A SUITE OF CASE STUDIES IN RELATIONAL DATABASE DESIGN

A SUITE OF CASE STUDIES IN RELATIONAL DATABASE DESIGN

By
WEIGUANG ZHANG

Computer Science

A Thesis

Submitted to the School of Graduate Studies

In Partial Fulfillment of the Requirements

For the Degree

Master of Science

McMaster University

MASTER OF SCIENCE (Jan, 2012) McMaster University

(Computer Science) Hamilton, Ontario

TITLE: A SUITE OF CASE STUDIES IN RELATIONAL DATABASE DESIGN

AUTHOR: WEIGUANG ZHANG

SUPERVISOR: Dr. Antoine Deza and Dr. Frantisek Franek

NUMBER OF PAGES: x & 107

Abstract

Typical relational database design examples in textbooks and undergraduate courses are small and do not provide any real opportunity to practice the design, they simply illustrate and illuminate the principles. On the other end of the spectrum are typical industrial databases whose designs are complex and extensive, and so not suitable as a project for a one term database course. The objective of this thesis is to design and develop a collection of ten projects that would be usable as term projects in relational database system design for a typical undergraduate database course. To this end a suite of ten case studies are presented. Each project is taken from its informal specification to a relational schema using entity-relationship modeling and its translation to relational model, to database schema, to implementation of the database, to interactive SQL querying of the installed database and finished with a simple application programmed in C using the installed database and accessing it via embedded SQL.

Acknowledgments

I would like to express my gratitude to all of those who made it possible to complete this thesis, in particular to my supervisors Dr. Antoine Deza and Dr. Frantisek Franek.

I appreciate the great aid and support from all the members of the Advanced Optimization Laboratory.

I would also like to thank my family for their understanding and continuous support.

Abbreviations

IBM:	International Business Machines
ACM:	Association of Computing Machinery
DBMS:	Database Management System
SQL:	Structured Query Language
ODBC:	Open Database Connectivity
JDBC:	Java Database Connectivity
ER:	Entity Relationship
PHP:	Personal Home Page
API:	Application Programming Interface
CLI:	Call Level Interface:
ESQL:	Embedded Structured Query Language
IE:	Information Engineering
IDEF1X:	Integrated Definition for Information Modeling

Terminologies and Symbols of ERwin IE Format

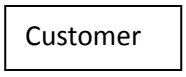
Identifying relationship: Shown with a solid line. An identifying relationship is a relationship between two entities in which the child entity is dependent on its associated parent entity, and the primary key of the parent entity is the part of the primary key of the child entity.

Non-identifying Relationship: Shown with a dashed line. A non-identifying relationship is a relationship between two entities in which the child entity is independent on its associated parent entity, and the primary key of the parent entity is the non-key attribute instead of the key attribute in the child entity.

Max relationship cardinality: Shown with a short perpendicular line across the relationship near its line end to signify "one" and with a "crow's foot" on the line end to signify "many".

Min relationship cardinality: Shown with a small circle near the end of the line to signify "zero" (participation in the relationship is optional) or with a short perpendicular line across the relationship line to signify "one" (participation in the relationship is mandatory).

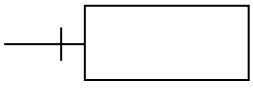
Dependent Entity:



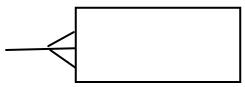
Independent Entity:

Zero:

One:



Many:



List of Figures

Figure 1: Summary of aspects of the Logical and Physical Models 7
Figure 2: The University System Logical Model 10
Figure 3: The University System Physical Model 11
Figure 4: The Airline Reservation Logical Model 27
Figure 5: The Airline Reservation Physical Model 28
Figure 6: The Movie Rental Logical Model 35
Figure 7: The Movie Rental Physical Model 36
Figure 8: The Car Rental Logic Model 44
Figure 9: The Car Rental Physical Model 45
Figure 10: The Course Registration Logical Model 52
Figure 11: The Course Registration Physical Model 53
Figure 12: The Emergency Room Logical Model 61
Figure 13: The Emergency Room Physical Model 62
Figure 14: The Property Rental Logical Model 70
Figure 15: The Property Rental Physical Model 71
Figure 16: The Software Project Logical Model 81
Figure 17: The Software Project Physical Model 82
Figure 18: The Tour Operator System Logical Model 89
Figure 19: The Tour Operator System Physical Model 90
Figure 20: The Warehouse System Logical Model 99
Figure 21: The Warehouse System Physical Model 100

Contents

Abstract iiii

Acknowledgments ivv

Abbreviations viv

Terminologies and Symbols of ERwin IE Format iv

List of Figures viiii

Chapter 1: Introduction 1

Chapter 2: Objectives, Decisions, and Methods 4

Chapter 3: University System 8

3.1 University System Informal Description 8
3.2 University System Logical Model 10
3.3 University System DB2 Physical Model 11
3.4 University System DB2 Schema 12
3.5 University System Interactive Queries 14
3.6 University System Application Program 16

Chapter 4: Airline Reservation 25

4.1 Airline Reservation informal description 25
4.2 Airline Reservation Logical Model 27
4.3 Airline Reservation DB2 Physical Model 28
4.4 Airline Reservation DB2 Schema 29
4.5 Airline Reservation interactive queries 31

Chapter 5: Movie Rental 34

5.1 MOVIE RENTAL INFORMAL DESCRIPTION 34
5.2 MOVIE RENTAL LOGICAL MODEL 35
5.3 MOVIE RENTAL PHYSICAL DB2 MODEL 36
5.4 MOVIE RENTAL DB2 SCHEMA 37
5.5 MOVIE RENTAL INTERACTIVE QUERIES 39

Chapter 6: Car Rental 42

6.1 CAR RENTAL INFORMAL DESCRIPTION 42
6.2 CAR RENTAL LOGICAL MODEL 44
6.3 CAR RENTAL PHYSICAL DB2 MODEL 45
6.4 CAR RENTAL DB2 SCHEMA 46
6.5 CAR RENTAL INTERACTIVE QUERIES 48

Chapter 7: Course Registration 51

7.1 COURSE REGISTRATION INFORMAL DESCRIPTION 51
7.2 COURSE REGISTRATION LOGICAL MODEL 52
7.3 COURSE REGISTRATION PHYSICAL DB2 MODEL 53
7.4 COURSE REGISTRATION DB2 SCHEMA 54
7.5 COURSE REGISTRATION INTERACTIVE QUERIES 58

Chapter 8: Emergency Room 60

8.1 EMERGENCY ROOM INFORMAL DESCRIPTION 60
8.2 EMERGENCY ROOM LOGICAL MODEL 61
8.3 EMERGENCY ROOM PHYSICAL DB2 MODEL 62
8.4 EMERGENCY ROOM DB2 SCHEMA 63
8.5 EMERGENCY ROOM INTERACTIVE QUERIES 66

Chapter 9: Property Rental 69

9.1 PROPERTY RENTAL INFORMAL DESCRIPTION 69
9.2 PROPERTY RENTAL LOGICAL MODEL 70
9.3 PROPERTY RENTAL PHYSICAL DB2 MODEL 71
9.4 PROPERTY RENTAL DB2 SCHEMA 72
9.5 PROPERTY RENTAL INTERACTIVE QUERIES 77

Chapter 10: Software Project 80

10.1 SOFTWARE PROJECT INFORMAL DESCRIPTION 80
10.2 SOFTWARE PROJECT LOGICAL MODEL 81
10.3 SOFTWARE PROJECT PHYSICAL DB2 MODEL 82
10.4 SOFTWARE PROJECT DB2 SCHEMA 83

10.5 SOFTWARE PROJECT INTERACTIVE QUERIES 85

Chapter 11:Tour Operator System 87

11.1 TOUR OPERATOR SYSTEM INFORMAL DESCRIPTION 87
11.2 TOUR OPERATOR SYSTEM LOGICAL MODEL 89
11.3 TOUR OPERATOR SYSTEM PHYSICAL DB2 MODEL 90
11.4 TOUR OPERATOR SYSTEM DB2 SCHEMA 91
11.5 TOUR OPERATOR SYSTEM INTERACTIVE QUERIES 94

Chapter 12: Warehouse System 97

12.1 WAREHOUSE SYSTEM INFORMAL DESCRIPTION 97
12.2 WAREHOUSE SYSTEM LOGICAL MODEL 99
12.3 WAREHOUSE SYSTEM PHYSICAL DB2 MODEL 100
12.4 WAREHOUSE SYSTEM DB2 SCHEMA 101
12.5 WAREHOUSE SYSTEM INTERACTIVE QUERIES 103

Conclusion and Future Work 106

Bibliography 107

Chapter 1: Introduction

Database Management Systems are really ubiquitous in this age of Internet commerce. Although the development of relational database system theory and practice can be traced to the 1970 seminal paper ***A Relational Model of Data for Large Shared Data Banks*** by E.F. Codd [1], the explosive growth of the use of relational database management systems came with the advent of Internet. Databases not only represent significant infrastructure for computer applications, but they also process the transactions and exchanges that drive most of the world economy. A significant and growing segment of the software industry, known as the database industry includes IBM Corporation, Oracle Corporation, Informix Corporation, Sybase Incorporated, Teradata Corporation and Microsoft Corporation.

E. F. Codd found the database technologies of the late 1960's taking the old-fashioned view "that the burden of finding information should be on users ..." unsatisfactory. He proposed what he called a "relational model" based on two fundamental premises: What Codd called the "relational model" rested on two key points:

1. It provided means of describing data with its natural structure only—that is, without any formatting aspects, i.e. superimposing any additional structure for machine representation purposes.

2. Between application programs on the one hand and the database system on the other. Accordingly, it provided a natural and consistent basis for a high level query language which facilitated maximal independence

These aspects are utilized every second all over the world as most of the Internet traffic and most of the Internet commerce rely on database access of some form. The relational databases allow various applications in various programming languages to access and modify a databases.

There are many drawbacks to the relational model and in many ways it is quite obsolete, see [2] or in particular the ACM blog by M. Stonebraker, one of the pioneers of the relational database management systems, [3]. In general , the disadvantages of the relational approach can be summarized by:

1. They are slow and cumbersome; in the early days they were slow - relational DBMS (database management systems) have to employ many tables to conform

to the various normalization rules. This can make them slow and resource inefficient. However most contemporary relational DBMS do not have performance problems now.

2. Restricted attribute sizes. Attribute lengths are usually capped by a maximum size. This can lead to occasional practical problems e.g. a company with a 400 character name - not frequent, but it can happen.

3. SQL does not provide an efficient way to browse alphabetically through an index. Thus some systems cannot provide a simple title A-Z browse.

4. To fine-tune a SQL query is not a simple task and its performance may often depend on the SQL query engine having up-to-date statistics of the database.

5. They do not "storing" of objects, in essence, objects must be "disassembled" before storing them in a relational database, and "assembled" when they are fetched.

Yet, despite these deficiencies an obsoleteness, the use of relational databases proliferates. The major advantages could be summarized as follows:

1. Overwhelmingly, the most popular type of DBMS in use and as a result technical development effort ensures that advances appear quickly and reliably.

2. A multitude of third party tools that are designed to work with the popular relational DBMS via standards such as Open Database Connectivity (ODBC) or Java Database Connectivity (JDBC).

3. A multitude of third party design and modeling tools for the relational model, such as ERwin© modeler used in this thesis.

4. Very well developed management tools and security with automatic data logging and recovery.

5. Referential integrity controls ensuring data consistency.

6. Transactional processing ensuring that incomplete transactions do not occur.

Based on the robust demand for relational databases, all Canadian universities offer database management course in their Computer Science programmes. A typical database management course covers mostly the relational database systems and a significant portion of the practical aspect of any such course deals with the modeling, design and

installation of relational databases, and querying of databases using SQL both interactively and via application programs.

The majority of textbooks dealing with databases as well as the majority of on-line database tutorials provide a comprehensive set of examples concerning all five aspects mentioned above: modeling, design, and installation of a database, and interactive SQL querying and SQL application programming. However, these examples are usually very simple and very small, just to illustrate or illuminate a concept or principle. They are not sufficient for practicing of these aspects. The real-world projects are on the other hand rather complex and extensive, not suitable to a typical one-term undergraduate course in database design and applications.

The objective of this thesis is to design and develop a suite of projects that are of some substance to provide a real practice ground for modeling, design, and installation of a relational database, and SQL interactive querying and SQL application programming, yet be of a complexity and extent that would allow it to be used in a one-term undergraduate course. To this end, a set of ten projects is presented, each leading to a reasonably small database of 20 to 30 tables, yet exhibiting a comprehensiveness of larger projects.

The structure of each project presented here is the same:

1. An informal specification of the project in simple English.
2. An Entity-Relationship (ER) model is constructed using Erwin © software and presented in the form of what is called in Erwin's terminology the Logical and the Physical models.
3. The DB2 database schema is produced based on the ER model.
4. Several SQL queries that can be used to query of the installed database are presented.
5. A simple application program in C is presented that utilize embedded SQL to work with the installed database -- usually performing the same queries used in the part 4.

Since the size of the thesis is already large enough and so only for the first project the source code of the C application programs is presented, for all other projects the source codes are listed in the Appendix.

Chapter 2: Objectives, Decisions, and Methods

One of important decisions of this thesis was whether the classical ER modeling diagrams were to be used, i.e. the diagrams as proposed by Peter Chen in his pivotal 1976 paper *The Entity-Relationship Model: Toward a Unified View of Data*, see [4]. The typical arguments in favour include the facts that majority of database textbooks present the original diagrams and that the original diagrams facilitate better understanding of the concepts and methods. They are simple to draw, so ER models can easily be done by hand as long as the overall size is not too big.

On the other hand, the original diagrams are bulky and unwieldy, and so even a simple design requires many pages. Moreover, the ER models must be translated to relational model before a database schema can be produced. The industrial and commercial software modelers typically use approach that is closer to the relational model than to the ER model, and for good reasons. Graphically, such models are much more compact and more wieldy and though conceptually a bit more challenging than the ER approach, they are still relatively easy to learn and master. The arguments in favour of employing the more industrial approach won and the models in this thesis were created using a professional design software Erwin ©.

A decision concerning application programming affected the selection of the DBMS for our projects. Any application program must access a database, usually using a software tool or an interface provided by the vendor of the DBMS. Among languages popular for database application programming, Python and PHP stand out. Python accesses the database using API -- application programming interface. If Python was just chosen as the language of application programming, the students would have to know or learn Python and learn the appropriate API -- a too steep requirement to fit into a single term, database course that should be focused on the DBMS rather than Python or API of the DBMS. PHP access the database using Microsoft ODBC. Similarly as for Python, the students would have to know or learn PHP and learn the "language" of ODBC, moreover the ODBC is a software that must be purchased independently from the DBMS. Another popular language for database application is Java that accesses a database using JDBC. Even though this is a less stringent requirement than PHP and ODBC, it still is too stringent.

Using embedded SQL in C programs is the least stringent requirement as the students do not have to learn any new language as by the time they take the database course, practically all students know enough of C, they cover SQL in the database course, so the expenditure for learning both static embedded SQL and dynamic embedded SQL is significantly less than learning ODBC, or JDBC, or API etc. Embedded SQL, though not limited to C, works best with C as the CLI -- call level interface -- to which embedded SQL are translated by the pre-processor is well defined in C as they are all actually programmed in C. For these reasons, we opted for application programming based on embedded SQL and C.

Another important decision concerned which of the concrete database management systems (DBMS for short) should be used for thesis. The original idea to include several major ones -- Oracle, DB2, and MySQL -- was rejected as the resulting thesis would be just too large. For each system, we looked at several criteria:

- How available it is to students, and McMaster students in particular?
- How commercially successful and practical the system is?
- Does ERwin provide modeling support for it?
- Does the system provide a native support -- i.e. set of tools -- for embedded SQL programming in C?

On availability to students, all three DBMS considered scored equally as they all offer free versions: DB2 has a free student version, Oracle provided free Oracle Database Express Edition, and MySQL that started as an Open Software is in public domain. On particular availability to McMaster students DB2 scored the highest as it is the DBMS of choice for the undergraduate database course and its graduate variant.

In commercial success and practicality, Oracle is a bit ahead of DB2 that is a bit ahead of MySQL, however the differences were not significant enough for our purposes.

Contemporary version of ERwin provides modeling support for all three DBMS considered equally.

For embedded SQL (ESQL for short), both DB2 and Oracle provide a very good support, while MySQL lacks any native support for ESQL, third-party providers have been promising ESQL tools (pre-processor) for some time, but none is still available.

Based on the above criteria and the fact that due to the size, for this thesis we had to consider a single DBMS, DB2 was chosen over Oracle due to better availability in particular to McMaster students. MySQL was rejected for the lack of ESQL support.

For the data modeling part of our projects, we decided to use ERwin modeling tool. In real world, most data models are the form of ER diagrams that visually represent entities, attributes, and relationships. As mentioned in the introduction, the original Chen's format is not very suitable as it is too bulky. Most of available software modeling tools thus provide ER models that are closer to the relational model than to the pure ER model. ERwin is no exception, and supports both the Information Engineering format (IE, most popular format) and the IDEF1X format. Moreover ERwin is user friendly and easy to learn, provides free Community Edition of Erwin Data Modeler, and for most of the DBMS we considered generates database schemas automatically.

ERwin models have two forms, so-called Logical Model and Physical Model. The Logical Model is a representation of an organization's data, organized in terms of entities and relationships and independent of any particular data management technology. It includes all entities and relationships among them, all primary keys for all entities are specified, i.e. weak entities are resolved, all attributes for each entity are specified, all foreign keys are specified. Normalization occurs at this level, if needed.

The basic steps in producing the Logical Model are:

- Specify all entities.
- Find attributes for each entity.
- Specify primary keys for all entities.
- Find the relationships between different entities.
- Resolve many-to-many relationships.
- Perform normalization.

The Physical Model represents how the model will be built/stored in the database. A physical database model shows all table structures, including column names, column data types, column constraints, primary keys, foreign keys, and relationships between tables . The features of the Physical Model include the specification all tables and columns, foreign keys are used to identify relationships between tables, intentional de-normalization may occur at this level based on user requirements. The physical considerations may cause the Physical Model to be quite different from the Logical Model, thus the Physical Model will be different for different DBMS. For example, data type for a column maybe different between DB2 and SQL Server.

The basic steps in producing the Physical Model are:

- Convert entities into tables.
- Convert relationships into foreign keys.

- Convert attributes into columns.
- Modify the physical data model based on physical constraints/requirements.

Logical Data Model	Physical Data Model
Entity, Attribute, Relationship.	Table, Column, Key, Data Type, Validation , Trigger, Stored Procedure, Constraint.
The name of Entity & Attribute are more business friendly.	The name of Table & Column are more defined and less generic specific.
Independent of technical platform.	Depend on technical platform.
Normalized	May de-normalized to meet performance requirements.

Figure 1: Summary of aspects of the Logical and Physical Models

Chapter 3: University System

3.1 University System Informal Description

ABC University is a large institution with several campuses. Each campus has a different name, address, distance to the city center and the only bus running to the campus. Each campus has one club. The name of the club, the building in which the club is located, the phone number of the club and the multiple sports which club offers, should all be recorded.

The University consists of a number of faculties, such as the Art Faculty, the Science Faculty, and so on. Each faculty has a name, dean and building. A faculty may be divided into a number of schools, for example, the Science Faculty has a School of Physics and a School of Chemistry. Each school belongs to one faculty only and is located on just one campus, but one campus maybe the location of many schools.

Every school has name and an building assigned to. Each school offers different programmes and each programme can be offered by only one school. Each programme has a unique code, title, level and duration. Each programme comprises several courses, different programmes have different courses. Each course has a unique code and course title. Some courses may have one or more prerequisite courses and one course can be the prerequisite course of some other courses.

Each of the students is enrolled in a single programme of study which involves a fixed core of courses specific to that programme as well as a number of electives taken from other programmes. Students work on courses and are awarded a grade in any course if he/she passes the course. Otherwise the student has to re-take the failed course. The system needs to record the year and term in which the course was taken and the grade awarded to the student. Every student has a unique ID. The system also keeps the student name, birthday and the year he/she enrolled in the course.

The school employs lecturers to teach the students. A lecturer is allowed to work for one school only. Each lecturer is assigned an ID which is unique across the whole university. The system keeps the lecturer's name, title and the office room. A supervisor maybe in charge of several lecturers, but a lecturer, however reports to only one supervisor. A lecturer can teach many different courses. A course may also have been taught by many different lecturers.

The university is operated by committees. Each faculty has to have a number of committees with the same titles across the university, such as the Faculty Executive, the Post Graduate Studies Committee, the Health and Sanity Committee, and so on. The committees meet regularly, such as weekly or monthly. The frequency is determined by the faculty involved. A committee's members are all lecturers. A lecturer may be a member of several committees.

3.2 University System Logical Model

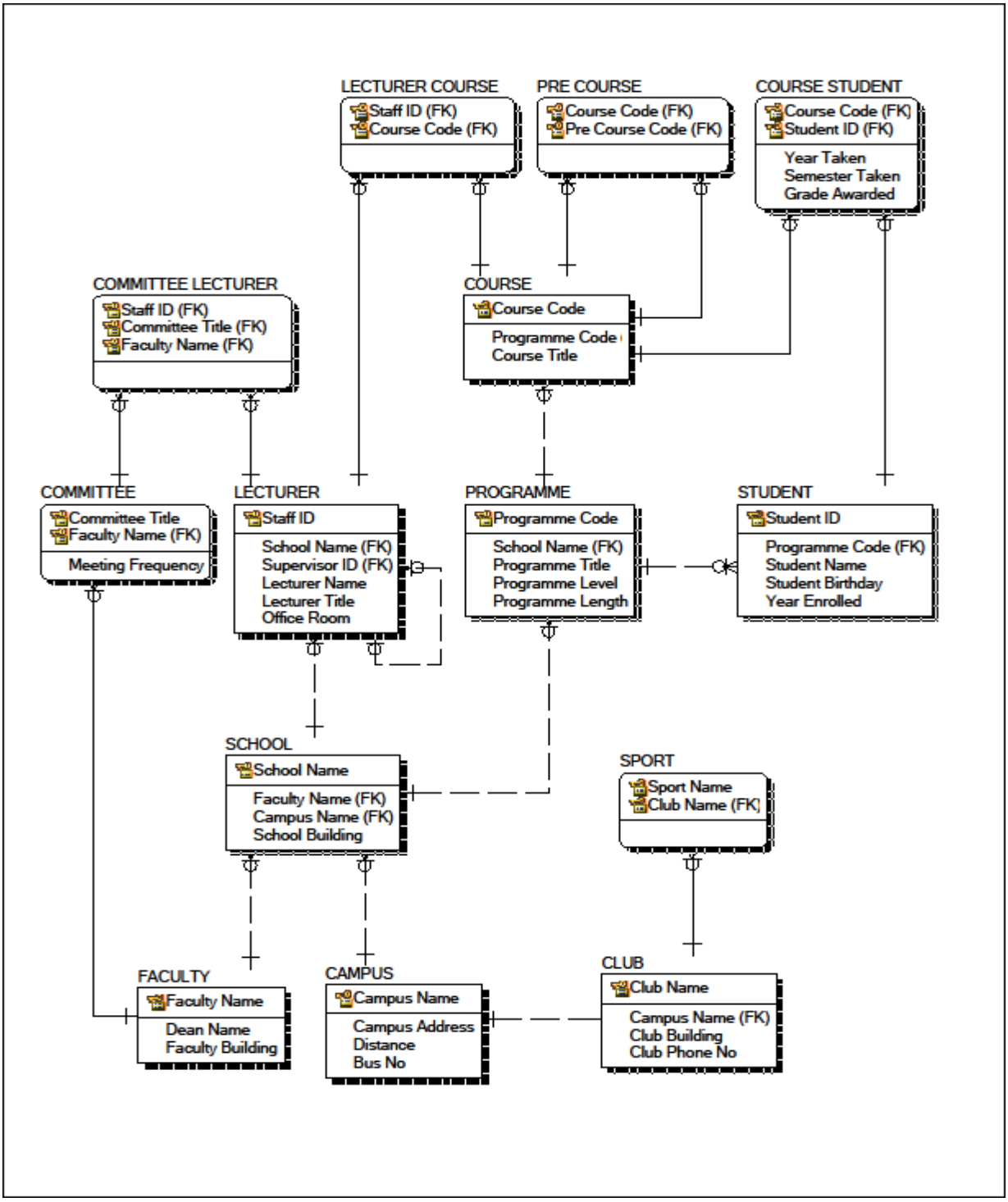


Figure 2: The University System Logical Model

3.3 University System DB2 Physical Model

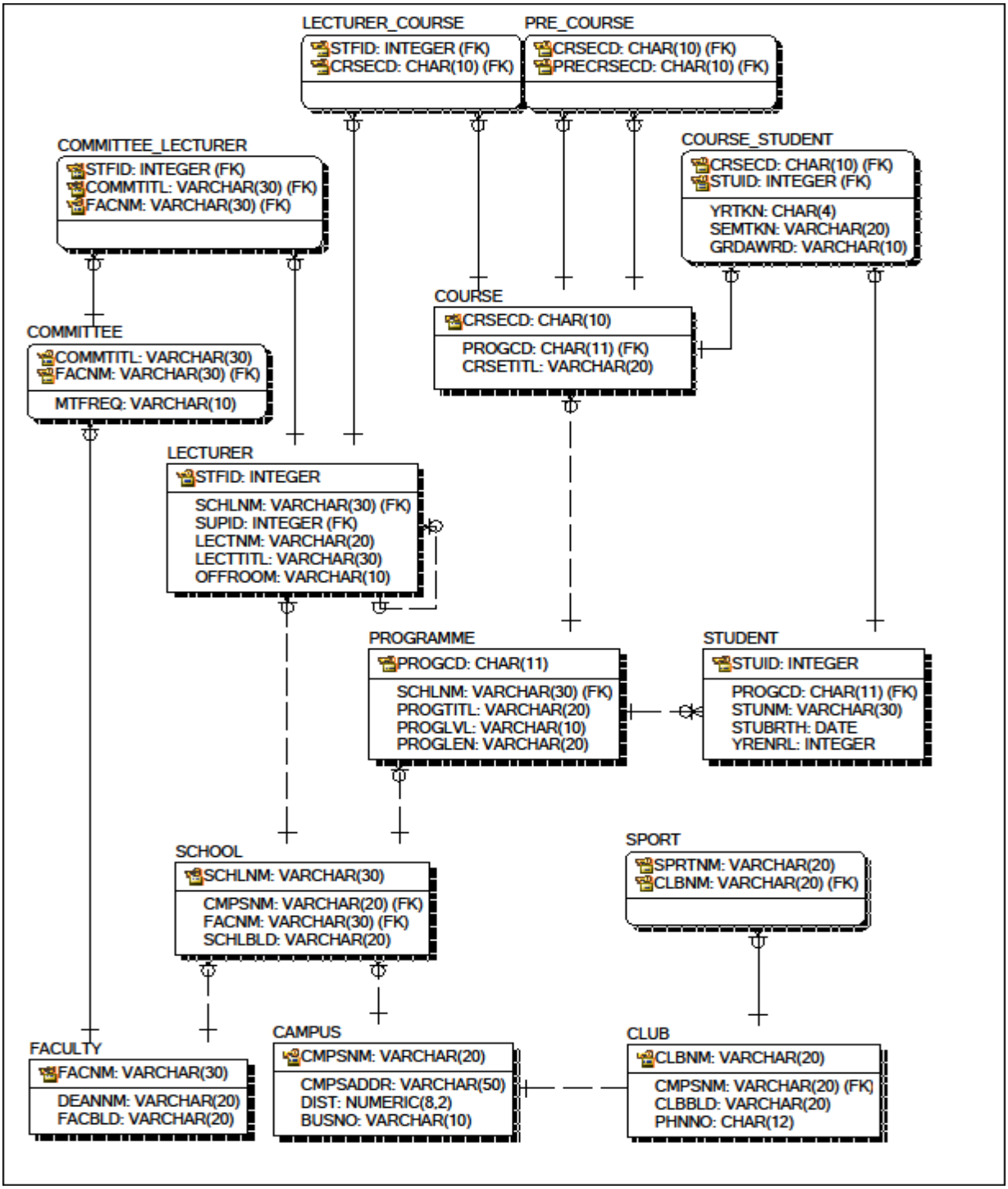


Figure 3: The University System Physical Model

3.4 University System DB2 Schema

```
CREATE TABLE CAMPUS(
       CMPSNM                VARCHAR(20) NOT NULL,
       CMPSADDR              VARCHAR(50),
       DIST                  NUMERIC(8, 2),
       BUSNO                 VARCHAR(10),
       PRIMARY KEY (CMPSNM)
)
CREATE TABLE FACULTY(
       FACNM                 VARCHAR(30) NOT NULL,
       DEANNM                VARCHAR(20),
       FACBLD                VARCHAR(20),
       PRIMARY KEY (FACNM)
)
CREATE TABLE SCHOOL(
       SCHLNM                VARCHAR(30) NOT NULL,
       CMPSNM                VARCHAR(20) NOT NULL,
       FACNM                 VARCHAR(30) NOT NULL,
       SCHLBLD               VARCHAR(20),
       PRIMARY KEY (SCHLNM),
       FOREIGN KEY (CMPSNM) REFERENCES CAMPUS  (CMPSNM),
       FOREIGN KEY (FACNM) REFERENCES FACULTY  (FACNM)
)
CREATE TABLE PROGRAMME(
       PROGCD                CHAR(11) NOT NULL,
       SCHLNM                VARCHAR(30) NOT NULL,
       PROGTITL              VARCHAR(20),
       PROGLVL               VARCHAR(10),
       PROGLEN               VARCHAR(20),
       PRIMARY KEY (PROGCD),
       FOREIGN KEY (SCHLNM) REFERENCES SCHOOL  (SCHLNM)
)
CREATE TABLE STUDENT(
       STUID                 INTEGER NOT NULL,
       PROGCD                CHAR(11) NOT NULL,
       STUNM                 VARCHAR(30),
       STUBRTH               DATE,
       YRENRL                INTEGER,
       PRIMARY KEY (STUID),
       FOREIGN KEY (PROGCD) REFERENCES PROGRAMME  (PROGCD)
)
CREATE TABLE COURSE(
       CRSECD                CHAR(10) NOT NULL,
       PROGCD                CHAR(11) NOT NULL,
       CRSETITL              VARCHAR(20),
       PRIMARY KEY (CRSECD),
       FOREIGN KEY (PROGCD) REFERENCES PROGRAMME  (PROGCD)
)
```

```
CREATE TABLE COURSE_STUDENT(
       CRSECD                CHAR(10) NOT NULL,
       STUID                 INTEGER NOT NULL,
       YRTKN                 CHAR(4),
       SEMTKN                VARCHAR(20),
       GRDAWRD               VARCHAR(10),
       PRIMARY KEY (CRSECD, STUID),
       FOREIGN KEY (STUID) REFERENCES STUDENT  (STUID),
       FOREIGN KEY (CRSECD) REFERENCES COURSE  (CRSECD)
)
CREATE TABLE CLUB(
       CLBNM                 VARCHAR(20) NOT NULL,
       CMPSNM                VARCHAR(20) NOT NULL,
       CLBBLD                VARCHAR(20),
       PHNNO                 CHAR(12),
       PRIMARY KEY (CLBNM),
       FOREIGN KEY (CMPSNM) REFERENCES CAMPUS  (CMPSNM)
)
CREATE TABLE SPORT(
       SPRTNM                VARCHAR(20) NOT NULL,
       CLBNM                 VARCHAR(20) NOT NULL,
       PRIMARY KEY (SPRTNM, CLBNM),
       FOREIGN KEY (CLBNM) REFERENCES CLUB  (CLBNM)
)
CREATE TABLE PRE_COURSE(
       CRSECD                CHAR(10) NOT NULL,
       PRECRSECD             CHAR(10) NOT NULL,
       PRIMARY KEY (CRSECD, PRECRSECD),
       FOREIGN KEY (CRSECD) REFERENCES COURSE  (CRSECD),
       FOREIGN KEY (PRECRSECD) REFERENCES COURSE  (CRSECD)
)
CREATE TABLE LECTURER(
       STFID                 INTEGER NOT NULL,
       SCHLNM                VARCHAR(30) NOT NULL,
       SUPID                 INTEGER,
       LECTNM                VARCHAR(20),
       LECTTITL              VARCHAR(30),
       OFFROOM               VARCHAR(10),
       PRIMARY KEY (STFID),
       FOREIGN KEY (SCHLNM) REFERENCES SCHOOL  (SCHLNM),
       FOREIGN KEY (SUPID) REFERENCES LECTURER  (STFID)
)
CREATE TABLE LECTURER_COURSE(
       STFID                 INTEGER NOT NULL,
       CRSECD                CHAR(10) NOT NULL,
       PRIMARY KEY (STFID, CRSECD),
       FOREIGN KEY (CRSECD) REFERENCES COURSE  (CRSECD),
       FOREIGN KEY (STFID) REFERENCES LECTURER  (STFID)
)
CREATE TABLE COMMITTEE(
       COMMTITL              VARCHAR(30) NOT NULL,
       FACNM                 VARCHAR(30) NOT NULL,
       MTFREQ                VARCHAR(10),
       PRIMARY KEY (COMMTITL, FACNM),
```

```
      FOREIGN KEY (FACNM) REFERENCES FACULTY  (FACNM)
)
CREATE TABLE COMMITTEE_LECTURER(
       STFID                INTEGER NOT NULL,
       COMMTITL             VARCHAR(30) NOT NULL,
       FACNM                VARCHAR(30) NOT NULL,
       PRIMARY KEY (STFID, COMMTITL, FACNM),
       FOREIGN KEY (STFID) REFERENCES LECTURER  (STFID),
       FOREIGN KEY (COMMTITL, FACNM) REFERENCES COMMITTEE  (COMMTITL,
                    FACNM)
)
```

3.5 University System Interactive Queries

Query 1: List all the schools are located in 'Toronto Campus', and sort them by school name.

```
SELECT SCHLNM AS SCHOOL_NAME
FROM SCHOOL
WHERE CMPSNM = 'Toronto Campus'
ORDER BY SCHLNM
```

Query 2: List all the programmes provided by 'science faculty'.

```
SELECT P.PROGCD AS PROGRAMME_CODE
FROM PROGRAMME P
INNER JOIN SCHOOL S ON P.SCHLNM = S.SCHLNM
INNER JOIN FACULTY F ON S.FACNM = F.FACNM
WHERE F.FACNM = 'science faculty'
```

Query 3: Give all the names of the lecturers who are the members of the committee and sort by their name.

```
SELECT DISTINCT L.LECTNM AS LECTURER_NAME
FROM COMMITTEE_LECTURER CL
JOIN LECTURER L ON CL.STFID = L.STFID
ORDER BY L.LECTNM
```

Query 4: List all supervisor's name and the name of the lecturer they manage. Please sort by supervisor name and lecturer name.

```
SELECT SUP.LECTNM AS SUPERVISOR_NAME,L.LECTNM AS LECTURER_NAME
FROM LECTURER SUP
INNER JOIN LECTURER L ON SUP.STFID = L.SUPID
ORDER BY SUP.LECTNM,L.LECTNM
```

Query 5: Give all the lecturers who are not the member of the committee.

```
SELECT STFID AS STAFF_ID
FROM LECTURER
WHERE STFID NOT IN (SELECT DISTINCT STFID FROM COMMITTEE_LECTURER)
```

Query 6: Give the total number of courses for each programme.

```
SELECT PROGCD AS PROGRAMME_CODE,COUNT(CRSECD) AS NUMBER_OF_COURSE
FROM COURSE
GROUP BY PROGCD
```

Query 7: Give all the lecturers with the courses they are teaching. Sort by lecturer name.

```
SELECT L.LECTNM AS LECTURER_NAME,C.CRSETITL AS COURSE_TITLE
FROM LECTURER_COURSE LC
INNER JOIN LECTURER L ON LC.STFID = L.STFID
INNER JOIN COURSE C ON LC.CRSECD = C.CRSECD
ORDER BY L.LECTNM
```

Query 8: Give all the course titles and their corresponding prerequisite course titles.

```
SELECT C1.CRSETITL AS COURSE_TITLE,C2.CRSETITL AS PRE_COURSE_TITLE
FROM PRE_COURSE PC
INNER JOIN COURSE C1 ON PC.CRSECD = C1.CRSECD
INNER JOIN COURSE C2 ON PC.PRECRSECD = C2.CRSECD
```

Query 9: Give the top 5 courses which have more students involved.

```
SELECT C.CRSECD AS COURSE_CODE,COUNT(SS.STUID) AS NUMBER_OF_STUDENTS
FROM COURSE_STUDENT SS
LEFT JOIN COURSE S ON SS.SUBJCD = S.SUBJCD
LEFT JOIN COURSE C ON S.CRSECD =  C.CRSECD
GROUP BY C.CRSECD
ORDER BY NUMBER_OF_STUDENTS DESC
FETCH FIRST 5 ROWS ONLY
```

Query 10**:** Give any of the prerequisite courses was not took by any of the students who enrolled into the university in 2010, and were taking the courses in 2011.

```
SELECT DISTINCT STUID,PC.PRECRSECD AS PRE_COURSE
FROM COURSE_STUDENT CS
RIGHT JOIN PRE_COURSE PC ON CS.CRSECD = PC.CRSECD
WHERE STUID IN (SELECT STUID FROM STUDENT WHERE YRENRL = 2010)
AND YRTKN = 2011
EXCEPT
SELECT STUID,CRSECD
FROM COURSE_STUDENT
WHERE STUID IN (SELECT STUID FROM STUDENT WHERE YRENRL = 2010)
AND YRTKN = 2010
```

3.6 University System Application Program

Write a simple C program using embedded SQL. The program is to excute and display the results on the screen in the same format as the DB2. It is to execute any of the ten queries listed in the previous section based on what action the user chooses, or all ten queries together. Assume the name of the databse is OURDB.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

EXEC SQL INCLUDE SQLCA;

void query1();
void query2();
void query3();
void query4();
void query5();
void query6();
void query7();
void query8();
void query9();
void query10();
void do_all();

int connected=0;

void sqlerr(char* x) {
  if (SQLCODE!=0) {
    printf("error %s (%d)\n",x,SQLCODE);
    if (connected) {
      EXEC SQL CONNECT RESET;
      printf("disconnected from OURDB\n");
    }
    exit(1);
  }
}

void connect() {
  EXEC SQL CONNECT TO OURDB;
  sqlerr("CONNECT TO OURDB");
  printf("Connected to OURDB\n");
  connected = 1;
}

void disconnect() {
  EXEC SQL CONNECT RESET;
  sqlerr("CONNECT RESET");
  printf("Disconnected from CS3DB3\n");
  connected = 0;
}
```

```
void depad(char* s) {
  int i;
  i=strlen(s)-1;
  while(s[i]==' ') s[i--]='\0';
}

char buffer[300];

int main() {
  int i, j;
  connect();
  while(1) {
     A: printf("enter number of query you want to execute "
              "(1-10),\n 0 to quit, anything else all queries\n");
        fgets(buffer,300,stdin);
        if (buffer[0]=='\n') {
          printf("nothing entered\n");
          continue;
        }
        i=j=0;
        while(buffer[j]!='\n') {
          if (buffer[j]>='0' && buffer[j]<='9')
            i = 10*i+buffer[j]-'0';
          else{
            printf("incorrect entry\n");
            goto A;
          }
          j++;
     }
     if (i == 0) break;
     else if (i == 1) query1();
     else if (i == 2) query2();
     else if (i == 3) query3();
     else if (i == 4) query4();
     else if (i == 5) query5();
     else if (i == 6) query6();
     else if (i == 7) query7();
     else if (i == 8) query8();
     else if (i == 9) query9();
     else if (i == 10) query10();
     else {
       do_all();
       break;
     }
  }

  disconnect();
  return 0;
}
int count, len, len1;
char buf1[80];

EXEC SQL BEGIN DECLARE SECTION;
```

```
  char school_name[30];
  char programme_code[11];
  char lecturer_name[20];
  char supervisor_name[20];
  sqlint32 staff_id;
  sqlint16 number_of_course;
  char course_title[20];
  char pre_course_title[20];
  sqlint16 number_of_students;
  sqlint32 student_id;
  char pre_course[10];
EXEC SQL END DECLARE SECTION;

void query1() {
  printf("displaying result of query 1\n");
  EXEC SQL DECLARE cur1 CURSOR FOR
       SELECT SCHLNM AS SCHOOL_NAME
       FROM SCHOOL
       WHERE CMPSNM = 'Toronto Campus'
       ORDER BY SCHLNM;
  sqlerr("OPEN cur1");

  count=0;
  while(1) {
    EXEC SQL FETCH cur1 INTO :school_name;

    if (SQLCODE!=0) break;
    if (count==0) {
      printf("SCHOOL_NAME\n");
      printf("----------------\n");
    }
    depad(school_name);
    printf("%s\n",school_name);
    count++;
  }

  EXEC SQL CLOSE cur1;
  sqlerr("CLOSE cur1");

  printf("\n  %d record(s) selected.\n",count);
}

void query2() {
  printf("displaying result of query 2\n");
  EXEC SQL DECLARE cur2 CURSOR FOR
       SELECT P.PROGCD AS PROGRAMME_CODE
       FROM PROGRAMME P
       INNER JOIN SCHOOL S ON P.SCHLNM = S.SCHLNM
       INNER JOIN FACULTY F ON S.FACNM = F.FACNM
       WHERE F.FACNM = 'science faculty';
  sqlerr("DECLARE cur2");

  EXEC SQL OPEN cur2;
  sqlerr("OPEN cur2");
```

```
  count=0;
  while(1) {
    EXEC SQL FETCH cur2 INTO :programme_code;
    if (SQLCODE!=0) break;
    if (count==0) {
      printf("PROGRAMME_CODE\n");
      printf("---------------\n");
    }
    depad(programme_code);
    printf("%s\n",programme_code);
    count++;
  }

  EXEC SQL CLOSE cur2;
  sqlerr("CLOSE cur2");

  printf("\n  %d record(s) selected.\n",count);
}

void query3() {
  printf("displaying result of query 3\n");
  EXEC SQL DECLARE cur3 CURSOR FOR
       SELECT DISTINCT L.LECTNM AS LECTURER_NAME
       FROM COMMITTEE_LECTURER CL
       JOIN LECTURER L ON CL.STFID = L.STFID
       ORDER BY L.LECTNM;
  sqlerr("DECLARE cur3");

  EXEC SQL OPEN cur3;
  sqlerr("OPEN cur3");

  count=0;
  while(1) {
    EXEC SQL FETCH cur3 INTO :lecturer_name;
    if (SQLCODE!=0) break;
    if (count==0) {
      printf("LECTURER_NAME\n");
      printf("-----------------\n");
    }
    printf("%s\n",lecturer_nam);
    count++;
  }

  EXEC SQL CLOSE cur3;
  sqlerr("CLOSE cur3");

  printf("\n  %d record(s) selected.\n",count);
}

void query4() {
  int i;
  printf("displaying result of query 4\n");

  EXEC SQL DECLARE cur4 CURSOR FOR
```

```
  SELECT SUP.LECTNM AS SUPERVISOR_NAME,L.LECTNM AS LECTURER_NAME
      FROM LECTURER SUP
      INNER JOIN LECTURER L ON SUP.STFID = L.SUPID
      ORDER BY SUP.LECTNM,L.LECTNM;
  sqlerr("DECLARE cur4");

  EXEC SQL OPEN cur4;
  sqlerr("OPEN cur4");

  count=0;
  while(1) {
    EXEC SQL FETCH cur4 INTO :supervisor_name,
                             :lecturer_name;
    if (SQLCODE!=0) break;
    if (count==0) {
     printf("SUPERVISOR_NAME           LECTURER_NAME\n");
      printf("----------------------------------------\n");
    }
    depad(supervisor_name);
    depad(lecturer_name);
    printf("%s",supervisor_name);
    for(i = 20-strlen(supervisor_name); i>=0; i--) fputc(' ',stdout);
    printf("%s\n",lecturer_name);
    count++;
  }
  EXEC SQL CLOSE cur4;
  sqlerr("CLOSE cur4");

  printf("\n  %d record(s) selected.\n",count);
}

void query5() {
  printf("displaying result of query 5\n");
  EXEC SQL DECLARE cur5 CURSOR FOR
       SELECT STFID AS STAFF_ID
       FROM LECTURER
       WHERE STFID NOT IN
                 (SELECT DISTINCT STFID FROM COMMITTEE_LECTURER);
  sqlerr("DECLARE cur5");

  EXEC SQL OPEN cur5;
  sqlerr("OPEN cur5");

  count=0;
  while(1) {
    EXEC SQL FETCH cur5 INTO :staff_id;
    if (SQLCODE!=0) break;
    if (count==0) {
      printf("STAFF_ID\n");
      printf("-----------\n");
    }
    depad(staff_id);
    printf("%s\n",staff_id);
    count++;
```

```
  }

  EXEC SQL CLOSE cur5;
  sqlerr("CLOSE cur5");

  printf("\n  %d record(s) selected.\n",count);
}

void query6() {
  int i;
  printf("displaying result of query 6\n");
  EXEC SQL DECLARE cur6 CURSOR FOR
       SELECT PROGCD AS PROGRAMME_CODE,COUNT(CRSECD) AS NUMBER_OF_COURSE
       FROM COURSE
       GROUP BY PROGCD;
  sqlerr("DECLARE cur6");

  EXEC SQL OPEN cur6;
  sqlerr("OPEN cur6");

  count=0;
  while(1) {
    EXEC SQL FETCH cur6 INTO :programme_code,
                             :number_of_course;

    if (SQLCODE!=0) break;
    if (count==0) {
      printf("PROGRAMME_CODE        NUMBER_OF_COURSE\n");
      printf("--------------------------------------\n");
    }
    depad(programme_code);
    depad(number_of_course);
    printf("%s",programme_code);
    for(i = strlen(programme_code); i<11; putchar(' '), i++);
    printf("%s",number_of_course);
    count++;
  }

  EXEC SQL CLOSE cur6;
  sqlerr("CLOSE cur6");

  printf("\n  %d record(s) selected.\n",count);
}

void query7() {
  int i;
  printf("displaying result of query 7\n");
  EXEC SQL DECLARE cur7 CURSOR FOR
       SELECT L.LECTNM AS LECTURER_NAME,C.CRSETITL AS COURSE_TITLE
       FROM LECTURER_COURSE LC
       INNER JOIN LECTURER L ON LC.STFID = L.STFID
       INNER JOIN COURSE C ON LC.CRSECD = C.CRSECD
       ORDER BY L.LECTNM;
  sqlerr("DECLARE cur7");
```

```
  EXEC SQL OPEN cur7;
  sqlerr("OPEN cur7");

  count=0;
  while(1) {
    EXEC SQL FETCH cur7 INTO :lecturer_name,
                             :course_title;
    if (SQLCODE!=0) break;
    if (count==0) {
      printf("LECTURER_NAME      COURSE_TITLE\n");
      printf("-----------------------------------\n");
    }
    depad(lecturer_name);
    depad(course_title);
    printf("%s",lecturer_name);
    for(i = strlen(lecturer_name); i<9; putchar(' '), i++);
    printf("%s",course_title);
    count++;
  }

  EXEC SQL CLOSE cur7;
  sqlerr("CLOSE cur7");

  printf("\n  %d record(s) selected.\n",count);
}

void query8() {
  int i;
  printf("displaying result of query 8\n");
  EXEC SQL DECLARE cur8 CURSOR FOR
       SELECT C1.CRSETITL AS COURSE_TITLE,C2.CRSETITL AS
              PRE_COURSE_TITLE
       FROM PRE_COURSE PC
       INNER JOIN COURSE C1 ON PC.CRSECD = C1.CRSECD
       INNER JOIN COURSE C2 ON PC.PRECRSECD = C2.CRSECD;
  sqlerr("DECLARE cur8");

  EXEC SQL OPEN cur8;
  sqlerr("OPEN cur8");

  count=0;
  while(1) {
    EXEC SQL FETCH cur8 INTO :course_title,
                             :pre_course_title;
    if (SQLCODE!=0) break;
    if (count==0) {
      printf("COURSE_TITLE     PRE_COURSE_TITLE\n");
      printf("--------- --------------------------------\n");
    }

    depad(course_title);
    printf("%s",course_title);
    for(i = strlen(course_title); i<10; putchar(' '), i++);
    printf("%.24f\n",pre_course_title);
```

```
    count++;
  }

  EXEC SQL CLOSE cur8;
  sqlerr("CLOSE cur8");

  printf("\n  %d record(s) selected.\n",count);
}

void query9() {
  int i;
  printf("displaying result of query 9\n");
  EXEC SQL DECLARE cur9 CURSOR FOR
       SELECT P.PROGCD AS PROGRAMME_CODE,COUNT(CS.STUID) AS
              NUMBER_OF_STUDENTS
       FROM COURSE_STUDENT CS
       LEFT JOIN COURSE C ON CS.CRSECD = C.CRSECD
       LEFT JOIN PROGRAMME P ON P.PROGCD =  C.PROGCD
       GROUP BY P.PROGCD
       ORDER BY NUMBER_OF_STUDENTS DESC
       FETCH FIRST 5 ROWS ONLY;
  sqlerr("DECLARE cur9");

  EXEC SQL OPEN cur9;
  sqlerr("OPEN cur9");

  count=0;
  while(1) {
    EXEC SQL FETCH cur9 INTO :programme_code,
                             :number_of_students;
    if (SQLCODE!=0) break;
    if (count==0) {
      printf("PROGRAMME_CODE    NUMBER_OF_STUDENTS\n");
      printf("-----------------------------------------\n");
    }
    depad(programme_code);
    depad(number_of_students);

    printf("%s",programme_code);
    for(i = strlen(programme_code); i<12; putchar(' '), i++);
    printf("%s\n",number_of_students);
    count++;
  }

  EXEC SQL CLOSE cur9;
  sqlerr("CLOSE cur9");

  printf("\n  %d record(s) selected.\n",count);
}

void query10() {
  int i;
  printf("displaying result of query 10\n");
```

```
EXEC SQL DECLARE cur10 CURSOR FOR
     SELECT DISTINCT STUID,PC.PRECRSECD AS PRE_COURSE
     FROM COURSE_STUDENT CS
     RIGHT JOIN PRE_COURSE PC ON CS.CRSECD = PC.CRSECD
     WHERE STUID IN (SELECT STUID FROM STUDENT WHERE YRENRL = 2010)
     AND YRTKN = 2011
     EXCEPT
     SELECT STUID,CRSECD
     FROM COURSE_STUDENT
     WHERE STUID IN (SELECT STUID FROM STUDENT WHERE YRENRL = 2010)
     AND YRTKN = 2010;
  sqlerr("DECLARE cur10");

  EXEC SQL OPEN cur10;
  sqlerr("OPEN cur10");

  count=0;
  while(1) {
    EXEC SQL FETCH cur10 INTO :student_id,
                              :pre_course;
    if (SQLCODE!=0) break;
    if (count==0) {
      printf("STUDENT_ID      PRE_COURSE\n");
      printf("-----------------------------------\n");
    }

    depad(student_id);

    printf("%s",student_id);
    for(i = strlen(student_id); i<10; putchar(' '), i++);
    printf("%d\n",pre_course);
    count++;
  }

  EXEC SQL CLOSE cur10;
  sqlerr("CLOSE cur10");

  printf("\n  %d record(s) selected.\n",count);
}

void do_all() {
  query1();
  query2();
  query3();
  query4();
  query5();
  query6();
  query7();
  query8();
  query9();
  query10();
}
```

Chapter 4: Airline Reservation

4.1 Airline Reservation informal description

There are 6 different airlines in 6 different countries: Canada – AirCan, USA - USAir, UK - BritAir, France - AirFrance, Germany - LuftAir, Italy - ItalAir. Their flights involve the following 12 cities: Toronto and Montreal in Canada, New York and Chicago in US, London and Edinburgh in UK, Paris and Nice in France, Bonn and Berlin in Germany, Rome and Naples in Italy. In each of the 12 cities, there is a (single) booking office. You are going to design a central air-reservation database to be used by all booking offices.

The flight has a unique flight number, air line code, business class indicator, smoking allowed indicator. Flight availability has flight number, date + time of departure, number of total seats available in business class, number of booked seats in business class, number of total seats available in economy class, and number of booked seats in economy class.

The customers may come from any country, not just the 6 above, and from any province/state, and from any city. Customer has first & last name, mailing address, zero or more phone numbers, zero or more fax numbers, and zero or more email addresses. Mailing address has street, city, province or state, postal code and country. Phone/fax number has country code, area code and local number. Email address has only one string, and no structure is assumed. A customer can book one or more flights. Two or more customers may have same mailing address and/or same phone number(s) and/or same fax number(s). But the email address is unique for each customer. First and last names do not have to be unique.

Booking has an unique booking number, booking city, booking date, flight number, date + time of departure (in local time, and time is always in hours and minutes), date + time of arrival (in local time), class indicator, total price (airport tax in origin + airport tax in destination + flight price – in local currency. The flight price for business class is 1.5 times of the listed flight price), status indicator (three types: booked. Canceled – the customer canceled the booking, scratched – the customer had not paid in full 30 days prior to the departure), customer who is responsible for payment, amount-paid-so far (in local currency), outstanding balance (in local currency), the first & last names to be printed on the ticket. The airport taxes must be stored in local currencies (i.e. Canadian

dollars, US dollars, British Pounds, French francs, German marks, and Italian Liras). Since the exchange rates change daily, they also must be stored for calculations of all prices involved.

Though France, Germany, and Italy have had a common currency for a while, we used the names of their original currencies to involve in this exercise currency exchange rates and their changes.

4.2 Airline Reservation Logical Model

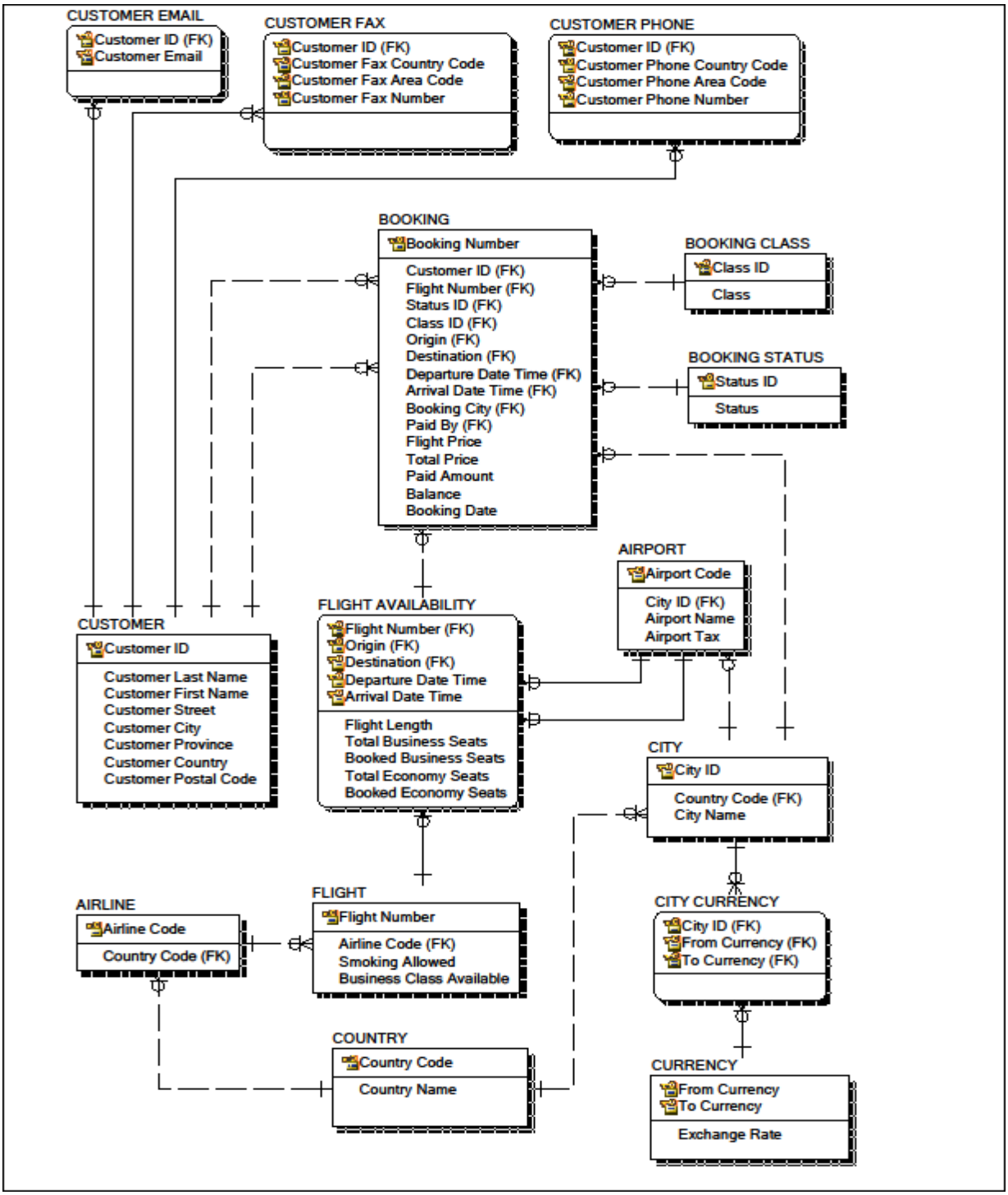


Figure 4: The Airline Reservation Logical Model

4.3 Airline Reservation DB2 Physical Model

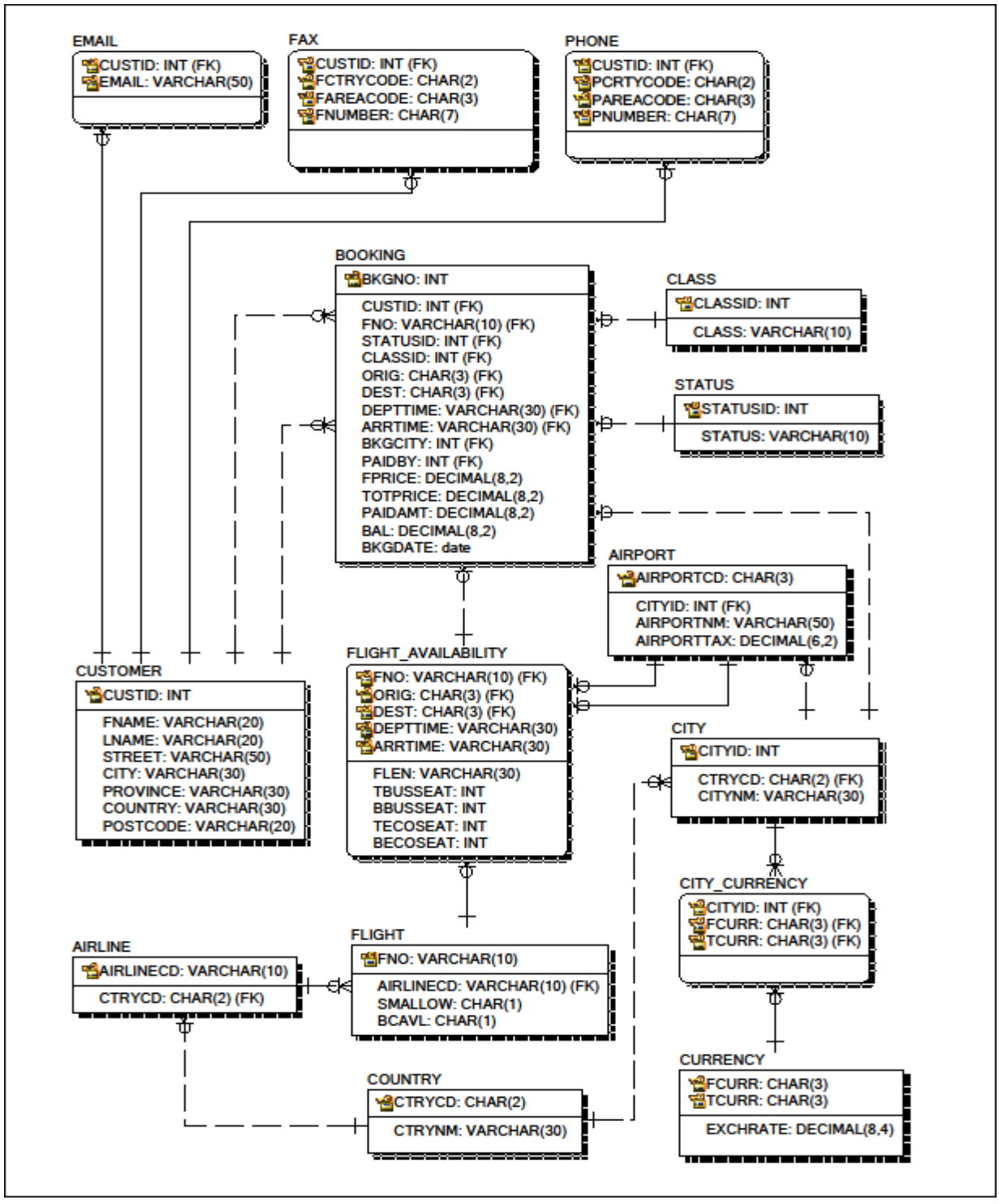


Figure 5: The Airline Reservation Physical Model

4.4 Airline Reservation DB2 Schema

```
CREATE TABLE CUSTOMER (
       CUSTID                INT NOT NULL,
       FNAME                 VARCHAR(20) NOT NULL,
       LNAME                 VARCHAR(20) NOT NULL,
       STREET                VARCHAR(50) NOT NULL,
       CITY                  VARCHAR(30) NOT NULL,
       PROVINCE              VARCHAR(30) NOT NULL,
       COUNTRY               VARCHAR(30) NOT NULL,
       POSTCODE              VARCHAR(20) NOT NULL,
       PRIMARY KEY (CUSTID)
)
CREATE TABLE PHONE (
       PCRTYCODE             CHAR(2) NOT NULL,
       PAREACODE             CHAR(3) NOT NULL,
       PNUMBER               CHAR(7) NOT NULL,
       CUSTID                INT NOT NULL,
       PRIMARY KEY (CUSTID, PCRTYCODE, PAREACODE, PNUMBER),
       FOREIGN KEY (CUSTID) REFERENCES CUSTOMER  (CUSTID)
)
CREATE TABLE FAX (
       FAREACODE             CHAR(3) NOT NULL,
       FCTRYCODE             CHAR(2) NOT NULL,
       FNUMBER               CHAR(7) NOT NULL,
       CUSTID                INT NOT NULL,
       PRIMARY KEY (CUSTID, FCTRYCODE, FAREACODE, FNUMBER),
       FOREIGN KEY (CUSTID) REFERENCES CUSTOMER  (CUSTID)
)
CREATE TABLE EMAIL (
       EMAIL                 VARCHAR(50) NOT NULL,
       CUSTID                INT NOT NULL,
       PRIMARY KEY (CUSTID, EMAIL),
       UNIQUE (EMAIL),
       FOREIGN KEY (CUSTID) REFERENCES CUSTOMER  (CUSTID)
)
CREATE TABLE COUNTRY (
       CTRYCD                CHAR(2) NOT NULL,
       CTRYNM                VARCHAR(30),
       PRIMARY KEY (CTRYCD)
)
CREATE TABLE AIRLINE (
       AIRLINECD             VARCHAR(10) NOT NULL,
       CTRYCD                CHAR(2) NOT NULL,
       PRIMARY KEY (AIRLINECD),
       FOREIGN KEY (CTRYCD) REFERENCES COUNTRY  (CTRYCD)
)
CREATE TABLE FLIGHT (
       FNO                   VARCHAR(10) NOT NULL,
       SMALLOW               CHAR(1) NOT NULL,
       BCAVL                 CHAR(1),
       AIRLINECD             VARCHAR(10) NOT NULL,
```

```
        PRIMARY KEY (FNO),
        FOREIGN KEY (AIRLINECD) REFERENCES AIRLINE  (AIRLINECD)
)
CREATE TABLE CURRENCY (
        FCURR                   CHAR(3) NOT NULL,
        TCURR                   CHAR(3) NOT NULL,
        EXCHRATE                DECIMAL(8, 4) NOT NULL,
        PRIMARY KEY (FCURR, TCURR)
)
CREATE TABLE CITY (
        CITYNM                  VARCHAR(30),
        CITYID                  INT NOT NULL,
        CTRYCD                  CHAR(2) NOT NULL,
        PRIMARY KEY (CITYID),
        FOREIGN KEY (CTRYCD) REFERENCES COUNTRY  (CTRYCD)
)
CREATE TABLE AIRPORT (
        CITYID                  INT NOT NULL,
        AIRPORTCD               CHAR(3) NOT NULL,
        AIRPORTNM               VARCHAR(50),
        AIRPORTTAX              DECIMAL(6, 2) NOT NULL,
        PRIMARY KEY (AIRPORTCD),
        FOREIGN KEY (CITYID) REFERENCES CITY  (CITYID)
)
CREATE TABLE CITY_CURRENCY (
        CITYID                  INT NOT NULL,
        FCURR                   CHAR(3) NOT NULL,
        TCURR                   CHAR(3) NOT NULL,
        PRIMARY KEY (CITYID,FCURR,TCURR),
        FOREIGN KEY (CITYID) REFERENCES CITY  (CITYID),
        FOREIGN KEY (FCURR,TCURR) REFERENCES CURRENCY  (FCURR,TCURR)
)
CREATE TABLE FLIGHT_AVAILABILITY (
        FLEN                    VARCHAR(30) NOT NULL,
        DEPTTIME                VARCHAR(30) NOT NULL,
        ARRTIME                 VARCHAR(30) NOT NULL,
        FNO                     VARCHAR(10) NOT NULL,
        BBUSSEAT                INT NOT NULL,
        BECOSEAT                INT NOT NULL,
        TBUSSEAT                INT NOT NULL,
        TECOSEAT                INT NOT NULL,
        DEST                    CHAR(3) NOT NULL,
        ORIG                    CHAR(3) NOT NULL,
        PRIMARY KEY (FNO, ORIG, DEST, DEPTTIME, ARRTIME),
        FOREIGN KEY (FNO) REFERENCES FLIGHT  (FNO),
        FOREIGN KEY (DEST) REFERENCES AIRPORT  (AIRPORTCD),
        FOREIGN KEY (ORIG) REFERENCES AIRPORT  (AIRPORTCD)
)
CREATE TABLE CLASS (
        CLASSID                 INT NOT NULL,
        CLASS                   VARCHAR(10),
        PRIMARY KEY (CLASSID)
)
CREATE TABLE STATUS (
```

```
        STATUSID              INT NOT NULL,
        STATUS                VARCHAR(10),
        PRIMARY KEY (STATUSID)
)
CREATE TABLE BOOKING (
        BKGNO                 INT NOT NULL,
        BKGDATE               date NOT NULL,
        BAL                   DECIMAL(8, 2),
        TOTPRICE              DECIMAL(8, 2),
        PAIDAMT               DECIMAL(8, 2) NOT NULL,
        FPRICE                DECIMAL(8, 2) NOT NULL,
        FNO                   VARCHAR(10) NOT NULL,
        DEPTTIME              VARCHAR(30) NOT NULL,
        ARRTIME               VARCHAR(30) NOT NULL,
        DEST                  CHAR(3) NOT NULL,
        ORIG                  CHAR(3) NOT NULL,
        BKGCITY               INT NOT NULL,
        CUSTID                INT NOT NULL,
        PAIDBY                INT NOT NULL,
        CLASSID               INT NOT NULL,
        STATUSID              INT NOT NULL,
        PRIMARY KEY (BKGNO),
        FOREIGN KEY (FNO, ORIG, DEST, DEPTTIME, ARRTIME) REFERENCES
        FLIGHT_AVAILABILITY  (FNO, ORIG, DEST, DEPTTIME, ARRTIME),
        FOREIGN KEY (BKGCITY) REFERENCES CITY  (CITYID),
        FOREIGN KEY (CUSTID) REFERENCES CUSTOMER  (CUSTID),
        FOREIGN KEY (PAIDBY) REFERENCES CUSTOMER  (CUSTID),
        FOREIGN KEY (CLASSID) REFERENCES CLASS  (CLASSID),
        FOREIGN KEY (STATUSID) REFERENCES STATUS  (STATUSID)
)
```

4.5 Airline Reservation Interactive Queries

Query 1: Give all the customers who lives in Canada and sort by customer_id.

```
SELECT CUSTID AS CUSTOMER_ID
FROM CUSTOMER
WHERE COUNTRY = 'Canada'
ORDER BY CUSTID
```

Query 2: List all different customers who made bookings.

```
SELECT DISTINCT CUSTID AS CUSTOMER_ID
FROM BOOKING
```

Query 3: Display all currency exchange rate is greater than 1. Please sort them by from_currency and to_currency.

```
SELECT FCURR AS FROM_CURRENCY,TCURR AS TO_CURRENCY,EXCHRATE AS
     EXCHANGE_RATE
FROM CURRENCY
WHERE EXCHRATE > 1
```

```
ORDER BY FCURR,TCURR
```

Query 4: List all the flight availabilities between Toronto (airport code is 'YYZ') and New York (airport code is 'JFK'). Please display flight_no, origin, destinatin, depature_time, and arrival_time. Please sort them by flight_no.

```
SELECT FNO AS FLIGHT_NO,ORIG AS ORIGIN,DEST AS DESTINATION,DEPTTIME
ASDEPATURE_TTIME,ARRTIME AS ARRIVAL_TIME
FROM FLIGHT_AVAILABILITY
WHERE ORIG = 'YYZ' AND DEST = 'JFK'
OR ORIG = 'JFK' AND DEST = 'YYZ'
ORDER BY FNO
```

Query 5: List all customers who did not place any booking. Please display customer_id only, and sort records by customer_id.

```
SELECT CUSTID AS CUSTOMER_ID
FROM CUSTOMER
WHERE CUSTID NOT IN (SELECT DISTINCT CUSTID FROM BOOKING)
ORDER BY CUSTID
```

Query 6: Display all customer's first_name, last_name, phone_no (format like 416-111-2222) and email. Please sort them by customer_id.

```
SELECT C.CUSTID AS CUSTOMER_ID,
       C.FNAME AS FIRST_NAME,
       C.LNAME AS LAST_NAME,
       P.PCRTYCODE||'-'||P.PAREACODE||'-'||P.PNUMBER AS PHONE_NO,
       E.EMAIL AS EMAIL
FROM CUSTOMER C
RIGHT JOIN PHONE P ON C.CUSTID = P.CUSTID
RIGHT JOIN EMAIL E ON C.CUSTID = E.CUSTID
ORDER BY C.CUSTID
```

Query 7: List all canceled bookngs. please display booking_no, customer_id, flight_no, origin, destination, class, status, and booking_city. Please also sort by booking_no, customer_id and flight_no.

```
SELECT BKGNO AS BOOKING_NO,CUSTID AS CUSTOMER_ID,FNO AS FLIGHT_NO,
       ORIG AS ORIGIN,DEST AS DESTINATION,C.CLASS AS CLASS,
       S.STATUS AS STATUS,CITY.CITYNM AS BOOKING_CITY
FROM BOOKING B
INNER JOIN STATUS S ON B.STATUSID = S.STATUSID
INNER JOIN CLASS C ON B.CLASSID = C.CLASSID
INNER JOIN CITY ON B.BKGCITY = CITY.CITYID
WHERE S.STATUS = 'Canceled'
ORDER BY BKGNO,CUSTID,FNO
```

Query 8: List total_price, total_payment and total_balance for each city. Please exclude canceled bookings and sort records by city_name.

```
SELECT C.CITYNM AS CITY,SUM(TOTPRICE)AS TOTAL_PRICE,
       SUM(PAIDAMT) AS TOTAL_PAYMENT,SUM(BAL) AS TOTAL_BALANCE
FROM BOOKING B
INNER JOIN CITY C ON B.BKGCITY = C.CITYID
WHERE STATUSID <> 2
GROUP BY C.CITYNM
ORDER BY C.CITYNM
```

Query 9: Calculate new total_price for each booking if origin airport tax increase by 0.01 and destination airport tax decrease by 0.005. Please display booking_no, origin, destination, flight_price, previous_total_price and new_total_price.

```
SELECT BKGNO AS BOOKING_NO,ORIG AS ORIGIN,
       DEST AS DESTINATION,FPRICE AS FLIGHT_PRICE,
            TOTPRICE AS PREVIOUS_TOTAL_PRICE,
            FPRICE*(1+(O.AIRPORTTAX+0.01)+(D.AIRPORTTAX-0.005)) AS
                  NEW_TOTAL_PRICE
FROM BOOKING B
INNER JOIN AIRPORT O ON B.ORIG = O.AIRPORTCD
INNER JOIN AIRPORT D ON B.DEST = D.AIRPORTCD
```

Query 10: List number_of_bookings, number_of_emails, number_of_phones and mumber_of_faxs for each customer.

```
SELECT C.CUSTID AS CUSTOMER_ID
       ,(SELECT COUNT(CUSTID) FROM BOOKING WHERE CUSTID=C.CUSTID) AS
         NUMBER_OF_BOOKINGS
      ,(SELECT COUNT(CUSTID) FROM EMAIL WHERE CUSTID=C.CUSTID) AS
        NUMBER_OF_EMAILS
      ,(SELECT COUNT(CUSTID) FROM PHONE WHERE CUSTID=C.CUSTID) AS
        NUMBER_OF_PHONES
      ,(SELECT COUNT(CUSTID) FROM FAX WHERE CUSTID=C.CUSTID) AS
        NUMBER_OF_FAXS
FROM CUSTOMER C
```

Chapter 5: Movie Rental

5.1 Movie Rental Informal Description

The movies are rented out in stores and there are several stores. Each store has a unique distributor that supplies the store with tapes. A distributor may supply more than one store. Each distributor has a name, an address, and a phone number. Each store has a name, an address, and a phone number. For each employee we must keep the following information: working store, a name, a supervisor, an address , a phone number, SIN (social insurance number) and the date when the employee was hired. For each customer we have to keep the following information: a name, an address, and a phone number (if any).

For each rental, we must keep track of which employee served the customer, which movie and which copy (i.e. type) the customer rented, information about payment, the date and the time of the rental, the status (rented, returned_in_time, returned_late), the rate (i.e. the price), and if applicable, due date and overdue charges. About the payment we have to keep which of the employees accepted the payment (does not have to be the same employee who rented the tape), the type of payment (i.e. cash, check, credit card, direct debit – for each type you must provide for relevant information to be kept, e.g. credit card number if credit card is used), the amount of the payment, date + time of the payment, payment status (completed if cash or the money have been received, approved if debit or credit card go through, pending if the check has not cleared yet). About each tape we have to keep information in what condition the tape is and what movie is on the tape. About each movie we have to keep its title, director's name, simple description, the name of a (single) major star, the movie's rating (use numbers 1-5).

5.2 Movie Rental Logical Model

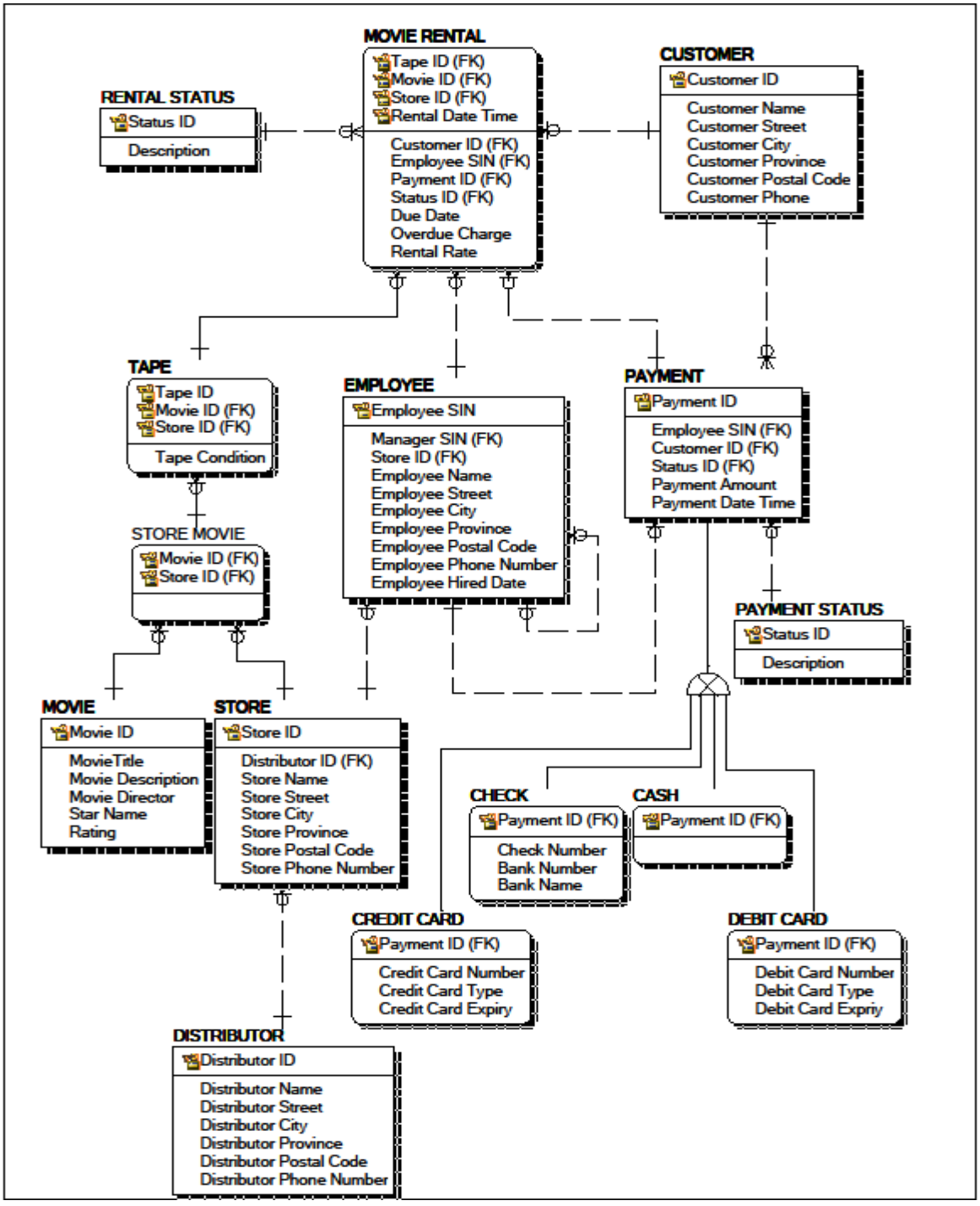


Figure 6: The Movie Rental Logical Model

5.3 Movie Rental Physical DB2 Model

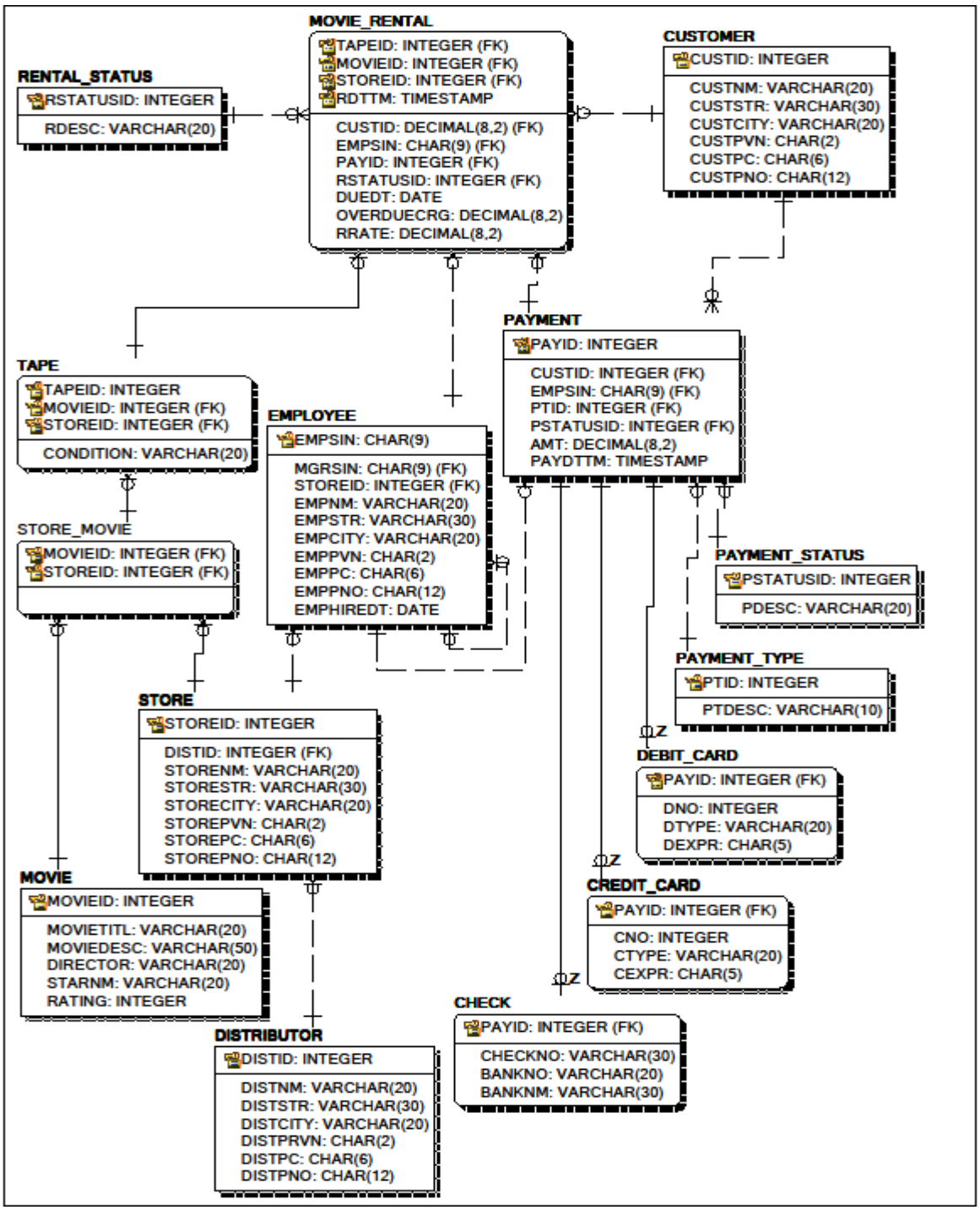


Figure 7: The Movie Rental Physical Model

5.4 Movie Rental DB2 Schema

```
CREATE TABLE DISTRIBUTOR(
       DISTID                INTEGER NOT NULL,
       DISTNM                VARCHAR(20),
       DISTSTR               VARCHAR(30),
       DISTCITY              VARCHAR(20),
       DISTPRVN              CHAR(2),
       DISTPC                CHAR(6),
       DISTPNO               CHAR(12),
       PRIMARY KEY (DISTID)
)
CREATE TABLE STORE(
       STORENM               VARCHAR(20),
       STORESTR              VARCHAR(30),
       STORECITY             VARCHAR(20),
       STOREPVN              CHAR(2),
       STOREPC               CHAR(6),
       STOREPNO              CHAR(12),
       STOREID               INTEGER NOT NULL,
       DISTID                INTEGER NOT NULL,
       PRIMARY KEY (STOREID),
       FOREIGN KEY (DISTID) REFERENCES DISTRIBUTOR  (DISTID)
)
CREATE TABLE EMPLOYEE(
       EMPSIN                CHAR(9) NOT NULL,
       MGRSIN                CHAR(9),
       EMPNM                 VARCHAR(20),
       EMPSTR                VARCHAR(30),
       EMPCITY               VARCHAR(20),
       EMPPVN                CHAR(2),
       EMPPC                 CHAR(6),
       EMPPNO                CHAR(12),
       EMPHIREDT             DATE,
       STOREID               INTEGER NOT NULL,
       PRIMARY KEY (EMPSIN),
       FOREIGN KEY (MGRSIN) REFERENCES EMPLOYEE  (EMPSIN),
       FOREIGN KEY (STOREID) REFERENCES STORE  (STOREID)
)
CREATE TABLE CUSTOMER(
       CUSTID                INTEGER NOT NULL,
       CUSTNM                VARCHAR(20),
       CUSTSTR               VARCHAR(30),
       CUSTCITY              VARCHAR(20),
       CUSTPVN               CHAR(2),
       CUSTPC                CHAR(6),
       CUSTPNO               CHAR(12),
       PRIMARY KEY (CUSTID)
)
CREATE TABLE MOVIE(
       MOVIETITL             VARCHAR(20),
       MOVIEDESC             VARCHAR(50),
```

```
        DIRECTOR              VARCHAR(20),
        RATING                INTEGER,
        STARNM                VARCHAR(20),
        MOVIEID               INTEGER NOT NULL,
        PRIMARY KEY (MOVIEID)
)
CREATE TABLE STORE_MOVIE(
        MOVIEID               INTEGER NOT NULL,
        STOREID               INTEGER NOT NULL,
        PRIMARY KEY (MOVIEID, STOREID),
        FOREIGN KEY (MOVIEID) REFERENCES MOVIE  (MOVIEID),
        FOREIGN KEY (STOREID) REFERENCES STORE  (STOREID)
)
CREATE TABLE TAPE(
        TAPEID                INTEGER NOT NULL,
        CONDITION             VARCHAR(20),
        MOVIEID               INTEGER NOT NULL,
        STOREID               INTEGER NOT NULL,
        PRIMARY KEY (TAPEID, MOVIEID, STOREID),
        FOREIGN KEY (MOVIEID, STOREID) REFERENCES STORE_MOVIE  (MOVIEID,
        STOREID)
)
CREATE TABLE PAYMENT_STATUS(
        PSTATUSID             INTEGER NOT NULL,
        PDESC                 VARCHAR(20),
        PRIMARY KEY (PSTATUSID)
)
CREATE TABLE PAYMENT_TYPE(
        PTID                 INTEGER NOT NULL,
        PTDESC               VARCHAR(10),
        PRIMARY KEY (PTID)
)
CREATE TABLE PAYMENT(
        PAYID                 INTEGER NOT NULL,
        AMT                   DECIMAL(8, 2),
        PAYDTTM               TIMESTAMP,
        EMPSIN                CHAR(9) NOT NULL,
        CUSTID                INTEGER NOT NULL,
        PSTATUSID             INTEGER NOT NULL,
        PTID                  INTEGER NOT NULL,
        PRIMARY KEY (PAYID),
        FOREIGN KEY (EMPSIN) REFERENCES EMPLOYEE  (EMPSIN),
        FOREIGN KEY (CUSTID) REFERENCES CUSTOMER  (CUSTID),
        FOREIGN KEY (PSTATUSID) REFERENCES PAYMENT_STATUS  (PSTATUSID),
        FOREIGN KEY (PTID) REFERENCES PAYMENT_TYPE  (PTID)
)
CREATE TABLE RENTAL_STATUS(
        RSTATUSID             INTEGER NOT NULL,
        RDESC                 VARCHAR(20),
        PRIMARY KEY (RSTATUSID)
)
CREATE TABLE MOVIE_RENTAL(
        DUEDT                 DATE,
        OVERDUECRG            DECIMAL(8, 2),
```

```
        RDTTM                   TIMESTAMP NOT NULL,
        EMPSIN                  CHAR(9) NOT NULL,
        CUSTID                  INTEGER NOT NULL,
        TAPEID                  INTEGER NOT NULL,
        RRATE                   DECIMAL(8, 2),
        PAYID                   INTEGER,
        MOVIEID                 INTEGER NOT NULL,
        STOREID                 INTEGER NOT NULL,
        RSTATUSID               INTEGER NOT NULL,
        PRIMARY KEY (TAPEID, MOVIEID, STOREID, RDTTM),
        FOREIGN KEY (EMPSIN) REFERENCES EMPLOYEE  (EMPSIN),
        FOREIGN KEY (CUSTID) REFERENCES CUSTOMER  (CUSTID),
        FOREIGN KEY (TAPEID, MOVIEID, STOREID) REFERENCES TAPE  (TAPEID,
                    MOVIEID, STOREID),
        FOREIGN KEY (PAYID) REFERENCES PAYMENT  (PAYID),
        FOREIGN KEY (RSTATUSID) REFERENCES RENTAL_STATUS  (RSTATUSID)
)
CREATE TABLE DEBIT_CARD(
        DNO                     INTEGER,
        DTYPE                   VARCHAR(20),
        DEXPR                   CHAR(5),
        PAYID                   INTEGER NOT NULL,
        PRIMARY KEY (PAYID),
        FOREIGN KEY (PAYID) REFERENCES PAYMENT  (PAYID)
)
CREATE TABLE CREDIT_CARD(
        CNO                     INTEGER,
        CTYPE                   VARCHAR(20),
        CEXPR                   CHAR(5),
        PAYID                   INTEGER NOT NULL,
        PRIMARY KEY (PAYID),
        FOREIGN KEY (PAYID) REFERENCES PAYMENT  (PAYID)
)
CREATE TABLE CHECK(
        CHECKNO                 VARCHAR(30),
        BANKNO                  VARCHAR(20),
        BANKNM                  VARCHAR(30),
        PAYID                   INTEGER NOT NULL,
        PRIMARY KEY (PAYID),
        FOREIGN KEY (PAYID) REFERENCES PAYMENT  (PAYID)
)
```

5.5 Movie Rental Interactive Queries

Query 1: Give all the customers who lives in Hamilton. Display customer_id and customer_name.

```
SELECT CUSTID AS CUSTOMER_ID,CUSTNM AS CUSTOMER_NAME
FROM CUSTOMER
WHERE CUSTCITY='Hamilton'
```

Query 2: Display the total payment are received by each employee, and sort by empsin.

```
SELECT EMPSIN AS EMPLOYEE_SIN,SUM(AMT) AS TOTAL_AMT
FROM PAYMENT
GROUP BY EMPSIN
ORDER BY EMPSIN
```

Query 3: Display the total movies are rented out by each store, and sort by storeid.

```
SELECT STOREID,COUNT(TAPEID) AS MOVIE_RENTED
FROM MOVIE_RENTAL
GROUP BY STOREID
ORDER BY STOREID
```

Query 4: Display all the tapes are never rented out in every store, and sort by movieid & tapeid.

```
SELECT DISTINCT T.MOVIEID,T.TAPEID
FROM TAPE T
LEFT JOIN MOVIE_RENTAL R ON T.MOVIEID=R.MOVIEID AND T.TAPEID=R.TAPEID
WHERE R.MOVIEID IS NULL
ORDER BY T.MOVIEID,T.TAPEID
```

Query 5: Display all customers who did not rent any movie so far and sort by custid.

```
SELECT CUSTID,CUSTNM
FROM CUSTOMER
WHERE CUSTID NOT IN (SELECT DISTINCT CUSTID FROM MOVIE_RENTAL)
ORDER BY CUSTID
```

Query 6: Display the total amount received by different payment type, and sort by ptdesc.

```
SELECT TP.PTDESC AS PAYMENT_TYPE,SUM(AMT) AS TOTAL_AMT
FROM PAYMENT P
INNER JOIN PAYMENT_TYPE TP ON P.PTID = TP.PTID
GROUP BY TP.PTDESC
ORDER BY TP.PTDESC
```

Query 7. Display the number of movies rented out based on the movie rating, and sort by rating.

```
SELECT M.RATING,COUNT(MR.MOVIEID) AS NO_OF_MOVIES
FROM MOVIE_RENTAL MR
INNER JOIN MOVIE M ON MR.MOVIEID = M.MOVIEID
GROUP BY M.RATING
```

Query 8: Display top 5 customers based on their total payment, and sort their total payment decreased.

```
SELECT CUSTID,SUM(AMT) AS TOTAL_AMT
```

```
FROM PAYMENT
GROUP BY CUSTID
ORDER BY TOTAL_AMT DESC
FETCH FIRST 5 ROWS ONLY
```

Query 9: List all the movies customer rented. Please display the columns: movie_title, rental_status, rental_rate, rental_employee, the employ accept the payment, payment_type and payment_status.

```
SELECT M.MOVIETITL AS MOVIE_TITLE,
       RS.RDESC AS RENTAL_STATUS,
       MR.RRATE AS RENTAL_RATE,
       E1.EMPNM AS RENTAL_EMPLOYEE,
       E2.EMPNM AS CASHIER_EMPLOYEE,
       PT.PTDESC AS PAYMENT_TYPE,
       PS.PDESC AS PAYMENT_STATUS
FROM MOVIE_RENTAL MR
INNER JOIN CUSTOMER C ON MR.CUSTID = C.CUSTID
INNER JOIN MOVIE M ON MR.MOVIEID = M.MOVIEID
INNER JOIN RENTAL_STATUS RS ON MR.RSTATUSID = RS.RSTATUSID
INNER JOIN EMPLOYEE E1 ON MR.EMPSIN = E1.EMPSIN
LEFT JOIN PAYMENT P ON MR.PAYID = P.PAYID
LEFT JOIN EMPLOYEE E2 ON P.EMPSIN = E2.EMPSIN
LEFT JOIN PAYMENT_STATUS PS ON P.PSTATUSID = PS.PSTATUSID
LEFT JOIN PAYMENT_TYPE PT ON P.PTID = PT.PTID
WHERE C.CUSTNM = 'customer1'
```

Query 10: List all the manager's name and the name of employee they manage. Please sort by manager sin & employee sin.

```
SELECT MGR.EMPNM AS MANAGER_NAME,E.EMPNM AS EMPLOYEE_NAME
FROM EMPLOYEE E
INNER JOIN EMPLOYEE MGR ON E.MGRSIN = MGR.EMPSIN
ORDER BY MGR.EMPSIN,E.EMPSIN
```

Chapter 6: Car Rental

6.1 Car Rental Informal Description

Our company does car rental business and has several locations with different address (address consist of street or rural route with the number, city, province and postal code). The cars are classified as subcompacts, compacts, sedans, or luxury. Each car has a particular make, model, year made, and color. Each car has a unique identification number and a unique license plate.

The cars rented in a particular location may be returned to a different location (so-called drop off). For every car we keep the odometer reading before it is rented and after it is returned. Since we trust our customers, we do not record the defect when the car is rent out and returned back. However, we rent the car with full tank and record the volume of gas in the tank when the car is returned, but we only indicate if the tank is empty, quarter full, half full, three quarters full, or full.

We keep track of which day a car was rented, but not of the time, similarly for car returning. If a customer requests a specific class (say sedan), we may rent the customer a higher-class car if we do not have the requested class in the stock, but we will price it at the level the customer requested (so-called upgrade). Each car class has its own pricing, but all cars in the same class are priced the same. We have rental policies for 1 day, 1 week, 2 weeks, and 1 month. Thus, if a customer rents a car for 8 days, it will be priced as 1 week + 1 day. The drop-off charge only depends on the class of the rented car, the location it was rented from and the location it is returned to.

About our customers, we keep their names, addresses, possibly all phone numbers, and the number of the driver's license (we assume a unique license per person). About our employees we keep the same information (we require that all our employees have a driver's license). We have several categories of workers, drivers, cleaners, clerks, and managers. Any of our employees can rent a car from our company for a 50% discount, if the rental is less than 2 weeks. However, for any longer rental they must pay 90% of the regular price. Every employee works in one location only. We have headquarters in Hamilton. The people who work there are all classified as managers, one of them is the president, two of them are the vice-presidents, one for operation, the other for marketing).

For certain weeks we have promotional rentals that are usually 60% of the regular price, but may be also of different percentage. They always affect only a single class of cars – i.e. we may have a promotion for subcompacts, but during that week we do not have any promotions for compacts, sedans or luxury cars. During some years we can have many promotions, in some we have none. The promotions cannot be applied to the employees.

6.2 Car Rental Logical Model

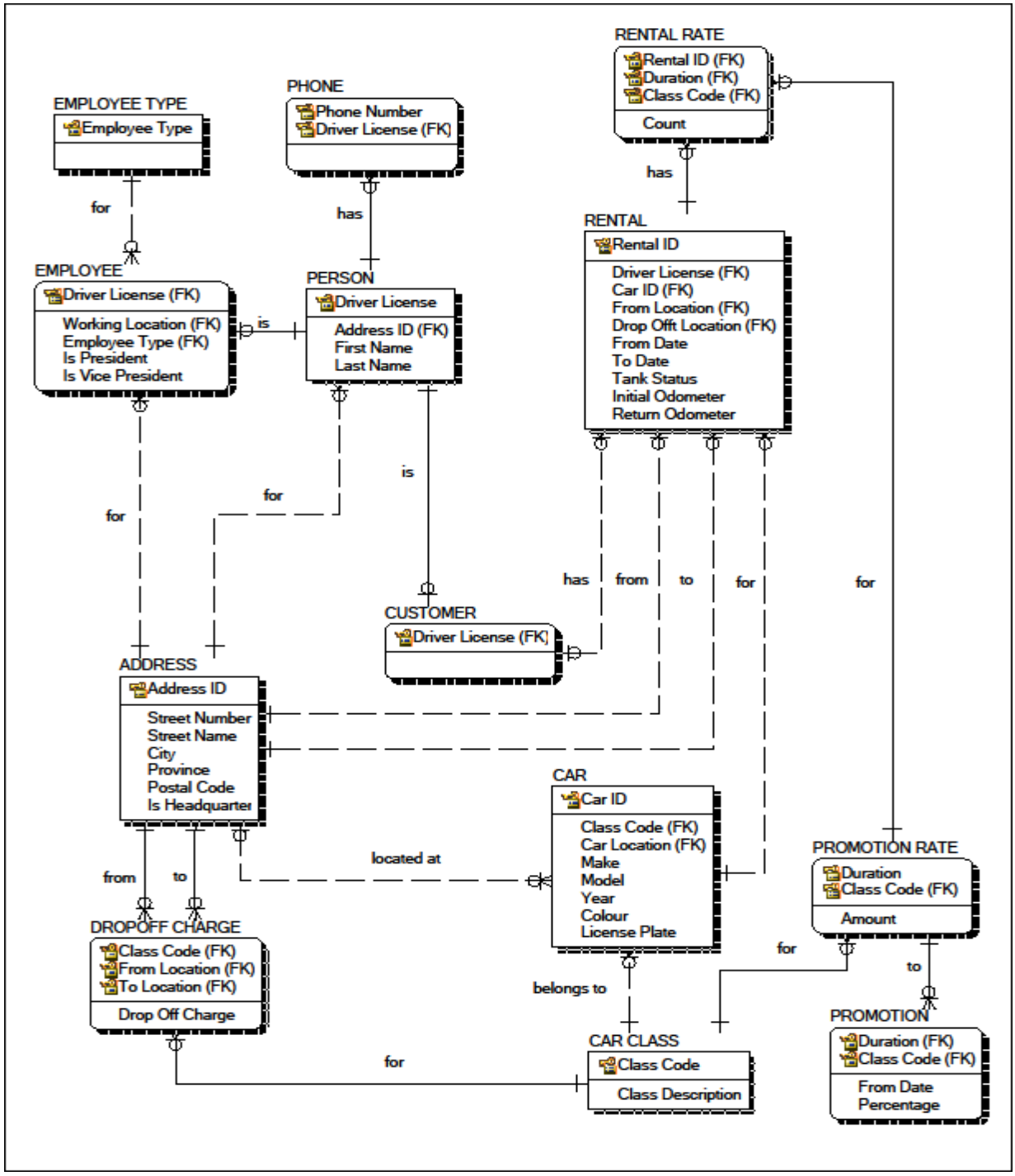


Figure 8: The Car Rental Logic Model

6.3 Car Rental Physical DB2 Model

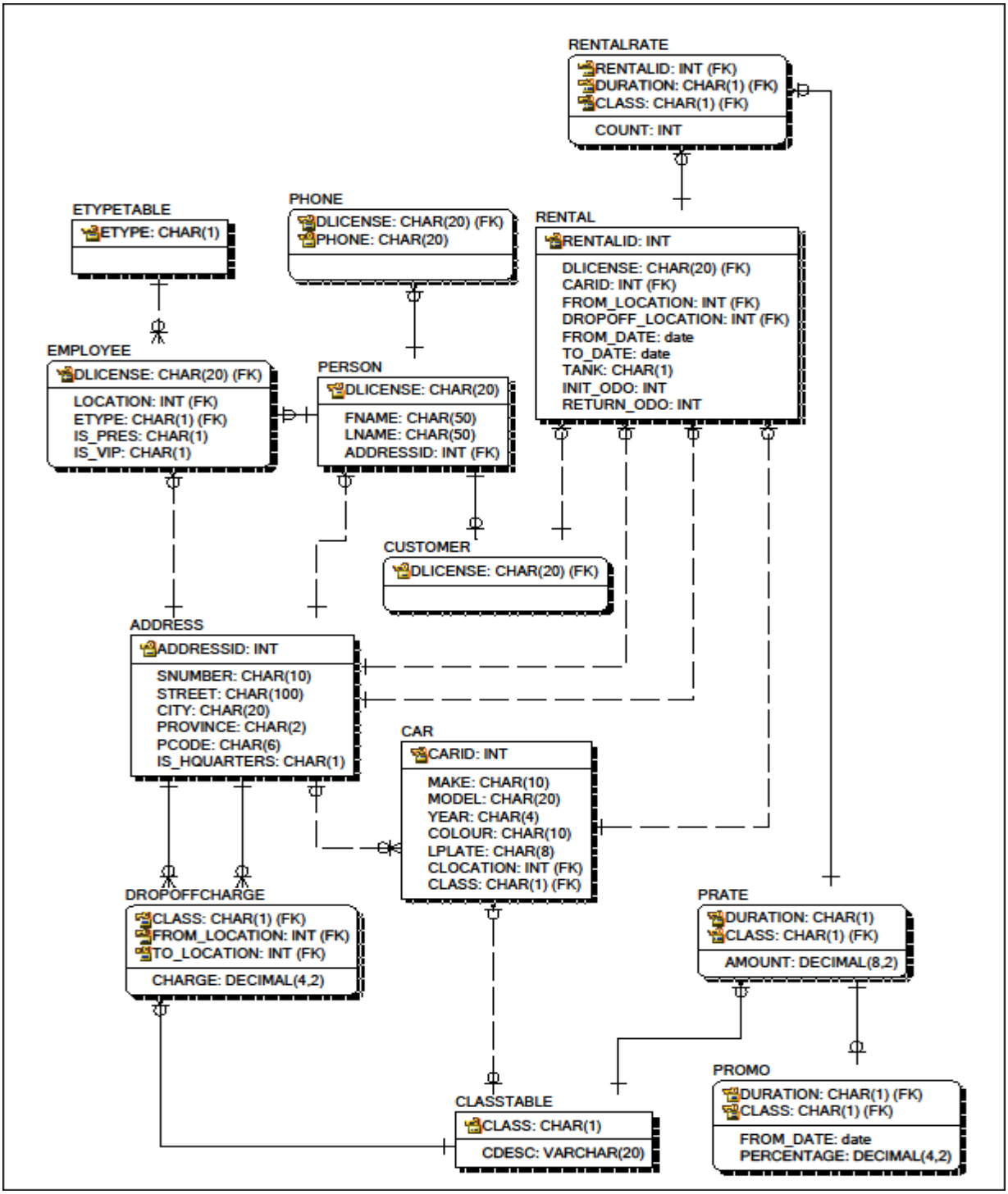


Figure 9: The Car Rental Physical Model

6.4 Car Rental DB2 Schema

```
CREATE TABLE ADDRESS (
       ADDRESSID INTEGER NOT NULL,
       SNUMBER CHAR(10) NOT NULL,
       STREET CHAR(100) NOT NULL,
       CITY CHAR(20) NOT NULL,
       PROVINCE CHAR(2) NOT NULL,
       PCODE CHAR(6) NOT NULL,
       IS_HQUARTERS CHAR(1) NOT NULL,
       PRIMARY KEY (ADDRESSID),
       UNIQUE (SNUMBER,STREET,CITY,PROVINCE),
       CHECK(ADDRESSID > 0),
       CHECK(PROVINCE IN ('AL', 'BC','MA','NB','NL','NT','NS','NU',
                          'ON','PE','QU','SA','YT')),
       CHECK (IS_HQUARTERS IN ('H','0'))
)
CREATE TABLE CLASSTABLE (
       CLASS CHAR(1) NOT NULL,
       CDESC VARCHAR(20) NOT NULL,
       PRIMARY KEY (CLASS),
       CHECK (CLASS IN ('B','C','S','L'))
)
CREATE TABLE CAR (
       CARID INTEGER NOT NULL,
       MAKE CHAR(10) NOT NULL,
       MODEL CHAR(20) NOT NULL,
       YEAR CHAR(4) NOT NULL,
       COLOUR CHAR(10) NOT NULL,
       LPLATE CHAR(8) NOT NULL,
       CLOCATION INTEGER,
       CLASS CHAR(1),
       PRIMARY KEY (carid),
       FOREIGN KEY (CLOCATION) REFERENCES ADDRESS (ADDRESSID),
       FOREIGN KEY (CLASS) REFERENCES CLASSTABLE,
       CHECK (CARID > 0)
)
CREATE TABLE PERSON (
       DLICENSE CHAR(20) NOT NULL,
       FNAME CHAR(50) NOT NULL,
       LNAME CHAR(50) NOT NULL,
       ADDRESSID INTEGER NOT NULL,
       PRIMARY KEY (DLICENSE),
       FOREIGN KEY (ADDRESSID) REFERENCES ADDRESS
)
CREATE TABLE PHONE (
       DLICENSE CHAR(20) NOT NULL,
       PHONE CHAR(20) NOT NULL,
       PRIMARY KEY (DLICENSE,PHONE),
       FOREIGN KEY (DLICENSE) REFERENCES PERSON
)
CREATE TABLE ETYPETABLE (
       ETYPE CHAR(1) NOT NULL,
```

```
        PRIMARY KEY (ETYPE),
        CHECK (ETYPE IN ('D', 'C', 'K', 'M'))
)
CREATE TABLE EMPLOYEE (
        DLICENSE CHAR(20) NOT NULL,
        LOCATION INTEGER NOT NULL,
        ETYPE CHAR(1) NOT NULL,
        IS_PRES CHAR(1) NOT NULL,
        IS_VIP CHAR(1) NOT NULL,
        PRIMARY KEY (DLICENSE),
        FOREIGN KEY (DLICENSE) REFERENCES PERSON,
        FOREIGN KEY (LOCATION) REFERENCES ADDRESS (ADDRESSID),
        FOREIGN KEY (ETYPE) REFERENCES ETYPETABLE,
        CHECK (IS_PRES IN ('P','0')),
        CHECK (IS_VIP IN ('M','P','0')),
        CHECK ((IS_PRES='P' AND ETYPE='M') OR
               (IS_VIP='M' AND ETYPE='M') OR
               (IS_VIP='P' AND ETYPE='M') OR
               (IS_VIP='0' AND IS_PRES='0')),
        CHECK ((IS_PRES='P' AND IS_VIP='0') OR
               (IS_PRES='0' AND IS_VIP='M') OR
               (IS_PRES='0' AND IS_VIP='P') OR
               (IS_PRES='0' AND IS_VIP='0'))
)
CREATE TABLE CUSTOMER (
        DLICENSE CHAR(20) NOT NULL,
        PRIMARY KEY (DLICENSE),
        FOREIGN KEY (DLICENSE) REFERENCES PERSON
)
CREATE TABLE PRATE (
        DURATION CHAR(1) NOT NULL,
        CLASS CHAR(1) NOT NULL,
        AMOUNT DECIMAL(8,2) NOT NULL,
        PRIMARY KEY (DURATION,CLASS),
        FOREIGN KEY (CLASS) REFERENCES CLASSTABLE,
        CHECK (DURATION IN ('D', 'W', 'T', 'M'))
)
CREATE TABLE PROMO (
        DURATION CHAR(1) NOT NULL,
        CLASS CHAR(1) NOT NULL,
        FROM_DATE DATE NOT NULL,
        PERCENTAGE DECIMAL(4,2) NOT NULL,
        PRIMARY KEY (DURATION,CLASS),
        FOREIGN KEY (DURATION,CLASS) REFERENCES PRATE
)
CREATE TABLE RENTAL (
        RENTALID INTEGER NOT NULL,
        DLICENSE CHAR(20) NOT NULL,
        CARID INTEGER NOT NULL,
        FROM_LOCATION INTEGER NOT NULL,
        DROPOFF_LOCATION INTEGER NOT NULL,
        FROM_DATE DATE NOT NULL,
        TO_DATE DATE,
        TANK CHAR(1) NOT NULL,
```

```
        INIT_ODO INTEGER NOT NULL,
        RETURN_ODO INTEGER,
        PRIMARY KEY (RENTALID),
        UNIQUE (CARID,FROM_DATE),
        FOREIGN KEY (CARID) REFERENCES CAR,
        FOREIGN KEY (DLICENSE) REFERENCES CUSTOMER,
        FOREIGN KEY (FROM_LOCATION) REFERENCES ADDRESS (ADDRESSID),
        FOREIGN KEY (DROPOFF_LOCATION) REFERENCES ADDRESS (ADDRESSID),
        CHECK (RENTALID > 0),
        CHECK (TANK IN ('F','T','H','Q','E')),
        CHECK ((FROM_DATE < TO_DATE) OR (TO_DATE IS NULL)),
        CHECK ((INIT_ODO < RETURN_ODO) OR (RETURN_ODO IS NULL))
)
CREATE TABLE RENTALRATE (
        RENTALID INTEGER NOT NULL,
        DURATION CHAR(1) NOT NULL,
        CLASS CHAR(1) NOT NULL,
        COUNT INTEGER NOT NULL,
        PRIMARY KEY (RENTALID,DURATION,CLASS),
        FOREIGN KEY (RENTALID) REFERENCES RENTAL,
        FOREIGN KEY (DURATION,CLASS) REFERENCES PRATE,
        CHECK (COUNT > 0)
)
CREATE TABLE DROPOFFCHARGE (
        CLASS CHAR(1) NOT NULL,
        FROM_LOCATION INTEGER NOT NULL,
        TO_LOCATION INTEGER NOT NULL,
        CHARGE DECIMAL(4,2) NOT NULL,
        PRIMARY KEY (CLASS,FROM_LOCATION,TO_LOCATION),
        FOREIGN KEY (CLASS) REFERENCES CLASSTABLE,
        FOREIGN KEY (FROM_LOCATION) REFERENCES ADDRESS (ADDRESSID),
        FOREIGN KEY (TO_LOCATION) REFERENCES ADDRESS (ADDRESSID)
)
```

6.5 Car Rental Interactive Queries

Query 1: Give last name of all customers who are now renting a car from our company.

```
SELECT LNAME
FROM CUSTOMER, PERSON, RENTAL
WHERE CUSTOMER.DLICENSE=RENTAL.DLICENSE AND CUSTOMER.DLICENSE =
PERSON.DLICENSE AND TO_DATE IS NULL
```

Query 2: Give make and color of all cars currently rented out.

```
SELECT MAKE,COLOUR
FROM CAR,RENTAL
WHERE CAR.CARID=RENTAL.CARID AND TO_DATE IS NULL
```

Query 3: For each completed rental, give the rental price and rental_id.

```
SELECT SUM(CHARGE), RENTALID
FROM (SELECT RENTALID,AMOUNT*COUNT
      FROM RENTALRATE,PRATE
      WHERE RENTALRATE.DURATION=PRATE.DURATION
      AND RENTALRATE.CLASS=PRATE.CLASS) AS T(RENTALID,CHARGE)
GROUP BY RENTALID
```

Query 4: List last name of all managers.

```
SELECT LNAME
FROM EMPLOYEE, PERSON
WHERE ETYPE='M' AND EMPLOYEE.DLICENSE=PERSON.DLICENSE
```

Query 5: List last and first names of all customers.

```
SELECT LNAME, FNAME
FROM CUSTOMER, PERSON
WHERE CUSTOMER.DLICENSE=PERSON.DLICENSE
```

Query 6: Give a query that answers the question "Is any of our employee also our customer"?

```
SELECT *
FROM EMPLOYEE, CUSTOMER
WHERE EMPLOYEE.DLICENSE=CUSTOMER.DLICENSE
```

Query 7: Does our president work in the headquarters?

```
SELECT *
FROM EMPLOYEE, ADDRESS
WHERE IS_PRES='P' AND LOCATION=ADDRESSID AND ADDRESS.IS_HQUARTERS='H'
```

Query 8: Find rental_id of all shortest (completed) rentals.

```
SELECT RENTALID
FROM RENTAL
WHERE TO_DATE IS NOT NULL AND (DAYS(TO_DATE)-DAYS(FROM_DATE)) IN
      (SELECT MIN(DAYS(TO_DATE) - DAYS(FROM_DATE))
       FROM RENTAL
       WHERE TO_DATE IS NOT NULL)
```

Query 9: Find the value of the cheapest (completed) rental. We will utilize query 3 as the inner query.

```
SELECT MIN(CHARGE)
FROM (SELECT SUM(CHARGE), RENTALID
      FROM (SELECT RENTALID,AMOUNT*COUNT
            FROM RENTALRATE,PRATE
            WHERE RENTALRATE.DURATION=PRATE.DURATION
            AND RENTALRATE.CLASS=PRATE.CLASS) AS T(RENTALID,CHARGE)
```

```
GROUP BY RENTALID) AS T1(CHARGE,RENTALID)
```

Query 10: Give makes of the cars that have never been rented.

```
SELECT DISTINCT MAKE FROM CAR
EXCEPT
SELECT DISTINCT MAKE FROM CAR, RENTAL WHERE CAR.CARID=RENTAL.CARID
```

Chapter 7: Course Registration

7.1 Course Registration Informal Description

In our college, students need to register for courses before new semester start. All of programs in our college take four years to finish.

Each course has a unique designation, title, description, year (in which year of study the course is to be taken, for instance 2nd year course), and classroom. A course can have no or many tutorial sections, and no or many lab sections. Each course is taught by exactly one instructor. Each instructor has a unique id, name, departmental affiliation, office room, phone extension, and a unique email address. Each student has a unique id, name, and the year of his/her study. A student cannot be an instructor. A course can have zero or many tutorial sections unique to the course (i.e. tutorial sections are not shared by different courses). Each tutorial section has exactly one TA assigned. A TA can tutor more than one tutorial section for the same course, and any number of tutorials for different courses. A TA cannot be an instructor, however a student can work as a TA (in that case his/her student id is used as TA id). A course can have zero or many lab sections unique to the course (i.e. lab sections are not shared by different courses). Each lab section has exactly one LA assigned. An LA can oversee more than one lab section for the same course, and any number of labs for different courses. An LA cannot be an instructor, however a student can work as a LA (in which case his/her student id is used as the LA id). In fact, a student can work as a TA and an LA simultaneously. Thus, a TA may or may not be a student, an LA may or may not be a student. A person can work as both, a TA and a LA. TA has the same attributes as instructor, the same goes for LA.

Each course has zero to many courses designated as its prerequisites and zero to many courses designated as its anti-requisites. Prerequisite courses are of the same or lower year, anti-requisite courses are of the same year. In the system we keep information of what courses a student has taken and what courses the student is registering. All courses are either Pass or Fail. A student can register a course only if he/she has passed all the prerequisites and has not passed any or is not registered in any of the anti-requisites. A student can only register a course of the appropriate year, i.e. a student in year X of study can only register and take course of year X.

7.2 Course Registration Logical Model

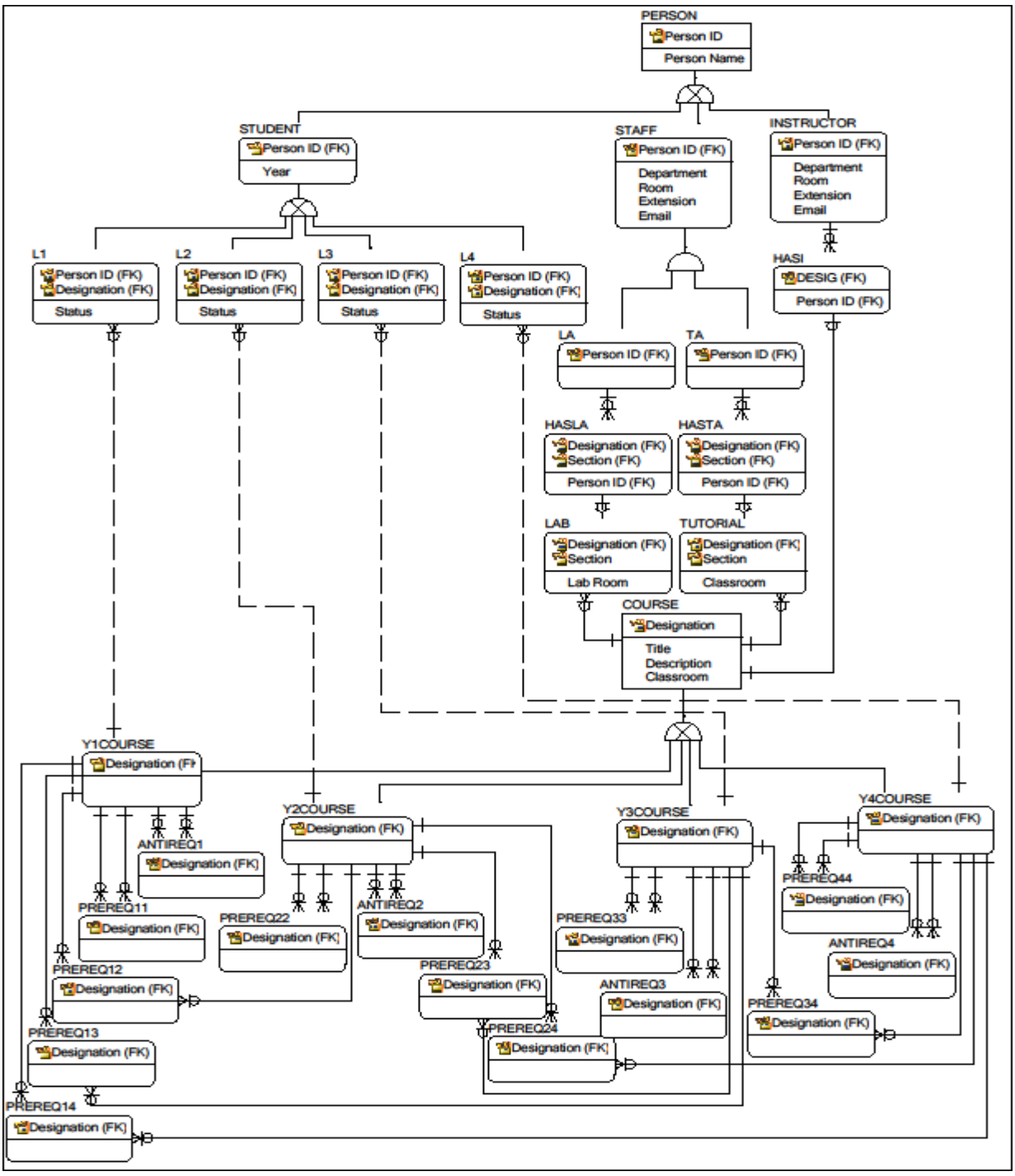


Figure 10:The Course Registration Logical Model

7.3 Course Registration Physical DB2 Model

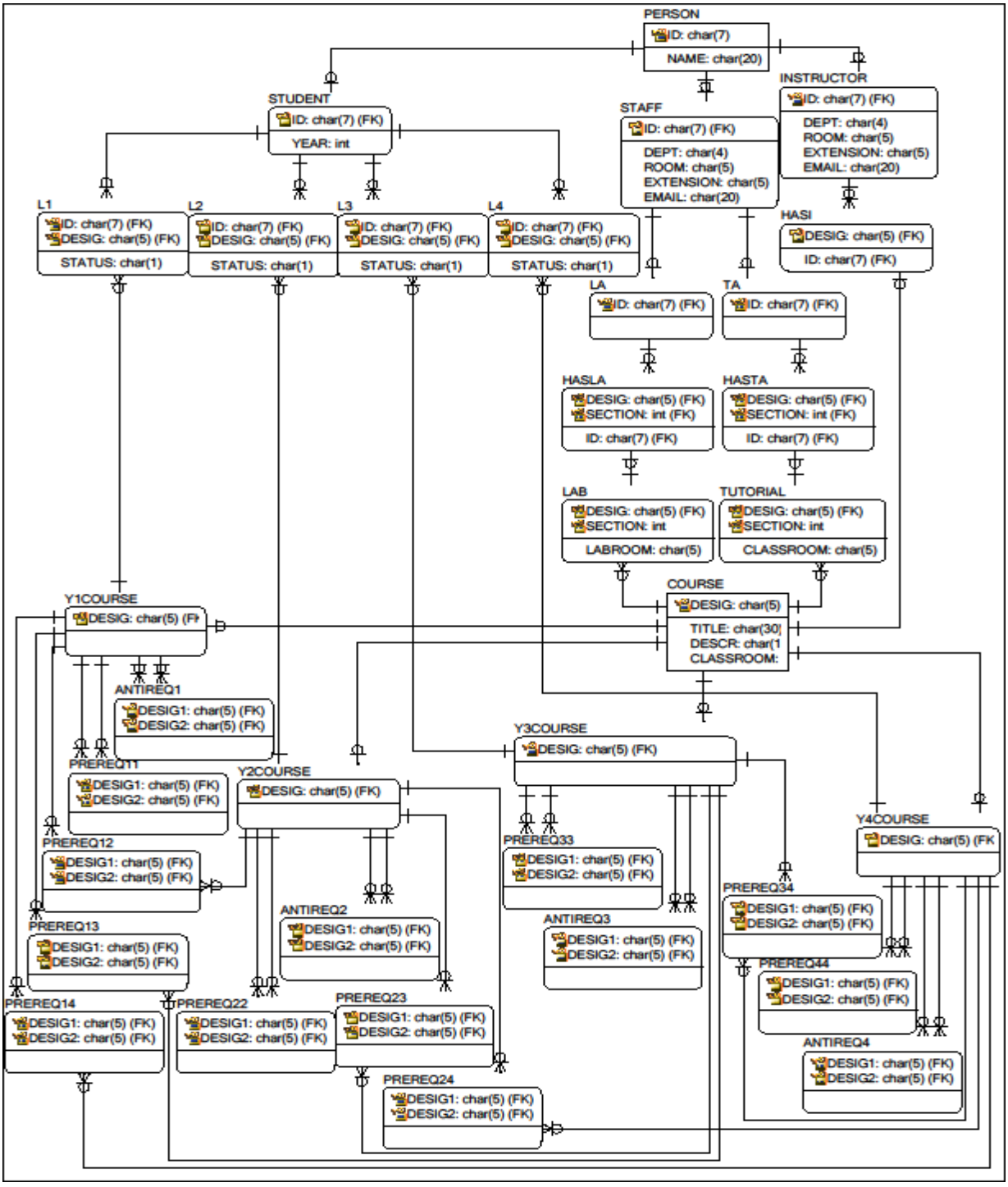


Figure 11: The Course Registration Physical Model

7.4 Course Registration DB2 Schema

```
CREATE TABLE COURSE (
       DESIG CHAR(5) NOT NULL,
       TITLE CHAR(30) NOT NULL,
       DESCR  CHAR(150) NOT NULL,
       CLASSROOM CHAR(5) NOT NULL,
       PRIMARY KEY(DESIG)
)
CREATE TABLE Y1COURSE (
       DESIG CHAR(5) NOT NULL,
       PRIMARY KEY(DESIG)
)
CREATE TABLE Y2COURSE (
       DESIG CHAR(5) NOT NULL,
       PRIMARY KEY(DESIG)
)
CREATE TABLE Y3COURSE (
       DESIG CHAR(5) NOT NULL,
       PRIMARY KEY(DESIG)
)
CREATE TABLE Y4COURSE (
       DESIG CHAR(5) NOT NULL,
       PRIMARY KEY(DESIG)
)
CREATE TABLE PREREQ11 (
       DESIG1 CHAR(5) NOT NULL,
       DESIG2 CHAR(5) NOT NULL,
       PRIMARY KEY(DESIG1,DESIG2)
)
CREATE TABLE PREREQ12 (
       DESIG1 CHAR(5) NOT NULL,
       DESIG2 CHAR(5) NOT NULL,
       PRIMARY KEY(DESIG1,DESIG2),
       FOREIGN KEY (DESIG1) REFERENCES Y1COURSE (DESIG),
       FOREIGN KEY (DESIG2) REFERENCES Y2COURSE (DESIG),
       CONSTRAINT PREREQ12_C3 CHECK (DESIG1<>DESIG2)
)
CREATE TABLE PREREQ13 (
       DESIG1 CHAR(5) NOT NULL,
       DESIG2 CHAR(5) NOT NULL,
       PRIMARY KEY(DESIG1,DESIG2),
       FOREIGN KEY (DESIG1) REFERENCES Y1COURSE (DESIG),
       FOREIGN KEY (DESIG2) REFERENCES Y3COURSE (DESIG),
       CONSTRAINT PREREQ13_C3 CHECK (DESIG1<>DESIG2)
)
CREATE TABLE PREREQ14 (
       DESIG1 CHAR(5) NOT NULL,
       DESIG2 CHAR(5) NOT NULL,
       PRIMARY KEY(DESIG1,DESIG2),
       FOREIGN KEY (DESIG1) REFERENCES Y1COURSE (DESIG),
       FOREIGN KEY (DESIG2) REFERENCES Y4COURSE (DESIG),
```

```
        CONSTRAINT PREREQ14_C3 CHECK (DESIG1<>DESIG2)
)
CREATE TABLE PREREQ22 (
        DESIG1 CHAR(5) NOT NULL,
        DESIG2 CHAR(5) NOT NULL,
        PRIMARY KEY(DESIG1,DESIG2),
        FOREIGN KEY (DESIG1) REFERENCES Y2COURSE (DESIG),
        FOREIGN KEY (DESIG2) REFERENCES Y2COURSE (DESIG),
        CONSTRAINT PREREQ22_C3 CHECK (DESIG1<>DESIG2)
)
CREATE TABLE PREREQ23 (
        DESIG1 CHAR(5) NOT NULL,
        DESIG2 CHAR(5) NOT NULL,
        PRIMARY KEY(DESIG1,DESIG2),
        FOREIGN KEY (DESIG1) REFERENCES Y2COURSE (DESIG),
        FOREIGN KEY (DESIG2) REFERENCES Y3COURSE (DESIG),
        CONSTRAINT PREREQ23_C3 CHECK (DESIG1<>DESIG2)
)
CREATE TABLE PREREQ24 (
        DESIG1 CHAR(5) NOT NULL,
        DESIG2 CHAR(5) NOT NULL,
        PRIMARY KEY(DESIG1,DESIG2),
        FOREIGN KEY (DESIG1) REFERENCES Y2COURSE (DESIG),
        FOREIGN KEY (DESIG2) REFERENCES Y4COURSE (DESIG),
        CONSTRAINT PREREQ24_C3 CHECK (DESIG1<>DESIG2)
)
CREATE TABLE PREREQ33 (
        DESIG1 CHAR(5) NOT NULL,
        DESIG2 CHAR(5) NOT NULL,
        PRIMARY KEY(DESIG1,DESIG2),
        FOREIGN KEY (DESIG1) REFERENCES Y3COURSE (DESIG),
        FOREIGN KEY (DESIG2) REFERENCES Y3COURSE (DESIG),
        CONSTRAINT PREREQ33_C3 CHECK (DESIG1<>DESIG2)
)
CREATE TABLE PREREQ34 (
        DESIG1 CHAR(5) NOT NULL,
        DESIG2 CHAR(5) NOT NULL,
        PRIMARY KEY(DESIG1,DESIG2),
        FOREIGN KEY (DESIG1) REFERENCES Y3COURSE (DESIG),
        FOREIGN KEY (DESIG2) REFERENCES Y4COURSE (DESIG),
        CONSTRAINT PREREQ34_C3 CHECK (DESIG1<>DESIG2)
)
CREATE TABLE PREREQ44 (
        DESIG1 CHAR(5) NOT NULL,
        DESIG2 CHAR(5) NOT NULL,
        PRIMARY KEY(DESIG1,DESIG2),
        FOREIGN KEY (DESIG1) REFERENCES Y4COURSE (DESIG),
        FOREIGN KEY (DESIG2) REFERENCES Y4COURSE (DESIG),
        CONSTRAINT PREREQ44_C3 CHECK (DESIG1<>DESIG2)
)
CREATE TABLE ANTIREQ1 (
        DESIG1 CHAR(5) NOT NULL,
        DESIG2 CHAR(5) NOT NULL,
        PRIMARY KEY(DESIG1,DESIG2),
```

```
        FOREIGN KEY (DESIG1) REFERENCES Y1COURSE (DESIG),
        FOREIGN KEY (DESIG2) REFERENCES Y1COURSE (DESIG),
        CONSTRAINT ANTIREQ1_C3 CHECK (DESIG1<>DESIG2)
)
CREATE TABLE ANTIREQ2 (
        DESIG1 CHAR(5) NOT NULL,
        DESIG2 CHAR(5) NOT NULL,
        PRIMARY KEY(DESIG1,DESIG2),
        FOREIGN KEY (DESIG1) REFERENCES Y2COURSE (DESIG),
        FOREIGN KEY (DESIG2) REFERENCES Y2COURSE (DESIG),
        CONSTRAINT ANTIREQ2_C3 CHECK (DESIG1<>DESIG2)
)
CREATE TABLE ANTIREQ3 (
        DESIG1 CHAR(5) NOT NULL,
        DESIG2 CHAR(5) NOT NULL,
        PRIMARY KEY(DESIG1,DESIG2),
        FOREIGN KEY (DESIG1) REFERENCES Y3COURSE (DESIG),
        FOREIGN KEY (DESIG2) REFERENCES Y3COURSE (DESIG),
        CONSTRAINT ANTIREQ3_C3 CHECK (DESIG1<>DESIG2)
)
CREATE TABLE ANTIREQ4 (
        DESIG1 CHAR(5) NOT NULL,
        DESIG2 CHAR(5) NOT NULL,
        PRIMARY KEY(DESIG1,DESIG2),
        FOREIGN KEY (DESIG1) REFERENCES Y4COURSE (DESIG),
        FOREIGN KEY (DESIG2) REFERENCES Y4COURSE (DESIG),
        CONSTRAINT ANTIREQ4_C3 CHECK (DESIG1<>DESIG2)
)
CREATE TABLE PERSON (
        ID CHAR(7) NOT NULL,
        NAME CHAR(20) NOT NULL,
        PRIMARY KEY(ID)
)
CREATE TABLE STUDENT (
        ID CHAR(7) NOT NULL,
        YEAR INTEGER NOT NULL,
        PRIMARY KEY(ID),
        FOREIGN KEY (ID) REFERENCES PERSON (ID)
)
CREATE TABLE INSTRUCTOR (
        ID CHAR(7) NOT NULL,
        DEPT CHAR(4) NOT NULL,
        ROOM CHAR(5) NOT NULL,
        EXTENSION CHAR(5) NOT NULL,
        EMAIL CHAR(20) NOT NULL,
        PRIMARY KEY(ID),
        FOREIGN KEY (ID) REFERENCES PERSON (ID),
        CONSTRAINT INSTRUCTOR_C2 UNIQUE (EMAIL)
)
CREATE TABLE STAFF (
        ID CHAR(7) NOT NULL,
        DEPT CHAR(4) NOT NULL,
        ROOM CHAR(5) NOT NULL,
        EXTENSION CHAR(5) NOT NULL,
```

```
        EMAIL CHAR(20) NOT NULL,
        PRIMARY KEY(ID),
        FOREIGN KEY (ID) REFERENCES PERSON (ID),
        CONSTRAINT STAFF_C2 UNIQUE (EMAIL)
)
CREATE TABLE LAB (
        DESIG CHAR(5) NOT NULL,
        SECTION INTEGER NOT NULL,
        LABROOM CHAR(5) NOT NULL,
        PRIMARY KEY(DESIG,SECTION),
        FOREIGN KEY (DESIG) REFERENCES COURSE (DESIG)
)
CREATE TABLE TUTORIAL (
        DESIG CHAR(5) NOT NULL,
        SECTION INTEGER NOT NULL,
        CLASSROOM CHAR(5) NOT NULL,
        PRIMARY KEY(DESIG,SECTION),
        FOREIGN KEY (DESIG) REFERENCES COURSE (DESIG)
)
CREATE TABLE TA (
        ID CHAR(7) NOT NULL,
        PRIMARY KEY(ID),
        FOREIGN KEY (ID) REFERENCES STAFF (ID)
)
CREATE TABLE LA (
        ID CHAR(7) NOT NULL,
        PRIMARY KEY(ID),
        FOREIGN KEY (ID) REFERENCES STAFF (ID)
)
CREATE TABLE HASTA (
        DESIG CHAR(5) NOT NULL,
        SECTION INTEGER NOT NULL,
        ID CHAR(7) NOT NULL,
        PRIMARY KEY(DESIG,SECTION),
        FOREIGN KEY (DESIG,SECTION) REFERENCES TUTORIAL (DESIG,SECTION),
        FOREIGN KEY (ID) REFERENCES TA (ID)
)
CREATE TABLE HASLA (
        DESIG CHAR(5) NOT NULL,
        SECTION INTEGER NOT NULL,
        ID CHAR(7) NOT NULL,
        PRIMARY KEY(DESIG,SECTION),
        FOREIGN KEY (DESIG,SECTION) REFERENCES LAB (DESIG,SECTION),
        FOREIGN KEY (ID) REFERENCES LA (ID)
)
CREATE TABLE HASI (
        DESIG CHAR(5) NOT NULL,
        ID CHAR(7) NOT NULL,
        PRIMARY KEY(DESIG),
        FOREIGN KEY (DESIG) REFERENCES COURSE (DESIG),
        FOREIGN KEY (ID) REFERENCES INSTRUCTOR (ID)
)
CREATE TABLE L1 (
        ID CHAR(7) NOT NULL,
```

```
        DESIG CHAR(5) NOT NULL,
        STATUS CHAR(1) NOT NULL CHECK (STATUS IN ('P','F','R')),
        PRIMARY KEY(ID,DESIG),
        FOREIGN KEY (ID) REFERENCES STUDENT (ID),
        FOREIGN KEY (DESIG) REFERENCES Y1COURSE (DESIG)
)
CREATE TABLE L2 (
        ID CHAR(7) NOT NULL,
        DESIG CHAR(5) NOT NULL,
        STATUS CHAR(1) NOT NULL CHECK (STATUS IN ('P','F','R')),
        PRIMARY KEY(ID,DESIG),
        FOREIGN KEY (ID) REFERENCES STUDENT (ID),
        FOREIGN KEY (DESIG) REFERENCES Y2COURSE (DESIG)
)
CREATE TABLE L3 (
        ID CHAR(7) NOT NULL,
        DESIG CHAR(5) NOT NULL,
        STATUS CHAR(1) NOT NULL CHECK (STATUS IN ('P','F','R')),
        PRIMARY KEY(ID,DESIG),
        FOREIGN KEY (ID) REFERENCES STUDENT (ID),
        FOREIGN KEY (DESIG) REFERENCES Y3COURSE (DESIG)
)
CREATE TABLE L4 (
        ID CHAR(7) NOT NULL,
        DESIG CHAR(5) NOT NULL,
        STATUS CHAR(1) NOT NULL CHECK (STATUS IN ('P','F','R')),
        PRIMARY KEY(ID,DESIG),
        FOREIGN KEY (ID) REFERENCES STUDENT (ID),
        FOREIGN KEY (DESIG) REFERENCES Y4COURSE (DESIG)
)
```

7.5 Course Registration Interactive Queries

Query 1: List all triples (student name, course designation, status) of courses taken or registered by students with id '0000041' and '0000042'. status is the status of each course (i.e. 'P' for passed, 'F' for failed, 'R' for registered).

```
SELECT  NAME,DESIG,STATUS
FROM  STUDENT, PERSON,L1
WHERE  STUDENT.ID=L1.ID
AND  (STUDENT.ID='0000041' OR STUDENT.ID='0000042')
AND  STUDENT.ID=PERSON.ID
UNION
SELECT  NAME,DESIG,STATUS
FROM  STUDENT, PERSON,L2
WHERE  STUDENT.ID=L2.ID
AND  (STUDENT.ID='0000041' OR STUDENT.ID='0000042')
AND  STUDENT.ID=PERSON.ID
```

Query 2: List names of all students in year 1.

```
SELECT PERSON.NAME
```

```
FROM PERSON, STUDENT
WHERE PERSON.ID=STUDENT.ID AND STUDENT.YEAR=1
```

Query 3: List names of all instructors teaching year 1 courses.

```
SELECT PERSON.NAME
FROM PERSON, INSTRUCTOR, HASI, Y1COURSE
WHERE PERSON.ID=INSTRUCTOR.ID
AND HASI.ID=INSTRUCTOR.ID AND HASI.DESIG=Y1COURSE.DESIG
```

Query 4: List designation of all courses that have tutorials. No designation can repeat.

```
SELECT DISTINCT DESIG FROM TUTORIAL
```

Query 5: List designation of all courses that have labs with more than 1 section. No designation can repeat.

```
SELECT DISTINCT DESIG FROM LAB WHERE SECTION=2
```

Query 6: List names of instructors that teach at least one course with multiple sections labs. No name can repeat.

```
SELECT DISTINCT NAME
FROM PERSON,INSTRUCTOR,HASI
WHERE (INSTRUCTOR.ID=PERSON.ID) AND (HASI.ID=PERSON.ID)
AND HASI.DESIG IN (SELECT COURSE.DESIG FROM COURSE,LAB
WHERE COURSE.DESIG=LAB.DESIG AND SECTION=2)
```

Query 7: List names of instructors that teach only courses with single-sections labs or no labs. No name can repeat.

```
SELECT DISTINCT NAME
FROM PERSON, INSTRUCTOR
WHERE (PERSON.ID=INSTRUCTOR.ID)
AND NAME NOT IN (SELECT DISTINCT NAME FROM PERSON,INSTRUCTOR,HASI
                 WHERE (INSTRUCTOR.ID=PERSON.ID) AND (HASI.ID=PERSON.ID)
                 AND HASI.DESIG IN (SELECT COURSE.DESIG FROM COURSE,LAB
                                    WHERE COURSE.DESIG=LAB.DESIG
                                    AND SECTION=2))
```

Chapter 8: Emergency Room

8.1 Emergency Room Informal Description

In our Emergency Room (ER), we have three distinct types of workers: receptionists, nurses, and doctors. Any of the workers can in fact be a patient. Each person in the proposed system, be it a patient or a worker has a last, a first, possibly a middle name, and one or more addresses. An address consists of a country, province, city, street and street number. Each person can have none or more email addresses, none or more telephone numbers.

The workers work in ER in shifts. A shift consists of start and end time. The shifts do not overlap, but they are consecutive, i.e. there is a shift on at any given time and day. We are assuming that the model we are creating (and eventually the database we will design) covers some extended period of time. Each worker will thus be assigned to many shifts in that period. Exactly two receptionists are assigned to each shift, a group of two or more nurses is assigned to each shift, a group of two or more doctors is assigned to each shift, one of the doctors assigned to a shift is the shift's triage doctor.

When a patient comes to ER, it happens during a particular shift. The patient is admitted by a particular receptionist, is seen by the triage doctor of the shift. The patient may be send home, prescribed some medication by the triage doctor and send home, or is staying in ER – in which case the patient is assigned a bed and case doctors (one of the doctors on each shift best qualified for the particular problem of the patient). Each bed is supervised by a single nurse during a shift, but a nurse may supervise many beds, or none at all. The case doctor(s) may prescribe a medication that is administered to the patient by a single nurse in each shift for the duration of the patient taking the medicine. Each medication has a name, and for each patient there may be a different dosage and different number of times a day to take it.

8.2 Emergency Room Logical Model

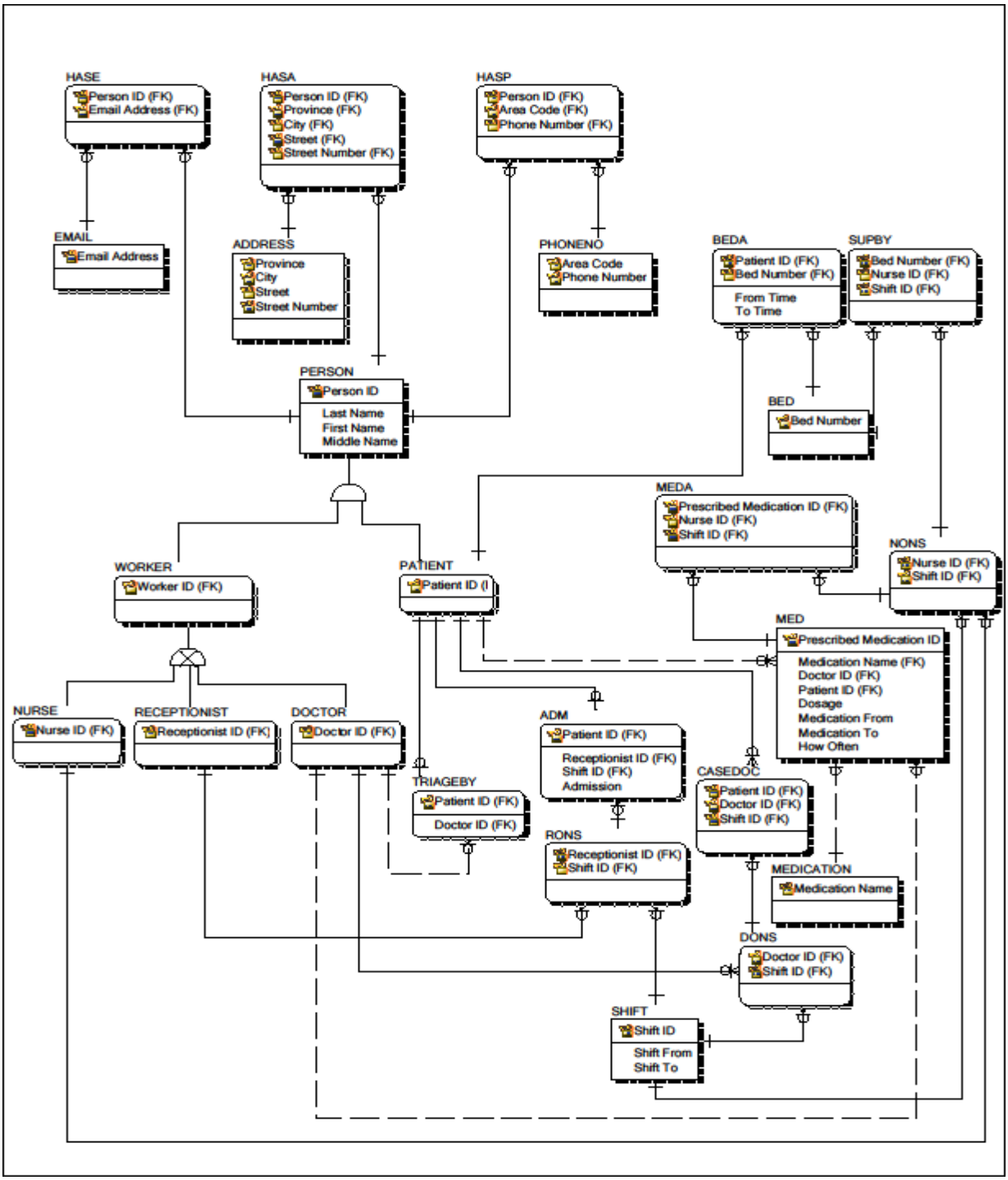


Figure 12: The Emergency Room Logical Model

8.3 Emergency Room Physical DB2 Model

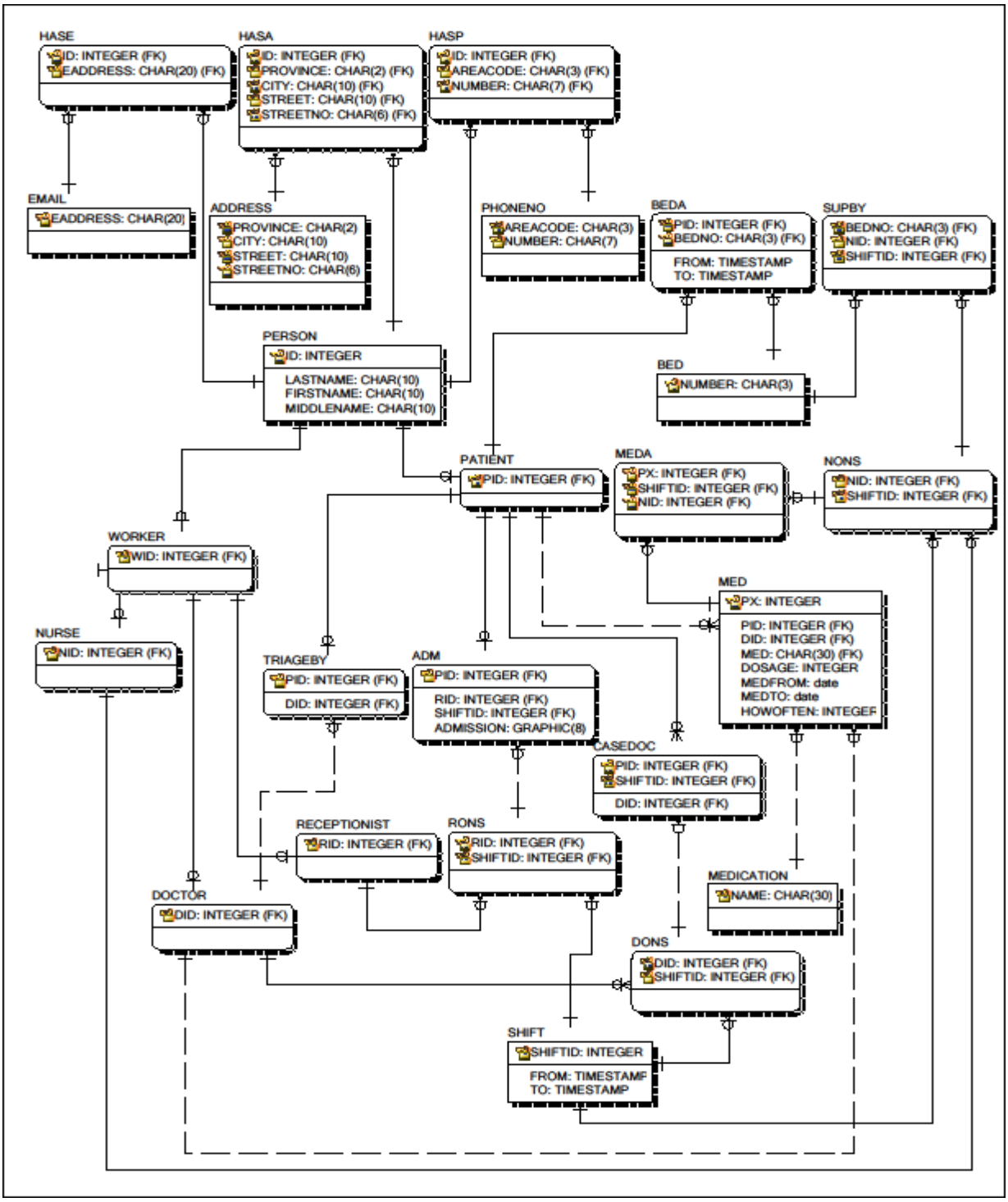


Figure 13: The Emergency Room Physical Model

8.4 Emergency Room DB2 Schema

```
CREATE TABLE PERSON(
       ID INTEGER NOT NULL,
       LASTNAME CHAR(10) NOT NULL,
       FIRSTNAME CHAR(10) NOT NULL,
       MIDDLENAME CHAR(10),
       PRIMARY KEY (ID)
)
CREATE TABLE PATIENT(
       PID INTEGER NOT NULL,
       PRIMARY KEY (PID),
       FOREIGN KEY (PID) REFERENCES PERSON (ID)
)
CREATE TABLE WORKER(
       WID INTEGER NOT NULL,
       PRIMARY KEY (WID),
       FOREIGN KEY (WID) REFERENCES PERSON (ID)
)
CREATE TABLE RECEPTIONIST(
       RID INTEGER NOT NULL,
       PRIMARY KEY (RID),
       FOREIGN KEY (RID) REFERENCES WORKER (WID)
)
CREATE TABLE NURSE(
       NID INTEGER NOT NULL,
       PRIMARY KEY (NID),
       FOREIGN KEY (NID) REFERENCES WORKER (WID)
)
CREATE TABLE DOCTOR(
       DID INTEGER NOT NULL,
       PRIMARY KEY (DID),
       FOREIGN KEY (DID) REFERENCES WORKER (WID)
)
CREATE TABLE EMAIL(
       EADDRESS CHAR(20) NOT NULL,
       PRIMARY KEY (EADDRESS)
)
CREATE TABLE PHONENO(
       AREACODE CHAR(3) NOT NULL,
       NUMBER CHAR(7) NOT NULL,
       PRIMARY KEY (AREACODE,NUMBER)
)
CREATE TABLE ADDRESS(
       PROVINCE CHAR(2) NOT NULL,
       CITY CHAR(10) NOT NULL,
       STREET CHAR(10) NOT NULL,
       STREETNO CHAR(6) NOT NULL,
       PRIMARY KEY (PROVINCE,CITY,STREET,STREETNO)
)
CREATE TABLE MEDICATION(
       NAME CHAR(30) NOT NULL,
       PRIMARY KEY (NAME)
```

```
)
CREATE TABLE BED(
       NUMBER CHAR(3) NOT NULL,
       PRIMARY KEY (NUMBER)
)
CREATE TABLE SHIFT(
       SHIFTID INTEGER NOT NULL,
       FROM TIMESTAMP NOT NULL,
       TO TIMESTAMP NOT NULL,
       PRIMARY KEY (SHIFTID),
       UNIQUE (FROM,TO),
       CHECK (FROM < TO)
)
CREATE TABLE HASE(
       ID INTEGER NOT NULL,
       EADDRESS CHAR(20) NOT NULL,
       PRIMARY KEY (ID,EADDRESS),
       FOREIGN KEY (ID) REFERENCES PERSON (ID),
       FOREIGN KEY (EADDRESS) REFERENCES EMAIL (EADDRESS)
)
CREATE TABLE HASP(
       ID INTEGER NOT NULL,
       AREACODE CHAR(3) NOT NULL,
       NUMBER CHAR(7) NOT NULL,
       PRIMARY KEY (ID,AREACODE,NUMBER),
       FOREIGN KEY (ID) REFERENCES PERSON (ID),
       FOREIGN KEY (AREACODE,NUMBER) REFERENCES PHONENO(AREACODE,NUMBER)
)
CREATE TABLE HASA(
       ID INTEGER NOT NULL,
       PROVINCE CHAR(2) NOT NULL,
       CITY CHAR(10) NOT NULL,
       STREET CHAR(10) NOT NULL,
       STREETNO CHAR(6) NOT NULL,
       PRIMARY KEY (ID,PROVINCE,CITY,STREET,STREETNO),
       FOREIGN KEY (ID) REFERENCES PERSON (ID),
       FOREIGN KEY (PROVINCE,CITY,STREET,STREETNO) REFERENCES ADDRESS
                    (PROVINCE,CITY,STREET,STREETNO)
)
CREATE TABLE RONS(
       RID INTEGER NOT NULL,
       SHIFTID INTEGER NOT NULL,
       PRIMARY KEY (RID,SHIFTID),
       FOREIGN KEY (RID) REFERENCES RECEPTIONIST (RID),
       FOREIGN KEY (SHIFTID) REFERENCES SHIFT (SHIFTID)
)
CREATE TABLE NONS(
       NID INTEGER NOT NULL,
       SHIFTID INTEGER NOT NULL,
       PRIMARY KEY (NID,SHIFTID),
       FOREIGN KEY (NID) REFERENCES NURSE (NID),
       FOREIGN KEY (SHIFTID) REFERENCES SHIFT (SHIFTID)
)
```

```
CREATE TABLE DONS(
       DID INTEGER NOT NULL,
       SHIFTID INTEGER NOT NULL,
       PRIMARY KEY (DID,SHIFTID),
       FOREIGN KEY (DID) REFERENCES DOCTOR (DID),
       FOREIGN KEY (SHIFTID) REFERENCES SHIFT (SHIFTID)
)
CREATE TABLE MED(
       PX INTEGER NOT NULL,
       PID INTEGER NOT NULL,
       DID INTEGER NOT NULL,
       MED CHAR(30) NOT NULL,
       DOSAGE INTEGER NOT NULL,
       MEDFROM DATE NOT NULL,
       MEDTO DATE NOT NULL,
       HOWOFTEN INTEGER NOT NULL,
       PRIMARY KEY (PX),
       UNIQUE (PID,MED),
       FOREIGN KEY (PID) REFERENCES PATIENT (PID),
       FOREIGN KEY (DID) REFERENCES DOCTOR (DID),
       FOREIGN KEY (MED) REFERENCES MEDICATION (NAME)
)
CREATE TABLE MEDA(
       PX INTEGER NOT NULL,
       NID INTEGER NOT NULL,
       SHIFTID INTEGER NOT NULL,
       PRIMARY KEY (PX,SHIFTID,NID),
       FOREIGN KEY (PX) REFERENCES MED (PX),
       FOREIGN KEY (NID,SHIFTID) REFERENCES NONS (NID,SHIFTID)
)
CREATE TABLE BEDA(
       PID INTEGER NOT NULL,
       BEDNO CHAR(3) NOT NULL,
       FROM TIMESTAMP NOT NULL,
       TO TIMESTAMP NOT NULL,
       PRIMARY KEY (PID, BEDNO),
       FOREIGN KEY (PID) REFERENCES PATIENT (PID),
       FOREIGN KEY (BEDNO) REFERENCES BED (NUMBER)
)
CREATE TABLE CASEDOC(
       PID INTEGER NOT NULL,
       DID INTEGER NOT NULL,
       SHIFTID INTEGER NOT NULL,
       PRIMARY KEY (PID,SHIFTID,DID),
       FOREIGN KEY (PID) REFERENCES PATIENT (PID),
       FOREIGN KEY (DID,SHIFTID) REFERENCES DONS (DID,SHIFTID)
)
CREATE TABLE SUPBY(
       BEDNO CHAR(3) NOT NULL,
       NID INTEGER NOT NULL,
       SHIFTID INTEGER NOT NULL,
       PRIMARY KEY (BEDNO,SHIFTID,NID),
       FOREIGN KEY (BEDNO) REFERENCES BED (NUMBER),
       FOREIGN KEY (NID,SHIFTID) REFERENCES NONS (NID,SHIFTID)
```

```
)
CREATE TABLE ADM(
       PID INTEGER NOT NULL,
       RID INTEGER NOT NULL,
       SHIFTID INTEGER NOT NULL,
       ADMISSION TIMESTAMP,
       PRIMARY KEY (PID),
       FOREIGN KEY (PID) REFERENCES PATIENT (PID),
       FOREIGN KEY (RID,SHIFTID) REFERENCES RONS (RID,SHIFTID)
)
CREATE TABLE TRIAGEBY(
       PID INTEGER NOT NULL,
       DID INTEGER NOT NULL,
       PRIMARY KEY (PID),
       FOREIGN KEY (PID) REFERENCES PATIENT (PID),
       FOREIGN KEY (DID) REFERENCES DOCTOR (DID)
)
```

8.5 Emergency Room Interactive Queries

Query 1: The query returns an empty set if con1 is satisfied.

```
((SELECT RID FROM RECEPTIONIST) INTERSECT (SELECT NID FROM NURSE))
UNION
((SELECT RID FROM RECEPTIONIST) INTERSECT (SELECT DID FROM DOCTOR))
UNION
((SELECT NID FROM NURSE) INTERSECT (SELECT DID FROM DOCTOR)
```

Query 2: The query returns an empty set if con2 is satisfied.

```
SELECT EADDRESS FROM EMAIL
EXCEPT
SELECT EADDRESS FROM HASE
```

Query 3: The query returns an empty set if con3 is satisfied.

```
SELECT AREACODE,NUMBER FROM PHONENO
EXCEPT
SELECT AREACODE,NUMBER FROM HASP
```

Query 4: The query returns an empty set if con4 is satisfied.

```
SELECT PROVINCE,CITY,STREET,STRETNO  FROM ADDRESS
EXCEPT
SELECT PROVINCE,CITY,STREET,STREETNO FROM HASA
```

Query 5: The query returns an empty set if con5 is satisfied.

```
SELECT ID FROM PERSON
EXCEPT
```

```
SELECT ID FROM HASA
```

Query 6: The query returns an empty set if con6 is satisfied.

```
((SELECT SHIFTID FROM RONS)
 EXCEPT
((SELECT T.SHIFTID FROM RONS AS T,RONS AS R
  WHERE T.RID <> R.RID AND T.SHIFTID = R.SHIFTID)
  EXCEPT
  (SELECT T.SHIFTID FROM RONS AS T,RONS AS R,RONS AS Q
   WHERE T.SHIFTID = R.SHIFTID AND T.SHIFTID = Q.SHIFTID
   AND T.RID <> R.RID AND T.RID<>Q.RID AND R.RID<>Q.RID))
)
```

Query 7: The query returns an empty set if con7 is satisfied.

```
(SELECT SHIFTID FROM SHIFT) EXCEPT (SELECT SHIFTID FROM RONS)
UNION
(SELECT RID FROM RECEPTIONIST) EXCEPT (SELECT RID FROM RONS)
```

Query 8: The query returns an empty set if con8 is satisfied.

```
SELECT SHIFTID FROM SHIFT
EXCEPT
SELECT T.SHIFTID FROM NONS AS T,NONS AS R
WHERE T.NID <> R.NID AND T.SHIFTID = R.SHIFTID
```

Query 9: The query returns an empty set if con9 is satisfied.

```
(SELECT SHIFTID FROM SHIFT) EXCEPT (SELECT SHIFTID FROM NONS)
UNION
(SELECT NID FROM NURSE) EXCEPT (SELECT NID FROM NONS)
```

Query 10: The query returns an empty set if con10 is satisfied.

```
((SELECT SHIFTID FROM SHIFT)
EXCEPT
(SELECT T.SHIFTID
 FROM DONS AS T,DONS AS R
 WHERE T.DID <> R.DID AND T.SHIFTID = R.SHIFTID))
```

Query 11: The query returns an empty set if con11 is satisfied.

```
((SELECT SHIFTID FROM SHIFT) EXCEPT (SELECT SHIFTID FROM DONS))
UNION
((SELECT DID FROM DOCTOR) EXCEPT (SELECT DID FROM DONS))
```

Query 12: The query returns an empty set if con12 is satisfied.

```
((SELECT PID FROM CASEDOC) EXCEPT (SELECT PID FROM BEDA))
```

Query 13: The query returns an empty set if con13 is satisfied.

```
((SELECT PID FROM BEDA) EXCEPT (SELECT PID FROM CASEDOC))
```

Query 14: The query returns an empty set if con14 is satisfied.

```
((SELECT BED.NUMBER,SHIFTID FROM BED,SHIFT)
 EXCEPT
 (SELECT BED.NUMBER,SHIFTID FROM BED,SUPBY
  WHERE BED.NUMBER=SUPBY.BEDNO))
```

Query 15: The query returns an empty set if con15 is satisfied.

```
(SELECT FROM,TO,ADMISSION FROM SHIFT,ADM
 WHERE SHIFT.SHIFTID=ADM.SHIFTID
 AND (ADMISSION < FROM OR TO < ADMISSION))
```

Query 16: The query returns an empty set if con16 is satisfied.

```
((SELECT PID FROM TRIAGEBY)
  EXCEPT
 (SELECT TRIAGEBY.PID FROM DONS,ADM,TRIAGEBY
  WHERE TRIAGEBY.PID=ADM.PID AND ADM.SHIFTID=DONS.SHIFTID
  AND DONS.DID=TRIAGEBY.DID))
```

Chapter 9: Property Rental

9.1 Property Rental Informal Description

Our company arranges rentals of properties owned by both private and business owners. We assign every property owner a unique owner number for identification, we record its address (consisting of a street, street number, town or city, and province), the owner's name (consisting of first, middle, and last name for a person or name of a business), and the owners email addresses and the owner phone numbers. For a business owner, we record the type (description) of its business. Each property is identified by a unique property number, we record its address and its type. Each property may be placed in several advertisements. Each such advertisement may be displayed in many newspapers on several dates. The newspapers are identified by unique names.

The term renter refers to a private person or a business who signed a rental agreement for a property. Each such rental agreement is identified in our database by a unique rental number. We record the date of the singing of the rental agreement, the starting and ending date of the rental agreement. A renter can rent many properties. A renter, prior to accepting the rental agreement may view the property repeatedly and we record the date of viewing. For each renter, we record its address, its name, its email address and phone numbers. Each renter has a unique renter number in our database.

Our agency is organized into branches and every staff member is allocated to exactly one branch. Each branch has one manager who is a member of the staff. In our database, we identify the staff by a unique staff number. For each staff member we record address, name, email address, phone numbers, sex, position, and salary. Each property is in care of one of our branches. Each renter refers to the branch that is in care of the property it rents. Each property is overseen by a unique staff member. Each branch has an address, phone number, and a unique branch number.

9.2 Property Rental Logical Model

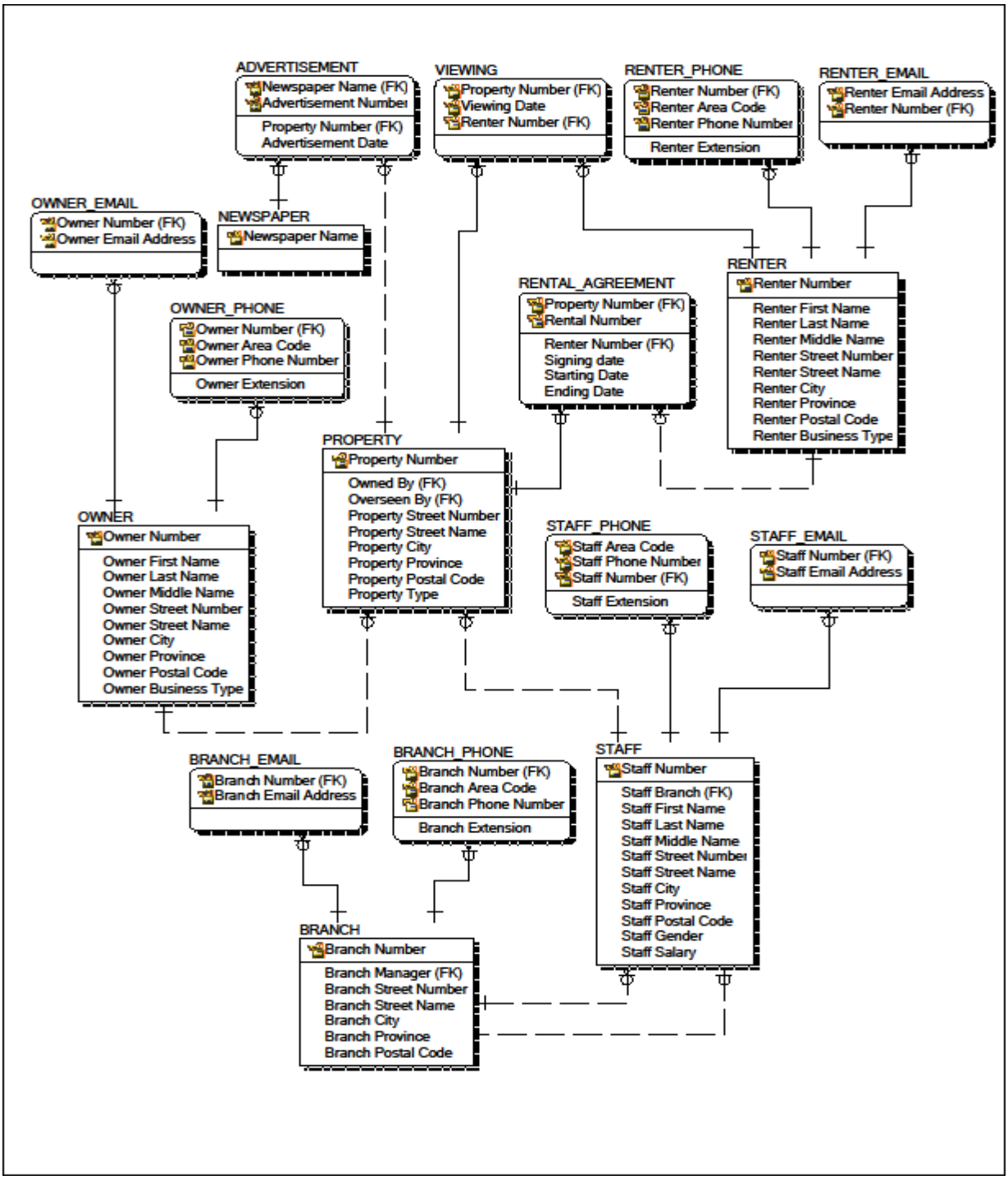


Figure 14: The Property Rental Logical Model

9.3 Property Rental Physical DB2 Model

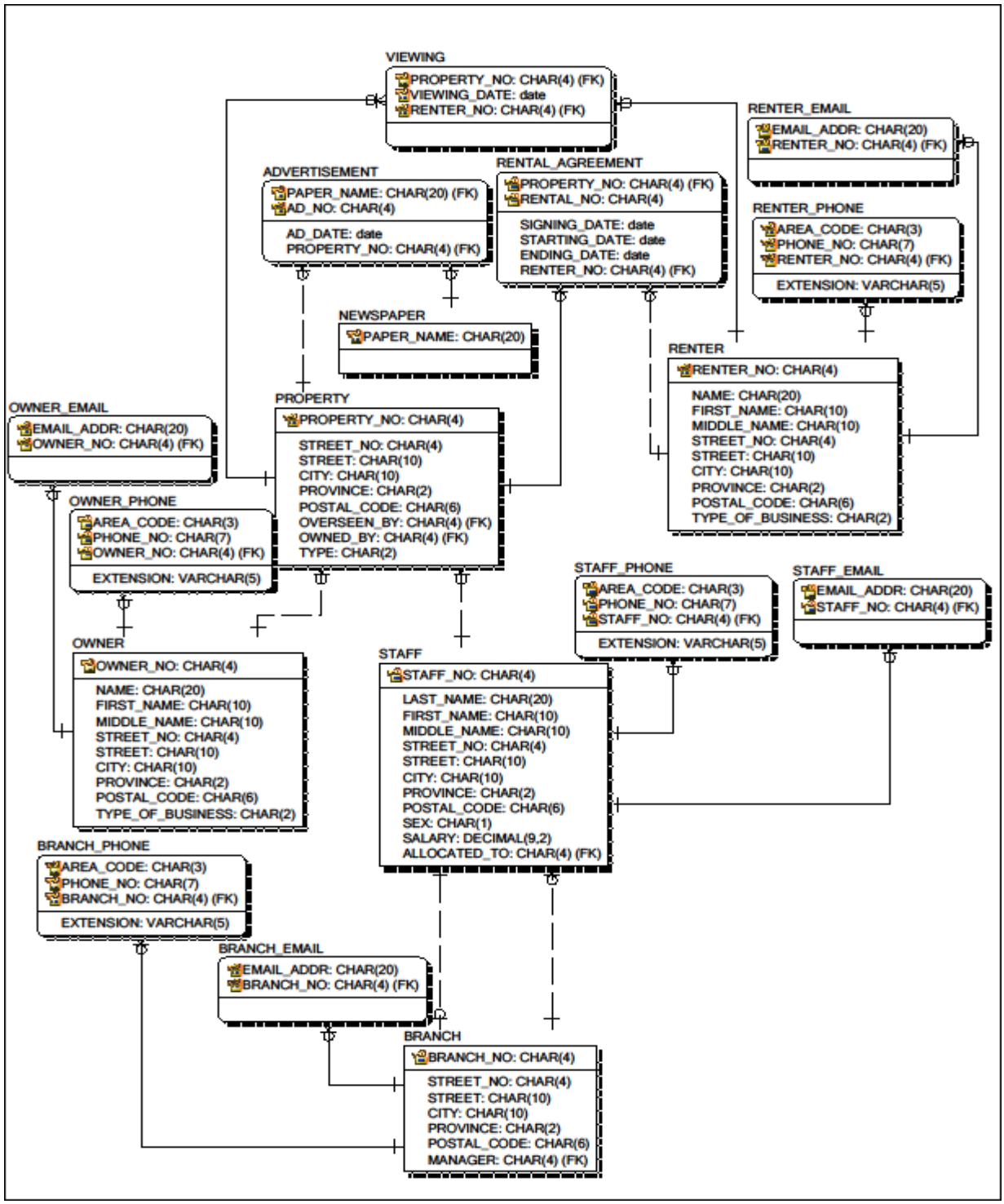


Figure 15: The Property Rental Physical Model

9.4 Property Rental DB2 Schema

```
CREATE TABLE BRANCH (
       BRANCH_NO CHAR(4) NOT NULL,
       STREET_NO CHAR(4) NOT NULL,
       STREET CHAR(10) NOT NULL,
       CITY CHAR(10) NOT NULL,
       PROVINCE CHAR(2) NOT NULL,
       POSTAL_CODE CHAR(6) NOT NULL,
       MANAGER CHAR(4) NOT NULL,
       PRIMARY KEY (BRANCH_NO),
       CHECK (PROVINCE IN ('AL','BC','MA','NB','NF','NT','NS', 'NU',
                           'ON','PE','QB','SA','YU')),
       CHECK (('A'<=SUBSTR(POSTAL_CODE,1,1) AND
             SUBSTR(POSTAL_CODE,1,1)<='Z') AND
         ('0'<=SUBSTR(POSTAL_CODE,2,1) AND SUBSTR(POSTAL_CODE,2,1)<='9')
         AND
         ('A'<=SUBSTR(POSTAL_CODE,3,1) AND SUBSTR(POSTAL_CODE,3,1)<='Z')
         AND
         ('0'<=SUBSTR(POSTAL_CODE,4,1) AND SUBSTR(POSTAL_CODE,4,1)<='9')
         AND
         ('A'<=SUBSTR(POSTAL_CODE,5,1) AND SUBSTR(POSTAL_CODE,5,1)<='Z')
         AND
         ('0'<=SUBSTR(POSTAL_CODE,6,1) AND
          SUBSTR(POSTAL_CODE,6,1)<='9')),
       UNIQUE(MANAGER)
)
CREATE TABLE STAFF (
       STAFF_NO CHAR(4) NOT NULL,
       LAST_NAME CHAR(20) NOT NULL,
       FIRST_NAME CHAR(10) NOT NULL,
       MIDDLE_NAME CHAR(10),
       STREET_NO CHAR(4) NOT NULL,
       STREET CHAR(10) NOT NULL,
       CITY CHAR(10) NOT NULL,
       PROVINCE CHAR(2) NOT NULL,
       POSTAL_CODE CHAR(6) NOT NULL,
       SEX CHAR(1) NOT NULL,
       SALARY DECIMAL(9,2) NOT NULL,
       ALLOCATED_TO CHAR(4) NOT NULL,
       PRIMARY KEY (STAFF_NO),
       FOREIGN KEY (ALLOCATED_TO) REFERENCES BRANCH,
       CHECK (PROVINCE IN ('AL','BC','MA','NB','NF','NT','NS',
                           'NU','ON','PE','QB','SA','YU')),
       CHECK (SEX IN ('F','M','N')),
       CHECK (SALARY > 0),
       CHECK (('A'<=SUBSTR(POSTAL_CODE,1,1)
       AND SUBSTR(POSTAL_CODE,1,1)<='Z')
       AND ('0'<=SUBSTR(POSTAL_CODE,2,1)
       AND SUBSTR(POSTAL_CODE,2,1)<='9')
       AND ('A'<=SUBSTR(POSTAL_CODE,3,1)
       AND SUBSTR(POSTAL_CODE,3,1)<='Z')
```

```
        AND ('0'<=SUBSTR(POSTAL_CODE,4,1)
        AND SUBSTR(POSTAL_CODE,4,1)<='9')
        AND  ('A'<=SUBSTR(POSTAL_CODE,5,1)
        AND SUBSTR(POSTAL_CODE,5,1)<='Z')
        AND ('0'<=SUBSTR(POSTAL_CODE,6,1)
        AND SUBSTR(POSTAL_CODE,6,1)<='9'))
)
CREATE TABLE OWNER (
        OWNER_NO CHAR(4) NOT NULL,
        NAME CHAR(20) NOT NULL,
        FIRST_NAME CHAR(10),
        MIDDLE_NAME CHAR(10),
        STREET_NO CHAR(4) NOT NULL,
        STREET CHAR(10) NOT NULL,
        CITY CHAR(10) NOT NULL,
        PROVINCE CHAR(2) NOT NULL,
        POSTAL_CODE CHAR(6) NOT NULL,
        TYPE_OF_BUSINESS CHAR(2),
        PRIMARY KEY (OWNER_NO),
        CHECK (PROVINCE IN ('AL','BC','MA','NB','NF','NT','NS','NU',
                            'ON','PE','QB','SA','YU')),
  CHECK (('A'<=SUBSTR(POSTAL_CODE,1,1) AND SUBSTR(POSTAL_CODE,1,1)<='Z')
  AND ('0'<=SUBSTR(POSTAL_CODE,2,1) AND SUBSTR(POSTAL_CODE,2,1)<='9')
  AND ('A'<=SUBSTR(POSTAL_CODE,3,1) AND SUBSTR(POSTAL_CODE,3,1)<='Z')
  AND ('0'<=SUBSTR(POSTAL_CODE,4,1) AND SUBSTR(POSTAL_CODE,4,1)<='9')
  AND ('A'<=SUBSTR(POSTAL_CODE,5,1) AND SUBSTR(POSTAL_CODE,5,1)<='Z')
  AND ('0'<=SUBSTR(POSTAL_CODE,6,1) AND SUBSTR(POSTAL_CODE,6,1)<='9')),
  CHECK(TYPE_OF_BUSINESS IS NULL OR (FIRST_NAME IS NULL AND MIDDLE_NAME
        IS NULL))
)
CREATE TABLE RENTER (
        RENTER_NO CHAR(4) NOT NULL,
        NAME CHAR(20) NOT NULL,
        FIRST_NAME CHAR(10),
        MIDDLE_NAME CHAR(10),
        STREET_NO CHAR(4) NOT NULL,
        STREET CHAR(10) NOT NULL,
        CITY CHAR(10) NOT NULL,
        PROVINCE CHAR(2) NOT NULL,
        POSTAL_CODE CHAR(6) NOT NULL,
        TYPE_OF_BUSINESS CHAR(2),
        PRIMARY KEY (RENTER_NO),
  CHECK (PROVINCE IN ('AL','BC','MA','NB','NF','NT','NS','NU','ON',
                      'PE','QB','SA','YU')),
  CHECK (('A'<=SUBSTR(POSTAL_CODE,1,1) AND SUBSTR(POSTAL_CODE,1,1)<='Z')
  AND ('0'<=SUBSTR(POSTAL_CODE,2,1) AND SUBSTR(POSTAL_CODE,2,1)<='9')
  AND ('A'<=SUBSTR(POSTAL_CODE,3,1) AND SUBSTR(POSTAL_CODE,3,1)<='Z')
  AND ('0'<=SUBSTR(POSTAL_CODE,4,1) AND SUBSTR(POSTAL_CODE,4,1)<='9')
  AND ('A'<=SUBSTR(POSTAL_CODE,5,1) AND SUBSTR(POSTAL_CODE,5,1)<='Z')
  AND ('0'<=SUBSTR(POSTAL_CODE,6,1) AND SUBSTR(POSTAL_CODE,6,1)<='9')),
  CHECK(TYPE_OF_BUSINESS IS NULL OR (FIRST_NAME IS NULL AND MIDDLE_NAME
  IS NULL))
```

```
)
CREATE TABLE PROPERTY (
       PROPERTY_NO CHAR(4) NOT NULL,
       STREET_NO CHAR(4) NOT NULL,
       STREET CHAR(10) NOT NULL,
       CITY CHAR(10) NOT NULL,
       PROVINCE CHAR(2) NOT NULL,
       POSTAL_CODE CHAR(6) NOT NULL,
       OVERSEEN_BY CHAR(4) NOT NULL,
       OWNED_BY CHAR(4) NOT NULL,
       TYPE CHAR(2) NOT NULL,
       PRIMARY KEY (PROPERTY_NO),
       FOREIGN KEY (OVERSEEN_BY) REFERENCES STAFF,
       FOREIGN KEY (OWNED_BY) REFERENCES OWNER,
  CHECK (PROVINCE IN ('AL','BC','MA','NB','NF','NT','NS','NU','ON',
                      'PE','QB','SA','YU')),
  CHECK (('A'<=SUBSTR(POSTAL_CODE,1,1) AND SUBSTR(POSTAL_CODE,1,1)<='Z')
  AND ('0'<=SUBSTR(POSTAL_CODE,2,1) AND SUBSTR(POSTAL_CODE,2,1)<='9')
  AND ('A'<=SUBSTR(POSTAL_CODE,3,1) AND SUBSTR(POSTAL_CODE,3,1)<='Z')
  AND ('0'<=SUBSTR(POSTAL_CODE,4,1) AND SUBSTR(POSTAL_CODE,4,1)<='9')
  AND ('A'<=SUBSTR(POSTAL_CODE,5,1) AND SUBSTR(POSTAL_CODE,5,1)<='Z')
  AND ('0'<=SUBSTR(POSTAL_CODE,6,1) AND SUBSTR(POSTAL_CODE,6,1)<='9'))
)
CREATE TABLE RENTAL_AGREEMENT (
       PROPERTY_NO CHAR(4) NOT NULL,
       RENTAL_NO CHAR(4) NOT NULL,
       SIGNING_DATE DATE NOT NULL,
       STARTING_DATE DATE NOT NULL,
       ENDING_DATE DATE NOT NULL,
       RENTER_NO CHAR(4) NOT NULL,
       PRIMARY KEY (PROPERTY_NO,RENTAL_NO),
       FOREIGN KEY (PROPERTY_NO) REFERENCES PROPERTY,
       FOREIGN KEY (RENTER_NO) REFERENCES RENTER,
       CHECK (SIGNING_DATE <= STARTING_DATE),
       CHECK (STARTING_DATE <= ENDING_DATE)
)
CREATE TABLE RENTER_EMAIL (
       EMAIL_ADDR CHAR(20) NOT NULL,
       RENTER_NO CHAR(4) NOT NULL,
       PRIMARY KEY (EMAIL_ADDR,RENTER_NO),
       FOREIGN KEY (RENTER_NO) REFERENCES RENTER
)
CREATE TABLE STAFF_EMAIL (
       EMAIL_ADDR CHAR(20) NOT NULL,
       STAFF_NO CHAR(4) NOT NULL,
       PRIMARY KEY (EMAIL_ADDR,STAFF_NO),
       FOREIGN KEY (STAFF_NO) REFERENCES STAFF
)
CREATE TABLE OWNER_EMAIL (
       EMAIL_ADDR CHAR(20) NOT NULL,
       OWNER_NO CHAR(4) NOT NULL,
       PRIMARY KEY (EMAIL_ADDR,OWNER_NO),
       FOREIGN KEY (OWNER_NO) REFERENCES OWNER
)
```

```
CREATE TABLE BRANCH_EMAIL (
       EMAIL_ADDR CHAR(20) NOT NULL,
       BRANCH_NO CHAR(4) NOT NULL,
       PRIMARY KEY (EMAIL_ADDR,BRANCH_NO),
       FOREIGN KEY (BRANCH_NO) REFERENCES BRANCH
)
CREATE TABLE RENTER_PHONE (
       AREA_CODE CHAR(3) NOT NULL,
       PHONE_NO CHAR(7) NOT NULL,
       EXTENSION VARCHAR(5),
       RENTER_NO CHAR(4) NOT NULL,
       PRIMARY KEY (AREA_CODE,PHONE_NO,RENTER_NO),
       FOREIGN KEY (RENTER_NO) REFERENCES RENTER,
  CHECK(('0'<=SUBSTR(AREA_CODE,1,1) AND SUBSTR(AREA_CODE,1,1)<='9')
  AND ('0'<=SUBSTR(AREA_CODE,2,1) AND SUBSTR(AREA_CODE,2,1)<='9')
  AND ('0'<=SUBSTR(AREA_CODE,3,1) AND SUBSTR(AREA_CODE,3,1)<='9')),
  CHECK(('0'<=SUBSTR(PHONE_NO,1,1) AND SUBSTR(PHONE_NO,1,1)<='9')
  AND ('0'<=SUBSTR(PHONE_NO,2,1) AND SUBSTR(PHONE_NO,2,1)<='9')
  AND ('0'<=SUBSTR(PHONE_NO,3,1) AND SUBSTR(PHONE_NO,3,1)<='9')
  AND ('0'<=SUBSTR(PHONE_NO,4,1) AND SUBSTR(PHONE_NO,4,1)<='9')
  AND ('0'<=SUBSTR(PHONE_NO,5,1) AND SUBSTR(PHONE_NO,5,1)<='9')
  AND ('0'<=SUBSTR(PHONE_NO,6,1) AND SUBSTR(PHONE_NO,6,1)<='9')
  AND ('0'<=SUBSTR(PHONE_NO,7,1) AND SUBSTR(PHONE_NO,7,1)<='9'))
)
CREATE TABLE STAFF_PHONE (
       AREA_CODE CHAR(3) NOT NULL,
       PHONE_NO CHAR(7) NOT NULL,
       EXTENSION VARCHAR(5),
       STAFF_NO CHAR(4) NOT NULL,
       PRIMARY KEY (AREA_CODE,PHONE_NO,STAFF_NO),
       FOREIGN KEY (STAFF_NO) REFERENCES STAFF,
  CHECK(('0'<=SUBSTR(AREA_CODE,1,1) AND SUBSTR(AREA_CODE,1,1)<='9')
  AND ('0'<=SUBSTR(AREA_CODE,2,1) AND SUBSTR(AREA_CODE,2,1)<='9')
  AND ('0'<=SUBSTR(AREA_CODE,3,1) AND SUBSTR(AREA_CODE,3,1)<='9')),
  CHECK(('0'<=SUBSTR(PHONE_NO,1,1) AND SUBSTR(PHONE_NO,1,1)<='9')
  AND ('0'<=SUBSTR(PHONE_NO,2,1) AND SUBSTR(PHONE_NO,2,1)<='9')
  AND ('0'<=SUBSTR(PHONE_NO,3,1) AND SUBSTR(PHONE_NO,3,1)<='9')
  AND ('0'<=SUBSTR(PHONE_NO,4,1) AND SUBSTR(PHONE_NO,4,1)<='9')
  AND ('0'<=SUBSTR(PHONE_NO,5,1) AND SUBSTR(PHONE_NO,5,1)<='9')
  AND ('0'<=SUBSTR(PHONE_NO,6,1) AND SUBSTR(PHONE_NO,6,1)<='9')
  AND ('0'<=SUBSTR(PHONE_NO,7,1) AND SUBSTR(PHONE_NO,7,1)<='9'))
)
CREATE TABLE OWNER_PHONE (
       AREA_CODE CHAR(3) NOT NULL,
       PHONE_NO CHAR(7) NOT NULL,
       EXTENSION VARCHAR(5),
       OWNER_NO CHAR(4) NOT NULL,
       PRIMARY KEY (AREA_CODE,PHONE_NO,OWNER_NO),
       FOREIGN KEY (OWNER_NO) REFERENCES OWNER,
  CHECK(('0'<=SUBSTR(AREA_CODE,1,1) AND SUBSTR(AREA_CODE,1,1)<='9')
  AND ('0'<=SUBSTR(AREA_CODE,2,1) AND SUBSTR(AREA_CODE,2,1)<='9')
  AND ('0'<=SUBSTR(AREA_CODE,3,1) AND SUBSTR(AREA_CODE,3,1)<='9')),
  CHECK(('0'<=SUBSTR(PHONE_NO,1,1) AND SUBSTR(PHONE_NO,1,1)<='9')
  AND ('0'<=SUBSTR(PHONE_NO,2,1) AND SUBSTR(PHONE_NO,2,1)<='9')
```

```
  AND ('0'<=SUBSTR(PHONE_NO,3,1) AND SUBSTR(PHONE_NO,3,1)<='9')
  AND ('0'<=SUBSTR(PHONE_NO,4,1) AND SUBSTR(PHONE_NO,4,1)<='9')
  AND ('0'<=SUBSTR(PHONE_NO,5,1) AND SUBSTR(PHONE_NO,5,1)<='9')
  AND ('0'<=SUBSTR(PHONE_NO,6,1) AND SUBSTR(PHONE_NO,6,1)<='9')
  AND ('0'<=SUBSTR(PHONE_NO,7,1) AND SUBSTR(PHONE_NO,7,1)<='9'))
)
CREATE TABLE BRANCH_PHONE (
       AREA_CODE CHAR(3) NOT NULL,
       PHONE_NO CHAR(7) NOT NULL,
       EXTENSION VARCHAR(5),
       BRANCH_NO CHAR(4) NOT NULL,
       PRIMARY KEY (AREA_CODE,PHONE_NO,BRANCH_NO),
       FOREIGN KEY (BRANCH_NO) REFERENCES BRANCH,
  CHECK(('0'<=SUBSTR(AREA_CODE,1,1) AND SUBSTR(AREA_CODE,1,1)<='9')
  AND ('0'<=SUBSTR(AREA_CODE,2,1) AND SUBSTR(AREA_CODE,2,1)<='9')
  AND ('0'<=SUBSTR(AREA_CODE,3,1) AND SUBSTR(AREA_CODE,3,1)<='9')),
  CHECK(('0'<=SUBSTR(PHONE_NO,1,1) AND SUBSTR(PHONE_NO,1,1)<='9')
  AND ('0'<=SUBSTR(PHONE_NO,2,1) AND SUBSTR(PHONE_NO,2,1)<='9')
  AND ('0'<=SUBSTR(PHONE_NO,3,1) AND SUBSTR(PHONE_NO,3,1)<='9')
  AND ('0'<=SUBSTR(PHONE_NO,4,1) AND SUBSTR(PHONE_NO,4,1)<='9')
  AND ('0'<=SUBSTR(PHONE_NO,5,1) AND SUBSTR(PHONE_NO,5,1)<='9')
  AND ('0'<=SUBSTR(PHONE_NO,6,1) AND SUBSTR(PHONE_NO,6,1)<='9')
  AND ('0'<=SUBSTR(PHONE_NO,7,1) AND SUBSTR(PHONE_NO,7,1)<='9'))
)
CREATE TABLE VIEWING (
       PROPERTY_NO CHAR(4) NOT NULL,
       RENTER_NO CHAR(4) NOT NULL,
       VIEWING_DATE DATE NOT NULL,
       PRIMARY KEY (PROPERTY_NO,VIEWING_DATE,RENTER_NO),
       FOREIGN KEY (PROPERTY_NO) REFERENCES PROPERTY,
       FOREIGN KEY (RENTER_NO) REFERENCES RENTER
)
CREATE TABLE NEWSPAPER (
       PAPER_NAME CHAR(20) NOT NULL,
       PRIMARY KEY (PAPER_NAME)
)
CREATE TABLE ADVERTISEMENT (
       PAPER_NAME CHAR(20) NOT NULL,
       AD_NO CHAR(4) NOT NULL,
       AD_DATE DATE NOT NULL,
       PROPERTY_NO CHAR(4) NOT NULL,
      PRIMARY KEY (PAPER_NAME,AD_NO),
      FOREIGN KEY (PAPER_NAME) REFERENCES NEWSPAPER,
      FOREIGN KEY (PROPERTY_NO) REFERENCES PROPERTY
)
ALTER TABLE BRANCH
ADD CONSTRAINT MANAGER_CNST FOREIGN KEY (MANAGER) REFERENCES
STAFF(STAFF_NO)
```

9.5 Property Rental Interactive Queries

Query 1: Give the staff number of each of the staff members whose salary is greater than 5000. Please, sort them by the staff number.

```
SELECT STAFF_NO FROM STAFF
WHERE SALARY > 5000
ORDER BY STAFF_NO
```

Query 2: Give the renter number of each of the renters who has a viewing record. Please avoid duplications.

```
SELECT DISTINCT RENTER_NO FROM VIEWING
```

Query 3: Give the dates of all the advertisements posted in THE GLOBE AND MAIL in 2005. Please, avoid duplications.

```
SELECT DISTINCT AD_DATE FROM ADVERTISEMENT
WHERE PAPER_NAME = 'THE GLOBE AND MAIL' AND AD_DATE>='2005-01-01'
AND AD_DATE<='2005-12-31'
ORDER BY AD_DATE
```

Query 4: Give the email addresses and the renter number for all the private renters. Please, sort them by the renter number.

```
SELECT EMAIL_ADDR, RENTER.RENTER_NO
FROM RENTER_EMAIL, RENTER
WHERE RENTER_EMAIL.RENTER_NO = RENTER.RENTER_NO
AND TYPE_OF_BUSINESS IS NULL
ORDER BY RENTER.RENTER_NO
```

Query 5: Find the properties that are already advertised but not yet rented. Please, avoid duplications.

```
SELECT DISTINCT PROPERTY_NO FROM ADVERTISEMENT
WHERE PROPERTY_NO NOT IN (SELECT DISTINCT PROPERTY_NO
                          FROM RENTAL_AGREEMENT)
```

Query 6: Give the names and the branch numbers of all the staff members working in the branch which is located in Hamilton. The names should be listed in an alphabetic order (by last, then by first, then by middle names).

```
SELECT FIRST_NAME,MIDDLE_NAME,LAST_NAME,BRANCH_NO
FROM STAFF,BRANCH
WHERE STAFF.ALLOCATED_TO=BRANCH.BRANCH_NO
AND BRANCH.CITY='HAMILTON'
ORDER BY LAST_NAME,FIRST_NAME,MIDDLE_NAME
```

Query 7: Give the staff numbers and the names of all the workers who live on the same street, city, and province as their manager. The names should be listed in an alphabetic order (by last, then by first, then by middle names).

```
SELECT STAFF.STAFF_NO, FIRST_NAME, MIDDLE_NAME, LAST_NAME
FROM STAFF,(SELECT STAFF_NO, BRANCH_NO, STAFF.STREET, STAFF.CITY,
            STAFF.PROVINCE FROM STAFF,BRANCH
            WHERE STAFF.STAFF_NO = BRANCH.MANAGER) AS T
WHERE STAFF.ALLOCATED_TO = T.BRANCH_NO AND
      STAFF.STAFF_NO != T.STAFF_NO AND
      STAFF.STREET = T.STREET AND
      STAFF.CITY = T.CITY AND
      STAFF.PROVINCE = T.PROVINCE
ORDER BY LAST_NAME, FIRST_NAME, MIDDLE_NAME
```

Query 8: Find the branch number and the average salary of the branch that has the highest average salary. Please, call the branch number as branch_no and the average salary as avg_salary.

```
(SELECT ALLOCATED_TO AS BRANCH_NO, AVG(SALARY) AS AVG_SALARY
 FROM STAFF GROUP BY ALLOCATED_TO)
EXCEPT
(SELECT T1.ALLOCATED_TO AS BRANCH_NO, T1.AVG_SALARY
 FROM (SELECT ALLOCATED_TO, AVG(SALARY) AS AVG_SALARY
       FROM STAFF GROUP BY ALLOCATED_TO) AS T1,
      (SELECT ALLOCATED_TO, AVG(SALARY) AS AVG_SALARY
       FROM STAFF GROUP BY ALLOCATED_TO) AS T2
 WHERE T1.AVG_SALARY < T2.AVG_SALARY)
```

Query 9: Find the owners and renters who have 2 or more phone numbers. Call the owner/renter number as customer_no, set the value of type_of_customer to 'owner' if the customer is an owner, and to 'renter' if he/she is a renter. Please, only list the customer_no and type_of_customer.

```
(SELECT OWNER_NO AS CUSTOMER_NO, 'OWNER' AS TYPE_OF_CUSTOMER
 FROM OWNER
 WHERE OWNER_NO IN (SELECT OWNER_NO FROM OWNER_PHONE
                    GROUP BY OWNER_NO HAVING COUNT(*) >= 2))
UNION
(SELECT RENTER_NO AS CUSTOMER_NO, 'RENTER' AS TYPE_OF_CUSTOMER
 FROM RENTER
 WHERE RENTER_NO IN (SELECT RENTER_NO FROM RENTER_PHONE
                     GROUP BY RENTER_NO HAVING COUNT(*) >= 2))
```

Query 10: Assuming that each advertisement costs 100 dollars, give the branch number and the amount spent on the advertisements for each branch. Name the branch number as branch_no, and the amount as ad_cost.

```
SELECT T2.ALLOCATED_TO AS BRANCH_NO, SUM(C)*100 AS AD_COST
```

```
FROM (SELECT PROPERTY_NO, COUNT(*) AS C
      FROM ADVERTISEMENT GROUP BY PROPERTY_NO) AS T1,
      (SELECT PROPERTY_NO, ALLOCATED_TO FROM PROPERTY,STAFF
WHERE PROPERTY.OVERSEEN_BY = STAFF.STAFF_NO) AS T2
WHERE T1.PROPERTY_NO = T2.PROPERTY_NO
GROUP BY T2.ALLOCATED_TO
```

Chapter 10: Software Project

10.1 Software Project Informal Description

Our website manages software projects for downloads to users. Each software project has a unique project id (8 characters long), can be assigned one or more categories (the categories are A, B ,C and D), has a status (D or P), and has a description (text of at most 256 characters). Some projects may depend on other projects and we keep track of the dependency. Each project is developed and owned by a single developer (who is our subscriber), and uploaded to our website in one or more transactions.

Our users are identified by name (at most 20 characters), email (at most 20 characters), and a unique user id (8 characters long). They can be either guest users or subscribed users (subscribers for short). The subscribers have passwords (at most 8 characters) and we keep the date of the subscription. They need the password to access our website to file bug reports or upload software projects or update patches. A user can download any project, the number of downloads per user per project is recorded. The subscribers can file bug reports for any project. Every bug identified has an id (a positive integer) and a description (text of at most 256 characters). The bug id's must be unique for all bugs concerning the same project. The date of filing of a bug report is recorded. Each bug report deals with a single project and can report a single bug. Each bug report is made by a single subscriber.

Some of our subscribers are developers. They develop the software projects and also software updates for their own projects. Each update for a project has an id (8 characters long), a name (at most 20 characters), a status (P or U), a description (text of at most 256 characters), and is assigned a particular type (the type are 1, 2 and 3). Each update for a project is created by a single developer, the one who originally created the project. Each update patch is uploaded to our website in a transaction.

Each transaction has an id (6 characters long) and a date when it took place. The transaction id's must be unique for all transactions concerning the same project.

10.2 Software Project Logical Model

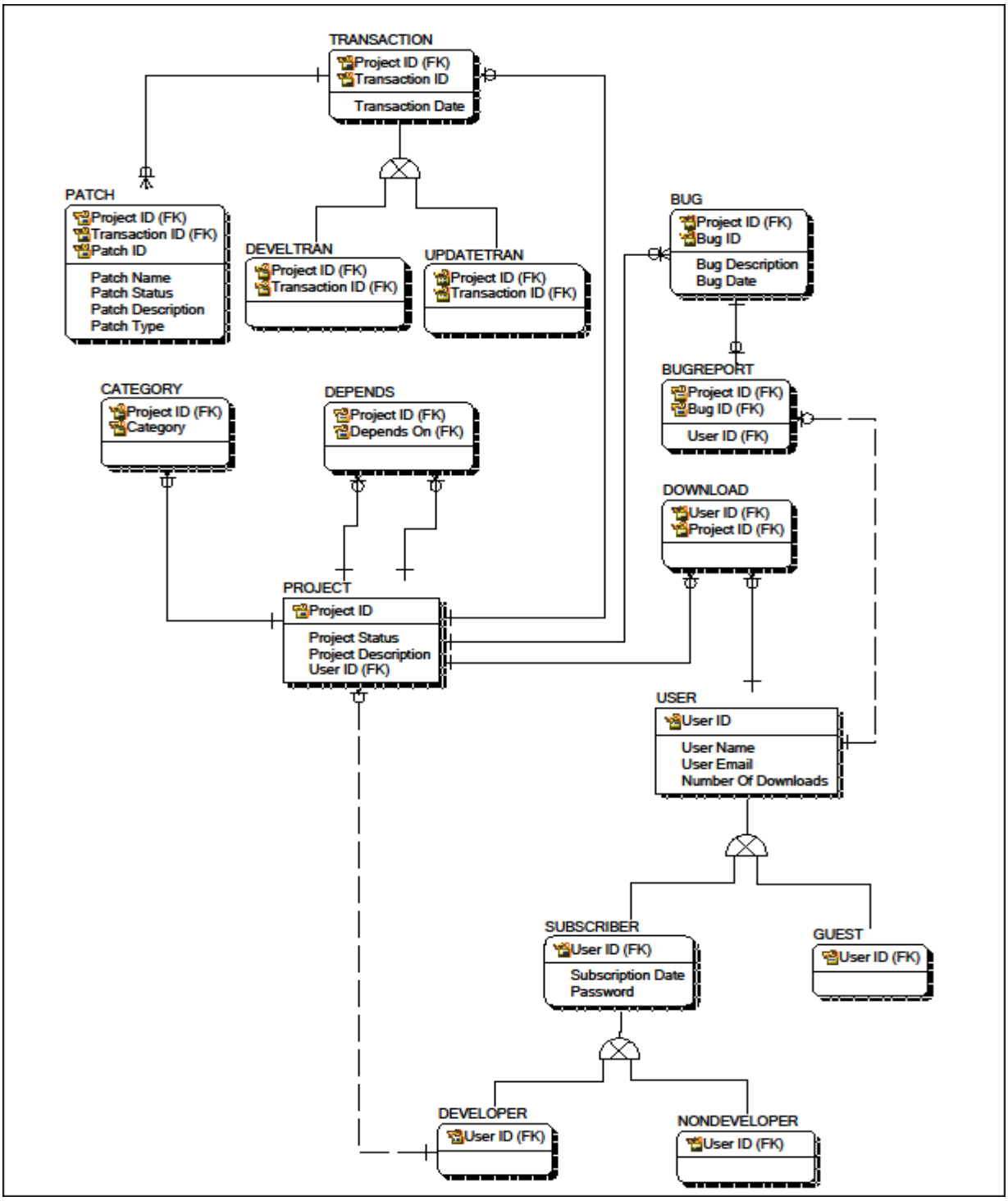


Figure 16: The Software Project Logical Model

10.3 Software Project Physical DB2 Model

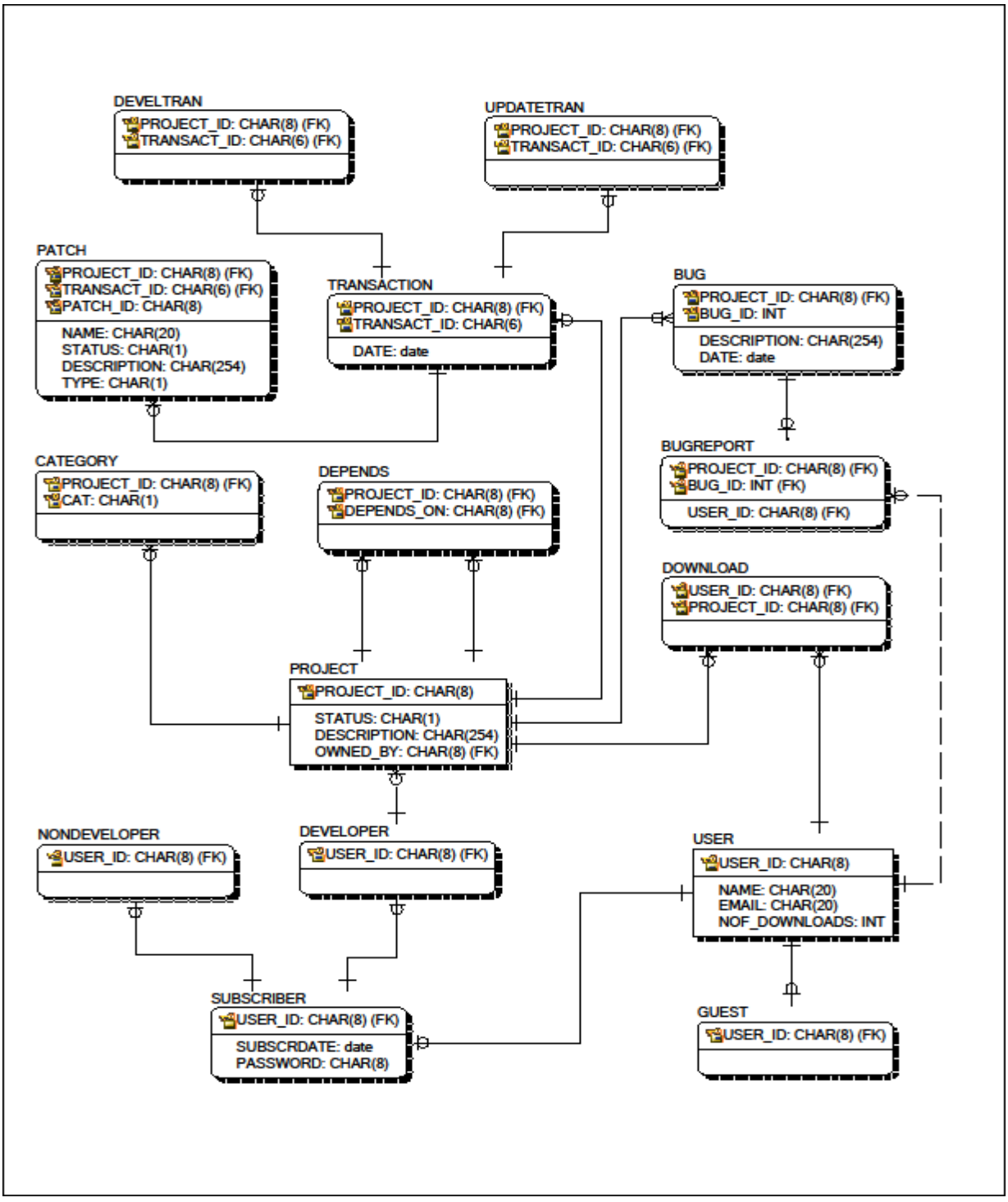


Figure 17: The Software Project Physical Model

10.4 Software Project DB2 Schema

```
CREATE TABLE USER (
       USER_ID CHAR(8) NOT NULL,
       NAME CHAR(20) NOT NULL,
       EMAIL CHAR(20),
       NOF_DOWNLOADS INTEGER NOT NULL,
       PRIMARY KEY (USER_ID)
)
CREATE TABLE GUEST (
       USER_ID CHAR(8) NOT NULL,
       PRIMARY KEY (USER_ID),
       FOREIGN KEY (USER_ID) REFERENCES USER
)
CREATE TABLE SUBSCRIBER (
       USER_ID CHAR(8) NOT NULL,
       SUBSCRDATE DATE NOT NULL,
       PASSWORD CHAR(8) NOT NULL,
       PRIMARY KEY (USER_ID),
       FOREIGN KEY (USER_ID) REFERENCES USER
)
CREATE TABLE NONDEVELOPER (
       USER_ID CHAR(8) NOT NULL,
       PRIMARY KEY (USER_ID),
       FOREIGN KEY (USER_ID) REFERENCES SUBSCRIBER
)
CREATE TABLE DEVELOPER (
      USER_ID CHAR(8) NOT NULL,
      PRIMARY KEY (USER_ID),
      FOREIGN KEY (USER_ID) REFERENCES SUBSCRIBER
)
CREATE TABLE PROJECT (
       PROJECT_ID CHAR(8) NOT NULL,
       STATUS CHAR(1) NOT NULL,
       DESCRIPTION CHAR(254),
       OWNED_BY CHAR(8) NOT NULL,
       PRIMARY KEY (PROJECT_ID),
       FOREIGN KEY (OWNED_BY) REFERENCES DEVELOPER
)
CREATE TABLE DEPENDS (
       PROJECT_ID CHAR(8) NOT NULL,
       DEPENDS_ON CHAR(8) NOT NULL,
       PRIMARY KEY (PROJECT_ID,DEPENDS_ON),
       FOREIGN KEY (PROJECT_ID) REFERENCES PROJECT,
       FOREIGN KEY (DEPENDS_ON) REFERENCES PROJECT
)
CREATE TABLE CATEGORY (
       PROJECT_ID CHAR(8) NOT NULL,
       CAT CHAR(1) NOT NULL,
       PRIMARY KEY (PROJECT_ID,CAT),
       FOREIGN KEY (PROJECT_ID) REFERENCES PROJECT
)

CREATE TABLE DOWNLOAD (
```

```
        PROJECT_ID CHAR(8) NOT NULL,
        USER_ID CHAR(8) NOT NULL,
        PRIMARY KEY (USER_ID,PROJECT_ID),
        FOREIGN KEY (USER_ID) REFERENCES USER,
        FOREIGN KEY (PROJECT_ID) REFERENCES PROJECT
)
CREATE TABLE BUG (
        PROJECT_ID CHAR(8) NOT NULL,
        BUG_ID INTEGER NOT NULL,
        DESCRIPTION CHAR(254),
        DATE DATE NOT NULL,
        PRIMARY KEY (PROJECT_ID,BUG_ID),
        FOREIGN KEY (PROJECT_ID) REFERENCES PROJECT
)
CREATE TABLE BUGREPORT (
        USER_ID CHAR(8) NOT NULL,
        PROJECT_ID CHAR(8) NOT NULL,
        BUG_ID INTEGER NOT NULL,
        PRIMARY KEY (PROJECT_ID,BUG_ID),
        FOREIGN KEY (PROJECT_ID,BUG_ID) REFERENCES BUG,
        FOREIGN KEY (USER_ID) REFERENCES USER
)
CREATE TABLE TRANSACTION (
        TRANSACT_ID CHAR(6) NOT NULL,
        PROJECT_ID CHAR(8) NOT NULL,
        DATE DATE NOT NULL,
        PRIMARY KEY (PROJECT_ID,TRANSACT_ID),
        FOREIGN KEY (PROJECT_ID) REFERENCES PROJECT
)
CREATE TABLE DEVELTRAN (
        TRANSACT_ID CHAR(6) NOT NULL,
        PROJECT_ID CHAR(8) NOT NULL,
        PRIMARY KEY (PROJECT_ID,TRANSACT_ID),
        FOREIGN KEY (PROJECT_ID,TRANSACT_ID) REFERENCES TRANSACTION
)
CREATE TABLE UPDATETRAN (
        TRANSACT_ID CHAR(6) NOT NULL,
        PROJECT_ID CHAR(8) NOT NULL,
        PRIMARY KEY (PROJECT_ID,TRANSACT_ID),
        FOREIGN   KEY   (PROJECT_ID,TRANSACT_ID)   REFERENCES   TRANSACTION
)
CREATE TABLE PATCH (
        PROJECT_ID CHAR(8) NOT NULL,
        TRANSACT_ID CHAR(6) NOT NULL,
        PATCH_ID CHAR(8) NOT NULL,
        NAME CHAR(20) NOT NULL,
        STATUS CHAR(1) NOT NULL,
        DESCRIPTION CHAR(254),
        TYPE CHAR(1) NOT NULL,
        PRIMARY KEY (PROJECT_ID,TRANSACT_ID,PATCH_ID),
        FOREIGN KEY (PROJECT_ID,TRANSACT_ID) REFERENCES TRANSACTION
)
```

10.5 Software Project Interactive Queries

Query 1: Give user id and name of all developers who own a project.

```
SELECT DEVELOPER.USER_ID,USER.NAME
FROM DEVELOPER,USER
WHERE DEVELOPER.USER_ID=USER.USEr_ID
AND DEVELOPER.USER_ID IN (SELECT OWNED_BY FROM PROJECT
                          WHERE OWNED_BY IS NOT NULL)
```

Query 2: Give project id of all projects that have more than two update patches.

```
SELECT PROJECT_ID
FROM (SELECT PROJECT.PROJECT_ID,C
      FROM PROJECT, (SELECT PROJECT_ID,COUNT(*) AS C
                     FROM PATCH GROUP BY PROJECT_ID) AS T
      WHERE PROJECT.PROJECT_ID=T.PROJECT_ID)
WHERE C > 2
```

Query 3: For each project, give the number of all update patches and the number of all downloads.

```
SELECT T.PROJECT_ID,PATCH_COUNT,DOWNLOAD_COUNT
FROM (SELECT PROJECT_ID,COUNT(*) AS PATCH_COUNT
      FROM PATCH GROUP BY PROJECT_ID) AS T,
     (SELECT PROJECT_ID,COUNT(*) AS DOWNLOAD_COUNT
      FROM DOWNLOAD GROUP BY PROJECT_ID) AS S
WHERE T.PROJECT_ID=S.PROJECT_ID
```

Query 4: Give the project id of the projects with the most downloads.

```
SELECT PROJECT_ID
FROM (SELECT PROJECT_ID,COUNT(*) AS DOWNLOAD_COUNT
      FROM DOWNLOAD GROUP BY PROJECT_ID),
     (SELECT MAX(DC) AS MAXDC
      FROM (SELECT COUNT(*) AS DC FROM DOWNLOAD
            GROUP BY PROJECT_ID))
WHERE DOWNLOAD_COUNT=MAXDC
```

Query 5: Give the project id of the projects with the most update patches.

```
SELECT PROJECT_ID
FROM (SELECT PROJECT_ID,COUNT(*) AS PATCH_COUNT
        FROM PATCH GROUP BY PROJECT_ID),
       (SELECT MAX(PC) AS MAXPC
        FROM (SELECT COUNT(*) AS PC
              FROM PATCH GROUP BY PROJECT_ID))
WHERE PATCH_COUNT=MAXPC
```

Query 6: For each project, give project id of all projects that the project depends on.

```
SELECT *
FROM DEPENDS
```

Query 7: Give project id of all projects that do not depend on any other project.

```
(SELECT distinct PROJECT_ID FROM PROJECT)
EXCEPT
(SELECT PROJECT_ID FROM DEPENDS)
```

Query 8: Give the project id of the projects that depend on the most other projects.

```
SELECT T.PROJECT_ID
FROM (SELECT PROJECT_ID,COUNT(*) AS DEPEND_COUNT
      FROM DEPENDS GROUP BY PROJECT_ID) AS T,
     (SELECT MAX(DC) AS MAXDC
      FROM (SELECT COUNT(*) AS DC
            FROM DEPENDS GROUP BY PROJECT_ID))
WHERE T.DEPEND_COUNT = MAXDC
```

Query 9: Give description, bug id, and project id of all bug reports for all projects that have most bug reports.

```
SELECT BUG_ID,BUG.PROJECT_ID,DESCRIPTION
FROM BUG, (SELECT T.PROJECT_ID,T.BC
           FROM (SELECT PROJECT_ID,COUNT(*) AS BC
                 FROM BUG GROUP BY PROJECT_ID) AS T,
                (SELECT MAX(BC1) AS MAXBC
                 FROM (SELECT COUNT(*) AS BC1
                 FROM BUG GROUP BY PROJECT_ID))
           WHERE T.BC=MAXBC) AS S
WHERE BUG.PROJECT_ID=S.PROJECT_ID
```

Query 10: Give user id of a developer with the least amount of bug reports.

```
SELECT USER_ID
FROM (SELECT USER_ID,COUNT(*) AS BPD
      FROM (SELECT BUG_ID,BUG.PROJECT_ID,USER_ID
            FROM BUG,PROJECT,DEVELOPER
            WHERE BUG.PROJECT_ID=PROJECT.PROJECT_ID
            AND OWNED_BY=USER_ID)
      GROUP BY USER_ID)
WHERE BPD IN (SELECT MIN(BPD) AS MINBPD
              FROM (SELECT USER_ID,COUNT(*) AS BPD
              FROM (SELECT BUG_ID,BUG.PROJECT_ID,USER_ID
                    FROM BUG,PROJECT,DEVELOPER
                    WHERE BUG.PROJECT_ID=PROJECT.PROJECT_ID
```

Chapter 11:Tour Operator System

10.1 Tour Operator System Informal Description

The system need to keep track of people. For each person, it records all his/her address, of which exactly one is designated as the mailing address (so each person has at least one address). Each address consists of country, province/state, city, street, street number, P.O. Box number, and a list (possible empty) of phone numbers to the location of the address and a list (possible empty) of fax numbers to the location of the address. In addition to the list of addresses for each person it records a list (possible empty) of cell phone numbers and a list (possible empty) of email address. Each person in the database can be an old customer (have taken a tour of the company), a current customer (is booked to take a tour or is on a tour right now), a tour guide, an employee (works for the tour company), or any mixture of these (for instance an employee can take a tour and so can be a customer as well, or an employee can work as a tour guide for a particular tour and hence be an employee and a guide at the same time etc.). The sex and age of each person must also be recorded, a date-of-birth is optional for an external worker, a contract reference for each of the tours the guide is doing must be included. A guide contract references the tour (see below) and the total amount the tour guide will be paid for the tour. The guides do not pay for the accommodation and the meals.

The system also keeps track of all tours, past and future. Each tour has a unique designation, itinerary, guide (at least one, but may be more than one), its status (completed, in-progress, in-the-future), and the list of participants (not including the guides). The itinerary consists of list of the dates the tour covers and for each date it includes the place of breakfast, the place of lunch, the place of diner, and the place of accommodation. For each of the places there is a contract reference. Each day in the itinerary also includes and a simple English description of the activities during that day.

An accommodation can be a hotel, or a rented room or rooms from a rental company, or a rented room or rooms from a private person. A meal (breakfast, lunch, dinner) can be in hotel, restaurant, or a private place. The contract for accommodation or meal must bear the date of the contract becomes valid, the date or dates it covers, what the contract is for (accommodation, breakfast, lunch, dinner) if the pricing is per person or per group or per room or per the whole facility, per night or per a certain period and the corresponding price. It also may stipulate the minimum and the maximum of people for the accommodation/meal for each day it covers, financial penalty if less than minimum

uses the accommodation. All prices are assumed to be in Canadian dollars, not conversion is involved, regardless where the place is. Each place is identified by a single address. Each provider of accommodation or meal has a unique designation.

11.2 Tour Operator System Logical Model

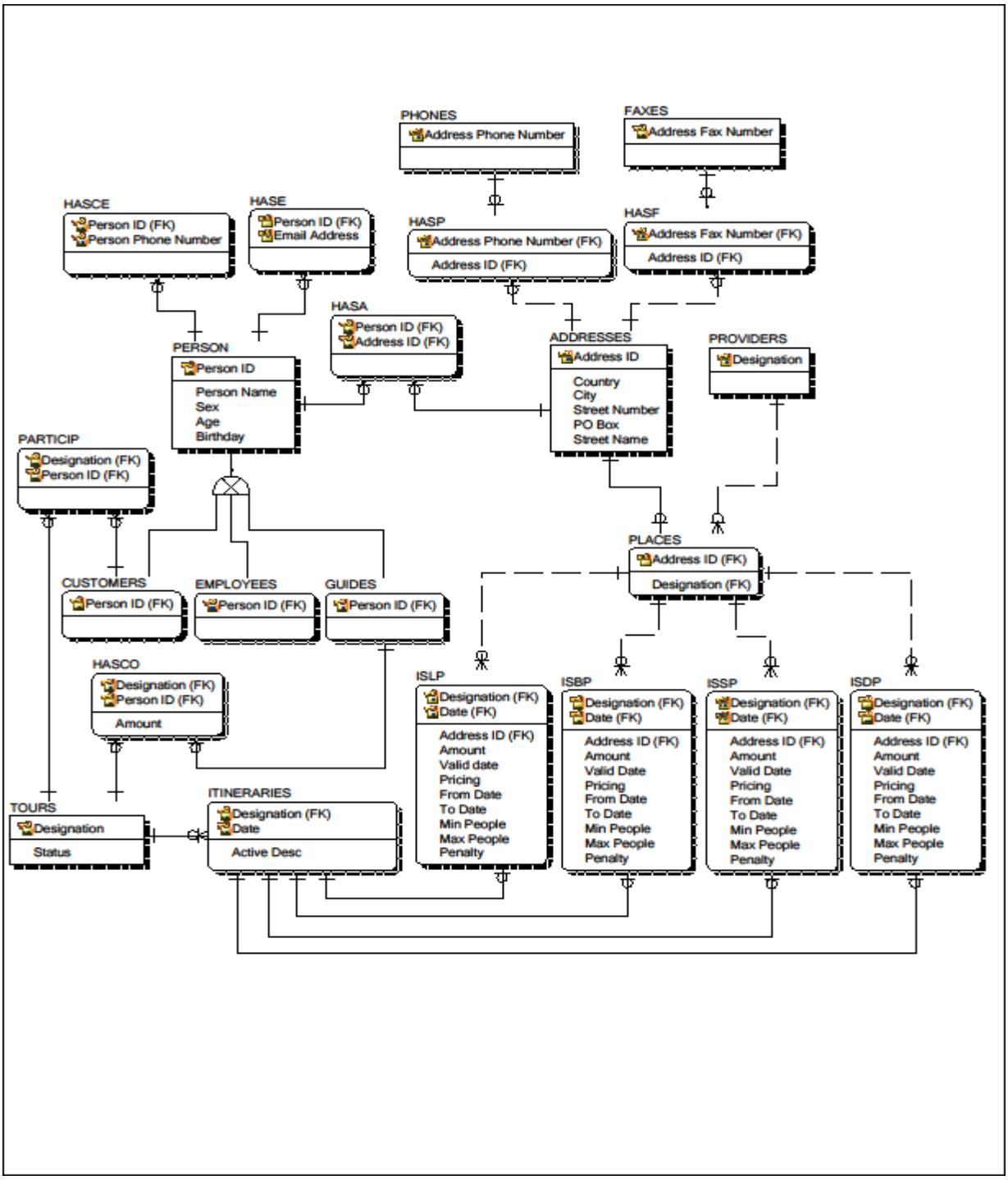


Figure 18: The Tour Operator System Logical Model

11.3 Tour Operator System Physical DB2 Model

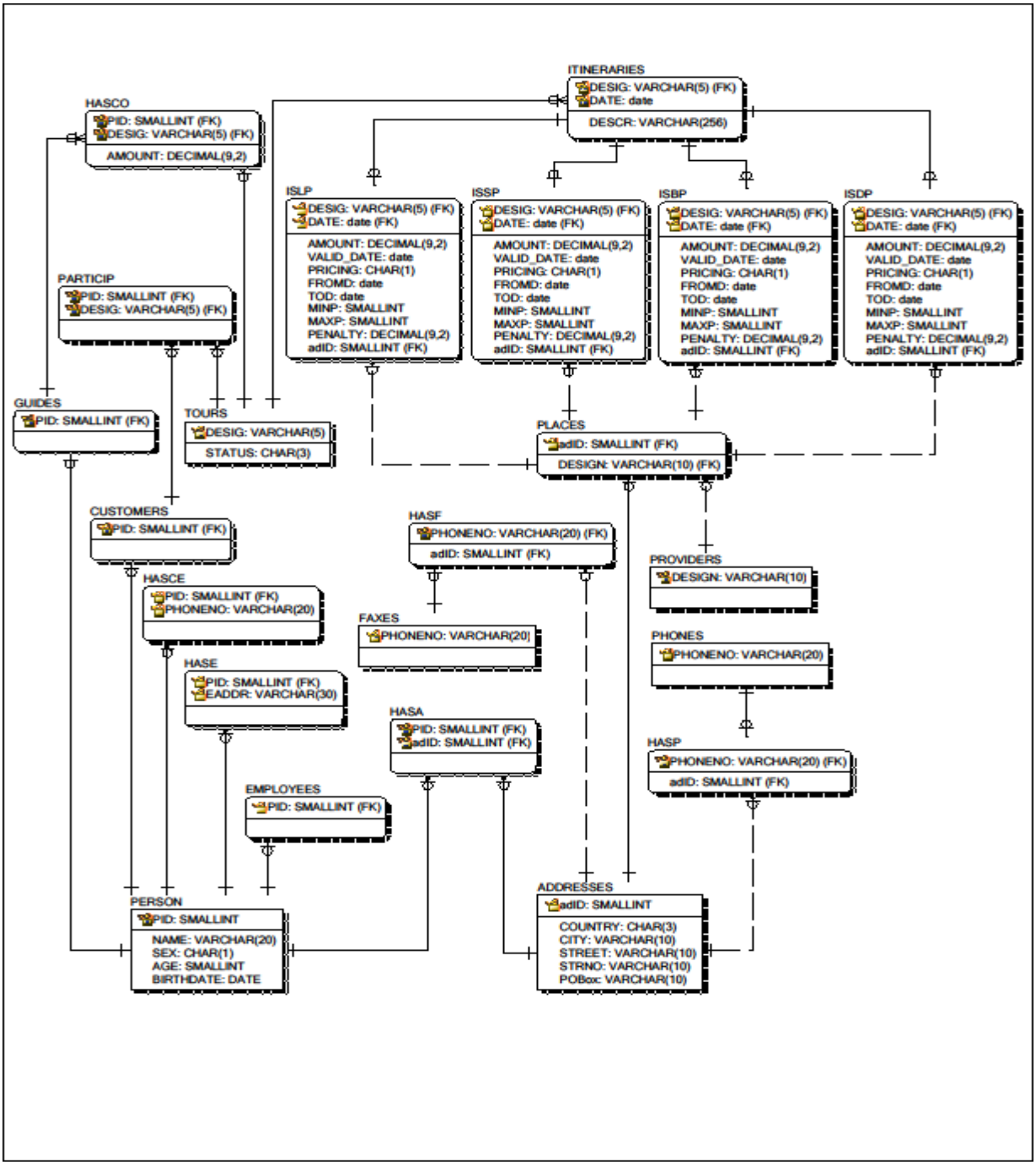


Figure 19: The Tour Operator System Physical Model

11.4 Tour Operator System DB2 schema

```
CREATE TABLE PERSONS(
       PID SMALLINT NOT NULL,
       NAME VARCHAR(20) NOT NULL,
       SEX CHAR(1) NOT NULL CHECK (SEX IN ('M','F')),
       AGE SMALLINT NOT NULL CHECK (AGE BETWEEN 0 AND 150),
       BIRTHDATE DATE NOT NULL,
       PRIMARY KEY(PID),
       CONSTRAINT AGECHECK CHECK (2003-YEAR(BIRTHDATE)=AGE)
)
CREATE TABLE HASE(
       PID SMALLINT NOT NULL,
       EADDR VARCHAR(30) NOT NULL,
       PRIMARY KEY(PID,EADDR),
       FOREIGN KEY (PID) REFERENCES PERSONS (PID)
)
CREATE TABLE HASCE(
       PID SMALLINT NOT NULL,
       PHONENO VARCHAR(20) NOT NULL,
       PRIMARY KEY(PID,PHONENO),
       FOREIGN KEY (PID) REFERENCES PERSONS (PID)
)
CREATE TABLE ADDRESSES(
       adID SMALLINT NOT NULL,
       COUNTRY CHAR(3) NOT NULL,
       CITY VARCHAR(10) NOT NULL,
       STREET VARCHAR(10),
       STRNO VARCHAR(10),
       POBox VARCHAR(10),
       PRIMARY KEY(adID),
       CONSTRAINT ADDRCHECK1
       CHECK (NOT ((STREET IS NOT NULL) AND (POBox IS NOT NULL))),
       CONSTRAINT ADDRCHECK2
       CHECK (NOT ((STRNO IS NOT NULL) AND (POBox IS NOT NULL))),
       CONSTRAINT ADDRCHECK3
       CHECK (NOT ((STREET IS NOT NULL) AND (STRNO IS NULL))),
       CONSTRAINT ADDRCHECK4
       CHECK (NOT ((STRNO IS NOT NULL) AND (STREET IS NULL)))
)
CREATE TABLE HASA(
       PID SMALLINT NOT NULL,
       adID SMALLINT NOT NULL,
       PRIMARY KEY(PID,adID),
       FOREIGN KEY (PID) REFERENCES PERSONS (PID),
       FOREIGN KEY (adID) REFERENCES ADDRESSES (adID)
)
CREATE TABLE PHONES(
             PHONENO VARCHAR(20) NOT NULL,
             PRIMARY KEY(PHONENO)
)
CREATE TABLE FAXES(
       PHONENO VARCHAR(20) NOT NULL,
```

```
        PRIMARY KEY(PHONENO)
)
CREATE TABLE HASP(
        adID SMALLINT NOT NULL,
        PHONENO VARCHAR(20) NOT NULL,
        PRIMARY KEY(PHONENO),
        FOREIGN KEY (adID) REFERENCES ADDRESSES (adID),
        FOREIGN KEY (PHONENO) REFERENCES PHONES (PHONENO)
)
CREATE TABLE HASF(
        adID SMALLINT NOT NULL,
        PHONENO VARCHAR(20) NOT NULL,
        PRIMARY KEY(PHONENO),
        FOREIGN KEY (adID) REFERENCES ADDRESSES (adID),
        FOREIGN KEY (PHONENO) REFERENCES FAXES (PHONENO)
)
CREATE TABLE EMPLOYEES(
        PID SMALLINT NOT NULL,
        PRIMARY KEY(PID),
        FOREIGN KEY (PID) REFERENCES PERSONS (PID)
)
CREATE TABLE CUSTOMERS(
        PID SMALLINT NOT NULL,
        PRIMARY KEY(PID),
        FOREIGN KEY (PID) REFERENCES PERSONS (PID)
)
CREATE TABLE GUIDES(
        PID SMALLINT NOT NULL,
        PRIMARY KEY(PID),
        FOREIGN KEY (PID) REFERENCES PERSONS (PID)
)
CREATE TABLE TOURS(
        DESIG VARCHAR(5) NOT NULL,
        STATUS CHAR(3) NOT NULL CHECK (STATUS IN ('P','I','F')),
        PRIMARY KEY(DESIG)
)
CREATE TABLE PARTICIP(
        PID SMALLINT NOT NULL,
        DESIG VARCHAR(5) NOT NULL,
        PRIMARY KEY (PID,DESIG),
        FOREIGN KEY (PID) REFERENCES CUSTOMERS (PID),
        FOREIGN KEY (DESIG) REFERENCES TOURS (DESIG)
)
CREATE TABLE HASCO(
        PID SMALLINT NOT NULL,
        DESIG VARCHAR(5) NOT NULL,
        AMOUNT DECIMAL(9,2) NOT NULL,
        PRIMARY KEY (PID,DESIG),
        FOREIGN KEY (PID) REFERENCES GUIDES (PID),
        FOREIGN KEY (DESIG) REFERENCES TOURS (DESIG)
)
CREATE TABLE ITINERARIES(
        DESIG VARCHAR(5) NOT NULL,
        DATE DATE NOT NULL,
```

```
        DESCR VARCHAR(256),
        PRIMARY KEY (DESIG,DATE),
        FOREIGN KEY (DESIG) REFERENCES TOURS (DESIG)
)
CREATE TABLE PROVIDERS(
        DESIGN VARCHAR(10) NOT NULL,
        PRIMARY KEY (DESIGN)
)
CREATE TABLE PLACES(
        adID SMALLINT NOT NULL,
        DESIGN VARCHAR(10) NOT NULL,
        PRIMARY KEY (adID),
        FOREIGN KEY (adID) REFERENCES ADDRESSES (adID),
        FOREIGN KEY (DESIGN) REFERENCES PROVIDERS (DESIGN)
)
CREATE TABLE ISBP(
        AMOUNT DECIMAL(9,2) NOT NULL,
        VALID_DATE DATE NOT NULL,
        PRICING CHAR(1) NOT NULL CHECK (PRICING IN ('G','P')),
        FROMD DATE NOT NULL,
        TOD DATE NOT NULL,
        MINP SMALLINT NOT NULL,
        MAXP SMALLINT NOT NULL,
        PENALTY DECIMAL(9,2) NOT NULL,
        DESIG VARCHAR(5) NOT NULL,
        DATE DATE NOT NULL,
        adID SMALLINT NOT NULL,
        PRIMARY KEY (DESIG,DATE),
        FOREIGN KEY (DESIG,DATE) REFERENCES ITINERARIES (DESIG,DATE),
        FOREIGN KEY (adID) REFERENCES PLACES (adID),
        CONSTRAINT ISBP_DATE1 CHECK (VALID_DATE < FROMD),
        CONSTRAINT ISBP_DATE2 CHECK (FROMD <= TOD),
        CONSTRAINT ISBP_DATE3 CHECK (FROMD <= DATE),
        CONSTRAINT ISBP_DATE4 CHECK (DATE <= TOD),
        CONSTRAINT ISBP_PER1 CHECK (MINP <= MAXP)
)
CREATE TABLE ISLP(
        AMOUNT DECIMAL(9,2) NOT NULL,
        VALID_DATE DATE NOT NULL,
        PRICING CHAR(1) NOT NULL CHECK (PRICING IN ('G','P')),
        FROMD DATE NOT NULL,
        TOD DATE NOT NULL,
        MINP SMALLINT NOT NULL,
        MAXP SMALLINT NOT NULL,
        PENALTY DECIMAL(9,2) NOT NULL,
        DESIG VARCHAR(5) NOT NULL,
        DATE DATE NOT NULL,
        adID SMALLINT NOT NULL,
        PRIMARY KEY (DESIG,DATE),
        FOREIGN KEY (DESIG,DATE) REFERENCES  ITINERARIES (DESIG,DATE),
        FOREIGN KEY (adID) REFERENCES PLACES (adID),
        CONSTRAINT ISLP_DATE1 CHECK (VALID_DATE < FROMD),
        CONSTRAINT ISLP_DATE2 CHECK (FROMD <= TOD),
        CONSTRAINT ISLP_DATE3 CHECK (FROMD <= DATE),
```

```
        CONSTRAINT ISLP_DATE4 CHECK (DATE <= TOD),
        CONSTRAINT ISLP_PER1 CHECK (MINP <= MAXP)
)
CREATE TABLE ISDP(
        AMOUNT DECIMAL(9,2) NOT NULL,
        VALID_DATE DATE NOT NULL,
        PRICING CHAR(1) NOT NULL CHECK (PRICING IN ('G','P')),
        FROMD DATE NOT NULL,
        TOD DATE NOT NULL,
        MINP SMALLINT NOT NULL,
        MAXP SMALLINT NOT NULL,
        PENALTY DECIMAL(9,2) NOT NULL,
        DESIG VARCHAR(5) NOT NULL,
        DATE DATE NOT NULL,
        adID SMALLINT NOT NULL,
        PRIMARY KEY (DESIG,DATE),
        FOREIGN KEY (DESIG,DATE) REFERENCES ITINERARIES (DESIG,DATE),
        FOREIGN KEY (adID) REFERENCES PLACES (adID),
        CONSTRAINT ISDP_DATE1 CHECK (VALID_DATE < FROMD),
        CONSTRAINT ISDP_DATE2 CHECK (FROMD <= TOD),
        CONSTRAINT ISDP_DATE3 CHECK (FROMD <= DATE),
        CONSTRAINT ISDP_DATE4 CHECK (DATE <= TOD),
        CONSTRAINT ISDP_PER1 CHECK (MINP <= MAXP)
)
CREATE TABLE ISSP(
        AMOUNT DECIMAL(9,2) NOT NULL,
        VALID_DATE DATE NOT NULL,
        PRICING CHAR(1) NOT NULL CHECK (PRICING IN ('G','P')),
        FROMD DATE NOT NULL,
        TOD DATE NOT NULL,
        MINP SMALLINT NOT NULL,
        MAXP SMALLINT NOT NULL,
        PENALTY DECIMAL(9,2) NOT NULL,
        DESIG VARCHAR(5) NOT NULL,
        DATE DATE NOT NULL,
        adID SMALLINT NOT NULL,
        PRIMARY KEY (DESIG,DATE),
        FOREIGN KEY (DESIG,DATE) REFERENCES
                        ITINERARIES (DESIG,DATE),
        FOREIGN KEY (adID) REFERENCES PLACES (adID),
        CONSTRAINT ISSP_DATE1 CHECK (VALID_DATE < FROMD),
        CONSTRAINT ISSP_DATE2 CHECK (FROMD <= TOD),
        CONSTRAINT ISSP_DATE3 CHECK (FROMD <= DATE),
        CONSTRAINT ISSP_DATE4 CHECK (DATE <= TOD),
        CONSTRAINT ISSP_PER1 CHECK (MINP <= MAXP)
)
```

11.5 Tour Operator System Interactive Queries

Query 1: list all customers, old and current.

```
SELECT CUSTOMERS.PID,NAME FROM CUSTOMERS,PERSONS
```

```
WHERE CUSTOMERS.PID=PERSONS.PID
```

Query 2: List all customers with all their addresses.

```
SELECT CUSTOMERS.PID,NAME,COUNTRY,CITY,STREET,STRNO,POBox
FROM CUSTOMERS,PERSONS,ADDRESSES,HASA
WHERE CUSTOMERS.PID=HASA.PID
AND HASA.adID=ADDRESSES.adID AND CUSTOMERS.PID=PERSONS.PID
```

Query 3: For a given guide, find all the tours he/she guided or will guide and the amount he/she got/will get for the guiding the tour

```
SELECT HASCO.PID,NAME,DESIG,AMOUNT FROM HASCO,PERSONS
WHERE HASCO.PID=0 AND PERSONS.PID=0
```

Query 4: List all customers that are also guides.

```
SELECT CUSTOMERS.PID,NAME FROM PERSONS,CUSTOMERS,GUIDES
WHERE CUSTOMERS.PID=GUIDES.PID AND CUSTOMERS.PID=PERSONS.PID
```

Query 5: List all guides that guided a tour that had a lunch in a given place.

```
SELECT DISTINCT HASCO.PID,NAME FROM HASCO,ISLP,PERSONS
WHERE HASCO.DESIG=ISLP.DESIG AND ISLP.adID=100 AND HASCO.PID=PERSONS.PID
```

Query 6: List all contracts that cover dinners in a given place.

```
SELECT DISTINCT AMOUNT,VALID_DATE,PRICING,FROMD,TOD,
                MINP,MAXP,PENALTY,DESIG
FROM ISDP WHERE adID=103
```

Query 7: List all providers that provide sleeping accommodation.

```
SELECT DISTINCT PLACES.DESIGN
FROM PLACES,ISSP WHERE PLACES.adID=ISSP.adID
```

Query 8: List the tours that will have breakfast at a given place on a given date .

```
SELECT TOURS.DESIG FROM TOURS,ISBP
WHERE TOURS.DESIG=ISBP.DESIG AND ISBP.DATE='12/18/2002' AND adID=100
```

Query 9: List all employees that have guided or will guide a tour or who have taken or will taken a tour.

```
SELECT DISTINCT EMPLOYEES.PID,NAME
FROM EMPLOYEES,GUIDES,CUSTOMERS,PERSONS
WHERE (EMPLOYEES.PID=GUIDES.PID OR   EMPLOYEES.PID=CUSTOMERS.PID)
AND EMPLOYEES.PID=PERSONS.PID
```

Query 10: List all customers booked for a tour starting later or on a given date.

```
SELECT DISTINCT PARTICIP.PID,NAME FROM PARTICIP,ITINERARIES,PERSONS
WHERE PARTICIP.DESIG=ITINERARIES.DESIG
AND ITINERARIES.DATE >= '12/18/2003' AND PARTICIP.PID=PERSONS.PID
AND NOT EXISTS (SELECT ITINERARIES.DATE FROM ITINERARIES
                WHERE ITINERARIES.DATE < '12/18/2003'
                AND ITINERARIES.DESIG=PARTICIP.DESIG)
```

Chapter 12: Warehouse System

12.1 Warehouse System Informal Description

Our company has several warehouses, each warehouse is designated by a unique 4-letter symbol (by a letter we mean a..z and A..Z). Each warehouse has several bins that are identified uniquely by numbers (unsigned integers), i.e. each warehouse has bins 0, 1, 2, 3, … Each bin has a particular capacity. In our warehouses (more precisely in the bins in our warehouses) we store parts. Each part is designated by a unique part number (a 5-symbol sequence of digits and letters). Several parts together can form another part. We call such a part "assembly". In the warehouses we store only the constituent parts, but we record the assemblies in our database as it were a part. Assemblies cannot be parts of other assemblies. A part can be a constituent part in at most in one assembly parts arrive in batches. Each batch for a particular part has a unique batch number (unsigned integer) and arrives on a particular date. Each batch has a size, i.e. the number of items in the batch. All items from the same batch are stored together in the same bin (no batch is stored in more than 1 bin). Each item in a batch has a unique item number (unsigned integer). For example: part A1, batch 27, item 1 or part A1, batch 23, item 1 etc.

When a batch arrives, its date-in is recorded. A particular manager checks its arrival, and this fact must be recorded in the database.

Some parts may be backordered. A part can be backordered only by a manager. The manager, the date of the backorder are recorded, and also the quantity backordered. When a backorder shipment arrives, the backorder's remaining quantity is updated (the number of items arrived is subtracted from the remaining quantity), and if it is less or equal to 0, the backorder is deleted, but must be kept for record. There may be only a single current (active) backorder for any parts. Assemblies cannot be backordered, only their constituent parts.

When an item is shipped out of the warehouse, its date-out is recorded together with the employee who checked its shipping.

Employee has a unique employee number (a 6-digit number), phone number(s) (it consists of a 3-digit area code and a 6-digit number an employee can have 0 to many phone numbers), name(s) (it consists of an up=to-10-characters fist name, an up-to-10-characters middle name, and an up-to-20-characters last name, an employee can have 1 to many names), address(s) (it consists of an up-to-6-characters street number, an up-to-20-

characters street name, an up-to-20-characters city name, and a 2-character abbreviation of the province, an employee can have 1 to many address). Some of the employees are managers. Every employee who is not a manager works under supervision of a single manager. Managers do not work under other managers.

12.2 Warehouse System Logical Model

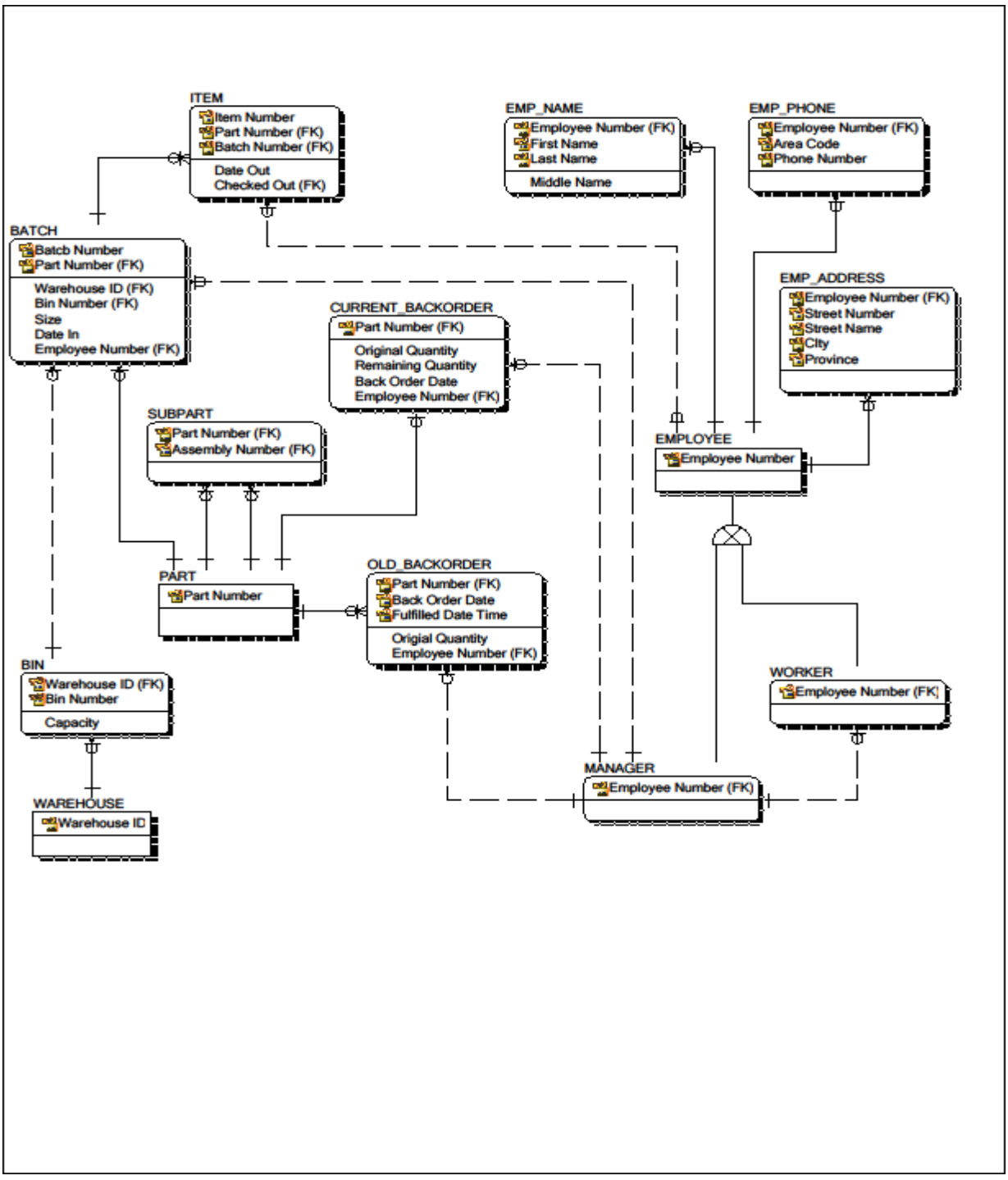


Figure 20: The Warehouse System Logical Model

12.3 Warehouse System Physical DB2 Model

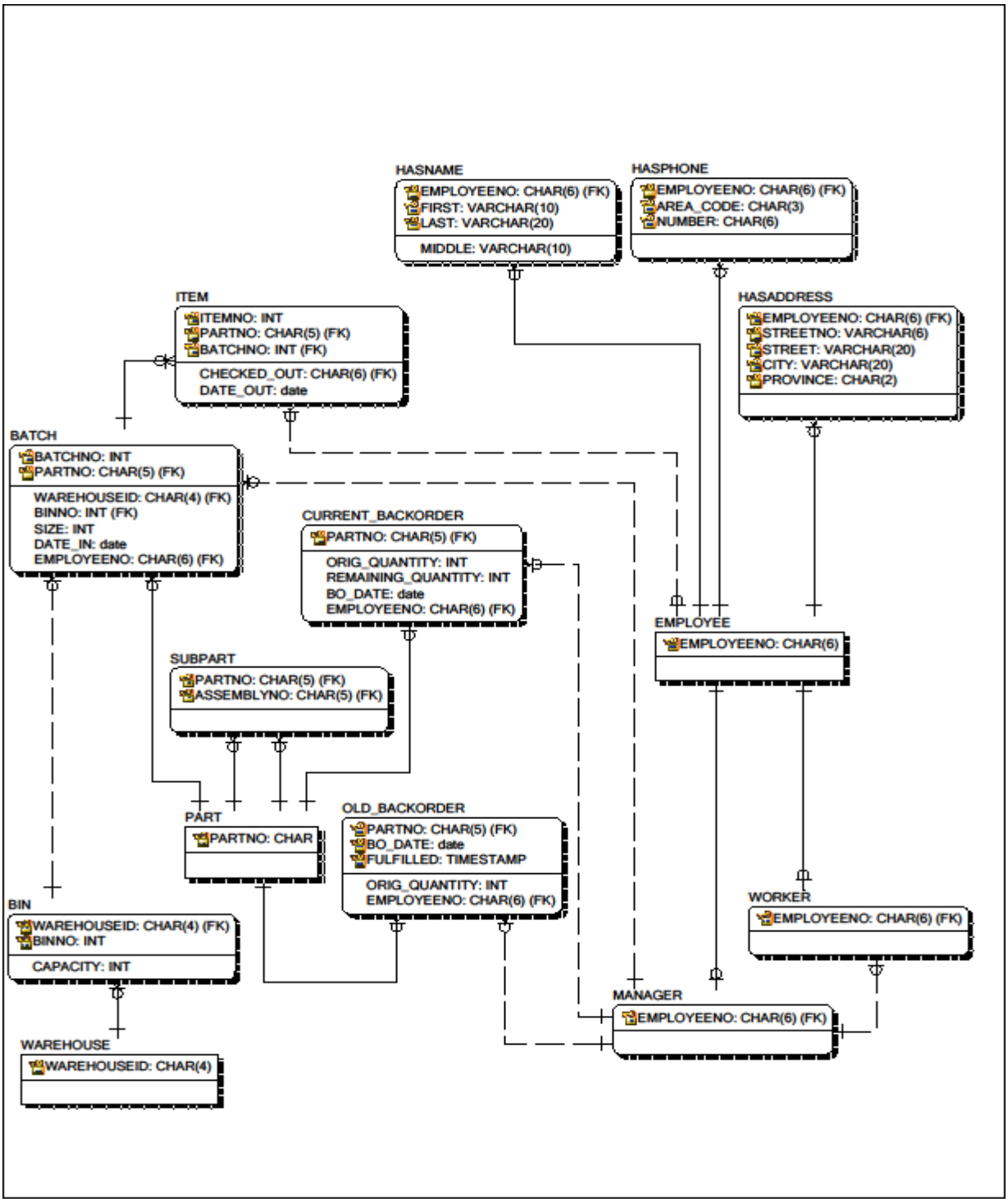


Figure 21: The Warehouse System Physical Model

12.4 Warehouse System DB2 Schema

```
CREATE TABLE EMPLOYEE (
       EMPLOYEENO CHAR(6) NOT NULL,
       PRIMARY KEY (EMPLOYEENO),
CHECK (('0' <= SUBSTR(EMPLOYEENO,1,1) AND SUBSTR(EMPLOYEENO,1,1) <= '9')
AND  ('0' <= SUBSTR(EMPLOYEENO,2,1) AND SUBSTR(EMPLOYEENO,2,1) <= '9')
AND  ('0' <= SUBSTR(EMPLOYEENO,3,1) AND SUBSTR(EMPLOYEENO,3,1) <= '9')
AND  ('0' <= SUBSTR(EMPLOYEENO,4,1) AND SUBSTR(EMPLOYEENO,4,1) <= '9')
AND  ('0' <= SUBSTR(EMPLOYEENO,5,1) AND SUBSTR(EMPLOYEENO,5,1) <= '9')
AND  ('0' <= SUBSTR(EMPLOYEENO,6,1) AND SUBSTR(EMPLOYEENO,6,1) <= '9'))
)
CREATE TABLE MANAGER (
       EMPLOYEENO CHAR(6) NOT NULL,
       PRIMARY KEY (EMPLOYEENO),
       FOREIGN KEY (EMPLOYEENO) REFERENCES EMPLOYEE
)
CREATE TABLE WORKER (
       EMPLOYEENO CHAR(6) NOT NULL,
       MANAGERNO CHAR(6) NOT NULL,
       PRIMARY KEY (EMPLOYEENO),
       FOREIGN KEY (EMPLOYEENO) REFERENCES EMPLOYEE,
       FOREIGN KEY (MANAGERNO) REFERENCES MANAGER(EMPLOYEENO)
)
CREATE TABLE HASPHONE (
       EMPLOYEENO CHAR(6) NOT NULL,
       AREA_CODE CHAR(3) NOT NULL,
       NUMBER CHAR(6) NOT NULL,
       PRIMARY KEY (EMPLOYEENO,AREA_CODE,NUMBER),
       FOREIGN KEY (EMPLOYEENO) REFERENCES EMPLOYEE,
  CHECK (('0' <= SUBSTR(AREA_CODE,1,1) AND SUBSTR(AREA_CODE,1,1) <= '9')
  AND ('0' <= SUBSTR(AREA_CODE,2,1) AND SUBSTR(AREA_CODE,2,1) <= '9')
  AND ('0' <= SUBSTR(AREA_CODE,3,1) AND SUBSTR(AREA_CODE,3,1) <= '9')),
  CHECK (('0' <= SUBSTR(NUMBER,1,1) AND SUBSTR(NUMBER,1,1) <= '9')
  AND ('0' <= SUBSTR(NUMBER,2,1) AND SUBSTR(NUMBER,2,1) <= '9')
  AND ('0' <= SUBSTR(NUMBER,3,1) AND SUBSTR(NUMBER,3,1) <= '9')
  AND ('0' <= SUBSTR(NUMBER,4,1) AND SUBSTR(NUMBER,4,1) <= '9')
  AND ('0' <= SUBSTR(NUMBER,5,1) AND SUBSTR(NUMBER,5,1) <= '9')
  AND ('0' <= SUBSTR(NUMBER,6,1) AND SUBSTR(NUMBER,6,1) <= '9'))
)
CREATE TABLE HASNAME (
       EMPLOYEENO CHAR(6) NOT NULL,
       FIRST VARCHAR(10) NOT NULL,
       MIDDLE VARCHAR(10),
       LAST VARCHAR(20) NOT NULL,
       PRIMARY KEY (EMPLOYEENO,FIRST,LAST),
       FOREIGN KEY (EMPLOYEENO) REFERENCES EMPLOYEE
)
CREATE TABLE HASADDRESS (
       EMPLOYEENO CHAR(6) NOT NULL,
       STRNO VARCHAR(6) NOT NULL,
       STREET VARCHAR(20) NOT NULL,
```

```
        CITY VARCHAR(20) NOT NULL,
        PROVINCE CHAR(2) NOT NULL,
        PRIMARY KEY (EMPLOYEENO,STRNO,STREET,CITY,PROVINCE),
        FOREIGN KEY (EMPLOYEENO) REFERENCES EMPLOYEE
)
CREATE TABLE PART (
        PARTNO CHAR(5) NOT NULL,
        PRIMARY KEY (PARTNO),
        CHECK ((('0' <= SUBSTR(PARTNO,1,1) AND SUBSTR(PARTNO,1,1) <= '9')
        OR ('a' <= SUBSTR(PARTNO,1,1) AND SUBSTR(PARTNO,1,1) <= 'z')
        OR ('A' <= SUBSTR(PARTNO,1,1) AND SUBSTR(PARTNO,1,1) <= 'Z'))
        AND (('0' <= SUBSTR(PARTNO,2,1) AND SUBSTR(PARTNO,2,1) <= '9')
        OR ('a' <= SUBSTR(PARTNO,2,1) AND SUBSTR(PARTNO,2,1) <= 'z')
        OR ('A' <= SUBSTR(PARTNO,2,1) AND SUBSTR(PARTNO,2,1) <= 'Z'))
        AND (('0' <= SUBSTR(PARTNO,3,1) AND SUBSTR(PARTNO,3,1) <= '9')
        OR ('a' <= SUBSTR(PARTNO,3,1) AND SUBSTR(PARTNO,3,1) <= 'z')
        OR ('A' <= SUBSTR(PARTNO,3,1) AND SUBSTR(PARTNO,3,1) <= 'Z')))
)
CREATE TABLE SUBPART (
        PARTNO CHAR(5) NOT NULL,
        ASSEMBLYNO CHAR(5) NOT NULL,
        PRIMARY KEY (PARTNO,ASSEMBLYNO),
        FOREIGN KEY (PARTNO) REFERENCES PART,
        FOREIGN KEY (ASSEMBLYNO) REFERENCES PART(PARTNO)
)
CREATE TABLE WAREHOUSE (
        WAREHOUSEID CHAR(4) NOT NULL,
        PRIMARY KEY (WAREHOUSEID),
        CHECK ((('a' <= SUBSTR(WAREHOUSEID,1,1)
        AND SUBSTR(WAREHOUSEID,1,1) <= 'z')
 OR('A' <= SUBSTR(WAREHOUSEID,1,1) AND SUBSTR(WAREHOUSEID,1,1) <= 'Z'))
 AND(('a' <= SUBSTR(WAREHOUSEID,2,1) AND SUBSTR(WAREHOUSEID,2,1) <= 'z')
 OR ('A' <= SUBSTR(WAREHOUSEID,2,1) AND SUBSTR(WAREHOUSEID,2,1) <= 'Z'))
 AND(('a' <= SUBSTR(WAREHOUSEID,3,1) AND SUBSTR(WAREHOUSEID,3,1) <= 'z')
 OR ('A' <= SUBSTR(WAREHOUSEID,3,1) AND SUBSTR(WAREHOUSEID,3,1) <= 'Z'))
 AND(('a' <= SUBSTR(WAREHOUSEID,4,1) AND SUBSTR(WAREHOUSEID,4,1) <= 'z')
 OR('A' <= SUBSTR(WAREHOUSEID,4,1) AND SUBSTR(WAREHOUSEID,4,1) <= 'Z')))
)
CREATE TABLE BIN (
        WAREHOUSEID CHAR(4) NOT NULL,
        BINNO INTEGER NOT NULL,
        CAPACITY INTEGER NOT NULL,
        PRIMARY KEY (WAREHOUSEID,BINNO),
        FOREIGN KEY (WAREHOUSEID) REFERENCES WAREHOUSE,
        CHECK (BINNO >= 0),
        CHECK (CAPACITY >= 0)
)
CREATE TABLE BATCH (
        PARTNO CHAR(5) NOT NULL,
        BATCHNO INTEGER NOT NULL,
        SIZE INTEGER NOT NULL,
        DATE_IN DATE NOT NULL,
        MANAGERNO CHAR(6) NOT NULL,
        WAREHOUSEID CHAR(4) NOT NULL,
```

```
        BINNO INTEGER NOT NULL,
        PRIMARY KEY (PARTNO,BATCHNO),
        FOREIGN KEY (PARTNO) REFERENCES PART,
        FOREIGN KEY (MANAGERNO) REFERENCES MANAGER(EMPLOYEENO),
        FOREIGN KEY (WAREHOUSEID,BINNO) REFERENCES BIN,
        CHECK (BATCHNO >= 0),
        CHECK (SIZE > 0)
)
CREATE TABLE ITEM (
        PARTNO CHAR(5) NOT NULL,
        BATCHNO INTEGER NOT NULL,
        ITEMNO INTEGER NOT NULL,
        CHECKED_OUT CHAR(6),
        DATE_OUT DATE,
        PRIMARY KEY (PARTNO,BATCHNO,ITEMNO),
        FOREIGN KEY (PARTNO,BATCHNO) REFERENCES BATCH,
        FOREIGN KEY (CHECKED_OUT) REFERENCES EMPLOYEE(EMPLOYEENO),
        CHECK (ITEMNO >= 0)
)
CREATE TABLE CURRENT_BACKORDER (
        PARTNO CHAR(5) NOT NULL,
        ORIG_QUANTITY INTEGER NOT NULL,
        REMAINING_QUANTITY INTEGER NOT NULL,
        BO_DATE DATE NOT NULL,
        BACKORDERED_BY CHAR(6) NOT NULL,
        PRIMARY KEY (PARTNO),
        FOREIGN KEY (PARTNO) REFERENCES PART,
        FOREIGN KEY (BACKORDERED_BY) REFERENCES MANAGER(EMPLOYEENO),
        CHECK (ORIG_QUANTITY > 0),
 CHECK (REMAINING_QUANTITY >= 0 AND REMAINING_QUANTITY <= ORIG_QUANTITY)
)
CREATE TABLE OLD_BACKORDER (
        PARTNO CHAR(5) NOT NULL,
        ORIG_QUANTITY INTEGER NOT NULL,
        BO_DATE DATE NOT NULL,
        BACKORDERED_BY CHAR(6) NOT NULL,
        FULFILLED TIMESTAMP NOT NULL,
        PRIMARY KEY (PARTNO,BO_DATE,FULFILLED),
        FOREIGN KEY (PARTNO) REFERENCES PART,
        FOREIGN KEY (BACKORDERED_BY) REFERENCES MANAGER(EMPLOYEENO),
        CHECK (DATE(FULFILLED) >= BO_DATE)
)
```

12.5 Warehouse System Interactive Queries

Query 1: give all employee_no for all the workers that work under manager with the first name Tony7 and the last name Tona7 with no middle name.

```
SELECT WORKER.EMPLOYEENO
FROM WORKER,
     (SELECT MANAGER.EMPLOYEENO
```

```
      FROM MANAGER,HASNAME
      WHERE MANAGER.EMPLOYEENO=HASNAME.EMPLOYEENO
      AND HASNAME.FIRST='TONY7' AND HASNAME.MIDDLE IS NULL
      AND HASNAME.LAST='TONA7') AS T
WHERE WORKER.MANAGERNO=T.EMPLOYEENO
```

Query 2: give all the names and employee_no for all the workers the names should be listed in an alphabetic order (by last, then by first, then by middle)

```
SELECT WORKER.EMPLOYEENO,FIRST,MIDDLE,LAST ROM WORKER,HASNAME
WHERE WORKER.EMPLOYEENO=HASNAME.EMPLOYEENO
ORDER BY LAST, FIRST, MIDDLE
```

Query 3: give all the phones and employee_no for all the managers.

```
SELECT MANAGER.EMPLOYEENO,AREA_CODE,NUMBER
FROM MANAGER,HASPHONE
WHERE MANAGER.EMPLOYEENO=HASPHONE.EMPLOYEENO
```

Query 4: list all parts that are assemblies they should be listed in a lexicographic order.

```
SELECT DISTINCT ASSEMBLYNO FROM SUBPART ORDER BY ASSEMBLYNO
```

Query 5: for each manager, list all current backorders done by the manager.

```
SELECT MANAGER.EMPLOYEENO,PARTNO,ORIG_QUANTITY,REMAINING_QUANTITY,
       BO_DATE, BACKORDERED_BY
FROM MANAGER,CURRENT_BACKORDER
WHERE MANAGER.EMPLOYEENO=CURRENT_BACKORDER.BACKORDERED_BY
```

Query 6: For each manager, list all current and old backorders done by the manager. For each backorder you have to list the part_no, backorder date, and fulfilled date. For current backorders, list a phony fulfilled date '2000-01-01'.

```
(SELECT MANAGER.EMPLOYEENO, PARTNO, BO_DATE, '2000-01-01' AS FD
 FROM MANAGER,CURRENT_BACKORDER
 WHERE MANAGER.EMPLOYEENO=CURRENT_BACKORDER.BACKORDERED_BY
 UNION
 SELECT MANAGER.EMPLOYEENO, PARTNO, BO_DATE,DATE(FULFILLED) AS FD
 FROM MANAGER,OLD_BACKORDER
 WHERE MANAGER.EMPLOYEENO=OLD_BACKORDER.BACKORDERED_BY
) ORDER  BY EMPLOYEENO
```

Query 7: For each warehouse bin, give the remaining capacity of the bin. Call the remaining capacity remaining_capacity.

```
SELECT T1.WAREHOUSEID, T1.BINNO, CAPACITY-C AS REMAINING_CAPACITY
FROM  (SELECT WAREHOUSEID, BINNO, CAPACITY FROM BIN) AS T1,
```

```
(SELECT BATCH.WAREHOUSEID, BATCH.BINNO, COUNT(ITEMNO) AS C
 FROM ITEM, BATCH  WHERE ITEM.PARTNO=BATCH.PARTNO
 AND ITEM.BATCHNO=BATCH.BATCHNO
 GROUP BY BATCH.WAREHOUSEID, BATCH.BINNO) AS T2
 WHERE T1.WAREHOUSEID=T2.WAREHOUSEID AND T1.BINNO=T2.BINNO
```

Query 8: Give employee_no and number of workers managed for all the managers with The smallest number of workers managed.

```
SELECT MANAGERNO, NUMBER_MANAGED
FROM (SELECT MANAGERNO,COUNT(*) AS NUMBER_MANAGED
      FROM WORKER GROUP BY MANAGERNO) AS T1
EXCEPT
SELECT T1.MANAGERNO, T1.NUMBER_MANAGED
FROM (SELECT MANAGERNO,COUNT(*) AS NUMBER_MANAGED
      FROM WORKER GROUP BY MANAGERNO) AS T1,
     (SELECT MANAGERNO,COUNT(*) AS NUMBER_MANAGED
      FROM WORKER GROUP BY MANAGERNO) AS T2
WHERE T1.NUMBER_MANAGED > T2.NUMBER_MANAGED
```

Conclusion and Future Work

In this thesis we presented a suite of ten projects designed for undergraduate students taking a database course to practise certain aspects of dealing with relational databases. Their main utility and our main objective was to provide projects that are not too simple and that are not too complex, projects that would allow students to practise the most important aspects of dealing with databases. To this end, each project takes a student through four major aspects of dealing with databases:

- Design of the database.
- Implementation of the database.
- SQL querying of the installed database.
- Application programming.

Our projects address some of what we consider most important issues associated with each of the four major aspects, and, of course, leave many other issues untouched.

- In design, every project practises the ER/relational modeling in the context of ERwin modeling tool.
- In implementation, every project practises the creation of the DB2 schema derived from the ER/relational model.
- In SQL querying, in every project simple to moderately complex SQL queries are practised in interactive form.
- In application programming, C and embedded SQL are used. For pedagogical reasons, queries that were first practiced in the interactive querying are used in the application program to illustrate and highlight the differences and the similarities of ESQL and SQL.

For the future work, Oracle and Microsoft SQL Server in addition to DB2 must be included. One of the important omissions in the projects concerns normalization on the level of ER/relational modeling and a problem related to, the intentional de-normalization of the design for performance gains. This omission must be addressed in the future work on this suite of projects. In order to allow more complex applications, ESQL-based application programming in C++ should be also considered for inclusion in near future.

Bibliography

[1] E.F. Codd, **A Relational Model of Data for Large Shared Data Banks**, Communications of ACM, Volume 13, Number 6, 1970, 377--387

[2] **Pioneer calls relational database technology obsolete,** http://www.computerworlduk.com/news/applications/5059/pioneer-calls-relational-database-technology-obsolete/

[3] Michael Stonebraker**, The End of a DBMS Era (Might be Upon Us)** http://cacm.acm.org/blogs/blog-cacm/32212-the-end-of-a-dbms-era-might-be-upon-us/fulltext

[4] Peter Pin-shan Chen, **The Entity-Relationship Model: Toward a Unified View of Data,** ACM Transactions on Database Systems, Volume 1, 1976, 9--36