Real life examples
(a bit disguised and a lot simplified)

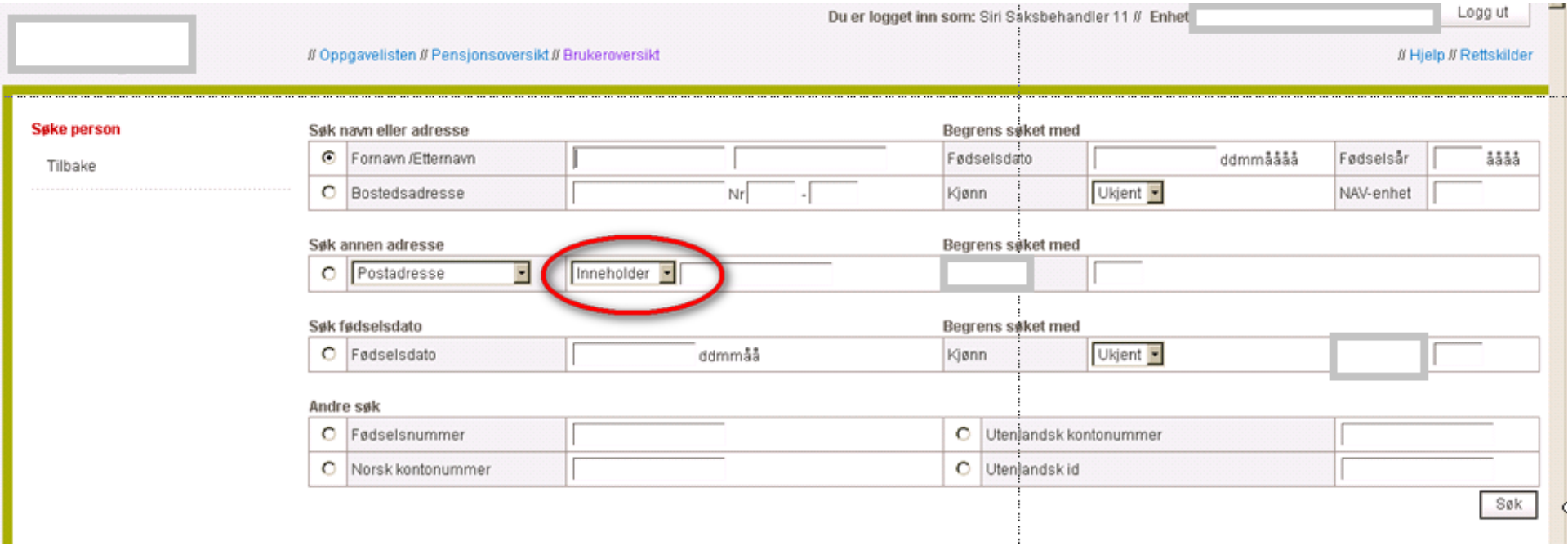
- Some may seem too trivial to mention
- But they did happen in companies with well educated people
- So please bear with me!

’Dishonest’ Queries
Most commonly used with COBOL

Example:

- A search dialog allows the user to enter an optional value-requirement for a column, eg a date. (Single value or value-range.)
- One query is used for several input-combinations:
  ”…AND SOME_DATE BETWEEN :DAY1 AND :DAY2…”
- During statement parsing DB2 decides that the best access method is index range scan using an index on SOME_DATE
- If the user enters no requirements for this column, the query is executed with paremeter values DAY1 = ’0001-01-01’ and DAY2 = ’9999-12-31’.
- What will happen? (Assume a table with several million rows).

Why does not the SQL use the index?



WHERE POSTADRESSE LIKE ‘%:E’

Change default to “Starts with”, and the SQL will be
WHERE POSTADRESSE LIKE ‘:E%’

Indexable Predicates revisited

Not indexable:

Parameter-driven comparator selection in SQL

```
…AND TIDSPKT_REG > :T
  AND (CASE
 WHEN :E = ‘ ‘
    THEN ENHET_BEHAND
     ELSE :E
 END) = ENHET_BEHAND…
```

Indexable:

Parameter-driven query selection in program

Query for E = ‘ ‘:

```
…AND TIDSPKT_REG > :T…
```

Query for other values of E:

```
…AND TIDSPKT_REG > :T
   AND ENHET_BEHAND = :E
```

Bad performance may be a symptom of incorrect logic

Bad performance:

```
select …
  from T_YTELSE
 where dato_ytel_iver_tom >= ?
       or dato_ytel_iver_tom is null
   and er_gyldig                = '1'
   and forhold_id               = ?
```

Did we really mean:

```
 where dato_ytel_iver_tom >= ?
       or (dato_ytel_iver_tom is null
   and er_gyldig                = '1'
   and forhold_id               = ?)
```

…or perhaps:

```
where (dato_ytel_iver_tom >= ?
       or dato_ytel_iver_tom is null)
   and er_gyldig                = '1'
   and forhold_id               = ?
```

Both performance and returned data may be totally different!

Q: Why are some executions of same query a lot slower than others?
A: Uneven key distributions!

```
Select x
  from t
 where y = :A
```

START_TIME	SQL	INDB2_TIME	INDB2_CPU	GETPAGE
08:44:36.946	124	01:09	00:03	431984
08:54:14.527	124	01:02	00:02	225995
09:05:41.789	124	00:26	00:03	431986
09:05:41.249	124	01:06	00:02	221913
09:18:41.141	124	01:02	00:02	221914

Y	# rows
<blank>	4 480 731
80000427901	109 197
80000438148	104 277
80000345435	103 698
80000423362	65 882
80000438116	17 744
80000432839	16 871
80000438118	14 053
80000366238	12 171
80000438132	11 171
80000364458	5

Question: Why are these straight-forward deletes so expensive?

```
delete
    from T_SJEKKLISTE
  where SJEKKLISTE_ID = ?
```

Yes, we have unique index on SJEKKLISTE_ID!

SQL_TEXT	USE COUNT	TIMEPCT	CPUPCT	INDB2_TIME	INDB2_CPU	GETPAGE
delete from T_SJEKKLISTE where	94	7,24 %	11,07 %	02:24	00:42	247 500
select oppgavedo0_.OPPGAVE_ID a:	1756	5,26 %	3,67 %	01:45	00:14	606 980
select oppgavedo0_.OPPGAVE_ID a:	1758	5,21 %	3,63 %	01:44	00:13	587 916
select oppgavedo0_.OPPGAVE_ID a:	313	4,43 %	3,47 %	01:28	00:13	624 083
select oppgavedo0_.OPPGAVE_ID a:	262	4,67 %	3,38 %	01:33	00:12	639 818
select oppgavedo0_.OPPGAVE_ID a:	1287	3,57 %	3,36 %	01:11	00:12	559 517
select oppgavedo0_.OPPGAVE_ID a:	1158	2,98 %	3,03 %	00:59	00:11	498 704

Answer: Because another table has a foreign key without index support

```
SELECT F.COLSEQ
      ,F.COLNAME
  FROM SYSIBM.SYSRELS         R
      ,SYSIBM.SYSFOREIGNKEYS F
 WHERE R.CREATOR   = 'GS606P'
   AND R.TBNAME    = 'T_OPPGAVE'
   AND R.REFTBNAME =
 'T_SJEKKLISTE'
   AND F.CREATOR   = R.CREATOR
   AND F.TBNAME    = R.TBNAME
   AND F.RELNAME   = R.RELNAME
```

COLSEQ	COLNAME
1	SJEKKLISTE_ID

TABLE	INDEX	TB_SEQ_GP	TB_IDX_GP	IS_GETP	IS_TBGETP
T_SJEKKLISTE_LINJE	XIE21VEC			2,0	0,0
T_SJEKKLISTE_KOLONNE	XIE21X7B			2,0	0,0
T_SJEKKLISTE	XPKTRSJE			4,0	1,8
T_SJEKKLISTE		0,0	1,8		
T_OPPGAVE		**2 645,6**	0,0		

The product of two numbers

- Not a very large number (1):
  - A batch executed a correlated subquery without index support.
  - Full table scan of 3 400 pages on every execution.
  - No disaster for a batch.
- Not a very large number (2):
  - The subquery was executed 110 000 times during each run of the batch.
  - Does not seem unreasonable for a batch
- A large number: (1) X (2)
  - Each batch run did 374 million page gets for this table
  - Execution time for each run exceeded 2,5 hours
  - A disaster for this batch. Unable to complete in batch-window
- A smaller factor:
  - Created new index tailored for the subquery
  - Now 3 page gets per subquery execution
- A smaller product:
  - 330 000 page gets for this table/index during a batch run
  - Execution time now 15 minutes

Redundancy is sometimes required…

Find payments with a certain 'current' status

1. Normalized tables:

```
SELECT …
  FROM PAYMENT P
 INNER JOIN PAYMENT_STATUS S
    ON P.PAYMENT_ID   = S.PAYMENT_ID
 WHERE S.STATUS_CODE = :H
   AND S.STATUS_TIME =
       (SELECT MAX(S2.STATUS_TIME)
          FROM PAYMENT_STATUS S2
         WHERE S2.PAYMENT_ID = S.PAYMENT_ID)
```

2. Redundant column CURRENT_STATUS_CODE:

```
SELECT P.PAYMENT_ID
  FROM PAYMENT P
WHERE P.CURRENT_STATUS_CODE = :H
```

- Assume:
  - Each payment has an average of 4 status-history rows.
  - Status-history contains 1 million rows with requested status value.
  - Requested value is most recent status for 1000 payments.
- Result:
  - With normalized tables and reasonable indexing we will need 2 – 5 million getpage operations to retrieve 1000 rows.
  - With redundant copy of current status and reasonable indexing we will need 2000 – 3000 getpage operations to retrieve 1000 rows.

…good understanding of data patterns may be even better…

Find task items on active tasks where approval is pending

```
SELECT …
  FROM TASK T
 INNER JOIN TASK_STATUS S
    ON S.TASK_ID = T.TASK_ID
 INNER JOIN TASK_ITEM I
    ON I.TASK_ID = T.TASK_ID
 WHERE I.APPROVAL    = 'PEND'
   AND S.STATUS_CODE = 'ACTV'
   AND S.STATUS_TIME =
 (SELECT MAX(S2.STATUS_TIME)
    FROM TASK_STATUS S2
   WHERE S2.TASK_ID
               = S.TASK_ID)
```

- Database skills:
  - Redundancy may help a lot
- Application skills:
  - Only active tasks have items with pending approval
  - Even most active tasks have no items with pending approval
- Combined skills:
  - Best solution is a new index on task_item with key ‘approval‘

… and synergy of creative minds may be best

Order-status history with sequence numbers:

```
SELECT …
  FROM ORDER O
 INNER JOIN ORDER_STATUS S
    ON S.ORDER_ID    = O.ORDER_ID
 WHERE S.STATUS_CODE = 'READY'
   AND S.SEQ_NO =
    (SELECT MAX(S2.SEQ_NO)
       FROM ORDER_STATUS S2
      WHERE S2.ORDER_ID = S.ORDER_ID)
```

- Database skills:
  - Redundancy may help a lot
- Application skills:
  - Why not change the rules: Always use SEQ_NO = 9999 for current status of an order
- Agreed result:

```
SELECT …
  FROM ORDER O
 INNER JOIN ORDER_STATUS S
    ON S.ORDER_ID    = O.ORDER_ID
 WHERE S.STATUS_CODE = 'READY'
   AND S.SEQ_NO      = 9999
```

Connection statement cache

- A DBMS must translate the SQL statements sent to it. This is a CPU-demanding process (finally…. till now we have mostly looked at IO and memory….).
  - Load into shared pool
  - Syntax parse (correct SQL as such)
  - Semantic parse (are all table & column names correct, check dictionary)
  - Optimisation (create access plan with info from db statistics)
  - Create executable
- You may set up each connection with a cache of SQL statements already translated,.
- Requires the SQL to be exact the same. Is case sensitive. Must use bind variables, not values.

```
select order_id, account_id
    from order_item
where account_id = :OrderId
```

Does not match neither

```
select order_id, account_id
    from order_item
where account_id = 158293
```

- Hint: Always user bind variables, even when you work with a constant. And use the same variable name

```
select Order_Id, Account_Id
    from Order_Item
where Account_Id = :OrderId
```

Search for exceptional values

DB2 Catalog info:

- SYSTABLES:
  1 million rows in table.
- SYSINDEXES:
  2 distinct key values for index on STATUS_CODE
- SYSCOLDIST for STATUS_CODE values:
  - 99.99% ’NORMAL’
  - 0.01% ’SPECIAL’

Probable access path selection:

Sequential scan of table or clustering index:

…WHERE STATUS_CODE = ?

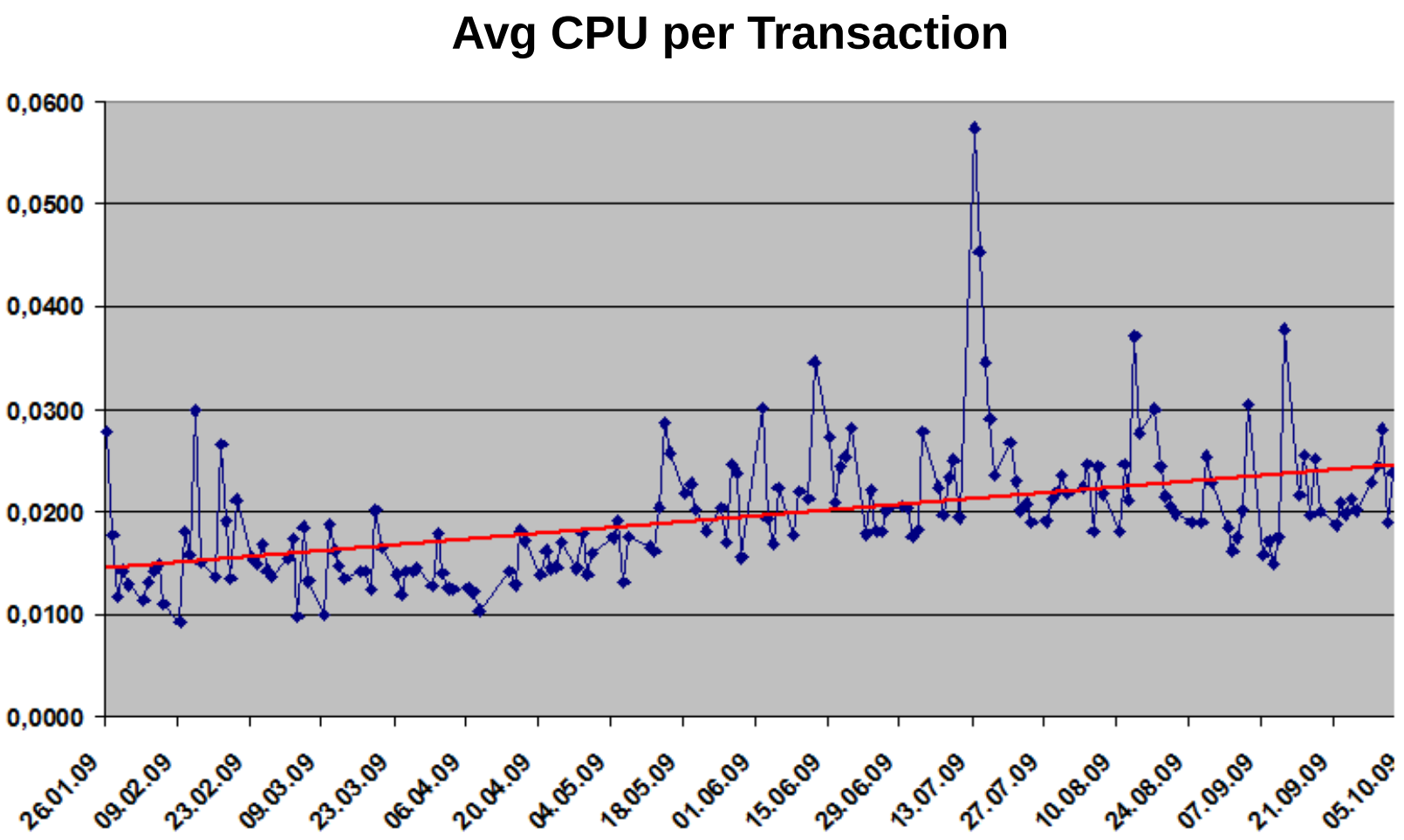
Index lookup on STATUS_CODE:

…WHERE STATUS_CODE = ’SPECIAL’

Controlling data growth

- Data growth may not impact transaction performance significantly if
  - Number of accessed rows per transaction is stable anyway
  - Every SQL is supported by indices that will hit only requested rows
- But real life is often different. In this case:
  - Some queries did sequential scans of entire tables or indices
  - One query accessed all rows in table x and joined them with other tables

Growing backlog for archive/delete of outdated information



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)
```