# Database Application Programs

**PL/SQL, Java and the Web**

As well as setting up the database and running queries, it is vital to be able to build programs which manage the database

- although they will only do so through the DBMS

This needed to:

- provide a user interface
- impose constraints which SQL cannot manage
- program complex data manipulation

Programs can be added to a database in several ways:

- having some kind of programming system inside the DBMS
- providing application building tools – e.g. the Access Switchboard
- connecting a programming language to a database



## A Bit More SQL – Assertions

These are database-schema elements, like relations or views, which are defined by:

The condition is a boolean expression which may refer to any relation or attribute in the database schema

- Not present in Oracle or MySQL

Example – there must be more tutorial groups than students

Example – no tutorial group may have more than 20 members



## A Bit More SQL – Triggers

A trigger specifies actions that are to be taken if the database gets into a particular state or if a particular event occurs, e.g.

- by use of data manipulation statements (update, insert or delete),
- by use of data definition commands (create, drop, etc.),
- or by general system or user actions (shutdown, logon, etc. )

They can be used to performs tasks such as generating derived columns, enforcing referential integrity, enforcing complex business rules, maintaining synchronous table replicates, etc.

A trigger has three parts:

- the **event** that triggers the rule – e.g. update or logoff
- the **condition** to be checked – e.g. "age less than 100" (note this part is optional, if doesn't appear the event always triggers the action)
- the **action** to be taken – some SQL data manipulation operations

Triggers can be made to execute either before or after the event



## Some Sample Triggers

These triggers maintain a derived total salary column in the Department table – note the keywords new and old to refer to the original and updated version of a record:

```
create trigger Total_Salary1
    after insert on Employee  // every time a new Employee record is created
      for each row
          when (new.deptno is not null)          // if the dept number is available
              update Department D                // add to the new emp's dept
                  set D.totalSalary = D.totalSalary + new.salary
                          where new.deptNo = D.number
```

```
create trigger Total_Salary2
    after update of Salary on Employee          // every an Emp's salary is
    changed
      for each row
          when (new.deptno is not null)          // if the dept number is available
              update Department D                // replace old salary with new
                  set D.totalSalary = D.totalSalary + new.salary – old.salary
                      where new.deptNo = D.number
```

```
create trigger Total_Salary3
    after update of deptno on Employee          // when an Emp's dept is changed
      for each row
          begin
              update Department D1              // add salary to new department
                    set D1.totalSalary = D1.totalSalary + new.salary
                        where new.deptNo = D1.number
              update Department D2              // add salary to new department
                    set D2.totalSalary = D2.totalSalary– old.salary
                        where old.deptNo = D2.number
          end                                   // like } in Java
```

```
create trigger Total_Salary4
    after delete on Employee            // when an Emp is removed from the database
      for each row
          when (old.deptno is not null)// if the dept number is available
              update Department D               // take away old salary
                    set D.totalSalary = D.totalSalary + old.salary
                        where new.deptNo = D.number
```



## PL / SQL

This is a programming language and system which manages data inside the database

It allows SQL commands to be placed inside a program as in:

```
declare
    number agevariable = 0;
for j in 1..10 loop
     agevariable := ... some calculation;
    insert into mytable (id, age) values(j, agevariable );
end loop;
commit;
```

These can be put into procedures (like methods) to be repeatedly called



## Database Connections

Originally a DBMS was used in isolation. It held all of the data centrally, provided multiple interfaces, multiple connections and all of the functionality to manage that data

Increasingly, enterprises need to connect databases:

- to each other - since a single (perhaps merged or devolved) enterprise is likely to have several database systems that it uses
- to other applications, such as spreadsheets, which also manage data
- to the web

We will now look at the techniques involved in:

- writing applications which span multiple relational database systems
- writing applications with embedded databases
- connecting a database to the web



## Interoperability

**Key Slide**

When writing a database application, there is an increasing need to make it independent of the underlying database system which it is using:

i) for an in-house application in an enterprise in which more than one DBMS is in use
ii) for a software house developing software of general use

Relational systems with SQL provide a single model for data and so **should** provide good support for such general application development,

But:

i) the programming interface to SQL varies
ii) the functionality provided varies
iii) the flavour of SQL varies
iv) the system catalogue varies - e.g. domain names (varchar, text, varchar2)
v) some data is not in a database
vi) DBMS vendors pride themselves on adding extra functionality

## Open Data Base Connectivity (ODBC)

The solution is to provide a single Application Programmer's Interface (API) to relational systems and other data sources

Microsoft's ODBC:

- provides a single API for sending SQL to a source of data, where an ODBC driver is available to manage the interaction between application and data
- permits the source of data to be relational or not
- standardises the SQL grammar
- assumes a core set of functions, but allows others to be supported
- provides querying functions to determine the nature of the DBMS, the meta-data and the catalogue

It is therefore possible to write a program which runs on top of a number of data sources unchanged

However, it will still be easier to write the application if you know in advance what those data sources are rather than try to provide for anything that might come up

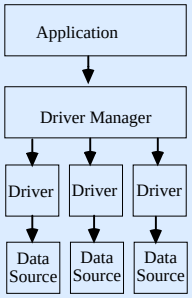


## ODBC Architecture

Key Slide

The architecture comprises four levels:

- The **application** carries out the processing and sends ODBC class to submit SQL statements and retrieve results.
- The **driver manager** identifies calls which identify the particular database and load and unload drivers appropriately.
- A **driver** is a DBMS-specific collection of code which implements ODBC calls, transforming them into the local DBMS flavour of SQL. The calls are then submitted to the data source and the data is retrieved and returned to the application.
- A **data source** is a collection of data (usually a database) which the driver has the ability to query.




Thus to get an application to use a database residing with a particular DBMS, a driver for the DBMS must be installed.



## The Structure of an Application which Uses ODBC

All ODBC applications have a similar structure:

1. An application begins by requesting and receiving a **connection** to a data source
2. Then it queries the data source for information about which facilities are actually available and how they are made available
3. It then allocates a **statement handle** in the context of the connection - this is a like a stream along which queries expressed as strings can be sent

4a. Having set up its local store, it sends SQL **queries** via statement handles to retrieve data and assign them to local variables

4b. Or (for updates), it sends the **updates** and checks the success of these by retrieving the number of rows affected

5. It processes these data, perhaps issuing more SQL calls to retrieve more data or update the data
6. It terminates the connection



## ODBC and SQL Statements

The execution of an SQL statement involves two phases:

i) A plan for executing the query is made by the DBMS - **preparation**

ii) The plan is carried out - **execution**

The application can separate these two by preparing the query using one call and executing it with another. Re-using the query will then save performing the preparation phase for all but the first use.

There are several different ways of building up SQL statements:

i) They are entered in the program complete as a single string

```
Query q = makeQuery( "select name from Student where matric=12345");
```

ii) They are built up gradually to become that complete string

```
int matric = ... code to get the number;
Query q = makeQuery( "select name from Student where matric= " + matric);
```

iii) They are parameterised, prepared and then only given the parameter just before execution

```
Query q = makeQuery( "select name from Student where matric = ?" );  // ? = parameter
q.bindParameter( 1, "12345");  // 12345 if the first parameter value
```

## Java and Databases

Increasingly, applications on top of databases are being programmed in Java

These slides are intended to illustrate how an application is programmed using ODBC through Java

The package for this is called **JDBC**

As a very simple example, we will assume the database is in Access and holds a table of student data, which looks like:

Student( name: text, age: number, matric: number )

The next slide has most of an application which will print the student names



## The Student Application

**Key Slide**

```
public class Student
{   String name;                          // one variable for each field
    int age;                              //          used in the application
    int matric;
    public Student( ResultSet RS )
      {   name = RS.getString(1);         // get the first column as a string
          age = RS.getInt(2);             // get second column as an integer
          matric = RS.getInt( “matric” ); // can use column name instead
          System.out.println( name );
      }
}

public class MainClass
{   public void main( String[] args )     // the next 2 lines are Access specific
      {   Connection con = ...  // ugly DBMS specific code to make the connection
          Statement stmt = con.createStatement();
          ResultSet RS = stmt.executeQuery( "select * from Student" );
          while ( RS.next() )   // The next method makes RS point to the next record and
          { Student dummy = new Student( RS ); }          // returns false at the end
      }
}
```



## Notes

The program has implemented a class *Student* which matches the table *Student*

- This will hold variables for as many columns as are required
- Each of these is retrieved using *getInt*, *getString*, etc.

Database access is managed through a **connection object**

- This is created using *getConnection*

**Statements handles** are then created in the context of a connection, using *createStatement*

- SQL is executed by passing a string holding the query to the statement handle
- The query returns a result set which is a list of records
- This list is retrieved one at a time by ***while*** *RS.next()*
- *getInt*, *getString* etc. also work on RS, getting the field of the current row from a numbered column



## Java Database Applications

An application written in Java is one good way of controlling a database

- You can write a nice GUI
- It also has all of the other packages which can be useful
- JDBC provides a solid way of accessing the database

It is also the basis of two of the main ways of writing web sites which use databases:

- Servlets
- JSP

## Databases and the Web
**(See also Lecture 10)**

Web sites are increasingly **dynamically** generated

- Instead of consisting of a number of pure HTML pages with client-side scripts (**static** pages), the web site contains programs which generate HTML when they are requested

The generating programs live on the server and:

- expect some parameterised input along with the request (usually generated by forms)
- produce a different HTML page depending on the parameter values
- usually, use the parameters to query a data source to get part of the content of the returned HTML page
- most often this data source will be a database and SQL will be used to manage the query

The programs are either:

- HTML extended with server side scripts (JSP); or
- written in a programming language (C, PERL, Java) and produce HTML as output (Servlets)



## A Typical Servlet

Access to the Request and Response Messages

```
public class lecturerSvlt extends HTTPServlet
{   public void doPost(HttpServletRequest req, HttpServletResponse res)

{   // First get an output stream to send stuff to the client
    PrintWriter out = res.getWriter();
    out.println("<html><head>……</head>\n<body> ");    // start of HTML
    String login = req.getParameter( "login");
    String sql = "select code, title, name from Course, Lecturer
                                  where login = '" + login + "'";
    ResultSet data = stmt.executeQuery( sql );
    out.println("<h1>The Courses Given by " + data.getString("name") +
            " </h1> <ul> " );              // get the value of name for this record
    ... while loop through the data set once for each course
            out.println("<li><a href="  + data.getString("code") + ".html"
                                          + data. getString("title") + "</a> </li>");
    out.println("</ul></body></html> ");                    // end of HTML
}
```

Response is just and output stream

Get the login from a form

Note name must by login

Database query as a string

Data extracted from the record



## JSP

```
<% page language="java" import="java.sql.*" %>
<% Connection DBcon = DriverMgr.getConnection( URL );
    Statement stmt = Dbcon.createStatement();
    String login = request.getParameter("login");
    String sql = "select code,title, name from Course, Lecturer
                                  where login = '" +  login +"'"
    ResultSet data = stmt.executeQuery( sql );  %>

<html><head>……</head>
<body>
<h1>The Courses Given by <%=data.getString("name")%> </h1>
<ul>
<% while data.next() {%>
     <li><a href=" <%= data. getString("code")%> .html">
          <%= data. getString("title")%> </a></li>
    <% } %> // end of while loop
</ul></body></html>
```



## A Servlet for Inserting a Student into the Database

```
public class addStudent extends HTTPServlet
{   public void doPost(HttpServletRequest req, HttpServletResponse res)

{   PrintWriter out = res.getWriter();                  // get output stream for HTML
    out.println("<html><head>……</head>\n<body> ");    // start of HTML
    String name = req.getParameter( "name");           // age and matric similar
    String sql = "insert into Student values( " +      // Build up insert statement
          name + ", " + age + ", " + matric +")";      // from form variables

    ResultSet nRecords = executeUpdate(sql);     // returns number of records inserted
    if (nrecords == 0)
          out.println("<p>Sorry – insertion failed </p> ");  // failure if nothing inserted
    else
          out.println("<p>Record inserted</p>");            // success
    out.println("</ul></body></html> ");                    // end of HTML
}
```

## PHP Connectivity with MySQL

variables all start $

```
$Advisor = $POST['Advisor'];
$db = mysql_connect("storo", "dymockvj", "4116")
                        or die ("Could not connect");
mysql_select_db("dymockvj") or die ("Cannot select db");

$query = "SELECT * FROM Student
    WHERE advisor = " . $Advisor . " ORDER BY Year, Name";
$result = mysql_query($query, $db) or die ("Error with query");

while ($row = mysql_fetch_array($result))
{   echo $row["name"]."  ".$row["matric"];
}
```

Create a MySQL Connection

Using database, dymockvj

get the student records for this advisor

. means string concatenation

loop once for each record

output the student name and matric



## PHP and MySQL Notes

This is a popular combination for building dynamic web sites

- It is cheap and efficient

The previous slide contained most of a PHP page:

- It starts by retrieving the form variable, *Advisor*
- It then opens a database connection
  - PHP has a set of methods for each major DBMS
- Again the query is a string
- $result is similar to a result set
- It uses a while loop again
  - $row holds a record (as an array indexed by column name)



## Other Tasks of the Server Side Programs

As well as accessing the database and building up the HTML, the program must:

- deal with **security**
- ensure processes are as **efficient** as possible - example
  - making connections is time consuming
  - so making one up every time a user enters a site would be slow
  - therefore, the site builds up a collection of connections when it is started up
  - and allocates a free one when the user logs in
  - this is called connection pooling
- deal with **sessions**
  - i.e. ensure that each min-program is connected to the whole by using the same data
- manage **constraint violations**
- etc.



## Security Example – SQL Injection

- SQL Injection is a security attack which exploits poorly written server programs
- Example, a page contains a request for an age and then returns data about people of that age
  - Inside the server program is the query:

    **select** * **from** Person **where** age = req.getParameter( "age")
  - i.e. get what's entered in the agetextbox and use it
  - but if someone enters this in the box:

    23; delete from Person

    then the table will be destroyed