A Reverse Life-Cycle Database Course With Mini-Projects

Kirby McMaster
kmcmaster@weber.edu

Nicole Anderson
nanderson1@weber.edu

Dona Bilyeu-Dittman
ddittman@weber.edu
Computer Science, Weber State University
Ogden, UT 84408 USA

Abstract

The usual approach to teaching an introductory database course--as presented in curriculum guidelines from professional societies, in database textbooks, and in papers and presentations--is to sequence the topics according to the database development life-cycle. Students proceed from data modeling to database design to database implementation and operations. In this approach, students are often assigned a semester-long project, where they perform life-cycle activities to develop a single database system. In this paper, some problems with the life-cycle approach are discussed, and an alternative reverse life-cycle approach is suggested. With the reverse life-cycle approach, students begin by performing operations on existing databases, and then learn how to implement their own databases. Data modeling and design topics are delayed until students become familiar with database systems. Instead of a semester-long project, students are given a sequence of mini-projects, where each mini-project involves activities within one stage of database development.

Keywords: database, life-cycle, data modeling, entity-relationship model, relational model, database design, SQL.

1. INTRODUCTION

In recent years, there has been a significant amount of activity to define the content of a first database course for Computer Science and Information Systems programs. Professional organizations such as ACM, IEEE, AIS, and AITP have provided lists of recommended topics for database courses in their curriculum guidelines. Authors of database textbooks carefully select topics to include in their texts, choosing those they feel are relevant to today's students. Papers presented at professional meetings have reviewed which topics are commonly included in database courses and what types of projects should be assigned to students.

Less has been written about the order in which database topics should be presented. The implicit topic ordering found in the curriculum guidelines, in most database textbooks, and in many papers follows the database development life-cycle. In the life-cycle approach, the normal topic sequence is data modeling, database design, database implementation, and database operations. The most common type of project in this approach is a semester-long group or individual project, in which a single database application is developed by performing life-cycle tasks.

This paper discusses some problems with the life-cycle approach for database courses, and offers an alternative topic sequence,

referred to as the reverse life-cycle approach. In the reverse life-cycle approach, the topic order is database operations, database implementation, followed by data modeling and database design. When the reverse life-cycle approach is used, a single semester-long project is impractical, since the database would have to be implemented before it is designed. Instead, we recommend a sequence of smaller mini-projects involving different databases. Each mini-project requires students to perform tasks within a specific stage of database development. It has been our experience that teaching a first database course using the reverse life-cycle approach can be very effective for students early in their academic program.

2. THE LIFE-CYCLE APPROACH

In the life-cycle approach, the topic sequence follows the stages in developing a new database system. The data modeling stage describes how to construct a preliminary data model, such as an entity-relationship model or object model. The database design stage covers the conversion of a preliminary data model into a normalized relational model. The implementation stage explains how to create tables and views and specify integrity constraints. The database operations stage focuses on performing queries and data entry.

In 2001, the ACM and IEEE organizations issued a set of Curriculum Guidelines for Computer Science courses (2001). For an introductory database course, the following topics were recommended:

IM1 Information models and systems
IM2 Database systems
IM3 Data modeling
IM4 Relational databases
IM5 Database query languages
IM6 Relational database design
IM7 Transaction processing
IM8 Distributed databases
IM9 Physical database design

The order in which sections IM3 through IM5 are listed approximates the life-cycle approach. The main difference is that database design topics are divided between IM3 (mapping conceptual schema to a relational schema) and IM6 (normal forms).

A Model Curriculum and Guidelines for Information Systems courses was prepared by the ACM, AIS, and AITP organizations in 2002 (Gorgone, 2002). In this curriculum, database topics are embedded within a two-course sequence. The first course--Analysis and Logical Design--includes data modeling and logical database design. The second course--Physical Design and Implementation with DBMS--covers additional data modeling topics (e.g. relational vs. object models), physical database design, and database implementation.

A team-oriented project is recommended for the two-course sequence, where students design and implement a departmental information system that includes a database. The database topics within the two-course sequence closely follow the life-cycle approach.

Each database textbook has its own choice of coverage and sequencing of topics. Fourteen recent database textbooks (versions published since 2000) were examined to see whether they follow the life-cycle approach or a different approach. As an indicator of how closely a textbook follows the life-cycle approach, we recorded which chapter covered data modeling and which chapter covered SQL queries. A textbook representing the life-cycle approach should cover data modeling before SQL queries. The results are summarized as a scatter diagram in Figure 1.

In this sample of database textbooks, nine covered data modeling early (in Chapters 2,3, or 4), before SQL queries (Elmasri, 2004; Garcia-Molina, 2002; Hoffer, 2005; Kifer, 2006; Ramakrishnan, 2003; Riccardi, 2001; Riccardi, 2003; Ricardo, 2004; Rob, 2004). The other five textbooks covered data modeling later, after SQL queries (Connolly, 2005; Date, 2004; Kroenke, 2006; O'Neil, 2000; Silberschatz, 2005). The split between modeling early vs. modeling later would have been greater (11 to 3) if the Kroenke (2006) and Silberschatz (2005) textbooks had not changed the topic ordering in their latest versions. The majority of database textbooks still favor the life-cycle approach, with data modeling presented in an early chapter. But there is recent movement toward later coverage of data modeling in two textbooks, as authors

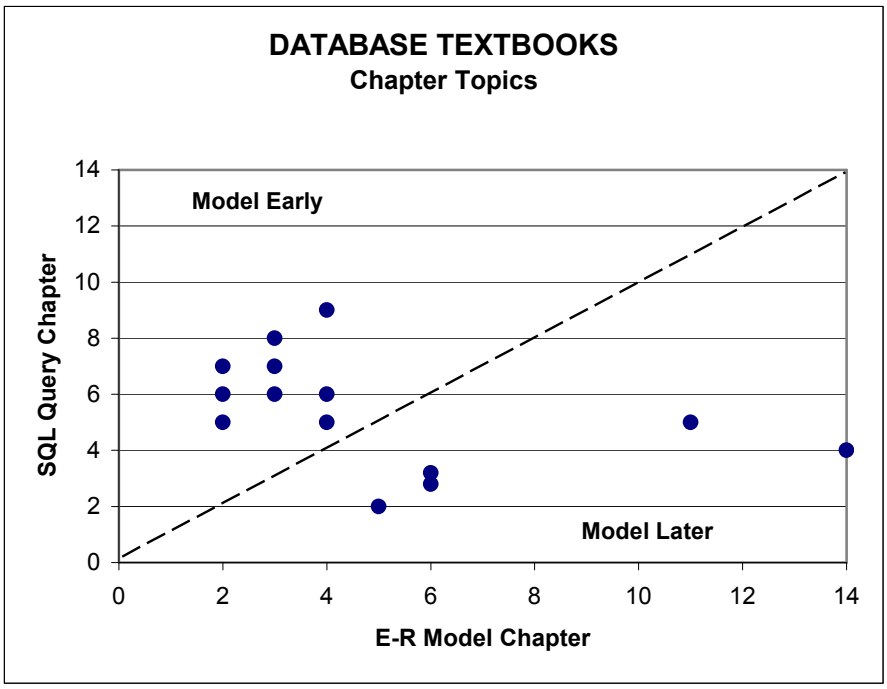

Figure 1: Modeling Early vs. Modeling Later

recognize benefits from offering a non-traditional sequence of topics. Database instructors are not forced to follow the topic sequence presented in the textbooks they adopt, but they are often influenced by it.

Recent papers and presentations on teaching the first database course have focused more on content than topic sequence. One such paper is "Trends in the Evolution of the Database Curriculum" by Robbert and Ricardo (2003), presented at ITiCSE 2003. The authors conducted surveys of database educators in 1999, 2001, and 2002, asking them which database topics are included in their database courses, and how many hours are spent on each topic. For the 106 respondents in the 2001 survey, the five topics with highest weighted average hours were SQL, database design, entity-relationship model, relational model, and normalization. The order in which database topics are or should be taught was not discussed in the paper.

A few papers do suggest or imply a database topic sequence, and seem to recommend the life-cycle approach with semester-long projects. Baugh (2003) presented a paper at ISECON 2003 entitled "A First Course in Database Management." The paper describes her organization of course topics, and how the concepts are taught in the context of individual semester-long projects. Baugh's course content includes:

Topic 1: Database Terminology
Topic 2: Database Design
Topic 3: Relational Database
Topic 4: SQL Language
Topic 5: Normalization
Topic 6: Database Management

This sequence is a slightly modified life-cycle approach, in which the normalization part of database design is covered after SQL (similar to the ACM Curriculum Guidelines). Baugh makes the following statement:

> "This course was designed to allow the student to work on an individual database project while learning the database theory in a concurrent manner."

Adams (2004) was moderator for a panel discussion at SIGCSE 2004 on "Managing the Introductory Database Course: What Goes In and What Comes Out?" Each of the four participants described what topics are important and what types of projects are given to students. There is a strong

consensus as to course content and projects among the four presenters.

> Liz Adams: "We cover both the ER model and the Semantic Object Model. We only consider the relational model and we cover normalization. ... We spend some time on relational algebra and SQL. ... Most of the labs are team projects as is the term project. The term project has a number of sequenced components, each with a specified due date."
>
> Don Goelman: "What does that leave as the 'sine qua non' topics for our course? My vote (and practice) goes to high-level modeling (ER, EER, and UML), principles of the relational model, mappings among the models, abstract query languages, SQL (primarily as DML), relational design theory, and the object data model. ... Three hourly exams, a final, and a semester group project are the chief assessment tools."
>
> Mary Granger: "This course focuses on logical database design, incorporating Entity-Relationship diagrams, relational database and normalization, relational algebra and SQL. ... The team project consists of students creating the entity-relationship diagrams, the tables and relationships in Access, implementing a certain level of functionality...."
>
> Catherine Ricardo: "For all students, it is vital to cover the essentials and to introduce the newer topics.... I assign a semester-long project, to be done in teams. Students design a database and implement it using Oracle."

The first three participants are describing a life-cycle approach for their database courses. Data modeling is covered early, both as a lecture topic and as a project activity. All four participants assign semester-long team projects.

3. PROBLEMS WITH THE LIFE-CYCLE APPROACH

When a database course follows the life-cycle approach, it is not uncommon for students to have difficulty developing entity-relationship models for their projects. Students also have trouble converting the entity-relationship model into a relational model, especially the process of normalizing the tables. This is not surprising, since most students have had very little experience with actual database systems.

During a semester-long project, students should have an easier time with concrete tasks such as creating tables, inserting data, and performing queries. However, student database designs are often poor, so the task of implementing them can be a challenge. Also, with a bad design, desired queries can be difficult or impossible to perform if the required data is not available or is in an inconvenient form.

The main issue here is one that appears in other areas of CS and IS, and it pervades Mathematics. The issue is: when should abstraction be taught? Should we teach abstract concepts first, followed by specific implementations? Or should we give students tangible examples first, and then generalize to abstract concepts? This issue applies to how we teach programming, how we teach object-oriented concepts, and how we teach database courses.

Data modeling is a form of abstraction. When we use the life-cycle approach, we teach students how to develop abstract models for data before they are familiar with actual database systems. Date (2004) does not agree with this approach.

> "Some reviewers of earlier editions complained that database design issues were treated too late. But it is my feeling that students are not ready to design databases properly or to appreciate design issues fully until they have some understanding of what databases are and how they are used."

Kroenke (2006) gives a different reason for not covering data modeling early.

> "Furthermore, today's students are too impatient to start a class with lengthy conceptual discussions on data modeling and database design. They want to do something, see a result, and obtain feedback."

4. THE REVERSE LIFE-CYCLE APPROACH

Consider the following alternative to the life-cycle approach. Because it is difficult for students to develop systems that are unfamiliar, the course starts with the database operations stage of the life-cycle. Students are given sample databases and asked to perform queries and data entry. This allows them to learn what a database system consists of and how it behaves.

Next, during the database implementation stage, students are provided with a well-designed relational model and asked to implement the tables, integrity constraints, and views of a relational database. The relational model can be in the form of a relational model diagram with a data dictionary, or it could be a working prototype (e.g. using Access).

For the analysis/design stages, students learn how to construct a preliminary data model (using entity-relationship modeling or object modeling), and then transform the preliminary data model into a normalized relational model. Another assignment would be to have students improve a badly designed relational model. The ultimate goal of this stage is to obtain a detailed relational model that is in a maintainable, anomaly-free normal form.

When this point in the course has been reached, the reverse life-cycle approach has been accomplished. As each development stage is discussed, students are familiar with the end products for that stage, since they have already experienced the next stage. Their focus can be on methods for creating these end products, or deliverables. Understanding the deliverables implies that students have seen examples of well-designed databases along the way and have a good idea of what they should look like. If time permits, students can be given a short final project that allows them to go through the development process in the usual life-cycle direction.

The reverse life-cycle approach for database courses is not mentioned in the ACM/IEEE Curriculum Guidelines. We also could not find any research papers that describe or recommend this approach. Of the five database textbooks described earlier that are in the Modeling Later group, the one that most closely follows the reverse life-cycle approach is *Database Systems*, by Connolly and Begg (2005). We have used this textbook in our course, and student evaluations of this course have been positive.

Two of the authors have used the reverse life-cycle approach in their database courses over the past four years. In addition to the positive feedback from students and the benefit of allowing students to understand stage deliverables, other advantages are gained using this approach. This topic ordering is very familiar to students, as it is typically followed when learning traditional software development. Students tend to understand the end products of general software development more easily. They often have an idea what software should do and how to test it.

Even so, educators don't ask students to develop a complex software system in the introductory programming course. Students are first taught building blocks such as variables, control statements, functions, and classes, and then build upon these concepts by creating small programs (e.g. a program to perform temperature conversion). Finally, they are asked to put all these ideas together, from requirements collection to the testing phases of development, to build larger software systems.

When this approach is compared to the way students are later exposed to databases in the business world, it is quite similar. They often encounter an existing database and are asked to query it or modify it. It is not until later in their career, when they have more experience, that they are asked to design a database system from the ground up. We see all of these situations as benefits of the reverse life-cycle approach.

5. USING MINI-PROJECTS

When the reverse life-cycle approach is used, it is not practical to assign students a semester-long project involving a single database, since they would have to start with the completed system. It is more instructive to have a number of smaller mini-projects involving different databases. Each mini-project requires students to

perform tasks within a particular stage of development.

Developing good database projects can be time consuming. To assist instructors, we have prepared eight mini-projects that can be used to support the reverse life-cycle approach. Each mini-project requires approximately two weeks to complete. Mini-projects have been prepared for the following topic areas:

1. Files vs. Databases
   data vs. metadata
   file processing vs. database processing
   data independence

2. Relational Databases and Relational Algebra
   attributes, domains, and tables
   relationships: primary keys and foreign keys
   relational algebra operations
   procedural query language

3. Relational Calculus and Query-By-Example
   predicate logic: domains, predicates, facts, rules
   relational calculus expressions
   Query-By-Example (QBE)
   nonprocedural query language

4. Structured Query Language: Queries
   SELECT statement: SELECT...FROM...WHERE
   aggregate functions
   grouping: GROUP BY...HAVING...
   sorting: ORDER BY...
   nested queries

5. Structured Query Language: Data Definition and Data Entry
   CREATE TABLE statement
   integrity constraints
   INSERT, UPDATE, and DELETE statements
   ALTER TABLE statement
   CREATE VIEW statement

6. Data Modeling: Entity-Relationship Model
   entities and attributes
   relationships
   cardinality: 1-1, 1-M, M-M
   entity subtypes

7. Data Modeling: Relational Model Design and Normalization
   update anomalies: insert, update, delete
   functional dependencies, determinants
   normal forms: 1NF, 2NF, 3NF, BCNF

8. Database Development Life Cycle
   modeling: develop preliminary data model (E-R model)
   design: develop normalized relational model
   implementation: create tables, views

Sample problem descriptions for several mini-projects are included in the Appendix. These mini-projects have been classroom tested and revised several times. The key to a suitable mini-project is that its main focus should be on topics and activities within a specific stage of development, although it may include some review of previous topics. Also, it should be possible to complete a mini-project within approximately two weeks.

6. SUMMARY AND CONCLUSIONS

The purpose of this paper is to propose a reverse life-cycle approach for teaching an introductory database course. The traditional life-cycle approach, which is recommended in CS and IS curriculum guidelines, in research papers and presentations, and in most database textbooks, presents topics in the order they are encountered in the process of developing a database system. A semester-long project usually accompanies the course topics in this approach. The main problem with the life-cycle approach is that abstract data modeling is covered before students have become familiar with database operations and implementation.

The reverse life-cycle approach delays data modeling and database design until after students have had experience with actual database systems. As a result of this experience, students are better prepared to make design decisions. Instead of having a single semester-long project, the reverse life-cycle approach is more effective when students are given a sequence of mini-projects involving different databases. Each mini-project has students perform tasks

within a particular stage of database development.

The life-cycle approach may be suitable when the first database course is offered in the senior year, as long as students have had some exposure to databases in previous courses. In this case, a semester-long database development project, especially a team project, allows the database course to serve as a capstone to the student's degree program. However, because the recognized importance of database concepts to CS and IS students is increasing, the trend is for schools to offer the first database course earlier than the senior year, and to offer additional database courses as electives.

In our degree program, the introductory database course is offered at the sophomore level, after students have completed a two-semester programming sequence. Scheduling the first database course early in our program allows us to offer advanced database courses (e.g. Distributed Database Development, Database Administration) as upper-division electives. It also enables other upper division courses such as Web Development and Software Engineering to utilize the knowledge gained in the first database course.

This paper has presented a new and innovative way to teach an introductory database course using a reverse life-cycle approach. This approach allows students to gain critical theoretical background information as well as practical application experience, including practice with modeling and implementing database systems. The reverse life-cycle approach is enhanced by the use of mini-projects that allow students to be more successful in their learning.

So far, there is little mention of the reverse life-cycle approach in the research literature and the curriculum guidelines. However, there is some movement toward the reverse life-cycle approach in two recent database textbooks. Since many teachers plan their courses based on the textbooks they choose, perhaps their approach to teaching database courses will change as textbooks evolve.

7. REFERENCES

Adams, Elizabeth, et al, "Managing the Introductory Database Course: What Goes In and What Comes Out?" SIGCSE 2004.

Baugh, Jeanne M., "A First Course in Database Management." ISECON 2003.

Connolly, Thomas and Carolyn Begg, Database Systems: A Practical Approach to Design, Implementation, and Management (4th ed). Harlow, England: Addison-Wesley, 2005.

Date, C. J., An Introduction to Database Systems (8th ed). Boston, MA: Addison-Wesley, 2004.

Elmasri, Ramez and Shamkant Navathe, Fundamentals of Database Systems (4th ed). Boston, MA: Addison-Wesley, 2004.

Garcia-Molina, Hector, et al, Database Systems: The Complete Book (1st ed). Upper Saddle River, NJ: Prentice Hall, 2002.

Gorgone, John T., et al, "IS 2002: Model Curriculum and Guidelines for Undergraduate Degree Programs in Information Systems." ACM/AIS/AITP, 2002.

Hoffer, Jeffrey, et al, Modern Database Management (7th ed). Upper Saddle River, NJ: Prentice Hall, 2005.

Kifer, Michael, et al, Database Systems: An Application Oriented Approach (2nd ed). Boston, MA: Addison-Wesley, 2006.

Kroenke, David, Database Processing: Fundamentals, Design, and Implementation (10th ed). Englewood Cliffs, NJ: Prentice Hall, 2006.

O'Neil, Patrick and Elizabeth O'Neil, Database: Principles, Programming, and Performance (2nd ed). San Francisco, CA: Morgan Kaufman, 2000.

Ramakrishnan, Raghu and Johannes Gehrke, Database Management Systems (3rd ed). New York: McGraw Hill, 2003.

Riccardi, Greg, Principles of Database Systems with Internet and Java Applications (1st ed). Boston, MA: Addison-Wesley, 2001.

Riccardi, Greg, Database Management: With Website Development Applications (1st ed). Upper Saddle River, NJ: Prentice Hall, 2003.

Ricardo, Catherine, Databases Illuminated (1st ed). Boston, MA: Jones and Bartlett Publishers, 2004.

Rob, Peter and Carlos M. Coronel, Database Systems: Design, Implementation, and Management (6th ed). : Course Technology, 2004.

Robbert, Mary Ann, and Catherine M. Ricardo, "Trends in the Evolution of the Database Curriculum." ITiCSE 2003.

Silberschatz, Abraham, et al, Database System Concepts (5th ed). New York: McGraw Hill, 2005.

The Joint Task Force on Computing Curricula, "Computing Curriculum 2001: Computer Science." ACM/IEEE, 2001.

APPENDIX: SAMPLE MINI-PROJECTS

Project #1. Files vs. Databases

Part A

You will be given a C (or Java) program that performs the following Reorder query on the STOCK.DAT and STKTYPE.DAT data files:

```
For all STOCK records, list StkNo, SType, StkName, QtyOnHand,
and (STKTYPE) ReorderPt and OrderSize in which the QtyOnHand
is at or below the ReorderPt and for which an order has not
yet been placed.
```

You will need to modify this program because the format of the data files has changed.

The old format of the STOCK file, as currently recognized by the program, is as follows:

Columns	Field	DataType
1-3	StkNo	Character (digits)
4	SType	Character (upper-case letters)
5-20	StkName	Character
21-23	QtyOnHand	Integer (>=0)
24	OnOrder	Character (N or Y)

The old format of the STKTYPE file is as follows:

Columns	Field	DataType
1	TType	Character (upper-case letters)
2-13	TypeName	Character
14-16	ReorderPt	Integer (>=0)
17-19	OrderSize	Integer (>=0)

The STOCK and STKTYPE files are ASCII text files. Each record ends with CR/LF bytes.

The new format of the two files includes the following changes:

a. The size of the StkName field has been increased to 18 characters in each STOCK record.
b. A 3-digit LeadTime field has been added at the end of each STKTYPE record.
c. The name and datatype of the OnOrder field in the STOCK table has been changed to QtyOnOrder, with 3-digit integer values instead of 'N' and 'Y'.
d. The number of records in the STKTYPE file has increased from 5 to 6, and 2 records have been added to the STOCK file.

1. Turn in a source code listing of your modified program and a printout of the query output.
2. Summarize (in words) the changes you made to the program. How does this part of the project illustrate the concept of **data independence**?

Part B

A Microsoft Access file called INVENTORY.mdb contains the STOCK and STKTYPE data as tables in the new format. An SQL statement that performs the Reorder query (described in Part A) on data in the old format is:

```
select StkNo, SType, StkName, QtyOnHand,
       ReorderPt, OrderSize
from STOCK, STKTYPE
where SType = TType
  and QtyOnHand <= ReorderPt
  and OnOrder = 'N';
```

Revise this SQL statement for the Reorder query so that it works correctly for the data in the new format. You can run this SQL query statement in the Access Query SQL View screen. I recommend that you type the revised SQL statement into a text file, and then copy and paste it into the SQL View screen to run it.

1. Turn in the revised SQL statement for the Reorder query, along with a printout of the query output.
2. Summarize (in words) the changes you made to the SQL statement that performs the query. How does this part of the project illustrate the concept of **data independence**?

Project #4. Structured Query Language: Queries

In this case, you will use SQL to perform queries on a *Time and Billing* application used by the X-Files group in the FBI. The database is implemented in Microsoft Access (or Oracle).

The X-Files group currently has four agents—Walter Skinner (Manager), Dana Scully, John Doggett, and you. The application keeps track of hours spent by the agents on a variety of cases. Each agent fills out timecards each day (8 hrs/day).

The database consists of four tables: DEPT, AGENT, CASES, and TIMECARD. The structure of each table is shown below. Primary key attributes are underlined. Note that the TIMECARD table has a composite primary key.

DEPT (DeptCode, DeptName)
AGENT (AgentID, LastName, FirstName, HireDate, DeptCode, Specialty, Rate)
CASES (CaseNum, CaseTitle, OpenDate, Budget, CloseDate)
TIMECARD (AgentID, CaseNum, WorkDate, Hours)

The following relationships are defined between the tables:

DEPT to AGENT:	DeptCode to DeptCode (1 - M)
AGENT to TIMECARD:	AgentID to AgentID (1 - M).
CASES to TIMECARD:	CaseNum to CaseNum (1 - M).

1. Edit the data in the AGENT table so that Agent FB340 has your last name and first name. Use an SQL UPDATE statement, if necessary.

2. Use the Access Query SQL View (or SQL*Plus) to define and run the following queries. Type the SQL SELECT statements for the queries into an editor. Then copy and paste the statements one at a time into the SQL View (or SQL*Plus) screen to run them.

   a. For WorkDate 07/12/2005, show the WorkDate, AgentID, LastName, CaseNum, and Hours. Order the results by LastName, CaseNum.
   b. For AgentID "FB270" for the days 07/12/2005 through 07/14/2005, show the AgentID, LastName, WorkDate, CaseNum, and Hours. Order the results by WorkDate, CaseNum.
   c. For the days 07/12/2005 and 07/15/2005, show the CaseNum, CaseTitle, WorkDate, AgentID, and Hours for all cases with "Alien" in the title. Order the results by CaseNum, WorkDate, AgentID.
   d. For LastName "Scully" for the week of 07/11/2005 through 07/15/2005, show the LastName, CaseNum, CaseTitle, and sum(Hours) as TotalHours. Order the results by decreasing TotalHours.
   e. For CaseNum 2802 for the week of 07/11/2005 through 07/15/2005, show the CaseNum, AgentID, LastName, sum(Hours) as TotalHours, sum(Charge) as TotalCharge. Order the results by LastName. The formula for the calculated field Charge is [Hours*Rate].
   f. For the week of 07/11/2005 through 07/15/2005, list the AgentID, LastName, and FirstName of all agents who either worked on Case 2803 or worked on a Case with a budget above $50,000. Do not show any duplicates in the output.

g. For the week of 07/11/2005 through 07/15/2005, list the AgentID, LastName, and FirstName of all agents who worked on neither Case 2804 nor Case 2805. Do not show any duplicates in the output.
h. For the week of 07/11/2005 through 07/15/2005, list the AgentID, LastName, and FirstName of all agents who worked at least 8 hours on Case 2804. Do not show any duplicates in the output.

Turn in the SQL SELECT statement for each query, along with the query output.

Project #7. Data Modeling: Relational Model Design and Normalization

Part A

The data requirements for a new database system for Integrity Auto Sales, a used car lot, are described as follows. Integrity purchases all cars at wholesale at various Auto Auctions. If a car hasn't been sold to a customer in 90 days, Integrity sells the car at wholesale at another Auto Auction.

Most cars are sold to customers. Each sale involves one or more salesreps, who are paid a commission. Cars may be sold with or without a warranty. All cars are "detailed" just before they are put on the lot. Some cars require special repairs before they can be sold.

The database system must be able to help manage the used-car inventory, keep track of past customers to encourage repeat sales, evaluate the performance of sales personnel and calculate their commissions, and determine the profitability of the business operations.

A preliminary data model consisting of an *Entity-Relationship Diagram* and an *attribute list* has already been prepared (see AutoRentalERD.doc). You are to continue the development of the database system by completing the following tasks.

1. Evolve the Entity-Relationship model into a **Relational Model Diagram**, showing all tables and their relationships (including cardinality). This model should have no Many-to-Many relationships. Specify the *primary key* for each table, and include *foreign keys* to link tables.

2. List the **functional dependencies** within each table to check that all of your tables are in *Third Normal Form*. Revise the tables as necessary until "all non-key fields depend on the key, the whole key, and nothing but the key."

3. Build a **data dictionary** for the model that lists all of the attributes. For each attribute, include the name, the domain, a description, and the tables that contain the attribute.

Part B

Genealogy Search Services (GSS) is a small business that helps individuals with research on their ancestors. GSS wants you to improve a database system that helps them track customers, orders, and current pricing of search services.

Each customer order lists one or more search services to be performed for a single ancestor. Different ancestors are placed on different orders. Order pricing is based on the current catalog prices (which can change over time). Each order includes a shipping & handling charge. The initial orders stored in the database are shown in a Word document file called GSSdata.doc.

The current (poorly designed) database is in the Microsoft Access database file GSSX.mdb. The database consists of two tables: GCUST and GORDER. The current structure of each table is described below. Primary keys are underlined, and GORDER contains CustCode as a foreign key.

GCUST (CustCode, CLName, CFname, Address, City, State, Zipcode, Email)
GORDER (OrderNo, OrdDate, CustCode, Ancestor, Shipping,
BirCode, BirDescr, BirPrice, BirCDate,
BapCode, BapDescr, BapPrice, BapCDate,
DeaCode, DeaDescr, DeaPrice, DeaCDate,
MarCode, MarDescr, MarPrice, MarCDate)

You are to improve the design of the GSS database by doing the following:

1. List the **functional dependencies** for the GORDER table. What normal form describes the current state of the GORDER table?

2. Restructure the GORDER table so that the resulting tables are in *Third Normal Form* (3NF).

3. Write a **CREATE TABLE** statement for each of your final tables, including primary key and foreign key constraints.

4. Write an SQL **CREATE VIEW** statement that displays the data in the format of the original GORDER table.