Understand locking behavior and analyze locking conflicts in IDS

* Part 1: Concepts and mechanisms
* [Part 2: Tracking and analysis](http://www.ibm.com/developerworks/data/library/techarticle/dm-0701herber/index.html?ca=drs-)

However, locking often results in conflicts and wait situations. These are one of the most common problem areas a DBA is faced with during his daily work. Analyzing locking problems without the appropriate scripts in place is complex and error-prone.

Examples are based on the stores\_demo database that can be created by executing the following command:

* dbaccessdemo stores\_demo -log

Lock types

IDS knows several different types of locks. Those types are as follows:

Shared lock

Shared locks could be placed on rows that do not have an exclusive lock on it. Other users might place additional shared locks or update locks on the same row, but no other exclusive locks are allowed.

Update lock

An update lock is a special kind of lock generated by a cursor that has been declared with the for update clause. Update locks could only be placed on a row that currently has no update or exclusive lock on it. Once an update lock has been placed on a row, it is promoted to an exclusive lock as soon as the row is updated.

Exclusive lock

An exclusive lock 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. It is exclusively reserved for this database session.

Intent lock

Intent locks are special kinds of locks. If, for example, a row is updated, an exclusive lock is placed on the row and an intent exclusive lock is placed on the table. The intent exclusive table lock ensures that no other session could place a share or exclusive lock on the table as long as individual rows in the table have been exclusively locked.

Locking granularity

IDS allows application developers to place locks on different objects.

Database locks

Databases could be locked explicitly in exclusive or share mode. An exclusive lock prevents anybody else from accessing the database. A share lock allows concurrent users to read and update data from this database but prevents the placement of an exclusive lock on this database.

* **Database share lock**

A share lock is automatically placed on the database as soon as you open the database. This ensures that no other session could place an exclusive lock on the database or drop the database. Utilities like **onunload** might also place a share lock on the database.

* + SQL Statement :
    - database stores\_demo;

Listing 1. Share lock on a database

Output from onstat -k:

----------------------

Locks

address wtlist owner lklist type tblsnum rowid key#/bsiz

300583dc 0 400d63d8 0 HDR+S 100002 207 0

|  |  |
| --- | --- |
|  |  |

Here you see a share lock (HDR+S) on a database. tblsnum=100002 represents the database tablespace. rowid=207 is the hexadecimal row ID of the row describing this database in table **sysdatabases** in the sysmaster database. Table **sysmaster:sysdatabases** represents the so called database tablespace.

Databases and their hexadecimal row IDs could be selected with the following SQL query:

Listing 2. Query on database sysmaster retrieving the row ID of a database tablespace entry

|  |  |
| --- | --- |
|  | Query:  ------  database sysmaster;  select name, hex(rowid) hex\_rowid from sysdatabases;  Result:  -------  name hex\_rowid  linux\_mag 0x00000204  onpload 0x00000206  stores\_demo 0x00000207  sysmaster 0x00000201  sysuser 0x00000203  sysutils 0x00000202  tpcb 0x00000208 |

* **Database Exclusive Lock**

An exclusive lock can be explicitly set on a database. Utilities like **dbexport** might also place an exclusive lock on the database to be exported.

* + **SQL Statement**
    - database stores\_demo **exclusive**

Listing 3. Exclusive lock on a database

|  |  |
| --- | --- |
|  | Output from onstat -k:  ----------------------  Locks  address wtlist owner lklist type tblsnum rowid key#/bsiz  300583dc 0 400d63d8 0 HDR+X 100002 207 0 |
|  |  |

Here you see an exclusive lock (HDR+X) on the database with the hexadecimal rowid=207.

Table locks

Like databases, tables could either be locked in exclusive or share mode. An exclusive lock prevents anybody else from reading or changing data in this table. An exception to this are sessions running with an isolation level of [dirty read](https://www.ibm.com/developerworks/data/library/techarticle/dm-0609herber/index.html#dirty-read) (read uncommitted). They are still able to read (possible inconsistent) data from the exclusively locked table. Locking a table in share mode allows others to select data from this table but prevents data modifications.

* **Table share lock**

A share lock can be explicitly placed on a table. Utilities like onunload or oncheck (depending if the locking mode of the table is page or row) might also place a share lock on the table they operate on.

* + SQL Statement
  + Database stores\_demo;
    - begin;
    - lock table orders in **share** mode;

Listing 4. Share lock on a table

|  |  |
| --- | --- |
|  | Output from onstat -k:  ----------------------  Locks  address wtlist owner lklist type tblsnum rowid key#/bsiz  300582e0 0 400d63d8 300583dc HDR+S 10013b 0 0 |

You see a share lock (HDR+S) and a row ID of zero (rowid=0). A zero row ID represents a table lock. The tablespace number (tblsnum=10013b) conforms to the hexadecimal partition number of this table.

The affected table could be identified in two ways:

* + oncheck command
  + Data Dictionary Query command

Listing 5. Identifying a table through its hexadecimal partition number

|  |  |
| --- | --- |
|  | Dictionary Query:  -----------------  database stores\_demo;  select st.tabname, dbinfo("dbspace", st.partnum), hex(st.partnum) from systables st where hex(st.partnum) = "0x0010013B";  Result:  ------  tabname dbspace hex\_partnum  orders rootdbs 0x0010013B  oncheck command:  ----------------  oncheck -pt 0x0010013b  Result:  -------  TBLspace Report for stores\_demo:informix.orders  Physical Address 1:1988  Creation date 08/21/2006 20:07:41  TBLspace Flags 801 Page Locking  TBLspace use 4 bit bit-maps  Maximum row size 80  Number of special columns 0 ... ... |

Notice that you need to put a 0x00 in front of the hexadecimal number gathered from the onstat -k output, and that you need to convert the letters to uppercase (10013 -&gt; **0x00**10013**B**) in order to find the table through the dictionary query.

If it is a fragmented table you have to query, the **sysfragments** dictionary table:

Listing 6. Identifying a fragmented table through its hexadecimal partition number

|  |  |
| --- | --- |
|  | Query:  ------  database stores\_demo;  select st.tabname, dbinfo("dbspace", sf.partn), hex(sf.partn) from systables st, sysfragments sf where st.tabid = sf.tabid and sf.fragtype = "T" and hex(sf.partn) = "0x0010013B";  Result:  -------  No rows found. |

The table with partition number 0x0010013B in this example is **not** fragmented, so there exists no entry in dictionary table **sysfragments** for it. However, fragmented tables have entries in sysfragments, so you are able to identify them with the above mentioned dictionary query.

The oncheck -pt &lt;hex\_partnum&gt; does **not** work for fragmented tables because it does not display the real tablename, only the partition information.

* **Table exclusive lock**

An exclusive lock can be explicitly set on a table or might be implicitly set by statements like alter table.

* + SQL Statement
    - begin;
    - lock table orders in **exclusive** mode;

Listing 7. Exclusive lock on table

|  |  |
| --- | --- |
|  | Output from onstat -k:  ----------------------  Locks  address wtlist owner lklist type tblsnum rowid key#/bsiz  300582e0 0 400d63d8 300583dc HDR+X 10013b 0 0 |
|  |  |

Here you see an exclusive lock (HDR+X) on the table with the hexadecimal tblsnum=10013b.

Page or row locks

The lock granularity (page or row) could be specified during the creation of a table or later be changed with the alter table statement.

* New Table
  + create table t1 (f1 int) lock mode **(row);**
* Existing Table
  + alter table t1 modify lock mode **(page);**

The default lock mode, which is used if no lock mode is specified during the creation of a table, could be configured with the onconfig parameter DEF\_TABLE\_LOCKMODE or through the environment variable IFX\_DEF\_TABLE\_LOCKMODE.. The lock mode setting in a create table statement takes precedence over the settings of the IFX\_DEF\_TABLE\_LOCKMODE environment variable and the DEF\_TABLE\_LOCKMODE configuration parameter.

Page locking means that IDS always locks a whole database page instead of an individual row even if only a single row is modified. Depending on the row size and configured page size of the underlying dbspace, this behavior might result in reduced concurrency. It is important to notice that page level locking not only affects data pages but also index pages. This further reduces concurrency because index pages normally hold a larger number of entries compared to data pages.

You can determine the current lock mode of a table in two ways:

Listing 8. Determining the lock mode of a table using oncheck (a) or a dictionary query (b)

|  |  |
| --- | --- |
|  | a) Command:  -----------  oncheck -pt stores\_demo:informix.orders  Output:  -------  TBLspace Report for stores\_demo:informix.orders  Physical Address 1:3905  Creation date 05/08/2006 16:47:43  TBLspace Flags 801 Page Locking ... ...  b) Query:  ---------  database stores\_demo;  select tabname, locklevel from systables where tabname = "orders";  Result:  -------  tabname locklevel  orders P |

The following abbreviations for the locklevel are shown by IDS:

* **B** -- Views (Check the locklevel of underlying tables)
* **P** -- Page locking
* **R** -- Row locking

Generating the necessary alter table commands for converting all tables to row locking can be done with the following meta SQL statement:

Listing 9. Using meta SQL to convert all tables to row locking

|  |  |
| --- | --- |
|  | Meta SQL:  ---------  database stores\_demo;  output to alter\_table.sql without headings  select "alter table '" || trim(owner) || "'." || trim(tabname) || " lock mode(row);" from 'informix'.systables where tabid &gt; 99 and tabtype = 'T' and locklevel = 'P';  Generated output file  'alter\_table.sql':  ----------------------------------------  alter table 'informix'.customer lock mode(row);  alter table 'informix'.orders lock mode(row); ... ... |

* **Page lock**

Page locks are placed on a database page during the processing of a row if the table has been configured for page level locking. They could be of type exclusive (insert, update, or delete) or share (select with an isolation level of repeatable read).

* + SQL Statement
    - begin;
    - update orders set ship\_charge = (ship\_charge\*1.2) where order\_num = 1005

Listing 10. Exclusive lock on a page

|  |  |
| --- | --- |
|  | Output from onstat -k:  ----------------------  Locks  address wtlist owner lklist type tblsnum rowid key#/bsiz  440cf9bc 0 4506c3d0 0 HDR+S 100002 207 0  440cfa14 0 4506c3d0 440cf9bc HDR+IX 10013b 0 0  440d082c 0 4506c3d0 440cfa14 HDR+X 10013b 100 00 |

This example shows three locks:

* + - Share lock (HDR+S) on a database. tblsnum=100002 is the partition number of the database tablespace. The share lock prevents other transactions from locking the database exclusively.
    - Intent exclusive (HDR+IX) lock on a table. tblsnum=0 represents a table lock. The intent exclusive lock prevents other transactions from locking the table in share or exclusive mode.
    - Exclusive lock on a page (rowid=100. Page locks are represented with a double zero (00) at the end of a row ID. The exclusive lock prevents other transactions from placing share or exclusive locks on this page.
* **Row lock**

Row locks are placed on a database page during the processing of a row if the table has been configured for row-level locking. They could be of type exclusive (insert, update, or delete) or type share (select with an isolation level of [repeatable read](https://www.ibm.com/developerworks/data/library/techarticle/dm-0609herber/index.html#repeatable-read)).

* + SQL Statement
    - begin; set isolation to repeatable read; select \* from customer where customer\_num = 110

Listing 11. Share lock on a row

|  |  |
| --- | --- |
|  | Output from onstat -k:  ----------------------  Locks  address wtlist owner lklist type tblsnum rowid key#/bsiz  440cf9bc 0 4506c3d0 0 HDR+S 100002 207 0  440cfa14 0 4506c3d0 440cf9bc HDR+IS 100139 0 0  440cfd84 0 4506c3d0 440d082c HDR+S 100139 10a 0  440d082c 0 4506c3d0 440cfa14 HDR+SR 10013a 10a K- 1 |

This example shows four locks:

* + - Share lock (HDR+S) on a database. tblsnum=100002 is the partition number of the database tablespace. The share lock prevents other transactions from locking the database exclusively.
    - Intent share (HDR+IS) lock on a table. tblsnum=0 represents a table lock. The intent share lock prevents other transactions from locking the table in exclusive mode.
    - Share lock on an individual row (rowid=10a). The share lock on the row prevents other transactions from placing an exclusive lock on the same row.
    - Share lock on index key (K-1). The share lock on the index key prevents other transactions from placing an exclusive lock on the same index key.

You are able to identify the locked row with the following data dictionary query:

Listing 12. Identifying a locked row

|  |  |
| --- | --- |
|  | Query:  ------  database stores\_demo;  set isolation to dirty read;  select \* from customer where hex(rowid) = "0x0000010A";  Result:  -------  customer\_num 110 fname Roy ... ... |

Index key locks

IDS also places locks on index keys like data rows in order to protect those index keys. Key value locking guarantees that unique keys remain unique because it doesn't allow a second transaction to insert a key with the same value until the transaction that deleted the unique key is committed.

Index key locks are identified by a K in the key#/bsiz column of the onstat -k output.

* Default isolation level
  + committed read
* The default isolation level for this type of database is [committed read](https://www.ibm.com/developerworks/data/library/techarticle/dm-0609herber/index.html#committed-read), but the user or application could select another one with the set isolation to &lt;isol\_level&gt;
* statement.
* SQL statement
  + create database stores\_demo with **buffered log**

Unbuffered logging

It is like buffered logging, but the log buffer is immediately flushed to disk as soon as a commit statement is received. You can't loose any transactions with this logging mode.

* Default isolation level
  + [committed read](https://www.ibm.com/developerworks/data/library/techarticle/dm-0609herber/index.html#committed-read)
* A user or application could select a different isolation level with the set isolation to &lt;isol\_level&gt; statement.
* SQL statement
  + create database stores\_demo with **log**

Log mode ANSI

The default isolation level for this type of database is [repeatable read](https://www.ibm.com/developerworks/data/library/techarticle/dm-0609herber/index.html#repeatable-read), which means that a shared lock is placed on every row processed. This behavior could lead to locking conflicts and timeouts. The isolation level could be switched with the set isolation to &lt;isolation\_level&gt; statement.

* Default isolation level
  + [repeatable read](https://www.ibm.com/developerworks/data/library/techarticle/dm-0609herber/index.html#repeatable-read)
* A user or application could select a different [isolation level](https://www.ibm.com/developerworks/data/library/techarticle/dm-0609herber/index.html#isolation-level) with the set isolation to &lt;isol\_level&gt; statement.
* SQL statement
  + create database stores\_demo with **log mode ansi**

Database logging modes and default isolation levels

| **Database logging mode** | **Create statement** | **Default isolation level** |
| --- | --- | --- |
| [No logging](https://www.ibm.com/developerworks/data/library/techarticle/dm-0609herber/index.html#no-log) | create database stores\_demo | [Dirty read](https://www.ibm.com/developerworks/data/library/techarticle/dm-0609herber/index.html#dirty-read) |
| [Buffered logging](https://www.ibm.com/developerworks/data/library/techarticle/dm-0609herber/index.html#buffered-log) | create database stores\_demo with buffered log | [Committed read](https://www.ibm.com/developerworks/data/library/techarticle/dm-0609herber/index.html#committed-read) |
| [Unbuffered logging](https://www.ibm.com/developerworks/data/library/techarticle/dm-0609herber/index.html#unbuffered-log) | create database stores\_demo with log | [Committed read](https://www.ibm.com/developerworks/data/library/techarticle/dm-0609herber/index.html#committed-read) |
| [Mode ANSI](https://www.ibm.com/developerworks/data/library/techarticle/dm-0609herber/index.html#mode-ansi) (unbuffered logging) | create database stores\_demo with log mode ansi | [Repeatable read](https://www.ibm.com/developerworks/data/library/techarticle/dm-0609herber/index.html#repeatable-read) |

The current logging mode of a database could be determined with either the onmonitor utility or a sysmaster query:

* onmonitor (status-databases)
  + **N** -- No logging
  + **B** -- Buffered logging
  + **U** -- Unbuffered logging
  + **U\*** -- Mode ANSI (unbuffered logging)
* sysmaster query

Listing 13. Determining the logging mode of a database

|  |  |
| --- | --- |
|  | Query:  ------  database sysmaster;  select name, is\_logging, is\_buff\_log, is\_ansi from sysmaster:sysdatabases;    Result:  -------  name stores\_ansi  is\_logging 1  is\_buff\_log 0  is\_ansi 1  name stores\_demo  is\_logging 1  is\_buff\_log 0  is\_ansi 0 |

You can switch the logging mode of a database with the **ontape** or **ondblog** utility.

Note that IDS does not allow you to switch a mode ANSI database back to any other logging mode (once it is an ANSI, it is forever an ANSI).

Isolation levels

IDS provides, depending on the [logging mode](https://www.ibm.com/developerworks/data/library/techarticle/dm-0609herber/index.html#db-logging) of the database, four different isolation levels. This section discusses those isolation levels in detail.

IDS isolation levels and their ANSI counterparts

| **Informix SQL** | **ANSI SQL** |
| --- | --- |
| [Dirty read](https://www.ibm.com/developerworks/data/library/techarticle/dm-0609herber/index.html#dirty-read) | Read uncommittted |
| [Committed read](https://www.ibm.com/developerworks/data/library/techarticle/dm-0609herber/index.html#committed-read) | Read committed |
| [Cursor stability](https://www.ibm.com/developerworks/data/library/techarticle/dm-0609herber/index.html#cursor-stability) | *Not available* |
| [Repeatable read](https://www.ibm.com/developerworks/data/library/techarticle/dm-0609herber/index.html#repeatable-read) | Serializable |

Dirty read (ANSI: Read uncommitted)

If you read data with this isolation level, you will not lock anything yourself and you will not be blocked by any existing locks from other users. However, you might read inconsistent data; data that has not been committed yet.

Dirty read is the only possible isolation level for databases created [without logging](https://www.ibm.com/developerworks/data/library/techarticle/dm-0609herber/index.html#no-log).

* Informix SQL
  + set isolation to **dirty read** [retain update locks]

Committed read (ANSI: Read committed)

This isolation level ensures that only committed data is read.

Sessions running with committed read check if it is possible to set a shared lock on a row but do not set it. This ensures that no rows could be read that are currently updated in a parallel running transaction. However, after reading a row, a parallel transaction might update it because a committed read does not set any locks to prevent this.

Committed read is the default isolation level for databases created with [buffered](https://www.ibm.com/developerworks/data/library/techarticle/dm-0609herber/index.html#buffered-log) or [unbuffered logging](https://www.ibm.com/developerworks/data/library/techarticle/dm-0609herber/index.html#unbuffered-log), but not [mode ANSI](https://www.ibm.com/developerworks/data/library/techarticle/dm-0609herber/index.html#mode-ansi).

* Informix SQL
  + set isolation to **committed read** [retain update locks]

Cursor stability (ANSI: --)

An isolation level of cursor stability is only relevant for cursors declared with the FOR UPDATE clause.

IDS places an update lock on the current fetched row. If the row is updated, the lock is converted to an exclusive lock that is held until the transaction ends, independent from the current position of the cursor. If the next row is fetched without updating the current one, the update lock on the previous row is released and a new one is placed on the current row. This behavior could be changed if you add the retain update locks to the set isolation statement. In this case, an update lock on a row is not released if you fetch the next row.

Update locks could only be placed on rows that currently have no update or exclusive locks on them. However, once the update lock is placed, other sessions are not prevented from placing a shared lock on the same row. This results in an error if you try to update that row, and IDS could not promote the update lock in an exclusive lock due to the shared lock(s) on it. This is the desired behavior. If you want to be sure that you can update the row later in the transaction, you have to perform a dummy update that promotes the update lock in an exclusive lock. This prevents parallel sessions from placing shared locks on the same row.

* Informix SQL
  + set isolation to **cursor stability** [retain update locks]
* ANSI SQL
  + --

Repeatable read (ANSI: Serializable)

An isolation level of repeatable read places a share lock on every row (or page if [page level locking](https://www.ibm.com/developerworks/data/library/techarticle/dm-0609herber/index.html#lock-page-row) is active for this table) processed. This prevents other users from changing the row.

The retain update locks clause is not supported with this isolation level because repeatable read automatically retains share or update locks.

Repeatable read is the default isolation level for databases created with [mode ANSI](https://www.ibm.com/developerworks/data/library/techarticle/dm-0609herber/index.html#mode-ansi) logging.

* Informix SQL
  + set isolation to **repeatable read**
* ANSI SQL
  + set transaction isolation level **serializable**

Dynamic lock allocation

How it works

The onconfig parameter LOCKS (max. **8,000,000**) specifies the initial number of available locks for the whole instance. However, IDS comes with the dynamic lock allocation feature. If the initial number of locks has been reached, IDS automatically doubles the capacity of the internal lock table. This procedure is repeated up to 15 times. No more than 100,000 additional locks are allocated by each increase, so the maximum number of locks available in IDS is limited to 9,500,000:

8.000.000 inital + (15 x 100.000) additions = **9.500.000** total

The dynamic lock allocation feature allows an application to continue processing although the initial number of configured locks has already been reached.

Suboptimal application code

Sometimes, the dynamic allocation of locks *hides* suboptimal application code. A reasonable IDS feature request would be the ability to control the maximum number of locks that a single session is allowed to hold. This would help the DBA to identify and block badly written applications without affecting other database sessions.

Currently, if IDS reaches the upper limit of 9.500.000 locks, *all* running applications receive a lock table overflow error when requesting a lock. This means all applications are affected by a problem created by a single, badly written application. This scenario is not ideal.

Lock wait time

Settings per database session

Each database session can set an individual lock wait time to prevent an overflow from affecting all applications. The IDS default is to set the lock mode to **not wait**, meaning as soon as a lock conflict is detected, IDS delivers these error codes to the application:

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

You can specify the lock wait behavior per session with the following Informix extension to ANSI SQL:

* set lock mode to **not** wait
* set lock mode to **wait**
* set lock mode to **wait** <**#sec**>

Distributed transactions

For distributed transactions, the onconfig [DEADLOCK\_TIMEOUT](https://www.ibm.com/developerworks/data/library/techarticle/dm-0701herber/index.html#deadlock-timeout) parameter specifies the maximum amount of time that IDS will wait for a lock on a remote IDS instance before returning an error code to the application.

Recommendation

Specifying an unlimited lock wait (set lock mode to **wait**) is not the optimal solution. It increases the possibility of lock conflicts and deadlocks, as well as masking suboptimal application code.

Setting a lock wait time of 5-10 seconds is good practice in OLTP environments. If IDS is not able to acquire the lock in the specified time, an appropriate SQL error is delivered to the application. The application can now decide whether to retry the operation or rollback the current transaction.

Check session settings

You can check the current isolation level and the lock mode of database sessions with the **onstat -g sql** command:

Listing 1. Session isolation level - onstat -g sql

|  |  |
| --- | --- |
|  | Output from the onstat -g sql command:  --------------------------  Sess  SQL         Current        Iso Lock       SQL  ISAM F.E.  Id    Stmt type   Database       Lvl Mode       ERR  ERR  Vers Explain  18    -           stores\_demo    CRNot Wait   0    0    9.03  Off  16    -           stores\_demo    RRWait 10    0    0    9.03  Off |
|  |  |

This example shows two database sessions with different isolation levels:

* Session 18, **CR** Committed Read, **Not Wait**
* Session 16, **RR** Repeatable Read, **Wait 10**

Another reasonable IDS feature request would be the ability to set a default lock wait method to use if the application does not execute an explicit **set lock mode to wait...** statement on the database or instance level.

Deadlocks

A deadlock occurs when two users hold locks, and each user wants to acquire a lock that the other user owns. To prevent deadlocks, IDS scans the internal lock table before granting new locks and delivers ISAM error code **-143** (**deadlock detected**) to the affected database session.

Number of deadlocks detected

The **onstat -p** command shows the total number of deadlocks (deadlks column) detected since IDS was started or since the **onstat -z** (zero statistics) command was executed. You can determine the last reset time of the IDS statistics using this SQL statement:

* **select dbinfo("utc\_to\_datetime", sh\_pfclrtime) from sysmaster:sysshmvals;**

Sysmaster tables

Unfortunately, IDS does not provide many details about detected deadlocks and involved applications. This makes it difficult to analyze the cause of a deadlock after it's resolved. The sysmaster database contains some tables that offer additional information:

* **sysprofile**: Contains the total number of deadlocks detected which corresponds to the deadlks column in the **onstat -p** command output
* **syssesprof**: Contains the number of deadlocks detected per individual database session
* **sysptprof**: Contains the number of deadlocks detected per individual table

Tracing deadlocks

The IDS error trapping feature is also useful in analyzing deadlocks. As already stated, IDS delivers ISAM error code **-143** to the application when detecting a possible deadlock. You can activate error trapping for deadlocks with the **onmode -I 143** command. IDS creates an assertion failure file and stores the **onstat -a** output in this file as soon as a deadlock is detected. You can analyze the contents of this file and check the SQL statement that the deadlocked session was executing as well as the locks held by that session and other concurrent sessions.

Deadlock timeout

The onconfig DEADLOCK\_TIMEOUT parameter has no influence on the deadlock detection time or the amount of time an application waits in the case of a local deadlock. IDS always immediately resolves local deadlocks.

If a lock wait occurs while selecting or updating data on a remote IDS instance, DEADLOCK\_TIMEOUT specifies the upper limit of seconds that the local IDS instance will wait for the requested lock, after which it assumes that a deadlock may have occurred. ISAM error code **-154** (**deadlock timeout expired - possible deadlock**) is then delivered to the application.

As stated, IDS does not crosscheck the internal lock table between instances; it uses the DEADLOCK\_TIMEOUT parameter, which is more of a distributed lock timeout. The DEADLOCK\_TIMEOUT parameter overwrites the application specific [set lock mode to wait](https://www.ibm.com/developerworks/data/library/techarticle/dm-0701herber/index.html#lock-wait-session) setting in case of distributed transactions.

Analyze lock conflicts

Often transactions try to access the same row concurrently in an OLTP environment with hundreds of parallel executing database sessions. Therefore, it is good practice to keep transactions as short as possible to avoid lock conflicts.

If you don't execute a [set lock mode to wait](https://www.ibm.com/developerworks/data/library/techarticle/dm-0701herber/index.html#lock-wait-session) statement in your application, you will receive an SQL error from IDS as soon as you request a lock on a row which is not compatible with the lock already placed on the same row from another session. A direct abort is often not desired, so applications normally execute the **set lock mode to wait** statement which instructs IDS to suspend the database session (sqlexec thread) until the lock can be granted or the agreed lock wait time has elapsed. However, this can lead to lock wait situations that can slow down the throughput of your IDS instance. As a result, interactive users might complain of slow response times or longer batch processes. Normally, you should not set the lock wait mode to "infinite" .

Analyzing lock wait situations is a real challenge in a dynamic environment with ongoing transactions. The best method is to have some scripts in place that allow the real-time analysis of lock wait situations.

Useful onstat commands

The **onstat -u** command is a good starter for looking after lock conflicts. There are two interesting values here:

* Number of locks currently **held** by each session
  + The locks column shows you the number of locks held by this session. Sessions with a huge number here could be responsible for lock conflicts. This is not always the case, but a large number is probably an indicator of a poorly written application.
* Sessions currently **waiting** for a lock
  + Database sessions waiting for a lock are marked with an **'L'** in the first position of the flags column.

Listing 2. Locks held and sessions waiting for a lock - onstat -u

|  |  |
| --- | --- |
|  | Output from the onstat -u command:  ---------------------  address  flags  sessid user  tty wait     tout locks nreads   nwrites  4506b44c L-BPR-- 20   informix 11 440cfac4 -1   17    19       0  4506b978 Y--P--D 16   informix -  4407d138  0    0     0       0  ...  ... |
|  |  |

You can execute the **onstat -k | grep 'L-'** command to identify all sessions currently waiting for a lock. With **onstat -g ses <sessid>**, you can monitor which SQL statement this database session is currently executing. You can also see the opened database (Current Database), the current isolation level (Iso Lvl), and the agreed lock wait time (Lock Mode). In the status column, you can see the number of seconds left before IDS returns a lock timeout error to the application.

Listing 3. Analyze sessions in lock wait status - onstat -g ses <sessid>

|  |  |
| --- | --- |
|  | Output from the onstat -g ses 20 command:  -----------------------------  ...  ...  42       sqlexec  4506b44c L-BPR--  7168     sleeping(secs: 9)  ...  ...  Sess  SQL         Current            Iso Lock       SQL  ISAM F.E.  Id    Stmt type   Database           Lvl Mode       ERR  ERR  Vers Explain  20    DELETE(all) stores\_demo CRWait 60        0    0  9.03  Off  ...  ...  Current SQL statement:     delete from customer |

Next, identify the session that is holding the lock and causing the lock wait. The **onstat -k** command delivers an overview of all locks currently allocated in this IDS instance:

Listing 4. Locks currently allocated in IDS - onstat -k

|  |  |
| --- | --- |
|  | Output from the onstat -k command:  --------------------------  Locks  address  wtlist   owner    lklist   type     tblsnum  rowid    key#/bsiz  440cf9bc 0        4506b44c 0        HDR+S    100002   207         0  440cfa14 0        4506b44c 440cf9bc HDR+IX   10015f   0           0  440cfa6c 0        4506b44c 440cfa14 HDR+X    10015f   101         0  440cfac4 4506b44c 0        440cfa6c HDR+     10015f   106         0  440cfb1c 0        4506b44c 440cfa6c HDR+X    100160   101      K- 1  440cfb74 0        4506b44c 440cfb1c HDR+X    100175   101      K- 1  440cfbcc 0        4506b44c 440cfb74 HDR+X    10015f   102         0  ...  ... |

The interesting column in this output is the second one, called wtlist. It contains the hexadecimal address of the user thread waiting for a lock. However, depending on the number of locks currently active in an IDS instance, an **onstat -k** output could be quiet large. You can use one of the following commands to locate the lock on which the database session with shared memory address **4506b44c** is currently waiting on:

* AWK : **onstat -k | awk '$2 ~/4506b44c/ { print }'**
* PERL: **onstat -k | perl -ane 'print if $F[1] eq "4506b44c"'**

The owner column gives you the shared memory address of the user thread holding the lock. Identifying this session is easy using the **onstat -u** command and piping the output to the grep utility. You can analyze what the cause of the lock wait is currently doing using the **onstat -g <sessid>** command. If the cause itself is waiting for a lock ('**L**' in the first position of the flags column in **onstat -u**), you have to **repeat** the steps mentioned above to find the cause of the second lock wait situation.

Lockwt utility

Using the Esql/C lockwt utility is a comfortable way to analyze lock wait situations. To use this utility, you need to install the Informix Client SDK and a C compiler in order to compile it. The lockwt utility searches through a pair of sysmaster tables to find lock wait situations.

The program reports each user session holding a lock and the session(s) waiting for this lock to be released. Execute the **lockwt -r <#sec>** command to repeat the lock-search in the specified time-interval (similarly to the **onstat -r** command).

The lockwt utility allows the real-time monitoring of complex lock wait situations, representing the collected information in an easy-to-read format.

Listing 5. Lockwt - Description of the output format

|  |  |
| --- | --- |
|  | Output from lockwt:  -------------------  (0) (1)  (2)  (3)    (4)           (5)                 (6)          (7)        (8)         (9)    WAIT SID :PID  PROCNAME    USERNAME           LKTYPE    DATABASE:TABLENAME   LKOBJ     0 -   13900:12303 workprocess3   dbuser                  X        rome    :orders          row   1 W   53600:23613 batchp12       dbuser                           rome    :orders      Colno Purpose(0)   Sequence number    (1)   Waiting or not waiting, possible values are:        "-" - this session is the holder of the lock and is always listed first.        "W" - this session(s) is(are) waiting for the above session.    (2)   Session ID of this session in the database server    (3)   Process ID  of the UNIX process, remote connections have pid -1    (4)   Process name of the UNIX process. If it is a remote connection        (pid = -1), no process name will be available.    (5)   UNIX username of this session    (6)   Type of lock, possible values are:        "X" - Exclusive Lock        "S" - Shared Lock        "U" - Update Lock        For additional lock types, execute the following sql-statement:        -> select txt from sysmaster:flags\_text where tabname = "syslcktab"(7)   Database name    (8)   Table name, the lock is on. If it is an index lock and the index is detached        from the table (has it's own partition number), the name of that index        is shown here.    (9)   Type of object locked, possible values are:        "table" - this is a table lock        "idx" - this is an index key lock        "page" - this is a page lock        "row" - this is a row lock |

Listing 6. Lockwt - Lock wait situation I

|  |  |
| --- | --- |
|  | Output from lockwt:  -------------------  WAIT   SID  :PID   PROCNAME     USERNAME LKTYPE DATABASE:TABLENAME       LKOBJ    0 -    13900:12303 workprocess3 dbuser   X      rome    :orders          row  1 W    53600:23613 batchp12     dbuser          rome    :orders |

In this example session, **13900** (process "workprocess3") is holding a lock on a specific row in the table **orders**. Session **53600** is waiting for this lock to be released. You need to analyze session **13900** by executing **onstat -g ses 13900**.

Listing 7. Lockwt - Lock wait situation II

|  |  |
| --- | --- |
|  | Output from lockwt:  -------------------  WAIT   SID  :PID   PROCNAME     USERNAME LKTYPE DATABASE:TABLENAME       LKOBJ    0 W     3894:   -1 (remote)     eherber1 X      rome    :status          row  1 W    17048: 3140 batchp3      dbuser          rome    :status    0 -    63296:   -1 (remote)     eherber1 X      rome    :customer\_order  row  1 W     3894:   -1 (remote)     eherber1        rome    :customer\_order |

This example is a little bit more complex. Session **17048** is waiting for session **3894** to release the lock on table **status**. But take a look at the second pair of locks. Session **3894** is waiting for session **63296**. This is a typical escalating lock situation, because session 3894 is holding a lock another session is waiting for, but session 3894 is also waiting for a lock to be released. Analyze what session **63296** is doing using the **onstat -g ses 63296** command.

In the source code of the lockwt utility, there are some interesting sysmaster queries that you can take as a starter for writing specific queries on your own.

The open cursor problem

You may have already encountered a strange problem when trying to alter a table. Even though you have explicitly locked a table in exclusive mode, you are not able to alter it. The following example demonstrates this issue:

Listing 8. Non-exclusive access on a table

|  |  |
| --- | --- |
|  | Output from dbaccess -e stores\_demo <script.sql>:  --------------------------------------------------  begin;    Started transaction.    lock table customer in exclusive mode;    Table locked.    alter table customer add (mycol integer);    242: Could not open database table (informix.customer).    106: ISAM error: non-exclusive access. |

This behavior occurs because a select cursor was opened on table **customer** by someone else. The cursor is not placing any locks on individual rows; otherwise we would have not been able to lock the table exclusively, but it prevents IDS from changing the partition information.

To solve the problem, identify the session that opened a cursor on table **customer**:

1. Determine the hexadecimal partition number of table **customer**:
   * Select **hex(partnum)** from systables where tabname = **"customer"**.
2. If the partition number of this table is zero, it is a fragmented table. You need to execute the following SQL statement ito find the partition numbers of the individual fragments:

Select **st.tabname, dbinfo("dbspace", sf.partn), hex(sf.partn)** from systables st, **sysfragments** sf, where st.tabid = sf.tabid and sf.fragtype = "T" and st.tabname = **"customer"**.

1. Take the hexadecimal partition number and search for it in all currently opened tables:

**onstat -g opn | grep -i <hex\_partnum>**

1. Take the rstcb column, which is a shared memory address of the respective user thread, and search for it using the **onstat -u** command.

**onstat -u | grep <rstcb\_without\_leading\_0x>**

After you identify the respective database session, you can terminate it with the **onmode -z <sessid>** command.