

The background of the slide features a collage of hexagonal images. These images depict various scenes: large industrial pipes in the upper left; modern residential buildings with illuminated windows at night; a bridge spanning a body of water; and a close-up view of a bridge's structural elements. The hexagonal pattern creates a tessellated effect across the entire slide.

Real life case studies

UiO 22.10.2025

Do not read data over and over again - advanced

Here we come back to what we discussed earlier: Do not read data over and over again.

Though in this case it is more hidden.

This case demonstrates best practise when doing mass updates in large database tables.

Problem

16.33.50 JOB007

--TIMINGS (MINS.)--

16.33.50 JOB007 -JOBNAME PROCSTEP

RC EXCP CPU SRB CLOCK

16.33.50 JOB007 -P007DBCH DBCHNG01

00 102 **8.15** .00 **121.56**

18.35.17 JOB007 IEF404I P007DBCH - ENDED - TIME=08.35.17

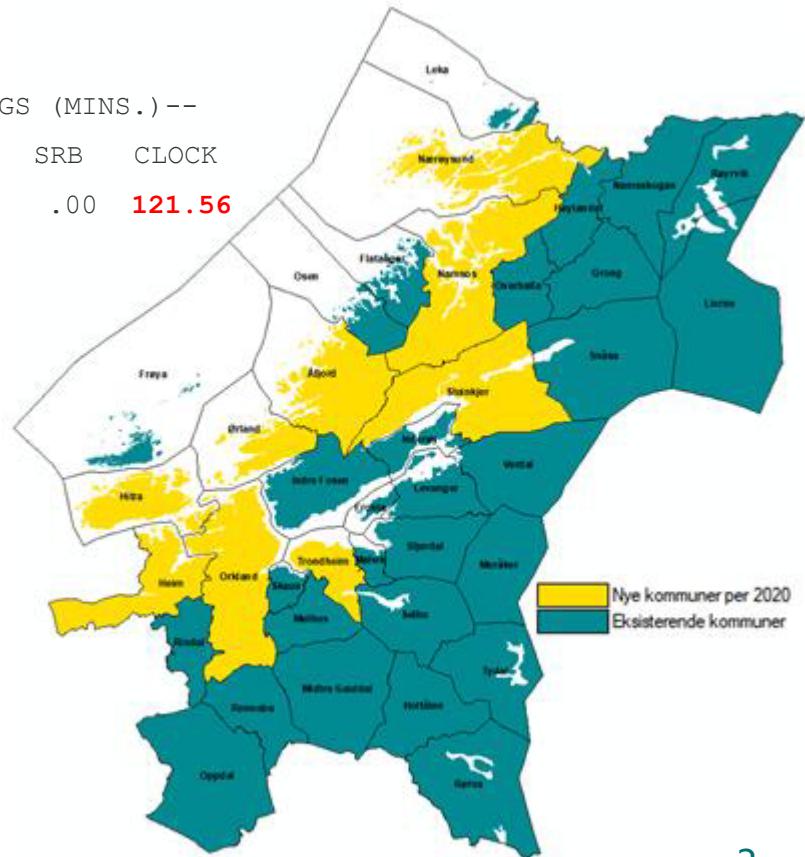
18.35.17 JOB007 -P007DBCH ENDED.

BATCH TOTAL CPU TIME = 8.15

TOTAL ELAPSED TIME = 121.56

18.35.17 JOB007 ÅHASP395 P007DBCH ENDED - RC=0000

121 minutes is just above 2 hours



The real problem

19.16.50 JOB007

--TIMINGS (MINS.)--

19.16.50 JOB007 -JOBNAME PROCSTEP

RC	EXCP	CPU	SRB	CLOCK
----	------	-----	-----	-------

19.16.50 JOB007 -P007DBCH DBCHNG01

00	102	160	.00	933
----	-----	------------	-----	------------

08.47.58 JOB007 IEF404I P007DBCH - ENDED - TIME=08.35.17

08.47.58 JOB007 -P007DBCH ENDED.

BATCH TOTAL CPU TIME = **160**

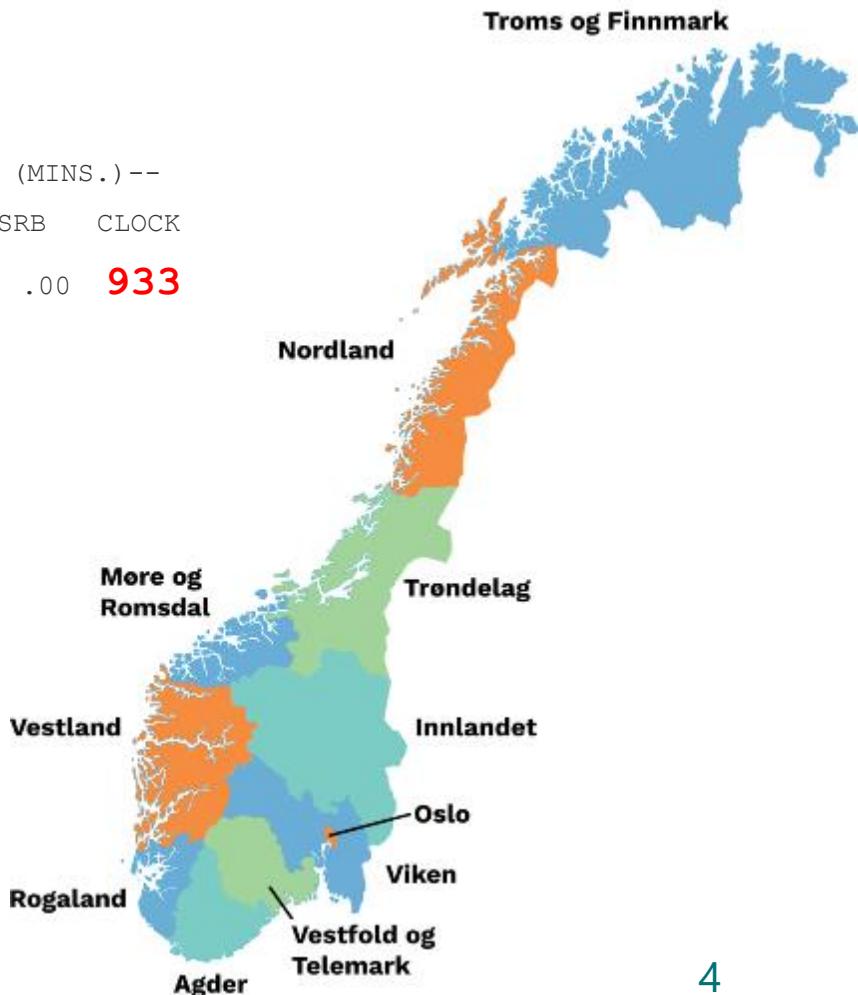
TOTAL ELAPSED TIME = **933**

08.47.58 JOB007 ÅHASP395 P007DBCH ENDED - RC=0000

933 minutes is just short of 16 hours.

We have 4 hours available.

So – make this run in 4 hours.



So, can an SQL monitor tell me anything?

SQL Text	Exec Count	CPU	Elapsed
UPDATE SERVICE_GRANTED SET KNR_MUN = :H1 WHERE..	49	03:00	54:03
UPDATE SERVICE_PAID SET KNR_MUN = :H1 WHERE..	49	02:41	43:05
UPDATE SERVICE_ITEM SET KNR_MUN = :H1 WHERE..	49	01:25	22:47
UPDATE SERVICE_APPROVAL SET KNR_MUN = :H1 WHERE..	49	00:17	1:09
UPDATE DECISION SET UNIT1 = :H1 WHERE UNIT1 = :H2	49	00:04	0:14
UPDATE DECISION SET UNIT2 = :H1 WHERE UNIT2 = :H2	49	00:04	0:13
Total		07:31	2:01:31

All SQLs in the program.

2:01:31 - Elapsed time in DB2

2:01:56 - For the whole job.

UPDATE SERVICE_GRANTED SET KNR_MUN = :H1 WHERE KNR_MUN = :H2

Elapsed	AvgX Elapsed	CPU	AvgX CPU	IO wait	AvgX IO Wait	GetPages	AvgX GetPages
54:03.825285	1:06.200516	3:00.845056	3,690715	51:50.478866	1:03.479160	14 779 820	301 629

The top SQL

Table	Rows	Pages
SERVICE_GRANTED	26 739 936	285 137

DB2 table information
from DB2ADM

How the program works.

SERVICE_GRANTED

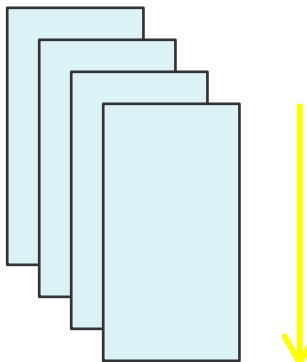
PERSON_NR	KNR_MUN	other info
060234	1208	lorem ipsum
081269	1210	dolor sit
050564	1603	amet consectetur
120123	1604	adipiscing elit
250302	1895	sed do eiusmod
310888	1603	tempor incididunt
170514	1603	ut labore et
070605	1751	dolore magna
121179	1802	aliqua ut enim
220684	1604	ad minim veniam
281089	1603	quis nostrud



OLD-KNR	NEW-KNR
1603	5081
1604	5082
1605	5083
1607	5084
++++	++++

49 rows in the sequential input file

UPDATE SERVICE_GRANTED
SET KNR_MUN = :new-knr
WHERE KNR_MUN = :old-knr
(repeat for each line in input file)



49 passes on SERVICE_GRANTED

And 49 passes on 4 more tables.

How the program works.

SERVICE_GRANTED

PERSON_NR	KNR_MUN	other info
060234	1208	lorem ipsum
081269	1210	dolor sit
050564	1603	amet consectetur
120123	1604	adipiscing elit
250302	1895	sed do eiusmod
310888	1603	tempor incididunt
170514	1603	ut labore et
070605	1751	dolore magna
121179	1802	aliqua ut enim
220684	1604	ad minim veniam
281089	1603	quis nostrud

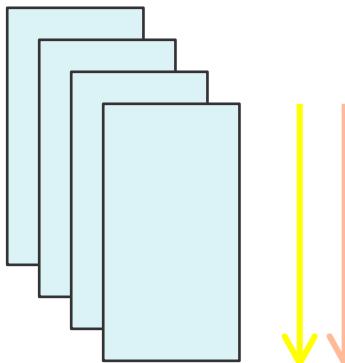


OLD-KNR	NEW-KNR
1603	5081
1604	5082
1605	5083
1607	5084
++++	++++

49 rows in the sequential input file

UPDATE SERVICE_GRANTED
SET KNR_MUN = :new-knr
WHERE KNR_MUN = :old-knr
(repeat for each line in input file)

49 passes on SERVICE_GRANTED



And 49 passes on 4 more tables.

Modified SQL

SERVICE_GRANTED

PERSON_NR	KNR_MUN	other info
060234	1208	lorem ipsum
081269	1210	dolor sit
050564	1603	amet consectetur
120123	1604	adipiscing elit
250302	1895	sed do eiusmod
310888	1603	tempor incididunt
170514	1603	ut labore et
070605	1751	dolore magna
121179	1802	aliqua ut enim
220684	1604	ad minim veniam
281089	1603	quis nostrud

The SQL is run only once.

The WHERE part picks out the rows to update. The SET part takes one and one row, look up in the temp table, and updates.

SQL-CHG-HSG

KNR_TAB, now a DB2 table.

OLD_KNR	NEW_KNR
1603	5081
1604	5082
1605	5083
1607	5084
++++	++++

49 rows a small DB2 table

UPDATE SERVICE_GRANTED SG

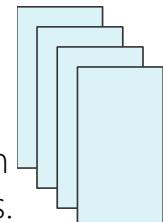
SET SG.KNR_MUN =

(SELECT TT.NEW_KNR
FROM KNR_TAB TT
WHERE SG.KNR_MUN = TT.OLD_KNR)

WHERE SG.KNR_MUN IN

(SELECT TX.OLD_KNR
FROM KNR_TAB TX)

And 1 pass on 4 more tables.



Modified SQL

SERVICE_GRANTED

PERSON_NR	KNR_MUN	other info
060234	1208	lorem ipsum
081269	1210	dolor sit
050564	1603	amet consectetur
120123	1604	adipiscing elit
250302	1895	sed do eiusmod
310888	1603	tempor incididunt
170514	1603	ut labore et
070605	1751	dolore magna
121179	1802	aliqua ut enim
220684	1604	ad minim veniam
281089	1603	quis nostrud

The SQL is run only once.

The WHERE part picks out the rows to update. The SET part takes one and one row, look up in the temp table, and updates.

SQL-CHG-HSG

KNR_TAB, now a DB2 table.

OLD_KNR	NEW_KNR
1603	5081
1604	5082
1605	5083
1607	5084
++++	++++

49 rows a small DB2 table

UPDATE SERVICE_GRANTED SG

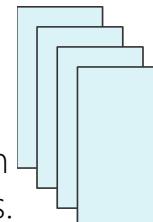
SET SG.KNR_MUN =

(SELECT TT.NEW_KNR
FROM KNR_TAB TT
WHERE SG.KNR_MUN = TT.OLD_KNR)

WHERE SG.KNR_MUN IN

(SELECT TX.OLD_KNR
FROM KNR_TAB TX)

And 1 pass on 4 more tables.



Results

Original solution				New solution		
SQL Text	Exec Count	CPU	Elapsed	Exec Count	CPU	Elapsed
UPDATE SERVICE_GRANTED SET TKNR_MUN = :H1..	49	03:00	54:03	1	01:58	04:03
UPDATE SERVICE_PAID SET TKNR_MUN = :H1....	49	02:41	45:05	1	01:45	03:36
UPDATE SERVICE_ITEM SET TKNR_MUN = :H1.....	49	01:25	22:47	1	00:51	02:06
UPDATE SERVICE_APPROVAL SET TKNR_NUM = :H1..	49	00:17	01:09	1	00:15	00:21
UPDATE DECISION SET UNIT1 = : H1 WHERE UNIT1 = :H2	49	00:04	00:14	1	00:05	00:08
UPDATE DECISION SET UNIT2 = : H1 WHERE UNIT2 = :H2	49	00:04	00:13	1	00:04	00:06
Sum	294	07:31	1:52:31	6	04:58	10:20

Know your problem
and your algorithm.
And your tool.

1 execution in the new SQL
does the same work as 49 in
the old - or 378 executions for
the whole country

Learning points

1. Look at the numbers – and try to understand them.
2. Do not just code straight ahead, the way your nose directs you.....
3. Find the cost driving issue, and try to avoid repeating that over and over again.



Case B: Correlated subselects when preparing data

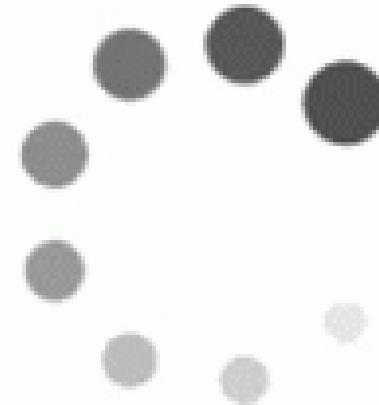
In the previous case, I showed you how to improve a large data conversion 10-100 times.

This time I will up the ante somewhat

I often use this example to explain people that not all problems can just be distributed to more processing nodes in a modern environment.

Some problems just mutate into huge impossible problem if you handle them poorly.

Code 6&7 – washing test data



Background: Code 6 & 7

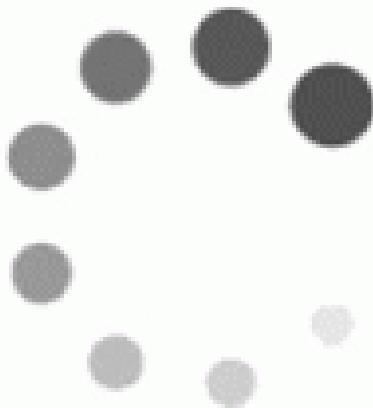


Persons with code 6 have their address removed from public registers. Only a handful of authorised persons there know their actual address.

Persons with code 7 have a secret address, though public registers have the address stored. They are not allowed to give out that information to any private persons or institutions.

When using live data in test environments, it is a requirement to remove all these persons from test data. (In order to reduce the risk of a leak, either intended or unintended).

Problem statement



Delete script had 42 SQLs.

The first SQL in the script took 40 hours to run, and the full script took 10 days (240 hours).

This should be used when building new test environment, something which was done quite often.

And every 3 month we anyhow rebuild the acceptance test environment with a copy from production.

So, what should have been a fairly simple and straight forward step in the building of an environment became – “wait for 10 days”.

This actually meant – it is impossible to build large test environments.

SQL

```
DELETE FROM T_AGREEMENT TA
WHERE EXISTS (SELECT 1 FROM T_RELATIONSHIP TR
              WHERE TR.RELATIONSHIP_ID = TA.RELATIONSHIP_ID
              AND EXISTS (SELECT 1 FROM T_REMOVELIST P67
                           WHERE P67.PERSON_ID = TR.PERSON_ID))
```

Table name	Rows
T_AGREEMENT	3 000 000
T_RELATIONSHIP	5 000 000
T_REMOVELIST	2 000

A correlated subselect cannot execute alone, it contains reference to the outer SQL (red arrows in example above). That means that the outer SQL has to call it one line at a time.

And here – in the outer SELECT there are no limitations before the SUBSELECT, meaning that each and every row must be sent into the next subselect. Here it finds 1 or more RELATIONSHIPS_IDs. And again, each of them must be sent into the bottom SQL for evaluation.

SQL

```
DELETE FROM T_AGREEMENT TA ←  
WHERE EXISTS (SELECT 1 FROM T_RELATIONSHIP TR  
    WHERE TR.RELATIONSHIP_ID = TA.RELATIONSHIP_ID →  
    AND EXISTS (SELECT 1 FROM T_REMOVELIST P67  
        WHERE P67.PERSON_ID = TR.PERSON_ID))
```

Table name	Rows
T_AGREEMENT	3 000 000
T_RELATIONSHIP	5 000 000
T_REMOVELIST	2 000

A correlated subselect cannot execute alone, it contains reference to the outer SQL (red arrows in example above). That means that the outer SQL has to call it one line at a time.

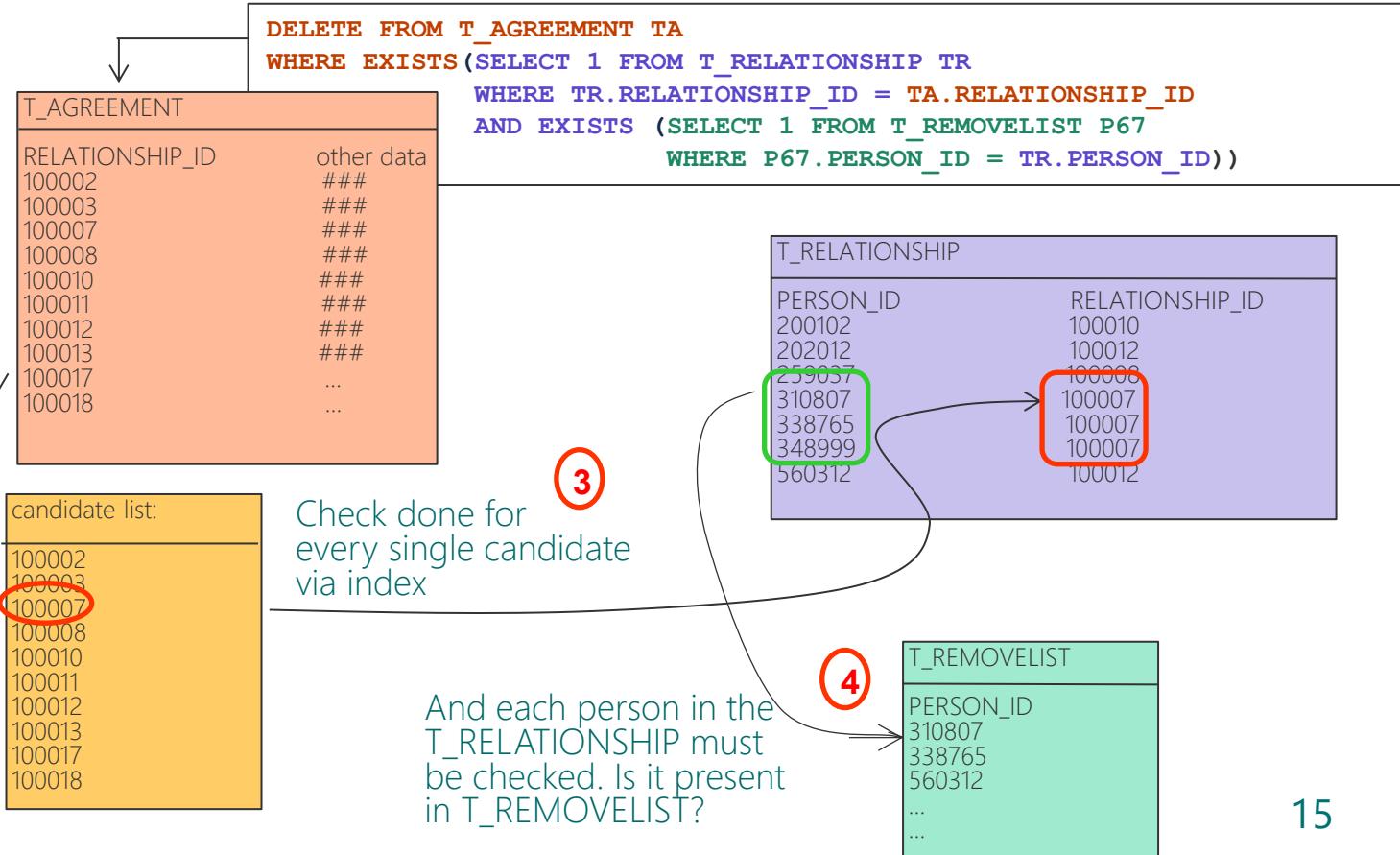
And here – in the outer SELECT there are no limitations before the SUBSELECT, meaning that each and every row must be sent into the next subselect. Here it finds 1 or more RELATIONSHIPS_IDs. And again, each of them must be sent into the bottom SQL for evaluation.

Execution of correlated subselect

①

Full table scan

SELECT RELATIONSHIP_ID
from T_AGREEMENT



Non-correlated subselect

```
DELETE FROM T_AGREEMENT TA
WHERE TA.RELATIONSHIP_ID IN
  (SELECT TR.RELATIONSHIP_ID
   FROM T_RELATIONSHIP TR, T_SPESREG_REMOVE P67
   WHERE WR.PERSON_ID = P67.PERSON_ID)
```

In the first subselect I changed the «WHERE EXISTS» to a «WHERE IN», and then it will get a list back from the underlying subselect. And the construction with a «WHERE EXISTS» in the second subselect, I just switched to a normal JOIN.

Now the inner subselect can execute ONCE with no input from the outer SQL. It will present a list to the outer sql, which will DELETE any rows which is referenced in the list of about 3000-4000 rows it gets back.

Execution of non-correlated subselect

①

Full table scan
SELECT PERSON_ID
from T_SPESREG_REMOVE

```
DELETE FROM T_AGREEMENT TA
WHERE TA.RELATIONSHIP_ID IN
  (SELECT TR.RELATIONSHIP_ID
   FROM T_RELATIONSHIP TR, T_REMOVELIST P67
   WHERE TR.PERSON_ID = P67.PERSON_ID)
```

②

Sets up a candidate list
(with all 2000 rows in the table)

T_REMOVELIST
PERSON_ID
310807
338765
560312
...
...

T_RELATIONSHIP	
PERSON_ID	RELATIONSHIP_ID
200102	100010
202012	100012
259037	100008
310807	100007
338765	100007
348999	100007
560312	100012

③

Join Each of the 2000 rows to a row in T_RELATIONSHIP. This is done via an index.

④

Then it creates a list of about 3000 RELATIONSHIP_ID to be DELETED

Candidate DELETE list:	
100007	
100012	
+++	
...	
...	

T AGREEMENT

RELATIONSHIP_ID	other data
100002	###
100003	###
100007	###
100008	###
100010	###
100011	###
100012	###
100013	###
100017	###
100018	...
...	...

Execution time of non-correlated subselect

Tablescan of T_SPESREG_REMOVE: 1 sec (5 pages = 5 IO)

2000 items to join, 9 ms pr item = 18 sec

3000 items to find and DELETE, 18+15 ms pr item = 99 sec

In total: 118 sec = 1:58 min:second.

That is 1221 times faster.

(Before – it took in total 40:02 hours:minutes.)

(And it is still 61 times as fast as 20 parallels)

GROUPING AND FINDING MAX

```
SELECT ACCOUNT_NO, ISIN,  
      STATEMENT_DATE , BALANCE  
   FROM HOLDINGS A  
 WHERE A.STATEMENT_DATE = (  
SELECT MAX ( STATEMENT_DATE )  
   FROM HOLDINGS B  
 WHERE B.ACCOUNT_NO = A.ACCOUNT_NO  
   AND B.REGIME = A.REGIME  
   AND B.ISIN = A.ISIN  
   AND B.STATEMENT_DATE <= :H )
```

Table HOLDINGS: 19,7 million rows.
298 450 pages.

Good index on
REGIME
ACCOUNT_NO
STATEMENT_DATE
ISIN

Index has 5 levels and 375 900 leaf pages

SQL_CALL	SQL	INDB2_TIME	INDB2_CPU	GETPAGE
OPEN	1	02:52:51.803	01:00:10.207	1 104 200 011

Why so many getpages? What do we do?

ACCOUNT_NO	REGIME	ISIN	STATEMENT_DATE	BALANCE
12312312312	CUST	NO9879879877	02.09.2024	1 100
12312312312	CUST	NO9879879877	04.09.2024	1 200
12312312312	CUST	NO9879879877	05.09.2024	980
12312312312	CUST	NO9879879877	09.09.2024	1 358
++++	++++	++++	++++	++++
12312312312	CUST	NO9879879877	27.09.2024	1 400
12312312312	CUST	NO9879879877	30.09.2024	1 150
12312312312	CUST	NO9879557888	02.09.2024	1 200
12312312312	CUST	NO9879557888	25.09.2024	1 450
12312312312	CUST	NO9879557888	26.09.2024	1 480
12912912912	CORP	SE8833883333	03.09.2024	134 760
12912912912	CORP	SE8833883333	09.09.2024	128 450
12912912912	CORP	SE8833883333	23.09.2024	176 000
12912912912	CORP	SE8833883333	24.09.2024	153 980
++++	++++	++++	++++	++++
12912912912	CORP	SE8833883333	27.09.2024	151 500
12912912912	CORP	SE8833883333	30.09.2024	149 230
13813813813	CUST	NO1114442222	04.09.2024	2 800
13813813813	CUST	NO1114442222	30.09.2024	2 850

```
SELECT ACCOUNT_NO, ISIN,
       STATEMENT_DATE , BALANCE
  FROM HOLDINGS A
 WHERE A.STATEMENT_DATE = (
SELECT MAX ( STATEMENT_DATE )
  FROM HOLDINGS B
 WHERE B.ACCOUNT_NO = A.ACCOUNT_NO
   AND B.REGIME = A.REGIME
   AND B.ISIN = A.ISIN
   AND B.STATEMENT_DATE <= :H )
```

The SQL finds the rows **within each REGIME, ACCOUNT, ISIN** with the maximum **STATEMENT_DATE**.

The indexing is close to perfect.

So why so slow?

Remember – SQL is executed sequentially....

Processing of the MAX

ACCOUNT_NO	REGIME	ISIN	STATEMENT_DATE	BALANCE
12312312312	CUST	NO9879879877	02.09.2024	1 100
12312312312	CUST	NO9879879877	04.09.2024	1 200
12312312312	CUST	NO9879879877	05.09.2024	980
12312312312	CUST	NO9879879877	09.09.2024	1 358
++++	++++	++++	++++	++++
12312312312	CUST	NO9879879877	27.09.2024	1 400
12312312312	CUST	NO9879879877	30.09.2024	1 150
12312312312	CUST	NO9879557888	02.09.2024	1 200
12312312312	CUST	NO9879557888	25.09.2024	1 450
12312312312	CUST	NO9879557888	26.09.2024	1 480
12912912912	CORP	SE8833883333	03.09.2024	134 760
12912912912	CORP	SE8833883333	09.09.2024	128 450
12912912912	CORP	SE8833883333	23.09.2024	176 000
12912912912	CORP	SE8833883333	24.09.2024	153 980
++++	++++	++++	++++	++++
12912912912	CORP	SE8833883333	27.09.2024	151 500
12912912912	CORP	SE8833883333	30.09.2024	149 230
13813813813	CUST	NO1114442222	04.09.2024	2 800
13813813813	CUST	NO1114442222	30.09.2024	2 850

```

WHERE A.STATEMENT_DATE = (
SELECT MAX ( STATEMENT_DATE )
FROM HOLDINGS B
WHERE B.ACCOUNT_NO = 12912912912
AND B.REGIME = 'CORP'
AND B.ISIN = 'SE8833883333'
AND B.STATEMENT_DATE <= '29.09.2024' )

```

5 index pages + 1 data page for each line.

A line is checked towards 10 others in average.

$10 * (5+1 \text{ getpages}) * 17,7 \text{ mill} = 1 062 \text{ million}$

Though –what are we actually doing?

ACCOUNT_NO	REGIME	ISIN	STATEMENT_DATE	BALANCE
12312312312	CUST	NO9879879877	02.09.2024	1 100
12312312312	CUST	NO9879879877	04.09.2024	1 200
12312312312	CUST	NO9879879877	05.09.2024	980
12312312312	CUST	NO9879879877	09.09.2024	1 358
++++	++++	++++	++++	++++
12312312312	CUST	NO9879879877	27.09.2024	1 400
12312312312	CUST	NO9879879877	30.09.2024	1 150
12312312312	CUST	NO9879557888	02.09.2024	1 200
12312312312	CUST	NO9879557888	25.09.2024	1 450
12312312312	CUST	NO9879557888	26.09.2024	1 480
12912912912	CORP	SE8833883333	03.09.2024	134 760
12912912912	CORP	SE8833883333	09.09.2024	128 450
12912912912	CORP	SE8833883333	23.09.2024	176 000
12912912912	CORP	SE8833883333	24.09.2024	153 980
++++	++++	++++	++++	++++
12912912912	CORP	SE8833883333	27.09.2024	151 500
12912912912	CORP	SE8833883333	30.09.2024	149 230
13813813813	CUST	NO1114442222	04.09.2024	2 800
13813813813	CUST	NO1114442222	30.09.2024	2 850

What are we actually doing?

We are finding the MAX smaller than
'29.09.2024' you may say.

But if we turn around the list, and only display
the rows hit by STATEMENT_DATE <=
'29.09.2024'

Then we just find the first for whenever
ACCOUNT_NO, REGIME, ISIN changes.
(Another way of describing what we want to
find).

This is a simple SELECT and then some
scanning in code.

Simple solution

ACCOUNT_NO	REGIME	ISIN	STATEMENT_DATE	BALANCE
12312312312	CUST	NO9879557888	26.09.2024	1 480
12312312312	CUST	NO9879557888	25.09.2024	1 450
12312312312	CUST	NO9879557888	02.09.2024	1 200
12312312312	CUST	NO9879879877	27.09.2024	1 400
++++	++++	++++	++++	++++
12312312312	CUST	NO9879879877	09.09.2024	1 358
12312312312	CUST	NO9879879877	05.09.2024	980
12312312312	CUST	NO9879879877	04.09.2024	1 200
12312312312	CUST	NO9879879877	02.09.2024	1 100
12912912912	CORP	SE8833883333	27.09.2024	151 500
++++	++++	++++	++++	++++
12912912912	CORP	SE8833883333	24.09.2024	153 980
12912912912	CORP	SE8833883333	23.09.2024	176 000
12912912912	CORP	SE8833883333	09.09.2024	128 450
12912912912	CORP	SE8833883333	03.09.2024	134 760
13813813813	CUST	NO1114442222	04.09.2024	2 800

```
DECLARE C-AGGREGATED CURSOR FOR  
SELECT ACCOUNT_NO, ISIN,  
       STATEMENT_DATE , BALANCE  
FROM HOLDINGS A  
WHERE STATEMENT_DATE <= '29.09.2024'  
ORDER BY STATEMENT DATE DESC;
```

Now we add some 'normal' code, that reads this cursor, and whenever ACCOUNT_NO, REGIME, ISIN changes, read the first line.

And we read the table once, with 298 450 pages, down from 1 100 millions.
298 450 is still a large number, but it is only 0,027% of 1 100 millions!

SQL_CALL	SQL	INDB2_TIME	INDB2_CPU	GETPAGE
OPEN	1	06:32.110532	00:26.110344	785 073

Compared with 2:53 hours:min
Or 173 min.

Complete solution

ACCOUNT_NO	REGIME	ISIN	STATEMENT_DATE	BALANCE	SEQNR
12312312312	CUST	NO9879557888	26.09.2024	1 480	1
12312312312	CUST	NO9879557888	25.09.2024	1 450	2
12312312312	CUST	NO9879557888	02.09.2024	1 200	3
12312312312	CUST	NO9879879877	27.09.2024	1 400	1
++++	++++	++++	++++	++++	2-7
12312312312	CUST	NO9879879877	09.09.2024	1 358	8
12312312312	CUST	NO9879879877	05.09.2024	980	9
12312312312	CUST	NO9879879877	04.09.2024	1 200	10
12312312312	CUST	NO9879879877	02.09.2024	1 100	11
12912912912	CORP	SE8833883333	27.09.2024	151 500	1
++++	++++	++++	++++	++++	2-4
12912912912	CORP	SE8833883333	24.09.2024	153 980	5
12912912912	CORP	SE8833883333	23.09.2024	176 000	6
12912912912	CORP	SE8833883333	09.09.2024	128 450	7
12912912912	CORP	SE8833883333	03.09.2024	134 760	8
13813813813	CUST	NO1114442222	04.09.2024	2 800	1

```

SELECT ACCOUNT_NO, ISIN,
STATEMENT_DATE, BALANCE
FROM
(SELECT A.ACCOUNT, A.ISIN,
A.STATEMENT_DATE, A.BALANCE,
ROW_NUMBER () OVER
(PARTITION BY A.ACCOUNT, A.ISIN, A.REGIME
ORDER BY STATEMENT_DATE DESC )
AS SEQNR
FROM HOLDINGS A
WHERE A.STATEMENT_DATE <= :H)
WHERE SEQNR = 1
    
```

SQL_CALL	SQL	INDB2_TIME	INDB2_CPU	GETPAGE
OPEN	1	01:06.678265	00:19.459510	785 073

Learning points

- Understand what the database is doing with the SQL.
- Understand what you are actually doing.
- When seeing huge IO numbers, try to understand how they may originate
- Is this a totally unique problem in the world? Or should there exist a fairly standard solution?
- Note how large gains we make. Not 20-30% It is 90-95-99% of runtime removed!
- This is done not by more hardware, but by clever work.

Assistance from AI



Here is my SQL query to optimize:

```
SELECT B.SOURCE_REF, A.QUESTION_CODE, B.ANSWER_GR_CODE, B.ANSWER_GR_NO, A.ANSWER_CODE, NULLIF( A.ANSWER_VALUE_NUM, 0 ), A.ANSWER_VALUE_TEXT, A.SYSTEMBANK, B.CHANGED_TIME, B.VALID_TO, C.ANSWER_ALT_NO, B.GROUP_SOURCE, B.GROUP_TYPE, B.CONTEXT, B.CONTEXT_ID
FROM ANSWER A, ANSWER_GR B, ANSWER_ALT C
WHERE B.OPPHAV = 'K98' [REDACTED]
AND B.CONTEXT IN ( 'XAL', 'K39', 'CDB', 'CDA', 'ODD', 'OBB' )
AND B.VALID_FROM < (SELECT DEC ( VERDI_TALL * 10000 , 12 , 0 )
FROM COMPARM
WHERE COMPARM = 'JOURNALDATE' )
AND B.VALID_TO > ( SELECT DEC ( VERDI_TALL * 10000 , 12 , 0 )
FROM COMPARM
WHERE COMPARM = 'JOURNALDATE' )
AND B.ANSWER_GR_ID = A.ANSWER_GR_ID [REDACTED]
AND C.QUESTION_GROUP_ID = A.QUESTION_GROUP_ID
AND C.QUESTION_NO = A.QUESTION_NO [REDACTED]
AND C.ANSWER_ALT_NO = (SELECT MAX ( ANSWER_ALT_NO )
FROM ANSWER_ALT D [REDACTED]
WHERE C.QUESTION_GROUP_ID = D.QUESTION_GROUP_ID
AND C.QUESTION_NO = D.QUESTION_NO ) WITH UR
```

This SQL is pretty similar to the last example, and I search a similar solution.



Assistance from AI

Thanks for sharing the query! Here's a breakdown of optimization suggestions for your SQL query, which appears to be written for **Db2** (based on syntax like WITH UR and DEC()):

General Observations

You're using **implicit joins** (comma-separated tables in FROM) — switching to **explicit JOIN syntax** improves readability and can help the optimizer.

There are **subqueries** in the WHERE clause that are repeated — these can be **factored out**.

```
SELECT  
    B.SOURCE_REF,  
    A.QUESTION_CODE,  
    B.ANSWER_GR_CODE,  
    B.ANSWER_GR_NO,  
    A.ANSWER_CODE, +++
```



Assistance from AI

```
SELECT B.SOURCE_REF, A.QUESTION_CODE, B.ANSWER_GR_CODE, B.ANSWER_GR_NO, A.ANSWER_CODE,  NULLIF (  
A.ANSWER_VALUE_NUM, 0 ), A.ANSWER_VALUE_TEXT, A.SYSTEMBANK, B.CHANGED_TIME,   B.VALID_TO, C.ANSWER_ALT_NO,  
B.GROUP_SOURCE, B.GROUP_TYPE, B.CONTEXT, B.CONTEXT_ID  
FROM [REDACTED]  
ANSWER_GR B  JOIN ANSWER A ON B.ANSWER_GR_ID = A.ANSWER_GR_ID  
JOIN ANSWER_ALT C  ON C.QUESTION_GROUP_ID = A.QUESTION_GROUP_ID  
AND C.QUESTION_NO = A.QUESTION_NO  
WHERE [REDACTED]  
B.OPPHAV = 'K98'  
AND B.CONTEXT IN ('XAL', 'K39', 'CDB', 'CDA', 'ODD', 'OBB')  
AND B.VALID_FROM < ( SELECT DEC(VERDI_TALL * 10000, 12, 0)  
                      FROM COMPARM  
                     WHERE COMPARM = 'JOURNALDATE' )  
AND B.VALID_TO > (SELECT DEC(VERDI_TALL * 10000, 12, 0)  
                      FROM COMPARM  
                     WHERE COMPARM = 'JOURNALDATE' )  
AND C.ANSWER_ALT_NO = ( SELECT MAX(D.ANSWER_ALT_NO)  
                      FROM ANSWER_ALT D  
                     WHERE D.QUESTION_GROUP_ID = C.QUESTION_GROUP_ID  
                       AND D.QUESTION_NO = C.QUESTION_NO )  
WITH UR;
```

This is a nice rewrite to explicit JOINS.
That is great, but there is absolutely no
CPU or elapsed time optimisation here.



Assistance from AI

avoid using SELECT MAX

Optimized Version Without SELECT MAX

Here's how you can rewrite your query using a **derived table join** instead of SELECT MAX(...): (AUF: This is Copilots word for a CTE – WITH)

Assistance from AI



```
WITH MaxAlt AS ( SELECT QUESTION_GROUP_ID, QUESTION_NO, MAX(ANSWER_ALT_NO) AS MAX_ANSWER_ALT_NO  
    FROM ANSWER_ALT  
   GROUP BY QUESTION_GROUP_ID, QUESTION_NO)
```

```
SELECT B.SOURCE_REF, A.QUESTION_CODE, B.ANSWER_GR_CODE, B.ANSWER_GR_NO, A.ANSWER_CODE,  
NULLIF(A.ANSWER_VALUE_NUM, 0), A.ANSWER_VALUE_TEXT, A.SYSTEMBANK, B.CHANGED_TIME, B.VALID_TO,  
C.ANSWER_ALT_NO, B.GROUP_SOURCE, B.GROUP_TYPE, B.CONTEXT, B.CONTEXT_ID  
FROM
```

```
    ANSWER_GR B JOIN ANSWER A ON B.ANSWER_GR_ID = A.ANSWER_GR_ID  
JOIN ANSWER_ALT C ON C.QUESTION_GROUP_ID = A.QUESTION_GROUP_ID  
    AND C.QUESTION_NO = A.QUESTION_NO
```

```
JOIN MaxAlt M ON C.QUESTION_GROUP_ID = M.QUESTION_GROUP_ID  
    AND C.QUESTION_NO = M.QUESTION_NO  
    AND C.ANSWER_ALT_NO = M.MAX_ANSWER_ALT_NO
```

```
WHERE B.OPPHAV = 'K98'  
    AND B.CONTEXT IN ('XAL', 'K39', 'CDB', 'CDA', 'ODD', 'OBB')  
    AND B.VALID_FROM < (SELECT DEC(VERDI_TALL * 10000, 12, 0)  
                           FROM COMPARM  
                           WHERE COMPARM = 'JOURNALDATE')  
    AND B.VALID_TO > ( SELECT DEC(VERDI_TALL * 10000, 12, 0)  
                           FROM COMPARM  
                           WHERE COMPARM = 'JOURNALDATE' )
```

```
WITH UR;
```

Well, I still have the SELECT MAX I wanted to remove, but it is placed in a CTE (WITH), and executed in this way It is quite effective.

I think we have 98% of the total optimization already here. But rather than the SELECT MAX, I want to use OLAP with row number ranked.



Assistance from AI

Use OLAP

Goal: Replace MAX() with OLAP

If your original use of MAX(ANSWER_ALT_NO) was to get the latest or highest alternative per question, you can use ROW_NUMBER() or RANK() instead.

OLAP-Based Rewrite Using ROW_NUMBER()



Assistance from AI

```
WITH RankedAlternatives AS ( SELECT QUESTION_GROUP_ID, QUESTION_NO, ANSWER_ALT_NO, ROW_NUMBER() OVER ( PARTITION BY QUESTION_GROUP_ID, QUESTION_NO ORDER BY ANSWER_ALT_NO DESC ) AS rn  
FROM ANSWER_ALT)
```

```
SELECT B.SOURCE_REF, A.QUESTION_CODE, B.ANSWER_GR_CODE, B.ANSWER_GR_NO, A.ANSWER_CODE,  
NULLIF(A.ANSWER_VALUE_NUM, 0), A.ANSWER_VALUE_TEXT, A.SYSTEMBANK, B.CHANGED_TIME, B.VALID_TO,  
RA.ANSWER_ALT_NO, B.GROUP_SOURCE, B.GROUP_TYPE, B.CONTEXT, B.CONTEXT_ID  
FROM ANSWER_GR B  
JOIN ANSWER A ON B.ANSWER_GR_ID = A.ANSWER_GR_ID  
JOIN RankedAlternatives RA ON RA.QUESTION_GROUP_ID = A.QUESTION_GROUP_ID  
    AND RA.QUESTION_NO = A.QUESTION_NO  
    AND RA.rn = 1
```

```
WHERE B.OPPHAV = 'K98'  
    AND B.CONTEXT IN ('XAL', 'K39', 'CDB', 'CDA', 'ODD', 'OBB')  
    AND B.VALID_FROM < (SELECT DEC(VERDI_TALL * 10000, 12, 0)  
        FROM COMPARM  
        WHERE COMPARM = 'JOURNALDATE' )  
    AND B.VALID_TO > (SELECT DEC(VERDI_TALL * 10000, 12, 0)  
        FROM COMPARM  
        WHERE COMPARM = 'JOURNALDATE' )  
WITH UR;
```

And this is exactly what I planned to write. This is a little bit more efficient than the last SQL.

When I rewrite these, it is tricky to get all JOIN expressions correct, it is tricky to avoid typos etc.

Conclusion: As long as we are using pure SQL rewrites where there is no extra input from the data distribution, Copilot does this better than you.

Will AI fix it all for you now?

Job runs for 46 minutes, burns loads of CPU

I look into Detector

SQL_CALL	SQL	INDB2_TIME	INDB2_CPU
SELECT	49048524	36:24.927427	33:05.766936

```
SELECT *
  INTO :H, :H , :H , :H, :H, :H, :H, :H, :H
  FROM EXCHANGE_RATE
 WHERE CURRENCTY = :H
   AND DATE = :H
 WITH UR
```

Will AI fix it all for you now?

Job runs for 46 minutes, burns loads of CPU

I look into Detector

SQL_CALL	SQL	INDB2_TIME	INDB2_CPU
SELECT	49048524	36:24.927427	33:05.766936

```
SELECT *
  INTO :H, :H, :H, :H, :H, :H, :H, :H, :H
  FROM EXCHANGE_RATE
 WHERE CURRENCTY = :H
   AND DATE = :H
 WITH UR
```

Hint:

We look up currency exchange rate about 50 million times. 95% of all currency payments in Norway are in SEK, DKK, EUR, USD, GBP.

Read those 5 currencies into a small internal table.

Check every payment. If SEK, DKK, EUR, USD, GBP – read the internal table. If some other, read the database table.

Learning points

1. Copilot needed some hints to get it correct.
2. As of now, we still need some SQL expert knowledge to guide it in the correct direction.
3. After just 3 tries I got the complex, efficient SQL with both CTE and OLAP. I typically spend 0,5 – 1 hour rewriting such an SQL, and always have a couple of retries to get all correct.
4. Of course, SQL is highly structured and as such “easy” for Copilot, though I am still impressed.
5. There are still problems where we must look beyond the SQL and look at more overall properties in order to see a solution.