JACOBS UNIVERSITY

# Database Application Development

Ramakrishnan & Gehrke, Chapter 6

# SQL Integration Approaches

JACOBS UNIVERSITY

- Create special API to call SQL commands
  - API = application programming interface
  - JDBC, PHP
- Embed SQL in the host language = extend language
  - Embedded SQL, SQLJ
- Move (part of) application code into database
  - Stored procedures, object-relational extensions, …

# Database APIs: A Coder Perspective

JACOBS UNIVERSITY

- Like in a PL: DB access = call to library function
  - Input: SQL string
  - Output: table
    - *…hm…data structure? Should be language-friendly!*
- Supposedly DBMS-neutral through encapsulating classes
  - "driver" translates into DBMS-specific code
- Ex:
  - PHP: "Private Home Page" -> "PHP Hypertext Processor"
  - JDBC: Java SQL API (Sun Microsystems)
    - *cf. ODBC by Microsoft*

# Overview

- SQL API
  - Example 1: PHP
  - Example 2: JDBC
- Embedded SQL
  - Basics; Cursors; Dynamic SQL – based on Example 1: C
  - Example 2: SQLJ
- Stored procedures

# PHP and (My)SQL

www.php.net

JACOBS UNIVERSITY

- PHP calls embedded within HTML as special tag
  - <?php *php-statement-sequence* ?>
- Execution (server-side!) of PHP:
- PHP statements → (HTML) text; complete file forwarded by Web server:

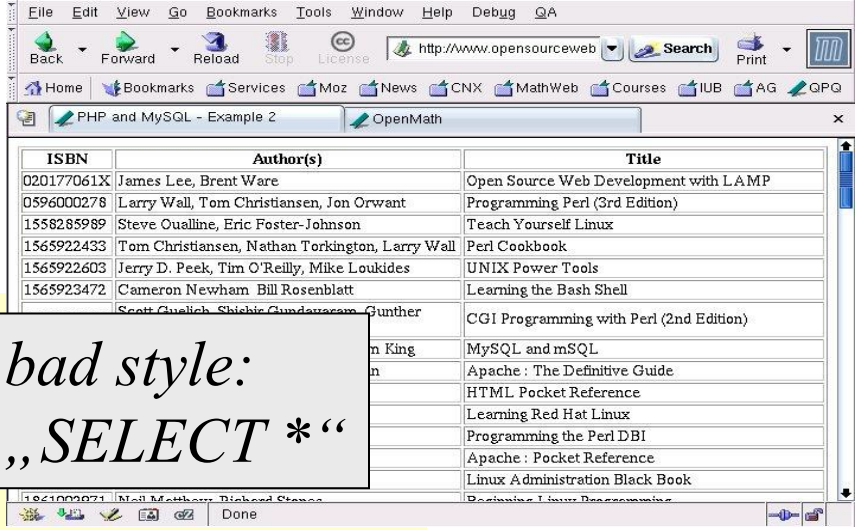
<h1><?php echo “Hello World”; ?></h1> → <h1>Hello World</h1>

- Example: connecting to mysql server on localhost

```
<?php
   $mysql = mysql_connect( “localhost”, “apache”, “DBWAisCool” )
      or die( “cannot connect to mysql” );
?>
```

variables have „$“ prefix

# PHP, HTML, and (My)SQL




*bad style: „SELECT *“*

```
<html>
  <head>
    <title>PHP and MySQL Example</title>
  </head>
  <body>
   <?php $mysql = mysqli_connect( “localhost” );
     $result = mysql_db_query( “books”, “SELECT isbn, author, title FROM book_info" )
        or die( “query failed - “ . mysql_errno() . “: “ . mysql_error(); )
    ?>
    <table>
      <tr>  <th>ISBN</th>  <th>Author(s)</th>  <th>Title</th>  </tr>
      <?php while ( $array = mysql_fetch_array($result) ); ?>
      <tr><td><?php echo $array[ "isbn"     ]; ?></td>
            <td><?php echo $array[ "author" ]; ?></td>
            <td><?php echo $array[ "title"    ]; ?></td>
      </tr>
      <?php endwhile; ?>
    </table>
    <?php mysql_close($mysql); ?>
  </body>
</html>
```

open

query

iterate over result set

close

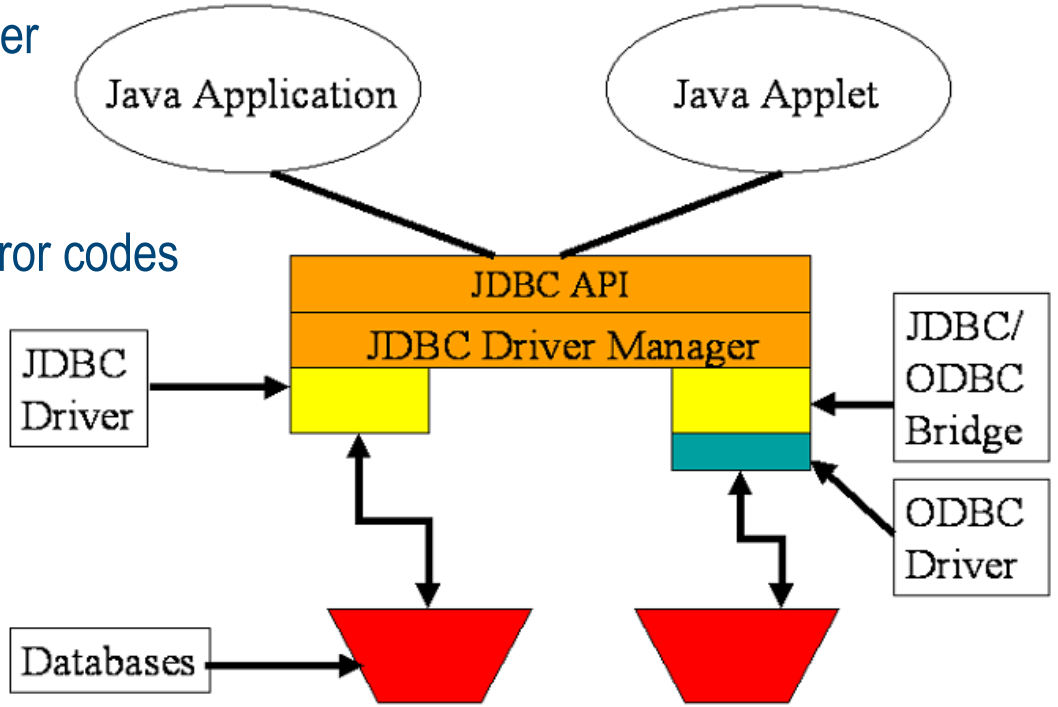
# Overview

- SQL API
  - Example 1: PHP
  - Example 2: JDBC
- Embedded SQL
  - Basics; Cursors; Dynamic SQL – based on Example 1: C
  - Example 2: SQLJ
- Stored procedures

not in exams

# JDBC: Architecture

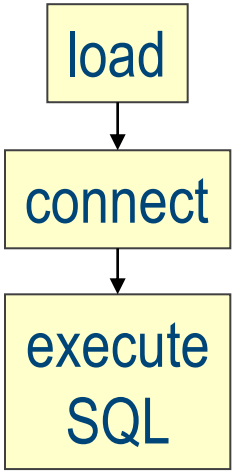
- Four architectural components:
  - Application: initiates / terminates connections, submits SQL statements
  - Driver manager: load JDBC driver
  - Driver: connects to data source, transmits requests, returns/translates results and error codes
  - Data source: processes SQL statements



not in exams

# JDBC Classes and Interfaces

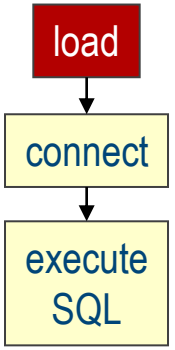
- Steps to submit a database query:
- Load the JDBC driver
- Connect to the data source
- Execute SQL statements



not in exams

# JDBC Driver Management

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

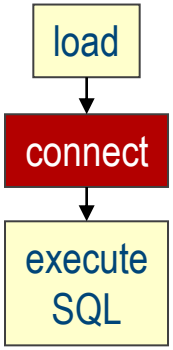


not in exams

# Connections in JDBC

JACOBS UNIVERSITY

- interact with data source through sessions
  - Each connection identifies a logical session
- Service identified through JDBC URL:
  jdbc:<subprotocol>:<otherParameters>
- Example:

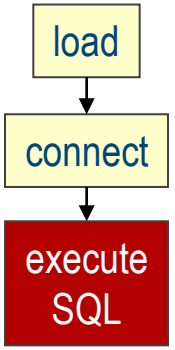



```
String url = "jdbc:oracle:www.bookstore.com:3083";
Connection con = DriverManager.getConnection( url, userId, password );
```

not in exams

# Executing SQL Statements

- Ways of executing SQL statements:
  - Static: complete query known at compile time
  - Prepared: precompiled, but parametrized
  - Dynamic: SQL string composed at runtime
  - Stored procedure: invoke query stored in server (later more)
- JDBC classes:
  - Statement (static and dynamic SQL statements)
  - PreparedStatement (semi-static SQL statements)
  - CallableStatement (stored procedures)



not in exams

# Prepared Statement: Example

```
String sql = "INSERT INTO Sailors VALUES(?,?,?,?)";
PreparedStatement pstmt=con.prepareStatement( sql );

pstmt.clearParameters();            // reset parameter list
pstmt.setInt( 1, sid );             // set attr #1 to value of sid
pstmt.setString( 2, sname );        // set attr #2 to sname
pstmt.setInt( 3, rating );          // set attr #3 to rating
pstmt.setFloat( 4, age );           // set attr #4 to age

// INSERT belongs to the family of UPDATE operations
// (no rows are returned), thus we use executeUpdate()
int numRows = pstmt.executeUpdate();
```

- Two methods for query execution:
  - PreparedStatement.executeUpdate() returns *number* of affected records
  - PreparedStatement.executeQuery() returns *data*

# ResultSets

not in exams



- Class ResultSet (aka cursor) for returning data to application

```
ResultSet rs = pstmt.executeQuery( sql );          // rs is a cursor
while ( rs.next() )
{
    System.out.println( rs.getString("name") + " has rating " + rs.getDouble("rating") );
}
```

- …but 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() moves to first or last row, resp.

not in exams

# JDBC: Error Handling

JACOBS UNIVERSITY

- Most of java.sql can throw an SQLException if an error occurs

```
try
{   rs = stmt.executeQuery(query);
    while (rs.next())
        System.out.println( rs.getString("name") + " has rating " + rs.getDouble("rating") );
}
catch (SQLException ex)
{   System.out.println( ex.getMessage () + ex.getSQLState () + ex.getErrorCode () );
}
```

- SQLWarning: subclass of SQLException not as severe
  - not thrown, existence has to be explicitly tested:

```
con.clearWarnings();
stmt.executeUpdate( queryString );
if (con.getWarnings() != null)
    /* handle warning(s) */;
```

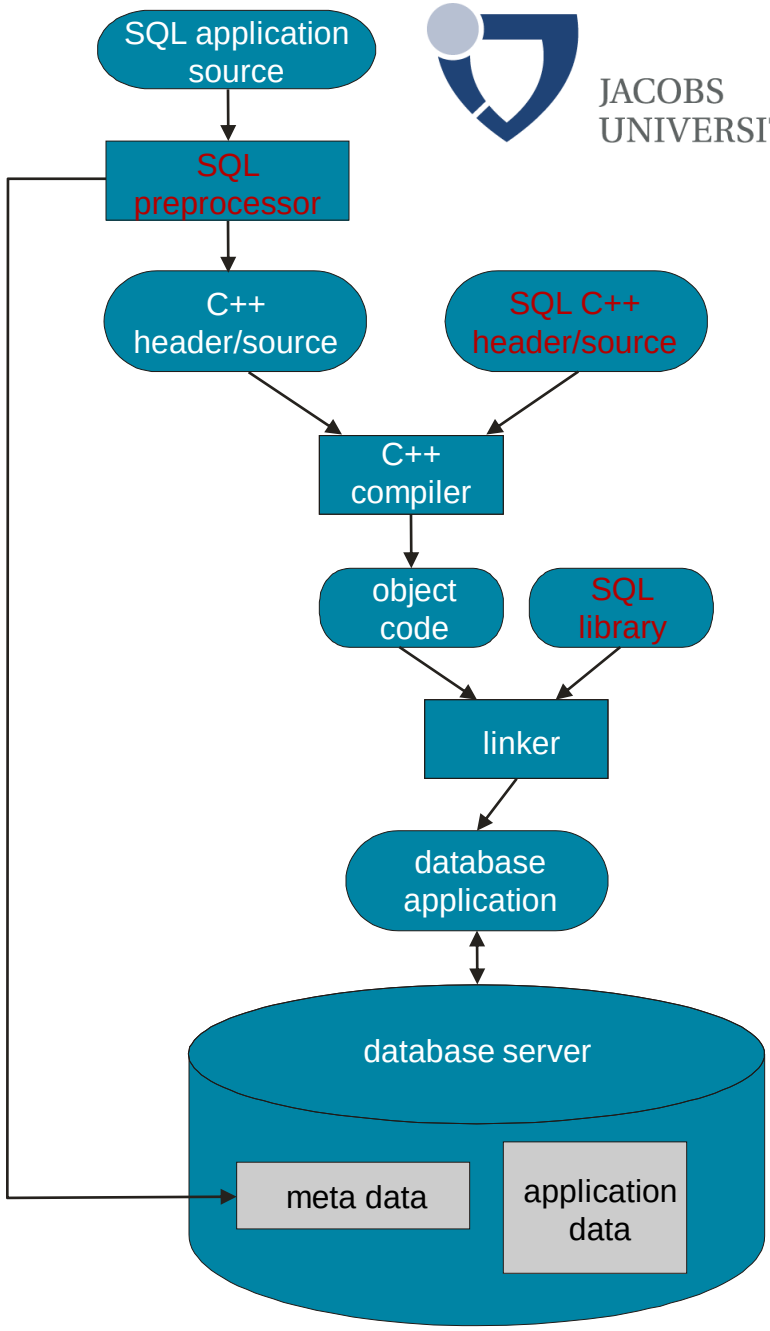
# Overview

- SQL API
  - Example 1: PHP
  - Example 2: JDBC
- Embedded SQL
  - Basics; Dynamic SQL
  - Example 2: SQLJ
- Stored procedures

not in exams

# Embedded SQL

- Approach: *make SQL statements part of host language*
  - Seems like language extention, but isn't
- Steps:
  - preprocessor converts SQL statements into sequences of API calls
    - *Source-to-source*
  - vanilla compiler for generating code
  - link code with vendor-supplied library
  - See www.knosof.co.uk/sqlport.html for tech details & issues



not in exams

# Embedded SQL Language Constructs

JACOBS UNIVERSITY

- Connecting to a database:
  - EXEC SQL CONNECT
- Declaring variables:
  - EXEC SQL BEGIN DECLARE SECTION
    …
    EXEC SQL END DECLARE SECTION
- Statements:
  - EXEC SQL Statement

```
EXEC SQL include sqlglobals.h;
EXEC SQL include "externs.h“

EXEC SQL BEGIN DECLARE SECTION;
        long rasver1;
        long schemaver1;
       char *myArchitecture = RASARCHITECTURE;
EXEC SQL END DECLARE SECTION;

EXEC SQL SELECT ServerVersion, IFVersion
       INTO :rasver1, :schemaver1
       FROM RAS_ADMIN
       WHERE Architecture = :myArchitecture;
if (SQLCODE != SQLOK)
{      if (SQLCODE == SQLNODATAFOUND) …;
}
```

not in exams

# 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:
  - long SQLCODE – set to negative value if error has occurred
  - char[6] SQLSTATE – error codes in ASCII

# Cursors

- Problem: How to iterate over result sets
  when procedural languages do not know “sets”?
- Cursor = aka generic iterator (C++, Java, python, …)
  - on relation, or query statement generating a result relation
- Can open cursor,
  and repeatedly fetch a tuple then move the cursor,
  until all tuples have been retrieved
- Ex:

```
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
```

not in exams

# Embedding SQL in C: An Example

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

c_minrating = random();          /* just for fun */

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;
        if ( SQLCODE == 0 )
                printf("%s is %d years old\n", c_sname, c_age);
} while ( SQLCODE >= 0 );
EXEC SQL CLOSE sinfo;
```

*Note ":" prefix!*
*Precompiler needs hint to distinguish program from SQL variables*

# Overview

- SQL API
  - Example 1: PHP
  - Example 2: JDBC
- Embedded SQL
  - Basics; Cursors; Dynamic SQL – based on Example 1: C
  - Example 2: SQLJ
- Stored procedures

not in exams

# SQLJ

JACOBS UNIVERSITY

- SQLJ = Java + embedded JDBC database access, nicely wrapped
  - ISO standard
  - eliminates JDBC overhead
    → compact & elegant database code, less programming errors
- SQLJ program ----[ SQLJ translator ]----> std Java source code
  - embedded SQL statements → calls to SQLJ runtime library
- (semi-) static query model: Compiler does
  - syntax checks, strong type checks
  - consistency wrt. schema
- Primer: http://archive.devx.com/dbzone/articles/sqlj/sqlj02/sqlj012102.asp

not in exams

# SQLJ Code Example

```
Int sid; String name; Int rating;
#sql iterator Sailors( Int sid, String name, Int rating );
Sailors sailors;

#sql sailors =
   { SELECT sid, sname INTO :sid, :name FROM Sailors WHERE rating = :rating };

while (sailors.next())
{    System.out.println( sailors.sid + ": " + sailors.sname) );
}

sailors.close();
```

not in exams

# SQLJ vs. JDBC

JACOBS UNIVERSITY

```
String vName; int vSalary; String vJob;
Java.sql.Timestamp vDate;

...

#sql { SELECT Ename, Sal
       INTO :vName, :vSalary
       FROM Emp
       WHERE Job = :vJob and HireDate = :vDate };
```

simplified:
no result set iteration

```
String vName; int vSalary; String vJob;
Java.sql.Timestamp vDate;
...
PreparedStatement stmt =
   connection.prepareStatement(
       "SELECT Ename, Sal "      +
       "INTO :vName, :vSalary "  +
       "FROM Emp "               +
       "WHERE Job = :vJob and HireDate = :vDate");

stmt.setString(1, vJob);
stmt.setTimestamp(2, vDate);

ResultSet rs = stmt.executeQuery();
rs.next();

vName   = rs.getString(1);
vSalary = rs.getInt(2);

rs.close();
```

not in exams

# SQLJ Iterators

- Named iterator
  - Needs both variable type and name, and then allows retrieval of columns by name
  - See example on previous slide:
    #sql iterator Sailors( Int sid, String name, Int rating );
- Positional iterator
  - Needs only variable *type* (not name), uses FETCH ... INTO construct:

```
#sql iterator Sailors( Int, String, Int );
Sailors sailors;
#sql sailors = { SELECT sid, sname INTO :sid, :name FROM Sailors WHERE rating = :rating };
do
{   #sql { FETCH :sailors INTO :sid, :name };
    if ( ! sailors.endFetch() )
        …; // process sailor
} while (! sailors.endFetch() );
```

# Overview

- SQL API
  - Example 1: PHP
  - Example 2: JDBC
- Embedded SQL
  - Basics; Cursors; Dynamic SQL – based on Example 1: C
  - Example 2: SQLJ
- Stored procedures

# SQL/PSM

JACOBS UNIVERSITY

- Most DBMSs allow users to write stored procedures in a simple, general-purpose language (close to SQL)
  - SQL/PSM standard is a representative
  - SQLJ worth considering
  - Other languages possible too, see vendor manuals
- Procedural constructs: procs/functions, variables, branches, loops
  - computationally complete
- Example: dock foreign code into database server:

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

# SQL/PSM Example

```
CREATE FUNCTION rateSailor (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;
```

# Calling Stored Procedures from Client

JACOBS UNIVERSITY

- Embedded SQL:
  - EXEC CALL IncreaseRating( :sid, :rating );
- JDBC:
  - CallableStatement cstmt = con.prepareCall( "{call ShowSailors}" );
- SQLJ:
  - #sql showsailors = { CALL ShowSailors };

# Summary: Connecting PL & DBMS

JACOBS UNIVERSITY

- Coupling techniques
  - API: library with DBMS calls = layer of abstraction between application and DBMS
  - Embedded SQL: extend PL with SQL statements
  - Stored procedures: execute application logic directly at the server
- Cursor mechanism for record-at-a-time traversal
  - bridge impedance mismatch
- Query flexibility
  - Static queries: fixed & checked a compile-time, only parameters can vary
  - Dynamic SQL: ad-hoc queries