Database Management Systems

Module 21: Application Design and Development/1

Partha Pratim Das

Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

ppd@cse.iitkgp.ernet.in

Srijoni Majumdar
Himadri B G S Bhuyan
Gurunath Reddy M

Database System Concepts, 6th Ed.

www.db-book.com

PPD

Week 04 Recap

- **Module 16: Relational Database Design/1**
  - Features of Good Relational Design
  - Atomic Domains and First Normal Form
  - Functional Dependencies
- **Module 17: Relational Database Design/2**
  - Decomposition Using Functional Dependencies
  - Functional Dependency Theory
- **Module 18: Relational Database Design/3**
  - Algorithms for Functional Dependencies
  - Lossless Join Decomposition
  - Dependency Preservation
- **Module 19: Relational Database Design/4**
  - Normal Forms
  - Decomposition to 3NF
  - Decomposition to BCNF
- **Module 20: Relational Database Design/5**
  - Multivalued Dependencies
  - Decomposition to 4NF
  - Database-Design Process
  - Modeling Temporal Data

PPD

Module Objectives

- To understand the requirements of Database Application Programs and User Interfaces
- To familiarize with the fundamentals notions and technologies of Web
- To learn about Servlets and Java Server Pages

PPD

Module Outline

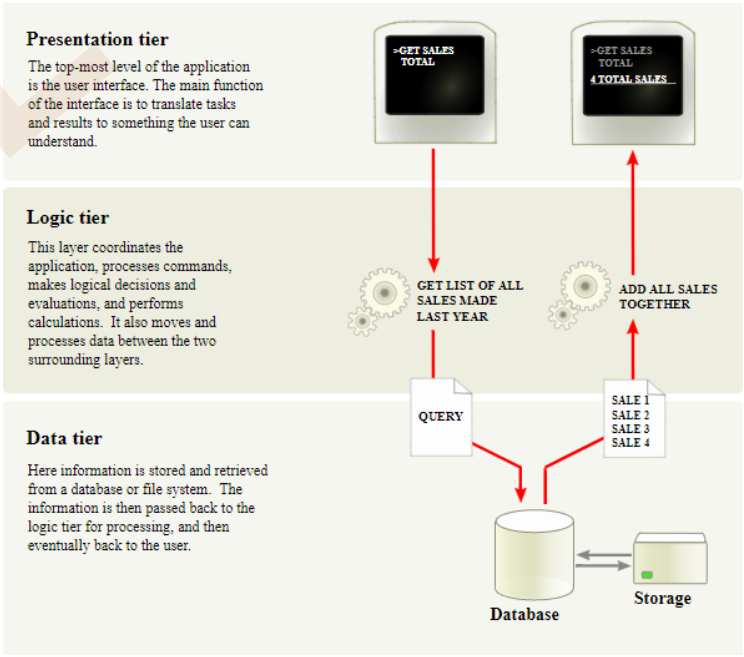
- Application Programs and User Interfaces
- Web Fundamentals
- Servlets and JSP

- **Application Programs and User Interfaces**
- Web Fundamentals
- Servlets and JSP

APPLICATION PROGRAMS AND USER INTERFACES

Application Programs and User Interfaces

- Most database users do *not* use a query language like SQL
- An application program acts as the intermediary between users and the database
  - Applications split into
    - frontend
    - middle layer
    - backend
- Frontend or Presentation Layer: user interface
  - Forms, Graphical user interfaces
  - Many interfaces are Web-based or Mobile App
- Middle Layer or Application / Business Logic Layer
  - Functionality of the Application – links front and backend
- Backend or Data Access Layer
  - Persistent data, large in volume, needs efficient access

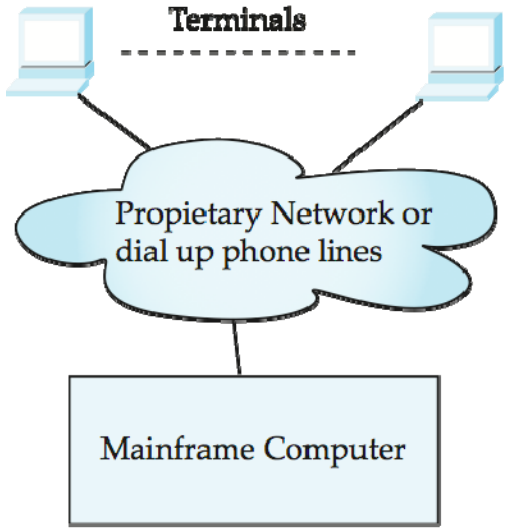


Overview of a 3-tier Architecture

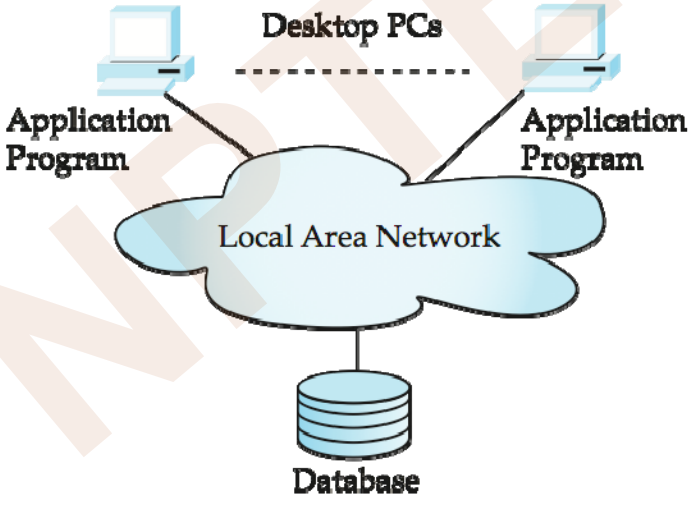
Source: https://en.wikipedia.org/wiki/Multitier_architecture

Application Architecture Evolution

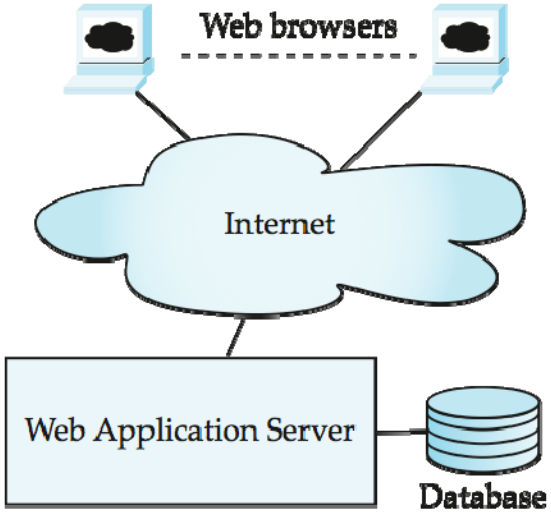
- Three distinct era's of application architecture
  - Mainframe (1960's and 70's)
  - Personal computer era (1980's)
  - Web era (1990's onwards)



(a) Mainframe Era



(b) Personal Computer Era



(c) Web era

Web Interface

- Web browsers have become the de-facto standard user interface to databases
  - Enable large numbers of users to access databases from anywhere
  - Avoid the need for downloading/installing specialized code, while providing a good graphical user interface
    - Javascript, Flash and other scripting languages run in browser, but are downloaded transparently
  - Examples: banks, airline and rental car reservations, university course registration and grading, an so on.
- Mobile Interfaces in Mobile Apps are getting popular
  - These are similar in architecture and workflow with web, but have significant differences
  - Will be discussed later

- Application Programs and User Interfaces
- **Web Fundamentals**
- Servlets and JSP

WEB FUNDAMENTALS

The World Wide Web

- The Web is a distributed information system based on hypertext
- Most Web documents are hypertext documents formatted via the HyperText Markup Language (HTML)
- HTML documents contain
  - text along with font specifications, and other formatting instructions
  - hypertext links to other documents, which can be associated with regions of the text
  - forms, enabling users to enter data which can then be sent back to the Web server

Uniform Resources Locators

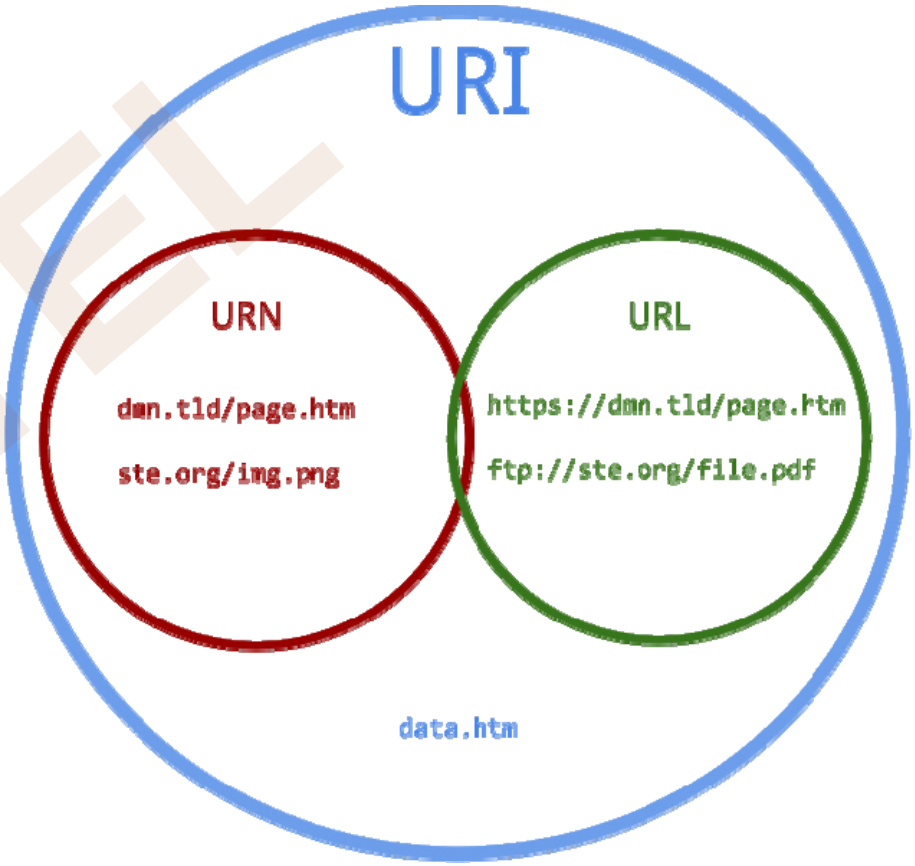
- In the Web, functionality of pointers is provided by Uniform Resource Locators (URLs).
- URL example:

  http://www.acm.org/sigmod

  - The first part indicates how the document is to be accessed (protocol)
    - "http" indicates that the document is to be accessed using the Hyper Text Transfer Protocol.
  - The second part gives the unique name of a machine on the Internet
  - The rest of the URL identifies the document within the machine
- The local identification can be:
  - The path name of a file on the machine: A file at C:/WINDOWS/media/Alarm01.wav of local machine can be accessed as:
    - file:///C:/WINDOWS/media/Alarm01.wav
    - file://localhost/c:/WINDOWS/media/Alarm01.wav
  - An identifier (path name) of a program, plus arguments to be passed to the program: Searching google.com with 'silberschatz' has the uri:
    - http://www.google.com/search?q=silberschatz

URI, URL, and URN

- Uniform Resource Identifier (URI)
- Uniform Resource Locator (URL)
- Uniform Resource Name (URN)
- Relationships:
  - URIs can be classified as locators (URLs), or as names (URNs), or as both.
  - URN functions like a person's name
  - URL resembles that person's street address.
  - URN defines an item's identity, while the URL provides a method for finding it



Source: https://danielmiessler.com/study/url-uri/

HTML and HTTP

- HTML provides formatting, hypertext link, and image display features
  - including tables, stylesheets (to alter default formatting), etc.
- HTML also provides input features
  - Select from a set of options
    - Pop-up menus, radio buttons, check lists
  - Enter values
    - Text boxes
  - Filled in input sent back to the server, to be acted upon by an executable at the server
- HyperText Transfer Protocol (HTTP) used for communication with the Web server

Sample HTML Source Text

```
<html>
<body>
 <table border>
    <tr> <th>ID</th> <th>Name</th> <th>Department</th> </tr>
    <tr> <td>00128</td> <td>Zhang</td> <td>Comp. Sci.</td> </tr>
    ….
 </table>
  <form action="PersonQuery" method=get>
    Search for:
      <select name="persontype">
         <option value="student" selected>Student </option>
         <option value="instructor"> Instructor </option>
      </select> <br>
    Name: <input type=text size=20 name="name">
    <input type=submit value="submit">
 </form>
</body> </html>
```

Display of Sample HTML Source

ID	Name	Department
00128	Zhang	Comp. Sci.
12345	Shankar	Comp. Sci.
19991	Brandt	History

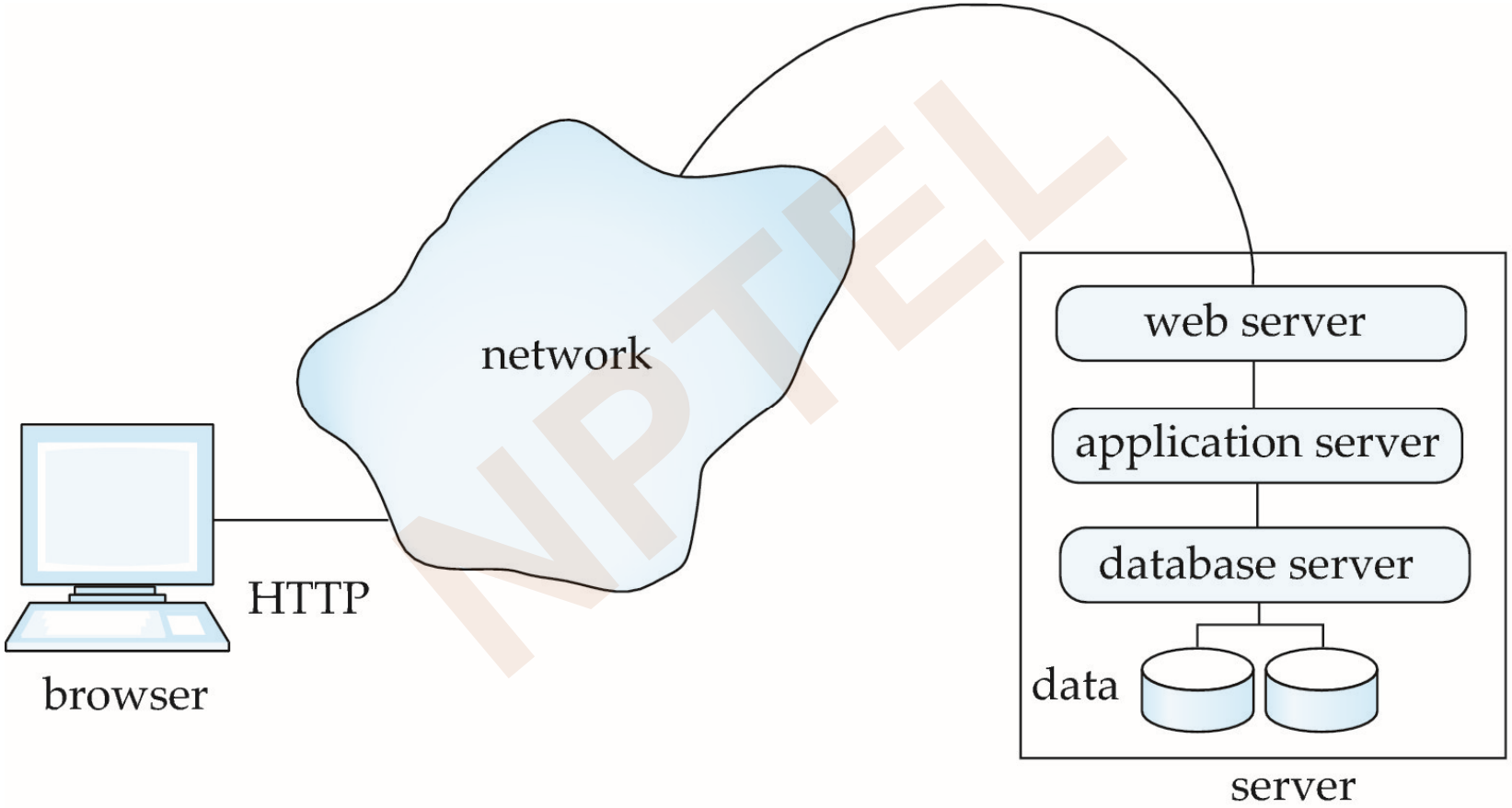
Search for: Student
Name:
submit

```
<html>
<body>
 <table border>
    <tr> <th>ID</th> <th>Name</th> <th>Department</th> </tr>
    <tr> <td>00128</td> <td>Zhang</td> <td>Comp. Sci.</td> </tr>
    ....
 </table>
 <form action="PersonQuery" method=get>
    Search for:
      <select name="persontype">
         <option value="student" selected>Student </option>
         <option value="instructor"> Instructor </option>
      </select> <br>
    Name: <input type=text size=20 name="name">
    <input type=submit value="submit">
 </form>
</body> </html>
```

Web Servers

- A Web server can easily serve as a front end to a variety of information services
- The document name in a URL may identify an executable program, that, when run, generates a HTML document
  - When an HTTP server receives a request for such a document, it executes the program, and sends back the HTML document that is generated
  - The Web client can pass extra arguments with the name of the document
- To install a new service on the Web, one simply needs to create and install an executable that provides that service
  - The Web browser provides a graphical user interface to the information service
- Common Gateway Interface (CGI): a standard interface between web and application server

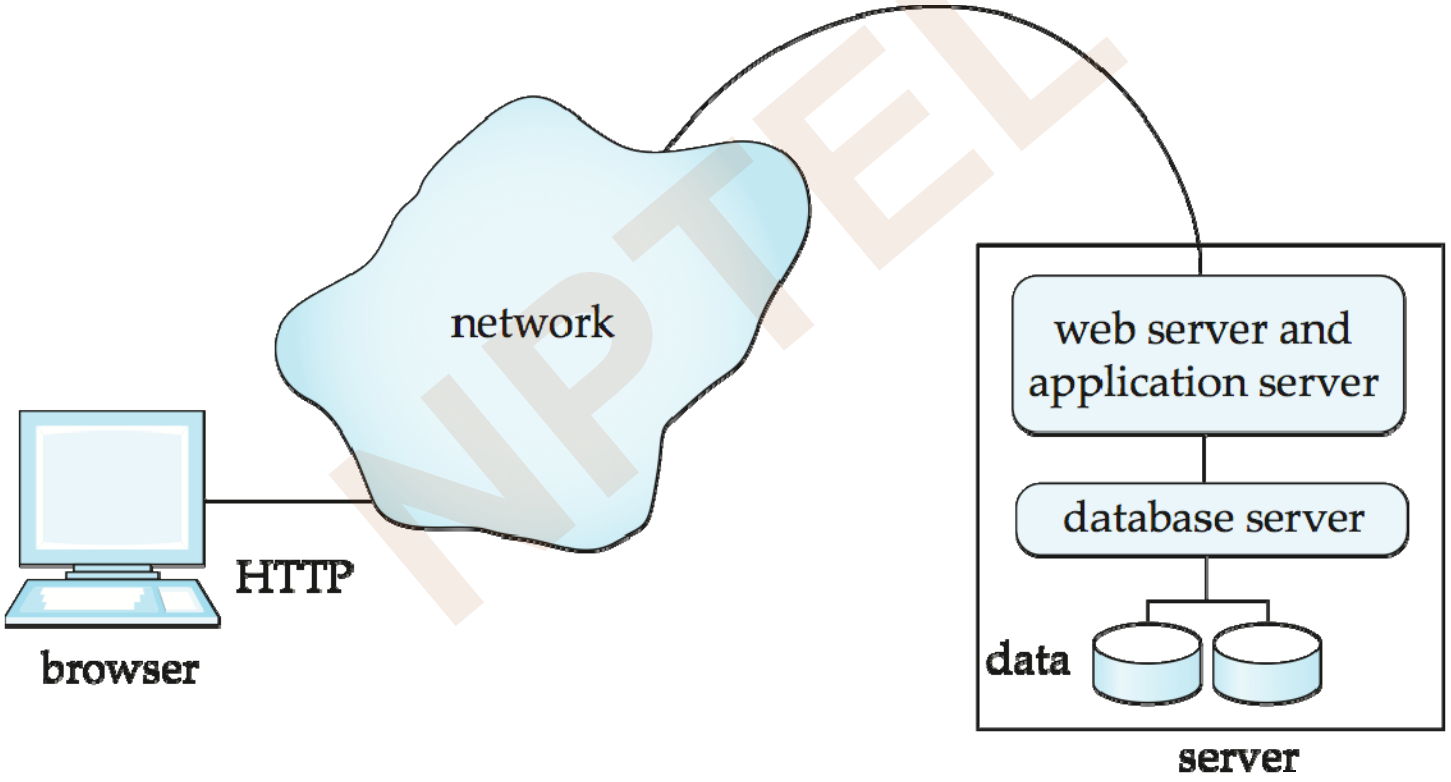
Three-Layer Web Architecture



Two-Layer Web Architecture

- Multiple levels of indirection have overheads

  Alternative: two-layer architecture



HTTP and Sessions

- The HTTP protocol is **connectionless**
  - That is, once the server replies to a request, the server closes the connection with the client, and forgets all about the request
  - In contrast, Unix logins, and JDBC/ODBC connections stay connected until the client disconnects
    - retaining user authentication and other information
  - Motivation: reduces load on server
    - operating systems have tight limits on number of open connections on a machine
- Information services need session information
  - E.g., user authentication should be done only once per session
- Solution: use a **cookie**

Sessions and Cookies

- A **cookie** is a small piece of text containing identifying information
  - Sent by server to browser
    - Sent on first interaction, to identify session
  - Sent by browser to the server that created the cookie on further interactions
    - part of the HTTP protocol
  - Server saves information about cookies it issued, and can use it when serving a request
    - E.g., authentication information, and user preferences
- Cookies can be stored permanently or for a limited time

- Application Programs and User Interfaces
- Web Fundamentals
- **Servlets and JSP**

SERVLETS AND JSP

Servlets

- Java Servlet specification defines an API for communication between the Web/application server and application program running in the server
  - E.g., methods to get parameter values from Web forms, and to send HTML text back to client
- Application program (also called a servlet) is loaded into the server
  - Each request spawns a new thread in the server
    - thread is closed once the request is serviced

Example Servlet Code

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class PersonQueryServlet extends HttpServlet {
   public void doGet (HttpServletRequest request, HttpServletResponse response)
                    throws ServletException, IOException
   {
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      out.println("<HEAD><TITLE> Query Result</TITLE></HEAD>");
      out.println("<BODY>");
         ….. BODY OF SERVLET (next slide) …
      out.println("</BODY>");
      out.close();
   }
}
```

Example Servlet Code

```
String persontype = request.getParameter("persontype");
String number = request.getParameter("name");
if(persontype.equals("student")) {
    ... code to find students with the specified name ...
    ... using JDBC to communicate with the database ..
    out.println("<table BORDER COLS=3>");
    out.println(" <tr> <td>ID</td> <td>Name: </td>" + " <td>Department</td> </tr>");
    for(... each result ...){
        ... retrieve ID, name and dept name
        ... into variables ID, name and deptname
        out.println("<tr> <td>" + ID + "</td>" + "<td>" + name + "</td>" + "<td>" + deptname
                + "</td></tr>");
    };
    out.println("</table>");
}
else {
    ... as above, but for instructors ...
}
```

Servlet Sessions

- Servlet API supports handling of sessions
  - Sets a cookie on first interaction with browser, and uses it to identify session on further interactions
- To check if session is already active:
  - if (request.getSession(false) == true)
    - .. then existing session
    - else .. redirect to authentication page
  - authentication page
    - check login/password
    - request.getSession(true): creates new session
- Store/retrieve attribute value pairs for a particular session
  - session.setAttribute("userid", userid)
  - session.getAttribute("userid")

Servlet Support

- Servlets run inside application servers such as
  - Apache Tomcat, Glassfish, JBoss
  - BEA Weblogic, IBM WebSphere and Oracle Application Servers
- Application servers support
  - deployment and monitoring of servlets
  - Java 2 Enterprise Edition (J2EE) platform supporting objects, parallel processing across multiple application servers, etc

Server-Side Scripting

- Server-side scripting simplifies the task of connecting a database to the Web
  - Define an HTML document with embedded executable code/SQL queries.
  - Input values from HTML forms can be used directly in the embedded code/SQL queries.
  - When the document is requested, the Web server executes the embedded code/SQL queries to generate the actual HTML document.
- Numerous server-side scripting languages
  - JSP, PHP
  - General purpose scripting languages: VBScript, Perl, Python

Java Server Pages (JSP)

- A JSP page with embedded Java code

```
<html>
<head> <title> Hello </title> </head>
<body>
<% if (request.getParameter("name") == null)
{ out.println("Hello World"); }
else { out.println("Hello, " + request.getParameter("name")); }
%>
</body>
</html>
```

- JSP is compiled into Java + Servlets
- JSP allows new tags to be defined, in tag libraries
  - such tags are like library functions, can are used for example to build rich user interfaces such as paginated display of large datasets

PHP

- PHP is widely used for Web server scripting
- Extensive libaries including for database access using ODBC

```
<html>
  <head> <title> Hello </title> </head>
  <body>
  <?php if (!isset($_REQUEST[‘name’]))
  { echo “Hello World”; }
  else { echo “Hello, ” + $_REQUEST[‘name’]; }
  ?>
  </body>
  </html>
```

Client Side Scripting

- Browsers can fetch certain scripts (client-side scripts) or programs along with documents, and execute them in "safe mode" at the client site
  - Javascript
  - Macromedia Flash and Shockwave for animation/games
  - VRML
  - Applets
- Client-side scripts/programs allow documents to be active
  - E.g., animation by executing programs at the local site
  - E.g., ensure that values entered by users satisfy some correctness checks
  - Permit flexible interaction with the user.
    - Executing programs at the client site speeds up interaction by avoiding many round trips to server

Client Side Scripting and Security

- Security mechanisms needed to ensure that malicious scripts do not cause damage to the client machine
  - Easy for limited capability scripting languages, harder for general purpose programming languages like Java
- E.g., Java's security system ensures that the Java applet code does not make any system calls directly
  - Disallows dangerous actions such as file writes
  - Notifies the user about potentially dangerous actions, and allows the option to abort the program or to continue execution.

Javascript

- Javascript very widely used
  - forms basis of new generation of Web applications (called Web 2.0 applications) offering rich user interfaces
- Javascript functions can
  - check input for validity
  - modify the displayed Web page, by altering the underling **document object model (DOM)** tree representation of the displayed HTML text
  - communicate with a Web server to fetch data and modify the current page using fetched data, without needing to reload/refresh the page
    - forms basis of AJAX technology used widely in Web 2.0 applications
    - E.g. on selecting a country in a drop-down menu, the list of states in that country is automatically populated in a linked drop-down menu

Javascript

- Example of Javascript used to validate form input

```
<html> <head>
    <script type="text/javascript">
        function validate() {
            var credits=document.getElementById("credits").value;
            if (isNaN(credits)|| credits<=0 || credits>=16) {
                alert("Credits must be a number greater than 0 and less than 16");
                return false
            }
        }
    </script>
</head> <body>
    <form action="createCourse" onsubmit="return validate()">
        Title: <input type="text" id="title" size="20"><br />
        Credits: <input type="text" id="credits" size="2"><br />
        <Input type="submit" value="Submit">
    </form>
</body> </html>
```

USP of JSP

- ***JSP vs. Active Server Pages (ASP)***
  - ASP is a similar technology from Microsoft and is proprietary (uses VB).
  - JSP is platform independent and portable.
- ***JSP vs. Pure Servlets***
  - JSP is a servlet but it is more convenient to write and to modify regular HTML than to have a million println statements that generate the HTML.
  - The Web page design experts can build the HTML, leaving places for the servlet programmers to insert the dynamic content.
- ***JSP vs. JavaScript*** JavaScript can generate HTML dynamically on the client.
  - "Client Side": *Java Script code is executed by the browser* after the web server sends the HTTP response. With the exception of cookies, HTTP and form submission data is not available to JavaScript.
  - "Server Side": *Java Server Pages are executed by the web server* before the web server sends the HTTP response. It can access server-side resources like databases, catalogs.
- ***JSP vs. Static HTML*** Regular HTML, of course, cannot contain dynamic information.

Module Summary

- Understood the requirements of Database Application Programs and User Interfaces
- Familiarized with the Fundamentals notions and technologies of Web
- Learnt the notions of Servlets and Java Server Pages

PPD

Instructor and TAs

Name	Mail	Mobile
Partha Pratim Das, Instructor	ppd@cse.iitkgp.ernet.in	9830030880
Srijoni Majumdar, TA	majumdarsrijoni@gmail.com	9674474267
Himadri B G S Bhuyan, TA	himadribhuyan@gmail.com	9438911655
Gurunath Reddy M	mgurunathreddy@gmail.com	9434137638

Slides used in this presentation are borrowed from http://db-book.com/ with kind permission of the authors.

Edited and new slides are marked with "PPD".

Database Management Systems

Module 22: Application Design and Development/2

Partha Pratim Das

Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

ppd@cse.iitkgp.ernet.in

Srijoni Majumdar
Himadri B G S Bhuyan
Gurunath Reddy M

PPD

Module Recap

- Application Programs and User Interfaces
- Web Fundamentals
- Servlets and JSP

PPD

Module Objectives

- To understand Architectures Database Applications in detail
- To explore the Rapid Application Development Process
- To understand the issues in Application Performance
- To understand the issues in Application Security
- To appreciate how Mobile Apps are similar to and different web applications

PPD

Module Outline

- Application Architectures
- Rapid Application Development
- Application Performance
- Application Security
- Mobile Apps

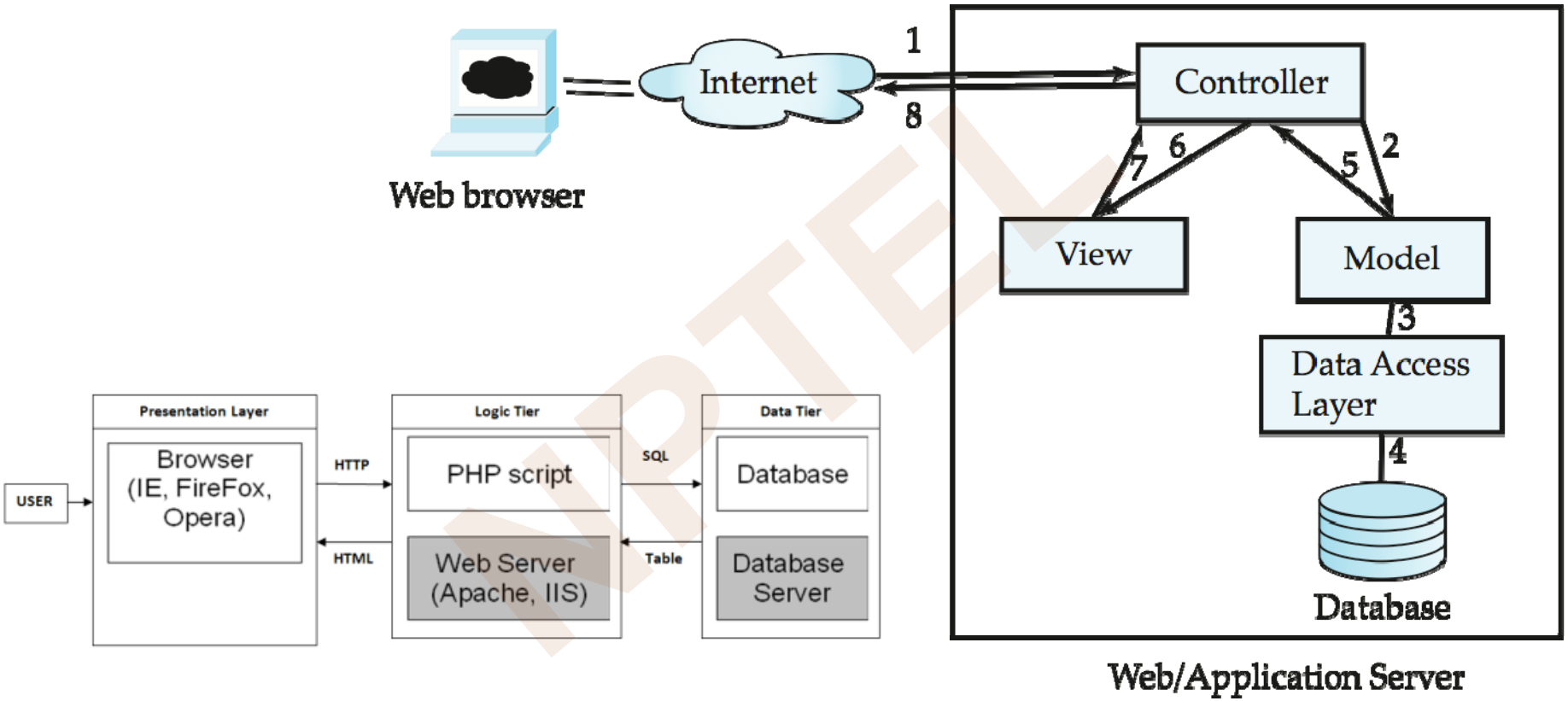
- **Application Architectures**
- Rapid Application Development
- Application Performance
- Application Security
- Mobile Apps

APPLICATION ARCHITECTURE

Application Architectures

- Application layers
  - Presentation or user interface
    - **model-view-controller (MVC)** architecture
      - **model**: business logic
      - **view**: presentation of data, depends on display device
      - **controller**: receives events, executes actions, and returns a view to the user
  - **business-logic** layer
    - provides high level view of data and actions on data
      - often using an object data model
    - hides details of data storage schema
  - **data access** layer
    - interfaces between business logic layer and the underlying database
    - provides mapping from object model of business layer to relational model of database

Application Architecture



Sample Applications in Multiple Tiers

Application	Presentation	Logic	Data	Functionality

Web Mail	• Login
	• Mail List View
	  • Inbox
	  • Sent Items
	  • Outbox
	  • Trash
	• Mail Composer
	• Filters
		• User Authentication
		• Connection to Mail Server (SMTP, POP, IMAP)
		• Encryption / Decryption
			• Mail Users
			• Address Book
			• Mail Items
				• Send / Receive Mails
				• Manage Address Book

Net Banking	• Login • Account View • Add / Delete Account • Add / Delete Beneficiary • Fund Transfer	• User Authentication • Beneficiary Authentication • Transaction Validation • Connection to Banks / Gateways • Encryption / Decryption	• Account Holders • Beneficiaries • Accounts • Debit / Credit Transactions	• Check Balance and Transactions • Transfer Funds

Timetable	• Login
	• Add / Delete Courses, Teachers, Rooms, Slots
	• Assignments:
	  • Teachers → Course
	• Allocations
	  • Course → Room, Slots
	• Views
		• User Authentication
		• Timetable Assignment Logic
		• Encryption / Decryption
			• Courses
			• Teachers
			• Rooms
			• Slots
			• Assignments
			• Allocations
				• Manage timetable for multiple courses taken by multiple teachers

Business Logic Layer

- Provides abstractions of entities
  - e.g. students, instructors, courses, etc
- Enforces **business rules** for carrying out actions
  - E.g. student can enroll in a class only if she has completed prerequsites, and has paid her tuition fees
- Supports **workflows** which define how a task involving multiple participants is to be carried out
  - E.g. how to process application by a student applying to a university
  - Sequence of steps to carry out task
  - Error handling
    - e.g. what to do if recommendation letters not received on time

Object-Relational Mapping

- Allows application code to be written on top of object-oriented data model, while storing data in a traditional relational database
  - alternative: implement object-oriented or object-relational database to store object model
    - has not been commercially successful
- Schema designer has to provide a mapping between object data and relational schema
  - e.g. Java class *Student* mapped to relation *student,* with corresponding mapping of attributes
  - An object can map to multiple tuples in multiple relations
- Application opens a session, which connects to the database
- Objects can be created and saved to the database using session.save(object)
  - mapping used to create appropriate tuples in the database
- Query can be run to retrieve objects satisfying specified predicates

Web Services

- Allow data on Web to be accessed using remote procedure call mechanism
- Two approaches are widely used
  - **Representation State Transfer (REST)**: allows use of standard HTTP request to a URL to execute a request and return data
    - returned data is encoded either in XML, or in **JavaScript Object Notation (JSON)**
  - **Big Web Services:**
    - uses XML representation for sending request data, as well as for returning results
    - standard protocol layer built on top of HTTP

- Application Architectures
- **Rapid Application Development**
- Application Performance
- Application Security
- Mobile Apps

RAPID APPLICATION DEVELOPMENT

Rapid Application Development

- A lot of effort is required to develop Web application interfaces
  - more so, to support rich interaction functionality associated with Web 2.0 applications
- Several approaches to speed up application development
  - Function library to generate user-interface elements
  - Drag-and-drop features in an IDE to create user-interface elements
  - Automatically generate code for user interface from a declarative specification
- Above features have been in used as part of **rapid application development (RAD)** tools even before advent of Web
- Web application development frameworks
  - Java Server Faces (JSF) includes JSP tag library
  - Ruby on Rails
    - Allows easy creation of simple **CRUD** (create, read, update and delete) interfaces by code generation from database schema or object model

ASP.NET and Visual Studio

- ASP.NET provides a variety of controls that are interpreted at server, and generate HTML code
- Visual Studio provides drag-and-drop development using these controls
  - E.g. menus and list boxes can be associated with DataSet object
  - Validator controls (constraints) can be added to form input fields
    - JavaScript to enforce constraints at client, and separately enforced at server
  - User actions such as selecting a value from a menu can be associated with actions at server
  - DataGrid provides convenient way of displaying SQL query results in tabular format

- Application Architectures
- Rapid Application Development
- **Application Performance**
- Application Security
- Mobile Apps

APPLICATION PERFORMANCE

Improving Web Server Performance

- Performance is an issue for popular Web sites
  - May be accessed by millions of users every day, thousands of requests per second at peak time
- Caching techniques used to reduce cost of serving pages by exploiting commonalities between requests
  - At the server site:
    - Caching of JDBC connections between servlet requests
      - a.k.a. **connection pooling**
    - Caching results of database queries
      - Cached results must be updated if underlying database changes
    - Caching of generated HTML
  - At the client's network
    - Caching of pages by Web proxy

- Application Architectures
- Rapid Application Development
- Application Performance
- **Application Security**
- Mobile Apps

APPLICATION SECURITY

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**
- Is the following prepared statemen secure?
  - conn.prepareStatement("select * from instructor where name = '" + name + "'")

Password Leakage

- Never store passwords, such as database passwords, in clear text in scripts that may be accessible to users
  - E.g. in files in a directory accessible to a web server
    - Normally, web server will execute, but not provide source of script files such as file.jsp or file.php, but source of editor backup files such as file.jsp~, or .file.jsp.swp may be served
- Restrict access to database server from IPs of machines running application servers
  - Most databases allow restriction of access by source IP address

Application Authentication

- Single factor authentication such as passwords too risky for critical applications
  - guessing of passwords, sniffing of packets if passwords are not encrypted
  - passwords reused by user across sites
  - spyware which captures password
- Two-factor authentication
  - e.g. password plus one-time password sent by SMS
  - e.g. password plus one-time password devices
    - device generates a new pseudo-random number every minute, and displays to user
    - user enters the current number as password
    - application server generates same sequence of pseudo-random numbers to check that the number is correct.

Application-Level Authorization

- Current SQL standard does not allow fine-grained authorization such as “students can see their own grades, but not other’s grades”
  - Problem 1: Database has no idea who are application users
  - Problem 2: SQL authorization is at the level of tables, or columns of tables, but not to specific rows of a table
- One workaround: use views such as

  create view *studentTakes* **as**
  select *
  from *takes*
  where *takes.ID = syscontext.user_id*()
  - where syscontext.user_id() provides end user identity
    - end user identity must be provided to the database by the application
  - Having multiple such views is cumbersome

Application-Level Authorization (Cont.)

- Currently, authorization is done entirely in application
- Entire application code has access to entire database
  - large surface area, making protection harder
- Alternative: **fine-grained (row-level) authorization** schemes
  - extensions to SQL authorization proposed but not currently implemented
  - Oracle Virtual Private Database (VPD) allows predicates to be added transparently to all SQL queries, to enforce fine-grained authorization
    - e.g. add *ID= sys_context.user_id()* to all queries on student relation if user is a student

Audit Trails

- Applications must log actions to an audit trail, to detect who carried out an update, or accessed some sensitive data
- Audit trails used after-the-fact to
  - detect security breaches
  - repair damage caused by security breach
  - trace who carried out the breach
- Audit trails needed at
  - Database level, and at
  - Application level

- Application Architectures
- Rapid Application Development
- Application Performance
- Application Security
- **Mobile Apps**

MOBILE APPS

What is a Mobile App?

- A type of application software designed to run on a mobile device, such as a smartphone or tablet computer
- Developed specifically for use on small, wireless computing devices, such as smartphones and tablets
- Designed with consideration for the demands and constraints of the devices and also to take advantage of any specialized capabilities
  - Form Factor – influences display and navigation
  - Limited Memory
  - Limited Computing Power
  - Limited Power
  - Limited Bandwidth
  - …
  + Availability of sensors like accelerometer
  + Availability of touchscreen – Gesture-based Navigation
  + …

Source: https://www.slideshare.net/hassandar18/architecture-of-mobile-software-applications?from_action=save

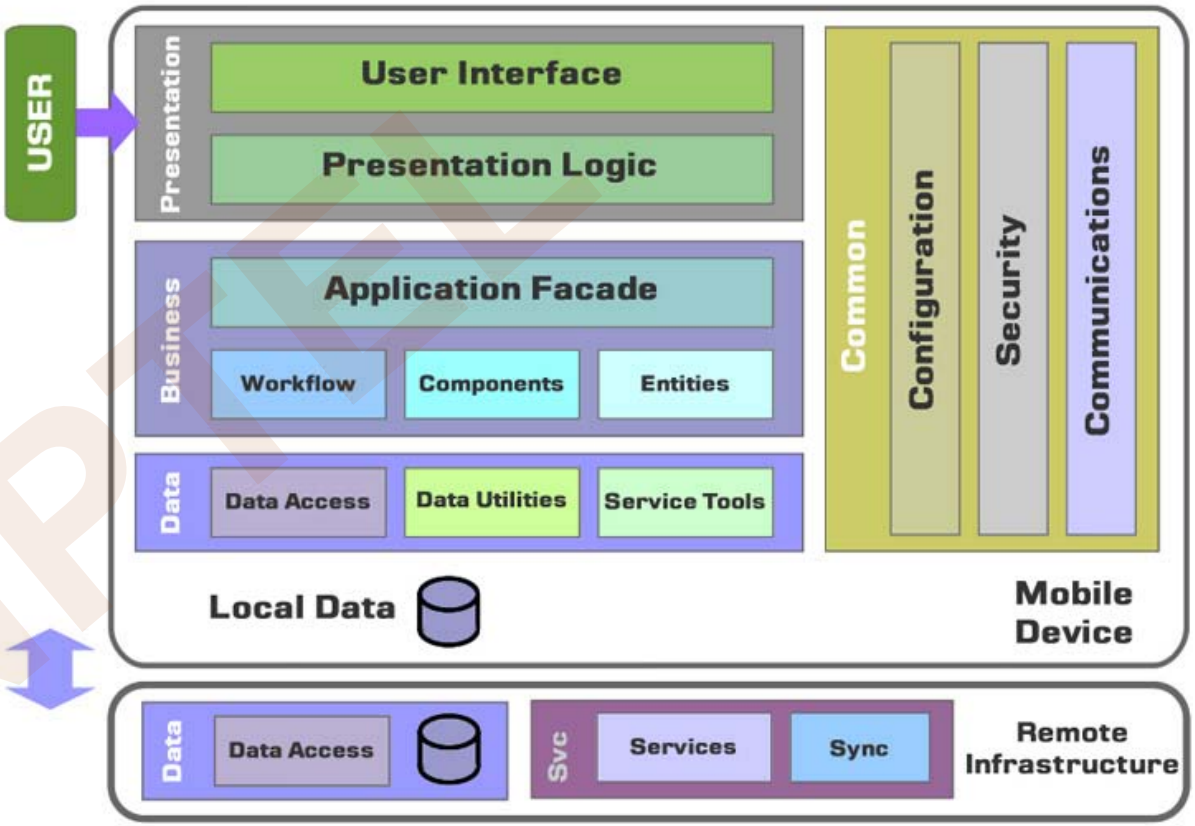
Mobile Website vis-à-vis Mobile App

- **Mobile Website**
  - Similar to any other website in that it consists of browser-based HTML pages
  - Can display text content, data, images and video
  - Typically accessed over WiFi or 3G or 4G networks
  - Designed for the smaller handheld display and touch-screen interface
  - Can also access mobile-specific features such as click-to-call (to dial a phone number) or location-based mapping
- **Mobile Apps**
  - Actual applications that are downloaded and installed on the mobile device
  - Users download apps from device-specific portals such as App Store, Google Play Store
  - The app may
    - pull content and data from the Internet, in similar fashion to a website, or
    - download the content so that it can be accessed without an Internet connection

Source: https://www.slideshare.net/hassandar18/architecture-of-mobile-software-applications?from_action=save

Architecture of Mobile App

- Typically 3 tier
  - Presentation
  - Business
  - Data
- Data Layer is often split between:
  - Local Data
  - Remote Data
- Needs customization for platform
  - Android
  - iOS
  - Windows



Source: https://www.slideshare.net/hassandar18/architecture-of-mobile-software-applications?from_action=save

Types of Mobile Apps

- **Native Apps**: Completely written in the native language of a platform
  - iOS → Objective-C; Android → Java or C/C++
  - Platform specific (heavily dependent on OS)
- **Web Apps**: Run completely inside of a Web browser.
  - Features interfaces built with HTML or CSS
  - Powered via Web programming languages →Ruby on Rails, JavaScript, PHP, or Python
  - Portable across any phone, tablet, or computer
- **Hybrid Apps**: Combines attributes of both native and Web apps.
  - Attempts to use redundant, common code that can be used across platforms, and
  - Tailors required attributes to the native system

Source: https://www.slideshare.net/hassandar18/architecture-of-mobile-software-applications?from_action=save

Design Issues

- Determine Device
- Note Device Resources – memory, power, speed
- Consider Bandwidth
- Decide on Architecture Layers
- Select Technology
- Define User Interface
- Select Navigation
- Maintain Flow

Source: https://www.slideshare.net/hassandar18/architecture-of-mobile-software-applications?from_action=save

Module Summary

- Studied the aspects of Database Applications Architectures
- Understood the steps in the Rapid Application Development Process
- Exposed to the issues in Application Performance
- Exposed to the issues in Application Security
- Learnt the distinctive features of Mobile Apps

PPD

Instructor and TAs

Name	Mail	Mobile
Partha Pratim Das, Instructor	ppd@cse.iitkgp.ernet.in	9830030880
Srijoni Majumdar, TA	majumdarsrijoni@gmail.com	9674474267
Himadri B G S Bhuyan, TA	himadribhuyan@gmail.com	9438911655
Gurunath Reddy M	mgurunathreddy@gmail.com	9434137638

Slides used in this presentation are borrowed from http://db-book.com/ with kind permission of the authors.

Edited and new slides are marked with "PPD".

Database Management Systems

Module 23: Application Design and Development/3

Partha Pratim Das

Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

ppd@cse.iitkgp.ernet.in

Srijoni Majumdar
Himadri B G S Bhuyan
Gurunath Reddy M

Database System Concepts, 6th Ed.

Module Recap

- Application Architectures
- Rapid Application Development
- Application Performance
- Application Security
- Mobile Apps

PPD

Module Objectives

- To design the schema for a Library Information System

Module Outline

- Library Information System

Library Information System (LIS)

An institute library has 200000+ books and 10000+ members. Books are regularly issued by members on loan and returned after a period. The library needs an LIS to manage the books, the members and the issue-return process. Every book has title, author (in case of multiple authors, only the first author is maintained), publisher, year of publication, ISBN number (which is unique for the publication), and accession number (which is the unique number of the copy of the book in the library). There may be multiple copies of the same book in the library. There are four categories of members of the library: undergraduate students, post graduate students, research scholars, and faculty members. Every student has name, roll number, department, gender, mobile number, date of birth, and degree (undergrad, grad, doctoral). Every faculty has name, employee id, department, gender, mobile number, and date of joining. Library also issues a unique membership number to every member. Every member has a maximum quota for the number of books she / he can issue for the maximum duration allowed to her / him. Currently these are set as:

- Each undergraduate student can issue up to 2 books for 1 month duration
- Each postgraduate student can issue up to 4 books for 1 month duration
- Each research scholar can issue up to 6 books for 3 months duration
- Each faculty member can issue up to 10 books for six months duration

The library has the following rules for issue:

A book may be issued to a member if it is not already issued to someone else (trivial). Also, a book may not be issued to a member if another copy of the same book is already issued to the same member. No issue will be done to a member if at the time of issue one or more of the books issued by the member has already exceeded its duration of issue. No issue will be allowed also if the quota is exceeded for the member.

Finally, it is assumed that the name of every author or member has two parts – first name and last name.

LIS Queries

LIS should support the following operations / query:

- Add / Remove members, categories of members, books.
- Add / Remove / Edit quota for a category of member, duration for a category of member.
- Check if the library has a book given its title (part of title should match). If yes: title, author, publisher, year and ISBN should be listed.
- Check if the library has a book given its author. If yes: title, author, publisher, year and ISBN should be listed.
- Check if a copy of a book (given its ISBN) is available with the library for issue. All accession numbers should be listed with issued or available information.
- Check the available (free) quota of a member.
- Issue a book to a member. This should check for the rules of the library.
- Return a book from a member.
- and so on

Entity Sets: books

- Every book has title, author (in case of multiple authors, only the first author is maintained), publisher, year of publication, ISBN number (which is unique for the publication), and accession number (which is the unique number of the copy of the book in the library). There may be multiple copies of the same book in the library.
- Entity Set:
  - **books**
- Attributes:
  - title
  - author_name (composite)
  - publisher
  - year
  - ISBN_no
  - accession_no

Entity Sets: students

- Every student has name, roll number, department, gender, mobile number, date of birth, and degree (undergrad, grad, doctoral).
- Entity Set:
  - **students**
- Attributes:
  - member_no – is unique
  - name (composite)
  - roll_no – is unique
  - department
  - gender
  - mobile_no – may be null
  - dob
  - degree

Entity Sets: faculty

- Every faculty has name, employee id, department, gender, mobile number, and date of joining.
- Entity Set:
  - **faculty**
- Attributes:
  - member_no – is unique
  - name (composite)
  - id – is unique
  - department
  - gender
  - mobile_no – may be null
  - doj

Entity Sets: members

- Library also issues a unique membership number to every member. There are four categories of members of the library: undergraduate students, post graduate students, research scholars, and faculty members.
- Entity Set:
  - **members**
- Attributes:
  - member_no
  - member_type (takes a value in ug, pg, rs or fc)

Entity Sets: quota

- Every member has a maximum quota for the number of books she / he can issue for the maximum duration allowed to her / him. Currently these are set as:
  - Each undergraduate student can issue up to 2 books for 1 month duration
  - Each postgraduate student can issue up to 4 books for 1 month duration
  - Each research scholar can issue up to 6 books for 3 months duration
  - Each faculty member can issue up to 10 books for six months duration
- Entity Set:
  - **quota**
- Attributes:
  - member_type
  - max_books
  - max_duration

Entity Sets: staff

- Though not explicitly stated, library would have staffs to manage the LIS.
- Entity Set:
  - **staff**
- Attributes: (speculated – to ratify from customer)
  - name (composite)
  - id – is unique
  - gender
  - mobile_no
  - doj

Relationships

- Books are regularly issued by members on loan and returned after a period. The library needs an LIS to manage the books, the members and the issue-return process.
- Relationship
  - **book_issue**
- Involved Entity Sets
  - **students / faculty / members**
    - member_no
  - **books**
    - accession_no
- Relationship Attribute
  - doi – date of issue
- Type of relationship
  - Many-to-one from **books**

Relational Schema

- **books**(title, author_fname, author_lname, publisher, year, ISBN_no, accession_no)
- **book_issue**(members, accession_no, doi)
- **members**(member_no, member_type)
- **quota**(member_type, max_books, max_duration)
- **students**(member_no, student_fname, student_lname, roll_no, department, gender, mobile_no, dob, degree)
- **faculty**(member_no, faculty_fname, faculty_lname, id, department, gender, mobile_no, doj)
- **staff**(staff_fname, staff_lname, id, gender, mobile_no, doj)

Schema Refinement

- **books**(title, author_fname, author_lname, publisher, year, ISBN_no, accession_no)
  - ISBN_no → title, author_fname, author_lname, publisher, year
  - accession_no → ISBN_no
  - Key: accession_no
- Redundancy of book information across copies
- Good to normalize:
  - **book_catalogue**(title, author_fname, author_lname, publisher, year, ISBN_no)
    - ISBN_no → title, author_fname, author_lname, publisher, year
    - Key: ISBN_no
  - **book_copies**(ISBN_no, accession_no)
    - accession_no → ISBN_no
    - Key: accession_no
- Both in BCNF. Decomposition is lossless join and dependency preserving

Schema Refinement

- **book_issue**(member_no, accession_no, doi)
  - member_no, accession_no → doi
  - Key: members, accession_no

Schema Refinement

- **quota**(member_type, max_books, max_duration)
  - member_type → max_books, max_duration
  - Key: member_type

Schema Refinement

- **members**(member_no, member_type)
  - member_no → member_type
  - Key: member_no
  - Value constraint on member_type
    - ug, pg, or rs: if the member is a student
    - fc: if the member is a faculty
  - How to determine the member_type?

Schema Refinement

- **students**(member_no, student_fname, student_lname, roll_no, department, gender, mobile_no, dob, degree)
  - roll_no → student_fname, student_lname, department, gender, mobile_no, dob, degree
  - member_no → roll_no
  - roll_no → member_no
  - 2 Keys: roll_no | member_no
- Issues:
  - member_no is needed for issue / return queries. It is unnecessary to have student's details with that.
  - member_no may also come from **faculty** relation.
  - member_type is needed for issue / return queries. This is implicit in degree – not explicitly given.

Schema Refinement

- **faculty**(member_no, faculty_fname, faculty_lname, id, department, gender, mobile_no, doj)
  - id → faculty_fname, faculty_lname, department, gender, mobile_no, doj
  - id → member_no
  - member_no → id
  - 2 Keys: id | member_no
- Issues:
  - member_no is needed for issue / return queries. It is unnecessary to have faculty details with that.
  - member_no may also come from **students** relation.
  - member_type is needed for issue / return queries. This is implicit by the fact that we are in faculty relation.

Schema Refinement

- Consider a query:
  - Get the name of the member who has issued the book having accession number = 162715
    - If the member is a student,
      - SELECT student_fname as First_Name, student_lname as Last_Name
      - FROM **students**, **book_issue**
      - WHERE accession_no = 162715 AND book_issue.member_no = students.member_no;
    - If the member is a faculty,
      - SELECT faculty_fname as First_Name, faculty_lname as Last_Name
      - FROM **faculty**, **book_issue**
      - WHERE accession_no = 162715 AND book_issue.member_no = faculty.member_no;
  - **Which query to fire!**

These are sample indicative queries not checked for syntax!

Schema Refinement

There are four categories of members of the library: undergraduate students, post graduate students, research scholars, and faculty members. This leads to the following specialization relationships.

- Consider the entity set **members** that represent the behavior of the member of a library and refine:
  - Attributes:
    - member_no
    - member_class – ‘student’ or ‘faculty’, used to choose table
    - member_type – ug,pg, rs, fc, …
    - roll_no (if member_class – ‘student’. Else null)
    - id (if member_class – ‘faculty’. Else null)
- We can then exploit some hidden relationship:
  - students IS_A members
  - faculty IS_A members
- Type of relationship
  - One-to-one

Schema Refinement

- Consider the old query again:
  - Get the name of the member who has issued the book having accession number = 162715

```
SELECT
    ((SELECT faculty_fname as First_Name, faculty_lname as Last_Name
    FROM faculty
    WHERE member_class = 'faculty' AND members.id = faculty.id)
    UNION
    (SELECT student_fname as First_Name, student_lname as Last_Name
    FROM students
    WHERE member_class = 'student' AND members.roll_no = students.roll_no))
FROM members, book_issue
WHERE accession_no = 162715 AND book_issue.member_no = members.member_no;
```


These are sample indicative queries not checked for syntax!

Schema Refinement

- **members**(member_no, member_class, member_type, roll_no, id)
  - member_no → member_type, member_class, roll_no, id
  - member_type → member_class
  - Key: member_no

Schema Refinement

- **students**(student_fname, student_lname, roll_no, department, gender, mobile_no, dob, degree)
  - roll_no → student_fname, student_lname, department, gender, mobile_no, dob, degree
  - Keys: roll_no
  - *Note*:
    - member_no is no longer used
    - member_type and member_class are set in **members** from degree at the time of creation of a new record.
- **faculty**(faculty_fname, faculty_lname, id, department, gender, mobile_no, doj)
  - id → faculty_fname, faculty_lname, department, gender, mobile_no, doj
  - Keys: id
  - *Note*:
    - member_no is no longer used
    - member_type and member_class are set in **members** at the time of creation of a new record

Schema Refinement – Final

- **book_catalogue**(title, author_fname, author_lname, publisher, year, ISBN_no)
- **book_copies**(ISBN_no, accession_no)
- **book_issue**(member_no, accession_no, doi)
- **quota**(member_type, max_books, max_duration)
- **members**(member_no, member_class, member_type, roll_no, id)
- **students**(student_fname, student_lname, roll_no, department, gender, mobile_no, dob, degree)
- **faculty**(faculty_fname, faculty_lname, id, department, gender, mobile_no, doj)
- **staff**(staff_fname, staff_lname, id, gender, mobile_no, doj)

Module Summary

- Using the specification for a Library Information System, we have illustrated how a schema can be designed and then refined for finalization
- Coding of various queries based on these schema are left as exercises

PPD

Instructor and TAs

Name	Mail	Mobile
Partha Pratim Das, Instructor	ppd@cse.iitkgp.ernet.in	9830030880
Srijoni Majumdar, TA	majumdarsrijoni@gmail.com	9674474267
Himadri B G S Bhuyan, TA	himadribhuyan@gmail.com	9438911655
Gurunath Reddy M	mgurunathreddy@gmail.com	9434137638

Slides used in this presentation are borrowed from http://db-book.com/ with kind permission of the authors.

Edited and new slides are marked with "PPD".

Database Management Systems

Module 24: Storage and File Structure/1: Storage

Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

ppd@cse.iitkgp.ernet.in

Srijoni Majumdar
Himadri B G S Bhuyan
Gurunath Reddy M

Module Recap

- Case Studies of Database Applications

PPD

Module Objectives

- To take a look at various Physical Storage Media for high volume, fast, reliable and inexpensive options for data storage for databases
- To understand the structure and basic functionality of Magnetic Disks
- To understand RAID – array of redundant disks in parallel to enhance speed and reliability
- To understand the options of Tertiary Storage for high volume, inexpensive backup options

PPD

Module Outline

- Overview of Physical Storage Media
- Magnetic Disks
- RAID
- Tertiary Storage

PPD

- **Overview of Physical Storage Media**
- Magnetic Disks
- RAID
- Tertiary Storage

OVERVIEW OF PHYSICAL STORAGE MEDIA

Classification of Physical Storage Media

- Speed with which data can be accessed
- Cost per unit of data
- Reliability
  - data loss on power failure or system crash
  - physical failure of the storage device
- Can differentiate storage into:
  - **volatile storage:** loses contents when power is switched off
  - **non-volatile storage**:
    - Contents persist even when power is switched off
    - Includes secondary and tertiary storage, as well as batter-backed up main-memory

Physical Storage Media

- **Cache**
  - fastest and most costly form of storage
  - volatile
  - managed by the computer system hardware
- **Main memory**
  - fast access (10s to 100s of nanoseconds; 1 nanosecond = 10^{-9} seconds)
  - generally too small (or too expensive) to store the entire database
    - capacities of up to a few Gigabytes widely used currently
    - Capacities have gone up and per-byte costs have decreased steadily and rapidly (roughly factor of 2 every 2 to 3 years)
  - **Volatile**
    - contents of main memory are usually lost if a power failure or system crash occurs

Physical Storage Media (Cont.)

- **Flash memory**
  - Data survives power failure
  - Data can be written at a location only once, but location can be erased and written to again
    - Can support only a limited number (10K – 1M) of write/erase cycles
    - Erasing of memory has to be done to an entire bank of memory
  - Reads are roughly as fast as main memory
  - But writes are slow (few microseconds), erase is slower
  - Widely used in embedded devices such as digital cameras, phones, and USB keys

Physical Storage Media (Cont.)

- **Magnetic-disk**
  - Data is stored on spinning disk, and read/written magnetically
  - Primary medium for the long-term storage of data
    - typically stores entire database
  - Data must be moved from disk to main memory for access, and written back for storage
    - Much slower access than main memory
  - **direct-access**
    - possible to read data on disk in any order, unlike magnetic tape
  - Capacities range up to roughly 16~32 TB
    - Much larger capacity and cost/byte than main memory/flash memory
    - Growing constantly and rapidly with technology improvements (factor of 2 to 3 every 2 years)
  - Survives power failures and system crashes
    - disk failure can destroy data, but is rare

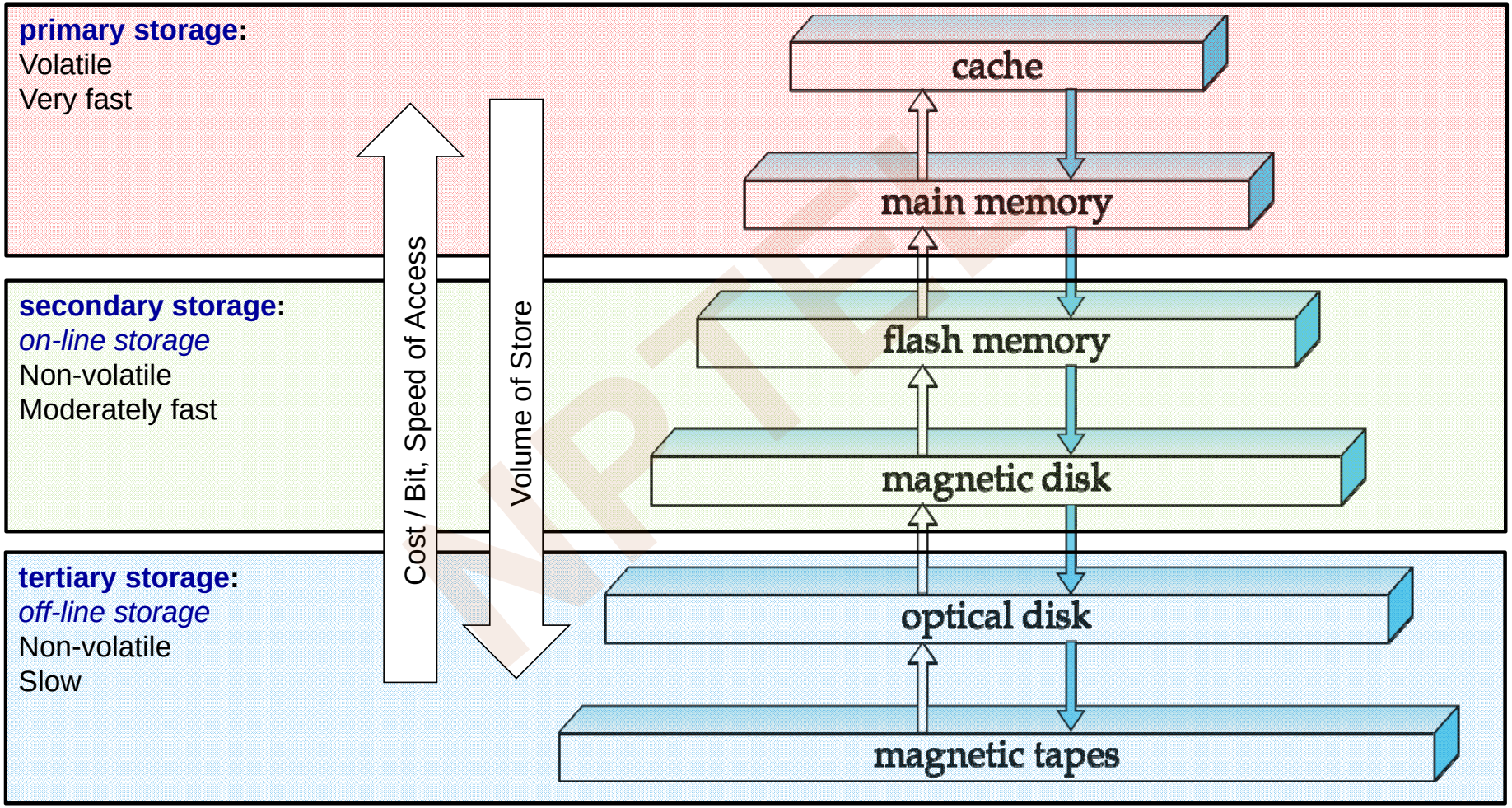
Physical Storage Media (Cont.)

- **Optical storage**
  - non-volatile, data is read optically from a spinning disk using a laser
  - CD-ROM (640 MB) and DVD (4.7 to 17 GB) most popular forms
  - Blu-ray disks: 27 GB to 54 GB
  - Write-one, read-many (WORM) optical disks used for archival storage (CD-R, DVD-R, DVD+R)
  - Multiple write versions also available (CD-RW, DVD-RW, DVD+RW, and DVD-RAM)
  - Reads and writes are slower than with magnetic disk
  - **Juke-box** systems, with large numbers of removable disks, a few drives, and a mechanism for automatic loading/unloading of disks available for storing large volumes of data

Physical Storage Media (Cont.)

- **Tape storage**
  - non-volatile, used primarily for backup (to recover from disk failure), and for archival data
  - **sequential-access**
    - much slower than disk
  - very high capacity (40 to 300 TB tapes available)
  - tape can be removed from drive ⇒ storage costs much cheaper than disk, but drives are expensive
  - Tape jukeboxes available for storing massive amounts of data
    - hundreds of terabytes (1 terabyte = 10^{12} bytes) to even multiple **petabytes** (1 petabyte = 10^{15} bytes)

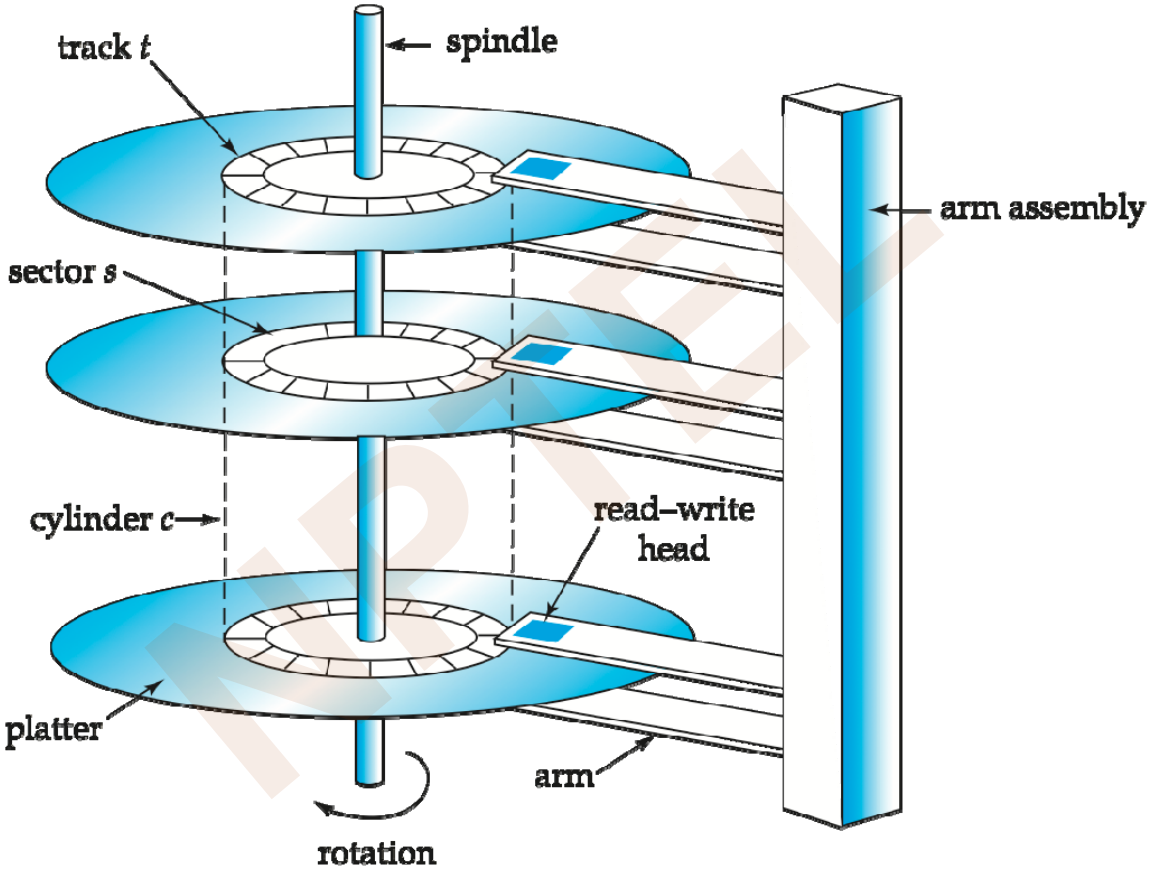
Storage Hierarchy



- Overview of Physical Storage Media
- **Magnetic Disks**
- RAID
- Tertiary Storage

MAGNETIC DISKS

Magnetic Hard Disk Mechanism

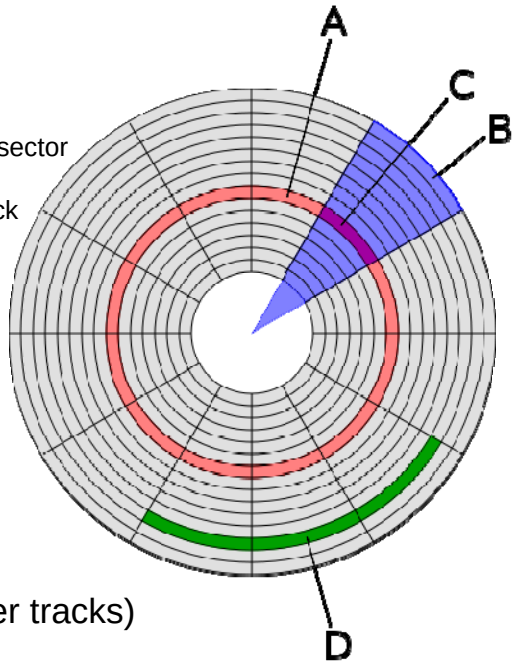


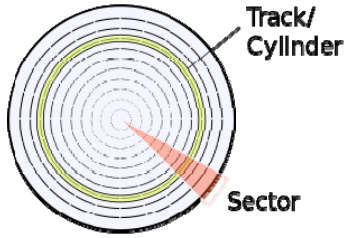
NOTE: Diagram is schematic, and simplifies the structure of actual disk drives

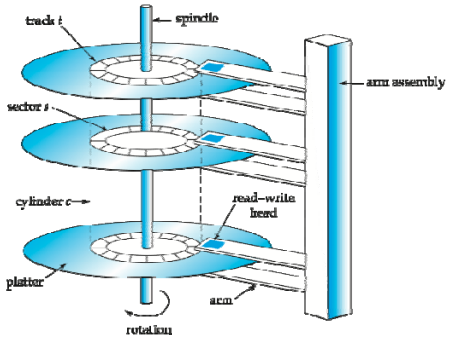
Magnetic Disks

- **Read-write head**
  - Positioned very close to the platter surface (almost touching it)
  - Reads or writes magnetically encoded information
- Surface of platter divided into circular **tracks**
  - Over 50K-100K tracks per platter on typical hard disks
- Each track is divided into **sectors**
  - A sector is the smallest unit of data that can be read or written.
  - Sector size typically 512 bytes
  - Typical sectors per track: 500 to 1000 (on inner tracks) to 1000 to 2000 (on outer tracks)
- To read/write a sector
  - disk arm swings to position head on right track
  - platter spins continually; data is read/written as sector passes under head
- Head-disk assemblies
  - multiple disk platters on a single spindle (1 to 5 usually)
  - one head per platter, mounted on a common arm.
- **Cylinder** *i* consists of *i*th track of all the platters

A: Track
B: Geometrical sector
C: Track sector
D: Cluster / Block



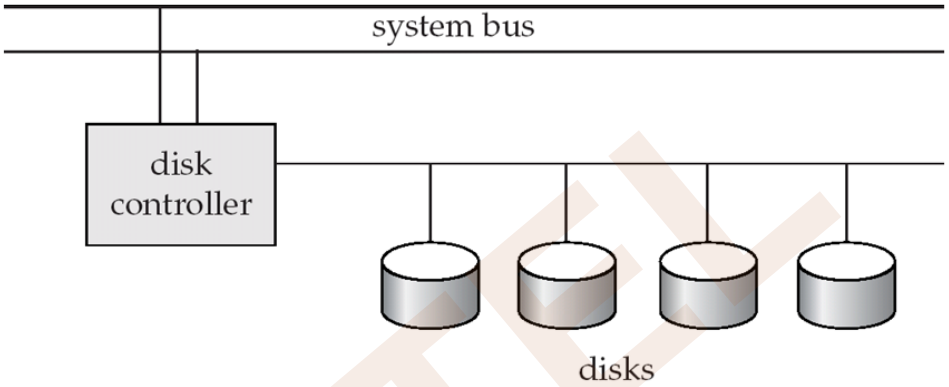




Magnetic Disks (Cont.)

- Earlier generation disks were susceptible to head-crashes
  - Surface of earlier generation disks had metal-oxide coatings which would disintegrate on head crash and damage all data on disk
  - Current generation disks are less susceptible to such disastrous failures, although individual sectors may get corrupted
- **Disk controller** – interfaces between the computer system and the disk drive hardware
  - accepts high-level commands to read or write a sector
  - initiates actions such as moving the disk arm to the right track and actually reading or writing the data
  - Computes and attaches **checksums** to each sector to verify that data is read back correctly
    - If data is corrupted, with very high probability stored checksum won't match recomputed checksum
  - Ensures successful writing by reading back sector after writing it
  - Performs remapping of bad sectors

Disk Subsystem



- Multiple disks connected to a computer system through a controller
  - Controllers functionality (checksum, bad sector remapping) often carried out by individual disks; reduces load on controller
- Disk interface standards families
  - ATA (AT adaptor) range of standards
  - SATA (Serial ATA)
  - SCSI (Small Computer System Interconnect) range of standards
  - SAS (Serial Attached SCSI)
  - Several variants of each standard (different speeds and capabilities)

Disk Subsystem

- Disks usually connected directly to computer system
- In **Storage Area Networks (SAN)**, a large number of disks are connected by a high-speed network to a number of servers
- In **Network Attached Storage (NAS)** networked storage provides a file system interface using networked file system protocol, instead of providing a disk system interface

Performance Measures of Disks

- **Access time** – the time it takes from when a read or write request is issued to when data transfer begins. It consists of:
  - **Seek time** – time it takes to reposition the arm over the correct track
    - Average seek time is 1/2 the worst case seek time
      - Would be 1/3 if all tracks had the same number of sectors, and we ignore the time to start and stop arm movement
    - 4 to 10 milliseconds on typical disks
  - **Rotational latency** – time it takes for the sector to be accessed to appear under the head
    - Average latency is 1/2 of the worst case latency.
    - 4 to 11 milliseconds on typical disks (5400 to 15000 rpm)
- **Data-transfer rate** – the rate at which data can be retrieved from or stored to the disk.
  - 25 to 100 MB per second max rate, lower for inner tracks
  - Multiple disks may share a controller, so rate that controller can handle is also important
    - E.g. SATA: 150 MB/sec, SATA-II 3Gb (300 MB/sec)
    - Ultra 320 SCSI: 320 MB/s, SAS (3 to 6 Gb/sec)
    - Fiber Channel (FC2Gb or 4Gb): 256 to 512 MB/s

Performance Measures (Cont.)

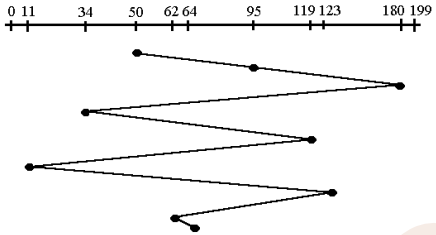
- **Mean time to failure (MTTF)** – the average time the disk is expected to run continuously without any failure
  - Typically 3 to 5 years
  - Probability of failure of new disks is quite low, corresponding to a "theoretical MTTF" of 500,000 to 1,200,000 hours for a new disk
    - E.g., an MTTF of 1,200,000 hours for a new disk means that given 1000 relatively new disks, on an average one will fail every 1200 hours
  - MTTF decreases as disk ages

Optimization of Disk-Block Access

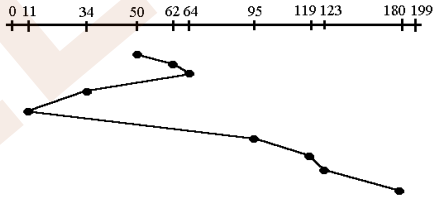
- **Block** – a contiguous sequence of sectors from a single track
  - data is transferred between disk and main memory in blocks
  - sizes range from 512 bytes to several kilobytes
    - Smaller blocks: more transfers from disk
    - Larger blocks: more space wasted due to partially filled blocks
    - Typical block sizes today range from 4 to 16 kilobytes

Optimization of Disk-Block Access (Cont.)

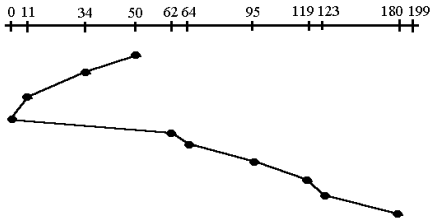
- **Disk-arm-scheduling** algorithms order pending accesses to tracks so that disk arm movement is minimized: Example: *Queue 95, 180, 34, 119, 11, 123, 62, 64 with the Read-write head initially at the track 50 and the tail track being at 199*



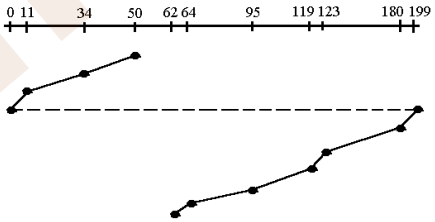
First Come -First Serve (FCFS)
640 Tracks, Oscillations



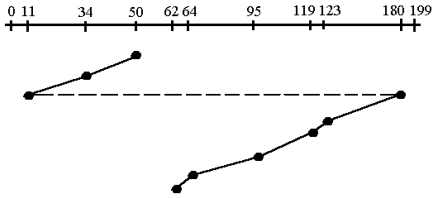
Shortest Seek Time First (SSTF)
236 Tracks, Starvation



Elevator (SCAN)
230 Tracks



Circular Scan (C-SCAN)
187 Tracks



C-LOOK
157 Tracks, Optimal

Optimization of Disk Block Access (Cont.)

- **File organization** – optimize block access time by organizing the blocks to correspond to how data will be accessed
  - E.g. Store related information on the same or nearby cylinders
  - Files may get **fragmented** over time
    - E.g. if data is inserted to/deleted from the file
    - Or free blocks on disk are scattered, and newly created file has its blocks scattered over the disk
    - Sequential access to a fragmented file results in increased disk arm movement
  - Some systems have utilities to defragment the file system, in order to speed up file access

Optimization of Disk Block Access (Cont.)

- **Nonvolatile write buffers** speed up disk writes by writing blocks to a non-volatile RAM buffer immediately
  - Non-volatile RAM: battery backed up RAM or flash memory
    - Even if power fails, the data is safe and will be written to disk when power returns
  - Controller then writes to disk whenever the disk has no other requests or request has been pending for some time
  - Database operations that require data to be safely stored before continuing can continue without waiting for data to be written to disk
  - *Writes can be reordered to minimize disk arm movement*
- **Log disk** – a disk devoted to writing a sequential log of block updates
  - Used exactly like nonvolatile RAM
    - Write to log disk is very fast since no seeks are required
    - No need for special hardware (NV-RAM)
- File systems typically reorder writes to disk to improve performance
  - **Journaling file systems** write data in safe order to NV-RAM or log disk
  - Reordering without journaling: risk of corruption of file system data

Flash Storage

- NOR flash vs NAND flash
- NAND flash
  - used widely for storage, since it is much cheaper than NOR flash
  - requires page-at-a-time read (page: 512 bytes to 4 KB)
  - transfer rate around 20 MB/sec
  - **solid state disks**: use multiple flash storage devices to provide higher transfer rate of 100 to 200 MB/sec
  - erase is very slow (1 to 2 millisecs)
    - erase block contains multiple pages
    - **remapp**ing of logical page addresses to physical page addresses avoids waiting for erase
      - **translation table** tracks mapping
        - also stored in a label field of flash page
      - remapping carried out by **flash translation layer**
    - after 100,000 to 1,000,000 erases, erase block becomes unreliable and cannot be used
      - **wear leveling**

PPD

- Overview of Physical Storage Media
- Magnetic Disks
- **RAID**
- Tertiary Storage

RAID: REDUNDANT ARRAY OF INDEPENDENT DISKS

RAID

- **RAID: Redundant Arrays of Independent Disks**
  - disk organization techniques that manage a large numbers of disks, providing a view of a single disk of
    - high capacity and high speed by using multiple disks in parallel,
    - high reliability by storing data redundantly, so that data can be recovered even if a disk fails
- The chance that some disk out of a set of *N* disks will fail is much higher than the chance that a specific single disk will fail
  - E.g., a system with 100 disks, each with MTTF of 100,000 hours (approx. 11 years), will have a system MTTF of 1000 hours (approx. 41 days)
  - Techniques for using redundancy to avoid data loss are critical with large numbers of disks
- Originally a cost-effective alternative to large, expensive disks
  - I in RAID originally stood for "inexpensive''
  - Today RAIDs are used for their higher reliability and bandwidth
    - The "I" is interpreted as independent

Improvement of Reliability via Redundancy

- **Redundancy** – store extra information that can be used to rebuild information lost in a disk failure
- **Mean time to data loss** depends on mean time to failure, and **mean time to repair**
  - E.g. MTTF of 100,000 hours, mean time to repair of 10 hours gives mean time to data loss of $500*10^6$ hours (or 57,000 years) for a mirrored pair of disks (ignoring dependent failure modes)
- **Mirroring** (or **shadowing**)
  - Duplicate every disk. Logical disk consists of two physical disks.
  - Every write is carried out on both disks
    - Reads can take place from either disk
  - If one disk in a pair fails, data still available in the other
    - Data loss would occur only if a disk fails, and its mirror disk also fails before the system is repaired
      - Probability of combined event is very small
        - » Except for dependent failure modes such as fire or building collapse or electrical power surges

Improvement of Reliability via Redundancy

- **Bit-level striping** – split the bits of each byte across multiple disks
  - In an array of eight disks, write bit *i* of each byte to disk *i*
  - Each access can read data at eight times the rate of a single disk
  - But seek/access time worse than for a single disk
    - Bit level striping is not used much any more
- **Block-level striping** – with *n* disks, block *i* of a file goes to disk (*i* mod *n*) + 1
  - Requests for different blocks can run in parallel if the blocks reside on different disks
  - A request for a long sequence of blocks can utilize all disks in parallel

Improvement of Reliability via Redundancy

- **Bit-Interleaved Parity** – a single parity bit is enough for error correction, not just detection, since we know which disk has failed
  - When writing data, corresponding parity bits must also be computed and written to a parity bit disk
  - To recover data in a damaged disk, compute XOR of bits from other disks (including parity bit disk)
- **Block-Interleaved Parity**: Uses block-level striping, and keeps a parity block on a separate disk for corresponding blocks from *N* other disks
  - When writing data block, corresponding block of parity bits must also be computed and written to parity disk
  - To find value of a damaged block, compute XOR of bits from corresponding blocks (including parity block) from other disks.

Choice of RAID Level

- Factors in choosing RAID level
  - Monetary cost
  - Performance: Number of I/O operations per second, and bandwidth during normal operation
  - Performance during failure
  - Performance during rebuild of failed disk
    - Including time taken to rebuild failed disk
- RAID 0 is used only when data safety is not important
  - E.g. data can be recovered quickly from other sources
- Level 2 and 4 never used since they are subsumed by 3 and 5
- Level 3 is not used anymore since bit-striping forces single block reads to access all disks, wasting disk arm movement, which block striping (level 5) avoids
- Level 6 is rarely used since levels 1 and 5 offer adequate safety for most applications

Choice of RAID Level (Cont.)

- Level 1 provides much better write performance than level 5
  - Level 5 requires at least 2 block reads and 2 block writes to write a single block, whereas Level 1 only requires 2 block writes
  - Level 1 preferred for high update environments such as log disks
- Level 1 had higher storage cost than level 5
  - disk drive capacities increasing rapidly (50%/year) whereas disk access times have decreased much less (x 3 in 10 years)
  - I/O requirements have increased greatly, e.g. for Web servers
  - When enough disks have been bought to satisfy required rate of I/O, they often have spare storage capacity
    - so there is often no extra monetary cost for Level 1!
- Level 5 is preferred for applications with low update rate, and large amounts of data
- Level 1 is preferred for all other applications

- Overview of Physical Storage Media
- Magnetic Disks
- **RAID**
- Tertiary Storage

TERTIARY STORAGE

Optical Disks

- Compact disk-read only memory (CD-ROM)
  - Removable disks, 640 MB per disk
  - Seek time about 100 msec (optical read head is heavier and slower)
  - Higher latency (3000 RPM) and lower data-transfer rates (3-6 MB/s) compared to magnetic disks
- Digital Video Disk (DVD)
  - DVD-5 holds 4.7 GB , and DVD-9 holds 8.5 GB
  - DVD-10 and DVD-18 are double sided formats with capacities of 9.4 GB and 17 GB
  - Blu-ray DVD: 27 GB (54 GB for double sided disk)
  - Slow seek time, for same reasons as CD-ROM
- Record once versions (CD-R and DVD-R) are popular
  - data can only be written once, and cannot be erased.
  - high capacity and long lifetime; used for archival storage
  - Multi-write versions (CD-RW, DVD-RW, DVD+RW and DVD-RAM) also available

Magnetic Tapes

- Hold large volumes of data and provide high transfer rates
  - Few GB for DAT (Digital Audio Tape) format, 10-40 GB with DLT (Digital Linear Tape) format, 100 GB+ with Ultrium format, and 330 GB with Ampex helical scan format
  - Transfer rates from few to 10s of MB/s
- Tapes are cheap, but cost of drives is very high
- Very slow access time in comparison to magnetic and optical disks
  - limited to sequential access.
  - Some formats (Accelis) provide faster seek (10s of seconds) at cost of lower capacity
- Used mainly for backup, for storage of infrequently used information, and as an off-line medium for transferring information from one system to another.
- Tape jukeboxes used for very large capacity storage
  - Multiple petabyes (10^{15} bytes)

Module Summary

- Looked at the options of Physical Storage Media for high volume, fast, reliable and inexpensive options for data storage for databases
- Understood the structure and basic functionality of Magnetic Disks
- Understood RAID – array of redundant disks in parallel to enhance speed and reliability
- Understood the options of Tertiary Storage for high volume, inexpensive backup options

PPD

Instructor and TAs

Name	Mail	Mobile
Partha Pratim Das, Instructor	ppd@cse.iitkgp.ernet.in	9830030880
Srijoni Majumdar, TA	majumdarsrijoni@gmail.com	9674474267
Himadri B G S Bhuyan, TA	himadribhuyan@gmail.com	9438911655
Gurunath Reddy M	mgurunathreddy@gmail.com	9434137638

Slides used in this presentation are borrowed from http://db-book.com/ with kind permission of the authors.

Edited and new slides are marked with "PPD".

Database Management Systems

Module 25: Storage and File Structure/2: File Structure

Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

ppd@cse.iitkgp.ernet.in

Srijoni Majumdar
Himadri B G S Bhuyan
Gurunath Reddy M

Module Recap

- Overview of Physical Storage Media
- Magnetic Disks
- RAID
- Tertiary Storage

PPD

Module Objectives

- To familiarize with the organization for database files
- To understand how records and relations are organized in files
- To learn how databases keep their own information in Data-Dictionary Storage – the metadata database of a database
- To understand the mechanisms for fast access of a database store

PPD

Module Outline

- File Organization
- Organization of Records in Files
- Data-Dictionary Storage
- Storage Access

- **File Organization**
- Organization of Records in Files
- Data-Dictionary Storage
- Storage Access

FILE ORGANIZATION

File Organization

- A database is
  - A collection of *files*. A file is
    - A sequence of *records.* A record is
      - A sequence of *fields*
- One approach:
  - assume record size is fixed
  - each file has records of one particular type only
  - different files are used for different relations

  This case is easiest to implement; will consider variable length records later

Fixed-Length Records

- Simple approach:
  - Store record *i* starting from byte $n * (i - 1)$, where *n* is the size of each record.
  - Record access is simple but records may cross blocks
    - Modification: do not allow records to cross block boundaries

- Deletion of record *i:* alternatives*:*
  - move records $i + 1, \ldots, n$ to $i, \ldots, n - 1$
  - move record *n* to *i*
  - do not move records, but link all free records on a *free list*

record 0	10101	Srinivasan	Comp. Sci.	65000
record 1	12121	Wu	Finance	90000
record 2	15151	Mozart	Music	40000
record 3	22222	Einstein	Physics	95000
record 4	32343	El Said	History	60000
record 5	33456	Gold	Physics	87000
record 6	45565	Katz	Comp. Sci.	75000
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000

Deleting record 3 and compacting

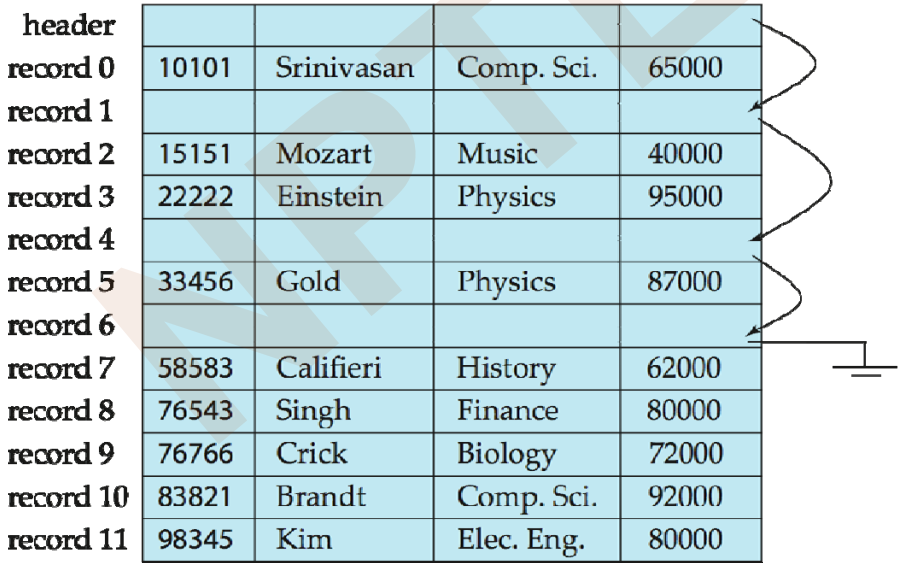
record 0	10101	Srinivasan	Comp. Sci.	65000
record 1	12121	Wu	Finance	90000
record 2	15151	Mozart	Music	40000
record 4	32343	El Said	History	60000
record 5	33456	Gold	Physics	87000
record 6	45565	Katz	Comp. Sci.	75000
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000

Deleting record 3 and moving last record

record 0	10101	Srinivasan	Comp. Sci.	65000
record 1	12121	Wu	Finance	90000
record 2	15151	Mozart	Music	40000
record 11	98345	Kim	Elec. Eng.	80000
record 4	32343	El Said	History	60000
record 5	33456	Gold	Physics	87000
record 6	45565	Katz	Comp. Sci.	75000
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000

Free Lists

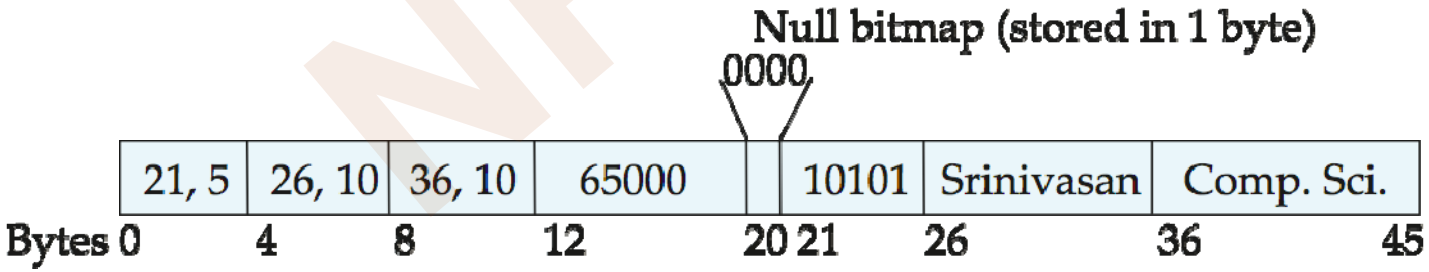
- Store the address of the first deleted record in the file header
- Use this first record to store the address of the second deleted record, and so on
- Can think of these stored addresses as pointers since they "point" to the location of a record
- More space efficient representation: reuse space for normal attributes of free records to store pointers (No pointers stored in in-use records)



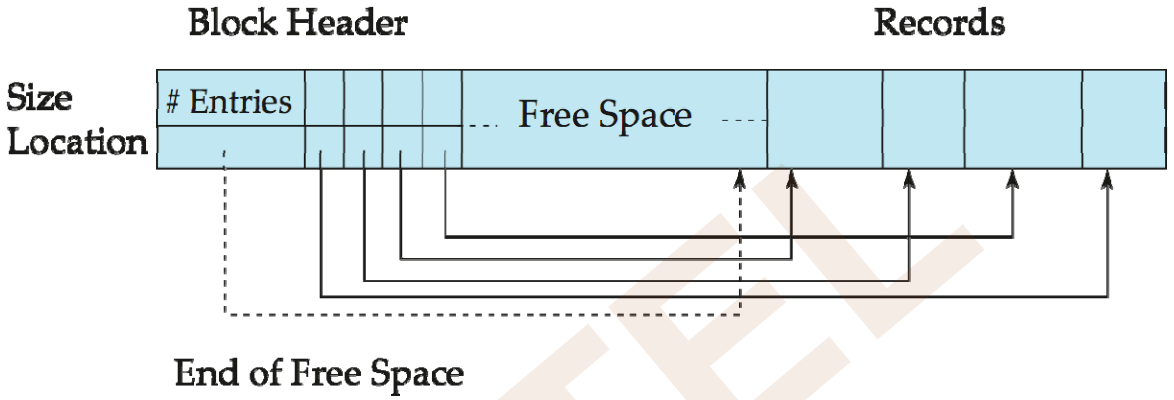
header				
record 0	10101	Srinivasan	Comp. Sci.	65000
record 1				
record 2	15151	Mozart	Music	40000
record 3	22222	Einstein	Physics	95000
record 4				
record 5	33456	Gold	Physics	87000
record 6				
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000

Variable-Length Records

- Variable-length records arise in database systems in several ways:
  - Storage of multiple record types in a file
  - Record types that allow variable lengths for one or more fields such as strings (**varchar**)
  - Record types that allow repeating fields (used in some older data models)
- Attributes are stored in order
- Variable length attributes represented by fixed size (offset, length), with actual data stored after all fixed length attributes
- Null values represented by null-value bitmap



Variable-Length Records: Slotted Page Structure



- **Slotted page** header contains:
  - number of record entries
  - end of free space in the block
  - location and size of each record
- Records can be moved around within a page to keep them contiguous with no empty space between them; entry in the header must be updated
- Pointers should not point directly to record — instead they should point to the entry for the record in header

- File Organization
- **Organization of Records in Files**
- Data-Dictionary Storage
- Storage Access

ORGANIZATION OF RECORDS IN FILES

Organization of Records in Files

- **Heap** – a record can be placed anywhere in the file where there is space
- **Sequential** – store records in sequential order, based on the value of the search key of each record
- **Hashing** – a hash function computed on some attribute of each record; the result specifies in which block of the file the record should be placed
- Records of each relation may be stored in a separate file. In a **multitable clustering file organization** records of several different relations can be stored in the same file
  - Motivation: store related records on the same block to minimize I/O

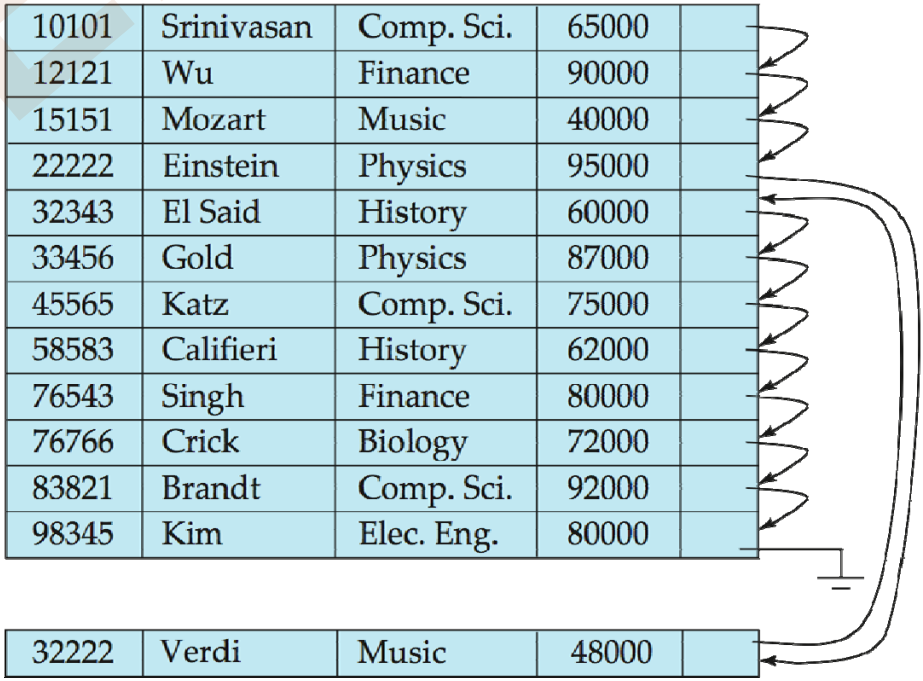
Sequential File Organization

- Suitable for applications that require sequential processing of the entire file
- The records in the file are ordered by a search-key

10101	Srinivasan	Comp. Sci.	65000	
12121	Wu	Finance	90000	
15151	Mozart	Music	40000	
22222	Einstein	Physics	95000	
32343	El Said	History	60000	
33456	Gold	Physics	87000	
45565	Katz	Comp. Sci.	75000	
58583	Califieri	History	62000	
76543	Singh	Finance	80000	
76766	Crick	Biology	72000	
83821	Brandt	Comp. Sci.	92000	
98345	Kim	Elec. Eng.	80000	

Sequential File Organization (Cont.)

- Deletion – use pointer chains
- Insertion –locate the position where the record is to be inserted
  - if there is free space insert there
  - if no free space, insert the record in an overflow block
  - In either case, pointer chain must be updated
- Need to reorganize the file from time to time to restore sequential order



10101	Srinivasan	Comp. Sci.	65000	
12121	Wu	Finance	90000	
15151	Mozart	Music	40000	
22222	Einstein	Physics	95000	
32343	El Said	History	60000	
33456	Gold	Physics	87000	
45565	Katz	Comp. Sci.	75000	
58583	Califieri	History	62000	
76543	Singh	Finance	80000	
76766	Crick	Biology	72000	
83821	Brandt	Comp. Sci.	92000	
98345	Kim	Elec. Eng.	80000	

32222	Verdi	Music	48000	

Multitable Clustering File Organization

Store several relations in one file using a **multitable clustering** file organization

department

dept_name	*building*	*budget*
Comp. Sci.	Taylor	100000
Physics	Watson	70000

instructor

ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
83821	Brandt	Comp. Sci.	92000

multitable clustering of *department* and *instructor*

Comp. Sci.	Taylor	100000
45564	Katz	75000
10101	Srinivasan	65000
83821	Brandt	92000
Physics	Watson	70000
33456	Gold	87000

Multitable Clustering File Organization (cont.)

- good for queries involving *department* ⋈ *instructor*, and for queries involving one single department and its instructors
- bad for queries involving only *department*
- results in variable size records
- Can add pointer chains to link records of a particular relation

Comp. Sci.	Taylor	100000
45564	Katz	75000
10101	Srinivasan	65000
83821	Brandt	92000
Physics	Watson	70000
33456	Gold	87000

- File Organization
- Organization of Records in Files
- **Data-Dictionary Storage**
- Storage Access

DATA DICTIONARY STORAGE

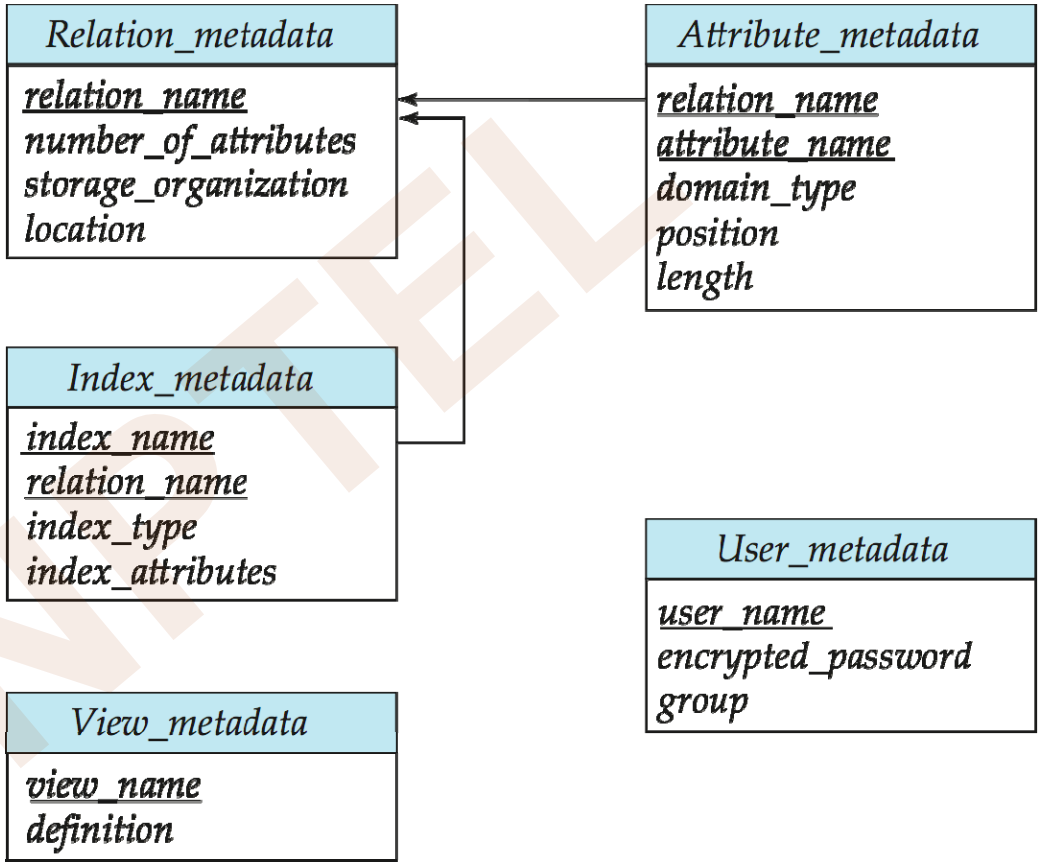
Data Dictionary Storage

The **Data dictionary** (also called **system catalog**) stores **metadata**; that is, data about data, such as:

- Information about relations
  - names of relations
  - names, types and lengths of attributes of each relation
  - names and definitions of views
  - integrity constraints
- User and accounting information, including passwords
- Statistical and descriptive data
  - number of tuples in each relation
- Physical file organization information
  - How relation is stored (sequential/hash/…)
  - Physical location of relation
- Information about indices

Relational Representation of System Metadata

- Relational representation on disk
- Specialized data structures designed for efficient access, in memory



- File Organization
- Organization of Records in Files
- Data-Dictionary Storage
- **Storage Access**

STORAGE ACCESS

Storage Access

- A database file is partitioned into fixed-length storage units called **blocks**
  - Blocks are units of both storage allocation and data transfer
- Database system seeks to minimize the number of block transfers between the disk and memory
  - We can reduce the number of disk accesses by keeping as many blocks as possible in main memory
- **Buffer** – portion of main memory available to store copies of disk blocks
- **Buffer manager** – subsystem responsible for allocating buffer space in main memory

Buffer Manager

- Programs call on the buffer manager when they need a block from disk.
  1. If the block is already in the buffer, buffer manager returns the address of the block in main memory
  2. If the block is not in the buffer, the buffer manager
     1. Allocates space in the buffer for the block
        1. Replacing (throwing out) some other block, if required, to make space for the new block
        2. Replaced block written back to disk only if it was modified since the most recent time that it was written to/fetched from the disk
     2. Reads the block from the disk to the buffer, and returns the address of the block in main memory to requester

Buffer-Replacement Policies

- Most operating systems replace the block **least recently used** (LRU strategy)
- Idea behind LRU – use past pattern of block references as a predictor of future references
- Queries have well-defined access patterns (such as sequential scans), and a database system can use the information in a user's query to predict future references
  - LRU can be a bad strategy for certain access patterns involving repeated scans of data
    - For example: when computing the join of 2 relations r and s by a nested loops
      for each tuple *tr* of *r* do
        for each tuple *ts* of *s* do
          if the tuples *tr* and *ts* match …
  - Mixed strategy with hints on replacement strategy provided by the query optimizer is preferable

Buffer-Replacement Policies (Cont.)

- **Pinned block** – memory block that is not allowed to be written back to disk
- **Toss-immediate** strategy – frees the space occupied by a block as soon as the final tuple of that block has been processed
- **Most recently used (MRU) strategy** – system must pin the block currently being processed. After the final tuple of that block has been processed, the block is unpinned, and it becomes the most recently used block.
- Buffer manager can use statistical information regarding the probability that a request will reference a particular relation
  - E.g., the data dictionary is frequently accessed. Heuristic: keep data-dictionary blocks in main memory buffer
- Buffer managers also support **forced output** of blocks for the purpose of recovery

Module Summary

- Familiarized with the organization for database files
- Understood how records and relations are organized in files
- Learnt how databases keep their own information in Data-Dictionary Storage – the metadata database of a database
- Understood the mechanisms for fast access of a database store

PPD

Instructor and TAs

Name	Mail	Mobile
Partha Pratim Das, Instructor	ppd@cse.iitkgp.ernet.in	9830030880
Srijoni Majumdar, TA	majumdarsrijoni@gmail.com	9674474267
Himadri B G S Bhuyan, TA	himadribhuyan@gmail.com	9438911655
Gurunath Reddy M	mgurunathreddy@gmail.com	9434137638

Slides used in this presentation are borrowed from http://db-book.com/ with kind permission of the authors.

Edited and new slides are marked with "PPD".