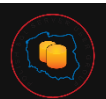
NASI SPONSORZY I PARTNERZY

Indexing for Denormalization

Why schema design matters

or

How to use indices to get the best from both (OLTP and DW) worlds



Introducing Marcin Szeliga

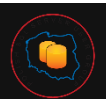
- Over 15 years' experience with SQL Server
- Trainer & Consultant
- Books & Articles Writer
- OLTP and DW/BI Systems Architect
- SQL Microsoft Most Valuable Professional since 2006
- Founder of SQLExpert.pl
  - marcin@sqlexpert.pl
  - http://blog.sqlexpert.pl/
  - https://www.facebook.com/SQLExpertpl

SQL EXPERT.pl

MVP Microsoft® Most Valuable Professional

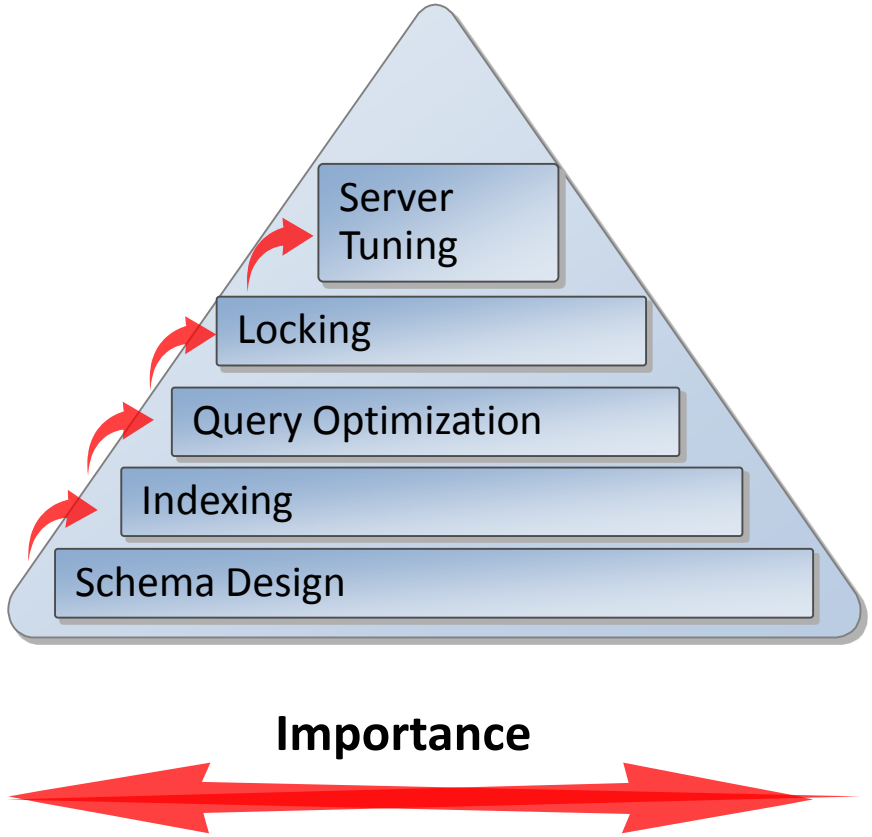
Agenda

- Performance Optimization in a Nutshell
- OLTP Databases Modeling Rules
- Denormalization for Optimal Query Performance
  - Non-Clustered Indices and SARG arguments
  - Triggers versus Indices on Computed Columns
- Denormalization for Real World OLTP Databases
  - Computed Column Limitations
  - Basic Indexing Strategy
  - POC (Partitioning, Ordering, and Covering) Indices
  - Indexed Views



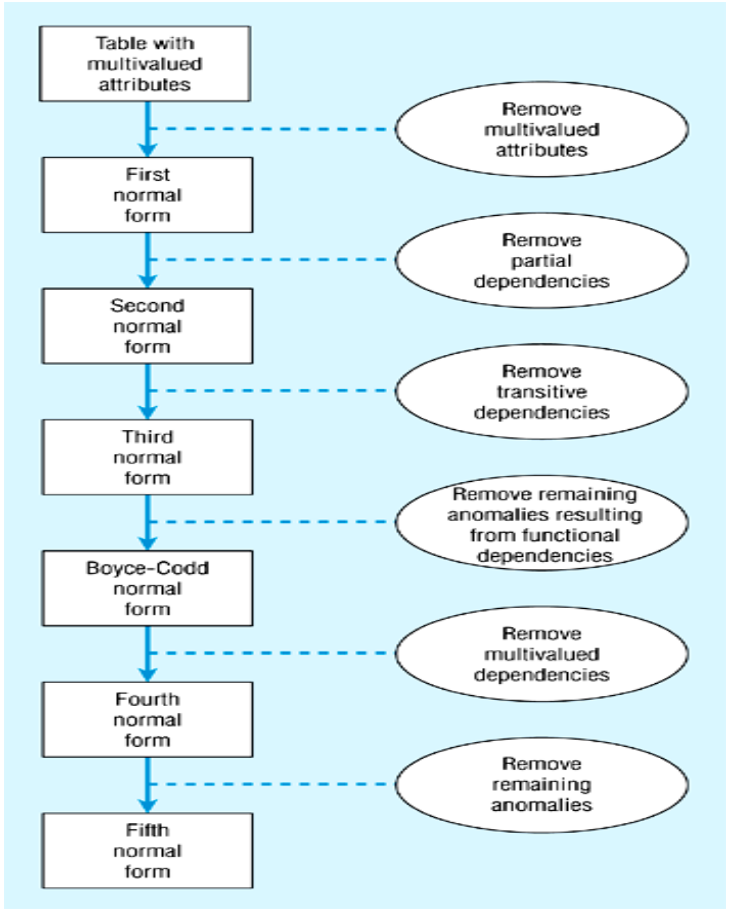
Performance Optimization Model

Tasks in the Performance Optimization Model



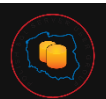
Problem with Normalization

- Well-structured relations:
  - Contain minimal redundancy
  - Allow insertion, modification, and deletion without errors or inconsistencies
- Normalization is all about avoiding I/U/D anomalies
  - Not at all about performance



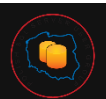
Solution - Denormalization

- Consider denormalization if:
  - Performance with the normalized design is unacceptable
  - **Anomalies caused by denormalization can be avoided** or at least predicted
- Use denormalization to improve performance by:
  - Moving calculations to server side and use precomputed results efficiently
  - Pre-aggregating data
  - Avoiding complex joins



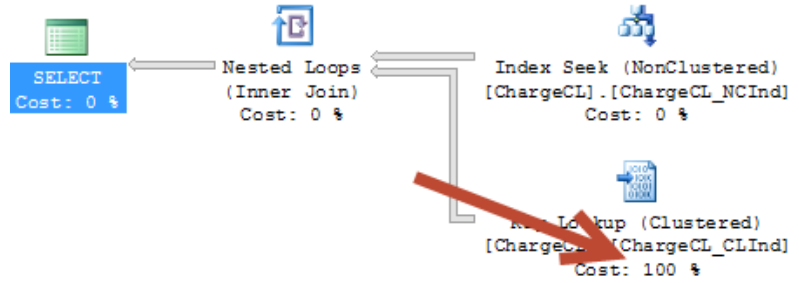
Indices for Optimal Query Performance

- Indices serve three main purposes:
  - Are used to limit (or minimize) the amount of data being read to execute queries
  - Can eliminate costly operators (like scan, sort or aggregate) from query execution plans
  - Can tremendously improve concurrency
- For denormalization, they shoud be built on computed columns and (sometimes) on views



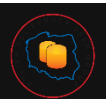
But Non-clustered Indices are Rarely Used

- Query must be selective enough:
  - If an index covers the query, SQL Server always use it
    - One-column non-clustered indices are not so useful
      - Foreign keys are exceptions
    - Otherwise, with too many lookups it will be too expensive
  - **For non-covering indices the tipping point is about 1 percent rows**
- SQL Server has to know this in advance
  - Estimations are based on statistics or assumptions
    - **Do you use positive search arguments?**
    - **Do you isolate the column to one side of the expression?**
      - » MonthlySalary > value/12 (constant, seekable)
      - » MonthlySalary * 12 > value (must scan)
      - » What about UPPER (Name) LIKE 'DO%' ?



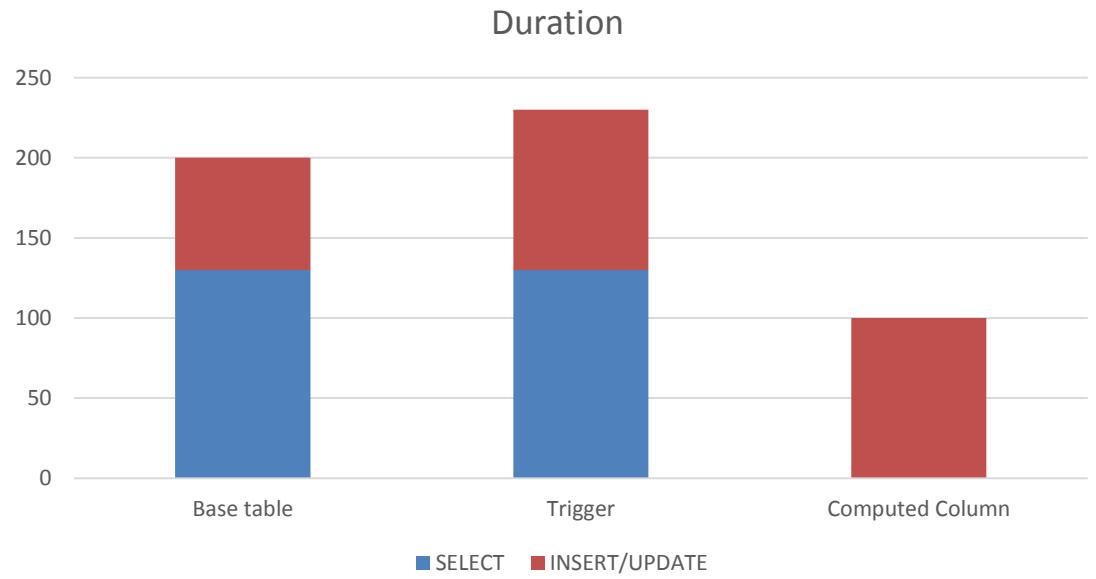
Indices for Limited Denormalization

- There is a big difference between unnormalized and denormalized schema
  - You have to know how you are moving away from the normalized design, one step at a time
- **Denormalization duplicates data**
  - You will have to deal with data bloat problem
  - DML operations will have to write to multiple tables
- **Indices also contain duplicate data**
  - They are easy to create and maintain without breaking (almost) any functionality
  - SQL Server will take care of data synchronization and will do this as fast as possible



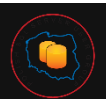
Demo

- Indices on Computed Columns
- Indices andUser Defined Functions



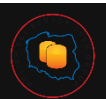
Recap - Denormalization with Non-Clustered Indices

- You have to somehow add redundant data
  - By adding an additional column, populate it with a trigger and putting an index on it
    - **This simply does not work**
  - By adding a computed column (an expression or UDF), and index it
    - **Clean and error-proof solution**
    - Queries don't have to be changed in any way, SQL Server is smart enough to figure out that exact the same expression/function was used in a query and in the computed column definition
  - There is also a hidden cost …
    - AskMicrosoft to fix this DBCC nuisance



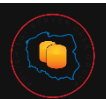
Denormalization for Real World OLTP Databases

- In real life there is no such thing as a pure OLTP database
- Computed columns have some limitations
  - Cannot be indexed if a function does user or system data access
- Indexed views are a hugely underused tool for achieving limited denormalization
  - Minimum amount of data being read
  - TempDB access not necessary
    - No worktables are necessary
  - But performance of data modification statements should be tested



Indexing for Aggregation
Starting Point

- Without a useful index SQL Server has to:
  - Scan the whole table — generally bad idea
  - Compute aggregate (i.e. sum) using hash aggregate — bad in OLTP environments for a couple of reasons:
    - This operator is a blocking one
    - It is computation heavy
    - It requires a memory grant
  - Sort the results



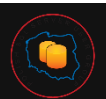
Indexing for Aggregation
With Just an Index

- With a covering, non-clustred index SQL Server has to do exactly the same work
  - But on limited amount of data

```
Query 1: Query cost (relative to the batch): 100%
SELECT [ProductID], SUM([UnitPrice]) AS TotalPrice FROM dbo.[SalesOrderDetail]
```

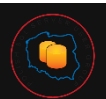
Indexing for Aggregation
With a POC Index

- The key columns order obviously matters
  - The best kind of indices for grouping queries are POC ones
    - The first key column(s) should be the one(s) used for partitioning
    - The second one(s) — for ordering
    - The last one(s) — for covering
- With a POC index SQL Server can:
  - Replace hash aggregate with stream one
    - It is a non-blocking operator, and does not require a memory grant
  - Omit sorting altogether



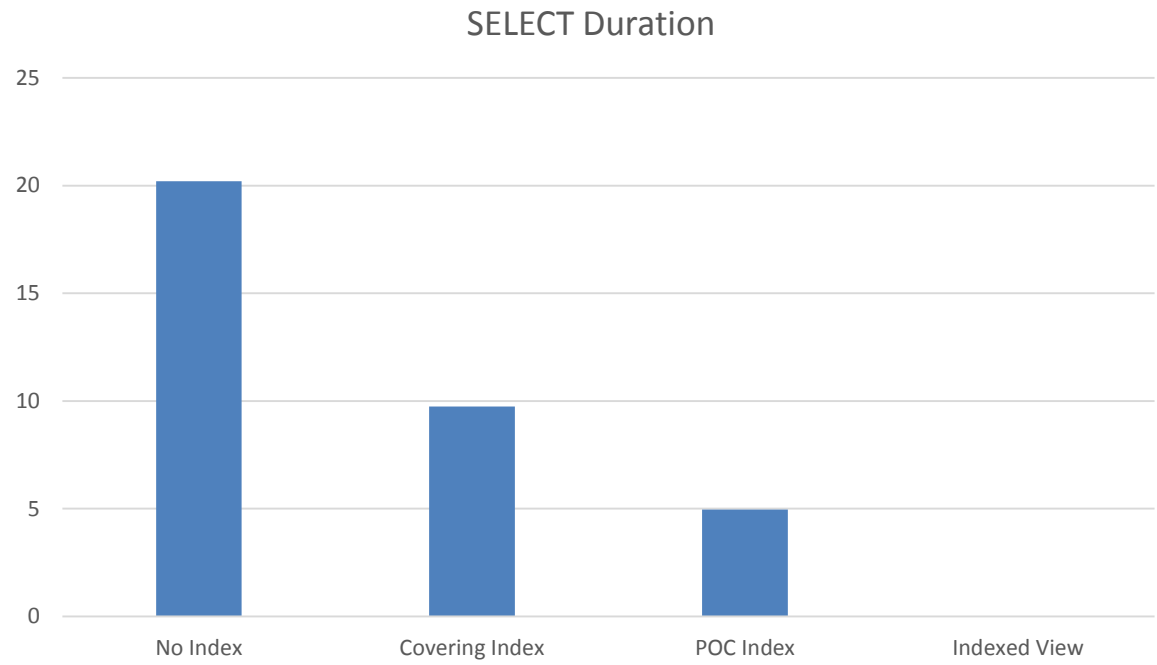
Indexing for Aggregation With an Indexed View

- With an indexed view SQL Server will just read pre-aggregated data
- This kind of views have a lot of requirements
  - Check “Create Indexed Views” topic in BOL
- If you are using Enterprise Edition, you just need to create one (at least, in theory)
- **All other editions will happily ignore this index, unless the NOEXPAND is used in a query**
- Because changing a client code is tedious, we have to put this hint on a server side
  - **The best place to place it is inside an another view**



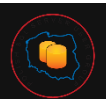
Demo

- Designing Indicies for Aggregation in OLTP Databases



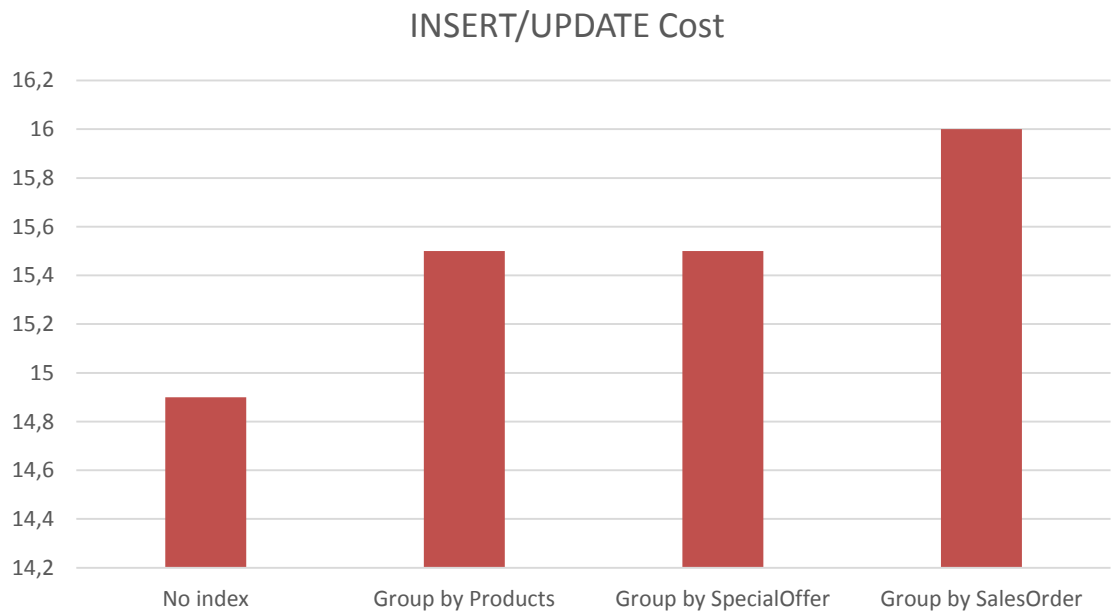
How Costly an Indexed View Can Be?

- There is always an additional cost of data modification associated with an indexed view
  - This cost varies greatly
- Aggregated set should be small but not too small to avoid
  - Create a hot row spot of activity (when set is too small)
  - Excessive blocking (when set is too big)
    - GROUP BY member_no – probably OK
    - GROUP BY state – too few rows in aggregate
    - GROUP BY country – avoid like the plague



Demo

- Assessing the Overhead of Data Manipulation for Different Aggregations



Take Home Messages

- Databases are normalized to minimize the impact of update anomalies
- After that databases should be denormalized to get performance gain
  - This gain does not come out of thin air
- Indices on computed columns are great for speed up queries that use calculations
- Indexed views are the ultimate solution for aggregations and joins
- **Briefly, normalize till it hurts, then denormalize till it works** …

NASI SPONSORZY I PARTNERZY

Organizacja: Polskie Stowarzyszenie Użytkowników SQL Server - PLSSUG