Datenbankadministration

12. Datenbankobjekte und analytische Funktionen

AG DBIS University of Kaiserslautern, Germany

Karsten Schmidt kschmidt@informatik.uni-kl.de
(Vorlage TU-Dresden)

Wintersemester 2008/2009

Datenbankobjekte - Tabelle

- **Tabelle** (*table*)
  - logisches Konstrukt aus Spalten verschiedenen Datentyps zur Speicherung von Daten
  - Informationen zu Tabellen
    - `LIST TABLES [FOR {ALL | SCHEMA <schemaname>}]`
    - `DESCRIBE TABLE <tablename>`
- **Tabellenpartitionierung** (*table (range) partitioning*)
  - Ziel: Performanzsteigerung
    - Tupel einer Tabelle auf mehrere Tabellenbereiche verteilen → Parallelität
    - selten benötigte Daten auf Tertiärspeicher auslagern → schnellerer Zugriff auf kleinere Tabellen
  - Partitionierung nach Partitionierungsspalte
    - auch mehrere Spalten und generierte Spalte möglich
    - keine XML-, LOB- oder LONG VARCHAR-Spalten

Datenbankobjekte - Tabelle

- **Tabellenpartitionierung** (Fortsetzung)
  - Bereiche definieren und auf Tabellenbereiche verteilen

```
CREATE TABLE bsptable (id int, ...)
PARTITION BY RANGE (id)
(
  [PARTITION p0] STARTING FROM (0) ENDING (10) IN tbspc1,
  [PARTITION p1] ENDING (20) IN tbspc2, ...
)
```

  - Kurzschreibweise, Reihum-Zuweisung auf Tabellenbereiche

```
CREATE TABLE bsptable (id int, ...)
IN tbspc1, tbspc2, ...
PARTITION BY RANGE (id)
(STARTING FROM (0) ENDING (80) EVERY (10))
```

  - Bereich abtrennen

```
ALTER TABLE <tblname>
DETACH PARTITION <partitionname> INTO TABLE <tblname>
```

  - Bereich hinzufügen

```
ALTER TABLE <tblname>
ATTACH PARTITION STARTING <start> ENDING <end>
        FROM TABLE <tblname>
```

Datenbankobjekte - Tabelle

- **Datenzeilenkomprimierung** (*table compression*)
  - Ziel: Einsparungen bei Plattenplatz und E/A-Operationen
  - Ersetzen von sich wiederholenden Mustern (auch über mehrere Spalten) einer Zeile durch kurzes Symbol
  - Datenzeilenkomprimierung aktivieren
    - `CREATE TABLE <table_name> ... COMPRESS YES`
    - `ALTER TABLE <table_name> COMPRESS YES`
  - Datenzeilenkomprimierung durchführen, Komprimierungswörterverzeichnis erstellen
    - `REORG TABLE <table_name> {RESETDICTIONARY | KEEPDICTIONARY}`
- **Einschränkung**
  - nicht für LONG-, LOB- und XML-Spalten

Datenbankobjekte - Sicht

- **Sicht** (*view*)
  - virtuelle Relation zur Vereinfachung von Anfragen bzw. nutzerspezifische Datendarstellung
  - auch Datenschutz durch Ausblenden von Tupeln / Attributen
  - Spezifikation des Inhaltes durch SQL-Anfrage
  - Sicht ungültig, wenn zugrunde liegende Sicht oder Tabelle gelöscht wird (`SYSCAT.VIEWS: VALID = X`)
  - je nach Spezifikation *updatable* oder *read-only*:
    - *updatable*: wenn Spalten der Sicht auf die der Tabelle abgebildet werden können
    - *read-only*: sonst (`VALUES, DISTINCT,` Verbund, ...)
    - `SYSCAT.VIEWS: READONLY=Y/N`

Datenbankobjekte - Sicht

- **Aktualisierung auf Sichten**
  - Einschränkung durch `WITH CHECK OPTION`
    - nur solche Aktualisierungen möglich, die mit Sichtdefinition konform sind
  - Sicht auf Sicht (*nested view*)
    - `WITH CASCADED CHECK OPTION: WITH CHECK OPTION` für alle zugrunde liegenden Sichten anwenden
    - `WITH LOCAL CHECK OPTION: WITH CHECK OPTION` nur für aktuelle Sicht anwenden

Datenbankobjekte - Integritätsbedingung

- **Integritätsbedingung** (*constraint*)
  - Regel, die vom Datenbankmanagementsystem erzwungen wird
  - Typen
    - Primärschlüssel/Disjunktheit (*primary key/unique key constraint*)
      - Eindeutigkeit über eine oder mehr Spalten
      - nur für `NOT NULL`-Spalten möglich
      - erzwingt Unique-Index auf betroffenen Spalten
    - wertebasiert (*check constraint*)
      - Vorgabe gültiger Werte
    - Informational Constraint
      - Regeln, die vom Optimierer verwendet werden können
      - Achtung: werden nicht erzwungen!
    - Fremdschlüssel (*foreign key constraint*)
      - Wertegleichheit von Attributen
      - beschreibt Beziehung zwischen oder innerhalb Tabellen

Datenbankobjekte - Integritätsbedingungen

- **Fremdschlüssel**
  - Verhalten bei Änderungen der Daten
    - Einfügen in abhängige Tabelle
      - Wert des Fremdschlüsselattributs muss in Elterntabelle existieren
    - Löschen aus Elterntabelle
      - `RESTRICT/NO ACTION`: nicht möglich, wenn abhängige Tupel vorhanden
      - `CASCADE`: abhängige Tupel aus abhängiger Tabelle löschen
      - `SET NULL`: Fremdschlüsselwert abhängiger Tupel NULL setzen
    - Ändern in Elterntabelle
      - `RESTRICT`: nicht mögich, wenn abhängige Tupel vorhanden
      - `NO ACTION`: nicht möglich, wenn nach Update abhängige Tupel ohne Elterntupel existieren

Datenbankobjekte - Index / Schema / Trigger

- **Index**
  - Einsatzgebiete
    - schneller Zugriff auf Daten
    - Erzwingen von Eindeutigkeit
  - Nachteile
    - zusätzlicher Aufwand für Indexaktualisierung notwendig
    - zusätzlicher Speicherplatz
- **Schema**
  - logische Klassifikation von Datenbankobjekten
  - erlaubt gleiche Tabellennamen in verschiedenen Schemata
- **Trigger**
  - Menge von Aktionen, die angestoßen werden, falls bestimmte Ereignisse eintreten
  - Ziel: Validierung, Datenaufbereitung, Integritätssicherung

Verwaltung

Verwaltung

- **Instanzverwaltung**
  - Instanz erzeugen: `db2icrt <instance_name>`
  - Instanz löschen: `db2idrop -f <instance_name>`
  - Instanz auflisten: `db2ilist`
- **Registrierdatenbankvariablen** (*profile registries*)
  - DB2-spezifische Variablen für Verwaltung, Konfiguration und Performanz
  - nach Änderung Neustart der Instanz erforderlich
  - alle unterstützten Variablen auflisten: `db2set -lr`
  - alle gesetzten Registrierdatenbankvariablen auflisten: `db2set -all`
  - setzen: `db2set <reg_variable>=<value>` (ohne Leerzeichen!)
- **Umgebungsvariablen**
  - vom System verwaltete DB2-Umgebungsvariablen (z.B. `DB2INSTANCE`)
- **Verbundene Anwendungen**
  - Anwendungen auflisten: `list applications`
  - alle Anwendungen trennen: `force application {all | <appl_handle>}`
  - alle Anwendungen trennen und Instanz stoppen: `db2stop force`

Analytische Funktionen

Analytische Funktionen

- **Grundlagen**
  - Skalarfunktionen (tupellokal): z.B. +-*/
  - Gruppierung: `GROUP BY`
    - Zusammenfassen mehrerer Tupel mit gleichen Ausprägungen (Verdichtung)
  - Aggregationsfunktionen: z.B. `SUM`, `COUNT`, `AVG`
    - Berechnung einer Kennzahl aus mehreren Einzelwerten
  - Gruppierung und Aggregation treten oft zusammen auf
- **OLAP-Funktionen** (*analytic functions*)
  - Trennung von Aggregation und Verdichtung
    - Aggregation auf Ergebnis einer Anfrage
    - keine weitere Verdichtung
  - Kernkonstrukt: `OVER`-Klausel
  - Umfasst
    - Attributlokale Partitionierung (Bildung dynamischer Fenster)
    - Ranking / Numbering

OVER-Klausel

- **OVER-Klausel**
  - u.a. Berechnung von Aggregationsfunktionen ohne weitere Verdichtung
  - `OVER ()` entspricht Aggregation über alle Tupel

R		
Jahr	**Quartal**	**Umsatz**
2004	1	10
2004	2	20
2004	3	10
2004	4	20
2005	1	30

```
SELECT Jahr,Quartal,
       Umsatz,
       SUM(Umsatz) OVER ()
FROM R
```

R			
Jahr	**Quartal**	**Umsatz**	**Gesamt-umsatz**
2004	1	10	90
2004	2	20	90
2004	3	10	90
2004	4	20	90
2005	1	30	90

Attributlokale Partitionierung

- **Attributlokale Partitionierung**
  - Auswahl der zu aggregierenden Tupel abhängig vom Kontext
  - `PARTITION BY <attribute-list>`
    - Aggregation aller Tupel, die in den angegebenen Attributen mit dem aktuellen Tupel übereinstimmen

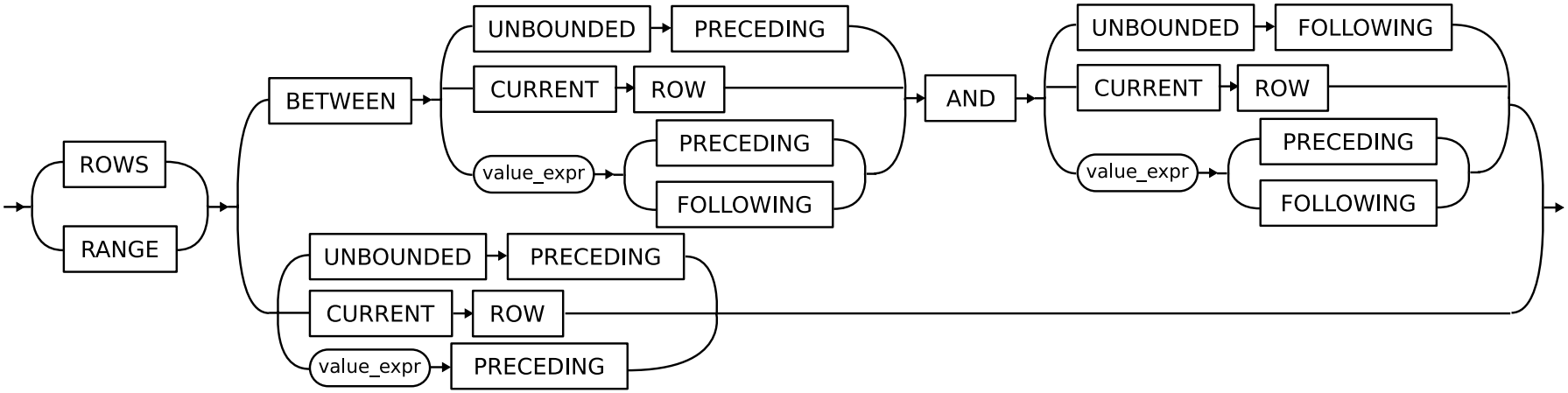
R		
Jahr	**Quartal**	**Umsatz**
2004	1	10
2004	2	20
2004	3	10
2004	4	20
2005	1	30

```
SELECT Jahr,Quartal,
       Umsatz,
       SUM(Umsatz) OVER (
         PARTITION BY Jahr
       )
FROM R
```

R			
Jahr	**Quartal**	**Umsatz**	**Jahres-umsatz**
2004	1	10	60
2004	2	20	60
2004	3	10	60
2004	4	20	60
2005	1	30	30

Attributlokale Partitionierung

- **Bildung dynamischer Fenster**
  - erfordert Sortierung der Tupel (lokal, in der `OVER`-Klausel angegeben)
  - `OVER (ORDER BY <attribute-list> [<window-spec>])`
  - Aggregation über benachbarte Tupel (=Fenster)
    - `ROWS`-Klausel = Anzahl vorhergehender / nachfolgender Tupel
    - `RANGE`-Klausel = Wertebereich (z.B. ± 5 Tage)



- Kombination mit `PARTITION BY`-Klausel möglich

Attributlokale Partitionierung

- **Beispiel: gleitendes Mittel**

```
SELECT Jahr,Quartal,
       Umsatz,
       AVG(Umsatz) OVER (
         ORDER BY Jahr, Quartal
         ROWS BETWEEN 1 PRECEDING AND CURRENT ROW
       )
FROM R
```

R		
Jahr	**Quartal**	**Umsatz**
2004	1	10
2004	2	20
2004	3	10
2004	4	20
2005	1	30

$\longrightarrow$

R			
Jahr	**Quartal**	**Umsatz**	**Gleit-umsatz**
2004	1	10	10
2004	2	20	15
2004	3	10	15
2004	4	20	15
2005	1	30	25

Ranking

- **Ranking**
  - Bestimmung der Position eines Tupels in der Ergebnismenge
  - erfordert Sortierung innerhalb der `OVER`-Klausel
  - Werte mit identischer Ausprägung erhalten denselben Rang
  - `RANK()` (mit Lücken) vs. `DENSE_RANK()` (ohne Lücken)
  - Spezialfall: Top-k-Anfragen (mit `FETCH FIRST n ROWS ONLY`)
  - mit `PARTITION BY`-Klausel kombinierbar

```
SELECT Jahr,Quartal,Umsatz
      RANK() OVER ( ORDER BY Umsatz DESC),
      DENSE_RANK() OVER ( ORDER BY Umsatz DESC)
FROM R
```

R		
Jahr	**Quartal**	**Umsatz**
2004	1	10
2004	2	20
2004	3	10
2004	4	20
2005	1	30

$\longrightarrow$

R				
Jahr	**Quartal**	**Umsatz**	**Rang**	**Rang (dicht)**
2004	1	10	4	3
2004	2	20	2	2
2004	3	10	4	3
2004	4	20	2	2
2005	1	30	1	1

Zusammenfassung

- **Datenbankobjekte**
  - Tabelle
  - Sicht
  - Integritätsbedingung
  - Index
  - Schema
  - Trigger
- **Verwaltung**
  - Instanzverwaltung
  - DB2-spezifische Parameter
- **Analytische Funktionen**
  - attributlokale Partitionierung
  - Ranking