Cardinality Estimation Issues with Large Tables
Miloš Radivojević

Data Platform MVP
Principal Database Consultant, bwin Vienna, Austria
Co-Founder: SQL Pass Austria
Conference Speaker, Book Author

Contact: MRadivojevic@gvcgroup.com
www.bwinparty.com
Twitter: @MilosSQL
Execution Plan depends on

Configuration
Hardware
Cardinality Estimations
Query Hints
Indexes
Joins
Filter Predicated
Statistics

Data Distribution (Distinct Values)
How query is written (preventing optimizations)
SQL Server Query Processing

PARSING
- Checking T-SQL Syntax, usage of reserved words, column and table names, generating a parse tree

BINDING
- Name resolution, checking permissions, aggregate and group binding, generating a query tree

OPTIMIZING
- Finding good enough way to retrieve the data, perform simplification and cost based optimization, generating an executable plan

EXECUTING
- Plan forwarded to the Execution Engine
Cardinality Estimator

Query Optimizer rely on accurate cardinality estimation (CE) models to produce reliable and efficient execution plans.

CE is used to help Query Optimizer to choose between competing execution plans.

QO should choose appropriate physical operators for logical operations.

There are 69 operators.

What is SQL Server Statistics?

Light-weight objects containing statistical information about the distribution of values in one or more columns.

The query optimizer uses these statistics to estimate the cardinality.

These cardinality estimates enable the query optimizer to create a high-quality query plan.

SQL Server estimates cardinalities come primarily from histograms.
Statistic Object in SQL Server

DBCC SHOW_STATISTICS ('Sales.Orders', 'FK_Sales_Orders_ContactPersonID');

SELECT * FROM sys.dm_db_stats_properties (object_id('Sales.Orders'), 3);

SELECT * FROM sys.dm_db_stats_histogram (object_id('Sales.Orders'), 5);
Creating Statistics in SQL Server

When stats is created?
- when an index is created
- when a column is used as predicate in a query and stats on it does not exist

Composite indexes creates multi column statistics
Histogram exists for leading column only

AUTO_CREATE_STATISTICS is ON => the Query Optimizer creates statistics on individual columns in the query predicate, as necessary, to improve cardinality estimates for the query plan
Updating Statistics in SQL Server

When stats is updated?
- Index rebuild
- When update threshold is reached
  - $CL < 130$ threshold $= 0.2 \times C + 500$ changed rows
  - $CL \geq 130$ threshold $= \sqrt{1000 \times C}$ changed rows, $C =$ table cardinality

AUTO_UPDATE_STATISTICS
- SYNC
- ASYNC, recommended for large OLTP tables
Usage of SQL Server Statistics

```
SELECT * FROM dbo.Orders WHERE orderdate = ?
```

---

**Q1**
```
SELECT * FROM dbo.Orders
WHERE orderdate = '20130629';
```

**Q2**
```
SELECT * FROM dbo.Orders
WHERE orderdate = '20130628';
```

**Q3**
```
DECLARE @date AS DATETIME = '20130629';
SELECT * FROM dbo.Orders WHERE orderdate = @date;
```
Single Table and Two Predicates

SQL Server 2012
Predicates are completely independent
Selectivity1 * Selectivity2 * NumberOfRows

SQL Server 2014+
Predicates are a little bit correlated
Selectivity1 * SQRT(Selectivity2) * NumberOfRows
## Single Table and Two Predicates

Cardinality Estimation Per Number of Predicates (Old CE / New CE)

<table>
<thead>
<tr>
<th>No of Predicates</th>
<th>Old CE (SQL Server 2012)</th>
<th>New CE (SQL Server 2014)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>( S_1 \times S_2 \times R )</td>
<td>( S_1 \times \sqrt{S_2} \times R )</td>
</tr>
<tr>
<td>3</td>
<td>( S_1 \times S_2 \times S_3 \times R )</td>
<td>( S_1 \times \sqrt{S_2} \times \sqrt{S_3} \times R )</td>
</tr>
<tr>
<td>4</td>
<td>( S_1 \times S_2 \times S_3 \times S_4 \times R )</td>
<td>( S_1 \times \sqrt{S_2} \times \sqrt{S_3} \times \sqrt{\sqrt{S_4}} \times R )</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( N )</td>
<td>( \left( \prod_{i=1}^{N} S_i \right) \times R )</td>
<td>( \left( \prod_{i=1}^{N} 2^{i-1} \sqrt{S_i} \right) \times R )</td>
</tr>
</tbody>
</table>

\( S_1, S_2, \ldots, S_N \) – ordered selectivities, where \( S_1 \) is the lowest selectivity
CREATE OR ALTER PROCEDURE dbo.usp_GetLoadedStatistics (@query NVARCHAR (3000))
AS
BEGIN

DECLARE @TSQL NVARCHAR (4000) = ''

SET NOCOUNT ON
SET @TSQL = CONCAT (@query, 
    ' option(recompile, 
    querytraceon 2363, 
    querytraceon 3604) 
)

EXEC (@TSQL);
SET NOCOUNT OFF
END
GO

---

Column: QCQL: [BetAndWin].[dbo].[tabSportsbookPP_EvalInfoSendingPacket2].frReaderID

Selectivity: 1

Stats collection generated:
CStCollFilter(ID=7, CARD=2688992)
CStCollBaseTable(ID=1, CARD=2688992 TBL: tabSportsbookPP_EvalInfoSendingPacket2)

End selectivity computation

Estimating distinct count in utility function

Input stats collection:
CStCollFilter(ID=7, CARD=2688992)
CStCollBaseTable(ID=1, CARD=2688992 TBL: tabSportsbookPP_EvalInfoSendingPacket2)

Columns to distinct on: QCQL: [BetAndWin].[dbo].[tabSportsbookPP_EvalInfoSendingPacket2].frReaderID QCQL: [BetAndWin].[dbo].[tabSportsbook

Plan for computation:
CIVCPlanUniqueKey

Result of computation: 2688992
Why Cardinality Estimator is so Important?

Cardinality estimation errors – reason #1 for poor execution plans
Discrepancy in estimations can lead to:
  - Inappropriate join strategies
  - Serial plan when a parallel plan would be more appropriate
  - Memory spills to disk or wasted memory
Sample Tables

100 M (11.2 GB)

**A**

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Allow Nulls</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>int</td>
<td></td>
</tr>
<tr>
<td>p1</td>
<td>int</td>
<td></td>
</tr>
<tr>
<td>c1</td>
<td>char(100)</td>
<td></td>
</tr>
</tbody>
</table>

Parent

335 M (37.5 GB)

**B**

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Allow Nulls</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>int</td>
<td></td>
</tr>
<tr>
<td>p1</td>
<td>int</td>
<td></td>
</tr>
<tr>
<td>c1</td>
<td>char(100)</td>
<td></td>
</tr>
</tbody>
</table>

Child
Sample Tables

CREATE NONCLUSTERED INDEX [ix1] ON [dbo].[A]
(
    [p1] ASC
)
GO

CREATE NONCLUSTERED INDEX [ix1] ON [dbo].[B]
(
    [p1] ASC
)
GO

CREATE NONCLUSTERED INDEX [ix1] (Non-Unique, Non-Clustered)
PK_xTGame (Clustered)

CREATE NONCLUSTERED INDEX [ix1] (Non-Unique, Non-Clustered)
PK_xTGame (Clustered)

CREATE NONCLUSTERED INDEX [ix1] (Non-Unique, Non-Clustered)
PK_xTGame (Clustered)
SELECT * FROM A
INNER JOIN B ON A.id = B.pid
WHERE A.pid = 413032;

Query returns 856 rows

Query execution took about 80 seconds!
SELECT * FROM A
WHERE pid = 413032;

Query 1: Query cost (relative to the batch): 100%
SELECT * FROM [A] WHERE [A].[pid]=@1
Missing Index (Impact 99.8772): CREATE NONCLUSTERED INDEX [<Name of Missing Index, sysname,>] ON [dbo].[A] ([p:

> 100x !!!
SELECT * FROM A
INNER JOIN B ON A.id = B.pid WHERE A.pid = 413032;

Query 1: Query cost (relative to the batch): 100%
SELECT * FROM A INNER JOIN B ON A.id = B.pid WHERE A.pid = 413032
Missing Index (Impact 98.1585): CREATE NONCLUSTERED INDEX [Name of Missing Index, sysnam]
SELECT * FROM A
INNER JOIN B ON A.id = B.pid WHERE A.pid = 413032;

SQL Server Execution Times:
  CPU time = 0 ms, elapsed time = 0 ms.
SQL Server parse and compile time:
  CPU time = 0 ms, elapsed time = 0 ms.

(856 rows affected)
Table 'Workfile'. Scan count 0, logical reads 0, physical reads 0, read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.
Table 'Worktable'. Scan count 0, logical reads 0, physical reads 0, read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.
Table 'B'. Scan count 1, logical reads 4915691, physical reads 10138, read-ahead reads 4976384, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.
Table 'A'. Scan count 1, logical reads 1184, physical reads 0, read-ahead reads 17, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.

(1 row affected)

SQL Server Execution Times:
  CPU time = 101437 ms, elapsed time = 211765 ms.
SQL Server parse and compile time:
  CPU time = 0 ms, elapsed time = 0 ms.

SQL Server Execution Times:
  CPU time = 0 ms, elapsed time = 0 ms.
```
SELECT * FROM A
INNER JOIN B ON A.id = B.pid WHERE A.pid = 413032;
```
### Statistics Discrepancy in SQL Server

**DBCC SHOW_STATISTICS ('A', 'ix1');**

<table>
<thead>
<tr>
<th>Name</th>
<th>Updated</th>
<th>Rows</th>
<th>Rows Sampled</th>
<th>Steps</th>
<th>Density</th>
<th>Average key length</th>
</tr>
</thead>
<tbody>
<tr>
<td>ix1</td>
<td>Oct 5 2018 12:08PM</td>
<td>100000000</td>
<td>440941</td>
<td>171</td>
<td>0.1763478</td>
<td>8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>All density</th>
<th>Average Length</th>
<th>Columns</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.286786E-05</td>
<td>4</td>
<td>pid</td>
<td>id</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RANGE_HI_KEY</th>
<th>RANGE_ROWS</th>
<th>EQROWS</th>
<th>DISTINCT_RANGE_ROWS</th>
<th>AVG_RANGE_ROWS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>201855,4</td>
<td>1807246</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>76852</td>
<td>1603931</td>
<td>248</td>
<td>814,0501</td>
</tr>
<tr>
<td>3</td>
<td>788206,1</td>
<td>7830,796</td>
<td>971</td>
<td>781,0396</td>
</tr>
<tr>
<td>4</td>
<td>50654,9</td>
<td>7906,701</td>
<td>523</td>
<td>974,6106</td>
</tr>
<tr>
<td>5</td>
<td>263999,6</td>
<td>32078,61</td>
<td>207</td>
<td>1276,564</td>
</tr>
<tr>
<td>6</td>
<td>356384,5</td>
<td>11069,38</td>
<td>306</td>
<td>1194,796</td>
</tr>
<tr>
<td>7</td>
<td>59330,3</td>
<td>4744,021</td>
<td>887</td>
<td>1125,021</td>
</tr>
<tr>
<td>8</td>
<td>86457</td>
<td>9036,229</td>
<td>659</td>
<td>1237,389</td>
</tr>
<tr>
<td>9</td>
<td>601909,9</td>
<td>9036,229</td>
<td>536</td>
<td>1272,196</td>
</tr>
<tr>
<td>10</td>
<td>483772,5</td>
<td>2936,774</td>
<td>401</td>
<td>1297,754</td>
</tr>
<tr>
<td>11</td>
<td>348370,6</td>
<td>11069,38</td>
<td>270</td>
<td>1290,023</td>
</tr>
<tr>
<td>12</td>
<td>202535,8</td>
<td>4969,926</td>
<td>155</td>
<td>1310,04</td>
</tr>
<tr>
<td>13</td>
<td>65523,1</td>
<td>11069,38</td>
<td>508</td>
<td>1289,908</td>
</tr>
<tr>
<td>14</td>
<td>329999,5</td>
<td>11069,38</td>
<td>233</td>
<td>1416,579</td>
</tr>
<tr>
<td>15</td>
<td>355655,1</td>
<td>11069,38</td>
<td>267</td>
<td>1332,6</td>
</tr>
<tr>
<td>16</td>
<td>187793,5</td>
<td>7003,078</td>
<td>130</td>
<td>1450,082</td>
</tr>
</tbody>
</table>
statistics discrepancy in SQL server

```sql
SELECT * FROM sys.dm_db_stats_histogram(OBJECT_ID('A'), 2);
```
Statistics Discrepancy in SQL Server

Actual number: 288, Estimation number: 32.078

Statistics is up-to-date, but is completely wrong
  Actual maximal number of rows with the same pid is 1.617 – it cannot be more than this!
  Statistics shows 20 more than this!

Why is it so wrong?
Wrong Statistics - Demo
Updating Statistics in SQL Server

DROP TABLE IF EXISTS dbo.T;
GO
CREATE TABLE dbo.T(
    id INT NOT NULL,
    c1 INT NOT NULL,
    c2 CHAR(100) NULL,
    c3 CHAR(100) NULL,
    CONSTRAINT PK_T PRIMARY KEY CLUSTERED (id ASC)
) GO

DECLARE @n INT = 1000000;
INSERT INTO dbo.T(id, c1, c2, c3)
SELECT n, 1 + ABS(CHECKSUM(NEWID())) % @n AS custid,
'cust' + CAST(n AS VARCHAR),
'test'
FROM dbo.GetNums(@n)
GO
CREATE INDEX ix1 ON T(c1)
GO
UPDATE STATISTICS T ix1;
GO
Updating Statistics in SQL Server

UPDATE STATISTICS T ix1;

SELECT StatMan([SC0], [SB0000]) FROM (SELECT TOP 100 PERCENT [SC0], step_direction([SC0]) over (order by NULL) AS [SB0000] FROM (SELECT [id] AS [SC0] FROM [dbo].[T] TABLESAMPLE SYSTEM (9.666242e-001 PERCENT) WITH (READUNCOMMITTED) ) AS _MS_UPDSTATS_TBL_HELPER ORDER BY [SC0], [SB0000] ) AS _MS_UPDSTATS_TBL OPTION (MAXDOP 16)
# Updating Statistics in SQL Server

<table>
<thead>
<tr>
<th>Rows</th>
<th>Sampled Rows</th>
<th>%</th>
<th>Max (Stat)</th>
<th>Max Key (Stat)</th>
<th>Key (Actual)</th>
<th>Max (Actual)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.000</td>
<td>10.000</td>
<td>100</td>
<td>7</td>
<td>4338</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>20.000</td>
<td>20.000</td>
<td>100</td>
<td>6</td>
<td>4347</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>30.000</td>
<td>30.000</td>
<td>100</td>
<td>6</td>
<td>1062</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>40.000</td>
<td>40.000</td>
<td>100</td>
<td>7</td>
<td>837</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>50.000</td>
<td>43.007</td>
<td>86.01</td>
<td>8</td>
<td>4492</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>100.000</td>
<td>46.472</td>
<td>46.47</td>
<td>12</td>
<td>8368</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>150.000</td>
<td>49.434</td>
<td>32.96</td>
<td>15</td>
<td>9976</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>200.000</td>
<td>51.319</td>
<td>25.66</td>
<td>15</td>
<td>3030</td>
<td>5</td>
<td>7</td>
</tr>
</tbody>
</table>
## Updating Statistics in SQL Server

<table>
<thead>
<tr>
<th>Rows</th>
<th>Sampled Rows</th>
<th>%</th>
<th>Max (Stat)</th>
<th>Max Key (Stat)</th>
<th>Key (Actual)</th>
<th>Max (Actual)</th>
</tr>
</thead>
<tbody>
<tr>
<td>300.000</td>
<td>53.095</td>
<td>17.70</td>
<td>22</td>
<td>30467</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>500.000</td>
<td>57.979</td>
<td>11.60</td>
<td>24</td>
<td>13136</td>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>1.000.000</td>
<td>68.080</td>
<td>6.81</td>
<td>36</td>
<td>27086</td>
<td>4</td>
<td>9</td>
</tr>
<tr>
<td>2.000.000</td>
<td>83.953</td>
<td>4.20</td>
<td>67</td>
<td>20535</td>
<td>3</td>
<td>9</td>
</tr>
<tr>
<td>5.000.000</td>
<td>111.148</td>
<td>2.22</td>
<td>71</td>
<td>32020</td>
<td>3</td>
<td>10</td>
</tr>
<tr>
<td>10.000.000</td>
<td>148.074</td>
<td>1.48</td>
<td>107</td>
<td>178009</td>
<td>3</td>
<td>10</td>
</tr>
<tr>
<td>20.000.000</td>
<td>192.992</td>
<td>0.96</td>
<td>164</td>
<td>33861</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>50.000.000</td>
<td>295.297</td>
<td>0.59</td>
<td>268</td>
<td>216580</td>
<td>4</td>
<td>13</td>
</tr>
</tbody>
</table>
# Updating Statistics in SQL Server

## Table A – Statistics IX1

<table>
<thead>
<tr>
<th>Rows</th>
<th>Sampled Rows</th>
<th>%</th>
<th>Max (Stat)</th>
<th>Max Key (Stat)</th>
<th>Key (Actual)</th>
<th>Max (Actual)</th>
</tr>
</thead>
<tbody>
<tr>
<td>100,000,000</td>
<td>440.941</td>
<td>0.44</td>
<td>32.078</td>
<td>413032</td>
<td>288</td>
<td>1.617</td>
</tr>
</tbody>
</table>

## Table B – Statistics IX1

<table>
<thead>
<tr>
<th>Rows</th>
<th>Sampled Rows</th>
<th>%</th>
<th>Max (Stat)</th>
<th>Max Key (Stat)</th>
<th>Key (Actual)</th>
<th>Max (Actual)</th>
</tr>
</thead>
<tbody>
<tr>
<td>335,305,134</td>
<td>803.920</td>
<td>0.24</td>
<td>29.498</td>
<td>50623957</td>
<td>136</td>
<td>394</td>
</tr>
</tbody>
</table>
Query, Table and Index HINTS
JOIN, OPTIMIZE FOR, MAX_GRANT_PERCENT = 0.001

Using table variable to “fix” cardinality errors
Be careful with SQL Server 2019

It is important to understand why discrepancy exists
OPTION (QUERYTRACEON 3604, QUERYTRACEON 2363)
Always UPDATE statistics with FULLSCAN or high sample rate

Use the PERSIST_SAMPLE_PERCENT option to ensure that for all future updates a full scan will be performed

```
UPDATE STATISTICS dbo.A ix1 WITH FULLSCAN, PERSIST_SAMPLE_PERCENT = ON;
```

Available from SQL Server 2016 SP1 CU4 and SQL Server 2017 CU1

Ensure that you updated all statistics with a given as a leading column (_WA…)

Check and judge statistics for large tables

Check loaded statistics (TF 2363, 3604)

Implement workarounds, but try to find root cause for a problem
Why people don’t update statistics with FULLSCAN?
• Because it is not default
• Because they think it is too expensive

How to calculate statistics on a clone database and import it (inject it) into the production database?
- Undocumented! On your own risk!

<update_stats_stream_option> ::= [ STATS_STREAM = stats_stream ] [ ROWCOUNT = numeric_constant ] [ PAGECOUNT = numeric_constant ]