Locking and Concurrency
Concept and Troubleshooting

Ani Patel
IBM Toronto Lab
anipatel@ca.ibm.com

WDUG
March 1, 2011
Platform: Linux, UNIX, Windows
Agenda

- Concurrency and isolation levels
  - Talk about different isolation levels
  - Transaction anomalies
  - Choosing right isolation level

- Locking mechanism
  - Lock granularity
  - Lock attributes
  - Symptoms of a locking problem

- Troubleshooting
  - Lock escalation
  - Lock wait and timeouts
  - Deadlocks
Agenda

- Concurrency and isolation levels
  - Talk about different isolation levels
  - Transaction anomalies
  - Choosing right isolation level

- Locking mechanism
  - Lock granularity
  - Lock attributes
  - Symptoms of a locking problem

- Troubleshooting
  - Lock escalation
  - Lock wait and timeouts
  - Deadlocks
What is concurrency?

- Concurrency is the ability for multiple users or application programs to interactively share resources at the same time.

- DB2 utilizes locking to control concurrency.

- Locking is required to maintain data integrity and to implement the supported isolation levels.

- The concept of concurrency is very important to a database. Users have to be able to share resources without affecting each other’s work, so proper concurrency control is vital.
Isolation levels provided by DB2

- The isolation level associated with an application process defines the degree of “separation” of that application process from other concurrently executing application processes.

- DB2 utilizes locking to control concurrency.
  - Uncommitted Read (UR)
  - Cursor Stability (CS)
  - Read Stability (RS)
  - Repeatable Read (RR)
# Transaction Anomalies

<table>
<thead>
<tr>
<th>Anomaly</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lost Updates</td>
<td>A transaction modifies data that was modified by another transaction which has not yet performed a COMMIT or ROLLBACK</td>
</tr>
<tr>
<td>Dirty Read</td>
<td>A transaction reads data that was modified by another transaction which has not yet performed a COMMIT or ROLLBACK</td>
</tr>
<tr>
<td>Non-Repeatable Read</td>
<td>A transaction that reads data does not see the same data which it had seen earlier. This phenomenon is also known as a fuzzy read.</td>
</tr>
<tr>
<td>Phantom Reads</td>
<td>A transaction that is reading data sees new data later in the same transaction. This occurs when another transaction inserts or updates data that would satisfy the transactions query</td>
</tr>
</tbody>
</table>
Choosing an Isolation Level

- Anomalies and Isolation Level

<table>
<thead>
<tr>
<th>Isolation Level</th>
<th>Dirty Reads</th>
<th>Non-Repeatable Reads</th>
<th>Phantom Reads</th>
</tr>
</thead>
<tbody>
<tr>
<td>RR</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>RS</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>CS</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>UR</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Agenda

- Concurrency and isolation levels
  - Talk about different isolation levels
  - Transaction anomalies
  - Choosing right isolation level

- Locking mechanism
  - Lock granularity
  - Lock attributes
  - Symptoms of a locking problem

- Troubleshooting
  - Lock escalation
  - Lock wait and timeouts
  - Deadlocks
Lock Granularity

- Tablespace locked only by some utilities and DDL, not always required. For example, Load and Reorg.

- For regular tables, table locking or table/row locking can be done. Generally, row level locking is used.

- The level of locking is determined by many factors such as isolation level, access plan (table or index scan), lock escalation etc.

  - Table scan in UR isolation will cause table level lock.
  - Lock escalation will force row level locks to be escalated to table level lock
  - Explicit LOCK TABLE statement or if a table's LOCKSIZE parameter is 'TABLE', it will cause table level locking
Lock Attributes

- We can determine the object numbers that a lock is requested/acquired on by looking at the lock name.

  - For example, a lock name of 0002000E00000008000000000052 maps to
    • table space x0002 =2
    • table object x000E=14
    • Lock type x52 =R
  - You can determine the name of the table from the syscat.tables catalog view
  - For a table lock, the first two fields will also show the tablespace id and table id, as here, but the final character will be x54 or ‘T’.

- The lock name is dependent on the type of lock. Each lock type has a slightly different lock name structure.
Symptoms of a Locking Problem

- A specific application seems to “hang” while other applications on the system continue to run.

- Applications appear to “hang” while regular DB2 commands return.

- Application “slow downs” when particular applications are run concurrently or when a DB2 utility is running.

- Locking problems can present as performance issues. Further diagnosis is required to determine whether locking is truly a factor in the slowdown.
Agenda

- Concurrency and isolation levels
  - Talk about different isolation levels
  - Transaction anomalies
  - Choosing right isolation level

- Locking mechanism
  - Lock granularity
  - Lock attributes
  - Symptoms of a locking problem

- Troubleshooting
  - Lock escalation
  - Lock wait and timeouts
  - Deadlocks
Lock Escalations

- Internal mechanism to reduce the number of locks held.

- Escalate from many row (and block) locks in a table to one table lock.

- A Lock is escalated when:

  - The total number of locks held by an application reaches the maximum amount of lock list space available to the application (maxlocks)

  - The lock list space consumed by all applications is approaching the total lock list space (locklist)

- Usually infrequent with a well-tuned database.
Lock Escalations: Symptoms

- A degradation in application performance is one of most common symptoms of a lock escalation. Lock escalations cause a reduction in concurrent access to the table.

- In some cases, high CPU usage during escalation.

- There may also be messages printed to the db2diag.log and/or the admin log, showing lock escalation has occurred, and a lock snapshot will show if a lock was requested as the result of an escalation.

FUNCTION: DB2 UDB, data management, sqldEscalateLocks, probe:3
MESSAGE: ADM5502W The escalation of "4089" locks on table "IFB .RIKOSTRK" to lock intent "X" was successful.
Lock Escalations: Diagnostics

- db2diag.log (Level 3)

- Event Table (Cobra) / Lock snapshot (pre-Cobra)

- db2pd –locks

- db2pd –wlocks (Cobra only) - displays the owner and waiter info for each lock being waited on

- Access plans
Reducing Lock Escalations

- Increase the number of locks allowed by increasing maxlocks or locklist for the database.

- Locate and adjust the offending processes (which may or may not be the one causing the escalation) and issue a LOCK TABLE explicitly.

- Change the degree of isolation if appropriate

- Increase the frequency of Commits.
Lock Waits and Timeouts

- There is the possibility that an application will have to wait indefinitely for a lock to be released.

- The locktimeout database configuration parameter can be used to set a maximum time an application can wait to get a lock.

- If the lock times out, the application will receive an error and the transaction will be rolled back. This helps avoid deadlocks.

- A lock timeout will result in an SQLCODE -911, same as a deadlock.

- These two conditions can be differentiated by their reason codes: a deadlock reports reason code 2, whereas a timeout uses reason code 68.
Lock Timeout Reporting – Before DB2 9.7

- New for v95, v91fp4, v8fp16

- Dynamic registry variable DB2_CAPTURE_LOCKTIMEOUT

- Captures information for
  - **Lock requestor**
    - Lock specifics, package information, isolation level, current operation, statement text if available
  - **Lock holder**
    - Application specifics, lock mode held, isolation level, list of active statements if available
    - List of inactive statements if there is an active detailed deadlock event monitor with history
Lock Reporting in DB2 9.7 Cobra

- Writes to a new target type – Unformatted Event Table:
  
  - CREATE EVENT MONITOR <name> FOR LOCKING WRITE TO UNFORMATTED EVENT TABLE

- Detailed information on lock timeouts, deadlocks, and lock waits for a specified duration (configurable).

- Deprecation of the default detailed deadlock event monitor DB2DETAILDEADLOCK and lock timeout report

- Default UE table name is the same as event monitor name unless specified. Query SYSCAT.EVENTTABLES for more information.

- Uses a new fast writer infrastructure
  - Increase in performance over the old infrastructure
  - Allows more detailed information to be collected faster
DB2 9.7 Cobra: Monitor Enablement

- Can be enabled at workload level (CREATE WORKLOAD) or at database level (MON_LOCKTIMEOUT, MON_DEADLOCK, MON_LOCKWAIT, MON_LW_THRESHOLD provide the default values when not specified)

- Example: Workload Level (CREATE):

  CREATE WORKLOAD employees
  APPLNAME( 'app1', 'app2')
  COLLECT LOCK TIMEOUT DATA WITH HISTORY

- Example: Workload Level (ALTER):

  ALTER WORKLOAD invoice
  COLLECT LOCK WAIT DATA FOR LOCKS
  WAITING MORE THAN 150000 MICROSECONDS
  WITH HISTORY AND VALUES

- Example: Database Level (CREATE):

  CREATE EVENT MONITOR EVMON FOR LOCKING
  WRITE TO UNFORMATTED EVENT TABLE (TABLE LOCK IN EVMON1_TS)
  SET EVENT MONITOR EVMON STATE 1
DB2 9.7 Cobra: Formatting Output Data

- Several ways to format the output data
  - db2evmonfmt (Java tool)
    ```
    javaw db2evmonfmt -d SAMPLE -ue LOCK -ftext -type deadlock -hours 1
    ```
  - EVMON_FORMAT_UE_TO_XML table function
    ```
    db2 "SELECT XMLPARSE(DOCUMENT evmon.xmlreport) FROM TABLE(EVMON_FORMAT_UE_TO_XML ( NULL, FOR EACH ROW OF ( select * from LOCK where EVENT_TYPE = 'DEADLOCK' and EVENT_TIMESTAMP >= CURRENT_TIMESTAMP - 2 hours order by EVENT_TIMESTAMP ))) as evmoN"
    ```
  - EVMON_FORMAT_UE_TO_TABLES stored procedure – see the next slide
    ```
    db2 "CALL EVMON_FORMAT_UE_TO_TABLES( 'LOCKING', NULL, NULL, NULL, NULL, NULL, NULL, -1, 'select * from lock1')"
    ```
DB2 9.7 Cobra: Relational Lock Data Tables

- EVMON_FORMAT_UE_TO_TABLES retrieves the data that was stored in an unformatted event table using the EVMON_FORMAT_UE_TO_XML table function, and then uses decomposition technology to move the XML document into a set of relational tables:

  - LOCK_EVENT - This table is the main control table, where each row of the table contains a unique locking event that has occurred in the database.

  - LOCK_PARTICIPANT - This table contains information regarding the participants involved in the lock event.

  - LOCK_PARTICIPANT_ACTIVITIES - This table contains information regarding the activities participants involved in the lock event.

  - LOCK_ACTIVITY_VALUES - This table contains information regarding each activity's input values.
Deadlocks

- **Example**
  - Process 1 locks table A in X mode
  - Process 2 locks table B in X mode
  - Process 1 attempts to lock table B in S mode
  - Process 2 attempts to lock table A in X mode
  - Both processes wait for the other to have their lock request granted, but neither can proceed or roll back – deadlock!

- **Deadlock Detector**
  - Asynchronous system background process to handle deadlocks
  - Periodically activated - period specified by dlchktime parameter

- If deadlock detected, detector chooses victim process to roll back
Deadlocks – Contributing Factors

- Lock escalations.

- Application is locking tables explicitly when system-generated row locks would have been sufficient.

- Application is using an inappropriate isolation level.

- Catalog tables are locked for RR.

- Applications are getting the same locks in different orders.

- Application with several processes that access the same table for reads then writes.
Deadlock Problem Determination

- Problem: Two users running an update statement on the same table. One user receives a SQL0911 RC=2.

- The RC=2 reveals that the error returned was due to a deadlock and not a lock timeout.

- Let’s walk through an example of a deadlock, and how to tackle it.
Deadlock Problem Determination

- The following tables are used in this example

<table>
<thead>
<tr>
<th>Table</th>
<th>Columns</th>
<th>Constraints</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parent_table</td>
<td>Parent_join_column</td>
<td>Primary key</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Parent_data_column</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Child_table</td>
<td>child_join_column</td>
<td>Foreign Key</td>
<td>Parent_join_column</td>
</tr>
<tr>
<td></td>
<td>child_data_column</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
#### Deadlock Problem Determination

- The following sample data is in the tables:

<table>
<thead>
<tr>
<th>Parent_join_column</th>
<th>Parent_data_column</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Data 1</td>
</tr>
<tr>
<td>2</td>
<td>Data 2</td>
</tr>
<tr>
<td>3</td>
<td>Data 3</td>
</tr>
<tr>
<td>4</td>
<td>Data 4</td>
</tr>
<tr>
<td>5</td>
<td>Data 5</td>
</tr>
<tr>
<td>6</td>
<td>Data 6</td>
</tr>
<tr>
<td>7</td>
<td>Data 7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Child_join_column</th>
<th>Child_data_column</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>One</td>
</tr>
<tr>
<td>2</td>
<td>Two</td>
</tr>
<tr>
<td>3</td>
<td>Three</td>
</tr>
<tr>
<td>4</td>
<td>Four</td>
</tr>
<tr>
<td>5</td>
<td>Five</td>
</tr>
</tbody>
</table>
Deadlock Problem Determination

- We reproduce the problem as follows:
  
  - Open a CLP Session (S1) at time (T1)
  - Issue the following SQL statement:
    
    ```
    db2 +c "update child_table set CHILD_DATA_COLUMN='n1' where CHILD_JOIN_COLUMN=5"
    ```

  - Open another CLP Session (S2) at time (T2)
  - Issue the following SQL statement:
    
    ```
    db2 +c "update parent_table set PARENT_JOIN_COLUMN=8 where PARENT_JOIN_COLUMN=6"
    ```

  - Back in (S1) at time (T3) and issue the following SQL statement:
    
    ```
    db2 +c "update parent_table set PARENT_JOIN_COLUMN=10 where PARENT_JOIN_COLUMN=6"
    ```
Deadlock Problem Determination

- At T1, update a row in child table

<table>
<thead>
<tr>
<th>Parent_join_column</th>
<th>Parent_data_column</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Data 1</td>
</tr>
<tr>
<td>2</td>
<td>Data 2</td>
</tr>
<tr>
<td>3</td>
<td>Data 3</td>
</tr>
<tr>
<td>4</td>
<td>Data 4</td>
</tr>
<tr>
<td>5</td>
<td>Data 5</td>
</tr>
<tr>
<td>6</td>
<td>Data 6</td>
</tr>
<tr>
<td>7</td>
<td>Data 7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Child_join_column</th>
<th>Child_data_column</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>One</td>
</tr>
<tr>
<td>2</td>
<td>Two</td>
</tr>
<tr>
<td>3</td>
<td>Three</td>
</tr>
<tr>
<td>4</td>
<td>Four</td>
</tr>
<tr>
<td>5</td>
<td>Five</td>
</tr>
</tbody>
</table>

- At T2, update a row in parent

- At T3, update a row in parent
Deadlock Problem Determination

- The deadlock that occurs dumps the following info in the db2diag.log:

```
2007-05-10-16.18.07.882000-240 I300214H929 LEVEL: Info
PID : 920 TID : 1968 PROC : db2syscs.exe
INSTANCE: DB2_01 NODE : 000 DB : SAMPLE
APPHDL : 0-160 APPID: *LOCAL.DB2_01.070510201712
AUTHID : DB2INST1
FUNCTION: DB2 UDB, lock manager, sqlplnfd, probe:80
DATA #1 : String, 170 bytes
Request for lock "REC: (3, 257) RID x0800806509000000" in mode ".NS" failed due to deadlock
Application caused the lock wait is "*LOCAL.DB2_01.070510201526"
Statement:
DATA #2 : Hexdump, 71 bytes
0x059AF5D0 : 7570 6461 7465 2070 6172 656e 745f 7461 update parent_ta
0x059AF5E0 : 626c 6520 7365 7420 5041 5245 4e54 5f4a ble set PARENT_J
0x059AF5F0 : 4f49 4e5f 434f 4c55 4d4e 3d38 2077 6865 OIN_COLUMN=8 whe-
0x059AF600 : 7265 2050 4152 454e 5f4a 454f 494e 5f43 re PARENT_JOIN_C
0x059AF610 : 554d 4e3d 36 OLMN=6
```
Deadlock Problem Determination

- To gather more information before the next occurrence we can use db2pdcfg –catch as follows:
  - Copy sqllib/cfg/db2cos to sqllib
  - Run db2pdcfg –catch -911,2
  - A file called db2cos<tid><pid>.<node> will be found in the sqllib/db2dump directory when the problem reoccurs.
# Deadlock Problem Determination

## Applications:

<table>
<thead>
<tr>
<th>AppHandl</th>
<th>NumAgents</th>
<th>CoorTid</th>
<th>Status</th>
<th>C-AnchID</th>
<th>L-AnchID</th>
</tr>
</thead>
<tbody>
<tr>
<td>160</td>
<td>1</td>
<td>1968</td>
<td>UOW-Executing</td>
<td>95</td>
<td>0</td>
</tr>
<tr>
<td>156</td>
<td>1</td>
<td>6036</td>
<td>Lock-wait</td>
<td>153</td>
<td>77</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>240</td>
<td>UOW-Waiting</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>2636</td>
<td>ConnectCompleted</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>4696</td>
<td>ConnectCompleted</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

## Transactions:

<table>
<thead>
<tr>
<th>Address</th>
<th>AppHandl</th>
<th>TranHdl</th>
<th>Locks</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x052A1880</td>
<td>156</td>
<td>2</td>
<td>7</td>
<td>WRITE</td>
</tr>
<tr>
<td>0x052A2480</td>
<td>8</td>
<td>3</td>
<td>0</td>
<td>READ</td>
</tr>
<tr>
<td>0x052A3080</td>
<td>9</td>
<td>4</td>
<td>0</td>
<td>READ</td>
</tr>
<tr>
<td>0x052A3C80</td>
<td>10</td>
<td>5</td>
<td>0</td>
<td>READ</td>
</tr>
<tr>
<td>0x052A4880</td>
<td>160</td>
<td>6</td>
<td>7</td>
<td>WRITE</td>
</tr>
</tbody>
</table>
Deadlock Problem Determination

Dynamic SQL Statements:
AnchID  Text
77      update child_table set CHILD_DATA_COLUMN='n1' where CHILD_JOIN_COLUMN=5
95      update parent_table set PARENT_JOIN_COLUMN=8 where PARENT_JOIN_COLUMN=6
153     update parent_table set PARENT_JOIN_COLUMN=10 where PARENT_JOIN_COLUMN=6

Dynamic SQL Environments:
Address  AnchID StmtUID EnvID Iso QOpt Blk
0x059AE8C0   77    2    1    CS    5    B
0x059AC9C0   95    2    1    CS    5    B
0x059AA320   153   2    1    CS    5    B
### Deadlock Problem Determination

#### Locks:

<table>
<thead>
<tr>
<th>TranHdl</th>
<th>Lockname</th>
<th>Type</th>
<th>Mode</th>
<th>Sts</th>
<th>Owner</th>
<th>Dur</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>0300000010900406509000000052</td>
<td>Row</td>
<td>..X</td>
<td>G</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0300000010900406509000000052</td>
<td>Row</td>
<td>.NS</td>
<td>W</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>030001010800806509000000052</td>
<td>Row</td>
<td>..X</td>
<td>G</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>0300010108008065090000000052</td>
<td>Row</td>
<td>.NS</td>
<td>W*</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>0300000100000000000000000054</td>
<td>Table</td>
<td>.IX</td>
<td>G</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>0300000010000000000000000054</td>
<td>Table</td>
<td>.IX</td>
<td>G</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0300010100000000000000000054</td>
<td>Table</td>
<td>.IX</td>
<td>G</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>0300010100000000000000000054</td>
<td>Table</td>
<td>.IS</td>
<td>G</td>
<td>6</td>
<td>1</td>
</tr>
</tbody>
</table>
Deadlock Problem Determination

- At this point the two applications are in a deadlock.

- The DB2 Deadlock detector detects the deadlock and picks a victim transaction to rollback. In this example its AppHandl 8 as can be seen in the db2diag.log.

- So why do we have a deadlock?
  - To answer this question we have to look at the DDLs of the tables.
  - Notice that we have a primary key constraint on the PARENT_TABLE and a foreign key constraint on the CHILD_TABLE.
  - There are no indexes on the CHILD_TABLE. This forces the optimizer to do a table scan and lock all the rows it access.
Deadlock Problem Determination

Access Plan:

```
| Total Cost: | 36.706 |
| Query Degree: | 1 |
```

Rows
RETURN
( 1)
Cost
I/O
| 3.84
UPDATE
( 2)
36.706
4.84

```
3.84
96
TBSCAN
(TABLE: AMANL
( 3)
CHILD_TABLE
7.65633
1
96
TABLE: AMANL
CHILD_TABLE
```

Original Statement:

```
update child_table set CHILD_DATA_COLUMN='n1'
where CHILD_JOIN_COLUMN=5
```

Predicates:

```
2) Sargable Predicate
Comparison Operator: Equal (=)
Subquery Input Required: No
Filter Factor: 0.04
```

Predicate Text:

```
(Q2.CHILD_JOIN_COLUMN = 5)
```
Deadlock Problem Determination

Access Plan:

-------------
Total Cost: 15.2946
Query Degree: 1

Original Statement:

-------------
update parent_table set PARENT_JOIN_COLUMN=6
where PARENT_JOIN_COLUMN=6

Predicates:

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

3) Sargable Predicate
Comparison Operator: Equal (=)
Subquery Input Required: No
Filter Factor: 0.04

Predicate Text:

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

(6 = Q3.CHILD_JOIN_COLUMN)

4) Sargable Predicate, Evaluate at Application Subquery
Comparison Operator: Not applicable
Subquery Input Required: Yes
Filter Factor: 0.5

Predicate Text:

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

NOT EXISTS(SELECT $RID$
FROM AMANL.PARENT_TABLE AS Q1
WHERE (6 = Q1.PARENT_JOIN_COLUMN))
Deadlock Problem Determination

- To avoid the table scan and ultimately the deadlock, we add the following index to the CHILD_TABLE:

```
ALTER TABLE CHILD_TABLE
    ADD CONSTRAINT PARENTPKEY PRIMARY KEY
    (CHILD_JOIN_COLUMN);
```
Summary and Conclusion

- Locking and concurrency is an important consideration in application and database design.

- The impact of locking and concurrency can be externally affected by:
  - Isolation level chosen
  - Application design and resulting processing and access plans
  - Several tuning parameters

- An understanding of DB2’s internal locking strategies helps to better design applications and interpret locking and concurrency issues.
Questions ?