Normal interview questions

1. What is Database?
2. What is DBMS?
3. What is RDBMS? How is it different from DBMS?
4. What is SQL?
5. What is the difference between SQL and MySQL?
6. What are Tables and Fields?
7. What are Constraints in SQL?
8. What is a Primary Key?

Q   =>   Write a SQL statement to add PRIMARY KEY 't\_id' to the table 'teachers'.

Q   =>   Write a SQL statement to add primary key constraint 'pk\_a' for table 'table\_a' and fields 'col\_b, col\_c'.

1. What is a UNIQUE constraint?
2. What is a Foreign Key?

Q   =>   What type of integrity constraint does the foreign key ensure?

Q   =>   Write a SQL statement to add a FOREIGN KEY 'col\_fk' in 'table\_y' that references 'col\_pk' in 'table\_x'.

1. What is a Join? List its different types.
2. What is a Self-Join?
3. What is a Cross-Join?
4. What is an Index? Explain its different types.
5. What is the difference between Clustered and Non-clustered index?
6. What is Data Integrity?
7. What is a Query?
8. What is a Subquery? What are its types?
9. What is the SELECT statement?
10. What are some common clauses used with SELECT query in SQL?
11. What are UNION, MINUS and INTERSECT commands?
12. What is Cursor? How to use a Cursor?
13. What are Entities and Relationships?
14. List the different types of relationships in SQL.
15. What is an Alias in SQL?
16. What is a View?
17. What is Normalization?
18. What is Denormalization?
19. What are the various forms of Normalization?
20. What are the TRUNCATE, DELETE and DROP statements?
21. What is the difference between DROP and TRUNCATE statements?
22. What is the difference between DELETE and TRUNCATE statements?
23. What are Aggregate and Scalar functions?
24. What is User-defined function? What are its various types?
25. What is OLTP?
26. What are the differences between OLTP and OLAP?
27. What is Collation? What are the different types of Collation Sensitivity?
28. What is a Stored Procedure?
29. What are user defined function?
30. What is a Recursive Stored Procedure?
31. How to create empty tables with the same structure as another table?
32. What is Pattern Matching in SQL?
33. [What is the difference between CHAR and VARCHAR2 datatype in SQL?](https://www.edureka.co/blog/interview-questions/sql-interview-questions#charvsvarchar)
34. What do you mean by “Trigger” in SQL?
35. What is the ACID property in a database?
36. Are NULL values same as that of zero or a blank space?
37. What are STUFF and REPLACE function?
38. Procedure vs Function
39. What are Local and Global variables?
40. What is view and why do we created view ?

Scenario based Questions:

**How can you fetch alternate records from a table?**

You can fetch alternate records i.e both odd and even row numbers. For example- To display even numbers, use the following command:

Select studentId from (Select rowno, studentId from student) where mod(rowno,2)=0

Now, to display odd numbers:

Select studentId from (Select rowno, studentId from student) where mod(rowno,2)=1

### How can you fetch first 5 characters of the string?

There are a lot of ways to fetch characters from a string. For example:

Select SUBSTRING(StudentName,1,5) as studentname from student

Q1 How will you go about identifying duplicate records in a table?  
A1 The following SQL query will do the trick.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | SELECT code, user\_name, COUNT(user\_name) AS NumOccurrences    FROM tbl\_user    GROUP BY code, user\_name    HAVING ( COUNT(user\_name) > 1 ) |

**Note:** Interviewer is testing if you understand that the aggregate queries need to have a “**GROUP BY**” clause on columns that are **NOT** aggregated. E.g. code and user\_name.

Q2 How would you go about deleting the duplicate records?  
A2 You could do it in a number of steps as shown below.

* Create a temporary table.
* Insert the unique records into the temporary table.
* Drop the original table.
* Rename the temp table to original table name.

#### DISTINCT keyword

Example based on Teradata.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | CREATE TABLE employee\_temp AS  (SELECT DISTINCT \* FROM employee)  WITH DATA AND STATS;    DROP TABLE employee;    RENAME TABLE employee\_temp TO employee; |

#### Window function

Example based on Teradata. **ROW\_NUMBER()** is a window function that assigns a sequential integer to each row within the **PARTITION BY** of a result set. In the below example, the rows are partitioned by name, designation, and dob. **QUALIFY** is like a WHERE clause for windowed records, and will filter out any duplicated records. Learn more in detail at [SQL window function for analytics & qualify interview Q&As](https://www.java-success.com/sql-window-function-analytics-interview-qas/).

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | INSERT INTO employee\_temp  SELECT \* FROM employee  QUALIFY ROW\_NUMBER() over (PARTITION BY name, designation, dob order by id desc) = 1;    DELETE FROM employee;    INSERT INTO employee SELECT \* FROM employee\_temp;    DROP TABLE employee\_temp; |

#### Window function & CTE

With databases that support **CTE** (i.e. **C**ommon **T**able **E**xpression). Learn more in detail at [Common Table Expressions (i.e. CTE) in SQL using the “WITH” clause](https://www.java-success.com/sql-common-table-expressions-e-cte/).

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | WITH CTE AS  (  SELECT id,  name,  designation,  dob,  ROW\_NUMBER() OVER (PARTITION BY name, designation, dob ORDER BY id desc) AS row\_num  FROM employee  )    DELETE FROM CTE WHERE row\_num<>1 |

#### GROUP BY & max(..)

Using the GROUP BY & MAX() without using CTE and ROW\_NUMBER().

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | DELETE  FROM employee  WHERE ID NOT IN  (  SELECT MAX(ID)  FROM employee  GROUP BY name, designation, dob) |

Q3 When you have a table that maintains history of record updates, how will you go about retrieving the latest records?

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | id      name     salary          updated  ------------------------------------------  1       John     35000.00        2018-05-01  2       Peter    42000.00        2019-04-01  3       Peter    54000.00        2017-01-22  4       Sam      58000.00        2018-01-01  5       John     62000.00        2016-01-01 |

A3 The CDC (i.e. Change Data Capture) & **SCD** (i.e. Slowly Changing Dimension) concepts are must know for Data Engineers & Data Analysts. These are discussed in detail at [10 Data warehouse interview Q&As](https://www.java-success.com/9-data-warehouse-interview-questions-answers/).

You can use a

#### #1 Subquery



|  |  |
| --- | --- |
| 1  2  3  4  5  6 | SELECT \* FROM user u1  where updated = (select max(u2.updated)                   from user u2                   where u1.name = u2.name  ); |

#### #2 Inner Join



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | SELECT u1.\* FROM user u1  join (select name, max(updated) as maxdate                   from user                   group by name) as u2  on u1.name = u2.name  and u1.updated = u2.maxdate; |

#### #3 Window function



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | SELECT \* FROM    (SELECT \*       , ROW\_NUMBER() OVER(PARTITION BY name ORDER BY updated desc) as row\_num      FROM user    ) as t  where t.row\_num = 1 |

Q4 How will you go about finding…”Which customers **didn’t** place an order in August”?  
A4 Use an **ANTI-JOIN**, which is a join between two tables that returns rows from the first table where no matches are found in the second table.

**1)** “**LEFT JOIN” – Lists all customers with non-null cust\_ids for those who placed orders in “Aug 2020”, with null cust\_ids for those who did not place orders.**

**2)** So “where o.cust\_id is NULL” will give all the customers who did not place orders in Aug 2020.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | SELECT c.id as customer\_id,         c.name as customer\_name    FROM customers c    LEFT JOIN orders o      ON c.id = o.cust\_id     AND o.ord\_date BETWEEN '2020-08-01' AND '2020-08-31 23:59:59'  WHERE o.cust\_id is NULL |

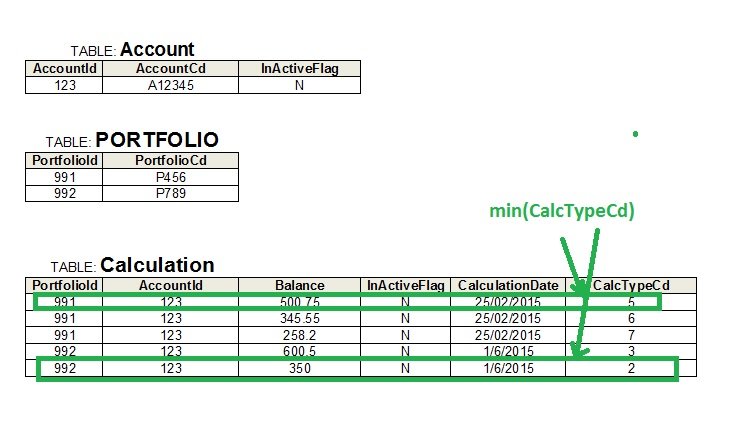
Similar approach can be used for scenarios like:

a) Which customers haven’t visited my website this year?  
b) Which real-estate agents didn’t close a deal last week?  
c) Which products are not sold in the last quarter?  
d) and so on …….

Q8 When will you use a full-outer-join? Is there another alternative to get the same results without using a full-outer-join?  
A8 The **full-outer-join** returns a result set that includes rows from both left and right tables. When no matching rows exist for the row in the left table, the columns of the right table will have nulls. Similarly, when no matching rows exist for the row in the right table, the column of the left table will have nulls.

Yes, the alternative is to do a **UNION** of **a)** an INNER JOIN, **b)** a LEFT JOIN (with right side IS NULL) and **c)** a RIGHT JOIN (with left side IS NULL). This approach is better as you can control each individual joins without having to use complex coalesce statements required with the full-outer-join.

Q9 How will you go about writing an **SQL**query for the following scenario?

[](https://www.java-success.com/wp-content/uploads/2015/04/sql-scenario.jpg)

database table structure

**Calculation table**with the following columns **PortfolioId**, **AccountId**, **Balance**, **InActiveFlag**, **CalculationDate**, and **CalcTypeCd**. The **Portfolio**table has columns **PortfolioId**, and **PortfolioCd**.  The **Account table**has columns **AccountId**and **AccountCd**.

Write an SQL query to extract out the **Accountcd**and the corresponding **Balance**for a given **Portfoliocd** and **CalculationDate**. Please note that there will be multiple balance records for each account, and your query must only extract out a single balance record per account based on the rule ‘extract the record with minimum value for **CalcTypeCd**‘.

A9 As you can see in the sample answer below, **inner joins** are used to join with the relevant tables. A **sub query** is used to calculate the **min(CalcTypeCd)**to extract the record with minimum value for **CalcTypeCd**.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | select  acc.AccountCd, calc.Balance  from Calculation calc          inner join Portfolio pf on pf.PortfolioId = calc.PortfolioId          inner join Account acc on acc.AccountId = calc.AccountId  where pf.PortfolioCd = 'P456'  and   calc.CalculationDate = '25 Feb 2015'  and   calc.InActiveFlag = 'N'  and   acc.InActiveFlag = 'N'  and   calc.CalcTypeCd = (select min(calc2.CalcTypeCd) from calculation calc2                            where calc2.CalculationDate = calc.CalculationDate                             and calc2.InActiveFlag = 'N'                             and calc2.AccountId = calc.AccountId                             group by AccountId)  order by acc.AccountCd |

Q10 If you need to map actual values retrieved from the database to some other value and then sort by these translated values as well, how will you go about accomplishing this in your SQL code?

**For example**, **StatusCd**is the column in the Portfolio table,  and it can have the values of New, and Processed. But the SQL query should return a status of ‘Excluded’ if the **ExcludedFlag**column is set yes, and ‘Sent’ if the SentDateTime is not null. iIf none of the above conditions are met, then return the **StatusCd** as in the database. The sorting needs to be carried out in the order of ‘**New**‘, ‘**Processed**‘, ‘**Sent**‘, and then ‘**Excluded**‘.

A10 This can be achieved with **case** statement. The syntax of switch/case statement can vary among databases. Here is a sample SQL based on Sybase database server.

case-when|else-end statements

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | SELECT PortfolioCd, SentDateTime, ExcludedFlag, StatusCd as ActualStatusCd,         case           when p.ExcludedFlag = 'Y' then 'Excluded'           else              case                  when p.SentDateTime is null then p.StatusCd                  else 'Sent'              end           end as EvaluatedStatusCd    FROM Portfolio p WHERE calculationdate > '09 Jan 2013'  and InActiveFlag = 'N'  ORDER BY case               when p.ExcludedFlag = 'Y' then '4'               else                  case                      when p.SentDateTime is not null then '3'                      else                         case                            when p.StatusCd = 'New' then '1'                            when p.StatusCd = 'Processed' then '2'                          end                  end             end,  PortfolioCd |

Q11 How would you retrieve a date time column converted to string and formatted as dd/mm/yy hh:mm:ss  
A11 You can use specific **functions**provided by your database server. These functions are specific to the database server you are using, hence your code cannot be ported to other database servers. Here is an example in Sybase.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | SELECT PortfolioCd, convert(char(11), p.SentDateTime, 103) + convert(char(12), p.SentDateTime, 108) as SentDateTime  FROM Portfolio p  WHERE calculationdate > '09 Jan 2013'  AND InActiveFlag = 'N' |

In the above example, the convert function is used to convert the date time field to **char**. The **103**in Sybase means **dd/mm/yy**format and and **108**to convert to the time format **hh:mm:ss**.

### ****Scenario 1 :****  What is Query to find Second highest salary for employee?

This is most asked Real Time Scenarios in SQL in many industries. There are lot of real time situation where user needs to deal with this kind of situation. User will try multiple queries to find out the same result.

Query 1 :

**Select distinct Salary from Employee e1 where 2=Select count(distinct Salary) from Employee e2 where e1.salary<=e2.salary**;

Query 2:

**select min(salary)from(select distinct salary from emp order by salary desc)where rownum<=2;**

Query 3:

**select \* from(Select S.\*,DENSE\_RANK() OVER (PARTITION BY DNO ORDER BY SALARY DESC) DR from Source) S Where S.DR=2;**

### ****Scenario 2 :**** Fetching Nth Record from the table.

There are some situations where user needs to find out the Nth records from the table. I will divide this scenario in to 3 parts for better understanding of people.

Query 1 :  Query to find First Record from the table.

**Select \* from Employee where [Rownum](http://www.complexsql.com/rowid-rownum/" \t "_blank)=1;**

Query 2: Query to find last record from the table.

**Select \* from Employee where Rowid= select max(Rowid) from Employee;**

Query 3 : Query to find Nth Record from the table.

**select \* from ( select a.\*, rownum rnum from ( YOUR\_QUERY\_GOES\_HERE — including the order by ) a where rownum <= N\_ROWS ) where rnum >= N\_ROWS;**

### ****Scenario 3 : Find and delete duplicate rows****

There are real world situations where user needs to find and delete duplicate rows from the table. These are most used SQL queries in real world to find the duplicate rows and delete it. When there is a situation where user needs to add unique constraint to column,user needs to delete duplicate rows.

Query 1 :  Query to find duplicate rows.

**select a.\* from Employee a where rowid !=**  
**(select max(rowid) from Employee b where  a.Employee\_num =b.Employee\_num;**

Query 2: Query to delete duplicate rows

**Delete from Employee a where rowid !=** **(select max(rowid) from Employee b where  a.Employee\_num =b.Employee\_num;**

### Scenario 4 : Find a table specific information

There are times where user needs to find out the table specific information. There are so many[system tables](http://www.complexsql.com/oracle-system-tables/) which will find a table specific information.

Query 1: How to Find table name and its owner?

Make sure that the database user have logged in with SYS user.

**Select table\_name,Owner from All\_tables order by table\_name,owner;**

Query2:How to find Selected Tables from a User?

**SELECT Table\_Name FROM User\_Tables WHERE Table\_Name LIKE ‘STU%’;**

### Scenario 6: How to create a table which has same structure  or how to create duplicate table.

There are so many situations where user needs to create duplicate tables for testing purpose. There are some needs where user needs to create the structure of the table. The following are 2 most important queries which are used in 90% of Real Time Scenarios in SQL.

Query 1: Create the duplicate table with data

**Create table Employee\_1 as Select \* from Employee;**

Query 2: Create the table structure duplicate to another table.

**Create table Employee\_1 as Select \* from Employee where 1=2;**

### Scenario 8: Scenario of Self Join

We need to check the table which are joined with itself.There are the situations where user needs to join the table with itself. I will try to give one query which explains the scenario of self join.

Query : The query to find out the manager of employee

**Select e.employee\_name,m.employee name from Employee e,Employee m where e.Employee\_id=m.Manager\_id;**

Library Management System

**Tables**

We have four tables

* BookCategory
* Book
* Student
* BookBorrow

For a bit experienced resources, the tables are self-explanatory that we have a table of books. Each book belongs to particular book category. A student will borrow the book that will be mentioned in the bookBorrow table which will hold borrowed books information.

**Relationship Between Tables**

**Attachment**

The scripts of database tables along with data are attached.

**Queries  
  
Basic Queries**

The first and basic scenario would be *simple records fetching*.

1. **SELECT** \* **FROM** BOOK
2. **SELECT** \* **FROM** STUDENT
3. **SELECT** \* **FROM** BOOKCATEGORY
4. **SELECT** \* **FROM** BOOKBORROW

**Where clause**

If we want to restrict result or apply some filtration on data.

1. **SELECT** \* **FROM** BOOK **WHERE** BOOKID = 1
2. **SELECT** \* **FROM** STUDENT **WHERE** STUDENTID = 1
3. **SELECT** \* **FROM** BOOKCATEGORY **WHERE** BOOKCATEGORYNAME = 'DATABASE'
4. **SELECT** \* **FROM** BOOKBORROW **WHERE** STUDENTID = 3

**Order clause**

The arrangement of the result set in ascending and descending way.

1. **SELECT** \* **FROM** BOOK **ORDER** **BY** BOOKNAME, BOOKID **DESC**
2. **SELECT** \* **FROM** STUDENT **ORDER** **BY** STUDENTDESCIPLINE, STUDENTNAME
3. **SELECT** \* **FROM** BOOKCATEGORY **ORDER** **BY** BOOKCATEGORYNAME
4. **SELECT** \* **FROM** BOOKBORROW **ORDER** **BY** STUDENTID, BOOKID

**Join Result**

In most of the cases, we are required to fetch data from multiple tables. I have tried to cover few cases like

Fetch Student’s id, name along with book id and book borrow id (student name with book id borrowed by the student),

1. **SELECT** S.STUDENTID, S.STUDENTNAME, B.BOOKID, B.BOOKBORROWID
2. **FROM** STUDENT S
3. **INNER** JOIN BOOKBORROW B
4. **ON** S.STUDENTID = B.STUDENTID
5. **WHERE** S.STUDENTID = 1

Fetch Book name with their Category,

1. **SELECT** B.BOOKID, B.BOOKNAME, C.BOOKCATEGORYID, C.BOOKCATEGORYNAME
2. **FROM** BOOK B **INNER** JOIN BOOKCATEGORY C
3. **ON** B.BOOKCATEGORYID = C.BOOKCATEGORYID

Fetch student name, book name and book borrow id (the books borrowed by the student),

1. **SELECT** BB.BOOKBORROWID, BB.BOOKID, B.BOOKNAME, BB.STUDENTID, S.STUDENTNAME, BB.BORROWDATE, BB.BOOKRETURN,
2. BB.ISACTIVE
3. **FROM** BOOKBORROW BB **INNER** JOIN BOOK B
4. **ON** BB.BOOKID = B.BOOKID
5. **INNER** JOIN STUDENT S
6. **ON** S.STUDENTID = BB.STUDENTID

Fetch student name, book name, book category and book borrow id (the books with their category name, borrowed by the student),

1. **SELECT** BB.BOOKBORROWID, BB.BOOKID, B.BOOKNAME, BC.BOOKCATEGORYNAME, BB.STUDENTID, S.STUDENTNAME, BB.BORROWDATE, BB.BOOKRETURN, BB.ISACTIVE
2. **FROM** BOOKBORROW BB **INNER** JOIN BOOK B
3. **ON** BB.BOOKID = B.BOOKID
4. **INNER** JOIN STUDENT S
5. **ON** S.STUDENTID = BB.STUDENTID
6. **INNER** JOIN BOOKCATEGORY BC
7. **ON** BC.BOOKCATEGORYID = B.BOOKCATEGORYID

**Group By**

Often, we are required to apply an aggregated function on a result set to fetch data like count, sum or average. Following are some of the group queries.

Count of books borrowed by student id

1. **SELECT** BB.STUDENTID, COUNT(BB.BOOKID) **AS** BOOKCOUNT
2. **FROM** BOOKBORROW BB
3. **GROUP** **BY** BB.STUDENTID

Fetch book count against each book category,

1. **SELECT** BC.BOOKCATEGORYID, BC.BOOKCATEGORYNAME, COUNT(B.BOOKID) BOOKCATEGORYCOUNT
2. **FROM** BOOKCATEGORY BC
3. **INNER** JOIN BOOK B
4. **ON** B.BOOKCATEGORYID = BC.BOOKCATEGORYID
5. **GROUP** **BY** BC.BOOKCATEGORYID, BC.BOOKCATEGORYNAME

*Because of Inner join, the above query will return those categories count which has at least one record(one book borrow) in book borrow table. What if, we required all categories list irrespective of records in book borrow table. Here we will use left join instead of inner*

1. **SELECT** BC.BOOKCATEGORYID, BC.BOOKCATEGORYNAME, COUNT(B.BOOKID) BOOKCATEGORYCOUNT
2. **FROM** BOOKCATEGORY BC
3. LEFT JOIN BOOK B
4. **ON** B.BOOKCATEGORYID = BC.BOOKCATEGORYID
5. **GROUP** **BY** BC.BOOKCATEGORYID, BC.BOOKCATEGORYNAME

The output of both queries will be as follow,

The student may be from various discipline/ technologies. (Count Students in each discipline)

1. **SELECT** S.STUDENTDESCIPLINE, COUNT(S.STUDENTDESCIPLINE) DISCIPLINECOUNT
2. **FROM** STUDENT S
3. **GROUP** **BY** STUDENTDESCIPLINE

Fetch students name with their book borrows count (count of the books borrowed by students)

1. **SELECT** S.studentid,
2. studentname,
3. Count(BB.bookborrowid) STUDENTBOOKBORROWCOUNT
4. **FROM**   student S
5. LEFT JOIN bookborrow BB
6. **ON** S.studentid = BB.studentid
7. **GROUP**  **BY** S.studentid,
8. studentname

Would you like to guess, why I am using left join instead of inner one? Yes, you are right because there may be any student who has not borrowed a single book.  
  
**Having**

In case of filtration or restriction on group data (count, sum, average), we have to use “having” to apply filtration/restriction on aggregate data. Following are some of the having queries.

**Fetch Student id who has borrowed at least 1 book**

1. **SELECT** BB.studentid,
2. Count(BB.bookid) **AS** BOOKCOUNT
3. **FROM**   bookborrow BB
4. **GROUP**  **BY** BB.studentid
5. **HAVING** Count(BB.bookid) >= 1

**Fetch Student id who has borrowed at least 3 book**

1. **SELECT** BB.STUDENTID, COUNT(BB.BOOKID) **AS** BOOKCOUNT
2. **FROM** BOOKBORROW BB
3. **GROUP** **BY** BB.STUDENTID
4. **HAVING** COUNT(BB.BOOKID) >= 3

**Fetch book’s categories list containing more than 1 books**

1. **SELECT** BC.BOOKCATEGORYID, BC.BOOKCATEGORYNAME, COUNT(B.BOOKID) BOOKCATEGORYCOUNT
2. **FROM** BOOK B
3. **INNER** JOIN BOOKCATEGORY BC
4. **ON** B.BOOKCATEGORYID = BC.BOOKCATEGORYID
5. **WHERE** B.ISACTIVE = 1
6. **GROUP** **BY** BC.BOOKCATEGORYID, BC.BOOKCATEGORYNAME
7. **HAVING** COUNT(B.BOOKID) > 1

**Fetch book categories containing more than 3 books**

1. **SELECT** BC.bookcategoryid,
2. BC.bookcategoryname,
3. Count(B.bookid) BOOKCATEGORYCOUNT
4. **FROM**   book B
5. **INNER** JOIN bookcategory BC
6. **ON** B.bookcategoryid = BC.bookcategoryid
7. **WHERE**  B.isactive = 1
8. **GROUP**  **BY** BC.bookcategoryid,
9. BC.bookcategoryname
10. **HAVING** Count(B.bookid) > 3

**Fetch Categories from which more than 3 books have been borrowed**

1. **SELECT** BC.bookcategoryid,
2. BC.bookcategoryname,
3. Count(BB.bookid) CATEGORYBOOKBORROW
4. **FROM**   bookcategory BC
5. **INNER** JOIN book B
6. **ON** BC.bookcategoryid = B.bookcategoryid
7. **INNER** JOIN bookborrow BB
8. **ON** B.bookid = BB.bookid
9. **GROUP**  **BY** BC.bookcategoryid,
10. BC.bookcategoryname
11. **HAVING** Count(BB.bookid) > 3

**Not Exists**

The EXISTS operator is used to check the existence of any record in a subquery. The best place to use exists/ Not exists is when you have to look data in other tables but don’t fetch any data (column) from that table to output records.

**Fetch student who has not borrowed any book**

1. **SELECT** S.studentid,
2. S.studentname
3. **FROM**   student S
4. **WHERE**  NOT EXISTS (**SELECT** 1
5. **FROM**   bookborrow BB
6. **WHERE**  BB.studentid = S.studentid)

**Sub Query/ Sub Result**

Sometimes, we are required to fetch data where we have to write the query within another query. Some are as following.

**Fetch student borrow book in different Category Count. (In how many categories, student have borrowed book)**

1. **SELECT** A.studentid,
2. A.studentname,
3. Count(bookcategoryid) CATEGORYCOUNTBORROW
4. **FROM**   (**SELECT** **DISTINCT** S.studentid,
5. S.studentname,
6. BC.bookcategoryid
7. **FROM**   student S
8. LEFT JOIN bookborrow BB
9. **ON** S.studentid = BB.studentid
10. LEFT JOIN book B
11. **ON** B.bookid = BB.bookid
12. LEFT JOIN bookcategory BC
13. **ON** B.bookcategoryid = BC.bookcategoryid) A
14. **GROUP**  **BY** A.studentid,
15. A.studentname

**Fetch Student borrowed book in more than 1 categories (Student who has borrowed in at least two categories)**

1. **SELECT** A.studentid,
2. A.studentname,
3. Count(bookcategoryid) CATEGORYCOUNTBORROW
4. **FROM**   (**SELECT** **DISTINCT** S.studentid,
5. S.studentname,
6. BC.bookcategoryid
7. **FROM**   student S
8. LEFT JOIN bookborrow BB
9. **ON** S.studentid = BB.studentid
10. LEFT JOIN book B
11. **ON** B.bookid = BB.bookid
12. LEFT JOIN bookcategory BC
13. **ON** B.bookcategoryid = BC.bookcategoryid) A
14. **GROUP**  **BY** A.studentid,
15. A.studentname
16. **HAVING** Count(bookcategoryid) > 1

**Fetch Student borrow book in different Category Name concatenated (in comma separated)**

1. **SELECT** **DISTINCT** S.studentid,
2. S.studentname,
3. (**SELECT** **DISTINCT** bookcategoryname + ', '
4. **FROM**   bookcategory BC
5. LEFT JOIN book B
6. **ON** B.bookcategoryid = BC.bookcategoryid
7. LEFT JOIN bookborrow BB
8. **ON** B.bookid = BB.bookid
9. LEFT JOIN student SS
10. **ON** SS.studentid = BB.studentid
11. **WHERE**  SS.studentid = S.studentid
12. **FOR** xml path('')) CATEGORIES
13. **FROM**   student S

**Fetch Student carrying distinct book (student with all borrowed book name in comma separated)**

1. **SELECT** **DISTINCT** S.studentid,
2. S.studentname,
3. (**SELECT** **DISTINCT** B.bookname + ', '
4. **FROM**   book B
5. LEFT JOIN bookborrow BB
6. **ON** B.bookid = BB.bookid
7. LEFT JOIN student SS
8. **ON** SS.studentid = BB.studentid
9. **WHERE**  SS.studentid = S.studentid
10. **FOR** xml path('')) BOOKNAME
11. **FROM**   student S

**Fetch a list of a book along with students who have borrowed the book in comma separated manner.**

1. **SELECT** B.bookname,
2. (**SELECT** **DISTINCT** S.studentname + ', '
3. **FROM**   student S
4. LEFT JOIN bookborrow BB
5. **ON** S.studentid = BB.studentid
6. **WHERE**  B.bookid = BB.bookid
7. **FOR** xml path('')) STUDENTNAME
8. **FROM**   book B

**NOT IN**

The “IN/ NOT IN” is also used when we have to look data in other tables but don’t want any column or record to fetch from that table to display.

**Fetch students who did not borrow the book.**

1. **SELECT** \*
2. **FROM**   student S
3. **WHERE**  studentid NOT IN (**SELECT** **DISTINCT** studentid
4. **FROM**   bookborrow)

**Fetch Book which is not borrowed by anyone.**

1. **SELECT** \*
2. **FROM**   book B
3. **WHERE**  B.bookid NOT IN (**SELECT** **DISTINCT** bookid
4. **FROM**   bookborrow)

**Note**  
Didn’t you find similarity between “Not in” and “Not Exists”?

Yes, but there is a difference between both of them. unlike IN and Exists, they are not equal in all the cases especially when a null value is involved. when the subquery returns even one null, NOT IN will not match any rows.Because inside of not in, its use AND logical gate.According to AND gate, all the options/values must be true but if its find any NULL value it returns UNKNOWN.

Hence, if any row of that subquery returns NULL, the entire NOT IN operator will evaluate to either FALSE or NULL and no records will be returned.On the other side, Exists cannot return null. It will only return true or false by checking presence or absence of a row.

For example, the below queries will display different result because, in book's table, I have added NULL book name.

1. **SELECT** \*
2. **FROM**   book B
3. **WHERE**  NOT EXISTS (**SELECT** bookname
4. **FROM**   book
5. **WHERE**  bookname = B.bookname)
7. **SELECT** \*
8. **FROM**   book B
9. **WHERE**  B.bookname NOT IN (**SELECT** bookname
10. **FROM**   book
11. **WHERE**  bookid = B.bookid)

**Rank**

Ranking functions are those who return/associates a ranking number (Value) for each row in a partition. For example, if I want to fetch records of most borrow book name and count. The following query will fetch the result.

1. **SELECT** **TOP** 1 BB.bookid,
2. B.bookname,
3. Count(BB.bookid) BORROWCOUNT
4. **FROM**   book B
5. LEFT JOIN bookborrow BB
6. **ON** B.bookid = BB.bookid
7. **GROUP**  **BY** BB.bookid,
8. B.bookname
9. **ORDER**  **BY** Count(BB.bookid) **DESC**

***But what if I have more than 1 same book count. At this moment, we will have to implement rank.***

1. **SELECT** \*
2. **FROM**   (**SELECT** BB.bookid,
3. B.bookname,
4. Count(BB.bookid)                    BORROWCOUNT,
5. Rank()
6. OVER (
7. **ORDER** **BY** Count(BB.bookid) **DESC**) **AS** RANK1
8. **FROM**   book B
9. LEFT JOIN bookborrow BB
10. **ON** B.bookid = BB.bookid
11. **GROUP**  **BY** BB.bookid,
12. B.bookname) TBL
13. **WHERE**  rank1 = 1

**Fetch top 3 most borrow book**

1. **SELECT** **TOP** 3 BB.bookid,
2. B.bookname,
3. Count(BB.bookid) BORROWCOUNT
4. **FROM**   book B
5. LEFT JOIN bookborrow BB
6. **ON** B.bookid = BB.bookid
7. **GROUP**  **BY** BB.bookid,
8. B.bookname
9. **ORDER**  **BY** borrowcount **DESC**

**Fetch Person borrow most book**

1. **SELECT** **TOP** 1 S.studentname,
2. Count(BB.bookid) BORROWCOUNT
3. **FROM**   student S
4. LEFT JOIN bookborrow BB
5. **ON** BB.studentid = S.studentid
6. **GROUP**  **BY** S.studentname
7. **ORDER**  **BY** borrowcount **DESC**

***Again, what if we have more than 1 person borrowing same books. The appropriate query would be.***

1. **SELECT** \*
2. **FROM**   (**SELECT** S.studentname,
3. Count(BB.bookid)                    BORROWCOUNT,
4. Rank()
5. OVER (
6. **ORDER** **BY** Count(BB.bookid) **DESC**) **AS** RANKVALUE
7. **FROM**   student S
8. LEFT JOIN bookborrow BB
9. **ON** BB.studentid = S.studentid
10. **GROUP**  **BY** S.studentname)TBL
11. **WHERE**  TBL.rankvalue = 1
12. **ORDER**  **BY** TBL.borrowcount **DESC**

**Fetch least borrow book Name and count**

1. **SELECT** \*
2. **FROM**   (**SELECT** BB.bookid,
3. B.bookname,
4. Count(BB.bookid)                BORROWCOUNT,
5. Rank()
6. OVER (
7. **ORDER** **BY** Count(BB.bookid) ) **AS** RANK1
8. **FROM**   book B
9. LEFT JOIN bookborrow BB
10. **ON** B.bookid = BB.bookid
11. **GROUP**  **BY** BB.bookid,
12. B.bookname) TBL
13. **WHERE**  rank1 = 1

**Fetch top 3 least books, borrow by student**

1. **SELECT** \*
2. **FROM**   (**SELECT** BB.bookid,
3. B.bookname,
4. Count(BB.bookid)                BORROWCOUNT,
5. Rank()
6. OVER (
7. **ORDER** **BY** Count(BB.bookid) ) **AS** RANK1
8. **FROM**   book B
9. LEFT JOIN bookborrow BB
10. **ON** B.bookid = BB.bookid
11. **GROUP**  **BY** BB.bookid,
12. B.bookname) TBL
13. **WHERE**  rank1 <= 3

**Fetch least 1 person who has borrow book (or didn’t borrow any book)**

1. **SELECT** \*
2. **FROM**   (**SELECT** S.studentname,
3. Count(BB.bookid)               BORROWCOUNT,
4. Rank()
5. OVER (
6. **ORDER** **BY** Count(BB.bookid)) **AS** RANKVALUE
7. **FROM**   student S
8. LEFT JOIN bookborrow BB
9. **ON** BB.studentid = S.studentid
10. **GROUP**  **BY** S.studentname)TBL
11. **WHERE**  TBL.rankvalue = 1
12. **ORDER**  **BY** TBL.borrowcount **DESC**

**Fetch list of books which at least one student has borrowed**

1. **SELECT** B.bookid,
2. B.bookname,
3. Count(BB.bookid) **AS** BOOKBORROWCOUNT
4. **FROM**   book B
5. **INNER** JOIN bookborrow BB
6. **ON** B.bookid = BB.bookid
7. **GROUP**  **BY** B.bookid,
8. B.bookname
9. **HAVING** Count(BB.bookid) > 1
10. **ORDER**  **BY** Count(BB.bookid) **DESC**

**Fetch most book borrowed by categories(Count all book against each category)**

1. **SELECT** \*
2. **FROM**   (**SELECT** BC.bookcategoryid,
3. BC.bookcategoryname,
4. Count(BB.bookid)                    BOOKBORROWCOUNT,
5. Rank()
6. OVER (
7. **ORDER** **BY** Count(BB.bookid) **DESC**) RANKCOUNT
8. **FROM**   bookcategory BC
9. LEFT JOIN book B
10. **ON** B.bookcategoryid = BC.bookcategoryid
11. LEFT JOIN bookborrow BB
12. **ON** BB.bookid = B.bookid
13. **GROUP**  **BY** BC.bookcategoryid,
14. BC.bookcategoryname) TBL
15. **WHERE**  TBL.rankcount = 1

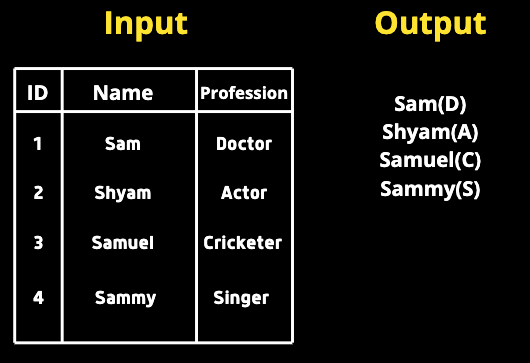
**Fetch the least book borrowed by category**

1. **SELECT** \*
2. **FROM**   (**SELECT** BC.bookcategoryid,
3. BC.bookcategoryname,
4. Count(BB.bookid)               BOOKBORROWCOUNT,
5. Rank()
6. OVER (
7. **ORDER** **BY** Count(BB.bookid)) RANKCOUNT
8. **FROM**   bookcategory BC
9. LEFT JOIN book B
10. **ON** B.bookcategoryid = BC.bookcategoryid
11. LEFT JOIN bookborrow BB
12. **ON** BB.bookid = B.bookid
13. **GROUP**  **BY** BC.bookcategoryid,
14. BC.bookcategoryname) TBL
15. **WHERE**  TBL.rankcount = 1

You can explore more dimension out of the discussed relational database. More queries can be written with multiple condition and criteria.

### Query 1

We are given a table consisting of two columns, **Name,** and**Profession**. We need to query all the names immediately followed by the first letter in the profession column enclosed in parenthesis.



**My Solution**

**SELECT**

**CONCAT**(Name, ’(‘, **SUBSTR**(Profession, 1, 1), ’)’)

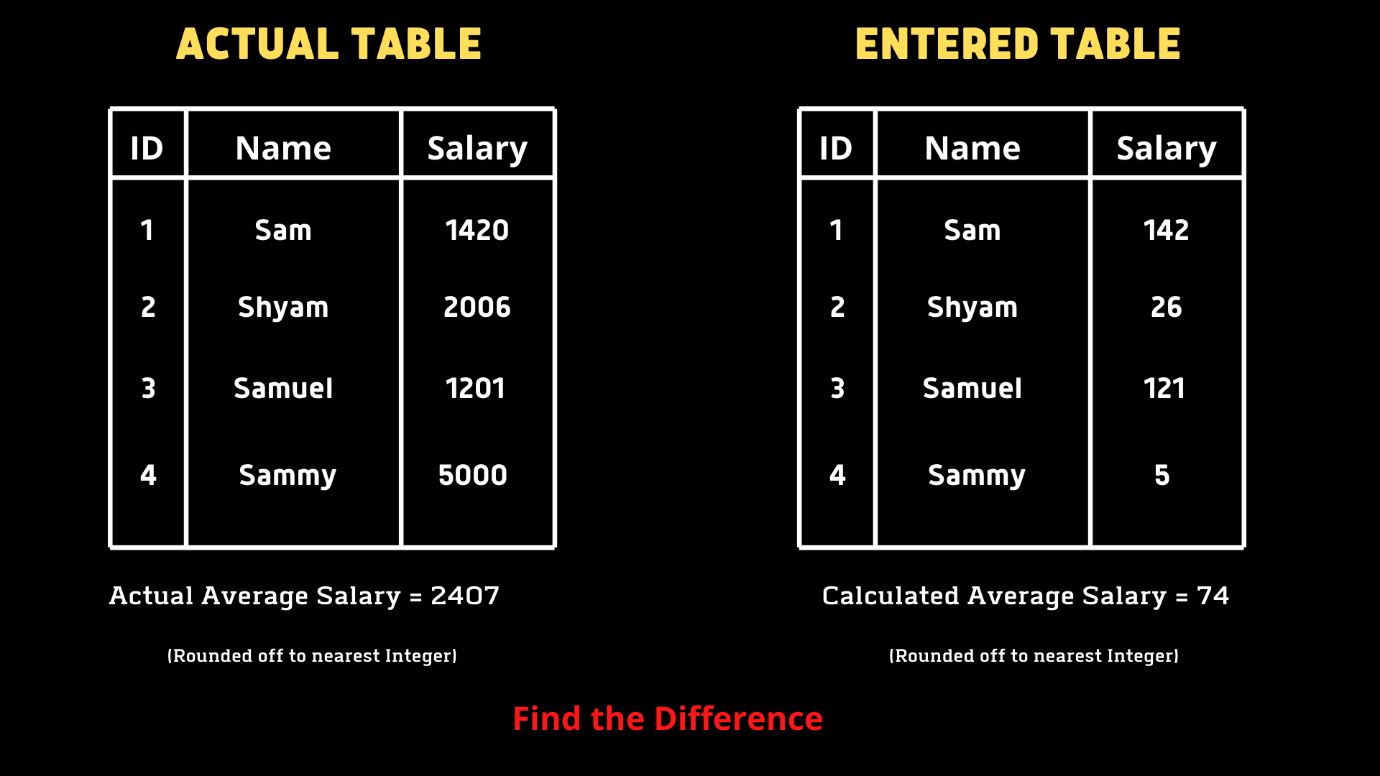
**FROM** table;

Since we need to combine the name and profession we can useCONCAT. We also need to have only one letter inside the parenthesis. Hence we will use SUBSTRand pass the column name, start index, end index. Since we need only the first letter we will pass 1,1(start index is inclusive and the end index is not inclusive)

### Query 2

Tina was asked to compute the average salary of all employees from the EMPLOYEES table she created but realized that the zero key in her keyboard is not working after the result showed a very less average. She wants our help in finding out the difference between miscalculated average and actual average.

We must write a query finding the error( Actual AVG — Calculated AVG).



**My Solution**

**SELECT**

**AVG**(Salary) - **AVG**(**REPLACE**(Salary, 0, ’’))

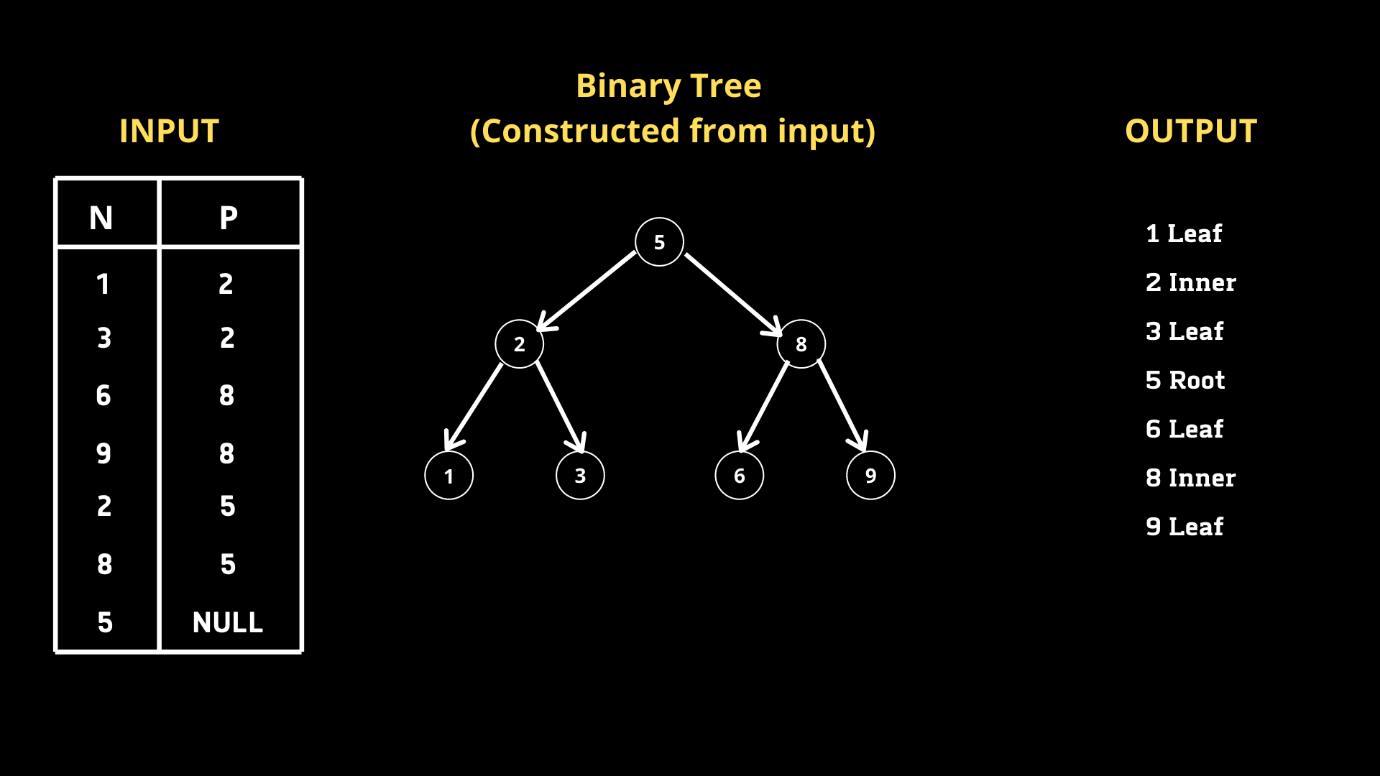
**FROM** table;

A point to note here is that we have only one table that consists of actual salary values. To create the error scenario we use REPLACE to replace 0’s. We will pass the column name, value to be replaced, and the value with which we will replace the REPLACE method. Then we find the difference in averages using the aggregate function AVG.

### Query 3

We are given a table, which is a [Binary Search Tree](https://en.wikipedia.org/wiki/Binary_search_tree) consisting of two columns **Node** and **Parent.**We must write a query that returns the node type ordered by the value of nodes in ascending order. There are 3 types.

1. Root — if the node is a root
2. Leaf — if the node is a leaf
3. Inner — if the node is neither root nor leaf.



**My Solution**

Upon initial analysis, we can conclude that if a given node N has its corresponding P-value as NULL it is the root. And for a given Node N if it exists in the P column it is not an inner node. Based on this idea let us write a query.

**SELECT CASE**

**WHEN** P **IS NULL** **THEN** **CONCAT**(N, ' Root')

**WHEN** N **IN** **(SELECT DISTINCT** P from BST) **THEN** **CONCAT**(N, ' Inner')

**ELSE** **CONCAT**(N, ' Leaf')

**END**

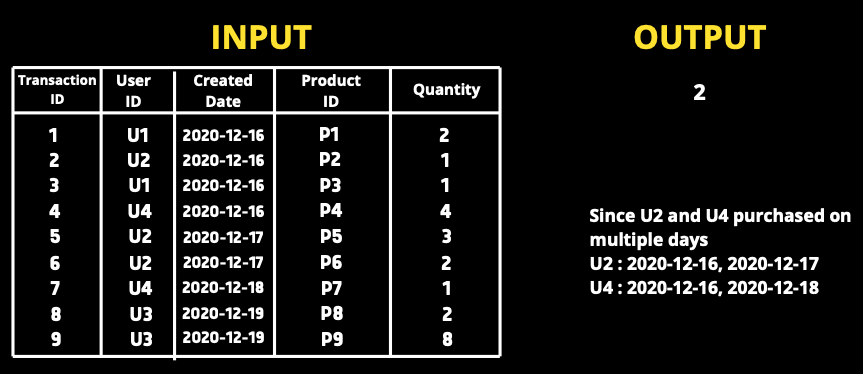
**FROM** BST

**ORDER BY** N asc;

We can use CASE which acts as a switch function. As I mentioned if P is null for a given node N then N is the root. Hence we usedCONCAT for combining the node value and label. Similarly, if a given node N is in column P it is an inner node. To get all nodes from column P we wrote a subquery which returns all the distinct nodes in column P. Since we were asked to order the output by node values in ascending order we used the ORDER BY Clause.

### Query 4

We are given a transaction table that consists of **transaction\_id, user\_id, transaction\_date, product\_id, and quantity**. We need to query the number of users who purchased products on multiple days(Note that a given user can purchase multiple products on a single day).



**My Solution**

To solve this query, we cannot directly count the occurrence of user\_id’s and if it is more than one return that user\_id because a given user can have more than one transaction on a single day. Hence if a given user\_id has more than one distinct date associated with it means he purchased products on multiple days. Following the same approach, I wrote a query. (Inner query)

**SELECT COUNT**(user\_id)

**FROM**

(

**SELECT** user\_id

**FROM** orders

**GROUP BY** user\_id

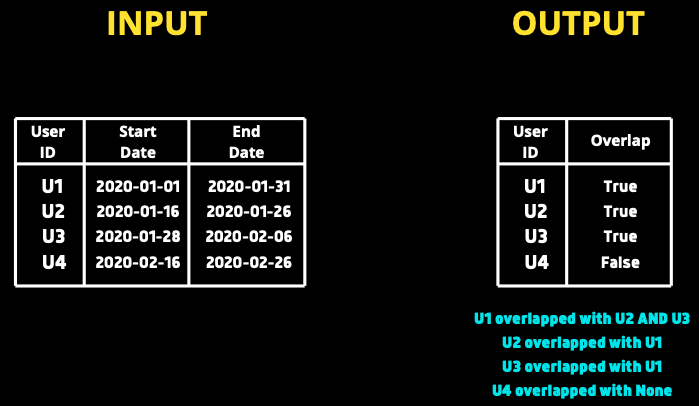
**HAVING COUNT**(**DISTINCT DATE**(date)) > 1

) t1

Since the question asked for the number of user\_ids and not the user\_id’s itself we use COUNT in the outer query.

### Query 5

We are given a subscription table which consists of subscription start and end date for each user. We need to write a query that returns true/false for each user based on the overlapping of dates with other users. For instance, If user1's subscription period overlaps with any other user the query must return **True** for user1.



**My Solution**

Upon initial analysis, we understand that we must compare every subscription against every other one. Let us consider start and end dates of **userA** as **startA** and **endA**, similarly for **userB**,**startB** and **endB.**

If **startA≤endB and endA≥startB** then we can say the two date ranges overlap. Let us take two examples. Let us compare U1 AND U3 first.

startA = 2020–01–01  
endA = 2020–01–31  
startB = 2020–01–16  
endB = 2020–01–26

Here we can see **startA**(2020–01–01) is less than **endB(**2020–01–26) and similarly, **endA**(2020–01–31) is greater than **startB**(2020–01–16) and hence can conclude that the dates overlap. Similarly, if you compare U1 and U4 the above condition fails and will return false.

We must also ensure that a user is not compared to his own subscription. We also want to run a left join on itself to match a user with each other user that satisfies our condition. We will create two replicas s1 and s2 of the same table now.

**SELECT** \*

**FROM** subscriptions **AS** s1

**LEFT** **JOIN** subscriptions **AS** s2

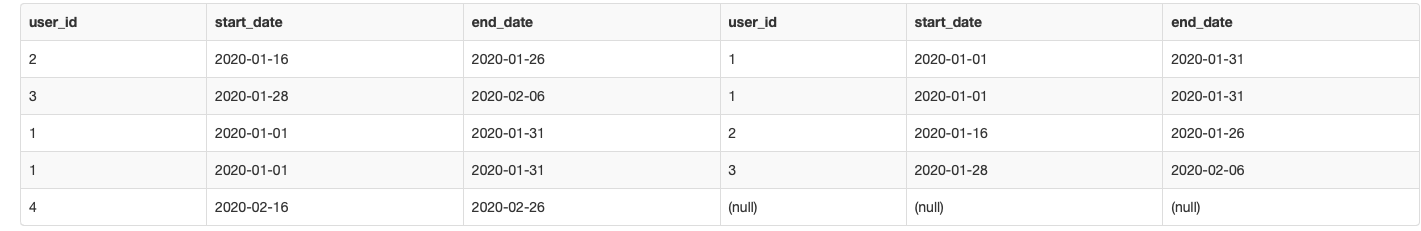
**ON** s1.user\_id != s2.user\_id

AND s1.start\_date <= s2.end\_date

AND s1.end\_date >= s2.start\_date

Given the conditional join, a user\_id from s2 should exist for each user\_id in s1 on the condition where there exists an overlap between the dates.

**Output**



We can see there exists another user for each user in case the dates overlap. For user1 there are 2 rows indicating that he matches with 2 users. For user 4 the corresponding id is null indicating that he does not match with any other user.

Wrapping it all together now, we can group by the s1.user\_id field and just check if any value exists true for a user where s2.user\_id IS NOT NULL.

**Final query**

**SELECT**

s1.user\_id

, (**CASE** **WHEN** s2.user\_id **IS** NOT NULL **THEN** 1 **ELSE** 0 **END**) **AS** overlap

**FROM** subscriptions **AS** s1

**LEFT** **JOIN** subscriptions **AS** s2

**ON** s1.user\_id != s2.user\_id

AND s1.start\_date <= s2.end\_date

AND s1.end\_date >= s2.start\_date

**GROUP** **BY** s1.user\_id

We used the CASE clause to label 1 and 0 depending on the s2.user\_id value for a given user. The final output looks like this -

