**Deposit**

Using SQL, create a table to identify

1. customer\_id: customer who reached cumulative minimum deposit of $500.
2. deposit\_date: on what day customer reached cumulative minimum deposit of $500.

account

|  |  |  |
| --- | --- | --- |
| customer\_id | account\_id | account\_type |
| 1 | 11 | Checking |
| 1 | 12 | Savings |
| 2 | 21 | Checking |
| 2 | 22 | Savings |
| 3 | 31 | Checking |

*customer\_id: 1*

*account\_id: 11, 12*

*deposit\_date: 2020-01-01 ~ 2020-01-05*

*deposit\_amount*

1. *group by customer\_id: 1*
2. *order by deposit\_date: 20120-01-01, 02*
3. *sum(deposit\_amount): 150, 500…*
4. *cumulative sum(deposit\_amount): 150, 650…*
5. *cumulative sum(deposit\_amount) >= 500: 650…*
6. *cumulative sum(deposit\_amount) >= 500: 650 (first row)*
7. *first date >=500: 2020-01-02*

deposit

|  |  |  |
| --- | --- | --- |
| account\_id | deposit\_date | deposit\_amount |
| 11 | 2020-01-01 | 100 |
| 11 | 2020-01-02 | 300 |
| 11 | 2020-01-05 | 200 |
| 12 | 2020-01-01 | 50 |
| 12 | 2020-01-02 | 200 |
| 12 | 2020-01-03 | 400 |
| 12 | 2020-01-04 | 100 |
| 21 | 2020-01-02 | 29 |
| 21 | 2020-01-04 | 18 |
| 21 | 2020-01-05 | 29 |
| 22 | 2020-01-03 | 100 |
| 22 | 2020-01-04 | 200 |
| 22 | 2020-01-05 | 300 |
| 31 | 2020-01-02 | 10 |
| 31 | 2020-01-03 | 40 |

Transcript

We used to call a bank savings-and-loans. Because that's what a bank does. A bank borrows from savers and lend to borrowers. Just how important deposits are... to a bank? Well... without deposits the bank can't lend. There are two kinds of deposits: demand deposits and term deposits. Demand deposits can be withdrawn any time without notice. Demand deposits include checking account and savings account. Term deposits are fixed term investments and are called guaranteed investment certificates (or GICs) in Canada. Checking account earns zero interest. High interest savings account (or HISA) is a joke and earns only 0.5% interest. I have no idea why it's called high interest. Guess 0.5% is considered high in a low interest rate environment. You keep thousands of dollars in checking and savings accounts because you want liquidity... or the convenience to access cash any time to pay everyday expenses and monthly bills. Demand deposits are the cheapest source of funding for the bank. Because you are not sensitive to interest rate movements and always maintain a significant balance of demand deposits, the bank can use these short-term deposits to fund long-term loans such as mortgages.

,,,,,,,,,,,,

Now let's try to understand demand deposit business and data. Data has three levels of details: customer, account and transaction. Imagine you have a third table: customer table... with columns like customer id (which is a primary key), first name, last name, date of birth, phone number, email address and home address. Now look at account table. It has 5 rows of sample data. How many customers do you see? Yes 3... not 5. Count duplicate values only once. Assuming you are a customer and your customer id is 1, how many accounts do you have? You have 2 accounts: 11 and 12. They are different types: 11 is checking account, 12 is savings account. In account table, account id is a primary key (which doesn't allow duplicate values), customer id is a foreign key (which allows duplicate values). Now look at deposit table. It has 15 rows of sample data. Remember your account id: 11 and 12, how many deposits have you made? 7. You have made 7 deposits on different dates to different accounts. 7 deposits are 7 credit transactions. In reality, cash comes in and goes out. Say you work part-time and wait tables at a restaurant. Tips you earn credit your accounts and are positive. You use debit card (which is linked to your checking account) to buy coffee or move money from savings to checking and write a check to pay rent. These transactions debit your accounts and are negative. Pretty easy to understand demand business and data... right? For simplicity's sake, we only look at deposit or credit transactions, transaction table becomes deposit table. Now let's read the question. Using SQL, create a table to identify customer who reached cumulative deposit of $500... and the date on which customer reached cumulative deposit of $500. Cumulative deposit, by definition, increases every time you make a deposit. The question asks for the first date on which customer's cumulative deposit >= $500. The best way to answer this question is assuming you are one of their customers so the experience becomes personal. Your customer id is 1, your account id's are 11, 12 and you have made 7 deposits from 2020-01-01 to 2020-01-05. Normally you solve this problem by running quick math in your head. I suggest you document your thought process and break it down into steps. 1) Follow the hierarchy: customer, account, deposit and find deposits made by you, group deposits by customer id and deposit date. 2) Order deposits by customer id and deposit date in ascending order from earliest to latest: from 2020-01-01 to 2020-01-05. 3) Sum deposit amount by customer id and deposit date: 100+50=150 (on 2020-01-01), 300+200=500 (on 2020-01-02), 400 (on 2020-01-03), 100 (on 2020-01-04), 200 (on 2020-01-05). 4) Calculate cumulative sum of deposit amount by customer and deposit date: 150 (on 2020-01-01), 150+500=650 (on 2020-01-02), 650+400=1050 (on 2020-01-03), 1050+100=1150 (on 2020-01-04), 1150+200=1350 (on 2020-01-05). 5) Evaluate the condition if cumulative sum of deposit amount >= 500: 150 (no), 650 (yes), 1050 (yes), 1150 (yes), 1350 (yes). First row meeting the condition is 650. First date meeting the condition is 2020-01-02.

,,,,,,,,,,,,

You realize this problem involves a little bit business, a little bit statistics and a little bit coding. So a data analyst is a hybrid role that requires you to have a hybrid skills set: business, statistics and coding. After you understand business and data, now is the right time to code logic into SQL. I want you to remember that SQL is more logic than coding. If you can think logically, then you can code. You code as a business and data analyst (data programming) not as a programmer or developer (application programming). Very different! My word of advice is never code in a word document. You should always use an interactive development environment (or IDE) to solve a SQL problem. And you should always test your solution before you submit it. So you open MySQL. Create a database called deposit. Create two tables. And insert values and populate two tables with sample data. The final solution looks complicated. But if you break it down into steps, you realize that it follows the exact same logic. It's a nested query: query within query within query. It is executed from innermost query to outermost query. It's easy to identify the innermost query: just look for select-from-where. Put your cursor before "select", you will see the beginning and the end of the query highlighted. Now select the query and click run, it completes step 1 to 3: join two tables - account and deposit on account id... group deposits by customer id and deposit date... order deposits by customer id and deposit date in ascending order from earliest to latest... sum deposit amount by customer id and deposit date. You run the innermost query, you get result table you expected: for customer id 1, 7 deposits get summed up to 5 dates: 150 (on 2020-01-01), 500 (on 2020-01-02), 400 (on 2020-01-03), 100 (on 2020-01-04), 200 (on 2020-01-05). Now look for the 2nd innermost query. Put your cursor before "select", you will see the beginning and the end of the query highlighted. Now select the query and click run. This query selects \* (star) and add a 4th column. Whenever you select \* (star), you create a copy of previous table, and add a new column cumulative sum of deposit amount. Here we use a window function, which is a function over a window. First it divides result table into 3 partitions by customer id (one partition for each customer id), then it orders rows within each partition by deposit date in ascending order from earliest to latest - here is the window. Then calculate cumulative sum of deposit amount over this window, you get: 150, 650, 1050, 1150, 1350. Do you follow? The 3rd innermost query selects \* (star), again create a copy of the previous table, and add a 5th column row number, which is another window function. Let me repeat. A window function is a function over a window. First it divides result table into 3 partitions by customer id (one partition for each customer id), then it orders rows within each partition by deposit date in ascending order from earliest to latest - here is the window. Then number the rows over this window. Hope you get it. Try run the query without the condition: cumulative sum of deposit amount >= 500, which is step 4, the 5th column has the row number: 1, 2, 3, 4, 5, five rows for five dates for customer id 1. Now run the query with the condition: cumulative sum of deposit amount >= 500, which is step 5, it will drop the rows with cumulative sum of deposit amount < 500, keep the rows with cumulative sum of deposit amount >= 500, and re-number the rows, 4 rows for customer id 1, 1 row for customer id 2, none for customer id 3. The last and also the outermost query find the first row meeting condition, using row number = 1. The logic is exactly the same as if you do the math in your head.

,,,,,,,,,,,,

In the end of this tutorial, I would like to tell you a story about window function. One of our students by the name Wilmer went for a technical interview, which included a SQL problem. He solved the problem. The hiring manager reviewed his query: Good, now can you improve it? Wilmer was like: What do you mean? Didn't I solve the problem? The hiring manager was like: Yes you solved the problem with complex self joins. Now assuming the table is huge, your query will take a long time to run. Can you modify the query to make it run faster. Sorry I don't know how. Well... then you are a junior. A senior would know how to improve performance of this query using window function. Wilmer was upset by the comment. After the interview, he searched on the Internet. He was like: window function isn't hard. It took me only half an hour to learn it. The hiring manager is so judgmental...... Well, the hiring manager isn't "judgmental". Just how did he conclude that you are junior? It's not because you don't know window function. It's because you haven't experienced real-world performance issues with complex queries on large databases. He called you junior because you don't know SQL performance tuning. So what have you learned from this tutorial? 1) real world data analytics problems involve business, statistics and coding that require you to have hybrid skills set: business, statistics and coding. 2) SQL is more logic than coding. If you can think logically, then you can code. 3) You should always use an IDE to solve a SQL problem and test your solution before you submit it. 4) It's not enough to write down SQL as a database tool on your resume. Nowadays even the first-year university students put every data tool they can think of on their resumes. To stand out from the crowd, you should write down SQL/Database with specific details such as complex joins, nested queries, window functions, stored procedures, performance tuning... and have a story handy to prove that you know how to use SQL to solve real world problems. Then you will sound like a senior and convince the hiring manager that you can do the job.