DB2 UDB for z/OS V8 Übersicht

Inhaltsverzeichnis

DB2 V8 mit zig Änderungen in SQL soll vor allem die Konsistenz der DB2 Failie sicherstellen. Einige Limits wurden aufgehoben: Nutzung des "64 bit memory", Nutzung konsistenter Längen von Tabellen- und Spüaltennamen, 2 MB grosse "SQL Statements", 4096 partitions und Verdreifachung des "log space".

Performance Verbesserungen liefern erhöhte Verfügbarkeit der DB2_umgebungen. Die Möglichkeit Datenbankänderungen ohne Anhalten von DB2 durchführen zu können erhöht die Verfügbarkeit der DB-Umgebung, z.B. Hinzufügen von "partitions". Verbesserungen in den Funktionen von Java, Konsistenz und Integration mit WebSphere machen z/OS zu einer besseren Plattform für Java. Erweiterungen zu "security" bis auf "row level" Ebene helfen dabei Anforderungen, die für "Web related applications" gestellt werden, zu erfüllen. Viele der Verbesserungen sind hifreich in Anwendfungen wie PeopleSoft, SAP und Siebel.

Hier erfahren Sie über die hauptsächlichen Verbesserungen, die im "New Function Mode" zur Verfügung gestellt werden, in den Bereichen:

- "Scalability"
- "Availability"
- SQL
- "e-business"
- Utilities
- Performance
- "Data sharing"
- Installation und Migration

1 Änderungen der DB2 Limits

- Virtual Storage 2 GB 2^{31} auf 2^{64}
- Table & view names von 18 auf 128 Zeichen
- Column names von 18 auf 30 Zeichen
- SQL Statement Länge von 32K auf 2 MB
- “Character Literals” von 255 auf 32704 Zeichen
- Tables in einem Join von 15 auf 225
- Anzahl Partitions in einem PTS von 254 auf 4096
- Anzahl von “Active logs” von 31 auf 93
- Anzahl “Archive logs“ von 1000 auf 10,000

2 Erweiterbarkeit

Mit V8, DB2 UDB for z/OS werden neue Perspektiven für "scalability" und Performance gesetzt. Hier die wichtigsten Verbesserungen zu diesem Thema:

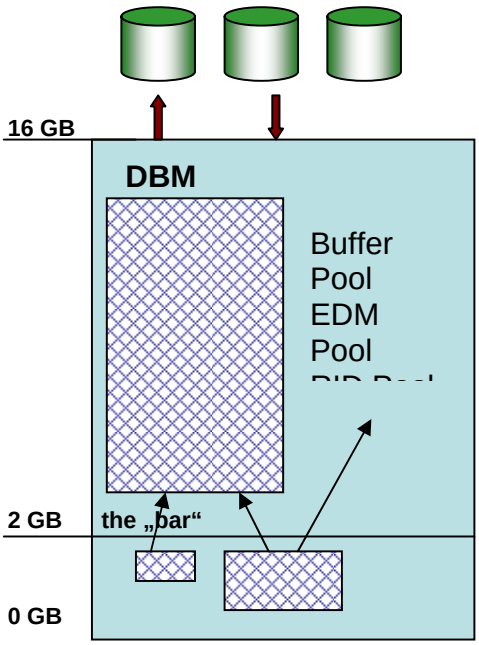
2.1 Erweiterung des "Virtual storage"

Die Nutzung der "zSeries 64-bit architecture" ermöglicht auch die Nutzung des "64-bit virtual storage".

Die "zSeries 64-bit architecture" gibt DB2 UDB for z/OS die Möglichkeit, diverse Speicherbereiche oberhalb der 2-GB Grenze anzulegen:

- Buffer pool
- EDM pool
- Sort pool
- RID pool
- "Compression dictionaries"

Ein einziger grosser "address space" von bis zu 2^{64} bytes (16 **exabytes**) ersetzt "hiper spaces" und "data spaces". Das ergebnis: Die Verwaltung des "virtual storage" wird vereinfacht und "scalability", "availability", und Performance verbessern sich gemäss den Anforderungen an "real storage" und dem Anwachsen der Benutzerzahlen.



2.2 Mehr partitions

Das Maximum der Anzahl Partitions in einem "partitioned table space" und "index space " wird von 254 auf 4096 erhöht.

Der DSSIZE-Wert bestimmt die maximale Anzahl der Partitions.

2.3 Mehr Tabellen beim Join

In V7 war die Anzahl Tabellen in der FROM Klausel eines SELECT Statement auf 225 bei einem "star join" beschränkt. Bei allen anderen Typen von JOIN's war das Limit 15 Tabellen. V8 erlaubt 225 Tabellen in allen JOIN-Typen.

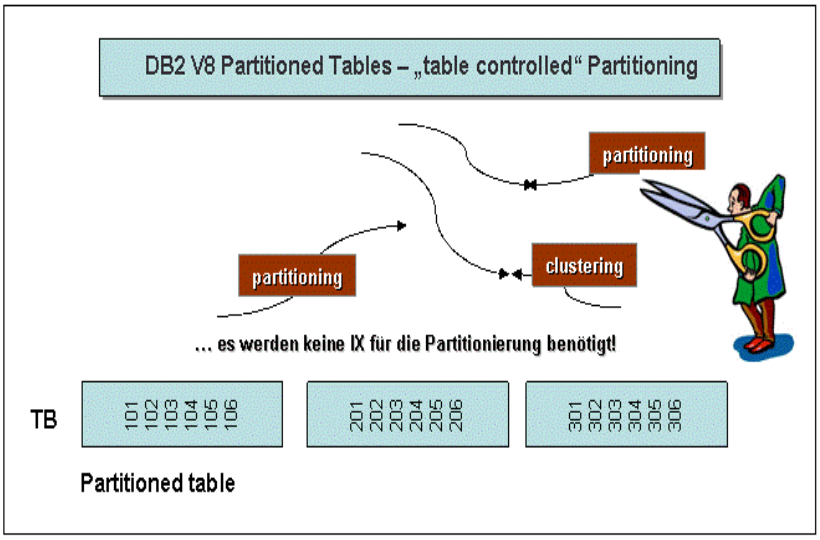
2.4 Mehr "log data sets"

Die maximale Anzahl von "active log data sets" pro "log copy" wurde von 31 auf 93 erhöht. Die maximale Anzahl von "archive log volumes", die im BSDS aufgezeichnet werden können, bevor ein "wrap around" passiert und der erste Eintrag überschrieben wird wurde von 1,000 auf 10,000 pro "log copy" erhöht.

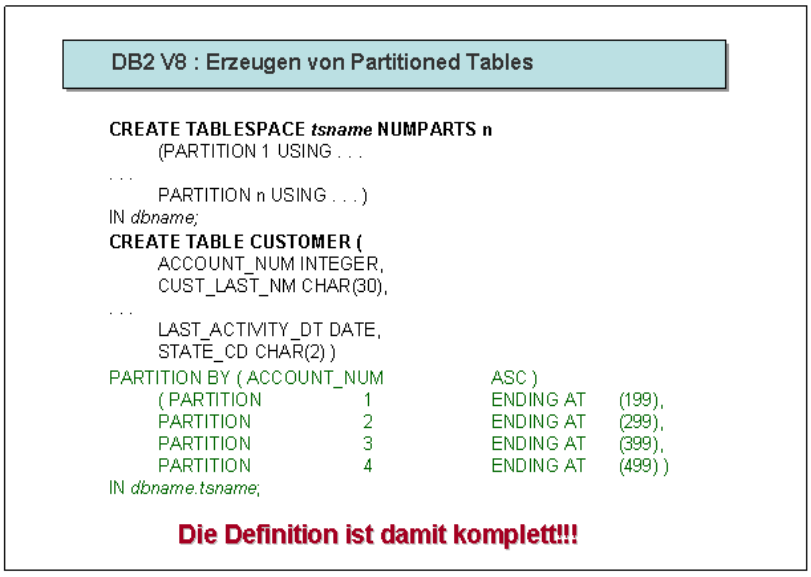
3 Verfügbarkeit

3.1 *"Partitioned secondary indexes"*

V8 führt "data partitioned secondary indexes" ein, um die Datenverfügbarkeit während der "partition level utility" Operationen zu erhöhen, z.B. REORG PART, LOAD PART, RECOVER PART.
Zudem wurden zusätzliche Funktionen auf "partition level" entwickelt: roll on/off part, rotate part. Diese werden über die "Online Schema Evolution" eingeführt.



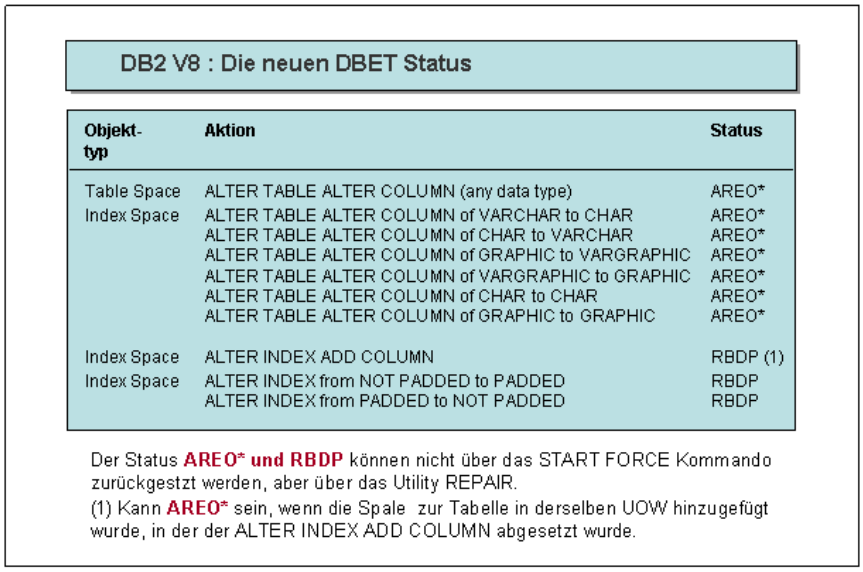
Die verbesserte Verfügbarkeit wird ergänzt über die sogenannten "partitioned secondary indexes" auf "partitioned tables", die entsprechend den unterliegenden daten ebenfalls partitioniert sein können. Dazu mgibt es weder eine BUILD2 Phase für REORG SHRLEVEL CHANGE, wenn alle "secondary indexes" partitioniert werden, noch gibt es Behinderungen bei LOAD PART Jobs, die auf unterschiedlichen Partitions eines TS durchgeführt werden. Aus Sicht von Queries gesehen ist ein "data partitioned secondary index" äusserst nützlich, wenn eine Query auf beiden IX-Typen Prädikate besitzt: den "secondary index column(s)" und den "partitioning index column(s)".



3.2 Online Schema Änderungen

Bestimmte Änderungen in DB2 Objekten können nun, ohne DB2 oder Teile davon anzuhalten, durchgeführt werden. Diese Objekte werden aber dann zunächst in einen eigenen Status versetzt, z.B. beim:

- Hinzufügen einer neuen "partition" zu einem bestehenden "partitioned table space" und bei "rotating partitions".
- Änderungen des "data type" von Spalten.



DB2 V8 : Die neuen DBET Status

Objekt-typ	Aktion	Status
Table Space	ALTER TABLE ALTER COLUMN (any data type)	AREO*
Index Space	ALTER TABLE ALTER COLUMN of VARCHAR to CHAR	AREO*
	ALTER TABLE ALTER COLUMN of CHAR to VARCHAR	AREO*
	ALTER TABLE ALTER COLUMN of GRAPHIC to VARGRAPHIC	AREO*
	ALTER TABLE ALTER COLUMN of VARGRAPHIC to GRAPHIC	AREO*
	ALTER TABLE ALTER COLUMN of CHAR to CHAR	AREO*
	ALTER TABLE ALTER COLUMN of GRAPHIC to GRAPHIC	AREO*
Index Space	ALTER INDEX ADD COLUMN	RBDP (1)
Index Space	ALTER INDEX from NOT PADDED to PADDED	RBDP
	ALTER INDEX from PADDED to NOT PADDED	RBDP

Der Status **AREO* und RBDP** können nicht über das START FORCE Kommando zurückgestzt werden, aber über das Utility REPAIR.
(1) Kann **AREO*** sein, wenn die Spale zur Tabelle in derselben UOW hinzugefügt wurde, in der der ALTER INDEX ADD COLUMN abgesetzt wurde.

In V5 konnte man die Grösse von "varchar columns" verändern.

DB2 V8 aber ermöglicht Änderungen, um "numeric" und/oder "character columns" zu vergrössern und Änderungen von "char" auf "varchar" und umgekehrt durchzuführen.

"Partitioning" und "clustering" waren Bestandteil in Versionen VOR DB2 V8.

Nun kann man "partitions" ohne eigenen (Extra-)Index definieren und Daten nach jedem beliebigen Index "clustern". Diese Änderungen können einen Index einsparen und die Zahl von "random I/O's" reduzieren.

3.3 “System level point-in-time recovery”

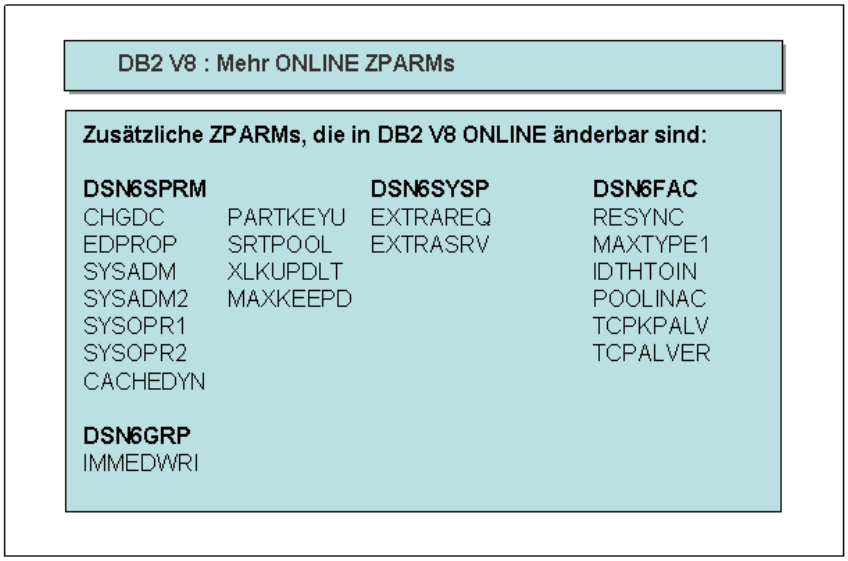
Das “system level point-in-time recovery” verbessert die Möglichkeiten das DB2 System auf jeden beliebigen Zeitpunkt in kürzester Zeit wiederherzustellen - unabhängig von “uncommitted units of work”. Diese Funktionalität wird ergänzt durch die Fähigkeit, ein Minimum an Objekten zu identifizieren, die in den „recovery process” mitaufgenommen warden müssen. Für grosse DB2 Systeme mit mehr als 30.000 Tabellen wird hiermit die “recovery time” erheblich verkürzt.

3.4 Online ZPARMs

In DB2 V7 war es bereits möglich bestimmte ZPARMs Online zu verändern, indem man das Kommando **SET SYSPARM** nutzte.

In DB2 V8 wurde diese Möglichkeit noch um einige ZPARMs erweitert.

Nicht „online“ geändert werden können z. B. DSNHDECP ZPARMs.



3.5 CI Size höher als 4 KB

DB2 V8 unterstützt CI Grössen von 8, 16, und 32 KB. Dies gilt für “user defined” und “DB2 defined” Tablespaces. Die neuen CI Grössen bedingen einige Einschränkungen bei “backup”, “concurrent copy”, und beim Einsatz von „striping“. Sie enthalten ebenfalls Potential zur Reduktion von „elapsed time“ bei „large table space scans“.

3.6 Überwachen des “system checkpoint” und Aufzeichnen der “offload activity“

In V8, überwacht den DB2 “system checkpoint” und die “log offload activity”. Es nutzt die “active log switch routine”, um feststellen zu können wann der “checkpoint process” oder die „offload task“ aufgebaut werden müssen.

- DSNJ016E csect-name WARNING -SYSTEM CHECKPOINT PROCESSOR MAY HAVE STALLED. LAST CHECKPOINT WAS TAKEN date-time
- DSNJ017E csect-name WARNING -OFFLOAD TASK HAS BEEN ACTIVE SINCE date-time AND MAY HAVE STALLED

3.7 Überwachen von langlaufenden "UR backouts"

In V7 ist es nicht möglich, festzustellen, wie lange das “backout” einer lang laufenden UR dauern könnte.

V8 liefert dazu alle 2 Minuten eine neue Prozess-Nachricht bis der „backout“ komplett ist.

3.8 Verbesserte LPL Recovery

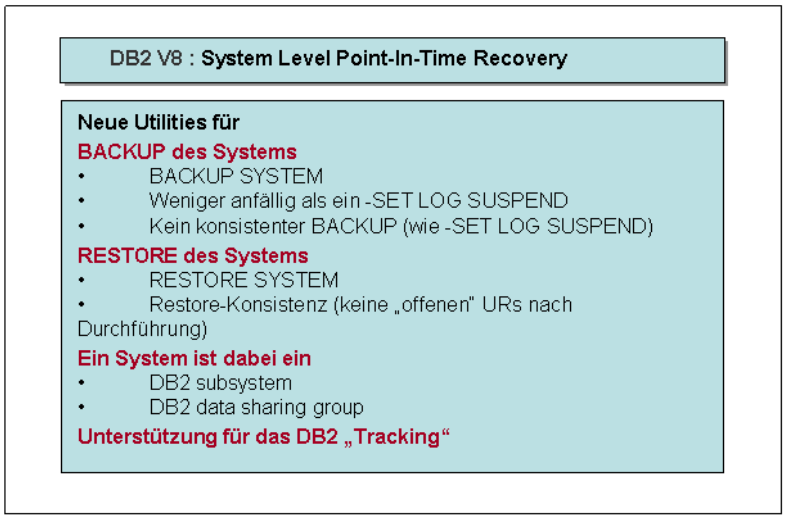
Dieses “feature” verbessert Verfügbarkeit, Performance, Nutzbarkeit und die Bedienbarkeit des DB2 LPL Recovery. Es geht speziell um folgende neuen Funktionen:

- **„Reason ID“ beim Einstellen einer „page“ in die LPL**

  Werden “pages” in den LPL gestellt, gibt DB2 die Meldung DSNB250E aus, um die „page range“ mitzuteilen, die „database“ und die „pageset/partition names“ der LPL „pages“. DB2 V8 gibt zusätzlich den Grund an, aus dem die „pages“ in den LPL gestellt wurden.

- **“Automatic recovery” von LPL “pages”**

  Nach dem Einstellen der “pages” in die LPL nimmt DB2 V8 an, dass das automatische „LPL Recovery“ initiiert werden muss. Wird das automatische „LPL Recovery“ erfolgreich abgeschlossen, werden die „pages“ wieder aus dem LPL gelöscht und DB2 gibt eine entsprechende Meldung - DSNI021I – aus. Schlägt das automatische „LPL Recovery“ fehl, meldet DB2 V8 DSNI005I um dies anzuzeigen.
  Eine neue Meldung, DSNB357I, wird gesetzt, um anzuzeigen, dass “pages” in die LPL gestellt wurden, dies aber nicht unbedingt zu einem automatischen „LPL Recovery“ führen muss.



- **Weniger fehlgeschlagene “LPL recovery” Aktionen**

  Wichtiger als das automatische „LPL Recovery“ ist, dass in DB2 V8 alle „pages“, die sich nicht im LPL befinden über SQL verarbeitet werden können. Bis zur DB2 V7 war die entsprechende TS oder “partition“ während des „LPL recovery“ nicht verfügbar.
  Der “LPL recovery processor” (beim -START DATABASE Kommando oder im neuen “automatic LPL recovery feature”) veranlasst lediglich einen “write claim” anstatt eines “drain” auf das Objekt das „recovered“ werden soll.
  Daraus resultiert dass alle „good pages” des Ovjekts den Benutzeren weiterhin zur Verfügung stehen und so die Performance verbessert wird, da ein „claim” weniger hinderlich als ein “drain” ist.

4 SQL

Hier die wichtigsten Verbesserungen im SQL Umfeld:

4.1 Long names

Die Änderungen an der Architektur von DB2 V8 befähigen den DB2 Katalog zur Unterstützung von sogen. “long names”: Ebenso zur Verwendung von längeren “string constants” (bis zu 32,704 Bytes), längerenr”index keys” (bis zu 2,000 Bytes), und längeren Prädikaten (bis zu 32,704 bBytes). Das trägt zur Kompatibilität von DB2 UDB for z/OS mit anderen Mitgliedern aus der DB2 Familie bei.

Item	Changed from	Changed to
Table check constraint name, correlation name, name of an alias, cursor (except for DECLARE CURSOR WITH RETURN), index, procedure, synonym, function, view, table	18 bytes	128 bytes
name of a column	18	30 bytes. Stored in DB2 catalog as VARCHAR(128), but limited to 30 Unicode bytes.
Name of cursor created with DECLARE CURSOR WITH RETURN	18 bytes	30 bytes
Name of a location	16 bytes	16 bytes. Stored in DB2 catalog as VARCHAR(128), but only 16 characters are used.
Name of package	8 bytes	8 bytes. Stored in DB2 catalog as VARCHAR(128), only 8 characters are used for packages created with BIND PACKAGE command, but can be 128 bytes long if created as a result of CREATE TRIGGER command.
Auxiliary table, collection, distinct type (both parts of two-part name), host identifier, JARs, parameter, sequence, specific, statement, savepoint, SQL parameter	18 bytes	128 bytes
SQL variable, SQL label, SQL condition	64 bytes	128 bytes
Maximum length of character constant	255 bytes	32704 UTF-8 bytes
Maximum length of hexadecimal constant	254 digits	32704 hexadecimal digits
Maximum length of graphic string constant	124 DBCS characters	32698 UTF-8 bytes

Item	Changed from	Changed to
Maximum length of hexadecimal graphic string constant	N/A	32704 hexadecimal digits
Maximum number of partitions in a partitioned table space or partitioned index	64 for table spaces that are not defined with LARGE or a DSSIZE greater than 2G 254 for table spaces that are defined with LARGE or a DSSIZE greater than 2G	64 for table spaces that are not defined with LARGE or a DSSIZE greater than 2G 4096 depending on what is specified for DSSIZE or LARGE and the page size.
Maximum size of a partition (table space or index)	For table spaces that are not defined with LARGE or a DSSIZE greater than 2G: 4 gigabytes, for 1 to 16 partitions 2 gigabytes, for 17 to 32 partitions 1 gigabyte, for 33 to 64 partitions For table spaces that are defined with LARGE: 4 gigabytes, for 1 to 254 partitions For table spaces that are defined with a DSSIZE greater than 2G: 64 gigabytes, for 1 to 254 partitions	For table spaces that are not defined with LARGE or a DSSIZE greater than 2G: 4 gigabytes, for 1 to 16 partitions 2 gigabytes, for 17 to 32 partitions 1 gigabyte, for 33 to 64 partitions For table spaces that are defined with LARGE: 4 gigabytes, for 1 to 4096 partitions For table spaces that are defined with a DSSIZE greater than 2G: 64 gigabytes, depending on the page size (1 to 256 partitions for 4KB, 1 to 512 partitions for 16KB, 1 to 1024 partitions for 32KB, and 1 to 2048 for 32KB)
Maximum length of an index key	255 bytes less the number of key columns that allow nulls	Partitioning index: 255 - n Non-partitioning index that is padded: 2000 - n Non-partitioning index that is not padded: 2000 - n -2m Where n is the number of columns in the key that allow nulls and m is the number of varying-length columns in the key
Longest SQL statement	32765 bytes	2097152 bytes
Maximum length of a sort key	4000 bytes	16000 bytes
Maximum length of the SQL path	254 bytes	2048 bytes
Largest table or table space	16 terabytes	128 terabytes

4.2 SQL Statements: 2 MB lang

Komplexe SQL Formulierungen, "SQL Procedures" und generierte SQL Statements, wie auch die geforderte Kopatibilität mit anderen Pattformen und konvertierte Objekte und Skripte von anderen Produkten erforderten die Erweiterungen der SQL Statements in DB2. DB2 V8 erweitert die maximale Grösse eines SQL Statements auf 2 Megabytes.

4.3 Enhanced scrollable cursors

Die Option DYNAMIC für "scrollable cursors" ermöglicht das Blättern direkt auf der DB2 Tabelle - ohne "temporary result table" (DTT).

Abhängig vom gewählten Zugriffspfad kann das "scrolling" über einen Index oder aber über "table space scan" erfolgen.

DB2 V8 : Dynamic Scrollable Cursors

„**Scrollable cursor**“, die den Zugriff auf die sogen. „base table" anstatt auf ein DTT ermöglichen... Das ermöglicht das Erkennen von ***UPDATE und/oder INSERTS***, die man selbst oder ein anderer User durchgeführt haben

Der „default" ist dabei ein „single row FETCH", sodass DDF Anwendungen folgende Techniken nutzen sollten:

- *Multi-row FETCH*
- *Positioned UPDATE/DELETE for multi-row FETCH*

```
DECLARE C1 SENSITIVE DYNAMIC SCROLL
    CURSOR FOR
    SELECT C1, C2
    FROM   T1;
```

4.4 Common table expression

DB2 V8 **führt "common table expression" und rekursives SQL** ein.

DB2 V8 : „common table expressions“

```
WITH
E AS
(
      SELECT EMPNO, LASTNAME, SALARY,
      SUBSTR(CHAR(HIREDATE,ISO),1,3) CONCAT '0 - 9'
         AS HIREDECADE
      FROM EMPLOYEE
),
M (HIREDECADE, MINIMUM_SALARY) AS
(
      SELECT HIREDECADE, MIN(SALARY)
      FROM E
      GROUP BY HIREDECADE
)
SELECT     E.EMPNO, E.LASTNAME, E.HIREDECADE,
           E.SALARY, M.MINIMUM_SALARY
FROM       E INNER JOIN M
           ON E.HIREDECADE = M.HIREDECADE
```

Dies ist ein übliches Vorgehen zur Fromulierung von komplexen Queries, da "common table expressions" anstelle von Views verwendet werden können.

Dies wiederum erspart sowohl dem User, als auch dem System die CREATE/DROP Aktivitäten auf Views und damit auf dem Katalog.

4.5 Rekursives SQL

Recursives SQL kann sehr nützlich sein, wenn Daten von hierarchisch angeordneten/abhängigen Tabellen wiedergelesen werden sollen.

DB2 V8 : „recursive SQL“

```
WITH
RPL (PART, SUBPART, QUANTITY) AS
(
```

Initialisierungs-SELECT

```
SELECT    ROOT.PART, ROOT.SUBPART, ROOT.QUANTITY
FROM      PARTLIST ROOT
WHERE     ROOT.PART = '01'
          UNION ALL
```

Iterations-SELECT

```
SELECT    CHILD.PART, CHILD.SUBPART, CHILD.QUANTITY
FROM      RPL PARENT, PARTLIST CHILD
WHERE     PARENT.SUBPART = CHILD.PART
)
```

Haupt-SELECT

```
SELECT    PART, SUBPART, SUM(QUANTITY) AS QUANTITY
FROM      RPL
GROUP     BY PART, SUBPART
```

Applikationen, die solche Tabellen verarbeiten werden auch als "Bill of Materials applications" bezeichnet.

Recursives SQL beinhaltet die Definition einer "common table expression" die auf sich selbst verweist („self-referencing").

Der iterative SELECT ist mit dem sogen. "initialization SELECT" über einen UNION ALL Ausdruck kombiniert.

Die Tiefe des rekursiven SELECT kann über die Angabe einer LEVEL-Spalte kontrolliert werden.

4.6 "Multi-row fetch and insert"

Das "multi-row FETCH/INSERT" Feature stellt eine Performance und eine Kompatibilitätsverbesserung dar. Dieses "feature" kann sowohl für "read only window scrolling" Applikationen, als auch für Benutzer, die bstimmte Daten sehen wollen und die Möglichkeit erwarten, diese auch zu ändern/zu ergänzen, nutzbar sein.

What is it? .....

1. **Multi-row FETCH:** Ein "single" FETCH Statement kann "multiple rows" von einer "result table" einer Query als "rowset" zurückliefern Ein "rowset" ist eine Gruppe von "rows", die gruppiert und als "set" verarbeitet werden. Unterstützt "dynamic" und "static" SQL (Fetch immer "static")

2. **Multi-row INSERT:** Ein "single" FETCH Statement kann eine oder mehrere Zeilen in eine Tabelle oder einen View einfügen (INSERT).
Multirow INSERT kann als "static" oder "dynamic" SQL implementiert sein.

DB2 V8 : DECLARE CURSOR Beispiel

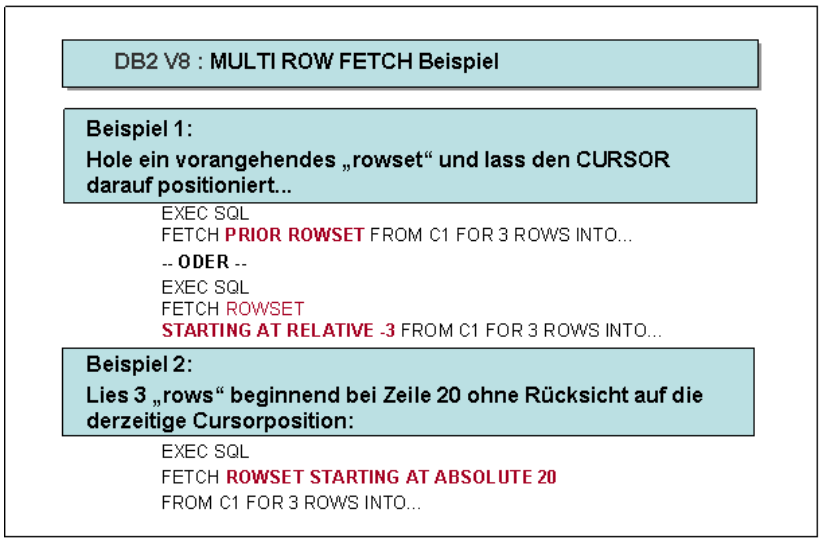
Deklaration von C1 als CURSOR einer Query, die ein „rowset“ aus der Tabelle EMP lesen soll

```
EXEC SQL
DECLARE C1 CURSOR
WITH ROWSET POSITIONING
FOR SELECT * FROM EMP;
```

WITH ROWSET POSITIONING gibt an, ob mehrere Datenzeilen von einem einzelnen FETCH Statement als sogenanntes „rowset“ zugegriffen werden können sollen.

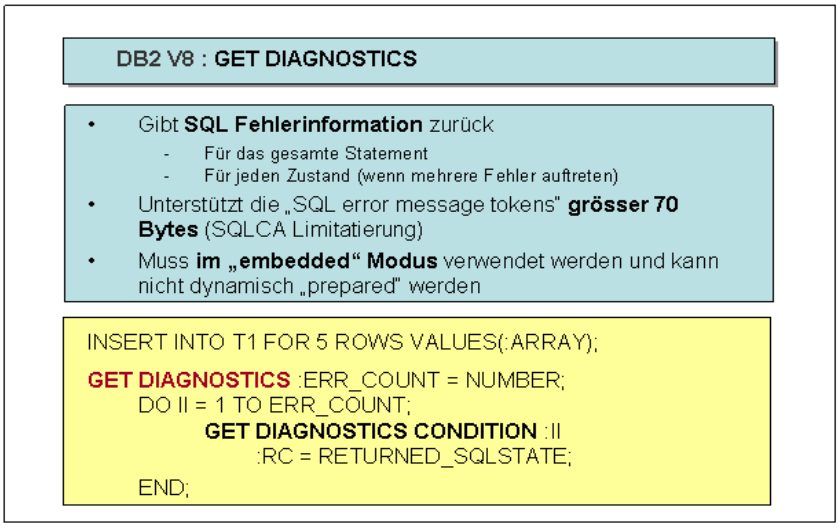
Benefits .....

1. Verbessert "Usability" und Mächtigkeit von SQL
2. Performance wird verbessert, wegen der Aufhebung multipler Aktionen zwischen Applikation und "database engine";
3. Bei "distributed access": wird der "network traffic" reduziert



4.7 Get diagnostics

Das neue **GET DIAGNOSTICS** Statement ist wichtig für die Informationen über alle “extended names” und die neuen Funktionen. Viele Entwickler werden ausgehend von den Informationen in der SQLCA diese zusätzlichen Standardinformationen nutzen wollen.



Das **GET DIAGNOSTICS Statement** gibt mehr Informationen über ein Statement und die damit verbundenen Bedingungen und „connections“. Beispielsweise kann das Statement die neuen “langen Namen”, die mehrfachen Bedingungsanzeigen für “multi-row statements” und die “error message” zu einem spezifischen Fehler ausgeben.

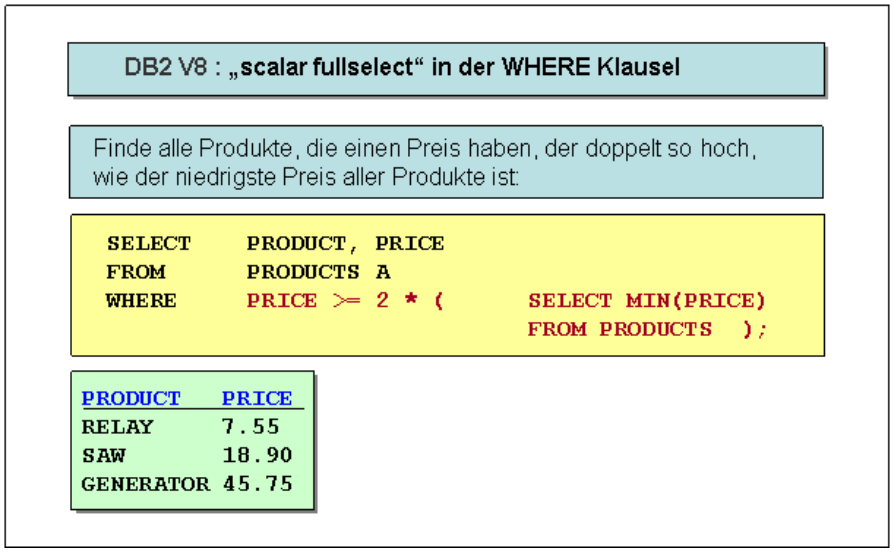
GET DIAGNOSTICS Beispiele

1. **Festellen wieviele “rows” in einem UPDATE Statement geändert wurden:**
   GET DIAGNOSTICS :rcount = ROW_COUNT;
2. **Behandeln von mehrfachen SQL Fehlern während eines NOT ATOMIC „multi-row Insert”**
   GET DIAGNOSTICS :numerrors = NUMBER;
   Dann folgt ein LOOP, um die anderen Fehler zu eruieren:
   GET DIAGNOSTICS CONDITION :i :retstate = RETURNED_SQLSTATE
3. **Ausgabe aller diagnostischen Informationen für ein SQL Statement**
   GET DIAGNOSTICS :diags = ALL STATEMENT -- beispielhafter “output” in :diags
   Number=1; Returned_SQLSTATE=02000;
   DB2_RETURNED_SQLCODE=+100; -- für alle “applicable items” und für alle
   -- Bedingungen

4.8 Scalar fullselect

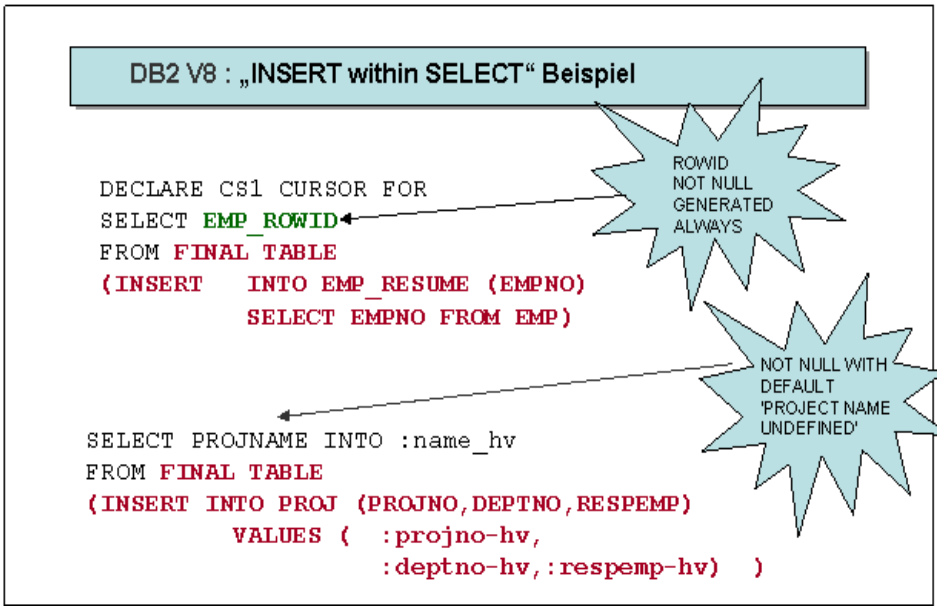
V8 beinhaltet die Möglichkeit eines **“scalar fullselect”** wo immer “expressions“ zulässig sind. Ein “scalar fullselect” kann maximal eine “row” und eine “column” zurückgeben.

Ein Beispiel:



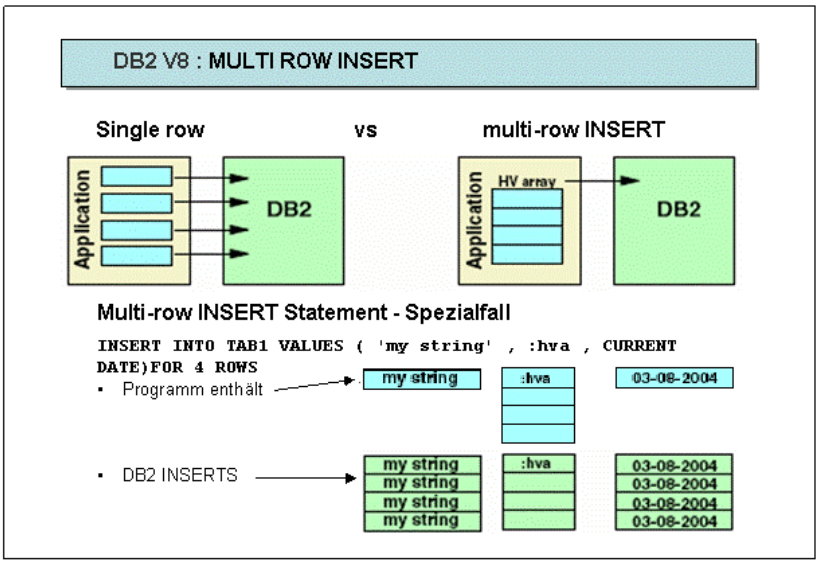
4.9 “Select from insert”

Diese Neuerung unterstützt die Anforderung “rows” oder eine “row”, die in eine Tabelle eingefügt werden soll(en) wie in folgenden Situationen vorher anzuzeigen:



- eine Applikation soll den Wert einer automatisch generierten Spalte (wie z.B. ROWID, „identity“) erkennen, falls eine neue „row“ in eine Tabelle „inserted“ wird
- eine Applikation kann Werte ,die eingefügt werden sollen, anfordern, obwohl sie von der rufenden Applikation nicht geliefert wurden (die Werte könnten aus „defaults“ bestehen oder sie wurden von einem „trigger“ geändert).
- eine einfacher syntaktischer Weg alle Werte einer “inserting row” darzustellen (ohne alle „column names“ zu spezifizieren).
- eine Möglichkeit, Werte von “multiple rows”, die “inserted” wurden, zu liefern

4.10 Multi-row INSERT



Der obere Teil des Bildes zeigt den Unterschiesd zwischen dem INSERT von einzelnen „rows“ und einem **„multi-row INSERT“**

Im ersten Fall **(single-row insert)** gibt es so viele Aufrufe an DB2, wie “rows” eingefügt werden sollen.

Im zweiten Fall **(multi-row insert)** ist nur ein einziger Zugang über das API erforderlich.

Es werden 4 “rows” eingefügt (das Maximum sind 32K “rows“ in einem einzigen INSERT).

4.11 “Qualified column names” in INSERT und UPDATE

DB2 V8 erlaubt die Qualifizierung der “column names” im INSERT Statement und in der SET Klausel des UPDATE Statement. Diese Erweiterung wurde aus Kompatibilitätsgründen eingeführt.

UPDATE T1 T SET T.C1 = C1 + 10 WHERE C1 = 2

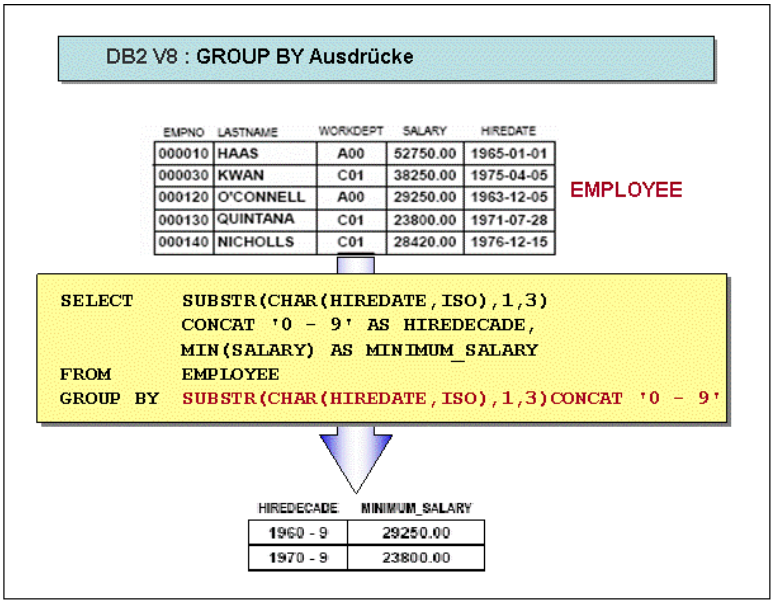
UPDATE MY.T1 SET MY.T1.C1 = C1 + 10 WHERE C1 = 4

4.12 “Expressions” im GROUP BY

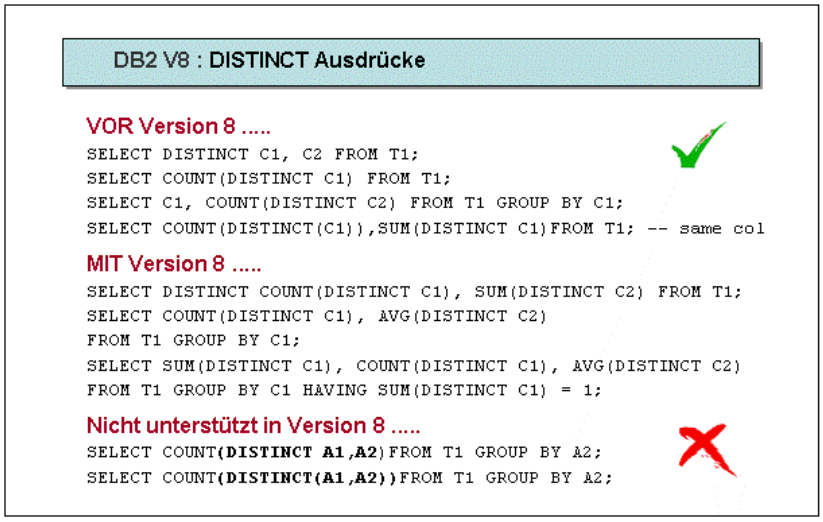
DB2 V8 unterstützt “expressions” in der GROUP BY Klausel.

Dies dient nicht nur der Kompatibilität, sondern hilft auch insofern, dass fremde Applikationen ohne Änderungen in DB2 lauffähig sind.

Das Beispiel zeigt die NUtzung von “expressions” im GROUP BY. (Minimales Gehalt, ausgezahlt an alle Mitarbeiter in der Dekade der Einstellung).



4.13 Multiple DISTINCT



Diese Erweiterung erlaubt die Verwendung des Schlüsselworts DISTINCT in mehr als einer „column function“ mit unterschiedlichen Ausdrücken.

Beispiel:

```
SELECT SUM(DISTINCT C1),
       AVG(DISTINCT C2)
FROM T1
```

4.14 IS NOT DISTINCT FROM

Per definitione, ist ein "null value" ein **undefinierter Wert** und dies macht ihn ungleich zu allen anderen Werten – andere "null values" eingeschlossen. Der einzige Weg auf "null values" zu testen ist die Verwendung des IS NULL Prädikates, wie in "**WHERE col IS NULL"**.

Das Prädikat : **"WHERE col = :hv :nullind"** trifft niemals einen Nullwert in "col", auch nicht, wenn die "host variable" "nullind" einen Null-Indikator (-1) enthält. Dies ist nicht intuitiv einsehbar. Angenommen man versucht "rows" zu lesen, in denen eine Spalte P1 den Wert NULL enthält. Man schreibt also:

"WHERE P1 IS NULL".

Um zwei Ausdrücke zu vergleichen und zu sehen ob sie gleich oder beide NULL sind muss eine Applikatione derzeut eine "compound search condition" formulieren:

(expr1 = expr2) OR (expr1 IS NULL AND expr2 IS NULL)

Um "rows" zu erhalten für die ein Wert für STADT existiert und dieser gleichzusetzen ist mit einem NULL-Wert, so kann man folgendes formulieren:

CITY = :CT :ctind

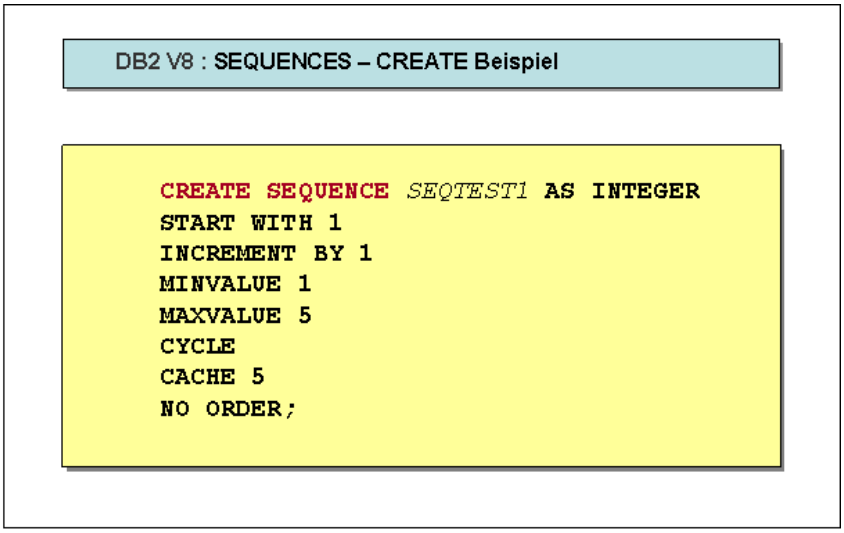
Aber die Suchbedingung wird niemals den Wert "true" annehmen, wenn der Wert in :CT NULL ist, auch nicht wenn die Hostvariable "ctind" den "null indicator" enthält. Dies deshalb, weil ein NULL-Wert niemals einem anderen NULL Wert gleich sein kann. Stattdessen muss man für diesen Fall folgende Kodierung verwenden:

CITY = :CT OR (CITY IS NULL AND :CT :ctind IS NULL)

Mit der Einführung des Prädikats **IS NOT DISTINCT FROM** vereinfacht sich die Suchbedingung wie folgt:

CITY IS NOT DISTINCT FROM :CT :ctind

4.15 Sequences

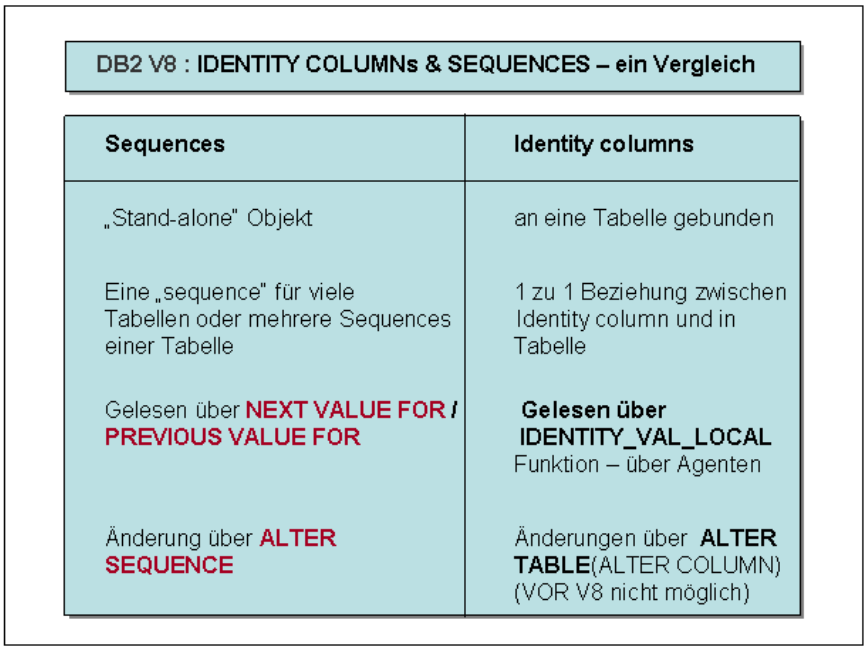


DB2 V8 unterstützt ein **neues Datenobjekt: die "sequence".** Eine "sequence" ist ein "user-defined object", das eine Folge von numerischen Werten generiert, je nach angegebenen "user-specifications". "Sequences" bilden eine ideale Möglichkeit schnelle vom DBMS generierte, "recoverable", garantiert eindeutige, sequentielle numerische Schlüsselwerte zu erhalten. Sie können auch dazu dienen, Schlüsselwerte über Tabellen hinweg zu koordinieren und in "data sharing environments" konsistent zu halten.

Eine "sequence" wird mit dem **CREATE SEQUENCE** Statement angelegt. Die Attribute können mit dem ALTER SEQUENCE Statement geändert werden. Weitere SQL statements im Zusammenhang mit "sequences" sind GRANT/REVOKE ON SEQUENCE, DROP SEQUENCE und COMMENT ON SEQUENCE.

Der aktuelle und der nächste Wert einer "sequence" wird erhalten über **PREV**ious **VALUE FOR SEQUENCE** und **NEXT VALUE FOR SEQUENCE**.

4.16 "Identity columns" Verbesserungen



Mit DB2 V8 werden die Möglichkeiten der "identity columns" im ALTER TABLE ALTER COLUMN Statement ergänzt.

Man kann zusätzliche "identity column" Spezifikationen mitgeben, dynamisch die Attribute bestehender "identity columns" ändern und es sind einige zusätzliche Definitionen zur "column specification" vorgesehen.

Der Unterschied zwischen "sequences" und "iudentity columns" wird auf dem nebenliegenden Bild kurz erläutert.

4.17 Session variables

"Session variables" stellen eine zusätzliche Methode dar, um Programmen Informationen über die DB2-Umgebung zu liefern. "Session variables" können von SQL Statements – wie "special registers" – verwendet werden. Sie können in "views", "triggers", "stored procedures" oder "constraints" gesetzt werden. Zum Beispiel:

```
CREATE VIEW V1 AS SELECT *
FROM T1
WHERE COL5 = GETVARIABLE(SYSIBM.SECLABEL);
```

"DB2-defined session variable"s

Die "session variables" werden von DB2 selbst definiert. Sie laufen alle unter dem Schema-Namen SYSIBM:

- _ **SYSIBM.DATA_SHARING_GROUP_NAME**
- _ **SYSIBM.PACKAGE_NAME**
- _ **SYSIBM.PACKAGE_SCHEMA**
- _ **SYSIBM.PACKAGE_VERSION**
- _ **SYSIBM.PLAN_NAME**
- _ **SYSIBM.SECLABEL**
- _ **SYSIBM.SYSTEM_NAME**
- _ **SYSIBM.SYSTEM_ASCII_CCSID**
- _ **SYSIBM.SYSTEM_EBCDIC_CCSID**
- _ **SYSIBM.SYSTEM_UNICODE_CCSID**
- _ **SYSIBM.VERSION**

5 Verbesserungen für die AE

5.1 Stored procedure und UDF Verbesserungen

In V8 kann man festlegen, dass eine "stored routine" in einen "stopped state" gesetzt wird, wenn eine bestimmte Anzahl Fehler sich ereignet haben.

STOP AFTER nn FAILURES

Setzt eine Routine in den "stopped state" nach *nn* Fehlern. Der Wert kann sein von 1 bis 32767.

STOP AFTER SYSTEM DEFAULT FAILURES

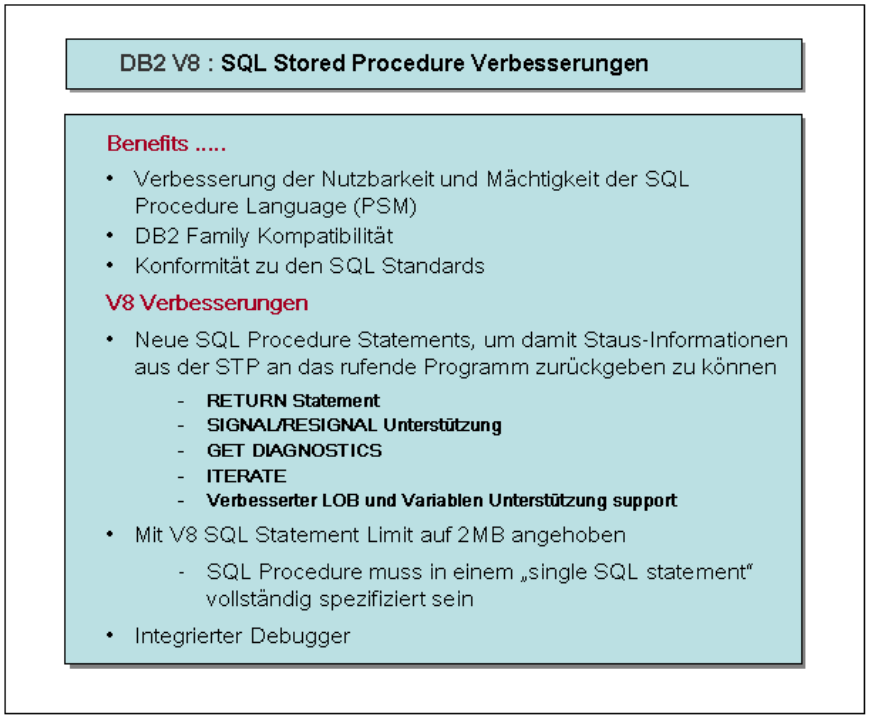
Setzt eine Routine in den "stopped state" nach der Anzahl von Fehlern, die im Systemparameter MAX ABEND COUNT (DSNZPARM STORMXAB) gesetzt wurden

CONTINUE AFTER FAILURE

Gibt an, dass die entsprechende Routine niemals in einen "stopped state" gesetzt werden soll. Diese Option ist nichtr möglich für "SQL functions" oder "sourced functions" (SQLSTATE

42849, SQLCODE -20102).

5.2 SQL stored procedure Verbesserungen



DB2 Version 8 liefert einige signifikante Verbesserungen zu "SQL stored procedures", auch **PSM (Persistent Stored Modules)** genannt (siehe links).

5.3 SET [CURRENT] SCHEMA

Für "static SQL Statements" kann die **DB2 QUALIFIER Option** beim BIND genutzt werden, um den impliziten Qualifier für "unqualified object names" festzulegen (ausgenommen für die, die den SQL PATH nutzen, um den Namen eines Objekts aufzulösen) . Die QUALIFIER Bind Option betrifft dann alle "unqualified object names". Ist die QUALIFIER Bind Option nicht angegeben, wird der PLAN/PACKAGE OWNER als impliziter "object qualifier" für "static SQL Statements" angenommen.

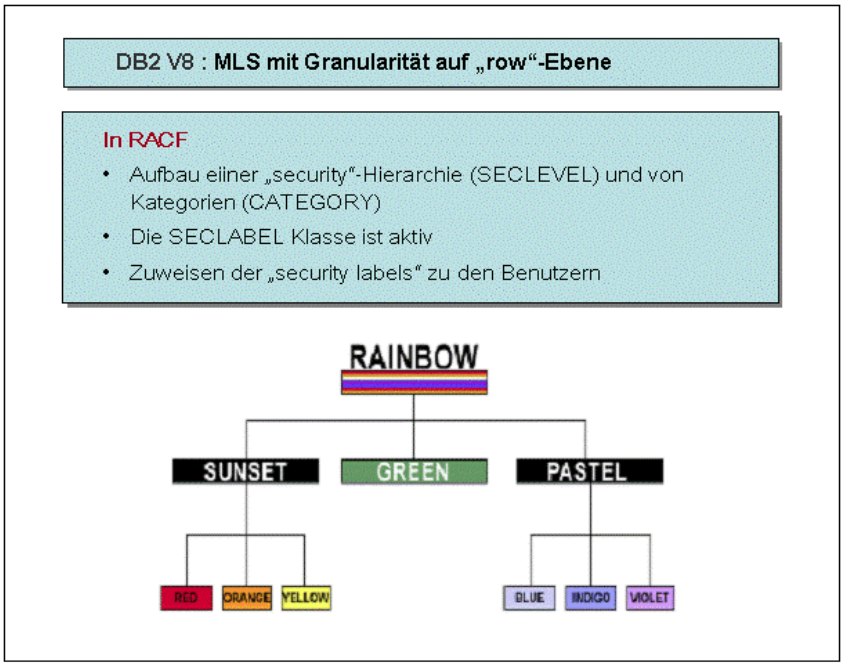
Für "dynamic SQL statements" kann das CURRENT SQLID Spezialregister benutzt werden, um einen impliziten Qualifier für "unqualified object names" zu bestimmen (ausgenommen für die, die den SQL PATH nutzen, um den Namen eines Objekts aufzulösen t).

Aber zum Unterschied zur QUALIFIER Bind Option hat die Nutzung des CURRENT SQLID Spezialregisters zusätzliche Auswirkungen, da es auch für "authorization checking" auf "dynamic" CREATE, ALTER, GRANT, und REVOKE Statements verwendet wird und der Wert auch als "owner" (oder "definer") von Objekten gilt, die über "dynamic" CREATE Statements erzeugt wurden.

Die Bündelung der Qualifizierung von Objektnamen und Authorisierungsprüfung, sowie der "ownership of objects" ist nicht ideal für "dynamic SQL statements".

DB2 for z/OS Version 8 bietet zur Lösung hierfür ein neues Statement - **SET (CURRENT) SCHEMA SQL -** und ein neues Spezialregister - **CURRENT SCHEMA** - . Es dient ausschliesslich der Qualifizierung von "object names" in "unqualified dynamic SQL statements" (wenn DYNAMICRULES(RUN) gesetzt ist – als "default".

5.4 Multilevel security



Eine wichtige Anforderung für "row-level security" in Anwendungen ist eine höhere Granularität der Sicherheitsebenen. Üblicherweise werden "views" und "joins" zur Anwendungslösung dieser Anforderungen. Die sogen. "multilevel security", die mit DB2 V8 eingeführt wird unterstützt hierarchische "security schemes" und vereint Erweiterungen in SQL mit der RACF Zugriffskontrolle bis hin zur **"row granularity"**.

5.5 MQSeries UDF

DB2 und MQSeries können genutzt werden,um Anwendungen zu konstruieren, die die Fähigkeiten beider Systeme in sich vereinen.

Es ist nun möglich "MQSeries messaging" Operationen innerhalb von SQL Statements zu verwenden.

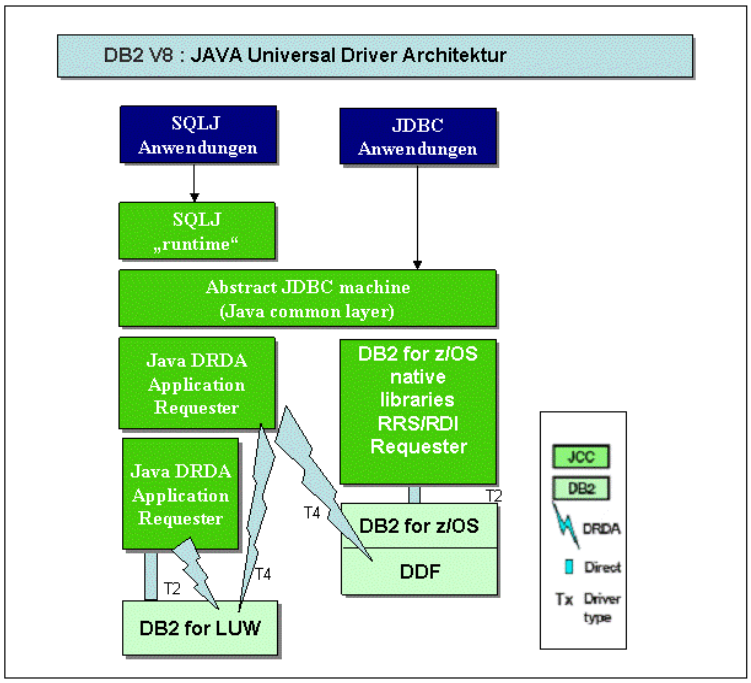
6 e-business

In diesem Kapitel werden die wichtigsten Änderungen in DB2 V8 bezüglich seiner "e-business"-Fähigkeit dargestellt.

6.1 Universal Driver für SQLJ und JDBC

Unternehmen verlangen mehr und mehr nach Zugriffen auf Datenquellen, die sich auf unterschiedlichen Plattformen befinden. Firmen kaufen eher Anwendungen als "database management systems". Die Auswahl ist geführt von den Kriterien wie der "Interoperabilität", dem Preis/Leistungsverhältnis und der "scalability" der Serverplattform.

6.1.1 JDBC



Dieses "feature" bietet einen **offenes und konsistentes Paket von "database protocols"** zum Zugriff auf Daten auf UNIX, Windows, und z/OS Plattformen. Tools und Applikationen können unter Nutzung eines konsistenten "sets of interfaces" entwickelt und genutzt werden, ohne Rücksicht darauf nehmen zu müssen, WO die Daten sich letztendlich befinden.

Endbenutzer können ihre "desktop tools" und andere Applikationen zu einem Paket bündeln, egal mit welchen Datenbanken (oder "multiple databases" gleichzeitig) sie auch arbeiten müssen. Das Ziel dieser Verbesserung in DB2 V8 ist die Implementierung der Version 3 der **"Open Group DRDA Technical Standards"**.

Dies erübrigt den Einsatz von "gateways" und verbessert die "desktop performance".

Unterstützte Plattformen und "Connectivity"

V 1.0 supported Type 4 connectivity to most DB2s
Bundled with DB2 for LUW v8.1, WAS, and Cloudscape
V 1.2 added Type 2 connectivity to DB2 for LUW, type 4 connectivity to iSeries
Bundled with DB2 for LUW v8.1 fix pack 2
V 1.3 added minimum JDBC 3 compliance
Bundled with DB2 for LUW v8.1 fix pack 3
V 1.4 added Beta Type 2 connectivity to z/OS v8
V 1.5 added full JDBC 3 compliance
Bundled with DB2 for LUW v8.1 fix pack 4
V 1.6 added Beta Type 4 XA (2 phase commit) connectivity to DB2 for z/OS V 7 and V 8
V 1.7 added Type 4 XA connectivity to DB2 V7 z/OS
V 2.1 GA of z/OS Type 2 JNI-based local connectivity
Released via APAR PQ80841 / PTF UQ85607 for DB2 for z/OS and OS/390 V7
Productional release via APAR PQ85053 / PTF UQ85928 for DB2 for z/OS V8

6.1.2 SQLJ

DB2 V8 : Vergleich – SQLJ und JDBC Code

Lesen einer einzelnen „row" von DB2.....

SQLJ

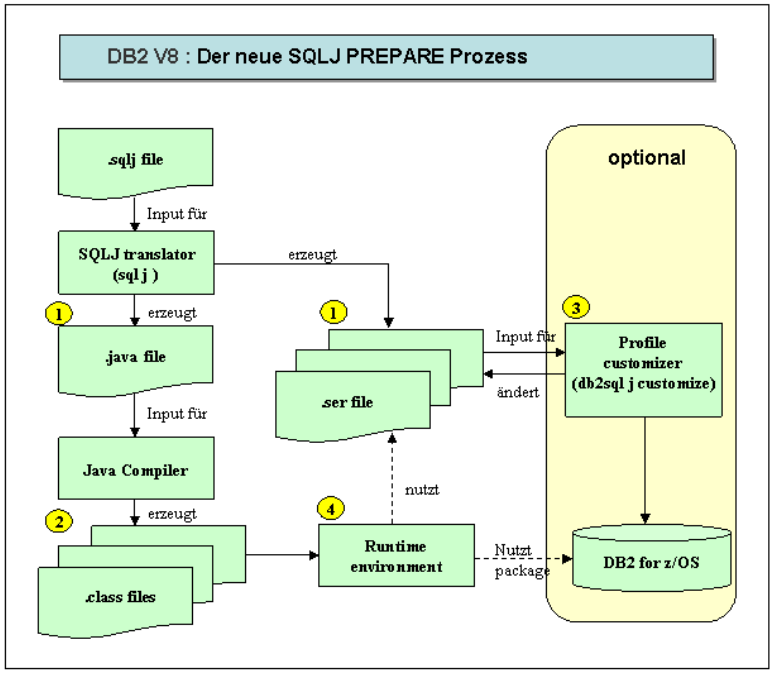
```
String addr; String name;
....
#sql [ctx] { SELECT        address INTO :addr
             FROM          emp WHERE name =:name };
```

JDBC

```
String addr; String name;
....
PreparedStatement pstmt = con.prepareStatement (
   "SELECT address FROM emp WHERE name = ?" );
pstmt.setString(1, name);
ResultSet names = pstmt.executeQuery();
names.next();
addr = names.getString(1);
names.close();
pstmt.close();
```

Ein Vergleich einer "single row Query" zwischen JDBC und SQLJ zeigt die Unterschiede der beiden Methoden:
In **JDBC**, muss man ein "result set" verfügbar machen und die NEXT (und einzige) "row über die "getxx Methoden" zur Verfügung stellen. Zusätzlich muss geprüft werden, ob genau eine einzige "row" gefunden wurde!
In **SQLJ** kann man die SELECT INTO Syntax nutzen; eine SQLException wird gesetzt, falls mehr als eine "row" gefunden wurde.
Die SQLJ Version ist also offensichtlich effizienter als JDBC.
JDBC muss zudem vier CALLs an DB2 absetzen (prepare statement, fetch row, fetch row, close statement) wohingegen SQLJ nur einen "single SELECT INTO" Call verwendet.

Anmerkung: Die SQLJ Variante ist nur dann effrizienter, wenn das Programm "customized" und "bound" ist. Läuft es "uncustomized", so emuliert der SELECT INTO "result sets" wie in der JDBC Variante.



Mit dem neuen **Universal Driver** werden keine DBRMs (oder .bnd files) mehr genutzt.
Über die Nutzung des **"db2sqljcustomize"** Kommandos kann man "serialized profiles" erzeugen and gleichzeitig den BIND der "packages" auf dem DB2 Zielsystem auslösen.
Mit dem **Type 4 Driver**, werden von jeder beliebigen Plattform DIREKT die Verbindungen auf das Zielsystem DB2 hergestellt, die "online"-Prüfung durchgeführt und die "packages" auf dem Ziel-DB2 gebunden.
Zusätzlich dazu behandelt der "Universal Driver" das "serialized profile" so, dass es in jedem Fall portable bleibt. So kann man dieselben Programmdateien gegen jede beliebige Plattform laufen lassen. Voraussetzung ist, dass das "db2sqlbind utility" genutzt wird, um zur neuen Lokation zu verbinden und die korrekten "program packages" zu binden.

6.2 Unicode support

Ein architektive Änderung in DB2 V8 betrifft die vollständige Konvertierung des DB2 Katalogs nach Unicode. Das bedeutet aber auch, dass diese Daten weltweit verwendet werden können. DB2 konvertiert zudem jedes SQL Statement erst nach Unicode bevor es "geparsed" wird.

6.2.1 Importance of Unicode knowledge

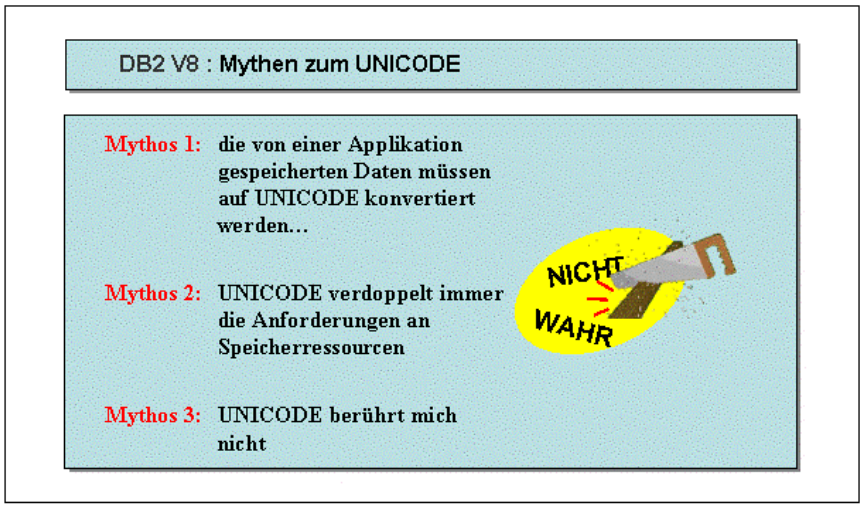
Der verbesserte Unicode Support ist eine der grössten Änderungen in DB2 for z/OS V8. Die folgenden Bereiche sin insbesondere vom "enhanced Unicode support" betroffen:

- Der DB2 Katalog wird in Unicode geführt
- Das "parsing" der SQL Statements wird in Unicode durchgeführt
- Der DB2 Precompiler generiert DBRMs, die in Unicode-Format gespeichert werden
- Nutzt man SQL Statements mit "multiple CCSID sets", so wird der Vergleich von Zeichen und Zeichen ketten oft in Unicode erfolgen
- Die Resultate von SQL Statements werden häufig in Unicode Format zurückgegeben
- Die Unicode Zeichenfolge unterscheidet sich von der EBCDIC-Folge
- Unicode daten können mehr Speicher erfordern. Dies ist wichtig für die Daten in DB2 und ebenso für die Anwendungsprogramme.

6.2.2 Drei Mythen um den UNICODE

Mythos 1: Die gespeicherten Daten der Applikation müssen auf Unicode konvertiert werden
Version 8 verbessert die Infrastruktur zum Speichern und Wiedergewinnen von Daten in UNICODE Format. Dies schliesst die Konvertierung des DB2 Katalogs in Unicode mit ein. Das Ziel ist jedoch, nicht alle Benutzerdaten in Unicode Format halten zu müssen. Es ist eine Wahlmöglichkeit.
Es gibt keine Notwendigkeit, alle Daten von EBCDIC nach Unicode zu konvertieren. Mit der Unterstützung mehrerer CCSIDs in einem einzigen SQL Statement ist die Notwendigkeit der Konvertierung von Daten in Unicode im Vergleich zur Version 7 deutlich gesunken. (Aber: Es gibt Einflüsse auf die Performance, wenn man "multiple CCSID" in einem SQL Statement setzt.)

Mythos 2: Unicode verdoppelt die Speicheranforderungen
Es gibt eine Reihe von Missverständnissen, was die Verdoppelung der Speicheranforderungen betrifft, falls Daten nach Unicode konvertiert werden. Die DB2 Family nutzt die Unicode Transformation Format UTF-8 und UTF-16.



Dabei ist es wichtig zu wissen, dass in UTF-8 die ersten 127 "code points" dieselben sind, wie in 7-bit ASCII ausgenommen 1 Byte das für Zeichen wie A-Z, a-z, und 0-9 gebraucht wird. Andere Zeichen werden in 1 bis 4 Bytes gespeichert. Meistens werrden nur 2 Bytes benötigt, bei fernöstlichen Zeichensätzen 3 bis 4 Bytes; d.h. der Anstieg des Speicherplatzes für das system hängt von der Art der daten ab, die nach Unicode konvertiert werden.

Mythos 3: Unicode geht mich nix an
Jeder ist von den Änderungen in DB2 Version 8 betroffen, egal ob die Daten in Unicode gespeichert werden oder nicht.

6.3 ODBC Verbesserungen

Einige der Verbesserungen betreffen ODBC u.a.:

- **Unicode support:** ODBC unterstützt nun die Unicode Formate UTF-8 und UCS-2. Dazu gibt es ein neues ODBC Initialisierungsschlüsselwort: **CURRENTAPPENSCH**. Es erlaubt die Angabe des "encoding scheme", das der ODBC Driver für Input und Output der Hostvariablen verwendet soll. Davon betroffen sind auch SQL statements und alle "character string" Argumente des ODBC Application Programming Interface. Angegeben werden kann: **Unicode, EBCDIC oder ASCII**.
- **ODBC SQL "Connect user" und "password" Unterstützung:** Mit V7 kann der DB2 ODBC Driver "userid" und "password" überprüfen. Aber die Werte der Argumente werden nicht für die Authentifikation der Endbenutzer genutzt. DB2 V8 implementiert die Unterstützung des USER/USING SQL CONNECT Statements und der ODBC Driver nutzt "userid" und "password" im API, um die Authentifikation durchzuführen

6.4 XML publishing functions

XML wurde in der Vergangenheit schnell zur de facto "Data Format Sprache" im Internet, in Intranets und für den Datenaustausch.

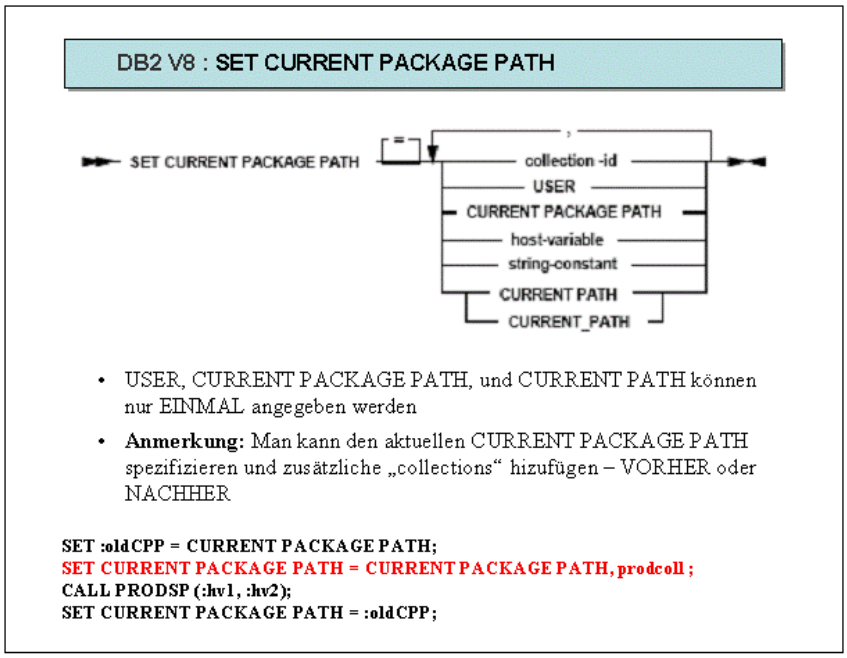
In vielen Fällen müssen Anwendungen XML Daten aus traditionellen relationalen Datenbanken generieren. Sie nutzen dazu "application packages" oder "middleware".

Diese DB2 Erweiterung enthält ein Set von SQL "built-in-functions", die dies mit hoher Performance m öglich machen. Das vermindert den Entwicklungsaufwand für die Generierung von XML Daten, Informationsaustausch und Web Services.

Beispiel:

```
SELECT VARCHAR( XML2CLOB( XMLElement(NAME "TABLE",
                    XMLATTRIBUTES('1' as "border"),
                    XMLElement(NAME CAPTION, 'Department-Employee Table'),
                    XMLElement(NAME TR, XMLFOREST('Dept No' as TH, 'Dept Name' as TH,
                                    'Emp No' as TH, 'Emp Name' as TH, 'Phone' as TH) ),
                    XMLAGG(
                    XMLCONCAT(
                    XMLELEMENT(NAME TR, XMLELEMENT(NAME TD,
                                     XMLATTRIBUTES( X.CNT+1 as "rowspan"),
                                        D.DEPTNO),
                                  XMLELEMENT(NAME TD,
                                           XMLATTRIBUTES( X.CNT+1 as "rowspan"),
                                                  D.DEPTNAME)
                                           ),
                    ( SELECT  XMLAGG(XMLElement(NAME TR,
                                     XMLForest(EMPNO as TD,
                                              FIRSTNME || ' ' || LASTNAME as TD,
                                              PHONENO as TD)  )  )
                      FROM DSN8810.EMP E
                      WHERE  E.WORKDEPT = D.DEPTNO )
                        ) ) ) ) )
FROM DSN8810.DEPT D, (SELECT WORKDEPT, COUNT(*)
                      FROM DSN8810.EMP GROUP BY WORKDEPT) X(DEPTNO, CNT)
WHERE D.DEPTNO = X.DEPTNO AND
             D.DEPTNO IN ('A00', 'C01')
```

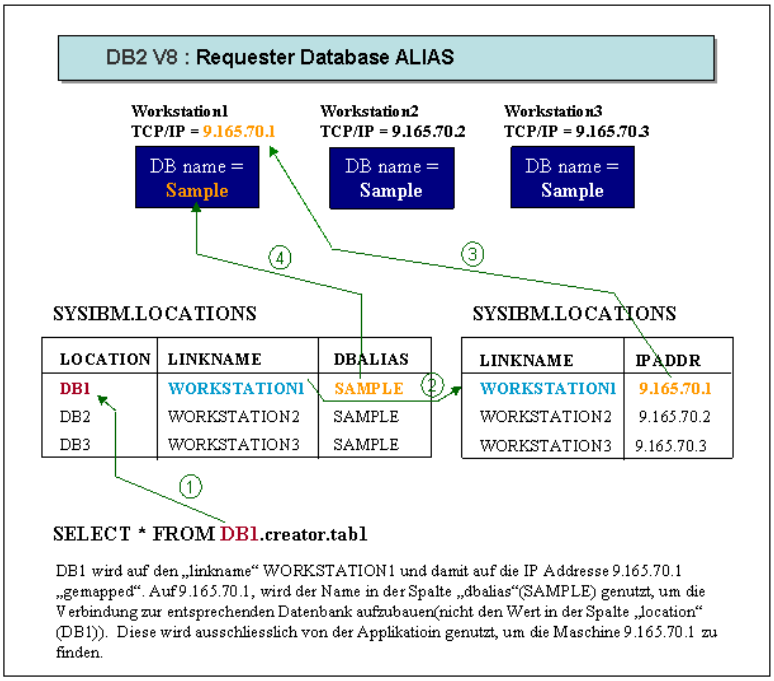
6.5 CURRENT PACKAGE PATH Spezialregister



"Package switching" und "Versionsführung" für "static SQL Applikationen ist immer kritisch. SQLJ Zugriffe erhöhen die Notwendigkeit für diese Art von Kontrolle. Diese Erweiterung bietet ein zusätzliches neues Spezialregister: **CURRENT PACKAGE PATH**. Dies bedeutet die Spezifikation einer Liste von "collections", die nach dem erforderlichen "package" durchsucht werden soll. Die Semantik gleicht der PKLIST Bind Option mit der die PACKAGE PATH Liste am Server verarbeitet wird. Das neue Spezialregister erlaubt diese Kontrolle für Applikationen, die nicht unter einem DB2 Plan laufen.

6.6 DDF Verbesserungen

Diese Erweiterungen im DDF umfassen:



- **Requester database ALIAS:** eine neue Spalte DBALIAS, die in der CDB-Tabelle SYSIBM.LOCATIONS eingefügt wurde. Sie zeigt auf die Korrekte TCP/IP Adresse für die Ziel Workstation. Man kann nun auf "databases" zugreifen, die auf den diversen LINUX/UNIX/Windows Systemen die gleichen Namen haben.
- **Server location alias:** Um die Migration dieser "location names" zu erleichtern, kann man nun die BSDS ergänzen , um die erforderlichen "server location aliases" einzustellen.
- **Member routing in a TCP/IP network:** in Kombination mit "server location alias" und der Definition in der neuen SYSIBM.IPLIST Tabelle kann man bestimmte "member" verlagern, damit WLM das "balancing" übernehmen kann.

6.7 Verbesserungen für “stored procedures” und UDFs

Derzeit ist es möglich einen maximale “abend”-wert für alle „stored procedures“ und „user defined functions“ auf einem einzelnen DB2 Subsystem zu setzen. Das ist nicht immer zufrieden stellend, da es sich meist um einen Mix aus etablierten Anwendungen und Anwendungen unter Testbedingungen handelt.

Diese Verbesserung ermöglicht das **Setzen eines Grenzwertes**, der bestimmt, wie oft eine „stored procedure“ oder eine „user defined function“ auf Fehler laufen kann, bevor sie gestopped wird. Dies unterstützt auch die Nutzung von Ressourcen in einem z/OS WLM Umfeld und liefert eine Methode, die Anzahl “tasks” im “stored procedure address space” zu kontrollieren.

DB2 V8 unterstützt **keine COMPJAVA “stored procedures”** mehr.

6.8 „SQL procedure“ Erweiterungen

Neben anderen Verbesserungen erlauben die Statements **SIGNAL und RESIGNAL** der SQL Prozedur einen spezifischen SQLSTATE Wert und einen Meldungstext event-gesteuert innerhalb der SQL Procedur zu erzeugen.

Ist die Bedingung nicht im „procedure body” verarbeitbar, wird die Information an den “caller” zurückgegeben. Diese Funktion ist sinnvoll für “packaged applications”, wie “extenders”, die eigene SQLSTATEs erzeugen und die sie an die aufrufende Applikation zurückgeben wollen.

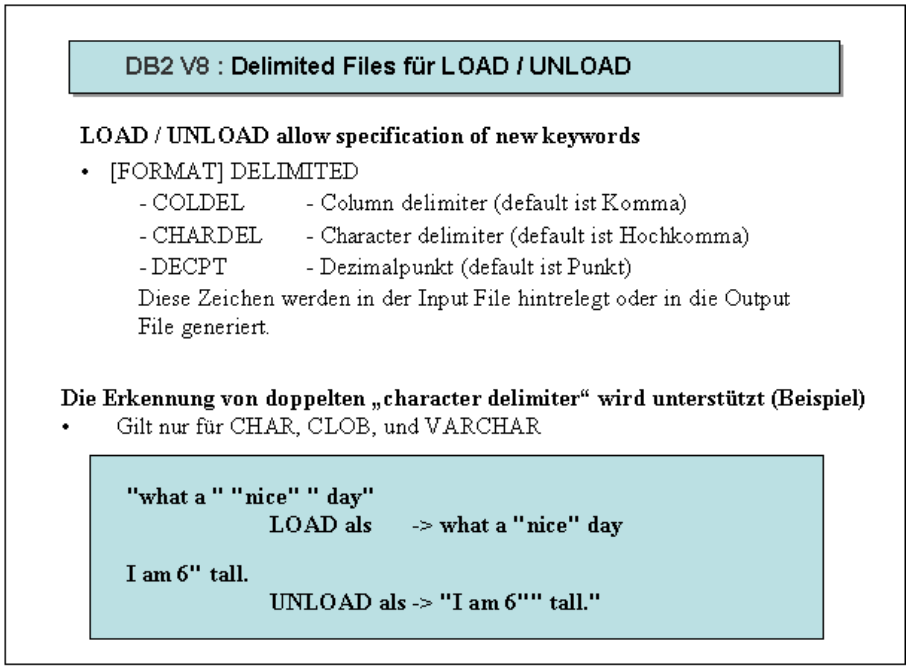
7 Utilities

7.1 Online Schema Änderungen

Da die Verfügbarkeit nach 24x7 immer kritischer für Applikationen wird, steigt auch die Anforderung DB-Objekte ändern zu können ohne diesen faktor nachhaltig beeinflussen zu müssen. "Online Schema Evolution" ermöglicht Änderungen an "table", "index" und "table space" Attributen attribute changes während grösstmöglicher Applikationsverfügbarkeit. Beispiel: "column types" und Längenangaben können geändert, Spalten zu einem IX hinzugefügt, Partitions ergänzt und "rotated" und eine Spezifikation abgegeben werden, um zu bestimmen welcher Index ("partitioning index" oder ein "non-partitioning index") als "clustering index" verwendet werden soll.

7.2 LOAD und UNLOAD mit “delimiter”

Diese Erweiterung setzt den LOAD Proz in die Lage, “input files”, in denen die Spalten über “delimiter” wie “,” identifiziert werden, anstatt von fixen Positionen innerhalb eines Satzes und /oder eines 2-Byte Längenfeldes für VARCHAR Eingaben zu identifizieren.



Dies eröffnet die Möglichkeit, Daten, die auf einer Workstation aus DB2, Oracle, Sybase, oder Informix entladen wurden, mit dem LOAD Utility zu verarbeiten und zu laden.

Auf der anderen Seite kann das UNLOAD Utility die Daten mit “delimiters” in eine Ausgabedatei entladen, um dieses „data set“ in einem anderen System wieder zu laden.

7.3 RUNSTATS und Verteilungsstatistiken

Dies ist eine neue Funktion zur Kalkulation von Vorkommenhäufigkeiten auf "non-indexed columns" im RUNSTATS. Die betroffenen Katalogtabellen werden mit den entsprechenden Werten bezüglich der Häufigkeiten versehen und optional auch mit den niedrigsten vorkommenden Häufigkeitswerten. Die neue Funktion sammelt auch Angaben über "multicolumn cardinality" für "non-indexed column groups".

Wann immer der DB2 Optimizer den erforderlichen "filter factor" genauer bestimmen kann, führt dies zu besseren Optimierungswahlmöglichkeiten in den Prädikaten und hilft damit der Performance von Queries. Das Utility RUNSTATS wurde in der Version 8 genau deshalb um die Kalkulation zusätzlicher Statistiken ergänzt:

- Verteilungsstatistiken für "non-indexed columns" oder Spaltengruppen
- Kardinalitätswerte für "non-indexed column"-Gruppen
- "LEAST frequently occurring values" und "MOST frequently occurring values" sowohl für "indexed" und "non-indexed columns" und deren Verteilungstypus in den Tabellen. Diese Information konnte in früheren Versionen von DB2 nicht genutzt werden.

7.4 “Back up” und “restore system”

Diese zwei neuen Utilities bieten “system level” und “point-in-time level” beim Recovery. Sie aktivieren neue Funktionen im neuen V1R5 DFSMShsms, die einen einfacheren und weniger gefährlichen Weg für “fast volume-level backup and recovery“ beim „disaster recovery“ und „system cloning“ erlauben. Die Funktion ist insbesondere für ERP Lösungen interessant, wo grosse Kopien von Recovery-Daten und Indizes auf grossen Mengen von Plattenlaufwerken zur aplikationsbezogenen Konsistenz synchronisiert sein müssen. Sie sind ein Eckpfeiler für jedwede Recovery Lösung.

7.5 REORG Verbesserungen

REORG utility is enhanced to allow you to specify that only partitions placed in Reorg Pending state should be reorganized. You do not have to specify the partition number or the partition range. You can also specify that the rows in the table space or the partition ranges being reorganized should be evenly distributed for each partition range when they are reloaded. Thus, you do not have to execute ALTER INDEX statement before executing the REORG utility. You can specify DISCARD with SHRLEVEL CHANGE. You can avoid BUILD2 phase during online REORG by using the new data partitioned secondary indexes.

- **REORG REBALANCE** gibt an, dass eine REORG TABLESPACE neue “partition Grenzen” setzt. So können alle “rows”, die an der Reorganisation beteiligt sind eine neu Verteilung über die Partitions erhalten.
- **REORG REBALANCE und ALTER + REORG Vergleich**
- **REORG TABLESPACE - SCOPE PENDING:** Der REORG TABLESPACE wird um das neue Schlüsselwort *SCOPE* erweitert, um anzuzeigen welchen Umfang die Reorganisation des TS und/oder der Partition betrifft.
- **REORG TABLESPACE ... DISCARD:** In Version 8 Kann das Schlüsselwort DISCARD auch beim REORG SHRLEVEL CHANGE angegeben werden.
- **REBUILD INDEX - SCOPE PENDING:** In DB2 V8 kann ähnlich zum REORG, auch der REBUILD INDEX mit dem neuen Schlüsselwort *SCOPE* versehen werden

7.6 Weitere Verbesserungen

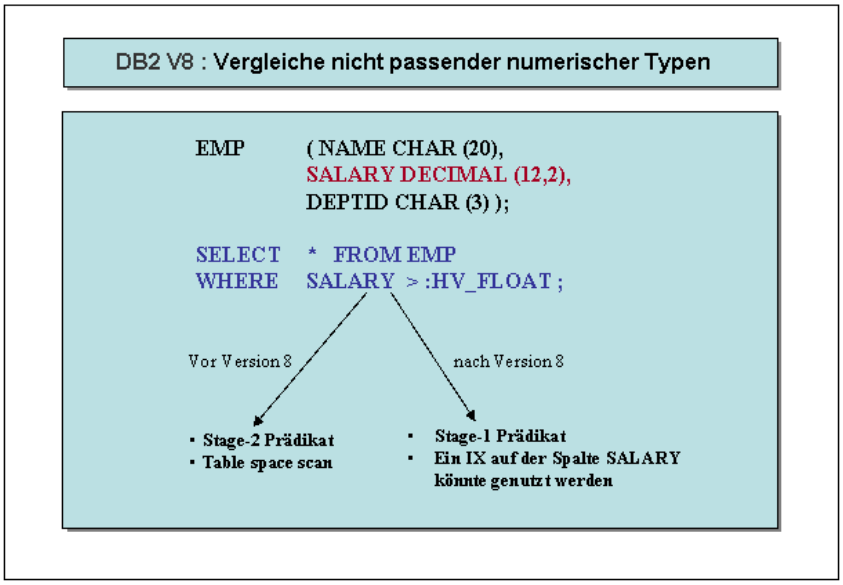
Zu den weiteren Verbesserungen in den DB2 Utilities gehören:

- Neue "utility defaults" für bessere Performance
- REBUILD INDEX - SCOPE PENDING
- Verbesserungen beim COPY
- REPAIR
- Utility Änderungen zur Unterstützun von sogen. "informational RI constraints"
- Utility Änderungen zur Unterstützung von DPSIs
- Änderungen an "Stand-alone utilities" (DSN1COMP, DSN1PRNT)
- DSNUTILU als Unicode Utility Stored Procedure

8 Performance

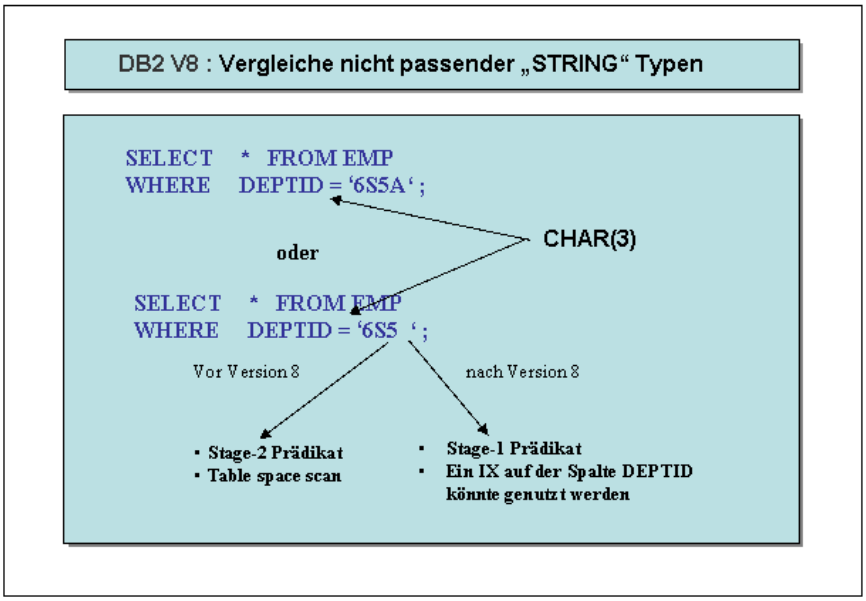
Übersicht über die Änderungen in V8

- Stage 1 und „indexable predicates“
- „Materialized query tables“
- Indexing Verbesserungen
- “Table UDF cardinality” Option und “block fetch”
- Trigger Verbesserungen
- “Distribution statistics” auf “non-indexed columns”
- “Cost-based parallel sor” für „single“ und „multiple“ Tabellen Abfragen
- Performance von “multi-row” Operationen
- Unterstützung von “volatile tables”



- “Data caching” und spezielle Indexsuche beim “star join”
- Weitere Performance Verbesserungen
- Visual Explain Erweiterungen

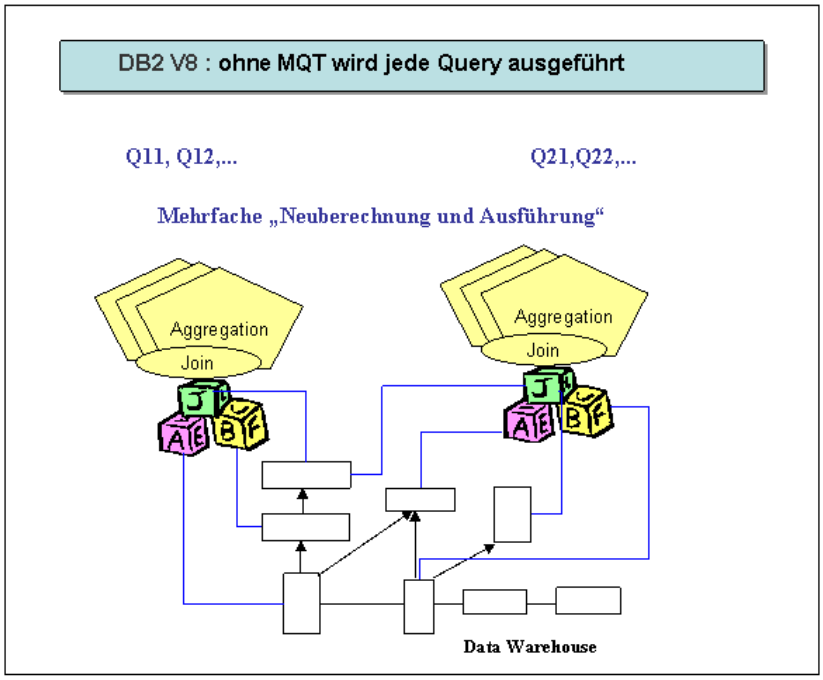
8.1 Vergleich von Daten ungleichen Typs



Diese Änderung verbessert die Performance von DB2 V8, indem die Prädikate in “stage 1” verarbeitet warden können, selbst, wenn die zu vergleichenden „columns/values“ unterschiedliche Datentypen aufweisen. Sie müssen lediglich kompatibel sein.

8.2 Materialized Query Tables(MQT's)

Was versteht man unter MQTs? – Eine “materialized query table” (MQT) enthält vorqualifizierte Daten. Diese “pre-computed data” sind ein Resultat einer Query, eines “fullselect” in Verbindung mit einer Tabelle, die als Teil eines CREATE/ALTER TABLE Statement spezifiziert worden ist.



Die **Ursache für eine “materialized query table”** können “base tables”, “views”, “table expressions” oder “user-defined table” Funktionen sein.

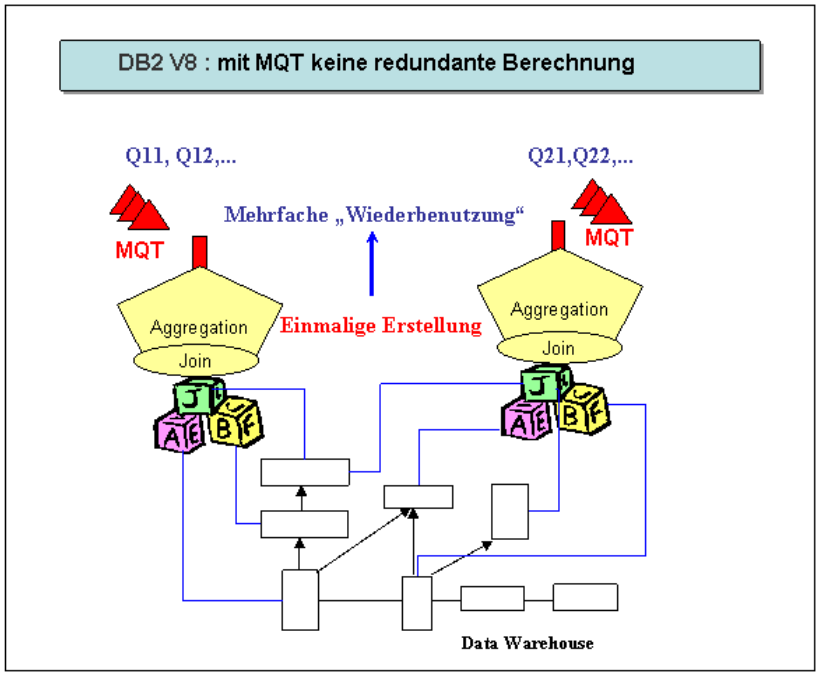
MQTs können entweder direkt über SQL oder vom Optimizer (über *„automatic query rewrite”*) verwendet werden.

Ein weiterer Fall kann über die Nutzung der DBMS’s Optimierungstechniken entstehen, wo entschieden wird, ob MQT’s angelegt werden. Das DBMS verwaltet diese vollständig transparent. Im zweiten Fall spricht man von ***“automatic materialized query tables”*** oder von ***“automatic summary tables”*** (ASTs) bzw. ***“materialized views”.***

Die Klassifizierung von MQTs kann auf unterschiedliche Art und Weise erfolgen:

- *system-maintained* und
- *user-maintained* MQTs.

Der “default” ist MAINTAINED BY SYSTEM (“system-maintained MQTs”), und dies heist, die MQTs können nicht über das LOAD Utility oder über INSERT, UPDATE bzw. DELETE SQL Statements manipuliert werden. Der einzige Weg, „system-maintained MQTs” zu ändern ist das (neue) **REFRESH TABLE** SQL Statement.



MQTs die als MAINTAINED BY USER (“user-maintained MQTs”) angelegt wurden, können über das LOAD Utility, INSERT, UPDATE oder DELETE Statements oder jede andere mögliche Art , z.B. über „triggers” geändert werden.

Diese Verbesserung liefert eine Menge von Funktionen die DB2 Applikationen die Möglichkeit geben, MQT's zu definieren, zu füllen und zu benutzen. Diese funktionalen Elemente beinhalten:

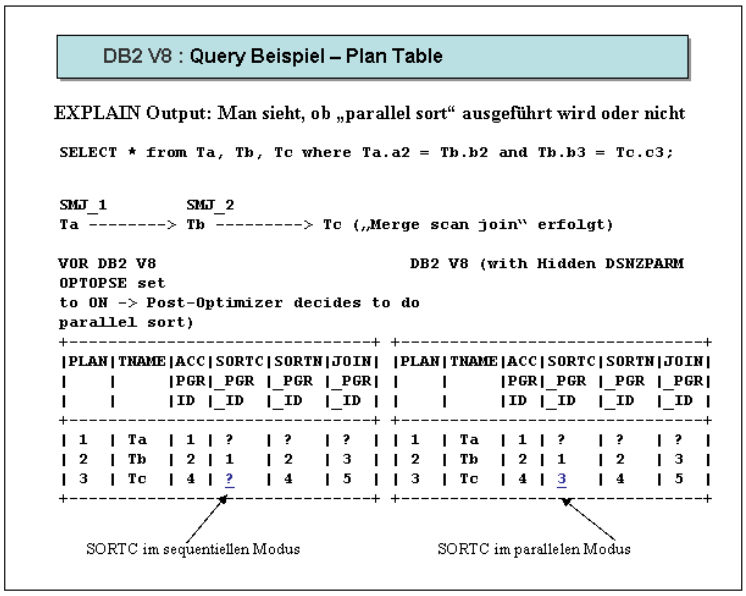
```
CREATE TABLE MQT1 AS (
 SELECT T.PDATE, T.TRANSID,
        SUM(QTY * PRICE) AS TOTVAL,
        COUNT(QTY * PRICE) AS CNT
 FROM SCNDSTAR.TRANSITEM TI, SCNDSTAR.TRANS T
 WHERE TI.TRANSID = T.TRANSID
 GROUP BY T.PDATE, T.TRANSID)
DATA INITIALLY DEFERRED
REFRESH DEFERRED
MAINTAINED BY SYSTEM
ENABLE QUERY OPTIMIZATION
IN MYDBMQT.MYTSMQT;
```

- Das erweiterte **CREATE TABLE** Statement wird zur Definition einer "materialized query table" verwendet.
- Das erweiterte **ALTER TABLE** Statement kann genutzt warden, um eine existente "base table" als MQT zu definieren. Man kann diese MQT "enablen" oder "disablen" bzw. zwischen "system-maintained" und "user-maintained" MQT-Tabellen zu wechseln. Mit dem ALTER Statement kann man aber auch eine "materialized query table" in eine "base table" umwandeln.
- Das Statement **REFRESH TABLE** dient dazu, eine "named materialized query table" zu aktualisieren. Dieses Statement die Daten in einer MQT, und führt dann einen "fullselect" für die MQT aus, um sie wieder zu füllen.
- Der **"automatic query rewrite"** Prozess ist ein Prozess, der eine Query prüft und, wenn es erforderlich ist, diese Query so umschreibt, dass sie gegen eine MQT, die aus einer "source table" erstellt wurde, laufen kann. Dieser „automatic query rewrite" führt in den moisten Fällen zu einer signifikanten Reduktion der Query Ausführungszeit.
- Das **CURRENT REFRESH AGE** Spezialregister wird genutzt, um zu prüfen, ob eine "materialized query table" mit einem bestimmten "refresh timestamp" für einen „automatic query rewrite" benutzt werden kann.
- Das **CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION** Spezialregister wird genutzt, um zu prüfen, ob „materialized query tables" (*system-maintained*, *user-maintained* oder beides) für einen „automatic query rewrite" ausgewählt wurden.

8.3 Multi-row INSERT and FETCH

Mit dieser Änderung an SQL Statements kann man ein einzelnes FETCH benutzen, um mehr al seine Datenzeile zurückzuerhalten und beim INSERT können in einem Statement mehr als eine „row" in eine Tabelle eingefügt werden. Dies reduziert die Übergabehäufigkeit der Kontrolle von DBMS an die Applikation und umgekehrt – ebenso, wie die Menge der „network trips" für "distributed requests".

8.4 Parallel sort



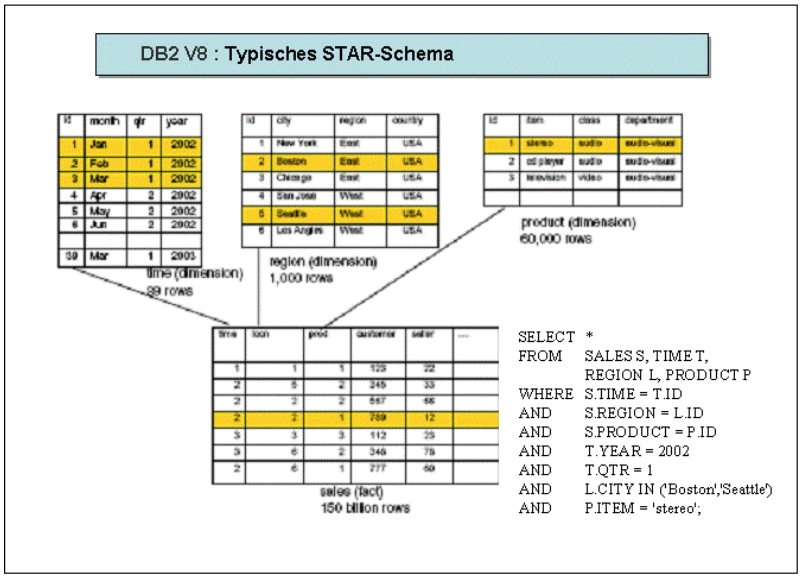
Der “sort process” kann nun mehrere Tabellen parallel sortieren.

Basieerend auf einer Entscheidung gemäss einem Kostenmodell, kann diese Parallelität von Sorts CPU Ressourcen besser nutzen und die „elapsed time vermindern.

8.5 Sparse index for star join

Die Implementierung des “star join” in DB2 UDB for z/OS muss sich potentiell mit einer grossen Menge von “work files”, speziell mit einem stark normalisierten “star schema”, das viele “snowflakes” beinhaltet und mit hohen Kosten beim Sort dieser „workfiles“ aufwartet. DB2 V8 erweitert die Nutzung der “sparse index” Funktion (eine dynamisch aufgebauten Index, der eine “range of values” bezeichnet) um optionales „data caching“ der „star join workfiles“.

Die Entscheidung diese Funktion zu nutzen wird auf Basis der Schätzung der Kosten für den Zugriffspfad getroffen.



8.6 Lange und variable lange “keys”

DB2 V8 erweitert die Unterstützung für lange “index keys”, variabel lange “index keys” und lange Prädikate. Die **maximale Länge eines “long index key” ist nun 2000 Bytes** – erweitert von 255 Bytes in DB2 V7.

Echte variable lange Schlüssel warden nun im Index unterstützt, d.h. variable lange Schlüsselfelder warden nicht auf die maximale Länge des Index Keys aufgefüllt. Die spart Speicher. Wichtiger aber ist die Tatsache, dass DB2 V8 damit „index-only“ Zugriffe auf diese Indizes zulässt. Die maximale Länge für Spalten in Prädikaten ist nun 32704 Bytes (die maximale Grösse einer VARCHAR „column“) erhöht von 255 Bytes in DB2 V7.

8.7 Backward index scan

DB2 V8 : „Backward Index Scan“

In V8 wählt DB2 einen aufsteigenden Index und kann diesen auch rückwärts durchsuchen, um so einen Sort in absteigender Sequenz zu vermeiden

In V8 wählt DB2 einen absteigenden Index und kann diesen auch rückwärts durchsuchen, um so einen Sort in aufsteigender Sequenz zu vermeiden

Um einen Index für „backward scan“ nutzen zu können gilt:

- Ein Index muss in der Folge der ORDER BY Spalten definiert sein
- Die Sequenz muss genau gegensätzlich den Angaben im ORDER BY festgelegt sein
  - Ist der IX wie folgt definiert DATE DESC, TIME ASC gilt:
    - „Forward scan“ für ORDER BY DATE DESC, TIME ASC
    - „Backward scan“ für ORDER BY DATE ASC, TIME DESC
- SORT erfolgt für:
  - ORDER BY DATE ASC, TIME ASC
  - ORDER BY DATE DESC, TIME DESC

Man muss nun keinen “ascending” und einen gleichen “descending” Index auf dieselben “table columns” definieren, nur um unterschiedliche SORTs in beide Richtungen unterstützen zu können.

8.8 Trigger Verbesserung

Immer, wenn ein **AFTER Trigger** mit einer WHEN Klausel gerufen wird, wird eine “work file“ erzeugt für „old“ und „new“ Transitionsvariable.

Diese “work file” wird immer erzeugt, auch wenn der “trigger” nicht aktiviert wird.

Diese Änderung speichert die Änderungen im Speicher, wenn es sich um einige wenige handelt, anstatt jedes Mal eine „work file“ zu erzeugen und zu löschen. Die Verbesserung gilt jetzt für AFTER und BEFORE Trigger.

8.9 Verringerte “lock contention” auf “volatile tables”

Eine **“volatile table”** ist eine Tabelle, deren Inhalte zur Laufzeit von **0 bis zu sehr grossen Zahlen** von “data rows” variieren können.

DB2 leitet oft einen “table space scan” oder einen “non-matching index scan” ein, wenn die Datenzugriffsstatistiken anzeigen, dass Tabellen klein sind – auch wenn ein “matching index access” möglich ware. Dies ist immer dann ein Problem, wenn die Tabelle zu Zeitpunkt der Erstellung von Statistiken klein oder leer ist, aber mit hohen Datenraten versehen ist, wenn die Queries laufen.

In diesem Fall sind die Statistiken nicht korrekt und dies wiederum kann DB2 veranlassen, einen falschen oder ungünstigen Zugriffspfad zu wählen. Ein “index access” kann für solche Tabellen nur wärmstens empfohlen warden.

Volatile (oder “cluster”) Tabellen werden vor allem in SAP Anwendungsumgebungen verwendet. Es sind in diesem Fall Tabellen, die Gruppen (oder “clusters”) von “rows” darstellen, die logisch zusammengehören. Innerhalb jeden „clusters” warden die “rows” jades mal in derselben Sequenz zugegriffen. „Lock contention” tritt auf, wenn DB2 unterschiedliche Zugriffspfade für unterschiedliche Anwendungen, die auf denselben “clusters” arbeiten, wählt.

DB2 V8 : „Cluster Table Concurrency“ - Beispiel

First name, Last name, Seq. # bilden den „primary key“

George Bush, Gray Davis, & Ron Gonzales bilden „cluster“

„Lock contention“ wird minimiert wenn die „rows“ in der Folge der „Sequence #“ innerhalb der jeweiligen „cluster“ zugegriffen wurde

Die Spezifikation von VOLATILE favorisiert den Index Zugriff und reduztiert die Wahrscheinlichkeit von „deadlocks“

First Name	Last Name	Sequence #	City
George	Bush	1	D.C.
George	Bush	2	Houston
George	Bush	3	Austin
Gray	Davis	1	Sacramento
Gray	Davis	2	Los Angeles
Ron	Gonzales	1	San Jose

Durch das Fehlen einer Unterstützung von „cluster tables” in DB2, müssen die User entweder systemweite Parameter ändern lassen, die alle Tabellen betreffen oder aber die Statistiken für jede derartige Tabelle ändern, um die “lock contention” zu vermindern.

“Cluster tables” warden in DB2 als “volatile tables” behandelt.
Durch Einführung eines neuen Schlüsselworts **VOLATILE** im CREATE TABLE und ALTER TABLE Statement wird dem DB2 angezeigt, welche tabellen als “volatile tables” behandelt werden sollen.
Für “volatile tables” wird “index access” gewählt, wann immer das möglich ist, ohne Rücksicht auf die Effizienz verfügbarer Indizes.
Das bedeutet, dass niemals einem “table scan” gegenüber einem ineffizienten Indexzugriff der Vorzug gegeben würde.

8.10 “Table UDF cardinality option”

Diese Änderung führt die sogenannte “cardinality option” für “user defined tables functions” ein. Der User kann damit in “user defined table functions” selbst die Schätzung des Aufwands direkt beeinflussen bevor dei Queries starten.

Die Performance Verbesserungen in derartigen Queries können vor allem in Verbindung mit den Features des “UDF block fetch“ erreicht werden . Bei dieser Option handelt es sich um eine “nonstandard SQL” Erweiterung und ist in der DB2 UDB for z/OS Implementierung enthalten.

Beispiel:

```
SELECT      *
FROM        BOOKS B,
            TABLE(CONTAINS(1,'cs') CARDINALITY MULTIPLIER 15.0) AS X1(ID),
            TABLE(CONTAINS(2,'database') CARDINALITY MULTIPLIER 2.0) AS X2(ID),
            TABLE(CONTAINS(3,'Clark') CARDINALITY MULTIPLIER 0.03) AS X3(ID)
WHERE       B.ID = X1.ID AND B.ID = X2.ID AND B.ID = X3.ID;
```

9 Data sharing

9.1 *“CF lock propagation reduction”*

Diese Änderung erlaubt es in einem “data sharing environment”, hierarchische **(“parent”) L-locks** lokal zu authorisieren, ohne das “global contention processing” aufrufen zu müssen.

Dabei wird der „locking overhead” entscheidend reduziert. Dadurch wird in DB2 die “data sharing performance” verbessert.

Der Performance Vorteil hängt dabei von Faktoren wie “commit interval”, “thread reuse”, Anzahl der in einem “commit interval” zugegriffener Tabellen, ist das „SQL processing“ „read-only“ oder „update“ usw.

9.2 *Andere Performance und Nutzungsverbesserungen*

- Reduktion der “overhead”-Kosten für “data sharing” Verarbeitung
- Batch-“updates” für Splits von “index pages”
- Verbessertes LPL Recovery
- Auflösung von “indoubt units” beim Recovery im “Restart light”
- Änderung des „default“ auf die IMMEDWRITE BIND Option
- Änderung des -DISPLAY GROUPBUFFERPOOL Output

10 Installation und migration

10.1 Voraussetzungen

Die DB2 V8 Umgebung ist ausschliesslich für z/OS Platform en verfügbar (DB2 auf zSeries Prozessoren mit z/OS V1R3 oder höher). Einige Funktionen erfordern eben ein höheres Release von z/OS.

10.2 Migration

Die Migration ist ausschliesslich von DB2 UDB for OS/390 und z/OS Version 7 Subsystemen möglich. Das Migration-SPE muss installiert und gestartet sein. Der Migrationsprozess besteht aus drei verschiedenen Schritten bzw. Phasen:

1. **Compatibility Mode (CM)**
   Die erste Phase. Während dieser Phase kann der User alle Tests, die nötig sind , um festzustellen, dass alle Applikationen ohne Probleme in der neuen Umgebung laufen, durchführen. „Fall back" auf DB2 V7 sind im Falle von Problemen möglich.
2. **Enable New Function Mode (ENFM)**
   Während dieser (hoffentlich kurzen) Zwiten Phase konvertiert der User den DB2 Katalog und das "directory" auf das neue Format. Getätigt wird dies über die "on-line Reorg" Ausführung. . „Fall back" auf DB2 V7 ist nicht mehr möglich.
3. **New Function Mode (NFM)**
   Dies ist die Ziel- und damit letzte Pase in Richtung DB2 V8. Alle neuen Funktionen von DB2 V8 sind ohne Einschränkung verfügbar.

11 Literaturhinweise

IBM Redbooks

IBM DB2 V8 Manuals