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.

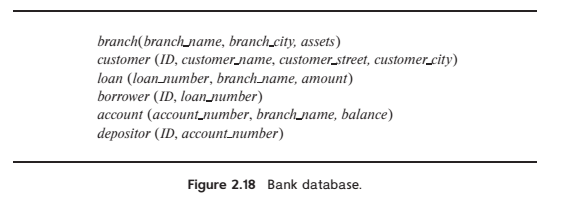
2.

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:



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.](https://en.wikipedia.org/wiki/Codd%27s_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](https://dbis-uibk.github.io/relax/landing) 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

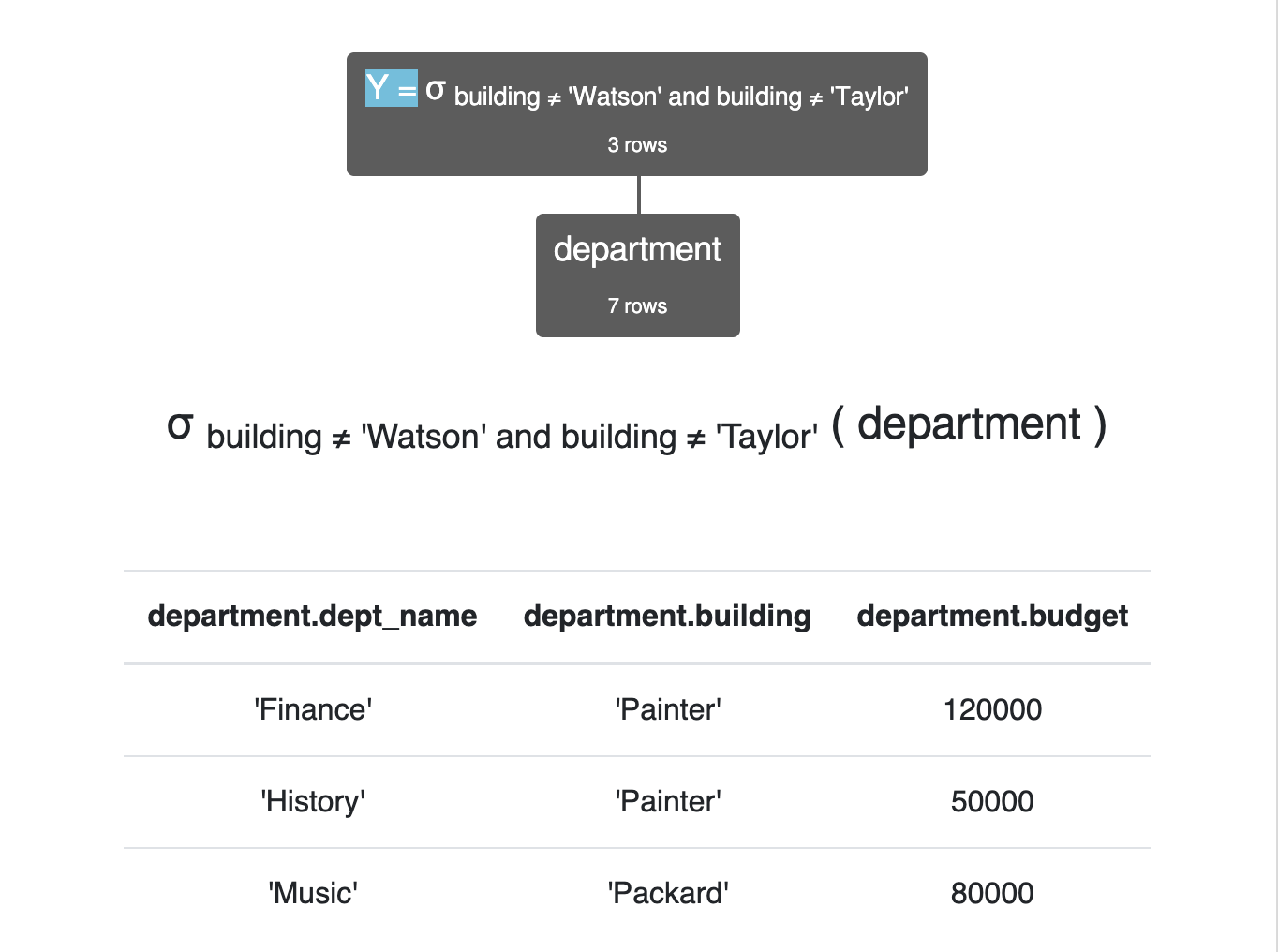
|  |  |  |
| --- | --- | --- |
| **department.dept\_name** | **department.building** | **department.budget** |
| 'Finance' | 'Painter' | 120000 |
| 'History' | 'Painter' | 50000 |
| 'Music' | 'Packard' | 80000 |

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

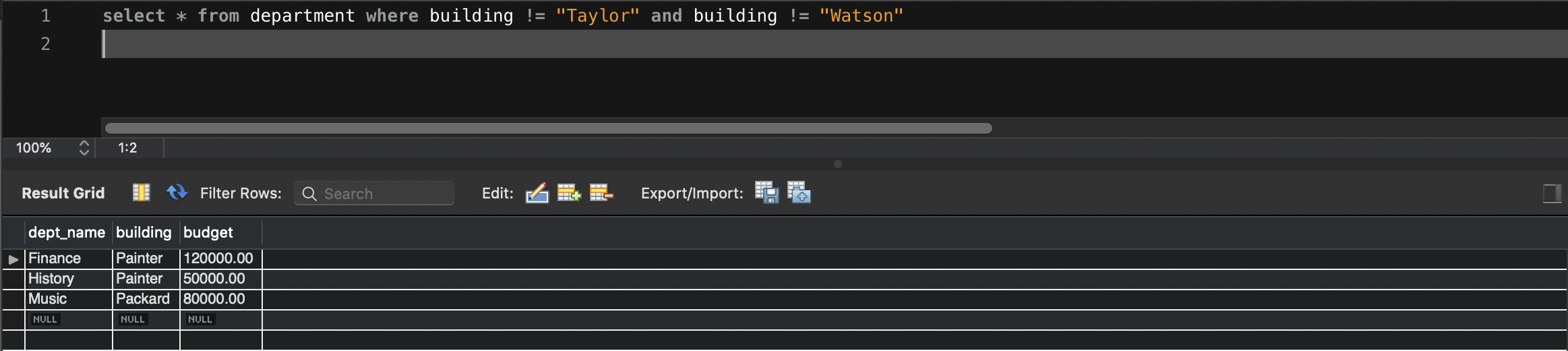
Answers:

a)

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



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

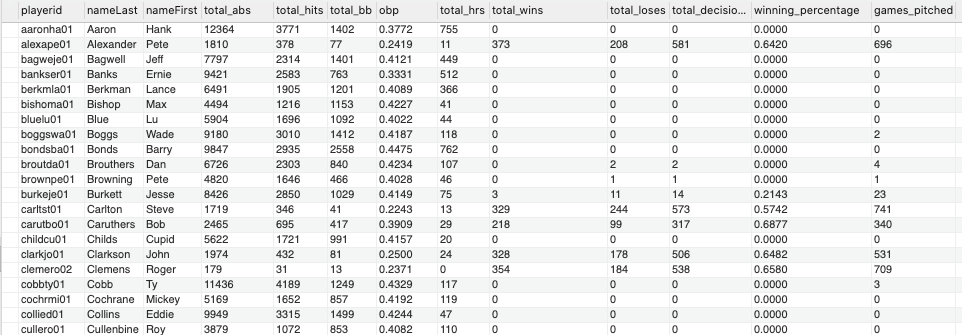
The relevant columns are:

* 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:



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;*

