BUAN 6320 Database Foundations for Business Analytics

Spring 2019

Instructor: Dr. James Scott

Assignment #3 – Intermediate MySQL

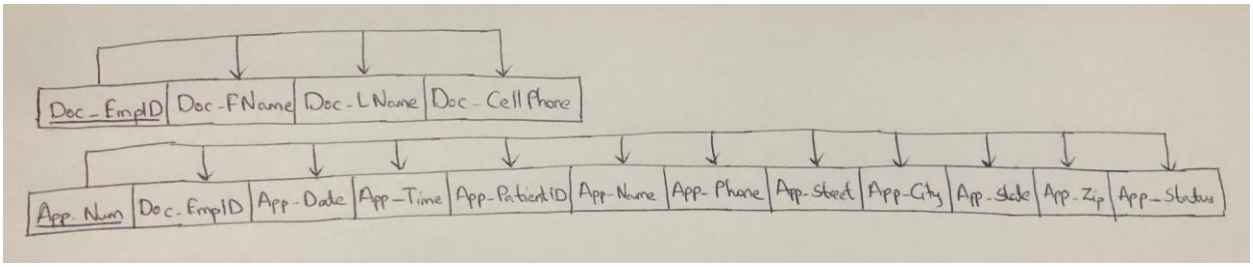
General Instructions

- Students may study together for the assignment and review each other's completed work
- Students must each complete the assignment by their own hand
- Please use the provided word document template
- Please save the completed word document into PDF format before uploading
- Please submit the PDF file electronically through eLearning before the due date and time
- Do not worry about variations among database vendors – you may write SQL to any vendor's dialect
- Do not include output – only the SQL
- Use table aliases for all tables in all queries (unless otherwise specified)
- Column aliases are required for all derived columns including aggregate columns (unless otherwise specified)
- Do not use column aliases unless required as stated previously
- If a problem does not ask for a specific sort order, use your best judgement to add a sort order

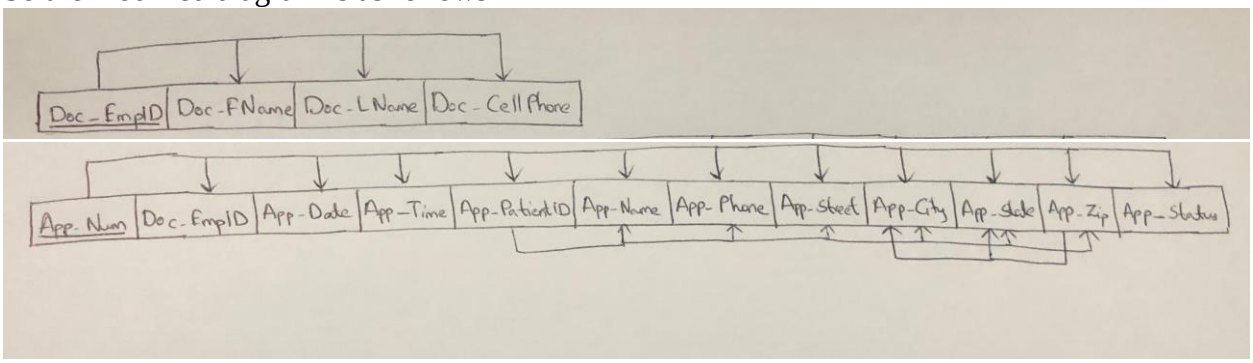
Chapter 6 Problems – Normalization of Database Tables

Do Problems 1-5 on page 224 of our textbook.

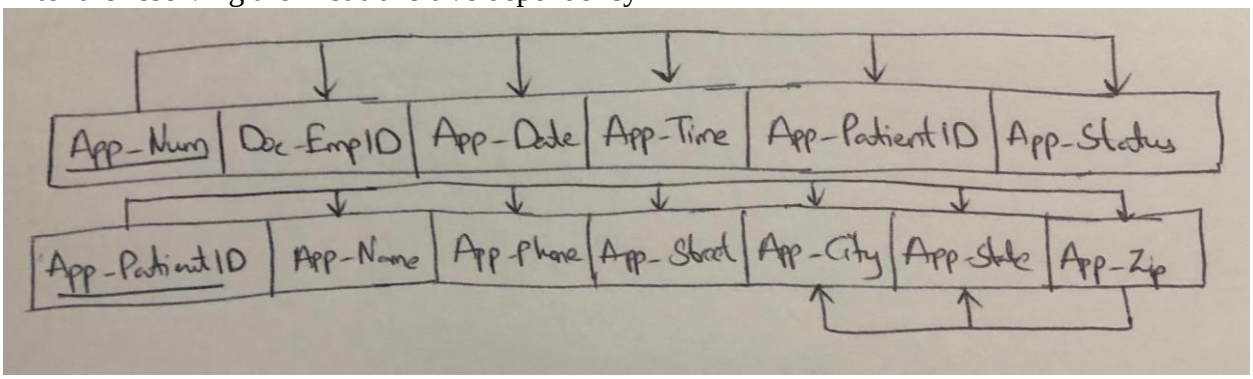
1. Using the descriptions of the attributes given in the figure, convert the ERD shown in Figure P6.1 into a dependency diagram that is in at least 3NF.
   The initial dependency diagram showing only primary key dependencies is as follows



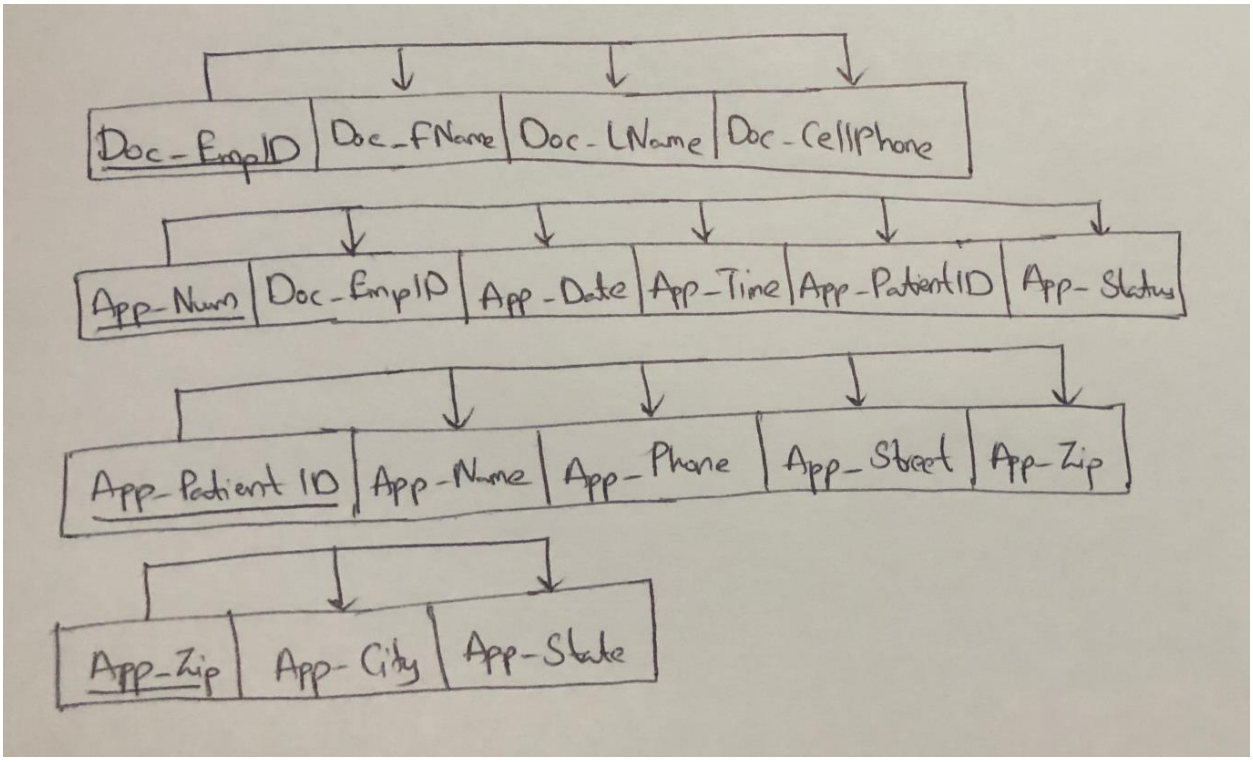
   No composite keys are there, so no problem of partial dependency exists. It is already in 2NF.
   But there are transitive dependencies as follows
   App_PatientID ⟶ (App_name, App_Phone, App_Street, App_City, App_State, App_Zip)
   Zip codes can also be used to determine City and State
   So App_Zip ⟶ (App_city, App_State)
   So the modified diagram is as follows



   After the resolving the first transitive dependency

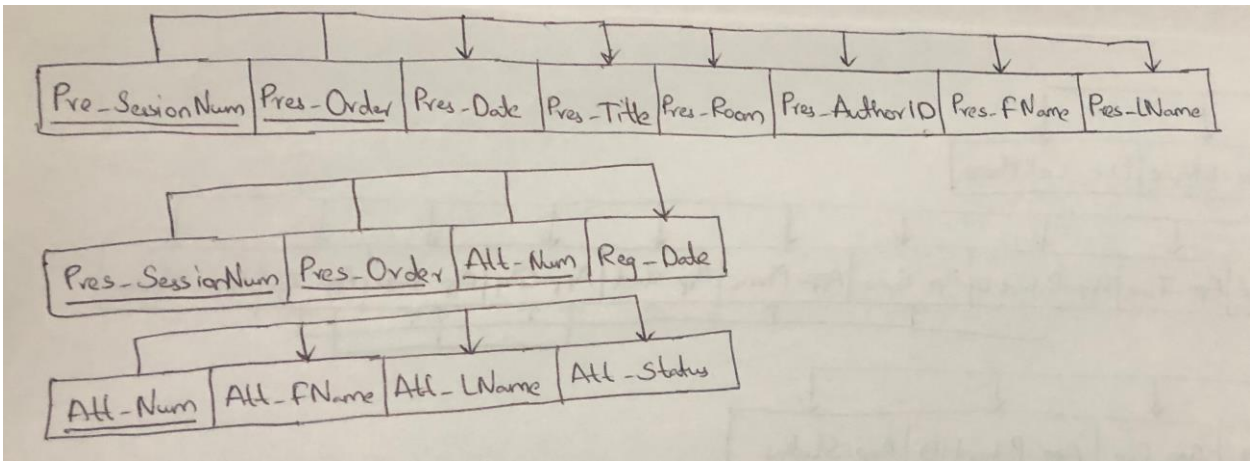


After resolving the second transitive dependency, the diagram changes to 3NF as follows



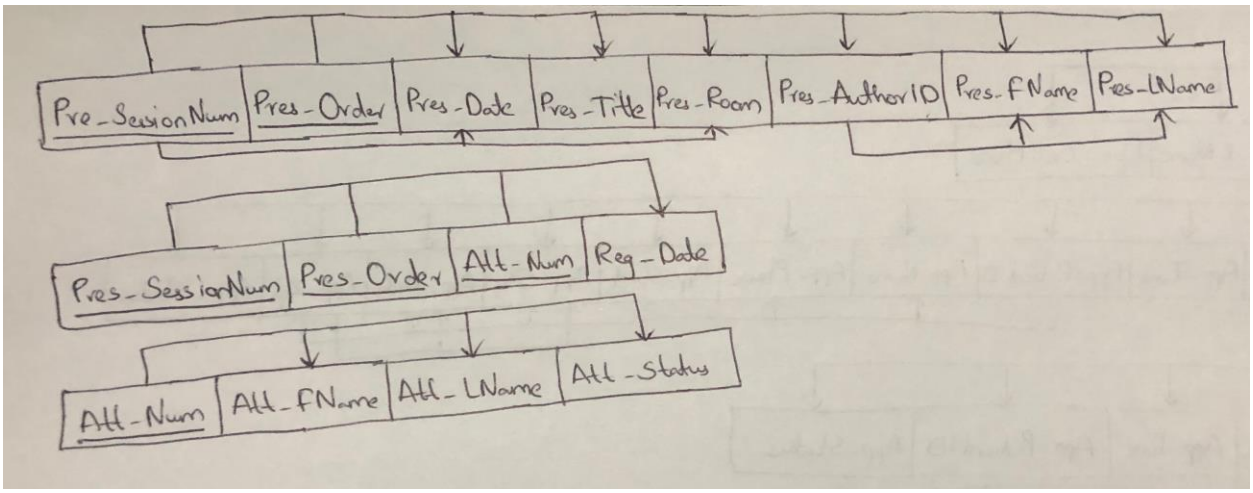
2. Using the descriptions of the attributes given in the figure, convert the ERD shown in Figure P6.2 into a dependency diagram that is in at least 3NF

The initial dependency diagram showing only primary key dependencies is as follows

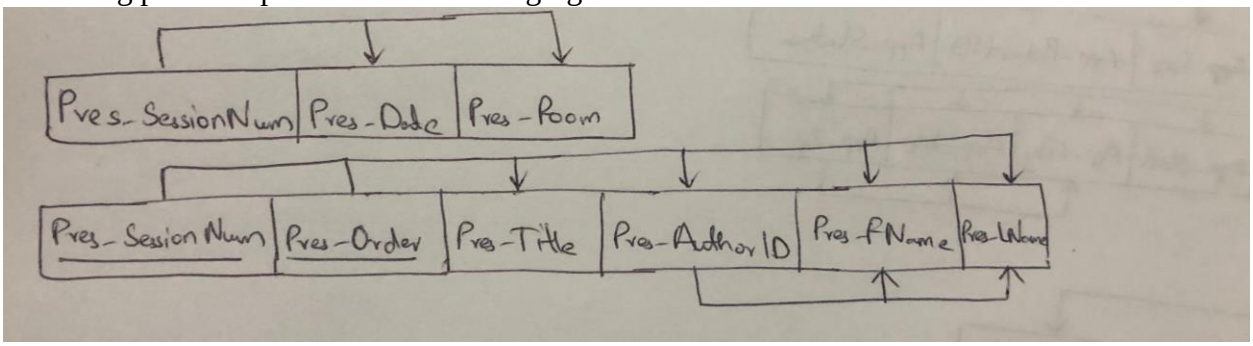


Based on the attribute descriptions partial dependencies are
Pres_SessionNum → (Pres_Date, Pres_Room)
Transitive dependencies are
Pres_AuthorID → (Pres_FName, Pres_LName)

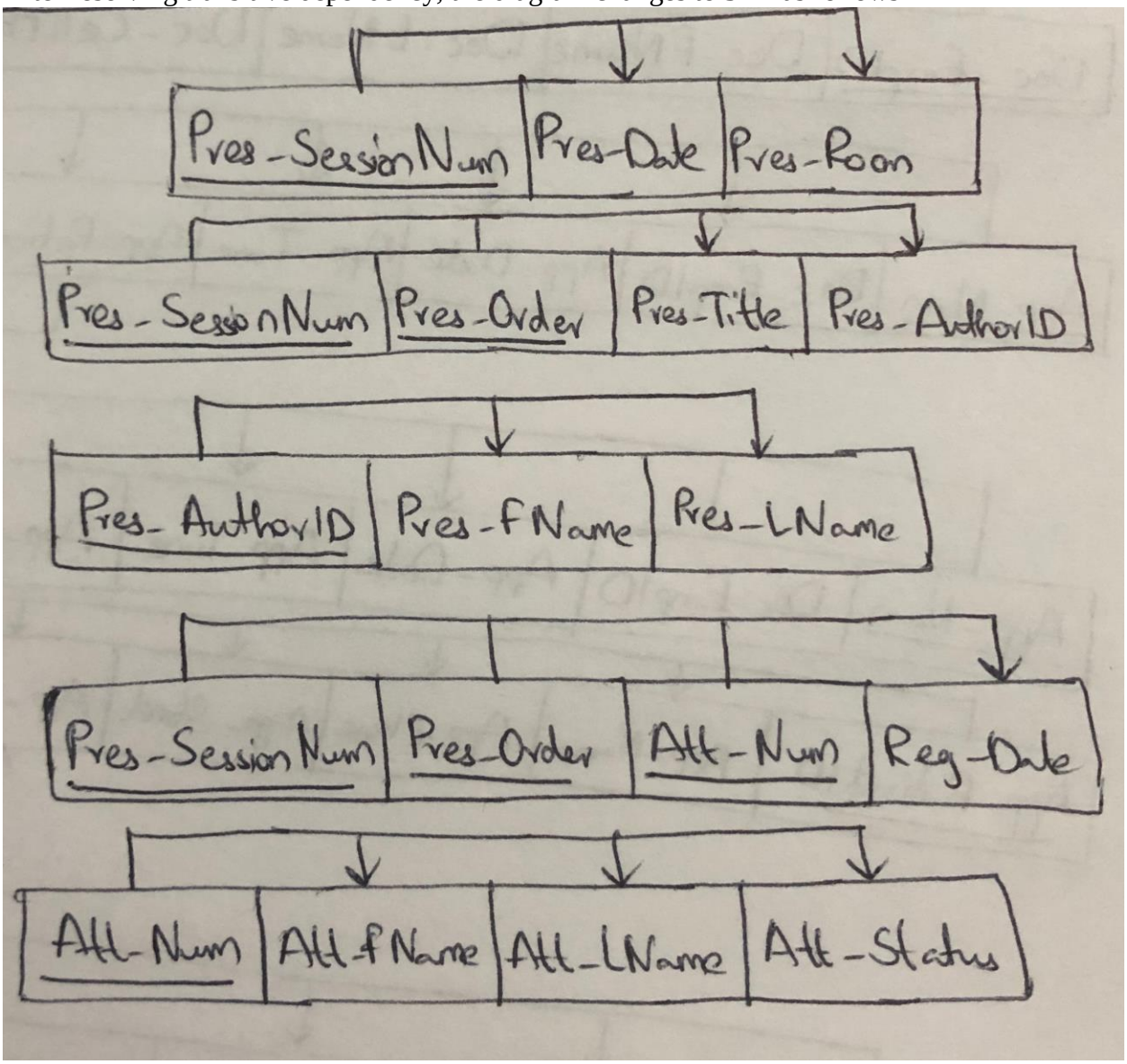
So the modified diagram showing partial and transirtive dependencies is as follows



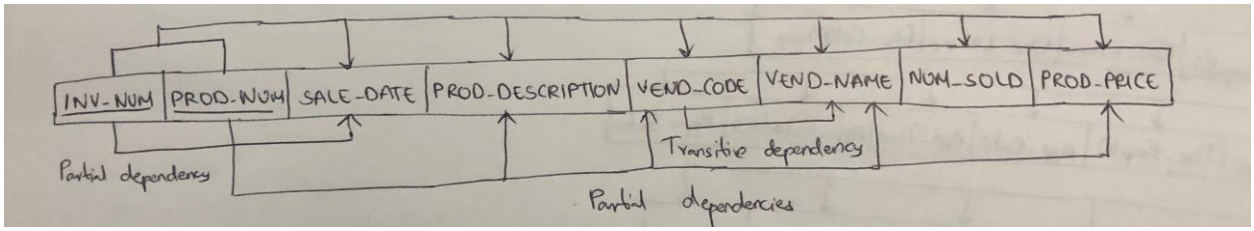
Resolving partial dependencies and changing to 2NF



After resolving transitive dependency, the diagram changes to 3NF as follows

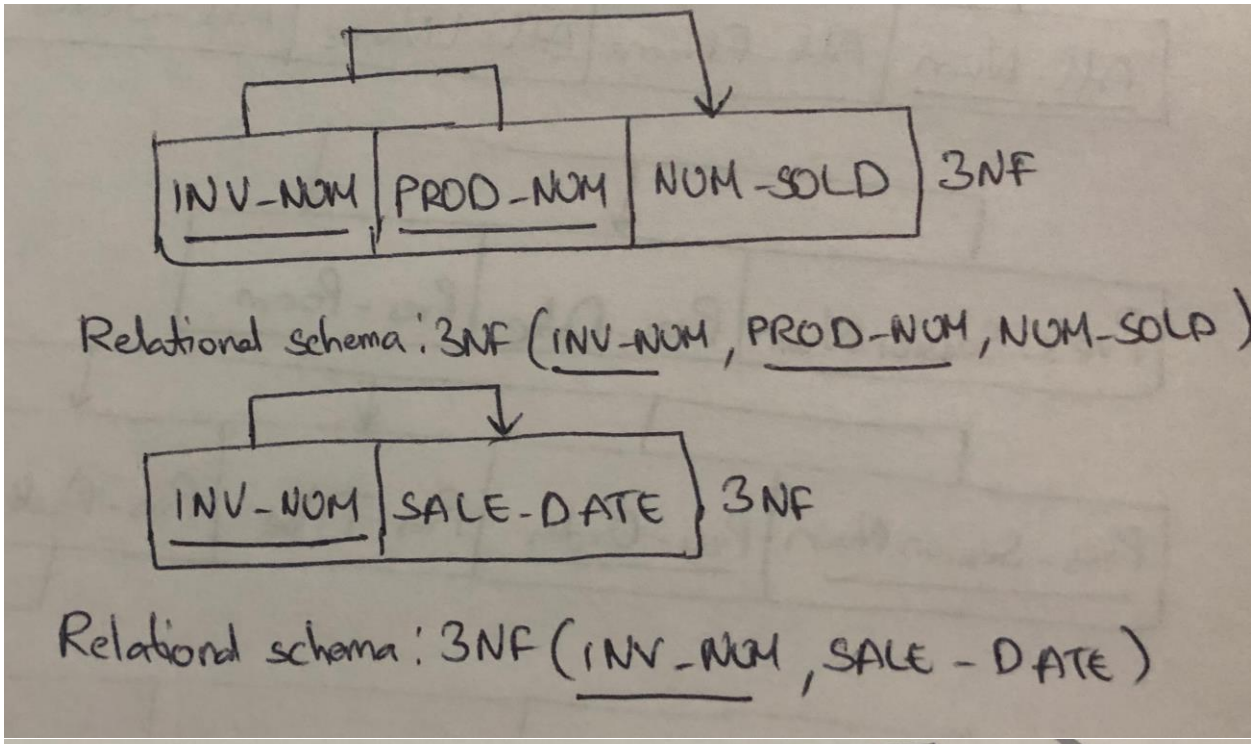


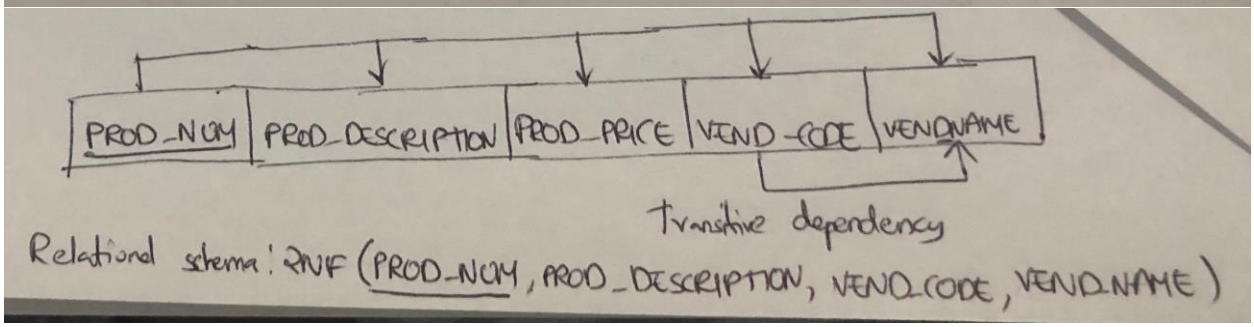
3. Using the INVOICE table structure shown in Table P6.3, do the following:
   a. Write the relational schema, draw its dependency diagram, and identify all dependencies, including all partial and transitive dependencies. You can assume that the table does not contain repeating groups and that an invoice number references more than one product. (Hint: This table uses a composite primary key.)



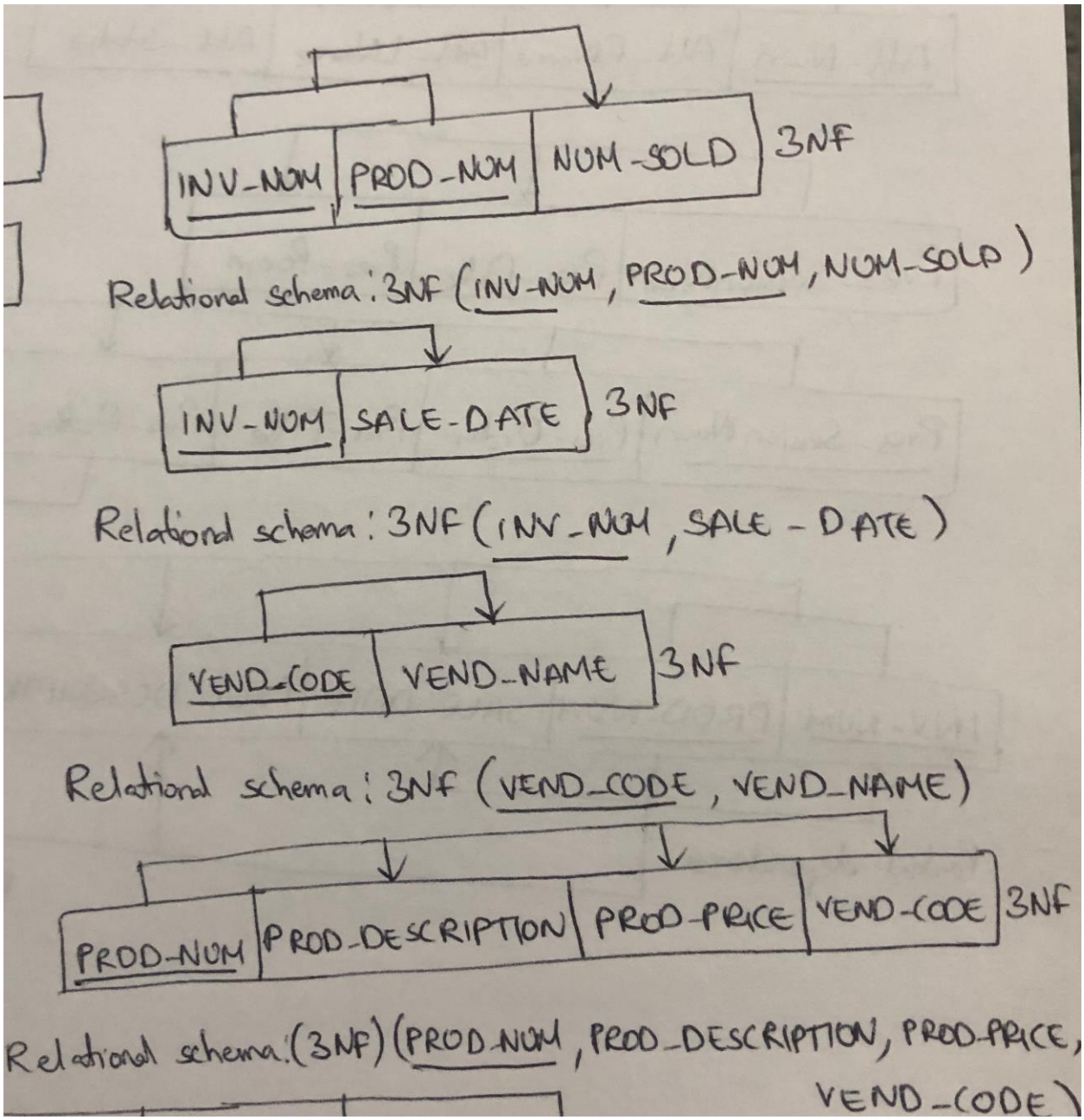
Relational Schema : 1NF (INV_NUM, PROD_NUM, SALE_DATE, PROD_DESCRIPTION, VEND_CODE, VEND_NAME, NUM_SOLD, PROD_PRICE)

b. Remove all partial dependencies, write the relational schema, and draw the new dependency diagrams. Identify the normal forms for each table structure you created.

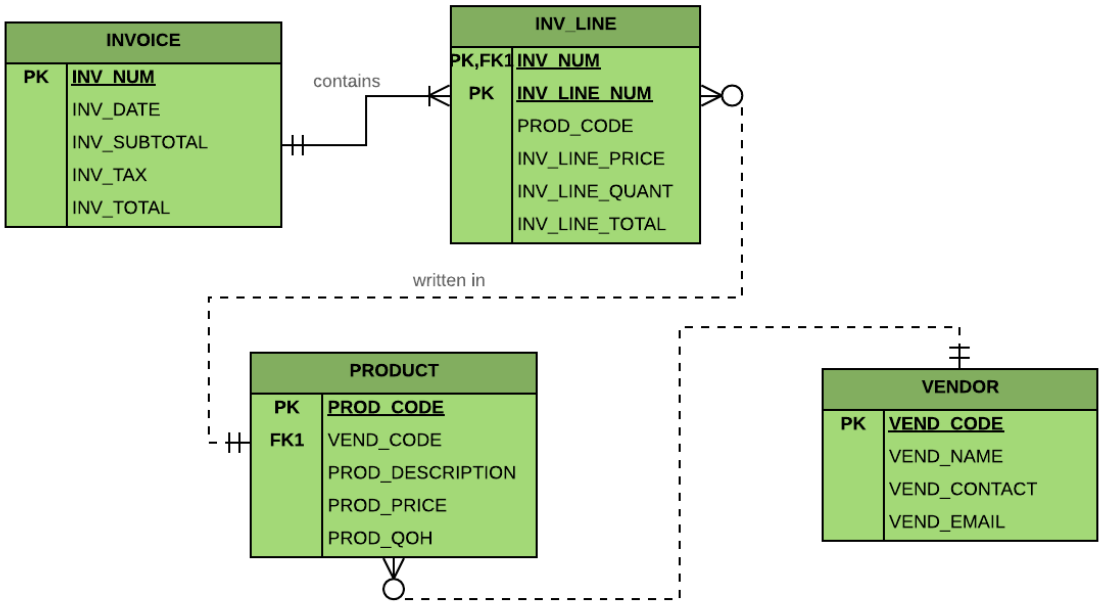




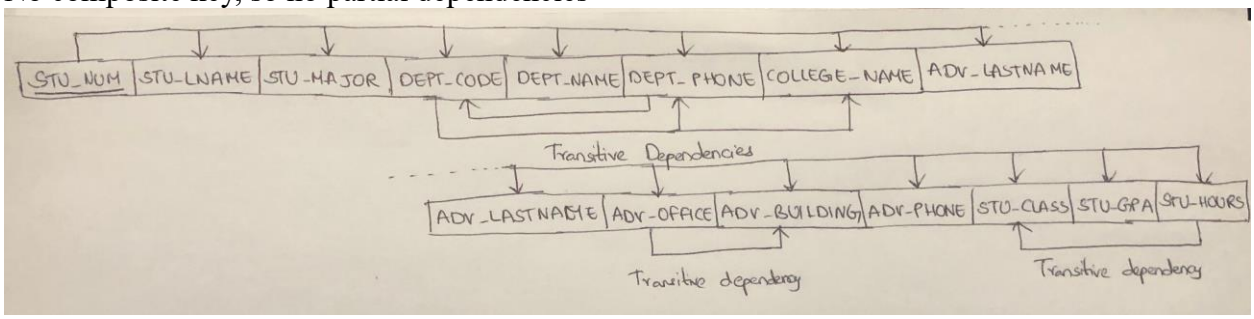
c. Remove all transitive dependencies, write the relational schema, and draw the new dependency diagrams. Also identify the normal forms for each table structure you created.



d. Draw the Crow's Foot ERD.

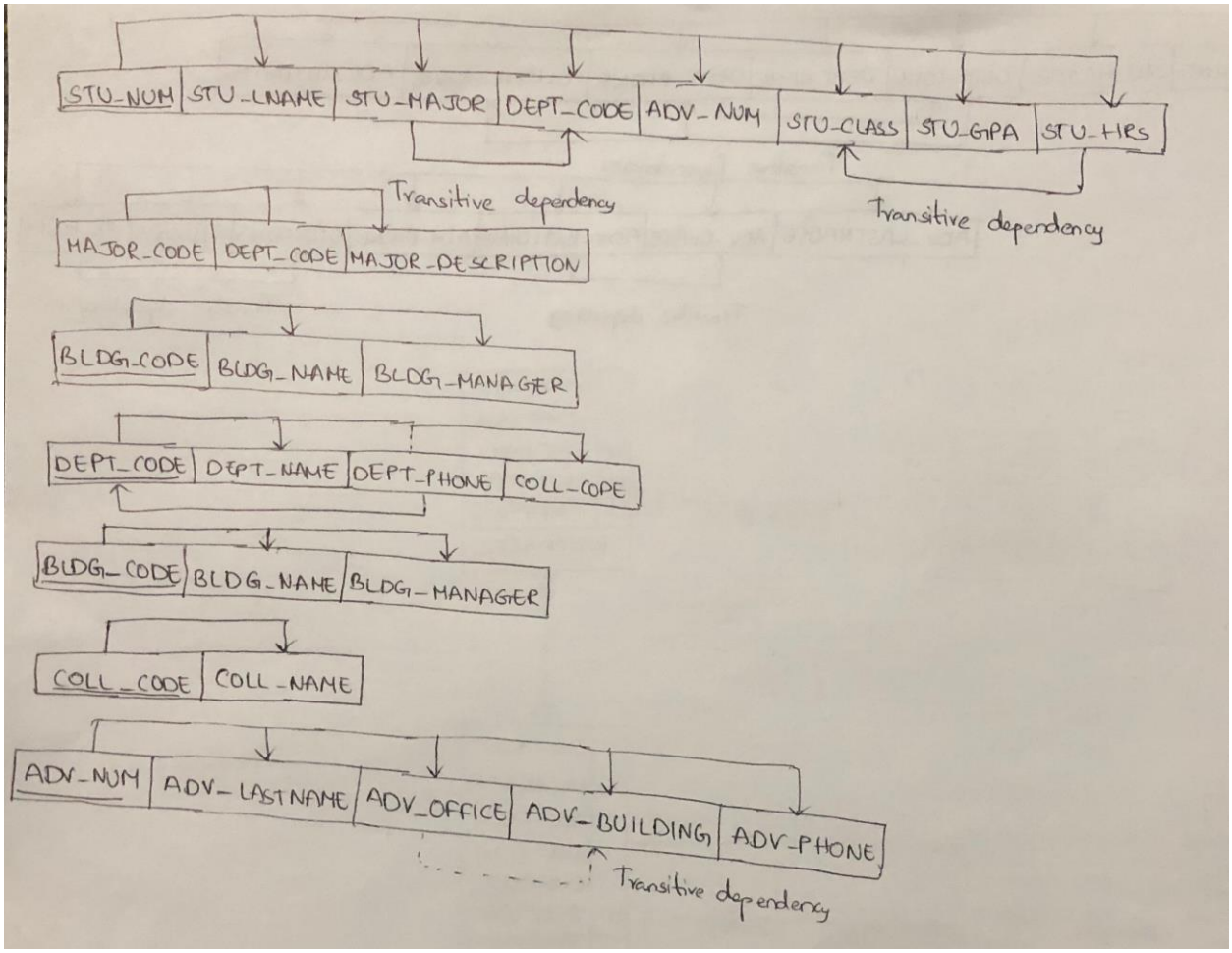


4. Using the STUDENT table structure shown in Table P6.4, do the following:
   a. Write the relational schema and draw its dependency diagram. Identify all dependencies, including all transitive dependencies.
   No composite key, so no partial dependencies



   Relational schema for the dependency diagram is
   Student → (STU_NUM, STU_LNAME, STU_MAJOR, DEPT_CODE, DEPT_NAME, DEPT_PHONE, COLLEGE_NAME, ADV_LASTNAME, ADV_OFFICE, ADV_BUILDING, ADV_PHONE, STU_CLASS, STU_GPA, STU_HOURS)

   b. Write the relational schema and draw the dependency diagram to meet the 3NF requirements to the greatest practical extent possible. If you believe that practical considerations dictate using a 2NF structure, explain why your decision to retain 2NF is appropriate. If necessary, add or modify attributes to create appropriate determinants and to adhere to the naming conventions.



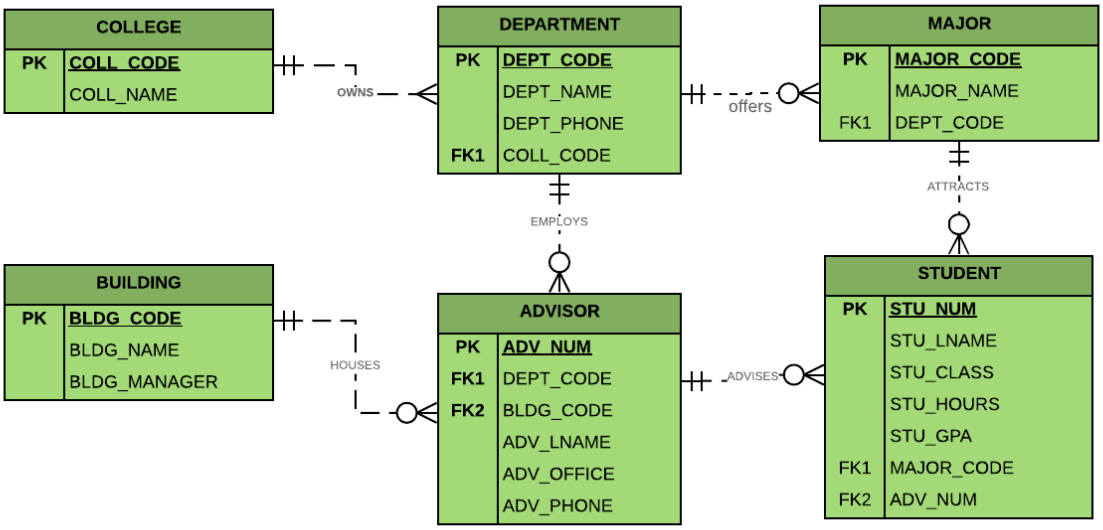
Student → (STU_NUM, STU_LNAME, STU_MAJOR, DEPT_CODE, ADVISOR_NUM, STU_CLASS, STU_GPA, STU_HOURS)
ADVISOR_NUM attribute is added. It acts as foreign key to the advisor attributes
MAJOR → (MAJOR_CODE, DEPT_CODE, MAJOR_DESCRIPTION)
BUILDING → (BLDG_CODE, BLDG_NAME, BLDG_MANAGER)
DEPARTMENT → (DEPT_CODE, DEPT_NAME, DEPT_PHONE, COLL_CODE)
COLLEGE → (COLL_CODE, COLL_NAME)

STUDENT table is in 2NF as it has transitive dependencies. Structure simplicity is the criteria considered here. We can create a separate table MAJOR, relate it to the DEPARTMENT table and take out the major code from STUDENT to remove the transitive dependency if there is business requirement to track the information of each major. But this create implementation complexity. So, we retain table in 2NF.

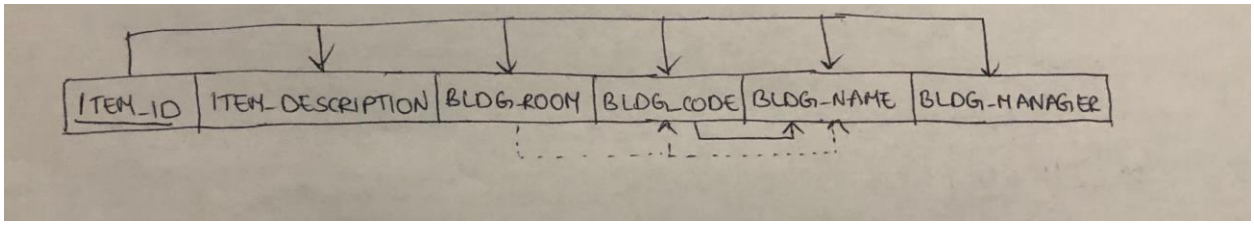
The dotted line in the ADVISOR table is just an interpretation of transitive dependency and can be ignored based on the practical business requirements

c. Using the results of Problem 4, draw the Crow's Foot ERD

-

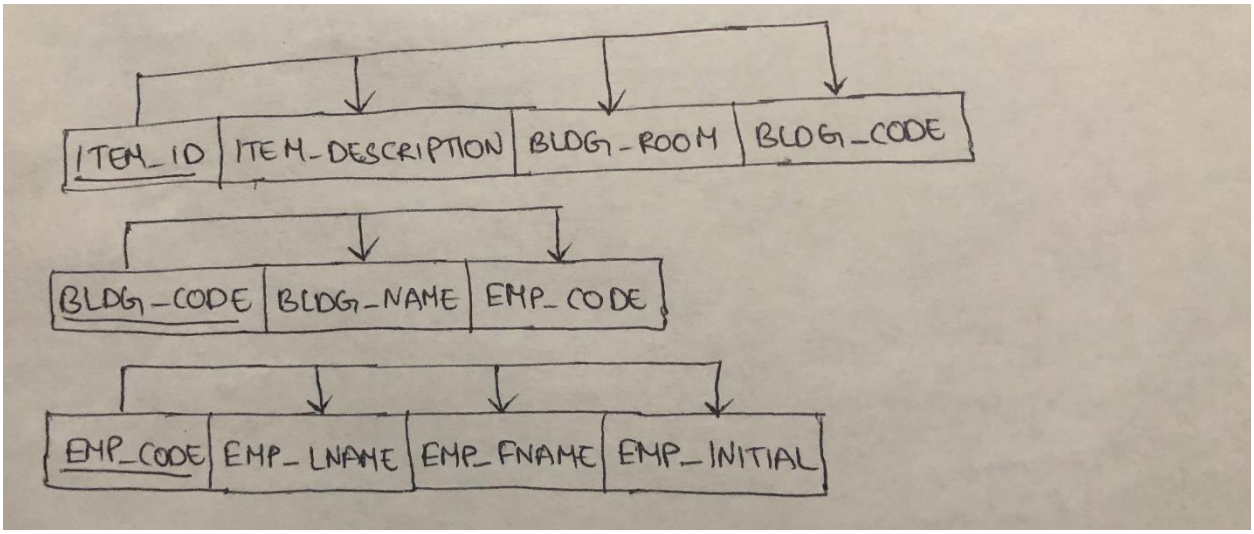


5. To keep track of office furniture, computers, printers, and other office equipment, the FOUNDIT Company uses the table structure shown in Table P6.5.
   a. Given that information, write the relational schema and draw the dependency diagram. Make sure that you label the transitive and/or partial dependencies.



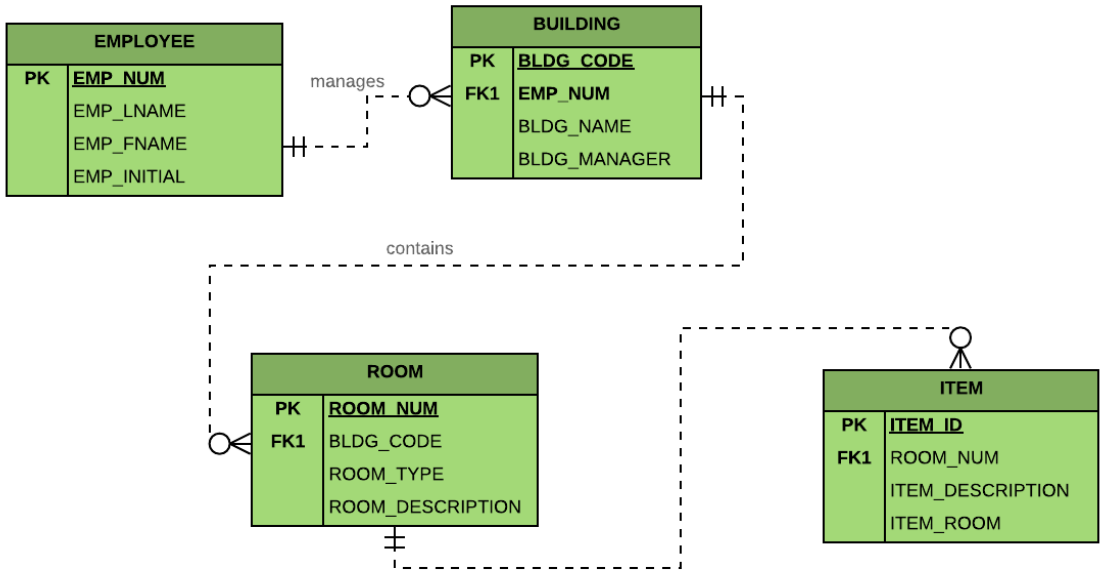
ITEM → (ITEM_ID, ITEM_ DESCRIPTION, ITEM_ROOM, BLDG_CODE, BLDG_NAME, BLDG_MANAGER)

   b. Write the relational schema and create a set of dependency diagrams that meet 3NF requirements. Rename attributes to meet the naming conventions, and create new entities and attributes as necessary.



EMPLOYEE → (EMP_CODE, EMP_LNAME, EMP_FNAME, EMP_INITIAL)
BUILDING → (BLDG_CODE, BLDG_NAME, EMP_CODE)
ITEM → (ITEM_ID, ITEM_DESCRIPTION, ITEM_ROOM, BLDG_CODE)

c. Draw the Crow's Foot ERD.



Chapter 7 Problems – Introduction to Structured Query Language (SQL)

Do Problems 1-8 on pages 295 of our textbook.

1. Write the SQL code that will create the table structure for a table named EMP_1. This table is a subset of the EMPLOYEE table. The basic EMP_1 table structure is summarized in the following table. (Note that the JOB_CODE is the FK to JOB.)

```
Create Table Emp_1 (
Emp_Num      Char(3)        Primary Key,
Emp_Lname Varchar(15)       Not Null,
Emp_Fname Varchar(15)       Not Null,
Emp_Initial Char(1),
Emp_Hiredate Date,
Job_Code Char(3),
Foreign Key (Job_Code) References Job);
```

2. Having created the table structure in Problem 1, write the SQL code to enter the first two rows for the table shown in Figure P7.2.

```
Insert Into Emp_1 Values ('101', 'News', 'John', 'G', '2000-11-8', '502');
Insert Into Emp_1 Values ('102', 'Senior', 'David', 'H', '1989-7-12', '501');
```

3. Assuming that the data shown in the EMP_1 table have been entered, write the SQL code that will list all attributes for a job code of 502

```
Select * from Emp_1 where Job_Code = '502' ;
```

4. Write the SQL code that will save the changes made to the EMP_1 table.

```
COMMIT;
```

5. Write the SQL code to change the job code to 501 for the person whose employee number (EMP_NUM) is 107. After you have completed the task, examine the results and then reset the job code to its original value.

```
Update Emp_1
Set Job_Code = '501'
Where Emp_Num= '107';

To examine the results
Select * from Emp_1 where Emp_Num = '107';

To reset
Rollback;
```

6. Write the SQL code to delete the row for William Smithfield, who was hired on June 22, 2004, and whose job code is 500. (Hint: Use logical operators to include all of the information given in this problem.)

```
Delete  From Emp_1
Where Emp_Lname = 'Smithfield'
And    Emp_Fname = 'William'
And    Emp_Hiredate = '22-June-04'
And    Job_Code = '500';
```

7. Write the SQL code that will restore the data to its original status; that is, the table should contain the data that existed before you made the changes in Problems 5 and 6.
Rollback;

8. Write the SQL code to create a copy of EMP_1, naming the copy EMP_2. Then write the SQL code that will add the attributes EMP_PCT and PROJ_NUM to the structure. The EMP_PCT is the bonus percentage to be paid to each employee. The new attribute characteristics are: EMP_PCT NUMBER(4,2) PROJ_NUM CHAR(3) (Note: If your SQL implementation allows it, you may use DECIMAL(4,2) or NUMERIC (4,2) rather than NUMBER(4,2).)

```
Create Table Emp_2 As Select * From Emp_1;

Alter Table Emp_2
Add (Emp_Pct Decimal (4,2)),
Add (Proj_Num Char(3));
```

Using MYSQL to answer the following eight (9-16) practical SQL questions from the UniversityDDL-Bad.

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) minimum,
max(S.StdGPA) maximum,
avg(S.StdGPA) average,
(avg(S.StdGPA) + 0.1* avg(S.StdGPA)) additional from Student S
group BY S.StdClass;
having avg(S.StdGPA) < 3.5;
```

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) minimum,
max(S.StdGPA) maximum,
avg(S.StdGPA) average,
(avg(S.StdGPA) + 0.1* avg(S.StdGPA)) additional from Student S
where S.StdMajor != 'IS'
group BY S.StdClass;
having avg(S.StdGPA) > 3;
```

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?

```
Select * from Faculty F
cross join Student S
cross join Offering O
cross join Enrollment E
cross join Course C;
```

No of columns we take in the select query to perform cartesian product
Here we can get maximum of 34 columns in total

How many rows are expected?
Product of number of rows
Select count(*) from Student;
Select count(*) from Offering;
Select count(*) from Enrollment;
Select count(*) from Course;
Select count(*) from Faculty;
11*13*37*7*6 = 222222

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 S.StdNo, E.StdNo, E.OfferNo, O.OfferNo, C.CourseNo,O.CourseNo, F.FacNo, O.FacNo
from Faculty F cross join Student S
cross join Offering O
cross join Enrollment E
cross join Course C
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 Syntax

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 s.StdNo, e.StdNo, E.OfferNo, o.OfferNo, c.CourseNo,o.CourseNo, f.FacNo, o.FacNo
from faculty f cross join student s
cross join offering o
cross join enrollment e
cross join course c
where s.StdNo = e.StdNo and
e.OfferNo = o.OfferNo and
c.CourseNo = o.CourseNo and
f.FacNo = o.FacNo
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 s.StdNo, e.StdNo, e.OfferNo, o.OfferNo, c.CourseNo,o.CourseNo, f.FacNo, o.FacNo
from student s inner join enrollment e
on
s.StdNo = e.StdNo
inner join offering o
on
e.OfferNo = o.OfferNo
inner join course c
on
c.CourseNo = o.CourseNo
inner join faculty f on
f.FacNo = o.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 2010 IS course offerings in which more than one student is enrolled.

```
select o.CourseNo, o.OfferNo, avg(s.StdGPA)
from student s join enrollment e
on s.StdNo = e.StdNo
join offering o
on e.OfferNo = o.OfferNo
where o.OffTerm = 'FALL' and o.OffYear = '2010' and  o.CourseNo like 'IS%'
group by 1,2
having count(s.StdNo) > 1;
```

No data for the given year and term

Using another year say, 2009 for which we have data

```
select o.CourseNo, o.OfferNo, avg(s.StdGPA)
from student s join enrollment e
on s.StdNo = e.StdNo
join offering o
on e.OfferNo = o.OfferNo
where o.OffTerm = 'FALL' and o.OffYear = '2009' and  o.CourseNo like 'IS%'
group by 1,2
having count(s.StdNo) > 1;
```