Oracle Database 12c Features für Datawarehouse

Frank Schneede
Oracle Deutschland B. V. & Co.KG
Hannover

Schlüsselworte
12c, Datawarehouse, Partitioning, Online Maintenance, Indexing, Statistiken, Optimizer

Einleitung
Seit Ende Juni 2013 steht die neue Oracle Database 12c, die bereits auf der Oracle Open World 2012 angekündigt worden ist, Anwendern offiziell zur Verfügung. Neben der erstmalig eingeführten Multitenant Architektur bietet auch diese Datenbank Version eine Vielzahl neuer Funktionen und Verbesserungen in nahezu allen Einsatzbereichen der Oracle Datenbank.
Ein gewissermaßen klassisches Einsatzgebiet der Oracle Datenbank ist die Verwendung als Plattform für große, unternehmenskritische Datawarehouses. Durch stark wachsende Datenvolumina und steigende Anforderungen an die Verfügbarkeit der Informationen werden Funktionen wie Partitioning oder auch Online Maintenance immer wichtiger. Zugleich werden Performanceaspekte angesichts der wachsenden Informationsflut immer wichtiger.
Dieser Vortrag stellt im Wesentlichen die Neuigkeiten in der Arbeit mit partitionierten Objekten vor und gibt zum Abschluß einen kurzen Überblick über andere im Datawarehouse Umfeld wichtige New Features der Oracle Database 12c.

Partitioning – eine lange Erfolgsgeschichte
Mit der Datenbankversion 8 wurde die Partitioning Option im Jahr 1997 erstmals eingeführt. Mit der zu der Zeit einzigen verfügbaren Strategie der *Range*-Partitionierung konnten große Datawarehouse Tabellen zum Beispiel nach der Dimension Zeit aufgeteilt werden. Einfache Maintenance-Aufgaben waren möglich und *statisches Partition Pruning* sorgte dafür, die Ergebnismenge bei Abfragen so zu verringern, dass die Antwortzeiten auch bei insgesamt sehr großen Datenmengen erträglich blieben.
Mit der 1999 eingeführten *Hash*-Partitionierung konnten *Partition-Wise Joins* ausgeführt werden, was dazu diente, Kreuzprodukte in Abfragen zu vermeiden bzw. deren Ergebnismenge deutlich zu verkleinern. *Dynamisches Partition Pruning* half ebenfalls, unnötige Daten zur Laufzeit der Query von der Ergebnisermittlung auszuschließen. Im Laufe der folgenden Datenbankversionen wurden weitere neue Partitionierungsstrategien eingeführt, die dabei halfen, Geschäftsabläufe genauer abzubilden und Wartungsarbeiten an partitionierten Objekten zu vereinfachen.
Mit der Datenbankversion 11g kam 2007 die Möglichkeit hinzu, Master/Detail Strukturen über *Reference*-Partitionierung abzubilden. Gleichzeitig wurden weitere Verfahren der zusammengesetzten Partitionierung (*Range-Range*, *List-List*, *List-Hash*, *List-Range*) eingeführt, um den Anforderungen der Kunden noch besser entsprechen zu können. Mit dem *Partition Advisor* haben die Kunden seit Version 11g die Möglichkeit, sich durch Analyse eines Workloads geeignete Partitionierungsstrategien vorschlagen zu lassen. In Version 11g minimieren neue Möglichkeiten wie *Interval*-Partitionierung, das im Grunde die automatisierte Partitionspflege von *Range*-Partitionen übernimmt, den Administrationsaufwand für partitionierte Objekte. Steigende Anforderungen an die Verfügbarkeit von Datawarehouses können mittlerweile durch effiziente online Operationen erfüllt werden. Auch die inkrementelle Fortschreibung von Statistikinformationen hilft seit 11g, Wartungsfenster im Datawarehouse fast vollständig zu vermeiden.

Partitioning in Oracle Database 12c – Gutes noch besser machen!
Die größte Herausforderung, vor der Datawarehouse Designer und Administratoren stehen, bilden stark wachsenden Datenmengen. Im Laufe des Lebenszyklus eines DWH sind wachsende Datenmenge die Hauptursache dafür, dass ohne administrative Eingriffe die Abfrageperformance stetig abnimmt. Mit Partitioning in Oracle Database 12c werden die bereits in den Vorgängerversionen eingeschlagenen Wege fortgeführt und :

- Bessere Abbildung der Geschäftsabläufe durch neue Partitionierungsstrategien
- Erhaltung oder Steigerung der Abfrageperformance durch geschickte Datenmodellierung und optimierten Datenzugriff
- Verkleinerung des notwendigen Zeitfensters für Wartungsarbeiten durch neue online DDL Befehle

Bessere Abbildung von Geschäftsabläufen
Die in Oracle Database 11g *Interval* Partitionierung automatisiert das Anlegen von Partitionen mit dem Einfügen von Daten in die betreffende Partition, handgeschriebene Skripte zur Partitionspflege fallen weg. Die ebenfalls in 11g neu vorgestellte Reference Partitionierung ermöglicht anhand der Foreign Key (FK) Beziehung eine identische Partitionierung für Parent / Child Tabellen, ohne den Partitionierungsschlüssel redundant halten zu müssen. Mit Oracle Database 12c ist es nun möglich, diese beiden Partitionierungsformen zu kombinieren. In dem in Abbildung 1 dargestellten Beispiel ist die Tabelle ORDERS nach der zeitlichen Dimension *Interval* partitioniert. Die über FK Beziehungen verbundenen Child Tabellen sind gleich partitioniert und werden erweitert, sobald Daten in die korrespondierenden Partitionen eingefügt werden.

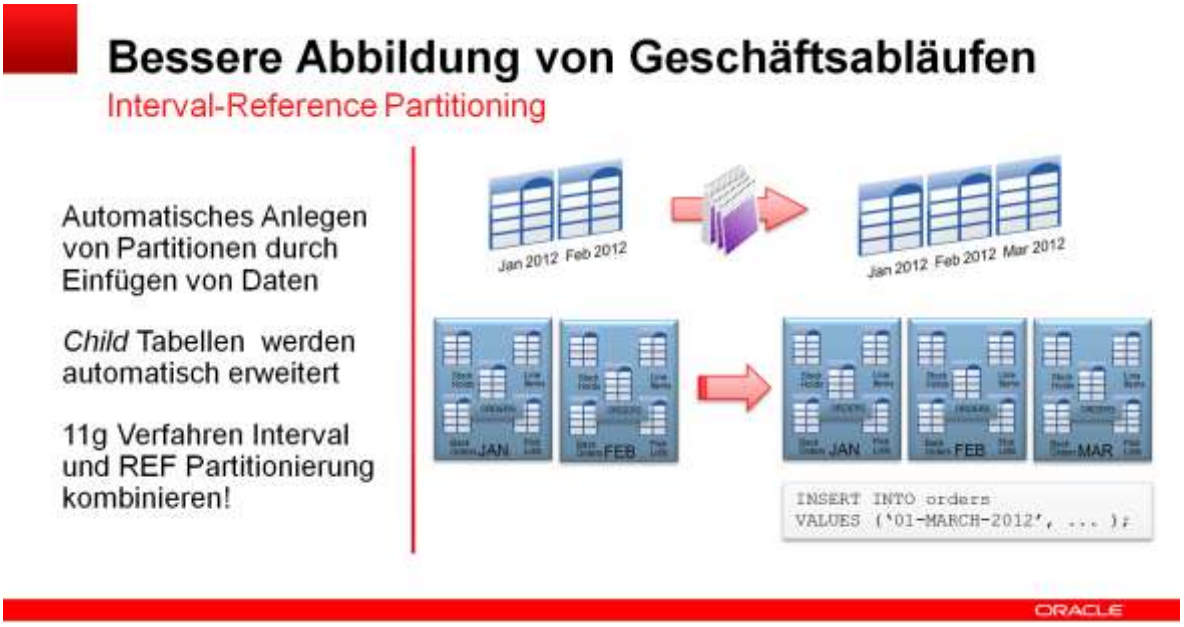

Abb. 1: Interval-Reference Partitionierung

Die Namensgebung folgt dabei der direkt referenzierten Partition, wenn die Partition der Parent Tabelle p100 heißt, dann heißt auch die Child Partition p100. Das Anlegen einer neuen Partition einer Child Tabelle erfolgt nur, wenn diese auch mit Daten versorgt wird, wie das folgende Listing zeigt:

```
SQL> CREATE TABLE intref_ptab(pk_id NUMBER NOT NULL
  2                          ,col2 VARCHAR2(100)
  3                          ,CONSTRAINT pk_intref_p PRIMARY KEY(pk_id))
```

```
  4   PARTITION BY RANGE(pk_id) INTERVAL (10)
  5  (PARTITION p1 VALUES LESS THAN (10));

SQL> CREATE TABLE intref_c1tab(pk_id NUMBER NOT NULL
  2                           ,col2 VARCHAR2(100)
  3                           ,fk_col NUMBER NOT NULL
  4                           ,CONSTRAINT pk_intref_c1 PRIMARY KEY(pk_id)
  5                           ,CONSTRAINT fk_intref_c1 FOREIGN KEY(fk_col) REFERENCES
intref_ptab(pk_id) ON DELETE CASCADE)
  6   PARTITION BY REFERENCE(fk_intref_c1);

SQL> CREATE TABLE intref_c2tab(pk_id NUMBER NOT NULL
  2                           ,col2 VARCHAR2(100)
  3                           ,fk_col NUMBER NOT NULL
  4                           ,CONSTRAINT pk_intref_c2 PRIMARY KEY(pk_id)
  5                           ,CONSTRAINT fk_intref_c2 FOREIGN KEY(fk_col) REFERENCES
intref_ptab(pk_id) ON DELETE CASCADE)
  6   PARTITION BY REFERENCE(fk_intref_c2);

SQL> INSERT INTO intref_ptab VALUES(10, 'Satz 1');
SQL> INSERT INTO intref_ptab VALUES(20, 'Satz 2');
SQL> INSERT INTO intref_ptab VALUES(30, 'Satz fuer TRUNC');
SQL> COMMIT;

SQL> SELECT table_name
  2  ,      partition_position
  3  ,      high_value
  4  ,      interval
  5  FROM USER_TAB_PARTITIONS
  6  WHERE table_name IN ('INTREF_PTAB', 'INTREF_C1TAB', 'INTREF_C2TAB')
  7  ORDER BY 1, 2;

TABLE_NAME       PARTITION_POSITION HIGH_VALUE INT
---------------- ------------------ ---------- ----
INTREF_C1TAB                     1            NO
INTREF_C2TAB                     1            NO
INTREF_PTAB                      1 10         NO
INTREF_PTAB                      2 20         YES
INTREF_PTAB                      3 30         YES
INTREF_PTAB                      4 40         YES

SQL> INSERT INTO intref_c1tab VALUES(10, 'Satz 1', 10);
SQL> INSERT INTO intref_c2tab VALUES(10, 'Satz 1', 10);
SQL> INSERT INTO intref_c2tab VALUES(20, 'Satz fuer TRUNC', 30);
SQL> COMMIT;

SQL> SELECT table_name
  2  ,      partition_position
  3  ,      high_value
  4  ,      interval
  5  FROM USER_TAB_PARTITIONS
  6  WHERE table_name IN ('INTREF_PTAB', 'INTREF_C1TAB', 'INTREF_C2TAB')
  7  ORDER BY 1, 2;

TABLE_NAME       PARTITION_POSITION HIGH_VALUE INT
---------------- ------------------ ---------- ----
INTREF_C1TAB                     1            NO
INTREF_C1TAB                     2            YES
INTREF_C2TAB                     1            NO
INTREF_C2TAB                     2            YES
INTREF_C2TAB                     3            YES
INTREF_PTAB                      1 10         NO
INTREF_PTAB                      2 20         YES
INTREF_PTAB                      3 30         YES
INTREF_PTAB                      4 40         YES
```

Besonders interessant bei Reference partitionierten Tabellen ist die in 12c geschaffene Möglichkeit, Operationen wie `TRUNCATE` oder `EXCHANGE` als sogenannte *Atomare Transaktion*, also in einem Schritt durchzuführen. Bisher konnte zum Beispiel ein `ALTER TABLE ... TRUNCATE`

PARTITION nur in mehreren Schritten unter Berücksichtigung der Abhängigkeiten über geeignetes Skripting ausgeführt werden. Eine Parent Partition konnte nur entfernt werden, sobald keine Child Partitionen mehr vorhanden sind. Mit Oracle Database 12c erfolgt dieser Schritt mit nur einem Kommando:

```
SQL> select * from intref_ptab;

     PK_ID COL2
---------- -----------------
        10 Satz 1
        20 Satz 2
        30 Satz fuer TRUNC

SQL> select * from intref_c1tab;

     PK_ID COL2                  FK_COL
---------- ----------------- ----------
        10 Satz 1                    10
        20 Satz fuer TRUNC           30

SQL> select * from intref_c2tab;

     PK_ID COL2                  FK_COL
---------- ----------------- ----------
        10 Satz 1                    10
        20 Satz fuer TRUNC           30

SQL> ALTER TABLE intref_ptab TRUNCATE PARTITION FOR(30) CASCADE UPDATE INDEXES;

SQL> select * from intref_ptab;

     PK_ID COL2
---------- -----------------
        10 Satz 1
        20 Satz 2

SQL> select * from intref_c1tab;

     PK_ID COL2                  FK_COL
---------- ----------------- ----------
        10 Satz 1                    10

SQL> select * from intref_c2tab;

     PK_ID COL2                  FK_COL
---------- ----------------- ----------
        10 Satz 1                    10
```

Die Operation TRUNCATE PARTITION wirkt auf alle referenzierten Partitionen – unabhängig von deren Struktur und kann auf jeder Ebene der REF partitionierten Tabelle ausgeführt werden. Voraussetzung für das kaskadierte Truncate ist, dass die Foreign Key Beziehungen mit ON DELETE CASCADE (siehe 1. Listing) definiert sind. Übrigens funktioniert die Operation auch für nicht-partitionierte Objekte, wobei die Abhängigkeiten durch Deaktivieren der FK Constraints unterbrochen werden können.

Analog funktioniert die Operation ALTER TABLE ... EXCHANGE PARTITION als *Atomare Transaktion*, wie Abbildung 2 schematisch darstellt.

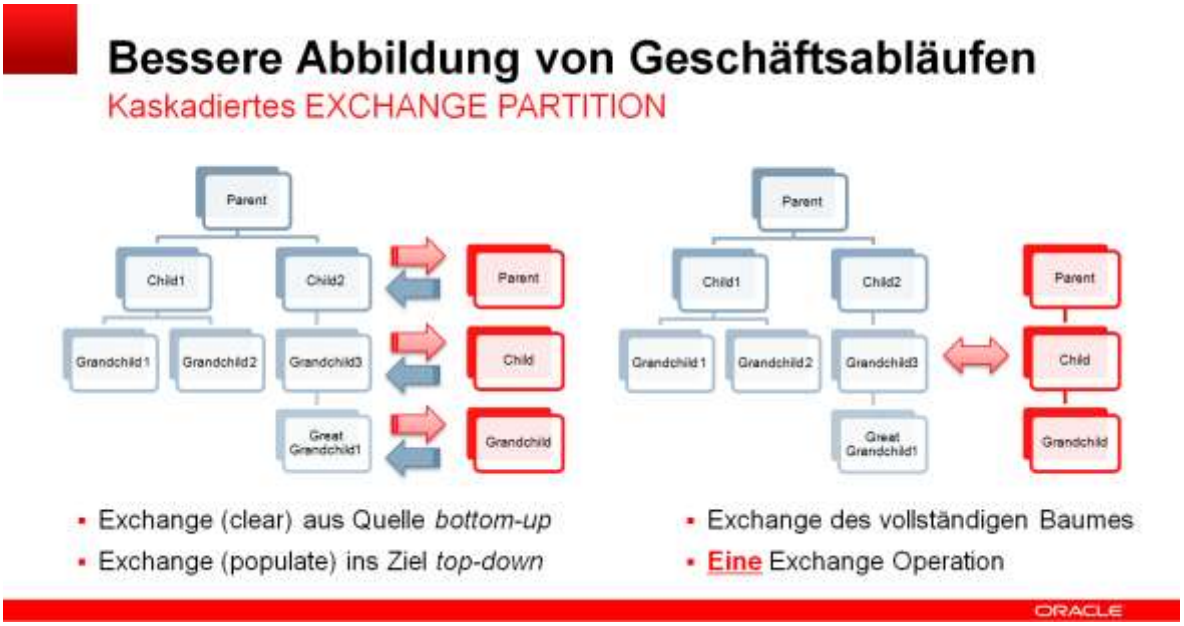

Abb. 2: Atomare Transaktion: kaskadiertes ALTER TABLE ... EXCHANGE PARTITION

Voraussetzung für kaskadiertes `EXCHANGE PARTITION` ist, dass die Strukturen der Quell- und Zieltabelle übereinstimmen müssen und es eine eindeutige Zuordnung von Child zu Parent Tabellen gibt.

Effiziente Verwaltung von Tabellen mit online Operationen
Durch die 7*24 Stunden Verfügbarkeit, die für viele Datawarehouses mittlerweile gefordert wird, kommt Wartungsoperationen, die online ausgeführt werden können, besonders hohe Bedeutung zu. Mit dem Befehl `ALTER TABLE ... MOVE PARTITION ONLINE` können Partitionen ohne Betriebseinschränkung verschoben und damit zum Beispiel komprimiert werden. Die während der Ausführungszeit des Statements auflaufenden DML Operationen werden journalisiert und nachgefahren, was natürlich eine entsprechend dimensionierte Plattenplatzreserve für die Journale voraussetzt. Aus diesem Grunde ist es sinnvoll, ein Verschieben von Partitionen primär in Zeiten geringer Last zu planen und durchzuführen.

Eine weitere mit Oracle Database 12c eingeführte Verbesserung ist die Möglichkeit, mehrere Partitionen in einem einzelnen Kommando zu bearbeiten, zum Beispiel die Zusammenfassung dreier Monatspartitionen zu einem Quartal. Abbildung 3 zeigt das Schema dieses Befehls.

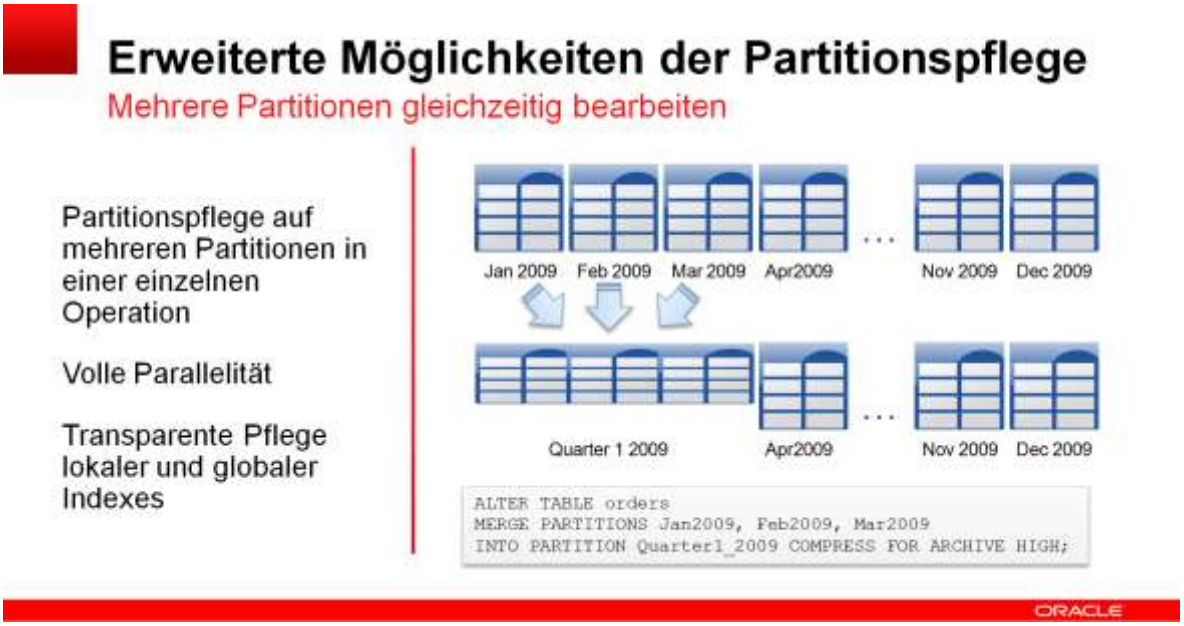

Abb. 3: Paralleles ALTER TABLE ... MERGE PARTITIONS

Gleiches gilt prinzipiell für alle DDL Operationen auf Partitionen, wie die folgenden Syntxbeispiele für `MERGE PARTITIONS` zeigen:

`MERGE PARTITIONS` mit Angabe der Reihenfolge der Partitionen:

```
SQL> ALTER TABLE pt MERGE PARTITIONS FOR(5), FOR(15), FOR(25) INTO PARTITION p30;
```

`MERGE PARTITIONS` mit Angabe eines Bereichs von Partitionen:

```
SQL> ALTER TABLE pt MERGE PARTITIONS p10 TO p30 INTO PARTITION p30;
```

Eine weitere Neuerung ist die asynchrone Pflege des Global Index auf einer partitionierten Tabelle. Vor 12c wurden Indexes nach `DROP` oder `TRUNCATE PARTITION` invalidiert und konnten nicht mehr genutzt werden. In der aktuellen Datenbankversion sind die betroffenen Partitionen bekannt und werden zur Laufzeit gefiltert, d. h. der globale Index bleibt benutztbar. Ein weiterer Vorteil dieses Features ist, dass `TRUNCATE` und `DROP` wesentlich schneller ablaufen, da nur Index-Metadatan betroffen sind. Die Pflege des Global Index erfolgt Scheduler-gesteuert mittels `ALTER INDEX REBUILD` oder `ALTER IDEX COALESCE`.

Partielle lokale und globale Indexes runden die neuen Partitioning Features in Oracle Database 12c ab. Mit Partiellen Indexes können auf einer Untermenge der Tabellenpartitionen Indexes definiert werden; die damit verbundene Flexibilität kann zum Beispiel genutzt werden, um teure Indexpflegeoperationen bei massenhaften DML auf bestimmten Partitionen zu vermeiden. Das folgende Beispiel zeigt, dass die Tabelle mit einer standardmäßigen Indizierungstrategie (`INDEXING OFF`) angelegt wird, die für automatisch angelegte Partitionen gilt:

```
SQL> CREATE TABLE idxpart_ptab (col1 NUMBER, col2 NUMBER, col3 NUMBER, col4 NUMBER)
  2  INDEXING OFF
  3  PARTITION BY RANGE (col1) INTERVAL (1000)
  4  (PARTITION p100 VALUES LESS THAN (101) INDEXING ON,
  5   PARTITION p200 VALUES LESS THAN (201) INDEXING ON,
  6   PARTITION p300 VALUES LESS THAN (301) INDEXING ON);
```

```
SQL> INSERT INTO idxpart_ptab VALUES(400, 500, 600, 700);
SQL> commit;

SQL> SELECT partition_name
  2  ,      high_value
  3  ,      indexing
  4  FROM user_tab_partitions
  5  WHERE table_name = 'IDXPART_PTAB';

PARTITION_NAME HIGH_VALUE INDEXING
-------------- ---------- -----------
P100           101        ON
P200           201        ON
P300           301        ON
SYS_P961       1301       OFF
```

Wenn nun, wie im folgenden Listing gezeigt, lokale und globale Indexes partiell und auf herkömmliche Weise angelegt werden, wird die Idee der partiellen Indizierung erkennbar:

```
SQL> CREATE INDEX idx_loc_part_ptab ON idxpart_ptab(col1) LOCAL INDEXING PARTIAL;
SQL> CREATE INDEX idx_loc_ptab ON idxpart_ptab(col4) LOCAL;
SQL> CREATE INDEX idx_glob_part_ptab ON idxpart_ptab(col2) INDEXING PARTIAL;
SQL> CREATE INDEX idx_glob_ptab ON idxpart_ptab(col3);

SQL> SELECT index_name
  2  ,      partition_name
  3  ,      status
  4  ,      NULL
  5  FROM user_ind_partitions
  6  WHERE index_name IN ('IDX_LOC_PART_PTAB', 'IDX_LOC_PTAB')
  7  UNION ALL
  8  SELECT index_name
  9  ,      indexing
 10  ,      status
 11  ,      orphaned_entries
 12  FROM user_indexes
 13  WHERE index_name IN ('IDX_GLOB_PART_PTAB', 'IDX_GLOB_PTAB');

INDEX_NAME          PARTITION_NAME STATUS    ORPHANED
------------------- -------------- --------- ----------
IDX_LOC_PTAB        SYS_P963       USABLE
IDX_LOC_PTAB        P300           USABLE
IDX_LOC_PTAB        P200           USABLE
IDX_LOC_PTAB        P100           USABLE
IDX_LOC_PART_PTAB   SYS_P962       UNUSABLE
IDX_LOC_PART_PTAB   P300           USABLE
IDX_LOC_PART_PTAB   P200           USABLE
IDX_LOC_PART_PTAB   P100           USABLE
IDX_GLOB_PTAB       FULL           VALID     NO
IDX_GLOB_PART_PTAB  PARTIAL        VALID     NO
```

Der lokale Index `IDX_LOC_PTAB` hat den Status `USABLE` für alle Partitionen, hier greift als die default `INDEXING OFF` Klausel nicht. Anders bei dem partiellen Index `IDX_LOC_PART_PTAB`, der für die automatisch angelegte Partition `SYS_P962 UNUSABLE` ist, wie es die default `INDEXING OFF` Klausel vorsieht. Der herkömmlich erzeugte globale Index `IDX_GLOB_PTAB` geht über alle Partitionen, während der partiell angelegte globale Index `IDX_GLOB_PART_PTAB` nur über die Partitionen geht, die mit `INDEXING ON` definiert sind.

Partielle lokale und globale Indexes sind also eine sehr nützliche Ergänzung zu einer vollständigen Indizierung von Datawarehouse Tabellen, wie Abblidung 4 anschaulich zeigt. Partielle Indizierung steht nur für Indexes zur Verfügung, die keine Eindeutigkeit erzwingen.

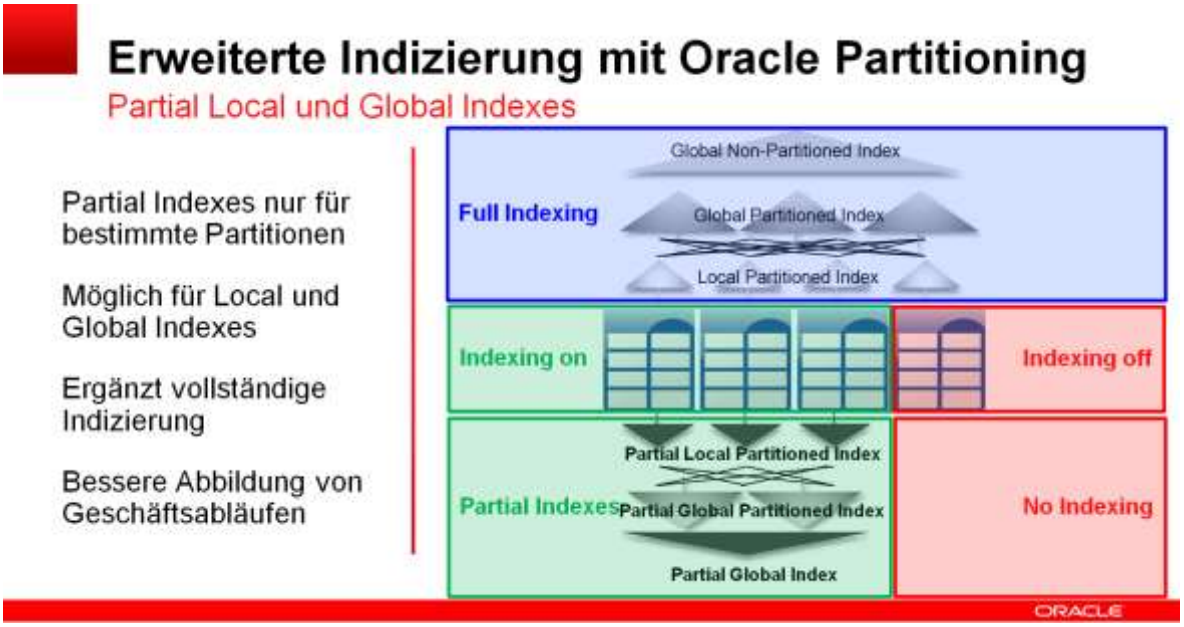

Abb. 4: Partial Local und Global Indexes

Statistiken in Oracle Database 12c
Eine ganze Reihe neuer Funktionen mit mehr Automatismen und Analyseverfahren helfen in Oracle Database 12c, den Optimizer in die Lage zu versetzen, die bestmöglichen Ausführungspläne für die Abarbeitung von SQL Abfragen zu ermitteln. Die neuen Features, von denen hier die fett gedruckten kurz erläutert werden, sind:

- Neue Histogramm Typen
- **Online Statistikberechnung**
- Statistiken auf Session-Ebene für Global Temporary Tables (GTT)
- **Erweiterung inkrementeller Statistiken**
- Gleichzeitge Statistikberechnung für mehrere Objekt
- Automatische Ermittlung von Column Groups
- Auswertungen über die Statistikberechnung

In ETL Strecken nimmt die Berechnung von Statistiken für den Optimizer häufig einen großen Raum ein. Durch die automatische Berechnung der Statistiken während *direct path loads* (CTAS, IAS) stehen diese sofort nach Abschluß des Ladelaufes zur Verfügung. Ein zusätzlicher Tabellenscan ist nicht mehr notwendig!

Inkrementelle Statistiken sind in Oracle Database 12c erweitert worden, dass globale Statistiken auf Basis der Statistiken auf Partitionsebene errechnet werden. Zur Berechnung der Kardinalität wird eine Synopse aus den bisherigen Analysen erzeugt, die Synopse wird in Tablespace `SYSAUX` abgelegt. Aus der Statistikberechnung für neu hinzukommende Partitionen und der Synopse wird dann die Kardinalität ermittelt. In Oracle Database 11g wurde die Statistik auch für alte Partitionen neu berechnet, wenn DML auf diesen Partitionen ausgeführt wurden. Die neue Präferenz `INCREMENTAL_STALENESS` dient dazu, die Neuberechnung zu verhindern (`USE_LOCKED_STATS`) oder erst ab einer Veränderung von 10% der Daten (`USE_STALE_PERCENT`) anzustoßen. Um inkrementelle Statistiken nutzen zu können, muss die

entsprechende Präferenz mit `DBMS_STATS.SET_TABLE_PREFS(<owner>, <table>, 'INCREMENTAL','TRUE')` gesetzt werden.

Fazit
Oracle Database 12c liefert mit den neuen Features in den Bereichen Partitioning und Optimizer Statistiken und durch viele andere Detailerweiterungen interessante Funktionen, die besonders für Datawarehouses großes Potential bieten, dem DBA das Leben erheblich zu vereinfachen. Besonders interessant sind in meinen Augen die Handlingsverbesserungen bei Parent / Child Tabellen.

Weiterführende Informationen:

- http://www.oracle.com/technetwork/database/bi-datawarehousing/dbbi-tech-info-part-100980.html
- http://www.oracle.com/technetwork/database/options/partitioning/partitioning-wp-12c-1896137.pdf
- http://www.oracle.com/technetwork/database/focus-areas/bi-datawarehousing/dbbi-tech-info-optmztn-092214.html
- http://www.oracle.com/technetwork/database/bi-datawarehousing/twp-optimizer-with-oracledb-12c-1963236.pdf
- http://www.oracle.com/technetwork/database/bi-datawarehousing/twp-statistics-concepts-12c-1963871.pdf
- http://blogs.oracle.com/optimizer

Kontaktadresse:
Frank Schneede
Oracle Deutschland B. V. & Co. KG
Thurnithistr. 2
D-30519 Hannover

Telefon:	+49 (0) 511 95787 - 250
Fax:	+49 (0) 511 95787 - 100
E-Mail	frank.schneede@oracle.com
Internet:	www.oracle.com