

Homework Assignment 2

(due through Blackboard before 6:00 PM on Wednesday, 2/7)

Things you should please do, lest you should lose my tender affection:

- Review the syllabus rules for submitting assignments
- Somehow incorporate your last name into the name of the file you upload to the Dropbox
- Make sure your name is inside the document itself, at the beginning or in the header

Certification of Academic Honesty (see syllabus for details)

=====

I certify that:

- **This homework assignment is entirely my own work.**
- **I have not quoted the words of any other person from a printed source or a website without indicating what has been quoted and providing an appropriate citation.**
- **I have not submitted this paper / project to satisfy the requirements of any other course.**

Signature: Joseph Mulray

Date : 2/7/2018

=====

1. Is that Normal? (40 points)

You have just been hired by BudgetBurners as their newest consultant. You have been asked to review an internal billing rate database that is having some data quality issues. The owner of the database provides you with a database schema, and you see that there is a single table defined as follows:

Consultants(ConsultantId, ProjectId, FirstName, LastName, BillingRate, ManagerId, FNm, LNm, Email, Skill1, Skill2, Skill3, Skill4)

Suddenly, a piece of paper falls from the ceiling. On it appears to be a list of functional dependencies:

ConsultantId, ProjectId → FirstName
ConsultantId, ProjectId → LastName
ConsultantId, ProjectId → BillingRate
ConsultantId, ProjectId → ManagerId
ConsultantId, ProjectId → FNm
ConsultantId, ProjectId → LNm
ConsultantId, ProjectId → Email
ConsultantId → FirstName
ConsultantId → LastName
ManagerId → FNm

Frein – 2/1/2018

ManagerId \rightarrow LNm
ManagerId \rightarrow Email

Following the example at the end of the “Normalization Foundation” lecture slides, normalize the above relation on the basis of these functional dependencies, showing the results of achieving each normal form (1st, 2nd, and 3rd) incrementally. That is, show what it would look like to make only the changes needed to achieve 1NF, then show the additional changes needed to get to 2NF, and then the changes needed to get to 3NF. Show the complete schema (all tables) for each normal form.

1st Normal Form

Consultants(ConsultantId, ProjectId, FirstName, LastName, BillingRate, ManagerId, FNm, LNm, Email)
ConsultantSkills(ConsultantId, Skill)

2nd Normal Form

Project(ConsultantId, ProjectId, BillingRate, ManagerId, FNm, LNm, Email)
Consultant(ConsultantId, FirstName, LastName)
ConsultantSkills(ConsultantId, Skill)

3rd Normal Form

Managers(ManagerId, FNm, LNm, Email)
Project(ConsultantId, ProjectId, ManagerId, BillingRate)
Consultant(ConsultantId, FirstName, LastName)
ConsultantSkills(ConsultantId, Skill)

2. So Help Me Codd (10 points)

Consider the following table:

GoatDominance(Goat, Mountain, Cave)

And the following functional dependencies:

Goat, Mountain \rightarrow Cave
Cave \rightarrow Mountain

Is this table compliant with Boyce-Codd Normal Form? If yes, say so. If not, normalize it to BCNF compliance and show the resulting schema.

No this is not in BCNF, this is something that would be allowed in 3NF but would not be represented in BCNF.

GoatCave(Goat, Cave)
CaveMountain (Cave, Mountain)

3. The Awesome Data Model (25 points)

AwesomeProducts (AP) is a brick-and-mortar chain which sells products that are awesome. (Go figure.) However, their database developers fall into the “distinctly otherwise than awesome” category. As AP looks to open an online storefront, they need some help in cleaning up their database. This is where you, the highly paid database consultant, burst onto the scene to craft some sweet data storage goodness and collect redonkulous *ka-ching*.

As it happens, AP grew up from a single-store operation. Unfortunately, whenever they would open a new store, the database folks would just make copies of the tables from the original store. As such, there is now a separate products table for each of the three AP stores. The tables are named Products_Store1, Products_Store2, and Products_Store3. Some of the products listed in these tables are common across all stores, whereas others are found in only one or two of the stores. If a product is found in more than one store, it has the same product id and name at each store. (These are the only two columns in the tables in question).

You can recreate these tables by running the code in the files that accompany this assignment – Products_Store1.txt, Products_Store2.txt, Products_Store3.txt.

Your assignment is as follows:

- a) Develop a list of products that are found in all three stores
- b) Develop a list of products that can be found in only one of the stores

2

Show the answers you will give to AP, along with the SQL you used to derive those answers.

a)

```
SELECT P1.ProductId,  
       P1.Name  
FROM Products_Store1 P1  
INNER JOIN Products_Store2 P2  
  on P1.ProductId = P2.ProductId  
INNER JOIN Products_Store3 P3  
  on P3.ProductId = P2.ProductId;
```

```
SQL> SELECT P1.ProductId,
P1.Name
FROM Products_Store1 P1
INNER JOIN Products_Store2 P2
on P1.ProductId = P2.ProductId
INNER JOIN Products_Store3 P3
on P3.ProductId = P2.ProductId; 2    3    4    5    6    7

PRODUCTID NAME
-----
1005 Super Great Widget
1013 Super Extraordinary Widget
1032 Fantastic Extraordinary Hammer
1034 Awesome Mega Glider
1036 Fantastic Mega Glider
1046 Awesome Extraordinary Glider
1055 Incredible Great Gum
1056 Fantastic Great Gum
1064 Fantastic Extraordinary Gum

9 rows selected.
```

b)

```
SELECT P1.ProductId, P2.ProductId,
P3.ProductId
FROM Products_Store1 P1
FULL OUTER JOIN Products_Store2 P2
ON P1.ProductId = P2.ProductId
FULL OUTER JOIN Products_Store3 P3
ON P3.ProductId = P2.ProductId
OR P3.ProductId = P1.ProductId
WHERE (P1.ProductId IS NULL AND P2.ProductId IS NULL)
OR (P1.ProductId IS NULL AND P3.ProductId IS NULL)
OR (P3.ProductId IS NULL AND P2.ProductId IS NULL);
```

```

SQL> SELECT P1.ProductId, P2.ProductId,
        P3.ProductId
        FROM Products_Store1 P1
        FULL OUTER JOIN Products_Store2 P2
        ON P1.ProductId = P2.ProductId
        FULL OUTER JOIN Products_Store3 P3
        ON P3.ProductId = P2.ProductId
        OR P3.ProductId = P1.ProductId
WHERE (P1.ProductId IS NULL AND P2.ProductId IS NULL)
OR (P1.ProductId IS NULL AND P3.ProductId IS NULL)
OR (P3.ProductId IS NULL AND P2.ProductId IS NULL)
  2      3      4      5      6      7      8      9     10     11     12 ;

```

PRODUCTID	PRODUCTID	PRODUCTID
1001		
1007		
1019		
1023		
1026		
1027		
1033		
1037		
1039		
1042		
1052		

PRODUCTID	PRODUCTID	PRODUCTID
1054		
1057		
	1002	
	1003	
	1006	
	1008	
	1010	
	1011	
	1012	
	1014	
	1018	

PRODUCTID	PRODUCTID	PRODUCTID
	1020	
	1021	
	1022	
	1024	
	1025	
	1029	
	1038	
	1040	
	1047	
	1053	
	1058	

PRODUCTID	PRODUCTID	PRODUCTID
	1060	
	1063	
		1004
		1015
		1016
		1017
		1028
		1035
		1041
		1045
		1048

PRODUCTID	PRODUCTID	PRODUCTID
		1050
		1051

46 rows selected.

4. The Begatitudes (25 points)

Suddenly, you are magically transported into a faraway land, into the service of an evil king named Badassius. (Deal with it.) Although this king is all swords and sorcery, he still has an Oracle server in his castle.

The king is upset over an ancient prophecy which states the following: *The tyrant king Badassius shall be overthrown by a member of the 16th generation of the House of Aaron.* The king somehow got hold of the family tree for the House of Aaron, but instead of being in a convenient graphical tree structure, it was actually just a big list of “So-and-so begat you-know-who” statements. Desperate to identify the 16th generation, the king commanded the Royal DBA to create a schema that could store all of the info from the family tree. The Royal DBA created the following table:

```
CREATE TABLE Generations
(
    Id NUMBER(4) CONSTRAINT Ye_Olde_Gen_PK PRIMARY KEY,
    Person VARCHAR(50),
    BegatBy NUMBER(4) CONSTRAINT Ye_Olde_Gen_FK REFERENCES Generations(Id)
);
```

(This table, along with all of its associated data, can be recreated by running the Begat.txt file that accompanies this assignment.)

The king’s minions entered all the information from the House of Aaron family tree into this table. Even though it takes two to tango, this family tree only records one “begatter” for every person in the tree. So, as far as this data is concerned, every person in the tree has only one parent worth worrying about, and this person is referenced in the BegatBy column.

In the house of Aaron, Aaron is the root of the family tree and is considered the first generation. Accordingly we have no BegatBy value for this person. However, everybody else in the tree / table was begat by some other person in the tree / table.

The king has given you two tasks:

- c) List the names of the members of the sixteenth generation of the house of Aaron, if such people even exist yet. (Here you should construe “generation” as a level of the family tree, irrespective of when particular individuals on that level were born.)
- d) Count the members of the house of Aaron who never begat anybody else.

Show the answers you will give to the king, along with the SQL you used to derive those answers.

c)

Couldn’t get this one figured out correctly. From my sql code I was trying to recursively count the children from the parents by doing a union from the generated table that included a count of which
Frein – 2/1/2018

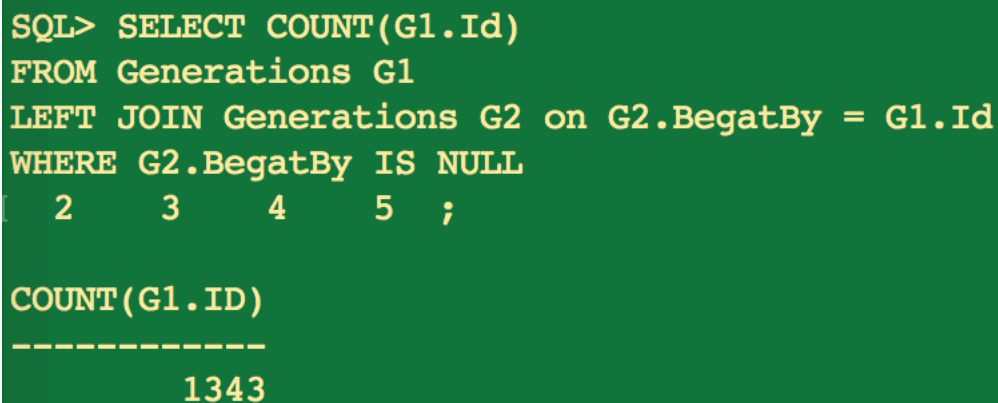
generation that person was on and increment for each child. Wasn't able to get it working in the end but gave it my best shot.

```
WITH TotalGenerations(Id, Person, BegatBy, depth) AS (  
  SELECT Id, Person, BegatBy, 0  
  FROM Generations  
  WHERE BegatBy is NULL  
  UNION ALL  
  SELECT Generations.Id,  
         Generations.Person,  
         Generations.BegatBy,  
         Generations.GenerateNum+1  
  FROM Generations  
        JOIN TotalGenerations ON Generations.BegatBy = TotalGenerations.Id  
)  
SELECT * FROM TotalGenerations  
WHERE GenerateNum=16;
```

d)

Members of House Aaron who never begat anybody else:
1343 Members

```
SELECT * FROM Generations G1  
LEFT OUTER JOIN Generations G2  
  ON G1.Id = G2.BegatBy  
WHERE G2.Id IS NULL;
```



```
SQL> SELECT COUNT(G1.Id)  
FROM Generations G1  
LEFT JOIN Generations G2 on G2.BegatBy = G1.Id  
WHERE G2.BegatBy IS NULL  
2      3      4      5 ;  
  
COUNT(G1.ID)  
-----  
1343
```

<End of Assignment>