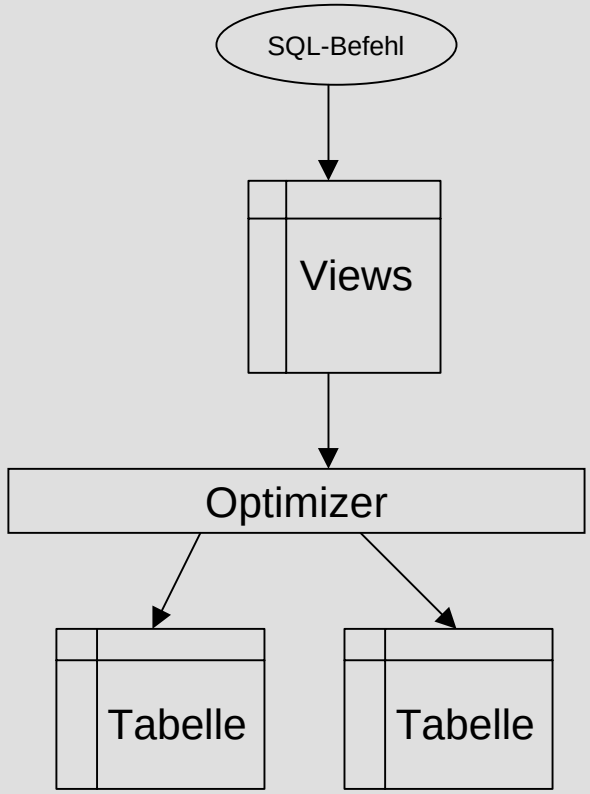
Materialized Views

Jan-Peter Timmermann

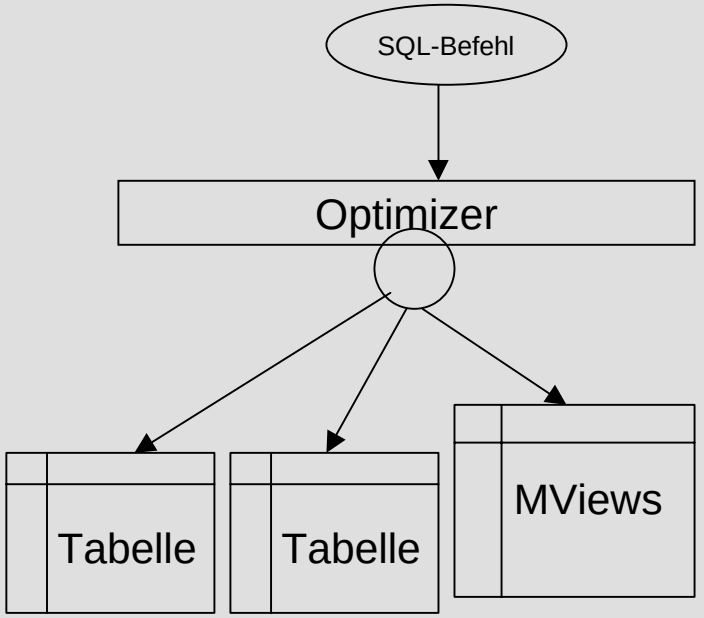
Klassische View

- **Sind virtuelle Tabellen**
- **Erstellung nicht aufwendig, da nur ein Eintrag im Data Dictionary vorgenommen werden muss**
- **Sie wird explizit im SQL-Befehl angesprochen**
- **Wird auf reale Tabelle umgesetzt**
- **in manchen Fällen direkte Änderung möglich**
- **Views führen keine Redundanzen ein**



Materialized View

- **Sind reale Tabellen**
- **Erstellung i.d.R. sehr aufwendig**
- **wird nicht explizit im SQL-Befehl angesprochen**
- **Basistabellen werden im SQL-Befehl angesprochen, können aber u.U. vom Optimizer umgeleitet werden**
- **direkte Änderung nicht möglich**
- **Änderungen in den Tabellen müssen nachgezogen werden (refresh)**
- **redundante Daten**



Einsatzgebiet für Materialized Views (1)

- **dort, wo aus Gründen der Performance Daten redundant gespeichert werden**
- **dort, wo man vormals SNAPSHOTS genutzt hat**
- **Data Warehouse**
  - **Vorberechnung und Speicherung aggregierter Daten (Dimensions-/Fakttabellen)**
  - **„Kostenbasierte Optimierung“ erhöht die Performanz. Der Optimizer entscheidet transparent für die Anwendung, ob die Materialized View oder aber die darunter liegenden Tabellen/Views genutzt werden sollen**

- **Distributed environments**
  - **hier werden Materialized Views auch *Snapshots* genannt**
  - **Datenreplikation und Synchronisierung von Updates mit Konfliktbehebungsstrategien**
  - **Materialized Views ermöglichen lokalen Zugriff auf Daten, auf die normalerweise als „entfernte Daten“ zugegriffen werden müßte**
- **Mobile computing environments**
  - **Herunterladen eines Teils der Daten eines zentralen Datenservers auf mobile Clients**
  - **bidirektionale periodische Aktualisierung der Daten**

Arten von Materialized Views

- Aggregate View
  - eine Tabelle als Quelle
  - aggregierte Daten mit Group by (sum,count,avg...)
- Join View
  - mehrere Tabellen
  - inner/outer Join
  - keine Aggregate
  - können große Tabllen werden
  - Rowid der Basis Tabellen mit in die MV
- Aggregate + Join View
  - mehrere Tabellen incl. Join und aggregierte Daten.

```
create materialized view Demo_1
build immediate
enable query rewrite as
select r.lear_id, sum (r.anzahl_seiten) as anzahl_seiten,
      sum (r.pos_preis_b  ) as betrag,
      count(r.pos_preis_b ) as c_betrag,
            count(*)            as c_gesamt
from rechnung_positionen r
where r.art='B'
group by lear_id
```

- Create materialized view <name>
- build deferred
- refresh fast
- on demand
- enable query rewrite

- Name der View
- Aufbau Zeitpunkt
- Refresh Modus
- Refresh Zeitpunkt
- Rewrite ein/aus

Es sollen die Anzahl und die Beträge für
einen Bezirk ausgegeben werden mit folgendem Select

```
select r.lear_id,
      sum (r.anzahl_seiten) as anzahl_seiten,
      sum (r.pos_preis_b  ) as betrag,
      count(r.pos_preis_b ) as c_betrag
from rechnung_positionen r
where r.art='B‘ and lear_id=39
group by lear_id
```

Demo_1_ohne_mv.sql

Anlegen der MV

```
create materialized view Demo_1
build immediate
enable query rewrite
as
select r.lear_id,
      sum (r.anzahl_seiten) as anzahl_seiten,
            sum (r.pos_preis_b  ) as betrag,
            count(r.pos_preis_b ) as c_betrag,
            count(*)              as c_gesamt
from rechnung_positionen r
where r.art='B'
group by lear_id
```

create_mv_demo_1.sql

Es sollen die Anzahl und die Beträge für
einen Bezirk ausgegeben werden mit folgendem Select

```
select r.lear_id,
      sum (r.anzahl_seiten) as anzahl_seiten,
      sum (r.pos_preis_b  ) as betrag,
      count(r.pos_preis_b ) as c_betrag
from rechnung_positionen r
where r.art='B‘ and lear_id=39
group by lear_id
```

Demo_1.sql

Neuer Select

Es wird nur Teil der MV benötigt. Hier ist es eine AVG() Funktion. Diese ist nicht in der MV vorhanden, sondern nur die Summen.

```
select r.lear_id,
        avg (r.pos_preis_b  ) as
betrag
from rechnung_positionen r
where r.art='B‘ and lear_id=39
group by lear_id
/
```

Trozt textueller Differenz wird hier die MV herangezogen

AVG-Werte können aus der MV gezogen werden.

Demo_3.sql

Es wird von den Rechnungspositionen zwar der Betrag, aber eine andere Gruppierung gefordert.

```
select trunc(r.insert_date),
        avg (r.pos_preis_b) as betrag
from rechnung_positionen r
where r.art='B'and trunc(insert_date) like '15.12.2001')
group by trunc(insert_date)
having sum(r.pos_preis_b) >100
/
```

Rewrite trotz anderer Gruppierung als in der MV.

Demo_4.sql

In diesen vorangegangen Beispielen war ein Rewrite möglich, obwohl in der MV kein AVG() Wert explizit abgespeichert worden ist.

Da jedoch in der MV sowohl die Summen als auch die Anzahl der Werte hinterlegt worden ist, kann der Optimizer hier die MV heranziehen.

Eine einfache Join-View

```
create materialized view Demo_2
build immediate
enable query rewrite as
select r.lear_id,     sum (r.anzahl_seiten) as anzahl_seiten,
                      sum (r.pos_preis_b  ) as betrag,
                      count(r.pos_preis_b ) as c_betrag,
                      count(*)              as c_gesamt,
                      rk.rechnungs_nummer  ,
                      trunc(rk.datum_rechnungsstellung)
from rechnung_positionen r , rechnung_koepfe rk
where r.art='B'
and r.reko_id = rk.reko_id
group by r.lear_id,trunc(rk.datum_rechnungsstellung),rk.rechnungs_nummer
/
```

```
select r.lear_id,sum (r.pos_preis_b  ) as betrag,
                    trunc(rk.datum_rechnungsstellung) Tag
from rechnung_positionen r , rechnung_koepfe rk
where r.art='B'
and r.reko_id = rk.reko_id
and r.lear_id=39
and trunc(rk.datum_rechnungsstellung) like '19.09.02'
group by r.lear_id,trunc(rk.datum_rechnungsstellung)
/
```

Demo_5.sql

Probleme

- **Regelmäßiges aktualisieren der MV**
  - eine Job anlegen mit dem Inhalt ‘dbms_refresh(‘DEMO_1‘)‘

- **in den Tabellen werden Daten eingefügt**
  - Die MV mit der Option on commit refresh

Demo_5.sql

On commit refresh

```
create materialized view Demo_5
build immediate
refresh on commit
enable query rewrite as
select r.lear_id,
       trunc(r.insert_date) as Datum,
       sum (r.anzahl_seiten) as anzahl_seiten,
      sum (r.pos_preis_b  ) as betrag,
      count(r.pos_preis_b ) as c_betrag,
      count(*)              as c_gesamt
from rechnung_positionen r
where r.art='B'
group by lear_id,trunc(r.insert_date)
```

Demo_6.sql