CHAPTER 10

Projects

This chapter presents some project suggestions that can be developed by the reader to get a better understanding of the material presented in this book. Each project will typically go through the following phases:

- *Phase 0.* Design the database (using entity-relationship diagrams).
- *Phase 1.* Create the tables, including constraints such as primary keys, foreign keys, check constraints, and `not null` constraints.
- *Phase 2.* Create triggers and active elements to maintain the integrity of the database and to perform appropriate actions on database updates.
- *Phase 3.* Populate the database using SQL `insert` statements or by writing programs in Java or Pro*C/C++.
- *Phase 4.* Write application programs in Java and/or Pro*C/C++ and/or PL/SQL.
- *Phase 5.* Document the project.

The application programs will have to implement some form of user interface. The simplest user interface is terminal based and involves menus and submenus. However, there are several tools and languages that support the development of fancier user interfaces, such as the `curses` package in Unix, the Java AWT toolkit, and the X-Windows libraries. The reader may choose to develop the user interfaces using these tools and languages.

The rest of this chapter discusses several application areas for which a database application can be developed. For each application area, a brief description of the

application is followed by a description of the relational tables and a sketch of the application program requirements in the form of menus and submenus. The reader is encouraged to use the ideas presented as a starting point for the definition of the problem and to modify it according to his or her understanding of the problem.

10.1 Airline Flight Information System

The airline flight information database consists of information relating to the operations of the airline industry. One possible design of the database results in the following relational tables:

```
AIRPORT(AIRPORT_CODE,name,city,state)
FLIGHT(NUMBER,airline,weekdays)
FLIGHT_LEG(FLIGHT_NUMBER,LEG_NUMBER,
  departure_airport_code,scheduled_departure_time,
  arrival_airport_code,scheduled_arrival_time)
LEG_INSTANCE(FLIGHT_NUMBER,LEG_NUMBER,LEG_DATE,
  number_of_available_seats,airplane_id,
  departure_airport_code,departure_time,
  arrival_airport_code,arrival_time)
FARES(FLIGHT_NUMBER,FARE_CODE,amount,restrictions)
AIRPLANE_TYPE(TYPE_NAME,max_seats,company)
CAN_LAND(AIRPLANE_TYPE_NAME,AIRPORT_CODE)
AIRPLANE(AIRPLANE_ID,total_number_of_seats,
  airplane_type)
SEAT_RESERVATION(FLIGHT_NUMBER,LEG_NUMBER,DATE,
  SEAT_NUMBER,customer_name,customer_phone)
```

Note: The primary key columns are shown in uppercase.

Each `FLIGHT` is identified by a `FLIGHT_NUMBER` and consists of one or more `FLIGHT_LEG`s with `LEG_NUMBER`s 1, 2, 3, and so on. Each leg has many `LEG_INSTANCE`s, one for each date on which the flight flies. `FARES` are kept for each flight, and `SEAT_RESERVATION`s are kept for each `LEG_INSTANCE`. Information about airports is kept in the `AIRPORT` table, and information about individual airplanes is kept in the `AIRPLANE` table. `AIRPLANE_TYPE` records the information about the airplane type, and the `CAN_LAND` table keeps information about which airplane type can land in which airport.

The following is an outline for the application program to be developed for the airline reservation system. It includes a main menu of options and several submenus.


```
MAIN MENU
(1) Customer functions
(2) Reporting functions
(3) Administrative functions
(4) Quit

CUSTOMER FUNCTIONS MENU
(1) Make a reservation
(2) Cancel a reservation
(3) Confirm a reservation
(4) Print trip itinerary
(5) Locate fare
(6) Quit

REPORTING FUNCTIONS MENU
(1) Print flight roster
(2) Print flight schedule (based on several criteria
    such as airline, departure city, arrival city)
(3) Print flight performance report
    (Given airline, print on-time and delayed flights)
(4) Quit

ADMINISTRATIVE FUNCTIONS MENU
(1) Add/drop flight
(2) Add airport
(3) Update fares
(4) Create leg instance
(5) Update leg instance (departure/arrival times)
(6) Quit
```

10.2 Library Database Application

Consider the operations of a public library system in a city. The library has many patrons who borrow books from one of its many branches. Each branch of the library holds a number of copies of a particular book. Books that are not returned on time are fined at a rate of 25 cents for each day after the due date. One possible design of a database for the public library system results in the following relational tables:

```
BOOKS(BOOK_ID,title,publisher_name)
BOOK_AUTHORS(BOOK_ID,AUTHOR_NAME)
PUBLISHERS(NAME,address,phone)
```

```
BOOK_COPIES(BOOK_ID,BRANCH_ID,no_of_copies)
BRANCHES(BRANCH_ID,branch_name,address)
BOOK_LOANS(BOOK_ID,BRANCH_ID,CARD_NO,
  date_out,date_due,date_returned)
BORROWERS(CARD_NO,name,address,phone,unpaid_dues)
```

Note: The primary key columns are shown in uppercase.

The BOOKS table records information about all the books that are available in the library system. Each book has a unique BOOK_ID. Information about the authors of books is kept in the table BOOK_AUTHORS, and information about the publishers is kept in the table PUBLISHERS. The number of copies of each book in a particular library branch is recorded in the BOOK_COPIES table. The branch information is kept in the BRANCHES table. Information about the patrons of the library is kept in the BORROWERS table, and the loaned books are recorded in the BOOK_LOANS table.

The following is an outline for an application program to be developed for a library database:

```
MAIN MENU
(1) Patron functions (ask for card number,
    then show submenu)
(2) Administrative functions
(3) Quit

PATRON FUNCTIONS MENU
(1) Book checkout
(2) Book return
(3) Pay fine
(4) Print loaned books list
(5) Quit

ADMINISTRATIVE FUNCTIONS MENU
(1) Add a book
(2) Update book holdings
(3) Search book
(4) New patron
(5) Print branch information
(6) Print top 10 frequently checked-out books
(7) Quit
```


10.3 University Student Database

Consider the data that are usually maintained by a typical university concerning students, courses, and enrollments. Students are admitted to the university, and they pursue a degree program in a particular department. The university catalog consists of courses that are offered every term. Students choose courses to take and enroll in them during registration. Instructors are assigned courses to teach, and they in turn assign grades. A possible database design results in the following relational tables:

```
COURSES(CNO,ctitle,hours,dept_id)
DEPARTMENTS(DEPT_ID,dept_name,college)
INSTRUCTORS(LAST_NAME,FIRST_NAME,
  dept_id,office,phone,email)
SECTIONS(TERM,LINENO,cno,instr_lname,instr_fname,
  room,days,start_time,end_time,capacity)
STUDENTS(SID,last_name,first_name,class,phone,
  street,city,state,zip,degree,dept_id,hours,gpa)
ENROLLMENT(SID,TERM,LINENO,grade)
```

Note: The primary key columns are shown in uppercase.

The COURSES table maintains the list of courses in the university catalog. Information about departments, instructors, and students is maintained in the DEPARTMENTS, INSTRUCTORS, and STUDENTS tables, respectively. Notice that some of the columns in the STUDENTS table are computed columns (gpa, hours)—i.e., their values are determined by other values in other tables. The SECTIONS table maintains information about the schedule of classes for each term. These are the sections of the various courses that are offered each term. The ENROLLMENT table keeps information about the enrollment of students in sections.

The following is an outline of a possible application program for the student database:

```
MAIN MENU
(1) Student functions
(2) Administrative functions
(3) Reporting functions
(4) Quit

STUDENT FUNCTIONS MENU
(1) Register for courses
(2) Add/drop a course
(3) Request transcript
(4) Pay fees (get a fee report)
(5) Quit
```



```
ADMINISTRATIVE FUNCTIONS MENU
(1) Create a new course/drop course
(2) Prepare term schedule (add sections)
(3) Add/drop instructors
(4) Alter term schedule (add/drop/update sections)
(5) Add/drop students
(6) Quit

REPORTING FUNCTIONS MENU
(1) Print schedule of classes (for a term)
(2) Print the catalog
(3) Print the honors list of students for a department
(4) Quit
```

10.4 Video Chain Database

Consider the operations of a video rental and sales chain. Such a company purchases videos from vendors and stocks them in one of many stores. Each store has several employees who rent or sell these movies to members. Members are required to return the rented movies by the due date; otherwise, a fine is imposed. Commissions are awarded to employees based on their sales volume. One possible design of a database for this application results in the following relational tables:

```
STORE(STORE_NUM,address)
EMPLOYEES(EID,name,store_num,commission_rate)
MOVIES(VID,STOCK_NUM,title,cost,category,rent_price,
       sale_price,purchase_date,vendor_name,qoh)
MEMBERS(MID,lname,fname,address,bonus_points)
RENTALS(RENTAL_TRANSACTION_NUMBER,mid,stock_num,
        vid,eid,date_out,frequency,date_in)
SALES(SALE_TRANSACTION_NUMBER,mid,stock_num,vid,eid,
      sale_date)
VENDORS(VENDOR_NAME,address,phone)
```

Note: The primary key columns are shown in uppercase.

The STORE table records the store numbers and addresses of the individual stores. The EMPLOYEES table records information about the employees and the store with which they are associated. The MOVIES table contains information about all the videos in the company. The STOCK_NUM column may indicate the store to which each videocassette belongs. Information about the members is kept in the MEMBERS

table. Members are given some bonus points each time they rent a movie. The accumulated points are recorded in this table. Members are eligible for a free rental after accumulation of a certain number of bonus points. The RENTALS and SALES tables record each transaction made. For the rental of movies, the date checked out, the duration of the checkout (how many days?), and the date returned are recorded. The VENDORS table records information about vendors from whom the videos are purchased.

A possible application program for the video company is outlined in the following menu/submenus:

```
MAIN MENU
(1) Member functions
(2) Reporting functions
(3) Administrative functions
(4) Quit

MEMBER FUNCTIONS MENU
(1) Video checkout
(2) New member signup
(3) List of outstanding videos
(4) Membership cancellation
(5) Video purchase
(6) Quit

ADMINISTRATIVE FUNCTIONS MENU
(1) Video return
(2) Add/delete employee
(3) Process new shipment of videos
(4) Open new store
(5) Quit

REPORTING FUNCTIONS MENU
(1) Print catalog (arranged by categories)
(2) Print due list of videos
(3) Print employee commission report
(4) Print rental summary
    (sorted based on frequency of rental)
(5) Quit
```

10.5 Banking Database

Consider the operations of a typical banking enterprise. A bank normally has many branches, and customers can open accounts at any of these branches. It is normal for more than one customer to have the same account and for one customer to have multiple accounts. The bank offers various types of services—from savings and checking accounts to loans. A possible design for the banking enterprise results in the following relational tables:

```
BRANCHES(BRANCH_NUM,branch_name,address)
CUSTOMERS(CUSTOMER_NUM,name,address,phone)
CHECKING_ACCOUNTS(ACCOUNT_NUM,branch_num,date_opened,
  balance,overdraft_amount,check_limit)
SAVINGS_ACCOUNTS(ACCOUNT_NUM,branch_num,date_opened,
  balance,interest_rate)
LOAN_ACCOUNTS(ACCOUNT_NUM,branch_num,date_opened,
  loan_type,interest_rate)
HAS_ACCOUNT(CUSTOMER_NUM,ACCOUNT_TYPE,ACCOUNT_NUM)
TRANSACTIONS(ACCOUNT_TYPE,ACCOUNT_NUM,TRANS_DATE,
  TRANS_AMT,TRANS_TYPE,trans_comments)
```

Note: The primary key columns are shown in uppercase.

The BRANCHES table keeps information about all the branches of the bank, and the CUSTOMERS table records information about all the customers of the bank. There are three tables for the three different types of accounts. Each has an ACCOUNT_NUM column, which may or may not be unique across the accounts. Hence, the table HAS_ACCOUNT, which keeps information about which customers own which accounts has the column ACCOUNT_TYPE to indicate what types of account the customers own. The TRANSACTIONS table records all the transactions that occur within the accounts.

The following is a sketch of an application for the banking enterprise:

```
      MAIN MENU
(1) Customer functions
(2) Administrative functions
(3) Reporting functions
(4) Quit

      CUSTOMER FUNCTIONS MENU
(1) Deposit
(2) Withdraw
(3) Transfer
```



```
(4) Loan payment
(5) Quit

      ADMINISTRATIVE FUNCTIONS MENU
(1) Process checks
    (assume a file containing checks received
     by the bank)
(2) Add/drop customer
(3) Open/close account
(4) Quit

      REPORTING FUNCTIONS MENU
(1) Print monthly statement
(2) Print loan payment schedule
(3) Print yearly tax statement (interest earned)
    (to be mailed out for each customer)
(4) Quit
```

10.6 BibTeX Database

Consider the TEX word processing system and the way it handles bibliography information. The bibliographic entries are created in a text file in a particular format and are consulted by the BibTEX program when processing the entries referenced in a document.[1] The problem is to keep the collection of bibliographic entries in an Oracle database and to allow manipulation of these entries via an application program.

The bibliographic entries are classified into various categories: article, book, inbook, proceedings, inproceedings, techreport, manual, conference, and so on. An entry in each of these categories has some required fields and some optional fields. Each entry is identified by a unique citekey.

One possible design of a relational database results in the following tables:

```
MASTER_ENTRIES(CITE_KEY,entry_type)
ARTICLE(CITE_KEY,author,title,journal,volume,number,
  pages,month,year,note)
BOOK(CITE_KEY,author,editor,title,publisher,address,
  volume,edition,series,month,year,note)
```

1. See Leslie Lamport, *LaTeX User's Guide and Reference Manual* (Reading, MA: Addison-Wesley, 1986), for details on this format and other specifics regarding the BibTEX program.


```
PROCEEDINGS(CITE_KEY,editor,title,publisher,
  organization,address,month,year,note)
INBOOK(CITE_KEY,author,editor,title,publisher,address,
  volume,edition,series,chapter,pages,month,year,note)
INPROCEEDINGS(CITE_KEY,author,editor,title,booktitle,
  publisher,organization,address,pages,month,year,note)
.
.
.
REQUIRED_FIELDS(ENTRY_TYPE,FIELD)
```

Note: The primary key columns are shown in uppercase.

The MASTER_ENTRIES table contains the citekey and the type of entry (article, book, etc.) for all the bibliographic entries. There is one table for each entry type that maintains all the entries under that particular category. For example, the table ARTICLE contains all the bibliographic entries that are articles. The REQUIRED_FIELDS table records information about the required fields for each type of entry. This information will have to be consulted when creating a new bibliographic entry.

An application program that manipulates the preceding database is sketched out here:

```
      MAIN MENU
(1) Update functions
(2) Search functions
(3) Reporting/utility functions
(4) Quit

      UPDATE FUNCTIONS MENU
(1) Add an entry
(2) Modify an entry
(3) Delete an entry
(4) Quit

      SEARCH FUNCTIONS MENU
(1) Search based on author
(2) Search based on keyword in title
(3) Search based on multiple search criteria
(4) Quit

      REPORTING/UTILITY FUNCTIONS MENU
(1) Print summary reports
(2) Read .bib files and load database
(3) Write all entries to .bib file
(4) Write selected entries to .bib file
(5) Quit
```


10.7 Music Store Database

Consider the operations of a company that sells prerecorded compact discs and related items. The company has many outlets in several states. Each outlet has been assigned a number and has its own manager, employees, inventory, sales, and returns. Company-wide product and customer lists are maintained. Based on this information, the following relational tables constitute one possible design of the database:

```
OUTLET(OUTLET_NUMBER,address,city,state,zip,phone)
EMPLOYEE(OUTLET_NUMBER,EMP_NUMBER,emp_name)
PRODUCT(PRODUCT_CODE,artist,title,cost,sale_price)
CUSTOMER(CUSTOMER_ID,customer_name,address,city,
  state,zip,phone)
MANAGER(OUTLET_NUMBER,emp_number)
INVENTORY(OUTLET_NUMBER,PRODUCT_CODE,quantity)
SALES(OUTLET_NUMBER,EMP_NUMBER,CUSTOMER_ID,PRODUCT_CODE,
  SALE_DATE,SALE_TIME,quantity)
RETURNS(OUTLET_NUMBER,PRODUCT_CODE,CUSTOMER_ID,
  RETURN_DATE,RETURN_TIME,quantity,reason,restock)
```

Note: The primary key columns are shown in uppercase.

The following is a sketch of an application program interface using menus and submenus:

```
      MAIN MENU
(1) Sale/return processing
(2) Outlet/employee/customer/product maintenance
(3) Reports
(4) Quit

      SALES/RETURNS MENU
(1) Process a sale
(2) Process a return
(3) View a sale (given date and customer id)
(4) View a return (given date and customer id)
(5) Quit

      MAINTENANCE MENU
(1) Add/modify/drop outlet
(2) Add/modify/drop employee
(3) Add/modify/drop customer
(4) Add/modify/drop product
```

```
(5) Process new shipment of products for an outlet
(6) Process returns
(7) Quit

      REPORTS MENU
(1) Produce yearly sales report for outlet
(2) Produce sales report for employee
(3) Produce the list of top 10 selling items
(4) Quit
```

10.8 Online Auctions Database

Consider the operations of an online auction company that offers members the opportunity to buy and sell computer-related hardware and software items. The seller lists the item for sale. A description of the item, along with the starting price and bid increments, is specified by the seller. Various other information, such as shipping mode and charges, category of the item, when the auction ends, and so on, is also provided at the time of the listing. Buyers make bids for the items they are interested in buying. The person placing the highest bid at the time of the end of the auction is declared the winner and a transaction between the buyer and seller may proceed soon after. Buyers and sellers can leave feedback regarding their experience with a purchase or sale on the system. This feedback is available for every member to view.

Based on this information, the following relational tables constitute one possible design of the database:

```
MEMBERS(USERID,password,name,address,phone,email)
ITEMS(INO,category,title,description,sellerID,quantity,
      startPrice,bidIncrement,lastBidReceived,closeTime)
SHIPPING(INO,SHIPTYPE,SHIPPRICE)
BID(BUYERID,INO,PRICE,QTYWANTED,BIDTIME)
RATING(INO,BUYERID,SELLERID,sComment,bComment,sScale,bScale)
```

Note: The primary key columns are shown in uppercase.

The `MEMBERS` table records information about all the members of the online auction company. These include both buyers and sellers, and the same member can be the seller of one item and the buyer of another item. The `ITEMS` table keeps track of items that are on the auction block. Each item is assigned a unique item number. The seller's user ID is included in this table. For each item on auction, there can be several shipping modes, and the `SHIPPING` table keeps track of this information. The bids placed by members for items are recorded in the `BID` table. The `RATING` table

records the ratings placed by buyers or sellers for a transaction. The rating includes a descriptive comment as well as a numeric value (say, −1 for a negative experience, 0 for a neutral experience, and +1 for a positive experience).

The following is a sketch of an application program interface using menus and submenus:

```
      MAIN MENU
(1) Member registration
(2) Member login
(3) Quit

      MEMBER MENU
(1) Place an item for auction
(2) Bid on an item
(3) Search for items
(4) Place a rating
(5) View rating
(6) Quit
```

10.9 Oracle Data Dictionary Browser

Oracle maintains information about all the database objects in its data dictionary, as briefly described in Section 2.8. The data dictionary includes the names and structures of all tables, constraints, indexes, views, synonyms, sequences, triggers, stored functions, procedures, and packages. Each individual Oracle user has access to the portion of the data dictionary that pertains to his or her schema, via predefined views. Some of these views are

```
user_objects   : description of user's objects
user_tables    : description of user's tables
user_indexes   : description of user's indexes
user_sequences: description of user's sequences
user_synonyms  : description of user's synonyms
user_source    : description of user's stored
                 functions, procedures, and packages
user_triggers  : description of user's triggers
user_views     : description of user's views
```

This project involves providing a convenient and powerful way for users to search and browse their database objects.

The following is a sketch of an application program interface for the data dictionary browser using menus and submenus:

```
      MAIN MENU
(1) Oracle user login
(2) Quit

      ORACLE USER MENU
(1) View tables
(2) View functions/procedures/packages
(3) View synonyms
(4) View sequences
(5) View indexes
(6) View triggers
(7) View views
(8) View objects
(9) Quit
```

To implement this project, the reader must get familiar with Oracle's data dictionary. Within SQL*Plus, issuing the `describe` command on each of the predefined views lists the data dictionary table structure for that view.

10.10 Oracle Data Browser on the Web

Using the PL/SQL Web Toolkit or Java servlets, implement an Oracle Data Browser. The Oracle Data Browser is a Web application that allows a user to examine his or her schema and data contents. The initial login screen should allow the user to enter a user schema name and password. After authenticating the name and password, the user is presented with a three-frame Web page, shown in Figure 10.1.

The top frame is always fixed. The right-hand frame is the work area where results are displayed. The left-hand frame contains the following five menu options:

- *Tables.* This option presents the user with a pull-down list of the tables in his or her schema. Upon choosing one and submitting it, the table schema should be displayed in the right-hand frame, as shown in Figure 10.2.

  The table schema display includes one checkbox per column of the table and a submit button. The user may choose zero or more checkboxes and click the submit button. The contents of the table (only those columns chosen or all columns if none of the checkboxes is chosen) should be displayed in the same frame upon submission.

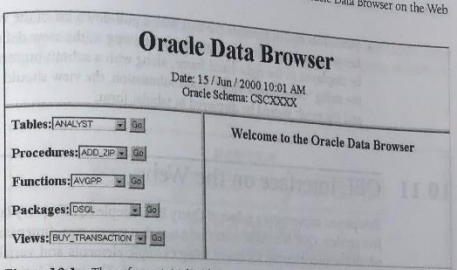

Figure 10.1 Three-frame initial Web page—Oracle Data Browser.

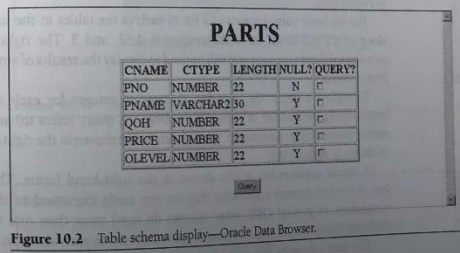

Figure 10.2 Table schema display—Oracle Data Browser.

- *Procedures.* This option presents the user with a pull-down list of the stored procedures in his or her schema. Upon choosing one and submitting it, the stored procedure source code should be displayed in the right-hand frame.
- *Functions.* Similar to Procedures.
- *Packages.* Similar to Procedures, except that only the package specification should be displayed, along with a submit button labeled with the string "Get Package Body." Upon submission, the package body should be displayed in the same frame.

- *Views.* This option presents the user with a pull-down list of the views in his or her schema. Upon choosing one and submitting it, the view definition should be displayed in the right-hand frame, along with a submit button labeled with the string "Get Current View." Upon submission, the view should be calculated and the result should be displayed in tabular form.

10.11 QBE Interface on the Web

This project implements a subset of Query By Example (QBE) query language using Java servlets. QBE is a visual query language that uses a two-dimensional syntax in which the user expresses a query using example elements and variables. You can learn about QBE from any standard database text. The initial login screen should allow the user to enter an Oracle schema name and password. After authenticating the name and password, the user is presented with a two-frame Web page, shown in Figure 10.3.

The left-hand frame consists of a list of each of the tables in the user's schema, along with a pull-down list of the integers 0, 1, 2, and 3. The right-hand frame contains a welcome message and will be used to display the results of any subsequent form submits.

To express a query, the user first chooses an integer for each of the tables (representing the number of copies of the table the query refers to) and clicks the submit button. Upon submission, table skeletons are shown in the right-hand frame, as shown in Figure 10.4.

A special *condition box* is also shown in the right-hand frame. The condition box is used to express conditions that are not easily expressed in the skeletons themselves. In general QBE, some queries do need more than one row in table skeletons; however, for this assignment, only one row in each skeleton is sufficient.

The user expresses the query by entering form elements in the skeletons and condition box. There are four types of elements: `P.`, `P._Variable`, `Constant`, and

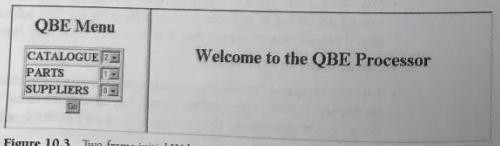

Figure 10.3 Two-frame initial Web page—QBE interface.

`_Variable`. Upon submission, the system should produce the results of the query along with the SQL statement obtained by translating the QBE query, as shown in Figure 10.5.

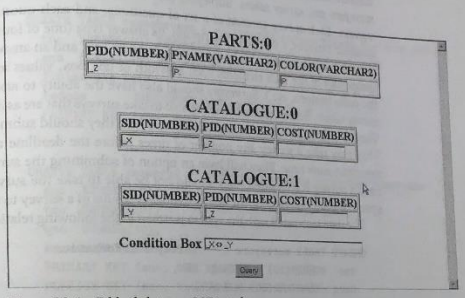

Figure 10.4 Table skeletons—QBE interface.

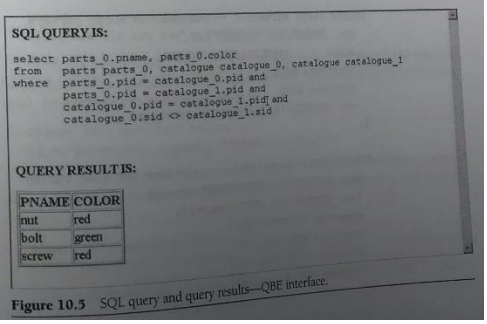

Figure 10.5 SQL query and query results—QBE interface.

10.12 A Web Survey Management System

Using PL/SQL or Java servlets, implement a Web application that manages the creation and deployment of online surveys. The system has two types of users: *surveyors* and *survey takers*. Surveyors should be able to create, edit, and delete surveys. Each survey consists of a set of questions, and each question contains a question number, a question description, its answer type (one of four possible GUI elements: text box, text area, radio button, or select list), and an answer description (properties of the GUI elements such as width of text box, values associated with the radio buttons, etc.). Surveyors should also have the ability to assign surveys to survey takers. Survey takers should be able to take surveys that are assigned to them. They usually will be given a deadline before which they should submit each survey. They may take a survey any number of times before the deadline and save their responses each time. They will have an option of submitting the survey; however, once a survey is submitted, they should not be able to take the survey again. The system should also provide the ability to display results of a survey to the surveyors.

The data associated with the system is stored in the following relational database:

```
CREATE TABLE surveyors ( -- surveyor information
  vno  NUMBER(10)   PRIMARY KEY,
  name  VARCHAR2(30) NOT NULL,
  email  VARCHAR2(50) UNIQUE,
  password VARCHAR2(20) NOT NULL
);

CREATE TABLE surveys ( -- information about surveys
  sno  NUMBER(10) PRIMARY KEY,
  sname      VARCHAR2(100) UNIQUE,
  firstDate DATE,
  lastDate DATE,
  vno  NUMBER(10) REFERENCES surveyors
);

CREATE TABLE questions( -- questions for each survey
  sno  NUMBER(10) REFERENCES surveys,
  qno  NUMBER(10) CHECK(qno>=1),
  question VARCHAR2(150),
  answtype VARCHAR2(15),
  answdesc VARCHAR2(400),
  PRIMARY KEY (sno, qno)
);
```



```
CREATE TABLE takers ( -- survey takers
  tno  NUMBER(10) PRIMARY KEY,
  email  VARCHAR2(50) UNIQUE,
  password VARCHAR2(20)
);

CREATE TABLE takesurveys ( -- records survey takers for each survey
  tno  NUMBER(10) REFERENCES takers,
  sno  NUMBER(10) REFERENCES surveys,
  finishdate DATE,
  isdone      CHAR CHECK(isdone IN ('Y', 'N')),
  PRIMARY KEY (sno, tno)
);

CREATE TABLE answers( -- records responses to survey questions
  sno  NUMBER(10),
  tno  NUMBER(10),
  qno  NUMBER(10),
  answer VARCHAR2(200),
  PRIMARY KEY (sno, tno, qno),
  FOREIGN KEY (sno, tno) REFERENCES takesurveys,
  FOREIGN KEY (sno, qno) REFERENCES questions
);
```

Some examples of rows in the question table are

```
insert into questions values
(1,1,'How old are you?','TextField','2,3');
```

Here, the first question in survey 1 is being described as a `TextField` question (a text box with a maximum of three characters and with the displayed width being two). An example of a text area question is

```
insert into questions values
(1,2,'Enter your comments','TextArea','80,5');
```

Here, the second question in survey 1 is being described as a `TextArea` question (a text area box with five rows and 80 columns). An example of a radio button question is

```
insert into questions values
(1,2,'Your Class','RadioButton','Freshman,Sophomore,Junior,Senior');
```

Here, a `RadioButton` question is being described with four possible values listed in the `answdesc` column (separated with commas). The user may choose

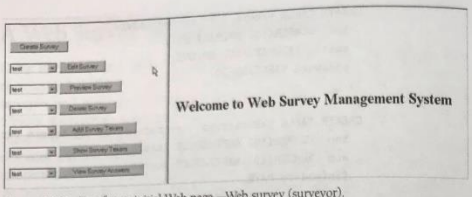

Figure 10.6 Two-frame initial Web page—Web survey (surveyor).

exactly one of these possibilities as an answer. A `SelectList` question is similar to a `RadioButton` question, except that the user may choose one or more of the options listed when answering this type of question.

There should be two different sign-in Web pages for the two types of users in the system. Once a surveyor has signed in successfully, the surveyor should be shown the two-frame Web page shown in Figure 10.6.

The right-hand frame is used as a workspace. The seven menu options in the left-hand frame are

- *Create Survey.* The user is able to name the survey and specify survey questions using the screen (shown in Figure 10.7) that appears in the right-hand frame when this option is chosen.

  Subsequent question screens will not have the survey name field.

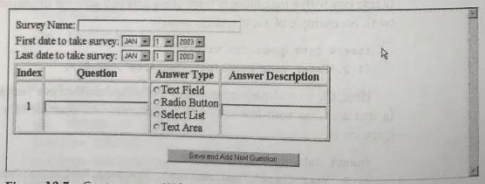

Figure 10.7 Create survey—Web survey management.

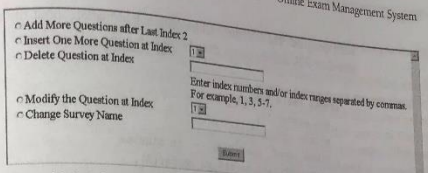

Figure 10.8 Edit survey—Web survey management.

- *Edit Survey.* The user is able to edit questions using the screen (shown in Figure 10.8) that appears in the right-hand frame when this option is chosen.
- *Preview Survey.* This option should display the survey as seen by the survey taker.
- *Delete Survey.* This option should delete the survey, including all its questions.
- *Add Survey Takers.* This option should allow the user to add emails of survey takers to the survey. An automatic email should be sent to the survey takers, giving them the URL to follow to take the survey, the deadline for taking the survey, and a system-generated password to use to sign in to the system to take the survey.
- *Show Survey Takers.* This option should display a list of current survey takers for the particular survey.
- *View Survey Answers.* This option should display the answers to the questions in a suitable format.

The sign-in page for survey takers contains text boxes for email and password and a submit button. Upon successfully signing in, the survey taker is shown a pull-down list of surveys assigned to him or her. The survey taker chooses one of the surveys and submits the request. The survey taker is then shown the survey questions along with two submit buttons labeled Save and Submit.

10.13 Online Exam Management System

Using PL/SQL or Java servlets, implement a Web-based application that allows multiple-choice exams to be created as well as administered. The data requirements for the project have already been analyzed, and six Oracle 10g relational tables have

been designed. The first three tables contain information about the online exams, and their SQL definitions are given below.

```
create table exam (
  eno number(5),
  etitle varchar2(50),
  timeAllowed  number(8), -- in minutes
  numberOfQuestionsPerPage number(3),
  primary key (eno)
);

create table question (
  eno number(5),
  qno number(5),
  qtext varchar2(2048),
  correctAnswer char(1), -- must be an answer option
  foreign key (eno) references exam,
  primary key (eno,qno)
);

create table answerOption (
  eno number(5),
  qno number(5),
  ono char(1) check (ono in ('A','B','C','D','E')),
  optionText varchar2(256),
  foreign key (eno,qno) references question,
  primary key (eno,qno,ono)
);
```

The `exam` table records high-level information about exams. Each exam has a unique identifier (`eno`), a title, the time allowed for the exam (in minutes), and the number of questions to be displayed per page.

The `question` table records information about questions in each exam. Each question is uniquely identified by a question number (`qno`) within an exam. The questions are numbered sequentially from `1`. So, the `eno, qno` combination serves as the key for the `question` table. For each question, the question text is recorded. The question text may contain formatting characters such as newline (`\n`) or tab (`\t`) so that they can be presented in a reasonable format. The correct answer is also recorded for each question. Since the exams are multiple-choice exams, there can

be exactly one correct answer, and it should be one of the answer choices for the question.

The `answerOption` table records the different choices or options from which the exam taker chooses their response. The answer option has a unique identifier (`ono`) for a given exam and a given question and must be one of the five values `'A'`, `'B'`, `'C'`, `'D'`, or `'E'`. So, the `eno,qno,ono` combination serves as the key for the `answerOption` table. For each option, the answer text is recorded. This may also contain some formatting characters so that it may be displayed in a reasonable format.

The remaining three tables contain information about exam takers, their enrollments in the exams, and their responses to questions in the exams they take. The SQL definitions of these tables are shown below.

```
create table user (
  uno number(5), -- system generated starting at 1
  email varchar2(64),  -- unique key used for signing in
  password varchar2(64),
  fname varchar2(64) not null,
  lname varchar2(64) not null,
  address1 varchar2(64),
  address2 varchar2(64),
  city varchar2(64),
  state varchar2(64),
  zip number(5),
  primary key (uno)
);

create table enroll (
  uno number(5),
  eno number(5),
  startTime date,
  finishTime date,
  foreign key (uno) references users,
  foreign key (eno) references exam,
  primary key (uno,eno)
);

create table userResponse (
  uno number(5),
  eno number(5),
```

```
    qno number(5),
    response char(1)
      check (response in ('A','B','C','D','E','N')),
                -- N for No Answer
    foreign key (uno,eno) references enrolls,
    foreign key (eno,qno) references question,
    primary key (uno,eno,qno)
);
```

The **user** table records basic information about potential exam takers. The **enroll** table records information about which user has enrolled in which exam. It also records the time when the exam taker starts taking the exam and the time when they finish taking the exam. The **userResponse** table records the responses of the exam taker to each question they provide an answer to. If they do not answer a particular question, a response of "N" is recorded.

The system should be built in two separate modules:

- *The admin module.* The admin module should allow the exam administrators to create, edit, and delete exams. The Create Exam option should present the user with a data input form for top-level exam details such as exam title, time allotted to the exam, and the number of questions per Web page. After this data is submitted by the user, an Add Question Web page should be presented to the user. This page should have data input form for question text and a select list of integers from 2 to 5 indicating the number of answer choices for this question. Upon selection of a number between 2 and 5, the appropriate number of text area elements should be created on the same page, and the user can enter the answer choices in the text area elements. The user should also be able to select one of the answer choices as the "correct" answer to the question using radio buttons. Once all information is input, the user can submit the question to be added to the database. In response to the submission, the user should be presented with the Add Question Web page in case they want to add the next question. The Delete Exam option should present the user with a select list of exams in which no user has enrolled and a submit button. The user can choose an exam from the select list and submit the exam for deletion. The Edit Exam option should allow the user to add new questions at a particular position and delete a question. In either case, care must be taken that all questions are numbered sequentially starting at 1, even after the add/delete operation has taken place.
- *The user module.* The user module should allow ordinary users to register, sign in, update profile, sign up for exams, take exams, and see their results. The Register, Update Profile, and Sign In options are similar to many Web applications. A

standard login page is provided, with email and password text boxes, along with an "*If you do not have an account, register here*" link. Once logged in successfully, the user should be presented with several options, including Update Profile, in which they can change some of the data about themselves such as password, address, etc. Another option is to Enroll in Exam, in which the user is presented with a select list of all available exams.

Note: If the student is already enrolled in the exam and has finished taking the exam or is currently taking the exam (i.e., has started taking the exam but not yet finished), a warning should be issued stating that his or her answers will be reset. You may confirm that the user wishes to reset the old exam.

Once enrolled, you should present the user with a confirmation that includes details about the exam he or she has just signed up for. The Take an Exam option should present the user with questions from where they left off the last time they signed on to take the exam. Questions should be presented in order, using the predefined number of questions per page. Once answers are submitted, they cannot be revisited. The user is then presented the next set of questions; this continues until time runs out or there are no more questions. The View Grade Report option allows the user to choose the exam for which they like to view results. Only the list of exams that have been completed should be presented. The grade report must include number of questions answered correctly, total number of questions, percentage correct, and a detailed listing of user responses and correct answers.

The data for a sample exam in the form of an SQL script is given below.

```
insert into exam values (3,'Elementary History',10,3);

insert into question values
  (3,1,'The Battle of Gettysburg was fought during which' ||
       ' war?','C');
insert into answerOption values (3,1,'A','World War II');
insert into answerOption values (3,1,'B',
                  'The Revolutionary War');
insert into answerOption values (3,1,'C','The Civil War');
insert into answerOption values (3,1,'D','World War I');

insert into question values
  (3,2,'Neil Armstrong and Buzz Aldrin walked how many \n' ||
       'minutes on the moon in 1966?','B');
insert into answerOption values (3,2,'A','123');
insert into answerOption values (3,2,'B','None');
```



```
insert into answerOption values (3,2,'C','10');
insert into answerOption values (3,2,'D','51');

insert into question values
  (3,3,'Which presidents held office during World War' ||
       ' II?','D');
insert into answerOption values (3,3,'A',
                  'Franklin D. Roosevelt');
insert into answerOption values (3,3,'B',
                  'Dwight D. Eisenhower');
insert into answerOption values (3,3,'C','Harry Truman');
insert into answerOption values (3,3,'D','Both A and C');

insert into question values
  (3,4,'In a communist economic system, people:','B');
insert into answerOption values (3,4,'A',
                  'Are forced to work as slaves');
insert into answerOption values (3,4,'B',
                  'Work for the common good');
insert into answerOption values (3,4,'C',
                  'Work from home computers');
insert into answerOption values (3,4,'D','Don''t work');

insert into question values (3,5,
       'Which president did not die while in office?','D');
insert into answerOption values (3,5,'A','John F. Kennedy');
insert into answerOption values (3,5,'B',
                  'Franklin D. Roosevelt');
insert into answerOption values (3,5,'C','Abraham Lincoln');
insert into answerOption values (3,5,'D','Ronald Reagan');
insert into answerOption values (3,5,'E',
                  'James A. Garfield');
```

10.14 Online Bulletin Board

Using PL/SQL or Java servlets, implement an online bulletin board system. The bulletin board is to be used only by registered members. The data requirements

for the system have already been analyzed, and the following two Oracle 10*g* tables and a sequence object have been designed:

```
create table bbusers (
  email varchar2(50), -- used as userid to sign in
  name varchar2(30),
  password varchar2(10),
  nickname varchar2(30),
  lastaccess date,
  primary key (email)
);

create sequence postSeq start with 1;

create table postings (
  postId number(5), -- use postSeq to generate this value
  postDate date,
  postedBy varchar2(50),
  postSubject varchar2(100),
  content varchar2(1024),
  ancestorPath varchar2(100),
  primary key (postId),
  foreign key (postedBy) references bbusers
);
```

The first table, **bbusers**, records basic information about the users of the system. The **postSeq** object is used to generate a unique posting number to be used whenever a new message or a reply to a previous message is posted to the bulletin board. The **postings** table records the details of each message posted to the bulletin board. Besides a unique posting id, the time and date of posting, the email of the user posting the message, the subject of the posting, and the text content of the posting, this table also records the path from root to parent node of the current posting. For example, `1:5:6:12` would indicate path `1 -> 5 -> 6 -> 12`, where `1`, `5`, `6`, and `12` are posting identifiers of nodes lying in the path from root node `1` to parent node `12` of the current message. Some sample data is shown below.

```
insert into bbusers values
  ('a@abc.com','Andy Smith','a123','baggy',null,null);
insert into bbusers values
  ('b@abc.com','Craig Smith','a123','runner',null,null);
insert into bbusers values
```

```
    ('c@abc.com','Craig Rich','a123','slogger',null,null);
insert into bbusers values
  ('d@abc.com','Mike Arlott','a123','ateam',null,null);
:
:
insert into postings values (1,'11-JAN-2003','a@abc.com',
  'Message 1','Welcome to the bulletin board',null);
insert into postings values (2,'12-JAN-2003','b@abc.com',
  'Follow up to Message 1','This is posting 2','1');
insert into postings values (3,'13-JAN-2003','c@abc.com',
  'Another follow up to Message 1','This is posting 3','1');
insert into postings values (4,'14-JAN-2003','a@abc.com',
  'Message 4','This is posting 4',null);
insert into postings values (5,'15-JAN-2003','a@abc.com',
  'Message 5','This is posting 5',null);
insert into postings values (6,'16-JAN-2003','a@abc.com',
  'Message 6','This is posting 6',null);
insert into postings values (7,'17-JAN-2003','c@abc.com',
  'Follow up to posting 2','This is posting 7','1:2');
insert into postings values (8,'18-JAN-2003','d@abc.com',
  'Second follow up to posting 2','This is posting 8','1:2');
insert into postings values (9,'19-JAN-2003','a@abc.com',
  'Follow up to posting 8','This is posting 9','1:2:8');
insert into postings values (10,'20-JAN-2003','b@abc.com',
  'Follow up to posting 6','This is posting 10','6');
```

The Web application should have a user sign-in and registration page. Upon successful sign-in to the bulletin board, all the messages should be listed in reverse chronological order with properly indented follow-up messages. The message subject should be hyperlinked, and the user nickname of the user who posted the message should be listed next to the subject. At the top and bottom of the message list, links to post a new message and a logout link should be provided. The New Message link should lead to a new Web page that accepts the subject and text content of the message for posting. Upon posting the new message, the Message List page should be displayed. Each message in the message list should be hyperlinked, and upon clicking the link the user should be taken to a new Web page, where the detailed text of the message is listed along with a place for the user to enter a follow-up message. Upon submission of a follow-up message, the Message List page should be displayed. A search feature should also be implemented. The user should be able to choose either SUBJECT or BODY or BOTH and enter a string in a text box to search for messages. Results should be displayed in a similar manner as the entire bulletin board (i.e., if a search string is found in a posting, its topmost ancestor message and the entire subtree under the topmost ancestor must be displayed). Finally, an automatic sign-out should be implemented with 2 minutes of inactivity.

10.15 Data Input Forms

A *data input form* is essentially an HTML form by which users submit data to be stored in a relational database. A data input form consists of one or more HTML form elements of the following types: text box, checkbox, radio button, select list, multiple select list, submit, and reset. Of these, the checkboxes and multiple select lists are considered multiple-valued elements, since the user may choose more than one item to be submitted. A data type is associated with text box, checkbox, radio button, select list, and multiple select list elements.

Consider a data input form titled "`Favorite Programming Languages`" that contains

- a text box labeled "`Student Name`" with a visible width of 20 characters and a maximum width of 50 characters,
- a radio button group that contains three radio buttons labeled "`BS`", "`MS`", and "`PhD`," and
- a checkbox group of four checkboxes labeled "`Java`", "`C++`", "`VBasic`", and "`Python`".

Let the names of the three elements be "`sname`," "`degree`," and "`language`." In addition, let the radio button group have a label "`Your degree level?`" and the checkbox group have a label "`Your favorite programming languages?`" associated with them. Such a data input form is described in XML (`form.xml`) as follows:

```
<?xml version = "1.0"?>
<dataInputForm id="1">
  <title>Favorite Programming Languages</title>

  <textbox datatype="string" key="key">
    <name>sname</name>
    <caption>Student Name:</caption>
    <size>20</size>
    <maxlength>50</maxlength>
  </textbox>
```



```
  <break></break> <break></break>

  <radio>
    <name>degree</name>
    <caption>Your degree level?</caption>
    <radiogroup>
      <radioelement>
        <value>BS</value>
        <caption>BS</caption>
      </radioelement>
      <break></break>
      <radioelement>
        <value>MS</value>
        <caption>MS</caption>
      </radioelement>
      <break></break>
      <radioelement>
        <value>PhD</value>
        <caption>PhD</caption>
      </radioelement>
    </radiogroup>
  </radio>
  <break></break> <break></break>

  <checkboxes>
    <name>language</name>
    <caption>Your favorite programming languages?</caption>
    <checkboxgroup>
      <checkbox status="checked">
        <value>Java</value>
        <caption>Java</caption>
      </checkbox>
      <break></break>
      <checkbox>
        <value>C++</value>
        <caption>C++</caption>
      </checkbox>
      <break></break>
      <checkbox>
        <value>VB</value>
        <caption>VB</caption>
      </checkbox>
      <checkbox>
        <value>Python</value>
        <caption>Python</caption>
      </checkbox>
    </checkboxgroup>
  </checkboxes>
  <break></break><break></break>

  <submit>
    <caption>Submit Data</caption>
  </submit>
</dataInputForm>
```


The XML document has the root element `<dataInputForm>` with an `id` attribute. It contains a `<title>` subelement that contains a title for the form. The `<title>` element is followed by one or more form elements. Each of the form elements is described by a subelement that has two optional attributes: `datatype`, whose value can be "`integer`", "`decimal`", or "`string`", and `key`, whose value is "`key`". If the `datatype` attribute is missing, the default value of "`string`" is assumed. The `key` attribute indicates that the value submitted in the form element is part of the primary key of the main relational table in which the data is stored. At least one single-valued element must have the `key` attribute if there is at least one multivalued element. This is to facilitate storing multivalued elements in a separate relational table. The format of the individual form elements is described below.

- *Text box:* The `textbox` element is described using the tag `<textbox>` and has subelements `name`, `caption`, and `maxlength` and an optional subelement `size`.
- *Checkbox:* The `checkbox` element is described using the tag `<checkboxes>` and must contain `name`, `caption`, and `checkboxgroup` subelements. Within the `checkboxgroup` element, one or more `checkbox` elements, each with a `value` and `caption` subelement, may be present. The `checkbox` element may have an optional attribute called `status` with a value of "`checked`."
- *Radio button:* Similar to checkbox.
- *Select list:* The `select` element is described using the tag `<select>` and has `name`, `caption`, and `options` subelements. The `options` element has one or more `option` subelements, each of which has a `value` and `caption` subelement.

- *Multiple select list:* The `multiselect` element, described using the tag `<multiselect>` and is similar to the `select` element except that it has an additional subelement, `size`, which indicates the size of the select window.
- *Submit and reset:* The `submit` and `reset` elements are described using the tags `<submit>` and `<reset>`, respectively. The `submit` element may contain an optional `caption` subelement.

In addition to the seven types of form elements, any number of `<break/>` subelements may be introduced between form elements as well as between different options in a checkbox or radio button group.

Implement a Java application that processes XML files containing information about data input forms. The Java application takes as command-line input an XML file name/URL and produces as output the following four files:

1. An HTML file that contains code to display the data input form on a browser. The HTML code must include Javascript code to perform some basic validation checks, such as data type checking for text box input.
2. An Oracle SQL script file that contains `create table` statements to create relational table(s) that can accept data submitted from the data input form. One main table containing the single-valued elements from the form as columns should be created. In addition, for each multivalued element (checkbox or multiselect), an additional table containing the "`key`" element(s) and the multivalued element as columns should be created.
3. A Java servlet source file that contains code to process data submitted by the user using the data input form. The program should simply insert the data into the database table(s).
4. A Java servlet source file that contains code to display contents of the database in tabular form.

For the sample data form shown earlier, the following two relational tables should be created:

```
create table formMain (
  sname varchar2(50), -- 50 is given maxlength
  degree varchar2(3), -- length of longest value (PhD)
  primary key (sname)
);

create table formlanguage (
  sname varchar2(50),
  language varchar2(6), -- length of longest value (Python)
  primary key (sname,language),
  foreign key (sname) references formMain
);
```


The Java application should use the SAX or DOM XML parser and must perform some basic semantic checks on the XML data input form file. Semantic errors such as a form element without a name or caption, more than one form element having the same name, data type mismatch, no key element indicated when at least one multivalued element is present, etc., should be reported. In case of any syntax or semantic error, no output is generated.