Université IBM i

10 et 11 mai 2016 – IBM Client Center de Bois-Colombes

S43 - VLDB and DB2 for i

Mercredi 11 mai – 15h15-16h45

Scott Forstie – IBM US
forstie@us.ibm.com
@Forstie_IBMi
DB2 for i

- Limits to Growth
- Very Large Database

IBM i 7.3
- ATTACH and DETACH PARTITION
- More objects within a library
- Raised maximum number of parameters and return columns on procedures and functions
- Priority change for parallel index builds
- More System Limits improvements

IBM i 7.2
- Increased Timestamp precision within SQL tables
- Multiple-action Triggers
- System Limits for IFS
- SQL support for Memory preference
- Automatic repositioning of updated rows within a partitioned table

IBM i 7.1
- XML Support
- EVI INCLUDE support for grouping set
- 1.7 Terabyte Indexes
- Tersaspace Storage Model
- Index Advice for OR predicates
- System Limits for DB2 tables
- SQL Plan Cache controls

IBM i 6.1
- DECIMAL
- Grouping sets / super groups
- Expression in Indexes
- Deferred Restore of MQT and Logicals
- Environmental limits
- Program Type SUB
- DB2 Statistical catalogs

IBM i V5R3
- Partitioned tables
- Online Reorganize
- 2M SQL statement
- 1000 tables in a query

V5R4
- SSD Memory Preference
- On Demand Performance Center
- Health Center
- Deferred Restore of MQT and Logicals
- Environmental limits
- Program Type SUB
- DB2 Statistical catalogs
- XML Support
- EVI INCLUDE support for grouping set
- 1.7 Terabyte Indexes
- Tersaspace Storage Model
- Index Advice for OR predicates
- System Limits for DB2 tables
- SQL Plan Cache controls

## Architectural Limits

<table>
<thead>
<tr>
<th>Database Constructs</th>
<th>FIPS</th>
<th>Oracle 12c</th>
<th>Microsoft SQL Server 2012</th>
<th>DB2 for i 7.2</th>
<th>DB2 for LUW V10.5</th>
<th>DB2 for z/OS V11</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length of an identifier (in bytes)</td>
<td>18</td>
<td>30</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>128</td>
</tr>
<tr>
<td>Length of CHARACTER data type (in bytes)</td>
<td>240</td>
<td>2000</td>
<td>8000</td>
<td>32766</td>
<td>254</td>
<td>255</td>
</tr>
<tr>
<td>Decimal precision of NUMERIC data type</td>
<td>15</td>
<td>38</td>
<td>38</td>
<td>63</td>
<td>31</td>
<td>31</td>
</tr>
<tr>
<td>Decimal precision of DECIMAL data type</td>
<td>15</td>
<td>38</td>
<td>38</td>
<td>63</td>
<td>31</td>
<td>31</td>
</tr>
<tr>
<td>Decimal precision of INTEGER data type</td>
<td>9</td>
<td>38</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>Decimal precision of SMALLINT data type</td>
<td>4</td>
<td>38</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Binary precision of FLOAT data type</td>
<td>20</td>
<td>126</td>
<td>53</td>
<td>53</td>
<td>53</td>
<td>53</td>
</tr>
<tr>
<td>Binary precision of REAL data type</td>
<td>20</td>
<td>63</td>
<td>24</td>
<td>24</td>
<td>24</td>
<td>24 or 24</td>
</tr>
<tr>
<td>Binary precision of DOUBLE PRECISION data type</td>
<td>30</td>
<td>126</td>
<td>53</td>
<td>53</td>
<td>53</td>
<td>53</td>
</tr>
<tr>
<td>Columns in a table</td>
<td>100</td>
<td>1000</td>
<td>1024</td>
<td>8000</td>
<td>1012</td>
<td>750</td>
</tr>
<tr>
<td>Values in an INSERT statement</td>
<td>100</td>
<td>1000</td>
<td>1024</td>
<td>8000</td>
<td>1012</td>
<td>750</td>
</tr>
<tr>
<td>Set clauses in an UPDATE statement</td>
<td>20</td>
<td>1000</td>
<td>1024</td>
<td>8000</td>
<td>1012</td>
<td>750</td>
</tr>
<tr>
<td>Length of a row (not including LOBs)</td>
<td>2000</td>
<td>3996739</td>
<td>8060</td>
<td>32766</td>
<td>32677</td>
<td>32714</td>
</tr>
<tr>
<td>Columns in a UNIQUE constraint</td>
<td>6</td>
<td>32</td>
<td>16</td>
<td>120</td>
<td>64</td>
<td>64</td>
</tr>
<tr>
<td>Length of a UNIQUE constraint</td>
<td>120</td>
<td>5</td>
<td>DB_BLOCK_SIZE</td>
<td>900</td>
<td>32K</td>
<td>8192</td>
</tr>
<tr>
<td>Length of foreign key column list</td>
<td>120</td>
<td>5</td>
<td>DB_BLOCK_SIZE</td>
<td>900</td>
<td>32K</td>
<td>8192</td>
</tr>
<tr>
<td>Columns in a GROUP BY clause</td>
<td>6</td>
<td>255</td>
<td>Limited by length</td>
<td>Limited by length</td>
<td>1012</td>
<td>750</td>
</tr>
<tr>
<td>Length of GROUP BY column list</td>
<td>120</td>
<td>DB_BLOCK_SIZE</td>
<td>8060</td>
<td>32766</td>
<td>32677</td>
<td>16000</td>
</tr>
<tr>
<td>Sort specifications in ORDER BY clause</td>
<td>6</td>
<td>255</td>
<td>Limited by length</td>
<td>Limited by length</td>
<td>1012</td>
<td>750</td>
</tr>
<tr>
<td>Length of ORDER BY column list</td>
<td>120</td>
<td>DB_BLOCK_SIZE</td>
<td>8060</td>
<td>32766</td>
<td>32677</td>
<td>16000</td>
</tr>
<tr>
<td>Columns in a referential integrity constraint</td>
<td>6</td>
<td>32</td>
<td>16</td>
<td>120</td>
<td>64</td>
<td>64</td>
</tr>
<tr>
<td>Tables referenced in a SQL statement</td>
<td>15</td>
<td>No limit</td>
<td>No limit</td>
<td>No limit</td>
<td>No limit</td>
<td>No limit</td>
</tr>
<tr>
<td>Cursors simultaneously open</td>
<td>10</td>
<td>OPEN_CURSORS</td>
<td>2 gig</td>
<td>No limit</td>
<td>No limit</td>
<td>No limit</td>
</tr>
<tr>
<td>Items in a select list</td>
<td>100</td>
<td>1000</td>
<td>4096</td>
<td>8000</td>
<td>1012</td>
<td>750</td>
</tr>
</tbody>
</table>

- **FIPS**: Federal Information Processing Standards
- **Oracle 12c**: Oracle version 12c
- **Microsoft SQL Server 2012**: Microsoft SQL Server version 2012
- **DB2 for i 7.2**: IBM DB2 version 7.2 for iSeries
- **DB2 for LUW V10.5**: IBM DB2 version 10.5 for Linux, UNIX, and Windows
- **DB2 for z/OS V11**: IBM DB2 version 11 for z/OS

*DB_BLOCK_SIZE values are for reference only.*
VLDB – Very Large Database

How large is “big data” on IBM i?

- Non-partitioned table: 1.7 terabytes
- Local Partitioned table: 435.2 terabytes
- Non-partitioned index: 1.7 terabytes
- Local Partitioned index: 435.2 terabytes

Local Partitioning

Enabled with the use of a priced option

- DB2 Symmetric Multiprocessing (57nnSS1 Boss option 26)
- DB2 Multisystem (57nnSS1 Boss option 27)
- HA Journal Performance (57nnSS1 Boss option 42)
- Query Manager and SQL Development Kit (57nnST1)
DB2 Multisystem – Boss Option 27

- **Local Partitioned tables**
  - Partitioning is transparent to SQL applications
  - Hash and range partitioning
  - Data is spread across local partitions (members)
  - Single table can be spread across 256 partitions

- **Distributed partitioned tables**
  - Partitioning is transparent to all applications
  - Hash partitioning (user can control location of hash values)
  - Data is spread across separate IBM i nodes (systems or partitions)
  - A single table can be spread across up to 32 systems

**Frequently Recommended**

**Typically Not Used**
DB2 for i Scalability – Example

OrdersTable (partitioned by year)

Media-preference
UNIT SSD

Memory-preference
KEEP IN MEMORY YES

OrderDateIndex (partitioned)


Hot data
Cold data
Data Partitioning - Concepts

- Partitions are **MEMBERS** of a physical file

- Two types of data partitioning
  - **Hash** *(database decides how to organize the rows)*
  - **Range** *(user decides how to organize the rows)*

- SQL alias, view, or override to refer to single partition

- SQL sees the table as if it were one big member

- Native access only sees a single member

- The SQL Query Engine (SQE) has built-in partition avoidance when a WHERE clause is built over the partition key(s)
# Benefits of partitioning data

<table>
<thead>
<tr>
<th>Benefits</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Performance</strong></td>
</tr>
<tr>
<td>• Queries can <em>efficiently</em> target specific partitions</td>
</tr>
<tr>
<td>• <strong>Tactical</strong> use of fast storage devices</td>
</tr>
<tr>
<td><strong>Administration</strong></td>
</tr>
<tr>
<td>• <strong>Fast</strong> roll-in and roll-out</td>
</tr>
<tr>
<td>• Backup/recovery if all <em>changed data only</em> affects a subset of the partitions</td>
</tr>
<tr>
<td><strong>High Availability &amp; Disaster Recovery</strong></td>
</tr>
<tr>
<td>• <strong>Reduction</strong> in journal traffic, for those using logical replication</td>
</tr>
</tbody>
</table>
Benefits of partitioning data - Example

<table>
<thead>
<tr>
<th>Hypothetical Scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>Business processes</td>
</tr>
<tr>
<td>▪ A transactional file has 1 million INSERTs per day</td>
</tr>
<tr>
<td>▪ The table contains a date or timestamp column</td>
</tr>
<tr>
<td>▪ Three years of data are kept in the production master table</td>
</tr>
<tr>
<td>▪ When a new year begins, the oldest year’s worth of data is purged (deleted)</td>
</tr>
<tr>
<td>▪ Save while active is performed on the production libraries every day</td>
</tr>
<tr>
<td>▪ High availability (HA) logical replication is performed to another system</td>
</tr>
</tbody>
</table>
Non-partitioned Example

After a year’s time, remove the oldest year, see what happens…
Partitioned Example

Removing the oldest partition is simple and light-weight
Drawbacks of Data Partitioning

- **Possible** performance degradation
  - In general, the larger the number of partitions in a partitioned table, the greater the overhead in SQL data change and SQL data statements. You should create a partitioned table with the minimum number of partitions that are required to minimize this overhead.
  - It is also highly recommended that a parallelism degree greater than one be considered when accessing a partitioned table.

- Native database operations may need to use a logical file and a Format Selector (FMTSLR) program for writes through the logical file

- **The transition from non-partitioned → partitioned**
Creating Partitioned Tables

CREATE TABLE part (  
c1 INTEGER DEFAULT NULL ,  
c2 CHAR(10) CCSID 37 DEFAULT NULL )  
PARTITION BY RANGE (c1 NULLS LAST)  
(PARTITION part1  
STARTING ( MINVALUE ) INCLUSIVE  
ENDING ( 100 ) INCLUSIVE ,  
PARTITION part2  
STARTING ( 101 ) INCLUSIVE  
ENDING ( MAXVALUE ) INCLUSIVE )

CREATE TABLE parthash (  
c1 INTEGER DEFAULT NULL ,  
c2 CHAR(10) CCSID 37 DEFAULT NULL )  
PARTITION BY HASH (c1) INTO 4 PARTITIONS
Creating Partitioned Tables

CREATE TABLE part2 (  
c1 DATE DEFAULT NULL ,  
c2 CHAR(10) CCSID 37 DEFAULT NULL )  
PARTITION BY RANGE ( c1 NULLS LAST ) (  
STARTING ( ‘2014-01-01’ ) INCLUSIVE  
ENDING ( ‘2019-01-01’ ) EXCLUSIVE  
EVERY ( 1 YEAR ) )

- Alternate syntax for creating several partitions
- In this example, each partition is one year
Creating Partitioned Tables

- **PLAN for partitioning**
  - Make sure you understand how the data will be distributed
  - Analyze how data will be distributed across the partitions
  - Work with Lab Based Services to create a VLDB strategy

- **Active** Restrictions on partitioned tables
  - Range partitions cannot overlap
  - Floating point, Decimal Floating point, LOB, DataLink, and ROWID data types cannot be used in a partitioning key
  - No more than 256 partitions
Altering Partitioned Tables

CREATE TABLE newpart (  
c1 DATE DEFAULT NULL ,  
c2 CHAR(10) CCSID 37 DEFAULT NULL )

ALTER TABLE newpart  
   ADD PARTITION BY RANGE ( c1 NULLS LAST )  
   ( PARTITION p2003  
      STARTING ( MINVALUE ) INCLUSIVE  
      ENDING ( '2013-01-01' ) INCLUSIVE ,  
      PARTITION p2004  
      STARTING ( '2014-01-01' ) INCLUSIVE  
      ENDING ( MAXVALUE ) INCLUSIVE )

ALTER TABLE newpart  
   DROP PARTITIONING
Altering Partitioned Tables

**ALTER TABLE** `newpart`
**ALTER PARTITION** `p2004`

**STARTING** `{‘2004-01-01’} INCLUSIVE**
**ENDING** `{‘2005-01-01’} EXCLUSIVE**

**ALTER TABLE** `newpart`
**ADD PARTITION** `p2005`

**STARTING** `{‘2005-01-01’} INCLUSIVE**
**ENDING** `{MAXVALUE} INCLUSIVE**

**ALTER TABLE** `newpart`
**DROP PARTITION** `p2003`
**DELETE ROWS**
Planning for Partitioned Tables

- **PLAN** for any partition change before attempting to alter
  - Make sure you understand how the data will be distributed

```sql
WITH x AS ( SELECT CASE WHEN c1 < '2013-01-01' THEN 'p2013' WHEN c1 < '2014-01-01' THEN 'p2014' ELSE 'p2015' END AS partname FROM newpart)
SELECT partname AS "Part Name", COUNT(*) AS "Count"
FROM x
GROUP BY partname
```

- Bundle operations into a single ALTER statement to minimize data movement
- Don’t underestimate how long it will take to do the alter
  - Adding or dropping a partition is quick unless a non-partitioned index exists
  - Altering ranges will cause movement of data and index rebuilds
  - Altering from non-partitioned to partitioned will cause data movement and index rebuilds
- **Restrictions**
  - DDS Created files cannot be altered. You must convert to an SQL table first
  - All existing data in the table must be assignable to the specified ranges
Partitioned Indexes

Restrictions

- If a **partitioned** index is UNIQUE, the columns of the index must be the same or a superset of the columns of the data partition key. Otherwise the UNIQUE index must be non-partitioned.
CREATE ALIAS part_alias1 FOR part
CREATE ALIAS part_alias2 FOR part (part1)
CREATE INDEX parti3 ON part_alias2 (c1)

Restrictions
- The alias cannot be used in SQL schema statements.
Temporal tables and partitioned history

ALTER TABLE account
ADD COLUMN instance_begin
TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS ROW BEGIN
ADD COLUMN instance_end
TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS ROW END
ADD COLUMN transaction_id
TIMESTAMP(12) GENERATED ALWAYS AS TRANSACTION START ID
ADD PERIOD SYSTEM_TIME (instance_begin, instance_end)

CREATE TABLE account_hist LIKE account
PARTITION BY RANGE (instance_end)
(PARTITION p2016 STARTING ('01/01/2016') INCLUSIVE ENDING ('01/01/2017')
EXCLUSIVE,
PARTITION p2017 STARTING ('01/01/2017') INCLUSIVE ENDING ('01/01/2018')
EXCLUSIVE,
PARTITION p2018 STARTING ('01/01/2018') INCLUSIVE ENDING ('01/01/2019')
EXCLUSIVE,
PARTITION p2019 STARTING ('01/01/2019') INCLUSIVE ENDING ('01/01/2020')
EXCLUSIVE)

ALTER TABLE account
ADD VERSIONING USE HISTORY TABLE account_hist

Establish birth/death of a row
Create a history table, organized by range partitioning
Enable Temporal tracking
ALTER TABLE ATTACH and DETACH Partitions

ALTER TABLE DETACH PARTITION allows for the efficient roll-out of a partition that is no longer needed to be kept online.

- **ALTER TABLE DROP PARTITION** – Delete the data
- **ALTER TABLE DETACH PARTITION** – Retain the data, creating a new single partition, partitioned table

```
ALTER TABLE orders DETACH PARTITION p2012 INTO Archived_OrdersTable
```

OrdersTable (partitioned by year)

![Diagram showing partitioning and detaching partitions](image)

Archived_OrdersTable
DETACH PARTITION – Dependent object rules

Dependent objects on the source table (OrdersTable)

- Views are rebuilt to use the remaining partitions
- DDS-created logical files that reference all partitions and Spanning SQL indexes are rebuilt to use the remaining partitions
- MQTs are retained, but need to be refreshed by the user

Usage details

- Cannot be a system-period temporal table (use DROP VERSIONING on the ALTER TABLE statement)
- An Identity column will not be an identity column in the target table
- Constraints are not added to the target table
- When RCAC is active, a default row permission is activated on the target table
- Privileges are not propagated to the target table
ATTACH PARTITION – Dependent object rules

Dependent objects on the source table (Archived_OrdersTable)
- Views and MQTs are discarded
- Partitioned indexes which correspond with partitioned indexes on the target are retained, as long as they have a matching logical page size, otherwise partitioned indexes are discarded
- Active RCAC must match on the source and target

Dependent objects on the target table (OrdersTable)
- Views are rebuilt to include the new partition
- Spanning indexes are rebuilt to include the new partition
- MQTs are retained, but need to be refreshed by the user
- Partitioned indexes, with no corresponding partitioned index on the source are modified to accommodate for the new partition
# Catalogs & Partitioning

<table>
<thead>
<tr>
<th>Catalog Name</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSPARTITIONDISK</td>
<td>Partition disk usage</td>
</tr>
<tr>
<td>SYSPARTITIONMQTS</td>
<td>MQTs built over table partitions</td>
</tr>
<tr>
<td>SYSPARTITIONSTAT</td>
<td>Statistics and attributes for table partitions</td>
</tr>
<tr>
<td>SYSPARTITIONINDEXDISK</td>
<td>Statistics for indexes built over table partitions</td>
</tr>
<tr>
<td>SYSPARTITIONINDEXES</td>
<td>Indexes built over table partitions</td>
</tr>
<tr>
<td>SYSPARTITIONINDEXSTAT</td>
<td>Statistics for indexes built over table partitions</td>
</tr>
</tbody>
</table>
Navigator for i – Partition definition

Also supported in the Table wizard
Navigator for i – Show Partitions
Data Partitioning - Resources

- **White Paper:**
  Table Partitioning Strategies for DB2 for i
  [https://ibm.biz/PartitionedTablesIBMi](https://ibm.biz/PartitionedTablesIBMi)

- **Mike Cain Blog thread:**
  [http://db2fori.blogspot.com/2013/03/living-large.html](http://db2fori.blogspot.com/2013/03/living-large.html)

- **DB2 for i - VLDB Consulting Workshop:**
  [https://ibm.biz/DB2CoEworkshops](https://ibm.biz/DB2CoEworkshops)
DB2 for i priced OS options – evaluation copy

Try before you buy! On any IBM i 7.x release!

DB2 Symmetric Multiprocessing – Option 26
DB2 Multisystem – Option 27

The IBM Lab Services DB2 for IBM i team has the ability to allow you to evaluate either of these options for up to 70 days, for no charge.

This is a simpler, no strings attached, way to evaluate these valuable database options.

Interested?

Contact…
Rob Bestgen (bestgen@us.ibm.com) or Scott Forstie (forstie@us.ibm.com)
www.ibm.com/developerworks/ibmi/techupdates/db2
Partitioning Scalar Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATAPARTITIONNAME</td>
<td>Returns the partition name where a row is located</td>
</tr>
<tr>
<td>DATAPARTITIONNUM</td>
<td>Returns the partition number of a row</td>
</tr>
<tr>
<td>DBPARTITIONNAME</td>
<td>Returns the relational database name where a row is located</td>
</tr>
<tr>
<td>DBPARTITIONNUM</td>
<td>Returns the node number of a row</td>
</tr>
<tr>
<td>HASH</td>
<td>Returns the partition number of a set of values</td>
</tr>
<tr>
<td>HASHED_VALUE</td>
<td>Returns the partition map index number of a row</td>
</tr>
</tbody>
</table>