.getInt(1);
if (sid==1) {
   sname=rs.getString(2);
}
else {
   sname2=rs.getString(2);
}
//we dynamically decide which host language variables
will hold the query result
```

- When writing SQLJ applications, we just write regular Java code and embed SQL statements according to a set of rules
- Important philosophical difference exist between Embedded SQL and SQLJ and JDBC

SQLJ

- When using Embedded SQL, it is tempting to use vendor-specific SQL contructs that offer functionality beyond the SQL-92 or SQL:1999 standards
- SQLJ and JDBC force adherence to the standards, and the resulting code is much more portable across different database systems

Writing 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
// execute the query and open the cursor
#sql 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

- What is a **stored procedure**?
  - Program executed through a single SQL statement
  - Locally 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 row-at-a-time return of records through cursors

Stored Procedures: 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
```

Stored procedures 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
```

Stored Procedures: Examples (Contd.)

- Stored procedure do not have to be written in SQL:

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

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 SQL CALL IncreaseRating(:sid,:rating);
```

Calling Stored Procedures (Contd.)

the calling subclass

JDBC:

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

SQLJ:

```
#sql iterator ShowSailors(…);
ShowSailors showsailors;
// call the stored procedure
#sql showsailors={CALL ShowSailors};
while (showsailors.next()) {
    …
}
```

SQL/PSM – Persistent Stored Modules

- Most DBMSs allow users to write stored procedures in a simple, general-purpose language (close to SQL)
- SQL/PSM standard is a representative of most vendor specific languages

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;
```

Main SQL/PSM Constructs

```
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;
```

Main SQL/PSM Constructs (Contd.)

- 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 parametrized 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

Summary (Contd.)

- **SQLJ**: Static model, queries checked a compile-time.
- **Stored procedures** execute application logic directly at the server
- **SQL/PSM** standard for writing stored procedures

Useful Websites

- http://java.sun.com/products/jdbc/ - the JDBC home page
- http://java.sun.com/j2se/1.4.2/docs/guide/jdbc/getstart/GettingStartedTOC.fm.html – *JDBC API Tutorial and Reference*
- http://java.sun.com/products/jdbc/overview.html – A brief overview
- http://www.php-mysql-tutorial.com/connect-to-mysql-using-php.php - This is where you start to put PHP and MySQL together. This page explains how to open and close MySQL connection with PHP.

Homework

- Read Chapter Six