W4111 Intro. To Databases
Section 02, Fall 2020 Homework 2

Name: Haoran Zhu
UNI: hz2712

Overview

This document has two sections for HW 2:

1. Written questions and problems from covering concepts from the class. (5 points)
2. Problems in relational algebra and SQL. (5 points)

There is no difference between Programming and Non-Programming for Homework 2.

Written Questions and Problems

Views (1)

Question:

1. List 3 benefits of defining views in a relational database.
2. What would be a scenario in which I would make a copy of the data instead of defining a view?

Answers:

1.
The first benefit is security. Views can be made accessible to users while the underlying tables are not directly accessible. Users only see the data they need and other data in the same table is protected.
Second benefit is simplicity. Views make querying logical model much simpler by hiding and reusing complex queries
Third benefit is the isolation between physical and logical schema. The database contains only the definition of a view, not a copy of all the data that it presents

2.
When the data is rarely updated but queried very frequently. In this case, making a copy the data speeds up the query significantly. Since the data is rarely updated, we won't speed too much time maintaining the consistency of the copied data.

Keys (2)

Question:

1. MySQL and other databases support the concept of a *uniqueness constraint.* How does this differ from a primary key constraint?

2. Briefly explain *compound key* and *composite key* and give an example of each using the example database schema associated with the textbook. If there are no examples for one or both, please state it.

Answers:
1.
Under the uniqueness constraint, data can be NULL. However, primary key constraint does not allow NULL data. Also, uniqueness constraint does not imply minimality while primary key constraint does.
2.
A composite key is a candidate key that consists of two or more attributes that together uniquely identify an entity occurrence (table row). Note that a composite key is minimal. A compound key is a composite key for which each attribute that makes up the key is a foreign key in its own right.
In textbook's example database schema, the section table includes following attributes: course_id, sec_id, semester, and year. These four attributes combined form a composite key that uniquely identify a table row. Also, this composite key is minimal. The prereq table has two attributes – course_id and prereq_id. These two attributes together form a composite key. Also, both of these attributes are foreign key that references the course table.

Schema and Keys (3)

Question:

branch(*branch_name*, *branch_city*, *assets*)
customer (*ID*, *customer_name*, *customer_street*, *customer_city*)
loan (*loan_number*, *branch_name*, *amount*)
borrower (*ID*, *loan_number*)
account (*account_number*, *branch_name*, *balance*)
depositor (*ID*, *account_number*)

Figure 2.18 Bank database.

Consider the bank database of Figure 2.18. Assume that branch names and customer names uniquely identify branches and customers, but loans and accounts can be associated with more than one customer.

1. What are the appropriate primary keys?
2. Given your choice of primary keys, identify appropriate foreign keys.

Answers:
1.
Branch: branch_name (according to the prompt)
Customer: ID (although customer_name can also uniquely identify customers, ID is a better choice for primary key as it will be used by other tables as foreign keys)
Loan: loan_number
Borrower: ID, loan_number

Account: accounr_number
Depositer: ID, account_number
2.
Loan: branch_name which references branch
Borrower: ID which references customer; loan_number which references loan
Account: branch_name which references branch
Depositor: ID which references customer; account_name which references account

Codd's Rules (4)

Question

Consider Codd's 12 rules.

Rule 3: *Systematic treatment of null values:*

Null values (distinct from the empty character string or a string of blank characters and distinct from zero or any other number) are supported in fully relational DBMS for representing missing information and inapplicable information in a systematic way, independent of data type.

Question: Briefly explain the importance of Rule 3 and some examples of how SQL realizes the rule.

Answer:
To know which value indicates not applicable or unknown would require understanding all domains and carefully writing SQL. By following Rule 3, we don't need the pre-understanding of the domain that the data is about to determine an appropriate value for missing/inapplicable information, which simplifies the process of designing database.
In SQL, null is propagated through mathematic operations and string operations. During comparison, null will not be treated as a normal value. Also, SQL has IS NULL and IS NOT NULL operators to test if a value is null. We are allowed to set the default value to null when declaring a table, regardless of the data type. Moreover, there are lots of functions in SQL designed specifically to handle null values such as IFNULL().

Rule 4*: Dynamic online catalog based on the relational model:*

The database description is represented at the logical level in the same way as ordinary data, so that authorized users can apply the same relational language to its interrogation as they apply to the regular data.

Question: In two or three sentences, explain what this means. Give two example queries from MySQL that show how MySQL realizes the concepts. You do not need to execute the query.

Answer:
Authorized users must be able to access the database's structure (catalog) using the same query language that they use to access the database's data.

Example 1: *select column_name from information_schema.columns*
where table_schema='lahmansbaseballdb2019'
Example 2: *select count(*) from information_schema.tables*
where table_name='appearances'

View Updates (5)

Question:

Use the Lahman 2019 schema for this question.

- Define a view for which *update* through the view is possible.
- Define a view for which *update* through the view is NOT possible.

Answers:

No need for SQL queries, a written explanation is fine.
Update is possible.
Consider following SQL query:
*create People_view as (select * from lahmansbaseballdb2019.People)*
The newly created People_view has exactly the same content as People table. In this case, update through the view is possible.

Update is NOT possible
Consider following SQL query:
create AwardWinner as (select playerID, count(*) from AwardsPlayers group by playerID)
The newly created AwardWinner view contains the number of times each player has won an award. This counting information was not present in the base table – AwardsPlayers. Updating through the view is NOT possible in this case.

Relational Algebra and SQL

Note: Use the same data model (sample university) that comes with the recommended textbook for these questions. You should have loaded the MySQL model into your database already. You can find the schema and instructions in a lecture. You will use the online relational calculator tool for the relational algebra questions. For the online relational calculator, choose the "Karlsruhe University of Applied Sciences" dataset.

Anti-Join (1)

Question:

"An anti-join is a form of join with reverse logic. Instead of returning rows when there is a match (according to the join predicate) between the left and right side, an anti-join returns those rows

from the left side of the predicate for which there is no match on the right." The Anti-Join Symbol is ▷.

Consider the following relational algebra expression and result.

/* (1) Set X = The set of classrooms in buildings Taylor or Watson. */
X = σ building='Watson' ∨ building='Taylor' (classroom)

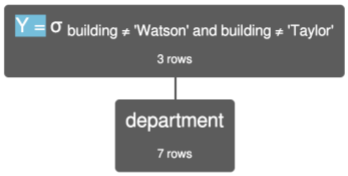
/* (2) Set Y = The Anti-Join of department and X */
Y = (department ▷ X)

/* (3) Display the rows in Y. */
Y

department.dept_name	department.building	department.budget
'Finance'	'Painter'	120000
'History'	'Painter'	50000
'Music'	'Packard'	80000

a) Find an alternate expression to (2) that computes the correct answer given X. Display the execution of your query below.
b) Write an equivalent SQL script (set of statements).

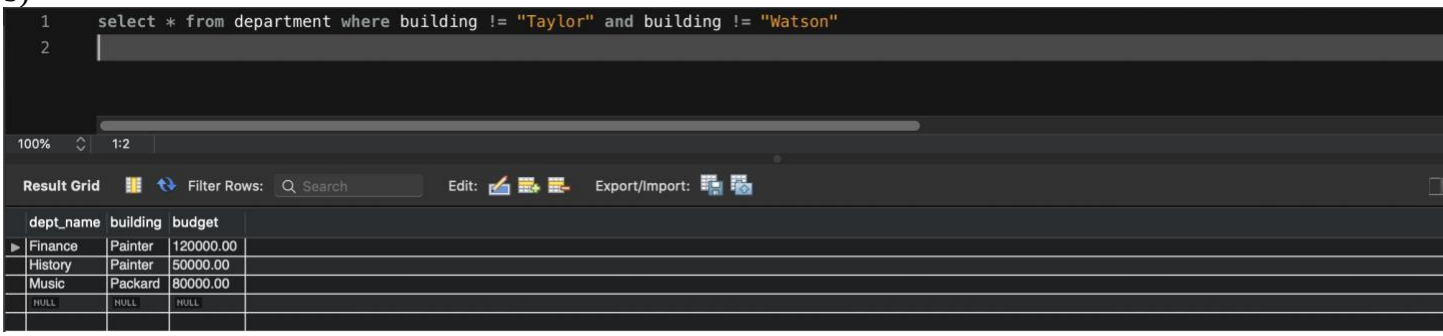
Answers:
a)
Y = σ building ≠ 'Watson' ∧ building ≠ 'Taylor' (department)



σ building ≠ 'Watson' and building ≠ 'Taylor' (department)

department.dept_name	department.building	department.budget
'Finance'	'Painter'	120000
'History'	'Painter'	50000
'Music'	'Packard'	80000

b)



Complex SQL (2)

Professor Ferguson's opinion is that a baseball player is a candidate for the Hall of Fame if:

1. For career batting statistics:
    a. The player has more than 500 home runs or
    b. The player
        i. Has more than 3,000 career at bats and
        ii. Has a career on-base percentage of 0.400 or better.
2. For pitching:
    a. The player has more than 300 career wins or
    b. The player has:
        i. More than 300 total decisions and
        ii. Has a winning percentage greater than or better than 0.666

The relevant columns are:

- Pitching:
    - W is wins.
    - L is loses.

    - Decisions is L + W.
- Batting:
    - AB is at bats.
    - H is hits.
    - BB is walks (bases on balls).
    - HBP is hit by pitch.
    - On-base percentage is (H + BB + HBP)/(AB + BB + HBP)
    - HR is home runs.

Produce the following table, which is Professor Ferguson's list of players that are candidates for the Hall of Fame.

Hint: Creating a view for batting and a view for pitching helps.

Sample Query Result:

playerid	nameLast	nameFirst	total_abs	total_hits	total_bb	obp	total_hrs	total_wins	total_loses	total_decisio...	winning_percentage	games_pitched
aaronha01	Aaron	Hank	12364	3771	1402	0.3772	755	0	0	0	0.0000	0
alexape01	Alexander	Pete	1810	378	77	0.2419	11	373	208	581	0.6420	696
bagweje01	Bagwell	Jeff	7797	2314	1401	0.4121	449	0	0	0	0.0000	0
bankser01	Banks	Ernie	9421	2583	763	0.3331	512	0	0	0	0.0000	0
berkmla01	Berkman	Lance	6491	1905	1201	0.4089	366	0	0	0	0.0000	0
bishoma01	Bishop	Max	4494	1216	1153	0.4227	41	0	0	0	0.0000	0
bluelu01	Blue	Lu	5904	1696	1092	0.4022	44	0	0	0	0.0000	0
boggswa01	Boggs	Wade	9180	3010	1412	0.4187	118	0	0	0	0.0000	2
bondsba01	Bonds	Barry	9847	2935	2558	0.4475	762	0	0	0	0.0000	0
broutda01	Brouthers	Dan	6726	2303	840	0.4234	107	0	2	2	0.0000	4
brownpe01	Browning	Pete	4820	1646	466	0.4028	46	0	1	1	0.0000	1
burkeje01	Burkett	Jesse	8426	2850	1029	0.4149	75	3	11	14	0.2143	23
carltst01	Carlton	Steve	1719	346	41	0.2243	13	329	244	573	0.5742	741
carutbo01	Caruthers	Bob	2465	695	417	0.3909	29	218	99	317	0.6877	340
childcu01	Childs	Cupid	5622	1721	991	0.4157	20	0	0	0	0.0000	0
clarkjo01	Clarkson	John	1974	432	81	0.2500	24	328	178	506	0.6482	531
clemero02	Clemens	Roger	179	31	13	0.2371	0	354	184	538	0.6580	709
cobbty01	Cobb	Ty	11436	4189	1249	0.4329	117	0	0	0	0.0000	3
cochrmi01	Cochrane	Mickey	5169	1652	857	0.4192	119	0	0	0	0.0000	0
collied01	Collins	Eddie	9949	3315	1499	0.4244	47	0	0	0	0.0000	0
cullero01	Cullenbine	Roy	3879	1072	853	0.4082	110	0	0	0	0.0000	0

Answer:

```
create or replace view battingView as
(
   select * from
        (
                select playerID, sum(AB) as total_abs, sum(H) as total_hits, sum(BB) as total_bb,

ifnull(sum(ifnull(H,0)+ifnull(BB,0)+ifnull(HBP,0))/sum(ifnull(AB,0)+ifnull(BB,0)+ifnull(HBP,0
)),0) as obp,
      sum(HR) as total_hrs
      from Batting group by playerID
   ) as a
);

create or replace view pitchingView as
(
```

```
        select * from
    (
            select playerID, sum(W) as total_wins, sum(L) as total_loses, sum(W + L) as
total_decisions,
            round(ifnull(sum(W)/sum(W+L),0),4)as winning_percentage, sum(G) as
games_pitched
            from Pitching group by playerID
        ) as a
);

select * from (select playerID, nameLast, nameFirst from People) as a natural join
(
        select playerID, total_abs, total_hits, total_bb, obp, total_hrs,
        ifnull(total_wins, 0) as total_wins, ifnull(total_loses, 0) as total_loses,
ifnull(total_decisions, 0) as total_decisions,
        ifnull(winning_percentage, 0) as winning_percentage, ifnull(games_pitched, 0) as
games_pitched
        from
        (
            select * from battingview left join pitchingview using (playerID)
        ) as b
        where (b.total_hrs > 500 or (b.total_abs > 3000 and b.obp >= 0.4))
        or (b.total_wins > 300 or (b.total_decisions > 300 and b.winning_percentage >= 0.666))
) as c order by c.playerID;
```

playerID	nameLast	nameFirst	total_abs	total_hits	total_bb	obp	total_hrs	total_wins	total_loses	total_decisions	winning_percentage	games_pitched
aaronha01	Aaron	Hank	12364	3771	1402	0.3772	755	0	0	0	0.0000	0
alexape01	Alexander	Pete	1810	378	77	0.2419	11	373	208	581	0.6420	696
bagweje01	Bagwell	Jeff	7797	2314	1401	0.4121	449	0	0	0	0.0000	0
bankser01	Banks	Ernie	9421	2583	763	0.3331	512	0	0	0	0.0000	0
berkmla01	Berkman	Lance	6491	1905	1201	0.4089	366	0	0	0	0.0000	0
bishoma01	Bishop	Max	4494	1216	1153	0.4227	41	0	0	0	0.0000	0
bluelu01	Blue	Lu	5904	1696	1092	0.4022	44	0	0	0	0.0000	0
boggswa01	Boggs	Wade	9180	3010	1412	0.4187	118	0	0	0	0.0000	2
bondsba01	Bonds	Barry	9847	2935	2558	0.4475	762	0	0	0	0.0000	0
broutda01	Brouthers	Dan	6726	2303	840	0.4234	107	0	2	2	0.0000	4
brownpe01	Browning	Pete	4820	1646	466	0.4028	46	0	1	1	0.0000	1
burkeje01	Burkett	Jesse	8426	2850	1029	0.4149	75	3	11	14	0.2143	23
carltst01	Carlton	Steve	1719	346	41	0.2243	13	329	244	573	0.5742	741
carutbo01	Caruthers	Bob	2465	695	417	0.3909	29	218	99	317	0.6877	340
childcu01	Childs	Cupid	5622	1721	991	0.4157	20	0	0	0	0.0000	0
clarkjo01	Clarkson	John	1974	432	81	0.2500	24	328	178	506	0.6482	531
clemero02	Clemens	Roger	179	31	13	0.2371	0	354	184	538	0.6580	709
cobbty01	Cobb	Ty	11436	4189	1249	0.4329	117	0	0	0	0.0000	3
cochrmi01	Cochrane	Mickey	5169	1652	857	0.4192	119	0	0	0	0.0000	0
collied01	Collins	Eddie	9949	3315	1499	0.4244	47	0	0	0	0.0000	0
cullero01	Cullenbine	Roy	3879	1072	853	0.4082	110	0	0	0	0.0000	0
cunnijo01	Cunning...	Joe	3362	980	599	0.4060	64	0	0	0	0.0000	0
delahed01	Delahanty	Ed	7510	2597	742	0.4113	101	0	0	0	0.0000	0
fainfe01	Fain	Ferris	3930	1139	904	0.4248	48	0	0	0	0.0000	0
fordwh01	Ford	Whitey	1023	177	113	0.2572	3	236	106	342	0.6901	498
foxxji01	Foxx	Jimmie	8134	2646	1452	0.4283	534	1	0	1	1.0000	10
galvipu01	Galvin	Pud	2748	552	38	0.2123	5	365	310	675	0.5407	705
gehrich01	Gehringer	Charlie	8860	2839	1186	0.4036	184	0	0	0	0.0000	0
gehrilo01	Gehrig	Lou	8001	2721	1508	0.4474	493	0	0	0	0.0000	0
giambja01	Giambi	Jason	7267	2010	1366	0.4035	440	0	0	0	0.0000	0
gilesbr02	Giles	Brian	6527	1897	1183	0.4034	287	0	0	0	0.0000	0
glavito02	Glavine	Tom	1323	246	101	0.2447	1	305	203	508	0.6004	682
greenha01	Greenberg	Hank	5193	1628	852	0.4118	331	0	0	0	0.0000	0
griffke02	Griffey	Ken	9801	2781	1312	0.3729	630	0	0	0	0.0000	0
grovele01	Grove	Lefty	1369	202	105	0.2093	15	300	141	441	0.6803	616
hamilbi01	Hamilton	Billy	6283	2164	1189	0.4552	40	0	0	0	0.0000	0