Islamic University of Technology (IUT)

Report on Lab 04

Submitted By

Shanta Maria (ID:200042172)

CSE 4308 Database Management Systems Lab

Submitted To

Md. Bakhtiar Hasan

Lecturer, Department of CSE

Zannatun Naim Srsity

Lecturer, Department of CSE

October 10, 2022

Introduction

In this lab class, we were given tasks based on advanced data manipulation techniques to solve using SQL command line to understand the basics of data definition and data manipulation. The given .sql file named banking.sql was executed before doing the following tasks.

1 Task 1

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

1.1 Solution

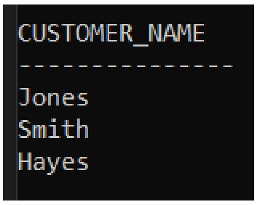
```
--number01(without intersect)--
select distinct customer.customer_name
from customer, depositor, borrower
where customer.customer_name=depositor.customer_name and
customer.customer_name=borrower.customer_name;

--number01(with intersect)--
select depositor.customer_name
from customer, depositor
where customer.customer_name=depositor.customer_name
intersect
select borrower.customer_name
from borrower, customer
where customer.customer_name=borrower.customer_name;
```

1.2 Analysis and Explanation

For the first part, I used the keyword `distinct` to remove any duplicate records and checked if the customer exists in both depositor table (for account) and borrower table (for loan).
For the next part, I learned a new keyword `intersect`. It shows the result like a set intersection where the result is the common records from 2 queries. Here, the first query gives the customers with accounts while the second query gives the customers with loans. So intersecting the queries would give all the customers with both accounts and loans.

1.3 Difficulties

I did not face any difficulties when doing this task except understanding how the new keyword `intersect` works.



```
CUSTOMER_NAME
---------------
Jones
Smith
Hayes
```

Figure 1: Output for Task 1

2 Task 2

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

2.1 Solution

```
--number02(without union)--
```

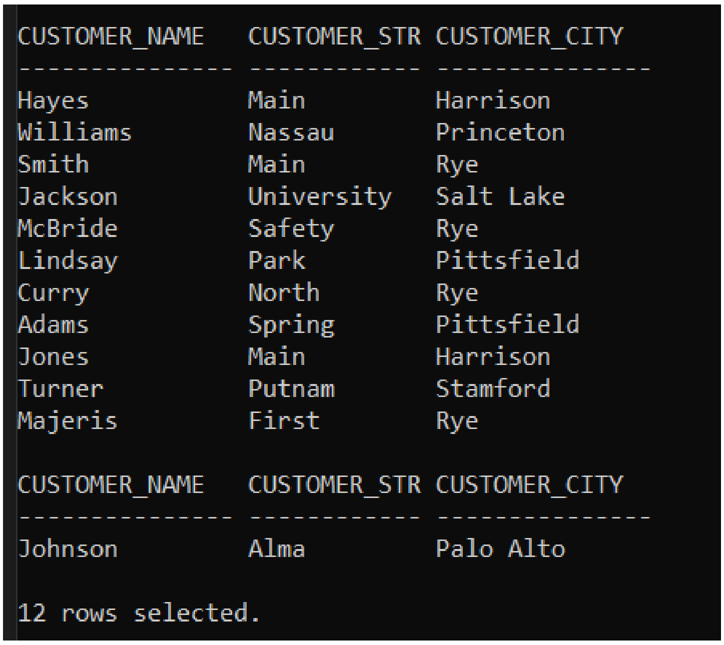
```
select distinct customer.customer_name, customer.customer_street,
customer.customer_city
from customer, depositor, borrower
where customer.customer_name=depositor.customer_name or
customer.customer_name=borrower.customer_name;

--number02(with union)--
select customer.customer_name, customer.customer_street,
customer.customer_city
from customer, depositor
where customer.customer_name=depositor.customer_name
union
select customer.customer_name, customer.customer_street,
customer.customer_city
from customer, borrower
where customer.customer_name=borrower.customer_name;
```

2.2 Analysis and Explanation

For the first part, I used the keyword `distinct` to remove any duplicate records and checked if the customer exists in either depositor table (for account) or borrower table (for loan).
For the next part, I learned a new keyword `union`. It shows the result like a set union where the result is all the records from 2 queries. Here, the first query gives the customers with accounts while the second query gives the customers with loans. So the union of the queries would give all the customers with either accounts or loans.

2.3 Difficulties

I did not face any difficulties when doing this task except understanding how the new keyword `union` works.



```
CUSTOMER_NAME   CUSTOMER_STR CUSTOMER_CITY
--------------- ------------ ---------------
Hayes           Main         Harrison
Williams        Nassau       Princeton
Smith           Main         Rye
Jackson         University   Salt Lake
McBride         Safety       Rye
Lindsay         Park         Pittsfield
Curry           North        Rye
Adams           Spring       Pittsfield
Jones           Main         Harrison
Turner          Putnam       Stamford
Majeris         First        Rye

CUSTOMER_NAME   CUSTOMER_STR CUSTOMER_CITY
--------------- ------------ ---------------
Johnson         Alma         Palo Alto

12 rows selected.
```

Figure 2: Output for Task 2

3 Task 3

Find all customer names and their cities who have a loan but not an account (with and without ‘minus’ clause).

3.1 Solution

```
--number03(without minus)--
select distinct customer.customer_name, customer.customer_city
from customer, depositor, borrower
where customer.customer_name=borrower.customer_name and
```

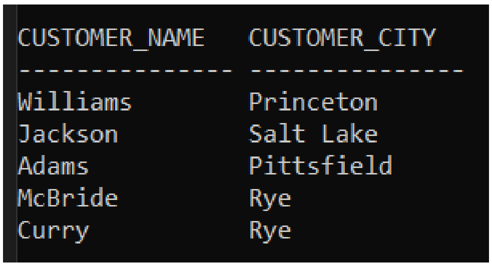
```
borrower.customer_name not in
    (select customer_name
    from depositor);

--number03(with minus)--
select customer.customer_name, customer.customer_city
from customer, borrower
where customer.customer_name=borrower.customer_name
minus
select customer.customer_name, customer.customer_city
from customer, depositor
where customer.customer_name=depositor.customer_name;
```

3.2 Analysis and Explanation

For the first part, I used the keyword `distinct` to remove any duplicate records. I checked if the customer has a loan by matching customer name in borrower and customer table. Then I checked if that customer existed in the depositor table using a nested query. The nested query returned the names of all the customers in the depositor table. I used `'not in'` to check if the customer was absent from the result of the sub query (the customer does not have an account). For the next part, I learned a new keyword `minus`. It shows the result like a mathematical subtraction operation where the result is the leftover records from after removing the results of the second query from the first one. Here, the first query returns all the customer names with loans while the second query returns the customer names with accounts. The result after using `minus` gives the records of the customer names with loans only.

3.3 Difficulties

I did not face any difficulties when doing this task except understanding how the new keyword `minus` works.



```
CUSTOMER_NAME   CUSTOMER_CITY
--------------- ---------------
Williams        Princeton
Jackson         Salt Lake
Adams           Pittsfield
McBride         Rye
Curry           Rye
```

Figure 3: Output for Task 3

4 Task 4

Find the total assets of all branches.

4.1 Solution

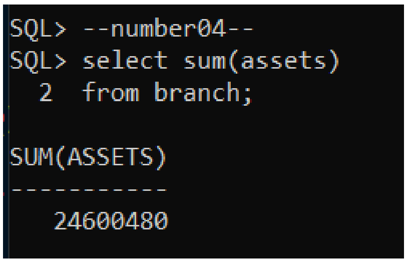
```
--number04--
select sum(assets)
from branch;
```

4.2 Analysis and Explanation

Here, the function `sum()` was used which returned the sum of a column with numeric records.

4.3 Difficulties

I did not face any difficulties when doing this task and no mentionable issues were encountered.



```
SQL> --number04--
SQL> select sum(assets)
  2  from branch;

SUM(ASSETS)
-----------
   24600480
```

Figure 4: Output for Task 4

5 Task 5

Find the total number of accounts at each branch city.

5.1 Solution

```
--number05--
select branch.branch_city, coalesce(count(account.account_number), 0)
as count_account_num
from branch
left join account on account.branch_name=branch.branch_name
group by branch.branch_city
order by branch.branch_city;
```

5.2 Analysis and Explanation

For total number of accounts, I used `count()` function with input as `account.account_number` to count the number of accounts. To show 0 for the branch cities with no account, I used `coalesce()` function to return 0 instead of null. This column was labelled as `count_account_num`. The branch table was then left joined with the account table on the basis of same branch names in both. This is done so that all the branch cities available are shown even if there exists no account in a particular city. This way 0 can be shown for a branch city with no accounts instead of dismissing the record entirely.

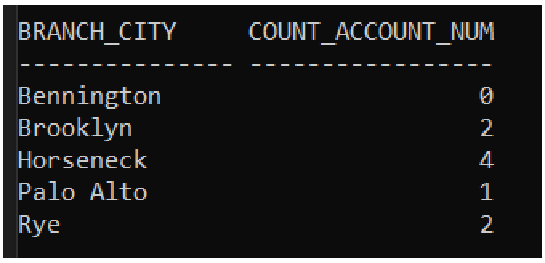
Lastly, I used `group by` to group the results by branch city so that the count of accounts is shown for each branch city and then used `order by` to organise the results alphabetically according to branch city.

5.3 Difficulties

I did face some difficulties while doing this task. I had trouble understanding how to show 0 for the records with count value as null.

I also mistakenly grouped the results by branch name instead of branch city at first.

I had trouble understanding the proper use of the group by clause since I kept getting error for using the wrong attribute that is an attribute not present in the select statement.



```
BRANCH_CITY     COUNT_ACCOUNT_NUM
--------------- -----------------
Bennington                      0
Brooklyn                        2
Horseneck                       4
Palo Alto                       1
Rye                             2
```

Figure 5: Output for Task 5

6 Task 6

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

6.1 Solution

```
--number06--
select branch.branch_name, coalesce(avg(account.balance), 0) as avg_balance
from branch
left join account on branch.branch_name=account.branch_name
group by branch.branch_name
order by avg_balance desc;
```

6.2 Analysis and Explanation

For average balance of accounts, I used `avg()` function with input as `account.balance` to calculate the average balance of some accounts. To show 0 for the branch cities with no account, I used `coalesce()` function to return 0 instead of null. This column was labelled as `avg_balance`.

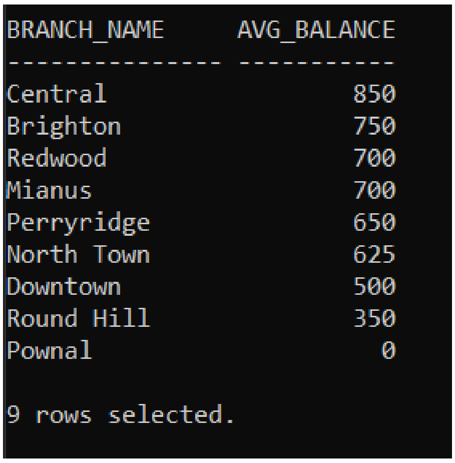
The branch table was then left joined with the account table on the basis of same branch names in both. This is done so that all the branch names available are shown even if there exists no account in a particular branch. This way 0 can be shown for a branch with no accounts instead of dismissing the record entirely.

Lastly, I used `group by` to group the results by branch name so that the average balance of the accounts for a particular branch is calculated and then used `order by` and `desc` to organise the results in descending order according to the calculated average balance.

6.3 Difficulties

I did face some difficulties while doing this task. I had trouble understanding how to show 0 for the records with count value as null.

I also had trouble understanding the proper use of the group by clause since I kept getting error for using the wrong attribute that is an attribute not present in the select statement.



```
BRANCH_NAME     AVG_BALANCE
--------------- -----------
Central                 850
Brighton                750
Redwood                 700
Mianus                  700
Perryridge              650
North Town              625
Downtown                500
Round Hill              350
Pownal                    0

9 rows selected.
```

Figure 6: Output for Task 6

7 Task 7

Find the total balance of accounts at each branch city.

7.1 Solution

```
--number07--
select branch.branch_city, coalesce(sum(account.balance), 0) as balance_sum
from branch
left join account on account.branch_name=branch.branch_name
group by branch.branch_city
order by branch.branch_city;
```

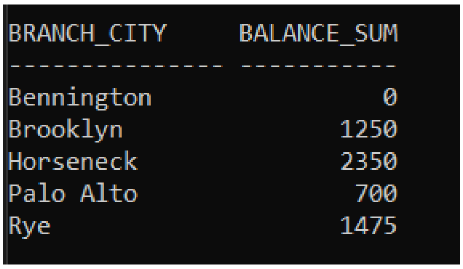
7.2 Analysis and Explanation

For total balance of accounts, I used `sum()` function with input as `account.balance` to calculate the total balance of some accounts. To show 0 for the branch cities with no account, I used `coalesce()` function to return 0 instead of null. This column was labelled as `balance_sum`.
The branch table was then left joined with the account table on the basis of same branch names in both. This is done so that all the branch names available are shown even if there exists no account in a particular branch. This way 0 can be shown for a branch city with no accounts instead of dismissing the record entirely.
Lastly, I used `group by` to group the results by branch city so that the total balance of the accounts for a particular branch city is calculated and then used `order by` to organise the results alphabetically according to branch city.

total balance not being calculated properly

7.3 Difficulties

I did face some difficulties while doing this task. I had trouble understanding how to show 0 for the records with count value as null.
I also had trouble understanding the proper use of the group by clause since I kept getting error for using the wrong attribute that is an attribute not present in the select statement.
The total balance for all the accounts for a particular city was also not being calculated properly at first because I had mistakenly grouped the results by branch name instead of branch city.



```
BRANCH_CITY     BALANCE_SUM
--------------- -----------
Bennington                0
Brooklyn               1250
Horseneck              2350
Palo Alto               700
Rye                    1475
```

Figure 7: Output for Task 7

8 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).

8.1 Solution

```
--number08(without having)--
select branch.branch_name, coalesce(avg(loan.amount), 0) as avg_loan
from branch
left join loan on branch.branch_name=loan.branch_name
where branch.branch_city!='Horseneck'
group by branch.branch_name
order by branch.branch_name;

--number08(with having)--
select branch.branch_name, coalesce(avg(loan.amount), 0) as avg_loan
from branch
left join loan on branch.branch_name=loan.branch_name
group by branch.branch_name
having branch.branch_name not in
    (select branch_name
    from branch
    where branch_city='Horseneck')
order by branch.branch_name;
```

8.2 Analysis and Explanation

Without `having` clause:

For average loan amount, I used `avg()` function with input as `loan.amount` to calculate the average amount of loan. To show 0 for the branch cities with no account, I used `coalesce()` function to return 0 instead of null. This column was labelled as `avg_loan`.

The branch table was then left joined with the loan table on the basis of same branch names in both. This is done so that all the branch names available are shown even if there exists no amount of loan in a particular branch. This way 0 can be shown for a branch with no loans instead of dismissing the record entirely.

I used the condition `branch.branch_city!='Horseneck'` in the `where` clause so that any branch records with this city is dismissed otherwise it would show 0 as the output for these.

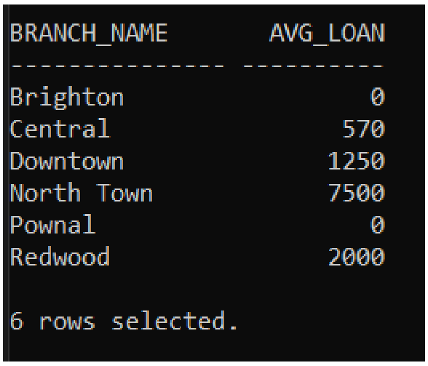
Lastly, I used `group by` to group the results by branch name so that the average loan for a particular branch is calculated and then used `order by` to organise the results alphabetically according to branch name.

With `having` clause:

For using the `having` clause, I removed the `where` clause from the previous code. To exclude `'Horseneck'` city, I used a nested query in the `having` clause where I gave the condition `branch.branch_name not in` followed by the nested query which returns all the `branch_names` with `'Horseneck'` city.

8.3 Difficulties

I faced difficulties when using `having` clause in this problem. First, I used `having branch.branch_city!='Horseneck'` which showed errors saying not a group by expression. This was because I used `branch.branch_name` to group the results and not `branch.branch_city` and `branch.branch_city` is also not included in the `select` statement.



```
BRANCH_NAME       AVG_LOAN
--------------- ----------
Brighton                 0
Central                570
Downtown              1250
North Town            7500
Pownal                   0
Redwood               2000

6 rows selected.
```

Figure 8: Output for Task 8

9 Task 9

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

9.1 Solution

```
--number09(without all)--
select depositor.customer_name, depositor.account_number
from depositor, account
where depositor.account_number=account.account_number and
account.balance =
    (select max(account.balance)
    from account);

--number09(with all)--
select depositor.customer_name, depositor.account_number
from depositor, account
where depositor.account_number=account.account_number and
account.balance>=all
    (select account.balance
    from account);
```

9.2 Analysis and Explanation

Without `all`:

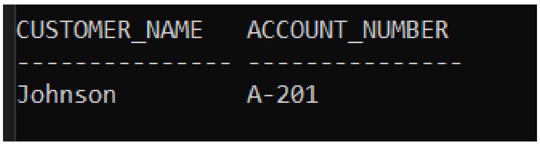
I used a nested query in the `where` clause. The nested query returns the max balance and in the `where` clause we check if the `account.balance` equals to that max balance. For this problem, cartesian product of the `depositor` and `account` tables were taken with the condition that the `account_number` in both are the same.

With `all`:

To solve the problem using the `all` keyword, the code is kept the same except the condition checking part for `account.balance`. Here we check the max balance by comparing all the balance with the current balance. If the current balance is greater or equal than all other account balances then it is the maximum balance. The current balance is checked against all other account balances using a nested query. The `all` keyword returns true for all values here.

9.3 Difficulties

I did not face any difficulties when doing this task except understanding how the new keyword `all` works.



```
CUSTOMER_NAME   ACCOUNT_NUMBER
--------------- ---------------
Johnson         A-201
```

Figure 9: Output for Task 9

10 Task 10

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

10.1 Solution

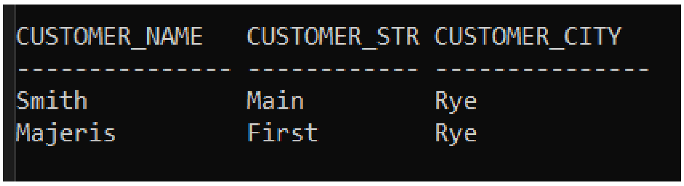
```
--number10--
select distinct customer.customer_name, customer.customer_street,
customer.customer_city
from customer, depositor, account, branch
where customer.customer_name=depositor.customer_name and
depositor.account_number=account.account_number and
branch.branch_name=account.branch_name and
branch.branch_city=customer.customer_city;
```

10.2 Analysis and Explanation

I used the keyword `distinct` to remove any duplicate records. For this problem, I accessed four of the given tables at once. First, I connected `customer` table with `depositor` table using `customer_name` to check if the customer has an account. Then I connected `depositor` table with `account` table using `account_number` to check if the account exists. Then `account` table was connected with `branch` table using `branch_name` to find the `branch_city` of those accounts. Finally, I connected `branch` table with `customer` table to check if the customer lives in the same city as that in which the account is present.

10.3 Difficulties

I did not face any difficulties when doing this task and no mentionable issues were encountered.



```
CUSTOMER_NAME   CUSTOMER_STR CUSTOMER_CITY
--------------- ------------ ---------------
Smith           Main         Rye
Majeris         First        Rye
```

Figure 10: Output for Task 10

11 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).

11.1 Solution

```
--number11(without having)--
select *
from
    (select branch.branch_city, avg(loan.amount) as avg_loan
    from branch natural join loan
    group by branch.branch_city) query
where query.avg_loan>=1500;

--number11(with having)--
select branch_city, avg(loan.amount) as avg_loan
```

```
from branch natural join loan
group by branch_city
having avg(amount)>=1500;
```

11.2 Analysis and Explanation

Without `having` clause:

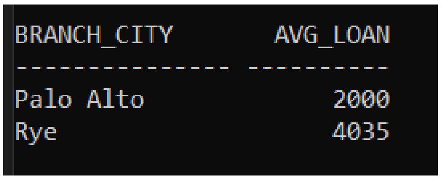
I used a nested query in the `from` clause. This allows us to put conditions on a calculated value such as the average loan amount. So to exclude results with the average loan less than 1500, I wrote the condition in the `where` clause. Inside the nested query, I natural joined the `branch` and `loan` tables that is they will join on the basis of a common attribute which is `branch_name` in this case.

With `having` clause:

When using the `having` clause, there is no need to store the computed result using a nested query. We use the same code as the nested query for the previous code and then add a `having` clause at the end with the condition that the average amount is greater than or equal to 1500.

11.3 Difficulties

I faced a few difficulties when doing this task. At first, I tried to solve the first part without using a nested query which resulted in many errors since we can not put conditions on a value calculated on the fly. Then, when using the `having` clause `avg_loan` shows error but `avg(loan.amount)` works correctly.



```
BRANCH_CITY       AVG_LOAN
--------------- ----------
Palo Alto             2000
Rye                   4035
```

Figure 11: Output for Task 11

12 Task 12

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

12.1 Solution

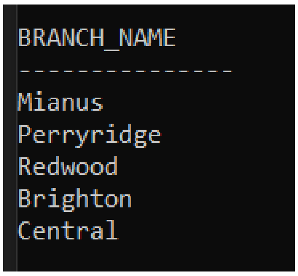
```
--number12--
select branch_name
from account natural join branch
group by branch_name
having sum(balance)>
(select avg(account.balance)
from account);
```

12.2 Analysis and Explanation

The `account` and `branch` tables were natural joined so that we can work on the records with the same `branch_name` only. We group the results according to `branch_name`. Then we use `having` clause to implement the condition of total account balance being greater than the average total balance among all the branches. Here a nested query is used to compute the average balance among all the accounts in the branch because we can not compare the sum of the balance with the average otherwise. So a nested query is used to compute the result beforehand and store it then we compare the sum of balance against the result of the nested query in the `having` clause.

12.3 Difficulties

I did not face any difficulties when doing this task and no mentionable issues were encountered.



```
BRANCH_NAME
---------------
Mianus
Perryridge
Redwood
Brighton
Central
```

Figure 12: Output for Task 12

13 Task 13

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

13.1 Solution

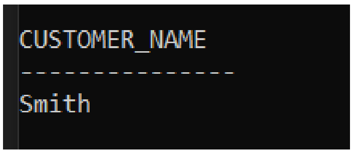
```
select distinct customer_name
from (select customer_name, sum(balance) as s_balance
from customer natural join (depositor natural join account)
group by customer_name) q1 natural join
(select customer_name, amount
from customer natural join (borrower natural join loan)) q2
where q2.amount<=q1.s_balance;
```

13.2 Analysis and Explanation

I used the keyword `distinct` to remove any duplicate records. I used to nested queries here to compare the results. The first nested query, `q1`, returns the total balance for each customer while the second one, `q2`, returns the individual loan amounts for each customer. The two queries were then natural joined to create one table with all the required information for comparison and printing. The total balance was then compared with the loan amounts with the condition that loan was less or equal than the total balance. That means it could be paid off so then that customer name would be printed.

13.3 Difficulties

I found it a bit difficult to come up with the nested queries and how to link those queries but other than that there were no mentionable issues encountered.



```
CUSTOMER_NAME
---------------
Smith
```

Figure 13: Output for Task 13

14 Task 14

Find the branch information for cities where at least one customer lives who do not have any account or any loans. The branch must have given some loans and has accounts opened by other customers.

14.1 Solution

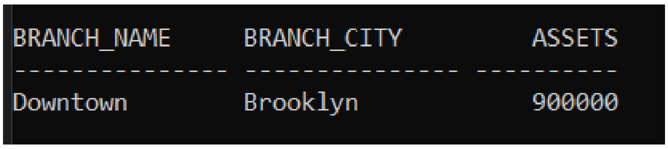
```
--number14--
select branch.branch_name, branch.branch_city, branch.assets
from branch, customer
where branch.branch_city=customer.customer_city and
customer.customer_name in
(--customers with no account or loan--
select customer_name
from customer
where customer_name not in
(select customer.customer_name
from customer, depositor, borrower
where customer.customer_name=depositor.customer_name or
customer.customer_name=borrower.customer_name)) and
branch.branch_name in
(--branch used for loans--
select branch_name
from loan natural join borrower) and
branch.branch_name in
(--branch used for account--
select branch_name
from account natural join depositor);
```

14.2 Analysis and Explanation

We need the branch information so we list all the branch attributes in the `select` clause. Then we check if the customer lives in that particular branch city in the `where` clause. To check if that customer has any accounts or loans or not, I used a nested query within another nested query. The first one returns all the customers who have no accounts or loans with which we cross check to see if the customer is in that list or not. Next we check the branch. The second nested query checks if the branch was used for making any loans while the third one checks is the branch was used for making any accounts.

14.3 Difficulties

I faced difficulty in figuring out how the subqueries will be connected but other than that there were no mentionable issues encountered.



```
BRANCH_NAME     BRANCH_CITY         ASSETS
--------------- --------------- ----------
Downtown        Brooklyn            900000
```

Figure 14: Output for Task 14

Conclusion

As shown in the report, I have solved and tested the solutions for all the tasks given in the lab. All the commands used were written in notepad which was then saved with `.sql` extension. The `.sql` file was then run through the `SQL` command line to execute all the commands.