Friday, March 12, 2021

SQL interview questions

This note is based on source.

What is the difference between UNION and UNION ALL?

- UNION joins datasets and removes the duplicated entries. Essentially apply SELECT DISTINCT to the combined datasets.
- UNION ALL joins the datasets without performing the work of de-duplication. Less intensive merging process, since the tables are simply merged regardless of the overlapping data.

What is the difference between inner and outer join?

- INNER JOIN returns the rows that matched in two tables based on the join predicate
- LEFT OUTER JOIN / LEFT JOIN returns all the rows that in the left table, regardless of whether the row has a match in the right table
- RIGHT OUTER JOIN / RIGHT JOIN returns all the rows that in the right table
- FULL OUTER JOIN / FULL JOIN returns all the rows in both tables
  - Rows that don't match will be filled with NULL

Definition of a key?

- A subset of columns in a table that allow a row to be uniquely identified
- Can be more than one column
- Each and every row has a unique value or a unique combination of values

Can a key have NULL values in SQL?

- NO according to SQL standard
- YES for foreign and unique keys, and NO for primary key in actual RDBMS implementations

Difference between primary, foreign and unique keys in SQL?

- One thing in common: each type of key can consist of more than just one column
- Difference:
  - A table can have multiple foreign and unique keys, while it can only have one primary key
  - Foreign and unique keys can hold NULL values, while primary key can never be NULL
  - Foreign key can reference a primary key or unique keys (non-primary keys)

- Primary and unique keys both enforce uniqueness on column(s), while foreign keys defines the relationship between two tables. A foreign key identifies column(s) in the referencing table that refers to column(s) in the referenced table.

What is natural key and surrogate key?

- A natural key is key composed of columns that have logical relationships to other columns within a table. The columns that belong to natural key are just naturally a part of the table and have a relationships with other columns in the table.
  - Also called business keys and domain keys
- Surrogate key is unnatural, the columns belong to the key does not have logical relationship to other columns in the table.
  - They are often being considered as artificial keys
  - They are usually just simple sequential numbers, where each number identifies a row. (Act as a primary key in most databases)

Simple key?

A simple key is a single attribute/column that can uniquely identify a row.

Secondary key?

A given table can have more than one option of primary keys. Any combination of columns that qualifies to be a primary key are candidates keys. The candidates that are not selected as primary keys are called secondary keys.

Superkey?

A set of columns in a table for which each row has unique combination of values.

- There are no two rows sharing the same combination of values
- The superkey is unique for each and every row
- Essentially all the superset combinations of keys that can uniquely identify each row

Minimal superkey?

Minimum number of columns that can be used to uniquely identify each row.

- A table can have multiple minimal superkeys: employee id, SSN
- Are candidate keys

Referential integrity?

A relational database concept that multiple tables share a relationship based on the data store in the tables, and that relationship must remain consistent.

- No row or record can be added to the referencing table unless the foreign key for that row points to an existing primary key in the referenced table.
- If a row/record is deleted in the referenced table, then all the corresponding matching rows in the referencing table must be deleted using a cascading delete.
- If the primary key for the referenced table changes, all the corresponding matching rows in the referencing table must be updated using a cascading update

What is the difference between HAVING and WHERE clause?

WHERE clause can not be used with aggregates (SUM, AVG, MAX, MIN…) while HAVING can.

- HAVING allows us to compare aggregates to other values

How do database indexes work? How do indexes help?

The whole point of having an index is to speed up search queries by essentially cutting down the number of records/rows in a table that need to be examined.

- Index: a data structure that stores the values for a specific column in a table. An index is created on a column of a table. (Index consists of column values of that specific column)
- Index data structure
  - B- trees (can be sorted): most commonly used
  - Hash table (can not be sorted):
    - extremely fast when looking up for values
    - hash index works in a way that the column value will be the key into the hash table and the actual values mapped to that key would just be a pointer to the row.
- An index also stores a pointer to the table row

How does a database know when to use an index?

When a column in a query has index created on it, the database will have to decide to use the index or not to find the values being searched. (Some scenarios it is less efficient to use database index)

Can you force database to use an index on a query?

Yes in most databases (Oracle, MySQL), you can specify that you want the index to be used.

What is a good analogy for a database index?

Similar to book index contains a page number, a database index contains a pointer to the row containing the value you are searching for in SQL.

What is the cost of having a database index?

- Takes up memory
- Same operation has to be done on the index whenever you change or update the table, index needs to contain the same up to the minute data as whatever is in the table columns that the index covers

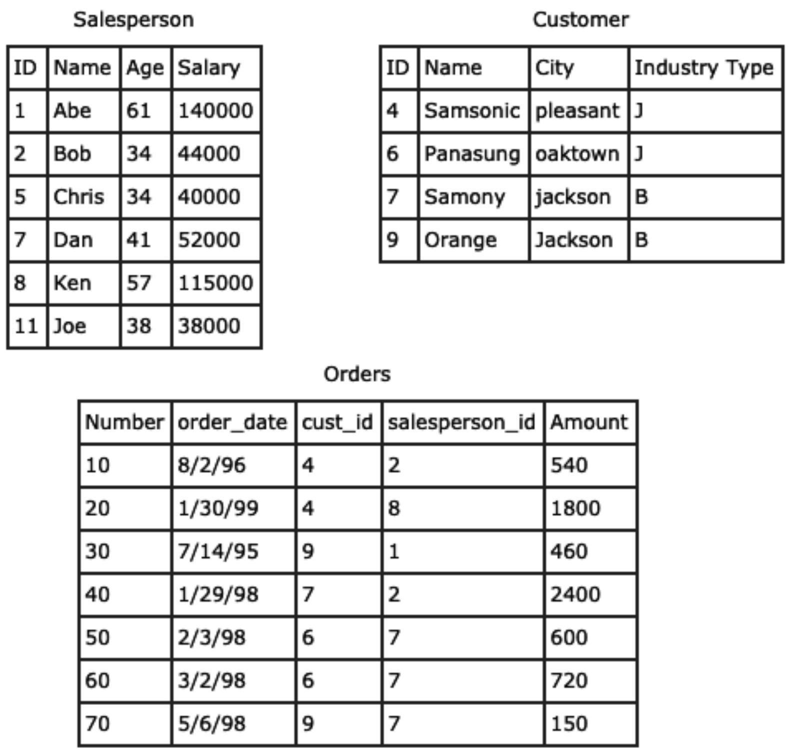
What is a self JOIN?

A self join is when a table is joined to itself. A join is performed between two identical copy of that table.

- Must have alias
- Predicate

Are self joins and inner joins the same?

No. Inner join is separate concept entirely from a self join. A self join can be inner or outer joins based on what the query needs.



Salesperson

ID	Name	Age	Salary
1	Abe	61	140000
2	Bob	34	44000
5	Chris	34	40000
7	Dan	41	52000
8	Ken	57	115000
11	Joe	38	38000

Customer

ID	Name	City	Industry Type
4	Samsonic	pleasant	J
6	Panasung	oaktown	J
7	Samony	jackson	B
9	Orange	Jackson	B

Orders

Number	order_date	cust_id	salesperson_id	Amount
10	8/2/96	4	2	540
20	1/30/99	4	8	1800
30	7/14/95	9	1	460
40	1/29/98	7	2	2400
50	2/3/98	6	7	600
60	3/2/98	6	7	720
70	5/6/98	9	7	150

Practice problem 1

Given the tables above, find the following:

a. The names of all salespeople that have an order with Samsonic.

b. The names of all salespeople that do not have any order with Samsonic.

c. The names of salespeople that have 2 or more orders.

d. Write a SQL statement to insert rows into a table called highAchiever(Name, Age), where a salesperson must have a salary of 100,000 or greater to be included in the table.

a.

```
SELECT Salesperson.Name FROM Salesperson, Orders

WHERE Salesperson.ID = Orders.salesperson_id

AND Orders.cust_id = (SELECT Customer.ID FROM Customer WHERE
Customer.Name = ‘Samsonic’)
```

b.

```
SELECT Salesperson.Name FROM Salesperson

WHERE Salesperson.ID NOT IN (

              SELECT Salesperson.ID FROM Orders, Customers

              WHERE Orders.cust_id = Customer.ID

              AND Customer.Name = ‘Samsonic’)
```

c.

```
SELECT name

  FROM Orders, Salesperson

WHERE Orders.salesperson_id = Salesperson.id

GROUP BY name, salesperson_id

HAVING COUNT( salesperson_id ) >1
```

d.

```
INSERT INTO highAchiever (name, age)

(SELECT name, age FROM Salesperson WHERE salary >= 100000)
```

Note: a regular insert, INSERT INTO highAchiever (name, age) VALUES (‘James’, 18)

Practice problem 2

This question was asked in a Google interview: Given the 2 tables below, User and UserHistory:

User user_id name phone_num
UserHistory user_id date action

1. Write a SQL query that returns the name, phone number and most recent date for any user that has logged in over the last 30 days (you can tell a user has logged in if the action field in UserHistory is set to "logged_on").

Every time a user logs in a new row is inserted into the UserHistory table with user_id, current date and action (where action = “logged_on").

Solving process:

- Join two tables
- Set two conditions
  - Date within 30 days (date_sub)
  - Action is logged_on
- Select the name, phone number and the max(date)
- group by user_id, name and phone number

Query:

```
SELECT User.name, User.phone_num, max(UserHistory.date)
  FROM User, UserHistory
WHERE  User.user_id = UserHistory.user_id
    AND  UserHistory.date >= DATE_SUB(CURDATE( ), INTERVAL 30 day)
    AND  UserHistoty.action = ‘logged_on’
GROUP BY User.user_id, User.name, User.phone_num
```

2. Write a SQL query to determine which user_ids in the User table are not contained in the UserHistory table (assume the UserHistory table has a subset of the user_ids in User table). Do not use the SQL MINUS statement. Note: the UserHistory table can have multiple entries for each user_id.

Note that your SQL should be compatible with MySQL 5.0, and avoid using subqueries.

Solving process:

- Left Join two tables, this will only retain the user_id in the User table
- Set condition: UserHistory.user_id null, which represents the user_ids that are not in the User table

Query:

```
SELECT DISTINCT u.user_id

FROM user AS u

LEFT JOIN userhistory AS uh

   ON  u.user_id = uh.user_id

WHERE uh.user_id IS NULL
```

Practice problem 3

Suppose we have 2 tables called Orders and Salesperson shown below:

Salesperson

ID	Name	Age	Salary
1	Abe	61	140000
2	Bob	34	44000
5	Chris	34	40000
7	Dan	41	52000
8	Ken	57	115000
11	Joe	38	38000

Orders

Number	order_date	cust_id	salesperson_id	Amount
10	8/2/96	4	2	540
20	1/30/99	4	8	1800
30	7/14/95	9	1	460
40	1/29/98	7	2	2400
50	2/3/98	6	7	600
60	3/2/98	6	7	720
70	5/6/98	9	7	150

Now suppose that we want to write SQL that must conform to the SQL standard.

We want to retrieve the names of all salespeople that have more than 1 order from the tables above. You can assume that each salesperson only has one ID.

If that is the case, then what (if anything) is wrong with the following SQL?

```
SELECT Name
FROM Orders, Salesperson
WHERE Orders.salesperson_id = Salesperson.ID
GROUP BY salesperson_id
HAVING COUNT( salesperson_id ) >1
```

The problem is that the SQL Standard says that we can not select a column that is not part of the group by clause unless it is also contained within an aggregate function.

Column 'Name' is invalid in the select list because it is not contained in either an aggregate function or the GROUP BY clause. We can fix that by either of the following modification.

```
SELECT Name
FROM Orders, Salesperson
WHERE Orders.salesperson_id = Salesperson.ID
GROUP BY salesperson_id, Name
HAVING COUNT( salesperson_id ) >1
```

```
SELECT MAX(Name) --put name in an aggregate function
FROM Orders, Salesperson
WHERE Orders.salesperson_id = Salesperson.ID
GROUP BY salesperson_id
HAVING COUNT( salesperson_id ) >1
```

Practice problem 4

Suppose we have some tables called Starbucks_Stores and Starbucks_Employees. In case you don't already know, Starbucks is a popular coffee shop/cafe in the USA:

Starbucks_Employees

ID	Name	Age	HourlyRate	StoreID
1	Abe	61	14	10
2	Bob	34	10	30
5	Chris	34	9	40
7	Dan	41	11	50
8	Ken	57	11	60
11	Joe	38	13	70

Starbucks_Stores

store_id	city
10	San Francisco
20	Los Angeles
30	San Francisco
40	Los Angeles
50	San Francisco
60	New York
70	San Francisco

Now, given the tables above let's say that we write some SQL like this:

```
SELECT count(*) as num_employees, HourlyRate
FROM Starbucks_Employees JOIN Starbucks_Stores
```

```
ON Starbucks_Employees.StoreID = Starbucks_Stores.store_id
GROUP BY city
```

What does the query want to do?

1. The 2 tables are joined on the condition that the Starbucks_Employees.StoreID column value is equal to the Starbucks_Stores.store_id column values.

2. Groups are then created for each city - which means that each distinct city will have it's own "group". So, there will be a total of 3 groups one each for San Francisco, New York, and Los Angeles.

3. The data we are interested in is selected from each group that is created in step 2.

What is the problem?

HourlyRate is not in an aggregate function or in the GROUP BY clause.

The real problem is that when SQL try to return the count of number employees and hourly rate grouped by city, it is confused regarding what to return for the Hourly rate, since for each city group, there might be multiple values. We can fix the query by:

```
SELECT count(*) as num_employees, MAX(HourlyRate)
FROM Starbucks_Employees JOIN Starbucks_Stores
ON Starbucks_Employees.StoreID = Starbucks_Stores.store_id
GROUP BY city
```

In this case, for each city group, the maximum HourlyRate will be returned.

```
SELECT count(*) as num_employees, HourlyRate
FROM Starbucks_Employees JOIN Starbucks_Stores
ON Starbucks_Employees.StoreID = Starbucks_Stores.store_id
GROUP BY city, HourlyRate
```

In this case, the turned data will grouped by city and hourly rate (combination).

MySQL – selecting non-aggregate columns not in the group by

MySQL allows having non-aggregated columns (columns that are not wrapped within an aggregate function) in the select list even if they are not part of the group by clause.

So we will not get error in MySQL for practice question 4 and 5. The code in Q4 will work just fine in MySQL since there is a one to one mapping between salesperson name to ID. Thus when we create groups based on ids, for each group will only have one and only one name.

While for Q5, even if MySQL does not return error, for each city group the returned Hourly rate would be random since there might be multiple values.

Reminder of Aggregate functions

```
AVG() - Returns the average value
COUNT() - Returns the number of rows
FIRST() - Returns the first value
LAST() - Returns the last value
MAX() - Returns the largest value
MIN() - Returns the smallest value
SUM() - Returns the sum
```