Kap. 2: Vertiefung von SQL

Grundaufbau SQL

Grundaufbau eines SQL Queries

```
SELECT <Spaltenliste>
FROM <Tabellenliste>
WHERE <Bedingung>
```

- Filterung doppelter Zeilen per **DISTINCT**

```
(SELECT DISTINCT year FROM movie)
```

- Vereinigung von Select-Ergebnissen mit **UNION**

```
(SELECT year FROM movie WHERE year = 2001
UNION
SELECT year FROM movie WHERE year = 2002)
```

  - Filter doppelte Ergebnisse (Unterdrückung der Filterung mit **UNION ALL**)

Join-Anfragen

Titel aller Kinofilme mit Genre-Angabe

```
SELECT m.title, g.genre
FROM movie m JOIN genre g ON (m.id = g.movie)
WHERE m.year = 1995
   and m.type = 'C'
```

Alternativ:

```
SELECT m.title, g.genre
FROM movie m, genre g
WHERE m.year = 1995
   and m.type = 'C'
   and m.id = g.movie
```

Kreuzprodukt:

```
SELECT *
FROM movie m, genre g
```

movie Cross Join genre g

Equi-Join:

```
SELECT *
FROM movie m, genre g
WHERE m.id = g.movie
```

JOIN .. ON ( )

Left Outer Join: (Oracle-Syntax)

```
SELECT *
FROM movie m, genre g
WHERE m.id = g.movie(+)
```

Left JOIN ... ON ( )

Join-Typen

Left Outer Join

- Auch die Datensätze der linken Tabelle, zu denen es keinen passenden Join-Partner gibt

Right Outer Join

- Analog mit allen Datensätzen der rechten Tabelle

Full Outer Join

- Kombinierter Right Outer Join und Left Outer Join

Natural Join

- Equi-Join auf Basis gleichbenannter Spalten
- Doppelte Spalten werden ausgeblendet

Theta Join

- Statt Gleichheit beliebiger Vergleichsoperator in der Join-Bedingung
- Beispiel: ... WHERE x.year < y.year ...

Gruppierung und Aggregierung

Gruppierung: Datensätze mit gemeinsamen Eigenschaften zusammenfassen.

Datengrundlage für die Beispiele:

id	title	year	type	votes	rating
461530	Simpsons, The	1989	S	392	9,2
30059	Lost	2004	S	958	9,1
352285	Lord of the Rings: The Two Towers	2002	C	272878	8,6
531609	Star Wars: Episode II	2002	C	138873	6,8

-
```
CREATE VIEW mr AS
SELECT id, title, year, type, votes, rating
FROM movie m JOIN rating r ON (m.id = r.movie);
```

```
SELECT year, avg(rating),
count(*) FROM MR
GROUP BY year
HAVING sum(votes) > 200000
```

Gruppe Datensätze

AVG() ...

```
SELECT ???
FROM mr
GROUP BY year
```

AVG() ...

```
SELECT ???
FROM mr
GROUP BY type, year
```

AVG() ...

```
SELECT ???
FROM mr
GROUP BY type
```

```
SELECT year, avg(rating),
count(*) FROM MR
WHERE votes > 200000
GROUP BY year
```

Einzelne GROUP Datensätze

Aggregations Funktionen

Sortierung und Unterabfragen

Sortierung und Top-N Abfragen

ROW_NUMBER: Beschränkung der Ergebnismenge

Kombination von Unterabfrage mit ROW_NUMBER:

```
SELECT * FROM
(
SELECT
   ROW_NUMBER() OVER (ORDER BY Rating DESC) AS rn,
   title
FROM mr
 WHERE type = 'C'
)
WHERE rn <= 10
ORDER BY rn
```

```
SELECT title
FROM mr
WHERE type = 'C'
ORDER BY rating DESC;
```

Verwendung von ROW_NUMBER

```
SELECT
      ROW_NUMBER() OVER (ORDER BY
      Rating DESC) AS rn,
      title
FROM mr
WHERE type = 'C'
ORDER BY 1
```

Maximumsbildung

Selektiere den / die besten Filme für jedes Jahr

Erster Versuch:

- `SELECT year, max(title), max(rating) FROM mr WHERE type = 'C' GROUP BY year;`

Problem:

- `max(title)` hat nichts mit `max(rating)` zu tun.

Lösung: Join auf Unterabfrage

```
(SELECT year, max(rating) rating
 FROM mr
 WHERE type = 'C' GROUP BY year)
```

```
SELECT *  FROM mr JOIN      X ON
(mr.rating = X.rating and mr.year = X.year)
```

Hierarchische Abfragen

Beispiel: follows-Relation

movie	successor
1	2
2	3
3	4
5	6
6	7
8	9
9	10

Wie selektiere ich alle Nachfolger von 1?

- Join funktioniert nicht, da Länge der Kette nicht bekannt.

Lösung Teil 1: Common Table Expressions

Idee: Sub-Queries vor den Query stellen und benennen

Unterschied:

```
SELECT * FROM mr JOIN
(SELECT year, max(rating) rating FROM mr
 WHERE type = 'C' GROUP BY year) X
ON (mr.rating = X.rating AND mr.year = X.year)
```

wird zu:

```
WITH SQ AS
(SELECT year, max(rating) rating FROM mr
WHERE type = 'C' GROUP BY year)
SELECT * FROM mr JOIN SQ X ON
(mr.rating = X.rating AND mr.year = X.year)
```

} **Common Table Expression (CTE)**

Lösung Teil 2: Common Table Expressions erlauben <u>Rekursionen</u>

```
WITH SuccessorCTE(movie, successor) AS
(
      -- Startpunkt
      SELECT movie, successor
      FROM moviedb.follows WHERE movie = 1

      UNION ALL

      -- Rekursive Abfrage
      SELECT m.movie, m.successor
      FROM moviedb.follows m
      JOIN SuccessorCTE r ON (r.successor = m.movie)
)
SELECT * FROM SuccessorCTE
```

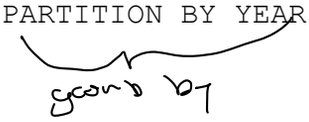
Analytische Funktionen

Idee:

- Funktionen über einem (ggf. sortierten) Teil der Ergebnismenge auswerten
- Im Gegensatz zu Funktionen, die nur über einer Zeile ausgewertet werden
- Ähnlich zu Aggregatsfunktionen, aber ohne `GROUP BY` Klausel

Beispiel: `ROW_NUMBER()`

- Nummeriert eine sortierte Ergebnismenge
- `ROW_NUMBER() OVER (ORDER BY rank DESC)`
- `ROW_NUMBER() OVER (PARTITION BY YEAR ORDER BY rank DESC)`



Anwendung

```
SELECT * FROM
(
      SELECT row_number() OVER ( PARTITION BY year
                                 ORDER BY rating DESC)rn,
            mr.*
      FROM mr
)
WHERE rn = 1
```

- Die Filme werden nach Jahr partitioniert
- dann nach Rating absteigend sortiert
- und dann pro Jahr durchnummeriert
- Es wird pro Jahr nur der erste Film (d.h. der mit dem höchsten Rating) selektiert

Idee hinter Analytischen Funktionen

- Funktion wird bezogen auf die ganze Partition ausgewertet
- Es wird aber trotzdem jede Zeile einzeln zurückgegeben

ID	Rating	Year	Funktion	
		2002		Partition 1
		2002		
		2002		
		2003		Partition 2
		2003		
		2004		Partition 3
		2004		
		2004		

Beispiele für Analytische Funktionen

Single Row Funktionen

`Funktion OVER (PARTITION BY Year ORDER BY Rating DESC)`

Funktion ist

- `Rank()`
- `Dense_Rank()`
- `Row_Number()`

ID	Rating	Year	Rank()	Dense_Rank()	Row_Number()
11	9.8	2002	1	1	1
2	9.7	2002	2	2	2
34	9.7	2002	2	2	3
16	8.6	2003	1	1	1
18	8.5	2003	2	2	2
75	8.8	2004	1	1	1
13	8.8	2004	1	1	2
9	8.6	2004	3	2	3

Vollständige Aggregationen

`Funktion OVER (PARTITION BY Year)`

Funktion ist

- `Sum(Rating)`
- `Count(*)`
- `Avg(Rating)`

ID	Rating	Year	Sum(Rating)	Count(*)	Avg(Rating)
11	9.8	2002	29.2	3	9.73
2	9.7	2002	29.2	3	9.73
34	9.7	2002	29.2	3	9.73
16	8.6	2003	17.1	2	8.55
18	8.5	2003	17.1	2	8.55
75	8.8	2004	26.2	3	8.73
13	8.8	2004	26.2	3	8.73
9	8.6	2004	26.2	3	8.73

Kumulierende Aggregationen

`Funktion OVER (PARTITION BY Year ORDER BY Rating DESC)`

Funktion ist

- `Sum(Rating)`
- `Count(*)`
- `Avg(Rating)`

ID	Rating	Year	Sum(Rating)	Count(*)	Avg(Rating)
11	9.8	2002	9.8	1	9.8
2	9.7	2002	29.2	3	9.73
34	9.7	2002	29.2	3	9.73
16	8.6	2003	8.6	1	8.6
18	8.5	2003	17.1	2	8.55
75	8.8	2004	17,6	2	8,8
13	8.8	2004	17,6	2	8,8
9	8.6	2004	26,2	3	8,73

Zusammenfassung Analytische Funktionen

- `ROW_NUMBER`: Fortlaufende Nummerierung innerhalb einer Partition Beispiel: 1, 2, 3, 4
- `RANK`: Wenn zwei Zeilen den gleichen Sortierschlüssel haben, bekommen die Zeilen den gleichen Rang; danach entsteht eine Lücke. Beispiel: 1, 2, 2, 4
- `DENSE_RANK`: Wie Rank, nur dass keine Lücke entsteht Beispiel: 1, 2, 2, 3
- `SUM` ohne Sortierung: Summierung über die ganze Partition
- `SUM` mit Sortierung: Summierung bis zur aktuellen Zeile inkl. aller Zeilen mit
- gleichem Sortierschlüssel
- `AVG, COUNT`: Analog zu SUM; Unterscheidung zwischen "mit Sortierung" und "ohne Sortierung"

Funktionen **DECODE, NVL, CASE**

Oracle:

```
DECODE(fakultaet,     'I', 'Elektro- und Informationstechnik',
                      ..., ...,
                      'IV', 'Wirtschaft und Informatik',
                      ..., ...,
                      'Unbekannte Fakultät');
```

- Ersetzungsliste mit Default

Standard-SQL:

```
CASE
      WHEN fakultaet = 'I' THEN 'Elektro- und Informationstechnik'
      WHEN fakultaet = 'IV' THEN 'Wirtschaft und Informatik'
      ...
      ELSE 'Unbekannte Fakultät'
END
```

NVL(x,y)

- Wenn x **NULL** ist, wird y geliefert, sonst x

Standard-SQL:

- **CASE WHEN** x **IS NULL THEN** y **ELSE** x **END**

Auch häufig verwendet:

- **COALESCE(x,y,z,...);**
- Liefert ersten Wert, der nicht NULL ist.

Zusammenfassung

SQL-Grundlagen

- Mengen vs. Multimengen

Join-Varianten:

- Cross-, Equi-, Theta-, (Left/Right/Full) Outer-, Natural-Join

Gruppierung und Aggregierung

- Select-Spalten bei Group By
- Unterschied WHERE und HAVING

Sortierung und Unterabfragen

- Order By
- Top-N Abfragen

Hierarchische Abfragen

- Common Table Expressions Analytische Funktionen