Large databases and performance

Lecture UiO 22. October 2025

Audun Faaberg - DNB

Agenda

1. Introduction.
2. Tune the SW
   a. Indices
   b. Efficient SQL
   c. Efficient code & design
3. Concurrency
   a. Isolation levels
   b. Clustering of tables
4. New directions
5. Real life examples

About me

- Performance architect at DNB – Den Norske Bank, Norway's largest bank..
- 1989 – 2016 Accenture Technology Consulting, leading the Performance engineering group in Norway & the Nordics. 2016-2018 Performance architect in NAV.
- Specialist in performance optimisation, SQL optimisation, coding optimisation.
- Make programs run faster!

- Bought my first computer, a Sinclair ZX-81 with 1 KB internal memory and 16 KB extended memory in September 1981.

DNB

DNB – Den norske bank

2 000 000
private customers

210 000
corporate customers

%

Market share Norway:

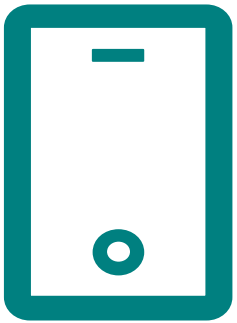
- Private:
  30% deposits
  23% mortgages
- Corporate:
  30% loan
  37% deposits

Around 11 700 employees, 9 500 of which are based in Norway.

- 48,5% women
- 51,5% men

Around 950 employees in the IT department.

We work with selected consultant companies.

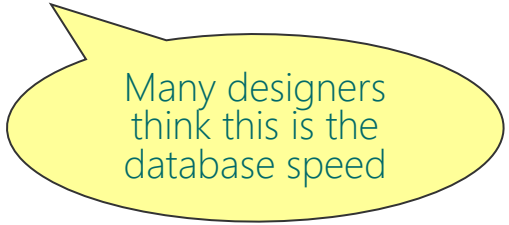


Disclaimer :-)

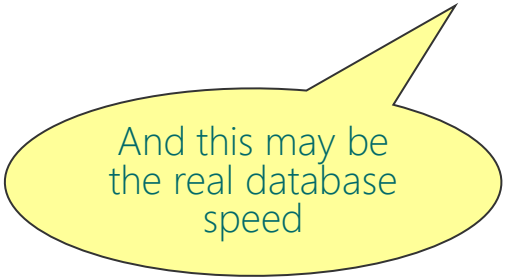
- I have worked in Norway, Denmark, Sweden, France, Italy, Brazil, The Czech Republic, Malaysia, Germany – MEANING – "a telco" is not Telenor, "a bank" is not DNB, "an oil company" is not Equinor…… and not all examples are from NAV and DNB!
- Specific numbers may be old (page size, cache size, IO speed). But the logic should still apply, though of course the massive technical improvements currently may lead to this logic producing another result…..
- This is not a full course of tuning in large databases, it is to make you aware of what is out there…

1. Introduction: Basic arithmetic

0.000 sec x 6,000,000 = 0 sec



0.012 sec x 6,000,000 = 72000 sec = 20 hours



1. Introduction: Poor performance

- Poor performance in the database and the code using the database is the most common reason for poor system performance.
- Poor performance may render an otherwise good system useless. Or despised. Or lead to ineffective organisations and numerous coffee breaks….
- Problems with performance may cause large delays in the final phases of a project, though should be more manageable than other issues that may occur at this stage.
- Finally, there is a lack of understanding that optimising code may drastically alter CPU & IO consumption and runtime. Therefore, more HW is the normal (though often wrong) answer.

DNB

1. Introduction: Why is there still a problem

- HW is faster. CPU, disks, network, memory, everything is much faster now than 10 years ago.
- Moore's law: Integrated circuits would double in performance every 18 months.
- Why is there still a problem?

- Niklaus Wirth's law: Software is getting slower more rapidly than hardware becomes faster.
- To be more specific – Bill Gate's law: The speed of commercial software generally slows by fifty percent every 18 months.

- And we let powerful hardware hide poor handcraft.

DNB

1. Introduction: Why is there still a problem

- Databases are now more finely modelled, catching more data, and creating complexities an order of magnitude greater.
- Flexible, parameterised, configurable system use many small parameter tables, which are used in every select. So earlier when a select referenced 2-3 tables, it may now reference 8-10 tables to fetch the same data. (And beware if a system is "generic")
- Greater ambitions – meaning – now much larger data volumes are stored. 20-25 years ago, we were stingy when designing databases.
- SQL code can be much more complex now. 10-15 years ago, you could look at an SQL and within minutes understand the functionality. Now…. Sometimes it takes days….

1. Introduction: Example of data modelling

Loan	
Cust_id	Balance
10	2,405,256
20	1,530,203

Earlier modelling,
1 record per loan

2 million customers

2 million rows

Loan	
Cust_id	Loan-_series_id
10	1,562,030
20	1,830,203

Recent modelling,
241 records per loan

2 million customers

482 million rows

Functionally more flexible, but at a cost

Loan_series (one record for every month throughout 20 years)					
Loan_series_id	Year	Month	Org_amount	Remaining	Interests
1,562,030	2018	01	2,405,256	2,395,256	23,952
1,562,030	2018	02	2,405,256	2,375,256	23,752
…	…	…	…	…	
1,562,030	2038	12	2,405,256	20,000	200

2. Tune the SW

Note: Tuning of SQL is typically a 40 hours introduction course
+ 5 years experience.
This is just a broad overview, so you will know there is more (much more) to know.

- Indices
- Efficient SQL
- Efficient code & design

Basic principles

- Divide and conquer
- Minimise the fetch of everything

DNB

2. Tune the SW: A DBMS model

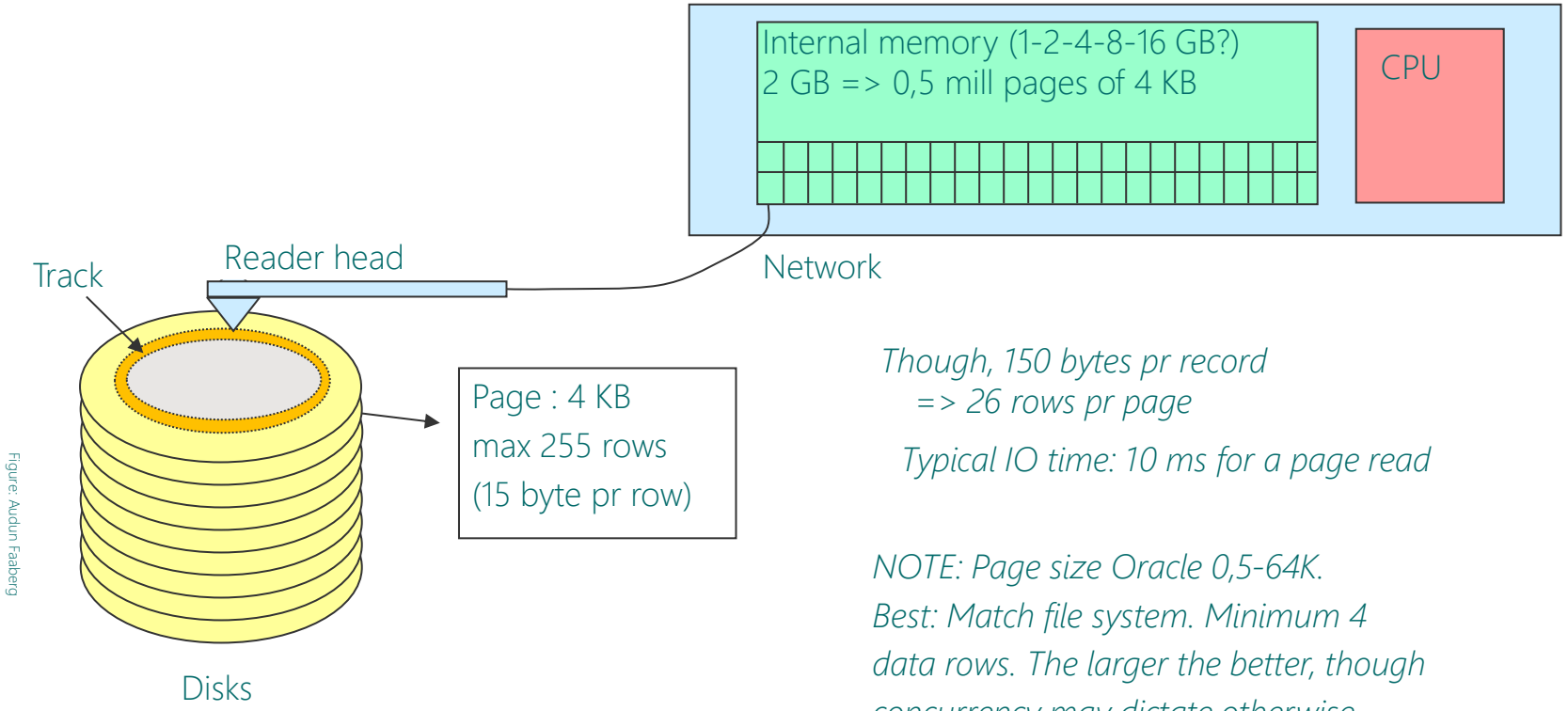


Figure: Audun Faaberg

Though, 150 bytes pr record
=> 26 rows pr page

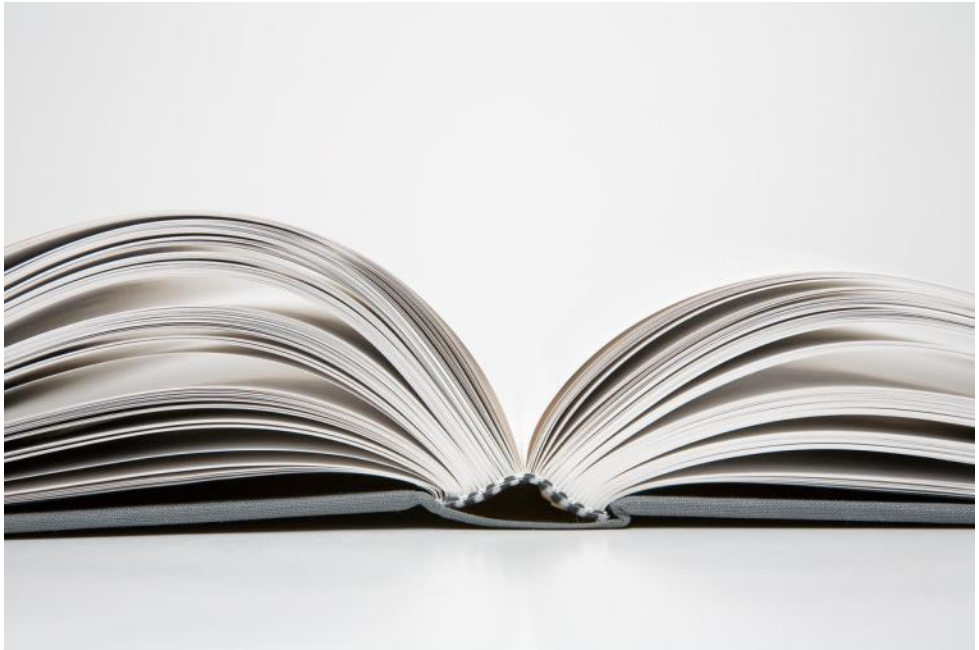
Typical IO time: 10 ms for a page read

NOTE: Page size Oracle 0,5-64K.
Best: Match file system. Minimum 4 data rows. The larger the better, though concurrency may dictate otherwise

2. Tune the SW – a. indices. What are indices – real life example

Non fiction book

- Content pages
- Index pages



Index

PERSONREGISTER
(Person index)

2. Tune the SW – a. indices. What are indices

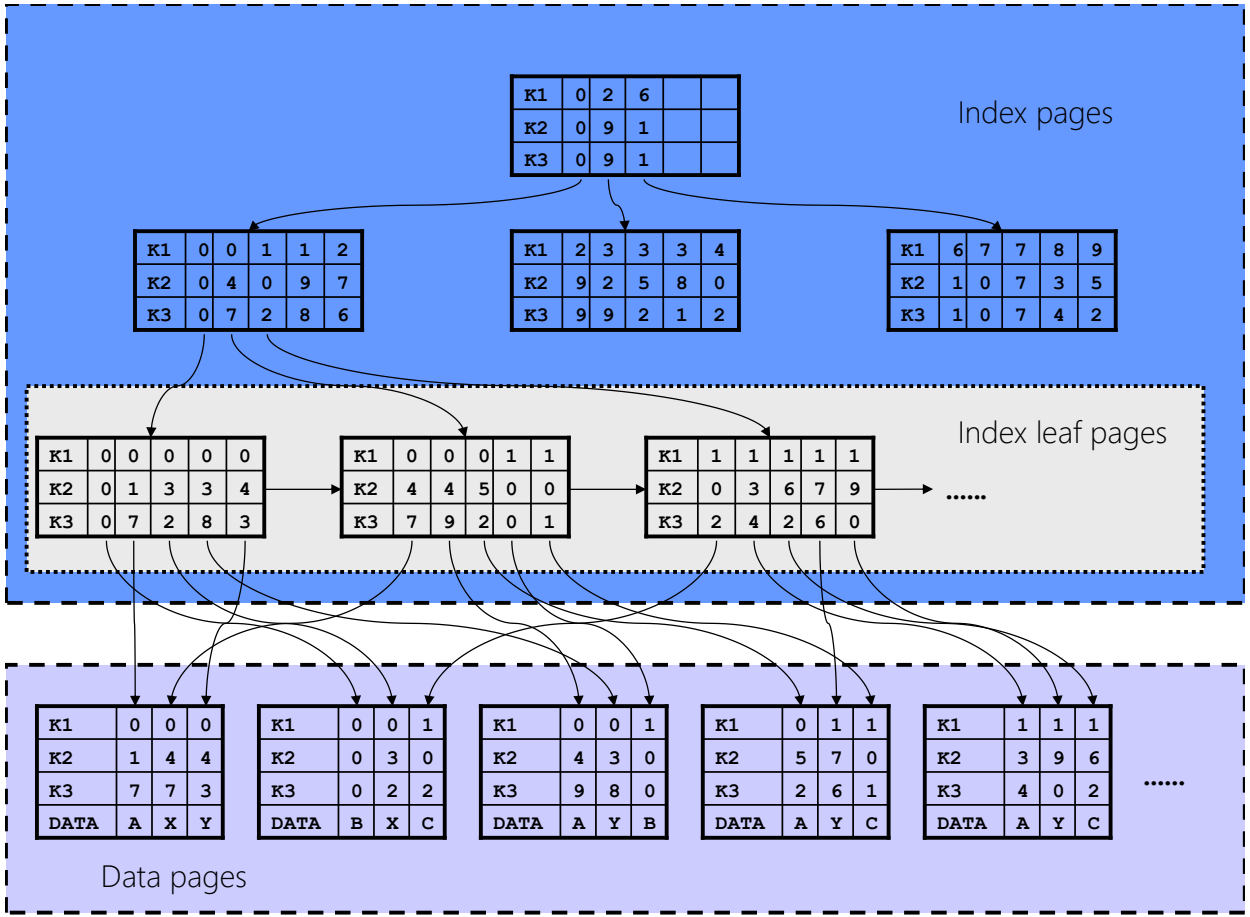


Figure: Jan Haugland

2. Tune the SW – a. indices. Full Table Scan

```
SELECT K2
  FROM SIMPLE_TABLE
 WHERE DATA = 'X'
```

- All yellow keys match
- All blue values returned
- All red pages scanned

- Let us assume the table has 50 mill rows, 20 rows pr page. (page 4K, row 200 bytes).
- In mean 1.250.000 page reads to find a random row.
- 10 ms pr page read..... 12.500 sec = 3,5 hours

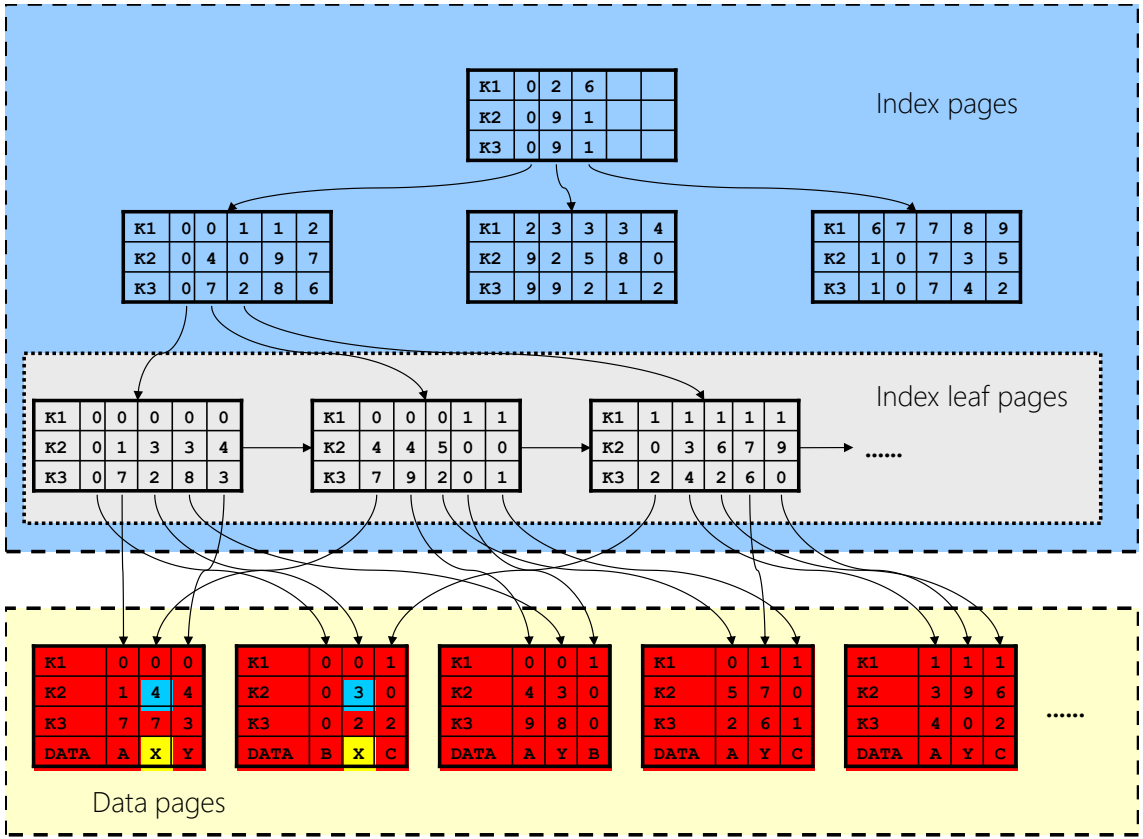


Illustration: Jan Haugland

2. Tune the SW – a. indices. Matching Index Scan

```
SELECT DATA
  FROM SIMPLE_TABLE
 WHERE K1 = 0
   AND K2 = 3
```

- All green keys match
- All yellow index entries used
- All blue values returned
- All red pages scanned

- Let us assume the table has 50 mill rows, 20 rows pr page.
- Indeces: 20 bytes – 200 on each page. 250.000 leaf pages, need 3 levels of index pages.

- 5 IOs -> 50 ms.

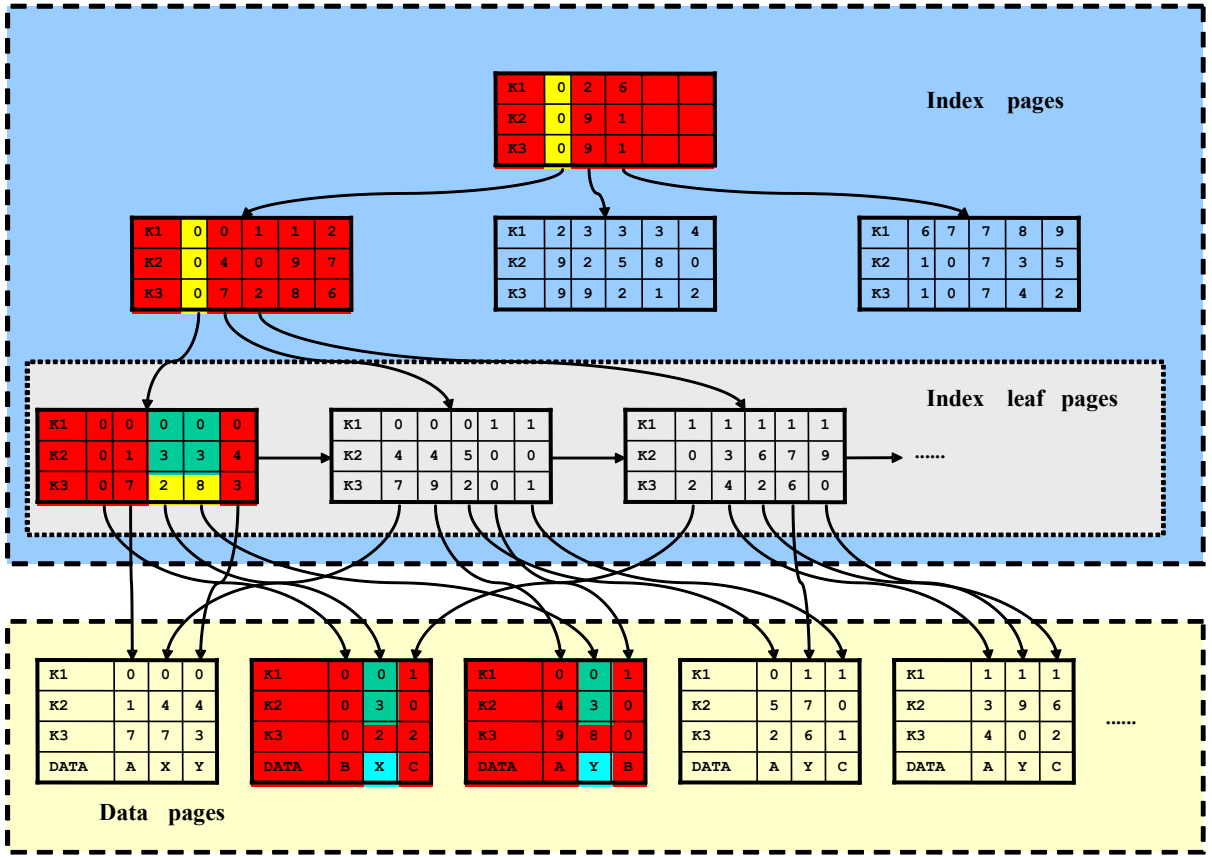


Illustration: Jan Haugland

2. Tune the SW – a. indices. Non-matching Index Scan

```
SELECT DATA
  FROM SIMPLE_TABLE
 WHERE K3 = 7
```

- All yellow keys match
- All blue values returned
- All red pages scanned

- Let us assume the table has 50 mill rows, 20 rows pr page.
- Indices: 20 bytes – 200 on each page. 250.000 leaf pages
- 125.000 IOs -> 1250 s.

Illustration: Jan Haugland

2. Tune the SW – a. indices. Specify your select

```
SELECT K2, K3
  FROM SIMPLE_TABLE
 WHERE K3 = 7
```

- Will result in a scan of the index leaf pages
- No read of data pages necessary.
- This is one reason to avoid SELECT * and rather specify the columns.
- Sometimes we add a missing column to the index
- If you have many hits, you may save 50% of the IO.
- Few hits, negligible gain.

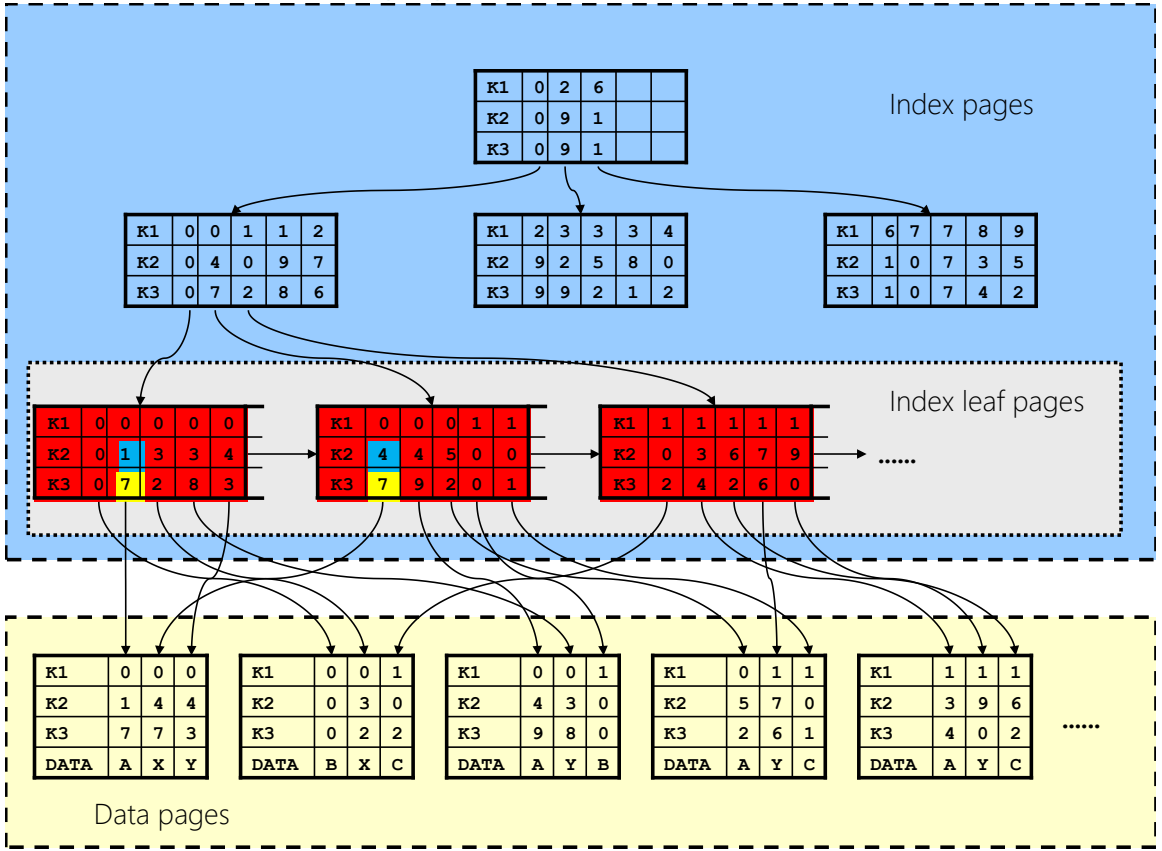
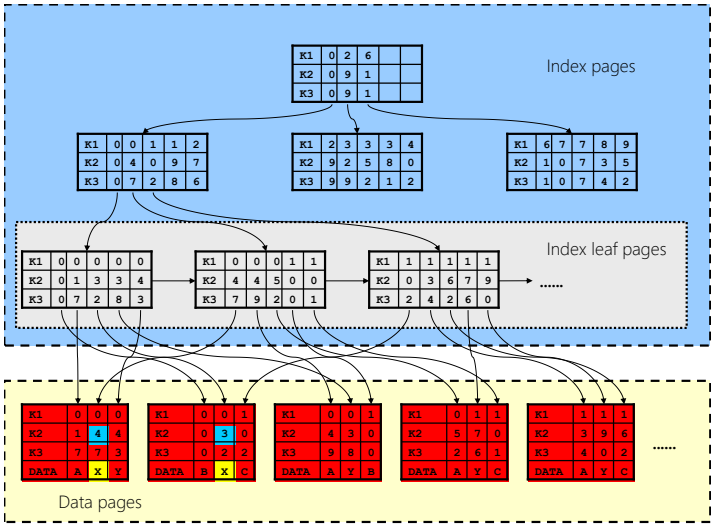


Illustration: Jan Haugland

2. Tune the SW – a. indices. Sequential prefetch

- A correction – the table scans are in fact more efficient than depicted in the earlier examples.
- A mechanism "Sequential prefetch" (or "scatter read") is invoked when the DBMS discovers that it is reading in sequence through the pages (typically 3 pages in sequence within 10 page reads).
- Starts to read 50 and 50 pages, typically at 30 ms (compared to 10 ms for one page). Leading to 25.000 read operations in a full table scan, or 750 / 2 sec = 6,25 minutes to find a random row (mean).



- Also the non-matching index scan will start prefetching. Leading to 4.000 read operations, or 120 / 2 sec = 60 sec.
- This can be utilised in large batch reads!
- Typically – if you try to select more than 10% of the rows in a table, the optimiser will go for a table scan.

2. Tune the SW – b. efficient SQL. Optimiser

- Cost based optimiser: Uses statistics, and weighs the different operations.

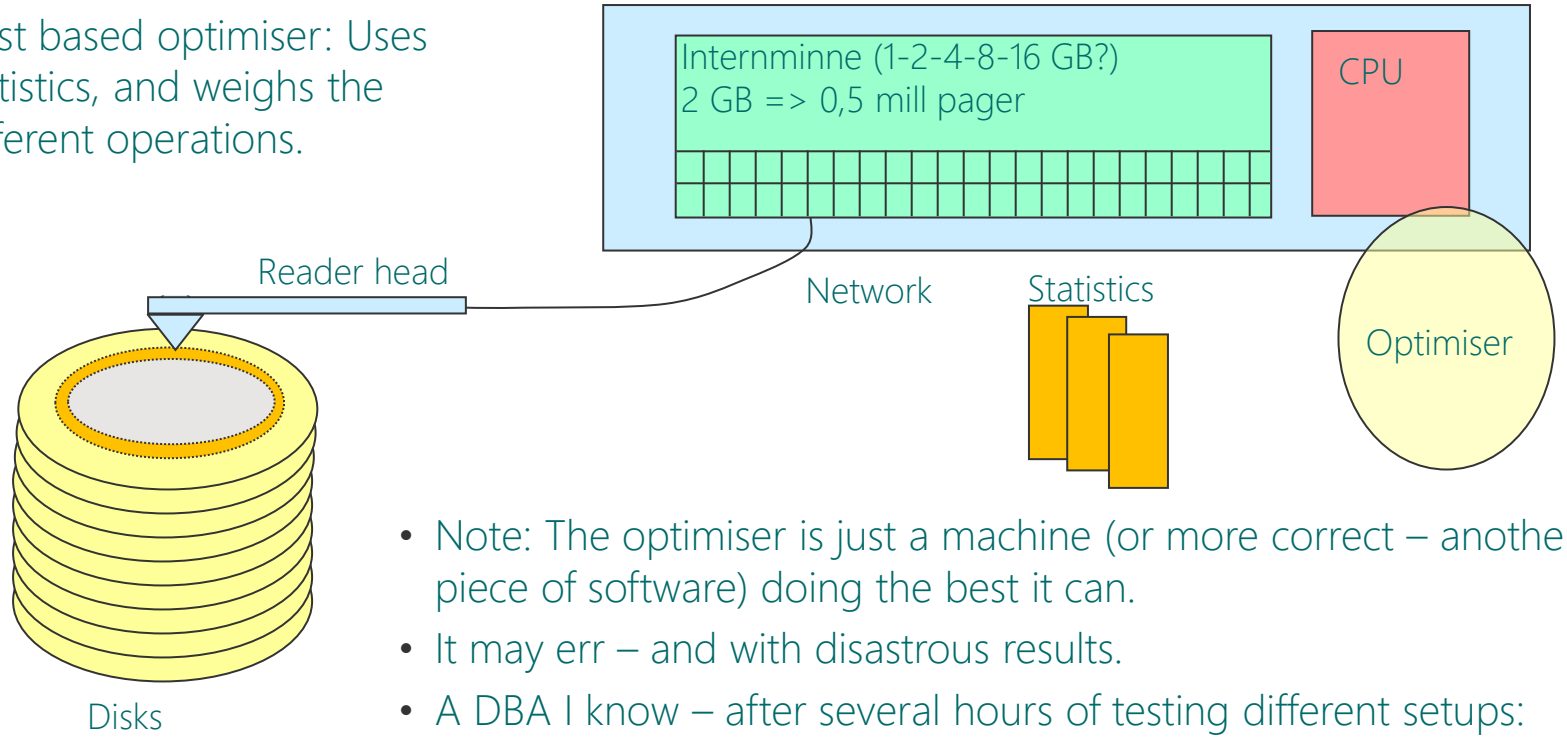


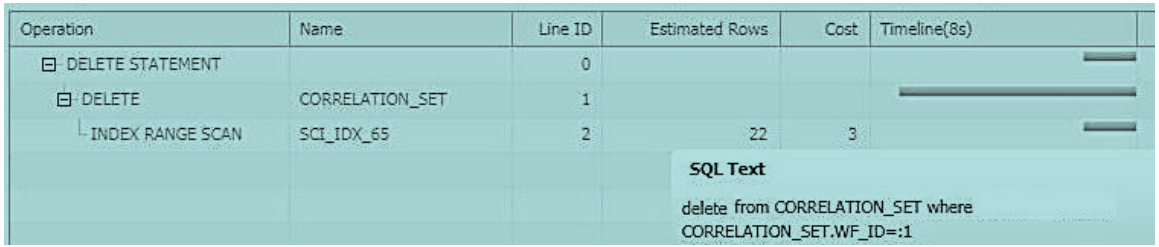
Figure: Audun Faaberg

- Note: The optimiser is just a machine (or more correct – another piece of software) doing the best it can.
- It may err – and with disastrous results.
- A DBA I know – after several hours of testing different setups: "Finally I framed the optimiser!"

2. Tune the SW – b. efficient SQL. Access path

- Access path is the way and sequence the DBMS applies rules. Using an index? Joins – in which order? Sort?
- It may be necessary to understand the access path.
- There are two ways of doing this
  - Many databases have an "explain plan" function
  - Many databases give you an overview of which tables and indexes are used, with #IOs on each. You may deduce the access plan by this.

Example 1: Graphical



Name	Type	GetPages	Elapsed	SyncRead	Table Name
IVEDE02	INDEX	35 237 679	39:21.600699	2 267 127	T_VENT_DETALJ
IVEBE02	INDEX	155 093	1:00.706364	24 198	T_VENT_BEREGNING
IVESS01U	INDEX	480	0,052551	104	T_VENT_STOPPSTATUS
T_VENT_STOPPSTATUS	TABLE	158	0,032234	17	
IKOVS01U	INDEX	31	0,00003	0	T_VENT_STATUSKODE
T_VENT_STATUSKODE	TABLE	31	0,000031	0	

Example 2: Tabular

2. Tune the SW – b. efficient SQL. Access path

Consider the employee table

```
select lname, empno, sal
from emp where
upper (lname) = 'FAABERG';
```

Eriksen	Erik	123
Faaberg	Audun	154
Faaberg	Rasmus	549
Horpen	Hallvor	798
Tallaksen	Tallak	101

With no index:

```
0 SELECT STATEMENT Optimizer=COST
1 0 TABLE ACCESS (FULL) OF 'EMPLOYEE_TABLE' 50
```

With a function index on upper(ename):

```
0 SELECT STATEMENT Optimizer=CHOOSE
1 0 INDEX (RANGE SCAN) OF 'UPPER_ENAME_IDX' (NON-UNIQUE) 1
```

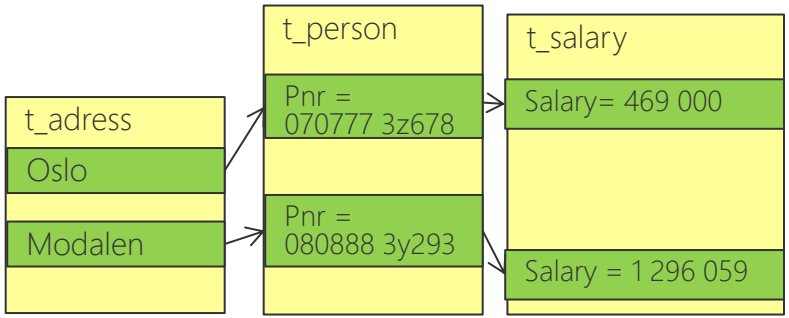
2. Tune the SW – b. efficient SQL. Looking for the millionaire

Before we start looking at SQLs and access paths - let us look at the real world. Tax is fun. And here we can study access paths.
How would you find the millionaires in Modalen municipality (one of the smallest municipalities in Norway). By hand - sifting through index cards.

a) Give me index cards of the millionaires in Norway, with the county added on. Scan through the index cards.

b) Give me all index cards of Modalen. I will scan through all of it.

And for Oslo?



Distribution info:

Modalen: 274 taxpayers

Oslo: 439 272 taxpayers

Over 1 million in Norway: 60 261

2. Tune the SW – b. efficient SQL. Can I predict the execution sequence of a compound statement?

No sequence granted, but most likely something like:

```
select mandatory1.x                        (7)
      ,optional.y
  from mandatory1                          (2 or 3)
inner join mandatory2                      (3 or 2)
    on mandatory1.z = mandatory2.z
left outer join optional                   (4)
    on optional.u = mandatory2.u
 where mandatory2.w = ?
   and mandatory1.a in
        (non-correlated subselect)         (1)
   and exists (correlated subselect)       (5)
order by mandatory.x                       (6)
```

2. Tune the SW – b. efficient SQL. SQL tuning

- The SQLs you meet in real life are often much more complex than the examples I have given.
- Most important tool – sql statistics (all DBMSs have some way for gathering this).
- A large system may have thousands of SQLs spread out in the code (or as stored procedures referenced in the code).
- In a problem situation, normally a handful (5-10-20) SQLs are causing problem. Though many more may be inefficient….
- First of all, identify them.
- Look for logical reads and physical reads in statistics, thus identifying the problem candidates.
- Candidates may be:
  - Light SQLs, somewhat inefficient, but very frequently executed.
  - Heavy SQLs with massive reads (logical and/or physical).

2. Tune the SW – b. efficient SQL. Tools – Detector (SQL monitor)

09.02.2018
Interval 0000-0400

PROGRAM	SQL	CPUPCT	INDB2_TIME	INDB2_CPU	GETPAGE
K411S024	131364	19.92%	43:20.142312	19:51.943399	75 035 827
K415B940	7206008	5.83%	12:53.232469	05:12.778640	42 480 939
K231B510	521364	4.97%	04:59.215339	04:26.795714	12 158 914
K278U950	4060	4.03%	53:12.498745	03:36.202277	93 502 081
K411S025	4072793	3.75%	08:08.467209	03:21.168218	10 610 520
K278BAN1	16086	2.79%	14:32.592899	02:29.905171	8 802 622
DSNESM68	8655	2.54%	09:12.452910	02:16.268729	23 541 223
K411S103	1966527	1.93%	05:12.101298	01:43.911804	4 951 095
K2300211	3068353	1.76%	20:32.410801	01:34.334010	3 433 748

SQL_CALL	SQL	TIMEPCT	CPUPCT	INDB2_TIME	INDB2_CPU	GETPAGE
SELECT	85191	88.86%	88.93%	01:53:30.21>	19:06.265410	74 983 286
FETCH	46173	11.12%	11.05%	00:14:12.86>	00:41.889843	52 541

```
SELECT  COUNT ( * )
INTO :H
FROM DB411.DUE_PAYMENTS
WHERE BANK_CUSTOMER_ID = :H
    AND ACCOUNTED_DATE = :H
    AND AMOUNT = :H
```

Start optimising from the top.
Use information in the tool.
Finds easily the culprit SQL.
Optimise CPU-consumption? IO (getpages)? Elapsed time?

2. Tune the SW – b. efficient SQL. Example 1

```
SELECT  COUNT ( * )
INTO :H
FROM DB411.DUE_PAYMENTS
WHERE ACCOUNT_NO = :H
     AND ACCOUNTED_DATE = :H
     AND AMOUNT = :H
```

Index used: **IA01**

ACCOUNT_NO
SWIFT_ADR
ACCOUNTED_DATE
PAYMENT_ID
AMOUNT
PAYERS_ACCOUNT
STATUS
MSG_TYPE

Very large table!

SQL_CALL	SQL	TIMEPCT	CPUPCT	INDB2_TIME	AvgIndDB2	INDB2_CPU	AvgCPU
SELECT	85191	88.86%	88.93%	01:53:30.21	0,0799	01:41:06.26	0,0712

TABLE	INDEX	TB_SEQ_GP	TB_IDX_GP	IS_GETP
DUE_PAYMENTS	IA01			17716.9

This is average pr SQL execution

2. Tune the SW – b. efficient SQL. Example 1

```
SELECT  COUNT ( * )
INTO :H
FROM DB411.DUE_PAYMENTS
WHERE ACCOUNT_NO = :H
     AND ACCOUNTED_DATE = :H
     AND AMOUNT = :H
```

Very large table!

Modified index: **IA01M**

```
ACCOUNT_NO
ACCOUNTED_DATE
AMOUNT
SWIFT_ADR
PAYMENT_ID
PAYERS_ACCOUNT
STATUS
MSG_TYPE
```

SQL_CALL	SQL	TIMEPCT	CPUPCT	INDB2_TIME	AvgIndDB2	INDB2_CPU	AvgCPU
SELECT	24	00,19%	00,23%	00,0012288	0,0000512	00,000924	0,0000385

TABLE	INDEX	TB_SEQ_GP	TB_IDX_GP	IS_GETP
DUE_PAYMENTS	IA01			1.9

This is 1800x faster.
With 1/9324 getpages.

2. Tune the SW – b. efficient SQL. Example 2

```
SELECT MIN (LIST.DATO_FOM) AS DATO_BER_FOM
FROM V_LINJE_STATUS LIST,
    V_OPPDRAG OPPD,
    V_OPPDRAGSLINJE OPLI,
    V_FAGOMRAADE FAGO
WHERE OPPD.OPPDRAG_GJELDER_ID = :H
  AND FAGO.KODE_FAGGRUPPE = :H
  AND FAGO.KODE_FAGOMRAADE = OPPD.KODE_FAGOMRAADE
  AND OPPD.FREKVENS = :H
  AND OPLI.OPPDRAGS_ID = OPPD.OPPDRAGS_ID
  AND OPLI.ATTESTERT = :H
  AND LIST.OPPDRAGS_ID = OPPD.OPPDRAGS_ID
  AND LIST.LINJE_ID = OPLI.LINJE_ID
  AND LIST.KODE_STATUS IN (:H, :H, :H, :H)
  AND LIST.TIDSPKT_REG = (SELECT MAX (LIS1.TIDSPKT_REG)
                          FROM V_LINJE_STATUS LIS1
                         WHERE LIS1.OPPDRAGS_ID =    LIST.OPPDRAGS_ID
                           AND LIS1.LINJE_ID = LIST.LINJE_ID
                           AND LIS1.DATO_FOM = LIST.DATO_FOM))
```

T_LINJE_STATUS LIST
Indexed fields
DATO_FOM
OPPDRAGS_ID
LINJE_ID
KODE_STATUS
TIDSPKT_REG
OPPDRAGS_ID
LINJE_ID
DATO_FOM

We have an index with
OPPDRAGS_ID,
LINJE_ID
KODE_STATUS
DATO_FOM

I see that most of the elapsed time is in T_LINJE_STATUS.

I am also reading an index ILIST01U on that table.

Name	Type	GetPages	Elapsed	Index Table Name
T_LINJE_STATUS	TABLE	16 123 812	12:28.117150	
ILIST01U	INDEX	15 515 944	5:32.999475	T_LINJE_STATUS
T_OPPDRAG	TABLE	14 665 872	9:50.858126	
T_FAGOMRAADE	TABLE	932 550	0,484374	
IOPLI01U	INDEX	443 206	1:11.982868	T_OPPDRAGSLINJE
IOPPD04	INDEX	221 640	1,324421	T_OPPDRAG
T_OPPDRAGSLINJE	TABLE	69 210	30,038437	
IFAGO01U	INDEX	1 842	0,001611	T_FAGOMRAADE

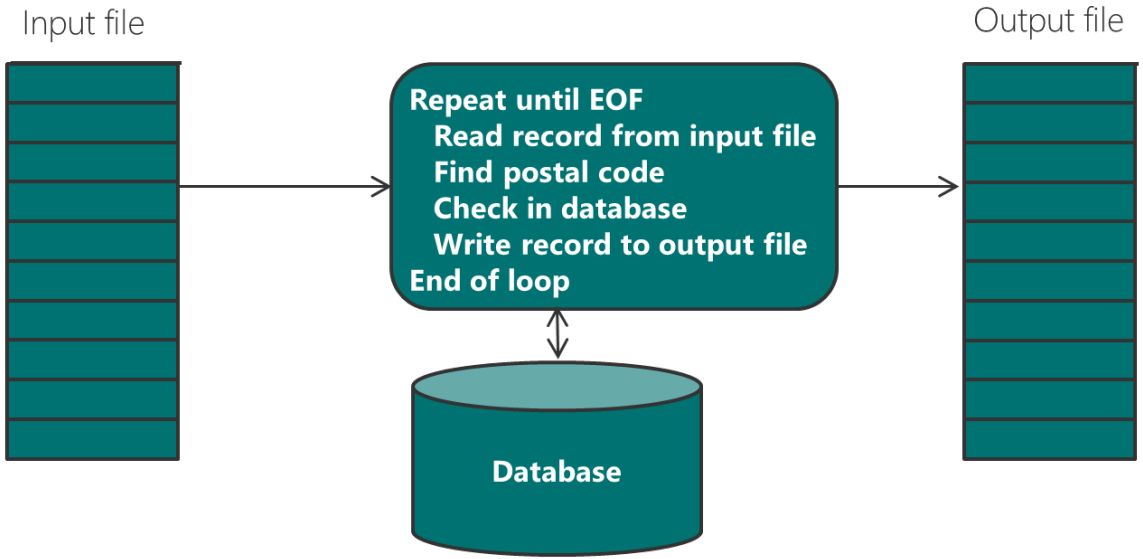
2. Tune the SW – c. Efficient code and design
Introduction

- Now we have looked into how to how to make the SQL to execute more efficient.
- Still, the DBMS has to execute the SQLs sent to it.
- Next focus should be to reduce the numbers of calls to SQL.
  (Remember the Axe Law: Don't use it if you don't mean it).
- Note: In a large project, this must be conveyed to the designers and the programmers early on. May be expensive to remove general problems afterwards.

DNB

2. Tune the SW – c. Efficient code and design
The post number lookup

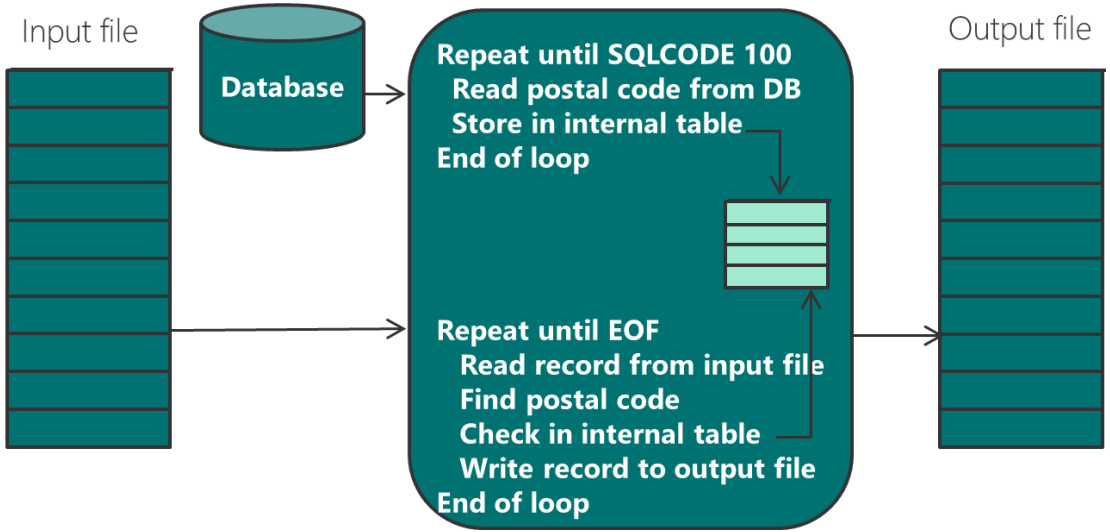
- Do not read over and over again the same value from the DB.
- Example: Verifying address information from 4 million customers.



- Reading the post number table pr customer record -> 4 million reads.
- This specific read may take 1-1,5 hours of a large run.

2. Tune the SW – c. Efficient code and design
The post number lookup

- Read the whole post number table into memory. 10.000 reads, after a short time a multiple page read (40 pages – 2 IOs of 50 ms) -> 0,1 second.

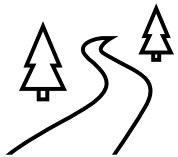


- The run time difference is virtually null on small volumes (on which the programmer typically test).
- On large volumes the difference is massive.

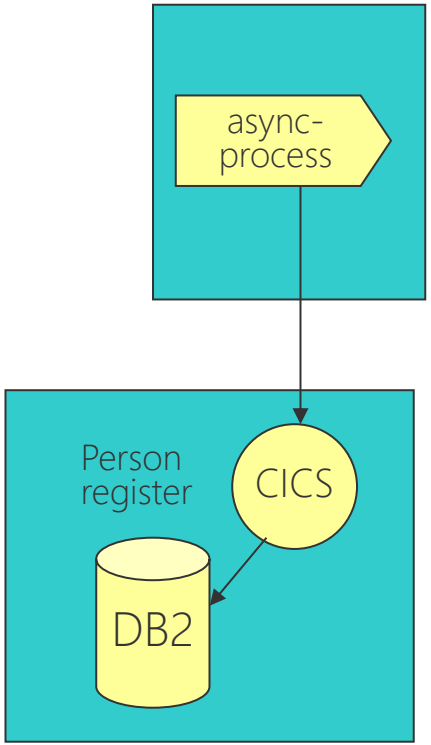
Database subsystem overhead

Databases are fine, though there is a certain overhead to execute a DB call.

I like to compare a database to a very chatty neighbour….it's OK to meet and talk, but try to keep it to a minimum.



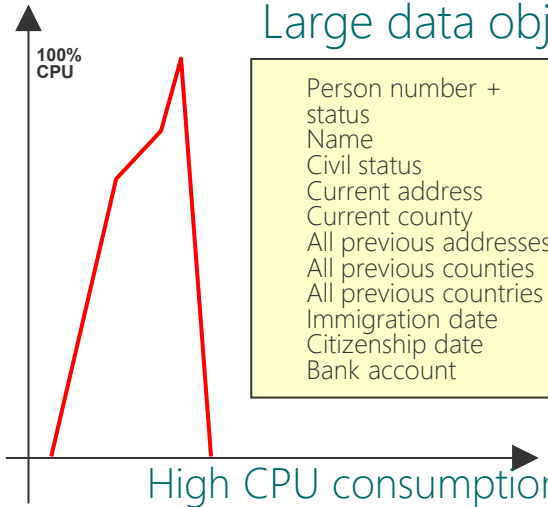
2. Tune the SW – c. Efficient code and design
The person info lookup



Person number +
status
Name
Civil status
Current address
Current county
All previous addresses
All previous counties
All previous countries
Immigration date
Citizenship date
Bank account

Phone number
Mobile number
Email address
Foreign stays
Person status
Description of
person status
Incapable and date
Filial used
and 7 more

95% is thrown away in the async-process (2)



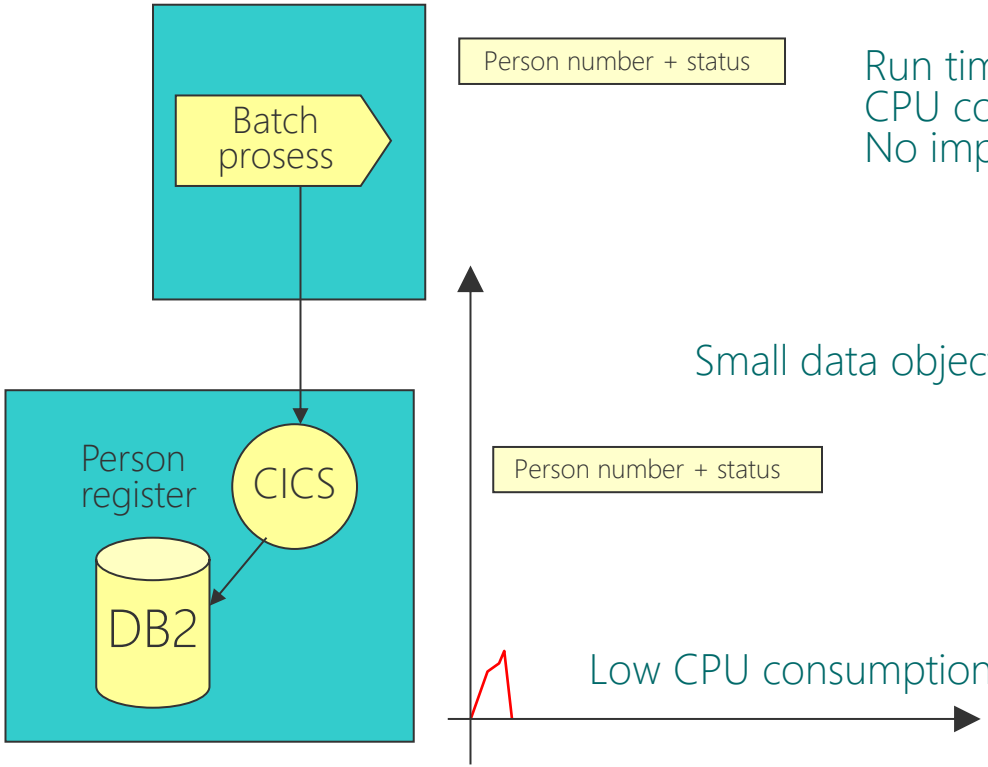
Large data object is created (1)

Person number +
status
Name
Civil status
Current address
Current county
All previous addresses
All previous counties
All previous countries
Immigration date
Citizenship date
Bank account

Phone number
Mobile number
Email address
Foreign stays
Person status
Description of
person status
Incapable and date
Filial used
and 7 more

High CPU consumption on DB server,
Massive impact on other systems for 15 minutes

2. Tune the SW – c. Efficient code and design
The person info lookup – new solution



Run time reduced 90%
CPU consumption reduced by 95%. (2)
No impact on other systems now.

Small data object created (1)

Low CPU consumption, short run time, less than 1 min

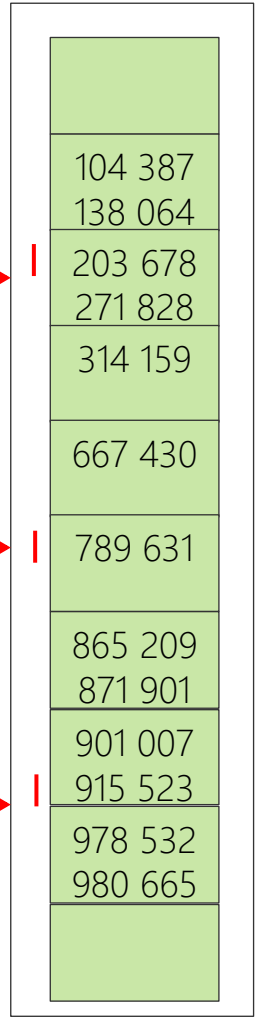
3. Concurrency

- So far, we have looked into the performance of a single SQL (and we have assumed it is running alone on the database).
- In real world transaction systems, that is not true.
- «Real world transaction systems» include bank systems, concert ticket ordering systems, netshop systems, warehousing systems, airplane ticketing systems ++++

- Two factors to consider regarding concurrency:
    - Isolation Level
    - Clustering in a table

Our understanding of database traffic

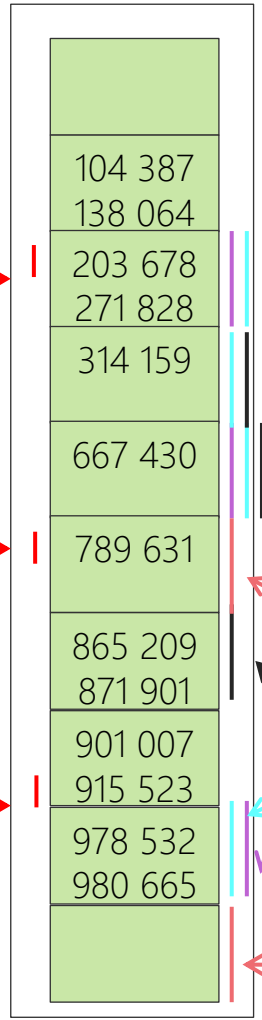
```
     IF  AIBRETRN EQUAL ZEROES
         CONTINUE
     ELSE
         display 'Klarer ikke deallokere PSB i S8212468 ' ws-tid
         MOVE '90' TO RETUR-KODE IN P8212468-IO (1)
         MOVE 'KLARER IKKE DEALLOKERE PSB' TO RETUR-REF-PUNKT(1)
         MOVE AIBRETRN TO WS-DIV-KVOT
         PERFORM X000-LAG-HEX-TALL
         DISPLAY 'DPSB AIBRETRN = X' WS-HEX-TALL
         MOVE AIBREASN TO WS-DIV-KVOT
         PERFORM X000-LAG-HEX-TALL
         DISPLAY 'DPSB AIBREASN = X' WS-HEX-TALL
         PERFORM X200-feil
      END-IF

   else
      initialize T-TEMP-FELT
      MOVE '96' TO T-RESP-KODE
      PERFORM B220-INSERT-PAA-T-BELOP
      PERFORM B230-OPEN-TEMP-BELOP-C
   END-IF

  IF C-OPEN-CURSOR-NEI
     EXEC SQL
          OPEN T-SP-STATUS-C
     END-EXEC
     IF SQLSTATE = ZERO
        CONTINUE
     ELSE
        DISPLAY 'FEIL I S8212468, KLARER IKKE åPNE CURSOR 2 !'
        DISPLAY 'SQLSTATE: ' SQLSTATE
        DISPLAY 'SQLCODE : ' SQLCODE
     END-IF
  END-IF
    .
B300-AVSLUTT-END.
    EXIT.

X000-LAG-HEX-TALL SECTION.
*****************************************************************

*****************************************************************
    PERFORM VARYING WS-HEX-IX FROM 8 BY -1 UNTIL WS-HEX-IX = ZERO
       DIVIDE WS-DIV-KVOT BY 16
       GIVING WS-DIV-KVOT
       REMAINDER WS-DIV-REST
       ADD 1 TO WS-DIV-REST 0  001120
       MOVE WS-HEX-SIFRE (WS-DIV-REST:1) TO
            WS-HEX-TALL (WS-HEX-IX:1)
    END-PERFORM
    .
X000-LAG-HEX-TALL-END.
    EXIT.

X100-FEIL-DB2 SECTION.
*****************************************************************
 Displayer feilmelding dersom feil i DB2, samt skrier på resulta.
*****************************************************************
    CALL 'DSNTIAR' USING SQLCA ERROR-MESSAGE ERROR-TEXT-LEN

    MOVE '99'            TO UT-RESULTAT
    MOVE RETUR-REF-PUNKT TO UT-REFERANSE
    MOVE 'x100-feil-db2' TO UT-REFERANSE
    STRING ' TIDSPUNKT FOR FEIL: ' DELIMITED BY SIZE
           WS-TID                  DELIMITED BY SIZE
                                   INTO UT-MELDING
```

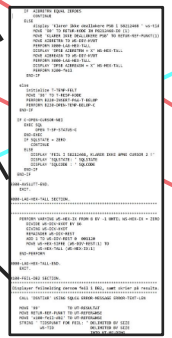


Our understanding

How it is in real life...

```
    IF  AIBRETRN EQUAL ZEROES
        CONTINUE
    ELSE
        display 'Klarer ikke deallokere PSB i S8212468 ' ws-tid
        MOVE '90' TO RETUR-KODE IN P8212468-IO (1)
        MOVE 'KLARER IKKE DEALLOKERE PSB' TO RETUR-REF-PUNKT(1)
        MOVE AIBRETRN TO WS-DIV-KVOT
        PERFORM X000-LAG-HEX-TALL
        DISPLAY 'DPSB AIBRETRN = X' WS-HEX-TALL
        MOVE AIBREASN TO WS-DIV-KVOT
        PERFORM X000-LAG-HEX-TALL
        DISPLAY 'DPSB AIBREASN = X' WS-HEX-TALL
        PERFORM X200-feil
     END-IF

   else
      initialize T-TEMP-FELT
      MOVE '96' TO T-RESP-KODE
      PERFORM B220-INSERT-PAA-T-BELOP
      PERFORM B230-OPEN-TEMP-BELOP-C
   END-IF

  IF C-OPEN-CURSOR-NEI
     EXEC SQL
          OPEN T-SP-STATUS-C
     END-EXEC
     IF SQLSTATE = ZERO
        CONTINUE
     ELSE
        DISPLAY 'FEIL I S8212468, KLARER IKKE åPNE CURSOR 2 !'
        DISPLAY 'SQLSTATE: ' SQLSTATE
        DISPLAY 'SQLCODE : ' SQLCODE
     END-IF
  END-IF
     .
B300-AVSLUTT-END.
    EXIT.

X000-LAG-HEX-TALL SECTION.
*****************************************************************

*****************************************************************
    PERFORM VARYING WS-HEX-IX FROM 8 BY -1 UNTIL WS-HEX-IX = ZERO
       DIVIDE WS-DIV-KVOT BY 16
       GIVING WS-DIV-KVOT
       REMAINDER WS-DIV-REST
       ADD 1 TO WS-DIV-REST 0  001120
       MOVE WS-HEX-SIFRE (WS-DIV-REST:1) TO
            WS-HEX-TALL (WS-HEX-IX:1)
    END-PERFORM
     .
X000-LAG-HEX-TALL-END.
    EXIT.

X100-FEIL-DB2 SECTION.
*****************************************************************
 Displayer feilmelding dersom feil i DB2, samt skriver på resulta.
*****************************************************************
    CALL 'DSNTIAR' USING SQLCA ERROR-MESSAGE ERROR-TEXT-LEN

    MOVE '99'            TO UT-RESULTAT
    MOVE RETUR-REF-PUNKT TO UT-REFERANSE
    MOVE 'x100-feil-db2' TO UT-REFERANSE
    STRING ' TIDSPUNKT FOR FEIL: ' DELIMITED BY SIZE
           WS-TID                  DELIMITED BY SIZE
                                   INTO UT-MELDING
```





3. Concurrency - Isolation levels

- Isolation levels is the most misunderstood concept in database theory.
- It is always presented as "you run an SQL….and later on you want to run the same SQL and get the same result….." That explanation does not help you at all.
- IBM DB2 uses this terminology:
    - repeatable read (RR)
    - read stability (RS)
    - cursor stability (CS) <<<<< THIS IS THE NORMAL ISOLATION LEVEL.
    - uncommitted read (UR)

The definitions and examples in all documentation is – to say the least – confusing.

But I found a definition from Steve Thomas, one of the IDUG organisers and a DB2 guru:

Isolation levels are about SHARE LOCKS (read locks), how many locks to take, and how long to hold these locks.

All the 3 upper levels take EXCLUSIVE locks by all UPDATES and keep them till commit.

But the differences are for the SHARE LOCKS.

DNB

3. Concurrency - Isolation levels

RR – Repeatable Read. Locks - with SHARE locks - ALL ROWS it peeks into to find the final qualifying (the selected) rows. If it has to scan 50 000 rows to find 10 qualifying rows, all 50 000 rows are locked. Does not allow INSERTS into a table where you do RR operations!

RR keeps all SHARE LOCKS till COMMIT. (Use only in very special cases).

RS – Read Stability. Much the same as RR, but locks only the rows that qualify. If it had to scan 50 000 rows to find 10 qualifying rows, the qualifying 10 rows are locked.

RS keeps all SHARE LOCKS till COMMIT. (Use only in very special cases).

CS – Cursor Stability: CS frees SHARE LOCK as it works through the data, it does not wait till COMMIT. (Is the one Isolation level to use, except for very, very special cases).

UC – Uncommited Read: Does not set SHARE lock when it reads a row. Meaning – it can read a row held with an EXCLUSIVE lock (under update, not committed).

DNB

3. Concurrency - Locking and lock escalation

Lock Modes for Page and Row Locks

- 3 Lock Modes available at Page and Row level
  - Application SQL determines which is used
- S (Shared)
  - Select, Fetch, Some Open Cursor
- X (Exclusive)
  - Any SQL that modifies data
  - Insert, Update, Delete, Merge
- U (Update)
  - Fetch for Update
  - Promoted to an X lock when the data is modified
  - Helps prevents deadlocks

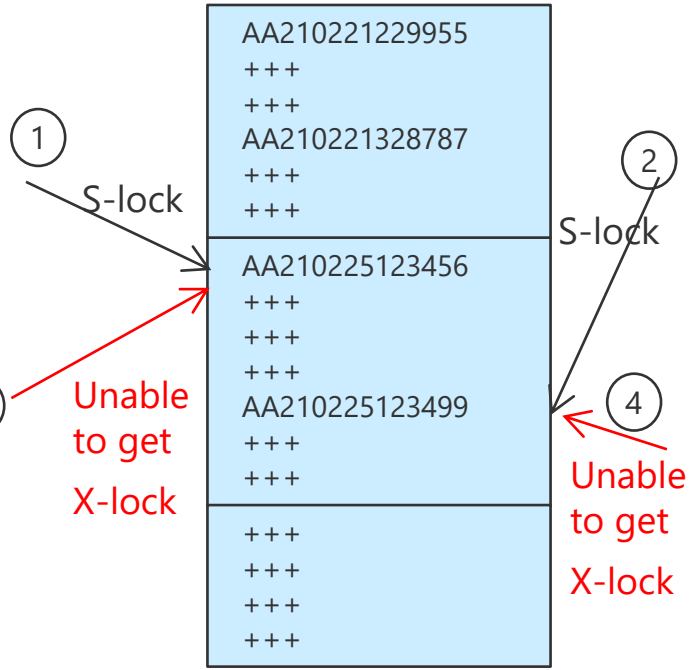
	S	U	X
S	OK	OK	NOK
U	OK	NOK	NOK
X	NOK	NOK	NOK

3. Concurrency - SQL locking with Read Stability

PROCESS A

```
SELECT * FROM  CUSTOMER_INFO
WHERE CASEID = 'AA210225123456'
  AND CUSTOMERID = 148267;
```

```
DELETE FROM CUSTOMER_INFO
WHERE CASEID = 'AA210225123456'
  AND CUSTOMERID = 148267 ;
```



PROCESS B

```
SELECT * FROM  CUSTOMER_INFO
WHERE CASEID = 'AA210225123499'
  AND CUSTOMERID = 314159;
```

```
DELETE FROM CUSTOMER_INFO
WHERE CASEID = 'AA210225123499'
  AND CUSTOMERID = 314159;
```

If SELECT in process A comes after the SELECT in process B, but before the DELETE in process A, we have a deadlock. If this run with Cursor Stability, no deadlocks will occur here.

3. Concurrency - Clustering of data

The clustering of a table tells us in what physical sequence the data is stored in the table.

Remember – the DB operates on pages. It will read pages, lock pages (or rows on pages), it will write pages.

In general – a transaction would prefer to have it's page alone.

There is no right or wrong clustering.

But it may affect locking and ultimately performance.

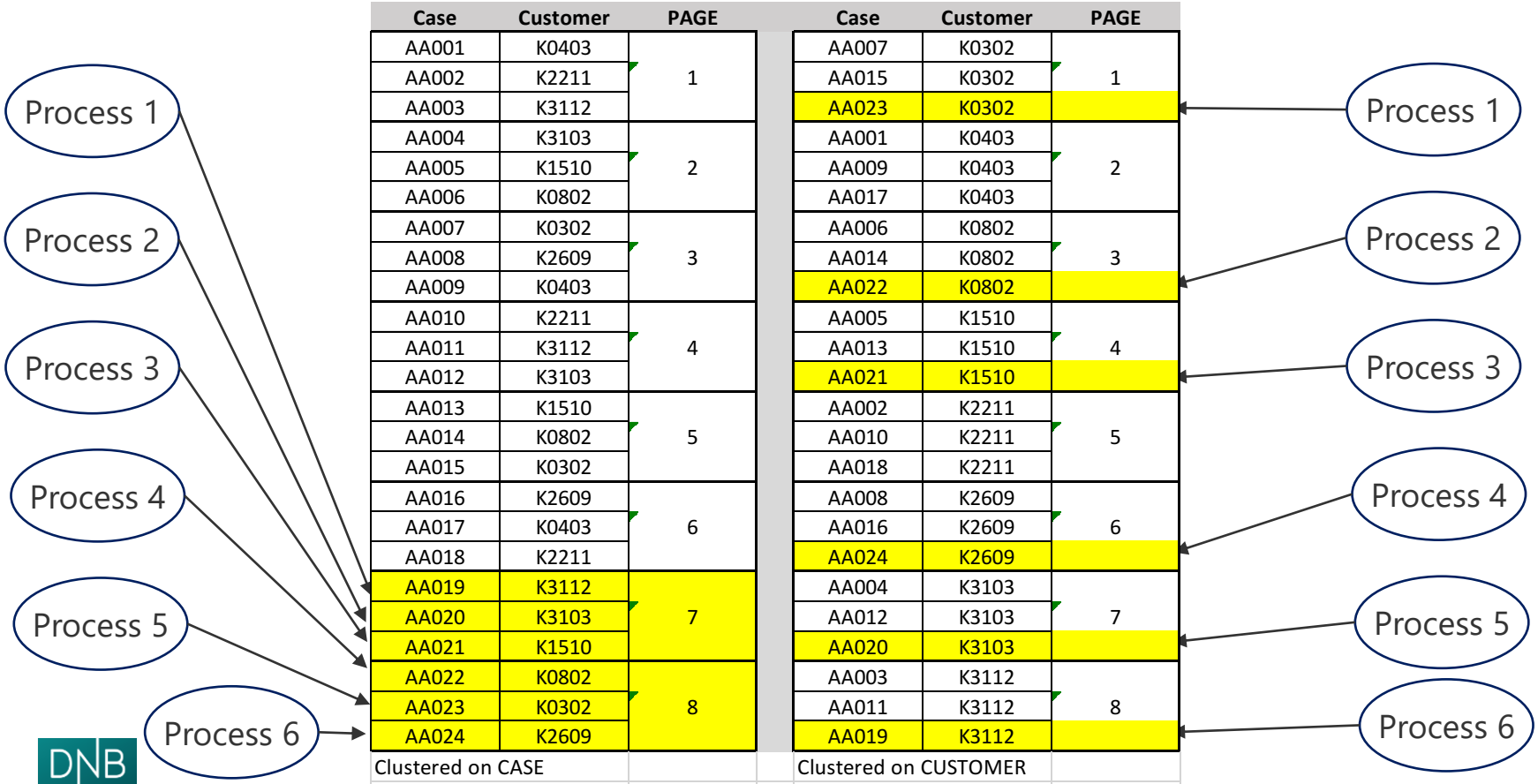
CaseID	CustomerID	PAGE
AA001	K0403	
AA002	K2211	1
AA003	K3112	
AA004	K3103	
AA005	K1510	2
AA006	K0802	
AA007	K0302	
AA008	K2609	3
AA009	K0403	
AA010	K2211	
AA011	K3112	4
AA012	K3103	
AA013	K1510	
AA014	K0802	5
AA015	K0302	
AA016	K2609	
AA017	K0403	6
AA018	K2211	
AA019	K3112	
AA020	K3103	7
AA021	K1510	
AA022	K0802	
AA023	K0302	8
AA024	K2609	

Clustered on CaseID

CaseID	CustomerID	PAGE
AA007	K0302	
AA015	K0302	1
AA023	K0302	
AA001	K0403	
AA009	K0403	2
AA017	K0403	
AA006	K0802	
AA014	K0802	3
AA022	K0802	
AA005	K1510	
AA013	K1510	4
AA021	K1510	
AA002	K2211	
AA010	K2211	5
AA018	K2211	
AA008	K2609	
AA016	K2609	6
AA024	K2609	
AA004	K3103	
AA012	K3103	7
AA020	K3103	
AA003	K3112	
AA011	K3112	8
AA019	K3112	

Clustered on CustomerID

3. Concurrency - Clustering of data – why it matters

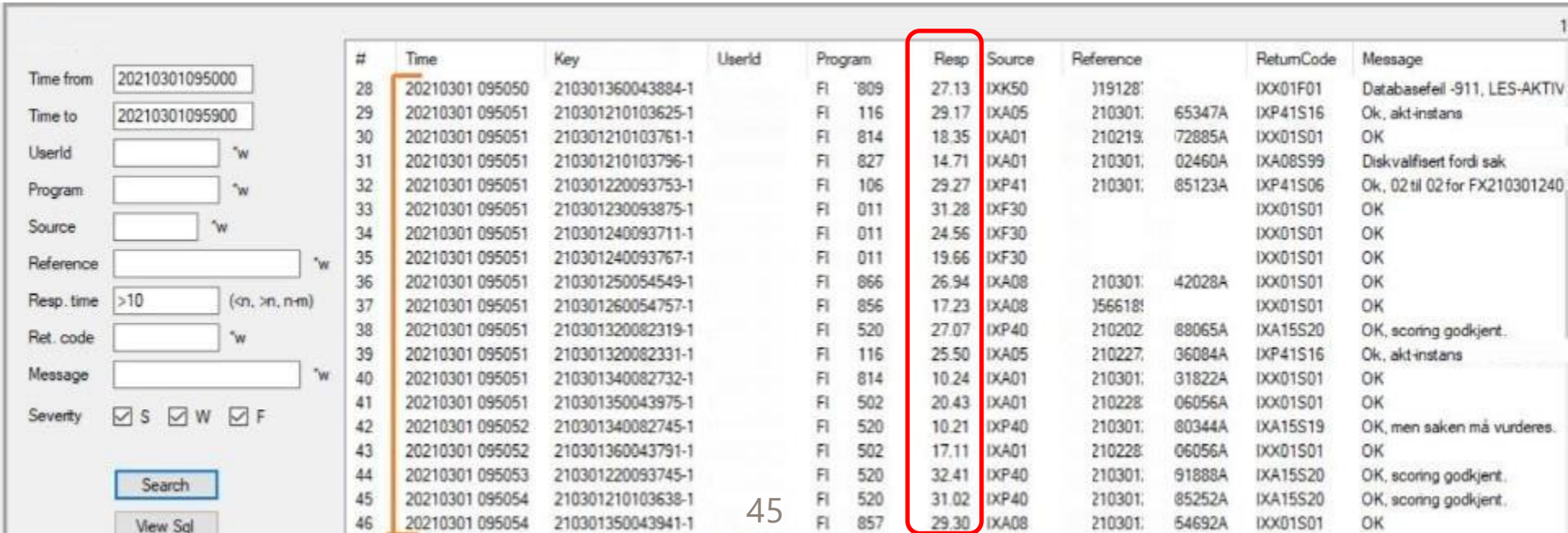


Case	Customer	PAGE
AA001	K0403	
AA002	K2211	1
AA003	K3112	
AA004	K3103	
AA005	K1510	2
AA006	K0802	
AA007	K0302	
AA008	K2609	3
AA009	K0403	
AA010	K2211	
AA011	K3112	4
AA012	K3103	
AA013	K1510	
AA014	K0802	5
AA015	K0302	
AA016	K2609	
AA017	K0403	6
AA018	K2211	
AA019	K3112	
AA020	K3103	7
AA021	K1510	
AA022	K0802	
AA023	K0302	8
AA024	K2609	
Clustered on CASE		
High deadlock risk		

Case	Customer	PAGE
AA007	K0302	
AA015	K0302	1
AA023	K0302	
AA001	K0403	
AA009	K0403	2
AA017	K0403	
AA006	K0802	
AA014	K0802	3
AA022	K0802	
AA005	K1510	
AA013	K1510	4
AA021	K1510	
AA002	K2211	
AA010	K2211	5
AA018	K2211	
AA008	K2609	
AA016	K2609	6
AA024	K2609	
AA004	K3103	
AA012	K3103	7
AA020	K3103	
AA003	K3112	
AA011	K3112	8
AA019	K3112	
Clustered on CUSTOMER		
Low deadlock risk		

3. Concurrency - Clustering of data – why it matters

- Isolation Levels and clustering are interesting, but this is about performance.
- What do Isolation Levels and clustering have to do with performance?
- Well, think of the previous page…. What happens when we have too many transactions fighting to access the same table pages or rows?



#	Time	Key	UserId	Program	Resp	Source	Reference		ReturnCode	Message
28	20210301 095050	210301360043884-1		FI '809	27.13	IXK50	)19128'		IXX01F01	Databasefeil -911, LES-AKTIV
29	20210301 095051	210301210103625-1		FI 116	29.17	IXA05	210301:	65347A	IXP41S16	Ok, akt-instans
30	20210301 095051	210301210103761-1		FI 814	18.35	IXA01	210219:	72885A	IXX01S01	OK
31	20210301 095051	210301210103796-1		FI 827	14.71	IXA01	210301:	02460A	IXA08S99	Diskvalifisert fordi sak
32	20210301 095051	210301220093753-1		FI 106	29.27	IXP41	210301:	85123A	IXP41S06	Ok, 02 til 02 for FX210301240
33	20210301 095051	210301230093875-1		FI 011	31.28	IXF30			IXX01S01	OK
34	20210301 095051	210301240093711-1		FI 011	24.56	IXF30			IXX01S01	OK
35	20210301 095051	210301240093767-1		FI 011	19.66	IXF30			IXX01S01	OK
36	20210301 095051	210301250054549-1		FI 866	26.94	IXA08	210301:	42028A	IXX01S01	OK
37	20210301 095051	210301260054757-1		FI 856	17.23	IXA08	)56618!		IXX01S01	OK
38	20210301 095051	210301320082319-1		FI 520	27.07	IXP40	210202	88065A	IXA15S20	OK, scoring godkjent.
39	20210301 095051	210301320082331-1		FI 116	25.50	IXA05	210227:	36084A	IXP41S16	Ok, akt-instans
40	20210301 095051	210301340082732-1		FI 814	10.24	IXA01	210301:	31822A	IXX01S01	OK
41	20210301 095051	210301350043975-1		FI 502	20.43	IXA01	210228:	06056A	IXX01S01	OK
42	20210301 095052	210301340082745-1		FI 520	10.21	IXP40	210301:	80344A	IXA15S19	OK, men saken må vurderes.
43	20210301 095052	210301360043791-1		FI 502	17.11	IXA01	210228:	06056A	IXX01S01	OK
44	20210301 095053	210301220093745-1		FI 520	32.41	IXP40	210301:	91888A	IXA15S20	OK, scoring godkjent.
45	20210301 095054	210301210103638-1		FI 520	31.02	IXP40	210301:	85252A	IXA15S20	OK, scoring godkjent.
46	20210301 095054	210301350043941-1		FI 857	29.30	IXA08	210301:	54692A	IXX01S01	OK

4. New directions

The traditional relational database is very good for certain operations, and not so good for others……

Excels in:

- Finding one or few rows via indexes. (That is often pre-defined searches).
- Transactional handling, for instance flight booking, concert ticketing.
- Storing structured data in a space efficient way.

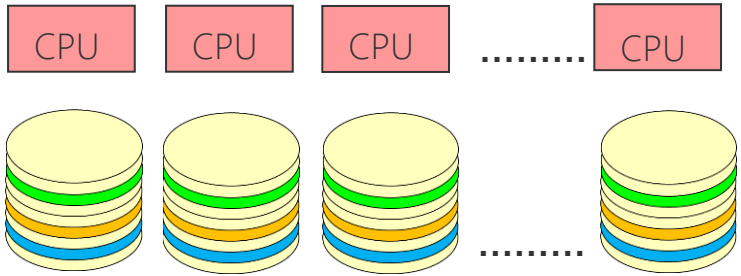
Not so good in:

- Searching through large data volumes with joins through multiple tables. (Analytics).
- Storing less structured data. (Comments in a blog, or Facebook).
- Storing and retrieving large volumes of read only data.

And many applications do not need consistent and controlled data…

4. New directions
SQL accelerators - divide and conquer

Lots of CPUs
with their own
disc rack



TableA
TableB
TableC

Tables are distributed (striped) over a large number of disc racks.

All SQLs are executed as table scans.
Data for all join tables are fetched in 1 scan.

The tables are copies of the actual operational database

Different strategies for synchronisation:
- Daily complete load.
- Continous synch

Advantage: Speed.

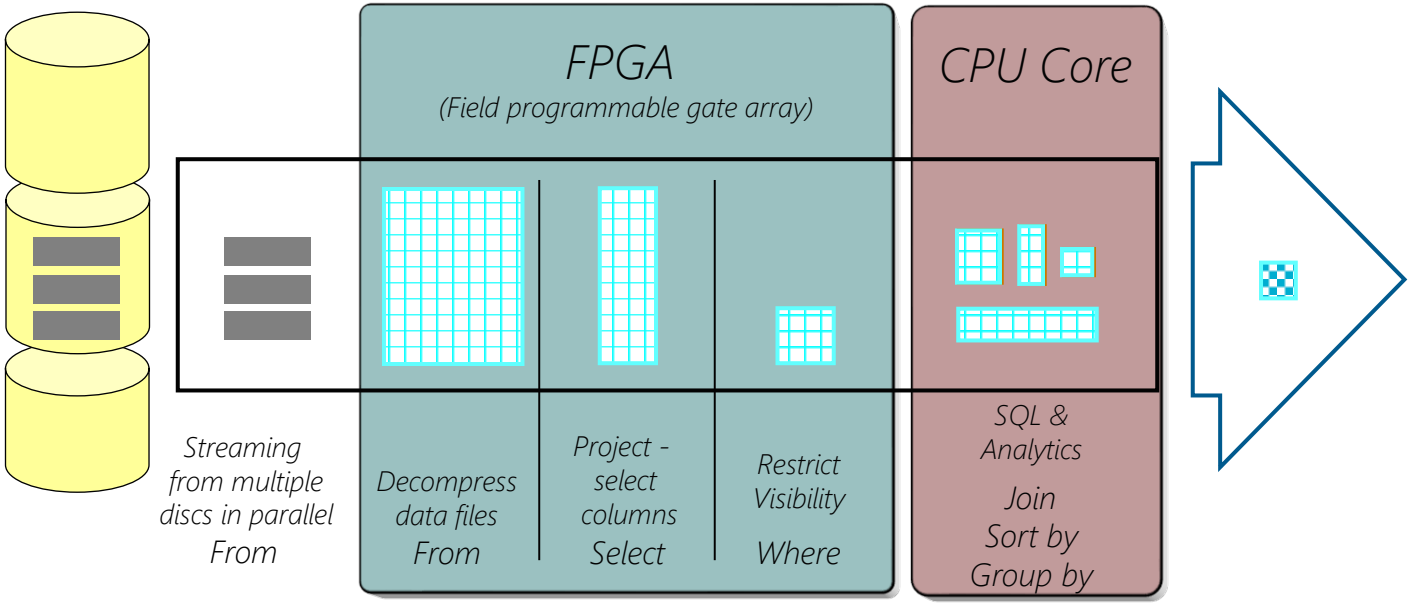
Before	-->>	After
16 hours		39 seconds
1 hour		8 seconds
∞		1:30 min:sec
16 hours		24 minutes

Disadvantage: Speed.

0,001 sec	1-2 seconds

I work currently with a SQL accellerator with 80 CPUs....though the range is
40 – 80 – 140 – 280 – 560 – 1120
CPUs and corresponding disc racks

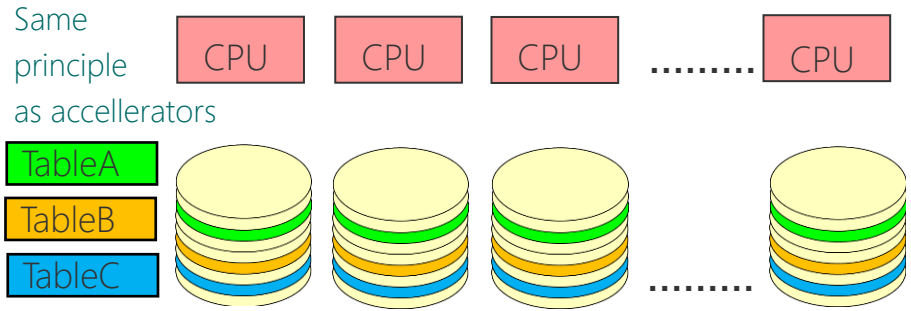
4. New directions
Accelerator Stream Processing




```
Select Etternavn, Fornavn, Bostedskommune, Fdato, Kjoenn, Ufoerestatus, count(*)
From Personregister.Tperson Left join Pensjon.Ufoere on Tperson.personid = Ufoere.personid Where Fdato < '01.01.1995'
And Bostedskommune in ('Valldal', 'Rauma, 'Vestnes)
And Ufoerestatus in ("PERM", "TEMP", "GRAD",
Group by Bostedskommune, Year(Fdato), Kjoenn
Order by Bostedskommune, Year(Fdato), Kjoenn
```

DNB

4. New directions
NoSQL – or NOSQL?



Not all data you want to store is highly structured and tabular.

Not all data is strictly transactional and must be persisted in an all or nothing strategy.

Not all data is of a type where you need 100% consistency control.

Not all data is write / update / delete. A lot is write once, read often.

No SQL or
Not Only SQL

Many different principles and solutions.

http://nosql-database.org

Examples:
Hadoop, Cassandra
MongoDB, GenieDB

Traditional applications where NoSQL may help: Payments archive in bank. Electricity metering. And many more, the industry is held back by traditional thinking.....

New applications: Social medias, mass data monitoring, data which is not updated, rather reentered (exam results? And many others). And where strict transactional control is not the issue.

4. New directions: We do not always need transactional consistency

Not all applications need the transactional consistency which is the hallmark of a relational database.

Think of air travel reservation: You reserve an air travel. You reserve a seat on an aircraft. While you do that, no other person may book that seat. When you pay, the seat is confirmed, and blocked for others.

And for full price tickets – you may change this order, meaning modifying the data. Maybe you change the ticket to another date.

A lot of our thinking stems from this….. But other applications do not have these requirements: Think of the position log on your smartphone. If you are at one street corner at a certain time, you do not block that position for other persons at the same time.

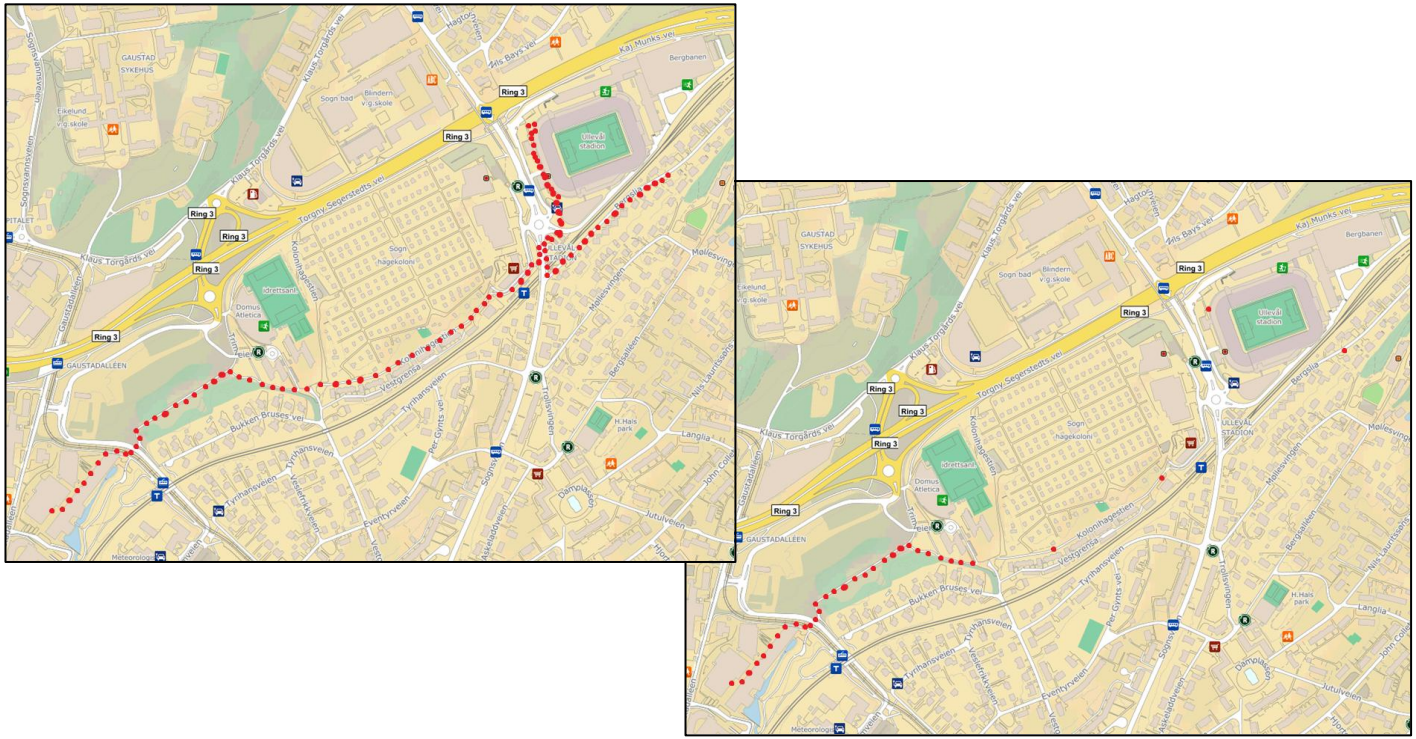
You do not change your historical positions. It is store once, read many times.

And – unlike air tickets – it is no major crises if you loose 10 or 100 data points.

This means: A lot of the complexities in traditional relation database IS NOT NEEDED!!!

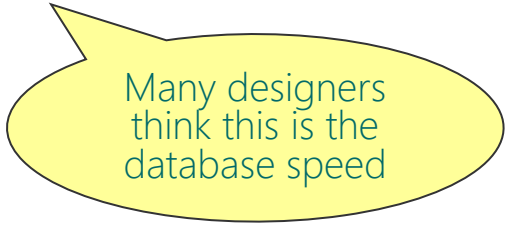
DNB

4. New directions. We do not always need transactional consistency

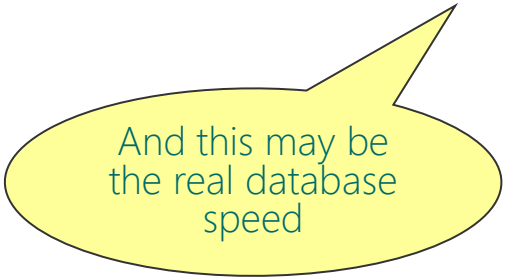


Summary

0.000 sec x 6,000,000 = 0 sec



0.012 sec x 6,000,000 = 72000 sec = 20 hours



Questions?

You are welcome to ask questions any time through November 2024.

Please send an email to audun “dot” faaberg “at” dnb “dot” no

(Change the words in “ ” and you get the mail address).