

Course: MCTE 630 – Database Systems
Subject: Homework #2
Task: Problems from text:
 The Concepts of Database Management
 Pratt & Adamski (1997)
Professor: Dr. Maxine Cohen
Student: Leanne C. Boyd
Usercode: boydl
Email: boydl@scis.acast.nova.edu
Due date: week of June 1, 1998

HOMEWORK #2

CONCEPTS OF DATABASE MANAGEMENT:

Chapter 4 # 2 (don't execute), 4, 6, 10, 12

2) **Define a view called SMLCUST. It consists of the customer number, name, address, balance, and credit limit for all customers whose credit limit is \$1,000 or less.**

a) **Write the view definition for SMLCUST.**

```

SQL> CREATE VIEW SMLCUST AS
  2 SELECT BOYD_CUST.CUSTOMER_NUMBER,
  3 LAST, FIRST, STREET,
  4 BALANCE, CREDIT_LIMIT
  5 FROM BOYD_CUST
  6 WHERE CREDIT_LIMIT <= '1000';
  
```

View created.

b) **Write an SQL query to retrieve the number and name of all customers in SMLCUST whose balance is over their credit limit.**

```

SQL> select customer_number,
  2 last, first
  3 from SMLCUST
  4 where balance > credit_limit;
  
```

```

CUS LAST          FIRST
--- -----
315 Daniels      Tom
622 Martin       Dan
  
```

c) **Convert the query from (b) to the query that actually will be executed.**

```
SQL> select boyd_cust.customer_number,
2 last, first
3 from boyd_cust
4 where balance > credit_limit;
```

CUS	LAST	FIRST
315	Daniels	Tom
622	Martin	Dan

4) **What are the advantages of using indexes? The disadvantages?**

An index, just as with a good index in a book, will help the user rapidly access data. In fact, within the relational model systems on both microcomputers and on mainframes, the primary mechanism for increasing the efficiency of data retrieval . . . is the use of indexes. Although an index greatly increases the efficiency, there ARE other ways of retrieving the data, so therefore one of the disadvantages of the index is that it DOES take up space that might be used for something else . . . and therefore TECHNICALLY isn't necessary. In this writer's opinion, the advantage outweighs the "space problem." The other disadvantage of the index is that when other data is added to the database, the index must be updated. If you don't have an index, you don't have to spend extra time in updating it whenever you add to the database itself.

6) **Describe the GRANT mechanism and explain how it relates to security. What types of privileges may be granted? How are they revoked?**

The GRANT mechanism is one of two security mechanisms in SQL systems – the other being VIEWS. The GRANT facility, however, is the main form of security. The basic theory is that users may be granted certain privileges in using the database, which can later be revoked, if necessary. These privileges would have to do with actions such as inserting rows, accessing certain tables or columns, the right to update or delete, etc. These privileges are accomplished through GRANT statements, such as GRANT SELECT ON PART TO BOYD. Conversely, the privilege may be eliminated by use of a REVOKE statement, such as REVOKE SELECT ON PART TO BOYD. The access rules are enforced by whatever security mechanism the DBMS provides. The privileges can range from very restrictive, to quite wide, as in granting to the PUBLIC. The actual privileges are: SELECT, UPDATE, DELETE, INSERT, INDEX, and ALTER, and are usually assigned by the database administrator.

10) **State the two integrity rules. Indicate the reasons for enforcing each rule.**

ENTITY INTEGRITY: This is the integrity rule that no column that participates in the primary key may accept null values.

The primary key uniquely identifies a certain, given row. If a NULL designation were to be allowed, this would no longer happen. In many instances, you would not be able to

ascertain the information, especially when it comes to customer ID numbers, student numbers, or any identifying factors that are unique to the person. If the value were NULL, it would cause much confusion and render the data useless. When it is not allowed for a primary key to accept NULL values, it is ensured that one record may be distinguished from another.

REFERENTIAL INTEGRITY: This is the integrity rule that if one table (Table A) contains a foreign key [a column or collection of columns in one table whose value is required to match the value of the primary key for some row in another table] that matches the primary key of a second table (Table B) – then, values of the foreign key must either match the value of the primary key for some row in Table B – or it must be NULL.

When working with relational models, relationships of the data are not always obvious – this is the first problem encountered with the entity integrity only. It is almost impossible to “see” the relationships of the data. A second problem is that there is nothing about the model that would prevent a user from storing information that doesn’t correspond with columns within the database. By the use of the foreign key, you are able to control that it matches a primary key in another table or that it is null – this makes the relationships between the two, very explicit. It defines the relationship. In fact, our book states that the principal failure in many systems today is that they lack support for referential integrity, which renders them incapable of being fully relational.

12) How can the structure of a table be changed in SQL? What general types of changes are possible? Which commands are used to implement these changes?

The way to change the table structure in SQL is through the ALTER TABLE command. This allows new columns to be added to the existing table. Some systems allow you to specify an initial value for these columns. Some systems also allow for columns to be deleted, using, again, the ALTER TABLE command, and then using DELETE [column-name]. Finally, some systems allow you to change the data type of a certain column, such as increasing the number of characters in the column. The command DROP might be considered a change . . . a permanent change! The table is erased!

CONCEPTS OF DATABASE MANAGEMENT:

Chapter 5 #2, 4, 8, 9, 12

2) Give an example of an attribute, A, and another attribute, B, such that B is functionally dependent on A. Give an example of an attribute, C, and an attribute, D, such that D is not functionally dependent on C.

- a) Example: For a local neighborhood watch group, they keep a table that lists a membership number, last name, first name, street, city, state, zip code, and membership dues paid, for every year. The membership number, which we will call Attribute A (the MEMB_NUMBER column), for a certain person, is 150. Attribute B (the LAST_NAME

column) is functionally dependent on Attribute A (or, A functionally determines B) because B (the last name) depends on A (the membership number). Any time the membership number column is given a certain value, in this case the #150, then you know you will find a SINGLE last name (a single value) in the LAST_NAME column. In this example, let's say the member is Mary Cochran. So, LAST_NAME of Cochran is functionally dependent on MEMB_NUMB of #150. Attribute A, furthermore, is the primary key for this example, as all other attributes are functionally dependent on it. The membership number MUST be tied to each of the other attributes.

WATCHGROUP (MEMB_NUMBER, LAST, FIRST, STREET, CITY, STATE, ZIP_CODE, DUES_PD, YEAR_PAID)

If this same member lives in Rhode Island, the ZIP_CODE column could be called Attribute C, and the DUES_PD column, Attribute D. Mary's dues paid for 1997 were \$585.00. This is NOT dependent on Mary's zip code of 09882. Therefore, Attribute D is not functionally dependent on Attribute C.

4) Define candidate key.

Sometimes there are more than one attributes that would qualify to be the primary key. For example, if the table mentioned in #2 also had the member's Driver's License number, this is a uniquely identifiable number. This would qualify it to also be a primary key. In this case, the organization designates the primary key, and the other attribute is designated as a CANDIDATE KEY. Actually, the primary key is then chosen from among two or more candidate keys, each of which would be aptly suitable for being primary key.

WATCHGROUP (MEMB_NUMBER, LAST, FIRST, STREET, CITY, STATE, ZIP_CODE, DUES_PD, YEAR_PAID, DL_NUMBER) ←The driver's license number could actually serve as a primary key, but membership number is chosen by the organization.

8) Consider a student table containing a student's number, student's name, student's major department, student advisor's number, student advisor's name, student advisor's office number, student advisor's phone number, student's number of credits, and student's class standing (freshman, sophomore, and so on). List the functional dependencies that exist, along with the assumptions that would support these dependencies.

- a) STUDENT_NUMBER determines student's name, student's major department, student advisor's number, student advisor's name, student advisor's office number, student advisor's phone number, student's number of credits, and student's class standing. This table might look like this:

STUDENT (STUDENT_NUMBER, STUDENT_NAME, MAJOR_DEPT, ADVISOR_NAME, ADV_OFFICE_NMBR, ADVISOR_PHONE, NUMBER_CREDITS, CLASS_STANDING)

Functional Dependencies:

STUDENT_NUMBER--> STUDENT_NAME, STUDENT_CREDITS, CLASS_STANDING,
STUDENT_ADVISOR, MAJOR_DEPT
STUDENT_ADVISOR-->ADVISOR_NAME, ADV_OFFICE_NMBR, ADVISOR_PHONE

- 9) Using the types of entities found in the Henry's Books database system (books, authors, and publisher), create an example of a table that is in 1NF but not in 2NF and an example of a table that is in 2NF but not in 3NF. In each case, justify the answers and show how to convert to the higher forms.

Note: I chose to create a system a bit different than the bookstores of Henry's Books. Staying in the realm of publishing, however, I created this scenario with poster sales:

- a) Example of table in 1NF, but not in 2NF:

Scenario setup:

```
SQL> --A graphics company produces unique posters for sale.
SQL> --The posters are printed on various fabrics on a high resolution
SQL> -- ink-jet printer, that has the ability to adjust the colors
SQL> -- as the printing process progresses. Between the texture of the
SQL> -- fabrics and the adjustment of color, each poster may be, and
SQL> -- usually is, truly one-of-a-kind. If special-ordered, two or
SQL> -- more almost-identical (for all intents and purposes) posters
SQL> -- may be created for a single customer.
SQL> --The posters are created in series, based on the creation date,
SQL> -- the color settings, and the fabric designation.
SQL> --These are the company's table's column names:
SQL> -- 1. ORDERNUMBER
SQL> -- 2. ORDERDATE
SQL> -- 3. SERIES_NUMBER
SQL> -- 4. NUMBER_ORDERED
SQL> -- 5. PRICE_QUOTED
SQL> -- 6. BUYER_FIRST
SQL> -- 7. BUYER_LAST
SQL> -- 8. ARTIST_PENNAME
```

For DigitalPosters, I created this table:

```
SQL> CREATE TABLE DIGITALPOSTERS
  2 (ORDERNUMBER CHAR(5) PRIMARY KEY,
  3 ORDERDATE DATE,
  4 SERIES_NUMBER CHAR(14),
  5 NUMBER_ORDERED CHAR(3),
  6 PRICE_QUOTED DECIMAL(7,2),
  7 BUYER_FIRST CHAR(15),
  8 BUYER_LAST CHAR(15),
  9 ARTIST_PENNAME CHAR(15) );
```

Table created.

I populated the table:

```
SQL> INSERT INTO POSTER_ORDERS
  2 VALUES
  3 SQL> INSERT INTO DIGITALPOSTERS
  2 VALUES
  3 ('10150', '19-JAN-98', '010598FF9933LN', 3, 69.95, 'Don', 'Juan', 'Kukla');

  1 row created.
```

```
SQL> INSERT INTO DIGITALPOSTERS
  2 VALUES
  3 (10151, '20-JAN-98', '010598FF36BBCO', 1, 69.95, 'Rhoe', 'Mich', 'Fran');

1 row created.

SQL> INSERT INTO DIGITALPOSTERS
  2 VALUES
  3 (10224, '14-FEB-98', '02019866FFBBRA', 1, 69.95, 'Cera', 'Noh', 'Ollie');

1 row created.

SQL> INSERT INTO DIGITALPOSTERS
  2 VALUES
  3 (10299, '18-FEB-98', '111897F9B633WL', 200, 39.95, 'Skye', 'Walker', 'Keystone');

1 row created.
```

Select * from DIGITALPOSTERS:

```
SQL> SELECT * FROM DIGITALPOSTERS;
```

ORDER	ORDERDATE	SERIES_NUMBER	NUM	PRICE_QUOTED	BUYER_FIRST	BUYER_LAST	ARTIST_PENNAME
10150	19-JAN-98	010598FF9933LN	3	69.95	Don	Juan	Kukla
10151	20-JAN-98	010598FF36BBCO	1	69.95	Rhoe	Mich	Fran
10224	14-FEB-98	02019866FFBBRA	1	69.95	Cera	Noh	Ollie
10299	18-FEB-98	111897F9B633WL	200	39.95	Skye	Walker	Keystone

This is the table:

DIGITALPOSTERS (ORDERNUMBER, ORDERDATE, SERIES_NUMBER, NUMBER_ORDERED, PRICE_QUOTED, BUYER_FIRST, BUYER_LAST, ARTIST_PENNAME)

This, then is in the 1NF form, but not 2NF:

DIGITALPOSTERS (ORDERNUMBER, ORDERDATE, BUYER_FIRST, BUYER_LAST, NUMBER_ORDERED, (SERIES_NUMBER, PRICE_QUOTED, ARTIST_PENNAME))

In 1NF form, any chance for repeating groups is removed

Justification and means to convert to higher forms:

Any tables not in 3NF form are prone to problems, such as update anomalies, so:

The primary key is a combination of Order Number and Series Number, which distinguishes one occurrence of the repeating group from another in any given row in the table. In order to convert to a higher form, for example 2NF, then its primary key must contain only a single attribute, which is the sign of automatically being in 2NF. In our 1NF form, above, the combination primary key (concatenation of Order Number and Series Number)) determines all the other attributes. There are “partial dependencies” which means there are dependencies on other than just one primary key. Therefore, you must take each subset as a set of attributes that make up the primary key, and begin a table with the subset as ITS primary key:

(ORDERNUMBER,
 (SERIES_NUMBER,
 (ORDERNUMBER, SERIES_NUMBER,

Then, place remaining attributes with the least collection on which they depend, and give the new tables appropriate names:

POSTER_ORDERS (ORDERNUMBER, SERIES_NUMBER, ORDERDATE,
NUMBER_ORDERED, PRICE_QUOTED)

DIGITALPOSTERS (SERIES_NUMBER, ARTIST_PENNAME)

ART_BUYER (ORDERNUMBER, BUYER_FIRST, BUYER_LAST)

(ORDERNUMBER

(SERIES_NUMBER

(ORDERNUMBER, SERIES_NUMBER)

Each table can now be given a descriptive name that accurately describes the table, such as Poster Orders, Series Number, etc. These codings would look like this:

(ORDERNUMBER, ORDERDATE)

(SERIES_NUMBER, POSTERTITLE)

(ORDERNUMBER, SERIES_NUMBER, NUMBER_ORDERED, PRICE_QUOTED)

Example of table in 2NF, but not in 3NF:

The example just shown above takes our table into the 2NF form, whereby the subsets of the set of attributes making up the primary key, allow us to develop tables that are based on placing each one with the most minimal collection of attributes on which it depends. In this way, the tables contain only the information needed, with no confusion of overlap, deletion, inconsistent data, no trouble in updating and no problems with addition of data. There are no design repetitions. Because data occurs in one place, you cannot accidentally add a multiple description, and you don't have to add a poster order to create a new row.

Justification and means to convert to higher forms:

Any primary or candidate key will be what is called a determinant, as it determines or defines another attribute. To take a form 2NF to a higher form, 3NF, the table must already be solidly a 2NF, and its only determinants MUST be only candidate keys. You must remove from this table, any determinants that are not candidate keys, and form a new table. In the table I created, for instance, if the columns Poster Title or Fabric Factory had been in the initial table, these would have to be removed, although Poster Title could qualify as a candidate key. BUT, Fabric Factory does not qualify. So it must be removed to form another table.

- 12) **Using your knowledge of a college environment, determine the functional dependencies that exist in the following table. After these have been determined, convert this table to an equivalent collection of tables that are in 3NF.**

STUDENT (STUDENT_NUMBER, STUDENT_NAME, NUMBER_CREDITS,
ADVISOR_NUMBER, ADVISOR_NAME, DEPTNUMB, DEPTNAME,
(COURSE_NUMBER, COURSE_DESCRIPTION, COURSE_TERM,
GRADE))

DETERMINE FUNCTIONAL DEPENDENCIES:

STUDENT_NUMBER--> STUDENT_NAME, NUMBER_CREDITS, ADVISOR_NUMBER,
ADVISOR_NAME, DEPTNUMB, DEPTNAME

ADVISOR_NUMBER--> ADVISOR_NAME, DEPTNUMB, DEPTNAME

STUDENT_NUMBER, COURSE_NUMBER--> COURSE_DESCRIPTION, COURSE_TERM, GRADE

SO THAT:

```
STUDENT (STUDENT_NUMBER, STUDENT_NAME, NUMBER_CREDITS, ADVISOR_NUMBER,
         ADVISOR_NAME, DEPTNUMB, DEPTNAME, COURSE_NUMBER,
         COURSE_DESCRIPTION, COURSE_TERM, GRADE)
```

So the table is now in 1NF, but not in 2NF as Student Name is only dependent on Student Number - only a portion of the primary key.

CONVERT TABLE TO 2NF BY STARTING A TABLE WITH EACH SUBSET AS ITS KEY:

```
(STUDENT_NUMBER,
 COURSE_NUMBER,
 STUDENT_NUMBER, COURSE_NUMBER)
```

With the least collection needed, put in the remaining attributes:

```
(STUDENT_NUMBER, STUDENT_NAME, NUMBER_CREDITS, ADVISOR_NUMBER,
 ADVISOR_NAME, DEPTNUMB, DEPTNAME)
(COURSE_NUMBER, COURSE_DESCRIPTION)
(STUDENT_NUMBER, COURSE_NUMBER, COURSE_TERM, GRADE)
```

Assign names to the new tables:

```
STUDENT (STUDENT_NUMBER, STUDENT_NAME, NUMBER_CREDITS,
         ADVISOR_NUMBER, ADVISOR_NAME, DEPTNUMB, DEPTNAME)
COURSE (COURSE_NUMBER, COURSE_DESCRIPTION)
GRADE (STUDENT_NUMBER, COURSE_NUMBER, COURSE_TERM, GRADE)
```

These tables are all in 2NF; but COURSE and GRADE are also in 3NF! STUDENT, however, is not, as it still contains a determinant - Advisor Number - that is not a candidate key. STUDENT still needs to be converted to 3NF.

Remove the attributes that depend on the determinant Advisor Number and put in a separate table.

```
(STUDENT_NUMBER, STUDENT_NAME, NUMBER_CREDITS,
 ADVISOR_NUMBER)
(ADVISOR_NUMBER, ADVISOR_NAME, DEPTNUMB, DEPTNAME)
```

Name these two tables, then put all the tables together:

```
STUDENT (STUDENT_NUMBER, STUDENT_NAME, NUMBER_CREDITS,
         ADVISOR_NUMBER)
ADVISOR (ADVISOR_NUMBER, ADVISOR_NAME, DEPTNUMB, DEPTNAME)
COURSE (COURSE_NUMBER, COURSE_DESCRIPTION)
GRADE (STUDENT_NUMBER, COURSE_NUMBER, COURSE_TERM, GRADE)
```

CONCEPTS OF DATABASE MANAGEMENT:

Chapter 6 #8, 10

8) A database at a college is required to support the following requirements:

a) For a department, store its number and name.

DEPARTMENT (DEPT_NUMBER, DEPT_NAME)

b) For an advisor, store his or her number and name and the number of the department to which he or she is assigned.

**ADVISOR (ADVISOR_NUMBER, DEPT_NUMBER)
DEPARTMENT (DEPT_NUMBER, DEPT_NAME)**

c) For a course, store its code and description (i.e., MTH110, ALGEBRA).

COURSE (COURSE_CODE, COURSE_DESCRIPTION)

d) For a student, store his or her number and name. For each course the student has taken, store the course code, the course description, and the grade received. Also, store the number and name of the student's advisor. Assume that an advisor may advise any number of students but that each student has just one advisor.

<p><u>STUDENT</u> STUDENT_NUMBER STUDENT_LAST STUDENT_FIRST</p>	<p><u>COURSE</u> COURSE_CODE COURSE_DESCRIPTION GRADE</p>	<p><u>ADVISOR</u> ADVISOR_NUMBER ADVISOR_LAST ADVISOR_FIRST STUDENT_NUMBER</p>
--	--	---

Complete the information-level design for this set of requirements. Use your own experience to determine any constraints you need that are not stated in the problem. Represent the answer in DBDL.

```

STUDENT (STUDENT_NUMBER, STUDENT_LAST, STUDENT_FIRST, ADVISOR_NUMBER)
ADVISOR (ADVISOR_NUMBER, ADVISOR_NAME)
COURSE (COURSE_NUMBER, COURSE_DESCRIPTION)
GRADE (STUDENT_NUMBER, COURSE_NUMBER, COURSE_CODE, GRADE)

AK SOC_SEC_NUMBER
SK STUDENT_LAST
FK STUDENT_NUMBER --> GRADE
FK COURSE_NUMBER --> GRADE
    
```

10) Suppose in addition to the requirements specified in Review Question 8, you must store the number of the department in which the student is majoring. Indicate the changes this would cause in the design in these two situations:

a) The student must be assigned an advisor who is in the department in which the student is majoring.

```
STUDENT (STUDENT_NUMBER, STUDENT_LAST, STUDENT_FIRST, DEPT_NUMB,
         STUDENT_MAJOR, ADVISOR_NUMBER)
ADVISOR (ADVISOR_NUMBER, ADVISOR_NAME, DEPT_NUMB, STUDENT_MAJOR)
COURSE (COURSE_NUMBER, COURSE_DESCRIPTION)
GRADE (STUDENT_NUMBER, COURSE_NUMBER, COURSE_CODE, GRADE)
```

```
AK SOC_SEC_NUMBER
SK STUDENT_LAST
FK STUDENT_MAJOR --> ADVISOR
FK STUDENT_NUMBER --> GRADE
FK COURSE_NUMBER --> GRADE
```

b) The student's advisor does not necessarily have to be in the department in which the student is majoring.

```
STUDENT (STUDENT_NUMBER, STUDENT_LAST, STUDENT_FIRST, DEPT_NUMB*,
         STUDENT_MAJOR*, ADVISOR_NUMBER)
ADVISOR (ADVISOR_NUMBER, ADVISOR_NAME, DEPT_NUMB)
COURSE (COURSE_NUMBER, COURSE_DESCRIPTION)
GRADE (STUDENT_NUMBER, COURSE_NUMBER, COURSE_CODE, GRADE)
```

```
AK SOC_SEC_NUMBER
SK STUDENT_LAST
FK STUDENT_NUMBER --> GRADE
FK COURSE_NUMBER --> GRADE
```