

Rug 2006

- V8 status Volvo
- Några problem på vägen
- V8 support i CAs verktyg
- Deadlock in DB2

LET'S MAKE SURE

V8 Status

- April 2004 migreras teknik DB2
- Februari 2005 migreras Kundtest till V8 comp
- April 2005 migreras sd och production till v8 comp
- Juni 2006 migreras sd till v8 nfm
- Oktober 2006 går vi med prod till nfm
- Status 060123:
 - 36 DB2 system kör DB2 v8 NFM
 - 43 -"- DB2 v8 Compatibility mode
 - 30 (13) -"- DB2 v7 (migreras innan 31/3 2006)

LET'S MAKE SURE

Några problem på vägen

- Många problem
 - Min lista täcker 56 problem
 - Apars 4
 - PTF 34
 - Vendor 3
 - Other reason 16
 - sql standard enforced
 - connect v8
 - rebind 00c90101
 - compile/link -189
 - select with union need column names
 - cobol coprocessor , -501

LET'S MAKE SURE

Några jobbiga

- IMS transar abendar mot DB2 – fix CA detector
- 2 fall där DDF trådar hänger, går ej att cancelera, tar ”bara” cpu ..
 - Business Object user ställer ”fel fråga”
 - Insert av binary data i CLOB
- JCC type 2 large resultset felar, workaround JCC typ 4
- IMS applikation använde sej av RTEREUS=YES orsakade allvarliga minnespb i DB2
- Reorg fails after Alter Index Partition

LET'S MAKE SURE

V8 support i CAs verktyg

- R11 är tolerant men bara till en viss del
 - Table versioning – stöds ej av PDA och RC/M
 - PDA skippar dessa tablespace – kör manuell reorg för att rätta till
- Vi kör beta på R11.5
 - Xmanager måste vara igång (detector)
 - Long name support – RCQ,RCM men ej i PFU!
 - Men fortfarande inget stöd för Online Schema i RCM
 - Inget stöd för DPSI (förutom i RCQ)

LET'S MAKE SURE

DB2 locking pb

- Lite om applikationen UCHP
 - Garanti och service åtaganden för lastbilar och bussar
 - 3-4 milj claimjobs 2007, 3000 anv, 32 milj PU i DB2, 400' i mån
 - WAS Unix – DB2 z/OS
 - Många system inblandade – bla z/OS och AS400
 - Jeeves, Volvo Java framework, Model driven, Persistent
- JCC type 4 (v8.2.1)
- Problemet
 - 2 användare samtidigt
 - SQLCode -911 or -913, RC 00C90088

LET'S MAKE SURE

Mera bakgrund

- Vad vet ni mer ?
 - Arkitekten säger att deadlock aldrig kan inträffa !
 - Vi kör REPEATABLE_READ !?
 - Vi kör med radlås ... Tror vi ?
- Låt oss titta på lite detaljer
- String url ="jdbc:db2:d2y2"
+ ":traceFile=c:/temp/jcctrace.txt;"
+ "traceLevel=-1;"
- Jdbc trace

LET'S MAKE SURE

Setting the isolation level for a JDBC transaction

Connection.setTransactionIsolation(int *level*)

JDBC value	DB2 isolation level
TRANSACTION_SERIALIZABLE	Repeatable read
TRANSACTION_REPEATABLE_READ	Read stability
TRANSACTION_READ_COMMITTED	Cursor stability
TRANSACTION_READ_UNCOMMITTED	Uncommitted read

LET'S MAKE SURE

[Connection@3d5104] clearWarnings () called

[Connection@3d5104] prepareStatement (**INSERT INTO ClaimRepairHeader**
(claimHeaderId, repairing.....

[Connection@3d5104] prepareStatement () returned PreparedStatement@**653566**

[PreparedStatement@653566] getFetchSize () returned 0

[Connection@3d5104] clearWarnings () called

[Connection@3d5104] prepareStatement (**SELECT NEXT VALUE FOR**
CLAIMHEADERSEQ FROM DUMMYSEQUENCE

[Connection@3d5104] prepareStatement () returned PreparedStatement@**94cd2e**

[PreparedStatement@94cd2e] getFetchSize () returned 0

[PreparedStatement@94cd2e] executeQuery () called

LET'S MAKE SURE

[PreparedStatement@94cd2e] executeQuery () returned ResultSet@1920ea7

...

[ResultSet@768ee6] next () called

[ResultSet@1920ea7] next () called

[ResultSet@768ee6] next () returned false

[ResultSet@1920ea7] next () returned true

[ResultSet@1920ea7] getLong (1) called

[ResultSet@1920ea7] getLong () returned 34

[ResultSet@1920ea7] close () called

[PreparedStatement@1d009d5] close () called

[PreparedStatement@653566] setLong (1, 34) called

[PreparedStatement@653566] setInt (2, 1320) called

LET'S MAKE SURE

Sequences?

SELECT NEXT VALUE FOR CLAIMHEADERSEQ FROM DUMMYSEQUENCE ??????

SELECT max(jobNo) + 1 as job FROM ClaimJob WHERE headClaimHeaderId = ?

SELECT importerNo, ... , objversion FROM ClaimRefNoSerie WHERE importerNo=?

UPDATE ClaimRefNoSerie SET importerNo=?, ... WHERE importerNo=? AND
objversion=?

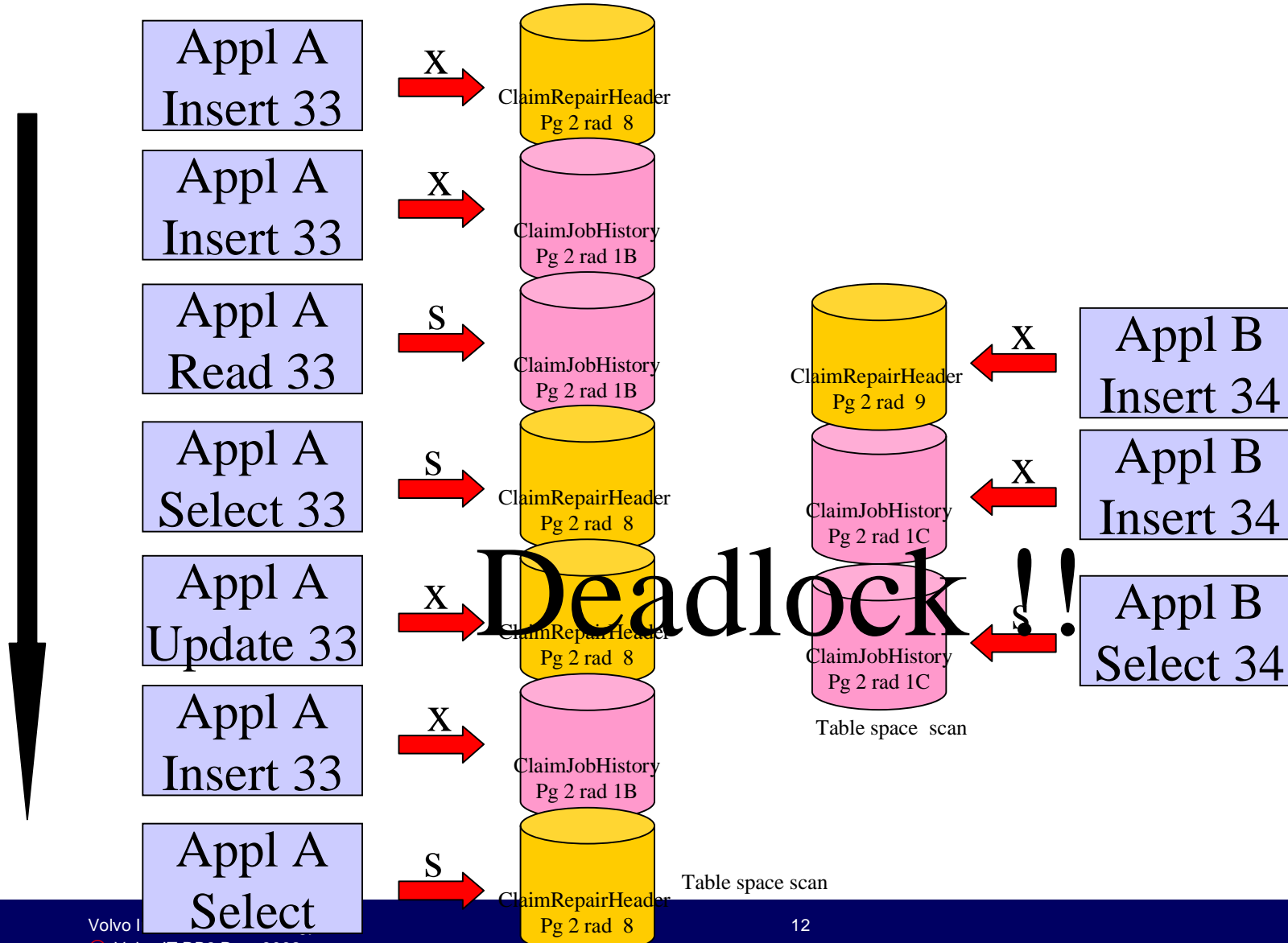
- Example of syntax

- CREATE SEQUENCE order_seq
START WITH 1 INCREMENT BY 1
- INSERT INTO orders (orderno,custno)
VALUES (NEXT VALUE FOR order_seq,123456);
- SELECT NEXT VALUE FOR order_seq
INTO :hv_seq from sysibm.sysdummy1;

LET'S MAKE SURE

Example of deadlock

LET'S MAKE SURE



Force index access

- Volatile table support
ALTER TABLE .. VOLATILE
 - Index access whenever possible. note: no list prefetch
- Svårt att hitta vad man har satt
 - RCQ R11.5 ok
 - Select SPLIT_ROWS from SYSIBM.SYSTABLES
Value is blank, except for VOLATILE tables, which will have 'Y' in the field to indicate to DB2 to use index access on this table whenever possible.

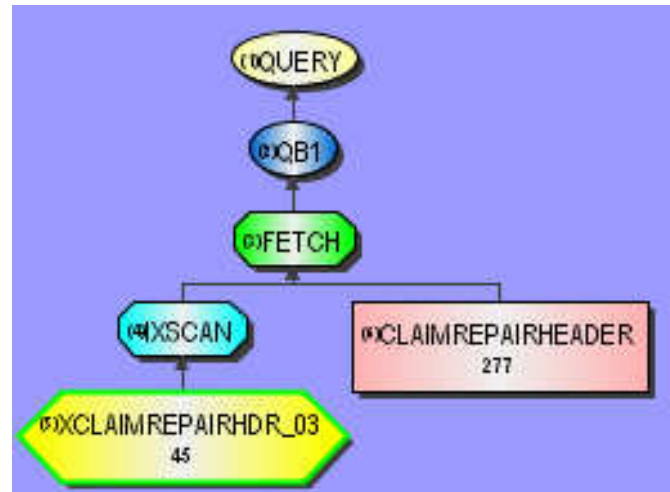
LET'S MAKE SURE

Några åtgärder

- ALTER TABLESPACE ... LOCKSIZE ROW ;
- ALTER TABLE ... VOLATILE ;
- Köra RUNSTATS – var aldrig körd i SD miljön !
- Thread c1 = **new**
C1(Connection.TRANSACTION_READ_COMMITTED);
- Men fortfarande Deadlock !!!

What about accesspath ?

**SELECT claimHeaderId FROM claimRepairHeader
WHERE repairingImporter = ? AND repairingDealer = ? AND refNo = ?**



**CREATE TYPE 2 INDEX XCLAIMREPAIRHDR_05 ON CLAIMREPAIRHEADER
(REPAIRINGIMPORTER ASC ,REPAIRINGDEALER ASC ,REFNO ASC)...**

Boken säger

ISOLATION (CS)

Allows maximum concurrency with data integrity.

However, after the process leaves a row or page, another process can change the data. With CURRENTDATA(NO), the process doesn't have to leave a row or page to allow another process to change the data. If the first process returns to read the same row or page, the data is not necessarily the same.

Lock avoidance:

With CURRENTDATA(NO), you have much greater opportunity for avoiding locks. DB2 can test to see if a row or page has committed data on it. If it has, DB2 does not have to obtain a lock on the data at all.

Hi Michael,

For this case, we never have the chance to use the Lock Avoidance mechanism. Because there are some restrictions for Lock Avoidance:

- 1) Must be FOR FETCH ONLY, Read Only or ambiguous
--> i.e Order by clauses for cursors to be read only
- 2) Must use Cursor Stability
- 3) Must be bound with CONRRENTDATA(NO)
--> Lock avoidance applies to non-returned rows for CURRENTDATA(YES)
- 4) Must not use Type 1 index access paths.

So for this case, the **DB2 must try to take the S-LOCK on the row because it contains uncommitted data.** And regardless of what Isolation Level we're using <Except for UR>, the deadlock can not be avoided.

To avoid the deadlock, we have two choices:

- 1) **Ensuring the Index Scan with matched indexed all the time.**
- 2) **Adjusting from the Application perspective, ie using the UR in the SELECT query.**