Database Applications Cont.: JDBC and Java Access to DBMS
MySQLdb and Python Access to MySQL

University of California, Berkeley
School of Information
IS 257: Database Management
Lecture Outline

• Review:
  – PHP
• Java and JDBC
• SQLite3 (and Python)
• MySQL and Python
Lecture Outline

• Review:
  – PHP
• Java and JDBC
• SQLite3 (and Python)
• MySQL and Python
PHP Combined with MySQL

• DBMS interface appears as a set of functions:

```php
<?php
mysql_connect("localhost", "usename", "password");
mysql_select_db("mydb");
$result = mysql_query("SELECT * FROM employees");
while ($r = mysql_fetch_array($result, MYSQL_ASSOC)) {
    printf("<center><H2>%s</H2></center> ", $r["LAST_NAME"]);
    printf("%s", $r["FIRST_NAME"]);
}
?>
```
Mysqli – an enhanced interface

```php
include 'msqlini.php';
$mysqli = new mysqli($host,$user,$pw,$dbname);
if ($mysqli->connect_error) {
    echo "Failed to connect to MySQL: (" . $mysqli->connect_errno . ") " . $mysqli->connect_error . 
}$cust_id = $_GET["cust_id"];
$ cust_id = mysql_real_escape_string($cust_id);

/* start first prepared statement */
$ stmt = $mysqli->stmt_init();
if ($stmt->prepare("SELECT * FROM DIVECUST where Customer_No= ? ")) {
    if (!$stmt->bind_param("i", $cid)) {
        echo "Binding parameters failed: (" . $stmt->errno . ") " . $stmt->error;
    }
    $cid = $cust_id;
    if (!$stmt->execute()) {
        echo "Execute failed: (" . $stmt->errno . ") " . $stmt->error;
    }
    $stmt->bind_result($custid,$name,$street,$city,$state,$zip,$country,$phone, $contact);
```
Connecting to returned values

if ($stmt->prepare("SELECT * FROM DIVECUST where Customer_No= ? ") ) {
    if (!$stmt->bind_param("i", $cid)) {
        echo "Binding parameters failed: (" . $stmt->errno . ") " . $stmt->error;
    }
    $cid = $cust_id;
    if (!$stmt->execute()) {
        echo "Execute failed: (" . $stmt->errno . ") " . $stmt->error;
    }
    $stmt->bind_result($custid,$name,$street,$city,$state,$zip,$country,$phone, $contact);
    while ($stmt->fetch()) {
        echo "<br>Customer ID: $custid";
        echo "<br>Name: <font size=+1><b>$name</b></font>";
        echo "<br>Street: $street";
        echo "<br>City: $city";
        echo "<br>State: $state";
        echo "<br>ZIP: $zip";
        echo "<br>Country: $country";
        echo "<br>Phone: $phone";
        echo "<br>Date of first contact: $contact";
    }
    $stmt->close();
}
More protection...

• For data input and passing parameters you will also need to use parameterized or “prepared” SQL statements to avoid the possibility of SQL Injection attacks:
Lecture Outline

• Review:
  – PHP
• MySQL functions and setup
• Java and JDBC
• SQLite3 (and Python)
• MySQL and Python
Java and JDBC

• Java was probably the high-level language used in most instruction and development in recent years.

• One of the earliest “enterprise” additions to Java was JDBC.

• JDBC is an API that provides a mid-level access to DBMS from Java applications.

• Intended to be an open cross-platform standard for database access in Java.

• Similar in intent to Microsoft’s ODBC.
JDBC Architecture

- The goal of JDBC is to be a generic SQL database access framework that works for any database system with no changes to the interface code.
JDBC

- Provides a standard set of interfaces for any DBMS with a JDBC driver – using SQL to specify the databases operations.
import java.sql.*;

public class JDBCCTestMysqlHarbinger {

    public static void main(java.lang.String[] args) {
        try {
            // this is where the driver is loaded
            Class.forName("com.mysql.jdbc.Driver").newInstance();
        }
        catch (InstantiationException i) {
            System.out.println("Unable to load driver Class");
            return;
        }
        catch (ClassNotFoundException e) {
            System.out.println("Unable to load driver Class");
            return;
        }
        catch (IllegalAccessException e) {

        }
        catch (IllegalArgumentException e) {

        }
        catch (IllegalAccessCompileException e) {

        }
    }
}
try {
//All DB access is within the try/catch block...
Connection con = DriverManager.getConnection("jdbc:mysql://localhost
/ray?user=ray&password=XXXXXXX");
// Do an SQL statement...
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery("SELECT name FROM DIVECUST");
// show the Results...
while(rs.next()) {
    System.out.println(rs.getString("Name"));
    System.out.println(StringUtils.EMPTY);
}

// Release the db resources...
rs.close();
stmt.close();
con.close();

} catch (SQLException se) {
    // inform user of errors...
    System.out.println("SQL Exception: " + se.getMessage());
    se.printStackTrace(System.out);
}
JDBC

• Once a connection has been made you can create three different types of statement objects

• Statement
  – The basic SQL statement as in the example

• PreparedStatement
  – A pre-compiled SQL statement

• CallableStatement
  – Permits access to stored procedures in the Database
JDBC Resultset methods

- Next() to loop through rows in the resultset
- To access the attributes of each row you need to know its type, or you can use the generic “getObject()” which wraps the attribute as an object
### JDBC “GetXXX()” methods

<table>
<thead>
<tr>
<th>SQL data type</th>
<th>Java Type</th>
<th>GetXXX()</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR</td>
<td>String</td>
<td>getString()</td>
</tr>
<tr>
<td>VARCHAR</td>
<td>String</td>
<td>getString()</td>
</tr>
<tr>
<td>LONGVARCHAR</td>
<td>String</td>
<td>getString()</td>
</tr>
<tr>
<td>NUMERIC</td>
<td>Java.math. BigDecimal</td>
<td>GetBigDecimal()</td>
</tr>
<tr>
<td>DECIMAL</td>
<td>Java.math. BigDecimal</td>
<td>GetBigDecimal()</td>
</tr>
<tr>
<td>BIT</td>
<td>Boolean</td>
<td>getBoolean()</td>
</tr>
<tr>
<td>TINYINT</td>
<td>Byte</td>
<td>getByte()</td>
</tr>
<tr>
<td>SQL data type</td>
<td>Java Type</td>
<td>GetXXX()</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------------</td>
<td>--------------------</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>Integer (short)</td>
<td>getShort()</td>
</tr>
<tr>
<td>INTEGER</td>
<td>Integer</td>
<td>getInt()</td>
</tr>
<tr>
<td>BIGINT</td>
<td>Long</td>
<td>getLong()</td>
</tr>
<tr>
<td>REAL</td>
<td>Float</td>
<td>getFloat()</td>
</tr>
<tr>
<td>FLOAT</td>
<td>Double</td>
<td>getDouble()</td>
</tr>
<tr>
<td>DOUBLE</td>
<td>Double</td>
<td>getDouble()</td>
</tr>
<tr>
<td>BINARY</td>
<td>Byte[]</td>
<td>getBytes()</td>
</tr>
<tr>
<td>VARBINARY</td>
<td>Byte[]</td>
<td>getBytes()</td>
</tr>
<tr>
<td>LONGVARBINARY</td>
<td>Byte[]</td>
<td>getBytes()</td>
</tr>
</tbody>
</table>
## JDBC GetXXX() Methods

<table>
<thead>
<tr>
<th>SQL data type</th>
<th>Java Type</th>
<th>GetXXX()</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATE</td>
<td>java.sql.Date</td>
<td>getDate()</td>
</tr>
<tr>
<td>TIME</td>
<td>java.sql.Time</td>
<td>getTime()</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>java.sql.Timestamp</td>
<td>getTimeStamp()</td>
</tr>
</tbody>
</table>
Large Object Handling

- Large binary data can be read from a resultset as streams using:
  - `getAsciiStream()`
  - `getBinaryStream()`
  - `getUnicodeStream()`

```java
ResultSet rs = stmt.executeQuery("SELECT IMAGE FROM PICTURES WHERE PID = 1223");
if (rs.next()) {
    BufferedInputStream gifData = new BufferedInputStream(rs.getBinaryStream("IMAGE"));
    byte[] buf = new byte[4*1024]; // 4K buffer
    int len;
    while ((len = gifData.read(buf,0,buf.length)) != -1) {
        out.write(buf, 0, len);
    }
}
```
JDBC Metadata

• There are also methods to access the metadata associated with a resultSet
  – ResultSetMetaData rsmd = rs.getMetaData();

• Metadata methods include…
  – getColumnCount();
  – getColumnLabel(col);
  – getColumnTypeName(col)
JDBC access to other DBMS

• The basic JDBC interface is the same, the only differences are in how the drivers are loaded...

```java
public class JDBCCTestMysql {
    public static void main(java.lang.String[] args) {
        try {
            // this is where the driver is loaded
            //Class.forName("com.mysql.jdbc.Driver").newInstance();
            DriverManager.registerDriver(new OracleDriver());
        } catch (InstantiationException i) {
            System.out.println("Unable to load driver Class");
            return;
        }
        catch (ClassNotFoundException e) {
            System.out.println("Unable to load driver Class"); ...
        }
    }
}
```
try {
    // All DB access is within the try/catch block...
    // make a connection to MySQL on Dream
    Connection con = DriverManager.getConnection("jdbc:oracle:thin:
        @dream.sims.berkeley.edu:1521:dev","ray", "XXXXXX");
    //Connection con = DriverManager.getConnection(
    //  "jdbc:mysql://localhost/MyDatabase?user=MyLogin&password=MySQLPW");
    // Do an SQL statement...
    Statement stmt = con.createStatement();
    ResultSet rs = stmt.executeQuery("SELECT NAME FROM DIVECUST");

    • Otherwise everything is the same as in the MySQL example
    • For connecting to the machine you are running the program on, you can use “localhost” instead of the machine name
Demo – JDBC for MySQL

• Demo of JDBC code on Harbinger
• Code will be available on class web site
Lecture Outline

• Review:
  – PHP
• Java and JDBC
• SQLite3 (and Python)
• MySQL and Python
Python and MySQL

• Python has a standard for database interfaces called the Python DB-API. Most Python database interfaces adhere to this standard.

• You can choose the right database for your application. Python Database API supports a wide range of database servers including MySQL, PostgreSQL, Microsoft SQL Server, Oracle, Sybase, etc.
SQLite3

• We have already mentioned the SQLite3 interface, which is an example of the DB-API

• Today we will look at it in a bit more detail, and see how an embedded DBMS might be used in your applications
SQLite3

- Light-weight implementation of a relational DBMS (~340Kb)
  - Includes most of the features of full DBMS
  - Intended to be embedded in programs
- Available on iSchool servers and for other machines as open source
- Used as the data manager in iPhone apps and Firefox (among many others)
- Databases are stored as files in the OS
SQLite3 Data types

- SQLite uses a more generic dynamic type system. In SQLite, the datatype of a value is associated with the value itself, not with its container.

- Types are:
  - **NULL**: The value is a NULL value.
  - **INTEGER**: The value is a signed integer, stored in 1, 2, 3, 4, 6, or 8 bytes depending on the magnitude of the value.
  - **REAL**: The value is a floating point value, stored as an 8-byte IEEE floating point number.
  - **TEXT**: The value is a text string, stored using the database encoding (UTF-8, UTF-16BE or UTF-16LE). (default max 1,000,000,000 chars)
  - **BLOB**: The value is a blob of data, stored exactly as it was input.
SQLite Command line

[dhcp137:~] ray% sqlite3 test.db
SQLite version 3.6.22
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> .tables
sqlite> create table stuff (id int, name varchar(30), address varchar(50));
sqlite> .tables stuff
sqlite> insert into stuff values (1,'Jane Smith','123 east st.');</sql
sqlite> select * from stuff;
1|Jane Smith|123 east st.
sqlite> insert into stuff values (2, 'Bob Jones', '234 west st.');
sqlite> insert into stuff values (3, 'John Smith', '567 North st.');
sqlite> update stuff set address = "546 North st." where id = 1;
sqlite> select * from stuff;
1|Jane Smith|546 North st.
2|Bob Jones|234 west st.
3|John Smith|567 North st.
Wildcard searching

sqlite> select * from stuff where name like '%Smith%';
1|Jane Smith|546 North st.
3|John Smith|567 North st.

sqlite> select * from stuff where name like 'J%Smith%';
1|Jane Smith|546 North st.
3|John Smith|567 North st.

sqlite> select * from stuff where name like 'Ja%Smith%';
1|Jane Smith|546 North st.

sqlite> select * from stuff where name like 'Jones';
sqlite> select * from stuff where name like '%Jones';
2|Bob Jones|234 west st.

sqlite> select name from stuff
   ...> ;
Jane Smith
Bob Jones
John Smith

sqlite>
Create backups

sqlite> .dump
PRAGMA foreign_keys=OFF;
BEGIN TRANSACTION;
CREATE TABLE stuff (id int, name varchar(30), address varchar(50));
INSERT INTO "stuff" VALUES(1,'Jane Smith','546 North st.'));
INSERT INTO "stuff" VALUES(2,'Bob Jones','234 west st.'));
INSERT INTO "stuff" VALUES(3,'John Smith','567 North st.'));
COMMIT;
sqlite> .schema
CREATE TABLE stuff (id int, name varchar(30), address varchar(50));
Creating Tables from Tables

sqlite> create table names as select name, id from stuff;
sqlite> .schema
CREATE TABLE names(name TEXT,id INT);
CREATE TABLE stuff (id int, name varchar(30),address varchar(50));
sqlite> select * from names;
  Jane Smith 1
  Bob Jones 2
  John Smith 3
sqlite> create table names2 as select name as xx, id as key from stuff;
sqlite> .schema
CREATE TABLE names(name TEXT,id INT);
CREATE TABLE names2(xx TEXT,"key" INT);
CREATE TABLE stuff (id int, name varchar(30),address varchar(50));
sqlite> drop table names2;
sqlite> .schema
CREATE TABLE names(name TEXT,id INT);
CREATE TABLE stuff (id int, name varchar(30),address varchar(50));
Using SQLite3 from Python

• SQLite is available as a loadable python library
  – You can use any SQL commands to create, add data, search, update and delete
SQLite3 from Python

[dhcp137:~] ray% python
Python 2.5.1 (r251:54869, Apr 18 2007, 22:08:04)
[GCC 4.0.1 (Apple Computer, Inc. build 5367)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import sqlite3
>>> sqlite3.version
'2.3.2'
>>> sqlite3.sqlite_version
'3.3.14'
>>>
SQLite3 from Python

[dhcp137:~] ray% python
Python 2.5.1 (r251:54869, Apr 18 2007, 22:08:04)
[GCC 4.0.1 (Apple Computer, Inc. build 5367)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import sqlite3 as lite
>>> import sys
>>> con = None
>>> try:
...     con = lite.connect('newtest.db')
...     cur = con.cursor()
...     cur.execute('SELECT SQLITE_VERSION()')
...     data = cur.fetchone()
...     print "SQLite version: %s" % data
... except lite.Error, e:
...     print "Error %s:" % e.args[0]
...     sys.exit(1)
... finally:
...     if con:
...         con.close()
...
<sqlite3.Cursor object at 0x46eb90>
SQLite version: 3.3.14
>>>
SQLite3 from Python

```python
#!/usr/bin/python2.7
# -*- coding: utf-8 -*-
import sqlite3 as lite
import sys

# our data is defined as a tuple of tuples...
cars = (
    (1, 'Audi', 52642),
    (2, 'Mercedes', 57127),
    (3, 'Skoda', 9000),
    (4, 'Volvo', 29000),
    (5, 'Bentley', 350000),
    (6, 'Hummer', 41400),
    (7, 'Volkswagen', 21600)
)
con = lite.connect('newtest.db')
with con:
    cur = con.cursor()
    cur.execute("DROP TABLE IF EXISTS Cars")
    cur.execute("CREATE TABLE Cars(\Id INT, Name TEXT, Price INT)")
    cur.executemany("INSERT INTO Cars VALUES(?,?,?)", cars)
```
Another Example

```python
#!/bin/env python
# -*- coding: utf-8 -*-
import sqlite3 as lite
import sys

con = lite.connect(':memory:)

with con:
    cur = con.cursor()
    cur.execute("CREATE TABLE Friends(Id INTEGER PRIMARY KEY,
        Name TEXT);")
    cur.execute("INSERT INTO Friends(Name) VALUES ('Tom');")
    cur.execute("INSERT INTO Friends(Name) VALUES ('Rebecca');")
    cur.execute("INSERT INTO Friends(Name) VALUES ('Jim');")
    cur.execute("INSERT INTO Friends(Name) VALUES ('Robert');")

    lid = cur.lastrowid
    print "The last Id of the inserted row is \%d" % lid
```
Retrieving Data

```
#!/bin/env python
# -*- coding: utf-8 -*-

import sqlite3 as lite
import sys

# connect to the cars database...
con = lite.connect('newtest.db')

with con:
    cur = con.cursor()
    cur.execute("SELECT * FROM Cars")
    rows = cur.fetchall()
    for row in rows:
        print row
```

```
ray% python2.7 retrnewtest.py
(1, u'Audi', 52642)
(2, u'Mercedes', 57127)
(3, u'Skoda', 9000)
(4, u'Volvo', 29000)
(5, u'Bentley', 350000)
(6, u'Hummer', 41400)
(7, u'Volkswagen', 21600)
(8, u'Citroen', 21000)
ray%
```
cur.execute("UPDATE Cars set Price = 450000 where Name = 'Bentley'")

cur.execute("SELECT * FROM Cars")
rows = cur.fetchall()
for row in rows:
    print row

(1, u'Audi', 52642)
(2, u'Mercedes', 57127)
(3, u'Skoda', 9000)
(4, u'Volvo', 29000)
(5, u'Bentley', 450000)
(6, u'Hummer', 41400)
(7, u'Volkswagen', 21600)
(8, u'Citroen', 21000)
Add another row...

```python
[dhcp137:~] ray% python2.7
Python 2.7.2 (default, Oct 11 2012, 20:14:37)
[ GCC 4.2.1 Compatible Apple Clang 4.0 …
>>> import sqlite3 as lite
>>> import sys
>>> 
>>> con = lite.connect('newtest.db')
>>> 
>>> with con:
...     cur = con.cursor()
...     cur.execute("INSERT INTO Cars VALUES(8,'Citroen',21000)")
...
<sqlite3.Cursor object at 0x107fafc00>
>>> 
```
From the SQLite3 command line

[dhcp137:~] ray% sqlite3 newtest.db
SQLite version 3.6.22
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> select * from cars;
1|Audi|52642
2|Mercedes|57127
3|Skoda|9000
4|Volvo|29000
5|Bentley|350000
6|Hummer|41400
7|Volkswagen|21600
8|Citroen|21000
sqlite>

INSERT more data…
sqlite> select * from cars;
1|Audi|52642
2|Mercedes|57127
3|Skoda|9000
4|Volvo|29000
5|Bentley|450000
6|Hummer|41400
7|Volkswagen|21600
8|Citroen|21000
10|Audi|51000
11|Mercedes|55000
12|Mercedes|56300
13|Volvo|31500
14|Volvo|31000
15|Audi|52000
17|Hummer|42400
16|Hummer|42400
Use Aggregates to summarize data

```python
#!/usr/bin/python2.7
# -*- coding: utf-8 -*-
import sqlite3 as lite
import sys

con = lite.connect('newtest.db')
with con:
    cur = con.cursor()
    cur.execute("SELECT Name, AVG(Price)
                  FROM Cars GROUP BY Name")
    rows = cur.fetchall()
    for row in rows:
        print row
```

ray% python2.7 aggnewtest.py
(u'Audi', 51880.666666666664)
(u'Bentley', 450000.0)
(u'Citroen', 21000.0)
(u'Hummer', 42066.666666666664)
(u'Mercedes', 56142.333333333336)
(u'Skoda', 9000.0)
(u'Volkswagen', 21600.0)
(u'Volvo', 30500.0)
Lecture Outline

• Review:
  – PHP
• MySQL functions and setup
• Java and JDBC
• SQLite3 (and Python)
• MySQL and Python
MySQLdb

• MySQLdb is a DB-API for MySQL
• The basic setup is fairly simple…
  – Pip install MySQL-python
  – Conda install mysql-python
• Or, if on harbinger it is already installed
• To use the interface…
#!/usr/bin/python

import MySQLdb

# Open database connection
db = MySQLdb.connect("ischool.berkeley.edu","ray","YOURPW","ray")

# prepare a cursor object using cursor() method
cursor = db.cursor()

# execute SQL query using execute() method.
cursor.execute("SELECT VERSION()")

# Fetch a single row using fetchone() method.
data = cursor.fetchone()

print "Database version : %s " % data

# disconnect from server
db.close()
#!/usr/bin/python
import MySQLdb

... 
cursor = db.cursor()
# Make a string of SQL commands...
sql = "SELECT * FROM DIVECUST"

try:
    # Execute the SQL command in a try/except in case of failure
cursor.execute(sql)
    # Fetch all the rows in a list of lists.
    results = cursor.fetchall()
    for row in results:
        custno = row[0]
        custname = row[1]
        street = row[2]
        city = row[3]
        state = row[4]
        zip = row[5]
        country = row[6]
        # Now print fetched result
        print "%s : %s, %s, %s, %s %s" % 
            (custname, street, city, state, zip, country)
except:
    print "Error: unable to fetch data"
Can run any SQL...

```python
#!/usr/bin/python
import MySQLdb

# Open database connection
db = MySQLdb.connect("localhost","testuser","test123","TESTDB")

# prepare a cursor object using cursor() method
cursor = db.cursor()

# Drop table if it already exist using execute() method.
cursor.execute("DROP TABLE IF EXISTS EMPLOYEE")

# Create table as per requirement
sql = '"""CREATE TABLE EMPLOYEE ( 
    FIRST_NAME  CHAR(20) NOT NULL,
    LAST_NAME  CHAR(20),
    AGE INT,
    SEX CHAR(1),
    INCOME FLOAT )"""
cursor.execute(sql)

# disconnect from server
db.close()
```
#!/usr/bin/python

import MySQLdb

# Open database connection
db = MySQLdb.connect("localhost","testuser","test123","TESTDB" )

# prepare a cursor object using cursor() method
cursor = db.cursor()

# Prepare SQL query to INSERT a record into the database.
sql = """"INSERT INTO EMPLOYEE(FIRST_NAME, LAST_NAME, AGE, SEX, INCOME)
VALUES ('Mac', 'Mohan', 20, 'M', 2000)""

try:
    # Execute the SQL command
    cursor.execute(sql)
    # Commit your changes in the database
    db.commit()
except:
    # Rollback in case there is any error
    db.rollback()

# disconnect from server
db.close()
#!/usr/bin/python

import MySQLdb

# Open database connection
db = MySQLdb.connect("localhost","testuser","test123","TESTDB" )

# prepare a cursor object using cursor() method
cursor = db.cursor()

# Prepare SQL query to UPDATE required records
sql = "UPDATE EMPLOYEE SET AGE = AGE + 1  
     WHERE SEX = '%c' " % ('M')

try:
    # Execute the SQL command
    cursor.execute(sql)
    # Commit your changes in the database
    db.commit()
except:
    # Rollback in case there is any error
    db.rollback()

# disconnect from server
db.close()