নিচে একজন **Lead Software Engineer** পদের জন্য SQL সম্পর্কিত গুরুত্বপূর্ণ ইন্টারভিউ প্রশ্ন ও উত্তর বাংলায় দেওয়া হলো। এই প্রশ্নগুলো বাস্তব অভিজ্ঞতা, পারফরমেন্স অপটিমাইজেশন, এবং জটিল কুয়েরি ডিজাইন করার দক্ষতা যাচাইয়ের জন্য ব্যবহৃত হয়।

**🔹 ১. SQL ও T-SQL এর মধ্যে পার্থক্য কী?**

**উত্তর:**

**SQL (Structured Query Language)** হলো একটি স্ট্যান্ডার্ড ল্যাঙ্গুয়েজ ডেটাবেইজ ম্যানিপুলেশন ও রিট্রিভ করার জন্য।  
**T-SQL (Transact-SQL)** হলো Microsoft SQL Server-এর একটি এক্সটেনশন, যেখানে অতিরিক্ত ফিচার যেমন DECLARE, BEGIN...END, লুপিং (WHILE), এবং কন্ডিশনাল লজিক (IF...ELSE) আছে।

| **SQL** | **T-SQL** |
| --- | --- |
| স্ট্যান্ডার্ড ভাষা | Microsoft-এর এক্সটেনশন |
| CRUD অপারেশন | লজিকাল অপারেশনসহ সম্পূর্ণ প্রোগ্রামিং সমর্থন |

**🔹 ২. INNER JOIN এবং LEFT JOIN এর মধ্যে পার্থক্য কী?**

**উত্তর:**

* INNER JOIN: শুধুমাত্র যেসব রেকর্ড দুইটি টেবিলে মিলে যায় সেগুলো রিটার্ন করে।
* LEFT JOIN: বাম পাশের টেবিলের সব রেকর্ড রিটার্ন করে, আর ডান পাশের টেবিলের মেলানো রেকর্ড থাকলে দেখায়, না থাকলে NULL দেখায়।

**উদাহরণ:**

-- INNER JOIN

SELECT a.Name, b.OrderDate

FROM Customers a

INNER JOIN Orders b ON a.CustomerID = b.CustomerID;

-- LEFT JOIN

SELECT a.Name, b.OrderDate

FROM Customers a

LEFT JOIN Orders b ON a.CustomerID = b.CustomerID;

**🔹 ৩. Index কী? কত প্রকার এবং কিভাবে পারফরমেন্স উন্নয়ন করে?**

**উত্তর:**

Index হলো ডেটাবেজের একটি স্ট্রাকচার যা দ্রুত সার্চ ও রিট্রিভ করার জন্য ব্যবহার হয়।

**প্রকারভেদ:**

* **Clustered Index**: ডেটা টেবিলে সরাসরি সাজানো থাকে।
* **Non-Clustered Index**: আলাদা স্ট্রাকচারে থাকে যা মূল ডেটার রেফারেন্স রাখে।

**উদাহরণ:**

-- Clustered Index তৈরি

CREATE CLUSTERED INDEX idx\_EmployeeID ON Employees(EmployeeID);

-- Non-clustered Index তৈরি

CREATE NONCLUSTERED INDEX idx\_EmpName ON Employees(EmployeeName);

**পারফরমেন্স উন্নয়ন করে কারণ:** সার্চ করার সময় পুরো টেবিল স্ক্যান না করে ইনডেক্স অনুসারে ডেটা খুঁজে পাওয়া যায়।

**🔹 ৪. ROW\_NUMBER() vs RANK() vs DENSE\_RANK() – ব্যাখ্যা করুন।**

**উত্তর:**

ROW\_NUMBER(): প্রতিটি রোকে ইউনিক সিকোয়েন্স নম্বর দেয়।

RANK(): সমান ভ্যালু পেলে একই র‍্যাংক দেয়, কিন্তু পরের র‍্যাংক স্কিপ করে।

DENSE\_RANK(): সমান ভ্যালু পেলে একই র‍্যাংক দেয়, কিন্তু কোনো র‍্যাংক স্কিপ করে না।

**উদাহরণ:**

SELECT Name, Salary,

ROW\_NUMBER() OVER(ORDER BY Salary DESC) AS RowNum,

RANK() OVER(ORDER BY Salary DESC) AS RankNum,

DENSE\_RANK() OVER(ORDER BY Salary DESC) AS DenseRank

FROM Employees;

**🔹 ৫. SQL Injection কী? এটি প্রতিরোধ করার উপায় কী?**

**উত্তর:**

**SQL Injection** হলো একটি নিরাপত্তা দুর্বলতা যেখানে অ্যাটাকার ইউজার ইনপুটের মাধ্যমে ম্যালিশিয়াস SQL চালাতে পারে।

**প্রতিরোধের উপায়:**

* Parameterized Query ব্যবহার করা
* Stored Procedure ব্যবহার করা
* ইনপুট ভ্যালিডেশন
* ORM (Entity Framework)-এর মাধ্যমে Access

**উদাহরণ (Safe):**

// C# with parameterized query

cmd.CommandText = "SELECT \* FROM Users WHERE Username = @username";

cmd.Parameters.AddWithValue("@username", userInput);

**🔹 ৬. CTE (Common Table Expression) কী এবং কখন ব্যবহার করা হয়?**

**উত্তর:**

CTE হলো একটি টেম্পোরারি নামকৃত রেজাল্ট সেট, যা SQL কুয়েরির মধ্যে ব্যবহার হয়।

**উদাহরণ:**

WITH EmployeeCTE AS (

SELECT EmployeeID, ManagerID, Name FROM Employees

)

SELECT \* FROM EmployeeCTE WHERE ManagerID IS NULL;

**ব্যবহার:**

* রিকার্সিভ কুয়েরিতে
* কমপ্লেক্স সাবকুয়েরির রিডেবিলিটি বাড়াতে
* পার্শিয়াল রেজাল্ট রিইউজ করতে

**🔹 ৭. Execution Plan কীভাবে বিশ্লেষণ করবেন?**

**উত্তর:**

Execution Plan দেখায় SQL Server কীভাবে কুয়েরি এক্সিকিউট করে। আপনি এটি SQL Server Management Studio (SSMS) থেকে "Actual Execution Plan" চালিয়ে দেখতে পারেন।

* **Table Scan** → স্লো, ইনডেক্স নেই
* **Index Seek** → ফাস্ট, ইনডেক্স ইউজ হচ্ছে
* **Key Lookup** → ইনডেক্স আছে কিন্তু অতিরিক্ত রিড হচ্ছে

**🔹 ৮. Deadlock কী এবং কীভাবে প্রতিরোধ করবেন?**

**উত্তর:**

**Deadlock** তখন ঘটে যখন দুই বা ততোধিক ট্রানজেকশন এমনভাবে একে অপরের লক চায়, যাতে কেউই এগোতে পারে না।

**প্রতিরোধ:**

* সবসময় একই অর্ডারে টেবিল অ্যাক্সেস করুন
* ট্রানজেকশন ছোট রাখুন
* নো-লক হিন্ট ব্যবহার করা যায় কিছু ক্ষেত্রে: WITH (NOLOCK)
* রিট্রাই লজিক যোগ করুন

**🔹 ৯. Stored Procedure vs Function পার্থক্য?**

| **বৈশিষ্ট্য** | **Stored Procedure** | **Function** |
| --- | --- | --- |
| রিটার্ন টাইপ | 0 বা একাধিক | একটাই |
| ব্যবহার | DML, Logic | SELECT-এর মধ্যে |
| Exception Handling | সমর্থন করে | সীমিত |
| ইনপুট/আউটপুট প্যারাম | আছে | শুধুমাত্র ইনপুট |

**🔹 ১০. Partitioning কী এবং Performance-এ কীভাবে সহায়ক?**

**উত্তর:**

Partitioning হচ্ছে বড় টেবিলকে লজিকালি ভাগ করা।  
**উদাহরণ:** Sales ডেটা প্রতি বছর ভাগ করা।

**পারফরমেন্স বেনিফিট:**

* Query শুধু প্রাসঙ্গিক পার্টিশনে চলে
* ইনডেক্স স্ক্যান সীমিত হয়
* মেইনটেনেন্স সহজ হয়

নিচে প্রতিটি SQL ইন্টারভিউ প্রশ্ন ও উত্তর আরও বিস্তারিতভাবে ব্যাখ্যা করা হলো যেন একজন **Lead Software Engineer** পদপ্রার্থী গভীরভাবে বুঝতে পারেন এবং বাস্তবে প্রয়োগ করতে পারেন।

**🔹 ১. SQL vs T-SQL এর মধ্যে পার্থক্য – বিশদ ব্যাখ্যা**

**SQL (Structured Query Language):**  
এটি একটি স্ট্যান্ডার্ড ভাষা যা ANSI এবং ISO দ্বারা সংজ্ঞায়িত। প্রায় সব রিলেশনাল ডেটাবেজেই SQL ব্যবহার করা যায় (যেমন: Oracle, MySQL, PostgreSQL, SQL Server)।

**T-SQL (Transact SQL):**  
Microsoft SQL Server-এর নিজস্ব এক্সটেনশন। SQL এর উপর ভিত্তি করে তৈরি কিন্তু এতে অতিরিক্ত কিছু ফিচার যোগ হয়েছে যেমন:

* Exception handling (TRY...CATCH)
* Variables (DECLARE, SET)
* Control-of-flow (IF...ELSE, WHILE)

**উদাহরণ:**

✅ SQL (স্ট্যান্ডার্ড):

SELECT FirstName, LastName FROM Employees WHERE Department = 'HR';

✅ T-SQL (উন্নত):

DECLARE @DeptName VARCHAR(50) = 'HR';

SELECT FirstName, LastName FROM Employees WHERE Department = @DeptName;

**🔹 ২. INNER JOIN vs LEFT JOIN – বিস্তারিত**

**INNER JOIN**  
যখন দুই টেবিলে matching রেকর্ড থাকে, তখনই সেই রেকর্ডগুলো রিটার্ন হয়।

**LEFT JOIN**  
বাম টেবিলের সব রেকর্ড রিটার্ন হয়, আর ডান টেবিলে যেটুকু মিল পাওয়া যায়, তা দেখায়। না পেলে NULL।

**ডায়াগ্রাম কনসেপ্ট:**

Customers (LEFT) Orders (RIGHT)

ID Name ID CustomerID

1 Alice 1 1

2 Bob 2 1

3 Charlie - -

✅ INNER JOIN Result:

| **Name** | **OrderID** |
| --- | --- |
| Alice | 1 |
| Alice | 2 |

✅ LEFT JOIN Result:

| **Name** | **OrderID** |
| --- | --- |
| Alice | 1 |
| Alice | 2 |
| Bob | NULL |
| Charlie | NULL |

**🔹 ৩. Index কী এবং এটি পারফরম্যান্সে কীভাবে সহায়তা করে – বিস্তারিত**

**➤ Index হচ্ছে ডেটার ম্যাপের মত**

যেমন আপনি একটি বইতে ইনডেক্স পেইজে দেখে দ্রুত বিষয়বস্তুতে পৌঁছাতে পারেন, ইনডেক্সও ডেটাবেজে সার্চ দ্রুত করে।

**➤ Index এর প্রকারভেদ:**

**✅ Clustered Index:**

* একটি মাত্র থাকতে পারে
* ডেটা ফিজিক্যালি সাজানো হয়
* Primary Key সাধারণত এটিই হয়

**✅ Non-Clustered Index:**

* অনেকগুলো থাকতে পারে
* আলাদা ভাবে ইনডেক্স থাকে, রেফারেন্স হিসাবে মূল ডেটার Row ID রেখে দেয়

**➤ পারফরম্যান্স উন্নয়ন:**

* WHERE, JOIN, ORDER BY তে ইনডেক্স থাকলে Table Scan-এর দরকার পড়ে না
* CPU ও I/O usage কম হয়

**উদাহরণ:**

-- Without Index: Table Scan

SELECT \* FROM Employees WHERE EmployeeName = 'John';

-- With Index: Index Seek

CREATE NONCLUSTERED INDEX idx\_name ON Employees(EmployeeName);

**🔹 ৪. ROW\_NUMBER vs RANK vs DENSE\_RANK – বিশদ ব্যাখ্যা**

| **ফাংশন** | **একই মানে র‍্যাংক** | **পরের মান স্কিপ করে** | **ব্যবহার** |
| --- | --- | --- | --- |
| ROW\_NUMBER | না | না | ইউনিক আইডেন্টিফায়ার |
| RANK | হ্যাঁ | হ্যাঁ | বিজয়ী র‍্যাংকিং |
| DENSE\_RANK | হ্যাঁ | না | টাই হ্যান্ডলিং |

**উদাহরণ ডেটা:**

| **Name** | **Salary** |
| --- | --- |
| A | 1000 |
| B | 900 |
| C | 900 |
| D | 800 |

**ROW\_NUMBER:**

| **Name** | **RowNum** |
| --- | --- |
| A | 1 |
| B | 2 |
| C | 3 |
| D | 4 |

**RANK:**

| **Name** | **RankNum** |
| --- | --- |
| A | 1 |
| B | 2 |
| C | 2 |
| D | 4 |

**DENSE\_RANK:**

| **Name** | **DenseRank** |
| --- | --- |
| A | 1 |
| B | 2 |
| C | 2 |
| D | 3 |

**🔹 ৫. SQL Injection – বিশদভাবে**

**কি ঘটে?**  
একজন অ্যাটাকার যদি ইউজার ইনপুটের মাধ্যমে নিজের SQL Code ইনজেক্ট করতে পারে তাহলে ডেটা মুছে ফেলতে বা এক্সেস করতে পারে।

**Example (Vulnerable):**

-- যদি ইউজার ইনপুট দেয়: ' OR 1=1 --

SELECT \* FROM Users WHERE Username = '' OR 1=1 --' AND Password = '...';

**Result:** সব ইউজার দেখায়

✅ **সমাধান:**

// C#

SqlCommand cmd = new SqlCommand("SELECT \* FROM Users WHERE Username = @username");

cmd.Parameters.AddWithValue("@username", userInput);

**🔹 ৬. CTE – বিস্তারিত বিশ্লেষণ**

**CTE (Common Table Expression)** হলো টেম্পোরারি রেজাল্ট যা নাম দিয়ে SQL ব্লকে ব্যবহার করা যায়।

**উপকারিতা:**

* কোড রিডেবল হয়
* রিকার্সিভ কুয়েরি সহজ হয়
* সাবকুয়েরি এড়ানো যায়

**উদাহরণ:**

WITH TopEmployees AS (

SELECT Name, Salary FROM Employees WHERE Salary > 50000

)

SELECT \* FROM TopEmployees WHERE Name LIKE 'A%';

**🔹 ৭. Execution Plan ব্যাখ্যা**

**Execution Plan** হলো SQL Server-এর পরিকল্পনা কিভাবে একটি কুয়েরি এক্সিকিউট করা হবে।

**প্রধান উপাদান:**

* **Table Scan**: পুরো টেবিল পড়ে → স্লো
* **Index Seek**: ইনডেক্স দিয়ে খুঁজে → ফাস্ট
* **Key Lookup**: ইনডেক্স পায়, কিন্তু আরও কলাম পড়তে হয়

✅ **SSMS এ কিভাবে দেখবেন?**

* “Actual Execution Plan” ট্যাব অন করুন

**🔹 ৮. Deadlock – বিস্তারিত**

**কেন ঘটে?**  
দুইটি বা একাধিক ট্রানজেকশন একে অপরের রিসোর্স লক করে রাখে, কেউই এক্সিকিউট করতে পারে না।

**উদাহরণ:**

* A → Table X লক → চায় Y
* B → Table Y লক → চায় X

**প্রতিকার:**

* সব কোডে টেবিল একসাথে একি অর্ডারে অ্যাক্সেস
* SET DEADLOCK\_PRIORITY LOW দিয়ে একটিকে ছাড় দেওয়া
* ট্রানজেকশন ছোট রাখা
* Retry Mechanism

**🔹 ৯. Stored Procedure vs Function – তুলনা**

| **বৈশিষ্ট্য** | **Stored Procedure** | **Function** |
| --- | --- | --- |
| রিটার্ন | 0 বা 1 বা অনেক | 1 |
| ইনপুট/আউটপুট | ইন, আউট প্যারাম | শুধু ইনপুট |
| DML (INSERT/UPDATE) | করা যায় | পারমিটেড না |
| ব্যবহার | অ্যাপ্লিকেশন লজিকে | শুধু SELECT-এ |

**🔹 ১০. Partitioning – বিস্তারিত**

**Partitioning** অর্থ এক টেবিলকে লজিক্যাল ভাগে ভাগ করা (উদাহরণস্বরূপ, SalesData কে বছরে ভাগ করা)।

**উদাহরণ:**

CREATE PARTITION FUNCTION SalesRange (int)

AS RANGE LEFT FOR VALUES (2019, 2020, 2021);

**উপকারিতা:**

* পার্টিশন প্রুনিং: কুয়েরি নির্দিষ্ট পার্টিশনে রান করে
* ব্যাকআপ, ডিলিট, আর্কাইভ দ্রুত হয়

Stored Procedure Optimization (স্টোর্ড প্রোসিজার অপ্টিমাইজেশন) একটি অত্যন্ত গুরুত্বপূর্ণ বিষয়, বিশেষ করে যখন আপনি একজন **Lead Software Engineer** হিসেবে কাজ করছেন। বড় সিস্টেমে স্টোর্ড প্রোসিজারের পারফরম্যান্স খারাপ হলে পুরো অ্যাপ্লিকেশনের রেসপন্স টাইম কমে যেতে পারে। নিচে এর বিস্তারিত বাংলা ব্যাখ্যা, কৌশল ও উদাহরণ দেওয়া হলো।

**🔧 Stored Procedure Optimization কেন প্রয়োজন?**

* ✅ Query Execution Time কমানো
* ✅ Server Resource (CPU, Memory, IO) ব্যবহার কমানো
* ✅ ডেটাবেজ Scalability বাড়ানো
* ✅ Deadlock/Blocking এড়ানো
* ✅ Better Maintainability

**🎯 Optimization কৌশলসমূহ (Step by Step)**

**✅ ১. Avoid SELECT \* — প্রয়োজন অনুযায়ী কলাম নির্বাচন করুন**

**কারণ:**

* অপ্রয়োজনীয় ডেটা ট্রান্সফার হয়
* ইনডেক্স ইউজ নাও হতে পারে

**❌ খারাপ:**

SELECT \* FROM Employees;

**✅ ভালো:**

SELECT EmployeeID, FirstName, LastName FROM Employees;

**✅ ২. উপযুক্ত ইনডেক্স ব্যবহার করুন (Index Seek vs Scan)**

**Execution Plan** দেখে বুঝতে হবে কুয়েরিতে **Index Seek** হচ্ছে নাকি **Scan**।

**বুঝবেন:**

* Table Scan → খারাপ পারফরমেন্স
* Index Seek → ভালো

**পরামর্শ:**

* WHERE, JOIN, ORDER BY, GROUP BY-তে ব্যবহৃত কলামের উপর Index দিন

**✅ ৩. SARGable Query লিখুন (Search ARGument Able)**

**❌ খারাপ (Index Use হবে না):**

WHERE YEAR(OrderDate) = 2023

**✅ ভালো (Index Use হবে):**

WHERE OrderDate >= '2023-01-01' AND OrderDate < '2024-01-01'

**✅ ৪. Temp Table vs Table Variable – সঠিক ব্যবহার করুন**

| **Feature** | **Temp Table (#)** | **Table Variable (@)** |
| --- | --- | --- |
| ইনডেক্স সাপোর্ট | হ্যাঁ | সীমিত |
| Estimated Rows | সঠিক | ভুল (একই Stat = 1) |
| বড় ডেটার জন্য | ভালো | খারাপ (Memory Pressure) |

**উদাহরণ:**

-- Better for large data

SELECT \* INTO #TempEmployees FROM Employees WHERE Status = 'Active';

-- Good for small rows

DECLARE @Emp TABLE (ID INT, Name NVARCHAR(50));

**✅ ৫. Avoid Cursors – Set-Based Query ব্যবহার করুন**

**Cursor খারাপ কারণ:**

* Row-by-row প্রসেস হয়
* পারফরমেন্স খারাপ
* CPU বেশি ব্যবহার করে

**❌ খারাপ:**

DECLARE EmpCursor CURSOR FOR SELECT Name FROM Employees;

**✅ ভালো:**

UPDATE Employees SET Bonus = Bonus + 1000 WHERE Department = 'Sales';

**✅ ৬. Execution Plan ও SQL Profiler ব্যবহার করে Analyze করুন**

* **Actual Execution Plan** → কীভাবে Query চলেছে
* **Estimated Plan** → কীভাবে Query চলতে পারে
* **SQL Server Profiler** → কোন SP কত বার চলে, কত সময় লাগে

**✅ ৭. Recompile ফোর্স করা দরকার হলে বুঝে করুন**

CREATE PROCEDURE MyProc

WITH RECOMPILE

AS

BEGIN

SELECT ... FROM ...

END

**Recompile কী করে?**

* প্রতিবার নতুন Execution Plan বানায়
* ভালো যখন ইনপুট ভ্যালুর ভিন্নতায় পারফরমেন্স বদলায়

⚠️ **বেশি ব্যবহার করলে CPU লোড বাড়বে**

**✅ ৮. Parameter Sniffing সমস্যা বোঝা এবং ঠিক করা**

**Parameter Sniffing:**  
যখন SQL Server প্রথমবার Execution Plan বানায়, তখনকার ইনপুটের উপর ভিত্তি করে Plan তৈরি করে। পরের বার ভিন্ন ইনপুটেও সেই পুরোনো Plan ব্যবহার করে — ফলে পারফরমেন্স খারাপ হয়।

**সমাধান:**

* OPTION (RECOMPILE)
* OPTIMIZE FOR UNKNOWN
* Local Variable এ Parameter রেখে কুয়েরি চালানো

**উদাহরণ:**

-- ভালো সমাধান

DECLARE @ID INT = @InputID;

SELECT \* FROM Orders WHERE CustomerID = @ID;

**✅ ৯. Result Set ছোট রাখা ও Pagination ব্যবহার**

**❌ খারাপ:**

SELECT \* FROM Orders; -- লাখ লাখ রেকর্ড!

**✅ ভালো (Pagination):**

SELECT \* FROM Orders

ORDER BY OrderDate

OFFSET 0 ROWS FETCH NEXT 50 ROWS ONLY;

**✅ ১০. Multiple Result Set এড়ানো এবং Output Parameters ব্যবহার করুন**

* অ্যাপ্লিকেশন লজিক সহজ হয়
* Network Load কমে

**🛠️ Optimization প্র্যাকটিস চেকলিস্ট:**

| **চেকপয়েন্ট** | **হয়েছে? ✅/❌** |
| --- | --- |
| SELECT \* বাদ দেওয়া হয়েছে? |  |
| ইনডেক্স তৈরি করা হয়েছে? |  |
| Execution Plan বিশ্লেষণ করা হয়েছে? |  |
| Cursor বাদ দিয়ে Set-based Query হয়েছে? |  |
| Parameter Sniffing ঠিক করা হয়েছে? |  |
| Temp Table vs Table Variable বোঝা গেছে? |  |

**📌 উপসংহার:**

একজন Lead Software Engineer হিসেবে, আপনাকে কেবল Stored Procedure লিখতে জানলেই হবে না — সেটি কীভাবে **Perform Efficiently**, **Scale** ও **Maintain** করা যায় সেটিও জানতে হবে।