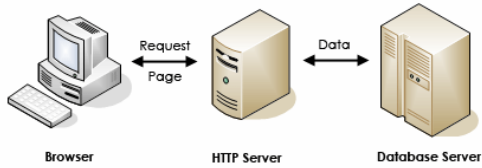


Three Tier Architecture



1

RDBMS

- Relational Database Management Systems
- A way of saving and accessing data on persistent (disk) storage.

2

Why Use an RDBMS

- Data Safety
 - data is immune to program crashes
- Concurrent Access
 - atomic updates via transactions
- Fault Tolerance
 - replicated dbs for instant fail-over on machine/disk crashes
- Data Integrity
 - aids to keep data meaningful
- Scalability
 - can handle small/large quantities of data in a uniform manner
- Reporting
 - easy to write SQL programs to generate arbitrary reports

3

Relational Model

- First published by Edgar F. Codd in 1970
 - Received Turing Award in 1981
- A relational database consists of a collection of tables
- A table consists of rows and columns
- Each row represents a record
- Each column represents an attribute of the records contained in the table

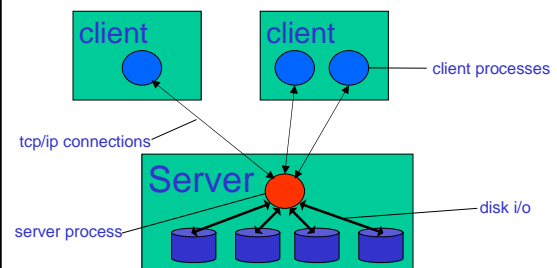
4

RDBMS Technology

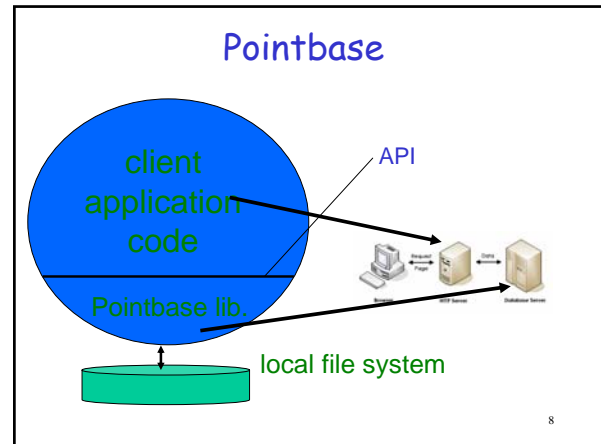
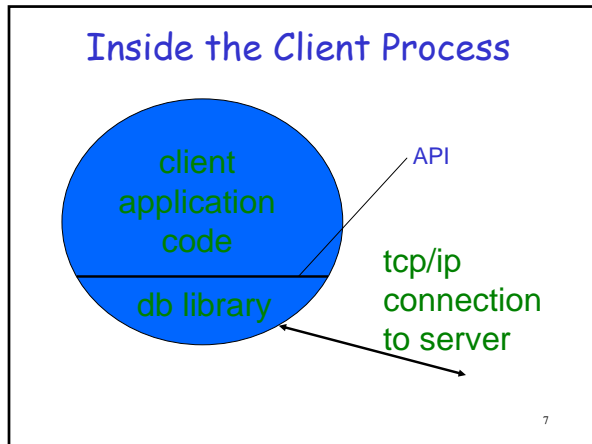
- Client/Server Databases
 - Oracle, Sybase, MySQL, SQLServer
- Personal Databases
 - Access
- Embedded Databases
 - Pointbase

5

Client/Server Databases



6

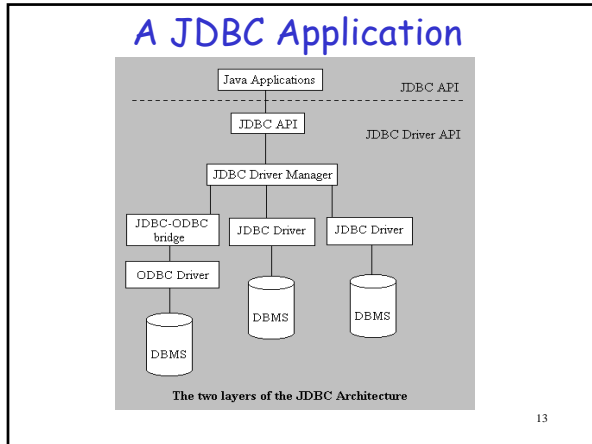


- ## Pointbase
- Libraries
 - pbembedded41ev.jar Implement database
 - pbtools41ev.jar DB management tools
 - Location on cdf
 /u/csc309h/lib/tomcat-5.0.28/common/lib
 - Tools
 - Make sure that both pbembedded41ev.jar and pbtools41ev.jar are in your CLASSPATH
 - Commander
 - Command-line utility
 - java com.pointbase.tools.toolsCommander
 - Console
 - Window-based utility
 - java com.pointbase.tools.toolsConsole

- ## Pointbase Configuration
- pointbase.ini
 - Configuration file
 - database.home=/h/u1/delara/tomcat/webapps/jdbc
 - Database file location
 - Pointbase expects pointbase.ini to be in the directory where the application loading the driver is located
 - Arbitrary java applications
 - e.g., com.pointbase.tools.toolsConsole
 - The directory from where we invoke the application
 - Tomcat web application
 - Directory from which Tomcat is launched
 - SCATALINA_HOME/bin

- ## Simple DB Manipulation
- Create a table
 create table student (id integer, name character (40) ,
 birthday date);
 - Add record
 insert into student values (1, 'John', date '1980-1-1');
 - Query table
 select * from student;
- Notes:
- Commander and Console require a ";" after each SQL command
 - By default, Commander runs with autocommit set to false
 set autocommit 1;

- ## JDBC
- JDBC (Java DataBase Connectivity)
 - Standard SQL database access interface.
 - Allows a Java program to issue SQL statements and process the results.
 - DB independent. Can replace underlying database with minimal code impact.
 - Defines classes to represent constructs such as database connections, SQL statements, result sets, and database metadata.



- ## JDBC Pieces
- A JDBC application consists of
 - Java client: code implementing the application
 - JDBC API. Provides DB independent abstraction to
 - establish a connection with a database
 - send SQL statements
 - process the results
- 14

- ## JDBC Pieces Continued
- JDBC Driver
 - Translates API calls to requests made against the specific database.
 - Specific driver for the each database.
 - Installed on the client. Usually a set of class files placed in the class path.
 - e.g., pbembedded41ev.jar
 - All large databases are now supported.
 - Database client software (optional)
 - Establishes communication between DB clients and the DB server.
 - Depending on the DB Manufacturer, you may need to install database client software on the client machine.
 - JDBC Driver may already include this functionality
- 15

- ## JDBC Pieces Continued
- Database server:
 - The actual database engine
 - Pointbase, Oracle, MSAccess, SQL Server, Postgresql etc.
- 16

- ## Alternatives (ODBC)
- **What is ODBC**
 - Microsoft's version of JDBC.
 - Many drivers exist for ODBC.
 - Sun provides a JDBC-ODBC driver to allow Java applications connectivity to databases that are only ODBC enabled.
- 17

- ## JDBC
- Package `java.sql`
java.sun.com/products/jdk/1.2/docs/api/java/sql/package-summary.html
 - **Classes and interfaces**
 - `DriverManager` `getConnection()`
 - `Connection` `Transactions`
 `commit(), rollback(), setAutoCommit()`
 `SQL statements`
 `createStatement(), prepareStatement()`
 - `Statement` `executeQuery(), executeUpdate()`
 - `ResultSet` `Scrolling over tuples`
 `next(), previous(), first(), last(), isLast()`
 `Values of attributes`
 `getString(int index), getTime(String name)`
 - `PreparedStatement` `executeQuery(), executeUpdate()`
 `setInt (int index, int x)`
- 18

API: Establish Connection

- 1 `Class.forName("com.pointbase.jdbc.jdbcUniversalDriver")`
make the driver class available
- 2 `String url = "jdbc:pointbase:embedded:sample";`
This is the connect string. The connect string, mentions the driver as well as the database. For the example above, the driver is the `jdbc:pointbase:embedded` bridge driver. The database is `sample`.
- 3 `Connection con=DriverManager.getConnection(url, uID, pw);`
Get a connection (session) with a specific database. Within the context of a Connection, SQL statements are executed and results are returned.
A Connection can be used to obtain information about the DB
By default a Connection automatically *commits* changes after each statement.
Typically, setting up a connection is an expensive operation.

19

API: Executing Queries

- A query can return many rows, each with many attributes
- Steps are
 - 1 Send query to the database
 - 2 Retrieve one row at a time
 - 3 For each row, retrieve attributes

20

API: Executing Queries

Example:

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

// Send the query to the DB, get back a ResultSet
ResultSet rs = stmt.executeQuery("SELECT * FROM PART;");

// Go through all rows returned by the query
while(rs.next()){
    // Pull out individual columns from the current row int
    pno=rs.getInt("PNO");
    String pname=rs.getString("PNAME");

    // Print out the values System.out.println(pno+"\t"+pname);
}
rs.close();
```

21

API: Updates

```
Int rowsAffected=
    stmt.executeUpdate(
        "DELETE * FROM ACCOUNTS;");
```

Executes a SQL INSERT, UPDATE or DELETE statement. Returns the number of rows affected.

22

API: Prepared Statements

- Is a parameterized SQL statement.
- Used to speedup query parsing (statement does not need to be reparsed)
- Used to simplify development (clumsy strings do not have to be created and recreated for each update).

Example:

```
String insert="INSERT INTO ACCOUNT(NAME,AMOUNT)VALUES(?,?)";
PreparedStatement ps =con.prepareStatement(insert);

ps.setString(1,"Charlie"); // Fill in the first ?
ps.setDouble(2,23.45); // Fill in the second ?

rowsEffecte=ps.executeUpdate();

ps.setInt(1,"Arnold"); // Fill in the first ?
ps.setInt(2,102.23); // Fill in the second ?

rowsEffecte=ps.executeUpdate();
```

23

Trivial First Example

- Servlet + JDBC + Pointbase
- Connects to database
- Prints out name of all tables



24

Scope of DB Connections

1. **Servlet:** Open/close connection on each servlet invocation
-
2. **Session:** Keep 1 open connection associated with the session: `HttpSession.getParameter("dbCon")`
-
-
-
3. **Web App:** Keep 1 open connection for the web application and re-use it
-

25

Connection Pooling

- The Solution:
 - Maintain a pool of open connections that time themselves out
- The `SharedPoolDataSource` allocates a new `Connection` object
 - It wraps it in a `PoolingConnection` object that delegates calls to its enclosed `Connection` object and returns it to the client.
 - It sets a timeout that will reclaim the `Connection` for a pool of free connections
 - If client accesses `PoolingConnection` after timeout, `PoolingConnection` will request a fresh connection from the pool.
- Client returns `Connection` to the pool when done with it.
 - `SharedPoolDataSource` does not close the `Connection`, rather saves it for the next request
- Client may block awaiting a freed connection if some maximum upper limit of open connections is reached.

26

Connection Pooling

- Init method of main webapp servlet:

```
// Create a data source
DriverAdapterCPDS ds = new DriverAdapterCPDS();
ds.setDriver("com.pointbase.jdbc.jdbcUniversalDriver");
ds.setUrl("jdbc:pointbase:embedded:sample");
ds.setUser("public");
ds.setPassword("public");
```

```
// Create a connection pool
SharedPoolDataSource dbcp = new SharedPoolDataSource();
dbcp.setConnectionPoolDataSource(ds);
```

```
//Add to webapp context
getServletContext().setAttribute("dbConPool",dbcp);
```

27

Connection Pooling

- Each servlet `doGet()` or `doPost()` calls:

```
SharedPoolDataSource dbcp = getServletContext().getAttribute("dbConPool");
Connection con = dbcp.getConnection();
...
Statement s = con.createStatement();
...
con.close();
```

28

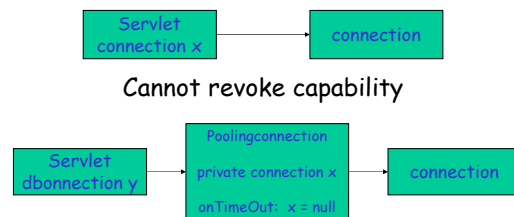
SQLGateway

| SCHEMAID | TABLENAME | TABLEID | TABLETYPE | SUBRELATION | TABLASTMODIFIED |
|----------|----------------------------|---------|-----------|-----------------------|-----------------------|
| 1 | SYNCHRONIZATIONCONSTRAINTS | 12 | 2 | 2004-11-08 09:08:18.0 | 2004-11-08 09:08:18.0 |
| 1 | SYNCHRONIZATIONINDEXES | 13 | 2 | 2004-11-08 09:08:18.0 | 2004-11-08 09:08:18.0 |
| 1 | SYNCHRONIZATION | 3 | 2 | 2004-11-08 09:08:17.0 | 2004-11-08 09:08:17.0 |
| 1 | SYNCHRONIZATIONPROPERTIES | 11 | 2 | 2004-11-08 09:08:18.0 | 2004-11-08 09:08:18.0 |

29

Indirection and Capabilities

Capability: enforces access control by providing a handle to a resource.



Cannot revoke capability

Indirection enables revoking capability 30

Atomic Transactions

- Recall ATM banking example:
 - Concurrent deposit/withdrawal operation
 - Need to protect shared account balance
- What about transferring funds between accounts?
 - Withdraw funds from account A
 - Deposit funds into account B

31

Properties of funds transfer

- Should appear as a single operation
 - Another process reading the account balances should see either both updates, or none
- Either both operations complete, or neither does
 - Need to recover from crashes that leave transaction partially complete

32

Transactions

- Definition: A transaction is a collection of DB modifications, which is treated as an atomic DB operation.
 - Transactions ensure that a collection of updates leaves the database in a consistent state (as defined by the application program); all updates take place or none do.
 - A sequence of **read** and **write** operations, terminated by a **commit** or **abort**
- Definition: Committed
 - A transaction that has completed successfully; once committed, a transaction cannot be undone
- Definition: Aborted
 - A transaction that did not complete normally
- JDBC: By default the Connection automatically commits changes after executing each statement. If auto commit has been disabled, the method `commit` must be called explicitly; otherwise, database changes will not be saved.

33

API: Transactions

Example:

```
// Change to transactional mode
con.setAutoCommit(false);
// Transaction A begins here
stmt.executeUpdate("DELETE * FROM ACCOUNT...");// 1
stmt.executeUpdate("INSERT INTO ACCOUNT ...");// 2
stmt.executeUpdate("INSERT INTO ACCOUNT ...");// 3
stmt.executeUpdate("INSERT INTO ACCOUNT ...");// 4
con.commit();
// Commit changes to database
// All of 1,2,3,4 take effect
```

34