NoSQL features of Db2 (LUW) revisited

Peter Vanroose

GSE Db2 BeLux User Group Meeting at IBM, Brussels, 18 October 2018
NoSQL features of Db2 revisited -- agenda:

• NoSQL, BigData, analytics
  - ACID versus BASE
  - “flat” data, versus XML / JSON
  - Db2 flexibility: BLOB, hash access, APPEND ON, MQTs, ...

• Parallelism and sharding
  - cluster-based model: data distribution & replication; shared-nothing
  - the CAP theorem
  - Db2: what about clone tables, columnar tables, HADR, pureScale, ... ?

• Weakening ACID in Db2
  - SET ISOLATION = UR; NOT ENFORCED; NOT LOGGED; circular logs; ...
  - restartable programs
  - pseudo-conversation
NoSQL - what’s in a name

Wikipedia:

• A NoSQL or Not Only SQL database provides a mechanism for
  - storage/retrieval of data, modelled otherwise than in RDBMS tables
  - motivations for this approach include: *simplicity* of design, *horizontal scaling*, higher *availability*, faster response

• Growing industry use in *big data* and *real-time web* applications

• Many NoSQL stores *compromise consistency* in favour of *availability* and *partition tolerance* ("CAP theorem")

• Most NoSQL stores lack true *ACID transactions*

Term “NoSQL” introduced 1998 by Carlo Strozzi (shell-interfaced RDBMS); term reintroduced 2009 in the context of *distributed DBs* (now meaning *not relational*)
NoSQL and Big Data

- **3 Vs (Gartner, 2001):** high-Volume, high-Velocity, high-Variety data
- (distributed) data *analysis* (data mining; statistical techniques)
- insight:
  - keep *all* data (sensor data, website clicks, blogs, ...)
  - in their *original* format (no ETL)
  - for potential later use (not yet decided at moment of collection)
    (pre-formatting may destroy or bias some information)
- as a consequence:
  - unstructured (or semi-structured, *non-flat*) data
  - less quality control/semantics during load => mainly useful for OLAP
  - interpretation & value judgement: done by ad-hoc *analysis* step(s)
Alledged problems/issues with “relational”

Some often heard arguments:

- 1. flat, tabular representation is unnatural
- 1b. need to convert to / from original (natural) representation
- 2. data modelling (DDL) beforehand => too rigid / restrictive / complex
- 2b. single column can only store similar data => too limiting
- 3. often need table joins => too heavy / complex / non-intuitive
- 4. may not scale well (horizontal scaling; large tables & growing)
- 5. too low concurrency (simultaneous users; parallelism)
- ...
Problem #1 - flat data

Statement: “flat, tabular representation is unnatural”

**Db2’s response:**

- store as XML (already since Db2 Version 9.1 -- that is: 2006 !)
  - Suppose table “companies” has column “empl” storing all employees for that company
  => one such “empl” should be of data type XML and could e.g. be:

  `<employees><person><name>Janssen</name><function>ANALYST</function></person>
  <person><name>Dupont</name><function>MANAGER</function></person></employees>`

- interrogate with XQuery or (even better) just with SQL:

  ```sql
  SELECT coname, XMLQUERY('count($E//function=[."ANALYST"]') PASSING empl AS e)
  FROM companies
  WHERE XMLEXISTS('$E/employees/person[function="ANALYST"]') PASSING empl AS e)
  ;
  SELECT c.coname, x.name AS employee_name, x.func AS employee_function
  FROM companies c, XMLTABLE('$E/employees/person' PASSING c.empl AS e)
  COLUMNS func VARCHAR(64) PATH 'function'
  , name VARCHAR(32) PATH 'name'
  ) x
  ;
  ```
Problem #1 - flat data (cont’d)

Db2’s response: (nr. 2)

- store as JSON (ECMA standard 2013; Db2 support since Version 10.5)
  - Suppose table “companies” has BLOB column “empl”, storing all employees for that company
    => one such “empl” could have the following value:

  
  ```json
  { employees: [ { name: "Janssen", function: "analyst" },
                  { name: "Dupont", function: "manager" } ]
  }  
  ```

- Interrogate with scalar function SYSTOOLS.JSON_VAL2, or with:
  
  ```sql
  SELECT c.coname, x.value AS function
  FROM companies c,
       TABLE(SYSTOOLS.JSON_TABLE(c.empl, 'employees.person.function', 's:64')) x
  ```

- or use the JSON-specific command line interface db2nosql.sh (!)
  
  Database has to be “enabled” (once) for using this interface with enable(true)

  ```bash
  db2nosql.bat -db MyDatabase
  nosql> db.companies.$find({})
  nosql> db.companies.$find({"employees.person.name":"Dupont"})
  ```
Problem #1b - convert to/from flat data

Db2 indeed does not require us to convert between XML & flat data!

but XML or JSON: probably still too rigid / too limited!

- How can we store anything whatsoever
- and yet easily
  - find it back and/or
  - aggregate on it (count/sum/avg/rank/top10/...)

“In search of a middle ground between file system & database”
=> one size does not fit all ... (Robert Greene, 2012)

Which brings us to Problem # 2 ...
Problem #2 - data modelling (DDL) beforehand

NoSQL wants:

- **schema-less** storage (=> dynamically add new attributes)
- but with **keys** & values (tuple store, ...) & possibly indexes

most NoSQL databases offer the possibility to work
- without a “schema”, i.e., without a predefined structure
- or with dynamically changing schema’s

**BUT** which **guarantees** can such a setup provide us?

*Db2’s response:*

- more flexible DDL changes
  - (more **ALTER support, esp. DROP COLUMN**)
- created global temporary tables
- common table expressions (CTEs), e.g. on top of CLOB/XML/JSON
Intermezzo: NoSQL database types

- **Key/Value Databases**
  - *Examples:* Berkeley DB, Oracle NoSQL, Dynamo, *MapReduce*

- **Document Stores**
  - *Examples:* MongoDB, CouchDB, MarkLogic, *IBM Lotus Notes (Domino)*

- **Columnar Databases**
  - *Examples:* Google Bigtable (2006), HBase, Cassandra, *Db2 BLU*

- **Graph (navigational) Data Model**
  - *Examples:* Neo4j, GraphDB, InfoGrid, *IMS*

- **Network DBMS**
  - *Examples:* IDMS
Intermezzo: NoSQL database types (cont’d)

Key/Value Database

- data stored based on programmer-defined keys [hash table approach]
- system is agnostic as to the semantics of the value
- requests are expressed in terms of keys: put(key, value), get(key): value
- indexes are defined over keys
Intermezzo: NoSQL database types (cont’d)

Key/Value Database - Db2’s related possibilities:

- **Hash access:**
  - Db2 table(space) which is not cluster-organized, but organized “by hash”
  - allows for fastest possible (single-page) access to a single row
  - hash “key” must be the primary key

- **The BYTE(n) and VARBYTE(n) datatypes**
  - similar to CHAR(n) and VARCHAR(n)
  - but no CCSID => no text interpretation, hence no auto-conversion

- **The BLOB datatype**

- **The Db2 transaction logs**
Intermezzo: NoSQL database types (cont’d)

**Document store**
- documents stored with programmer-defined key [“key-value”]
- system is aware of the arbitrary document structure
- support for lists, pointers and nested documents
- support for key-based & secondary indexes (with search possibility)
**Intermezzo: NoSQL database types (cont’d)**

Document store - Db2’s answer:

- **XML (again)**
- but not quite a “document store”
  - complicated way to assign an XML Schema to an XML document...
    - cf. SYSCAT.XSROBJECTS catalog view
    - need a stored procedure for registering new XML Schema
      CALL SYSPROC.XSR_REGISTER( 'AbisSchema', 'AbisCourseInfo',
    - after that, the following allows Schema validation upon insert:
      INSERT INTO MyTable(MyXmlCourseInfo)
      VALUES ( XMLVALIDATE( ?
      ACCORDING TO XMLSCHEMA URI 'http://abis.be/
      LOCATION 'http://abis.be/courses.xsd'
      ) )
  - impossible to more generally “link” XML documents within Db2
Intermezzo: NoSQL database types (cont’d)

### Columnar Database
- stores tables as sections of columns of data
- data stored together with meta-data (‘a map’)

**[typically including row id, attribute name & value, timestamp]**

<table>
<thead>
<tr>
<th>Name</th>
<th>Year</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peter</td>
<td>2013</td>
<td>75</td>
</tr>
<tr>
<td>Peter</td>
<td>2014</td>
<td>11</td>
</tr>
<tr>
<td>Peter</td>
<td>2014</td>
<td>3</td>
</tr>
<tr>
<td>Eliza</td>
<td>2014</td>
<td>70</td>
</tr>
<tr>
<td>Eliza</td>
<td>2014</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>2012</td>
<td>472</td>
</tr>
<tr>
<td></td>
<td>2011</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

Eliza: [4, 5]
Peter: [1, 2, 3]

2011: [7, ...]
2012: [6, ...]
2013: [1]
2014: [2, 4, 5]

1: [5]
3: [3]
11: [2, 7]
70: [4]
75: [1]
472: [6]

(name, Eliza, v1, 4)
(name, Eliza, v1, 5)
(name, Peter, v1, 1)
(name, Peter, v1, 2)
(name, Peter, v1, 3)
(date, 2011, v1, 7)
...
(date, 2012, v1, 6)
...
(date, 2013, v1, 1)
(date, 2014, v1, 2)
(date, 2014, v1, 4)
(date, 2014, v1, 5)
(amount, 1, v1, 5)
(amount, 3, v1, 3)
(amount, 4, v1, 2)
(amount, 11, v1, 7)
(amount, 11, v2, 2)
(amount, ...)

*NoSQL features of Db2 revisited*
1. NoSQL, BigData, analytics
   - XML / JSON
   - no DDL?
   - no joins?
   - NoSQL database types
2. Parallelism and sharding
   - cluster-based model:
     - distributed data & replication
     - shared-nothing
   - the CAP theorem
   - ACID versus BASE
3. Weakening ACID in Db2
   - syntactic possibilities
   - restartable programs
   - pseudo-conversation
Intermezzo: NoSQL database types (cont’d)

Columnar Database - Db2’s answer: “BLU acceleration” (since Db2 10.5):

- in-memory tables
- stored in a columnar fashion

**table dll**: ORGANIZE BY COLUMN **keyword**

=> better compression (similar data) & “sparse” (data skipping)

- is essentially an **indexes-only** table!

(one per column; sorted on timestamp)

Related Db2 technology:

**alter table dll:**

- **APPEND ON** **keyword**
- **COMPRESS YES** **keyword**
- **PREFETCHSIZE** **keyword** of the ALTER TABLESPACE statement
**Intermezzo: NoSQL database types (cont’d)**

**Graph (navigational) Data Model**
- data stored as *nodes & links*, both with (arbitrary) attributes
- requests through *system id’s* (or through indexes)
Graph (navigational) Data Model - Db2’s implementation:

- **This is exactly** the internal data representation of Db2!
  - RIDs
  - index is a hierarchy with internal & external pointers
  - z/OS: page sets (including space map pages)
  - fan sets (both for indexes and for foreign keys)
  - log records, log range info in the directory
- **Is even used exclusively** in the runtime environment
  - static SQL
  - packages & access paths
Problem #3 - table joins are heavy

Statement: “table joins: too often needed, too heavy, unnatural”

Db2’s response:

- normalisation (hence joins) avoids redundancy; one may denormalize
- use VIEWs to hide the “complexity” of joins
- use MQTs to additionally make join views “lighter” (performance)
  - but .. beware of refresh issues!  (consistency (ACID) jeopardised..)
- aggregate concatenation (since Db2 10.1):

  SELECT coname, LISTAGG(pname, ', ') 
  WITHIN GROUP (ORDER BY pname) AS employees 
  FROM companies JOIN persons ON cono = p_cono 
  GROUP BY p_cono ; 

  SELECT coname, (SELECT LISTAGG(pname, ', ') 
  FROM persons WHERE p_cono=c.cono) 
  FROM companies c
Problem #4 - scalability, parallelism, sharding

NoSQL wants:

- to use a **distributed** storage model (autonomous “nodes”; TCP/IP)
- with data **partitioning** ("sharding"), i.e.: *horizontally* splitting
- with **replication** for fault-tolerance (redundancy across nodes)

  ==> hence can afford “commodity hardware”

  ==> **scales linearly**: e.g. 10x more nodes for 10x more data or users

  => same response times promised ...

- **sharding & replication allow for parallelism:**
  - serve multiple clients in parallel (from different data copies),
  - and/or divide the work for 1 client over multiple workers
NoSQL features of Db2 revisited

1. NoSQL, BigData, analytics
   - XML / JSON
   - no DDL?
   - no joins?
   - NoSQL database types
2. Parallelism and sharding
   - cluster-based model: distributed data & replication
     shared-nothing
   - the CAP theorem
   - ACID versus BASE
3. Weakening ACID in Db2
   - syntactic possibilities
   - restartable programs
   - pseudo-conversation

Scalability, parallelism, sharding, replication

Data node = Worker

(Worker 1 may e.g. need data from Data node 2, though ...)
Sharding with Db2?

Db2’s implementation of “sharding”? 

- **Partitioning** => either PBG or PBR  
  - can imply (if wanted) that partitions are on different disks  
    => no shared disks; no replication though (except for backups+logs)  
  - but partitions cannot be in different buffer pools  
    (shared real memory)  
  - indexes can be partitioned or not  
    => note Db2 does not require any indexes!

- **HADR**  
  - High Availability Disaster Recovery => data replication  
  - primary vs standby database => will take over when needed ...  
  - *not sharding*: e.g. clients cannot connect to the standby server
Sharding with Db2 (cont’d)

- **Clone tables**? (atypical use case to implement 2-fold replication ...)
  
  ==> Always a shared something solution ...

- **pureScale**
  - since 2009, on AIX
  - several machines (members) together forming a *cluster*
  - no shared processor, no shared memory (buffer pools)
  - easily scales (more members => more parallel clients)
  - “easy” recovery from failing member
  - **but shared disks!** => so *not* sharding!
  - also a shared lock manager & a shared group buffer pool

- **Data sharing:**
  - available on Db2 for z/OS:
    - different LPARs, different Db2 instances
  - very similar to pureScale
Transactions, consistency and availability

- In a ‘shared something’ environment, ACID is wanted:
  - Pessimistic behaviour: force consistency at end of transaction!
  - Atomicity: all or nothing (of the n actions): commit or rollback
  - Consistency: transactions never observe or cause inconsistent data
  - Isolation: transactions are not aware of concurrent transactions
  - Durability: acknowledged transactions persist in all events (even in case of disaster)

- In a ‘shared nothing’ environment, BASE is implemented:
  - Optimistic behaviour: accept temporary database inconsistencies
  - Basically Available [guaranteed thanks to replication - no wait times]
  - Soft state [it’s user’s (application’s) task to guarantee consistency]
  - Eventually consistent (weakly consistent) ['stale’ data is OK]
Distributed data & processing

Why not have the best of both worlds?

=> **Consistency (ACID):** all clients see same data at same moment

=> **Availability (through N-fold replication):** no server timeouts

=> **speed (through sharding):** => **Partition tolerance**

**CAP theorem:**

- Brewer’s Conjecture (2000; proved in 2002; refined in 2012):

  *in any environment (shared-nothing or not) it is only possible to satisfy at most two of these requirements*

- \( C + A \Rightarrow \text{ACID} \);
- \( A + P \Rightarrow \text{BASE} \);
- \( C + P \Rightarrow \text{write } N \text{ read } 1 / \text{write } 1 \text{ read } N \)
NoSQL features of Db2 revisited

1. NoSQL, BigData, analytics
   - XML / JSON
   - no DDL?
   - no joins?
   - NoSQL database types

2. Parallelism and sharding
   - cluster-based model: distributed data & replication
     shared-nothing
   - the CAP theorem
   - ACID versus BASE

3. Weakening ACID in Db2
   - syntactic possibilities
   - restartable programs
   - pseudo-conversation

CAP theorem

![CAP diagram]

RDBMS
(Oracle, Db2, MySQL, SQLServer, PostgreSQL, ...)

*) KEY VALUE
(Berkley DB, Redis)

*) COLUMN/TABULAR
(BigTable, Hypertable, Hbase)

*) DOCUMENT
(MongoDB, Terrastore)

*) KEY VALUE
(Dynamo, Voldemort, KAI)

*) COLUMN/TABULAR
(Cassandra)

*) DOCUMENT
(SimpleDB, CouchDB, Riak)
Weakening ACID in Db2

- **Atomicity**: transaction (consisting of the \( n \) actions): all or nothing
  - long-running transactions => might be problematic!
    - logs span multiple log data sets => active log ( & log buffers) too large
    - locks of long duration -- either SHARED or EXCLUSIVE
  - 2 “old” solutions:
    - regularly commit (say every 5 seconds)=>breaks atomicity: a bit BASE!
    - use ISOLATION=UR for long running reads => see also Consistency ...
      or use **WITH UR keyword with SELECT**
  - and a “newer” one:
    - optimistic locking, lock avoidance, ...
    - idea: don’t place exclusive locks, but verify “last modified” time on read
      => data page timestamp, row change timestamp column, ...
Weakening ACID in Db2 (cont’d)

- **Consistency**: transactions *never* observe or cause inconsistent data
  - *READ* locks should last at least until effective read
    => SET ISOLATION = CS (or WITH CS)
  - what about e.g. *phantom reads*?
    => ACID would require ISOLATION=RR!!
  - *WRITE* inconsistency:
    - use NOT ENFORCED foreign key constraints (or no FKs at all ...)
    - not using cursor FOR UPDATE yet update (without CURRENT OF): evil!
    - after LOAD:
      Integrity Pending state
      => SET INTEGRITY FOR table IMMEDIATE UNCHECKED
      (might make sense for e.g. a test environment)

- **Isolation**: transactions are not aware of concurrent other transactions
  - weakened through (again) ISOLATION=UR, or regular commits
  - NoSQL would use *replication* though ... => mimic with MQTs?
Weakening ACID in Db2 (cont’d)

- **Durability**: acknowledged transactions persist in all events
  - also in case of a disaster (e.g. disk crash)
  - Db2 guarantees this through **Backups & transaction/archive logs**
  - “circumventing” the Db2 default behaviour:
    - ALTER TABLESPACE ... NOT LOGGED (only for LOB data)
    - LOAD ... COPY NO
      => BACKUP PENDING state => *Db2 does not allow data changes*
    - LOAD ... NONRECOVERABLE
    - not making backups
    - set database to *circular logging* only (logarchmeth1&2 set to OFF)
“NoSQL” application scenario’s with Db2

Some typically considered “application design” scenario’s which contain aspects which are not 100% “ACID”:

• Long running applications (typically: batch jobs)
  - need to “commit regularly”
  - should also apply to read-only applications! (often forgotten ...)
• Risk of inconsistent data, when application ends abnormally!
  - incomplete updates/inserts
  - duplicate updates/inserts on restart of job! => even worse ...
• Solution: make application restartable => programming skill!

NoSQL features of Db2
revisited

1. NoSQL, BigData, analytics
   - XML / JSON
   - no DDL?
   - no joins?
   - NoSQL database types
2. Parallelism and sharding
   - cluster-based model: distributed data & replication
     shared-nothing
   - the CAP theorem
   - ACID versus BASE
3. Weakening ACID in Db2
   - syntactic possibilities
   - restartable programs
   - pseudo-conversation
“NoSQL” application scenario’s with Db2 (cont’d)

- Long running interactive applications
  - graphical front-end, e.g. “paging” application: one screen at a time
  - cursor locks must be kept ... => unacceptable
  - solution: pseudo-conversation
    - application retrieves data for just 1 screen from Db2
    - application closes connection with Db2 after each screen
    - application reconnects to Db2 on “page down” or “page up” request
  - This requires ORDER BY and additional WHERE key > :last_seen
    - note the (Db2 10) handy “paging” syntax for when key is multi-column!
      WHERE (key1, key2) > (:last_seen_1, :last_seen_2)
      “syntactic sugar” for:
      WHERE key1 > :last_seen_1
      OR key1 = :last_seen_1 AND key2 > :last_seen_2
Restartability

- Not a new issue:
  - has been used for “batch” application development since “ages”
  - non-restartable programs are often rewritten to become restartable
- but typical for a “NoSQL” approach: because it’s a client decision
- What is restartability?
  - When a batch application returns normally => RC=0, no problem
  - When a batch application returns abnormally (crashes, or RC > 0):
    - Could e.g. be a “disk full” problem, or an “unavailable file” issue
    - Can the operator safely restart the program, after fixing the cause?
    - In general, no: risk of e.g. partial duplicate updates in Db2 ...
    - Unless either no intermediate commits, or program is restartable!
Restartability - Example

SELECT STATUS INTO :ExecutionStatus FROM SYNCTable;
if (ExecutionStatus == NormalEnd) { NormalStart(); } else { PrepareProgramRestart(); }

NormalStart():
    ProdNo <- 0; OrdNo <- 0; Totals <- 0;
    UPDATE SYNCTable SET STATUS = :Running;

PrepareProgramRestart():
    SELECT PRNO,ORDNO,TOTALS INTO :ProdNo, :OrdNo, :Totals
    FROM SYNCTable;

DECLARE prod CURSOR WITH HOLD FOR
    SELECT ... FROM ... WHERE ... AND (PRDNO,ORDNO) > (:ProdNo, :OrdNo)
    ORDER BY PRODNO, ORDNO;

• Note: restart info is saved in Db2 “syncpoint” table !!
Restartability - Example (cont’d)

NormalProgramEnd():
   UPDATE SYNCTable SET PRNO=0, ORDNO=0, STATUS=:NormalEnd ;
   COMMIT ;

• If the batch program modifies data,
   COMMIT processing (e.g. every 5 seconds) might already be in place;
   modify it as follows:

SyncpointProcessing() :
   UPDATE SYNCTable SET PRNO=:ProdNo, ORDNO=:OrdNo, Totals = :Totals ;
   COMMIT ; -- of both the data modifications and the synpoint info
Pseudo-conversational programs

- Not a new issue -- *but* typical for a “NoSQL” approach: client decision
- Typical situation:
  - User wants to scroll through a Db2 result set
  - The program shows only (say) 10 results (one screenful) at a time
  - Programs might allow for updates/inserts or might be read-only
  - Scroll-forward “next screen” & scroll-backward “previous screen”
- Pseudo-conversational approach:
  - Program reads just 10 rows from cursor, then *disconnects from Db2*
  - On “next screen”, it reconnects, runs cursor *with additional WHERE*
  - Program needs to remember “last entry seen”
Pseudo-conversational programs (cont’d)

• Example:

-- “data-dependent pagination”:
DECLARE nextscreen CURSOR FOR
    SELECT ... FROM ... WHERE ... AND (PRODNO,ORDNO) > (:ProdNo, :OrdNo)
    ORDER BY PRODNO, ORDNO
    FETCH FIRST 10 ROWS ONLY ;

OPEN nextscreen ;
FETCH nextscreen INTO :ProdNo, :OrdNo, ... ;
while (SQLCODE == 0) :
    Display_data() ;
    FETCH nextscreen INTO :ProdNo, :OrdNo, ... ;
CLOSE nextscreen ;
-- at this point, ProdNo and OrdNo are ready for the next “OPEN CURSOR”
Pseudo-conversational programs (cont’d)

- Scrolling backwards:
  DECLARE prevscreen CURSOR FOR
  SELECT ... FROM ... WHERE ... AND (PRODNO,ORDNO) < (:FirstProdNo, :FirstOrdNo)
  ORDER BY  PRODNO DESC, ORDNO DESC
  FETCH FIRST 10 ROWS ONLY ;

  OPEN prevscreen ;
  FETCH prevscreen INTO :LastProdNo, :LastOrdNo, ... ;
  FirstProdNo <- LastProdNo; FirstOrdNo <- LastOrdNo;
  while (SQLCODE == 0) :
    Display_data_backward() ;
    FETCH prevscreen INTO :FirstProdNo, :FirstOrdNo, ... ;
  CLOSE prevscreen ;

  (will also need FirstProdNo&FirstOrdNo on forward cursor traversal)
In summary ...

- **NoSQL, BigData, analytics**
  - Db2 supports non-flat data: XML (and JSON)
  - more Db2 flexibility: BLOB, hash access, APPEND ON, MQTs, ...

- **Parallelism and sharding**
  - pureScale cluster: comes close to a NoSQL setup
  - CAP theorem: cannot be 100% ACID and 100% sharded ...
  - Db2 features for “mimicking” some NoSQL functionality: clone tables, no indexes, columnar tables, HADR replication

- **Weakening ACID in Db2**
  - SET ISOLATION = UR; NOT ENFORCED; LOG NO
  - how to make Db2 batch programs restartable
  - how to make interactive programs pseudo-conversational
NoSQL features of Db2 (LUW) revisited

Thank you!

Peter Vanroose
ABIS Training & Consulting
pvanroose@abis.be