Databases 1

Course Outline

1. ~~Introduction to database approach~~
2. ~~The database environment~~
3. ~~Introduction to The Relational Model~~
4. ~~Views~~
5. ~~Transactions~~
6. ~~SQL Constraints~~
7. ~~Relational Database Design. Theory and practice~~
8. **An Introduction to Database Performance. ~~Indexing~~**
9. JSON Support in Relational Database Management Systems
10. NoSQL Databases

Week 13

Database Performance. An Introduction

Agenda

1. Query analysis
   1. Execution plans
      - Demo (let’s learn how to read execution plans)
   2. Query resource consumption

2. Index design
   1. When to use indexes
   2. Query optimizer
   3. (Some) Types of indexes and when to use them
      - Demo (let’s play with indexes)
   4. Maintaining our indexes using Dynamic Management Views

3. (Bonus) Interesting underperforming queries

1.Query Analysis

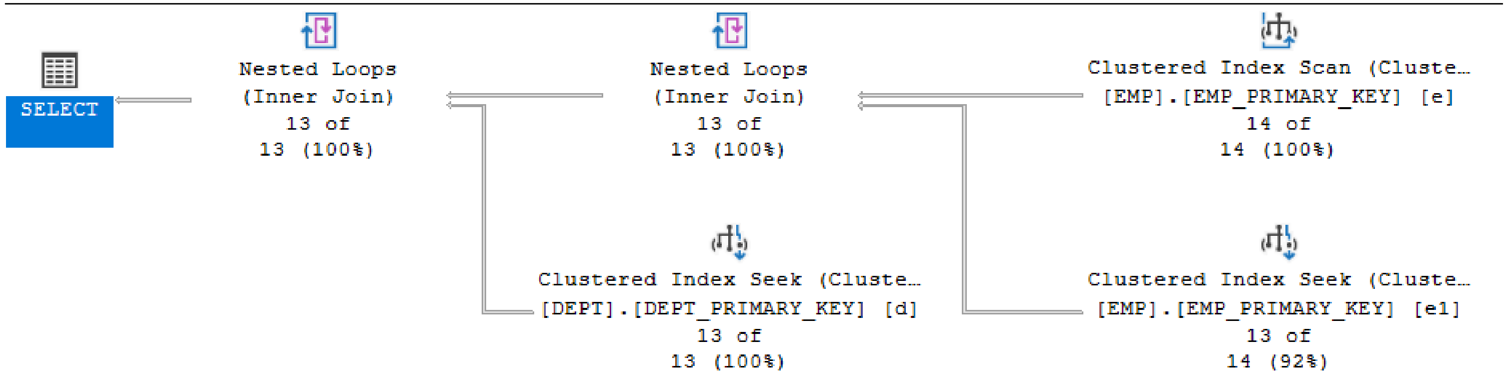
1. Execution plans

- What are execution plans?

“The way SQL Server thinks”

- How does an execution plan look like?

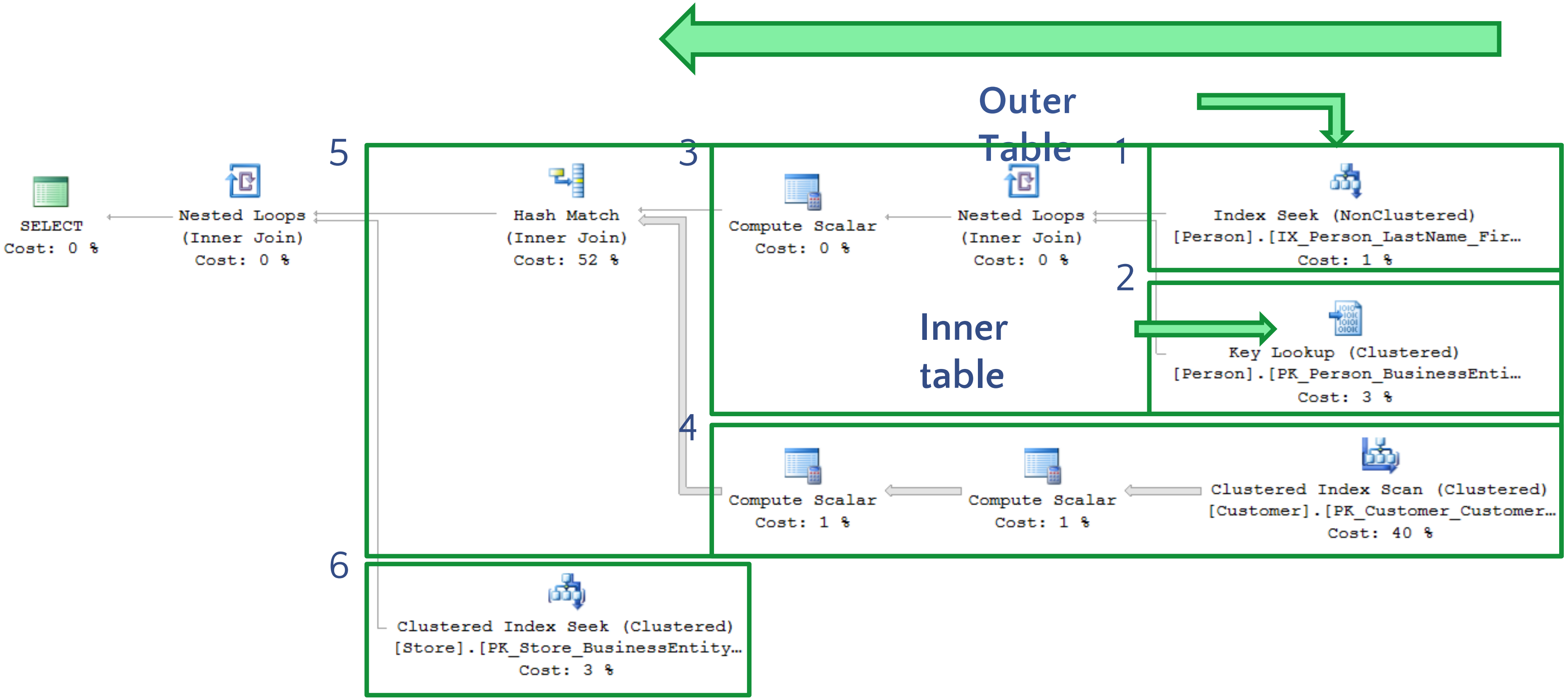
Something like this


SELECT
Nested Loops
(Inner Join)
13 of
13 (100%)
Nested Loops
(Inner Join)
13 of
13 (100%)
Clustered Index Scan (Cluste…
[EMP].[EMP_PRIMARY_KEY] [e]
14 of
14 (100%)
Clustered Index Seek (Cluste…
[DEPT].[DEPT_PRIMARY_KEY] [d]
13 of
13 (100%)
Clustered Index Seek (Cluste…
[EMP].[EMP_PRIMARY_KEY] [e1]
13 of
14 (92%)

- How do we read execution plans?

“Like Arabic and Hebrew people. From ____ to ____”

Graphical Showplan Flow



Resultset 1 and 2 are joined using a nested loops join, creating resultset 3
Resultset 3 and 4 are joined using a hash match join, creating resultset 5
Resultset 5 and 6 are joined using a nested loops join, creating a resultset for the Select clause

DEMO

1.Query Analysis

1. Execution plans

Let’s see how the following execution plan looks for the following queries

```
-- Simple query
SELECT *
FROM EMP e

-- One join
SELECT *
FROM EMP e
JOIN DEPT d ON e.DEPTNO = d.DEPTNO


-- Two Joins
SELECT *
FROM EMP e
JOIN DEPT d ON e.DEPTNO = d.DEPTNO
JOIN Emp e1 ON e.MGR = e1.EMPNO

-- Simple WHERE clause
SELECT *
FROM EMP
WHERE ENAME = 'KING'

-- JOin with WHERE clause
SELECT *
FROM EMP e
JOIN DEPT d ON e.DEPTNO = d.DEPTNO
WHERE e.ENAME = 'KING'
```

```
-- ORDER clause
SELECT *
FROM EMP e
ORDER BY SAL DESC

-- Group By and Calculation
SELECT e.DEPTNO, COUNT(e.Ename) AS CountEmp
FROM EMP e
GROUP BY e.DEPTNO

-- Everything
SELECT d.DNAME, COUNT(e.ENAME) AS CountEmp
FROM EMP e
JOIN DEPT d ON e.DEPTNO = d.DEPTNO
WHERE d.LOC = 'CHICAGO'
GROUP BY d.DNAME
HAVING COUNT(e.Ename) > 1
ORDER BY d.DNAME DESC
```

1.Query Analysis

1. Query resource consumption

As any system, databases use resources from the system it is hosted on. The main resources that a SQL server uses for each query is:

- –
- –
- –

1.Query Analysis

1. Query resource consumption

As any system, databases use resources from the system it is hosted on. The main resources that a SQL server uses for each query is:

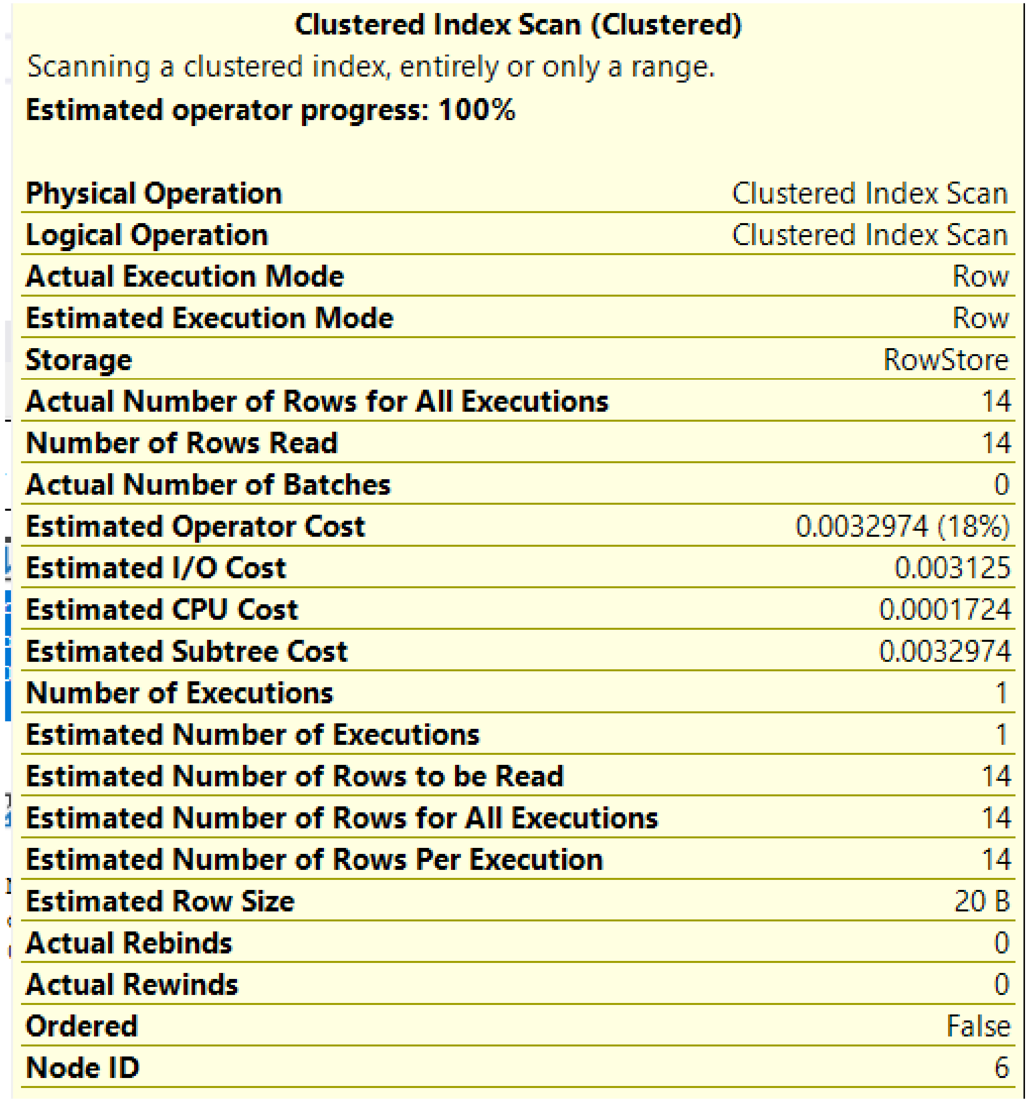
- CPU (Processor)
- Memory (RAM)
- IO (Disk writes/reads)

Our goal is to design our database and write queries in such way that we use the least resources. For this we need to examine the query execution plan for our queries and “work” with either fine tuning our queries or our database structure/indexes/etc.

1.Query Analysis

1. Query resource consumption

To understand the resources one query is using we can inspect the execution plan to understand all the required resources for each operation in a query



Clustered Index Scan (Clustered)

Scanning a clustered index, entirely or only a range.

Estimated operator progress: 100%

Physical Operation	Clustered Index Scan
Logical Operation	Clustered Index Scan
Actual Execution Mode	Row
Estimated Execution Mode	Row
Storage	RowStore
Actual Number of Rows for All Executions	14
Number of Rows Read	14
Actual Number of Batches	0
Estimated Operator Cost	0.0032974 (18%)
Estimated I/O Cost	0.003125
Estimated CPU Cost	0.0001724
Estimated Subtree Cost	0.0032974
Number of Executions	1
Estimated Number of Executions	1
Estimated Number of Rows to be Read	14
Estimated Number of Rows for All Executions	14
Estimated Number of Rows Per Execution	14
Estimated Row Size	20 B
Actual Rebinds	0
Actual Rewinds	0
Ordered	False
Node ID	6

2. Index design

1. When to use indexes

Indexes come with a lot of benefits but also come with their cost. Having that in mind we need to establish if an index is worth adding.

- Indexes inquire costs related to:

1. Disk spaces required
2. RAM
3. Fragmentation
4. Slows down INSERT/UPDATE/DELETE operation

- Benefits of an index depends on:

1. Size of table
2. Data distribution
3. Query vs. update load

2. Index design recommendations

1. Bigger the table, index is more valuable

2. If not often SELECT then the cost of index maintenance may be greater than the benefits

3. Data distribution – do not create indexes on Boolean or fixed set values (e.g. days of week etc)

4. Delete unused indexes

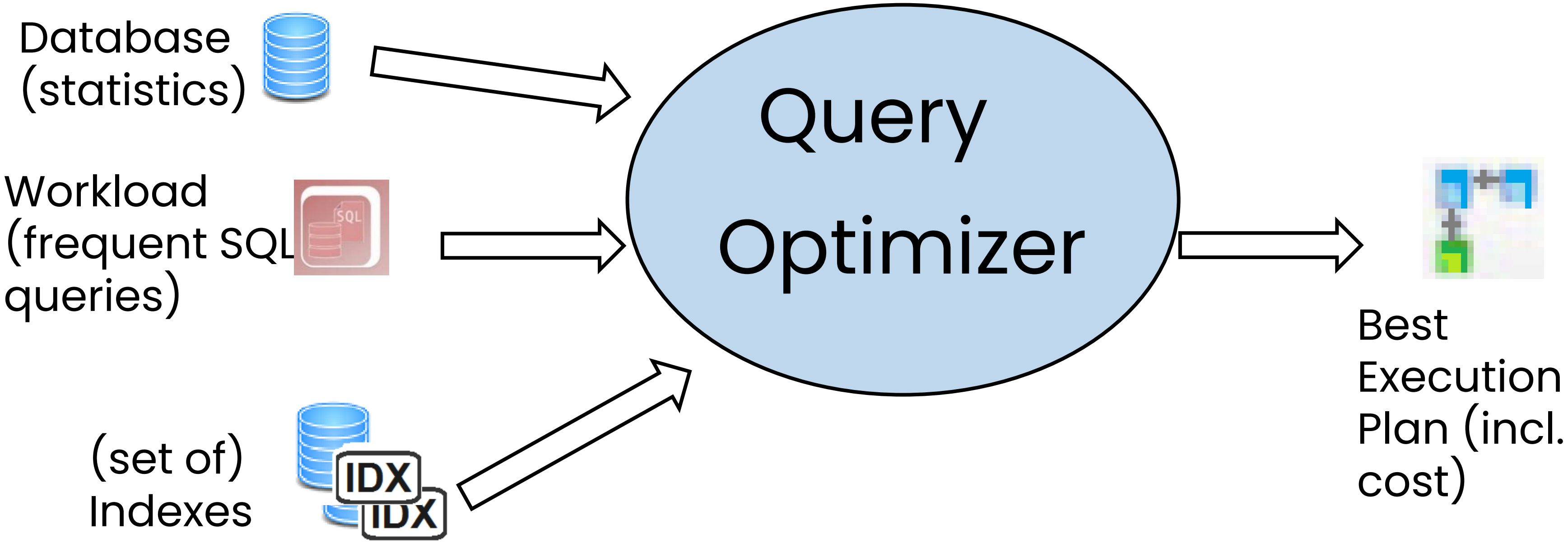
5. Transform indexes (from clustered to non-clustered/columnar) to better suite the workload

6. For a massive import operation, disable the impacted indexes before running the ingest process and rebuild them afterwards

2. Index design

2. Query optimizer

- Component of DBMS
- It’s used by Physical Design Adviser

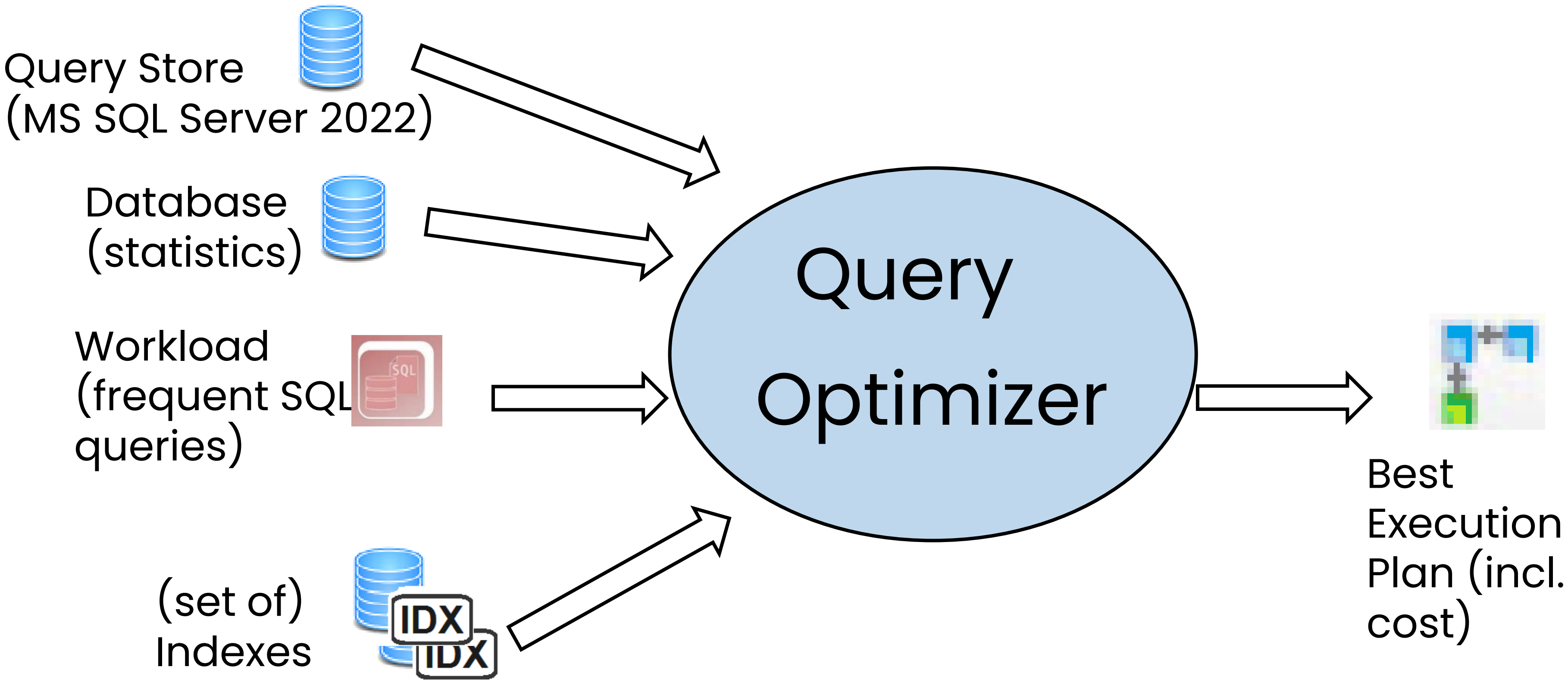


Database (statistics)
Workload (frequent SQL queries)
(set of) Indexes
IDX
IDX
Query Optimizer
Best Execution Plan (incl. cost)

2. Index design

2. Query optimizer

- Component of DBMS
- It’s used by Physical Design Adviser



Query Store (MS SQL Server 2022)

Database (statistics)

Workload (frequent SQL queries)

(set of) Indexes

Query Optimizer

Best Execution Plan (incl. cost)

2. Index design

3. Statistics

- What are statistics?
    - Distribution of values within a column
    - Density, Cardinality

- Why are statistics important?
    - Execution plan calculation

- Update statistics

2. Index design

3. (Some) Types of indexes and when to use them

1. Clustered Index.
2. Non-Clustered Index.
3. Unique Index.
4. Filtered Index.
5. Columnstore Index.
6. Hash Index.

DEMO

2. Index design

3. (Some) Types of indexes and when to use them

In the last course you have started to look at a bigger table and looked at the execution time comparison for looking at a table with a Clustered index and without it. We will start from that and experiment with a few more indexes

```
-- Create one non-clustered index
CREATE NONCLUSTERED INDEX [NonClusteredIndexDemo]
ON [dbo].[BigTable]
(
[UnitPrice] ASC,
[ModifiedDate] ASC
)
INCLUDE ([OrderQty])
GO
```

```
-- Create one column store index
CREATE NONCLUSTERED COLUMNSTORE INDEX
[NonClusteredColumnStoreIndexDemo] ON [dbo].[BigTable]
(
[SalesOrderID],
[SalesOrderDetailID],
[CarrierTrackingNumber],
[OrderQty],
[ProductID],
[SpecialOfferID],
[UnitPrice],
[UnitPriceDiscount],
[LineTotal],
[ModifiedDate]
) WITH (DROP_EXISTING = OFF, COMPRESSION_DELAY = 0) ON
[PRIMARY]
GO
```

2. Index design

```sql
-- Lookup for a record SalesOrderDetailID
-- Now, this is using 'by default' the
[CLI_SalesOrderDetailID] index
SELECT * FROM BigTable WHERE SalesOrderDetailID=120

-- Slow: Table Scan (using the clustered index)
SELECT OrderQty FROM [dbo].[BigTable]
WITH (INDEX(CLI_SalesOrderDetailID))
WHERE UnitPrice = 63.90

-- Fast: Index Seek (using a non-clustered index)
SELECT OrderQty FROM [dbo].[BigTable]
WITH (INDEX(NonClusteredIndexDemo))
WHERE UnitPrice = 63.90

-- Slow: Table Scan (using the clustered index) and also
looking for ProductId
SELECT OrderQty,[ProductID] FROM [dbo].[BigTable]
WITH (INDEX(CLI_SalesOrderDetailID))
WHERE UnitPrice = 63.90

-- Fast: Index Seek (using a non-clustered index) and also
looking for ProductId
SELECT OrderQty,[ProductID] FROM [dbo].[BigTable]
WITH (INDEX(NonClusteredIndexDemo))
WHERE UnitPrice = 63.90
```

```sql
-- Aggregating a column using a column-store index vs.
clustered index
SELECT AVG(UnitPrice) FROM [dbo].[BigTable]
WITH (INDEX(CLI_SalesOrderDetailID))

SELECT AVG(UnitPrice) FROM [dbo].[BigTable]
WITH (INDEX(NonClusteredColumnStoreIndexDemo))
```

2. Index design

4. Maintaining our indexes using Dynamic Management Views

As time passes and databases get bigger, it gets harder to keep track and maintain your indexes. For this we can use the Dynamic Management Views.

As indexes can bring also downsides, we need to be careful to what indexes we have in our database. Also we need to be mindful of possible missing indexes as well.

For that we can use MS SQL Server’s Dynamic Management Views

- Use system tables stored in database’s system catalogue (sys.* schema)
- May require elevated user clearance (permissions) to access them
- May change from one release to another

2. Index design

4. Maintaining our indexes using Dynamic Management Views

Missing indexes

```
 select
      'Missing indices' as Output_Type
      , db.name as database_name
      , m.name as schema_name
      , o.name as object_name
      , [total_cost_savings]  =
               round(s.avg_total_user_cost * s.avg_user_impact * (s.user_seeks + s.user_scans),0) /100
      , s.avg_total_user_cost
      , s.avg_user_impact
      , s.user_seeks
      , s.user_scans
      , unique_compiles
      , last_user_seek
      , last_user_scan
      --, last_system_seek
      --, last_system_scan
      , d.equality_columns
      , d.inequality_columns
      , d.included_columns
from  sys.dm_db_missing_index_groups g
      inner join  sys.dm_db_missing_index_group_stats s on s.group_handle = g.index_group_handle
      inner join  sys.dm_db_missing_index_details     d on d.index_handle = g.index_handle
      inner join  sys.objects                         o on o.object_id    = d.object_id
      inner join  sys.schemas                         m on m.schema_id    = o.schema_id
      inner join  sys.databases                      db on db.database_id = d.database_id
order by total_cost_savings desc
```

2. Index design

4. Maintaining our indexes using Dynamic Management Views

Index low usage

```
SELECT sc.name as schema_name
       , o.name as object_name
       , s.object_id
       , indexname=i.name
       , i.index_id
       , user_seeks
       , user_scans
       , user_lookups
       , user_updates
       , user_seeks + user_scans + user_lookups as total_reads
FROM sys.dm_db_index_usage_stats s
   JOIN sys.indexes  i ON i.object_id = s.object_id  AND i.index_id = s.index_id
   join sys.objects  o on o.object_id = i.object_id
   join sys.schemas sc on sc.schema_id = o.schema_id
WHERE o.type = 'U' -- user table
   and user_seeks + user_scans + user_lookups < 20
ORDER BY (user_seeks + user_scans + user_lookups) ASC
```

2. (Bonus) Interesting underperforming queries

Field from SELECT vs JOIN

Let’s compare and discuss on the following execution plans from the following queries

```
SELECT e.ENAME
, d.DNAME
FROM EMP e
JOIN DEPT d ON d.DEPTNO = e.DEPTNO

SELECT e.ENAME
, (SELECT d.DNAME FROM DEPT d WHERE d.DEPTNO
= e.DEPTNO)
FROM EMP e
```

2. (Bonus) Interesting underperforming queries

Window Functions

Let’s compare and discuss on the following execution plans from the following queries

```
SELECT *
FROM EMP

SELECT ROW_NUMBER() OVER(ORDER BY SAL DESC) AS RowNumber
       ,*
FROM EMP
```

2. (Bonus) Interesting underperforming queries

Finding top 10 resource consuming queries

```
SELECT TOP 10
    execution_count,
    statement_start_offset AS stmt_start_offset,
    total_logical_reads / execution_count AS avg_logical_reads,
    total_logical_writes / execution_count AS avg_logical_writes,
    total_physical_reads / execution_count AS avg_physical_reads,
    total_elapsed_time  / (execution_count * 1000)  AS avg_duration_ms,
    total_worker_time / (execution_count * 1000) AS avg_CPU_ms,
    total_rows / execution_count AS avg_rows_retuned,
    t.TEXT ,
    qp.query_plan
FROM
    sys.dm_exec_query_stats AS s
    CROSS APPLY sys.dm_exec_sql_text(s.sql_handle) AS t
    CROSS APPLY sys.dm_exec_query_plan(s.plan_handle) AS qp
ORDER BY
    avg_duration_ms DESC
```

Summary

- Indexes are the primary mechanism to improve the performance
- Implemented as hash tables or search trees
- Run a cost/benefit analysis to decide what indexes are needed
- Query planning and optimization – an important activity in relational database design
- Query Optimizer