**Chapter 4 Problems – Entity Relationship (ER) Modeling – 5 Problems**

Do Review Questions 8-12 on page 150 of our textbook.

1. **Discuss the difference between a composite key and a composite attribute. How would each be indicated in an ERD?**

A composite key is one that consists of more than one attribute. If the ER diagram contains the attribute names for each of its entities, a composite key is indicated in the ER diagram by the fact that more than one attribute name is underlined to indicate its participation in the primary key.

A composite attribute is one that can be subdivided to yield *meaningful* attributes for each of its components. For example, the composite attribute CUS\_NAME can be subdivided to yield the CUS\_FNAME, CUS\_INITIAL, and CUS\_LNAME attributes. There is no ER convention that enables us to indicate that an attribute is a composite attribute.

1. **What two courses of action are available to a designer when encountering a multivalued attribute?**

The discussion that accompanies the answer to question 3 is valid as an answer to this question.

1. **What is a derived attribute? Give an example.**

A derived attribute is an attribute whose value is calculated (derived) from other attributes. The derived attribute need not be physically stored within the database; instead, it can be derived by using an algorithm. For example, an employee’s age, EMP\_AGE, may be found by computing the integer value of the difference between the current date and the EMP\_DOB. If you use MS Access, you would use INT((DATE() – EMP\_DOB)/365).

Similarly, a sales clerk's total gross pay may be computed by adding a computed sales commission to base pay. For instance, if the sales clerk's commission is 1%, the gross pay may be computed by

EMP\_GROSSPAY = INV\_SALES\*1.01 + EMP\_BASEPAY

Or the invoice line item amount may be calculated by

LINE\_TOTAL = LINE\_UNITS\*PROD\_PRICE

1. **How is a relationship between entities indicated in an ERD? Give an example, using the Crow’s Foot notation.**

Use Figure Q4.7 as the basis for your answer. Note the distinction between the dashed and solid relationship lines, then tie this distinction to the answers to question 7c and 7d.

1. **Discuss two ways in which the 1:M relationship between COURSE and CLASS can be implemented. (*Hint*: Think about relationship strength.)**

Note the discussion about weak and strong entities in questions 7c and 7d. Then follow up with this discussion:

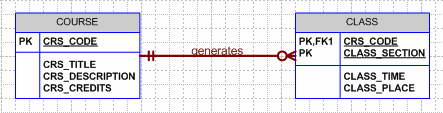
The relationship is implemented as *strong* when the CLASS entity’s PK contains the COURSE entity’s PK. For example,

COURSE(**CRS\_CODE**, CRS\_TITLE, CRS\_DESCRIPTION, CRS\_CREDITS)

CLASS(**CRS\_CODE**, **CLASS\_SECTION**, CLASS\_TIME, CLASS\_PLACE)

Note that the CLASS entity’s PK is CRS\_CODE + CLASS\_SECTION – and that the CRS\_CODE component of this PK has been “borrowed” from the COURSE entity. (Because CLASS is existence-dependent on COURSE and uses a PK component from its parent (COURSE) entity, the CLASS entity is weak in this strong relationship between COURSE and CLASS. The Visio Crow’s Foot ERD shows a strong relationship as a solid line. (See Figure Q4.12a.) Visio refers to a strong relationship as an *identifying* relationship.

**Figure Q4.12a Strong COURSE and CLASS Relationship**



Sample data are shown next:

**Table name: COURSE**

|  |  |  |  |
| --- | --- | --- | --- |
| **CRS\_CODE** | **CRS\_TITLE** | **CRS-DESCRIPTION** | **CRS\_CREDITS** |
| ACCT-211 | Basic Accounting | An introduction to accounting. Required of all business majors. | 3 |
| CIS-380 | Database Techniques I | Database design and implementation issues. Uses CASE tools to generate designs that are then implemented in a major database management system. | 3 |
| CIS-490 | Database Techniques II | The second half of CIS-380. Basic Web database application development and management issues. | 4 |

**Table name: CLASS**

|  |  |  |  |
| --- | --- | --- | --- |
| **CRS\_CODE** | **CLASS\_SECTION** | **CLASS\_TIME** | **CLASS\_PLACE** |
| ACCT-211 | 1 | 8:00 a.m. – 9:30 a.m. T-Th. | Business 325 |
| ACCT-211 | 2 | 8:00 a.m. – 8:50 a.m. MWF | Business 325 |
| ACCT-211 | 3 | 8:00 a.m. – 8:50 a.m. MWF | Business 402 |
| CIS-380 | 1 | 11:00 a.m. – 11:50 a.m. MWF | Business 415 |
| CIS-380 | 2 | 3:00 p.m. – 3:50 a.m. MWF | Business 398 |
| CIS-490 | 1 | 1:00 p.m. – 3:00 p.m. MW | Business 398 |
| CIS-490 | 2 | 6:00 p.m. – 10:00 p.m. Th. | Business 398 |

The relationship is implemented as *weak* when the CLASS entity’s PK does not contain the COURSE entity’s PK. For example,

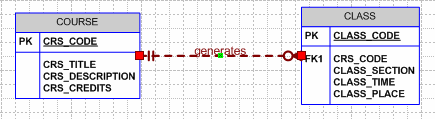
COURSE(**CRS\_CODE**, CRS\_TITLE, CRS\_DESCRIPTION, CRS\_CREDITS)

CLASS(**CLASS\_CODE**, CRS\_CODE, CLASS\_SECTION, CLASS\_TIME, CLASS\_PLACE)

(Note that CRS\_CODE is no longer part of the CLASS PK, but that it continues to serve as the FK to COURSE.)

The Visio Crow’s Foot ERD shows a weak relationship as a dashed line. (See Figure Q4.12b.) Visio refers to a weak relationship as a *non-identifying* relationship.

**Figure Q4.12b Weak COURSE and CLASS Relationship**



Given the weak relationship depicted in Figure Q4.13b, the CLASS table contents would look like this:

**Table name: CLASS**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **CLASS\_CODE** | **CRS\_CODE** | **CLASS\_SECTION** | **CLASS\_TIME** | **CLASS\_PLACE** |
| 21151 | ACCT-211 | 1 | 8:00 a.m. – 9:30 a.m. T-Th. | Business 325 |
| 21152 | ACCT-211 | 2 | 8:00 a.m. – 8:50 a.m. MWF | Business 325 |
| 21153 | ACCT-211 | 3 | 8:00 a.m. – 8:50 a.m. MWF | Business 402 |
| 38041 | CIS-380 | 1 | 11:00 a.m. – 11:50 a.m. MWF | Business 415 |
| 38042 | CIS-380 | 2 | 3:00 p.m. – 3:50 a.m. MWF | Business 398 |
| 49041 | CIS-490 | 1 | 1:00 p.m. – 3:00 p.m. MW | Business 398 |
| 49042 | CIS-490 | 2 | 6:00 p.m. – 10:00 p.m. Th. | Business 398 |

The advantage of the second CLASS entity version is that its PK can be referenced easily as a FK in another related entity such as ENROLL. Using a single-attribute PK makes implementation easier. This is especially true when the entity represents the “1” side in one *or more* relationships. **In general, it is advisable to avoid composite PKs whenever it is practical to do so**.

**Chapter 6 Problems – Normalization of Database Tables – 5 Problems**

Do Review Questions 6-8 on page 223 and Problems 1-2 on pages 224 of our textbook.

1. **Given the dependency diagram shown in Figure Q6.6, answer items 6a-6c:**

**FIGURE Q5.6 Dependency Diagram for Question 6**



* 1. **Identify and discuss each of the indicated dependencies.**

C1 **→** C2 represents a *partial dependency*, because C2 depends only on C1, rather than on the entire primary key composed of C1 and C3.

C4 **→** C5 represents a *transitive dependency*, because C5 depends on an attribute (C4) that is not part of a primary key.

C1, C3 **→** C2,C4, C5 represents a set of proper functional dependencies, because C2, C4, and C5 depend on the primary key composed of C1 and C3.

* 1. **Create a database whose tables are at least in 2NF, showing the dependency diagrams for each table.**

The normalization results are shown in Figure Q6.6b.

**Figure Q6.6b The Dependency Diagram for Question 6b**



* 1. **Create a database whose tables are at least in 3NF, showing the dependency diagrams for each table.**

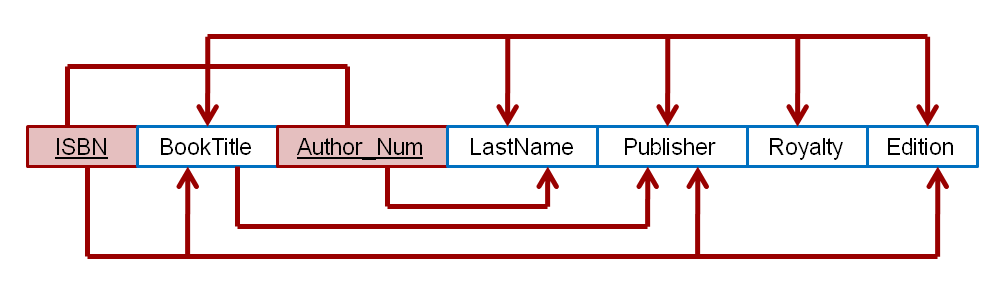
The normalization results are shown in Figure Q6.6c.

**Figure Q6.6c The Dependency Diagram for Question 6c**



1. **The dependency diagram in Figure Q6.7 indicates that authors are paid royalties for each book that they write for a publisher. The amount of the royalty can vary by author, by book, and by edition of the book.**

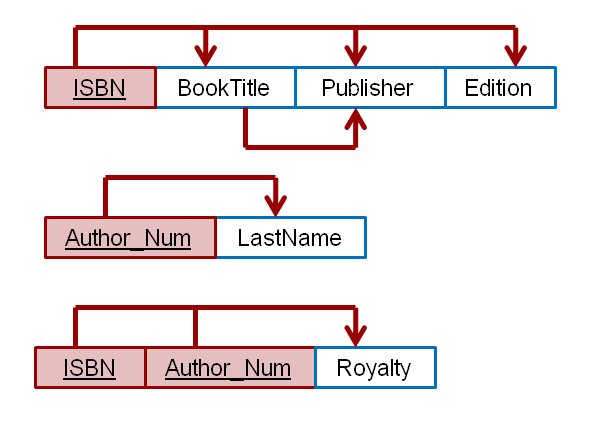
**Figure Q6.7 Book royalty dependency diagram**

****

**a. Based on the dependency diagram, create a database whose tables are at least in 2NF, showing the dependency diagram for each table.**

The normalization results are shown in Figure Q6.7a.

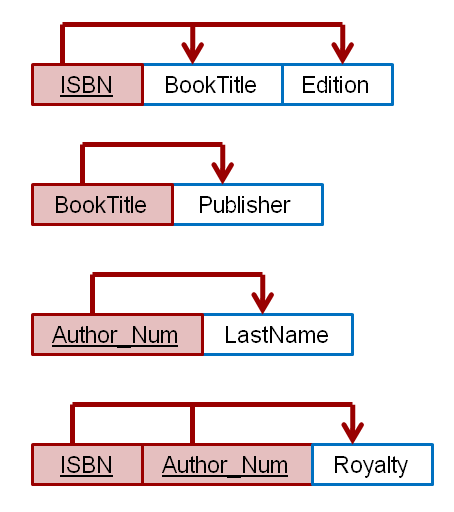
**Figure Q6.7a The 2NF normalization results for Question 7a.**

****

**b. Create a database whose tables are at least in 3NF, showing the dependency diagram for each table.**

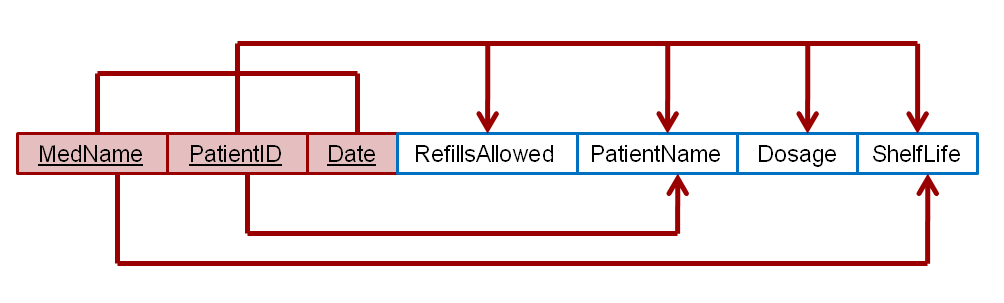
The normalization results are shown in Figure Q6.7a.

**Figure Q6.7b The 3NF normalization results for Question 7b.**

****

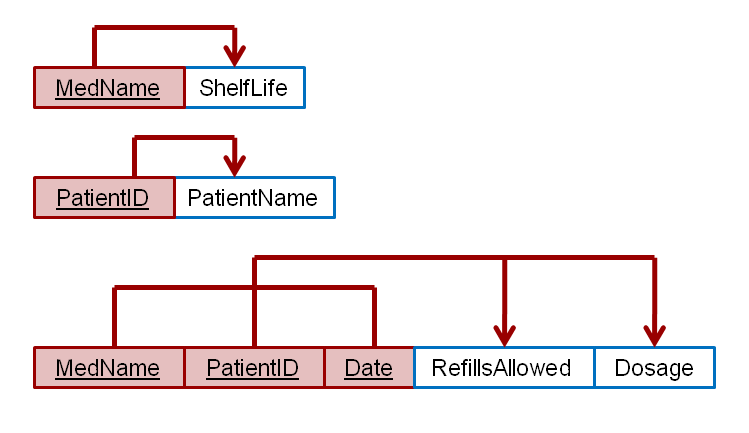
1. **The dependency diagram in Figure Q6.8 indicates that a patient can receive many prescriptions for one or more medicines over time. Based on the dependency diagram, create a database whose tables are in at least 2NF, showing the dependency diagram for each table.**

**Figure Q6.8 Prescription dependency diagram**

****

The normalization results are shown in Figure Q6.8a.

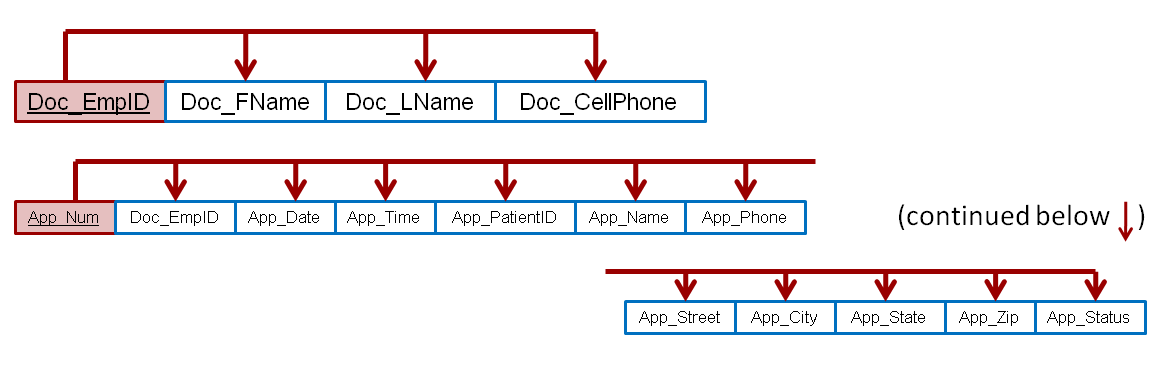
**Figure Q6.8a The 2NF normalization results for Question 8.**

****

* + 1. **Using the descriptions of the attributes given in the figure, convert the ERD shown in Figure P6.1 into a dependency diagram that is in at least 3NF.**

An initial dependency diagram depicting only the primary key dependencies is shown in Figure P6.1a below.

**Figure P6.1a Initial dependency diagram for Problem 1.**

****

There are no composite keys being used, therefore, by definition, there is not an issue with partial dependencies and the entities are already in 2NF. Based on the descriptions of the attributes, it appears that the patient name, phone number, and address can be determined by the patient id number. Therefore, the following transitive dependency can be determined.

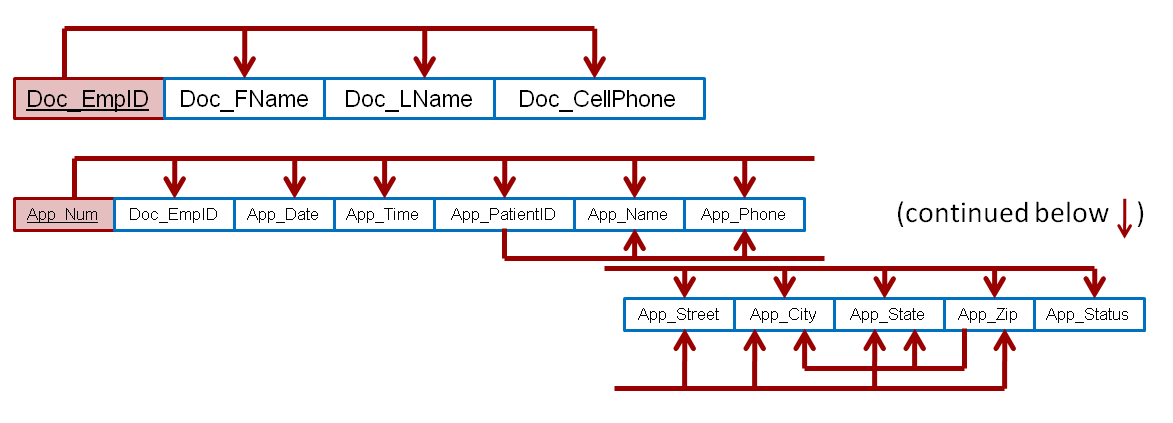
App\_PatientID 🡪 (App\_Name, App\_Phone, App\_Street, App\_City, App\_State, App\_Zip)

As discussed in the chapter, ZIP\_Codes can be used to determine a city and state; therefore, we also have the transitive dependency:

App\_Zip 🡪 App\_City, App\_State

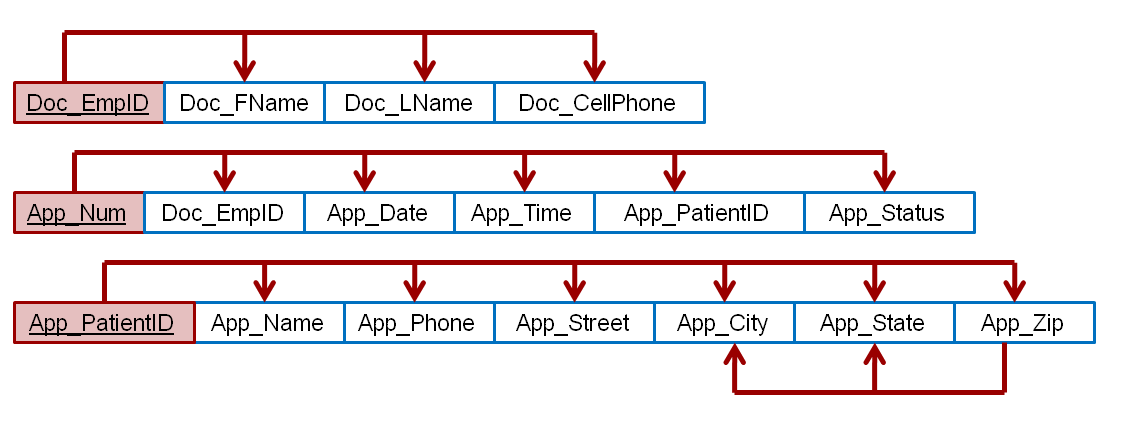
Figure P6.1b depicts the dependency diagram with these transitive dependencies included.

**Figure P6.1b Revised dependency diagram for Problem 1.**



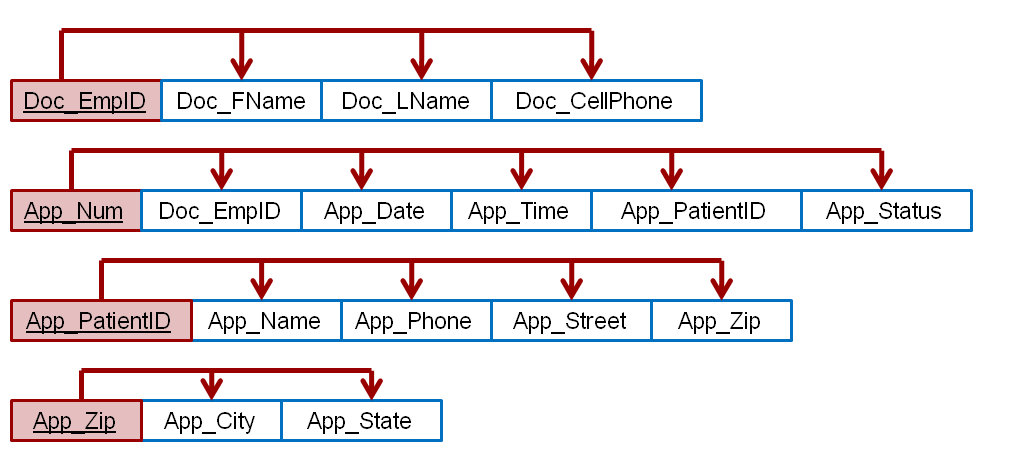
Since the first transitive dependency completely encloses the second transitive dependency, it is appropriate to resolve the first transitive dependency before resolving the second. Figure P6.1c shows the results of resolving the first transitive dependency.

**Figure P6.1c Resolving the first transitive dependency**



Finally, the second and final transitive dependency can now be resolved as shown in the final dependency diagram in Figure P6.1d.

**Figure P6.1d Final dependency diagram for Problem 1**



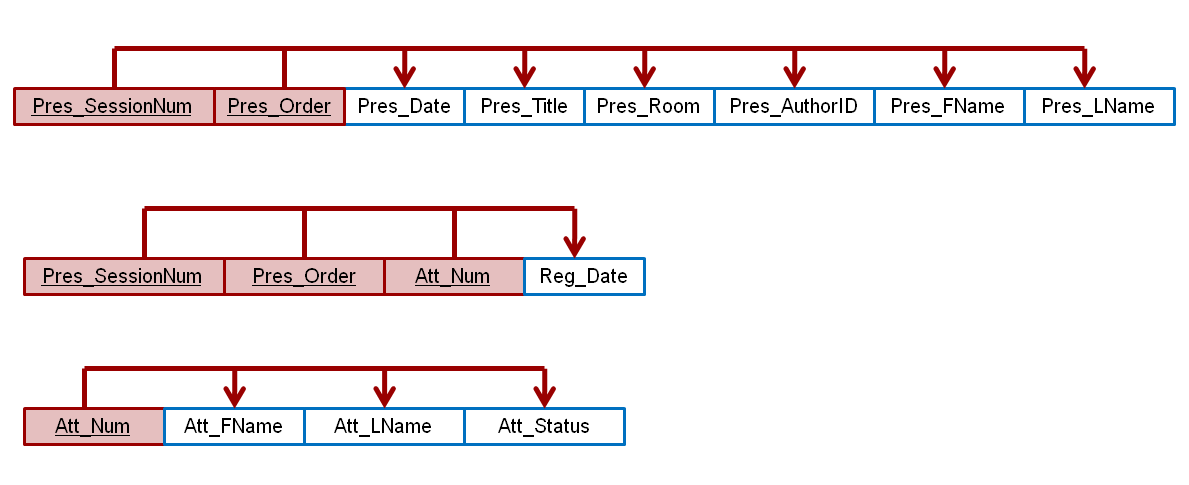
Note that at this time we have resolved all of the transitive dependencies. Decisions on whether or not to denormalize, and perhaps not remove the final transitive dependency, have yet to be made. Also, the structures have not yet had the benefit of additional design modifications such as achieving proper naming conventions for the attributes in the new tables. However, creating the fully normalized structures is an important set toward making informed decisions about the compromises in the design that we may choose to make.

NOTE: Please note that we are making the assumption that a zip code only determines one city and state. Unfortunately, this is not true, there are a handful of zip codes that traverse states. In these cases, it would be appropriate not to use the [App\_zip, App\_City, App\_State] relation and instead add these attributes to the previous relation. Hence, the relation would be: [App\_PatiendID, App\_Name, App\_Phone, App\_Street, App\_City, App\_Zip, App\_State]

* + 1. **Using the descriptions of the attributes given in the figure, convert the ERD shown in Figure P6.2 into a dependency diagram that is in at least 3NF.**

An initial dependency diagram depicting only the primary key dependencies is shown in Figure P6.2a below.

**Figure P6.2a Initial dependency diagram for Problem 2.**

****

Based on the descriptions of the attributes given, the following partial dependency can be determined:

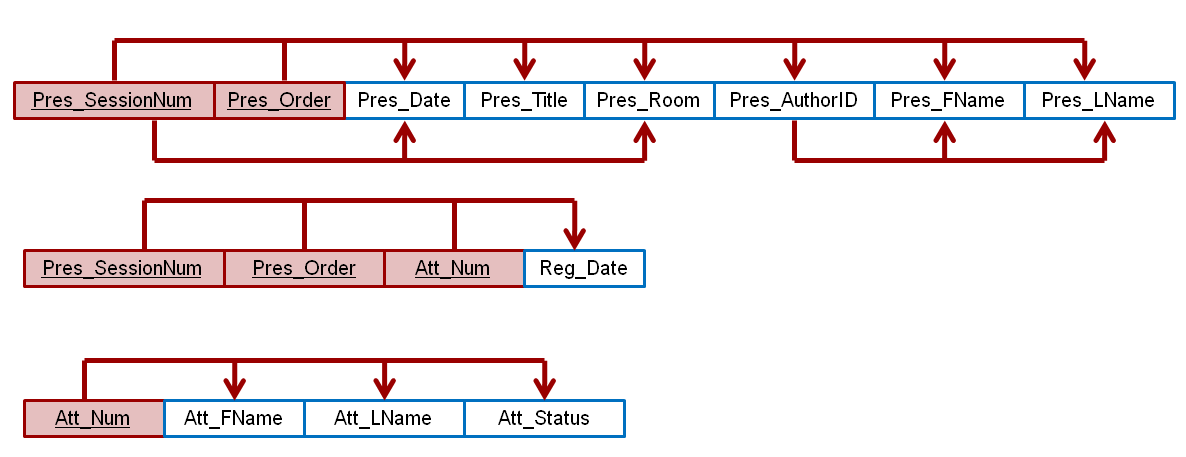
Pres\_SessionNum 🡪 (Pres\_Date, Pres\_Room)

Also, the following transitive dependencies can be determined:

Pres\_AuthorID 🡪 (Pres\_FName, Pres\_LName)

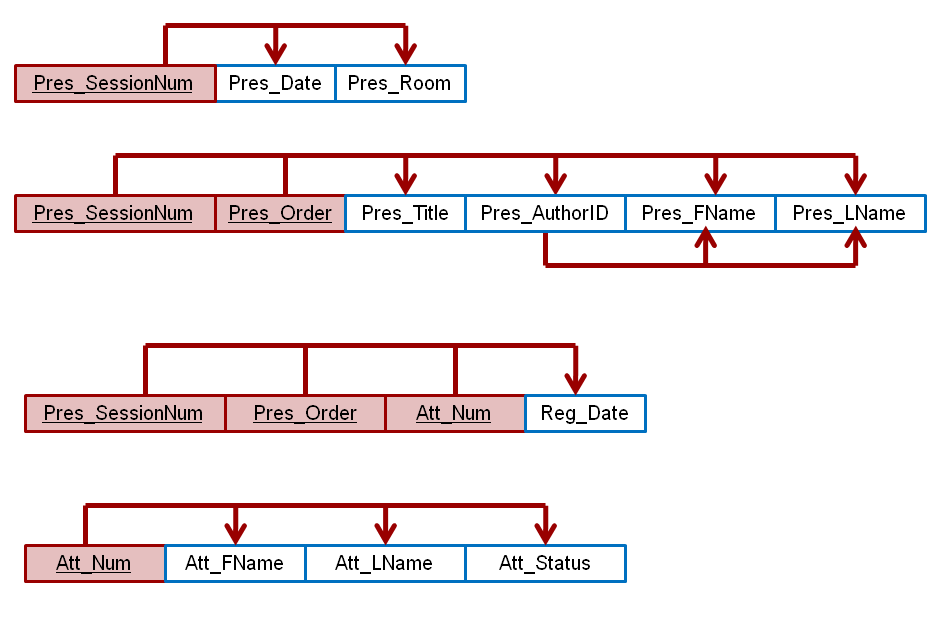
Figure P6.2b shows the revised dependency diagram including the partial and transitive dependencies.

**Figure P6.2b Revised dependency diagram for Problem 2**



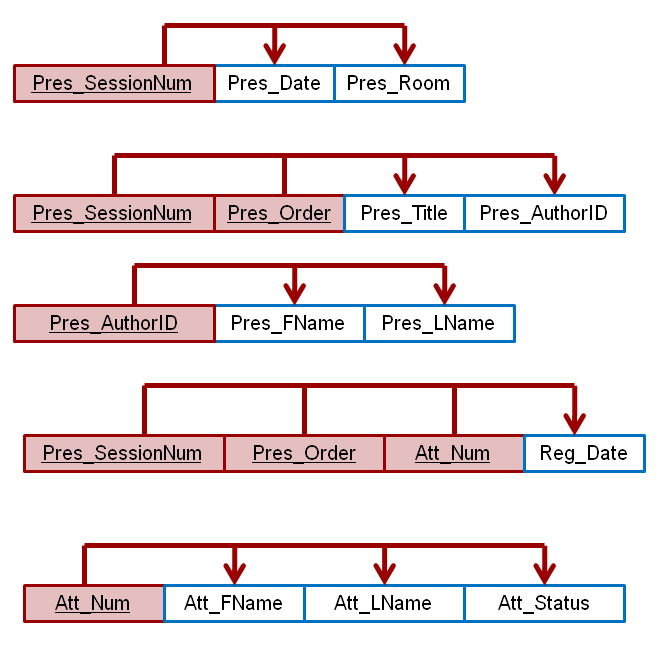
Resolving the partial dependency to achieve 2NF yields the dependency diagram shown in Figure P6.2c.

**Figure P6.2c 2NF dependency diagram for Problem 2**

****

Finally, the transitive dependency is resolved to achieve the 3NF solution shown in the final dependency diagram in Figure P6.2d.

**Figure P6.2d Final dependency diagram for Problem 2**



# Problem #16 – Aggregates on a subset of rows that are grouped (using a WHERE clause and a GROUP BY clause)

Retrieve the class name, minimum GPA, maximum GPA, average GPA, and average GPA plus 10% for each class but only for non-IS majors

SELECT s.StdCLass, MIN(s.StdGPA) AS MinGPA, MAX(s.StdGPA) AS MaxGPA, AVG(s.StdGPA) AS AvgGPA, AVG(s.StdGPA) \* 1.1 AS BoostAverageGPA

FROM Student s

WHERE s.StdMajor <> 'IS'

GROUP BY s.StdClass

ORDER BY 4 DESC

# Problem #17 – Aggregates that are grouped and subsetted (using a GROUP BY clause and a HAVING clause)

Retrieve the class name, minimum GPA, maximum GPA, average GPA, and average GPA plus 10% for each class but only for classes with an average GPA less than 3.5

SELECT s.StdCLass, MIN(s.StdGPA) AS MinGPA, MAX(s.StdGPA) AS MaxGPA, AVG(s.StdGPA) AS AvgGPA, AVG(s.StdGPA) \* 1.1 AS BoostAverageGPA

FROM Student s

GROUP BY s.StdClass

HAVING AVG(s.StdGPA) < 3.5

ORDER BY 4 DESC

# Problem #18 – Aggregates of a subset of rows that are grouped and subsetted (using a WHERE clause, a GROUP BY clause, and a HAVING clause)

Retrieve the class name, minimum GPA, maximum GPA, average GPA, and average GPA plus 10% for each class but only for non-IS majors and only for classes with an average GPA greater than 3 for non-IS majors

SELECT s.StdCLass, MIN(s.StdGPA) AS MinGPA, MAX(s.StdGPA) AS MaxGPA, AVG(s.StdGPA) AS AvgGPA, AVG(s.StdGPA) \* 1.1 AS BoostAverageGPA

FROM Student s

WHERE s.StdMajor <> 'IS'

GROUP BY s.StdClass

HAVING AVG(s.StdGPA) > 3

ORDER BY 4 DESC

# Problem #19 – Cartesian Products, how many rows expected

Perform a Cartesian Product between tables Student, Offering, Enrollment, Course, and Faculty How many columns are expected?

How many rows are expected?

SELECT c.\*, f.\*, o.\*, s.\*, e.\*

FROM Course c, Faculty f, Offering o, Student s, Enrollment e

We will have the sum of all column counts for all tables

For row counts, we will need to perform the row counts of individual tables and multiple them. We can verify this with the last query given below:

SELECT COUNT(\*)

FROM Course

SELECT COUNT(\*)

FROM Faculty

SELECT COUNT(\*)

FROM Offering

SELECT COUNT(\*)

FROM Student

SELECT COUNT(\*)

FROM Enrollment

SELECT COUNT(\*)

FROM Course c, Faculty f, Offering o, Student s, Enrollment e

# Problem #20 – Cartesian Products, figuring out which rows match

Perform a Cartesian Product between tables

Student, Offering, Enrollment, Course, and Faculty

Retrieve only the columns which are needed to show matching based on the relationship between the five tables and order in such a way as to tell the matching records

SELECT c.CourseNo, o.CourseNo,

f.FacNo, o.FacNo,

o.OfferNo, e.OfferNo,

s.StdNo, e.StdNo

FROM Course c, Faculty f, Offering o, Student s, Enrollment e

ORDER BY 1,2,3,4,5,6,7,8

# Problem #21 – Turning a Cartesian Product into an Inner Join by adding a WHERE clause to the Cross Product Synatx

Start with a Cartesian Product between tables

Student, Offering, Enrollment, Course, and Faculty

Retrieve only the columns which are needed to show matching based on the relationship between the five tables and order in such a way as to tell the matching records Add a WHERE clause to turn the Cartesian Product into an Inner Join

SELECT c.CourseNo, o.CourseNo,

f.FacNo, o.FacNo,

o.OfferNo, e.OfferNo,

s.StdNo, e.StdNo

FROM Course c, Faculty f, Offering o, Student s, Enrollment e

WHERE c.CourseNo= o.CourseNo

AND f.FacNo= o.FacNo

AND o.OfferNo= e.OfferNo

AND s.StdNo= e.StdNo

ORDER BY 1,2,3,4,5,6,7,8

# Problem #22 – Converting an Inner Join from Cross Product Syntax to Join Operator Syntax

Start with the Inner Join using Cross Product Syntax for the tables:

Student, Offering, Enrollment, Course, and Faculty

Convert to Join Operator Syntax

SELECT c.CourseNo, o.CourseNo,

f.FacNo, o.FacNo,

o.OfferNo, e.OfferNo,

s.StdNo, e.StdNo

FROM (((Course c INNER JOIN Offering o

ON c.CourseNo= o.CourseNo)

INNER JOIN Enrollment e ON o.OfferNo= e.OfferNo)

INNER JOIN Student s ON e.StdNo= s.StdNo)

INNER JOIN Faculty f ON o.FacNo= f.FacNo

ORDER BY 1,2,3,4,5,6,7,8

# Problem #23 – Combining Inner Join and WHERE, GROUP BY, and HAVING clauses

List the course number, offer number, and average grade of students enrolled in fall 2012 IS course offerings in which more than one student is enrolled

SELECT o.CourseNo, o.OfferNo, AVG(e.EnrGrade) AS AvgGrade

FROM Offering o INNER JOIN Enrollment e ON o.OfferNo= e.OfferNo

WHERE o.OffTerm= 'Fall' AND o.OffYear= 2012

AND o.CourseNoLIKE 'IS\*'

GROUP by o.CourseNo, o.OfferNo

HAVING COUNT(\*) > 1

ORDER BY 1, 3 DESC

# Problem #24 – Join not involving a Primary Key to a Foreign Key

List faculty who are also students. Include all student columns in the result.

SELECT s.\*

FROM Student s INNER JOIN Faculty f ON s.StdNo= f.FacNo

ORDER BY 1,2,3

# Problem #25 – Self Join

List faculty members who have a higher salary than their supervisor

List the faculty number, last and first names, and salary for both

SELECT f.FacNo, f.FacLastName, f.FacFirstName, f.FacSalary,

s.FacNo, s.FacLastName, s.FacFirstName, s.FacSalary

FROM Faculty f INNER JOIN Faculty s

ON f.FacSupervisor= s.FacNo

WHERE f.FacSalary> s.FacSalary

ORDER BY 1,2,3