CS352 Lecture - Database Application Development

Last revised 10/9/06

Objectives:

1. To discuss possible architectures for a client-server application that uses a database
2. To discuss use of embedded SQL

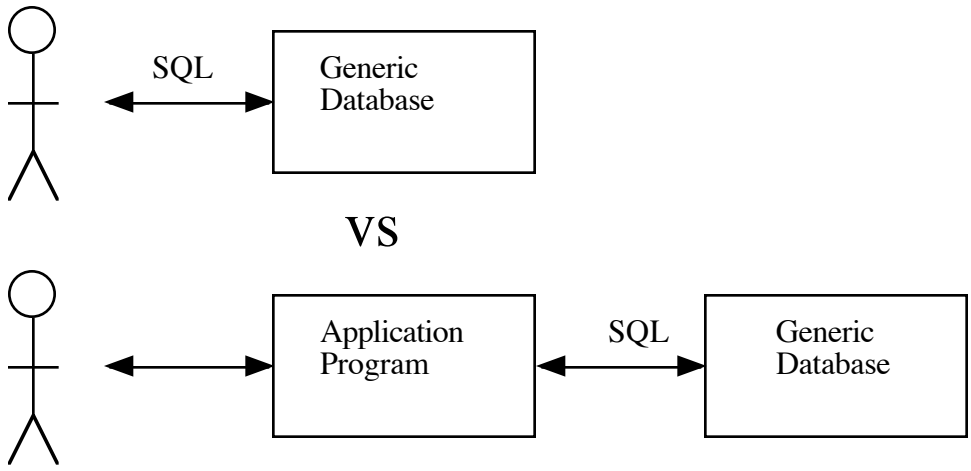
Materials:

1. Projectable of three and two tier architectures
2. Projectable of book figure 8.7
3. Projectable of db2 create procedure
4. Projectable of JDBC code example + variant using a prepared statement
5. Projectables of sqlj example - sqlj + pure Java

I. Introduction

A. Thus far, we have used SQL as the means of actually accessing/modifying the database.

B. Of course, the majority of people accessing information stored in a database don't do so directly using SQL. Instead, they run an application program. While the application program may store its data in various kinds of application-specific files (the file processing approach), frequently it stores its information in a "generic" (e.g. SQL-based) database.



Examples of the latter?

ASK

Numerous - many web-based ecommerce systems use a database to actually store the data; also bank tellers, insurance agents ...

II. Architectural Alternatives

A. When a user interacts with an application program that stores its data in a generic database, there are three kinds of tasks that are performed:

1. Database tasks - tasks related to accessing/modifying information in the database (e.g. tasks corresponding to SQL `select, insert, update, delete` or the equivalent in some other DML).

2. "Business logic" tasks - tasks related to the actual logic of the application (which vary widely from application to application, of course.) Quite a few things might fall into this category, including

   a) Carrying out the task(s) that the software is designed to do: displaying information, recording purchases/reservations/..., etc.

   b) Ensuring that the appropriate "business rules" are adhered too - e.g., for example, if a system is registering students for courses one important "rule" that needs to be enforced is that a student cannot be signed up for two different courses meeting at the same time (at all, or perhaps without some sort of special permission)

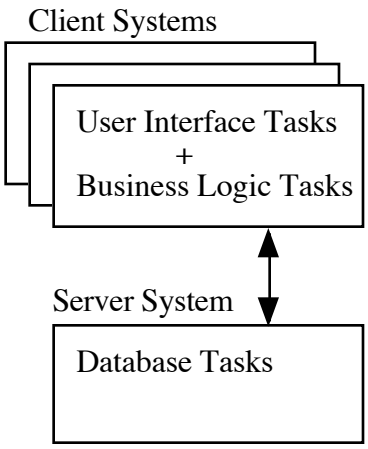
   c) Ensuring that users are properly authenticated if sensitive information is being made available or data is being modified.

   d) etc.

3. User interface tasks - tasks concerned with presenting information to the user and accepting commands from the user. This could be via a command line interface or some kind of dedicated hardware (e.g an ATM), but is often done through a GUI.

B. While it is certainly possible for all three kinds of tasks to be done by the same program, many applications make use of some variant of a client-server model, where the database tasks are performed by a "database server" and the user interface tasks are performed by a client computer - typically a PC or the like. Where there is considerable variation is in the matter of the placement of the business logic tasks.

1. Thick-client architectures:



2. Thin-client architectures: in a thin client architecture, only the user interface tasks are performed by the client. The business logic tasks and database tasks are performed by server(s).

   Often (though not always), the client is a web browser. In this case, we need an additional server - the web server that delivers up html upon request from the client. A couple of configurations are common:

   a) It is possible to actually have three servers (though they might actually be separate processes on the same computer). In this case, when the client requests a computation, it is received by the web server and passed on to the application server which in turn accesses/updates the database as needed before passing a response back to the client via the web server.

      This is often called a "three-tier" architecture.

      PROJECT: Authors powerpoint

   b) It is possible for the web server to actually perform the business logic computation, contacting the database server as needed to access/update the database.

      This is often called a "two-tier" architecture.

      PROJECT: Authors powerpoint

   c) It is also possible for the database server to actually perform the computation on request from the web server, using stored procedures.

C. Each of these Architectures has advantages

1. Thick client:

   ASK

   a) Much of the computation is done on the client systems, minimizing load on servers

2. Thin Client

   a) Easier to secure, since the business logic software is on server systems. (Business logic software on client systems is more easily attacked/spoofed; and the business logic software may need to contain some sort of authentication information that should not be generally available.)

   b) Updating the business logic software is easier, since it resides only on server(s).

D. Software support for the various architectures

1. Thick client

   a) Business logic software needs to reside on the client. This can be an actual program installed on the client which contains both user interface and business logic code, communicating with the database server as needed.

      Example: The Video Store project in CS211 could be configured this way.

      Example: Your programming project in this course will be configured this way

   b) Alternately, the business logic can be contained in code that is downloaded when needed - e.g. a Java applet or Java web start. (This is more appropriate when the client will run the system relatively infrequently.)

2. Thin client

   a) If a separate application server/program is used, typically, the URL sent by the client tells the web server to send a message to the application server that invoke the business logic.

   b) It the web server actually does the business logic tasks itself, then one of two approaches may be used:

(1) The web server may utilize small programs called "servlets". These are similar to applets, except that they run on the server system, rather than the client system.

In this case, the business logic is a Java program that runs on the server.

(2) The web pages themselves may contain embedded scripts that the server executes while sending the page. (e.g. JSP, ASP, ColdFusion).

In this case, the .html file that the server sends the client in response to a URL is actually generated from the html file stored on the server's disk. Some of it may be literal text that is just passed through to the client, while another part may be a script that the server executes whose output is the html that is sent to the client (perhaps representing in html table form the result of some database query.) As the server processes the page, it recognizes the script portions by an appropriate tag.

That is, the business logic is written in some scripting or programming language embedded in html.

PROJECT: Figure 8.7 from book

c) If the database server does the business logic tasks, then the the business logic is contained in stored procedures / SQL modules that are stored in the database. In this case, the business logic is actually written in SQL, and the UI code running on the client invokes these procedures through a connection to the database server.

PROJECT: Example of SQL create procedure from db2 examples

E. A key issue in a web-based system is the notion of a session.

1. The http protocol is a connection-less protocol. This means that no long-lasting connection is established between the client and server; rather, each request is treated as a separate operation, independent of all others.

   There is a good reason for this in terms of the general architecture of the web. Popular web servers may handle requests from thousands of clients. Often, a client will request just one or two pages from a given server, before following a link to another site. Keeping a connection open consumes system resources, with no reason in many cases.

2. On the other hand, when the web is used for ecommerce or other operations requiring some form of user authentication, a problem arises if each access is treated as independent of all others, even from the same client.

   a) One solution is to require each access to include authentication information (e.g. a username and password). This would either require that the user be asked to supply this information for each operation (which would be painful, to say the least), or the browser would need to store the information and append it to each request (which is not supported directly in http).

   b) More typically, this issue is addressed by the notion of a session.

      (1) When a user first accesses a site - or perhaps when the user chooses an operation such as "login" - the user is required to supply authentication information - e.g. a username and password.

      (2) When the server receives such a request from the client, if the authentication information supplied is valid, it creates a session object which records the results of the authentication (i.e. who the user has proved himself/herself to be.)

      (3) Future requests from the same client are associated with this same session object, and are handled based on the authentication previously established for the session.

      A common way to handle this is using a cookie - a small string stored by the browser. (This is not the only way)

         (a) A server can request a browser to set a cookie as part of its reply to a request from that browser.

         (b) A cookie always stores the url (or portion of the url) of the server that set it.

         (c) A cookie can be requested from the browser by the server; but the browser will only send a cookie back to the url that set it.

         (d) When a cookie is used as part of authentication, it typically stores a session-id - a large, random number that the server uses to locate the session object [using some sort of hashtable] when subsequent requests arrive.

3. A crucial security issue arises with regard to sessions, though. Suppose a user authenticates and a session is created. Now suppose a malicious user is somehow able to "hijack" this session (perhaps by obtaining the session id by sniffing a packet that is part of the communication between the true client and the server) The malicious user would then be able to do anything the original user was allowed to do. Here are some measures used to address this.

   a) A session is typically associated with a specific IP number, and traffic purporting to be from that session from a different IP number is rejected. (But, of course, IP numbers can be spoofed)

   b) Encryption of session ids to prevent sniffing.

   c) The ability for the user to terminate a session by explicitly logging out.

   d) Fairly short timeouts for a session, so that it is automatically terminated even if the user forgets to log out.

III. Accessing the Database from Within a Program

A. Regardless of which architecture is used, it necessary for some portion of the program to access the database. Unless the business logic is entirely written in SQL, this will typically take the form of code written in some other language executing SQL statements.

   1. In a thick client system, this program may run on the client, either standalone or as an applet or web start application.

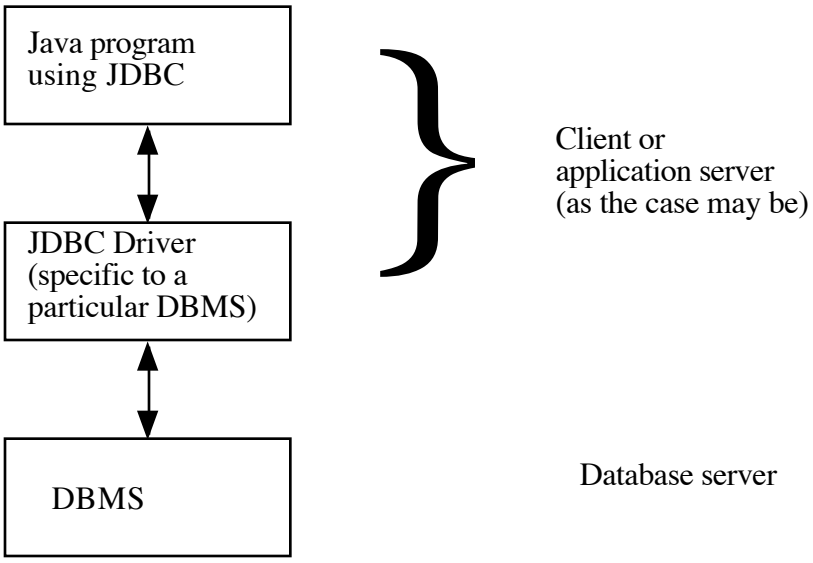
   2. In a thin client system, this program may run either on an application server or within the web browser

B. In either case, as we noted at the start of the course, one of two approaches is typically used to allow the application program to access the database

   1. One approach - called <u>dynamic SQL</u> - involves the application program generating SQL statements as character strings. Such a string is then passed to the database, where it is interpreted.

      a) In the case of Java, the technology used is called JDBC - Java Database Connectivity. (You have used this for a lab in CS211.)

b) A system that uses JDBC has the following architecture:



(1) The Java program itself will work on any platform that supports Java and with any DBMS that supports JDBC.

(2) The JDBC driver - a collection of classes typically furnished by the database vendor - serve to translate operations in the Java program into network messages to the database server in a form that is appropriate to the specific DBMS being used. (Thus, each brand of DBMS needs its own JDBC driver; but since the JDBC driver is often written in Java, it could be platform-independent. However, some drivers do use platform-specific native code, in which case they are platform-specific as well.).

c) Just to review, here is an example of JDBC code (from CS211 lab).

PROJECT

(1) A SQL statement is constructed as a character string, using string concatenation as with any Java string.

(2) A SQL statement is executed by passing it as a parameter to a method of class `java.sql.Statement`, which in turn passes it to the JDBC driver, which in turn passes it on to the database - with the result travelling back to the program via the reverse of the same path.

(3) The process of parsing the SQL statement is computationally expensive. For some statements (e.g. those involving joins), some effort may also be expended by the DBMS on planning a good strategy for executing the query (we will see later in the course how much of a difference this can make). This can also be computationally expensive.

Because of the compiutation potentially involved in parsing and strategy planning, JDBC incorporates the notion of a "prepared statement", which allows this work to be performed once - when the statement is prepared - rather than each time it needs to be executed. A prepared statement can have parameters, which are values to be supplied when the prepared statement is actually executed.

PROJECT - revision of JDBC to use a prepared statement

(a) Note the use of ? as a placeholder in original statement

(b) Note the use of an appropriate method (in this case `setString()`, though there are many others) to set the parameters.

(c) Note the use of `executeUpdate()` to actually execute the prepared statement with the specified parameters., (There is also an `executeQuery()` that returns a `ResultSet`, which is used with `select` statements.)

d) JDBC is actually based on an earlier technology (originated by Microsoft) called ODBC - Open Database Connectivity - which supports database access by application programs written in a variety of different programming languages through a library. (The architecture is similar to JDBC, except for the presence of an ODBC library between the program and the driver; for this reason, ODBC drivers are platform-specific).

2. The other approach - called <u>static SQL</u> - involves the application program being a mixture of SQL and some other programming language.

a) The language in which the SQL is embedded is called the <u>host language.</u>

(1) The syntax details for any particular language are common to that language - regardless of what brand of DBMS is being used.

(2) Of course, SQL itself is also (relatively standard).

(3) However, the actual implementation of embedded SQL for a specific language/DBMS is typically furnished by the DBMS vendor.

b) Embedded SQL implementations exist for a variety of host languages. For example, db2 comes with support for embedding SQL in C/C++, COBOL, FORTRAN, Perl, and REXX.

c) We will be using this approach for the programming project in the course.

d) Here is an example of embedded SQL code, using Java as the host language.

PROJECT

(1) The overall program is a standard Java program.

(2) The program contains embedded SQL statements. In the case of Java, the embedded SQL is bracketed by `#sql { ... }`. (Other languages typically bracket the SQL by a construct like `exec SQL ... ;`).

(3) Information flows between the two languages via host variables. A host variable is a variable in the syntax of the host language, and can be used as such by the host language code. In the SQL code, it is a marked as being a host variable by being preceeded by a colon - e.g. `categoryName` is a Java parameter that is used in the SQL code under the name `:categoryName`.

(a) A host variable may serve to pass information from the Java program into the SQL code (e.g. `:categoryName`).

(b) A host variable may serve to pass information from the SQL code back to the Java program. (e.g. `:checkoutPeriod`).

(c) Not illustrated here - but also possible - is a host variable that serves both purposes.

(4) A SQL statement is executed when it is encountered in the normal flow of execution of the host language code - i.e. it is executed just as if it were a host language statement. In fact, embedded SQL statements can appear in host language control structures, in which case they may be executed conditionally or repeatedly.

C. When a program uses embedded SQL, a more complex process is needed to prepare it for execution. For example, here is the process used by db2 for SQL code embedded in Java - referred to as sqlj. (The process details will vary from DBMS to DBMS).

1. The process begins with a source file that contains a mixture of Java and SQL code (as in the example just projected). Such a file typically has the file type .sqlj.

2. The sqlj file is processed by the sqlj compiler (furnished by IBM), which splits it into two files:

   a) A "pure java" file that contains the Java code plus calls to SQL procedures.

   b) A SQL module file.

   c) For example, the following is an excerpt from the results of invoking sqlj on the file projected earlier. (This excerpts correspond to the routine we just looked at.)

   PROJECT - Java code for example procedure;

   (In the case of the db2 implementation of sqlj, the SQL module code.is not created in a human-readable form, so I can't project that)

3. The "pure java" file is then translated by the standard Java compiler into one or more class files.

4. The SQL code is bound to the database, using a program furnished by IBM. In effect, what happens is that stored procedures are created in the database that correspond to the SQL code in the original program.

   As part of the translation process, the textual form of the SQL statements is replaced by a compiled form. Thus, the task of parsing the SQL and constructing an optimized strategy for processing a query is done once at build time, rather than each time a given SQL statement is executed (as is the case with dynamic SQL).

5. When the program runs, it establishes a connection to the DBMS, and then executes the SQL statements embedded in the Java code as they are encountered during execution. An embedded SQL statement is executed by sending a message to the DBMS requesting it to execute the appropriate stored procedure that was created by translating the SQL.