Storage, Indexing and Views

Samantha Krall

SYM-400

There are a few different reasons that the queries can be returning results slowly. The SQL code could be written badly, affecting the query times. Despite many databases having query optimizers, human inefficiency can not be ignored. One example of this is SELECT \*. Although it may get the right result from a query, it may take a long time in production with real data. Performing several table joins on a dev database with small sets of data will always run quickly, though executing the same query in production with millions of rows in tables could grind an application to halt.

A badly designed database schema will often contribute to bad database performance. Too little or too much normalization can mean the database and your queries must work harder to get the data you need. A bad database design can increase the complexity of SQL queries and the cost of change down the road. Joins can be expensive operations on big data sets, so it is best to fully understand the needs of the application(s) and its data before jumping into building schemas. You can not predict requirements or change but you can plan and mitigate how your data model may evolve over time so that altering schemas in the future is less painful.

Use the built-in database features. Stored procedures require a bit more effort but make applications less complex and costly to maintain. Databases are good at processing, managing and querying data- way more efficient than any developer embedding SQL queries in application code, which can cause result sets to be dumped in application server memory so they can be iterated through and processed. Less is more when it comes to accessing the database- so keep your hits low, let the database do the work for you and only bring back the data you need. Persisting lots of data back and forth between the application server and database is a prime reason why OutOfMemory exceptions can occur, along with frequent garbage collection.

Most databases have query optimizers to figure out the most efficient way of retrieving data. The output from a query optimizer is known as an execution plan that describes the optimum method and steps to retrieve the data. You might have heard of the Cost Based Optimizer (CBO) in Oracle which the vendor itself recommends. Things like database dictionaries and statistics store important meta-data (e.g. table sizes, row counts, index info) which help the optimizer decide on the optimal execution plan. For example, it is a lot faster to scan a table with one hundred rows than a million rows, but if the optimizer does not know that a table has a million rows, it produces an execution plan blind to this fact. This is common in test environments where database schemas are copied from production and imported without statistics. Everything looks great, but the application SQL queries run slow because the database statistics are out of date. Assuming database statistics are accurate, the most common cause of query latency is full table scans where the data being queried has not been indexed. The database must fetch and scan through all records in the table you are accessing, and you will see a lot of I/O wait as a result. Indexes can often be a silver bullet, and it is true that they can dramatically improve business transaction response time and throughput. However, they can not be placed everywhere because indexes require disk space and need to be maintained over time as data changes and grows. Indexes can therefore speed up read latency but may introduce latency when data is written or updated.

Just as you get contention in the JVM/CLR by implementing thread synchronization on application logic and data, you can get the same performance hit in the database when queries read and write data from tables. Relational databases provide great atomicity and read/write consistency but at a cost. Good database design and considerations around indexes will ensure operations like insert, update and delete do not introduce high latency and locking in the database.

At the end of the day, databases store data on a disk. Traditional spinning disk is getting larger and cheaper, but it is not getting any faster and that can impact the performance of database queries, especially when disk is shared. For example, when a DBA provisions a database, they need one or more logical volumes to mount the database data files, index files, and redo logs. A storage administrator can provision logical volumes from anywhere. It might be allocated from a SAN or be cobbled together by chunks of disk from multiple I/O devices. So, whilst databases can be shared by many applications, so can its underlying storage and I/O capacity. If the same physical I/o device is serving two databases or applications, you can end up with hot spots on a disk where the same physical spindles are being over worked- thus creating I/O contention and read/ write latency for queries. As applications move to the cloud, storage will become more critical to both relational databases and NoSQL technologies as they rely on shared storage provisioned by cloud providers.

CREATE VIEW Production.RedView

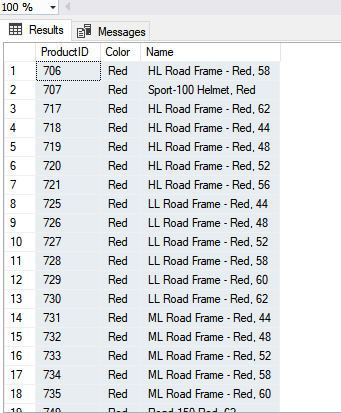
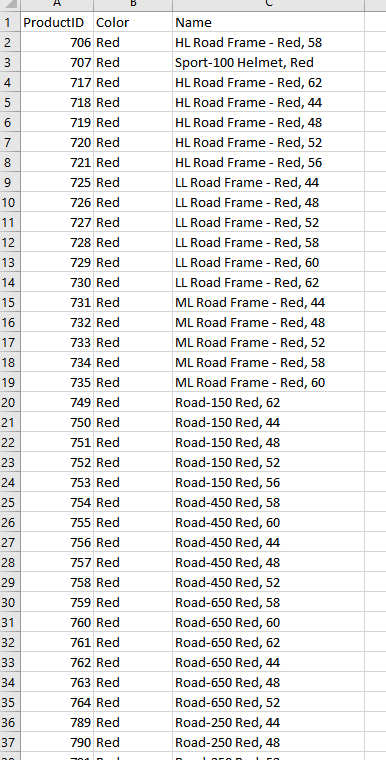
AS SELECT ProductID, Color, Name

FROM Production.Product

WHERE Color = 'Red'

SELECT \*

FROM Production.RedView

CREATE VIEW Production.FinishedProductionColorView

AS SELECT Product.ProductID,

Product.Name AS [Product Name],

Product.Color,

ProductInventory.Quantity,

Location.LocationID,

Location.Name AS [Location Name]

FROM

((Production.Product JOIN Production.ProductInventory

ON ProductInventory.ProductID = Product.ProductID)

JOIN Production.Location

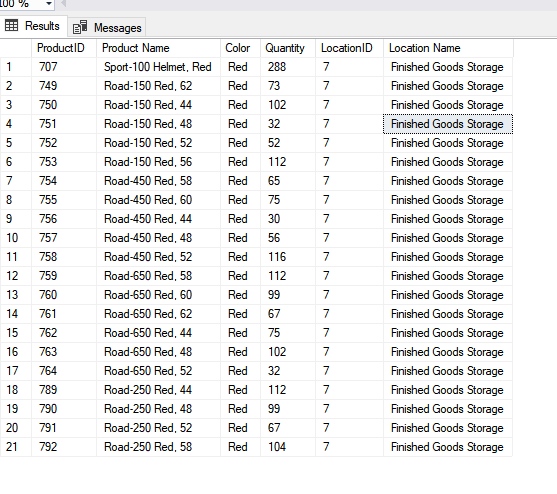
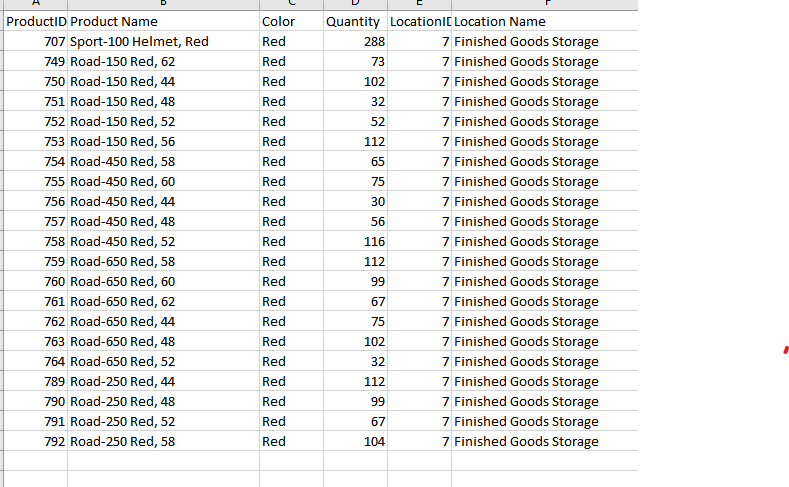
ON ProductInventory.LocationID = Location.LocationID)

WHERE Color = 'Red'

AND Location.LocationID = '7'

SELECT \*

FROM Production.FinishedProductionColorView

**References**:

<https://www.appdynamics.com/blog/engineering/slow-sql-a-usual-suspect/>

<https://www.programmerinterview.com/database-sql/what-is-an-index/>

<http://www.devx.com/vb2themax/Tip/18579>

<https://docs.microsoft.com/en-us/sql/t-sql/statements/create-view-transact-sql?view=sql-server-2017>

<https://viewer.gcu.edu/eZhFRY>