



JACOBS
UNIVERSITY

Database Application Development

Ramakrishnan & Gehrke, Chapter 6

SQL Integration Approaches

- 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, ...

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

DB APIs: the Alternative to Embedding



JACOBS
UNIVERSITY

- No "syntactic sugar" through precompiler, but direct access to **library** with database calls
 - Pass SQL string from language, present results in language-friendly way
- Supposedly DBMS-neutral through encapsulating classes
 - "driver" translates into DBMS-specific code
- **PHP**: "Private Home Page" -> "PHP Hypertext Processor"
- **JDBC**: Java SQL API (Sun Microsystems)
 - cf. ODBC by Microsoft

PHP and (My)SQL



- PHP calls embedded within HTML as special tag
 - `<?php php-statement-sequence ?>`
- Execution (server-side!) of PHP statements generates text which substitutes PHP code snippets; all then is forwarded by Web server:

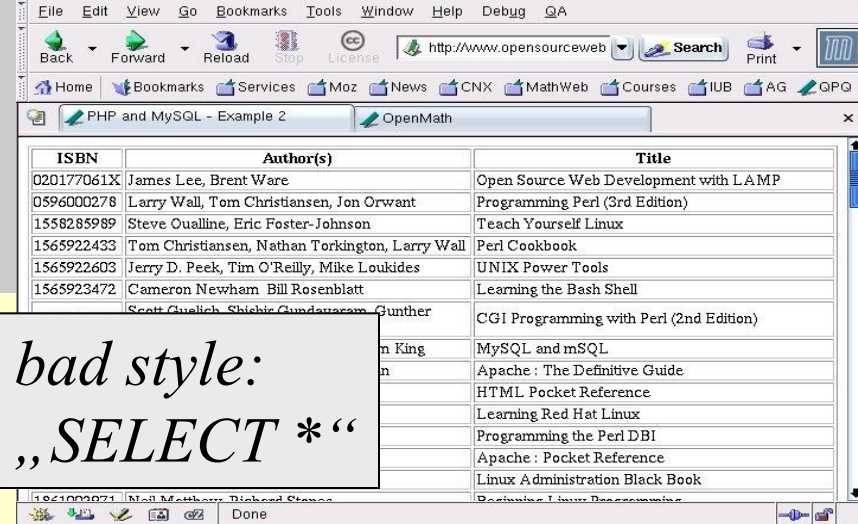
`<h1><?php echo "Hello World"; ?></h1>` → `<h1>Hello World</h1>`

- Example: connecting to mysql server on localhost

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

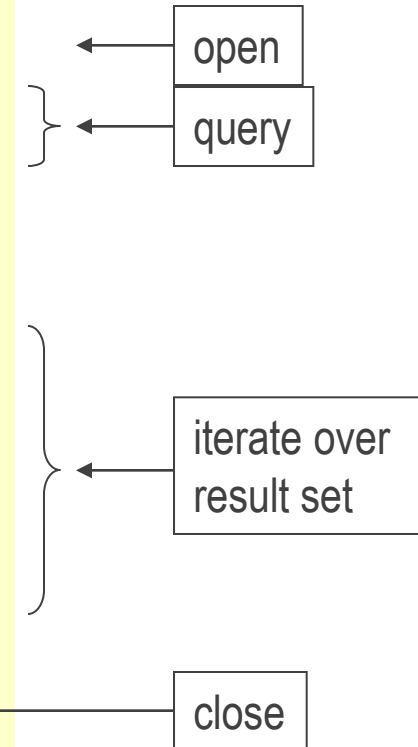
variables
have „\$“
prefix

PHP, HTML, and (My)SQL



bad style:
„*SELECT **“

```
<html>
<head>
  <title>PHP and MySQL Example</title>
</head>
<body>
  <?php $mysql = mysql_connect( "localhost", "apache", "DBWAisCool" );
  $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>
```

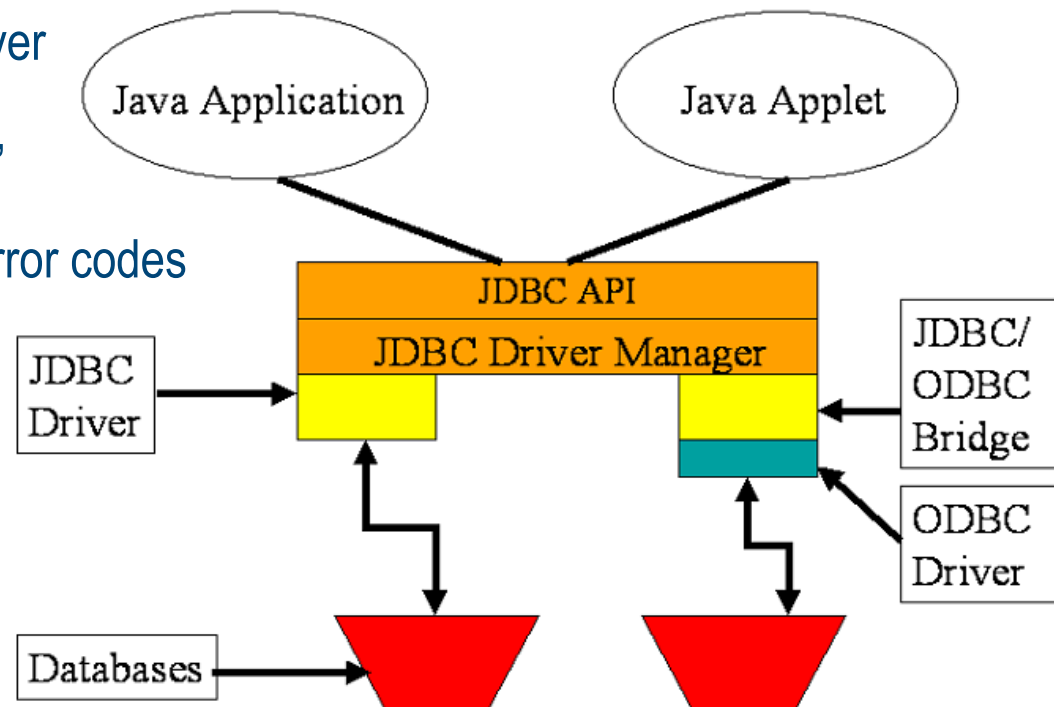


- SQL API
 - Example 1: PHP
 - Example 2: JDBC
- Embedded SQL
 - Basics; Cursors; Dynamic SQL – based on Example 1: C
 - Example 2: SQLJ
- Stored procedures

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



JDBC Architecture: Driver Types

■ 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*

■ Network bridge:

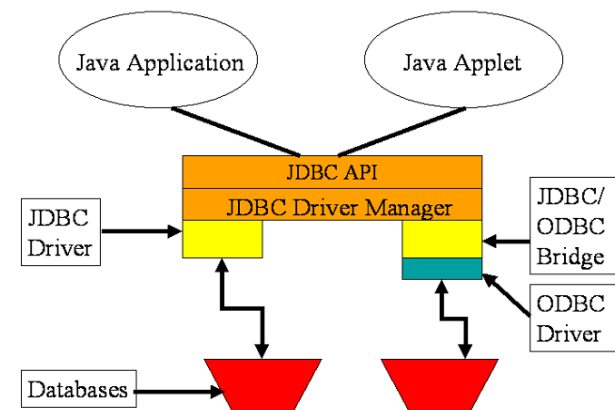
- Send commands over the network to a middleware server that talks to the data source
- *Needs only small JDBC driver at each client*

■ Direct 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*

■ Direct translation to native API, non-Java driver:

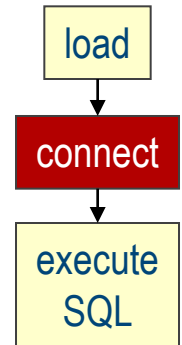
- Translates SQL commands to native API of data source
- *Need OS-specific binary on each client*



Connections in JDBC

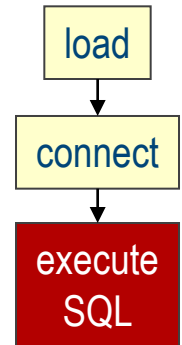
- interact with data source through **sessions**
 - Each connection identifies a logical session
- JDBC URL:
jdbc:<subprotocol>:<otherParameters>
- Example:

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



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)



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();
```

not in exams

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

ResultSets

not in exams



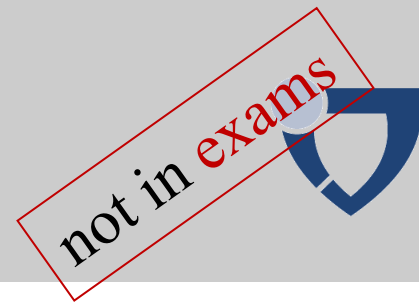
JACOBS
UNIVERSITY

- 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.

JDBC: Error Handling



- 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

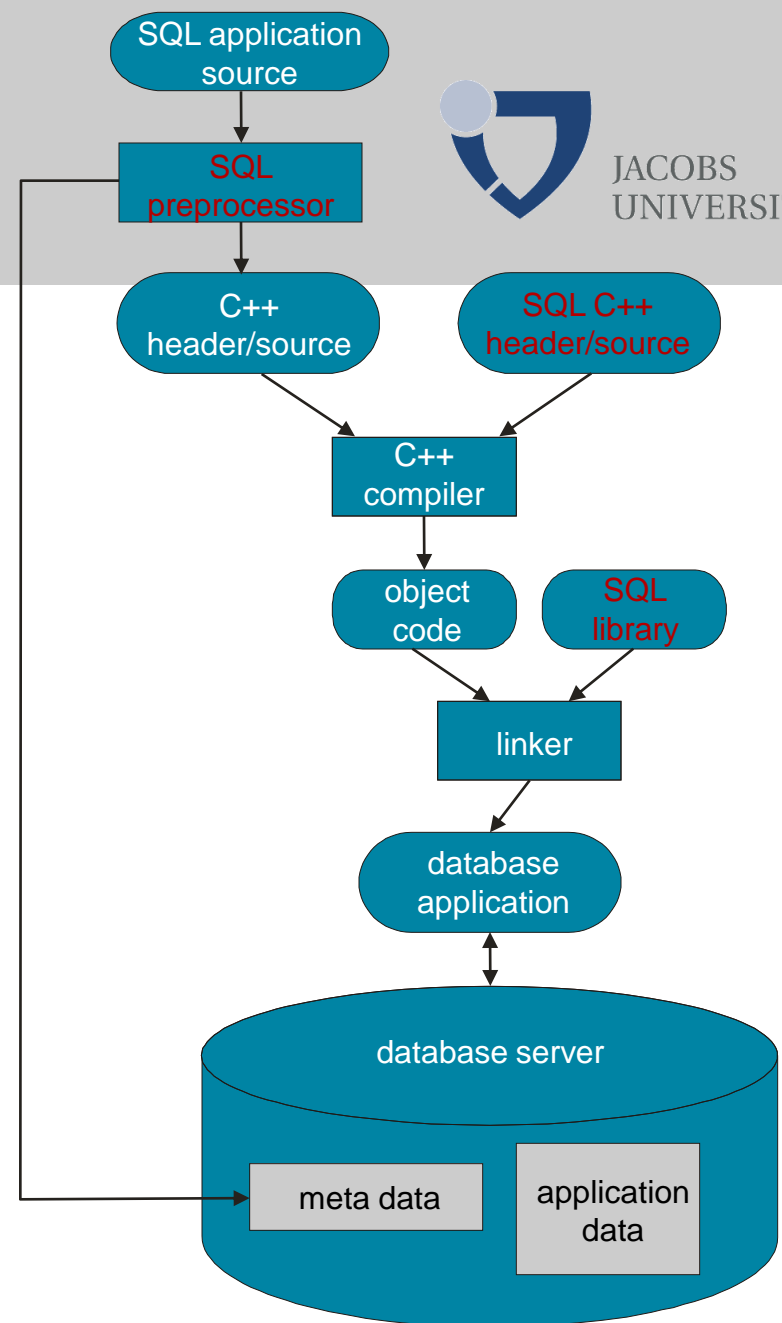
- not thrown, existence has to be explicitly tested:

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

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

Embedded SQL

- Approach:
make SQL statements
part of host language
 - **preprocessor** converts SQL statements into sequences of API calls (can look into that source code)
 - regular compiler for generated code
 - link code with vendor-supplied library
 - See www.knosof.co.uk/sqlport.html for tech details & issues



Embedded SQL Language Constructs

- 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) ...;  
}
```

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

- Problem: How to iterate over result sets when procedural languages do not know “sets”?
- **Cursor** = aka generic iterator (C++!)
 - 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
- special clause **ORDER BY** to control order in which tuples are returned
 - Fields in ORDER BY clause **must** also appear in SELECT clause
- Can also **modify/delete** tuple pointed to by a cursor
 - ...but no update of attributes mentioned in ORDER BY clause (obviously)

Names of sailors who have 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
```

- 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

```
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
that hint to distinguish
program from SQL
variables*

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

- **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>

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();
```


SQLJ vs. JDBC

```
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();
```

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()
REAL	Double	getDouble()
INTEGER	Integer	getInt()
DATE	java.sql.Date	getDate()
TIME	java.sql.Time	getTime()
TIMESTAMP	java.sql.Timestamp	getTimestamp()

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

- 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;
```



Summary: Connecting PL & DBMS

- 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
 - (parametrized) **static queries**, checked at compile-time
 - **Dynamic SQL**: ad-hoc queries within host language