ADVANCED DATABASE

Performance tuning

Dr. NGUYEN Hoang Ha

Email: nguyen-hoang.ha@usth.edu.vn

“THE RIGHT THING AT A WRONG TIME IS A WRONG THING”

Joshua Harris (2012).

“I Kissed Dating Goodbye: A New Attitude Toward Relationships and Romance”

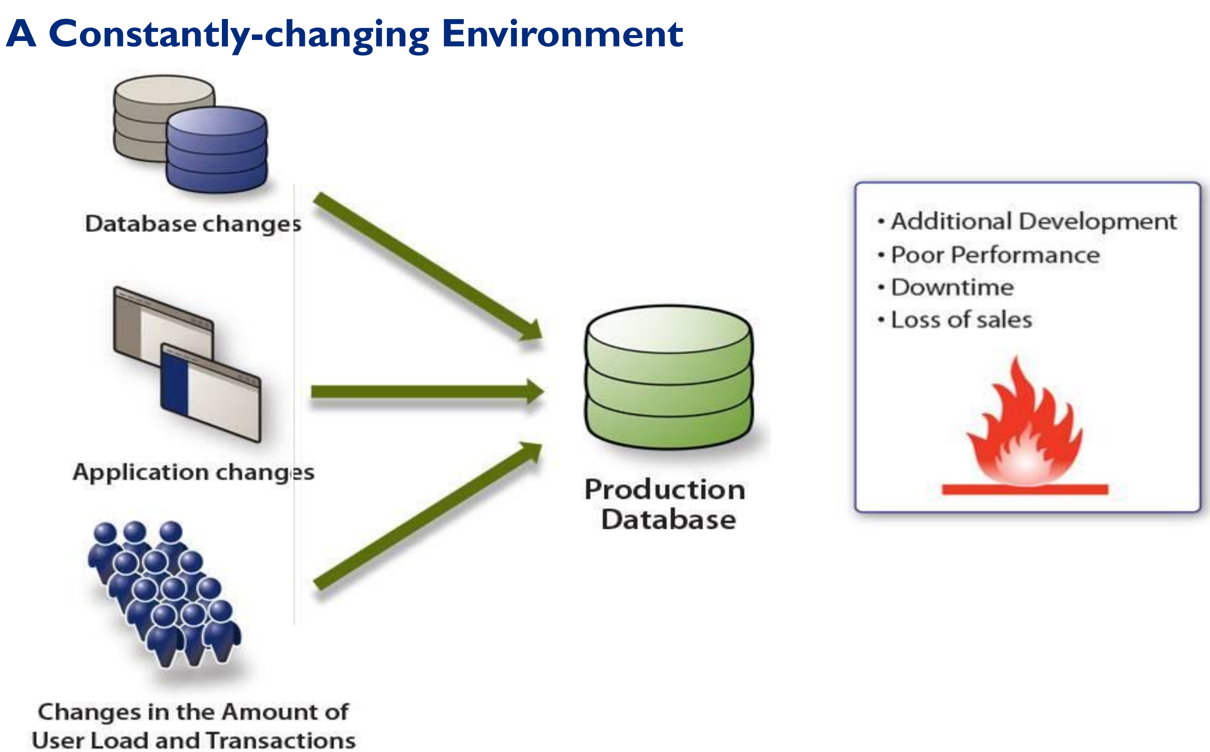
Agenda

I. Overview
II. Diagnostic
III. System and Hardware
IV. Design strategies
V. Index use
VI. Programming techniques

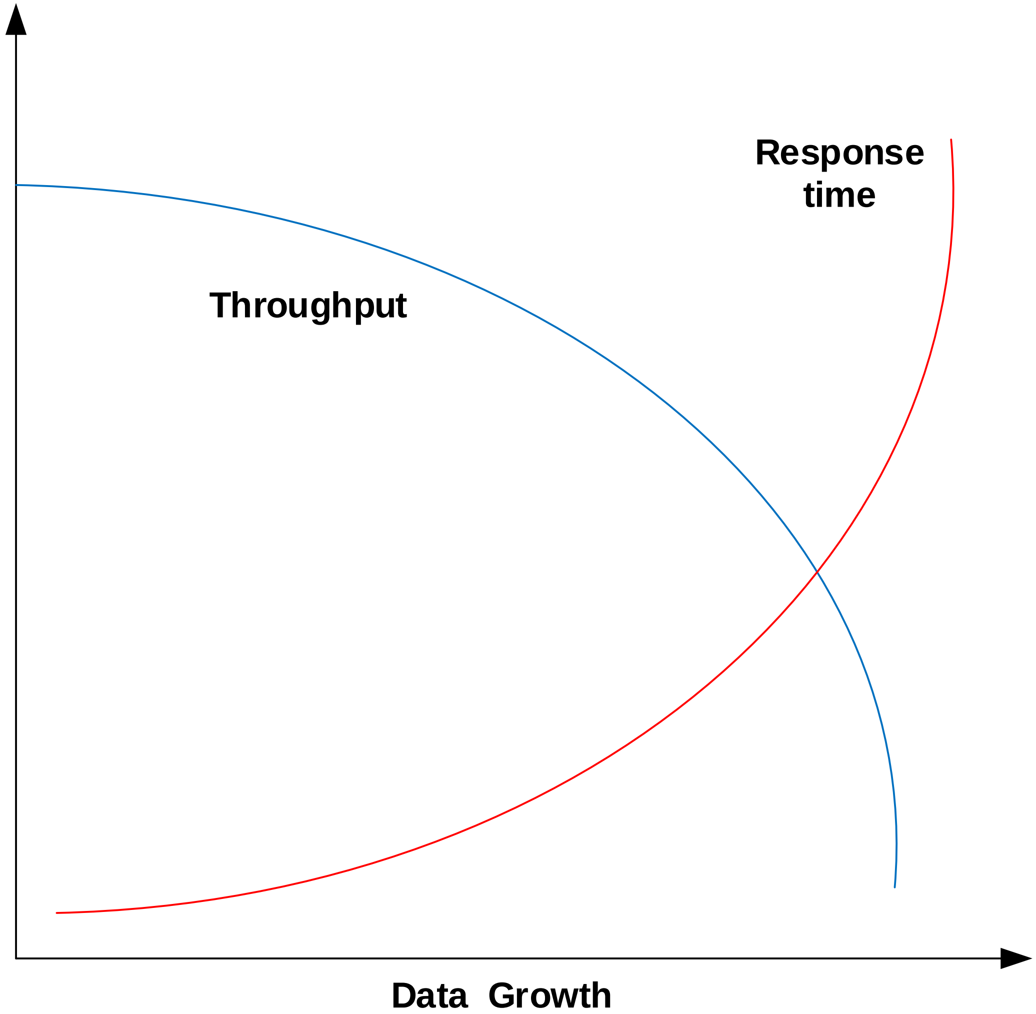
1. OVERVIEW

Needs vs Problems

- Needs
  - More business → Higher speed
  - More concurrent users → Less resource consumption per user
- Problems



Problems



Goals

- The all-encompassing goal of the computer industry is speed

→ Make a set of DB applications execute faster

- Definition: DB performance tuning is a set activities and procedure to optimize:
  - Response time
  - Throughput
- What we should do?
  - Make queries run faster
  - Make updates run faster
  - Minimize congestion due to concurrency
- Facts:
  - Functional SQL → not difficult
  - Write efficient, high performance SQLs → harder

What affects performance

- SQL statements
- Indexes
- DB design
- Server settings
- OS
- Hardware

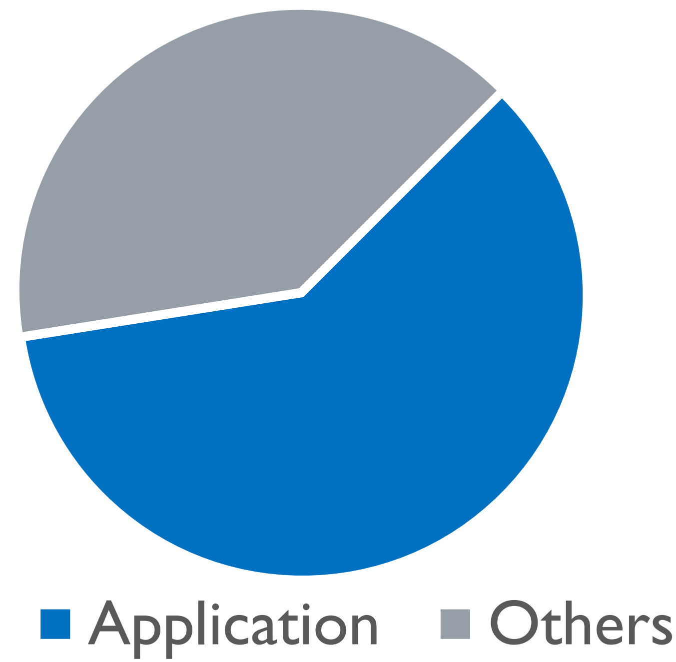
Application programmers

Business analysist, data architect, …

DBA, tuner

Possible causes

- Weak hardware
- Lack of proper and meaningful maintenance
- Poor monitoring and scheduling etc..
- Bad server settings
- Applications
  - Poor design
  - Bad SQL statements
    - By developers, users



Rules of thumb

- Optimize the DB before upgrading the hardware
- Try to have **good DB design** and **well written code**
- Focus the optimization effort on the **most frequently run code**, rather than the slowest code
- Focus on fixing the worst performing aspect of the application first
- Keep a **list of possible optimization ideas**, even if you do not have time to implement them now
- **Spend time** for using the application as a user

10

Tuning strategy

- Keep it Simple
- Small changes with low impact but with high performance benefits
- Localized changes
- No change in logic
- Easy to understand, test and deploy

Fallacies

- Too busy now. I'll do it later.
- I'm a Java or C# not an SQL, programmer.
- Is there optimizer tool is for?
- I don't know how.
- Let tool generate SQL → hard to control.
- It works. I've got my data. I'm happy.

Tuning modes

Proactive

- Is planned
- Low time pressure
- No scope
- Sometimes no target

Reactive

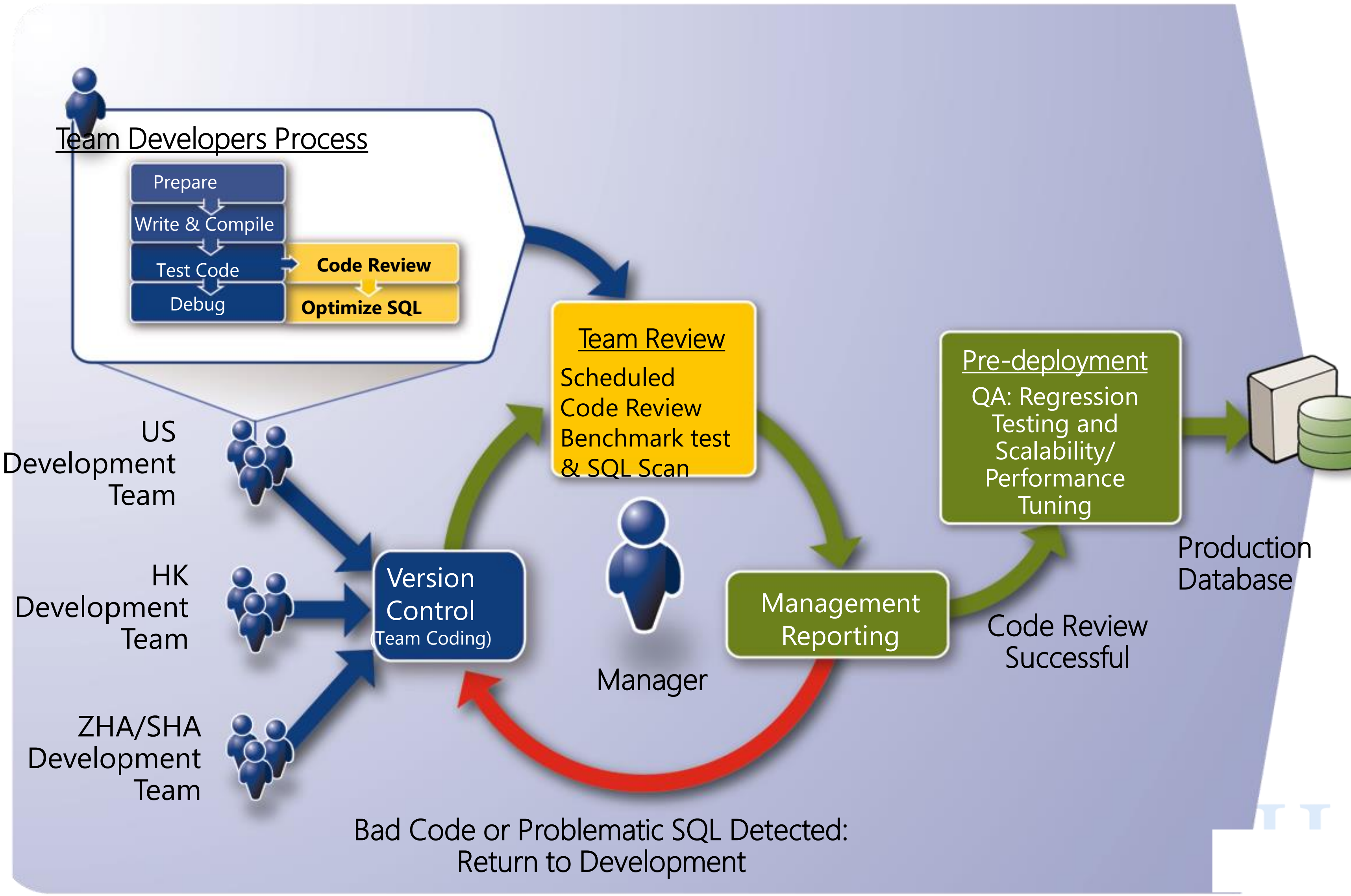
- Cannot be planned
- High time pressure
- Scope limited to specific problems
- Clear target

- When to tune?
  - At the time it is written
  - As the database changes

Considered factors

- Budgets
- Time frame
- Functional requirements
- Required performance
- Critical nature of the system to the core business
- Risks
  - Acceptable
  - Unacceptable

DB development good practice



Tuning process

Diagnosis
- Monitor
- Benchmark

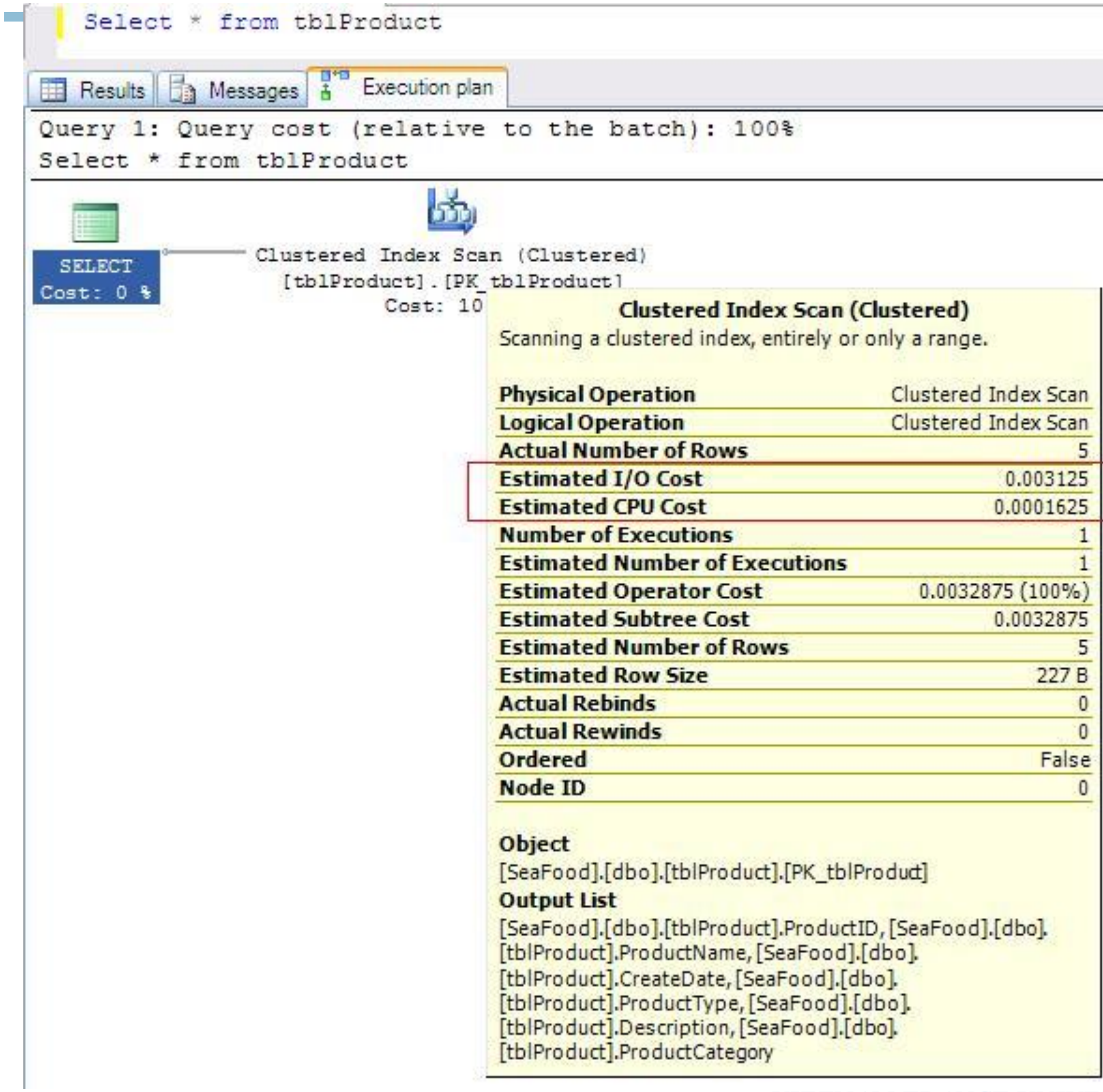
Prescribe

Change

Test
- Re-benchmark

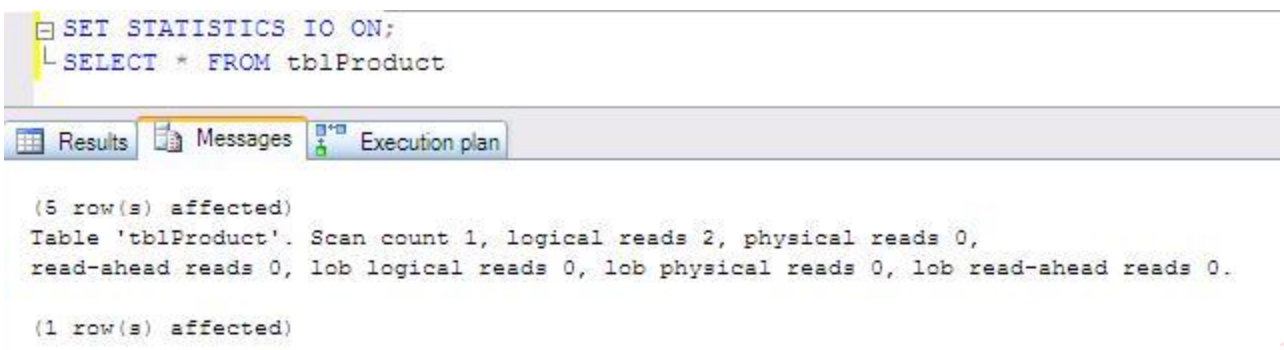
II. DIAGNOSTIC

Metric 1: Query cost



Metric 2: Page reads

- Number of read pages
- SQL server: Page size = 8KB
- To see: put SET STATISTIC IO ON before the query



Metric 3: Query Execution Time

- How long a statement executes
- To see: put SET STATISTICS TIME ON before the statement

```
SET STATISTICS TIME ON;
SELECT * FROM tblProduct
```

Results | Messages | Execution plan

```
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.
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.

(5 row(s) affected)

(1 row(s) affected)

 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.

 SQL Server Execution Times:
   CPU time = 0 ms,  elapsed time = 0 ms.
```

Dynamic Management Views

- SQL Server counters

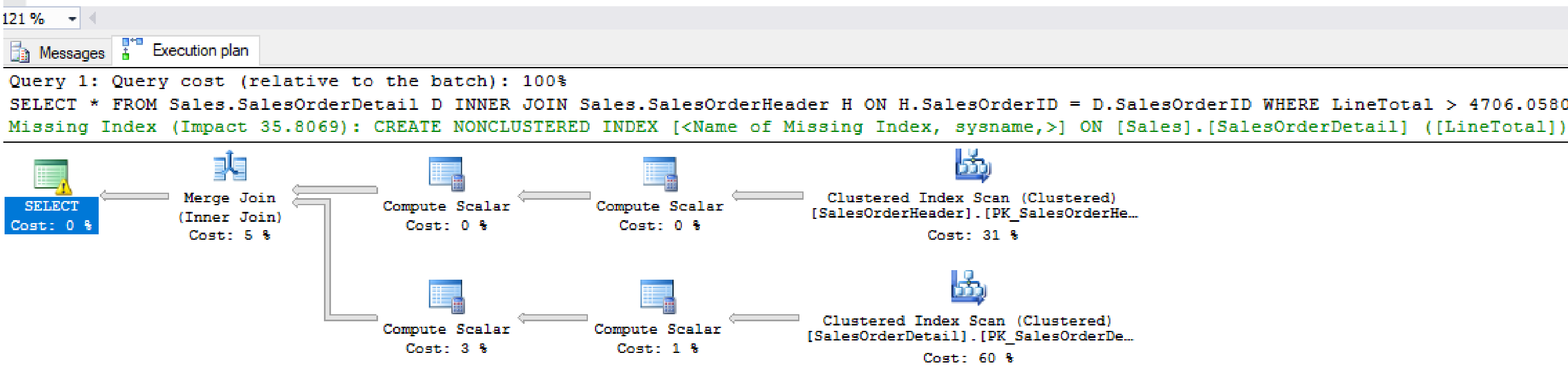
```
SELECT *
FROM sys.dm_os_performance_counters
```

- Sessions
  - sys.dm_exec_sessions
- Connection
  - sys.dm_exec_connections
- Mission indices
  - Sys.dm_db_missing_index_group_stats
  - Sys.dm_db_missing_index_groups
  - Sys.dm_db_missing_index_details
  - Sys.dm_db_missing_index_columns

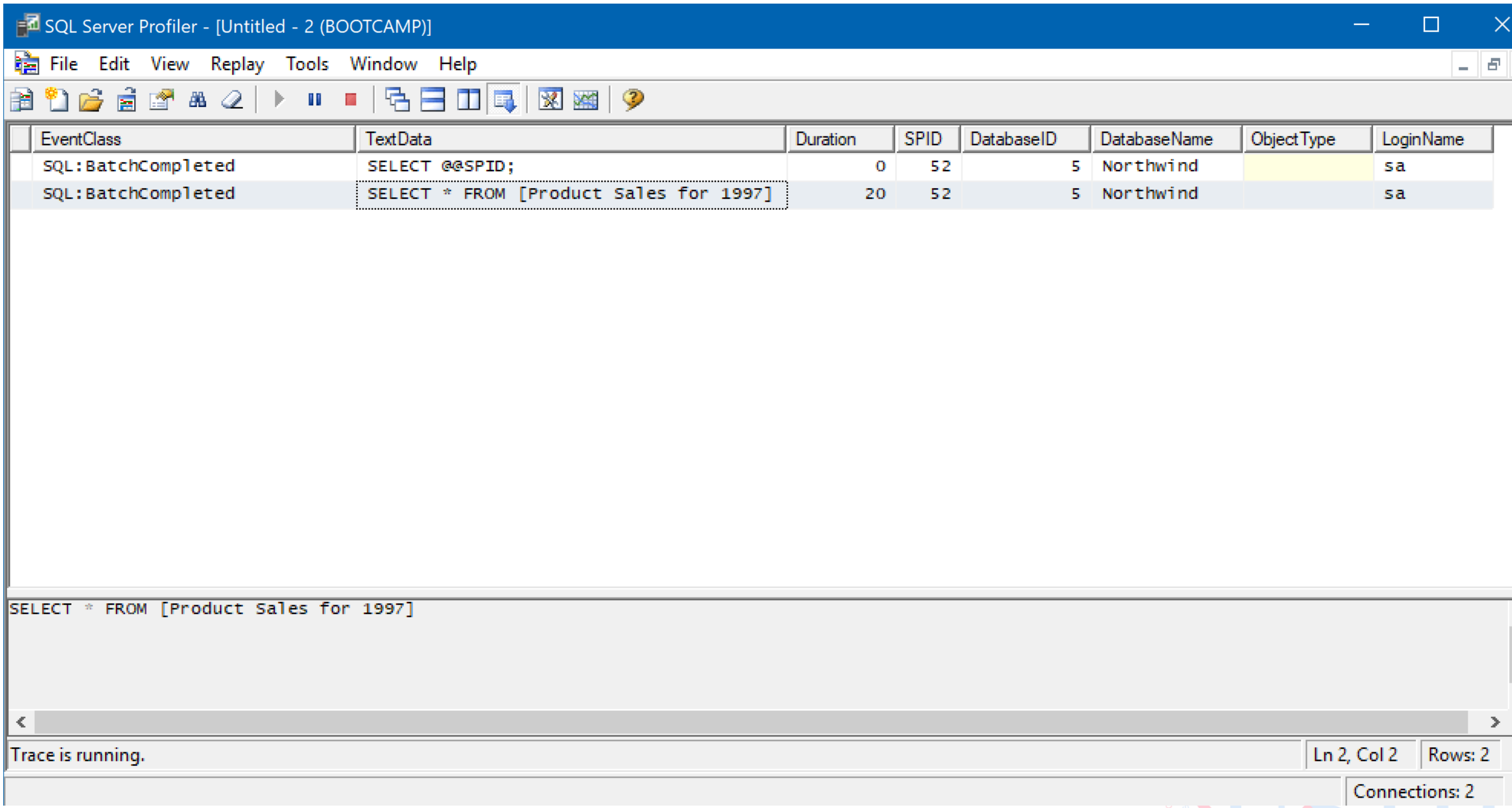
Analyze query plan

- Index seek vs. Index Scan

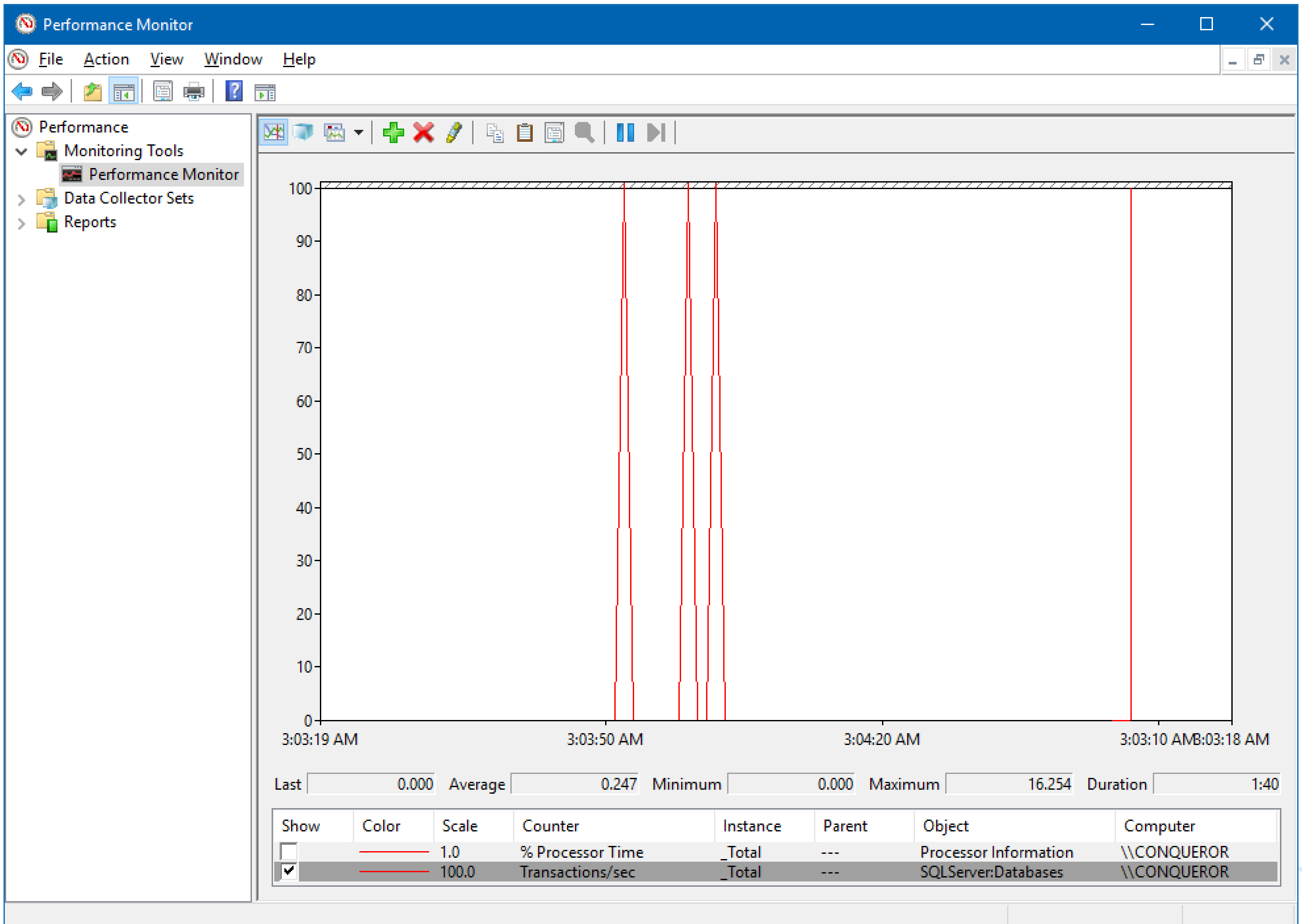
```sql
SELECT *
FROM Sales.SalesOrderHeader H INNER JOIN Sales.SalesOrderDetail D ON H.SalesOrderID = D.SalesOrderID
WHERE LineTotal > 4706.058000
```



Monitor tools: SQL Server Profiler

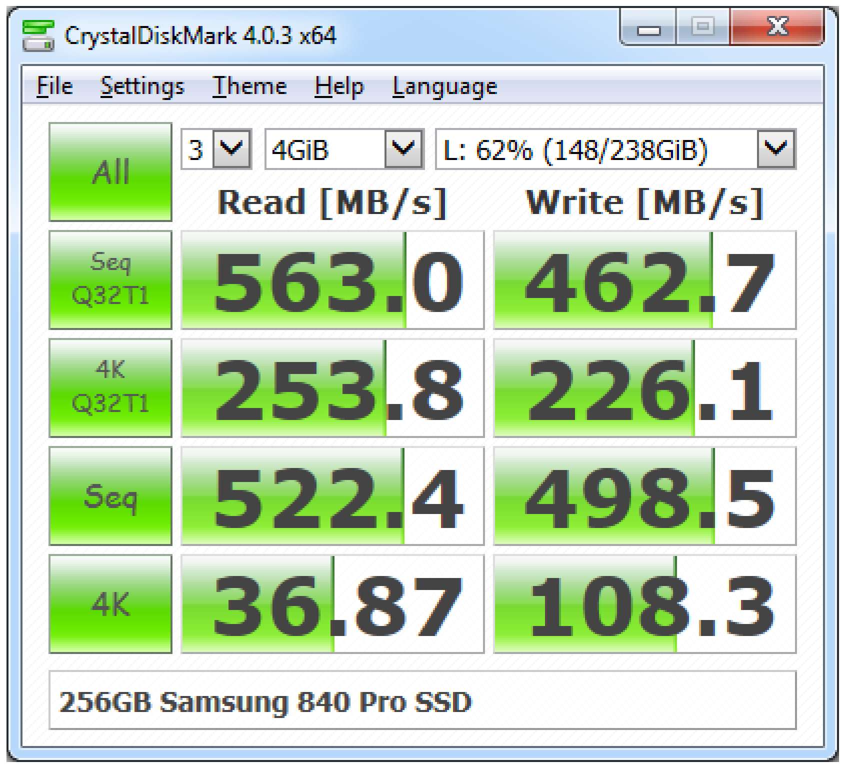


Monitor tools: Performance Monitor



I/O performance of system

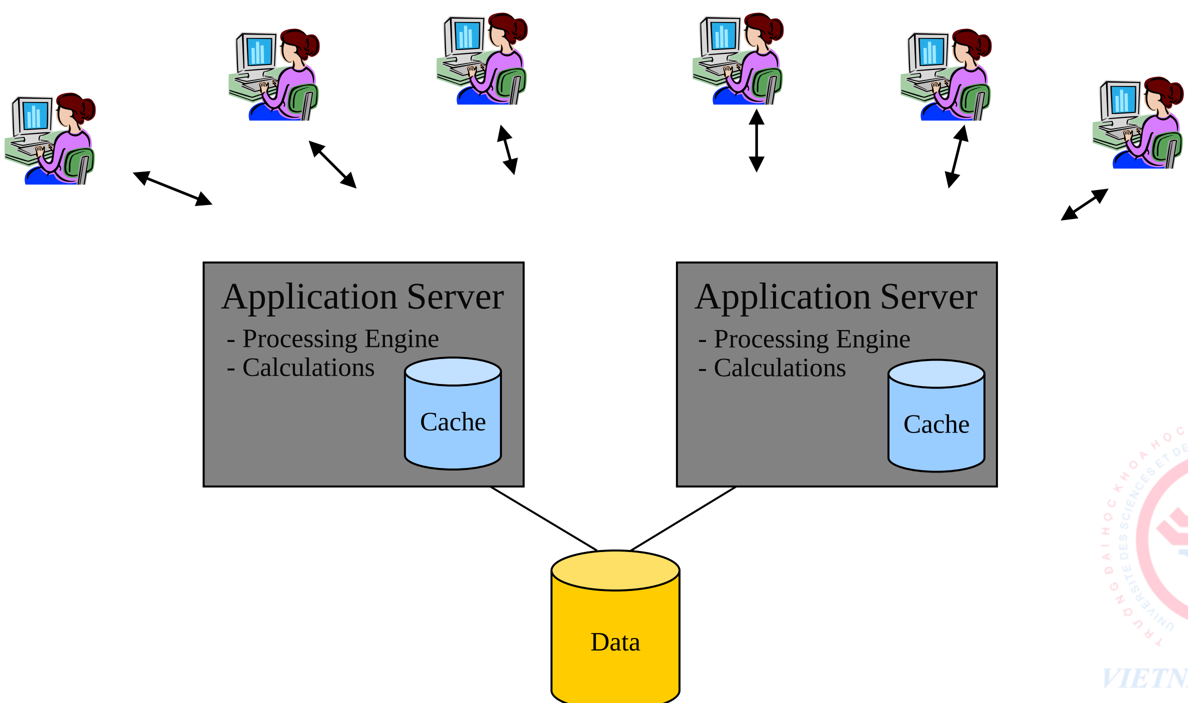
- CrystalDiskMark



III. SYSTEM AND HARDWARE

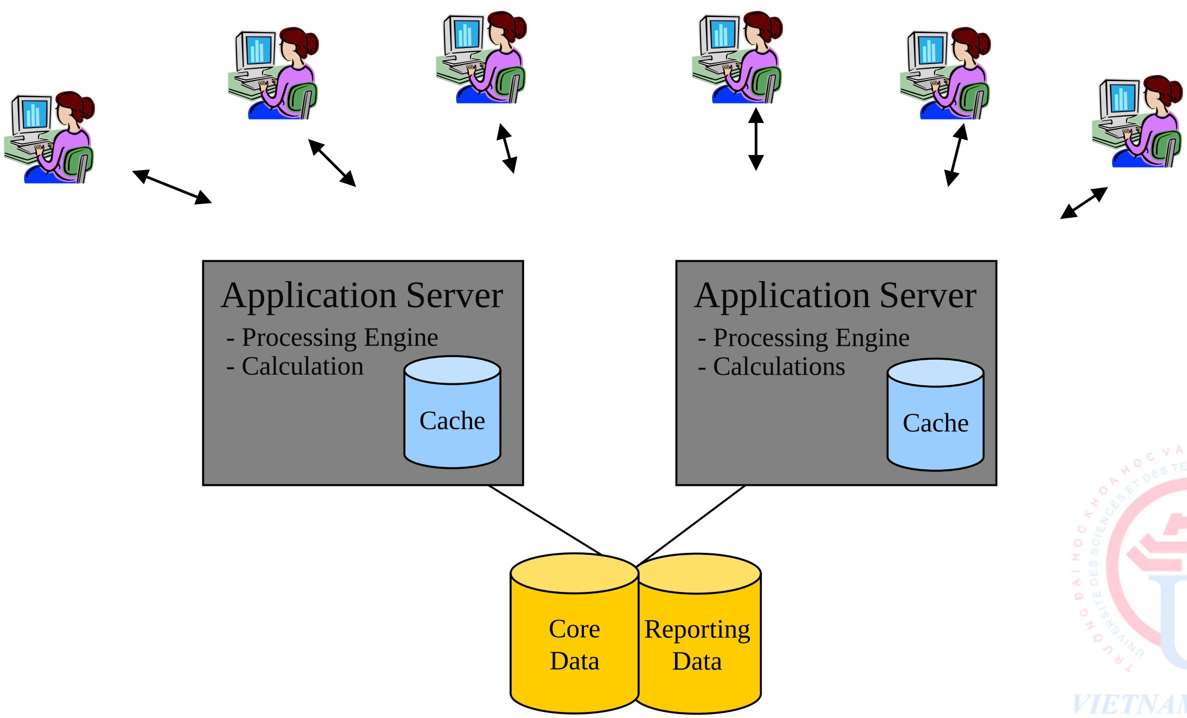
Load Balanced Asynchronous Processing

- Scale up by distribution the system
  - When real time analysis of large volumes of data is required, move the calculations into a middle-tier
- Allow several servers to run the middle tier objects and federate the data to be processed



Mirrored Data

- Use techniques for mirroring data between n servers to separate analysis transactions from OLTP transactions
- Techniques can include using replication and double commit of transactions

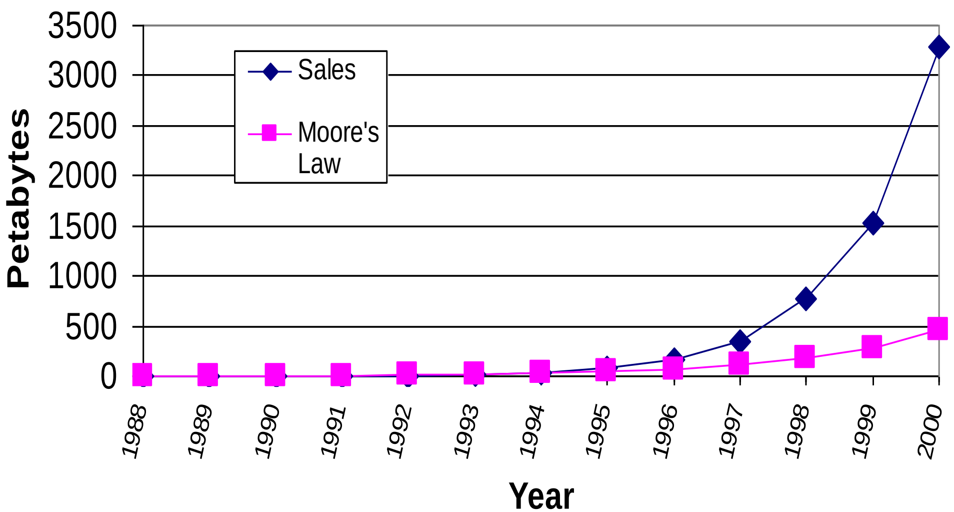


Consequences of “Moore’s law” on Hardware

- Over the last decade:
  - 10x better access time
  - 10x more bandwidth
  - 100x more capacity
  - 4000x lower media price
  - Scan takes 10x longer (3 min vs 45 min)
  - Data on disk is accessed 25x less often (on average)
- → Consider upgrading RAM, Storage, CPU, Network

Data Flood

- Disk Sales double every nine months
  - Because volume of stored data increases



uch faster than areal density.

Graph courtesy of Joe Hellerstein
Source: J. Porter, Disk/Trend, Inc.
http://www.disktrend.com/pdf/portrpkg.pdf

Magnetic Disks

- Access Time (2001)
  - Controller overhead (0.2 ms)
  - Seek Time (4 to 9 ms)
  - Rotational Delay (2 to 6 ms)
  - Read/Write Time (10 to 500 KB/ms)
- Disk Interface
  - IDE (16 bits, Ultra DMA - 25 MHz)
  - SCSI: width (narrow 8 bits vs. wide 16 bits) - frequency (Ultra3 - 80 MHz).

http://www.pcguide.com/ref/hdd/

RAID Levels

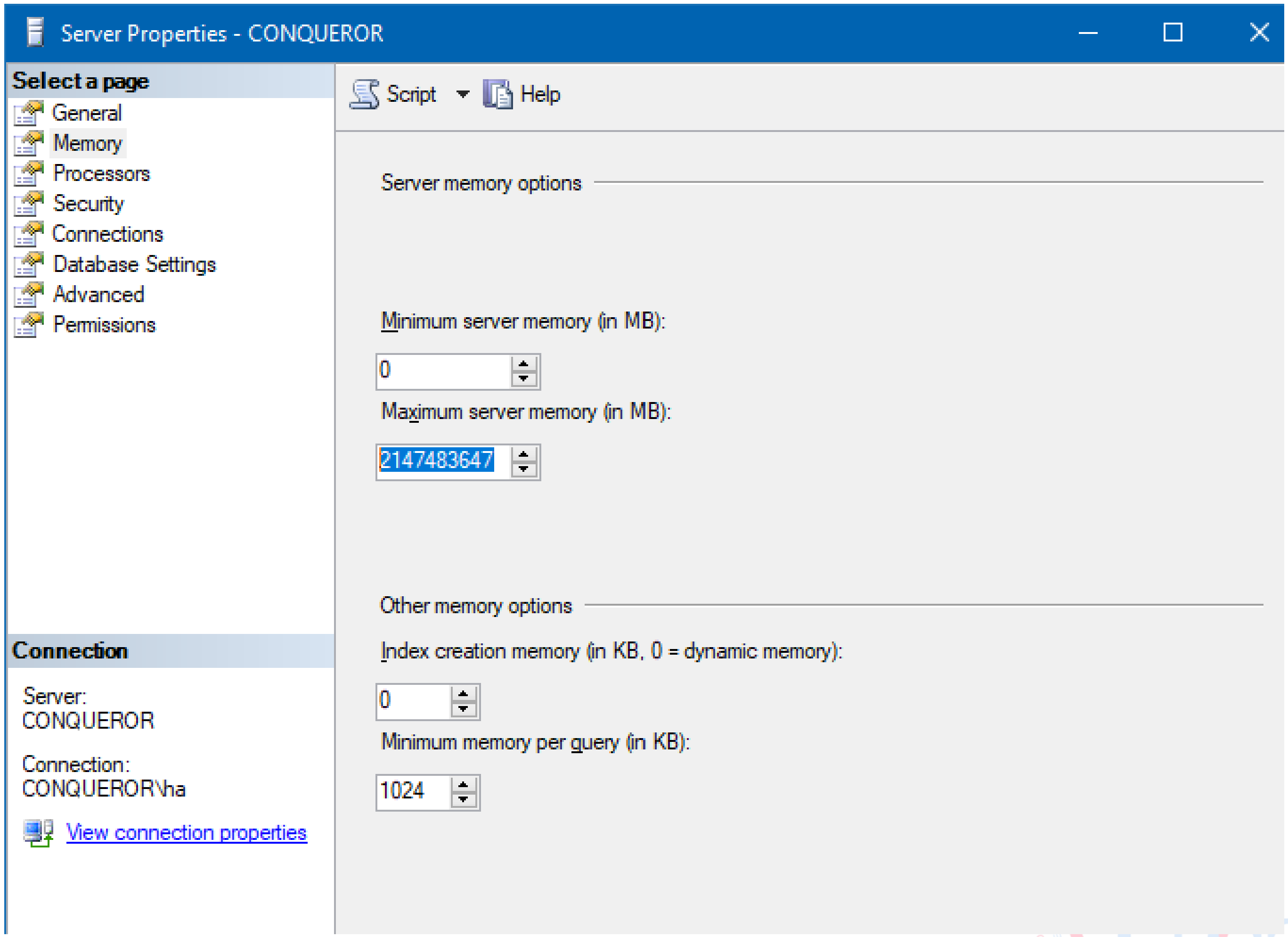
- RAID 0: striping (no redundancy)
- RAID 1: mirroring (2 disks)
- RAID 5: parity checking
  - Read: stripes read from multiple disks (in parallel)
  - Write: 2 reads + 2 writes
- RAID 10: striping and mirroring
- Software vs. Hardware RAID:
  - Software RAID: run on the server's CPU
  - Hardware RAID: run on the RAID controller's CPU

32

OS and software

- 64-bit OS are suggested
- Keep updating OS and DBMS
- Proper level of firewall
- Antivirus

Server memory options



IV. DESIGN STRATEGY

DB design affects performance

- The foundation of an application is the database design. It affects the type of queries
- Databases that are not properly normalized require additional code to maintain data integrity.
- Databases that use composite primary keys require multiple join condition.
- Database without comprehensive constraints require extra codes to validate the data during data inputting

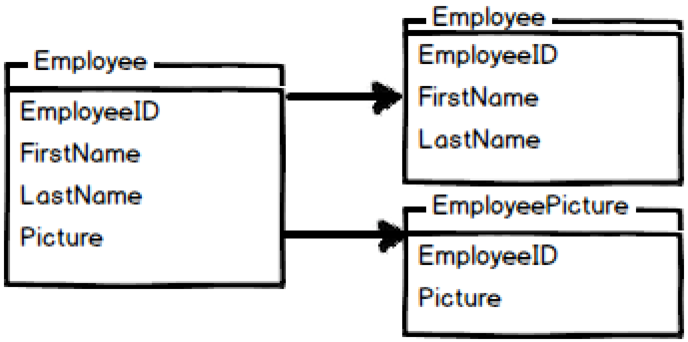
Guidelines

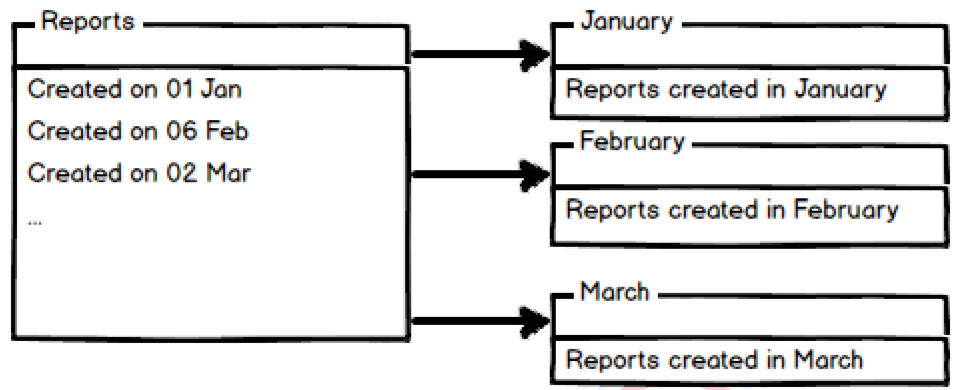
- Normalize the database to 3NF
- Don't over-normalize or over-complicate the database. Keep working until a simple and elegant design is found
- Avoid database designs that move data from table to table in a transactional manner
- Use a data-driven database design rather than designs with any hard-coded values
- Avoid temporary tables
- Design the DB schema with queries in mind.
- When necessary, do duplicate data from Normalized tables to DeNormalized read-only tables for faster reading

Denormalization example

- Normalized
  - Students (StudentID, FirstName, LastName….)
  - Subjects (SubjectID, SubjectName). Suppose cardinality is 3
  - Grades (StudentID, SubjectID, grade)
  - → Join queries are expensive
- Denormalize Grades into 0NF
  - Students (StudentID, FirstName, LastName….)
  - Subjects (SubjectID, SubjectName)
  - Grades (StudentID, Subject1, Subject2, Subject3)

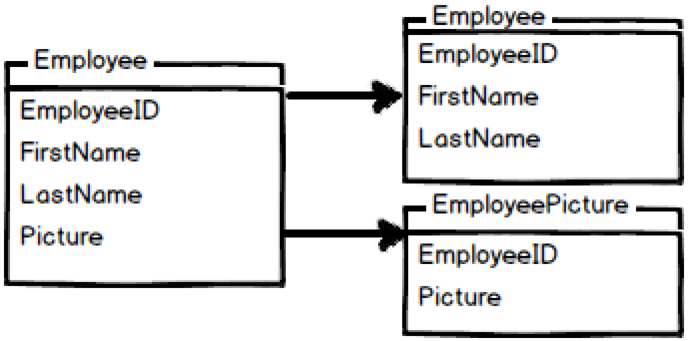
Horizontal and Vertical partitioning





Recomposing with VIEW

```
CREATE VIEW Emp AS
SELECT E.*, P.Picture
FROM Employee E INNER JOIN EmployeePicture P
        ON E.EmployeeID = P.EmployeeID
```



Recomposing with VIEW

CREATE VIEW Reports AS

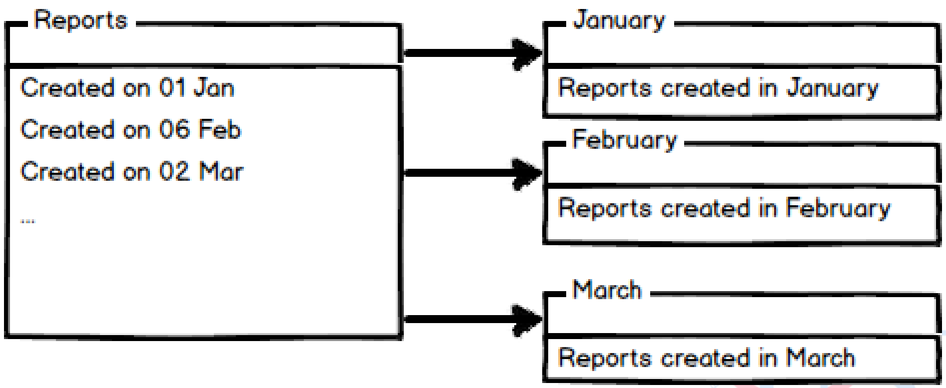
SELECT * FROM January

UNION

SELECT * FROM February

UNION

SELECT * FROM March



Precomputed columns

- System takes time to compute aggregated, inferred values
- Eg: Purchasing.PurchaseOrderDetail of Adventure Works

	PurchaseOrderDetailID	OrderQty	UnitPrice	LineTotal
1	1	4	50.26	201.04
2	2	3	45.12	135.36
3	3	3	45.5805	136.7415
4	4	550	16.086	8847.30
5	5	3	57.0255	171.0765
6	6	550	37.086	20397.30
7	7	550	26.5965	14628.075
8	8	550	27.0585	14882.175
9	9	550	33.579	18468.45
10	10	550	46.0635	25334.925
11	11	3	47.4705	142.4115
12	12	3	45.3705	136.1115

- Use trigger to update precomputed columns

V. INDEX USE

Index Implementations in some major DBMS

- SQL Server
  - B+Tree data structure
  - Clustered indexes are sparse
  - Indexes maintained as updates/insertions/deletes are performed
- DB2
  - B+Tree data structure, spatial extender for R-tree
  - Clustered indexes are dense
  - Explicit command for index reorganization
- Oracle
  - B+tree, hash, bitmap, spatial extender for R-Tree
  - clustered index
    - Index organized table (unique/clustered)
    - Clusters used when creating tables.
- TimesTen (In-memory DBMS)
  - T-tree

EXEC sp_helpindex [table name] to list all indexes

Clustered Indexes

- In a clustered index, the actual data rows that comprise the table are stored at the leaf level of the index
- The physical row order of the table and the order of rows in the index are the same

  → Each table can have only one clustered index

- PK columns are good candidates for clustered indexes

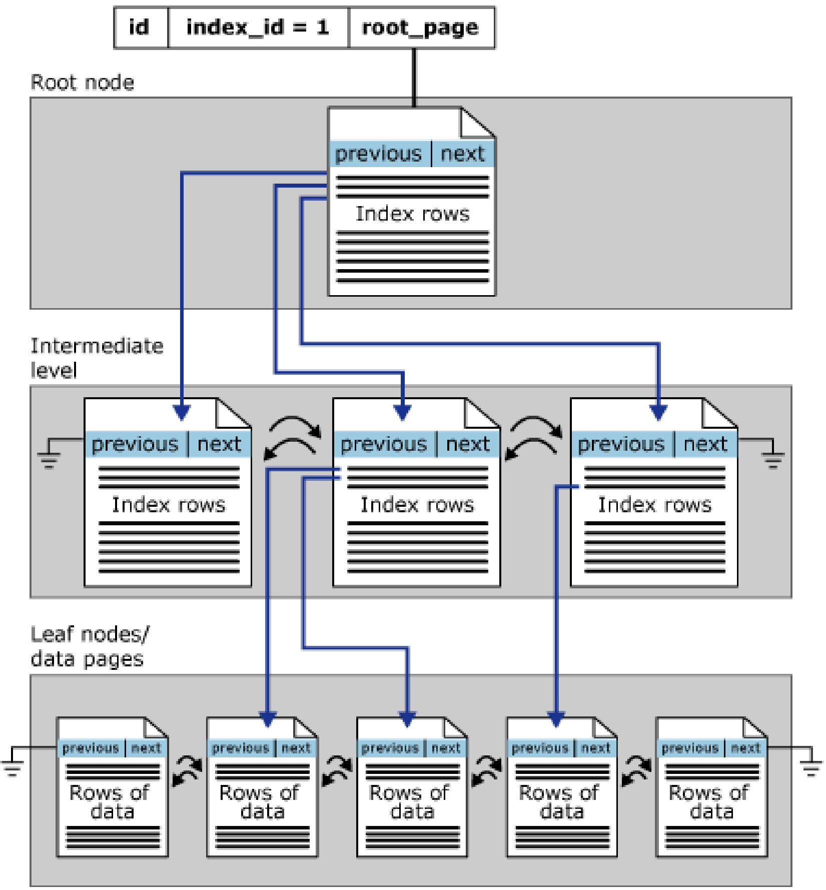
Non-clustered index

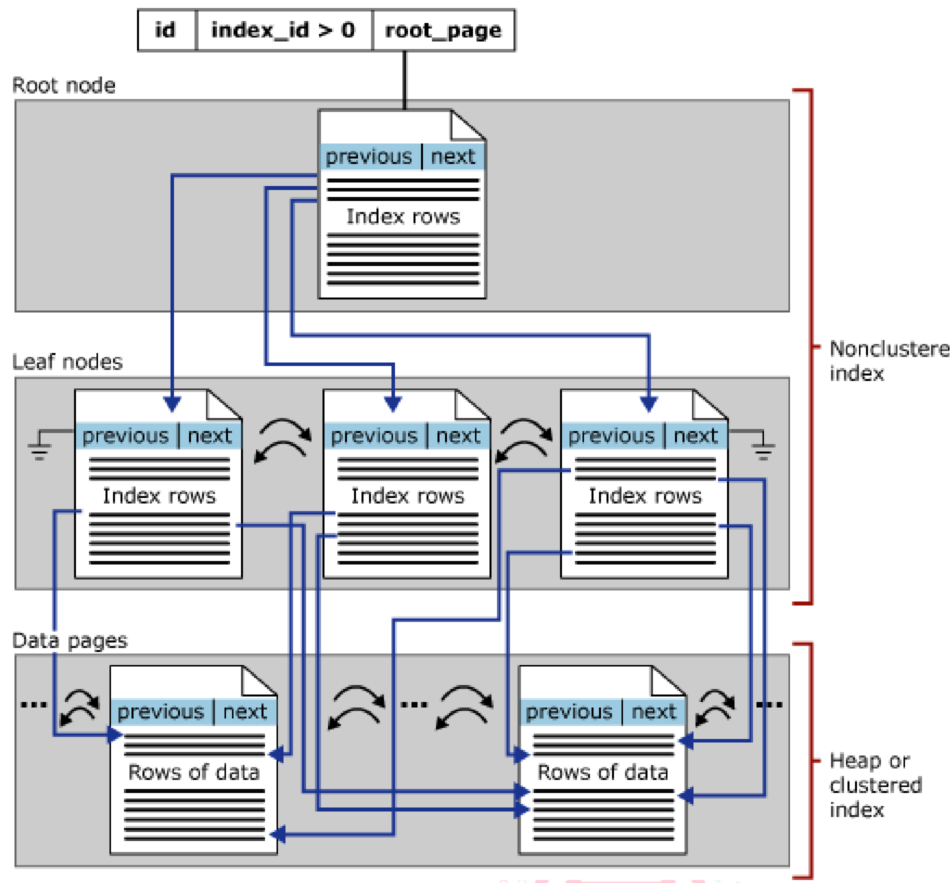
- The Non-Clustered index is an index structure separate from the data stored in a table

→ A table can have more than one non-clustered index

- Non-clustered indexes are slower than clustered indexes because the DMBS must follow a pointer to retrieve the actual data row.
- The leaf nodes of a non-clustered index can optionally contain values from non-indexed columns

Clustered vs. Nonclustered indexes





Using the `FILLFACTOR` Option

- Specifies how much to fill the page
- Impacts leaf-level pages

Data Pages Full

Con	...	470401
Funk	...	470402
White	...	470403
Rudd	...	470501
White	...	470502
Barr	...	470503

Akhtar	...	470601
Funk	...	470602
Smith	...	470603
Martin	...	470604
Smith	...	470701
Ota	...	470702

Martin	...	470801
Phua	...	470802
Jones	...	470803
Smith	...	470804
Ganio	...	470901
Jones	...	470902

Data Pages
50% Fillfactor

Con	...	470401
Funk	...	470402
White	...	470403

Rudd	...	470501
White	...	470502
Barr	...	470503

Akhtar	...	470601
Funk	...	470402
Smith	...	470603

Martin	...	470604
Smith	...	470701
Ota	...	470702

Martin	...	470801
Phua	...	470802
Jones	...	470803

Smith	...	470804
Ganio	...	470901
White	...	470902

Using the `PAD_INDEX` Option

- PAD_INDEX ON means applying FILLFACTOR to all NonLeaf Level of B-tree

- Must use with `FILLFACTOR` option

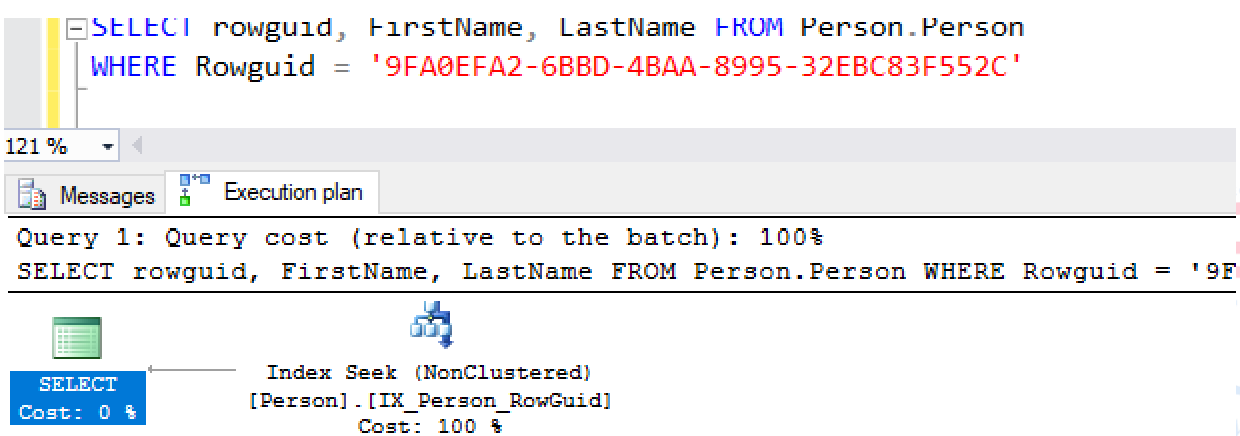
Index selectivity

- Create every primary key as a non-clustered index
- Create a clustered index for every table.
  - Primary tables: cluster the most common ORDER BY columns, don't cluster the primary key.
  - Secondary tables: create a clustered index for the most important foreign key
- Create non-clustered indexes for the columns of every foreign key
- Create single-column index for every column referenced in a WHERE clause or an ORDER BY clause
- If a table is heavily updated, index as few columns as possible
- If a table is updated rarely, use as many indexed columns as necessary to achieve maximum query performance

Covering index

- An non-clustered index which can satisfy all requested columns in a query without performing a further lookup into the clustered index. → save time
- Non-clustered index can include some other columns so that the query can fetch enough columns from the index
- Eg

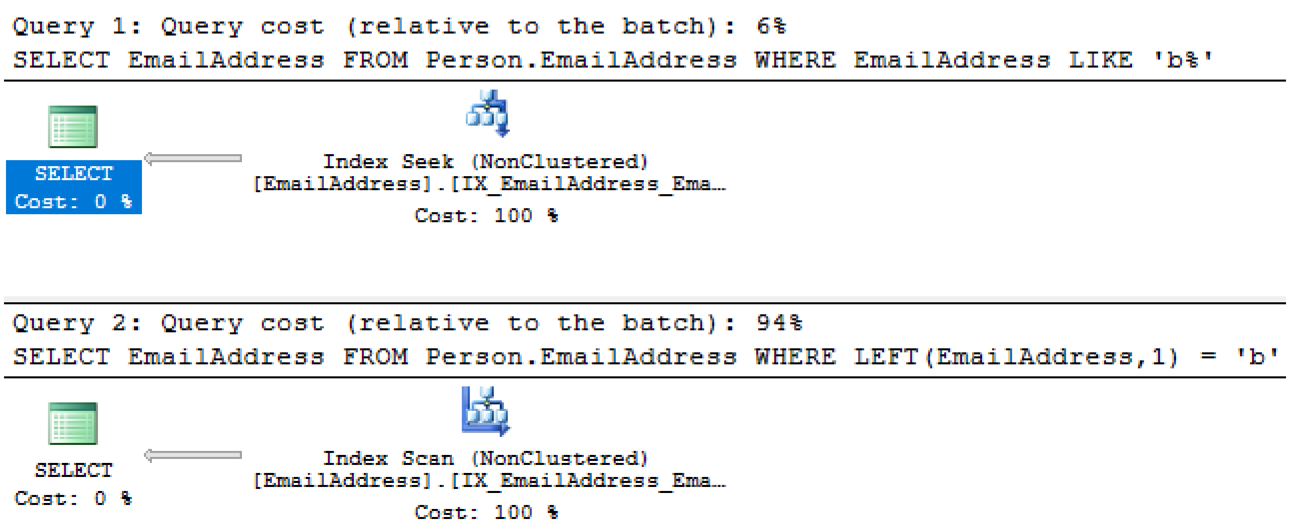
```
CREATE NONCLUSTERED INDEX IX_Person_RowGuid
ON Person.Person(rowguid) INCLUDE (FirstName,LastName)
```



Exploit index when available

```sql
Use AdventureWorks2014;
SELECT EmailAddress FROM Person.EmailAddress
WHERE EmailAddress LIKE 'b%'

SELECT EmailAddress FROM Person.EmailAddress
WHERE LEFT(EmailAddress,1) = 'b'
```



Data types cannot be indexed

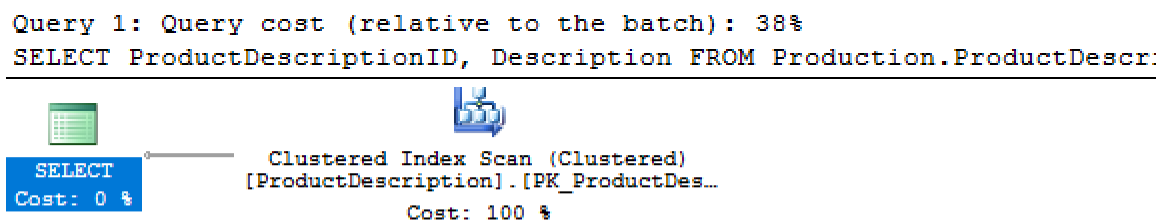
- Image
- Varbinary(max)
- Text
- Ntext
- Varchar (max)
- Nvarchar(max)

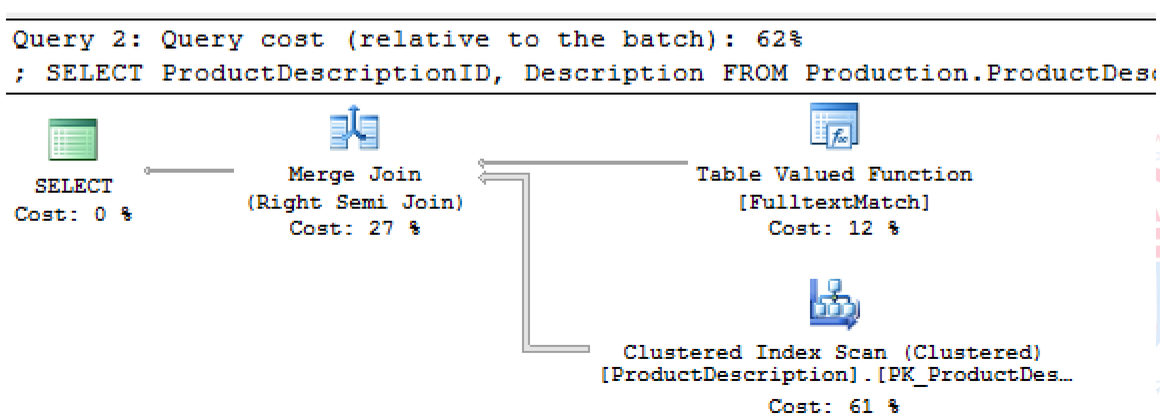
Fulltext search

Example

```sql
SELECT ProductDescriptionID, Description FROM Production.ProductDescription
WHERE [Description] LIKE  N'%bike%';

SELECT ProductDescriptionID, Description FROM Production.ProductDescription
WHERE FREETEXT(Description, N'bike');
```





What is REAL Fulltext search

- Allows searching for text/words in columns
  - Similar words
  - Plural of words
- Based on special index
  - Full-text index (Full text catalog)

FREETEXT vs. CONTAINS

Demo.sql - (local).Quitanda (ALEPH\Paulo (51))

```sql
SELECT * FROM Textos
WHERE CONTAINS(*, '
FORMSOF(INFLECTIONAL, Azul) OR FORMSOF(THESAURUS, Azul)')

SELECT * FROM Textos
WHERE FREETEXT(*, 'Azul')
```

100 %

Results | Messages

	Codigo	Nome	Texto
1	1	Azul	Modelo de texto
2	7	Azulado	Textos obrigatórios
3	8	Azuis	Textos obrigatórios

	Codigo	Nome	Texto
1	1	Azul	Modelo de texto
2	7	Azulado	Textos obrigatórios
3	8	Azuis	Textos obrigatórios

Query executed successfully. | (local) (10.50 RTM) | ALEPH\Paulo (51) | Quitanda | 00:00:00 | 6 rows

Full-text search Predicates

- CONTAINS
- CONTAINSTABLE
  - Return a table Key
  - Rank: 0 to 1000 shows how well the results match

```
DECLARE @SearchWord nvarchar(30)
SET @SearchWord = N'performance'
SELECT ProductDescriptionID, Description
FROM Production.ProductDescription
WHERE CONTAINS(Description, @SearchWord);

SELECT * FROM CONTAINSTABLE (Production.ProductDescription  , [Description], @SearchWord)
```

- FREETEXT: adds inflectional forms of a word to the search
- FREETEXTTABLE

Full-Text Search Terminologies

- Full-text index
  - Information about words and their location in columns
  - Used in full text queries
- Full-text catalog
  - Group of full text indexes (Container)
- Token
  - Word identified by word breaker
- Word breaker
  - Tokenizes text based on language

Full-Text Search Terminologies (cont')

- Stopwords/Stoplists
  - not relevant word to search
  - e.g. 'and', 'a', 'is' and 'the' in English
  - Some languages without stop list supported

LCID	Language Name
1042	Korean
1066	Vietnamese
3076	Chinese (Hong Kong SAR, PRC)
4100	Chinese (Singapore)
5124	Chinese (Macau SAR)

- Accent insensitivity
  - cafè = cafe

FREETEXT

- Fuzzy search (less precise ☺)
  - Inflectional forms (Stemming)
  - Related words (Thesaurus)

```
SELECT ProductDescriptionID, Description FROM Production.ProductDescription
WHERE [Description] LIKE  N'%bike%';
```

```
SELECT ProductDescriptionID, Description FROM Production.ProductDescription
WHERE FREETEXT(Description, N'bike');
```

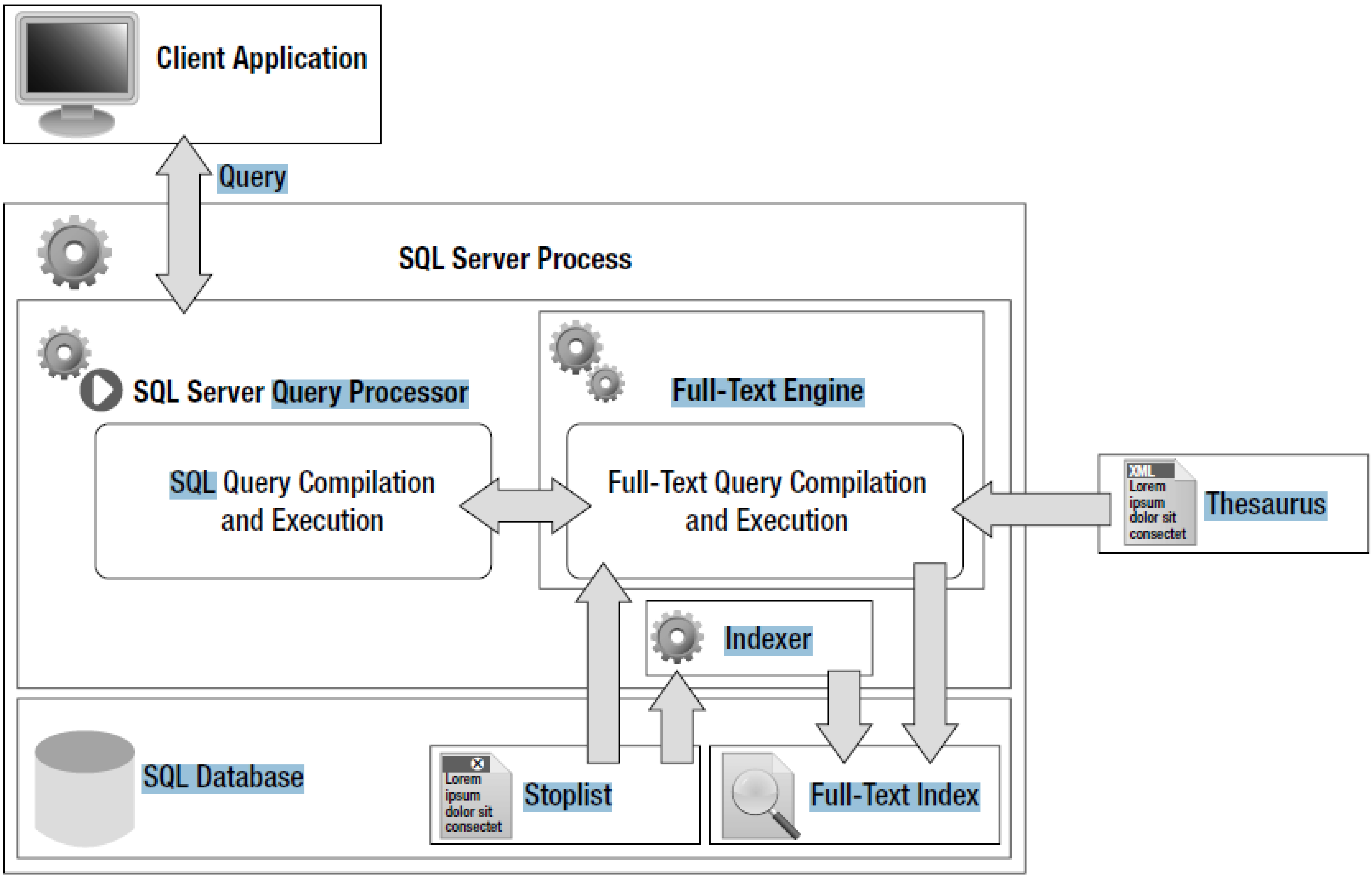
Fulltext search – Under the hood

Full-text index data

- *Now is the time for all good men to come to the aid of the party*

Word	Document ID	Occurrence
Now	1	1
is	1	2
the	1	3
time	1	4
for	1	5
all	1	6
good	1	7
men	1	8

Architecture



Index vs. Full-text index

Full-text indexes	**Regular SQL Server indexes**
Stored in the file system, but administered through the database..	Stored under the control of the database in which they are defined.
Only one full-text index allowed per table.	Several regular indexes allowed per table.
Addition of data to full-text indexes, called population, can be requested through either a schedule or a specific request, or can occur automatically with the addition of new data.	Updated automatically when the data upon which they are based is inserted, updated, or deleted.

Populating a Full-Text **Index**

SQL 2005

Because of the external structure for storing full-text indexes, changes to underlying data columns are not immediately reflected in the full-text index. Instead, a background process enlists the word breakers, filters and noise word filters to build the tokens for each column, which are then merged back into the main index either automatically or manually. This update process is called population or a crawl. To keep your full-text indexes up to date, you must periodically populate them.

From **SQL 2008** this is not a problem. You can choose from there modes for full-text population:

- Full
- Incremental
- Update

Populating a Full-Text **Index**

- Full
  - Read and process all rows
  - Very resource-intensive
- Incremental
  - Automatically populates the index for rows that were modified since the last population
  - Requires timestamp column
- Update
  - Uses changes tracking from SQL Server (inserts, updates, and deletes)
  - Specify how you want to propagate the changes to the index
    - AUTO automatic processing
    - MANUAL implement a manual method for processing changes

Examples

```
SELECT ProductDescriptionID, Description FROM Production.ProductDescription
    WHERE CONTAINS(Description, N' FORMSOF (INFLECTIONAL, ride) ');

SELECT ProductDescriptionID, Description FROM Production.ProductDescription
    WHERE CONTAINS(Description, N' FORMSOF (THESAURUS, ride) ');
```

Word proximity

- NEAR (~)
- How near words are in the text/document

```
SELECT ProductDescriptionID, Description FROM Production.ProductDescription
    WHERE CONTAINS(Description, N'mountain NEAR bike');

SELECT ProductDescriptionID, Description FROM Production.ProductDescription
    WHERE CONTAINS(Description, N'mountain ~ bike');

SELECT ProductDescriptionID, Description FROM Production.ProductDescription
    WHERE CONTAINS(Description, 'ISABOUT (mountain weight(.8), bikes weight (.2) )');
```

Disadvantages

- Full text catalogs
  - Disk space
  - Up-to-date
  - Continuous updating → performance hit
- Queries
  - Complicated to generate
  - Generated as a string

Advantages

- Much more powerful than LIKE
  - Specific
  - Ranking
  - Performance
- Pre-computed ranking (FREETEXTTABLE)
- Configurable Population Schedule
  - Continuously track changes, or index when the CPU is idle

VI. PROGRAMMING TECHNIQUES

Some guidelines

- Exploit precompiled, loaded code
  - Stored procedure, function
  - Avoid Embedded SQL
- Avoid coding loops
- Minimal Use of Cursors
  - Use set-based instead of row-based operations
  - Row-based can be unknowingly implemented by:
    - Cursors
    - DTS Lookup
    - Functions to perform lookups

Bad loop

```
for (int i = 0; i < 1000; i++)
{
    SqlCommand cmd = new SqlCommand("INSERT INTO TBL (A,B,C) VALUES...");
    cmd.ExecuteNonQuery();
}
```

```
INSERT INTO TableName (A,B,C) VALUES (1,2,3),(4,5,6),(7,8,9)
```

Table join is better than sub-query

- If A,B is many to one or one to one relationship
- Replace

```
SELECT * FROM A
WHERE A.CITY IN
(SELECT B.City FROM B)
```

- With

```
SELECT A.* FROM A INNER JOIN B ON A.City
```

In a join, small table should precede larger

- If A is a large table and B is small. Small table should drive the large table. This changes the table driving path.
- Replace

```sql
SELECT * FROM A,B
WHERE A.STATE = B.STATE
```

- With

```sql
SELECT * FROM B,A
WHERE A.STATE = B.STATE
```

Use indexed/materialized views

- A indexed/materialized view is a replica of base tables
  - → base tables change → must update data on view
- Can be queried like a normal view
- How they speed up queries
  - Perform JOINs and calculation in advance
  - Can be indexed to access faster

Lock problem

Effective Locking

- Use the lowest necessary isolation level
- For transaction
  - Keep short
  - Use the same resource use to avoid dead lock
- Don't hold locks while waiting for user Input!
  - Someone in service department wants to use an update screen to view data
  - Then goes on to view a work order
  - Then forgets and goes to lunch
- Not just user input, but any process that may have an open ended wait