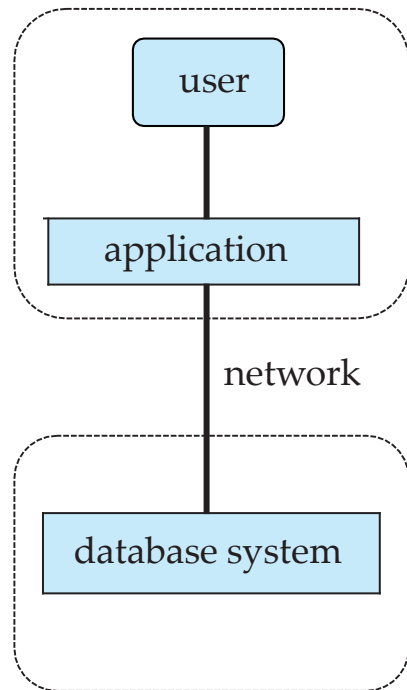


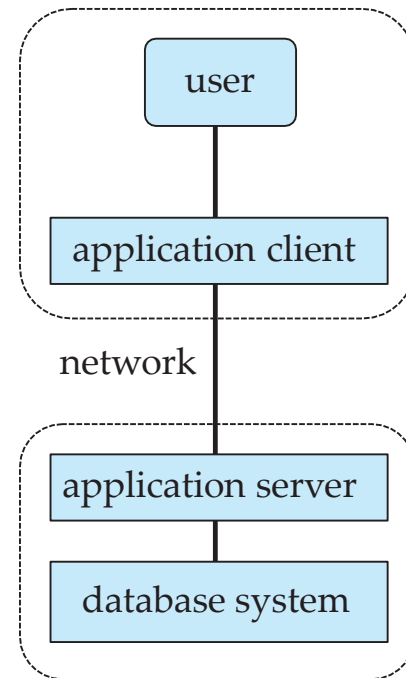
# Client-server Architectures

- ▶ Many different possibilities to build end-to-end app
  - 2-tier and 3-tier architectures are popular



(a) Two-tier architecture

client



server

(b) Three-tier architecture

# Three-tier architecture

## Presentation tier

The top-most level of the application is the user interface. The main function of the interface is to translate tasks and results to something the user can understand.



## Logic tier

This layer coordinates the application, processes commands, makes logical decisions and evaluations, and performs calculations. It also moves and processes data between the two surrounding layers.



GET LIST OF ALL  
SALES MADE  
LAST YEAR



ADD ALL SALES  
TOGETHER

QUERY

SALE 1  
SALE 2  
SALE 3  
SALE 4

## Data tier

Here information is stored and retrieved from a database or file system. The information is then passed back to the logic tier for processing, and then eventually back to the user.



Database



Storage

- ▶ E.g. browser
- ▶ E.g. Ruby on Rails, Java EE, ASP.NET, PHP, ColdFusion, etc.
- ▶ E.g. PostgreSQL, Oracle, MySQL, SQL Server, etc.

# JDBC and ODBC

- ▶ API (application-program interface) for a program to interact with a database server
- ▶ Application makes calls to
  - Connect with the database server
  - Send SQL commands to the database server
  - Fetch tuples of result one-by-one into program variables
- ▶ ODBC (Open Database Connectivity) works with C, C++, C#, and Visual Basic
  - Other API's such as ADO.NET sit on top of ODBC
- ▶ JDBC (Java Database Connectivity) works with Java

# JDBC Code

```
public static void JDBCexample(String dbid, String userid, String passwd)
{
    try (Connection conn = DriverManager.getConnection(
        "jdbc:oracle:thin:@db.umd.edu:2000:univdb", userid, passwd);
        Statement stmt = conn.createStatement();
    )
    {
        ... Do Actual Work ....
    }
    catch (SQLException sqle) {
        System.out.println("SQLException : " + sqle);
    }
}
```

**NOTE:** Above syntax works with Java 7, and JDBC 4 onwards.

Resources opened in “try (...)” syntax (“try with resources”) are automatically closed at the end of the try block

# JDBC Code (Cont.)

- ▶ Update to database

```
try {  
    stmt.executeUpdate(  
        "insert into instructor values('77987', 'Kim', 'Physics', 98000)");  
} catch (SQLException sqle)  
{  
    System.out.println("Could not insert tuple. " + sqle);  
}
```

- ▶ Execute query and fetch and print results

```
ResultSet rset = stmt.executeQuery(  
    "select dept_name, avg (salary)  
    from instructor  
    group by dept_name");  
while (rset.next()) {  
    System.out.println(rset.getString("dept_name") + " " +  
        rset.getFloat(2));  
}
```

# JDBC Code Details

- ▶ Getting result fields:
  - `rs.getString("dept_name")` and `rs.getString(1)` equivalent if `dept_name` is the first argument of select result.
- ▶ Dealing with Null values  
`int a = rs.getInt("a");`  
`if (rs.isNull()) Systems.out.println("Got null value");`

# Metadata Features

- ▶ ResultSet metadata
- ▶ E.g. after executing query to get a ResultSet rs:
  - `ResultSetMetaData rsmd = rs.getMetaData();`  
    `for(int i = 1; i <= rsmd.getColumnCount(); i++) {`  
        `System.out.println(rsmd.getColumnName(i));`  
        `System.out.println(rsmd.getColumnTypeName(i));`  
    `}`
- ▶ How is this useful?

# Prepared Statement

- ▶ `PreparedStatement pStmt = conn.prepareStatement("insert into instructor values(?,?,?,?)");`  
`pStmt.setString(1, "88877");`  
`pStmt.setString(2, "Perry");`  
`pStmt.setString(3, "Finance");`  
`pStmt.setInt(4, 125000);`  
`pStmt.executeUpdate();`  
`pStmt.setString(1, "88878");`  
`pStmt.executeUpdate();`
- ▶ WARNING: always use prepared statements when taking an input from the user and adding it to a query
  - NEVER create a query by concatenating strings
  - `"insert into instructor values(' " + ID + " ', ' " + name + " ', " + " ' + dept name + " ', " ' salary + ")"`
  - What if name is "D'Souza"?



# SQL Injection

- ▶ Suppose query is constructed using
  - "select \* from instructor where name = " + name + ""
- ▶ Suppose the user, instead of entering a name, enters:
  - X' or 'Y' = 'Y
- ▶ then the resulting statement becomes:
  - "select \* from instructor where name = " + "X' or 'Y' = 'Y" + ""
  - which is:
    - select \* from instructor where name = 'X' or 'Y' = 'Y'
  - User could have even used
    - X'; update instructor set salary = salary + 10000; --
- ▶ Prepared statement internally uses:  
"select \* from instructor where name = 'X\' or \'Y\' = \'Y'"
  - **Always use prepared statements, with user inputs as parameters**

# SQL Injection: XKCD

