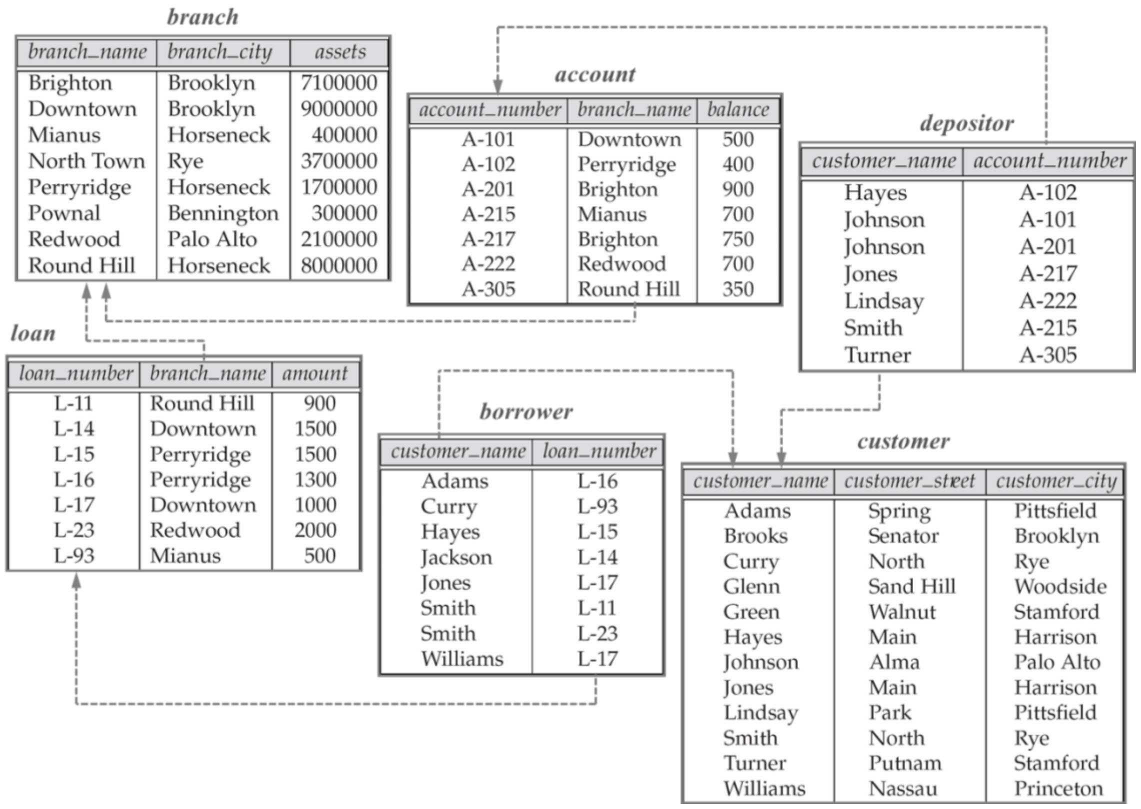
Lab Session 6: Advanced SQL and Indexing

In this lab, we are going to try out more complicated queries. As usual, we will use our simple bank database as an example.



branch

branch_name	branch_city	assets
Brighton	Brooklyn	7100000
Downtown	Brooklyn	9000000
Mianus	Horseneck	400000
North Town	Rye	3700000
Perryridge	Horseneck	1700000
Pownal	Bennington	300000
Redwood	Palo Alto	2100000
Round Hill	Horseneck	8000000

account

account_number	branch_name	balance
A-101	Downtown	500
A-102	Perryridge	400
A-201	Brighton	900
A-215	Mianus	700
A-217	Brighton	750
A-222	Redwood	700
A-305	Round Hill	350

depositor

customer_name	account_number
Hayes	A-102
Johnson	A-101
Johnson	A-201
Jones	A-217
Lindsay	A-222
Smith	A-215
Turner	A-305

loan

loan_number	branch_name	amount
L-11	Round Hill	900
L-14	Downtown	1500
L-15	Perryridge	1500
L-16	Perryridge	1300
L-17	Downtown	1000
L-23	Redwood	2000
L-93	Mianus	500

borrower

customer_name	loan_number
Adams	L-16
Curry	L-93
Hayes	L-15
Jackson	L-14
Jones	L-17
Smith	L-11
Smith	L-23
Williams	L-17

customer

customer_name	customer_street	customer_city
Adams	Spring	Pittsfield
Brooks	Senator	Brooklyn
Curry	North	Rye
Glenn	Sand Hill	Woodside
Green	Walnut	Stamford
Hayes	Main	Harrison
Johnson	Alma	Palo Alto
Jones	Main	Harrison
Lindsay	Park	Pittsfield
Smith	North	Rye
Turner	Putnam	Stamford
Williams	Nassau	Princeton

Figure 1. Bank database

Part I: Customers with accounts in every branch of Brooklyn

In the slides of this course, we have seen how to find the customers who have accounts in <u>every</u> branch in the city of Brooklyn.

1. Write a query to show the branches in the city of Brooklyn. (Show the branch names.)
2. Expand the previous query to show the accounts that belong to those branches. (Show branch names and account numbers.)
3. Expand the previous query to show also the customers who own those accounts. (Show branch names, account numbers, and customer names.)

4. In the result of the previous query, can you find any customer who has accounts in every branch of Brooklyn?
5. Execute the following query to find those customers directly:

```
select distinct customer_name
from depositor as d
where not exists (
          select branch_name
          from branch as b
          where branch_city = 'Brooklyn'
             and branch_name not in (
                         select branch_name
                         from account as a, depositor as d2
                         where a.account_number = d2.account_number
                            and d2.customer_name = d.customer_name));
```

6. The query above returns a customer from table depositor (d) if and only if the customer obeys a certain condition. Can you explain this condition in your own words?

   *(Hint: the third **select** in the query returns the branches where that customer has accounts.)*

7. The query in step 5 has a double negation with **not exists** and **not in**. Rewrite the query using **not exists** twice (i.e. replace the **not in** by a second **not exists**).

Part II: Customers with accounts in every branch of the city where they live

We are now interested in finding the customers who have accounts in every branch of the same city where they live.

8. Can you formulate this question using a double negation?

   (Hint: if a customer has accounts in every branch of the city where he/she lives, then we cannot find a branch in that city where the customer does not have an account.)

9. Change the query from step 5 so that, instead of selecting the branches in the city of Brooklyn, you select the branches in the same city where the customer lives.

   *(Hint: in the first **select**, you may want to join the tables depositor and customer so that you can access the customer city in the second **select**.)*

   When you run the query, you may get the following result:

```
+---------------+
| customer_name |
+---------------+
| Hayes         |
| Jones         |
| Lindsay       |
| Turner        |
+---------------+
```

10. Run the following query to check where these customers live, and where they have accounts:

```
select c.customer_name, c.customer_city, b.branch_city
from customer as c, depositor as d, account as a, branch as b
where c.customer_name = d.customer_name
  and d.account_number = a.account_number
  and a.branch_name = b.branch_name
  and c.customer_name in ('Hayes', 'Jones', 'Lindsay', 'Turner');
```

    In the result of this query, you will see that the cities (customer_city and branch_city) do not match. Then how is it possible that those customers appear in the result of the previous query? What is going on?

    (Hint: in the database, you will find that there no branches in the cities where these customers live, so we cannot find a branch in that city where the customer does not have an account.)

11. Improve the query you wrote in step 9 to make sure that you get the expected result:

```
Empty set (0.00 sec)
```

    i.e. there are no customers who have accounts in every branch in the same city where they live.

    (Hint: you have to enforce another condition, which is that there must exist branches in the city where the customer lives.)

Part III: Date and time

In the slides of this course, we have seen how to work with dates, times, and timestamps.

12. Use the following instructions to create a database about movies and tickets in a movie theater:

```
create table movie (
        title varchar(255),
        year year,
        length time,
        primary key(title, year) );
create table ticket (
        number integer,
        showdate date,
        showtime time,
        title varchar(255),
        year year,
        primary key(number),
        foreign key(title, year) references movie(title, year) );
```

13. Use the following instructions to insert some records in these tables:

```
insert into movie values ('Back to the Future', 1985, '01:56:00');
insert into movie values ('Back to the Future II', 1989, '01:48:00');
insert into movie values ('Back to the Future III', 1990, '01:58:00');
insert into ticket values (123456, '2016-11-12', '18:25:00', 'Back to the Future', 1985);
insert into ticket values (123457, '2016-11-12', '21:15:00', 'Back to the Future', 1985);
insert into ticket values (123458, '2016-11-13', '00:05:00', 'Back to the Future', 1985);
```

14. Run the following query to discover how much time you have left before each of these movie sessions:

```
select showdate, showtime,
       datediff(showdate, current_date) as daysuntil,
       timediff(showtime, current_time) as timeuntil
from ticket;
```

15. Run the following query to determine at what time the movie will end, including 20 minutes of trailers and advertisements:

```
select t.title, t.showtime as begins,
       addtime(addtime(t.showtime, m.length), sec_to_time(20*60)) as ends
from ticket as t, movie as m
where t.title = m.title and t.year = m.year;
```

16. Check out the documentation for these and other date and time functions:

https://dev.mysql.com/doc/refman/5.1/en/date-and-time-functions.html

Part IV: Indexing

In the slides of this course, we have seen how to use indexes.

(a) Use the script named **index_data.sql** to populate the table named **account.**

(b) Execute the following query and notice the time involved in its execution:
SELECT * FROM account WHERE account_number='A-012345';

(c) Delete the primary key with the following command:
ALTER TABLE account DROP PRIMARY KEY;

(d) Repeat the query from step (b) and notice the difference in the execution time. How do you explain this result?

(e) Put back the primary key constraint in the table with the following command:
ALTER TABLE account ADD PRIMARY KEY(account_number);
Notice the time involved in executing this command. How do you explain the result?

(f) Write a query to retrieve the number of accounts with a balance equal to 1000 € and another to retrieve the maximum balance. Execute the queries and take note of the time involved on their execution.

SELECT account_number FROM account WHERE balance = 1000;
SELECT MAX(balance) FROM account;

(g) Create an índex over the balance atribute, with the following command:
CREATE INDEX balance_idx ON account (balance);
Is this index going to be primary or secondary? Why?

(h) Repeat the queries from step (f) and take note of the execution time. How do you explain an eventual difference from the original results?

(i) Consider the following relation: **Emp (<u>eid</u>, ename, address, salary, age)**

Suppose the following queries are very frequent.

a. What is the identifier, name, and address for employees with ages in a given interval?
b. What is the identifier and address of employees with a given name?
c. What is the maximum salary of employees?
d. What is the average salary of employees, by age?

State which indexes could be created to make the execution of each query more efficient. Justify your answers.