Database Applications

CMPT 354

- Database Programming
- Application Architecture
- Objects and Relational Databases

Database Programming

Database Applications

- Users do not usually interact directly with a database via the DBMS
  - The DBMS provides a "back-end" that stores the data on a database server
- A separate application is built on top of the database
  - In a general purpose programming language
  - That retrieves and manipulates the data in the database as necessary

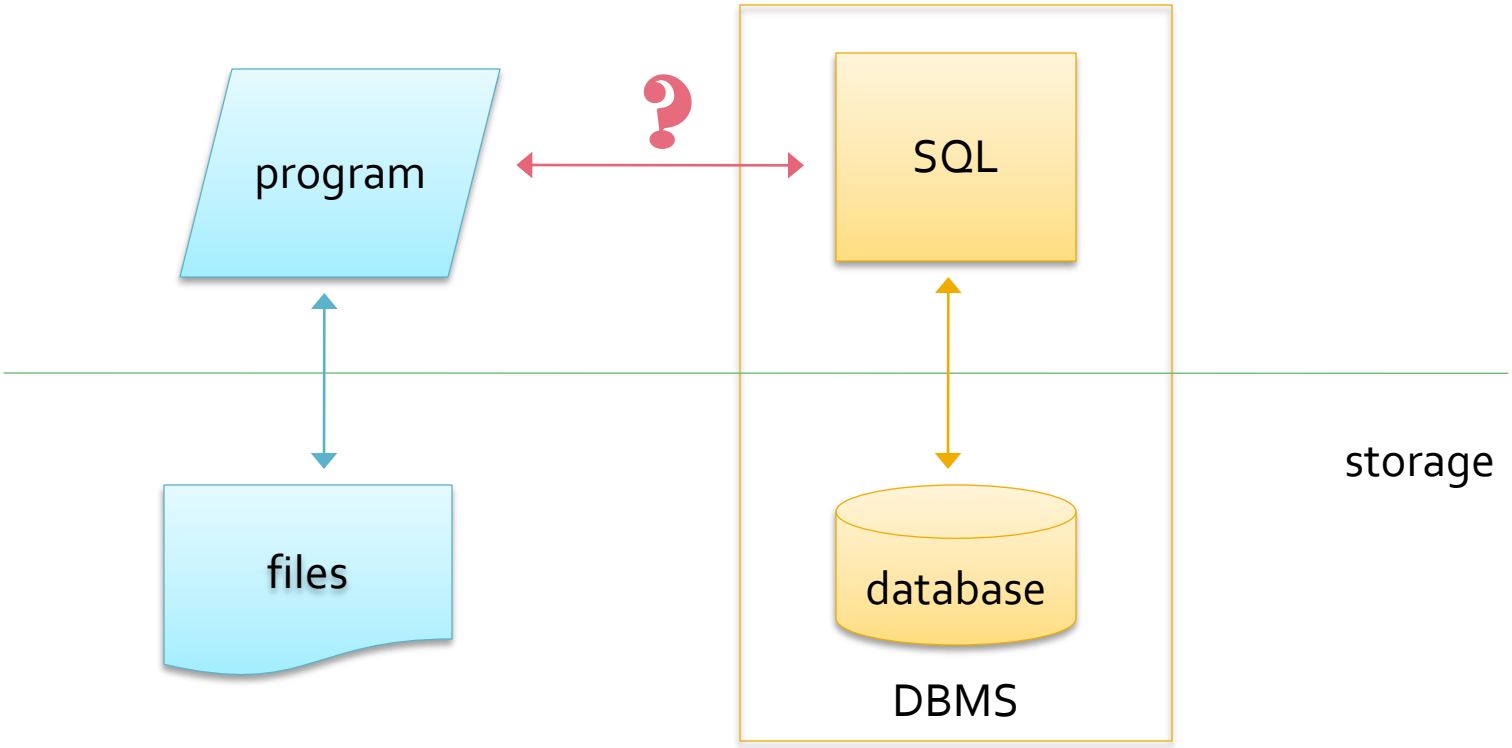
Application Types

- Many different types of applications require data stored in a database
- The applications can therefore vary in depth and sophistication and might include
  - Interface to provide appropriate access to data
  - Business logic to ensure consistency of data
    - Providing constraints that are not easily implemented in the underlying DBMS
  - Sophisticated presentation of data
  - Substantial processing and computation of data

Accessing a Database

- In an application an SQL DB may have to be accessed from a programming language
- The database has to be connected to the host language
  - The host language is the programming language that the application is written in
  - Through a variety of connection protocols
  - Data from the database has to be read into programming language constructs

Programming Environment

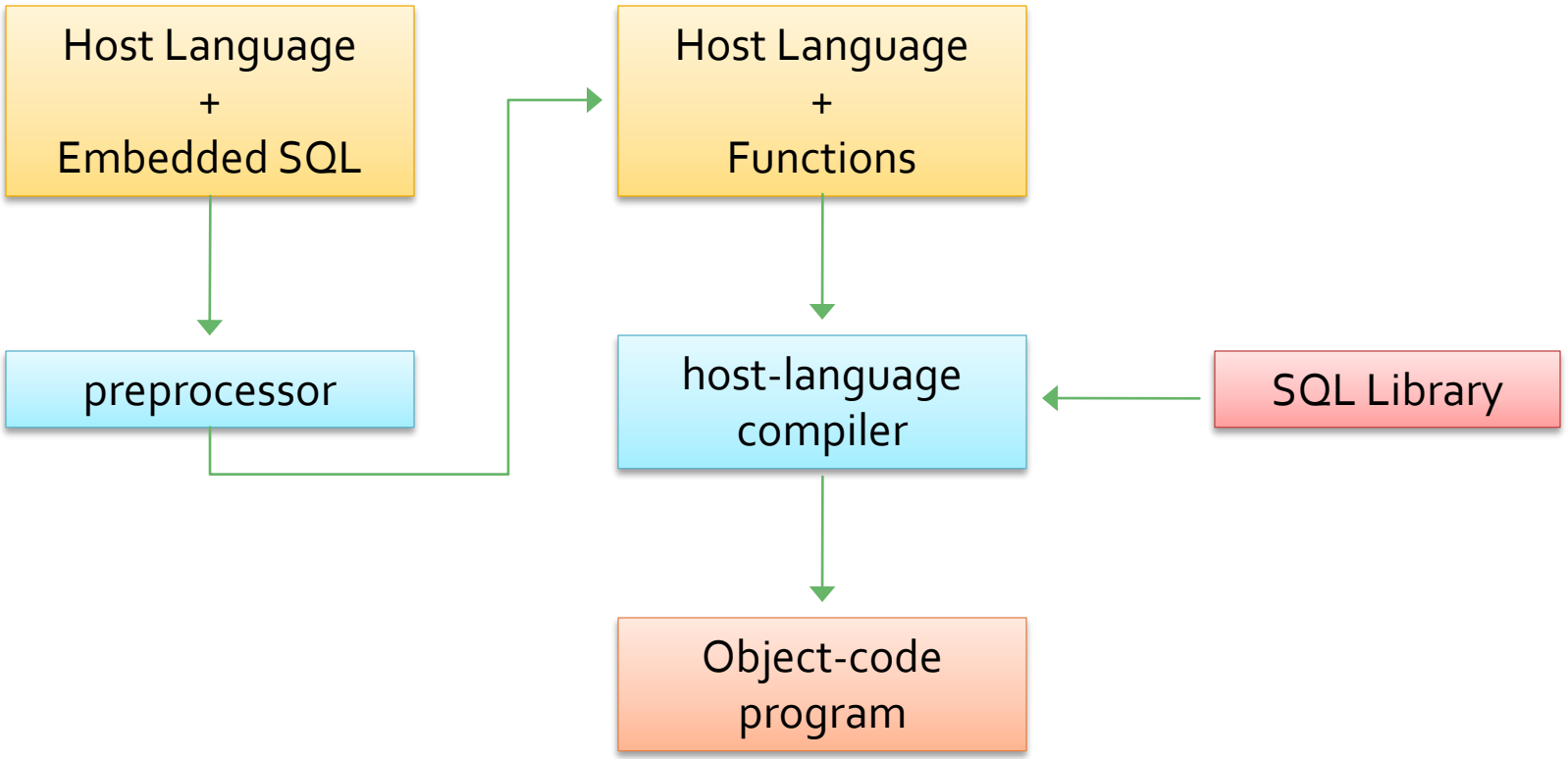


Connecting to the DB

- There are a number of ways to access DB data from within a host language application
  - Embedded SQL
    - SQL statements are identified at compile time with a preprocessor
  - Dynamic SQL
    - Allows a program to construct SQL queries at runtime
    - e.g. ODBC and JDBC
  - Mapping DB tables to host language constructs
    - e.g. CRecordSet
- *Stored procedures* can be called from an application to perform to perform previously specified tasks

Embedded SQL

Embedding SQL



Embedded SQL

- SQL statements can be *embedded* in some host language
  - The SQL statements have to be marked so that they can be dealt with by a preprocessor
  - Host language variables used to pass arguments into an SQL command must be declared in SQL
    - However SQL data types may not be recognized by the host language, and vice versa, and
    - SQL is set-oriented
- The syntax for embedded SQL varies by language, assume that the host language is C

Declaring Variables

- To declare host language variables to be used in SQL statements
  - `EXEC SQL BEGIN DECLARE SECTION;`
    - `long sin;`
    - `char[20] fname;`
    - `char[20] lname;`
    - `int age;`
  - `EXEC SQL END DECLARE SECTION;`
- Embedded SQL statements must be clearly marked
  - In C they must be preceded by EXEC SQL
  - When used in SQL variables must be prefixed by a colon

Embedded SQL Statements

- This embedded SQL statement inserts a row into the Customer table
  - The statement uses values that have been assigned to host language variables:

```
EXEC SQL
INSERT INTO Customer VALUES
    (:sin, :fname, :lname, :age);
```

- Note that the entire statement is one line of C code, and is terminated by a semi-colon (;)
  - Just like any normal statement in C

SQL Statement Status

- How do we know if an update or other statement succeeds?
  - SQLSTATE connects the host language program with the SQL execution system
- SQLSTATE is an array of five characters
  - Whenever a function of the SQL library is called the value of SQLSTATE is set to a code
    - '00000' – no error occurred
    - '02000' – a requested record was not found
  - Can be stored as a string and analyzed

Storing SQL Query Results

- As long as an SQL statement only returns one record the value can be stored in a variable
- The keyword INTO is used in the SELECT clause to insert data into a variable

```
EXEC SQL SELECT AVG(income)
FROM Customer WHERE age = :age
INTO :avgIncome;
```

Cursor Introduction

- SQL operates on sets of records, and SQL queries frequently return sets of records
- Consider this embedded SQL statement

```
EXEC SQL SELECT sin, lname
FROM Customer WHERE age = :age
INTO :sin, :lname;
```

- The idea is to read SINs and last names into the given variables
  - However, the query returns a set of *sin*, *lname* pairs, and the set cannot be cast into the variables
- The solution is to use a *cursor* to retrieve the set of records one record at a time

Cursors

- Cursors can be declared on any table or query
  - A cursor has a position in a relation so can be used to retrieve one row at a time
  - Cursors can be used to both query and modify tables
- Cursors can be declared with a number of different properties
  - `FOR READ ONLY`
  - `FOR UPDATE`
- Cursors must be opened before being used

Cursors and Queries

- Consider creating a cursor to query customer data
  - `EXEC SQL DECLARE custInfo CURSOR FOR`
    - `SELECT sin, age`
    - `FROM Customer`
    - `FOR READ ONLY`
  - `END_EXEC`
- The cursor has to be opened to evaluate the query
  - `EXEC SQL OPEN custInfo;`
- The `FETCH` command reads the first row of the table into the given variables, …
  - `EXEC SQL FETCH custInfo INTO :sin, :age;`
  - and moves the cursor to the next row

Closing Cursors

- The SQLSTATE is set to '02000' when no more data is available
- The close statement causes the database system to delete the query result
  - `EXEC SQL close custInfo;`
- The details for using cursors vary by language

Cursors and CURRENT

- The `CURRENT` keyword allows the row currently referred to by a cursor to be accessed
  - `EXEC SQL UPDATE Customer`
    - `SET income = 200000`
    - `WHERE CURRENT of custData;`
- Assuming that `custData` refers to the Customer table and has been set `FOR UPDATE`
- Note that the examples shown would require other host language constructs to behave as desired
  - Loops to iterate through every row in a table, and
  - If statements to update the desired record

Dynamic SQL

- Embedded SQL allows the use of variables to change the *values* being referenced in a query
  - But does not allow the program to create *new* SQL queries
- *Dynamic SQL* allows entire queries to be created at run-time
  - The query is first created as a string then parsed and executed
    - `char sqlString[] = {"DELETE FROM Customer WHERE age < 19"}`
    - `EXEC SQL PREPARE sqlCommand FROM :sqlString`
    - `EXEC SQL EXECUTE sqlCommand`

Little Bobby Tables

- Dynamic SQL allows users to write and run queries at run time
  - Without having to re-write program source code
- This is often performed through a user-friendly interface
  - Where queries are generated from choices entered by the user
- Be careful to prevent injection attacks

Connection Protocols

Database Connections

- A database connection allows a database server and client software to communicate
  - The client uses the connection to send commands and receive replies from the DB server
- Connections are built by supplying a driver with a connection string
  - The connection string gives the address of a DB

Making a Connection

- It is common for an application program to run on a different machine to the database
- The application has to
  - Open a connection to the database server
    - Which entails specifying the URL of the server's machine
    - And providing logon information
  - Once the connection is made the program can interact with the database
  - The connection should be closed when finished

Multiple Database Connectivity

- Embedded SQL programs allow the same source code to be compiled to work with different DBMSs
  - However, the embedded SQL calls are translated into host language functions by a DBMS specific preprocessor
  - Therefore the final executable code only works with one DBMS
- Various APIs allow a single executable to access different DBMSs without recompilation
  - *ODBC* and *JDBC* for example

ODBC and JDBC

- ODBC and JDBC integrate SQL with programming languages
  - ODBC – Open Database Connectivity
  - JDBC – Sun trademark (not Java Database Connectivity)
- The integration is through an *API* (*A*pplication *P*rogramming *I*nterface) and allows
  - Access to different DBMSs without recompilation and
  - Simultaneous access to several different DBMSs
- Achieved by introducing an extra level of indirection
  - A DBMS specific driver interacts with the DBMS
  - The drivers are loaded dynamically on demand

ODBC Components

- The *application* starts and ends connection with the DB
- The *driver manager* loads ODBC drivers and passes calls to the appropriate driver
- The *driver* makes a connection to the *data source* and translates between the data source and the ODBC standard
- The DB processes commands from the driver and returns the results

Making a Connection

- First create an ODBC data source
  - In Windows, using the Windows Data Source Administrator
- The connection methods vary by language and implementation
  - Depending on your choice of language and compiler there may be library classes to assist with connection
    - e.g. the Microsoft Foundation Classes CRecordSet classes
  - In addition the IDE may provide support for database applications

JDBC Components

- Like ODBC, JDBC has four main components
  - Application
  - Driver manager
  - Data source specific drivers
  - Data source
- There are different types of JDBC drivers
  - The types are dependent on the relationship between the application and the data source

JDBC classes and Interfaces

- JDBC contains classes and interfaces that support
  - Connecting to a remote data source
  - Executing SQL statements
  - Transaction management
  - Exception handling
- The classes and interfaces are part of the java.sql package
  - Programs must be prefaced with `import java.sql.*` to allow access to these classes and interfaces

Other APIs

- There are numerous APIs that assist in rapid development of DB applications
  - One example is Django
- Django is an open-source web application framework
  - Written in Python
  - Includes classes that aid in accessing data from the underlying DB

Cursors Re-visited

- There are many class libraries that are written for database programming
  - In many different programming languages
- Such libraries often include implementations of cursors
  - MS C++ CRecordSet
  - Java ResultSet
  - Python pymssql cursor
  - These classes have methods which allow records in tables to be accessed and modified

Stored Procedures

Stored Procedures

- A *stored procedure* is a program executed with a single SQL statement
  - The procedure is executed at the DB server and
  - The result of the stored procedure can be returned to the application (if necessary)
- This contrasts with the use of cursors
  - Which may require that DB objects be locked while the cursor is in use
  - DBMS resources (locks and memory) may be tied up while an application is processing records retrieved by a cursor
- Stored procedures are beneficial for software engineering reasons

Benefits of Stored Procedures

- Stored procedures are modular
  - It is easier to change a stored procedure than it is to edit an embedded query
  - This makes it easier to maintain stored procedures, and to
  - Change the procedure to increase its efficiency
- Stored procedures are registered with the DB server
  - They can be used by multiple users applications and
  - Separate server side functions from client side functions
- Stored procedures are written by DB developers
  - Who are more likely than application developers to have the SQL experience to write efficient procedures

Stored Procedure Support

- Many commercial DBMSs include support for stored procedures
  - Including SQL Server
- The SQL Standard includes a specification for stored procedures and functions
  - SQL/PSM – Persistent Stored Modules
    - Includes both procedure and function definition
    - Stored functions return values, procedures do not

Overview

	Static Queries Query form known at compile time	Dynamic Queries
Execution in Application Space	Embedded SQL	API: Dynamic SQL ODBC, JDBC, …
Server Execution	Stored Procedure SQL/PSM	

Application Architecture

Architectures

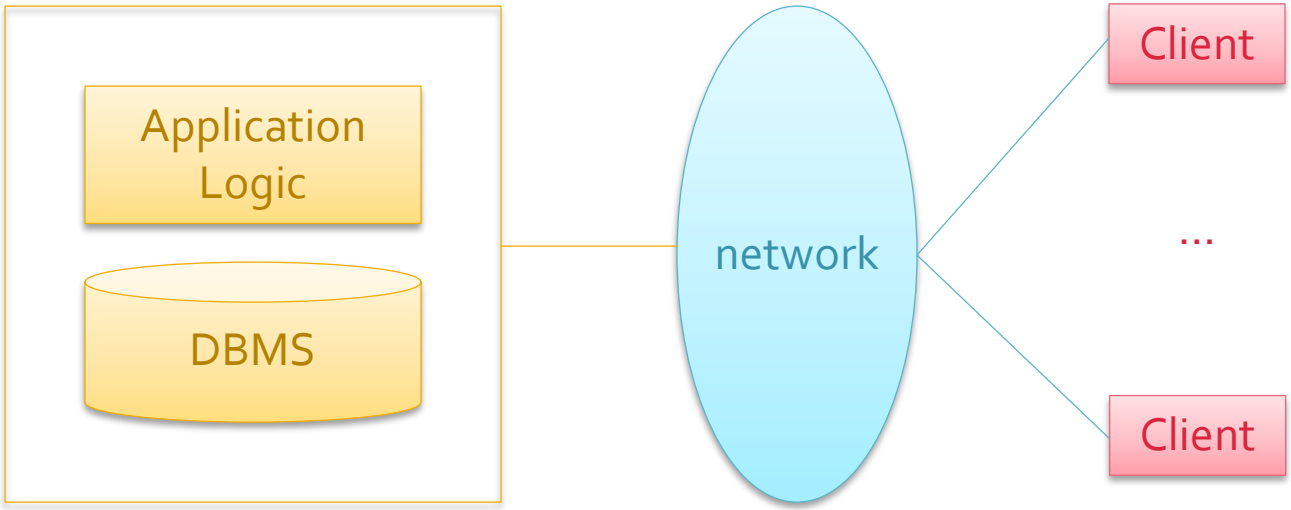
- Three broad categories of application architecture
  - Single tier
    - How things used to be …
  - Two tier
    - Client-server architecture
  - Three tier (and multi-tier)
    - Used for many web systems
    - Very scalable

Single Tier Architecture

- Historically, data intensive applications ran on a single tier which contained
  - The DBMS,
  - Application logic and business rules, and
  - User interface
- Typically, such applications ran on a mainframe and were accessed by users through *dumb terminals*
- Dumb terminals do not have the computational power to support GUIs
  - Centralizing computation of GUIs makes it impossible for a single server to support thousands of users

Client Server Architecture

- Two-tier architectures consist of client and server computers
- Typically, the client implements the GUI
  - Referred to as *thin clients*, e.g. streaming services such as Netflix
- The server implements the business logic and data management



Thin vs. Thick Clients

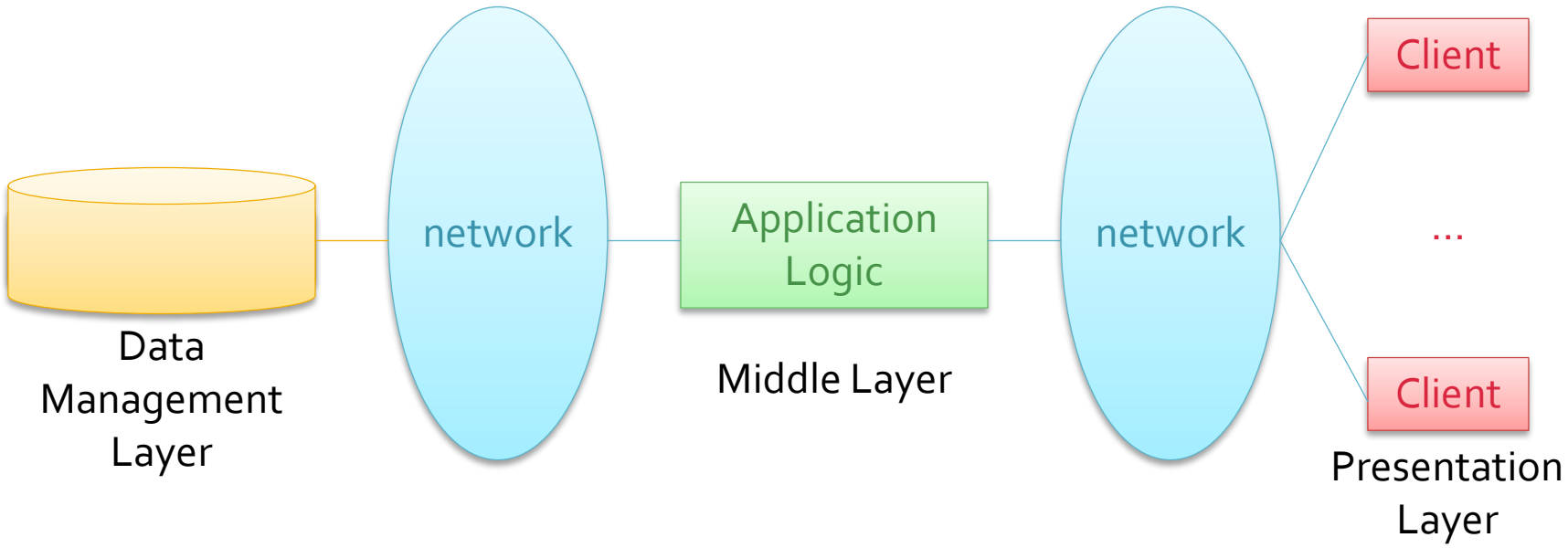
- A *thick client* is one where the client software implements the UI and part of the business logic
  - Example – computer game
- Thick clients are less common because
  - There is no central place to update and maintain the business logic
  - The server has to trust that the client application code will run correctly (and is not tampered with)
  - Thick clients do not scale as well
    - More communication is required between the application and the DB than between the UI and the application

Clients

- Clients are not responsible for data processing
  - Request input from users and data from server
  - Analyze and present the data from the server
- Clients are not dependent on the location of the data
- Clients can be optimized for the presentation of the data
  - And can display data differently dependent on the client processor

Three-Tier Architecture

- The thin-client two-tier architecture separates presentation from the rest of the application
- Three-tier architecture further separates the application logic from the data management



Presentation Layer

- Responsible for handling the user's interaction with the middle tier
  - One application may have multiple versions that correspond to different interfaces
    - Web browsers, mobile phones, …
    - Style sheets can assist in controlling versions
- The presentation layer itself may be further broken up into layers

Style Sheets

- Different clients may have widely differing displays
  - e.g. black and white screens, or phone displays
- A style sheet is a method to format the same document in different ways
  - The same document can be displayed differently depending on the context allowing reuse
  - Documents can be tailored to the reader's preference
  - Documents can de displayed differently on different output devices
  - Display format can be standardized by using the same style sheet conventions to multiple documents

Business logic Layer

- The middle layer is responsible for running the business logic of the application which controls
  - What data is required before an action is performed
  - The control flow of multi-stage actions
  - Access to the database layer
- Multi-stage actions performed by the middle tier may require database access
  - But will not usually make permanent changes until the end of the process
    - e.g. adding items to a shopping basket in an Internet shopping site

Data Management Layer

- The data management tier contains one, or more databases
  - Which may be running on different DBMSs
- Data needs to be exchanged between the middle tier and the database servers
  - This task is not required if a single data source is used but,
  - May be required if multiple data sources are to be integrated
  - *XML* is a language which can be used as a data exchange format between database servers and the middle tier

Example: Airline reservations

- Consider the three tiers in a system for airline reservations
- Database System
  - Airline info, available seats, customer info, etc.
- Application Server
  - Logic to make reservations, cancel reservations, add new airlines, etc.
- Client Program
  - Log in different users, display forms and human-readable output

Example: Course Enrollment

- Student enrollment system tiers
- Database System
  - Student information, course information, instructor information, course availability, pre-requisites, etc.
- Application Server
  - Logic to add a course, drop a course, create a new course, etc.
- Client Program
  - Log in different users (students, staff, faculty), display forms and human-readable output

Three-Tier Advantages

- Allows heterogeneous systems
  - Applications can use different platforms and software components at the different tiers
  - It is easy to modify or replace code at one tier without affecting other tiers
- Thin clients – clients only require enough processing power for the presentation layer
- Integrated data access
  - In some cases the data must be accessed from several sources
  - The middle tier can manage connections to all databases and integrate the sources

Three-Tier Advantages 2

- Scalability
  - Clients are thin and access to the system is controlled by the middle tier
  - If the middle tier becomes a bottleneck, more resources can be deployed
    - And clients can connect to any middle tier server
- Software development benefits
  - Dividing the application into natural components makes it easier to maintain
  - Interaction between tiers can occur through standardized APIs
    - Allowing for reuse, and more efficient development

Web Databases

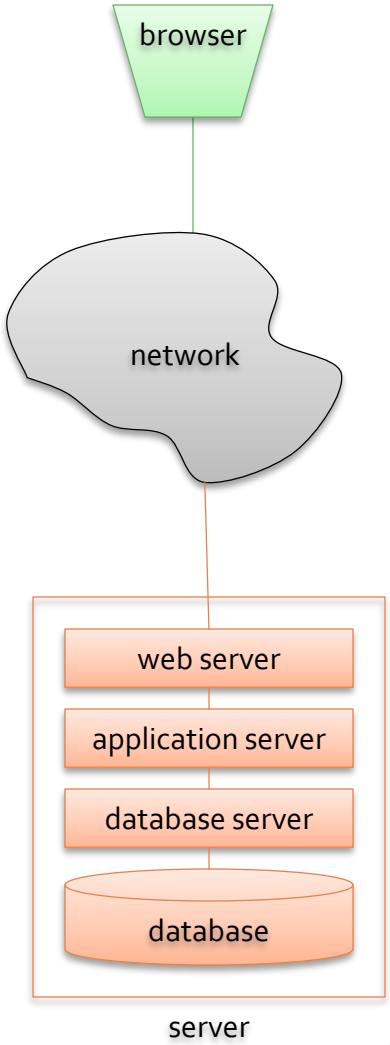
- Web interfaces to databases are prevalent
- The web is an important front end to many databases
  - Web browsers allow access to data on servers located anywhere in the world
  - Web browsers can run on any system and users do not need special purpose software
  - Each document that can be accessed on the web has a unique name (URL)

3 Tier Architecture and the Web

- In the domain of web applications three tier architecture usually refers to
  - Web server
  - Application server
  - Database server
- In this architecture the client accesses the web server
  - Sometimes referred to as 4 tier
    - Or, generically as *n* tier

Web Fundamentals

- A web server runs on the server machine
  - It takes requests from a browser and sends back results as html
    - The browser and web server communicates via http
- The web server also communicates with applications providing a service
  - The Common Gateway Interface defines how web servers communicate with applications
  - Applications communicate with a database server
    - Usually via ODBC, JDBC or other protocols



Hypertext Transfer Protocol

- What is a communication protocol?
  - A set of standards that defines the structure of messages
  - Examples: TCP, IP, HTTP
- What happens if you click on http://www.cs.sfu.ca/CourseCentral/354/johnwill/?
  - Client (web browser) sends HTTP request to server
  - Server receives request and replies
  - Client receives reply

HTTP has no states

- HTTP is stateless
  - No sessions
  - Every message is completely self-contained
  - No previous interaction is remembered by the protocol

Maintaining State

- The *state* of a user's interaction often needs to be maintained across different web pages
  - e.g. a web shopping site
- State can be maintained at the middle tier
  - Either in main memory
    - But such state is volatile, and may use a lot of space
  - Or in local files, or even in the database tier
  - Generally state is only stored in the middle tier if the data is required for many user sessions
- State can be maintained at the presentation tier
  - cookies …

Client State: Cookies

- Storing text on the client which will be passed to the application with every HTTP request.
  - Can be disabled by the client
  - Are perceived as "dangerous”
  - May scare away site visitors if asked to enable cookies
- Are a collection of (Name, Value) pairs

Client State: Cookies

- Advantages
  - Easy to use in Java Servlets / JSP
  - Provide a simple way to keep non-essential data on the client side even when the browser has closed
- Disadvantages
  - Limit of 4 kilobytes of information
  - Users can (and often will) disable them
- Should use cookies to store interactive state
  - The current user's login information
  - The current shopping basket
  - Any non-permanent choices the user has made

Multiple state methods

- Typically multiple methods of state maintenance are used
  - User logs in and information is stored in a cookie
  - User issues a query which is stored in the URL
  - User places an item in a shopping basket cookie
  - User purchases items and credit-card information is stored and retrieved from a database
  - User leaves a click-stream which is kept in a log on the web server

Connectionless Sessions

- When a web server receives a request a *temporary* connection is created
  - The connection is closed after the response is received from the server
  - Leaving connections available for other requests
  - Information has to be stored at the client and returned with each request, in a *cookie*
- In contrast to an ODBC or JDBC connection
  - Session information is retained at the server and client until the session is terminated

Server Side Scripting

- Scripting can provide an alternative to writing web applications in languages like Java or C++
- Scripting languages allow constructs to be embedded in html documents
  - Before a web page is delivered the server executes the embedded scripts
  - Scripts may contain SQL code
- Scripting languages include JSP, ASP, PHP, …
  - Many of these come with tools and libraries to give a framework for web application development

Client Side Scripting

- Scripting languages can also add programs to webpages that run directly at the client
  - e.g.Javascript, PHP, ColdFusion
- Scripting languages are often used to generate dynamic content
  - Browser detection to detect the browser type and load a browser-specific page
  - Form validation to perform checks on form fields
  - Browser control to open pages in customized windows (such as pop-ups)

Objects and Relational DBs

Material from Ted Neward's blog article – The Vietnam of Computer Science

Object Relational Mapping

- Most modern programming languages are object oriented
  - Classes contain complex types that have composite and multivalued attributes
  - A relational database only allows atomic attributes
- Object relational mapping (ORM) is the process of converting data between the two systems
  - There are a number of automated ORM tools available on the market
  - Conversion from an OO to relational data types is a non trivial problem

OR Impedance Mismatch

- An impedance mismatch is a system where inputs and outputs do not match
  - In our case the object and relational data models
- Object systems have four basic components
  - Identity
  - State
  - Behaviour
  - Encapsulation
- Relational systems contain relations in tables
  - A relation is a statement of facts about the world
  - From which other statements can be derived using set operations

Object to Table Mapping

- How should classes be matched to tables?
  - Tables to classes
  - Columns to member variables
- How can inheritance be dealt with?
  - Table for each class,
  - Table for each concrete class, or
  - Table for each class family
- We have discussed the first of these three alternatives
  - The most obvious solution, but it can lead to problems

Complex Data Types

- Complex data types are composed of composite or multi-valued attributes
  - A composite attribute has multiple component attributes
    - e.g. address composed of street, city, province
  - Multi-valued attributes contain sets of values
    - e.g. a person's phone numbers
- First normal form requires that all attributes have atomic domains
  - Complex attributes are not atomic

Mapping Subclasses to Tables

- If each class in an inheritance hierarchy gets its own table the hierarchy requires multiple tables
- An object in an OO system is a sequence of memory that contains all of its member variables
  - To extract the equivalent set of data from a DB involves joining each of the class hierarchy tables
  - If the class hierarchy is large the sequence of joins can be very expensive
- Therefore the other approaches (e.g. one table per hierarchy) are often used
  - Even though these approaches are more complex

Schema Ownership Conflict

- Who owns the DB schema?
  - In many organizations the DB schema will not be under the direct control of developers
  - But will be owned by the DBA group
- Typically, at some point, the DB schema will be frozen at some point
  - Creating a barrier to object model refactoring
- Note that this is more of a political than a technical problem

Dual Schema Problem

- In an ORM system the metadata is held in two places
  - In the database schema and
  - In the object model
- Updates or refactoring in one schema force updates in the other
  - It is usually considered easier to refactor code to match the DB schema than vice versa
  - An application often serves a single purpose whereas DBs are often used by many applications
  - It may become necessary to allow object models to diverge from the schema to avoid expensive global changes

Entity Identity

- Objects have an implicit sense of identity
  - The *this* pointer
- In a relational DB identity is implicit in the state of the data
  - Two rows with identical data are usually not permitted
- It is possible that an attempt may be made to insert two records with the same data
  - That correspond to two different objects

Data Retrieval

- Once an entity is stored in a DB how is it retrieved?
  - In OOP objects are created through the use of constructors
  - And an object should be responsible for its own data
- Retrieving data from a DB may entail some form of embedded SQL queries
  - Which are easy to write incorrectly
    - They involve strings which are easy to mistype
  - And require a functioning DB to test

Partial Object Problem

- Satisfying an SQL request is relatively expensive compared to local network calls
  - It typically involves traversing a network
  - In SQL query optimization only required rows and columns and retrieved
- This suggests that in the interests of efficiency only part an object is retrieved
  - Objects must therefore allow nullable fields,
  - All object variables should be filled out on retrieval with the associated performance issues, or
  - Object variables should be loaded on demand

ORM Solutions

- Abandonment – give up on objects!
- Acceptance – give up on relational storage
  - And maybe use NoSQL
- Manual mapping
- Acceptance of ORM limitations
- Integration of relational concepts into the language