PAYAP UNIVERSITY

SQL in a Server Environment

Vaidė Narváez

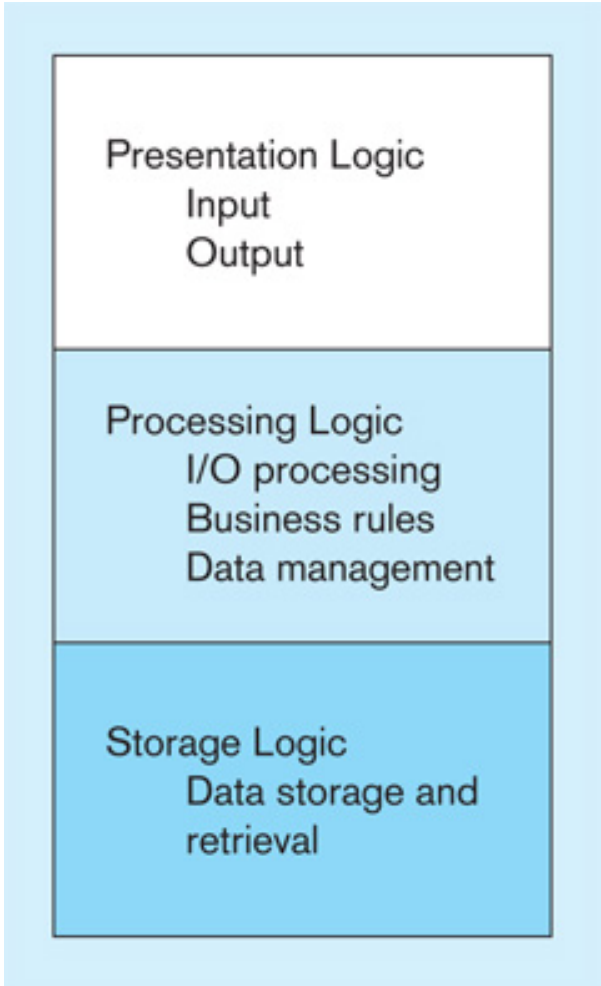
Computer Information Systems

January 13th, 2011

The Three-Tier Architecture

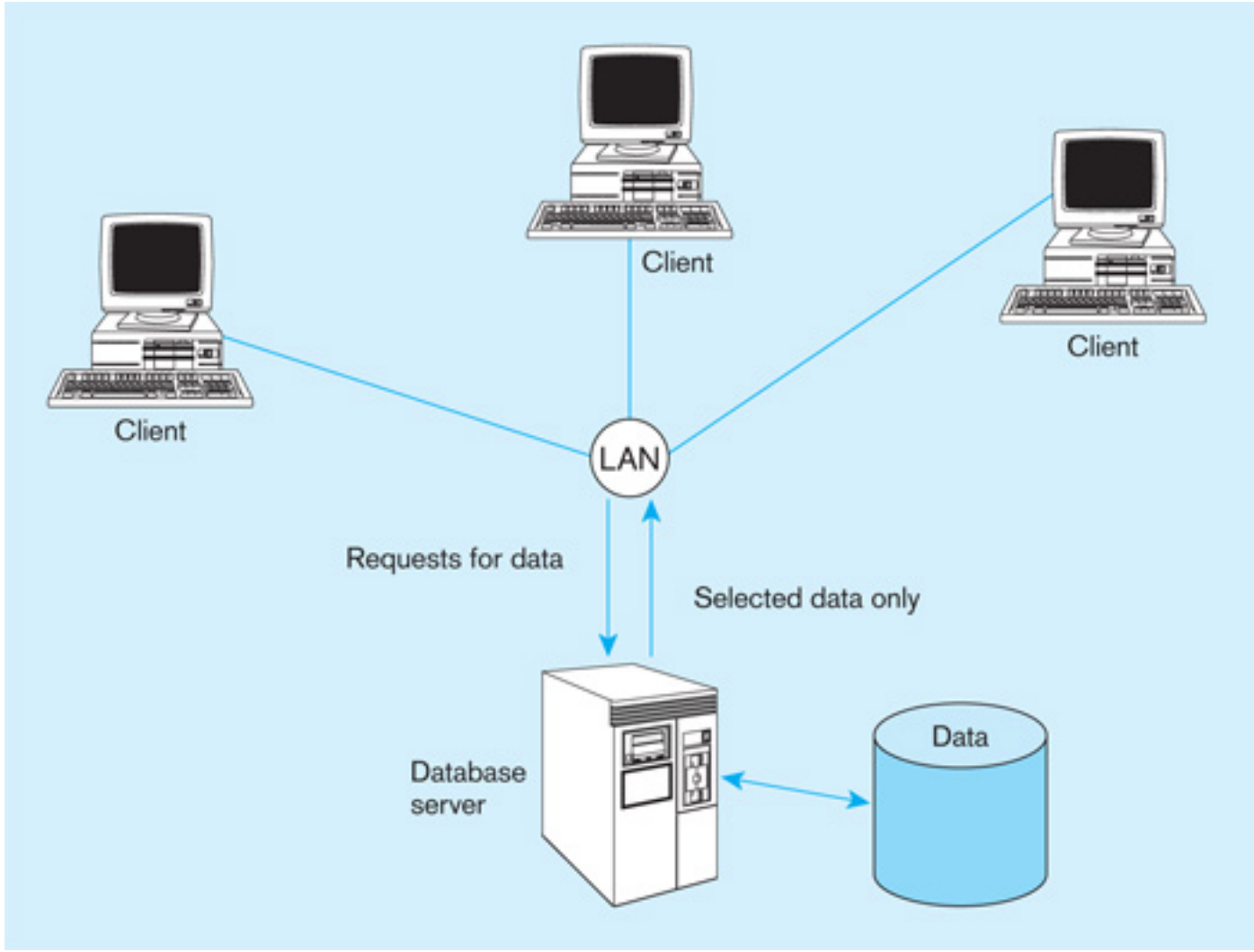
PAYAP UNIVERSITY

Application logic components



Database Server Architecture

Two-tier architecture



PAYAP UNIVERSITY

Two-tier architecture

- Client is responsible for
  - I/O processing logic
  - Some business rules logic
- Server performs all data storage and access processing

Advantages of two-tier architectures

- Clients do not have to be very powerful
- Greatly reduces data traffic on the network
- Improved data integrity since it is all processed centrally

PAYAP UNIVERSITY

Three-tier architectures

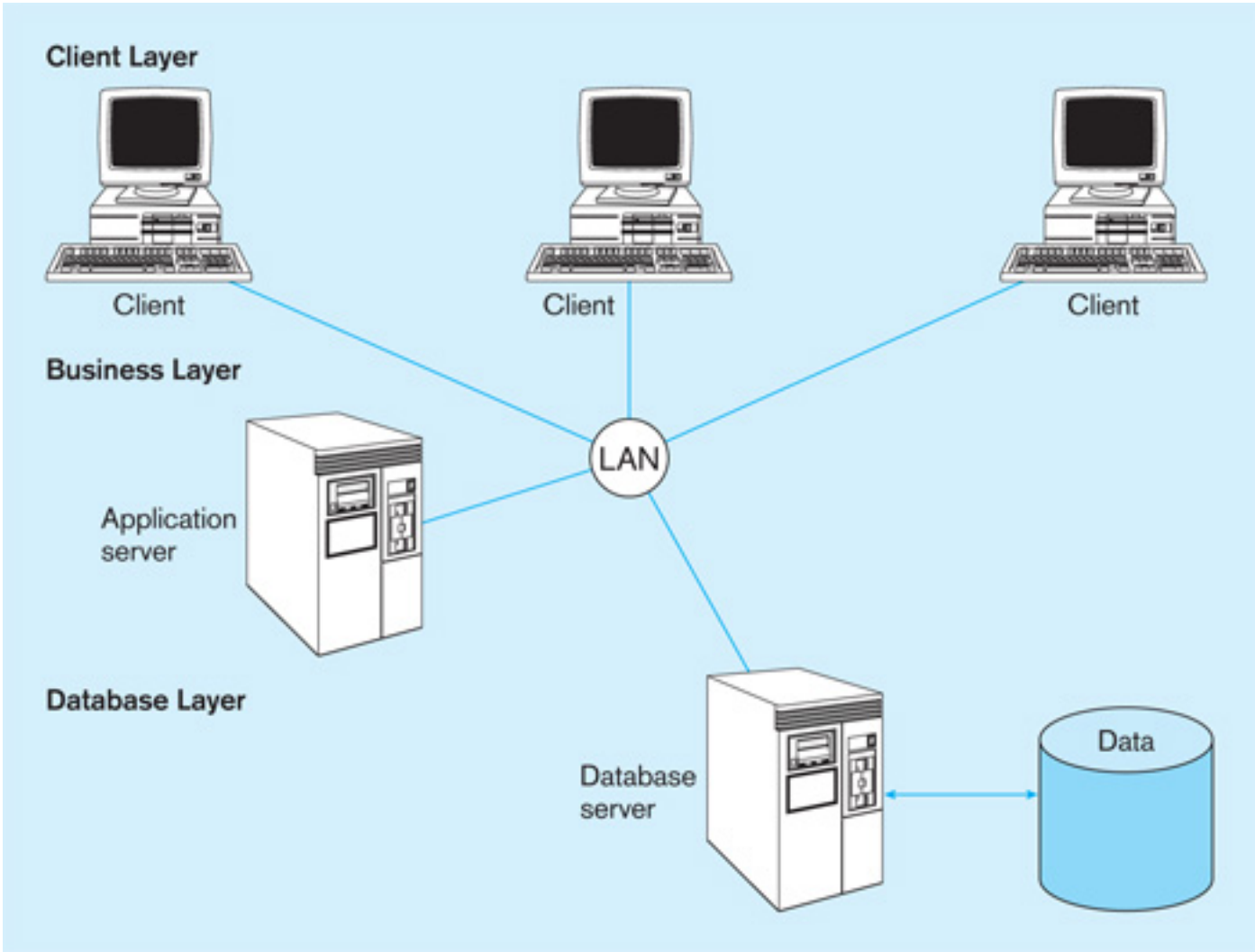
A common environment for using a database has three tiers of processors:

- *Web servers* — talk to the user.
- *Application servers* — execute the business logic.
- *Database servers* — get what the app servers need from the database.

Client	GUI, IO	Browser
Application Server	Business Rules	Web Server
Database Server	Data storage	DBMS

PAYAP UNIVERSITY

Three-tier Architecture



PAYAP UNIVERSITY

The Web Server

The web server processes manage interactions with the user

Example?

PAYAP UNIVERSITY

The Application Tier

The job of the application tier is to turn data, from the database, into a response to the request that it receives from the web server.
(perfoms *business logic* of the organization)

PAYAP UNIVERSITY

The Database Tier

The database tier executes queries that are requested from the application tier and may also provide some buffering of data.

Keep a large number of connections open!

JDBC

PAYAP UNIVERSITY

Introduction to ODBC/JDBC

ODBC - Open Database Connectivity
JDBC - Java Database Connectivity

ODBC/JDBC expose database capabilities in a standardized way to the application developer through an API.

requires a DBMS-specific driver

PAYAP UNIVERSITY

Architecture of JDBC

1. Application (initiates and terminates connections, submits SQL statements)
2. Driver manager (load JDBC driver)
3. Drivers (connects to data source, transmits requests and returns/translates results and error codes)
4. Data sources or DBMSs (processes SQL statements)

PAYAP UNIVERSITY

JDBC classes and interfaces

java.sql
javax.sql

```
import java.sql.*
```

Steps to submit a database query:

- Load the JDBC driver
- Connect to the data source
- Execute SQL statements

PAYAP UNIVERSITY

JDBC Driver Management

All drivers are managed by the *DriverManager* class

Loading a JDBC driver:

- In the Java code:
  `Class.forName("com.mysql.jdbc.Driver");`
- When starting the Java application:
  `-Djdbc.drivers=com.mysql.jdbc.Driver`

PAYAP UNIVERSITY

Connections in JDBC

We interact with a data source through sessions. Each connection identifies a logical session.

JDBC URL:
`jdbc:<subprotocol>:<otherParameters>`

Example:

```
1
2       String url="jdbc:mysql://localhost/sailing";
3
4       Connection con;
5
6               try{
7                       con =DriverManager.getConnection(url,userId,password);
8               }catch(SQLException excpt){
9                       System.out.println(excpt.getMessage());
10                      return;
11              }
```

PAYAP UNIVERSITY

Connection Class Interface

- `public int getTransactionIsolation()` and `void setTransactionIsolation(int level)`
  Sets isolation level for the current connection.
  TRANSACTION_READ_UNCOMMITTED
  TRANSACTION_READ_COMMITTED
  TRANSACTION_REPEATABLE_READ
  TRANSACTION_SERIALIZABLE

PAYAP UNIVERSITY

Connection Class Interface

- `public int getTransactionIsolation()` and
  `void setTransactionIsolation(int level)`
  Sets isolation level for the current connection.
  TRANSACTION_READ_UNCOMMITTED
  TRANSACTION_READ_COMMITTED
  TRANSACTION_REPEATABLE_READ
  TRANSACTION_SERIALIZABLE
- `public boolean getReadOnly()` and
  `void setReadOnly(boolean b)`
  Specifies whether transactions in this connection are read-only

PAYAP UNIVERSITY

Connection Class Interface

- `public int getTransactionIsolation()` and
  `void setTransactionIsolation(int level)`
  Sets isolation level for the current connection.
  TRANSACTION_READ_UNCOMMITTED
  TRANSACTION_READ_COMMITTED
  TRANSACTION_REPEATABLE_READ
  TRANSACTION_SERIALIZABLE
- `public boolean getReadOnly()` and
  `void setReadOnly(boolean b)`
  Specifies whether transactions in this connection are read-only
- `public boolean getAutoCommit()` and
  `void setAutoCommit(boolean b)`
  If autocommit is set, then each SQL statement is considered its own transaction. Otherwise, a transaction is committed using commit(), or aborted using rollback().

PAYAP UNIVERSITY

Connection Class Interface

- `public int getTransactionIsolation()` and
  `void setTransactionIsolation(int level)`
  Sets isolation level for the current connection.
  TRANSACTION_READ_UNCOMMITTED
  TRANSACTION_READ_COMMITTED
  TRANSACTION_REPEATABLE_READ
  TRANSACTION_SERIALIZABLE
- `public boolean getReadOnly()` and
  `void setReadOnly(boolean b)`
  Specifies whether transactions in this connection are read-only
- `public boolean getAutoCommit()` and
  `void setAutoCommit(boolean b)`
  If autocommit is set, then each SQL statement is considered its own transaction. Otherwise, a transaction is committed using commit(), or aborted using rollback().
- `public boolean isClosed()`
  Checks whether connection is still open.

PAYAP UNIVERSITY

Executing SQL Statements

Three different ways of executing SQL statements:

- Statement(both static and dynamic SQL statements)
- PreparedStatement(semi-static SQL statements)
- CallableStatement(stored procedures)

`PreparedStatement` class:
Precompiled, parametrizedSQL statements:

- Structure is fixed
- Values of parameters are determined at run-time

PAYAP UNIVERSITY

Prepared statements example

```
1
2           int sid =10;
3           String sname="Thomas"
4           int rating =4;
5           Double age = 20.2;
6
7           String sql="INSERT INTO Sailors VALUES(?,?,?,?)";
8
9           PreparedStatment pstmt=con.prepareStatement(sql);
10
11                  pstmt.clearParameters();
12                  pstmt.setInt(1,sid);
13                  pstmt.setString(2,sname);
14                  pstmt.setInt(3, rating);
15                  pstmt.setFloat(4,age);
16  // we know that no rows are returned, thus we use executeUpdate()
17          int numRows=pstmt.executeUpdate();
```

PAYAP UNIVERSITY

ResultSet

`PreparedStatement.executeUpdate()` only returns the number of affected records

`PreparedStatement.executeQuery()` returns data, encapsulated in a `ResultSet` object.

```
1 ResultSet rs=pstmt.executeQuery(sql);
2 // rs is now a cursor
3         while (rs.next()) {
4             // process the data
5          }
```

PAYAP UNIVERSITY

Resultset cont.

We can move in the query answer as follows:

- `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()`

PAYAP UNIVERSITY

A (semi) complete example

```
1       ...
2     String url="jdbc:mysql://localhost/sailing";
3        // connect
4     Connection con =  DriverManager.getConnection(url, "login", "pass");
5
6     Statement stmt = con.createStatement(); // set up stmt
7
8     String query = "SELECT name, rating FROM Sailors";
9
10    ResultSet rs= stmt.executeQuery(query);
11
12            try { // handle exceptions
13                  // loop through result tuples
14                  while (rs.next()) {
15                                    String s =rs.getString("name");
16                                    int n =rs.getFloat("rating");
17                                   System.out.println(s + "   " + n);
18                   }
19               } catch(SQLExceptionex) {
20                  System.out.println(ex.getMessage() + ex.getSQLState()
21                  + ex.getErrorCode());
22               }
23                ...
```