Interview Questions You Should Know – SQL

Obviously you should know basic SELECT, INSERT, UPDATE, and DELETE query syntax as well as WHERE clause expression syntax. Review the slide decks if you don't.

What is the difference between an inner join and an outer join?

A joins are used to combine rows from two tables and create a result set that can contain columns from both. An **inner join** only returns records that have matching data in the join condition (typically one or more columns that are compared).

```
SELECT LastName, DepartmentName
FROM Employee
    INNER JOIN Department
        on Employee.DepartmentID = Department.DepartmentID
```

The above query would generate a list of all employee last names and their department. If an employee is not assigned to a department, that employee would not be returned.

In an **outer join** matching rows will be returned from both table, as well as any unmatched rows from one of the tables.

```
SELECT LastName, DepartmentName
FROM Employee
    LEFT JOIN Department
        on Employee.DepartmentID = Department.DepartmentID
```

The query now would select all employees regardless of whether they had a department. If an employee was not assigned to a department, the department name would be null.

What is the difference between Join and Union?

Where joins allow us to create result sets for matching records between two tables, unions take two similar data sets and combine them into a single set that contains all of the data from both sources. Say for example we had two tables, employees and contractors, both having LastName and Email columns. We could combine both tables as follows:

```
SELECT LastName, Email, 'Employee' as PersonType
FROM Employees
UNION
SELECT LastName, Email, 'Contractor' as PersonType
FROM Contractors
```

What is a self join?

A self join is the act of joining a table to itself. It is typically used in a hierarchical structure (ex: managers of each employee, categories and subcategories, etc.)

```
SELECT e1.LastName, e2.LastName as ManagerLastName
FROM Employees e1
    LEFT JOIN Employees e2
        ON e1.ManagerID = e2.EmployeeID
```

What is the difference between a WHERE clause and a HAVING clause?

A where clause can only be used on table columns while a having can only be used on columns aggregated with one of the aggregate functions (SUM, MAX, MIN, etc).

```
SELECT DepartmentName, avg(Salary) AS AverageSalary
FROM Department
    inner join Employee
        on Department.DepartmentID = Employee.EmployeeID
WHERE DepartmentName in ('Finance', 'Marketing', 'IT')
GROUP BY DepartmentName
HAVING avg(Salary) > 50000
```

What do we mean by Primary Key and Foreign Key?

When you specify a **primary key (PK)** constraint for a table, the Database Engine enforces data uniqueness by automatically creating a unique index for the primary key columns. This index also permits fast access to data when the primary key is used in queries. If a primary key constraint is defined on more than one column, values may be duplicated within one column, but each combination of values from all the columns in the primary key constraint definition must be unique. A primary key defaults to a clustered index and is frequently an identity column.

A **foreign key (FK)** is a column or combination of columns that is used to establish and enforce a link between the data in two tables to control the data that can be stored in the foreign key table. In a foreign key reference, a link is created between two tables when the column or columns that hold the primary key value for one table are referenced by the column or columns in another table. This column becomes a foreign key in the second table.

Foreign keys are not indexed by default, so it is very common to declare an index on foreign key columns commonly used in join conditions.

In our example above, the DepartmentID is a primary key on the Department table and a Foreign Key on the Employee table. If the foreign key column does not allow null values, then it means every employee must be assigned to a valid department and that a Department record cannot be deleted unless no employees are assigned to it. This is called **referential integrity** and is a key feature of relational databases.

What is a NULL value?

A value of NULL indicates that the value is unknown. A value of NULL is different from an empty or zero value. No two null values are equal. Comparisons between two null values, or between a NULL and any other value, return unknown because the value of each NULL is unknown. NULL column values used in aggregate or math expressions always return NULL. A NULL value cannot be placed in a primary key column.

Null values generally indicate data that is unknown, not applicable, or that the data will be added later. For example, a customer's middle initial may not be known at the time the customer places an order.

To test for NULL values in a query, we can use the **IS NULL** or **IS NOT NULL** expressions in the WHERE clause.

What is a SQL Injection Attack and how are they prevented?

SQL Injection is one of the top 10 web application vulnerabilities. It is a code injection technique in which malicious SQL statements are inserted into an entry field or URL parameter to dump database contents to the attacker. It is effective when user input is incorrectly filtered for string literal escape characters.

Imagine a variable that concatenates a value into a SQL statement like so:

```
statement = "SELECT * FROM users WHERE name ='" + userName + "';"
```

A malicious user could enter text into the username field to select all of the users by injecting 'or '1'='1', making the final statement something like this:

```
SELECT * FROM users WHERE name = '' OR '1'='1';
```

The common way to prevent SQL injection is to parameterize your queries via stored procedures or by declaring variables in text commands:

```
SqlCommand cmd = new SqlCommand("Select * from users where username =
@User and password = @Password", cn);
cmd.CommandType = CommandType.Text;
cmd.Parameters.AddWithValue("@User", UserName;
cmd.Parameters.AddWithValue("@Password", Password;
```

Stored procedures are generally preferred because by granting the application access to only stored procedures and not the tables themselves, it greatly limits the ability of an attacker to navigate through your database.

What is an index?

An index is a data structure stored on disk that is used to speed up retrieval of rows from a table or view. Indexes are made up of one or more columns in the table or view. There are many different types of indexes, but the most common are clustered and non-clustered.

A **clustered index** is typically the primary key of the table, but it does not have to be. It is the physical order of the “real” table data on the disk, so there can only be one clustered index per table. If a table has no clustered index, then rows are stored in an unordered structure called a **heap**. Heaps are very inefficient for lookups and typically require the entire table to be scanned to find queried information.

A **nonclustered index** are stored in a data structure separate from the data rows. Any number of columns can make up the key of a nonclustered index and it is used to quickly retrieve data rows for commonly searched fields that are not the primary key. These indexes are typically put on columns commonly used in joins (foreign keys) and columns that are frequently searched on (if your users look up customers by phone number, then it may make sense to index the PhoneNumber column).

Both indexes can have a **unique constraint** placed on them (this happens automatically for primary keys). A unique constraint prevents duplicate data from being placed in a column.

Indexes do not always improve performance. It is the job of a DBA to analyze query patterns for the organization and tune indexes and queries for the best performance in the most important scenarios. Because data changing queries (insert, update, delete) necessitate changes to indexes, having many unnecessary indexes on a table can slow down those types of queries. Indexes also take up disk space, so if you have 1 million rows in the table, your index will also have 1 million rows with at minimum two pieces of data- the indexed column and the primary key of the row the column is stored in.

What is the transaction log?

Every SQL Server database has a transaction log that records all transactions and the database modifications made by each transaction. The transaction log must be truncated on a regular basis to keep it from filling up.

The transaction log is a critical component of the database and, if there is a system failure, the transaction log might be required to bring your database back to a consistent state. The transaction log should never be deleted or moved unless you fully understand the ramifications of doing this.

A database in the simple recovery mode will automatically reuse transaction log space. This setting means the database can only be restored up to the last full backup.

A database in full recovery mode will keep growing the transaction log until the log is backed up. It is possible in this mode to restore the last backup, then restore each transaction log back up to near the point in time when the database failed. So if you back up your transaction log every 5 minutes, your exposure to data loss is 4 minutes and 59 seconds of data.

What are some common T-SQL Functions?

T-SQL has many useful built in functions available. Some of the most common ones are as follows:

- **GETDATE()** – Gets the current date/time on the server
- **DATEADD()** – Allows for adding hours, days, and years to an existing date
- **LEFT(), RIGHT()** – Select the leftmost or rightmost # of characters in a string
- **SUBSTRING()** – Return a portion of a string based on start position and length
- **REPLACE()** – Replace all instances of one string with another
- **SCOPE_IDENTITY()** – Gets the most recently created identity column value (useful for getting the value of an automatically created primary key)

There are also a lot of math functions built in including (but not limited to) ABS, CEILING, EXP, FLOOR, LOG, PI, POWER, RAND, ROUND, SIN, COS, TAN, SQRT.

What is a Subquery?

A subquery is a query that is nested inside another query (or subquery). Subqueries can be used **anywhere an expression is allowed**.

An example of this in a WHERE clause. Say we wanted to find all the customers in Northwind who are in the country with the fewest number of orders:

```
select Country, CompanyName, ContactName, ContactTitle, Phone
from Customers
where Country =
   (select top 1 country
        from Customers C
           inner join Orders O on C.CustomerId = O.CustomerID
    group by country
    order by count(*))
```

We can also use them in a FROM clause, like if we wanted the product information for our top 10 best-selling products:

```
SELECT p.ProductID, ProductName, TopTenSold.NumberSold
FROM Products p
    INNER JOIN
        (SELECT TOP 10 ProductID, SUM(Quantity) as NumberSold
         FROM [Order Details]
         GROUP BY ProductID
         ORDER BY NumberSold DESC) AS TopTenSold
    ON p.ProductID = TopTenSold.ProductID
```

What is a Stored Procedure and why do we use them?

A stored procedure is a database object that contains one or more SQL statements that is stored and invoked later. Stored procedures have several significant advantages, not limited to the following:

1. Maintainability- Because scripts are on the database, the database team can easily locate and modify them without having to change the source code. This also means that database updates can be pushed separately from the source code in some cases.
2. Security- By limiting direct access to tables and providing an interface to the underlying data structure all implementation and even the data itself is shielded. It is often easier to secure the stored procedures in the database than applying security in the application logic itself.
3. Set Processing- SQL is design to quickly and efficiently perform set based operations (insert/update/delete/select large amounts of rows). In large sets it is almost always faster than the coding equivalent (iterative looping).

The server/query is slow, what should I do?

This is a symptom that could be pointing to many problems. Some possible reasons are:

1. Slow network communication
2. Other queries being run concurrently on the server, particularly blocking ones or ones that use up a large amount of resources.
3. Lack of useful indexes
4. Not enough memory on the SQL Server
5. Write contention (the disk is doing too much work)

If a query is always slow it is likely a poorly written query, lack of indexes, or not enough resources on the server. If it is a sporadic problem it is likely network communication, write contention, or not enough resources due to other queries being run at the time.

A major part of a DBA's job is being aware of what is going on in the server so they can properly investigate the root cause.