DB2 for z/OS: Dynamic Statement Cache

1 Summary
The following document is based on IBM DB2 12 for z/OS but also refers to functions that were added in previous versions of DB2 for z/OS.
It provides information about dynamic statement caching (DSC) and describes the features and functions associated with it.

1.1 High level point of view
The DSC was introduced in DB2 for OS/390 Version 5 to support applications that use dynamic SQL, such as PeopleSoft and SAP.
Each SQL statement must be prepared before it is executed. For static SQL, most of this work is done during the BIND process and is performed only once. For dynamic SQL, the PREPARE request is done before the SQL statement is executed. Before the DSC existed, a dynamic SQL statement had to be prepared every time it was executed, even if it was identical to a previously prepared statement.
The main reason for the DSC is based on the fact that the PREPAREs for the same SQL statement should be preserved: the statement is prepared only once and executed multiple times. Of course, this approach brings up the following questions:

- When is a SQL statement considered to be the same?
- What happens if the underlying database design is changed?
- What does the selected access path look like?

This document attempts to answer all these questions and outlines the features that were introduced after DB2 for OS/390 Version 5 to meet the challenges posed by these questions. It also explains new functionality that was introduced by IBM DB2 Analytics Accelerator or by DB2 12 for z/OS, e.g., Dynamic Plan Stability.

This document provides suggested best practices. It is not intended to be exhaustive and is provided on an as-is basis.
2 Table of Contents

1 Summary ................................................................................................................. 1

1.1 High level point of view .................................................................................. 1

2 Table of Contents .................................................................................................. 2

3 How does Dynamic Statement Caching work? .................................................. 4

3.1 Enable DSC ........................................................................................................... 4

3.1.1 CACHEDYN ...................................................................................................... 4

3.1.2 EDMSTMTC ..................................................................................................... 4

3.2 Match the cache ................................................................................................... 5

3.2.1 SQL statement text .......................................................................................... 5

3.2.2 Query rewrite considerations ......................................................................... 6

3.2.3 Authorization ID ............................................................................................. 6

3.2.4 Special register ................................................................................................ 8

3.2.5 CURSOR with HOLD ...................................................................................... 9

3.2.6 Isolation levels ................................................................................................ 10

3.2.7 DSNZPARM changes .................................................................................... 10

3.3 Invalidation of a cache entry ............................................................................. 12

3.3.1 Running an index DDL .................................................................................. 13

3.3.2 Running an utility against an object ................................................................. 14

3.3.3 Running a REVOKE ........................................................................................ 15

4 Looking into the DSC ............................................................................................. 17

4.1 EXPLAIN statement .......................................................................................... 17

4.2 SET CURRENT EXPLAIN MODE register ....................................................... 19

4.3 IFCID 316/317/318 ............................................................................................... 19

5 Dynamic plan stability ............................................................................................ 20

5.1 Implementation ..................................................................................................... 20

5.1.1 DSNZPARM .................................................................................................... 20

5.1.2 Commands ...................................................................................................... 20

5.1.3 Catalog tables .................................................................................................. 23

5.2 Comparison DSC entry: Stabilized DSC entry .................................................. 23

6 Special considerations .............................................................................................. 25

6.1 Additional PREPARE keywords ....................................................................... 25

6.1.1 Statement concentration ................................................................................. 25

6.1.2 Extended indicators ........................................................................................ 27

6.2 Re-optimization of statements .......................................................................... 29

6.3 Keep dynamic statements ................................................................................... 30

6.4 DB2 Analytics Accelerator and dynamic statement caching ......................... 31

6.4.1 Short introduction ........................................................................................... 31

6.4.2 Dynamic Statement Caching for accelerated queries .................................... 31

6.4.3 DSC checking for accelerator queries ............................................................ 32

6.4.4 Adding to or removing tables from the accelerator .......................................... 33

6.4.5 Accelerator-only tables .................................................................................. 33

7 Appendixes ............................................................................................................... 34

7.1 Recent APARs ..................................................................................................... 34

7.2 DSNZPARM settings .......................................................................................... 36

7.3 Performance numbers .......................................................................................... 36

8 References .................................................................................................................. 38
3 How does Dynamic Statement Caching work?

3.1 Enable DSC
Set the DSNZPARMs CACHEDYN to YES and EDMSTMTC to an appropriate value to enable the DSC. The DSC exists separately on each member of a data sharing group.

3.1.1 CACHEDYN
The global switch to enable the DSC is the DSNZPARM CACHEDYN. Disabling caching by setting CACHEDYN=NO will not prevent DB2 from executing any dynamic SQL, but it will have a significant impact on performance and concurrency. CACHEDYN is an online changeable DSNZPARM. Changing the value from NO to YES enables the DSC. Changing the value from YES to NO does not clear the cache; it only prevents new statements from going into the cache.

3.1.2 EDMSTMTC
The DSNZPARM EDMSTMTC is also an online changeable DSNZPARM, which can be increased as shown in the following figure:

```
DSNZ006I -DB2A DSNZCMD1 SUBSYS DB2A SYSTEM
PARAMETERS LOAD MODULE NAME DB2AZDSN IS BEING LOADED
DSNG012I -DB2A EDM STMT POOL HAS
  OLD LIMIT  10240000
  NEW LIMIT  20480000
  AlLOCATED SIZE  7483392
DSNZ007I -DB2A DSNZCMD1 SUBSYS DB2A SYSTEM
PARAMETERS LOAD MODULE NAME DB2AZDSN LOAD COMPLETE
DSN9022I -DB2A DSNZCMD0 'SET SYSPARM' NORMAL COMPLETION
```

Or decreased, as shown in the following figure:

```
DSNZ006I -DB2A DSNZCMD1 SUBSYS DB2A SYSTEM
PARAMETERS LOAD MODULE NAME DB2AZDSN IS BEING LOADED
DSNG012I -DB2A EDM STMT POOL HAS
  OLD LIMIT  20480000
  NEW LIMIT  5120000
  AlLOCATED SIZE  5103616
DSNZ007I -DB2A DSNZCMD1 SUBSYS DB2A SYSTEM
PARAMETERS LOAD MODULE NAME DB2AZDSN LOAD COMPLETE
DSN9022I -DB2A DSNZCMD0 'SET SYSPARM' NORMAL COMPLETION
```

The storage area for one statement includes the following parts:
- Storage for control blocks
- Storage for a PT (=package table) header
- Storage for the statement (above the bar)
- Storage for the explain blocks if needed
In the case of a simple and short SQL statement, the storage amount is approximately 50K per statement.

If the cache is full, statements that are not in use are replaced by LRU algorithm. Only the following statements can be cached:

- DELETE
- INSERT
- MERGE
- SELECT and common table expressions (WITH ...)
- UPDATE
- TRUNCATE

Statements that reference Declared Global Temporary Tables are not cached.

3.2 Match the cache

To reuse an existing DSC entry, the currently executed SQL statement must match several criteria. Of course, the most important one is the SQL statement text, but it is not the only criterion. The following sections discuss the criteria for a cache match:

3.2.1 SQL statement text

The currently executed statement text is considered a cache match if it is exactly the same as the cached statement text.

It is NOT a match if it has more or fewer blanks:

```
User-A
```

```
SELECT * FROM SYSIBM.SYSTABLESPACE
WHERE PARTITIONS = 00002
  AND DBNAME <> 'DSNDB06'
WITH UR;

SELECT * FROM SYSIBM.SYSTABLESPACE
WHERE PARTITIONS = 00002
  AND DBNAME <> 'DSNDB06'
WITH UR;
```

```
<table>
<thead>
<tr>
<th>STMT_ID</th>
<th>... STMT_TEXT</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>SELECT * FROM SYSIBM.SYSTABLESPACE WHERE PARTITIONS = 00002 AND DBNAME &lt;&gt; 'DSNDB06' WITH UR</td>
</tr>
<tr>
<td>3</td>
<td>SELECT * FROM SYSIBM.SYSTABLESPACE WHERE PARTITIONS = 00002 AND DBNAME &lt;&gt; 'DSNDB06'</td>
</tr>
</tbody>
</table>
```

Dynamic Statement Cache

Figure 1: Equal statements - 1

Consider some special tooling behaviour; for example, DSNTEP2 strips the blanks, whereas SPUFI does not strip the blanks.

It is NOT a match if it is syntactically the same but has:

- Additional correlation names
- Exchanged predicates, such as B=A instead of A=B
- ...

3.2.2 Query rewrite considerations

When preparing a SQL statement, DB2 first parses the statement text and tries to find a semantically identical statement that might give the DB2 optimizer more choices for the best access path. This phase is called query rewrite. By explaining the SQL statement, a user can see how the original statement text has been transformed.

For statement matching, DB2 always compares the original statement text (which is subject to the PREPARE statement) with existing entries in the DSC. Externalizing the DSC with the EXPLAIN STMTCACHE ALL statement always dumps the original statement text into DSN_STATEMENT_CACHE_TABLE.

3.2.3 Authorization ID

If the statement is executed by another current SQLID, it is inserted into the cache a second time, but now under the new SQLID.

Following are some examples:
Figure 3: Different user IDs

The same statement now the same current SQLID:

```
User-A
SELECT * FROM
SYSBM.SYSTABLESPACE
WHERE PARTITIONS = 00002
AND DBNAME <> 'DSNDB06'
WITH UR ;
```

```
User-B
SELECT * FROM
SYSBM.SYSTABLESPACE
WHERE PARTITIONS = 00002
AND DBNAME <> 'DSNDB06'
WITH UR ;
```

Dynamic Statement Cache

```
<table>
<thead>
<tr>
<th>STMT_ID</th>
<th>PRIMAUTH</th>
<th>CURSQLID</th>
<th>STMT_TEXT</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>User-A</td>
<td>User-A</td>
<td>SELECT * FROM SYSBM.SYSTABLESPACE WHERE PARTITIONS = 00002 AND ...</td>
</tr>
<tr>
<td>3</td>
<td>User-B</td>
<td>User-B</td>
<td>SELECT * FROM SYSBM.SYSTABLESPACE WHERE PARTITIONS = 00002 AND ...</td>
</tr>
</tbody>
</table>
```

Figure 4: Current SQLID

The following figure shows the current path register difference. Refer to the CURRENT PATH topic in the *DB2 for z/OS SQL Reference* for information about how this register is set in conjunction with CURRENT SQLID:
### 3.2.4 Special register

A statement can be reused only if the following special registers do not change:

- CURRENT SQLID
- CURRENT PATH (refer to previous section)
- CURRENT APPLICATION COMPATIBILITY
- CURRENT DECFLOAT Rounding MODE
- CURRENT DEGREE
- CURRENT LOCALE LC_CTYPE
- CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION
- CURRENT OPTIMIZATION HINT
- CURRENT PRECISION
- CURRENT REFRESH AGE
- CURRENT RULES

The following example shows that the same SQL statement, which is executed by the same user, is cached twice. This is due to the fact that one of the above registers change: CURRENT DEGREE
### 3.2.5 CURSOR with HOLD

For statement matching, DB2 also considers whether a cursor was declared with or without the HOLD option. It creates separate cache entries if all other criteria match but the cursor declaration is different.

#### Figure 6: Special registers

<table>
<thead>
<tr>
<th>STMT_ID</th>
<th>BIND_DEGREE</th>
<th>STMT_TEXT</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>A</td>
<td>SELECT * FROM SYSBM.SYSTABLESPACE WHERE PARTITIONS = 00002 AND DBNAME &lt;&gt; 'DSNDB06' WITH UR ;</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>SELECT * FROM SYSBM.SYSTABLESPACE WHERE PARTITIONS = 00002 AND DBNAME &lt;&gt; 'DSNDB06' WITH UR ;</td>
</tr>
</tbody>
</table>

Dynamic Statement Cache

#### Figure 7: Cursor with HOLD

Note that the DB2 default for the DECLARE statement is WITHOUT HOLD. For Java connections, DB2 uses `HOLD_CURSORS_OVER_COMMIT as the default`. You must specify explicitly `CLOSE_CURSOR_OVER_COMMIT` for the result set or as default property for your connection if your application can tolerate this.

For Java, different cursor hold usage also results in different package names. The fifth character in the package name typically indicates whether a hold cursor was used (e.g., package name SYSLLH200) or the cursor was closed at commit (e.g., package name SYSLN200).
3.2.6  Isolation levels

Different isolation levels for the same statement also result in different cache entries. Note that for JDBC applications which use the standard JDBC packages the package name will also differ from isolation level to isolation level.

The following example shows a code example from REXX. For REXX, the isolation level can dynamically be set with the CURRENT PACKAGESET special register. For Java, the isolation level can be specified as a method of a connection object.

```
User-A

SET CURRENT PACKAGESET='DSNREXUR';

DECLARE C01 CURSOR FOR
SELECT * FROM SYSEIBM.SYSTABLESPACE
WHERE PARTITIONS = 00002
AND DBNAME = 'DSNB00';

SET CURRENT PACKAGESET='DSNREXCS';

DECLARE C02 CURSOR FOR
SELECT * FROM SYSEIBM.SYSTABLESPACE
WHERE PARTITIONS = 00002
AND DBNAME = 'DSNB00';
```

<table>
<thead>
<tr>
<th>STMT_ID</th>
<th>PROGRAM_NAME</th>
<th>BIND_ISO</th>
<th>STMT_TEXT</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>DSNREXX</td>
<td>UR</td>
<td>SELECT * FROM SYSEIBM.SYSTABLESPACE WHERE PARTITIONS = 00002 AND DBNAME ...</td>
</tr>
<tr>
<td>3</td>
<td>DSNREXX</td>
<td>CS</td>
<td>SELECT * FROM SYSEIBM.SYSTABLESPACE WHERE PARTITIONS = 00002 AND DBNAME ...</td>
</tr>
</tbody>
</table>

Dynamic Statement Cache

3.2.7  DSNZPARM changes

Be cautious when you make online changes to DSNZPARM values because DSNZPARM keywords such as BIF_COMPATIBILITY can affect the result of a statement. The statements are not automatically invalidated if DSNZPARM values are changed by, for example, SET SYSPARM RELOAD. If a statement exists before the online DSNZPARM change, it is reused with all its implications. The following flow shows such an example:
BIF_COMPATIBILITY=V9.Decimal_VARCHAR

User-A

```sql
SELECT 
  CAST(PARTITIONS AS DECIMAL,5) AS #PART
, DBNAME FROM SYSIBM.SYSTABLESPACE
WHERE PARTITIONS = 00003
AND DBNAME = 'DSNDB06'
WITH UR;
```

BIF_COMPATIBILITY=CURRENT

Online change by SET SYSPARM

```sql
SELECT 
  CAST(PARTITIONS AS DECIMAL,5) AS #PART
, DBNAME FROM SYSIBM.SYSTABLESPACE
WHERE PARTITIONS = 00003
AND DBNAME = 'DSNDB06'
WITH UR;
```

Figure 9: BIF_COMPATIBILITY

The following flow shows the behaviour in the case of an APPLCOMPAT change by REBIND:

```
User-A runs package ... with APPLCOMPAT(V12R1M500)
```

```
SELECT * FROM 
  SYSIBM.SYSTABLESPACE
WHERE PARTITIONS = 00002
AND DBNAME = 'DSNDB06'
WITH UR;
```

Rebind Package ... APPLCOMPAT(V11R1)

```
User-A runs package ... with APPLCOMPAT(V11R1)
```

```
SELECT * FROM 
  SYSIBM.SYSTABLESPACE
WHERE PARTITIONS = 00002
AND DBNAME = 'DSNDB06'
WITH UR;
```

<table>
<thead>
<tr>
<th>STMT_ID</th>
<th>APPLCOMPAT</th>
<th>STMT_TEXT</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>V12R1M500</td>
<td>SELECT * FROM SYSIBM.SYSTABLESPACE WHERE PARTITIONS = 00002 AND DBNAME ...</td>
</tr>
<tr>
<td>5</td>
<td>V11R1</td>
<td>SELECT * FROM SYSIBM.SYSTABLESPACE WHERE PARTITIONS = 00002 AND DBNAME ...</td>
</tr>
</tbody>
</table>

Dynamic Statement Cache

Figure 10: APPLCOMPAT change
Note: The column APPLCOMPAT was added to DSN_STATEMENT_CACHE_TABLE in DB2 12 for z/OS. In version 11, you might see identical statements in the DSC that differ only in the non-visible APPLCOMPAT setting that was active during prepare.

3.3 Invalidation of a cache entry

The previous example shows how important it is that statements in the cache are automatically invalidated or can be invalidated by special means.

The following actions are a few examples that result in automatic invalidation:

- An ALTER is issued against objects that are referred by the DSC entry (e.g., setting a new data type, changing the AUDIT flag, etc.).
- A CREATE/DROP index is done for objects that are referred by the DSC entry.
- A REVOKE is issued for the user who inserted the DSC entry.
- Utilities are run against the objects in the statement (e.g., materializing REORG, REBUILD index, collecting statistics by RUNSTATS, etc.).

Basically, this process is equal to the package invalidation process as in case of static SQL.

You can invalidate the cache with an explicit, but otherwise a dummy, action by using RUNSTATS. You can use the following parameters:

- INVALIDATECACHE YES (new with DB2 12)
- REPORT NO UPDATE NONE (except HISTORY ALL is used – see APAR PI66197)
- RESET ACCESSPATH (new with DB2 11)

INVALIDATECACHE YES is the default if REPORT NO UPDATE NONE or RESET ACCESSPATH is used, except if HISTORY ALL is used (PI66197).

This default invalidation cannot be overwritten by INVALIDATECACHE NO. In this case, the DSNU070I message is issued instead:

```
DSNU050I 139 05:38:38.69 DSNUGUTC - RUNSTATS TABLESPACE DSNDB06.SYSTSTSP
      TABLE(ALL) INDEX(ALL) REPORT NO UPDATE NONE
DSNU1383I -DB2A 139 05:38:38.82 DSNUSIVC - DYNAMIC STATEMENT CACHE WAS
      INVALIDATED,
      REASON = INVALIDATECACHE YES
DSNU010I 139 05:38:38.83 DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST
      RETURN CODE=0
```

The default for INVALIDATECACHE in the utilities is NO. This setting results in different behaviour of RUNSTATS after change to DB2 12: Statements are not invalidated unless INVALIDATECACHE=YES is specified or enforced.

The following examples illustrate the automatic invalidation process:
### 3.3.1 Running an index DDL

The following figure shows the flow for creating an index. Even if the CREATE INDEX statement is done on another member, the cache is invalidated on all members. Where appropriate, this notify processing of other members also occurs for other DDL or utility processes.

If the index is created with DEFER YES, the cache is also invalidated, although it is not necessary. While an index that was created with DEFER YES is in rebuild pending (RBDP) status, dynamic statements that are not dependent on the index (e.g., for uniqueness checking) are cached again in the DSC. REBUILD INDEX makes the index available for access path selection and invalidates the DSC.

```sql
SELECT * FROM SYSIBM.SYSTABLESPACE WHERE PARTITIONS = 00002 AND DBNAME <> 'DSND806' WITH UR ;
```

---

**Member A**

**Member B**

**STMT_ID** … **STMT_TEXT**

| 2   | … SELECT * FROM SYSIBM.SYSTABLESPACE WHERE PARTITIONS = 00002 AND … |

**Dynamic Statement Cache**

**INVALIDATION request also send to other members**

**STMT_ID** … **STMT_TEXT**

**Dynamic Statement Cache**

---

**Figure 11: Invalidation by CREATE INDEX**

The following figure shows an example for an ALTER INDEX:
Figure 12: Invalidation by ALTER INDEX

Note: As long as an index is NOT UNIQUE or uniqueness does not need to be enforced (e.g., during UPDATE), an index can be bypassed during access path selection if this index is in RBDP or Page set REBUILD-pending (PSRBD) status.

3.3.2 Running an utility against an object

Utilities such as LOAD REPLACE, REBUILD, and REORG can also invalidate the statement cache. In DB2 12, the INVALIDATECACHE keyword was introduced to enforce this. However, some utility processes enforce automatic invalidation independent of the keyword setting, as shown in the following figure:

```
DSNU050I 145 06:50:19.34 DSNUGUTC - REBUILD INDEX(ALL) TABLESPACE DSNDB06.SYSTSTSP
DSNU1383I -DB2B 145 06:50:20.08 DSNUINV - DYNAMIC STATEMENT CACHE WAS INVALIDATED, REASON = OBJECT IN RESTRICTIVE STATE
```

Exception: RECOVER to a point in time (=PIT) does not invalidate the statement cache.

Consider the following example:

- Assume a table T1 (in table space TS1) has a unique index, X1.
- The following statement is executed: SELECT ... FROM T1 WHERE unique key = 10. This statement is stored in the cache.
- RECOVER TABLESPACE ... TOLASTCOPY is done.
- The index, X1, is set to RBDP:
The statement `SELECT ... FROM T1 WHERE unique key = 10` is executed again, but now returns:

```
SELECT * FROM SZI10T WHERE PNR = 10 WITH UR ;
```

SQLERROR ON SELECT COMMAND, FETCH FUNCTION RESULT OF SQL STATEMENT:

```
DSNT408I SQLCODE = -904, ERROR: UNSUCCESSFUL EXECUTION CAUSED BY AN UNAVAILABLE RESOURCE. REASON 00C900AE, TYPE OF RESOURCE 00000201, AND RESOURCE NAME SZI10D .SZI10X
```

However, another statement, `SELECT ... FROM T1 WHERE unique key = 20`, returns the expected result:

```
SELECT * FROM SZI10T WHERE PNR = 20 WITH UR ;
... SUCCESSFUL RETRIEVAL OF 1 ROW(S)
```

The reason is that the last statement is newly prepared because it is not already in the statement cache, and during PREPARE the IX1 (in RBDP) is bypassed.

### 3.3.3 Running a REVOKE

After the SELECT privilege on SYSIBM.SYSTABLESPACE is revoked, the cache entry is invalidated for the specified user.
Figure 13: Invalidation by REVOKE

As a consequence, authority checking is not performed if a statement is found in the cache.
4 Looking into the DSC

4.1 EXPLAIN statement

The EXPLAIN statement can be used to look into the dynamic statement cache. The EXPLAIN statement works only on the member level and exists in three flavours:

- **EXPLAIN STMTCACHE ALL**, which stores all statements into `sqlid.DSN_STATEMENT_CACHE_TABLE`.

- The STMTCACHE STMTID clause can be specified on the EXPLAIN statement. STMTID has the value of STMT_ID from the appropriate `sqlid.DSN_STATEMENT_CACHE_TABLE` entry. If EXPLAIN is used with STMT_ID, the following tables are populated:
  - `sqlid.PLAN_TABLE`
  - `sqlid.DSN_STATEMENT_TABLE`
  - `sqlid.DSN_FUNCTION_TABLE`
  - `sqlid.DSN_STATEMENT_CACHE_TABLE`

- **EXPLAIN STMTCACHE STMTTOKEN** can be used in a way that's similar to STMTID. The appropriate column in `sqlid.DSN_STATEMENT_CACHE_TABLE` is STMT_TOKEN. The same tables listed above are populated. The value of STMTTOKEN can be set by RRSAF SET_ID or by the sqleseti API if the program is connected remotely.

EXPLAIN STMTCACHE STMTID and STMTTOKEN put the same information into DSN_STATEMENT_CACHE_TABLE as EXPLAIN STMTCACHE ALL does, but limited to the specified statement only.

Note that EXPLAIN STMTCACHE ALL requires SQLADM, SYSADM, or System DBADM authority to dump the complete DSC to the DSN_STATEMENT_CACHE_TABLE. Users without that authority can execute the EXPLAIN but will see the statements with the same authorization ID as their current SQLID only. You can bypass this restriction by enabling the stored procedure SYSPROC.OPT_RUNSQL. This procedure allows invokers to explain the complete DSC. The load module, the associated DBRM, and the DDL for the procedure are delivered with the no-charge version of IBM Data Studio. See the IBM Data Studio in Knowledge Center for more information.

If you look at an access path with the one of the above methods and compare the access path with a current EXPLAIN PLAN, both access paths might have differences. Any differences are due to the fact that the EXPLAIN statement generates the access path at the current point in time, and the access path stored in the DSC can have an older creation date. Operations like DROP INDEX, ... do not generate such a discrepancy because they invalidate the DSC entry, but for example DSNZPARM changes of bufferpool settings can have this effect.

The following example shows RUNSTATS and the default behaviour that was introduced in DB2 12: INVALIDATECACHE NO.
In this example, RUNSTATS with INVALIDATECACHE NO preserves the existing cache entry with STMT_ID 4 and its access path (via index USRIXTS). But the most recent statistics would bring the DB2 optimizer to choose a different access path (via index DSNDSX01) the next time that the statement is prepared. This access path will be chosen and cached only if the existing cache entry is removed.

Tips:

- You can link between the different tables by using the QUERYNO clause, which contains the STMTID (or STMT_ID) value. The COLLID of such rows has the value of DSNDYNAMICSQLCACHE.

- The sqlid.DSN_STATEMENT_CACHE_TABLE also contains runtime accounting data. This data becomes visible if tracing for IFCID 318 is active (e.g., command `-STA TRACE(PERFM) CLASS(30) IFCID(318)` is issued). The appropriate accounting trace does not need to be started; this DSC data is collected independently. The accounting data is cumulative. The accounting data, when used together with the counter STAT_EXECB, can also provide averages.

The accounting data is helpful to get a deeper understanding of the performance metrics of individual SQL statements. Be aware that dumping the DSC into DSN_STATEMENT_CACHE_TABLE provides a snapshot of the current state of the DSC only. Even if you take several snapshots in a certain time period, you might miss statements that have been cached for a short time only.

Besides accounting data (e.g., CPU- and suspension times), DSC also provides information about RID failures or the statement’s selectivity (columns STAT_EROWB and STAT_PROWB).
4.2 SET CURRENT EXPLAIN MODE register

If the special register SET CURRENT EXPLAIN MODE is set to YES or EXPLAIN, appropriate information is directly inserted into sqlid.PLAN_TABLE and also into sqlid.DSN_STATEMENT_CACHE_TABLE.

In both cases, nothing is added to the DSC itself, although the statement is executed if SET CURRENT EXPLAIN MODE = YES.

These rows can be separated or located by COLLID=DSNEXPLAINMODEYES or COLLID=DSNEXPLAINMODEEXPLAIN.

When you set CURRENT EXPLAIN MODE to YES, consider the fact that explain tables are populated in the same unit of work as the SQL statement itself. So a long-running SQL statement generates a long-running unit of work with all its implications.

The default of the register SET CURRENT EXPLAIN MODE is NO.

The following table summarizes the different behaviours:

<table>
<thead>
<tr>
<th>SET CURRENT EXPLAIN MODE setting</th>
<th>NO</th>
<th>YES</th>
<th>EXPLAIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>sqlid.PLAN_TABLE</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>sqlid.DSN_STATEMENT_CACHE</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Entry in DSC</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Statement is executed</td>
<td>Yes</td>
<td>Yes</td>
<td>No, SQLCODE+217</td>
</tr>
</tbody>
</table>

Table 1: SET CURRENT EXPLAIN MODE register

4.3 IFCID 316/317/318

Another option that you can use to look into the dynamic statement cache is to write a monitor program that executes a READS command by calling DSNWLI, the IFI interface. You can control the data that is returned by setting the appropriate fields in the DSNDQWAL macro.

Note the following considerations for using this option:

- The monitor trace class 29 must be active or the appropriate IFCIDs.
- The monitor program must request IFCID 316. To get the full SQL statement the monitor program must start and read IFCID 317 with the appropriate DSNDQWAL settings.
- READS is a synchronous interface, which means that statements that are removed from the dynamic statement cache are not visible to the monitor program.
- If the IFCID trace is started against GTF/SMF (e.g., -STA TRA(PERFM) CLASS(30) IFCID(316,318) DEST(SMF)), invalidated statements are written to the destination. IFCID 316 captures the first 60 bytes of SQL text. The full SQL text, as well as the attribute string, are provided by IFCID 317.

For a complete description, see the Knowledge Center topic “Monitoring the dynamic statement cache with READS calls”.

5 Dynamic plan stability

Starting with DB2 12, you can freeze dynamic SQL statements from the dynamic statement cache into the DB2 catalog tables that are related to dynamic plan stability (DPS). This feature allows you to reuse the prepared form of the dynamic queries even after they are removed from the cache or DB2 is recycled. This behaviour is similar to how bound static statements can be saved during the static bind process. The matching criteria for finding an appropriate entry in the catalog are the same as for matching an entry in the DSC (refer to section 3.2, Match the cache).

The DB2 catalog tables related to DPS enlarge the storage hierarchy for prepared dynamic SQL statements, as shown in the following figure:

![Dynamic statement hierarchy diagram](figure15.png)

**Figure 15: Dynamic statement hierarchy**

Currently the following restrictions exist:

- Only statements from DSC can be frozen in the DB2 catalog tables (no stabilization from “plain SQL text” is possible).
- Statement concentration is excluded.
- Temporal and transparent archive queries are excluded.
- Queries that are inserted under REOPT(AUTO) control are excluded.

5.1 Implementation

5.1.1 DSNZPARM

You can use an online changeable DSNZPARM called CACHEDYN_STABILIZATION to control dynamic plan stability. You can set this parameter to CAPTURE (DSC into DPS catalog tables), LOAD (DPS catalog tables into DSC), or BOTH. The default for CACHEDYN_STABILIZATION is BOTH. To disable this functionality, set this parameter to NONE.

5.1.2 Commands

The command `START DYNQUERYCAPTURE` starts the collection process from the DSC into the appropriate catalog tables. The set of queries to be captured is assigned to a STBLGRP (stabilization group) name. This name allows all of these statements to be handled in a single operation, e.g., during SELECT or during FREE. The command allows filtering on CURRENT SQLID and on individual STMT_IDS or statement tokens.
The decisive parameter is THRESHOLD, which correlates to the counter STAT_EXECB in the DSC. The counter STAT_EXECB is maintained only if IFCID 318 is active. Consider the following terms:

- The default value for THRESHOLD is 2.
- A value of 0 cannot be specified for THRESHOLD.

Therefore, you can stabilize queries only if IFCID 318 is on. If it is not on, all STAT_EXECB counters are 0, and 0 cannot be used as an input value for THRESHOLD.

The following scenario illustrated how this counter is populated: assume that a statement has already been stabilized statement by data sharing member A, and that data sharing member B wants to stabilize the same query again. Now the stabilized task encounters the statement as an ALREADY STABILIZED statement. If the same member wants to stabilize the same statement again, it does not schedule the stabilization because the DSC entry of this statement is flagged as STABILIZED = Y.

If the DSNZPARM setting does not allow the capture, the -START DYNQUERYCAPTURE command is rejected:

```
DSNZ018I -DB2A DSNXESTC REQUESTED ACTION 'START DYNQUERYCAPTURE' IS DISABLED BY THE 'CACHEDYN_STABILIZATION' SUBSYSTEM PARAMETER SETTING
DSN9023I -DB2A DSNXESTC 'START DYNQUERYCAPTURE' ABNORMAL COMPLETION
```

Another option is to start the collection as a background task (MONITOR YES) and stabilize the queries, which fulfills the criteria. The following -DISPLAY DYNQUERYCAPTURE command shows the background task and the selection criteria:
This background task is related to the correlation ID 014.RTSTST00 with plan name RTS in case resources such as DSNDB06.SYSTSDQY are not available. The correlation ID of the foreground task (MONITOR NO) is 022.XESTC 01, and the associated plan name is DPS.

You can end these background processes by run the `STOP DYNQUERYCAPTURE` command:

```
-STOP DYNQUERYCAPTURE CNO(17) SCOPE(GROUP)
DSNX224I -DB2B DSNXESPC STOP DYNAMIC QUERY CAPTURE FOR COMMAND NUMBER (17) COMPLETED SUCCESSFULLY, WITH 1 QUERY CAPTURES STOPPED.
```

To delete the stabilized statements, run the DSN subcommand `FREE STABILIZED DYNAMIC QUERY` command. It deletes the entries from SYSIBM.SYSDYNQRY and SYSIBM.SYSDYNQRYDEP and also from the DSC:

```
FREE STABILIZED DYNAMIC QUERY STBLGRP(TEST)
DSNT340I -DB2A FREE STABILIZED DYNAMIC QUERY COMPLETED SUCCESSFULLY FOR 3 STATEMENTS.
```

**Note:** This counter represents the number of statements that were freed. This does not need to be equal to the number of rows, because some statements can have a valid entry (COPYID=0) and an invalid entry (COPYID=4).

If the stabilized group is not found, the job ends with RC=8, and the following messages are issued:

```
FREE STABILIZED DYNAMIC QUERY STBLGRP(TEST)
DSNT346I -DB2A FREE ERROR
STABILIZED DYNAMIC QUERY NOT FOUND FOR STBLGRP TEST
DSNT341I -DB2A UNSUCCESSFUL FREE FOR STABILIZED DYNAMIC QUERY
```
5.1.3 Catalog tables

The stabilized queries are stored in the table SYSIBM.SYSDYNQRY in table space DSNDB06.SYSTSDQY. In addition, the dependent objects are stored in SYSIBM.SYSDYNQUERYDEP in table space DSNDB06.SYSTSDQD. The last table is required when objects are altered so that the associated entry in SYSIBM.SYSDYNQRY can be invalidated. If an entry in SYSIBM.SYSDYNQRY is invalidated, the associated SYSIBM.SYSDYNQRYDEP rows are deleted. The RUNSTATS utility and other utilities mentioned in section 3.3.2 do not invalidate the stabilized entries.

The column LASTUSED is updated every 24 hours or during DB2 shutdown.

5.2 Comparison DSC entry: Stabilized DSC entry

The following table summarizes the similarities and differences between dynamic statement cache entries and stabilized dynamic statements:

<table>
<thead>
<tr>
<th></th>
<th>DSC entry</th>
<th>Stabilized DSC entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Persistent</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Statement must be equal, etc.</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Special register must be equal, etc.</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Invalidation by DDL</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Invalidation by utilities (RUNSTATS, etc.)</td>
<td>Depends on utility and options of utility job</td>
<td>No</td>
</tr>
<tr>
<td>Invalidation by REVOKE</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Invalid entry</td>
<td>Removed</td>
<td>Kept (only one entry)</td>
</tr>
<tr>
<td>FREE STABILIZED QUERY</td>
<td>Removed</td>
<td>Removed</td>
</tr>
<tr>
<td>EXPLAIN on statement is possible</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 2: Comparison DSC entry versus stabilized DSC entry

If a statement is stabilized from DSC, the following fields of sqlid.DSN_STATEMENT_CACHE_TABLE are updated as follows:
- STABILIZED = Y
- PER_STMT_ID = nnn
- STBLGRP (with the STBLGRP name of -START DYNQUERYCAPTURE command)

If a stabilized entry is fetched from the DB2 catalog, the following fields of sqlid.DSN_STATEMENT_CACHE_TABLE (left side) are populated as follows:
- STABILIZED = Y
- PER_STMT_ID = SDQ_STMT_ID (from SYSIBM.SYSDYNQRY)
- STBLGRP = STBLGRP (from SYSIBM.SYSDYNQRY)

The previous fields can be used externally to identify movement from DSC to DPS tables and vice versa.
Be aware that the usual LRU algorithm for removal of statements from the DSC still applies. Stabilized queries can also be subject to removal and will be loaded from the DB2 catalog if they are prepared again.

An existing stabilized query is not replaced in SYSIBM.SYSDYNQRY if the same statement is stabilized again, which can happen in a data sharing environment or when the DSNZPARM CACHEDYN_STABILIZATION is set to CAPTURE. This situation is true even if the access path is different.
6 Special considerations

6.1 Additional PREPARE keywords

6.1.1 Statement concentration

If dynamic SQL statements are used with literals, and the literals change from one statement execution to another, the cache is not used very efficiently. Each statement is different, which means that each statement needs a full prepare and a separate entry in the DSC. If the access path is always the same, it makes sense to store the statement only once and reuse it. A common best practice is to use parameter markers instead of literals for best DSC efficiency, but sometimes applications run as “black box”. They are bought from third parties or vendors and cannot be changed easily. Using statement concentration can mitigate that problem. The appropriate parameter settings are:

- In SQL PREPARE statement, the ATTRIBUTES setting of CONCENTRATE STATEMENTS WITH LITERALS. Consider that ATTRIBUTES options must be in a host variable.
- In JDBC on connection level, setDBStatementConcentrator(2)
- In ODBC initialization file, LITERALREPLACEMENT=1
- Beginning with DB2 12, CONCENTRATESTMT can be specified as a BIND option

The following figures show an example without statement concentration and an example with statement concentration:

```
User-A
SET CURRENT SQUALID = 'DB2';
SELECT * FROM SYSIBM.SYSTABLESPACE
WHERE PARTITIONS = 00002
AND DNAME <> 'DSNDB06'
WITH UR ;

User-B
SET CURRENT SQUALID = 'DB2';
SELECT * FROM SYSIBM.SYSTABLESPACE
WHERE PARTITIONS = 00003
AND DNAME <> 'DSNDB01'
WITH UR ;
```

```
<table>
<thead>
<tr>
<th>STMT_ID</th>
<th>PRIMAUTH</th>
<th>CURSQUALID</th>
<th>STMT_TEXT</th>
<th>STMT_TEXT</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>User-A</td>
<td>DB2</td>
<td>SELECT * FROM SYSIBM.SYSTABLESPACE WHERE PARTITIONS = 00002 AND...</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>User-B</td>
<td>DB2</td>
<td>SELECT * FROM SYSIBM.SYSTABLESPACE WHERE PARTITIONS = 00003 AND...</td>
<td></td>
</tr>
</tbody>
</table>
```

Dynamic Statement Cache

Figure 16: Without statement concentration

With statement concentration, the appropriate literal is replaced by the “&” sign:
After the statement is concentrated, it is searched as usual in the cache. If the statement is not found, it is inserted; otherwise, it is reused. Consider the following situation: if DB2 determines that the statement with the new literal instance cannot share the cached statement because of incompatible literal reusability criteria, DB2 inserts a new statement into the cache that has both '&' substitution and a different set of literal reusability criteria. This new statement is different from the cached statement, even though both statements have the same statement text with ampersand characters ('&'). Now, both statements are in the cache, but each has different literal reusability criteria that makes these two cached statements unique.

Consider also that the variable sign & cannot be used in normal predicates, as shown in the following example:

```sql
SELECT * FROM SYSIBM.SYSTABLESPACE WHERE PARTITIONS = &
AND DBNAME <> '&'
WITH UR
```

If the variable sign & is used in string literals, the statement is concentrated as expected:

```sql
SELECT * FROM SYSIBM.SYSTABLESPACE WHERE PARTITIONS = 00002
AND DBNAME <> '&'
WITH UR
```

is cached as

```sql
SELECT * FROM SYSIBM.SYSTABLESPACE WHERE PARTITIONS = 
AND DBNAME <> &
WITH UR
```

Statements with a mix of literals and parameter markers are not subject for literal replacement.

Figure 17: With statement concentration
Note: If columns have a field procedure on, the statement concentration has the following limitations:

- If the same user executes the same INSERT statements with literals, it is cached with the substitution character “&” only once.
- If the same user executes a different INSERT statements with literals, but the value for the field proc column does not change, it is cached with the substitution character “&” only once.
- If the same user executes a different INSERT statements with literals and the value for the field proc column changes: it is cached with the substitution character “&” for each different value.

The previous logic applies only if field procedures are used, but not for edit procedures or valid procedures. Statement concentration is helpful for JDBC or ODBC applications that cannot be changed easily (e.g., packaged applications from a third party vendor) because it is possible to activate statement concentration at the data source level – no application changes are required.

Nonetheless, you should keep the following considerations in mind:

- Whenever possible and appropriate, application developers should work with parameter markers instead of literals. Replacing literals with “&” and then a subsequent lookup in the cache results in more effort for DB2 in the preparation process, compared to working with parameter markers directly.
- For some situations, statement concentration might not be the best choice, for example, when data is skewed and you want DB2 to prefer a certain access path. Consider a table that contains a column ORDER_STATUS, which stores the status of an order. If 95% of your orders have ORDER_STATUS = ‘COMPLETED’ but your query is looking for all orders with status ‘IN_PROGRESS’, the DB2 optimizer will probably generate a better performing access path if the query asks for ORDER_STATUS='IN_PROGRESS’ instead of ORDER_STATUS=&.

### 6.1.2 Extended indicators

Another technique that increases efficiency of the DSC is the usage of extended indicators (available with DB2 10). This function is either activated by using the PREPARE attribute WITH EXTENDED INDICATORS or by the BIND option EXTENDEDINDICATOR(YES). In context of dynamic statement cache, only the PREPARE attribute is important. It enables the programmer to code the complete INSERT, UPDATE, or MERGE statement and decide at runtime which columns should be used. This can be done by using the value of -7 in the indicator variable.

The definitions for the indicator variables are:

- 0 (zero), or positive value specifies that the associated, first host-identifier provides the value of this host variable reference.
- -1, -2, -3, -4, or -6 specifies the null value.
- -5 specifies that the target column for this host variable is to be set to its default value.
- -7 specifies that the target column for this host variable is to be treated as if it had not been specified in the statement.

For Java programs, the IBM Data Server Driver for JDBC and SQLJ provides the method setDBUnassigned for DB2PreparedStatement objects. In the following example, it is assumed that a user can modify several values of a person record. The application allows updating city, zip code, street, bank account, phone number, mobile phone number, and so on. If the SQL statement contained only the columns that are updated, several variations of the statement would be found in the DSC:
By using extended indicators, you can ensure that only one version of the update statement is stored in the cache.

Figure 18: Caching without extended indicators

Note that this technique is not applicable to SELECT statements and CURSOR declarations.
6.2 Re-optimization of statements

If dynamic SQL statements are used with host variables or parameter markers, you can adjust the access path during runtime by using the REOPT (RE-)BIND parameter. This approach is useful when the input of the host-variables or parameter markers changes.

The default (REOPT(NONE)) prevents re-optimization at runtime. The access paths that were determined with the full prepare of the statement remain stable as long as this statement is kept in the cache. The following re-optimizations are possible:

- REOPT(ONCE): Access path re-optimization with the first execution of the statement after caching.
- REOPT(AUTO): Check for access path optimization with every execution of the statement based on the current values for parameter markers (host variables). If the current values provide a better access path, the old access path will be removed from the cache and the new path will be inserted. **Note:** This also resets the execution statistics for the statement and creates a new statement ID.
- REOPT(ALWAYS): Creates a new access path with every execution of the statement based on the values for parameter markers or host variables. This access path is unique for every execution so there is no caching in the DSC.

The following table summarizes the effect of the bind options for the usage of the DSC:

<table>
<thead>
<tr>
<th></th>
<th>REOPT(NONE)</th>
<th>REOPT(ONCE)</th>
<th>REOPT(AUTO)</th>
<th>REOPT(ALWAYS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PREPARE</td>
<td>STMT cached</td>
<td>not executed</td>
<td>not executed</td>
<td>not executed</td>
</tr>
<tr>
<td>EXECUTE</td>
<td></td>
<td>STMT cached</td>
<td>STMT cached</td>
<td>STMT not cached</td>
</tr>
</tbody>
</table>

**Table 3: REOPT parameters**

The statement "not executed" means that, although the code runs over the PREPARE statement, it is quasi bypassed (e.g., the appropriate accounting or statistics counters are not incremented). The counters are incremented if the statement is actually executed, which is normally the case.

Also consider the following counters, which describe the caching. The example runs over 10 PREPARES of the same statement:

<table>
<thead>
<tr>
<th>DYNAMIC SQL STMT</th>
<th>QUANTITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>PREPARE REQUESTS</td>
<td>10.00</td>
</tr>
<tr>
<td>FULL PREPARES</td>
<td>1.00</td>
</tr>
<tr>
<td>SHORT PREPARES</td>
<td>9.00</td>
</tr>
<tr>
<td>SHORT PREPARES</td>
<td>9.00</td>
</tr>
<tr>
<td>BASED ON CACHE</td>
<td>9.00</td>
</tr>
<tr>
<td>BASED ON CATALOG</td>
<td>0.00</td>
</tr>
<tr>
<td>LOOK-UP IN CATALOG</td>
<td>1.00</td>
</tr>
<tr>
<td>CACHE HIT RATIO (%)</td>
<td>90.00</td>
</tr>
<tr>
<td>CACHE+CATALOG HIT RATIO (%)</td>
<td>90.00</td>
</tr>
<tr>
<td>TOTAL PREPARES</td>
<td>10.00</td>
</tr>
<tr>
<td>EXPlicit PREPARES</td>
<td>10.00</td>
</tr>
<tr>
<td>Implicit PREPARES</td>
<td>0.00</td>
</tr>
</tbody>
</table>
In the case of REOPT(ALWAYS), none of these counters are maintained. For the cases of REOPT(ONCE) and REOPT(AUTO), the following counters are maintained:

<table>
<thead>
<tr>
<th>DYNAMIC SQL STMT</th>
<th>AVERAGE</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>REOPTIMIZATION</td>
<td>1.00</td>
<td>1</td>
</tr>
<tr>
<td>NOT FOUND IN CACHE</td>
<td>1.00</td>
<td>1</td>
</tr>
<tr>
<td>FOUND IN CACHE</td>
<td>9.00</td>
<td>9</td>
</tr>
</tbody>
</table>

For more information, see the Knowledge Center topic “Including dynamic SQL in your program”.

6.3 Keep dynamic statements

An option to bypass the PREPARE statement after COMMIT is to use KEEPDYNAMIC(YES) during package bind. Using this option allows DB2 to keep prepared statements in the local thread cache after COMMIT. The total number of statements in the complete system is controlled by DSNZPARM MAXKEEPD. Also consider the accounting counters (IMPLICIT PREPARE or CACHE_LIMIT_EXCEEDED) in the following figure.

This figure shows the impact of the KEEPDYNAMIC parameter. This example assumes that the same statement is always addressed in the PREPARE or EXECUTE statement:

<table>
<thead>
<tr>
<th>KEEPDYNAMIC=NO</th>
<th>BIND PACKAGE</th>
<th>KEEPDYNAMIC=YES</th>
</tr>
</thead>
<tbody>
<tr>
<td>PREPARE ...</td>
<td>PREPARE ...</td>
<td>PREPARE ...</td>
</tr>
<tr>
<td>EXECUTE ...</td>
<td>EXECUTE ...</td>
<td>EXECUTE ...</td>
</tr>
<tr>
<td>COMMIT ...</td>
<td>COMMIT ...</td>
<td>COMMIT ...</td>
</tr>
<tr>
<td>PREPARE ...</td>
<td>EXECUTE ...</td>
<td>EXECUTE ...</td>
</tr>
<tr>
<td>EXECUTE ...</td>
<td>SQLCODE-518</td>
<td></td>
</tr>
</tbody>
</table>

Figure 20: KEEPDYNAMIC bind option

Before DB2 12, applications bound with KEEPDYNAMIC(YES) needed to pay attention for ROLLBACK processing. ROLLBACK used to invalidate the statement in the local cache. So it was necessary to issue a PREPARE request before the next statement execution. This behaviour was changed in DB2 12. Now statements are kept in the cache even if a ROLLBACK occurs.

KEEPDYNAMIC(YES) and REOPT(ALWAYS) are mutually exclusive. However, KEEPDYNAMIC(YES) can be used with REOPT(ONCE) or REOPT(AUTO).
6.4  DB2 Analytics Accelerator and dynamic statement caching

6.4.1  Short introduction

DB2 Analytics Accelerator for z/OS based on IBM Netezza technology was introduced in 2011. It is a hardware appliance with a dedicated connection to one or several DB2 for z/OS subsystems, and is designed to run complex and/or analytical queries much quicker than DB2 for z/OS native processing.

DB2 Analytics Accelerator allows acceleration of static as well as dynamic queries. Users can influence the routing of queries to the accelerator with BIND parameter QUERYACCELERATION (for static queries) or special register CURRENT QUERY ACCELERATION. In cases where this parameter or special register is not set explicitly, the setting of ZPARM QUERY_ACCELERATION determines DB2's behaviour for query routing.

The following values can be specified:

- NONE = Queries are not routed to the accelerator.
- ENABLE = The DB2 optimizer checks whether the formal prerequisites for accelerator usage are met and performs a cost calculation. If the expected cost of running the query in DB2 natively is higher than the cost of query routing, the query will be routed to the accelerator.
- ENABLE WITH FAILBACK = Same as ENABLE with the exception that queries are sent back to DB2 when an error occurs in the preparation phase of the query on the accelerator.
- ELIGIBLE = DB2 will route the query to the accelerator without considering the cost of query execution in DB2 versus the cost of routing to the accelerator. Only format prerequisites need to be met.
- ALL = DB2 will always try to route the query to accelerator regardless of whether formal prerequisites are met or whether cost savings can be expected.

When DB2 Analytics Accelerator was originally introduced, the focus was on dynamic SQL acceleration because there was no support for dynamic statement caching and only very limited support for measurement of execution metrics. Monitoring products such as OMEGAMON XE Performance Expert for DB2 can provide overall information on accelerator usage, which can be used for capacity planning and cost chargeback. Nonetheless, it was quite difficult to measure individual queries.

6.4.2  Dynamic Statement Caching for accelerated queries

DB2 10 and DB2 11 delivered PTFs in their maintenance stream to enable caching of accelerated queries in the global DSC. The following APARs / PTFs are required:

- PI22531 (UI24539) : Enablement of DSC for DB2 10
- PI28660 (UI28976) : Enablement of DSC for DB2 11
- PI23083 (UI24713) : Collection of statement level metrics for accelerated queries in DB2 10 (via IFCID 316 and 401)
- PI30005 (UI29032) : Same for DB2 11

DB2 Analytics Accelerator Version 4.1 or later is also required.

With the introduction of caching of accelerated queries, eleven new columns have been added to DSN_STATEMENT_CACHE_TABLE. Eight columns (with names that start with STAT_ACC…) contain metrics for the execution of queries on an accelerator. The ACCELERATOR_NAME column indicates on which accelerator this query was executed (or contains the name of a virtual accelerator if the user performs accelerator modelling). The ACCEL_OFFLOAD_ELIGIBLE column is also used for accelerator modelling.

The ACCELERATED column is most relevant for caching of accelerated queries in the DSC. It can contain three different values: "NO", "YES" or "NEVER". The following table indicates their meaning.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO</td>
<td>Queries are not cached.</td>
</tr>
<tr>
<td>YES</td>
<td>Queries are cached.</td>
</tr>
<tr>
<td>NEVER</td>
<td>Queries are never cached.</td>
</tr>
</tbody>
</table>
6.4.3 DSC checking for accelerator queries

Users of DB2 Analytics Accelerator might see that there are up to three entries in the DSC for the same query in parallel – differentiated in their value for column ACCELERATED. All of these entries can be reused by DB2 and can accumulate execution statistics. As long as the setting of special register CURRENT QUERY ACCELERATION allows (or tries to enforce) the routing of a query to an accelerator, DB2 will consider entries with ACCELERATED=YES or NEVER in the prepare process for the statement.

As soon as an existing cache entry can be reused by DB2, the execution statistics in the DSC are also updated after successful query execution – whether the query was executed on DB2 natively or on the accelerator.

There are special cases when statistics in the DSC are not updated. For example, when query routing to the accelerator is allowed (or enforced) and all routing criteria are met, but at least one of the involved tables is not defined on the accelerator, or the query contains SQL syntax that is not supported for query acceleration. This value is also set if query acceleration is optional (special register setting ENABLE or ENABLE WITH FAILBACK), but the optimizer regards query routing to an accelerator as not beneficial.

### Rules for statement matching

The basic rules for whether DB2 can reuse an existing cache entry and therefore avoid a full prepare apply to environments with a DB2 Analytics Accelerator as well. This means that the statement text, CURRENT SQLID, CURRENT PATH, CURRENT DEGREE, or package name must match an existing statement in the DSC. For queries that might be routed to an accelerator, the value of the ACCELERATED column for existing matching records must be considered.

### Special register setting ENABLE or ENABLE WITH FAILBACK

Before statement caching for the DB2 Analytics Accelerator was introduced, every statement that executed with CURRENT QUERY ACCELERATION = ENABLE (or ENABLE WITH FAILBACK) had to pass a complete prepare process in DB2. This was needed for the optimizer to calculate whether the query routing to an accelerator would be beneficial.

---

### ACCELERATED in DSN_STATEMENT_CACHE_TABLE

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
</table>
| NO     | When this statement has been inserted into DSC:  
  - Query acceleration was not requested (due to special register / BIND option / ZPARM setting) OR  
  - Query acceleration was allowed, but the query was not eligible for acceleration (e.g., at least one table was defined in the accelerator but was not enabled for acceleration). |
| YES    | This statement has been inserted into the DSC after there was a successful query execution on the accelerator. |
| NEVER  | When this statement was prepared, DB2 detected that the query can never be accelerated. This can happen when query acceleration was allowed but at least one of the referenced tables is not defined on the accelerator, or the query contains SQL syntax that is not supported for query acceleration. This value is also set if query acceleration is optional (special register setting ENABLE or ENABLE WITH FAILBACK), but the optimizer regards query routing to an accelerator as not beneficial. |
| ACCEL_ONLY | This statement has been inserted because it has processed an Accelerator Only-Table. By definition, these kinds of statements can run on an accelerator only. |
With statement caching, the cost for this preparation process can be significantly reduced. DB2 first checks whether a matching cache entry with ACCELERATED=NEVER exists. If so, DB2 will reuse that entry, avoid further cost checking, and run that query natively in DB2. If that entry does not exist, it will check for an entry with ACCELERATED=YES. If such an entry exists, DB2 will also avoid a full prepare and route that query to the accelerator. A full prepare will be necessary only if none of these entries exist in the DSC. Workloads with many queries running under CURRENT QUERY ACCELERATION=ENABLE or ENABLE WITH FAILBACK will benefit most from statement caching.

6.4.4 Adding to or removing tables from the accelerator

When tables are added to the accelerator, queries that are already cached in the DSC and flagged as non-eligible for acceleration (ACCELERATED=NEVER) can become eligible. In this case, the DB2 stored procedure ACCEL_ADD_TABLES (which is part of the DB2 Analytics Accelerator software) invalidates DSC entries that refer to the table being added.

On the other hand, removing a table from an accelerator does not invalidate any entries in the DSC. Entries with ACCELERATED=YES will also be kept in the DSC. If the query is executed with CURRENT QUERY ACCELERATION=ELIGIBLE and at least one of tables is no longer available in the accelerator, DB2 will execute the query natively and update the statistics for an existing entry with ACCELERATED=NO (or create a new entry in the DSC).

6.4.5 Accelerator-only tables

Accelerator-only tables (AOTs) were introduced with Version 4.1 PTF 5 of IBM DB2 Analytics Accelerator. They differ from other types of tables as they physically exist inside the accelerator only. DB2 keeps only a catalog entry for them.

SQL statements with AOTs must run on the accelerator. For example, inserts into an AOT must reference other tables that are available on the accelerator (in the SELECT part of an insert) or that can specify a value list. Similar rules apply to other types of statements. As a consequence, the DB2 optimizer has no choice when preparing a statement that references AOTs: the statement must be eligible for the accelerator, and it is executed on the accelerator.

Statements that reference AOTs (no matter what kind of statement) will be cached in DB2’s dynamic statement cache like other accelerated and non-accelerated statements. They have “ACCEL_ONLY” as value for column ACCELERATED and therefore can easily be distinguished from other statements in the DSC.

Entries with ACCELERATED=ACCEL_ONLY are reused by DB2. Although these entries are not relevant for DB2’s access paths (as the statements are executed on the accelerator), DB2 accumulates the execution statistics from these statements gathered from the accelerator.

If a cached statement refers to an AOT and at least one DB2 table that is shadowed on the accelerator, RUNSTATS of this shadowed table removes the entry from the DSC. The DSC entry will be kept after RUNSTATS only if all tables in the statement refer to AOTs (e.g., INSERT INTO AOT_1 SELECT … FROM AOT2).

Note: this document assumes standard RUNSTATS functionality; DB2 12 allows stabilizing statements in the DSC as well as avoidance of statement invalidation.

These entries for AOTs are removed from the DSC when the cache gets filled up and other statements replace them or when the AOT is removed from the accelerator (which means that the table is dropped).
## Appendixes

### 7.1 Recent APARs

The following table provides a list of important and available APARs associated with the processes described in this document. It is provided strictly for informational purposes and is not a substitute for a comprehensive maintenance strategy nor intended to be exhaustive. Use keyword SQLDYNSTMTCACHE for search in the appropriate IBM databases.

<table>
<thead>
<tr>
<th>#APAR</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PI28660</td>
<td>This new function APAR now extends Dynamic Statement Cache (DSC) support to dynamic queries that are accelerated to an accelerator.</td>
</tr>
<tr>
<td>PI30005</td>
<td>This new function APAR now enables Accelerator Statement Level Monitoring support in IFCID 316 and 401.</td>
</tr>
<tr>
<td>PI32291</td>
<td>RENAME TABLE ENHANCEMENT</td>
</tr>
<tr>
<td>PI33417</td>
<td>REOPT(ALWAYS) NOT WORKING WITH STATEMENT LEVEL REOPTIMIZATION Suggestion APAR</td>
</tr>
<tr>
<td>PI35224</td>
<td>IMPROVE RC00E7910D HANDLING</td>
</tr>
<tr>
<td>PI42350</td>
<td>-START PROFILE PREVENTS EXISTING STATEMENTS IN THE DSC FROM BEING (RE)USED</td>
</tr>
<tr>
<td>PI46386</td>
<td>A USER CANCEL DURING LONG CACHE PREPARE CAN RESULT IN ABEND04E WITH VARIOUS MODULES AND REASON CODES</td>
</tr>
<tr>
<td>PI48283</td>
<td>NO DYNAMIC STATEMENT CACHE HIT FOR SQL STATEMENTS USING TEMPORAL TABLES</td>
</tr>
<tr>
<td>PI49489</td>
<td>STORAGE LEAK IN QW0225DMH RESULTING IN SQLCODE904 RC00E7910D FOR SELECT FROM INSERT / UPDATE / DELETE</td>
</tr>
<tr>
<td>PI49581</td>
<td>EXTEND RANGE OF VALUES FOR SYSTEM PARAMETER DDLTOX FOR TIMEOUT RESOLUTION</td>
</tr>
<tr>
<td>PI50207</td>
<td>DROP ROLE CAUSES EXTRA DYNAMIC STATEMENT CACHE SEARCH</td>
</tr>
<tr>
<td>PI52619</td>
<td>DYNAMIC CACHE PERFORMANCE DEGRADATION WHEN USING DB2 SYSTEM PARM GET_ACCEL_ARCHIVE = YES + ‘CONCENTRATE STATEMENTS WITH LITERALS’</td>
</tr>
<tr>
<td>PI55444</td>
<td>ABEND04E RC00C90110 IN DSNISRTI ERQUAL5016 DURING INSERT OF JSON DATA</td>
</tr>
<tr>
<td>PI57200</td>
<td>SQLCA.SQLERRD3 FILLED UNEXPECTED FOR IDAA CASE OF ENABLE WITH FAILBACK AND QUERY FAILS BACK</td>
</tr>
<tr>
<td>APAR Code</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>PI58436</td>
<td>INCORROUT WITH SQLPARALLELISM AND CONCENTRATE STATEMENTS WITH LITERALS CLAUSE.</td>
</tr>
<tr>
<td>PI63576</td>
<td>ABEND0C4 RC10 DSNXEDSC +273A FOR QUERY ACCELERATION ENABLE + ARCHIVE=YES USING A VIRTUAL ACCELERATOR &amp; DYN STMT CACHING</td>
</tr>
<tr>
<td>PI64733</td>
<td>SQLCODE84 DURING PREPARE OF A STATEMENT WHICH USES CONCENTRATE WITH LITERALS AND STATEMENT GETS INVALIDATED</td>
</tr>
<tr>
<td>PI66197</td>
<td>MSGDSNU070I DSNUGDFL - KEYWORD OR OPERAND 'INVALIDATECACHE NO' INVALID WITH 'REPORT NO UPDATE NONE'</td>
</tr>
<tr>
<td>PI67477</td>
<td>SQLCODE401 FROM DSNXOEXD WHEN AN SQL STATEMENT HAS AN ARITHMETIC EXPRESSION AND CONCENTRATE STATEMENT WITH LITERALS PREPARE ATTR</td>
</tr>
<tr>
<td>PI67545</td>
<td>SQL IS RE-PREPARED AFTER ROLLBACK WHEN KEEPDYNAMIC=1</td>
</tr>
<tr>
<td>PI67915</td>
<td>INCORRECT DSN_STATEMENT_CACHE_TABLE.GROUP_MEMBER WHEN DYNAMIC PLAN STABILITY IS USED</td>
</tr>
<tr>
<td>PI68144</td>
<td>ABEND0C4 RC10 DSNXEDSC +273A FOR QUERY ACCELERATION ENABLE + ARCHIVE=YES USING A VIRTUAL ACCELERATOR &amp; DYN S</td>
</tr>
<tr>
<td>PI68393</td>
<td>DROP INDEX CAUSES STABILIZED QUERIES ON TABLE TO BE INVALIDATED</td>
</tr>
<tr>
<td>PI68875</td>
<td>INCORROUT with CONCENTRATE STATEMENTS WITH LITERALS</td>
</tr>
<tr>
<td>PI69844</td>
<td>CREATE INDEX INCORRECTLY SKIPS QUIESCING OF CACHED DYNAMIC STATEMENTS THAT ARE DEPENDENT ON THE UNDERLYING TABLE</td>
</tr>
<tr>
<td>PI70165</td>
<td>SQLCODE904 ISSUED FROM DSNXIDMH WITH RC00E70081 FOR AN ALTER ADD DDL WHEN THERE IS A DEADLOCK BETWEEN DDL AND DML</td>
</tr>
<tr>
<td>PI71488</td>
<td>SQLCODE84 DURING PREPARE OF A STATEMENT WHICH USES CONCENTRATE WITH LITERALS AND STATEMENT GETS INVALIDATED</td>
</tr>
<tr>
<td>PI71495</td>
<td>ABEND04E RC00E70005 AT DSNXGRDS DSNXOB2 M105 ON SQL STATEMENT WITH CONCENTRATE STATEMENTS WITH LITERALS</td>
</tr>
</tbody>
</table>

Table 4: Recent APARs
7.2 DSNZPARM settings
The following table summarizes the V12 parameters that influence the DSC and its usage.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Values</th>
<th>Default</th>
<th>Recommended setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>CACHEDYN</td>
<td>YES NO</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>CACHEDYN_STABILIZATION</td>
<td>CAPTURE LOAD BOTH NONE</td>
<td>BOTH</td>
<td>BOTH</td>
</tr>
<tr>
<td>EDMSTMTTC</td>
<td>5000-4194304 (in K)</td>
<td>113386 (in K)</td>
<td>Dependent on the hit ratio in the cache</td>
</tr>
<tr>
<td>MAXKEEPD</td>
<td>0-204800</td>
<td>5000</td>
<td>Dependent on the usage of bind option KEEPDYNAMIC(YES)</td>
</tr>
</tbody>
</table>

Table 5: DSNZPARM settings
Consider the information APAR II14811 for tips on the DSNZPARM CACHE_DEP_TRACK_STOR_LIM. This DSNZPARM is no longer supported in V12.

7.3 Performance numbers
The following tests were executed on a LPAR that runs under z/VM. Therefore the numbers are not accurate but show at least the performance benefit of the dynamic statement cache.

1000 equal statements are prepared with CONCENTRATE STATEMENT WITH LITERALS
**Without** dynamic statement cache:  

<table>
<thead>
<tr>
<th></th>
<th>APPL(CL.1)</th>
<th>DB2 (CL.2)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ELAPSED TIME</strong></td>
<td>2.114564</td>
<td><strong>2.109352</strong></td>
</tr>
<tr>
<td><strong>NONNESTED</strong></td>
<td>2.114564</td>
<td><strong>2.109352</strong></td>
</tr>
<tr>
<td><strong>CP CPU TIME</strong></td>
<td>1.641826</td>
<td><strong>1.638438</strong></td>
</tr>
<tr>
<td><strong>AGENT</strong></td>
<td>1.641826</td>
<td><strong>1.638438</strong></td>
</tr>
<tr>
<td><strong>NONNESTED</strong></td>
<td>1.641826</td>
<td><strong>1.638438</strong></td>
</tr>
</tbody>
</table>

**DYNAMIC SQL STMT**  

<table>
<thead>
<tr>
<th></th>
<th>AVERAGE</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>REOPTIMIZATION</strong></td>
<td>0.00</td>
<td>0</td>
</tr>
<tr>
<td><strong>NOT FOUND IN CACHE</strong></td>
<td>0.00</td>
<td>0</td>
</tr>
<tr>
<td><strong>FOUND IN CACHE</strong></td>
<td>0.00</td>
<td>0</td>
</tr>
<tr>
<td><strong>IMPLICIT PREPARES</strong></td>
<td>0.00</td>
<td>0</td>
</tr>
<tr>
<td><strong>PREPARES AVOIDED</strong></td>
<td>0.00</td>
<td>0</td>
</tr>
<tr>
<td><strong>CACHE_LIMIT_EXCEEDED</strong></td>
<td>0.00</td>
<td>0</td>
</tr>
<tr>
<td><strong>PREP_STMT_PURGED</strong></td>
<td>0.00</td>
<td>0</td>
</tr>
<tr>
<td><strong>CSWL - STMTS PARSED</strong></td>
<td>0.00</td>
<td>0</td>
</tr>
<tr>
<td><strong>CSWL - LITS REPLACED</strong></td>
<td>0.00</td>
<td>0</td>
</tr>
<tr>
<td><strong>CSWL - MATCHES FOUND</strong></td>
<td>0.00</td>
<td>0</td>
</tr>
<tr>
<td><strong>CSWL - DUPLS CREATED</strong></td>
<td>0.00</td>
<td>0</td>
</tr>
</tbody>
</table>

**SQL DML**  

<table>
<thead>
<tr>
<th></th>
<th>AVERAGE</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PREPARE</strong></td>
<td>1000.00</td>
<td>1000</td>
</tr>
</tbody>
</table>

---

**With** dynamic statement cache:  

<table>
<thead>
<tr>
<th></th>
<th>APPL(CL.1)</th>
<th>DB2 (CL.2)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ELAPSED TIME</strong></td>
<td>0.698583</td>
<td><strong>0.695075</strong></td>
</tr>
<tr>
<td><strong>NONNESTED</strong></td>
<td>0.698583</td>
<td><strong>0.695075</strong></td>
</tr>
<tr>
<td><strong>CP CPU TIME</strong></td>
<td>0.240693</td>
<td><strong>0.238901</strong></td>
</tr>
<tr>
<td><strong>AGENT</strong></td>
<td>0.240693</td>
<td><strong>0.238901</strong></td>
</tr>
<tr>
<td><strong>NONNESTED</strong></td>
<td>0.240693</td>
<td><strong>0.238901</strong></td>
</tr>
</tbody>
</table>

**DYNAMIC SQL STMT**  

<table>
<thead>
<tr>
<th></th>
<th>AVERAGE</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>REOPTIMIZATION</strong></td>
<td>0.00</td>
<td>0</td>
</tr>
<tr>
<td><strong>NOT FOUND IN CACHE</strong></td>
<td>0.00</td>
<td>0</td>
</tr>
<tr>
<td><strong>FOUND IN CACHE</strong></td>
<td>999.00</td>
<td>999</td>
</tr>
<tr>
<td><strong>IMPLICIT PREPARES</strong></td>
<td>0.00</td>
<td>0</td>
</tr>
<tr>
<td><strong>PREPARES AVOIDED</strong></td>
<td>0.00</td>
<td>0</td>
</tr>
<tr>
<td><strong>CACHE_LIMIT_EXCEEDED</strong></td>
<td>0.00</td>
<td>0</td>
</tr>
<tr>
<td><strong>PREP_STMT_PURGED</strong></td>
<td>0.00</td>
<td>0</td>
</tr>
<tr>
<td><strong>CSWL - STMTS PARSED</strong></td>
<td>1000.00</td>
<td>1000</td>
</tr>
<tr>
<td><strong>CSWL - LITS REPLACED</strong></td>
<td>1000.00</td>
<td>1000</td>
</tr>
<tr>
<td><strong>CSWL - MATCHES FOUND</strong></td>
<td>999.00</td>
<td>999</td>
</tr>
<tr>
<td><strong>CSWL - DUPLS CREATED</strong></td>
<td>0.00</td>
<td>0</td>
</tr>
</tbody>
</table>

**SQL DML**  

<table>
<thead>
<tr>
<th></th>
<th>AVERAGE</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PREPARE</strong></td>
<td>1000.00</td>
<td>1000</td>
</tr>
</tbody>
</table>
8 References

8.1 Authors
This document was written by: Chris Crone, Christoph Theisen, Eric Radzinski, Frank Bower, Laura Kunioka-Weis, Long Tu, Manogari Simanjuntak, Patrick Bossman, Paul McWilliams, Peter Hartmann, Peter Kuang, Regina Liu, and Tammie Dang
Send any comments or corrections to Christoph Theisen (ctheisen@de.ibm.com) and Peter Hartmann (peterhar@de.ibm.com).

Company: IBM
Status: 06th of March 2017

8.2 Documentation
- GC27-8847: DB2 12 for z/OS Codes
- GC19-8851: DB2 12 for z/OS Installation and Migration Guide
- GC27-8855: DB2 12 for z/OS Messages
- SC27-8859: DB2 12 for z/OS SQL Reference
- SC27-8860: DB2 12 for z/OS Utility Guide and Reference
- SC27-8848: DB2 12 for z/OS Command Reference
- SC27-8857: DB2 12 for z/OS Managing Performance
- SH12-7061: IBM DB2 Analytics Accelerator for z/OS V5.1 User’s Guide

8.3 List of Tables
Table 1: SET CURRENT EXPLAIN MODE register................................................................. 19
Table 2: Comparison DSC entry versus stabilized DSC entry.................................................. 23
Table 3: REOPT parameters.................................................................................................. 29
Table 5: Recent APARs........................................................................................................ 35
Table 6: DSNZPARM settings.............................................................................................. 36

8.4 List of Figures
Figure 1: Equal statements - 1 .............................................................................................. 5
Figure 2: Equal statements - 2 .............................................................................................. 6
Figure 3: Different user IDs .................................................................................................. 7
Figure 4: Current SQLID ...................................................................................................... 7
Figure 5: Current path .......................................................................................................... 8
Figure 6: Special registers .................................................................................................... 9
Figure 7: Cursor with HOLD ............................................................................................... 9
Figure 8: Isolation levels ..................................................................................................... 10
Figure 9: BIF_COMPATIBILITY ......................................................................................... 11
Figure 10: APPLCOMPAT change ..................................................................................... 11
Figure 11: Invalidation by CREATE INDEX......................................................................... 13
Figure 12: Invalidation by ALTER INDEX.......................................................................... 14
Figure 13: Invalidation by REVOKE ................................................................. 16
Figure 14: Out-dated access path in the cache ................................................. 18
Figure 15: Dynamic statement hierarchy ......................................................... 20
Figure 16: Without statement concentration .................................................. 25
Figure 17: With statement concentration ....................................................... 26
Figure 18: Caching without extended indicators .............................................. 28
Figure 19: Caching with extended indicators .................................................. 28
Figure 20: KEEP_DYNAMIC bind option ......................................................... 30