Merge und andere schnelle Statements

Dr. Andrea Kennel

InfoPunkt Kennel GmbH

November 2009

Agenda

- **Einleitung**
- **Analytische Funktionen**
- **Merge Statement**
- **Multitable Insert**
- **Fazit**
- **Fragen & Antworten**

Einleitung

- Wie finde ich die Lösung zu einem Problem?
- Wir zeigen Lösungen anhand von Beispielen

Analystische Funktionen: Problem

Tabelle COSTS

PROD_ID	PROMO_ID	CHANNEL_ID	UNIT_COST	UNIT_PRICE	VALID_FROM	VALID_TO
116	999	4	9,93	12,73	20.12.01	
116	999	4	9,93	12,73	22.12.01	
116	999	4	9,93	12,73	24.12.01	
116	999	4	9,93	12,73	25.12.01	
116	999	4	9,93	12,73	28.12.01	
116	999	4	9,93	12,73	29.12.01	
116	999	4	9,93	12,73	31.12.01	

Wie kann das VALID_TO berechnet werden?

Analytische Funktionen: Lösung mit SQL

```
SELECT a.prod_id, a.promo_id, a.channel_id,
       a.unit_cost, a.unit_price, a.time_id
       valid_from, min(b.time_id) -1 valid_to
FROM costs a, costs b
WHERE a.prod_id = b.prod_id AND
      a.promo_id = b.promo_id AND
      a.channel_id = b.channel_id AND
      a.time_id < b.time_id
GROUP BY a.prod_id, a.promo_id, a.channel_id,
         a.unit_cost, a.unit_price, a.time_id
```

Analytische Funktionen:
Lösung analytische Funktionen

```
SELECT a.prod_id, a.promo_id, a.channel_id,
       a.unit_cost, a.unit_price, a.time_id
       valid_from,
LEAD(a.time_id) OVER (PARTITION BY a.prod_id,
     a.promo_id, a.channel_id ORDER BY a.time_id)
     - 1 valid_to
FROM costs a
```

Analystische Funktionen: Performancevergleich

Executionplan mit Join

Operation	Name	Rows	Bytes	Cost	Time	CPU Cost	I/O Cost
SELECT STATEMENT (ALL_ROWS)		520245	23411025	6567	00:01:19	1653355013	6283
HASH GROUP BY		520245	23411025	6567	00:01:19	1653355013	6283
HASH JOIN		520245	23411025	579	00:00:07	1108327672	389
PARTITION RANGE ALL		82112	1560128	63	00:00:01	19137823	60
TABLE ACCESS FULL (ANALYZED)	SH.COSTS (TABLE)	82112	1560128	63	00:00:01	19137823	60
PARTITION RANGE ALL		82112	2134912	64	00:00:01	22422303	60
TABLE ACCESS FULL (ANALYZED)	SH.COSTS (TABLE)	82112	2134912	64	00:00:01	22422303	60

Laufzeit: 00:00:31.95

Executionplan mit LEAD

Operation	Name	Rows	Bytes	Cost	Time	CPU Cost	I/O Cost
SELECT STATEMENT (ALL_ROWS)		82112	2134912	684	00:00:09	98315824	667
WINDOW SORT		82112	2134912	684	00:00:09	98315824	667
PARTITION RANGE ALL		82112	2134912	64	00:00:01	22422303	60
TABLE ACCESS FULL (ANALYZED)	SH.COSTS (TABLE)	82112	2134912	64	00:00:01	22422303	60

Laufzeit: 00:00:01.48

Analytische Funktionen: Weiteres Beispiel

```
SELECT prod_id, prod_subcategory_id sub_cat,
       prod_name, prod_list_price price,
       SUM(prod_list_price) OVER (PARTITION BY
           prod_subcategory_id ORDER BY
  prod_list_price)
           running,
       SUM(prod_list_price) OVER (PARTITION BY
          prod_subcategory_id) sum_sub_cat,
       SUM(prod_list_price) OVER () sum_total,
       RANK() OVER (ORDER BY prod_list_price) rank
FROM products
ORDER BY prod_subcategory_id, prod_list_price
```

Analystische Funktionen: Resultat

PROD_ID	SUB_CAT	PROD_NAME	PRICE	RUNNING	SUM_SUB_CAT	SUM_TOTAL	RANK
46	2031	Standard Mouse	22,99	22,99	189,95	344,91	1
22	2031	Keyboard	24,99	47,98	189,95	344,91	2
47	2031	Deluxe Mouse	28,99	76,97	189,95	344,91	3
27	2031	Speakers- 3"	44,99	121,96	189,95	344,91	7
32	2031	Speakers- 5"	67,99	189,95	189,95	344,91	9
38	2032	Internal 6X CD-ROM	29,99	29,99	154,96	344,91	4
39	2032	Internal 8X CD-ROM	34,99	64,98	154,96	344,91	5
34	2032	6X CD-ROM	39,99	104,97	154,96	344,91	6
35	2032	8X CD-ROM	49,99	154,96	154,96	344,91	8

SUM(prod_list_price) **OVER** (**PARTITION BY** prod_subcategory_id **ORDER BY** prod_list_price)
running,
SUM(prod_list_price) **OVER** (**PARTITION BY** prod_subcategory_id) sum_sub_cat,
SUM(prod_list_price) **OVER** () sum_total,
RANK() **OVER** (**ORDER BY** prod_list_price) **rank**

Merge Statement: Aufgabenstellung

Tabelle Costs

- Kosten und Preise
  je Verkaufskanal
  je Werbeweg (Promotion)

⇒Neue Tabelle COST_02
Kosten neu berechnen

(Alle Kosten des Produktes in diesem Monat um 10 % der Maximalkosten erhöhen.)

Merge Statement:
Daten vor Update

PROD_ID	TIME_ID	PROMO_ID	CHANNEL_ID	UNIT_COST	UNIT_PRICE
148	08.07.01	999	3	18,32	23,55
148	18.07.01	999	2	17,83	23,69
148	18.07.01	999	4	17,52	22,37
148	23.07.01	999	4	17,52	22,37
148	02.07.01	999	3	18,32	23,55
148	08.07.01	999	4	17,52	22,37
148	02.07.01	999	4	17,52	22,37
148	27.07.01	999	3	18,32	23,55
148	27.07.01	999	4	17,52	22,37
148	27.07.01	999	2	17,83	23,69
148	08.07.01	999	2	17,83	23,69
148	02.07.01	999	2	17,83	23,69
148	23.07.01	999	3	18,32	23,55
148	18.07.01	999	3	18,32	23,55
148	23.07.01	999	2	17,83	23,69

18,32 + 1,83 = 20,15

17,83 + 1,83 = 19,66

Merge Statement: Lösung mit SQL

```
UPDATE costs_2 a
SET unit_cost = unit_cost +
    (SELECT max(unit_cost) * 0.1
    FROM costs_2 b
WHERE a.prod_id = b.prod_id AND
      to_char(a.time_id,'YYYY-MM') =
      to_char(b.time_id,'YYYY-MM'))
```

Merge Statement:
Daten nach Update

PROD_ID	TIME_ID	PROMO_ID	CHANNEL_ID	UNIT_COST	UNIT_PRICE
148	08.07.01	999	3	20.15	23.55
148	18.07.01	999	2	19.66	23.69
148	18.07.01	999	4	19.35	22.37
148	23.07.01	999	4	19.35	22.37
148	02.07.01	999	3	20.15	23.55
148	08.07.01	999	4	19.35	22.37
148	02.07.01	999	4	19.35	22.37
148	27.07.01	999	3	20.15	23.55
148	27.07.01	999	4	19.35	22.37
148	27.07.01	999	2	19.66	23.69
148	08.07.01	999	2	19.66	23.69
148	02.07.01	999	2	19.66	23.69
148	23.07.01	999	3	20.15	23.55
148	18.07.01	999	3	20.15	23.55
148	23.07.01	999	2	19.66	23.69

Merge Statement:
Lösung mit Merge

```
MERGE INTO costs_2 d USING
 (SELECT detail.prod_id, detail.time_id, detail.promo_id,
  detail.channel_id, month.max_cost
  FROM costs_2 detail,
      (SELECT prod_id, to_char(time_id,'YYYY-MM') month,
      max(unit_cost) * 0.1 max_cost FROM costs_2
      GROUP BY prod_id, to_char(time_id,'YYYY-MM'))
      month
      WHERE detail.prod_id = month.prod_id AND
            to_char(detail.time_id,'YYYY-MM') =
            month.month) s
ON (d.prod_id = s.prod_id AND
    d.time_id = s.time_id AND
    d.promo_id = s.promo_id AND
    d.channel_id = s.channel_id)
WHEN MATCHED THEN
UPDATE SET d.unit_cost = d.unit_cost + s.max_cost
```

Merge Statement:

Lösung mit Merge + analytischen Funktionen

```
MERGE INTO costs_2 d USING
 (SELECT detail.prod_id, detail.time_id,
         detail.promo_id, detail.channel_id,
         sum(unit_cost) OVER (PARTITION BY prod_id,
         to_char(time_id,'YYYY-MM')) max_cost
         FROM costs_2 detail) s
ON (d.prod_id = s.prod_id AND
    d.time_id = s.time_id AND
    d.promo_id = s.promo_id AND
    d.channel_id = s.channel_id)
WHEN MATCHED THEN
UPDATE SET d.unit_cost = d.unit_cost + s.max_cost
```

Merge Statement: Performancevergleich

Executionplan mit Update

Operation	Name	Rows	Bytes	Cost	Time	CPU Cost	I/O Cost
UPDATE STATEMENT (ALL_ROWS)		84069	2942415	88	00:00:02	22013531	84
UPDATE	SH.COSTS_2	0	0	0	00:00:00	0	0
TABLE ACCESS FULL	SH.COSTS_2 (TABLE)	84069	2942415	88	00:00:02	22013531	84
SORT AGGREGATE		1	35	0	00:00:00	0	0
TABLE ACCESS FULL	SH.COSTS_2 (TABLE)	8	280	88	00:00:02	22056035	84

Laufzeit: 00:45:40.00

Executionplan mit Merge und analytische Funktion

Operation	Name	Rows	Bytes	Cost	Time	CPU Cost	I/O Cost
MERGE STATEMENT (ALL_ROWS)		266801	10405239	2270	00:00:28	203054961	2235
MERGE	SH.COSTS_2	0	0	0	00:00:00	0	0
VIEW		0	0	0	00:00:00	0	0
HASH JOIN		266801	39219747	2270	00:00:28	203054961	2235
VIEW		90521	5521781	1439	00:00:18	117773216	1419
WINDOW SORT		90521	5521781	1439	00:00:18	117773216	1419
TABLE ACCESS FULL	SH.COSTS_2 (TABLE)	90521	5521781	88	00:00:02	23497491	84
TABLE ACCESS FULL	SH.COSTS_2 (TABLE)	90521	7784806	88	00:00:02	25307911	84

Laufzeit: 00:00:08.46

Merge Statement: Performancevergleich

Executionplan mit Merge und Join

Operation	Name	Rows	Bytes	Cost	Time	CPU Cost	I/O Cost
MERGE STATEMENT (ALL_ROWS)		3354306	130817934	3059	00:00:37	602908213	2955
MERGE	SH.COSTS_2	0	0	0	00:00:00	0	0
VIEW		0	0	0	00:00:00	0	0
HASH JOIN		3354306	553460490	3059	00:00:37	602908213	2955
HASH JOIN		266800	35751200	863	00:00:11	106377963	845
TABLE ACCESS FULL	SH.COSTS_2 (TABLE)	90521	4345008	88	00:00:02	21687071	84
TABLE ACCESS FULL	SH.COSTS_2 (TABLE)	90521	7784806	88	00:00:02	25307911	84
VIEW		90521	2806151	101	00:00:02	96471726	84
SORT GROUP BY		90521	3168235	101	00:00:02	96471726	84
TABLE ACCESS FULL	SH.COSTS_2 (TABLE)	90521	3168235	88	00:00:02	23497491	84

Laufzeit: 00:00:08.62

Multi Table Insert: Aufgabenstellung

Tabelle Costs

- Preise und die neu berechneten Kosten der Produkte

  je Verkaufskanal

  je Werbeweg (Promotion)

⇒Daten sollen je Proudktgruppe auf neue Tabellen verteilt werden

Multi Table Insert: Lösung mit SQL

```
INSERT INTO costs_201 (prod_id, time_id, promo_id,
            channel_id, unit_cost, unit_price)
SELECT c.prod_id, c.time_id, c.promo_id, c.channel_id,
       c.unit_cost, c.unit_price
FROM costs_2 c, products p
WHERE c.prod_id = p.prod_id AND
      p.prod_category_id = 201;
INSERT INTO costs_202 (prod_id, time_id, promo_id,
            channel_id, unit_cost, unit_price)
SELECT c.prod_id, c.time_id, c.promo_id, c.channel_id,
       c.unit_cost, c.unit_price
FROM costs_2 c, products p
WHERE c.prod_id = p.prod_id AND
      p.prod_category_id = 202;
…
```

Multi Table Insert:
Lösung mit Mulit Table Insert

```
INSERT ALL
WHEN prod_category_id = 201 THEN
  INTO costs_201 VALUES (prod_id, time_id, promo_id, channel_id,
       unit_cost, unit_price)
WHEN prod_category_id = 202 THEN
  INTO costs_202 VALUES (prod_id, time_id, promo_id, channel_id,
       unit_cost, unit_price)
WHEN prod_category_id = 203 THEN
  INTO costs_203 VALUES (prod_id, time_id, promo_id, channel_id,
       unit_cost, unit_price)
WHEN prod_category_id = 204 THEN
  INTO costs_204 VALUES (prod_id, time_id, promo_id, channel_id,
       unit_cost, unit_price)
WHEN prod_category_id = 205 THEN
  INTO costs_205 VALUES (prod_id, time_id, promo_id, channel_id,
       unit_cost, unit_price)
SELECT c.prod_id, c.time_id, c.promo_id, c.channel_id, c.unit_cost,
       c.unit_price, p.prod_category_id
FROM costs_2 c, products p
WHERE c.prod_id = p.prod_id
```

Fazit

- Einsatz von analystischen Funktionen und anderen Features sind sinnvoll
- Performanceverbesserungen
- einfachere Programmierung
- Weiterbildung lohnt sich

Fragen & Antworten

?