Fachhochschule Nordwestschweiz
Hochschule für Technik

dbarc Lab 5:
Speicherstruktur von Tabellen

In den folgenden Übungen geht es darum, die Funktionsweise von Partitioned Tables, Full Table Scans und Table Compression zu verstehen. Alle Aufgaben kannst Du unter Deinem persönlichen User ausführen.

Hinweis: Wenn Du mit SQL Developer arbeitest und die Execution Plans mit «Autotrace» anzeigen möchtest, musst Du dem eigenen User die Rolle SELECT_CATALOG_ROLE zuweisen.

1. Partitionierung

RANGE Partitioning

Erstelle in Deinem eigenen Schema eine neue Tabelle CUSTOMERS. Sie hat die gleichen Attribute und Datentypen wie die Tabelle im Beispielschema, ist jedoch partitioniert nach Geburtsjahren:

```
CREATE TABLE customers
( id NUMBER(8) NOT NULL
, first_name VARCHAR2(30 BYTE) NOT NULL
, last_name VARCHAR2(30 BYTE) NOT NULL
, date_of_birth DATE NOT NULL
, title VARCHAR2(10 BYTE)
, gender VARCHAR2(1 BYTE)
, marital_status VARCHAR2(10 BYTE)
, member_flag VARCHAR2(1 BYTE)
, active_flag VARCHAR2(1 BYTE)
, email_address VARCHAR2(100 BYTE)
, language_code VARCHAR2(2 BYTE)
)
PARTITION BY RANGE(date_of_birth)
   (PARTITION p_gen_silent VALUES LESS THAN (TO_DATE('01.01.1946', 'dd.mm.yyyy'))
   ,PARTITION p_gen_boomer VALUES LESS THAN (TO_DATE('01.01.1965', 'dd.mm.yyyy'))
   ,PARTITION p_gen_x      VALUES LESS THAN (TO_DATE('01.01.1981', 'dd.mm.yyyy'))
   ,PARTITION p_gen_y      VALUES LESS THAN (TO_DATE('01.01.1997', 'dd.mm.yyyy'))
   ,PARTITION p_gen_z      VALUES LESS THAN (TO_DATE('01.01.2011', 'dd.mm.yyyy'))
   ,PARTITION p_gen_alpha  VALUES LESS THAN (TO_DATE('01.01.2026', 'dd.mm.yyyy')));
```

Fülle die Tabelle mit Daten aus dem Beispielschema:

```
INSERT INTO customers
SELECT * FROM dbarc_base.customers;

COMMIT;
```

Für die nachfolgenden Auswertungen ist es wichtig, dass wir für die Tabelle aktuelle Optimizer-Statistiken haben (Theorie dazu folgt später). Die Statistiken werden normalerweise automatisch (jeweils abends um 22:00) berechnet. Für unsere Tests müssen wir sie manuell neu berechnen lassen. Dazu kann folgende Prozedur aufgerufen werden:

```
EXECUTE dbms_stats.gather_table_stats(USER, 'CUSTOMERS');
```

Überprüfe mit folgender Abfrage auf den Data Dictionary, wie die Daten auf die unterschiedlichen Partitionen verteilt sind.

```
SELECT partition_position, partition_name, num_rows, high_value
  FROM user_tab_partitions
 WHERE table_name = 'CUSTOMERS';
```

Abfragen auf partitionierte Tabelle (Partition Pruning)

Schreibe eine SQL-Abfrage, welche die Kunden selektiert, die im gleichen Jahr wie Du geboren sind.

- Versuche die Abfrage so zu formulieren, dass Partition Pruning verwendet wird
- Überprüfe mit dem Execution Plan, welche Partition(en) für die Abfrage gelesen werden
- Führe die gleiche Abfrage für andere Geburtsjahre aus (z.B. Deiner Eltern oder Grosseltern) und überprüfe die Execution Plans.

Partitionierungsstrategie ändern auf LIST Partitioning

Die Partitionierungsstrategie einer Tabelle kann mittels ALTER TABLE ... MODIFY PARTITION auch nachträglich geändert werden. In unserem Beispiel möchten wir die Tabelle CUSTOMERS nach Geschlecht partitionieren:

```
ALTER TABLE customers
MODIFY PARTITION BY LIST(gender)
   (PARTITION p_male   VALUES ('M')
   ,PARTITION p_female VALUES ('F'));
```

- Das SQL-Statement führt zu einem Fehler. Was muss geändert werden, um die Tabelle gemäss Vorgaben zu partitionieren?
- Korrigiere den Fehler und führe den ALTER TABLE Befehl nochmals aus.
- Berechne die Optimizer-Statistiken neu und überprüfe die Datenverteilung auf die einzelnen Partitionen.
- Welche Datensätze haben zum Fehler beim ersten Versuch geführt?

INTERVAL Partitioning

Erstelle in Deinem eigenen Schema eine neue Tabelle ORDERS. Sie hat die gleichen Attribute und Datentypen wie die Tabelle im Beispielschema, ist jedoch nach ORDER_DATE partitioniert. Dabei wird INTERVAL Partitioning verwendet, um jeweils automatisch eine Partition pro Monat zu erstellen.

Erstelle die Tabelle mit folgendem SQL-Statement:

```
CREATE TABLE orders
( id NUMBER(8) NOT NULL
, cust_id NUMBER(8) NOT NULL
, order_date DATE NOT NULL
)
PARTITION BY RANGE(order_date)
INTERVAL(NUMTOYMINTERVAL(1, 'MONTH'))
(PARTITION p_old_data VALUES LESS THAN (TO_DATE('01.01.2020', 'dd.mm.yyyy')));
```

Überprüfe, welche Partitionen für die Tabelle ORDERS vorhanden sind:

```
SELECT partition_position, partition_name, num_rows, high_value
  FROM user_tab_partitions
 WHERE table_name = 'ORDERS';
```

Füge die Bestellungen des Jahres 2020 in die Tabelle ein und berechne die Optimizer-Statistiken auf der Tabelle ORDERS:

```
INSERT INTO orders
SELECT * FROM dbarc_base.orders
 WHERE order_date BETWEEN TO_DATE('01.01.2020', 'dd.mm.yyyy')
                      AND TO_DATE('31.12.2020', 'dd.mm.yyyy');

COMMIT;

EXECUTE dbms_stats.gather_table_stats(USER, 'ORDERS');
```

Überprüfe nochmals, welche Partitionen nun vorhanden sind.

Wiederhole diese Schritte für die Jahre 2021, 2022 und 2023. Am Ende sollten 38 Partitionen vorhanden sein.

2. Full Table Scan

Erstelle in Deinem eigenen Schema eine neue Tabelle FULL_TABLE_SCAN. Da die Tabelle ziemlich gross ist, dauert der Vorgang ca. 1 Minute. Die Optimizer-Statistiken werden bei CREATE TABLE ... AS SELECT ... automatisch erstellt.

```
CREATE TABLE full_table_scan AS
SELECT oi.order_id
     , oi.line_no
     , o.order_date
     , oi.delivery_date
     , c.title
     , c.first_name
     , c.last_name
     , c.email_address
     , p.prod_name
     , p.prod_desc
     , p.prod_category
     , oi.quantity
  FROM dbarc_base.customers c
  JOIN dbarc_base.orders o ON (o.cust_id = c.id)
  JOIN dbarc_base.order_items oi ON (oi.order_id = o.id)
  JOIN dbarc_base.products p ON (p.id = oi.prod_id);
```

Führe die folgenden beiden SQL-Abfragen aus und überprüfe die Execution Plans. Notiere folgende Informationen:

- Ausführungszeit in Sekunden
- Kosten des Execution Plans

Werden die Ausführungszeiten bei mehrmaliger Ausführung kürzer?

```
SELECT COUNT(*) FROM full_table_scan;
SELECT COUNT(*) FROM full_table_scan WHERE order_date >= TO_DATE('01.01.2023', 'dd.mm.yyyy');
```

Table Compression

Komprimiere die Tabelle FULL_TABLE_SCAN mit folgendem Befehl und berechne die Optimizer-Statistiken neu:

```
ALTER TABLE full_table_scan MOVE COMPRESS;

EXECUTE dbms_stats.gather_table_stats(USER, 'FULL_TABLE_SCAN');
```

Führe die beiden SQL-Abfragen erneut aus und notiere Ausführungszeiten und Kosten der Execution Plans. Was fällt Dir auf?

Table Partitioning

Teile die Tabelle FULL_TABLE_SCAN in Jahrespartitionen auf und berechne die Optimizer-Statistiken neu:

```
ALTER TABLE full_table_scan
MODIFY PARTITION BY RANGE(order_date)
INTERVAL(NUMTOYMINTERVAL(1, 'YEAR'))
(PARTITION p_old_data VALUES LESS THAN (TO_DATE('01.01.2020', 'dd.mm.yyyy')));

EXECUTE dbms_stats.gather_table_stats(USER, 'FULL_TABLE_SCAN');
```

Führe die beiden SQL-Abfragen erneut aus und notiere Ausführungszeiten und Kosten der Execution Plans. Was fällt Dir auf?