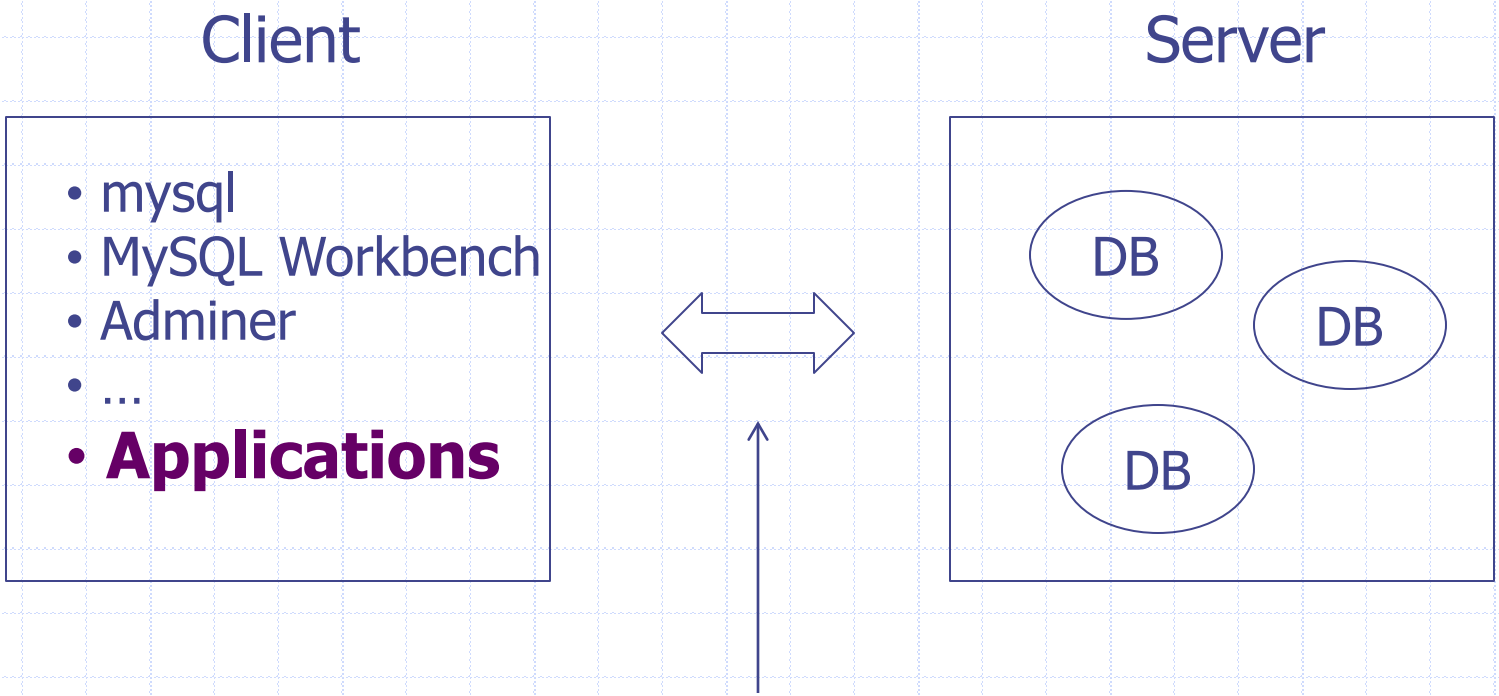
CS3220 Web and Internet Programming

Database Access with JDBC

Chengyu Sun

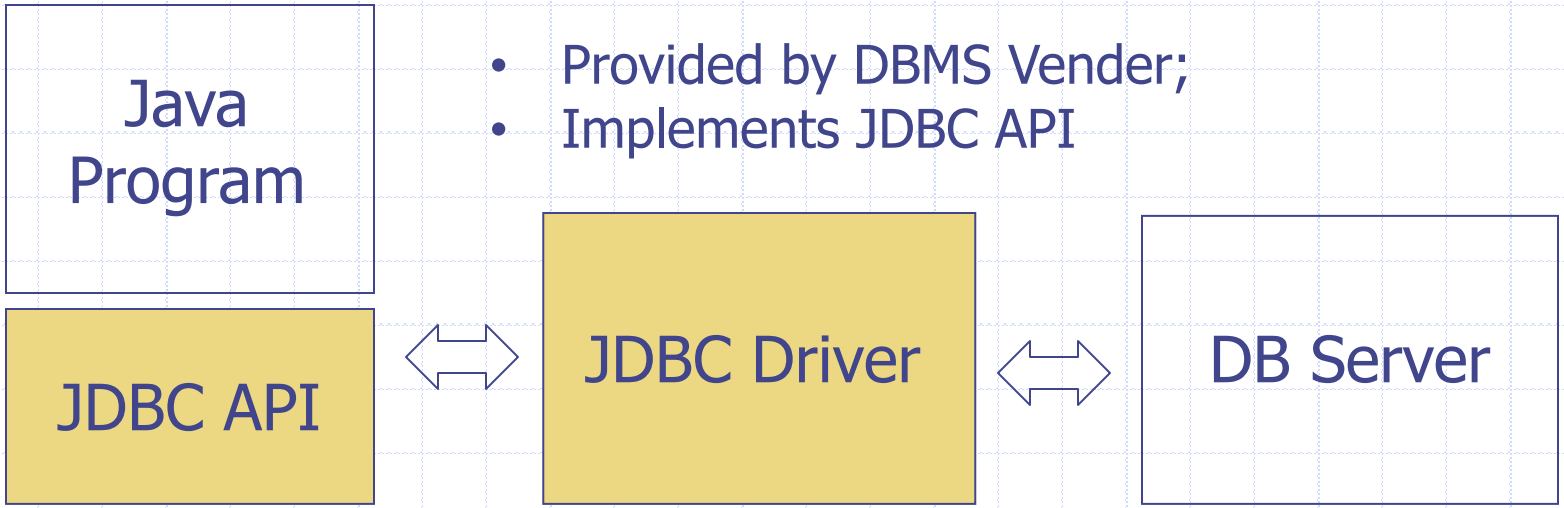
California State University, Los Angeles

Client-Server Architecture of Databases



A *library* that sends SQL and other commands from client to server, and get the results from server to client.

Java DataBase Connectivity (JDBC)



- Provided by DBMS Vender;
- Implements JDBC API

- Part of JDK
- DBMS Independent

MySQL JDBC Driver

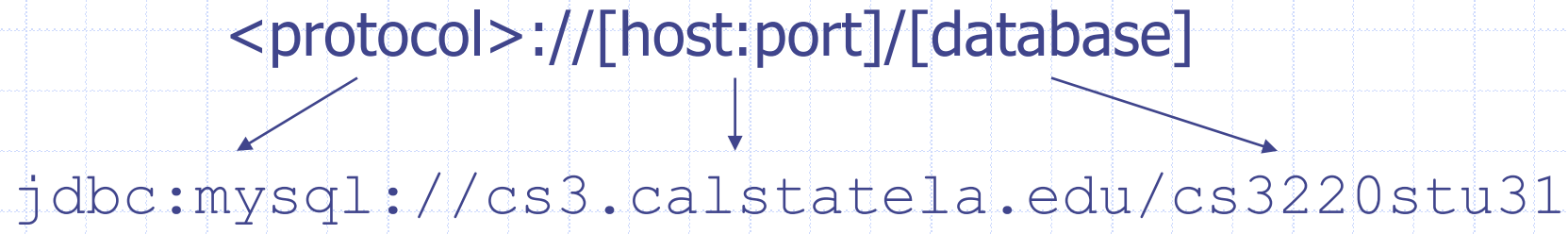
- https://dev.mysql.com/downloads/connector/j/
- The jar file should be placed under Tomcat's `/lib` folder

Example: HelloJDBC.java

- Make connections
- Execute SQL statements
- Process results
- Handle exceptions and close connections

Making Connections …

Connection URL (a.k.a. Connection String):



- DriverManager.getConnection(url, username, password)

… Making Connections

- Username and password can also be specified as request parameters in connection URL
- MySQL 8 default authentication plugin may require additional request parameters (see example)

Executing SQL Statements

- Statement stmt = c.creaetStatement()
  - stmt.executeQuery(String sql)
  - stmt.executeUpdate(String sql)
- Difference between *query* and *update*??

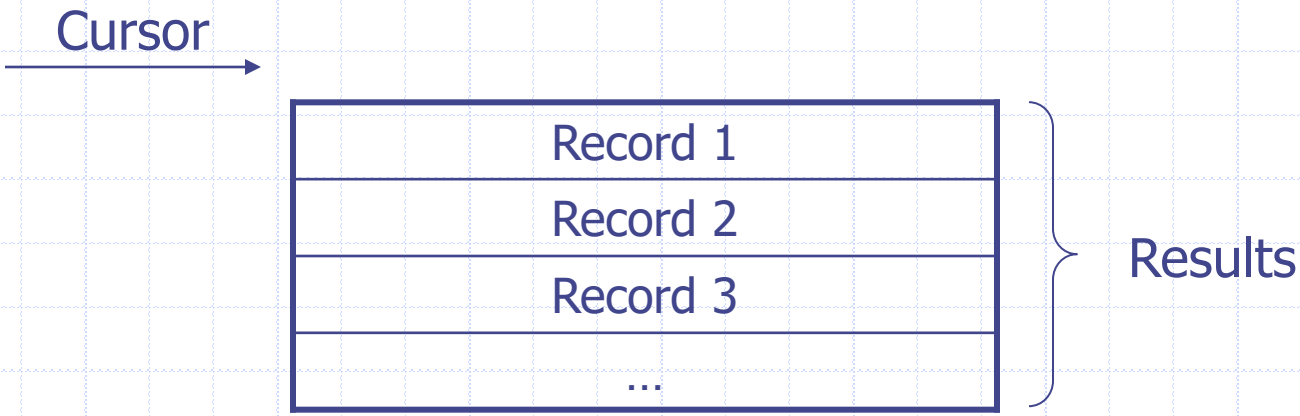
DB Query Results

- In a program, we want to
  - Access each record
  - Access each attribute in a record
  - Access the name of each attribute

select * from items;

name	price	quantity
Milk	3.89	2
Beer	6.99	1

JDBC ResultSet – Row Access



- `next()` – move cursor down one row
  - Cursor starts from *before the 1st record*
  - `true` if the current record is valid
  - `false` if no more records

Common Code for Processing ResultSet

- Process each row
  - `while(rs.next()) {…}`
- Check whether a result set is empty
  - `if(rs.next()) {…}`

JDBC ResultSet – Column Access

- Access the columns of *current row*
- getXxx(String columnName)
  - E.g. getString("user");
- getXxx(int columnIndex)
  - columnIndex starts from 1
  - E.g. getString(1);

https://docs.oracle.com/javase/8/docs/api/java/sql/ResultSet.html

JDBC ResultSet – Access Column Names

```
ResultSetMetaData meta = rs.getMetaData();
```

- ResultSetMetaData
  - getColumnName(columnIndex)
    - Column name
  - getColumnLabel(columnIndex)
    - Column title for display or printout

Handle Exceptions

```
catch( SQLException e )
{
    e.printStackTrace();
}
finally
{
    try
    {
        if( c != null ) c.close();
    }
    catch( SQLException e )
    {
        e.printStackTrace();
    }
}
```

About MySQL Connections to CS3

- Each account can have at most **4** concurrent MySQL connections to the server
- Close a connection in code after you are done using it
  - You can run as many statements as you want using the same connection
- Avoid using multiple tabs in MySQL Workbench because each tab will take up one connection

Example: GuestBook (JDBC) – Display

- Create a `guestbook` table
- Retrieve the entries from database, *and convert them into List<GuestBookEntry>*
- Display the entries in a JSP – same as before

Example: GuestBook (JDBC) – Add

- Save new guest book entries to database
  - executeQuery() vs. executeUpdate()
  - Get auto-generated IDs after an insert

Getting Auto-Generated IDs

- `Statement`
  - int executeUpdate(sql, autoGeneratedKeys)
  - ResultSet getGeneratedKeys()
- `PreparedStatement`
  - connection.prepareStatement(sql, autoGeneratedKeys)

Potential Problems

- Special characters, e.g. `O'Brien`
- SQL Injection attack

Example: SQL Injection Attack

- User input should NOT be trusted
- Regular user input
  - Username: `john`
  - Password: `abc`
- Malicious user input
  - Username: `something`
  - Password: `something' or '1' = '1`
- *Prevent SQL injection attack?*

Prepared Statements

- Statements with parameters

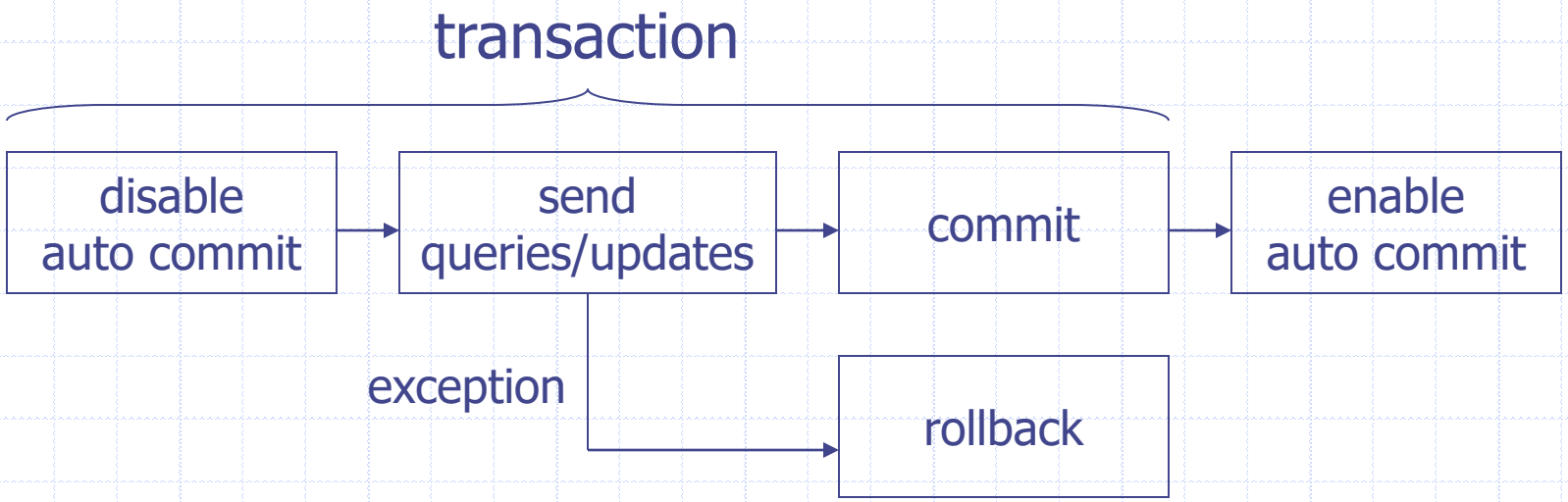
```
String sql = "insert into items values (?, ?, ?)";

PreparedStatement pstmt =c.prepareStatement(sql);

pstmt.setString(1, "orange");
pstmt.setBigDecimal(2, 0.59);
pstmt.setInt(3, 4);

pstmt.executeUpdate();
```

Benefits of Prepared Statements

- Special characters are properly handled
- Secure if the SQL statement is constructed from user input
- The SQL statement is more readable
- Better performance

Beyond the Basics ...

- Transaction
  - ACID



… Beyond the Basics …

- It's rather expensive to open a db connection
  - So how about once we open a connection, we leave it open forever??
- Connection Pool
  - Max number of connections
  - Max number of idle connections
  - And many other configurable parameters
  - http://tomcat.apache.org/tomcat-9.0-doc/jndi-datasource-examples-howto.html

… Beyond the Basics

- Mismatch between an OO design and a relational design
- Object-Relational Mapping
  - JPA and Hibernate - http://hibernate.org/orm/