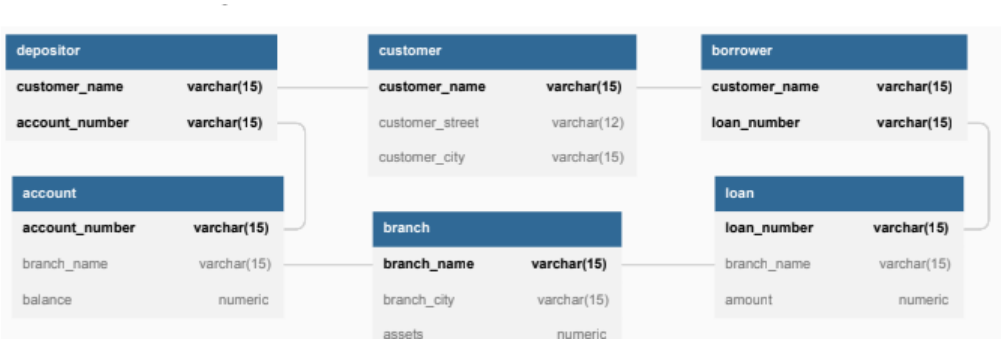
Database Management Systems Lab
Lab 4
CSE 4308

A Z Hasnain Kabir
200042102
Software Engineering

14 September 2022

Given database and files

We are given a database with 6 tables which has the following schema. We are also provided with a *banking.sql* file which contains the values we need to insert inside our given tables. Here, the boldfaces denote the primary keys and the arcs denote the foreign key relationships. In this lab, we have to write all SQL statements in an editor first and save them with *.sql* extension.



Tasks

Write SQL statements to execute given queries.

Task 1

Find all customer names who have an account as well as a loan (with and without 'intersect' clause).

Analysis of the Problem

The tables depositor and borrower contains the information related to accounts and loans of a customer as well as the customer names. To find out the names of the customers, we need to work with these two tables.

Code

Without using INTERSECT

```
SELECT distinct depositor.customer_name FROM depositor,borrower
WHERE depositor.customer_name = borrower.customer_name;
```

Using INTERSECT

```
SELECT customer_name FROM depositor
INTERSECT
SELECT customer_name FROM borrower;
```

Explanation of the Solution

If we want to write queries without using INTERSECT clause, we find cartesian product of depositor and borrower and check if customer names in both tables are equal. If we use INTERSECT clause, we select customer name from both tables and do an intersection with the two. Intersection returns rows only from the first SELECT statement that are identical to a row in the second SELECT statement.

Task 2

Find all customer-related information who have an account or a loan (with and without 'union' clause).

Analysis of the Problem

The tables depositor and borrower contains the information related to accounts and loans of a customer and the customer table contains information related to customers. To find out the names of the customers with accounts or loans, we need to work with these three tables.

Code

Without using UNION

```
SELECT distinct customer.customer_name,customer_city,customer_street
FROM customer,depositor,borrower
WHERE customer.customer_name = depositor.customer_name OR
customer.customer_name = borrower.customer_name ;
```

Using UNION

```
SELECT customer.customer_name,customer_street,customer_city
↪  FROM customer,borrower
WHERE customer.customer_name = borrower.customer_name
UNION
SELECT customer.customer_name,customer_street,customer_city
↪  FROM customer,depositor
WHERE customer.customer_name = depositor.customer_name;
```

Explanation of the Solution

If we want to write queries without using UNION clause, we find cartesian product of depositor, customer and borrower and check if customer names in both tables are equal. If we use UNION clause, we select customer name after cartesian product of customer, borrower and UNION it with cartesian product of cusomer and depositor tables.

Task 3

Find all customer-related information who have an account or a loan (with and without 'union' clause).

Analysis of the Problem

If we don't use MINUS, we use the NOT IN clause to specify that the records belong to one table and not the other

Code

Without using MINUS

```
SELECT distinct customer.customer_name, customer_city
FROM customer,borrower,depositor
WHERE customer.customer_name = borrower.customer_name AND
customer.customer_name NOT IN (
SELECT distinct customer.customer_name
FROM customer,borrower,depositor
WHERE customer.customer_name = borrower.customer_name AND
↪  customer.customer_name = depositor.customer_name);
```

Using MINUS

```
SELECT distinct customer.customer_name, customer_city
FROM customer, borrower
WHERE customer.customer_name = borrower.customer_name
MINUS
SELECT distinct customer.customer_name, customer_city
FROM customer, depositor
WHERE customer.customer_name = depositor.customer_name;
```

Explanation of the Solution

MINUS returns rows that are present in one table but not in another. Also, using NOT IN clause does the same thing without using MINUS.

Task 4

Find the total assets of all branches.

Analysis of the Problem

We only have to work with branch table.

Code

```
SELECT sum(assets) FROM branch;
```

Explanation of the Solution

sum() returns the summation of the entire column.

Task 5

Find the total number of accounts at each branch city.

Analysis of the Problem

We need to work with tables branch and account.

Code

```
SELECT branch_city,count(account_number) FROM branch,
account WHERE account.branch_name = branch.branch_name
GROUP BY branch_city;
```

Explanation of the Solution

group by() categorizes records based on another column. count() function returns a value after counting the entire column.

Task 6

Find the average balance of accounts at each branch and display them in descending order of average balance.

Analysis of the Problem

account table contains all the information we need to execute the operation.

Code

```
SELECT  branch_name,avg(balance) FROM account
GROUP BY branch_name ORDER BY avg(balance) DESC;
```

Explanation of the Solution

GROUP BY() categorizes records based on another column. avg() function returns average value after counting the entire column. ORDER BY orders the column according to ascending or descending order.

Task 7

Find the total balance of accounts at each branch city.

Analysis of the Problem

account and branch tables contains all the information we need to execute the operation.

Code

```
SELECT branch_city,sum(balance) FROM account,
branch WHERE branch.branch_name = account.branch_name
GROUP BY branch_city;
```

Explanation of the Solution

GROUP BY() categorizes records based on another column. sum() funtion returns sum of the entire column of a table.

Task 8

Find the average loan amount at each branch. Do not include any branch which is located in 'Horseneck' city (with and without 'having' clause). 'union' clause).

Analysis of the Problem

We have to group loan and branch tables and find out if any name of a city is called 'Horseneck'.

Code

Without using HAVING

```
SELECT loan.branch_name, avg(amount) FROM loan
,branch WHERE branch.branch_name = loan.branch_name AND
branch_city != 'Horseneck'
GROUP BY loan.branch_name;
```

Using HAVING

```
SELECT avg(amount), max(branch.branch_name)
FROM branch,loan
WHERE branch.branch_name=loan.branch_name
GROUP BY branch.branch_city
HAVING branch.branch_city!='Horseneck';
```

Explanation of the Solution

HAVING clause is a substitute of WHERE clause in case of a GROUP BY operation.

Task 9

Find the customer name and account number of the account which has the highest balance (with and without 'all' clause).

Analysis of the Problem

We can extract the required information from the tables customer and account.

Code

Without using ALL

```
SELECT customer_name, account.account_number FROM
 depositor, account WHERE depositor.account_number
= account.account_number AND account.balance IN (SELECT
↪  max(balance)
FROM account);
```

Using ALL

```
SELECT customer_name, account.account_number FROM
 depositor, account WHERE depositor.account_number
= account.account_number AND account.balance = all (SELECT
↪  max(balance)
FROM account);
```

Explanation of the Solution

ALL clause is used to select all tuples of a SELECT statement.

Task 10

Find all customer-related information who have an account in a branch, located in the same city as they live.

Analysis of the Problem

We have to extract information from four tables customer, branch, account, depositor

Code

```
SELECT distinct c.customer_name, c.customer_city,
↪  c.customer_street
FROM customer c, branch b, account a,depositor d
WHERE c.customer_city = b.branch_city
AND d.account_number = a.account_number
AND d.customer_name = c.customer_name AND a.branch_name =
↪  b.branch_name;
```

Explanation of the Solution

We find the cartesian product of four tables and then use multiple boolean operations to find our required information.

Task 11

For each branch city, find the average amount of all the loans opened in a branch located in that branch city. Do not include any branch city in the result where the average amount of all loans opened in a branch located in that city is less than 1500. (with and without using 'having' clause).

Analysis of the Problem

We can extract the required information from the tables branch and loan.

Code

Without using HAVING

```
SELECT * FROM
(SELECT branch_city, avg(amount) as av FROM branch b, loan l
↪  WHERE b.branch_name = l.branch_name
GROUP BY branch_city)A WHERE A.av>1500;
```

Using HAVING

```
SELECT customer_name, account.account_number FROM
 depositor, account WHERE depositor.account_number
= account.account_number AND account.balance = all (SELECT
↪  max(balance)
FROM account);
```

Explanation of the Solution

For query without the WHERE clause we use a subquery and a table alias A to find our required information.

Task 12

Find those branch names which have a total account balance greater than the average total balance among all the branches.

Analysis of the Problem

account and branch tables contains all the information we need to execute the operation.

Code

```
SELECT distinct b.branch_name FROM branch b, account a
WHERE b.branch_name = a.branch_name GROUP BY
b.branch_name HAVING sum(balance) > avg(balance);
```

Explanation of the Solution

In our query, after the HAVING clause, we use two aggregate functions sum() and avg() to find sum of balance and average balance.

Task 13

Find the name of the customer who has at least one loan that can be paid off by his/her total balance.

Analysis of the Problem

We need to use customer, borrower, loan, account tables to find the name of the customers

Code

```
SELECT  d.customer_name, max(a.balance), min(l.amount) FROM
↪  depositor d,account a, borrower b, loan l
WHERE d.customer_name = b.customer_name AND d.account_number =
↪  a.account_number
AND b.loan_number = l.loan_number GROUP BY (d.customer_name)
HAVING sum(balance) > min(amount);
```

Explanation of the Solution

We use multiple aggregate functions, use GROUP BY operations and HAVING clause to check if sum of balance is greater than minimum amount of loan for each customer. Those customers are able to pay off at least one loan.