SQL Server Statistics - Fundamentals

The Real Brain Behind the Query Optimizer

Indexes help SQL Server **locate** data,
but **Statistics tell SQL Server *how* to look for it**.
Without accurate statistics, even the best index becomes useless.

🔷 What Are Statistics?

Statistics are metadata objects that describe **data distribution** inside a table or index.
The **Query Optimizer** relies on statistics to estimate:

- 🔢 Number of rows returned
- 🎯 Data selectivity (how rare/common values are)
- 💰 Cost of different execution plans
- ⚖️ Join strategies (Nested Loop vs Hash vs Merge)
- 🧠 Memory grants (critical in SQL Server 2019+)

📌 **No statistics = guesses.**
Bad statistics = wrong guesses.

🔷 Key Components of Statistics

Every statistics object contains:

1) Histogram

Shows **distribution of values** (up to 200 steps).

- Used for **range predicates**
- Drives **seek vs scan** decisions

2) Density

Represents **data uniqueness**.

- Used when histogram cannot help
- Impacts joins and equality predicates

3) Header

Metadata including:

- Last updated date
- Number of rows sampled
- Modification counter

🔷 Types of Statistics

1) Auto-Created Statistics

- Created automatically when AUTO_CREATE_STATISTICS = ON
- Created on **non-indexed columns**
- Name pattern: _WA_Sys_*

2) User-Created Statistics

- Manually created by DBAs
- Used for **filtered queries** or complex predicates
- Critical for reporting databases

3) Index Statistics

- Created automatically when an index is created
- One-to-one relationship with the index

🔷 **Real-Time Scenario (AdventureWorks2019)**

🎯 **Scenario: Slow Query on Sales.SalesOrderDetail**

```
SELECT *
FROM Sales.SalesOrderDetail
WHERE OrderQty = 1;
```

Problem:

- Column OrderQty is **not indexed**
- SQL Server initially creates **auto statistics**
- Data distribution later changes (bulk insert)

👉 **Result:**

- Estimated rows = 10
- Actual rows = 120,000
- Execution plan switches to **Table Scan**
- CPU spikes 🔥

🔷 **Check Existing Statistics**

```
SELECT
   s.name AS StatsName,
   STATS_DATE(s.object_id, s.stats_id) AS LastUpdated
FROM sys.stats s
WHERE s.object_id = OBJECT_ID('Sales.SalesOrderDetail');
```

📌 If stats are old → optimizer decisions are wrong.

🔷 **View Histogram (Live Insight)**


```
DBCC SHOW_STATISTICS
(   'Sales.SalesOrderDetail',
   '_WA_Sys_00000002_44CA3770'
);
```

This shows:

- Range values
- Equal rows
- Average rows per step

💡 **This is exactly what the optimizer reads.**

🔷 **Fix with User-Created Statistics**

```
CREATE STATISTICS Stats_OrderQty
ON Sales.SalesOrderDetail(OrderQty);
```

✓ Query plan improves

✓ Accurate row estimation

✓ CPU usage drops

🔷 **Why Statistics Matter (Production Reality)**

Bad or Outdated Statistics Cause:

- ❌ Table scans instead of seeks
- ❌ Wrong join types
- ❌ Excessive memory grants
- ❌ Parameter sniffing disasters
- ❌ Query timeouts in reports

📉 **One stale statistic can bring down a reporting server.**

🔷 **When Should You Update Statistics?**

✅ **Best Times:**

- After **large data changes** (>20–30%)
- After **bulk inserts / ETL loads**
- When **query performance suddenly degrades**
- Before major reporting jobs

🔁 **Index Operations Impact**

Operation	Statistics Updated
Index Rebuild	✅ YES (FULLSCAN)
Index Reorganize	❌ NO
Bulk Insert	❌ NO (unless auto-update triggers)

🔷 **Manual Update (Best Practice)**

```
UPDATE STATISTICS Sales.SalesOrderDetail
WITH FULLSCAN;
Or entire database:
EXEC sp_updatestats;
```

⚠️ sp_updatestats uses **sampling**, not FULLSCAN.

🔷 **SQL Server 2019 → 2025 Enhancements**

🚀 **Modern Optimizer Improvements:**

- **Adaptive Query Processing**
- **Batch Mode on Rowstore**
- **Improved Cardinality Estimator**
- **Automatic Plan Correction**

📌 **But ALL of these still depend on statistics.**

No stats → no intelligence.

🔷 **DBA Pro Tip** 💡

Index Rebuild updates statistics automatically.

Index Reorganize does NOT.

👉 Always follow reorganize with:

```
UPDATE STATISTICS <table>;
```

🔚 **Final Thought**

🏗️ **Indexes are the roads**

🗺️ **Statistics are the maps**

You can build the widest highways,

but **without accurate maps, SQL Server still gets lost**.

Statistics end-to-end, exactly how **real DBAs are evaluated** — with **questions, labs, deep dives, and a one-page cheat sheet**.

🔥 **SQL Server Statistics – Interview Q&A (Real DBA Level)**

1) What are statistics in SQL Server?

Answer:

Statistics are metadata objects that describe **data distribution** in columns or indexes.

The Query Optimizer uses them to estimate **row counts, selectivity, and execution cost**.

2) How are statistics different from indexes?

Answer:

Indexes help **locate data**.

Statistics help SQL Server **decide which index or access path to use**.

👉 Index = structure

👉 Statistics = intelligence

3) What happens if statistics are outdated?

Answer:

- Wrong cardinality estimates
- Table/Index scans instead of seeks
- Bad join choices
- Excessive memory grants
- CPU spikes & slow queries

4) What is a histogram?

Answer:

A histogram stores **value distribution** (up to 200 steps) and is used for **range predicates** like >, <, BETWEEN.

5) What is density?

Answer:

Density represents **data uniqueness** and is used when:

- Histogram can't be used
- Equality predicates or joins occur

6) Difference between AUTO and USER statistics?

Answer:

- **Auto stats**: Created automatically on non-indexed columns
- **User stats**: Created manually for better control and optimization

7) Does index rebuild update statistics?

Answer:

✅ **YES – FULLSCAN stats**

❌ Index reorganize does NOT update stats

8) How does SQL Server decide to auto-update stats?

Answer:

Based on **row modification threshold**:

- Small tables: ~20%
- Large tables: dynamic threshold (SQL 2016+)

9) What is ASC (Async Stats Update)?

Answer:

Query runs with **old stats**, update happens **in background**.

Prevents blocking but may return suboptimal plans temporarily.

🔟 **How do statistics affect joins?**

Answer:

They decide:

- Nested Loop vs Hash vs Merge Join
- Join order
- Memory grant size

🧪 **Live Lab: Broken Statistics (AdventureWorks2019)**

🎯 **Goal: See how bad stats break performance**

Step 1) Create a copy table

```
SELECT *
INTO Sales.SalesOrderDetail_BrokenStats
FROM Sales.SalesOrderDetail;
```

Step 2) Create skewed data (real-world scenario)

```
INSERT INTO Sales.SalesOrderDetail_BrokenStats
SELECT *
FROM Sales.SalesOrderDetail
WHERE OrderQty = 1;
GO 50
```


👉 Massive data skew introduced

Step 3) Run query (bad plan expected)

```
SELECT *
FROM Sales.SalesOrderDetail_BrokenStats
WHERE OrderQty = 1;
```

📉 Estimated rows ≠ Actual rows

📉 Table scan

📉 High CPU

Step 4) Check statistics date

```
SELECT
   name,
   STATS_DATE(object_id, stats_id) AS LastUpdated
FROM sys.stats
WHERE object_id = OBJECT_ID('Sales.SalesOrderDetail_BrokenStats');
```

Step 5) Fix stats

```
UPDATE STATISTICS Sales.SalesOrderDetail_BrokenStats
WITH FULLSCAN;
```

✅ Accurate estimates

✅ Better plan

✅ CPU drops

📊 Parameter Sniffing + Statistics (Deep Dive)

🎯 What is Parameter Sniffing?

SQL Server **sniffs first parameter value** and builds a plan based on stats for that value.

Real Example

```
CREATE PROCEDURE dbo.usp_GetOrders
  @OrderQty INT
AS
SELECT *
FROM Sales.SalesOrderDetail
WHERE OrderQty = @OrderQty;
```

First Execution:

```
EXEC dbo.usp_GetOrders 1;
```

- Returns many rows
- Optimizer chooses **scan**

Next Execution:

```
EXEC dbo.usp_GetOrders 50;
```

- Returns few rows
- But still uses scan ❌

👉 **Stats + first parameter caused bad plan**

Fix Options (Interview Gold)

Option 1) Update stats (first step)

```
UPDATE STATISTICS Sales.SalesOrderDetail;
```


Option 2) OPTIMIZE FOR

```
OPTION (OPTIMIZE FOR (@OrderQty = 10));
```

Option 3) RECOMPILE

```
OPTION (RECOMPILE);
```

Option 4) Filtered statistics (Best for skew)

```
CREATE STATISTICS Stats_OrderQty_One
ON Sales.SalesOrderDetail(OrderQty)
WHERE OrderQty = 1;
```

🔥 **Principal DBA level answer**

🧠 ONE-PAGE DBA CHEAT SHEET (Save This)

Statistics Basics

- AUTO_CREATE_STATISTICS → ON
- AUTO_UPDATE_STATISTICS → ON
- Histogram = ranges
- Density = uniqueness

When Plans Go Bad

✓ Check stats date

✓ Compare estimated vs actual rows

✓ Check data skew

✓ Look for parameter sniffing

Index vs Stats

Action	Stats Updated
Index Rebuild	✅ FULLSCAN
Index Reorganize	❌ NO
Bulk Insert	❌ Usually NO
sp_updatestats	✅ Sampling

Golden Commands

```
DBCC SHOW_STATISTICS ('table', 'statname');
UPDATE STATISTICS table WITH FULLSCAN;
EXEC sp_updatestats;
```

Interview One-Liners 💥

- “Indexes find data, statistics choose the plan”
- “Bad stats cause bad joins, not bad queries”
- “Parameter sniffing is a statistics problem first”

🎤 30-Minute Live Mock Interview

Topic: SQL Server Statistics (2019 → 2025)

⏱️ Minutes 0–5: Core Fundamentals

Q1) What are statistics in SQL Server?

A:

Statistics are metadata objects that describe data distribution in columns or indexes.
The Query Optimizer uses them to estimate row counts, selectivity, and plan cost.

Q2) Why are statistics more important than indexes?

A:

Indexes help locate data, but statistics decide **which index and access method** to use.
Without accurate statistics, SQL Server can choose a bad plan even with perfect indexes.

Q3) What happens when statistics are outdated?

A:

Row estimation becomes incorrect, leading to table scans, wrong join types, excessive memory grants, and high CPU usage.

Q4) What is inside a statistics object?

A:

- Histogram – value distribution
- Density – data uniqueness
- Header – last updated date and row counts

Q5) How many steps can a histogram have?

A:

Up to **200 steps**, regardless of table size.

⏱️ Minutes 5–10: Statistics Types & Behavior

Q6) Types of statistics in SQL Server?

A:

- Index statistics
- Auto-created statistics
- User-created statistics

Q7) When does SQL Server auto-create statistics?

A:

When a query references a non-indexed column and AUTO_CREATE_STATISTICS is ON.

Q8) Does index rebuild update statistics?

A:

Yes. Index rebuild updates statistics with **FULLSCAN**.

Q9) Does index reorganize update statistics?

A:

No. Reorganize only defragments pages. Statistics remain unchanged.

Q🔟 What is asynchronous statistics update?

A:

The query runs with old stats while the stats update happens in the background to avoid blocking.

⏱️ **Minutes 10–15: Real-World DBA Scenarios**

Q11) A query suddenly becomes slow after ETL load. First thing you check?

A:

Statistics freshness and estimated vs actual row mismatch in the execution plan.

Q12) How do you check when stats were last updated?

A:

Using STATS_DATE() or querying sys.stats.

Q13) How do bad statistics affect joins?

A:

They cause SQL Server to choose inefficient joins like Nested Loops instead of Hash or Merge joins.

Q14) Why do large tables suffer more from stale statistics?

A:

Because auto-update thresholds are higher, so stats update less frequently even after large data changes.

Q15) Is sp_updatestats enough for production?

A:

Not always. It uses sampling. Critical tables often require FULLSCAN updates.

⏱️ **Minutes 15–20: Parameter Sniffing & Advanced Topics**

Q16) What is parameter sniffing?

A:

SQL Server creates an execution plan based on the first parameter value passed, using statistics for that value.

Q17) How are statistics involved in parameter sniffing?

A:

The optimizer uses histogram data for the sniffed value, which may not represent other parameter values.

Q18) How do you identify parameter sniffing?

A:

Same query plan reused with drastically different actual row counts and performance.

Q19) Name three ways to fix parameter sniffing.

A:

- Update statistics
- OPTION (RECOMPILE)
- OPTIMIZE FOR / Filtered statistics

Q20) Best fix for skewed data?

A:

Filtered statistics or filtered indexes targeting the skewed values.

⏱ **Minutes 20–25: Execution Plans & Memory**

Q21) How do statistics affect memory grants?

A:

Bad row estimates cause over- or under-allocation of memory, leading to spills or wasted memory.

Q22) What is a memory spill?

A:

When SQL Server underestimates memory and spills data to tempdb, slowing query execution.

Q23) What plan warning often indicates bad stats?

A:

- Spill to tempdb
- Excessive memory grant
- Cardinality estimate warnings

Q24) Does Adaptive Query Processing remove the need for stats?

A:

No. AQO improves plan behavior but still depends on accurate statistics.

Q25) How does SQL Server 2019+ improve stats handling?

A:

Dynamic thresholds, batch mode on rowstore, adaptive joins, and better memory feedback—but stats remain critical.

⏱ **Minutes 25–30: Principal DBA Level**

Q26) How do you design a stats maintenance strategy?

A:

- FULLSCAN on critical tables
- Scheduled updates after ETL
- Sampling for large non-critical tables

Q27) What's worse: missing index or bad statistics?

A:

Bad statistics. A missing index can be identified, but bad stats silently cause wrong plans.

Q28) Can statistics exist without indexes?

A:

Yes. Auto-created and user-created statistics can exist independently of indexes.

Q29) How do you explain statistics to non-DBAs?

A:

Indexes are roads; statistics are maps. Without maps, SQL Server takes the wrong route.

Q30) One final DBA rule about statistics?

A:

"If performance suddenly drops, always suspect statistics before rewriting queries."

🧪 LIVE TROUBLESHOOTING

🧠 Whiteboard Scenario 1: "Query Suddenly Slowed Down"

🔴 Problem Statement

A query that normally runs in **2 seconds** is now taking **2 minutes** after last night's data load.

Interviewer:
Walk me through your troubleshooting steps.

✅ Ideal Whiteboard Answer (Step-by-Step)

1) **Confirm the issue**
   - Capture execution plan (actual)
   - Compare estimated vs actual rows
2) **Check statistics**
   - Look at STATS_DATE()
   - Identify skewed columns
3) **Check plan reuse**
   - Was the plan compiled before or after ETL?
4) **Look for parameter sniffing**
   - Same plan, different row counts?
5) **Validate indexes**
   - Ensure correct index still exists
6) **Check memory grants**
   - Look for spills or excessive grants
7) **Fix**
   - Update statistics (FULLSCAN if critical)
   - Recompile or adjust procedure

🧠 Key Interview Line:

"I never tune queries before validating statistics."

🧠 Whiteboard Scenario 2: "Index Exists But Scan Is Used"

🔴 Problem Statement

The correct index exists, but SQL Server is still doing a table scan.

Ideal Answer

Possible causes:

- Outdated statistics
- Low selectivity (high-density column)
- Parameter sniffing
- Implicit conversions
- Incorrect index order

Fix:

- Update stats
- Review histogram
- Validate parameter data types

💡 **Interviewer Impression:** Strong optimizer understanding

🧠 **Whiteboard Scenario 3: "High CPU but Low IO"**

🔴 **Problem Statement**

CPU is pegged at 90%, IO is low.

Ideal Thought Process

- Likely poor execution plans
- Cardinality estimation errors
- Excessive parallelism
- Scalar UDFs or row-by-row operations

👉 **First action:** Check execution plans and statistics

🔥 **P1 PRODUCTION OUTAGE SIMULATION**

(This is where interviews are won or lost)

🚨 **P1 Alert (02:17 AM)**

All reports are timing out.

CPU at 95%.

Business is blocked.

⏱️ **Minute 0–5: Stabilize**

Immediate Actions

- Confirm scope (all users vs subset)
- Identify top CPU queries
- Capture plans & wait stats

🧠 **Golden Rule:**

"Stabilize first. Fix later."

⏱️ **Minute 5–10: Identify Root Cause**

Findings:

- Same stored procedure dominating CPU
- Large discrepancy:
  Estimated rows = 100
  Actual rows = 5,000,000

👉 **Conclusion:** Bad statistics / parameter sniffing

⏱️ **Minute 10–15: Emergency Mitigation**

Fast, Low-Risk Fixes

```
EXEC sp_recompile 'dbo.usp_GenerateSalesReport';
OR
UPDATE STATISTICS Sales.SalesOrderDetail WITH FULLSCAN;
```

✓ CPU drops

✓ Queries recover

✓ Users unblocked

⏱️ **Minute 15–20: Validate & Monitor**

- Confirm execution plan change
- Check memory grants
- Monitor waits (CXPACKET, RESOURCE_SEMAPHORE)

⏱️ **Minute 20–30: Root Cause & Prevention**

Root Cause:

- ETL inserted skewed data
- Statistics not updated
- Cached plan reused incorrectly

Permanent Fix:

- Scheduled FULLSCAN stats after ETL
- Filtered statistics for skewed columns
- Query Store plan forcing (if needed)

🧠 **Executive-Level Explanation (Critical)**

“The outage was caused by stale statistics leading to incorrect cardinality estimates.
SQL Server selected a plan optimized for small data but reused it for large volumes, exhausting CPU.”

💥 **This line screams Principal DBA.**