Essential SQL Server Interview Questions

Q. Is the following a valid SQL query? Why or why not?

SELECT TOP 5 YEAR(BillingDate) AS BillingYear, COUNT(\*) AS NumberOfInvoices

FROM Invoices

WHERE CustomerId = 42

GROUP BY YEAR(BillingDate)

HAVING COUNT(\*) > 1

ORDER BY BillingYear;

As part of your answer, list the phrases of this statement in the order that SQL Server logically processes them.

A. One might think that this SQL statement is invalid since it uses the alias BillingYear in the ORDER BY phrase. Even though BillingYear cannot be used in the WHERE or GROUP BY phrases, it can be used in the ORDER BY phrase.

Listing the phrases in the logical order in which they are processed helps make clear why this is so:

FROM Invoices

WHERE CustomerId = 42

GROUP BY YEAR(BillingDate)

HAVING COUNT(\*) > 1

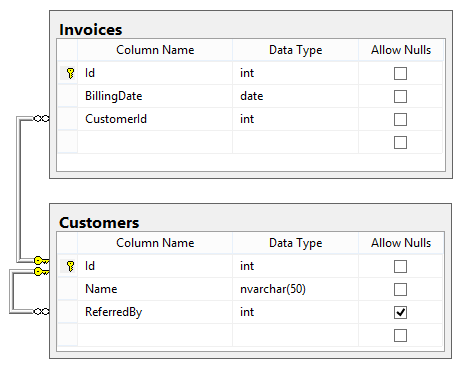
SELECT YEAR(BillingDate) AS BillingYear, COUNT(\*) AS NumberOfInvoices

ORDER BY BillingYear

TOP 5

Although the reordered statement is not valid T-SQL syntax, it is critical for candidates to be able to reorder the statement in their heads so they can work out any problems they are having when composing complex queries. Most T-SQL programmers never need to explicitly memorize this processing order. There is a logic to it that becomes very familiar with experience, so this question is designed to gauge that experience in a candidate.

Q. Considering the database schema displayed in the diagram below, write a SQL query which lists all invoices billed to the customer who has the highest number of invoices. Note that there may be more than one customer tied for most invoices, in which case the invoices for all of them should be listed.



A. The point here is to see if the candidate can break down a complex query into simpler parts. The first step is to find out who the customer with the highest number of invoices is, taking into account there may be a tie:

SELECT TOP 1 WITH TIES CustomerId

FROM Invoices

GROUP BY CustomerId

ORDER BY COUNT(\*) DESC;

Now that we know who the “winner(s)” are, we can query for that customer(s) orders. In order to query the orders of a list of customers we know by ID, we can use the IN operator:

SELECT \*

FROM Invoices

WHERE CustomerId IN (…);

So now inserting the first query as a subquery into the second one, we get:

SELECT \* FROM Invoices

WHERE CustomerId IN (

SELECT TOP 1 WITH TIES CustomerId

FROM Invoices

GROUP BY CustomerId

ORDER BY COUNT(\*) DESC

);

Extremely complicated queries can be broken down into subqueries in this fashion, then used as building blocks to produce the final query.

Q. Can records be deleted from a View in SQL Server?

A. It depends.

There are two types of Views in SQL Server. One is a “simple” view that contains data from one table only, and the other is a “complex” view that contains data from multiple tables.

A delete operation can be performed on records in a simple view, but not in a complex view.

Q. What is the difference between the Merge Join and Lookup components in SSIS in SQL Server?

A. The use of Merge Join and Lookup components is based on the type of data relationship between two tables, i.e. a one-to-one relationship or a one-to-many relationship.

The Merge Join is used for joining data from two or more datasources and inserting the merged data into one denormalized target datasource. The Merge Join component takes two input sources and the merged data is transfered to a single target data output.

The Lookup component is used in SSIS where there is for one input record one matched record in the target source. Both the source and target satisfy the one-to-one relation.

Q. What would be the SQL Server command to get the position of the letter ‘o’ in the name ‘John’ from an Employee table

A. Select CHARINDEX('o',FIRST\_NAME,0) from employee where first\_name='John'

CHARINDEX is the SQL Server Equivalent of INSTR in Oracle.

Q. When should a developer use the NOLOCK hint? What problems can happen when using this hint and how can these problems be addressed?

NOLOCK improves performance by not locking rows as they are read/processed by SQL. It should be used when accessing large tables and tables with data that rarely changes.

If too many locks are created, row-level locks can be escalated to block-level then table locks. Table locks especially on core tables will pause most processing on that database instance until that SQL operation completes.

NOLOCK allows the query to access both committed and uncommitted data and may return dirty data. Dirty data can lead to data integrity issues if a system is not coded to handle this situation.

Data integrity issues can be avoided by adding a TIMESTAMP column and referencing it in UPDATE and DELETE statements. This prevents data changes from occurring if a query reads dirty data. This requires more code to determine that an update or delete did not occur and then to take the appropriate action to recover. Comparing all original values of every column in a row on every UPDATE or DELETE may be performed instead of verifying the TIMESTAMP column has not changed.