**Homework Assignment 2**

Due: 11:59PM Saturday, May 2, 2020

|  |
| --- |
| * This is an individual work; Please be clear with HGU CSEE Standard:   + Submitting assignments or program codes written by others or acquired from the internet without explicit approval of the professor is regarded as cheating.   + Showing or lending one’s own homework to other student is also considered cheating that disturbs fair evaluation and hinders the academic achievement of the other student.   + It is regarded as cheating if two or more students conduct their homework together and submit it individually when the homework is not a group assignment. * Read the assignment carefully. You are **allowed to re-use any of the queries from the lecture slides** while developing solutions to the problems. * When finished, submit your work to *LMS.* |

**1. (1 pt. per each blank – total 17 blanks) Read textbook sections 2.3-2.5, 3-3.6 and 4-4.4, and fill in the blanks.**

(a) (SQL) is the principal language used to describe and manipulate relational databases. It makes a distinction between three kinds of relations: (Stored) relations, (Views), and (Temporary) tables.

(b) A (referential integrity constraint) asserts that a value appearing in one table also appears in another related table.

(c) We say one or more attributes is a (key) for a relation if the attributes (functionally determine) all other attributes of the relation, while no subset of the attributes does so.

(d) Sometimes, a relation has more than one key, where one of them is designated as the (primary key).

(e) Redundancies in database design may occur different kinds of anomalies. (Deletion Anomalies) refer to the case when a set of values becomes empty, one may lose other information as a side effect. (Update Anomalies) refer to the case when one change information in one tuple but leave the same information in another tuple unchanged.

(f) An (E/R diagram) is a graph representing entity sets, attributes, and relationships. It is a notation for describing (schemas) of databases.

(g) An (entity) is an abstract object of some sort, and a collection of similar entities forms an (entity set).

(h) (Relationships) are connections among two or more entity sets.

(i) A (key) for an entity set is an attribute or a set of attributes such that, given any two distinct entities, both cannot have identical values for each of the attributes in it.

(j) For (weak) entity sets, the key belongs to another entity set.

**2. Draw the E-R diagrams for the following databases. Be sure to indicate the cardinalities to indicate the multiplicity of a relationship.**

(a) Design a database for a bank, including information about customers and their accounts. Information about a customer includes their name, address, phone, and customer ID. An Account has an account nu mber, balance, and opening date. Also record the customer(s) who own an account. Note that:  
- A customer can have multiple accounts.

- An account can have only one customer.

- Customer ID and Account number are unique to each customer and account, respectively.

|  |
| --- |
| **Customer** |
| name  address  phone  customer ID |

|  |
| --- |
| **Account** |
| account number  balance  opening date |

Own

Own

(b) Modify your original diagram of Problem 2(a) such that a customer can have multiple phone numbers. Recall that we do not allow attributes to have non-primitive data types.

|  |
| --- |
| **Customer** |
| name  address  {phone}  customer ID |

|  |
| --- |
| **Account** |
| account number  balance  opening date |

Own

(c) Change your diagram of Problem 2(b) such that a customer has an address represented by composite attributes (which are street-city-province triplets). Note that multiple customers may live at a single address.

|  |
| --- |
| **Customer** |
| name  address  street  city  province  {phone}  customer ID |

|  |
| --- |
| **Account** |
| account number  balance  opening date |

Own

(d) Convert your solution E-R diagram from Problem 2(c) into database schemas (*e.g.*, TableName(Attribute1, Attributes2, Attribute3)).

Before Normalization

Customer(name, address\_street, address\_city, address\_province, {phone}, customer ID)

Account(account number, balance, opening date)

= After Normalization =>

Customer(name, customer ID)

phones(phone, customer ID)

Account(account number, balance, opening date)

(e) Convert your solution to Problem 2(d) into SQL DDL (CREATE TABLE statements).

**CREATE TABLE** Customer(

name VARCHAR(20) **NOT NULL**,

customer ID CHAR(8),

**PRIMARY KEY**(customer ID));

**CREATE TABLE** phone(

customer ID CHAR(8),

phone CHAR(16) **NOT NULL,**

**FOREIGN KEY**(customer ID) **REFERENCES** Customer);

**CREATE TABLE** Account(

account number VARCHAR (20),

balance VARCHAR(20) **NOT NULL,**

opening date CHAR(16) **NOT NULL,**

**PRIMARY KEY**(account number));

3. Normalization.

(a) Is every relation in 3NF also in BCNF? If yes, explain why. If no, given a counter example.

Answer : no

Counter example

|  |  |  |
| --- | --- | --- |
| Section ID | subject | instructor |
| 101 | C++ | Dr. Cpp |
| 103 | Java | Dr. Java |
| 102 | C++ | Dr. C |
| 104 | Python | Dr. Python |

|  |  |
| --- | --- |
| Student ID | Section ID |
| 218000999 | 101 |
| 218009999 | 103 |
| 218000998 | 102 |
| 21800997 | 104 |
| 21800996 | 101 |

This is a counter example of BCNF shows that every relation in 3NF is not always in BCNF.

subject attributes can indicate the instructor attributes and vise versa, even if they are not Primary key. This is a transitive dependency and it does not allow in 3NF.

(b) Is every relation in 4NF also in BCNF? If yes, explain why. If no, given a counter example.

Answer : yes

Explanation

: 4NF is condition that relation should not have multi-valued dependency. Moreover, it should satisfy the BCNF(=3.5NF) as the base condition. That is the reason every relation in 4NF also in BCNF.

(c) Consider the following relation. Does the relation violate any of the normal forms {1NF, 2NF, 3NF, 4NF, BCNF}? If yes, indicate which of the NFs is violated, explain why it is violated and how it could be fixed.

R1(employee\_id, name, branch, project)

|  |  |  |  |
| --- | --- | --- | --- |
| **employee\_id** | **name** | **branch** | **project** |
| 1001 | Brown | Seoul | Mars, Saturn |
| 1004 | Green | Seoul | Saturn |
| 1005 | Hopkins | Pohang | Jupiter, Saturn |
| 2001 | Kim | Seoul | Mars |
| 3002 | Walker | Seoul | Venus |
| 3004 | Kim | Pohang | Venus |
| 3005 | Clermont | Pohang | Jupiter, Saturn |

1NF is violated, for the multiple data in one domain(ex: Mars, Saturn values are in one attribute).

|  |  |  |
| --- | --- | --- |
| **employee\_id** | **name** | **branch** |
| 1001 | Brown | Seoul |
| 1004 | Green | Seoul |
| 1005 | Hopkins | Pohang |
| 2001 | Kim | Seoul |
| 3002 | Walker | Seoul |
| 3004 | Kim | Pohang |
| 3005 | Clermont | Pohang |

|  |  |
| --- | --- |
| **employee\_id** | **project** |
| 1001 | Mars |
| 1001 | Saturn |
| 1004 | Saturn |
| 1005 | Jupiter |
| 1005 | Saturn |
| 2001 | Mars |
| 3002 | Venus |
| 3004 | Venus |
| 3005 | Jupiter |
| 3005 | Saturn |

(d) Consider the following relation. Does the relation violate any of the normal forms {1NF, 2NF, 3NF, 4NF, BCNF}? If yes, indicate which of the NFs is violated, explain why it is violated and how it could be fixed.

R2(employee\_id, name, position, previous\_branch)

|  |  |  |  |
| --- | --- | --- | --- |
| **employee\_id** | **name** | **position** | **previous\_branch** |
| 1001 | Brown | Sales representative | Pohang |
| 1001 | Brown | Sales representative | Busan |
| 1005 | Hopkins | Software engineer | Seoul |
| 2001 | Kim | Software engineer | Busan |
| 3004 | Kim | Product manager | Seoul |
| 3004 | Kim | Product manager | Wonju |
| 3005 | Clermont | Project administrator | Seoul |

2NF is violated, for the partial dependencies. (we expect to find out the position data by using employee\_id and previous\_branch, but employee\_id can find out the position data)

|  |  |  |
| --- | --- | --- |
| **employee\_id** | **name** | **previous\_branch** |
| 1001 | Brown | Pohang |
| 1001 | Brown | Busan |
| 1005 | Hopkins | Seoul |
| 2001 | Kim | Busan |
| 3004 | Kim | Seoul |
| 3004 | Kim | Wonju |
| 3005 | Clermont | Seoul |

|  |  |
| --- | --- |
| **employee\_id** | **position** |
| 1001 | Sales representative |
| 1005 | Software engineer |
| 2001 | Software engineer |
| 3004 | Product manager |
| 3005 | Project administrator |

(e) Consider the following relation. Does the relation violate any of the normal forms {1NF, 2NF, 3NF, 4NF, BCNF}? If yes, indicate which of the NFs is violated, explain why it is violated and how it could be fixed.

R3(employee\_id, name, branch, branch\_address)

|  |  |  |  |
| --- | --- | --- | --- |
| **employee\_id** | **name** | **branch** | **branch\_address** |
| 1001 | Brown | Seoul | Garosu-gil 233 |
| 1004 | Green | Seoul | Garosu-gil 233 |
| 1005 | Hopkins | Pohang | Handong-ro 501 |
| 2001 | Kim | Seoul | Garosu-gil 233 |
| 3002 | Walker | Seoul | Garosu-gil 233 |
| 3004 | Kim | Pohang | Handong-ro 501 |
| 3005 | Clermont | Pohang | Handong-ro 501 |

2NF is violated, for the partial deependencies. (we expect to find out the branch\_address data by using employee\_id, name, and branch, but employee\_id can find out the branch\_address data)

|  |  |
| --- | --- |
| **employee\_id** | **branch\_address** |
| 1001 | Garosu-gil 233 |
| 1004 | Garosu-gil 233 |
| 1005 | Handong-ro 501 |
| 2001 | Garosu-gil 233 |
| 3002 | Garosu-gil 233 |
| 3004 | Handong-ro 501 |
| 3005 | Handong-ro 501 |

|  |  |  |
| --- | --- | --- |
| **employee\_id** | **name** | **branch** |
| 1001 | Brown | Seoul |
| 1004 | Green | Seoul |
| 1005 | Hopkins | Pohang |
| 2001 | Kim | Seoul |
| 3002 | Walker | Seoul |
| 3004 | Kim | Pohang |
| 3005 | Clermont | Pohang |

**4. In this problem we use the "*sakila*" (DVD rental) database, which consists of 16 tables regarding movie inventory, actors, customers, rental history, payment information, *etc*. For each of the following questions, find answer based on the information recorded in the database and write a query that shows how you obtained the answer.**

(a) How many *stores* are found in the database?

Query:

SELECT *COUNT*(DISTINCT store\_id) Store\_Num  
FROM store;

Answer:

검은색, 하얀색, 표지판, 공이(가) 표시된 사진

자동 생성된 설명

(b) How many unique *last names* are found in the *actor* relation?

Query:

SELECT *COUNT*(DISTINCT last\_name) Last\_Name  
FROM actor;

Answer:

검은색, 쥐고있는, 여자, 하얀색이(가) 표시된 사진

자동 생성된 설명

(c) According to the database, how many *inventories* (DVDs) have not been returned (the inventories that have not been returned do not have *return\_date*)?

Query:

SELECT *COUNT*(DISTINCT inventory\_id) Not\_Returned  
FROM rental  
WHERE return\_date IS NULL;

Answer:

검은색, 하얀색, 플레이어, 쥐고있는이(가) 표시된 사진

자동 생성된 설명

(d) How many distinct *customer*s have rented a movie from *staff\_id*=1?

Query:

SELECT *COUNT*(DISTINCT customer\_id) Customer\_Num  
FROM rental  
WHERE staff\_id = 1;

Answer:

공, 플레이어, 하얀색, 그리기이(가) 표시된 사진

자동 생성된 설명

(e) How many distinct *film*s rated 'PG' are available?

Query:

SELECT *COUNT*(DISTINCT film\_id) Film\_Num  
FROM film  
WHERE rating = 'PG';

Answer:

검은색, 시계, 쥐고있는, 하얀색이(가) 표시된 사진

자동 생성된 설명

(f) List *three* most frequent *categories* of film available at *store\_id*=2 (if a store has multiple copies of the same film, consider each copy as an individual inventory).

Query:

SELECT name, C.category\_id AS CID, *COUNT*(FC.film\_id) AS D  
FROM film\_category AS FC  
 JOIN category AS C  
 ON FC.category\_id = C.category\_id,  
 (SELECT film\_id, *COUNT*(DISTINCT inventory\_id) AS A  
 FROM inventory  
 WHERE store\_id = 2  
 GROUP BY film\_id ORDER BY A DESC) AS INVEN  
WHERE FC.film\_id = INVEN.film\_id  
GROUP BY CID ORDER BY D DESC LIMIT 3;

Answer:

시계, 측정기, 디스플레이, 표지판이(가) 표시된 사진

자동 생성된 설명

(g) How many *active* *customer*s are living in the *district* of 'England'?

Query:

SELECT *COUNT*(customer\_id) AS Total  
FROM customer AS C JOIN address AS A ON C.address\_id = A.address\_id  
WHERE C.active = 1 AND A.district = 'England';

Answer:

공, 플레이어, 쥐고있는, 시계이(가) 표시된 사진

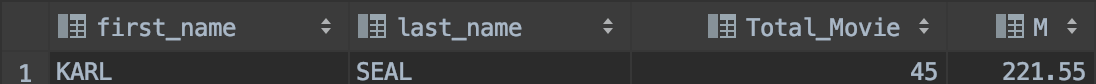
자동 생성된 설명

(h) Considering the rental history (*rental*) and payment history (*payment*), who has paid the largest amount of money for renting movies? List the *first* and *last name* of the *customer*, the total number of movie rentals, and total amount of money s/he has paid.

Query:

SELECT C.first\_name, C.last\_name, P.Total\_Movie, P.Rental\_Sum  
FROM customer AS C JOIN  
 (SELECT customer\_id, *COUNT*(rental\_id) AS Total\_Movie, *SUM*(amount) AS Rental\_Sum  
 FROM payment  
 GROUP BY customer\_id ORDER BY Rental\_Sum DESC) AS P  
 ON C.customer\_id = P.customer\_id  
ORDER BY P.Rental\_Sum DESC LIMIT 1;

Answer:

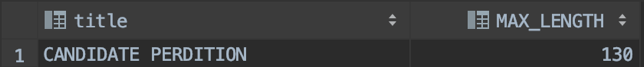


(i) What is the *title* of the movie that has the longest *description* (*film\_text.description*) among the rental store with *store\_id*=2 has?

Query:

SELECT FT.title, *MAX*(*CHAR\_LENGTH*(FT.description)) AS MAX\_LENGTH  
FROM film\_text AS FT JOIN  
 (SELECT DISTINCT film\_id  
 FROM inventory  
 WHERE store\_id = 2) AS INVEN  
 ON FT.film\_id = INVEN.film\_id  
GROUP BY FT.title ORDER BY MAX\_LENGTH DESC LIMIT 1;

Answer:



(j) Which of the *film*s starred by 'FRED COSTNER' rented the most? Write the *title* of the film.

Query:

SELECT F.title, *COUNT*(R.rental\_id) AS T  
FROM film AS F JOIN (SELECT FA.film\_id  
 FROM actor AS A, film\_actor AS FA  
 WHERE first\_name = 'FRED' AND last\_name = 'COSTNER'  
 AND A.actor\_id = FA.actor\_id) AS FID ON F.film\_id = FID.film\_id,  
 rental AS R JOIN inventory AS I ON R.inventory\_id = I.inventory\_id  
WHERE I.film\_id = F.film\_id  
GROUP BY F.title ORDER BY T DESC LIMIT 1;

Answer:

그리기이(가) 표시된 사진

자동 생성된 설명

(k) Using the '*customer\_list*' view, list all *name*s of people whose *address* is in the *city* of 'London'.

Query:

CREATE VIEW Customer\_List AS  
 SELECT CT.first\_name, CT.last\_name, CT.customer\_id  
 FROM customer AS CT, address AS A, city AS C  
 WHERE C.city\_id = A.city\_id  
 AND A.address\_id = CT.address\_id  
 AND C.city = 'London';  
  
SELECT CL.customer\_id, CL.first\_name, CL.last\_name  
FROM Customer\_List AS CL;

Answer:

검은색, 쥐고있는, 하얀색, 플레이어이(가) 표시된 사진

자동 생성된 설명

(l) Write a query that uses only tables (does not use any views) and returns the same information as in Problem 4(k).

Query:

SELECT customer\_id, first\_name, last\_name  
FROM customer AS CT JOIN  
 (SELECT address\_id  
 FROM address AS A, city AS C  
 WHERE C.city = 'London'  
 AND A.city\_id = C.city\_id) AS AC ON AC.address\_id = CT.address\_id;

검은색, 쥐고있는, 하얀색, 플레이어이(가) 표시된 사진

자동 생성된 설명