

Fundamentals of Database Systems
Laboratory Manual¹

Rajshekhar Sunderraman
Georgia State University

August 2010

¹ To accompany Elmasri and Navathe, Fundamentals of Database Systems, 6th Edition, Addison-Wesley, 2010.

Preface

This laboratory manual accompanies the popular database textbook *Elmasri and Navathe, Fundamentals of Database Systems, 6th Edition, Addison-Wesley, 2010*. It provides supplemental materials to enhance the practical coverage of concepts in an introductory database systems course. The material presented in this laboratory manual complement many of the chapters of the Elmasri/Navathe text typically covered in most introductory database systems courses.

Chapter Mappings

The laboratory manual consists of 8 chapters and the following table shows the mapping to the chapters in the Elmasri/Navathe textbook:

Laboratory Manual Chapter	Elmasri/Navathe 6th Edition Chapter(s)
Chapter 1	Chapters 7, 8, and 9
Chapter 2	Chapters 3, 6, and 26
Chapter 3	Chapters 4, 5, and 13
Chapter 4	Chapters 4, 5, and 14
Chapter 5	Chapters 15 and 16
Chapter 6	Chapter 11
Chapter 7	Chapter 12
Chapter 8	Chapters 13 and 14

Chapter 1 presents ERWin, a popular data modeling software that allows database designers to represent Entity-Relationship diagrams and automatically generate relational SQL code to create the database in one of several commercial relational database management systems such as Oracle or Microsoft SQLServer. The material presented in this chapter is tutorial in nature and covers the COMPANY database design of the Elmasri/Navathe text in detail.

Chapter 2 presents three interpreters that can be used to execute queries in Relational Algebra, Domain Relational Calculus, and Datalog. These interpreters are part of a Java package that includes a rudimentary database engine capable of storing relations and able to perform basic relational algebraic operations on these relations. It is hoped that these interpreters will allow the student to get a better understanding of abstract query languages.

Chapter 3 presents techniques to interact and program with Oracle database management system. A popular data-loading tool for Oracle databases called SQL Loader is introduced and the COMPANY database of the Elmasri/Navathe text is extended with additional data to make it more interesting to program with. Programming applications that access Oracle databases is then introduced in Java using the JDBC interface. Several non-trivial example programs are discussed.

Chapter 4 covers MySQL database management system, a popular open source database system that is increasing used by small and medium sized organizations. Programming Web applications in PHP that accesses MySQL databases is introduced with a complete database browser application for the COMPANY database as well as a complete Online Address Book application.

Chapter 5 introduces a Prolog-based toolkit for relational database design. The toolkit, called Database Designer (DBD), allows the student to work with numerous concepts and algorithms that deal with functional dependency theory and data normalization. The student may use DBD to verify answers to many questions related to functional dependency theory and normalization algorithms.

Chapter 6 presents programming with a popular open source Object-Oriented Database Management system, db4o. Creating and populating objects in db4o is covered as well various methods to query and retrieve data from the object-oriented database is introduced. Db4o supports various object-oriented programming interfaces, but the Java interface is covered in the lab manual.

Chapter 7 presents XML and its related technologies. Query languages XPath and XQuery are covered as well as schema specification language XML Schema. Numerous examples are presented including a complete specification of the company database in XML along with a XML Schema.

Chapter 8 presents several semester-long projects for students in introductory database courses to complete. These projects may be implemented in Java, PHP or any other favorite programming language and may access Oracle, MySQL or any other relational database management system.

Code

The laboratory manual comes with all the code and data presented in the different chapters. The software for the relational query interpreters as well as the database designer (DBD) also accompanies the laboratory manual.

Software

The software systems discussed and used in the laboratory manual are ERWin from Computer Associates, Oracle DBMS from Oracle, and MySQL, PHP, db4o, and SWI-Prolog from open source. Both Computer Associates and Oracle have educational pricing for their software and we expect the individual universities and colleges that use this laboratory manual to provide the software for use by their students.

Rajshekhar Sunderraman
Atlanta, Georgia
August 2010

Contents

ER MODELING TOOLS	6
1.1 STARTING WITH ERWIN	6
1.2 ADDING ENTITY TYPES	7
1.3 ADDING RELATIONSHIPS	10
1.4 FORWARD ENGINEERING	12
1.5 SUPERTYPE/SUBTYPE EXAMPLE	15
EXERCISES	17
ABSTRACT QUERY LANGUAGES	21
2.1 CREATING THE DATABASE	21
2.2 RELATIONAL ALGEBRA INTERPRETER.....	23
2.2.1 <i>Relational Algebra Syntax</i>	23
2.2.2 <i>Naming of Intermediate Relations and Attributes</i>	25
2.2.3 <i>Relational Algebraic Operators Supported by the RA Interpreter</i>	26
2.2.4 <i>Examples</i>	27
2.3 DOMAIN RELATIONAL CALCULUS INTERPRETER.....	30
2.3.1 <i>Domain Relational Calculus Syntax</i>	30
2.3.2 <i>Safe DRC Queries</i>	32
2.3.3 <i>DRC Query Examples</i>	34
2.4 DATALOG INTERPRETER	35
2.4.1 <i>Datalog Syntax</i>	35
2.4.2 <i>Datalog Query Examples</i>	36
EXERCISES	42
RELATIONAL DATABASE MANAGEMENT SYSTEM: ORACLE™	45
3.1 COMPANY DATABASE.....	45
3.2 <i>SQL*PLUS</i> UTILITY.....	49
3.3 <i>SQL*LOADER</i> UTILITY	50
3.4 PROGRAMMING WITH ORACLE USING THE JDBC API	53
EXERCISES	63
RELATIONAL DATABASE MANAGEMENT SYSTEM: MYSQL	69
4.1 COMPANY DATABASE.....	69
4.2 <i>MYSQL</i> UTILITY	73
4.3 <i>MYSQL</i> AND <i>PHP</i> PROGRAMMING	75
4.4 <i>ONLINE ADDRESS BOOK</i>	87
EXERCISES	100
DATABASE DESIGN (DBD) TOOLKIT	103
5.1 CODING RELATIONAL SCHEMAS AND FUNCTIONAL DEPENDENCIES	103
5.2 INVOKING THE SWI-PROLOG INTERPRETER	103
5.3 DBD SYSTEM PREDICATES	105
5.3.1 <i>xplus(R,F,X,Xplus)</i>	105
5.3.2 <i>finfplus(R,F,[X,Y])</i>	106
5.3.3 <i>fplus(R,F,Fplus)</i>	106
5.3.4 <i>implies(R,F1,F2) and equiv(R,F1,F2)</i>	107
5.3.5 <i>superkey(R,F,K) and candkey(R,F,K)</i>	108
5.3.6 <i>mincover(R,F,FC)</i>	109
5.3.7 <i>ljd(R,F,R1,R2), ljd(R,F,D), and fpd(R,F,D)</i>	110
5.3.8 <i>is3NF(R,F) and threenf(R,F,D)</i>	113

5.3.9 <i>isBCNF(R,F) and bcnf(R,F,D)</i>	113
EXERCISES	114
OBJECT-ORIENTED DATABASE MANAGEMENT SYSTEMS: DB4O	119
6.1 DB4O INSTALLATION AND GETTING STARTED	119
6.2 A SIMPLE EXAMPLE	120
6.3 DATABASE UPDATES AND DELETES	123
6.4 COMPANY DATABASE.....	123
6.5 DATABASE QUERYING	125
6.5.1 <i>Query by Example</i>	125
6.5.2 <i>Native Queries</i>	125
6.5.3 <i>SODA (Simple Object Database Access) Queries</i>	126
6.6 COMPANY DATABASE APPLICATION	129
6.6.1 <i>CreateDatabase.java</i>	129
6.6.2 <i>createEmployees</i>	130
6.6.3 <i>createDependents</i>	131
6.6.4 <i>createDepartment</i>	132
6.6.5 <i>createProjects</i>	133
6.6.6 <i>createWorksOn</i>	134
6.6.7 <i>setManagers</i>	135
6.6.8 <i>setControls</i>	136
6.6.9 <i>setWorksFor</i>	137
6.6.10 <i>setSupervisors</i>	138
6.6.11 <i>Complex Retrieval Example</i>	139
6.7 WEB APPLICATION	140
6.7.1 <i>Client-Server Configuration</i>	140
EXERCISES	146
XML.....	153
7.1 XML BASICS	153
7.2 COMPANY DATABASE IN XML	155
7.3 XML EDITOR EDITIX.....	157
7.4 XPATH	159
7.5 XQUERY	163
7.6 XML SCHEMA	173
EXERCISES	178
PROJECTS	180
8.1 STUDENT REGISTRATION SYSTEM (GOLUNAR)	180
8.2 ONLINE BOOK STORE DATABASE SYSTEM.....	189
8.3 ONLINE SHOPPING SYSTEM	198
8.4 ONLINE BULLETIN BOARD SYSTEM.....	204
8.5 ONLINE EXAM MANAGEMENT SYSTEM.....	207
8.6 ONLINE AUCTIONS	211
BIBLIOGRAPHY	215

CHAPTER 1

ER Modeling Tools

This chapter introduces ERWin, a popular data-modeling tool used in the industry. ERWin is a powerful tool that allows database designers to enter their Entity Relationship (ER) diagrams in a graphical form and produce physical database designs for popular relational database management systems such as Oracle and Microsoft SQLServer.

The use of ERWin is illustrated in this chapter using the ER schema diagram for the COMPANY database shown in Figure 7.2 of the Elmasri/Navathe text.

1.1 Starting with ERWin

The ERWin Data Modeler workspace is shown in Figure 1.1.

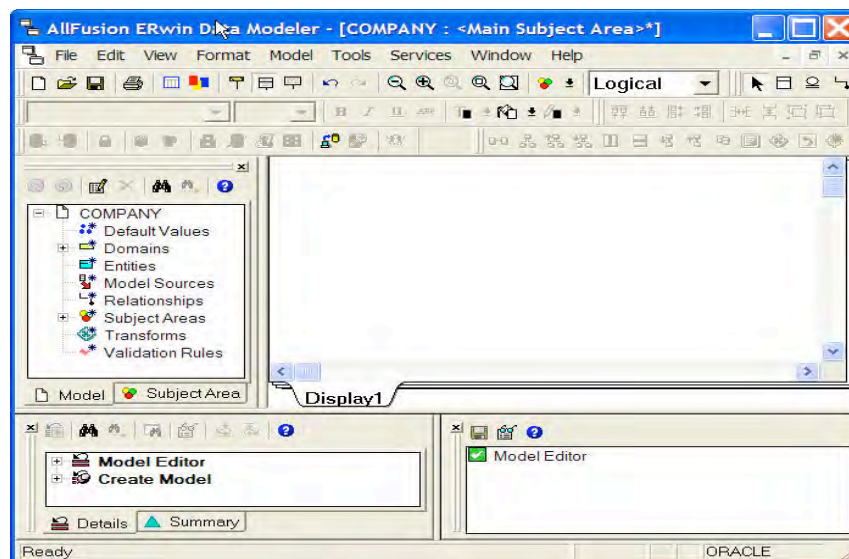


Figure 1.1: ERWin Data Modeler Workspace

The top part of the workspace consists of Menu and Toolbars. The middle part of the workspace consists of two panes: the *model explorer panel* on the left providing a text based view of the data model and the *diagram window panel* on the right providing a graphical view of the data model. The lower part of the workspace consists of two panes: the *action log panel* on the left that displays a log of all changes made to the data model under design and the *advisories panel* that displays messages associated with the actions performed on the data model under design.

ERWin supports three model types for use by the database designer:

1. Logical: A conceptual model that includes entities, relationships, and attributes. This model type is essentially at the ER modeling level.

2. Physical: A database specific model that contains relational tables, columns and associated data types.
3. Logical/Physical: A single model that includes both the conceptual level objects as well as physical level tables. In this chapter we will use this model type.

To create a model in ERWin, one should launch the program and then choose the “New” option from the File menu. The Create Model dialog appears as shown in Figure 1.2.

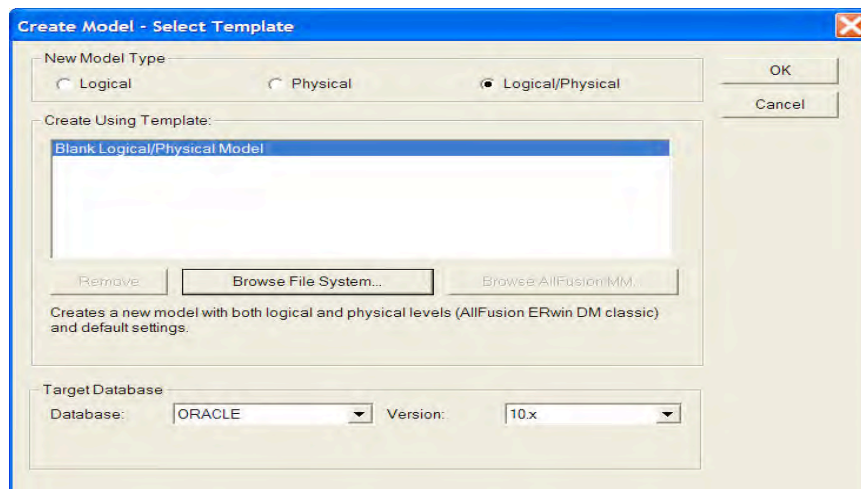


Figure 1.2: Create Model dialog window

In this dialog window, the user should choose the type of model. Typically the Logical/Physical model type should be chosen if the final goal is to produce a relational design for the database. The target database may also be chosen. In this case, Oracle 10.x version is chosen as the target database. In a future step, we will illustrate how ERWin can be used to generate SQL code to create the database objects in Oracle 10.x database.

The workspace for the new model will be populated by the system with a default name of Model_n. This name may be changed in the model explorer pane by right clicking the model name and choosing the Properties option. This brings up a new window in which the name and other properties of the model may be changed. Besides changing the model name, the “Transform” options should be checked. This would allow for many-to-many relationships to be transformed correctly into separate relational tables in the physical model. In addition any sub-type/super-type relationships will also be transformed correctly in the physical model.

1.2 Adding Entity Types

To add an entity type to the database design, the user may either right click the “Entities” entry in the model explorer pane and choose “New” or choose the “Entity” icon in the Menus and Toolbars section of the workspace and click in the diagram window panel. An entity box shows up in the diagram window panel with a default entity name (E/n) that can be changed either in the diagram window panel or in the model explorer pane. Figure 1.3 shows the addition of the EMPLOYEE entity type in the COMPANY database.

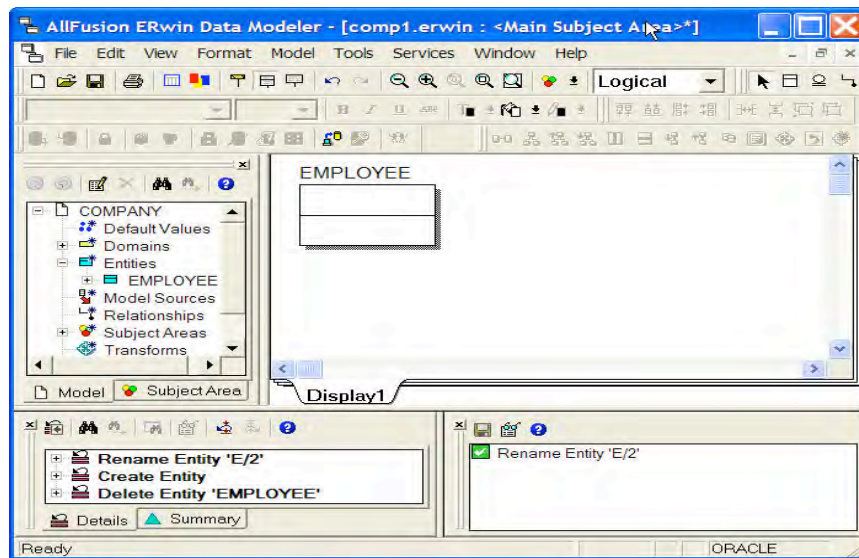


Figure 1.3: Add EMPLOYEE entity to the COMPANY database

To add attributes to the EMPLOYEE entity type, the user may right click within the EMPLOYEE entity box in the diagram window panel and choose “Attributes”. This brings up a separate window using which new attributes may be added. The attribute window is shown in Figure 1.4.

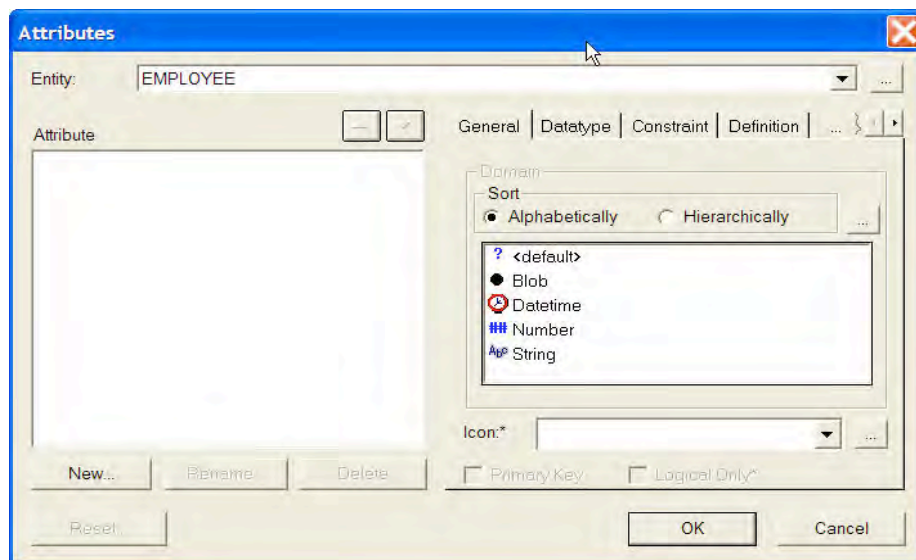


Figure 1.4: Attribute Window

The user may now add attributes one at a time by clicking the “New” button. A separate window pops up as shown in Figure 1.5.

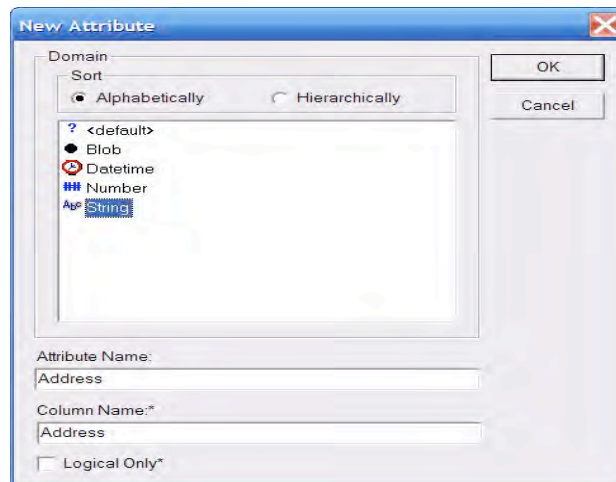


Figure 1.5: New Attribute Window

The user may choose an appropriate Domain (data type) and enter the Attribute Name and click OK. The data type may be further refined in the Attribute Window by choosing the Datatype tab and entering a precise data type. The user may also choose to designate this attribute as a primary key by selecting this option in the Attribute window.

After adding a few attributes to the EMPLOYEE entity type the Attributes window is shown in Figure 1.6.

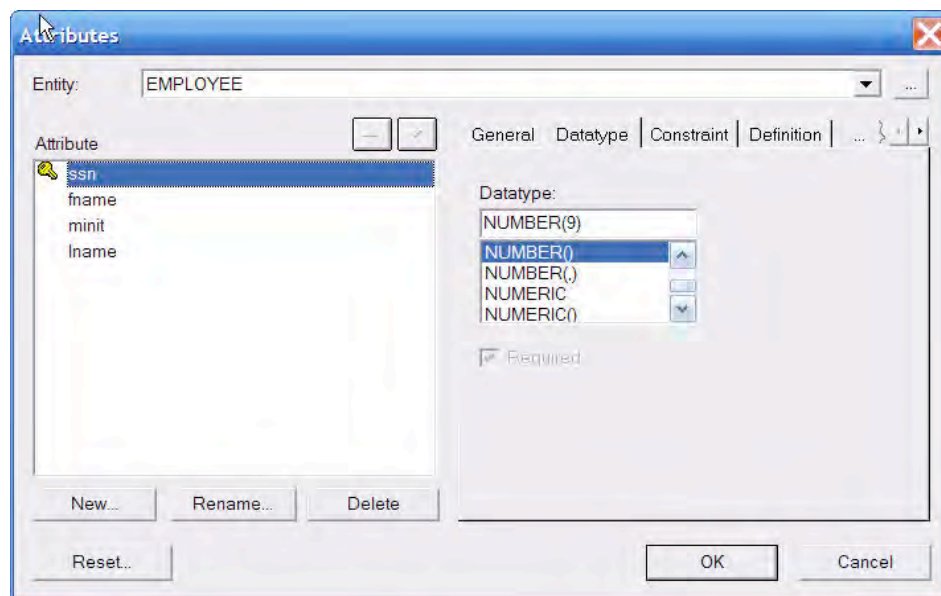


Figure 1.6: Attribute Window with four attributes

In this way, we can create each of entity types: EMPLOYEE, DEPARTMENT, PROJECT, and DEPENDENT for the COMPANY database.

Weak Entity Sets

By default any entity type created as discussed so far is classified as an independent entity type. ERWin will classify an entity type as “weak” as soon as it participates in an identifying relationship. For example, the entity type DEPENDENT will be classified as “weak” in a subsequent step when we add the identifying relationship from EMPLOYEE to DEPENDENT in the next section. Weak entity types are denoted by rounded rectangles in the diagram window panel.

Multi-Valued Attributes

Multi-valued attributes such as the locations attribute for the DEPARTMENT entity type cannot be modeled easily with ERWin. To handle such attributes, a separate entity type LOCATIONS is created and a many-to-many relationship between DEPARTMENT and LOCATIONS will be established in the next section.

1.3 Adding Relationships

Three types of relationships are supported in ERWin: identifying, non-identifying, and many-to-many. ERWin classifies the child entity type in an identifying relationship as “weak”. To add a relationship, the user may simply right click the Relationships entry in the model explorer pane and choose “New”. This pops up a new relationship window as shown in Figure 1.7.

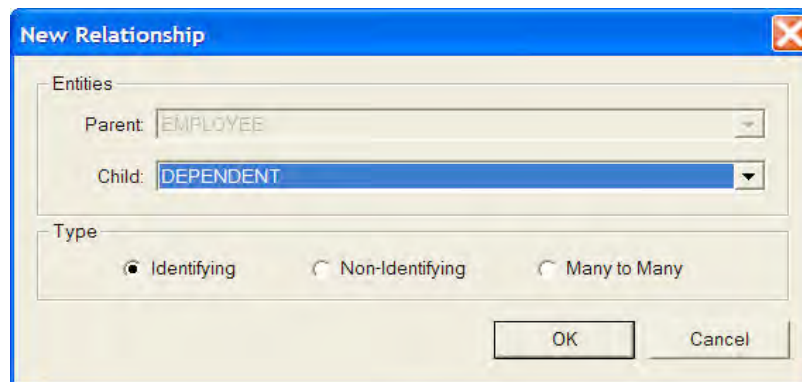


Figure 1.7: New Relationship Window

After choosing the parent and child entity types and the type of relationship and clicking OK, the new relationship is added and is reflected by a line connecting the two entity types in the diagram window panel. The many-to-many relationships are denoted by solid connecting lines, with two black dots at the two ends. Non-identifying relationships are denoted by a dashed connecting line with a black dot at many-end and a square-shaped symbol at the one-end. Identifying relationships are denoted by a solid connecting line with a black dot at the many-end and nothing special at the one-end.

After creating a new relationship, the user may add verb phrases and other properties of the relationship by right clicking the connecting line in the diagram and choosing properties.

In the case of the COMPANY database, we create the following relationships:

- One identifying relationship from EMPLOYEE to DEPENDENT.
- Two many-to-many relationships, one from EMPLOYEE to PROJECT and the other from DEPARTMENTS to LOCATIONS, and
- Four non-identifying relationships: from EMPLOYEE to DEPARTMENT (one-to-one for manages), from DEPARTMENT to EMPLOYEE (one-to-many for works for relationship), from EMPLOYEE to EMPLOYEE (one-to-many for supervisor/supervisee relationship), and from DEPARTMENT to PROJECT (one-to-many for the controls relationship).

The final logical ER diagram from the diagram window panel is shown in Figure 1.8.

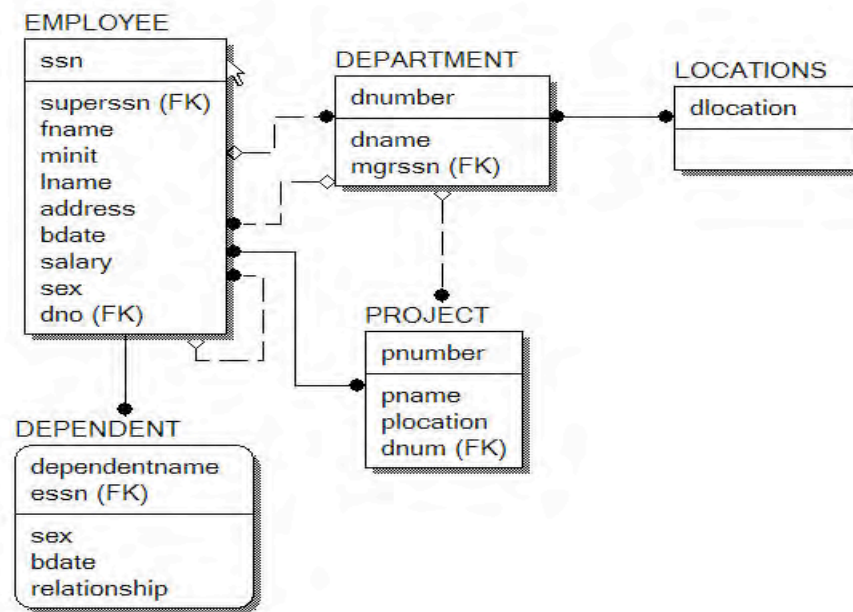


Figure 1.8: Final Logical ER Diagram

Notice that the two many-to-many relationships do not have the transforms applied yet. The transforms are shown in the physical ER diagram (obtained by switching from Logical to Physical in the Menu and Toolbar section) in Figure 1.9. Notice the introduction of the two new “entity types” for the two many-to-many relationships. These entity types are introduced because the transforms are defined at the model level.

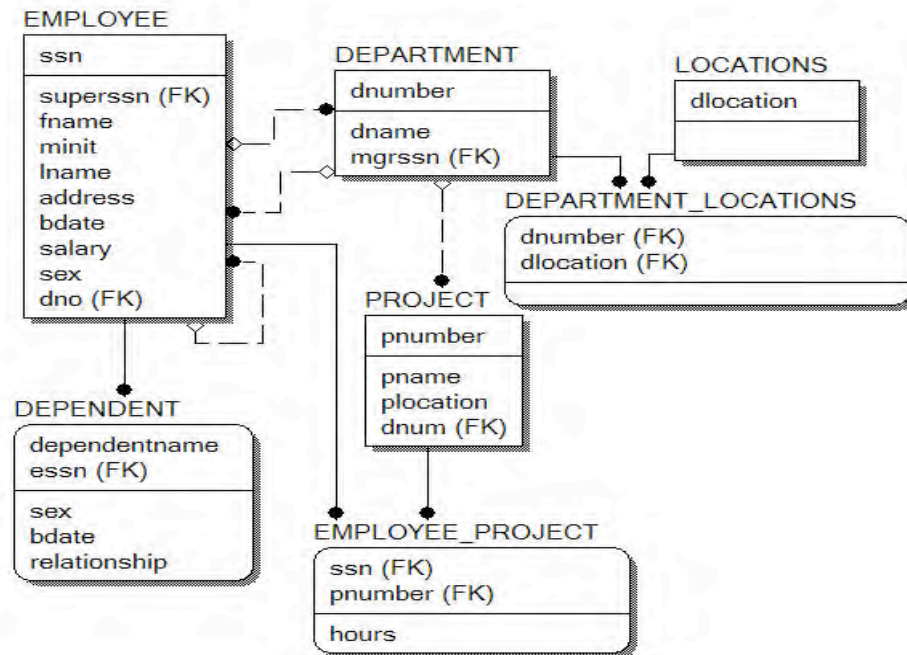


Figure 1.9: Final Physical ER Diagram

1.4 Forward Engineering

ERWin provides a powerful feature called forward engineering that allows the database designer to convert the ER design into a schema generation SQL script for one or more target relational databases. The following SQL script is obtained for the COMPANY database by choosing Tools→Forward Engineering→Schema-Generation option in the Menus and Toolbars section and clicking the “Preview” button.

```
CREATE TABLE DEPARTMENT
(
    dname VARCHAR2(20) NOT NULL ,
    dnumber INTEGER NOT NULL ,
    mgrssn NUMBER(9) NULL
);

ALTER TABLE DEPARTMENT
    ADD PRIMARY KEY (dnumber);

CREATE TABLE DEPARTMENT_LOCATIONS
(
    dnumber INTEGER NOT NULL ,
    dlocation VARCHAR2(20) NOT NULL
);

ALTER TABLE DEPARTMENT_LOCATIONS
    ADD PRIMARY KEY (dnumber,dlocation);
```

```
CREATE TABLE DEPENDENT
(
    dependentname VARCHAR2(20) NOT NULL ,
    sex CHAR NULL ,
    bdate DATE NULL ,
    relationship VARCHAR2(20) NULL ,
    essn NUMBER(9) NOT NULL
);
```

```
ALTER TABLE DEPENDENT
    ADD PRIMARY KEY (dependentname,essn);
```

```
CREATE TABLE EMPLOYEE
(
    ssn NUMBER(9) NOT NULL ,
    superssn NUMBER(9) NULL ,
    fname VARCHAR2(20) NULL ,
    minit CHAR NULL ,
    lname VARCHAR2(20) NOT NULL ,
    address VARCHAR2(50) NULL ,
    bdate DATE NULL ,
    salary NUMBER(8) NULL ,
    sex CHAR NULL ,
    dno INTEGER NULL
);
```

```
ALTER TABLE EMPLOYEE
    ADD PRIMARY KEY (ssn);
```

```
CREATE TABLE EMPLOYEE_PROJECT
(
    ssn NUMBER(9) NOT NULL ,
    pnumber INTEGER NOT NULL ,
    hours NUMBER(3) NULL
);
```

```
ALTER TABLE EMPLOYEE_PROJECT
    ADD PRIMARY KEY (ssn,pnumber);
```

```
CREATE TABLE LOCATIONS
(
    dlocation VARCHAR2(20) NOT NULL
);
```

```
ALTER TABLE LOCATIONS
    ADD PRIMARY KEY (dlocation);
```

```

CREATE TABLE PROJECT
(
    pnumber INTEGER NOT NULL ,
    pname VARCHAR2(20) NULL ,
    plocation VARCHAR2(20) NULL ,
    dnum INTEGER NULL
);

ALTER TABLE PROJECT
    ADD PRIMARY KEY (pnumber);

ALTER TABLE DEPARTMENT
    ADD ( FOREIGN KEY (mgrssn) REFERENCES EMPLOYEE(ssn) ON DELETE SET NULL);

ALTER TABLE DEPARTMENT_LOCATIONS
    ADD ( FOREIGN KEY (dnumber) REFERENCES DEPARTMENT(dnumber));

ALTER TABLE DEPARTMENT_LOCATIONS
    ADD ( FOREIGN KEY (dlocation) REFERENCES LOCATIONS(dlocation));

ALTER TABLE DEPENDENT
    ADD ( FOREIGN KEY (essn) REFERENCES EMPLOYEE(ssn));

ALTER TABLE EMPLOYEE
    ADD ( FOREIGN KEY (superssn) REFERENCES EMPLOYEE(ssn) ON DELETE SET
NULL);

ALTER TABLE EMPLOYEE
    ADD ( FOREIGN KEY (dno) REFERENCES DEPARTMENT(dnumber) ON DELETE SET
NULL);

ALTER TABLE EMPLOYEE_PROJECT
    ADD ( FOREIGN KEY (ssn) REFERENCES EMPLOYEE(ssn));

ALTER TABLE EMPLOYEE_PROJECT
    ADD ( FOREIGN KEY (pnumber) REFERENCES PROJECT(pnumber));

ALTER TABLE PROJECT
    ADD ( FOREIGN KEY (dnum) REFERENCES DEPARTMENT(dnumber) ON DELETE SET
NULL);

```

The above SQL script contains table definitions and basic primary and foreign key constraints definitions. ERWin does provide a number of options to generate views, triggers, indices etc. and these can be set in the forward engineering schema generation window.

1.5 Supertype/Subtype Example

ERWin supports the creation of sub-type/super-type relationships between entity types. Consider the example in Figure 8.3 of the Elmasri/Navathe text in which a super-type entity VEHICLE consists of two sub-types CAR and TRUCK. To create this design in ERWin, the three entity types are created first. Then, the user may click the sub-type button (a circle with two parallel lines below the circle) in the Menus and Toolbars section, followed by clicking the super-type entity (VEHICLES) in the diagram window pane, followed by clicking the sub-type entity (CAR) in the diagram window pane. This process may be repeated for adding other sub-types (TRUCK in this example). The logical model for this example is shown in Figure 1.10.

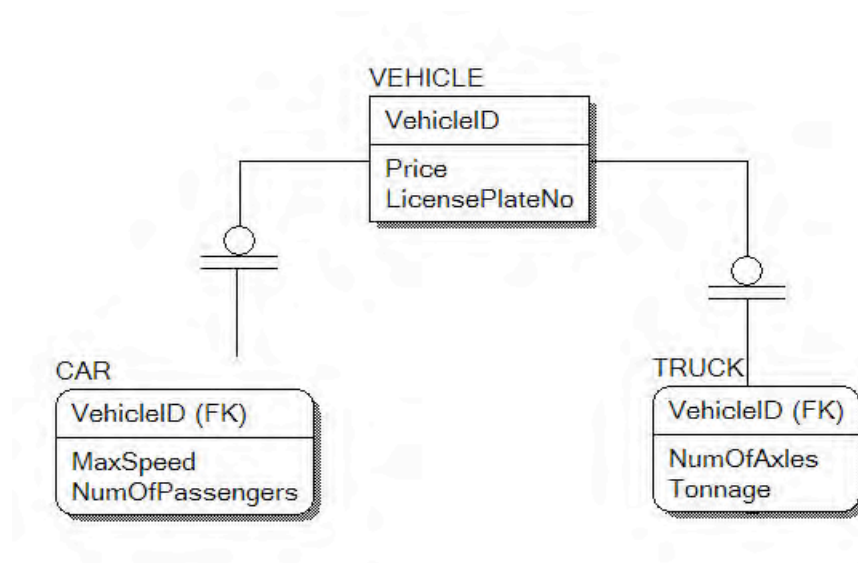


Figure 1.10: Sub-type/Super-type Logical ER Diagram

To customize the properties of the sub-type/super-type relationship, the user may right click the relationship symbol (circle with two parallel lines) and choose Subtype Relationship. This brings up a window shown in Figure 1.11. The user may choose “Complete” subtype (when all categories are known) or “Incomplete” subtype (when all categories may not be known). The user may also add verb phrases etc by right-clicking the relationship line and choosing properties as was done for ordinary relationships. ERWin also allows the user to choose a “discriminator” attribute for the sub-types (an attribute in the super-type whose values determine the sub-type object). If no discriminator attribute is defined, the user may choose “-- -- --”.

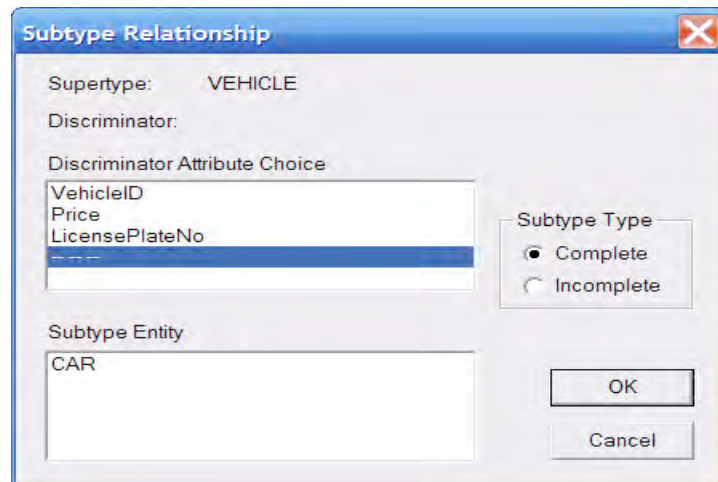


Figure 1.11: Subtype Relationship Properties

The following SQL script is produced using the forward engineering feature of ERWin for the Vehicles example:

```
CREATE TABLE CAR
(
    MaxSpeed INTEGER NULL ,
    NumOfPassengers INTEGER NULL ,
    VehicleID INTEGER NOT NULL
);

ALTER TABLE CAR
    ADD PRIMARY KEY (VehicleID);

CREATE TABLE TRUCK
(
    NumOfAxles INTEGER NULL ,
    Tonnage INTEGER NULL ,
    VehicleID INTEGER NOT NULL
);

ALTER TABLE TRUCK
    ADD PRIMARY KEY (VehicleID);

CREATE TABLE VEHICLE
(
    VehicleID INTEGER NOT NULL ,
    Price NUMBER(8,2) NULL ,
    LicensePlateNo VARCHAR2(20) NULL
);

ALTER TABLE VEHICLE
    ADD PRIMARY KEY (VehicleID);
```



```
ALTER TABLE CAR
  ADD ( FOREIGN KEY (VehicleID) REFERENCES VEHICLE(VehicleID));
```

```
ALTER TABLE TRUCK
  ADD ( FOREIGN KEY (VehicleID) REFERENCES VEHICLE(VehicleID));
```

Exercises

ER Modeling Problems

1. Consider the *university* database described in Exercise 7.16 of the Elmasri/Navathe text. Enter the ER schema for this database using a data-modeling tool such as ERWin.
2. Consider a *mail order* database in which employees take orders for parts from customers. The data requirements are summarized as follows:
 - The mail order company has employees identified by a unique employee number, their first and last names, and a zip code where they are located.
 - Customers of the company are uniquely identified by a customer number. In addition, their first and last names and a zip code where they are located are recorded.
 - The parts being sold by the company are identified by a unique part number. In addition, a part name, their price, and quantity in stock are recorded.
 - Orders placed by customers are taken by employees and are given a unique order number. Each order may contain certain quantities of one or more parts and their received date as well as a shipped date is recorded.

Design an Entity-Relationship diagram for the mail order database and enter the design using a data-modeling tool such as ERWin.

3. Consider a *movie* database in which data is recorded about the movie industry. The data requirements are summarized as follows:
 - Movies are identified by their title and year of release. They have a length in minutes. They also have a studio that produces the movie and are classified under one or more genres (such as horror, action, drama etc). Movies are directed by one or more directors and have one or more actors acting in them. The movie also has a plot outline. Each movie also has zero or more quotable quotes that are spoken by a particular actor acting in the movie.
 - Actors are identified by their names and date of birth and act in one or more movies. Each actor has a role in the movie.
 - Directors are also identified by their names and date of birth and direct one or more movies. It is possible for a director to act in a movie (not necessarily in a movie they direct).

- Studios are identified by their names and have an address. They produce one or more movies.

Design an Entity-Relationship diagram for the movie order database and enter the design using a data-modeling tool such as ERWin.

4. Consider a *conference review* system database in which researchers submit their research papers for consideration. The database system also caters to reviewers of papers who make recommendations on whether to accept or reject the paper. The data requirements are summarized as follows:

Authors of papers are uniquely identified by their email id. Their first and last names are also recorded.

- Papers are assigned unique identifiers by the system and are described by a title, an abstract, and a file name containing the actual paper.
- Papers may have multiple authors, but one of the authors is designated as the contact author.
- Reviewers of papers are uniquely identified by their email id. Their first and last names are also recorded.
- Each paper is assigned between two and four reviewers. The reviewer rates the paper assigned to him on a scale of 1 to 10.
- Each review contains two types of written comments: one to be seen by the review committee only and the other by the author(s) as well.

Design an Entity-Relationship diagram for the conference review database and enter the design using a data-modeling tool such as ERWin.

5. Consider the ER diagram for the AIRLINE database shown in Figure 7.20 of the Elmasri/Navathe text. Enter this design using a data-modeling tool such as ERWin.

Enhanced ER Modeling Problems

6. Consider a *grade book* database in which instructors within an academic department maintain scores/points obtained by individual students in their classes. The data requirements are summarized as follows:

- Students are identified by a unique student id, their first and last names, and an email address.
- The instructor teaches certain courses each term. The courses are uniquely identified by a course number, a section number, and the term in which they are taught. The instructor also assigns grade cutoffs (example 90, 80, 70, and 60) for letter grades A, B, C, D, and F for each course he or she teaches.
- Students are enrolled in courses taught by the instructor.
- Each course being taught by the instructor has a number of grading components (such as mid-term, final exam, project, etc.). Each grading component has a maximum number of points (such as 100 or 50) and a weight (such as 20% or 10%). The weights of all the grading components of a course usually add up to 100.

- Finally, the instructor records the points earned by each student in each of the grading components in each of the courses. For example, student with id=1234 earns 84 points for the grading component mid-term for the course CSc 2310 section 2 in the fall 2005 term. The mid-term grading component may have been defined to have a maximum of 100 points and a weight of 20% of the course grade.

Design an enhanced Entity-Relationship diagram for the grade book database and enter the design using a data-modeling tool such as ERWin.

7. Consider an *online auction* database system in which members (buyers and sellers) participate in the sale of items. The data requirements for this system are summarized as follows:

- The online site has members who are identified by a unique member id and are described by an email address, their name, a password, their home address, and a phone number.
- A member may be a buyer or a seller. A buyer has a shipping address recorded in the database. A seller has a bank account number and routing number recorded in the database.
- Items are placed by a seller for sale and are identified by a unique item number assigned by the system. Items are also described by an item title, an item description, a starting bid price, bidding increment, the start date of the auction, and the end date of the auction.
- Items are also categorized based on a fixed classification hierarchy (for example a modem may be classified as /COMPUTER/HARDWARE/MODEM).
- Buyers make bids for items they are interested in. A bidding price and time of bid placement is recorded. The person at the end of the auction with the highest bid price is declared the winner and a transaction between the buyer and the seller may proceed soon after.
- Buyers and sellers may place feedback ratings on the purchase or sale of an item. The feedback contains a rating between 1 and 10 and a comment. Note that the rating is placed by the buyer or seller involved in the completed transaction.

Design an Entity-Relationship diagram for the auction database and enter the design using a data-modeling tool such as ERWin.

8. Consider a database system for a baseball organization such as the major leagues. The data requirements are summarized as follows:

- The personnel involved in the league include players, coaches, managers, and umpires. Each is identified by a unique personnel id. They are also described by their first and last names along with the date and place of birth.
- Players are further described by other attributes such as their batting orientation (left, right, or switch) and have a lifetime batting average (BA).
- Within the players group is a subset of players called pitchers. Pitchers have a lifetime ERA (earned run average) associated with them.

- Teams are uniquely identified by their names. Teams are also described by the city in which they are located and the division and league in which they play (such as Central division of the American league).
- Teams have one manager, a number of coaches, and a number of players.
- Games are played between two teams with one designated as the home team and the other the visiting team on a particular date. The score (runs, hits, and errors) are recorded for each team. The team with more number of runs is declared the winner of the game.
- With each finished game, a winning pitcher and a losing pitcher are recorded. In case there is a save awarded, the save pitcher is also recorded.
- With each finished game, the number of hits (singles, doubles, triples, and home runs) obtained by each player is also recorded.

Design an Entity-Relationship diagram for the baseball database and enter the design using a data-modeling tool such as ERWin.

9. Consider the ER diagram for the university database shown in Figure 8.9 of the Elmasri/Navathe text. Enter this design using a data-modeling tool such as ERWin.
10. Consider the ER diagram for the small airport database shown in Figure 8.12 of the Elmasri/Navathe text. Enter this design using a data-modeling tool such as ERWin.

CHAPTER 2

Abstract Query Languages

This chapter introduces Java-based interpreters for three abstract query languages: Relational Algebra (RA), Domain Relational Calculus (DRC), and Datalog. The interpreters have been implemented using the parser generator tools JCup and JFlex. In order to use these interpreters, one needs to only download two jar files: `dbengine.jar` and `aql.jar` and include them in the Java CLASSPATH. The JCup libraries are included as part of the jar files and hence the only other software that is required to use the interpreters is a standard Java environment.

The system is simple to use and comes with a database engine that implements a set of basic relational algebraic operators. The interpreter reads a query from the terminal and performs the following three steps:

- (1) **Syntax Check:** The query is checked for any syntax errors. If there are any syntactic errors, the interpreter reports these to the terminal and waits to read another query; otherwise the interpreter proceeds to the second step.
- (2) **Semantics Check:** The syntactically correct query is checked for semantic errors including type mismatches, invalid column references, and invalid relation names. In addition, the DRC and Datalog interpreters check the queries for safety. If there are any semantic errors or if the DRC/Datalog query is unsafe, the interpreter reports these to the terminal and waits to read another query; otherwise the interpreter proceeds to the third step.
- (3) **Query Evaluation:** The query is evaluated using the primitives provided by the database engine and the results are displayed.

2.1 Creating the Database

Before the user can start using the interpreters, they must create a database against which they will submit queries. The database consists of several text files all stored within a directory. The directory is named after the database name. For example, to create a database identified with the name `db1` and containing two tables:

```
student(sid:integer, sname:vchar, phone:vchar, gpa:decimal)
skills(sid:integer, language:vchar)
```

a directory called `db1` should be created along with the following three files (one for the catalog description and the remaining two for the data for the two tables):

```
catalog.dat
STUDENT.dat
SKILLS.dat
```

The file names are case sensitive and should strictly follow the convention used, i.e. `catalog.dat` should be all lower case and the data files should be named after their relation name in upper case followed by the file suffix, `.dat`, in lower case.

The `catalog.dat` file contains the number of relations in the first line followed by the descriptions of each relation. The description of each relation begins with the name of the relation in a separate line followed by the number of attributes in a separate line followed by attribute descriptions. Each attribute description includes the name of the attribute in a separate line followed by the data type (`VARCHAR`, `INTEGER`, or `DECIMAL`) in a separate line. All names and data types are in upper case. There should be no leading or trailing white space in any of the lines. The `catalog.dat` file for database `db1` is shown below:

```
2
STUDENT
4
SID
INTEGER
SNAME
VARCHAR
PHONE
VARCHAR
GPA
DECIMAL
SKILLS
2
SID
INTEGER
LANGUAGE
VARCHAR
```

The `db1` directory must include one data file for each relation. In the case of `db1`, they should be named `STUDENT.dat` and `SKILLS.dat`. The data file for relations contains the number of tuples in the first line followed by the description of each tuple. Tuples are described by the values under each column with each value in a separate line. For example, let the `SKILLS` relation have three tuples:

```
(111, Java)
(111, C++)
(222, Java)
```

These tuples will be represented in the `SKILLS.dat` data file as follows:

```
3
111
Java
```

111
C++
222
Java

Again, there should be no leading or trailing white spaces in any of the lines. Some pre-defined databases are available along with this laboratory manual. New data may be added to existing databases as well as new databases may be created when needed.

2.2 Relational Algebra Interpreter

The RA interpreter is invoked using the following terminal command:

```
$ java edu.gsu.cs.ra.RA company
```

Here \$ is the command prompt and `company` is the name of the database (as well as the name of the directory where the database files are stored). This command assumes that the `company` directory is present in the same directory where this command is issued. Of course, one can issue this command in a different directory by providing the full path to the database directory.

The interpreter responds with the following prompt:

```
RA>
```

At this prompt the user may enter a Relational Algebra query or type the exit command. Every query is terminated by a “;”. Even the exit command must end with a semi-colon. Queries may span more than one line; upon typing the ENTER key the interpreter prints the RA> prompt and waits for further input unless the ENTER key is typed after a semi-colon, in which case the query is processed by the interpreter.

2.2.1 Relational Algebra Syntax

A subset of Relational Algebra that includes the union, minus, intersect, Cartesian product, natural join, select, project, and rename operators is implemented in the interpreter. The context-free grammar for this subset is shown below:

```
<Query> ::= <Expr> SEMI;
<Expr>  ::= <ProjExpr>   | <RenameExpr>       | <UnionExpr> |
              <MinusExpr> | <IntersectExpr>  | <JoinExpr> |
              <TimesExpr> | <SelectExpr>      | RELATION
<ProjExpr>  ::= PROJECT [<AttrList>] (<Expr>)
<RenameExpr> ::= RENAME [<AttrList>] (<Expr>)
<AttrList>  ::= ATTRIBUTE | <AttrList> , ATTRIBUTE
<UnionExpr> ::= (<Expr> UNION <Expr>)
<MinusExpr> ::= (<Expr> MINUS <Expr>)
<IntersectExpr> ::= (<Expr> INTERSECT <Expr>)
```

```

<JoinExpr>      ::= (<Expr> JOIN <Expr>)
<TimesExpr>     ::= (<Expr> TIMES <Expr>)
<SelectExpr>    ::= SELECT [<Condition>](<Expr>)
<Condition>     ::= <SimpleCondition> |
                   <SimpleCondition> AND <Condition>
<SimpleCondition> ::= <Operand> <Comparison> <Operand>
<Operand>       ::= ATTRIBUTE | STRING-CONST | NUMBER-CONST
<Comparison>    ::= < | <= | = | <> | > | >=

```

The terminal strings in the grammar include

- Keywords for the relational algebraic operators: PROJECT, RENAME, UNION, MINUS, INTERSECT, JOIN, TIMES, and SELECT. These keywords are case-insensitive.
- Logical keyword AND (case-insensitive).
- Miscellaneous syntactic character strings such as (,), <, <=, =, <>, >, >=, ,, and comma (,).
- Name strings: RELATION and ATTRIBUTE (case-insensitive names of relations and their attributes).
- Constant strings: STRING-CONST (a string enclosed within single quotes; e.g. 'Thomas') and NUMBER-CONST (integer as well as decimal numbers; e.g. 232 and -36.1).

An example of a well-formed syntactically correct query for the company database of the Elmasri/Navathe text is:

```

( project[ssn] (select[lname=' Jones' ] (employee))
  union
  project[superssn] (select[dno=5] (employee))
);

```

All relational algebra queries must be terminated by a “;”.

A relational algebra query in the simplest form is a “relation name”. For example the following terminal session with the interpreter illustrates the execution of this simple query form:

```

$ java edu.gsu.cs.ra.RA company
RA> departments;
SEMANTIC ERROR in RA Query: Relation DEPARTMENTS does not exist
RA> department;
DEPARTMENT (DNAME:VARCHAR, DNUMBER:INTEGER, MGRSSN:VARCHAR, MGRSTARTDATE:VARCHAR)

```

```

Number of tuples = 6
Research:5:333445555:22-MAY-1978:
Administration:4:987654321:01-JAN-1985:
Headquarters:1:888665555:19-JUN-1971:
Software:6:111111100:15-MAY-1999:
Hardware:7:444444400:15-MAY-1998:

```



```
Sales:8:555555500:01-JAN-1997:
```

```
RA> exit;
$
```

In response to a query, the interpreter displays the schema of the result followed by the answer to the query. Individual values within a tuple are terminated by a “:”. The simplest query form is useful to display the database contents.

More complicated relational algebra queries involve one or more applications of one or more of the several operators such as select, project, times, join, union, etc. For example, consider the query “*Retrieve the names of all employees working for Dept. No. 5*”. This would be expressed by the query execution in the following RA session:

```
RA> project[fname,lname](select[dno=5](employee));
temp1 (FNAME:VARCHAR, LNAME:VARCHAR)
```

```
Number of tuples = 4
Franklin:Wong:
John:Smith:
Ramesh:Narayan:
Joyce:English:
```

```
RA>
```

2.2.2 Naming of Intermediate Relations and Attributes

The RA interpreter assigns temporary relation names such as temp0, temp1, etc. to each intermediate relation encountered in the execution of the entire query. The RA interpreter also employs the following rules as far as naming of attributes/columns of intermediate relations:

1. Union, Minus, and Intersect: The attribute/column names from the left operand are used to name the attributes of the output relation.
2. Times (Cartesian Product): Attribute/Column names from both operands are used to name the attributes of the output relation. Attribute/Column names that are common to both operands are prefixed by relation name (tempN).
3. Select: The attribute names of the output relation are the same as the attribute/column names of the operand.
4. Project, Rename: Attribute/Column names present in the attribute list parameter of the operator are used to name the attributes of the output relation. Duplicate attribute/column names are not allowed in the attribute list.
5. Join (Natural Join): Attribute/Column names from both operands are used to name the attributes of the output relation. Common attribute/column names appear only once.

As another example, consider the query “*Retrieve the social security numbers of employees who either work in department 5 or directly supervise an employee who works in department 5*”. The query is illustrated in the following RA session:

```
RA> (project[ssn] (select[dno=5] (employee)))
RA> union project[superSSN] (select[dno=5] (employee))) ;
temp4 (SSN:VARCHAR)
```

```
Number of tuples = 5
333445555:
123456789:
666884444:
453453453:
888665555:
```

```
RA>
```

2.2.3 Relational Algebraic Operators Supported by the RA Interpreter

Select: As can be noted from the grammar, the select operator supported by the interpreter has the following syntax:

```
select[condition] (expression)
```

where `condition` is a conjunction of one or more simple conditions involving comparisons of attributes or constants with other attributes or constants. The attributes used in the condition must be present in the attributes of the relation corresponding to `expression`.

Project: The project operator supported by the interpreter has the following syntax:

```
project[attribute-list] (expression)
```

where `attribute-list` is a comma separated list of attributes, each of which is present in the attributes of the relation corresponding to `expression`.

Rename: The syntax for the rename operator is

```
rename[attribute-list] (expression)
```

where `attribute-list` is a comma separated list of attribute names. The number of attributes mentioned in the list must be equal to the number of attributes of the relation corresponding to `expression`.

Join: The syntax for the join operator is

```
(expression1 join expression2)
```

There is no restriction on the schemas of the two expressions.

Times: The syntax for the times operator is

```
(expression1 times expression2)
```

There is no restriction on the schemas of the two expressions.

Union: The syntax for the union operator is

```
(expression1 union expression2)
```

The schemas of the two expressions must be compatible (same number of attributes and same data types; the names of the attributes may be different).

Minus: The syntax for the minus operator is

```
(expression1 minus expression2)
```

The schemas of the two expressions must be compatible (same number of attributes and same data types; the names of the attributes may be different).

Intersect: The syntax for the intersect operator is

```
(expression1 intersect expression2)
```

The schemas of the two expressions must be compatible (same number of attributes and same data types; the names of the attributes may be different).

2.2.4 Examples

The queries from Section 6.5 of the Elmasri/Navathe text modified to work with the RA interpreter are shown below:

Query 1: Retrieve the name and address of employees who work for the "Research" department.

```
project[fname,lname,address] (
  (rename[dname,dno,mgrssn,mgrstartdate] (
    select[dname='Research'] (department))
  join
  employee
)
);
```

Query 2: For every project located in "Stafford", list the project number, the controlling department number, and the department manager's last name, address, and birth date.

```
project[pnumber,dnum,lname,address,bdate] (
  (
    (select[plocation='Stafford'] (projects)
    join
    rename[dname,dnum,ssn,mgrstartdate] (department)
  )
  join employee
)
);
```

Query 3: Find the names of employees who work on all the projects controlled by department number 5.

```
project[lname,fname] (
  (employee
  join
  (project[ssn] (employee)
  minus
  project[ssn] (
    (
      (project[ssn] (employee)
      times
      project[pnumber] (select[dnum=5] (projects))
    )
    minus
    rename[ssn,pnumber] (project[essn,pno] (works_on))
  )
  )
  )
  )
);
```

Query 4: Make a list of project numbers for projects that involve an employee whose last name is "Smith", either as a worker or as a manager of the department that controls the project.

```
( project[pno] (
  (rename[essn] (project[ssn] (select[lname='Smith'] (employee)))
  join
  works_on
  )
  )
union
project[pnumber] (
  ( rename[dnum] (project[dnumber] (select[lname='Smith'] (
```

```

        (employee
        join
        rename[dname,dnumber,ssn,mgrstartdate] (department)
        )
    )
    join
    projects
)
);

```

Query 5: List the names of all employees with two or more dependents.

```

project[lname,fname] (
  (rename[ssn] (
    project[essn] (
      select[essn1=essn2 and dname1<>dname2] (
        (rename[essn1,dname1] (project[essn,dependent_name] (dependent))
        times
        rename[essn2,dname2] (project[essn,dependent_name] (dependent)))
      )
    )
  )
  join
  employee)
);

```

Query 6: Retrieve the names of employees who have no dependents.

```

project[lname,fname] (
  ( ( project[ssn] (employee)
    minus project[essn] (dependent)
  )
  join
  employee
)
);

```

Query 7: List the names of managers who have at least one dependent.

```

project[lname,fname] (
  ((rename[ssn] (project[mgrssn] (department))
  join
  rename[ssn] (project[essn] (dependent))
  )
);

```

```

join
employee
)
);

```

Important Tip: Since many of the queries shown above are long and span multiple lines, the best way to use the interpreter is to create a text file in which the queries are typed. These queries are then cut and pasted into the interpreter prompt. Any errors in syntax or semantics should be corrected in the text file and then the process of cut and paste should be repeated until a correct solution is reached.

2.3 Domain Relational Calculus Interpreter

The DRC interpreter is invoked using the following terminal command:

```
$ java edu.gsu.cs.drc.DRC company
```

Here \$ is the command prompt and `company` is the name of the database (as well as the name of the directory where the database files are stored). This command assumes that the `company` directory is present in the same directory where this command is issued. Of course, one can issue this command in a different directory by providing the full path to the database directory.

The interpreter responds with the following prompt:

```
DRC>
```

At this prompt the user may enter a Domain Relational Calculus query or type the `exit` command. Each DRC query is expressed in a set-notation using a pair of curly brackets as follows:

```
{ variable-list | P(variable-list) }
```

where `variable-list` is a comma-separated list of variables which must all be present in the body predicate of the query `P(variable-list)` as free-variables.

The `exit` command must end with a semi-colon. Queries may span more than one line; upon typing the ENTER key the interpreter prints the `DRC>` prompt and waits for further input unless the ENTER key is typed after a right curly bracket (`}`), in which case the query is processed by the interpreter.

2.3.1 Domain Relational Calculus Syntax

The context-free grammar for DRC queries implemented within the DRC interpreter is shown below:

```
Query ::= LBRACE VarList BAR Formula RBRACE;
```

```

VarList ::= NAME | VarList COMMA NAME;
Formula ::= AtomicFormula |
           Formula AND Formula |
           Formula OR Formula |
           NOT LPAREN Formula RPAREN |
           LPAREN EXISTS VarList RPAREN LPAREN Formula RPAREN |
           LPAREN FORALL VarList RPAREN LPAREN Formula RPAREN;
AtomicFormula ::=
           NAME LPAREN ArgList RPAREN | Arg Comparison Arg;
ArgList ::= Arg | ArgList COMMA Arg;
Arg ::= NAME | STRING | NUMBER;
Comparison ::= < | <= | = | <> | > | >=

```

The terminal strings in the grammar include

- Keywords for the logical operators: AND, OR, and NOT. These keywords are case-insensitive.
- Quantifier keywords EXISTS and FORALL (case-insensitive).
- Miscellaneous syntactic character strings such as (,), <, <=, =, <>, >, >=, and comma (,).
- NAME strings: used for named relations and variables (case-insensitive).
- Constant strings: STRING (a string enclosed within single quotes; e.g. 'Thomas') and NUMBER (integer as well as decimal numbers; e.g. 232 and -36.1).

An example of a well-formed syntactically correct query on the company database of the Elmasri/Navathe text is:

```

{ x | (exists a1,a2,a3,a4,a5,a6,a7,a8) (
        employee(a1,a2,'Jones',x,a3,a4,a5,a6,a7,a8)) or
      (exists a1,a2,a3,a4,a5,a6,a7,a8) (
        employee(a1,a2,a3,x,a4,a5,a6,a7,a8,5)) }

```

All DRC queries must be enclosed within a pair of matching curly brackets.

The simplest DRC query displays the contents of a relation. For example the following terminal session with the interpreter illustrates the execution of this simple query form that displays the contents of the DEPARTMENT relation:

```

$ java edu.gsu.cs.drc.DRC company
DRC> { a,b,c,d | department(a,b,c,d) }
ANSWER (A:VARCHAR,B:INTEGER,C:VARCHAR,D:VARCHAR)

```

```

Number of tuples = 6
Research:5:333445555:22-MAY-1978:
Administration:4:987654321:01-JAN-1985:
Headquarters:1:888665555:19-JUN-1971:
Software:6:111111100:15-MAY-1999:

```

Hardware:7:444444400:15-MAY-1998:
 Sales:8:555555500:01-JAN-1997:

DRC>

In response to a query, the interpreter displays the schema of the result followed by the answer to the query.

2.3.2 Safe DRC Queries

The DRC interpreter checks for the “safety” of queries and evaluates only those that are determined to be safe. An error message is generated for unsafe queries.

For the discussion of safe DRC queries, we will assume that the formula defining the query does not contain the `forall` quantifier. If the `forall` quantifier does appear in the formula, the user can convert such a formula to an equivalent one without the `forall` quantifier using the logical equivalence:

$$(\text{forall } X) (F) \equiv \text{NOT } ((\text{exists } X) (\text{NOT } (F)))$$

It is almost always the case that the `F` in the `forall` quantified formula above is of the form

$$\text{NOT } (P) \text{ or } Q$$

In case the user does not eliminate the `forall` quantifier, the DRC interpreter would automatically convert all `forall` quantified formulas into equivalent `exists` quantified formulas using the above equivalence. In addition, the interpreter would also apply the DeMorgan’s law:

$$\text{NOT } (P \text{ or } Q) \equiv \text{NOT } (P) \text{ and } \text{NOT } (Q)$$

to push the `NOT` further inside the formula.

As an example of this automatic transformation, consider the following query provided by the user:

```
{a,b | (exists c) (r(a,b,c) and
              (forall d,e) (not(s(a,d,e)) or (exists f) (t(d,f)))) }
```

The DRC interpreter would convert the above query to:

```
{a,b | (exists c) (r(a,b,c) and
              not(exists d,e) (s(a,d,e) and not(exists f) (t(d,f)))) }
```


Definition: A DRC query (without `forall` quantifiers) is defined to be *safe* if it satisfies the following three conditions:

- (a) For every sub-formula in the query connected with an “or”, the two operand formulas have the same set of free variables, i.e. the “or” formula is of the form:

$$F(X_1, \dots, X_n) \text{ or } G(X_1, \dots, X_n)$$

- (b) All free variables appearing in “maximal sub-conjuncts”, F_1 and ... and F_n , must be “limited” in that they either appear in (i) a positive sub-formula F_i or (ii) as X in an sub-formula of the form $X=a$ or $a=X$ or (iii) as X in a sub-formula of the form $X=Y$ where Y is determined to be “limited”.
- (c) The NOT operator may be applied only to a term in a maximal sub-conjunct of type discussed in (b), i.e. all free variables in the NOT term must be shown to be “limited” in the positive terms of the maximal sub-conjunct.

Some examples follow. The following query would be considered safe as it satisfies condition (a).

$$\{a, b \mid (\text{exists } c) (r(a, b, c)) \text{ or } s(a, b) \}$$

But the following would not be safe:

$$\{a, b \mid (\text{exists } b, c) (r(a, b, c)) \text{ or } s(a, b) \}$$

This is because the free variables on the left operand of the “or” formula consists of only one variable, a , and the free variables on the right operand consists of two variables, a and b .

The query formula from an earlier query:

$$(\text{exists } c) (r(a, b, c) \text{ and } \text{not}(\text{exists } d, e) (s(a, d, e) \text{ and } \text{not}(\text{exists } f) (t(d, f))))$$

is safe. The formula has the following two maximal sub-conjuncts (ignoring atomic formulas which are maximal sub-conjuncts of size 1):

- (1) $s(a, d, e) \text{ and } \text{not}(\text{exists } f) (t(d, f))$
all three free variables a, d , and e are limited as they appear in $s(a, d, e)$
- (2) $r(a, b, c) \text{ and } \text{not}(\text{exists } d, e) (s(a, d, e) \text{ and } \text{not}(\text{exists } f) (t(d, f)))$
all three free variables a, b , and c are limited as they appear in $r(a, b, c)$.

The free variables in each of the maximal sub-conjuncts are shown to be “limited” and hence the overall query is safe.

The following query formula is unsafe:

$$p(a,b) \text{ and not } ((\text{exists } c) (q(b,c,d)))$$

This is because the free variable d is not “limited” as it is not grounded in a positive term in the maximal sub-conjunct.

2.3.3 DRC Query Examples

The queries from Section 6.7 of the Elmasri/Navathe text modified to work with DRC interpreter are shown below:

Query 0: Retrieve the birthdate and address of the employees whose name is "John B. Smith".

```
{ u,v | (exists t,w,x,y,z) (
    employee('John', 'B', 'Smith', t,u,v,w,x,y,z) ) }
```

Query 1: Retrieve the name and address of all employees who work for the "Research" department.

```
{ q,s,v | (exists r,t,u,w,x,y,z,n,o) (
    employee(q,r,s,t,u,v,w,x,y,z) and
    department('Research',z,n,o) ) }
```

Query 2: For every project located in "Stafford", list the project number, the controlling department number, and the department manager's last name, birth date, and address.

```
{ i,k,s,u,v | (exists h,q,r,t,w,x,y,z,l,o) (
    projects(h,i,'Stafford',k) and
    employee(q,r,s,t,u,v,w,x,y,z) and
    department(l,k,t,o) ) }
```

Query 6: List the names of employees who have no dependents.

```
{ q,s | (exists r,t,u,v,w,x,y,z) (
    employee(q,r,s,t,u,v,w,x,y,z) and
    not ((exists m,n,o,p) (dependent(t,m,n,o,p))) ) }
```

The following is not SAFE and would not work

```
{ q,s | (exists r,t,u,v,w,x,y,z) (
    employee(q,r,s,t,u,v,w,x,y,z) and
    (forall l,m,n,o,p) (not (dependent(l,m,n,o,p)) or t<>l) ) }
```

Query 7: List the names of managers who have at least one dependent.

```
{ s,q | (exists r,t,u,v,w,x,y,z,h,i,k,m,n,o,p) (
    employee(q,r,s,t,u,v,w,x,y,z) and
    department(h,i,t,k) and
    dependent(t,m,n,o,p) ) }
```

2.4 Datalog Interpreter

The DLOG interpreter is invoked using the following terminal command:

```
$ java edu.gsu.cs.dlg.DLOG company
```

Here \$ is the command prompt and `company` is the name of the database (as well as the name of the directory where the database files are stored). This command assumes that the `company` directory is present in the same directory where this command is issued. Of course, one can issue this command in a different directory by providing the full path to the database directory.

The interpreter responds with the following prompt:

```
DLOG>
```

At this prompt the user may enter the query execution command `@file-name` or type the `exit` command, where `file-name` contains the Datalog query. Each command is to be terminated by a semi-colon. Even the `exit` command must end with a semi-colon.

2.4.1 Datalog Syntax

Datalog is a rule-based logical query language for relational databases. The syntax of Datalog is defined below:

An *atomic formula* is of one of the following two forms:

1. $p(x_1, \dots, x_n)$ where p is a relation name and x_1, \dots, x_n are either constants or variables, or
2. $x <op> y$ where x and y are either constants or variables and $<op>$ is one of the six comparison operators: $<, <=, >, >=, =, !=$.

A *Datalog rule* is of the form:

$$p :- q_1, \dots, q_n.$$

Here p is an atomic formula and q_1, \dots, q_n are either atomic formulas or negated atomic formulas (i.e. atomic formula preceded by `not`). p is referred to as the head of the rule, and q_1, \dots, q_n are referred to as sub-goals.

A Datalog rule $p \text{ :- } q_1, \dots, q_n$ is said to be *safe* if

1. Every variable that occurs in a negated sub-goal also appears in a positive sub-goal, and
2. Every variable that appears in the head of the rule also appears in the body of the rule.

A *Datalog query* is set of safe Datalog rules with at least one rule having the `answer` predicate in the head. The `answer` predicate collects all answers to the query.

Note: Variables that appear only once in a rule can be replaced by anonymous variables (represented by underscores). Every anonymous variable is different from all other variables.

2.4.2 Datalog Query Examples

The following are examples of Datalog queries against the company database:

Query 1: Get names of all employees in department 5 who work more than 10 hours/week on the ProductX project.

```
answer(F,M,L) :-
    employee(F,M,L,S,_,_,_,_,5),
    works_on(S,P,H),
    projects('ProductX',P,_,_),
    H >= 10.
```

Query 2: Get names of all employees who have a dependent with the same first name as their own first names.

```
answer(F,M,L) :-
    employee(F,M,L,S,_,_,_,_,_),
    dependent(S,F,_,_,_).
```

Query 3: Get the names of all employees who are directly supervised by Franklin Wong.

```
answer(F,M,L) :-
    employee(F,M,L,_,_,_,_,_,S,_),
    employee('Franklin',_, 'Wong', S,_,_,_,_,_,_).
```

Query 4: Get the names of all employees who work on every project.

```
temp1(S,P) :-
    employee(_,_,_,S,_,_,_,_,_),
    projects(_,P,_,_).
temp2(S,P) :-
    works_on(S,P,_).
temp3(S) :-
    temp1(S,P), not temp2(S,P).
answer(F,M,L) :-
```

```
employee(F,M,L,S,_,_,_,_,_,_) , not temp3(S) .
```

In this query, temp1 (S, P) collects all combinations of employees, S, and projects, P; temp2 (S, P) collects only those pairs where employee S works on project P; temp3 (S) collects employees, S, who do not work for a particular project (these employees should not be in the answer). A second negation in the final rule gets the answers to the query.

Query 5: Get the names of employees who do not work on any project.

```
temp1(S) :-
    works_on(S,_,_) .
answer(F,M,L) :-
    employee(F,M,L,S,_,_,_,_,_,_) , not temp1(S) .
```

Query 6: Get the names and addresses of employees who work for at least one project located in Houston but whose department does not have a location in Houston.

```
temp1(S) :-
    works_on(S,P,_) , project(_,P,'Houston',_) .
temp2(S) :-
    employee(_,_,_,S,_,_,_,_,_,D) ,
    not dept_locations(D,'Houston') .
answer(F,M,L,A) :-
    employee(F,M,L,S,_,A,_,_,_,_) , temp1(S) , temp2(S) .
```

temp1 (S) collects employee S who work for a project located in Houston; temp2 (S) collects employees S whose department do not have a location in Houston; the final rule intersects the two temp predicates to get the answer to the query.

Query 7: Get the names and addresses of employees who work for at least one project located in Houston or whose department does not have a location in Houston. (Note: this is a slight variation of the previous query with 'but' replaced by 'or').

```
temp1(S) :-
    works_on(S,P,_) ,
    project(_,P,'Houston',_) .
temp2(S) :-
    employee(_,_,_,S,_,_,_,_,_,D) ,
    not dept_locations(D,'Houston') .
answer(F,M,L,A) :-
    employee(F,M,L,S,_,A,_,_,_,_) , temp1(S) .
answer(F,M,L,A) :-
    employee(F,M,L,S,_,A,_,_,_,_) , temp2(S) .
```

Query 8: Get the last names of all department managers who have no dependents.

```

templ(S) :-
    dependent(S,_,_,_,_).
answer(L) :-
    employee(_,_,L,S,_,_,_,_,_),
    department(_,_,S,_),
    not templ(S).

```

To execute the above queries using the Datalog interpreter, each must be placed in a separate file with a \$ symbol appearing at the end of the file. Assume that the queries are placed in files named q1, q2, ..., q8. The following is a terminal session showing the execution of the above queries:

```

[raj@tinman ch2]$ java edu.gsu.cs.dlg.DLOG company
type "help;" for usage...
Message: Database Provided: Database Directory is ./company
DLOG> @q1;
-----
answer(F,M,L) :-
    employee(F,M,L,S,_,_,_,_,_,5),
    works_on(S,P,H), H >= 10,
    projects('ProductX',P,_,_).$
-----
ANSWER(F:VARCHAR,M:VARCHAR,L:VARCHAR)

Number of tuples = 2
John:B:Smith:
Joyce:A:English:

DLOG> exit;
Exiting...

```

```

[raj@tinman ch2]$ java edu.gsu.cs.dlg.DLOG company
type "help;" for usage...
Message: Database Provided: Database Directory is ./company
DLOG> @q2;
-----
answer(F,M,L) :-
    employee(F,M,L,S,_,_,_,_,_,_),
    dependent(S,F,_,_,_).$
-----
ANSWER(F:VARCHAR,M:VARCHAR,L:VARCHAR)

Number of tuples = 1
Alec:C:Best:

DLOG> exit;
Exiting...

```

```
[raj@tinman ch2]$ java edu.gsu.cs.dlg.DLOG company
type "help;" for usage...
Message: Database Provided: Database Directory is ./company
DLOG> @q3;
```

```
-----
answer(F,M,L) :-
    employee(F,M,L,_,_,_,_,_,S,_),
    employee('Franklin',_, 'Wong',S,_,_,_,_,_).$
-----
ANSWER(F:VARCHAR,M:VARCHAR,L:VARCHAR)
```

```
Number of tuples = 3
John:B:Smith:
Ramesh:K:Narayan:
Joyce:A:English:
```

```
DLOG> exit;
Exiting...
```

```
[raj@tinman ch2]$ java edu.gsu.cs.dlg.DLOG company
type "help;" for usage...
Message: Database Provided: Database Directory is ./company
DLOG> @q4;
```

```
-----
temp1(S,P) :-
    employee(_,_,_,S,_,_,_,_,_),
    projects(_,P,_,_).
temp2(S,P) :-
    works_on(S,P,_).
temp3(S) :-
    temp1(S,P), not temp2(S,P).
answer(F,M,L) :-
    employee(F,M,L,S,_,_,_,_,_), not temp3(S).$
-----
ANSWER(F:VARCHAR,M:VARCHAR,L:VARCHAR)
```

```
Number of tuples = 0
```

```
DLOG> exit;
Exiting...
```

```
[raj@tinman ch2]$ java edu.gsu.cs.dlg.DLOG company
type "help;" for usage...
Message: Database Provided: Database Directory is ./company
DLOG> @q5;
```

```

temp1(S) :-
    works_on(S,_,_).
answer(F,M,L) :-
    employee(F,M,L,S,_,_,_,_,_), not temp1(S).$
-----
ANSWER(F:VARCHAR,M:VARCHAR,L:VARCHAR)

```

Number of tuples = 2
 Bob:B:Bender:
 Kate:W:King:

DLOG> exit;
 Exiting...

```

[raj@tinman ch2]$ java edu.gsu.cs.dlg.DLOG company
type "help;" for usage...
Message: Database Provided: Database Directory is ./company
DLOG> @q6;

```

```

-----
temp1(S) :-
    works_on(S,P,_), projects(_,P,'Houston',_).
temp2(S) :-
    employee(_,_,_,S,_,_,_,_,D),
    not dept_locations(D,'Houston').
answer(F,M,L,A) :-
    employee(F,M,L,S,_,A,_,_,_), temp1(S), temp2(S).$
-----
ANSWER(F:VARCHAR,M:VARCHAR,L:VARCHAR,A:VARCHAR)

```

Number of tuples = 1
 Jennifer:S:Wallace:291 Berry, Bellaire, TX:

DLOG> exit;
 Exiting...

```

[raj@tinman ch2]$ java edu.gsu.cs.dlg.DLOG company
type "help;" for usage...
Message: Database Provided: Database Directory is ./company
DLOG> @q7;

```

```

-----
temp1(S) :-
    works_on(S,P,_),
    projects(_,P,'Houston',_).
temp2(S) :-
    employee(_,_,_,S,_,_,_,_,D),
    not dept_locations(D,'Houston').
answer(F,M,L,A) :-
    employee(F,M,L,S,_,A,_,_,_), temp1(S).

```



```

answer(F,M,L,A) :-
    employee(F,M,L,S,_,A,_,_,_,_), temp2(S).$
-----
ANSWER(F:VARCHAR,M:VARCHAR,L:VARCHAR,A:VARCHAR)

```

```

Number of tuples = 38
James:E:Borg:450 Stone, Houston, TX:
Franklin:T:Wong:638 Voss, Houston, TX:
Jennifer:S:Wallace:291 Berry, Bellaire, TX:
Ramesh:K:Narayan:971 Fire Oak, Humble, TX:
Alicia:J:Zelaya:3321 Castle, Spring, TX:
Ahmad:V:Jabbar:980 Dallas, Houston, TX:
Jared:D:James:123 Peachtree, Atlanta, GA:
Alex:D:Freed:4333 Pillsbury, Milwaukee, WI:
John:C:James:7676 Bloomington, Sacramento, CA:
Jon:C:Jones:111 Allgood, Atlanta, GA:
Justin:null:Mark:2342 May, Atlanta, GA:
Brad:C:Knight:176 Main St., Atlanta, GA:
Evan:E:Wallis:134 Pelham, Milwaukee, WI:
Josh:U:Zell:266 McGrady, Milwaukee, WI:
Andy:C:Vile:1967 Jordan, Milwaukee, WI:
Tom:G:Brand:112 Third St, Milwaukee, WI:
Jenny:F:Vos:263 Mayberry, Milwaukee, WI:
Chris:A:Carter:565 Jordan, Milwaukee, WI:
Kim:C:Grace:6677 Mills Ave, Sacramento, CA:
Jeff:H:Chase:145 Bradbury, Sacramento, CA:
Bonnie:S:Bays:111 Hollow, Milwaukee, WI:
Alec:C:Best:233 Solid, Milwaukee, WI:
Sam:S:Snedden:987 Windy St, Milwaukee, WI:
Nandita:K:Ball:222 Howard, Sacramento, CA:
Bob:B:Bender:8794 Garfield, Chicago, IL:
Jill:J:Jarvis:6234 Lincoln, Chicago, IL:
Kate:W:King:1976 Boone Trace, Chicago, IL:
Lyle:G:Leslie:417 Hancock Ave, Chicago, IL:
Billie:J:King:556 Washington, Chicago, IL:
Jon:A:Kramer:1988 Windy Creek, Seattle, WA:
Ray:H:King:213 Delk Road, Seattle, WA:
Gerald:D:Small:122 Ball Street, Dallas, TX:
Arnold:A:Head:233 Spring St, Dallas, TX:
Helga:C:Pataki:101 Holyoke St, Dallas, TX:
Naveen:B:Drew:198 Elm St, Philadelphia, PA:
Carl:E:Reedy:213 Ball St, Philadelphia, PA:
Sammy:G:Hall:433 Main Street, Miami, FL:
Red:A:Bacher:196 Elm Street, Miami, FL:

```

```

DLOG> exit;
Exiting...

```

```

[raj@tinman ch2]$ java edu.gsu.cs.dlg.DLOG company
type "help;" for usage...
Message: Database Provided: Database Directory is ./company
DLOG> @q8;
-----
templ(S) :-
    dependent(S,_,_,_,_) .
answer(L) :-
    employee(_,_,L,S,_,_,_,_,_),
    department(_,_,S,_),
    not templ(S).$
-----
ANSWER(L:VARCHAR)

Number of tuples = 2
Borg:
James:

DLOG> exit;
Exiting...
[raj@tinman ch2]$

```

Exercises

1. Specify and execute the following queries using the RA interpreter on the COMPANY database schema.
 - a. Retrieve the names of all employees in department 5 who work more than 10 hours per week on the 'ProductX' project.
 - b. List the names of all employees who have a dependent with the same first name as themselves.
 - c. Find the names of employees who are directly supervised by 'Franklin Wong'.
 - d. Retrieve the names of employees who work on every project.
 - e. Retrieve the names of employees who do not work on any project.
 - f. Retrieve the names and addresses of all employees who work on at least one project located in Houston but whose department has no location in Houston.
 - g. Retrieve the last names of all department managers who have no dependents.
2. Consider the following MAILORDER relational schema describing the data for a mail order company:

```

parts(pno,pname,qoh,price,olevel)
customers(cno,cname,street,zip,phone)
employees(eno,ename,zip,hdate)
zipcodes(zip,city)

```

```
orders(ono, cno, eno, received, shipped)
odetails(ono, pno, qty)
```

The attribute names are self-explanatory. “qoh” stands for quantity on hand. Specify and execute the following queries using the RA interpreter on the MAILORDER database schema.

- a. Retrieve the names of parts that cost less than \$20.00.
 - b. Retrieve the names and cities of employees who have taken orders for parts costing more than \$50.00
 - c. Retrieve the pairs of customer number values of customers who live in the same zip code.
 - d. Retrieve the names of customers who have ordered parts only from employees living in the city of Wichita.
 - e. Retrieve the names of customers who have ordered all parts costing less than \$20.00.
 - f. Retrieve the names of customers who have not placed a single order.
 - g. Retrieve the names of customers who have placed exactly two orders.
3. Consider the following GRADEBOOK relational schema describing the data for a grade book of a particular instructor (Note: The attributes A, B, C, and D store grade cutoffs.)

```
catalog(cno, ctitle)
students(sid, fname, lname, minit)
courses(term, secno, cno, A, B, C, D)
enrolls(sid, term, secno)
```

Specify and execute the following queries using the RA interpreter on the GRADEBOOK database schema.

- a. Retrieve the names of students enrolled in the ‘Automata’ class in the term of Fall 1996.
 - b. Retrieve the SID values of students who have enrolled in CSc226 as well as CSc227.
 - c. Retrieve the SID values of students who have enrolled in CSc226 or CSc227.
 - d. Retrieve the names of students who have not enrolled in any class.
 - e. Retrieve the names of students who have enrolled in all courses in the `catalog` table.
4. Consider the database consisting of the following relations:

```
supplier(sno, sname)
part(pno, pname)
project(jno, jname)
supply(sno, pno, jno)
```

The database records information about suppliers, parts, and projects and includes a ternary relationship between suppliers, parts, and projects. This relationship is a many-many-many relationship. Specify and execute the following queries using the RA interpreter.

- a. Retrieve part numbers of parts that are supplied to exactly two projects.
 - b. Retrieve supplier names of suppliers who supply more than two parts to project 'J1'.
 - c. Retrieve part numbers of parts that are supplied by every supplier.
 - d. Retrieve project names of projects that are supplied only by suppliers 'S1'.
 - e. Retrieve supplier names of suppliers who supply at least two different parts each to at least two different projects.
5. Specify and execute the following queries for the database in Exercise 5.16 of the Elmasri/Navathe text using the RA interpreter.
 - a. Retrieve the names of students who have enrolled in a course that uses a textbook published by Addison Wesley.
 - b. Retrieve the course names of courses which have changed their textbook at least once.
 - c. Retrieve the names of departments that only adopt textbooks from Addison Wesley.
 - d. Retrieve the names of departments that have adopted all textbooks written by Navathe and published by Addison Wesley in their courses.
 - e. Retrieve the names of students who have never used a book (in a course) written by Navathe and published by Addison Wesley.
6. Repeat Exercises 1 through 5 in domain relational calculus (DRC) by using the DRC interpreter.
7. Repeat Exercises 1 through 5 in Datalog (DLOG) by using the DLOG interpreter.