Innehåll Optimering

- Var ligger problemet?
- Performance Killers
- Index
- Query Design Analysis

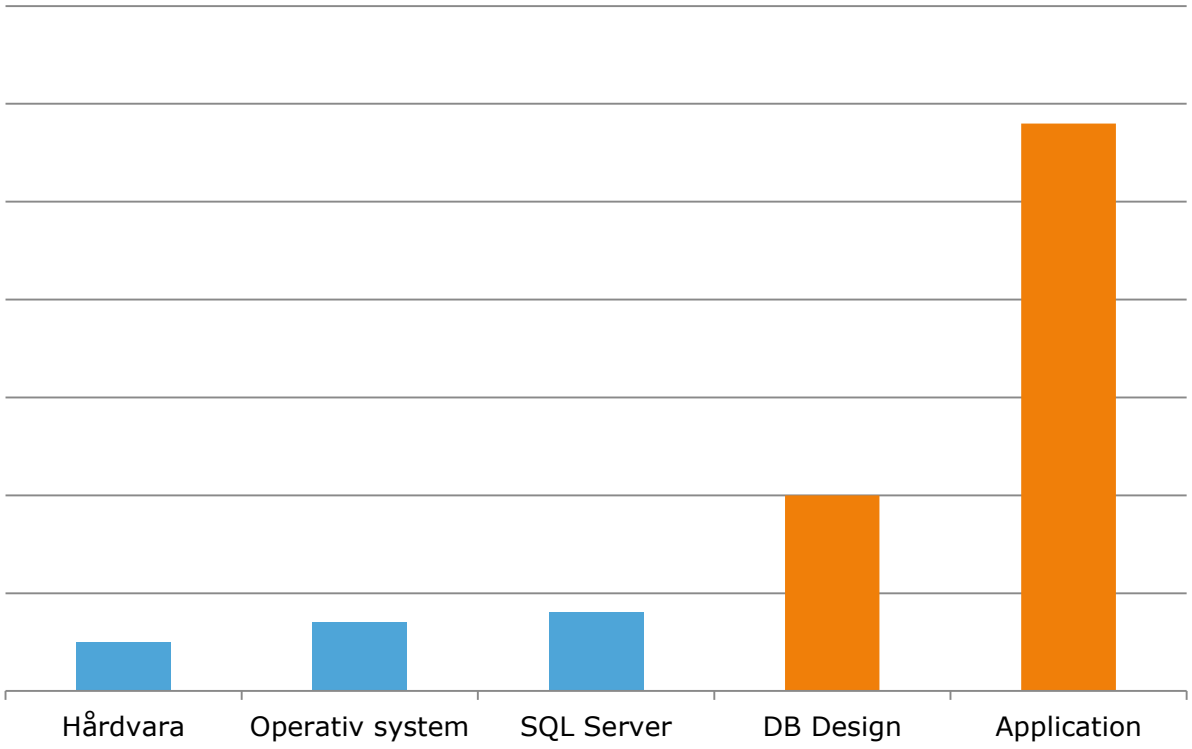
Optimering

Optimering av prestanda är ett brett område. Det omfattar bland annat:

- Att kunna bygga SQL satser utan att göra sånt som är vedertaget känt för att vara resurskrävande
- Att förstå den fysiska strukturen och tillvägagångssättet i en typisk databas
- Att kunna lösa de verkliga problemen istället för de imaginära problemen.

”De bästa sökningarna är de som påverkar det minsta antalet rader i en tabell och med sökvillkor som har en låg kostnad.
Med låg kostnad avses låg komplexitet, enkelt utförande och snabb exekvering”

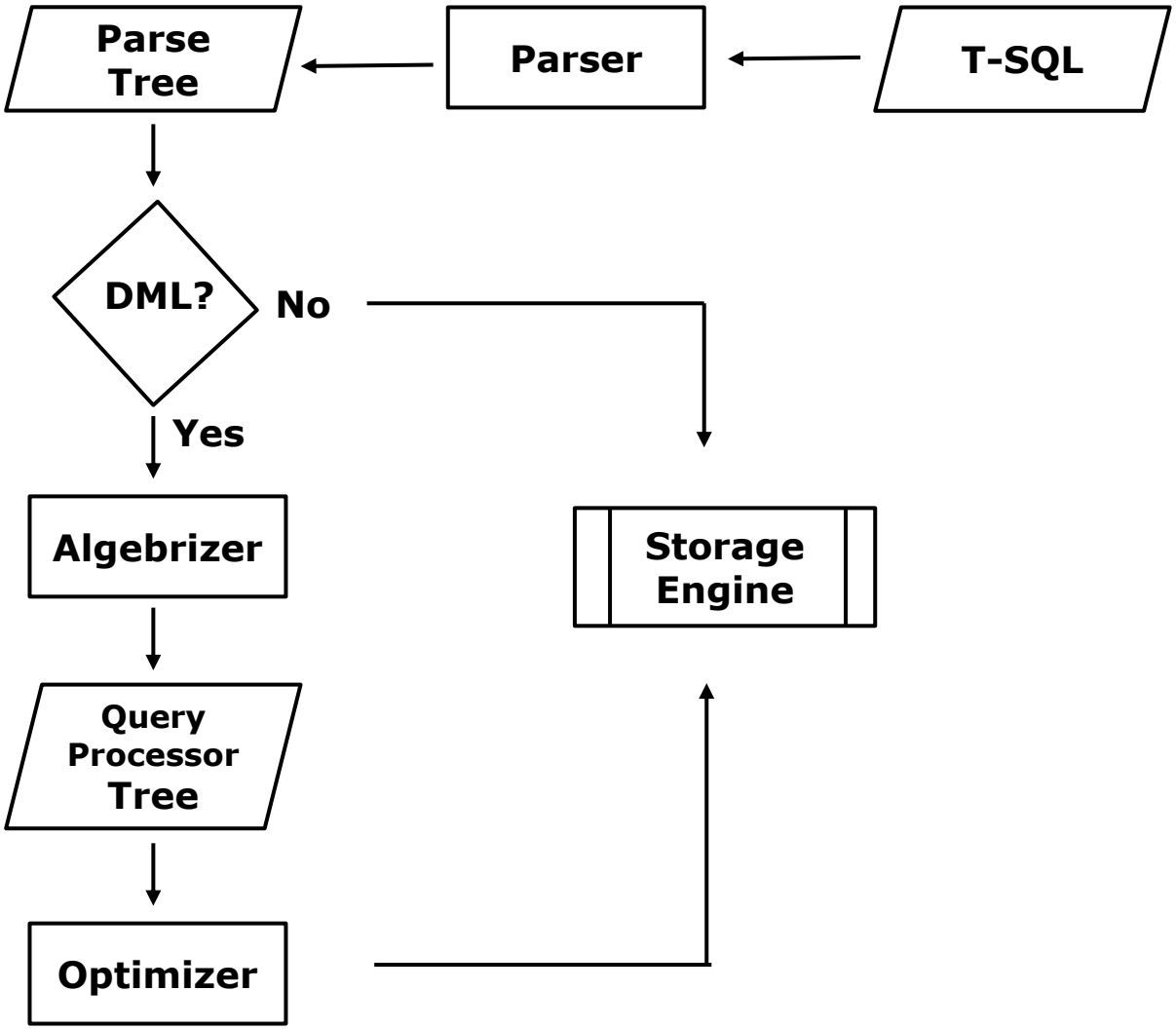
Problematiken



Performance killers

1. Index. Saknas, felaktiga eller för många
2. Onödiga låsningar (Blocking and Deadlocks)
3. Dålig databasdesign
4. Dålig utformning av SQL frågor
5. Excecution Plan

Query Execution Plan



Index – att tänka på

- ✓ Undersök WHERE och JOIN villkors kolumner
- ✓ Använd smala index
- ✓ Kontrollera om värden unika i kolumn/er
- ✓ Kontrollera datatyp, tal, text?
- ✓ Tänk på eventuell sortering, ORDER BY
- ✓ Typ av index, clustered eller non clustered

WHERE / JOIN optimeraren

1

När du exekverar en SQL sats så tar optimeraren hand om denna och försöker hitta ett bra index som förbättrar svarstiden.

Följande steg genomförs:

- ✓ Optimeraren identifierar kolumner som finns i WHERE och JOIN
- ✓ Optimerar undersöker sedan om det finns index på dessa kolumner
- ✓ Optimeraren värderar användbarheten på indexet/indexen för den aktuella frågan
- ✓ Optimeraren uppskattar den minst kostsamma metoden för att lämna resultatet i retur

Därför är det viktigt att du anpassar dina index till de frågor du kör i din applikation.

Datatyp i index

Index bör byggas i första hand på tal och i andra hand på fasta textfält och i tredje hand ?

ID	int
ENAMN	varchar(30)
FNAMN	varchar(20)
conamn	varchar(100)
GATUNR	int
UPPGANG	varchar(10)
GATA	varchar(40)
Fastnr	int
POSTNR	int

ID	int
MEDLEMID	int
Personnr	nvarchar(255)
ENAMN	nvarchar(30)
FNAMN	nvarchar(20)
conamn	nvarchar(255)
GATUNR	int
UPPGANG	nvarchar(10)
GATA	nvarchar(40)
POSTNR	int
ORT	nvarchar(20)
ANNDATUM	int
BETALTOM	int
INTDATUM	int
Fastnr	nvarchar(255)
Fastnrgl	nvarchar(255)
EPOST	nvarchar(80)
TELENR	nvarchar(16)
ENDDATUM	int
VERVKOD	nvarchar(2)
BETALKOD	nvarchar(4)
INPABET	int
TIDNING	tinyint
ANDRAHAND	tinyint
FORHANDL	tinyint
MEDLKORT	tinyint
HUVKORTDATUM	int

Smala index

Undvik att använda breda index, dvs index som är breda pga av ett brett fält eller flera smala fält – eller ännu värre flera breda fält i ett index. Indexen bör dessutom användas på ett riktigt sätt:

Tabellen

ID	Namn	a1	b1
1	Anna	1	2
2	Otto	4	1
3	Stina	9	7
4	Nicklas	2	3
5	lena	6	9

Index

ID	a1	b1
1	1	2
4	2	3
2	4	1
5	6	9
3	9	7

Dålig

```
SELECT *
FROM Tabell
WHERE b1=9 AND a1=6
```

Bra

```
SELECT *
FROM Tabell
WHERE a1=6 AND b1=9
```

Unika värden i index

1

Ett index ska helst ha enbart unika värden. Du ska undvika index på ett fält som innehåller samma värde på många poster.

ID	Enamn	Fnamn	Personnr	Kön	Grupp
1	Karlsson	Sven	780602-2550	M	Null
2	Olsson	Anna	741201-6542	K	1
3	Svensson	Stina	901015-2578	K	Null
4	Karlsson	Olle	780405-1528	M	1

Vilka fält ska jag helst inte sätta index på?

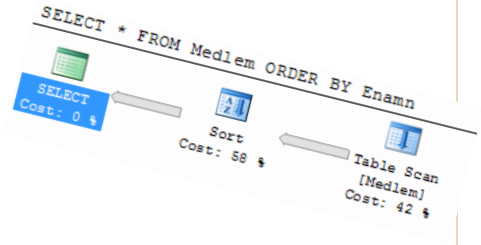
Sortering och index

Ett index används i samband med sortering (ex ORDER BY, DISTINCT , GROUP BY, JOIN). Finns ett index på det fält du gör order by på så används det per automatik men du kan också tvinga optimeraren till detta.

```
SELECT Kategori, SUM(Pris*antal) as Lagerpris
FROM Artikel
GROUP BY Kategori;
```

```
SELECT *
FROM Artikel WITH (INDEX(IX_Plats))
WHERE Plats Like 'Hylla%';
```

Det är först och främst tre saker som påverkar sorteringens hastighet. Nedan är de nämnda i viktighetsordning.



1. Antalet rader sorteringen omfattar
2. Antalet kolumner som nämns i ORDER BY satsen
3. Databredden på de kolumner som nämns i ORDER BY satsen

Execution Plan


```
SELECT * FROM Medlem ORDER BY Enamn
```

7,3 s

SELECT	
Cached plan size	32 B
Degree of Parallelism	0
Memory Grant	300496
Estimated Operator Cost	0 (0%)
Estimated Subtree Cost	42,4098
Estimated Number of Rows	293899

Sort

Sort the input.

Physical Operation	Sort
Logical Operation	Sort
Actual Number of Rows	293899
Estimated I/O Cost	0,0112613
Estimated CPU Cost	24,5046
Number of Executions	1
Estimated Number of Executions	1
Estimated Operator Cost	24,5158 (58%)
Estimated Subtree Cost	42,4098
Estimated Number of Rows	293899
Estimated Row Size	796 B

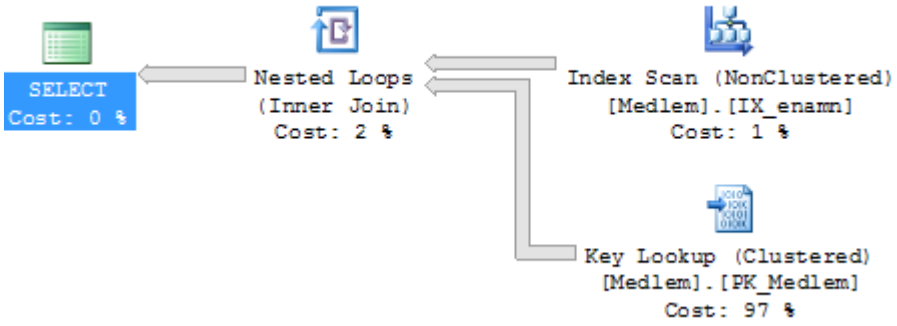
Table Scan

Scan rows from a table.

Physical Operation	Table Scan
Logical Operation	Table Scan
Actual Number of Rows	293899
Estimated I/O Cost	17,5705
Estimated CPU Cost	0,323446
Number of Executions	1
Estimated Number of Executions	1
Estimated Operator Cost	17,894 (42%)
Estimated Subtree Cost	17,894
Estimated Number of Rows	293899
Estimated Row Size	796 B

SELECT	
Cached plan size	40 B
Degree of Parallelism	0
Estimated Operator Cost	0 (0%)
Estimated Subtree Cost	75,3951
Estimated Number of Rows	293899

```
SELECT * FROM Medlem WITH (INDEX(IX_enamn)) ORDER BY Enamn
```



6,5 s

Typ av index

Det finns i grunden två typer av index.

Clustered som är sorterad i sin fysiska ordning, normalt Pk (=bäst).

Non Clustered som är parallell tabell med relation till ordinarie tabell, normalt Fk.

Clustered Index

Medlem

ID	Namn	...	...	...
1	Anna			
2	Ole			
3	Stina			
4	Bertil			
5	Sussi			
6	Oskar			

Non Clustered Index

Anna	1
Bertil	4
Ole	2
Oskar	6
Stina	3
Sussi	5

```
SELECT *
FROM Medlem
WHERE Namn='Oskar';
```

Sökning sker efter Oskar.

Indexet kopplas in och Oskar finns. Med hjälp av Index värdet hittas motsvarande post i Medlem.

Optimering Kolumnvis delning

Då viss information sällan efterfrågas kan man dela tabellen i två tabeller
"Viktiga data" + "Sällan använda data" =>Mindre och snabbare tabeller

Kund

KundID	Namn	Bla Bla	**Noteringar**
14	AB S&P	…	Diverse för det mesta ointressant information
20	Context AB	…	

Kund

KundID	Namn
14	AB S&P
20	Context AB

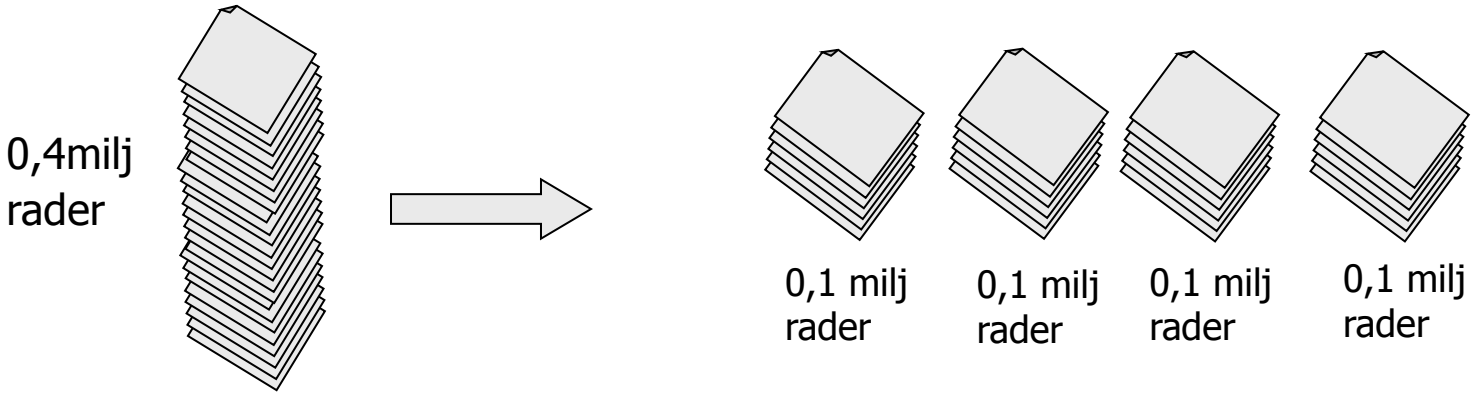
KundDetalj

KundID	Bla Bla	**Noteringar**
14	…	Diverse för det mesta ointressant information

Kan också vara ett sätt att separera publik data från säkerhetskänslig data.
=>Säkrare tabeller

Kolumnvis delning (Dekomponering)

Optimering Radvis delning



Tabellerna delas upp i flera kortare tabeller

- Aktuell info / Gammal info (ex årets fakturor o tidigare års fakturor)
- En tabell per år
- Aktiva / Passiva kunder

Fördelar:

- Korta tabeller - Snabb access
- Lätt att organisera arkivering av gamla data

Radvis delning (Segmentering/Partitionering)

Nackdelar:

- Måste byta tabeller
- Svårare att göra statistik över flera tabeller.

Optimering Kolumnvis sammanslagning

Slå samman två eller flera normaliserade tabeller till en onormaliserad.

Kund

Kundnr	Namn	Postadress	Tel	Distrikt
1	Direct AB	Gatan 33 Hjo	123 44	01
2	Direct LTD	Vägen 33 Ystad	55 66 77	03

Distrikt

Distrikt	Distriktsnamn
01	Mellan
03	Södra

Kund

Kundnr	Namn	Postadress	Tel	Distrikt	Distriktsnamn
1	Direct AB	Gatan 33 Hjo	123 44	01	Mellan
2	Direct LTD	Vägen 33 Ystad	55 66 77	03	Södra

Fördelar:
- Snabbare access
- Enklare – minskar antalet joinoperationer

Nackdelar:
- Ökad redundans
- Problem vid uppdatering av distriktnamn
- Svårighet att lagra distrikt utan kunder
- Applikationen måste göra mer omfattande validering

Man kan också dubbel-lagra distrikts-tabellen för att underlätta uppslag.

Optimering Reduntanta tabeller

Variant på denormalisering
- frekventa omfattande frågor lagras fysiskt i en tabell som kan läsas snabbt .
- måste genereras om då bakomliggande tabeller ändras.

Student - Alla studenter

StudID	Namn	Postadress	Tel	ProgID
1	Anders			1
2	Stina			2
7	Anna			1

Program - Utbildningslinje

ProgID	Program
1	Maskiningenjör
2	Webbprogrammerare

Maskin

StudID	Namn	Program
1	Anders	Maskiningenjör
7	Anna	Maskiningenjör

Redundant tabell med studenter i viss klass.

Syntax – SQL Satser

Det är även viktigt att hålla en konsekvent stil när man skriver SQL förfrågningar. Ta följande fyra uttryck som exempel:

```
SELECT column1*4 FROM Table1 WHERE COLUMN1 = COLUMN2 + 7
SELECT Column1 * 4 FROM Table1  WHERE column1=(column2 + 7)
```

Istället för att exekvera de två ovanstående uttryck bör man istället använda sig av följande:

```
SELECT column1 * 4 FROM Table1 WHERE column1 = column2 + 7
SELECT column1 * 4 FROM Table1 WHERE column1 = column2 + 7
```

Alla uttryck ger samma resultat, så kommer ibland det sista uttrycket att köra snabbare. Detta beror på att vissa databashanterare sparar vad som kan kallas förkompileringar av tidigare körda förfrågningar i en buffert, ibland även riktiga resultat om datan ännu inte har ändrats sen den senaste förfrågningen. En grundförutsättning för att denna buffert skall kunna utnyttjas är att förfrågningen ser exakt likadan ut som den tidigare, inklusive alla mellanslag och stora och små bokstäver.

Förutom att förfrågningarna blir lättare att läsa och förstå, så kan alltså en konsekvent skrivstil även bidra till att höja förfrågningens prestanda. Här kommer några tips för en läsbar och entydig syntax:

- **Nyckelord med stora bokstäver men kolumner med små bokstäver**
- **Tabellnamn har stor första bokstav**
- **Enkla mellanslag mellan varje ord och runtom varje aritmetisk operator**

Rangordning

Tabellerna nedan visar en typisk rangordning av sökvillkor (tabellen baserar sig på tillverkarnas egna manualer och är vedertagen känd i databassammanhang). Desto högre poäng en operator eller en operand har, desto effektivare är de att använda i sökningar.

Operator	Poäng
=	10
>	5
>=	5
<	5
<=	5
LIKE	3
<>	0

Operand	Poäng
Ensam direkt given operand	10
Ensam kolumn	5
Ensam parameter	5
Multioperand	3
Exakt numerisk datatyp	2
Annan numerisk datatyp	1
Temporär datatyp	1
Textbaserad datatyp	0
Null	0

Rangordning exempel


```
… WHERE antal=12345        -- datatyp smallint
```

Exemplet skulle enligt poängtabellen få hela 27 poäng!

- 5 poäng för att kolumnen (antal) är ensam på vänster sida
- 2 poäng för kolumnens datatyp (exakt numerisk datatyp)
- 10 poäng för likhetsoperatorn (=)
- 10 poäng för den direkta operanden (12345) på höger sida

```
… WHERE ort>=kommun || 'X'   -- ort=char,kommun=varchar
```

- 5 poäng för att kolumnen (ort) är ensam på vänster sida
- 0 poäng för kolumnens datatyp (char) på vänster sida
- 5 poäng för operatorn (>=)
- 3 poäng för multioperand (|| 'X') på höger sida
- 0 poäng för den textbaserade datatypen (kommun) på höger sida

Användning av konstanter

Vid sammanslagning av flera konstanter i samma uttryck kan konstantöverföring uppstå.

Därför är sammanslagning av konstanter en lönsam operation. Det enkla uttrycket

... WHERE a – 3 = 5

skall alltså hellre skrivas som

... WHERE a = 8 -- a – 3 = 5 -> a = 5 + 3

för att uppnå en högre poäng.

WHERE

... WHERE column1 = ’A’ AND column2 = ’B’

omformas till

... WHERE column2 = ‘B’ AND column1 = ‘A’

om det är mindre sannolikt att column2 är lika med ’B’ än att column1 är lika med ’A’.

Vilket innebär att optimeraren tittar på vilket som är minst sannolikt.

... WHERE column2 = ’B’ OR column1 = ’A’

omformas till

... WHERE column1 = ‘A’ OR column2 = ‘B’

om det är mera sannolikt att column1 är lika med ’A’ än att column2 är lika med ’B’.