Database applications

Introduction

Relational databases are specialist components of larger systems. They rarely stand on their own and are much more often incorporated into applications. The overall application benefits from the characteristics of relational databases:

- Support for multiple concurrent users
- Fine-grained access rights model
- Secure and reliable backup facilities
- Portability across multiple platforms (DBMS and operating systems)

There are different ways that databases can be used to provide specialist data handling capabilities. At one end of the scale are traditional programs built with java or C# for example where connections and operations are built into the code and compiled. At the other end of the scale are loosely-coupled applications in which an end user may use tools such as Excel to access relational data.

Objects and relations

The current standard for application development is the object-oriented model in which data and processing are tightly connected. A software application for use in a bank, for example, might represent a customer as a software object with certain attributes and also certain behaviours. When a bank teller is responding to a customer the application represents that person as a software object which includes the customer name, account number, current balance, latest statement details and so on. If the customer wishes to make a deposit, that could be represented as a behaviour of the the *customer* object. Other behaviours would represent other actions such as *make withdrawal* and *request overdraft*.

Representing things as software objects makes the application structure easier to understand. Because the behaviour of the software object is defined along with its attributes, application development is simplified because object definitions can be re-used without any need to re-implement the same behaviours in the new application.

Both objects and relations are used to represent things in the problem domain. However, there are some major differences in the way they do that. Firstly, objects can be complex. In the example of the bank application, the customer object includes several recent transactions as part of the statement data. This is a one to many relationship which would have to be disaggregated in a relational database with the personal details stored in one table and the statement records in another.

Another difference is that the object model allows for relationships between object types (or *classes* to use the appropriate term) which have no direct counterpart in a relational structure. We have already seen one of these in week 4 which offered three different ways of representing the specialisation/generalisation relationship. The table below summarises some of the other main differences.

Objects	Relations
Objects comprise data and procedures	Relations define data and relationships
Object model elaborated using generalisation, inheritance etc.	ER model elaborated using normalisation
Application is made up of objects	Database is separate from the application
Objects "die" when application terminates	Relations comprise persistent storage of data
Attributes can be complex types or other classes	Attributes comprise elementary data types

Persistence

A major issue for object-oriented application programs is that although some objects are transient and can be discarded when the application terminates, other objects must persist from one use of the application to the next. In the bank example, it is clearly important that the customer details are retained even if the system is switched off. Pairing a database with the object-oriented application can provide a solution to the persistence problem, but only if the differences between the two models can be resolved.

Relational databases are not the only way that data can be carried over from one invocation of an application to the next. Ordinary files could also be used, but there are a number of problems with this when large amounts of data are involved, and the advantages of database systems were covered in week 2. For small amounts of data, however, they can be appropriate.

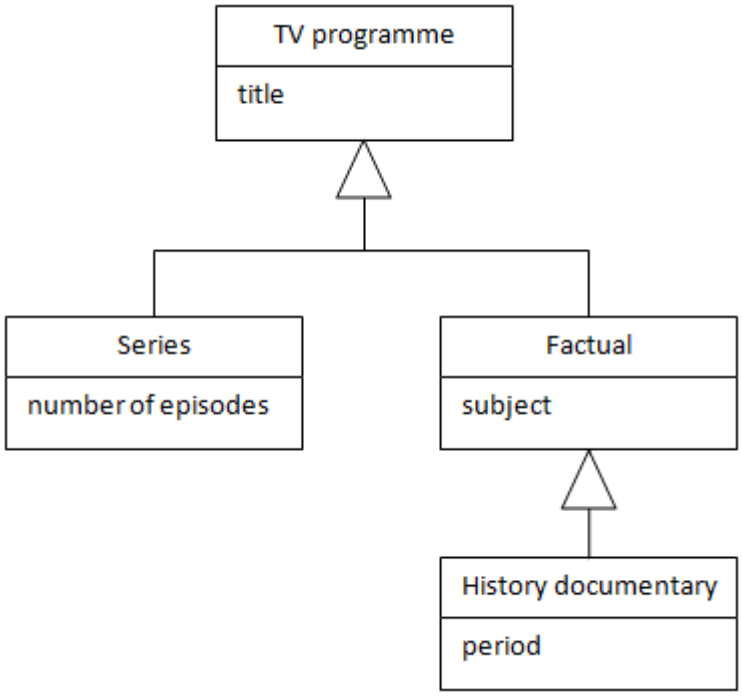
A further option is to use a so-called object database which is capable of storing complex structures. A number of such systems exist, and details of several can be found at http://www.odbms.org/ along with a range of other related materials. However, the application programmer may be required to use an existing relational database, or may prefer a relational database on performance grounds.

The choice of a persistence architecture is therefore part of the overall design of the application.

Some relational database platforms such as Oracle provide special features for handling object-like structures in a transparent way. Data in these object relational systems is still stored in a relational schema, however, and behaviours must be implemented using the native procedural extensions to SQL. This can limit the portability of objects from one system to another.

UML class diagrams

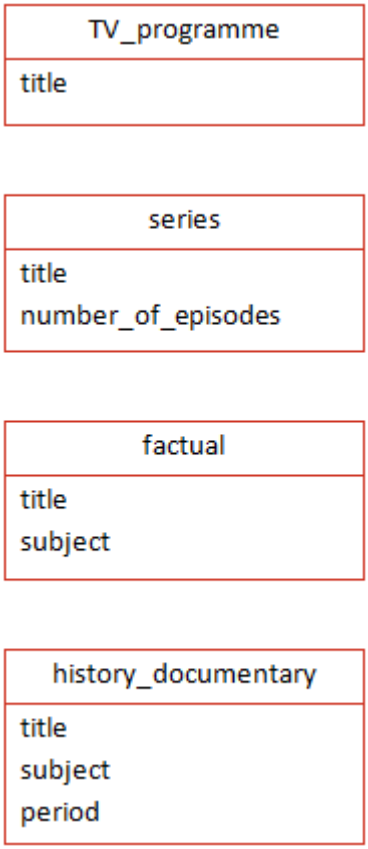
UML class diagrams extend the basic entity-relationship to include specialisation/generalisation and inheritance. This is done as shown below with a large triangular arrowhead pointing towards the superclass and a branching relationship line leading to the subclasses.



TV programme
title
type
subject
number_of_episodes
period

Because relational databases cannot handle this type of relationship directly, equivalent structures have to be used, and there are different ways that this can be done. The first option is to use a single table for all of the classes in a hierarchy, typically including a *type* column to differentiate them as shown on the right. The main disadvantage of this solution is that any record that corresponds to an object high up in the hierarchy will contain many null values. For example, a feature film would have a title, but none of the other columns shown here would have a value.

The second solution is to represent each member of the class hierarchy by its own table as shown below. The main disadvantage here is that similar entities (ie TV programmes) are held in different tables. Thus the generalisation is lost, and any general query - on title, for example - would be very complicated to perform. A possible solution to this lies in creating views which reproduce the different hierarchical levels.




```
CREATE VIEW all_programmes AS
SELECT title, null, null, null
FROM   TV_programme
UNION
SELECT title, number_of_episodes,
       null, null
FROM   series
UNION
SELECT title, null, subject, null
FROM   factual
UNION
SELECT title, null, subject,
period
FROM   history_documentary
```

Object mapping

The structure of a relational database capable of holding the data for a set of persistent objects can be derived from the structure of the objects themselves. Using the same example as we did on week 6 when we looked at normalisation, a project could be represented as shown below. The consultants who work on the project are represented as a collection of nested objects.

A set of mapping rules can be used to transform the object structure into the equivalent relational structure:

- Classes with simple data structure become tables
- Object IDs become primary keys
- Where classes contain another class as an attribute create a table for the embedded class
- For collections create two tables, one for the objects in the collection, the other to hold object IDs of the containing objects and the contained objects - this is equivalent to creating a link table for a many-to-many relationship
- One-to-many associations can be treated like collections
- Many-to-many associations become two separate tables for the objects and a table to hold pairs of object IDs
- One-to-one associations are implemented as foreign-key attributes - each class gains an extra attribute for the object ID of the other
- Inheritance (generalisation/specialisation) can be represented in three different ways, each of which has its disadvantages:
  - only implement the superclass as a table including all subclass attributes
  - only implement the subclasses as tables, duplicating superclass attributes in each
  - implement superclass and subclasses as tables with shared primary keys

In the project case, we will end up with the same structure as we did using normalisation.

To simplify the process of addressing relational structures from an object-oriented program, several frameworks can be used including JDO, Hibernate and LINQ

Basic operations

Summarising what you already know, there are four basic operations that can be performed on databases. They can be remembered using the acronym CRUD which stands for Create, Read, Update and Delete. In SQL these operations correspond to the three type of DML command along with the SELECT statement. They are so fundamental that they are also built into other standard technologies such as the http protocol which defines the operations shown in the table below.

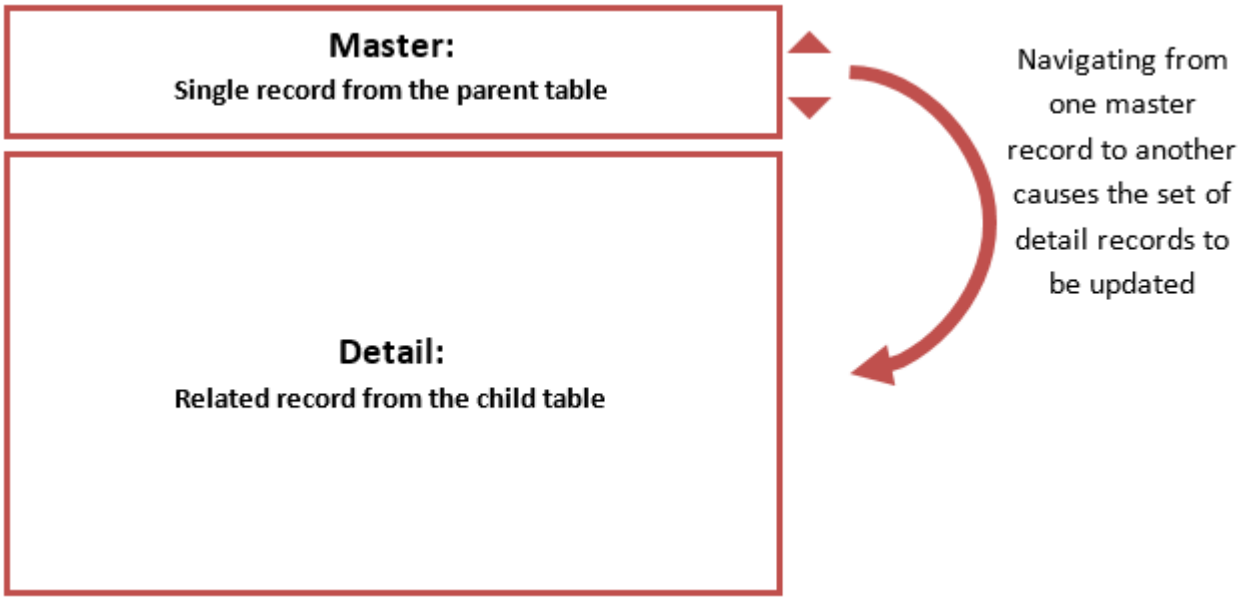
Operation	SQL	HTTP
Create	INSERT	POST
Read	SELECT	GET
Update	UPDATE	PUT
Delete	DELETE	DELETE

At the very least, an appilcation must support the CRUD operations. For example, the HR schema contains seven tables:

- COUNTRIES
- DEPARTMENTS
- EMPLOYEES
- JOBS
- JOB_HISTORY
- LOCATIONS
- REGIONS

A very simple interface would provide a way to view each table independently and would allow the user to change the data by inserting, updating and deleting records as required. Many application frameworks now provide this functionality by default and this is essentially what is provided by SQL Developer; however, a basic CRUD interface is very limited. For example, some operations should be performed automatically by the application rather than by the user. Consider the records in the **JOB_HISTORY** table: a new record should be automatically inserted when an employee record is updated with a new job title. Such additional functionality typically needs to be built into the application, although simple operations can be provided by the database y using triggers.

A further issue is that a CRUD interface does not present the data in a particularly useful form. Consider the tables **DEPARTMENTS** and **EMPLOYEES**: the data is clearly related, and in certain circumstances it would be useful to present the list of employees alongside the details of the department they work for. This would simply be a view based on the one-to-many relationship between departments and employees. To accommodate this natural relationship, a commonly used design pattern is the *master-detail* page which presents one record from the parent table along with the related records from the child table. If the user navigates to the next row in the parent table, the detail records from the child table will automatically synchronise to show the related records:



The master-detail pattern is one of several recommended by Oracle for database-driven applications: http://www.oracle.com/webfolder/ux/middleware/alta/patterns/index.html

Embedded SQL

Whereas there are many languages available for writing complete applications, SQL is the only standard language for handling relational structures. Most languages therefore provide ways of embedding SQL commands into structured programs. A particularly good example of this is provided by PHP which has dedicated sets of commands for linking to MySQL databases.

PHP (Pre-Hypertext Processing) is a scripting language that can be used to produce dynamic Web content. If you are not familiar with PHP the examples below should still make sense. The important aspect is that PHP is a procedural language with the full range of control statements and data structures. Likewise, the examples refer to MySQL databases because of their tight integration with PHP. However, PHP can also be used with other databases including Oracle, and if you are interested in following this topic up, you can follow the resources link on the left.

Connecting to a database

Connecting to a database from a programming language is just the same as connecting with a tool like SQL Developer. You need to tell the software where the database is and what account to use. PHP provides a simple command to do this for MySQL databases. Note that PHP variables always start with a dollar sign and that a double slash indicates a comment:

```
$db_host = "server_1";
$username = "hr";
$password = "hr";

// Connect to MySQL
mysql_connect($db_host, $username, $password);
```

Constructing and sending a query

A text string representing a query can be defined in a single assignment, or it can be built up gradually from literal text and variable values. The example below illustrates the second case. When the query text is complete, it is executed and an array of results is created. The results array is two-dimensional since each row is an array element in the vertical dimension, but because it is made up of columns, it is itself an array in the horizontal dimension.

In the example below, note that the string concatenation operator is a dot and that the operator **.=** appends text to the named variable. Also note that because the query string is built up over several concatenation operations, it is important that each part ends in a space character.

```
$query  = "SELECT employee_id, first_name, last_name ";
$query .= "FROM employees ";
$query .= "WHERE department_id = " . $deptId . " ";
$query .= "ORDER BY last_name";
$result = mysql_query($query);
```

If you are using text values in the *where* clause of the query, you will also need to remember to handle any problem characters just as you would in any other situation. For example, single quote characters would need to be doubled to prevent them from being interpreted as string delimiters.

Processing query results

Many programming languages have the concept of a *cursor* which is a local array variable containing the values for the current row in a result set. As the results are processed the cursor is moved to each row in turn. PHP does not use this term explicitly, but the processing follows the same sequence and is controlled using standard language constructs. The example below shows the use of a *while* loop to iterate through each result row in turn. Each iteration of the loop generates a new row in an HTML table, and therefore illustrates how a dynamic Web page might be produced. Note that the PHP *echo*command sends the subsequent values to the standard output channel.

```
echo "<table>";
while ($loc = mysql_fetch_assoc($result)) {
   echo "<tr>";
   echo "<td>" . $loc{employee_id} . "</td>";
   echo "<td>" . $loc{first_name} . "</td>";
   echo "<td>" . $loc{last_name} . "</td>";
   echo "</tr>";
}
echo "</table>";
```

The standard term *fetch* is used to describe the assignment of a set of database values to a local variable.

Null values

Missing values in a database are shown as null values in a set of results and this can cause problems when trying to process those results in an application program. For example, a null value in a number column may produce an error if the application program attempts to compare the value with another number. To avoid such problems, it may be advisable to ensure that potential null values are handled by the query so that appropriate defaults are used instead of nulls in the results. The SQL function COALESCE (available in both Oracle and MySQL) can be used to do this.

Error handling

You will already be used to seeing database error message that tell you when an operation is not successful for some reason. Because the same errors can occur when using SQL inside a program, it is important to have a way of passing those messages back to the user. When a PHP operation fails, the construct or die(<message>) can be used to generate error messages, and the example below shows how the most recent SQL error can be embedded into the message string. This same method can be used for any embedded SQL operation.

```
$result = mysql_query($query) or
   die("Unable to retrieve employee information: " . mysql_error());
```

DML

Although the PHP function is called `mysql_query( )`, it can actually be used to execute any arbitrary SQL statement including DML. The example below shows how a record can be updated. It includes error checking, and an additional message to the user showing how many rows were updated using the function `mysql_affected_rows( )`.

```
$query  = "UPDATE employees SET commission_pct = " . $newCommission . " ";
$query .= "WHERE employee_id = " . $empId;
$result = mysql_query($query) or
          die("Unable to update commission: " . mysql_error());

echo mysql_affected_rows() . " row(s) updated";
```

Closing the database connection

Typically a database allows only a limited number of simultaneous connections. Once the program has completed its database operations it is important to close the connection in order to avoid blocking other users. In PHP this is accomplished as follows:

```
mysql_close();
```

Interoperability

ODBC

Standard desktop applications such as Microsoft Excel are very popular with non-technical users because they offer very powerful data manipulation features combined with an accessible and intuitive user interface. With so much functionality readily available, it makes little sense to try to duplicate it in context-specific database applications. It is much more efficient to provide a connection which allows data to be queries from the database and then processed further an application such as a spreadsheet. This approach also takes advantage of the existing skills of end users without the need for any new application-specific training.

Open Database Connectivity (ODBC) is a standard interface for programs such as desktop applications to access databases. ODBC provides a *driver* as a translation layer between an application and the DBMS. The application uses standard ODBC functions, and the driver passes the appropriate query to the DBMS. The existence of the ODBC standard means that all vendors of desktop software can provide support for the standard, and so offer a high degree of connectivity without having to develop proprietary interfaces.

Microsoft Excel comes equipped with ODBC drivers for major database vendors such as Oracle, and drivers are readily available for download on the Internet for any more obscure platforms. This week's practical exercise takes you through the process of setting up and using a database connection in Excel.

XML

Another current technology that provides a simple and flexible mechanism for transporting relational data is eXtensible Markup Language (XML). XML can be used to structure any type of data in text form by using markup tags in the same way that HTML tags identify the elements of a Web page.

Because XML is structured text, it can be produced and consumed very easily by any software system. It also has the added advantage that its structure can be validated so that ill-formed data can be rejected. By defining a set of XML tags specific to a particular domain, structures of arbitrary complexity can be built which correspond directly to relational structures.

For the above reasons, XML makes an excellent communication medium. However, as a storage medium it suffers from many of the same problems as standard files storage, and cannot compete with relational databases for efficiency and flexibility