Assignment 2 - DB2

Daniel Brand Andreas Kostecka

1. CONCURRENCY CONTROL MECHANISMS

Describe briefly the concurrency control mechanisms supported by your DBMS.

For concurrency control and prevention from uncontrolled data access the DB2 database manager offers locking for bufferpools, tables, data partitions, blocks and rows. A lock associates a resource of the database manager from an application. This application is called the lock owner. A lock controls the concurrent access of the same resource from other applications held by the lock owner.

The database manager uses locking at a row or table level but always attempts to acquire row-level locks. The decision of the locking level is based on several factors:

- Active isolation level
- Database access method (index scan or table scan)
- Specified **LOCKSIZE** attribute for a certain table (see section 3)
- Available space for locks depending on the **LOCKLIST** size and/or **MAXLOCKS** parameter
  - LOCKLIST: The amount of memory, defined in 4KiB pages, reserved to manage locks for all connected applications
  - MAXLOCKS: Maximum percentage of the entire lock list a single application can hold

The duration a lock is alive depends on the isolation level:

- Uncommitted-Read-Search: If row data isn't altered, row locking is deactivated
- Cursor-Stability-Search: Row locks are active as long as de cursor is positioned at the corresponding row. In this case it is possible that no locks are active
- Read-Stability-Search: Row locks for rows that fits the search criterion are valid as long as the transaction is active
- Repeatable-Read-Search: All row locks are valid as long as the transaction is active

2. LOCKING MODES

What lock modes are supported by your DBMS (shared, exclusive, others)?

DB2 supports the following lock modes:

1. Intent None (IN)
   - Applicable-Objects: Tablespaces, tables, blocks, data partitions
   - Description: Lock owner can read locked data in the object, also uncommitted data, but isn't allowed to alter this data. Lock owner doesn't acquire row-level locks. Other concurrent running applications can read and alter data.
2. Intent Share (IS)
   - Applicable-Objects: Tablespaces, tables, blocks, data partitions
   - Description: Lock owner can read locked data in the table, but isn't allowed to alter this data. Other applications can read and alter data in the table.
3. Intent Exclusive (IX)
   - Applicable-Objects: Tablespaces, tables, blocks, data partitions
   - Description: Lock owner and concurrent running applications can read and alter data. Other concurrent running applications can read and update the table.
4. Scan Share (NS)
   - Applicable-Objects: Rows
   - Description: Lock owner and concurrent running applications can read the locked rows, but aren't allowed to alter these rows. When the isolation level of the application is Cursor-Stability (CS) or Read-Stability (RS) this lock mode is used instead of a normal Share (S) lock.
5. Next Key Weak Exclusive (NW)
   - Applicable-Objects: Tablespaces, tables, blocks, data partitions
   - Description: If a row is inserted into an index the next row acquires a NW lock. This occurs only when the next row is locked by an Repeatable-Read-Search. The lock owner can read the row but not alter.
6. Share (S)
   - Applicable-Objects: Rows, tables, blocks, data partitions
   - Description: Lock owner and concurrent running applications can read the locked data but not alter.
7. Share with Intent Exclusive (SIX)

- Applicable-Objects: Tables, blocks, data partitions
- Description: Lock owner can read and alter data. Other concurrent running applications can only read the table.

8. Update (U)
    - Applicable-Objects: Rows, tables, blocks, data partitions
    - Description: Lock owner can alter data. Other transactions (units of work) can read the locked data in the object but aren't allowed to alter the data.
9. Exclusive (X)
    - Applicable-Objects: Rows, tables, blocks, data partitions, bufferpools
    - Description: Lock owner can read and alter data in the locked object. Only applications running with the isolation level Uncommitted-Read (UR) can access the locked object.
10. Super Exclusive (Z)
    - Applicable-Objects: Tablespaces, tables, blocks, data partitions
    - Description: Lock owner can alter, drop the table or create, drop an index. This lock is automatically granted whenever a transaction uses one of these operations. No other concurrent running application can read or alter this table.

	IN	IS	NS	S	IX	SIX	U	X	Z	NW
IN	Y	Y	Y	Y	Y	Y	Y	Y	X	Y
IS	Y	Y	Y	Y	Y	Y	Y	X	X	X
NS	Y	Y	Y	Y	X	X	Y	X	X	Y
S	Y	Y	Y	Y	X	X	Y	X	X	X
IX	Y	Y	X	X	Y	X	X	X	X	X
SIX	Y	Y	X	X	X	X	X	X	X	X
U	Y	Y	Y	Y	X	X	X	X	X	X
X	Y	X	X	X	X	X	X	X	X	X
Z	X	X	X	X	X	X	X	X	X	X
NW	Y	X	Y	X	X	X	X	X	X	X

Table 1: Lock mode compatibility matrix (Y - compatible, X - not compatible)

3. RESOURCE LOCKING

Which resource types can be locked in your DBMS (schema, tables, rows, others)?
Show on simple examples how (if possible) to explicitly lock resources (for example, tables and tuples)?

DB2 supports locking for the following types:

- Bufferpools
- Tables
- Data-Partitions (partitioned tables)
- Blocks
- Rows

With the command **ALTER TABLE** a global lock granularity can be set:

- Rows

  ```
  ALTER TABLE TABLE_NAME LOCKSIZE ROW
  ```

- Blockinserts

  ```
  ALTER TABLE TABLE_NAME LOCKSIZE BLOCKINSERT
  ```

- Tables

  ```
  ALTER TABLE TABLE_NAME LOCKSIZE TABLE
  ```

With the command **LOCK TABLE** an application can lock a table on application level:

- Share-Mode:

  ```
  LOCK TABLE TABLE_NAME IN SHARE MODE
  ```

  Prevents concurrent application processes from executing any except read-only operations on the table.

- Exclusive-Mode:

  ```
  LOCK TABLE TABLE_NAME IN EXCLUSIVE MODE
  ```

  Prevents concurrent application processes from executing any operations on the table. EXCLUSIVE MODE does not prevent concurrent application processes running at isolation level Uncommitted Read (UR) from executing read-only operations on the table.

A bufferpool lock is granted as soon as a bufferpool is created, altered or deleted. Bufferpools have always exclusive locks.

4. BOOKING EXPERIMENT

4.1 Remarks

For IBM's database system DB2 the *ROW LOCKING* can be set by the SQL - statement **ALTER TABLE LOCKSIZE ROW**. Since DB2 is based on the SQL92 standard, no boolean data type is supported (not for columns at least), so we simulated them by using an Integer with the values 1 and 0. The simulation utility was written in Java. The test was done on a 6-core processor with a local DB2 installation on a SSD.

4.2 Results

4.2.1 *Execution times and booking tries*

In figure 1 we can see the almost identical execution times, if it were not for the single transaction, serialized test. The execution times for this test were significantly larger than for any other test. The times for other tests vary within a few hundred milliseconds.

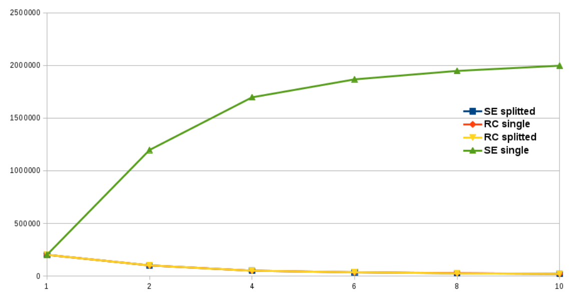

Figure 1: Line plot for all tests

To see the curve for the tests better, we omitted the single transaction, serialize test for figure 2;

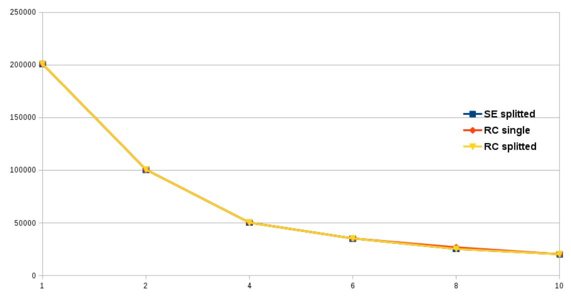

Figure 2: Line plot without single transaction, serialize test

The tables 2 3 show the average, minimal and maximal booking tries for the isolation levels read committed and serializable respectively.

Threads:	1	2	4	6	8	10
Read committed single						
min:	1	1	1	1	1	1
max:	1	1	1	1	1	1
avg:	1	1	1	1	1	1
Read committed splitted						
min:	1	1	1	1	1	1
max:	1	1	1	1	1	1
avg:	1	1	1	1	1	1

Table 2: Booking tries for isolation level **Read committed**

Threads:	1	2	4	6	8	10
Serializable single						
min:	1	1	1	1	1	1
max:	1	101	151	168	176	181
avg:	1	1.5	2.5	3.525	4.5	5.5
Serializable splitted						
min:	1	1	1	1	1	1
max:	1	1	1	1	1	1
avg:	1	1	1	1	1	1

Table 3: Booking tries for isolation level **Serializable**

The numbers of (re-)tries for booking a seat seem to correlate with the observed execution times. The only way for DB2 and our program to trigger a retry was setting the isolation level to serializable and using a single transaction. We are unsure whether our implementation has issues or this behavior simply follows through the nature of DB2 and the properties of this exercise.

4.2.2 Resulting tables

We decided to look at that aspect too. Since at the end of a test each row should have value *false* for column *availability*, we looked at the remaining *true* entries. Table 4 shows the row counts for tables with an availability field set to *true*.

Threads:	1	2	4	6	8	10
Read committed						
Single:	0	4	8	10	15	24
Splitted:	0	3	5	9	21	15
Serializable						
Single:	0	0	0	0	0	0
Splitted:	0	1	7	9	14	15

Table 4: Absolute failure rates

Looking at the results for a single thread, a zero mark was expected, since only one thread accesses the database at a time. With an increasing thread count, the failures, i.e. a seat doubly set to *not available*, rise too, with the worst value 24 for isolation level read committed on 10 threads. The only way to guarantee a correct booking for our implementation and using DB2 was using a single transaction with isolation level serializable.

4.3 Results with PreparedStatements

For the previous results we used the class *java.sql.Statement* throughout the program. Because the execution times were that bad, we tried replacing the *java.sql.PreparedStatement* class for all taxing commands, i.e. *SELECT*, *UPDATE* and *INSERT*. The overall times did indeed decrease by a small margin, but 1 second at most. The failure rates seem to be better though, for isolation level serializable at least.

Threads:	1	2	4	6	8	10
Read committed						
Single:	0	3	5	15	16	20
Splitted:	0	1	9	11	21	20
Serializable						
Single:	0	0	0	0	0	0
Splitted:	0	1	3	5	12	16

Table 5: Absolute failure rates

5. VIEWING LOCKS [OPTIONAL]

DB2 ships with a utility called **db2pd**, which is intended for error handling or correction. With that tool you can display a massive amount of information like memory usage, accessing applications and locks currently held.

To show the currently held locks:

```
db2pd -db DATABASE\_NAME -locks
```

An example output of that would be:

```
Database Member 0 -- Database FSR -- Active --
Up 0 days 00:26:33 -- Date 2015-11-08 ...

Locks:
Address            TranHdl    Lockname          ...
0x00007F45D791A600 11         0700000002000000
```

```
0x00007F45D7918880 12         0700000002000000
0x00007F45D791A100 12         0700000001000000
0x00007F45D7918B00 11         0700000001000000
0x00007F45D791B500 12         5359534C564C3031
0x00007F45D7911B00 11         5359534C564C3031
0x00007F45D7913280 12         0200040000000000
0x00007F45D7917E80 11         0200040000000000  ...
```

These are locks held during execution of our program in exercise 4 for this assignment. To show further information on those locks the option **showlocks** can be added. By using

```
db2 "SELECT SUBSTR(TABSCHEMA,1,8) AS TABSCHEMA,\
SUBSTR(TABNAME,1,15) AS TABNAME, LOCK_OBJECT_TYPE,\
LOCK_MODE, LOCK_MODE_REQUESTED, AGENT_ID_HOLDING_LK\
FROM SYSIBMADM.LOCKWAITS WHERE AGENT_ID = 000"
```

the currently held locks of a process with the AGENT_ID 000 are emitted.