Deep Dive into IDS Locking Mechanisms

Eric Herber
Datenbank-Consulting

A05 (49)
Day, April 28, 2008 • 03:30 p.m. – 04:30 p.m.

2008 IIUG Informix Conference

www.iiug.org

Who am I ?

- A german Informix fan
- Working with Informix products since 1989, since 1996 as an independent consultant. No I'm not that old, I was actually 22 during my first contact with Informix technology
- Wrote several articles for IT magazines about Informix technology, but you might need to learn german to read them ☺:
  - http://www.herber-consulting.de/drupal/?q=fachartikel
- Owner of the website:
  - http://www.informix-zone.com

Agenda

1. Lock types
2. Lock granularities
3. Database logging modes
4. Isolation levels
5. Dynamic lock allocation in IDS
6. Lock wait time
7. Deadlocks
8. Analyzing lock conflicts
9. IDS Utilities and locking
10. IDS 11: Optimistic concurrency – your new friend

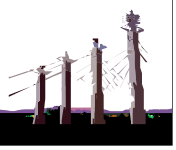
- **Lock types**
- Lock granularities
- Database logging modes
- Isolation levels
- Dynamic lock allocation in IDS
- Lock wait time
- Deadlocks
- Analyzing lock conflicts
- IDS Utilities and locking
- IDS 11: Optimistic concurrency – your new friend

Lock types I

- Share lock
  - Share locks can be placed on objects that do not have an exclusive lock already placed on them. More than one share lock can be placed on the **same object** at the **same time**
- Update lock
  - Update locks are created by cursors that have the '**for update**' extension specified and can only be placed on a row that does not already have an exclusive or update lock on it. The update lock will be converted to an **exclusive** lock as soon as the update is actually performed

Lock types II

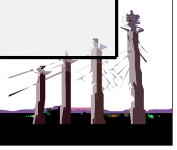
- Exclusive lock
  - Exclusive locks can only be placed on rows that do **not** have **any other kind** of lock on it. Once an exclusive lock is placed on a row, **no other locks** can be placed on the same row anymore
- Intent lock
  - Intent locks are **automatically** set by IDS. If a row in a table is updated, an exclusive lock is placed on the row and an **intent-exclusive** lock is placed on the table. This ensures that **no other session** could place a share or exclusive lock on the table itself as long as an **individual table row** is exclusively locked



Lock types III

- Lock compatibility matrix

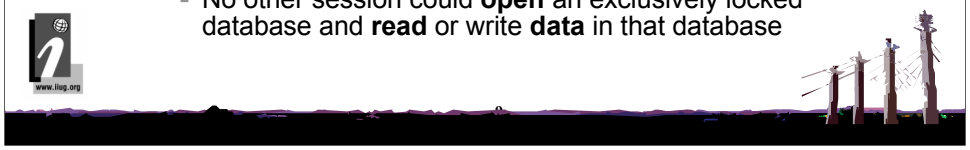
Current lock is:	Place share lock ?	Place update lock ?	Place exclusive lock ?
None	Yes	Yes	Yes
Share lock	Yes	Yes	No
Update lock	Yes	No	No
Exclusive lock	No	No	No



- Lock types
- **Lock granularities**
- Database logging modes
- Isolation levels
- Dynamic lock allocation in IDS
- Lock wait time
- Deadlocks
- Analyzing lock conflicts
- IDS Utilities and locking
- IDS 11: Optimistic concurrency – your new friend

Lock granularities I

- Database locks
  - Share lock
    - As soon as you **open** a database, a share lock will be placed on it. This ensures that no other session could place an **exclusive** lock on the database or **drop** the database
      - *database stores_demo;*
  - Exclusive lock
    - An exclusive lock must be **explicitely** set on a database. Utilities like *dbexport* will also place an exclusive lock on the database
      - *database stores_demo **exclusive**;*
    - No other session could **open** an exclusively locked database and **read** or write **data** in that database



Lock granularities II

- Table locks
  - Share lock
    - A share lock can be **explicitly** placed on a table. Utilities like *onunload* or *oncheck* might also place a share lock on a table
      - *begin work; lock table customer in **share** mode;*
    - Other sessions can read data from the table but **cannot** change any data
  - Exclusive lock
    - An exclusive lock can be **explicitly** set on a table or might be implicitly set by statements like *alter table:*
      - begin work; lock table customer in ***exclusive*** mode;
    - Only sessions with an isolation level of *dirty read* are able to **read** data from an exclusively locked table but are **not** allowed to **change** data. All other sessions are **not** allowed to **read** or **write** data in this table

Lock granularities III

- Page/Row lock
  - If the table is configured for lock mode **page**, IDS will lock a **whole page** instead of an individual row. The same is true for index pages.
  - If you have a small rowsize or/and a large pagesize, **many** rows might be affected by a single page lock
  - The **default** table lock mode could be set in the *onconfig* file or as environment variable:
    - **onconfig:** DEF_TABLE_LOCKMODE <row|page>
    - **Environment:** export IFX_TABLE_LOCKMODE=<row|page>
  - If DEF_TABLE_LOCKMODE is **not** set and the environment variable IFX_TABLE_LOCKMODE is also **not** set and the CREATE TABLE statement does **not** declare the lock-mode, IDS will choose lockmode **page** as the default

Lock granularities IV

- For OLTP systems lock mode **row** is in most cases the best solution
- Determining the current lock mode of a table:

1. oncheck –pt <db>:<tabname>
2. dbschema –d <db> -t <tabname> -ss
3. select locklevel from systables where tabname = “<tabname>";

- Changing **all** tables in a database to lock mode **row**:

```
output to alter_table.sql without headings
select "alter table '" || trim(owner) || "'." || trim(tabname) || " lock mode(row);"
from   'informix'.systables
where  tabid > 99
and    tabtype = 'T'
and    locklevel = 'P';
```

- Lock types
- Lock granularities
- **Database logging modes**
- Isolation levels
- Dynamic lock allocation in IDS
- Lock wait time
- Deadlocks
- Analyzing lock conflicts
- IDS Utilities and locking
- IDS 11: Optimistic concurrency – your new friend

Database logging modes I

Database logging mode	**Create statement**	**Default isolation level**	**Switch DB loggging mode ?**
No logging	create database stores_demo;	Dirty read (the **only one** available)	Yes
Buffered logging	create database stores_demo with buffered log;	Committed read	Yes
Unbuffered logging	create database stores_demo with log;	Committed read	Yes
Mode ANSI	create database stores_demo with log mode ansi;	Repeatable read	No

Database logging modes II

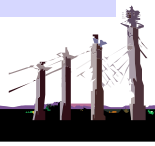
- How to discover the current logging mode of a database ?
  - onmonitor -> [status] -> databases
    - **N** -> No logging
    - **B** -> Buffered logging
    - **U** -> Unbuffered logging
    - **U*** ->Unbuffered logging mode ANSI
  - sysmaster query:

```
select name, is_logging, is_buff_log, is_ansi
    from sysmaster:sysdatabases;
```

  - Database logging mode changes will be documented in the **online.log**

Database logging modes III

- How to change the logging mode of a database ?
  - ondblog
    - *ondblog* ***nolog*** *<db>*
    - *ondblog* ***buf*** *<db>*
    - *ondblog* ***unbuf*** *<db>*
    - *ondblog* ***ansi*** *<db>*
    - *ondblog* ***cancel*** *<db>*
  - ontape
    - *ontape* ***–N*** *<db>*
    - *ontape* ***–B*** *<db> [-s –L 0]*
    - *ontape* ***–U*** *<db> [-s –L 0]*
    - *ontape* ***–A*** *<db> [-s –L 0]*
  - dbexport/dbimport for Mode ANSI databases



Database logging modes IV

- What logging mode changes are supported and do they require a backup ?

Initial state	No Logging	Buffered logging	Unbuffered Logging	Mode ANSI
No Logging		OK+Backup	OK+Backup	OK+Backup
Buffered Logging	OK		OK	OK
Unbuffered Logging	OK	OK		OK
Mode ANSI	Invalid	Invalid	Invalid	

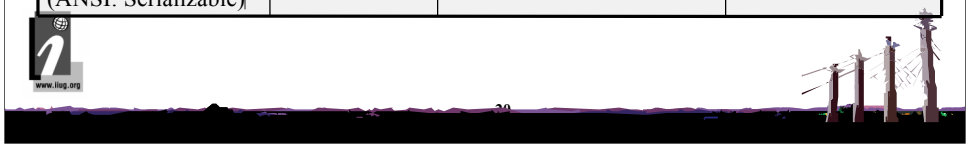
- Lock types
- Lock granularities
- Database logging modes
- **Isolation levels**
- Dynamic lock allocation in IDS
- Lock wait time
- Deadlocks
- Analyzing lock conflicts
- IDS Utilities and locking
- IDS 11: Optimistic concurrency – your new friend

Isolation levels I

Informix SQL	ANSI SQL	Remarks
Dirty read *set isolation to dirty read [retain update locks]*	**Read uncommitted** *set transaction isolation level read uncommitted*	No locks are placed during reading data and no locks from other sessions will block this reader (Exception: retain update locks)
Committed read *set isolation to committed read [retain update locks]*	**Read committed** *set transaction isolation level read committed*	Checks for locks being held by other sessions, but does not place a lock itself (Exception: retain update locks)
Cursor stability *set isolation to cursor stability [retain update locks]*	**Not available**	A share or update lock is placed on the current fetched row. It will be promoted to an exclusive lock if the update is actually performed.
Repeatable read *set isolation to repeatable read*	**Serializable (and Repeatable read)** *set transaction isolation level serializable*	A share lock is placed on every row read to make sure that this query returns the same result set if it is being re-executed in the same transaction

Isolation levels II

Isolation level	Dirty reads possible ?	Nonrepeatable reads possible ?	Phantom reads possible ?
Dirty read (ANSI: Read uncommitted)	Yes	Yes	Yes
Committed read (ANSI: Read Committed)	No	Yes	Yes
Cursor Stability (ANSI: Not supported)	No	Yes	Yes
Repeatable Read (ANSI: Serializable)	No	No	No



Isolation levels III

- Differences between the Informix '*set isolation to…*' and the ANSI '*set transaction isolation…*' statements:
  - Informix does **not support** an ANSI *'repeatable read'* isolation level. The Informix *'repeatable read'* isolation level equals to ANSI *'serializable'*
  - Informix implementation is **session based**. Once set, the isolation level is used until a new '*set isolation to…'* statement is executed or until the end of the session
  - ANSI implementation is **transaction based**. Once set, it is only active for the current transaction and will be automatically **reset** to the **default** as soon as the transaction ends
  - Informix implementation allows to **switch** between different isolation levels **inside** a transaction
  - ANSI implementation does **not** allow to switch between isolation levels in the **same** transaction

Isolation levels IV

- How to determine the current isolation level of a database session ?

1. *onstat –g sql [sid] / onstat –g ses <sid>*
   - ***DR**=Dirty read*
   - ***CR**=Committed read*
   - ***CS**=Cursor stability*
   - ***RR**=Repeatable read*
   - ***DRU**=Dirty read retain update locks*
   - ***CRU**=Committed read retain update locks*
   - ***CSU**=Cursor stability retain update locks*
   - ***LC**=Last Committed*
2. *select od.odb_sessionid, od.odb_dbname, ft.txt*
   from sysmaster:sysopendb od, sysmaster:flags_text ft
   where od.odb_isolation = ft.flags
   and ft.tabname = "sysopendb";

Isolation levels V

- Changing the default isolation level **without** modifying the application
  - **> IDS 9.21 < IDS 11**

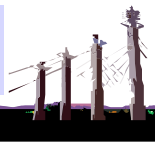
```
create procedure p_isol_level()
        set isolation to dirty read;
end procedure;
create trigger t_cust
        select on customer
        before ( execute procedure p_isol_level() );
```

  - **>= IDS 11**
    - For **all** users

```
create procedure "public".sysdbopen()
        set isolation to dirty read;
end procedure;
```

    - For **individual** users

```
create procedure "<user>".sysdbopen()
        set isolation to dirty read;
end procedure;
```



- Lock types
- Lock granularities
- Database logging modes
- Isolation levels
- **Dynamic lock allocation in IDS**
- Lock wait time
- Deadlocks
- Analyzing lock conflicts
- IDS Utilities and locking
- IDS 11: Optimistic concurrency – your new friend

Dynamic lock allocation I

- Max. configurable size of IDS lock table is:
  - IDS **32**-Bit
    - 8.000.000 locks
  - IDS **64**-Bit
    - 500.000.000 locks
- IDS automatically **doubles** the size of the lock table up to 15 times if the lock table becomes full:
  - Each lock table increase is **limited** to **100.000** (this has been changed to **1.000.000** at least in 10.00.[UF]C6 but was never documented by IBM):
    - **32-Bit:** 8.000.000 + (15x100.000) = **9.500.000**
    - **64-Bit:** 500.000.0000 + (15x100.000) = **501.500.000**

Dynamic lock allocation II

- Every time IDS **increases** the size of the lock table, a **message** will be written to the **online.log**

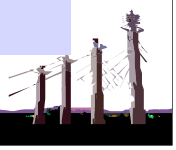
```
13:02:18  dynamically allocated 32000 locks
13:02:19  dynamically allocated 64000 locks
13:02:20  dynamically allocated 100000 locks
13:02:22  dynamically allocated 100000 locks
```

- Checking for sessions holding **many locks**

```
select ss.sid, ss.username, ss.pid, ssp.lockreqs, ssp.locksheld
from syssessions ss, syssesprof ssp
where ss.sid = ssp.sid
order by ssp.locksheld desc;
```

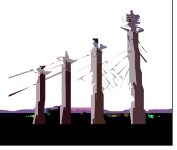
- Determine the **current size** of the lock table

```
select sh_maxlocks
from sysmaster:sysshmvals;
```



Dynamic lock allocation III

- Currently **all sessions** are affected if a single session exhausts the lock table
  - every session that wants to allocate an additional lock receives a “lock table overflow” error and does – depending on the application code – probably perform a **rollback**
  - no **new** sessions can connect to any database in the IDS server because a share lock on the database could not be acquired anymore
  - the database server is virtually frozen, no new transactions (exception: **dirty read** access) are possible until the rollback of the causing application is finished
- An IDS feature request has been formulated in order to allow the DBA to **limit** the max. number of locks a **single session** is allowed to allocate. Hopefully we will see it in the next IDS release



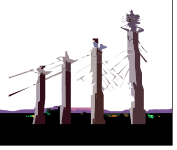
- Lock types
- Lock granularities
- Database logging modes
- Isolation levels
- Dynamic lock allocation in IDS
- **Lock wait time**
- Deadlocks
- Analyzing lock conflicts
- IDS Utilities and locking
- IDS 11: Optimistic concurrency – your new friend

Lock wait time I

- Every session can set an individual period of time to **wait on a lock** before IDS returns an error code to the application:
  - Set lock mode to **not wait**;
  - Set lock mode to **wait**;
  - Set lock mode to **wait <#sec>**;
- The **default** is NOT WAIT which means that IDS **immediately** return an error to the application if a lock could not be aquired:

-244: Could not do a physical-order read to fetch the next row
107: ISAM error: record is locked

- For **distributed** transaction the maximum lock wait time is specified thru the onconfig parameter DEADLOCK_TIMEOUT



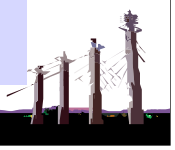
Lock wait time II

- An unlimited lock wait time is **not** a good idea for OLTP environments as it **increases** the possibility of **lock wait** situations and **deadlocks**
- Good practice in OLTP environments is setting a lock wait time between **5-10** seconds
- The IDS 11 **sysdbopen()** procedure might be used to set up a reasonable default lock wait time:

```
create procedure "public".p_isol_level()
   set lock mode to wait 5;
end procedure;
```

- The current lock wait time could be determined by:

```
1.  onstat –g sql [sid] / onstat –g ses <sid>
2.  select rst.sid, rst.username, txp.lkwait
        from sysmaster:sysrstcb rst, sysmaster:systxptab txp
        where rst.address = txp.owner;
```



Lock wait time III

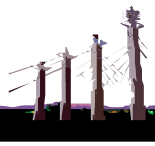
- Exclusive is **not** always exclusive:

```
begin;
Started transaction.

lock table customer in exclusive mode;
Table locked.

alter table customer modify (lname char (20));
  242: Could not open database table (eherber.customer).
  106: ISAM error:  non-exclusive access.
```

  - The problem is that another session has currently a **select cursor open** on this table
  - You could set the environment variable IFX_DIRTY_WAIT in order to **wait** until this session has **closed** the select cursor or the specified number of seconds has elapsed
    - *export IFX_DIRTY_WAIT=<#seconds_to_wait>*



Lock wait time IV

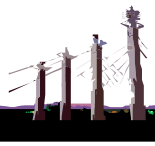
- Identifying the session that has an **open cursor** on the table you wish to alter:

```
# Retrieving the partition number
select hex(partnum)
from systables
where tabname = "customer";
(expression)
0x001002C1

# Identifying the thread id that has opened this partition
onstat -g opn | egrep -i "^tid|1002C1"
tid  rstcb      isfd  op_mode    op_flags   partnum    ucount ocount lockmode
48   0x44812578 3     0x00000400 0x00000017 0x001002c1 1      1      0

# Identifying the session that has an open cursor on this partition
onstat -u | egrep -i "^address|44812578"
address  flags    sessid   user     tty      wait     tout locks nreads
nwrites
44812578 Y--P---- 27       informix 10       456e7760 0    1     75
0
```

- Lock types
- Lock granularities
- Database logging modes
- Isolation levels
- Dynamic lock allocation in IDS
- Lock wait time
- **Deadlocks**
- Analyzing lock conflicts
- IDS Utilities and locking
- IDS 11: Optimistic concurrency – your new friend



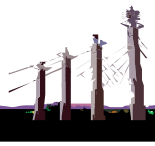
Deadlocks I

- A deadlock occurs if two sessions hold a lock and each session wants to acquire a lock that the other sessions already owns
- A deadlock is **automatically** solved by IDS
  - before granting a new lock, IDS scans the internal lock table and delivers ISAM error code **143** to one of the involved sessions if it detects a possible deadlock situation
- Deadlock statistics can be found in the following sysmaster tables:
  - **sysprofile** (Total number of deadlocks detected, see 'onstat –p')
  - **syssesprof** (Number of deadlocks per individual session)
  - **sysptprof** (Number of deadlocks per individual table)

Deadlocks II

- Analyzing deadlocks:
  - Add command **onstat –g ses 0** to $INFORMIXDIR/etc/evidence.sh
  - Activate trap mode: **onmode –I 143**
  - Wait for a deadlock situation
  - Deactivate trape mode: **onmode –I**
  - Analyze **af-File** generated by IDS
- onconfig parameter DEADLOCK_TIMEOUT
  - will only be used in **distributed** transactions
  - Specifies the upper limit of seconds that the local IDS instance will wait to acquire a lock on a **remote** IDS instance before assuming that a deadlock has occurred
  - ISAM error code **–154** (deadlock timeout expired) will be delivered to the application if the time limit will be **exceeded**

- Lock types
- Lock granularities
- Database logging modes
- Isolation levels
- Dynamic lock allocation in IDS
- Lock wait time
- Deadlocks
- **Analyzing lock conflicts**
- IDS Utilities and locking
- IDS 11: Optimistic concurrency – your new friend



Analyzing lock conflicts I

- Lock wait situations can **only** occur if the application has **executed** one of the following statements:
  - '*set lock mode to wait'* or
  - '*set lock mode to wait <#sec>*'
- Otherwise no lock wait situation will occur, because IDS will **immediately** return an error code to the application if a lock could not be granted
- Execute '*onstat –u'* and pay particular attention to the following columns for each database session:
  - **locks**: Number of locks currently held by this session
    - A session holding a **large number of locks** might probably be the causer of lock wait situations. However this must not always be the case
  - **flags**: Check for an **'L'** in the first position of the flags column
    - Sessions marked with a capital **'L'** are currently waiting for a lock to be released

Analyzing lock conflicts II

- To find the **holder** of the lock on which a session is **waiting** on, we need to execute *'onstat –k'* which shows a list of all locks currently set:
  - write down the **wait** column of the 'onstat –u' output of the session that is waiting for a lock (the one that has an 'L' in the first position of the flags column)
  - run 'onstat –k | **perl** –ane 'print if $F[0] eq "<wait_column>"' **or** 'onstat –k | **awk** '$1 ~/<wait_column>/ {print}'
  - Write down the **owner** column (position 3) of the row returned
  - Execute 'onstat –u | grep <owner_column>'
- Step 4 will give you the session **holding** the lock. If this session has also an **'L'** in the flags column, you might need to **repeat** those 4 steps in order to identify the next lock holder
- Take a deeper look at the **lockwt** Esql/C utility which queries the sysmaster database to present you a detailed overview about current lock wait situations in your IDS instance:
  - http://www.herber-consulting.de/drupal/?q=lock-wait

Analyzing lock conflicts III

```
1. onstat -u | egrep "^address|L-"
address  flags   sessid  user     tty      wait     tout locks nreads
   nwrites
448136d0 L--PR-- 36       eherber  11       440fe25c -1   1     11       0
```

```
2. onstat -k | egrep  "^address|440fe25c"
address  wtlist   owner    lklist   type     tblsnum  rowid    key#/bsiz
440fe25c 448136d0 44812b40 440fe13c HDR+X    1002c1   10c         0
```

```
3. onstat -u | egrep "^address|44812b40"
address  flags   sessid  user     tty      wait     tout locks nreads
   nwrites
44812b40 Y-BP--- 34       informix 6        45baf840 0    3     28       0
```

```
4. onstat -g sql 34
Sess  SQL            Current            Iso Lock       SQL  ISAM F.E.
Id    Stmt type      Database           Lvl Mode       ERR  ERR  Vers  Explain
34    -              stores_demo        CR  Not Wait   0    0    9.24  Off
Last parsed SQL statement :
  update customer set fname = "eric" where customer_num = 112
```

Analyzing lock conflicts IV

- Interesting columns in '*onstat –k*' output

```
onstat -k:
   address  wtlist   owner      lklist   type      tblsnum  rowid    key#/bsiz
1. 440fde9c 0        44810e58   0        HDR+S     100002   20b      0
   440fdefc 0        448119e8   0             S    100002   20b      0
2. 440fe13c 0        44812b40   440fe0dc HDR+IX    1002c1   0        0
```

- This is a **share** lock (HDR+**S**) on a database (**tblsnum=100002)**. You can identify the database using the rowid:
  - *select rowid, hex(rowid), name from **sysmaster:sysdatabases** where **hex(rowid)** like **upper("%20b%");***
- This is an **intent-exclusive** (HDR+**IX**) on a table (**rowid=0**). You can identify the table using the **tblsnum**:
  - *oncheck –pt **0x0**1002c1*

Analyzing lock conflicts V

```
onstat -k:
   address  wtlist   owner      lklist   type      tblsnum  rowid    key#/bsiz
3. 440fe25c 448136d0 44812b40   440fe13c HDR+U     1002c1   10c        0
4. 440fe49c 0        44813c98   440fe3dc HDR+X     1002c3   0          0
   440fe61c 0        44812578   440fe01c HDR+IX    1002cf   0          0
5. 440fe67c 0        44812578   440fe61c HDR+S     1002cf   100        0
6. 440fe6dc 0        44812578   440fe67c HDR+X     1002d0   135      K- 1
```

- This is an **update** lock (HDR+**U**) on an individual row (**rowid=10c)**. You can identify the table using the tblsnum and the row thru the hexadecimal rowid
- This is an **exclusive** (HDR+**X**) on a table (**rowid=0)**
- This is a **share** (HDR+S) lock on a page (**rowid=100**). If the rowid is ending with double zeros, it is a page lock
- This is an **exclusive** lock on an individual index item belonging to the **first index** (**K- 1**) of the table with tblsnum 1002d0

Analyzing lock conflicts VI

```
Output from lockwt:
WAIT    SID   :PID    PROCNAME     USERNAME  LKTYPE  DATABASE:TABLENAME
   LKOBJ
0 -     200   :12303  workprocess3 dbuser    X       rome     :orders          row
1 W     100   :23613  batchp12     dbuser            rome     :orders
```

- In this example session **200** (process "workprocess3") is holding an **exclusive** lock on an individual **row** in the table orders. Session **100** is **waiting** for this lock to be released. You need to analyze session 200 by executing '***onstat -g ses 200***'

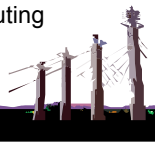
Analyzing lock conflicts VII

```
Output from lockwt:
WAIT   SID  :PID   PROCNAME     USERNAME LKTYPE DATABASE:TABLENAME
   LKOBJ

0 W      400:   -1 (remote)     eherber1 X      rome    :status          row
1 W      300: 3140 batchp3      dbuser          rome    :status

0 -      500:   -1 (remote)     eherber1 X      rome    :customer_order  row
1 W      400:   -1 (remote)     eherber1        rome    :customer_order
```

- Session **300** is **waiting** for session **400** to release the exclusive lock on table status. But there is a **second pair** of locks. Session **400** is **waiting** for session **500** which holds a lock on table customer_order.
- This is a typical **escalating** lock situation, because session 400 is holding a lock another session is waiting for, but session 400 is **also waiting** for a lock to be released. Analyze what session 500 is doing by executing '***onstat -g ses 500***'



- Lock types
- Lock granularities
- Database logging modes
- Isolation levels
- Dynamic lock allocation in IDS
- Lock wait time
- Deadlocks
- Analyzing lock conflicts
- **IDS Utilities and locking**
- IDS 11: Optimistic concurrency – your new friend

Utilities and locking I

Utility	Type of lock	Remarks
dbexport/dbimport	exclusive	Not configurable, dbexport/dbimport will always place an **exclusive** lock on the database
onunload	share	Places a **share** lock on the table that is being processed. If onunload is run against a database, every table will be locked in share mode
load/unload	none	**No** lock will be placed on the table. User must **explicitely** lock table if desired
dbload	none/exclusive	If switch **–k** is used, an **exclusive** lock will be placed on the table during the load. If switch **–r** is used, only individual row locks will be set (default).
oncheck –cd/-cD	share	A share lock is placed on the table/fragment that is checked for consistency
oncheck –ci/-cI	none/share	If table is configured for **row** locking, **no** lock will be placed on it during the check. If table uses **page** locking or switch **–x** is used, a **share** lock will be placed on the table during the check

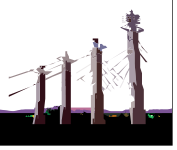
Utilities and locking II

Utility	Type of lock	Remarks
oncheck -pt	none	**No** lock will be placed on table
oncheck -pT	share	**Share** lock will be placed on table. This command can take some time on big tables with many indexes
onpladm (deluxe mode)	none	**No** lock will be placed on table during load/unload in deluxe mode
onpladm (express mode)	none/exclusive	**Loading** in express mode will place an **exclusive** lock on the table. **Unloading** in express mode will **not** place a lock on the table

- Lock types
- Lock granularities
- Database logging modes
- Isolation levels
- Dynamic lock allocation in IDS
- Lock wait time
- Deadlocks
- Analyzing lock conflicts
- IDS Utilities and locking
- **IDS 11: Optimistic concurrency – your new friend**

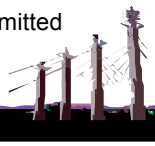
IDS 11: Optimistic concurrency I

- If you want to read row that is currently **exclusively** locked by another session, you might end up with:
  - a possible **inconsistent** version of the row (*dirty read*)
  - a **lock-wait** situation (*isolation level != dirty read* and *set lock mode to wait [#sec]* set)
  - an **error message** that the row is currently locked (*isolation level != dirty read* and *set lock mode to not wait* set)
- IDS 11 however is able to perform a **time travel** and deliver the **last consistent** version of the row if:
  - the table has been configured for **row-level**- not page-level-locking
  - you are trying to **read** the row, not update or delete it
  - the new USELASTCOMMITTED feature has been **enabled** and your current isolation level ist *dirty read* or *committed read*
- This feature ensures that writers don't block readers



IDS 11: Optimistic concurrency II

- Three ways to **enable** optimistic concurrency:
  1. set isolation to committed read last committed;
  2. onconfig parameter: USELASTCOMMITTED [dirty read|committed read|all|none]
  3. set environment 'USELASTCOMMITTED' [dirty read|committed read|all|none];
- USELASTCOMMITTED settings:
  - **dirty read:** IDS will automatically add the 'last committed' option if dirty read is active
  - **committed read:** IDS will automatically add the 'last committed' option if committed read is active
  - **all:** IDS will automatically add the 'last committed' option for dirty read and committed read
  - **none:** IDS will not change the normal dirty read or committed read behaviour (default)



IDS 11: Optimistic concurrency III

- Enabling optimistic concurrency for a user 'eherber' for committed reads:

```
create procedure "eherber".sysdbopen()
   set environment uselastcommitted "committed read";
end procedure;
```

- Enabling optimistic concurrency for **all** users for dirty read and committed read:

1. Database level

```
create procedure "public".sysdbopen()
   set environment uselastcommited "all";
end procedure;
```

2. Instance level

onconfig: ***USELASTCOMMITTED all***

online: ***onmode –wf USELASTCOMMITTED=all***

Logos might change over years - but superb technology lives on…

Thank you for your attention !!