

Concepts for Project 2: Updates, JDBC, and Transactions

CSE 444 section
October 21, 2010

Today

- Database updates
- Java Database Connectivity
- Transactions in SQL

Modifying the database

Three kinds of modifications in SQL:

- insertions
- updates
- deletions

Sometimes they are all called “updates”

□

Insertions

General form:

```
INSERT INTO R(A1,...., An) VALUES (v1,...., vn)
```

Insertions

Product (name, listPrice, category)

Purchase (buyer, seller, product, price)

Example: Insert a new purchase to the database:

```
INSERT INTO Purchase (buyer, seller, product, price)
VALUES ('Joe', 'Fred', 'wakeup-clock-espresso-machine',
         199.99)
```

Missing attributes → NULL.

May drop attribute names if you give them in order.

Inserting results of a query

```
INSERT INTO Product (name)  
  
  SELECT DISTINCT Purchase.product  
  FROM Purchase  
  WHERE Purchase.date > "10/26/01";
```

The query replaces the **VALUES** keyword.
Here we insert *many* tuples into Product

Updates

Example:

```
UPDATE Product
SET price = price/2
WHERE Product.name IN
      (SELECT product
       FROM Purchase
       WHERE Date = 'Oct, 25, 1999');
```

WHERE works the same as in a query (SELECT).
It chooses the tuples whose values are to be updated

Deletions

Similar to UPDATE but without the SET clause:

```
DELETE FROM Purchase  
  
WHERE seller = 'Joe' AND  
        product = 'Brooklyn Bridge'
```

Always specify a WHERE clause (in fact, write it first!)
Otherwise, *every tuple* will be deleted!

Today

- Database updates
- **Java Database Connectivity**
- Transactions in SQL

JDBC (Java Database Connectivity)

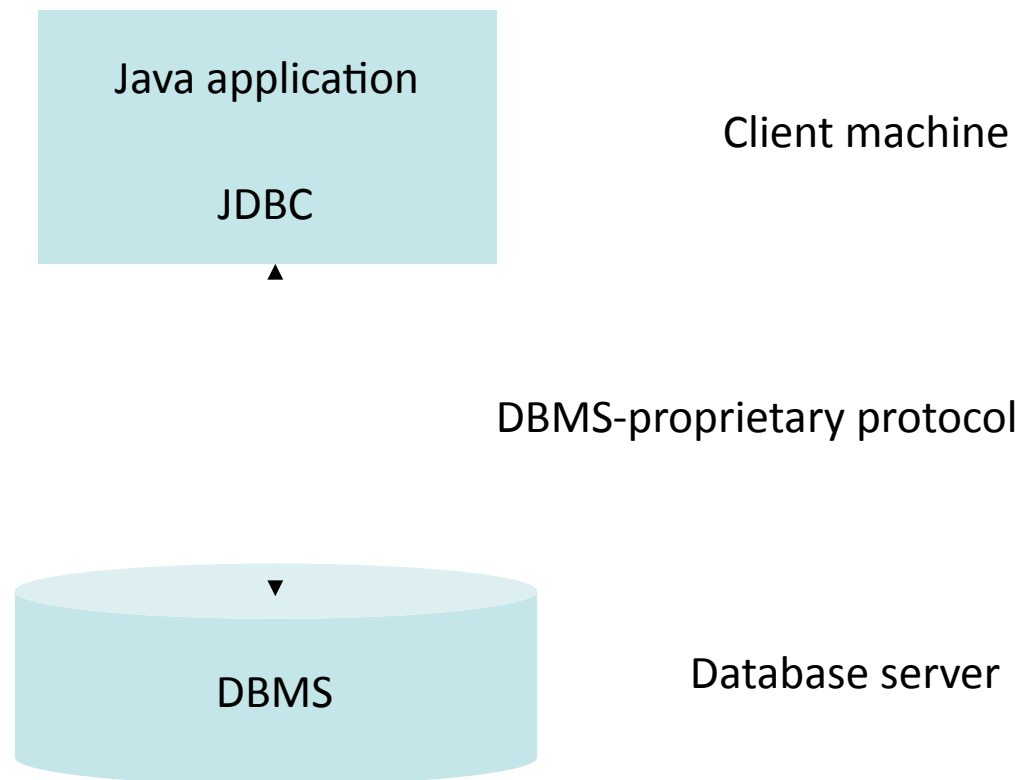
A Java API to access a database:

- connect to a data source
- send queries and update statements
- retrieve and process results

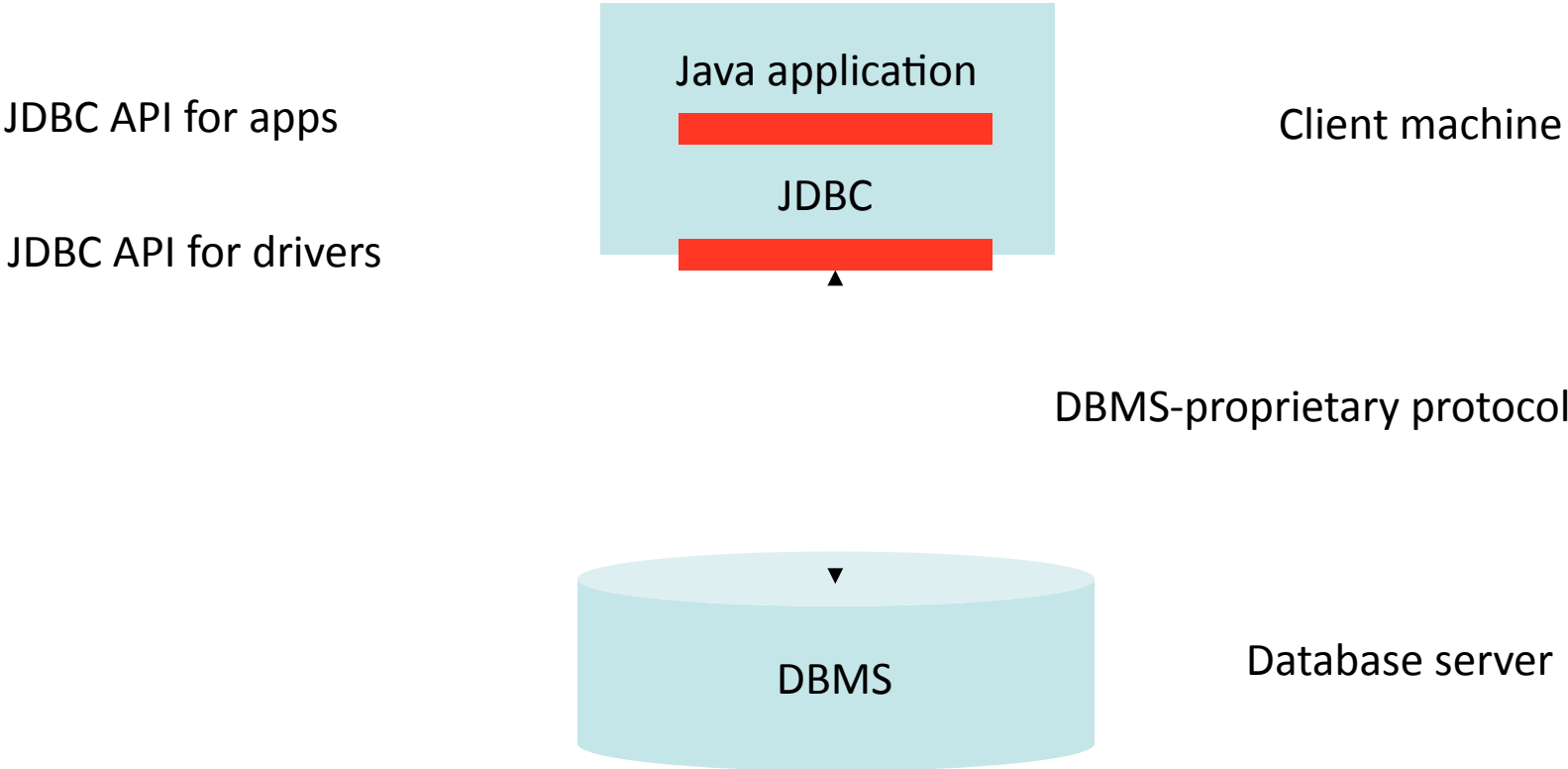
Documentation:

<http://java.sun.com/javase/6/docs/technotes/guides/jdbc/>

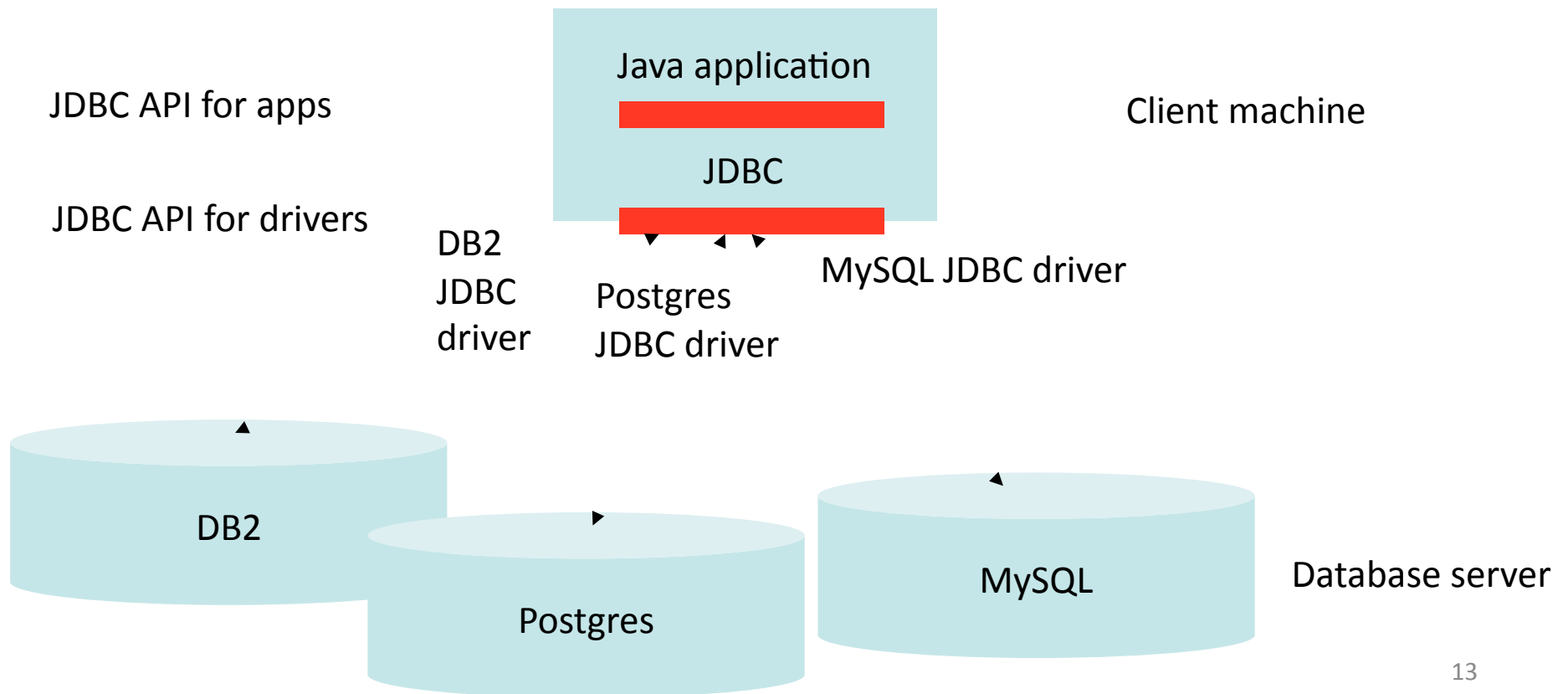
JDBC lets Java talk to your database



DBMS vendors make JDBC drivers...



... letting JDBC talk to *any* database



First, load the driver

- For Project 2, look in project2.tar.gz
 - SQL Server driver
sqljdbc4.jar
 - PostgreSQL driver
postgresql-8.4-701.jdbc4.jar
 - Already installed on Lab PCs (use 444shell.cmd)
- Put on class path, then tell Java to load it
Class.forName
("com.microsoft.sqlserver.jdbc.SQLServerDriver");
Class.forName ("org.postgresql.Driver");
 - Class.forName() optional in current versions of Java

JDBC example

```
Connection conn = DriverManager.getConnection
    ("jdbc:sqlserver://iisqlsrv;database=username",
     "username", "password");

Statement stmt = conn.createStatement();

ResultSet rs = stmt.executeQuery
    ("SELECT * FROM hw1_data WHERE month='feb'");

while (rs.next()) {
    String productName = rs.getString("name");
    String discount = rs.getString(2);
    int price = rs.getInt(4);
}
```

Modifying the database

Use `Statement.executeUpdate()`:

```
Statement stmt = conn.createStatement();

int rowsUpdated = stmt.executeUpdate (
    "UPDATE hw1_data " +
    "SET name = 'jane''s super gizmo' " +
    "WHERE name = 'gizmo3' "
);
```

- Works with any database modification, not just UPDATE
- Warning – will throw if you run it with a query!

Close all JDBC objects when done

```
Connection conn = DriverManager.getConnection(...);
Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery
    ("SELECT * FROM hw1_data");

// do work with rs...

rs.close();
stmt.close();
conn.close();
```

```
Class.forName
    ("com.microsoft.sqlserver.jdbc.SQLServerDriver");

Connection conn = null;

try {
    conn = DriverManager.getConnection( ... );

    ...

} catch (Exception e) {
    e.printStackTrace();
} finally {
    conn.close();
}
```

Parameterized queries - PreparedStatement

```
PreparedStatement pstmt = conn.prepareStatement  
    ("SELECT * from hw1_data WHERE month = ? ");
```

...

```
pstmt.setString(1, "may");  
ResultSet rs1 = pstmt.executeQuery();
```

...

```
pstmt.setString(1, "aug");  
ResultSet rs2 = pstmt.executeQuery();
```

...

Parameterized queries - PreparedStatement

No need to worry about quotes ‘, “

```
PreparedStatement pstmt = conn.prepareStatement  
    ("SELECT website FROM shops " +  
     "WHERE name = ? OR owner = ? ");
```

...

```
pstmt.setString(1, "George's");  
pstmt.setString(2, "Oh \"wow\"!");
```

...

Parameterized queries - PreparedStatement

No need to worry about quotes ‘, “

```
PreparedStatement pstmt = conn.prepareStatement  
    ("SELECT website FROM shops " +  
     "WHERE name = ? OR owner = ? ");
```

...

```
pstmt.setString(1, "George's"); < Single quotes without escaping!  
pstmt.setString(2, "Oh \"wow\"!"); Parameterizing lets plan be cached
```

...

```
Statement stmt = conn.createStatement();  
ResultSet rs = stmt.executeQuery  
    ("SELECT website FROM shops "  
     + "WHERE name = 'George's' OR ...");
```

Must escape single quotes.
What if this came from user?

Today

- Database updates
- Java Database Connectivity
- Transactions in SQL

SQL transactions

2 reasons to put related actions in a transaction:

- *Recovery*: either everything happens, or nothing does
- *Concurrency control*: make sure unrelated actions don't interfere with each other

In project 2, we mostly need the latter

SQL transaction syntax

Start a transaction:

- Standard/Postgres: `START TRANSACTION;`
- Postgres/SQL Server: `BEGIN TRANSACTION;`

Commit the transaction:

`COMMIT;`

Abort the transaction:

`ROLLBACK;`

By default: “auto-commit” (no transaction used)

Transactions in JDBC – option 1

Execute the SQL code to start, end transactions:

```
PreparedStatement pBeginTx    =  
    con.prepareStatement("BEGIN TRANSACTION");  
PreparedStatement pCommitTx  =  
    con.prepareStatement("COMMIT");  
PreparedStatement pRollbackTx = con.prepareStatement  
("ROLLBACK");  
...  
pBeginTx.executeUpdate();  
// transaction started  
...  
  
if (ok) pCommitTx.executeUpdate();  
else pRollbackTx.executeUpdate();  
// transaction finished or reverted
```

Transactions in JDBC – option 2

Use JDBC methods to work with transactions:

```
conn.setAutoCommit(false);  
// From now on, everything is in a transaction  
...  
  
if (ok) conn.commit();  
else conn.rollback();  
// Old transaction done/reverted, new one started  
...  
  
conn.setAutoCommit(true);  
// Now each statement executes by itself again
```