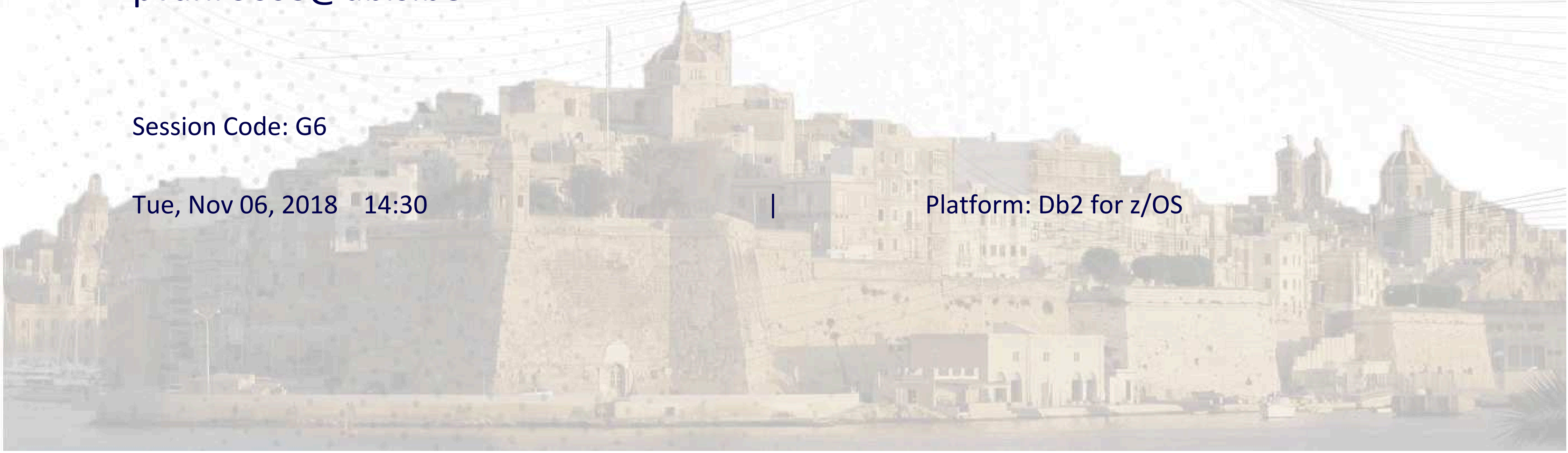
IDUG
Leading the DB2 User Community since 1988

IDUG EMEA Db2 Tech Conference
St. Julians, Malta | November 4 - 8, 2018

#IDUGDb2

From ACID to BASE: NoSQL with Db2

Peter Vanroose
ABIS Training & Consulting
pvanroose@abis.be



Session Code: G6

Tue, Nov 06, 2018 14:30 | Platform: Db2 for z/OS

Agenda

- NoSQL, BigData, analytics
  - ACID versus BASE
  - “flat” data, versus XML / JSON
  - Db2 flexibility: BLOB, hash access, APPEND YES, MQTs, ...
- Parallelism and sharding
  - cluster-based model: data distribution & replication; shared-nothing
  - the CAP theorem
  - Db2: what about data sharing, clone tables, IDAA, ... ?
- Weakening ACID in Db2
  - ISOLATION(UR); NOT ENFORCED; LOG NO; ...
  - 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-**V**olume, high-**V**elocity, high-**V**ariety 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 or 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 9 -- that is: 2007 !)
  - interrogate with XQuery or (even better) just **with 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.func AS employee_function
FROM     companies c, XMLTABLE('$E/employees/person' PASSING c.empl AS E)
                            COLUMNS func VARCHAR(64) PATH 'function' ) x
```

- store as **JSON** (some support since Db2 11)

```
SELECT value FROM SYSTOOLS.JSON_TABLE(c.empl, 'employees.person.function', 's:64') x
```

Problem #1b - convert to/from flat data

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

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 predefined structure
- or with dynamically changing schema's

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

Db2's response:

- more flexible DDL changes (e.g. `DROP COLUMN`); created GTTs; CTEs

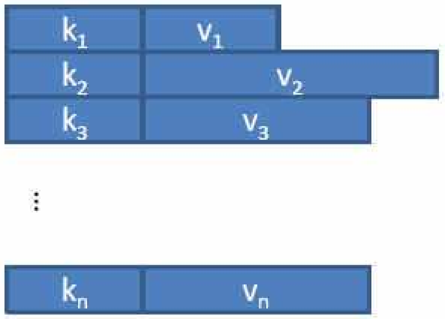
Intermezzo: NoSQL database types

- 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)

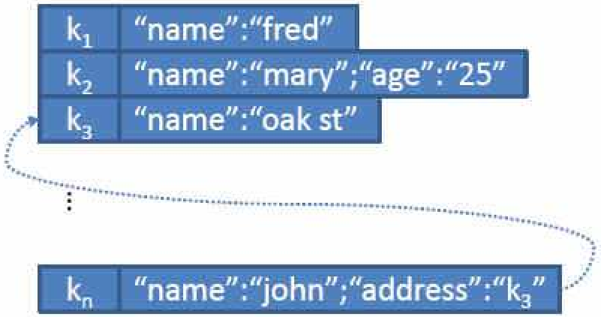
Key/Value Database - Db2's related possibilities:

- **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)



k_1	“name”:“fred”
k_2	“name”:“mary”;“age”:“25”
k_3	“name”:“oak st”
⋮	
k_n	“name”:“john”;“address”:“k_3”

Intermezzo: NoSQL database types (cont'd)

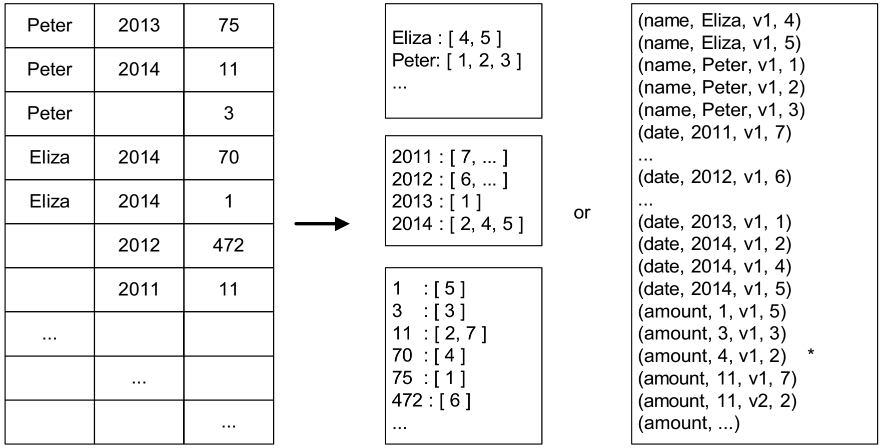
Document store - Db2's answer:

- XML (again)
- but not quite a "document store"
  - note the complicated way to assign an XML Schema to an XML document ...
    cf. SYSIBM.XSROBJECTS catalog table
  - 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]



Peter	2013	75
Peter	2014	11
Peter		3
Eliza	2014	70
Eliza	2014	1
	2012	472
	2011	11
...		
	...	
		...

→

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]
...

or

(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, ...)

Intermezzo: NoSQL database types (cont'd)

Columnar Database - Db2's answer:

- Db2 for LUW has so-called "BLU acceleration":
  - in-memory tables
  - stored in a columnar fashion
    => better compression (similar data) & "sparse" (data skipping)
- no counterpart (yet) in Db2 for z/OS
- is essentially an **indexes-only** table! (one per column; sorted on ts)

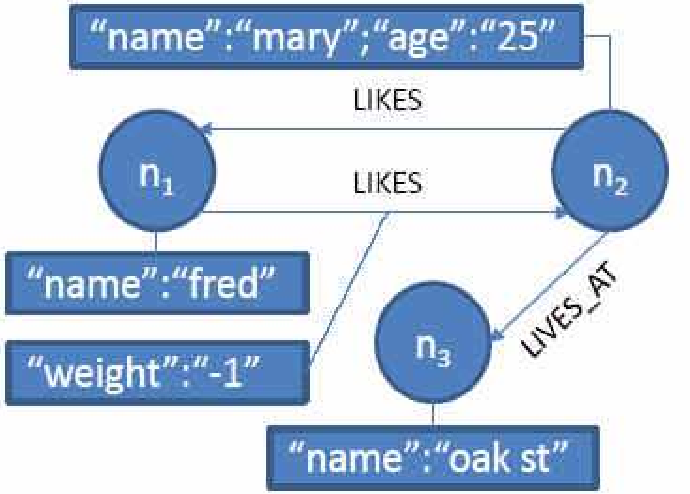
Related Db2 technology:

- in-memory buffer pools (`PGSTEAL(NONE)`) since Db2 12
- table ddl: `APPEND YES` keyword; or `MEMBER CLUSTER` on tablespace

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)



Intermezzo: NoSQL database types (cont'd)

Graph (navigational) Data Model - Db2's implementation:

- This is *exactly* the internal data representation of Db2 !
  - index: hierarchic structure, with internal & external pointers (RIDs)
  - page sets (including space map pages)
  - fan sets (both for indexes and for foreign keys)
  - log records, RBAs/LRSNs, 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** (Db2 12 FL 501):

```
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

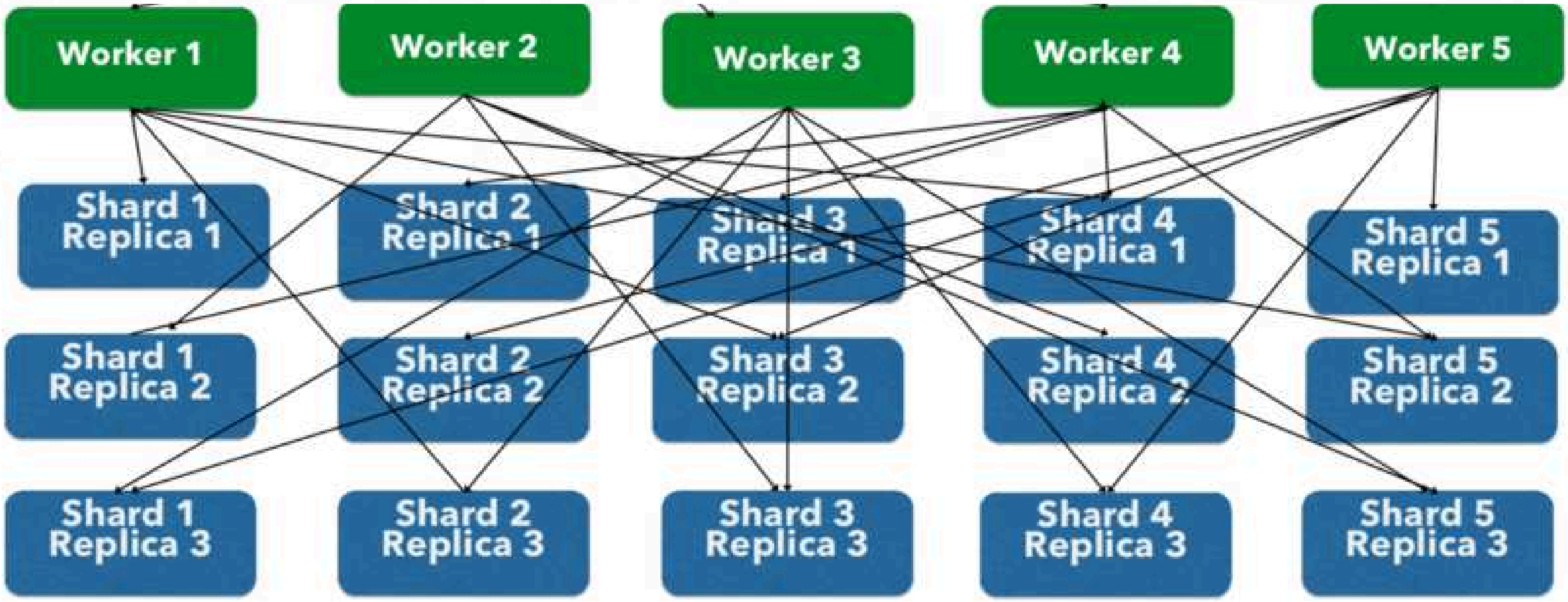
NoSQL wants:

- 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

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 volumes
    => no shared **disks**; no *replication* though (except for backups + logs)
  - but partitions *cannot* be in different buffer pools (shared **real memory**)
  - also need single Db2 subsystem (shared **LPAR**)
  - indexes: **DPSI** or not => note Db2 *does not require any indexes*!
- Data sharing: (=> note that *data sharing* is **not** sharding !)
  - no shared processor, no shared real memory (buffer pools)
  - **but** shared disks! => lock coordination (CF); use `MEMBER CLUSTER` ?

Sharding with Db2 ? (cont'd)

- **Clone** tables ? (atypical use case to implement 2-fold replication ...)
  ==> Always a **shared something** solution ...
- **IDAA**
  - a.k.a. Netezza / Sailfish
  - a "black box" appliance, accessible (only) by the Db2 optimizer
  - implements:
    - replication of (some) Db2 data
    - internal replication & sharding (multi-processor)
    - analytic processing (distributed) on this data
  - is a real **NoSQL** implementation!

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 *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 the 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?

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

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

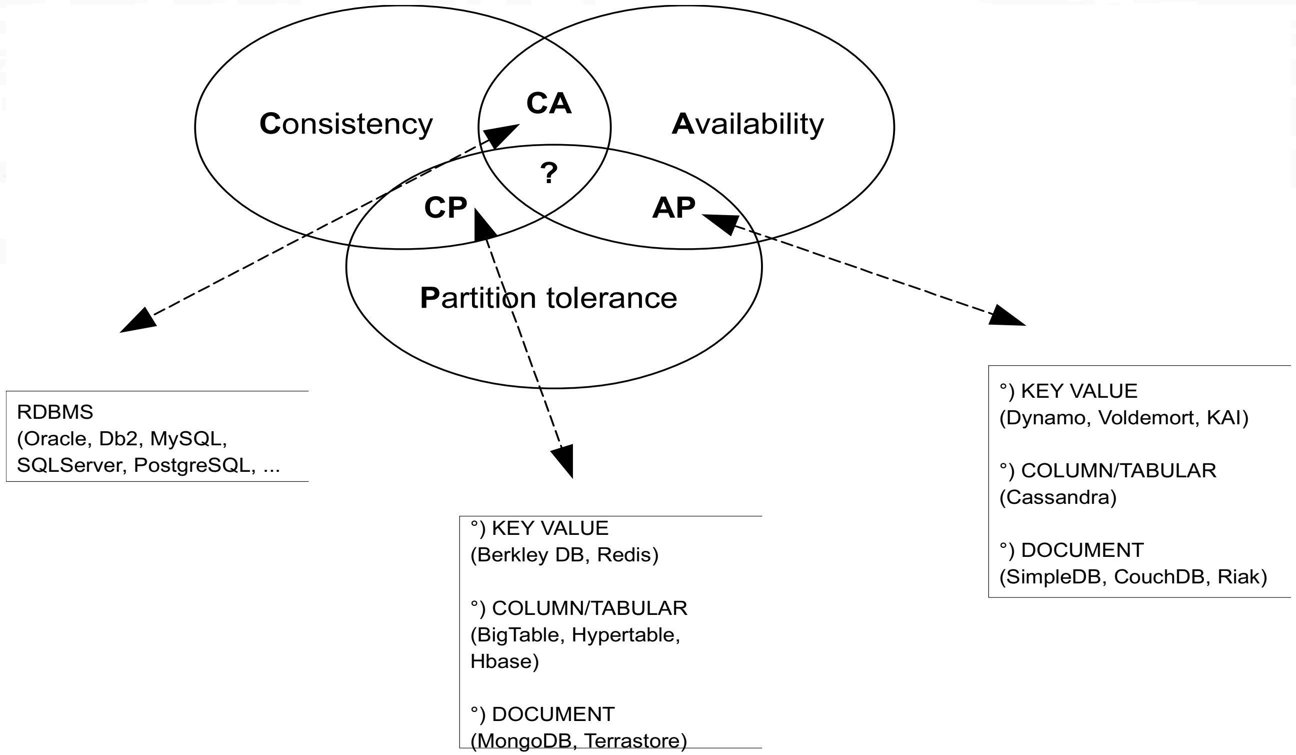
=> speed (through sharding) => **Partition tolerance**

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 => *ACID*; A + P => *BASE*; C + P => write N read 1 / write 1 read N

CAP theorem



Consistency

CA

Availability

?

CP

AP

Partition tolerance

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 ...
  - and a “newer” one:
    - **optimistic locking**, lock avoidance, latches, ...
    - 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 => ISOLATION(CS)
  - what about e.g. **phantom reads**? => ACID would require ISOLATION(RR) !!
  - *WRITE* inconsistency:
    - using NOT ENFORCED foreign key constraints (or no FKs at all ...)
    - not using cursor FOR UPDATE, yet update (without CURRENT OF): *evil!*
    - load ENFORCE NO, then `-START DB(xx) SP(yy) ACCESS(FORCE)` (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 Image **Copies** & transaction **logs**
  - “circumventing” the Db2 default behaviour:
    - ALTER TABLESPACE ... NOT LOGGED
    - LOAD ... LOG NO
    - not making image copies (or deleting them)
  - => COPY PENDING state => *Db2 does **not** allow data changes*
    - `-START DATABASE(...) SPACENAM(...) ACCESS(FORCE)`

“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 **abend**s !
  - incomplete updates/inserts
  - **duplicate** updates/inserts on restart of job! => even worse ...
- Solution: make application **restartable** => programming skill!

“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 > :LastSeen`
    - Db2 12 has new handy “paging” syntax for when key is multi-column!

Restartability

- Not a new issue:
  - has been used for mainframe 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* (abend, or RC > 4):
    - Could e.g. be a “disk full” problem, or an “unavailable dataset” 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

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

NormalStart():
    ProdNo <- 0; OrdNo <- 0; Totals <- 0; EXEC SQL UPDATE SYNCTable SET STATUS = :Running ;
PrepareProgramRestart():
    EXEC SQL SELECT PRNO,ORDNO,TOTALS  INTO :ProdNo, :OrdNo, :Totals FROM SYNCTable ;

EXEC SQL   DECLARE prod CURSOR WITH HOLD FOR
    SELECT ... FROM ... WHERE ... AND  (PRODNO,ORDNO) > (:ProdNo, :OrdNo) -- Db2 12
    ORDER BY      PRODNO, ORDNO ;
```

- Note: restart info is saved in Db2 “syncpoint” table !!

Restartability - Example (cont'd)

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

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

```
SyncpointProcessing():
    EXEC SQL    UPDATE SYNCTable SET PRNO=:ProdNo, ORDNO=:OrdNo, Totals = :Totals ;
    EXEC SQL    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 cond*
  - Program needs to remember “last entry seen”

Pseudo-conversational programs (cont'd)

- Example:

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

EXEC SQL    OPEN nextscreen ;
EXEC SQL    FETCH nextscreen INTO :ProdNo, :OrdNo, ... ;
while (SQLCODE == 0) :
    Display_data() ;
    EXEC SQL    FETCH nextscreen INTO :ProdNo, :OrdNo, ... ;
EXEC SQL    CLOSE nextscreen ;
-- at this point, ProdNo and OrdNo are ready for the next "OPEN CURSOR"
```

Pseudo-conversational programs (cont'd)

- Scrolling backwards:

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

EXEC SQL    OPEN prevscreen ;
EXEC SQL    FETCH prevscreen INTO :LastProdNo, :LastOrdNo, ... ;
FirstProdNo <- LastProdNo; FirstOrdNo <- LastOrdNo;
while (SQLCODE == 0) :
    Display_data_backward() ;
    EXEC SQL    FETCH prevscreen INTO :FirstProdNo, :FirstOrdNo, ... ;
EXEC SQL    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 YES, MQTs, ...
- Parallelism and sharding
  - only **IDAA** implements a really “shared-nothing” NoSQL setup
  - **CAP** theorem: cannot be 100% ACID and 100% sharded ...
  - Db2 features for “mimicing” NoSQL: data sharing, clone tables, no indexes
- Weakening ACID in Db2
  - ISOLATION(UR); NOT ENFORCED; LOG NO; -START ACCESS(FORCE); ...
  - how to make Db2 batch programs **restartable**
  - how to make interactive programs **pseudo-conversational**

From ACID to BASE: NoSQL with Db2

Peter Vanroose
ABIS Training & Consulting
pvanroose@abis.be

Session code: **G6**

Please fill out your session evaluation before leaving!