Teil 7: Data Warehouses und neuere SQL-Konstrukte

Literatur:

- Kemper/Eickler: Datenbanksysteme, 7. Aufl., Kap. 17, Oldenbourg, 2009.
- Elmasri/Navathe: Fundamentals of Database Systems, 3rd. Ed., Addison-Wesley, 2000. Kapitel 26: Data Warehousing And Data Mining (Seiten 841–872).
- ISO/IEC 9075-2: Information Technology - Database Languages - SQL - Part 2 (2003).
- Oracle Database SQL Language Reference, 11g Release 2 (11.2), Part No E26088-01. `http://download.oracle.com/docs/cd/E11882_01/server.112/e26088/toc.htm`
- Oracle Database Data Warehousing Guide, 11g Release 2 (11.2) Part No E25554-01. `http://docs.oracle.com/cd/E11882_01/server.112/e25554/toc.htm`
- IBM DB2 Database for Linux, UNIX and Windows — Informationszentrale. `http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp`
- Transact SQL Reference (Database Engine), SQL Server 2008 R2. `http://msdn.microsoft.com/en-us/library/bb510741.aspx`
- MySQL 5.1 Reference Manual. `http://dev.mysql.com/doc/refman/5.1/en/index.html`
- Chaudhuri, S./Dayal, U.: An Overview of Data Warehousing and OLAP Technology. SIGMOD Record, Vol. 26, No. 1, March 1997.
- Wikipedia: Data Warehouse, OLAP, OLAP Cube, Essbase, MDX. `http://en.wikipedia.org/wiki/Data_Warehouses`
- Stefan Deßloch: Recent Developments for Data Models (Folien, TU Kaiserslautern, 2010). `http://wwwlgis.informatik.uni-kl.de/cms/courses/datenmodelle/`

Lernziele

Nach diesem Kapitel sollten Sie Folgendes können:

- Unterschiede von OLAP- zu OLTP-Datenbanken erklären.

  Wofür stehen die beiden Abkürzungen?
- Das mehrdimensionale Datenmodell erklären.
- Die erweiterten `GROUP BY` Möglichkeiten in SQL verwenden (`CUBE`, `ROLLUP`).
- Top-n Anfragen in SQL schreiben können.
- Die Möglichkeiten von Window Funktions in SQL für eine konkrete Anwendung einschätzen.

Inhalt

1. Einführung in Data Warehouses
2. Fortgeschrittene Gruppierung: Cube, Rollup
3. Top-N Anfragen
4. Window Functions

Data Warehouses (1)

- Der Zweck eines Data Warehouses ist es, durch Zusammenstellung und Auswertung von Daten Manager in ihren Entscheidungen zu unterstützen.
- Z.B. können Verkaufszahlen in einem Supermarkt genutzt werden, um Fragen zu beantworten wie
  - ◇ Welche Waren sollen in welcher Menge bestellt werden (ggf. saisonal unterschiedlich)?
  - ◇ Ist der Effekt von Werbemaßnahmen und Sonderangeboten in den Verkaufzahlen spürbar?
  - ◇ Wie wirken Preisunterschiede zur Konkurrenz?

Data Warehouses (2)

- Kreditkartenunternehmen müssen
  - ◇ einen Kreditrahmen für Kunden festlegen,
  - ◇ Betrugsfälle möglichst schnell erkennen.
- Telekommunikationsunternehmen müssen u.a. folgende Entscheidungen treffen:
  - ◇ Welche Leitungen/Bandbreiten mieten?
  - ◇ Preise senken/erhöhen?
  - ◇ Neue Produkte anbieten?
  - ◇ Werbemaßnahmen durchführen? Zielgruppe?

Data Warehouses (3)

- Für alle diese Entscheidungen sind Daten nützlich, die das Unternehmen bereits hat oder hatte.

  Erfahrungen aus der Vergangenheit: Historische Daten, die für das aktuelle Tagesgeschäft nicht mehr wichtig sind.
- Zum Teil müssen auch Daten speziell für diese Analysen erfasst oder hinzugekauft werden.

  Preise der Konkurrenz, Einwohnerzahlen (nach Alter/Einkommen).
- Auch Daten des eigenen Unternehmens liegen z.T. in Word/Excel vor, und stehen nicht in der DB.

  Unterschiedliche Softwarepakete je nach Aufgabe / Unternehmensbereich: Ggf. jeweils eigene DB. Daten müssen integriert werden.

Data Warehouses (4)

- Data Warehouses dienen zum “Online Analytical Processing” (OLAP), also der Analyse der Daten.

  Sie speisen “Decision Support Systems”, “Data Mining” Anwendungen, “Executive Information Systems” mit Daten. Der ganze Bereich sind “Business Intelligence” Tools (Aufbereitung von Rohdaten in nützliche Informationen zum Zwecke der Entscheidungsfindung).
- Klassische Produktiv-Datenbanken dienen dagegen dem “Online Transaction Processing” (OLTP).

  Z.B. Abwicklung von Bestellungen. “Operatives Tagesgeschäft”.
- Die Anforderungen von OLAP und OLTP unterscheiden sich deutlich, deswegen werden meist unterschiedliche Datenbanken aufgebaut.

Data Warehouses (5)

OLAP vs. OLTP:

- Eine OLTP Datenbank spiegelt nur den aktuellen Stand des modellierten Weltausschnitts wieder, eine OLAP Datenbank enthält die ganze historische Entwicklung.
- OLAP Datenbanken sind meist sehr groß.
- Bei einer OLTP-Datenbank gibt es ständig kleine Updates (wenige Tupel), eine OLAP Datenbank muß nicht ganz aktuell sein.

  Typischerweise werden größere Datenmengen auf einmal in die OLAP-Datenbank geladen, z.B. einmal täglich oder seltener.

Data Warehouses (6)

OLAP vs. OLTP, Forts.:

- Eine OLTP-Datenbank wird fast nur über Anwendungsprogramme benutzt, so dass die Anfragen bekannt sind. Eine OLAP-Datenbank muß auch unerwartete Analysen (ad hoc Anfragen) erlauben.
- Anfragen in OLTP-Datenbanken benötigen meist nur wenige Tupel, OLAP-DBen müssen auch Auswertungen großer Datenmengen unterstützen.
- Eine OLAP-Datenbank muß eine integrierte Sicht auf Datenbestände aus mehreren Quellen bieten.

Data Cube (1)

- Im OLAP-Bereich werden die Daten häufig als n-dimensionaler Würfel aufgefasst ("Hypercube").

  Man kann das als neues Datenmodell verstehen oder als Visualisierung.

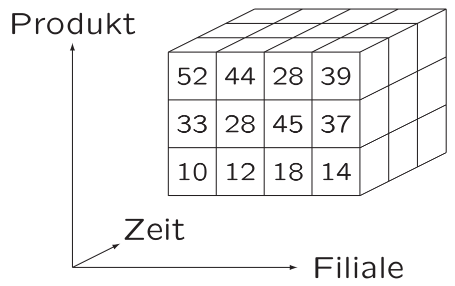
- Ein zweidimensionaler Würfel ist eine Matrix:

Aufgaben →

Studenten ↓	H1	H2	Z1
Lisa Weiss	10	8	12
Michael Grau	9	9	10
Daniel Sommer	5		7
Iris Winter			

Data Cube (2)

- Ein klassisches Beispiel sind Verkaufszahlen bzw. Umsätze abhängig von den drei Dimensionen Produkt, Filiale/Region und Zeit:



Data Cube (3)

- Die Einheiten für die verschiedenen Dimensionen sind hierarchisch strukturiert:
  - ◇ Studenten können nach Studiengang und/oder Semester zusammengefasst werden.

    Dieses und andere Beispiele zeigen, dass es nicht unbedingt eine lineare Hierarchie sein muss.
  - ◇ Aufgaben: Kategorie (Hausaufgaben, Zwischenklausur, u.s.w.), Thema, Schwierigkeitsgrad.
  - ◇ Produkte: Produktkategorien, Preisklasse.
  - ◇ Zeit: Tage, Wochen, Monate, Quartale, Jahre.
  - ◇ Filialen: Städte, Regionen, Bundesländer.

Data Cube (4)

- Typische Operationen mit Data Cubes sind:
  - ◇ Roll-up: Vergröbern des Detaillierungsgrades (“herauszoomen”), z.B. Übergang von einzelnen Filialen zu Regionen.

    Dabei findet eine Aggregierung der Daten statt (meist Summe, eventuell auch Durchschnitt).
  - ◇ Drill-down: Verfeinern des Detaillierungsgrades (“hineinzoomen”).
  - ◇ Rotation/Pivoting: Vertauschen von Achsen.

    Dabei können auch Achsen sichtbar werden bzw. nach hinten verschwinden. Die Darstellung von mehr als zwei Dimensionen ist ja schwierig.

Data Cube (5)

- Typische Operationen mit Data Cubes, Forts.:
  - ◇ Slicing: Herausschneiden einer Scheibe aus dem Daten-Würfel (Auswahl eines bestimmten Wertes in einer Dimension).
  - ◇ Dicing: Herausschneiden eines kleineren Würfels (Mengen von Werten/Intervalle ggf. in mehreren Dimensionen auswählen).
  - ◇ Drill-Accross: Wechsel auf einen anderen Attributwert (in einer Dimension, von der nur jeweils ein Wert angezeigt wird).

Data Cube (6)

- Die Daten heißen entsprechend auch “multidimensionale Daten”. Mathematisch sind sie eine Funktion von den Dimensionen in den abhängigen Wert:

  $$\text{Verkaufszahlen:}\ \text{Produkt} \times \text{Filiale} \times \text{Datum} \rightarrow \text{int}$$

- Oft enthält nicht jede Zelle des Würfels tatsächlich Daten (dünnbesetzte Matrix).

  Z.B. könnten Kunde und Produkt Dimensionen sein. Jeder Kunde kauft normalerweise nur wenige Produkte. Besonders bei hohen Dimensionen enthalten die meisten Zellen des Würfels die Zahl 0 oder einen Nullwert. Eine direkte Speicherung als Array, die sich zunächst anbieten würde, wird dann sehr ungünstig.

Data Cube (7)

- Bei Praktikern sind Spreadsheets sehr beliebt,
  - ◇ vermutlich sind multidimensionale Daten als Verallgemeinerung entstanden,
  - ◇ oder zur Verwaltung einer größeren Menge von untereinander abhängigen Spreadsheets.

    Ein wichtiges multidimensionales DBMS ist “Essbase” (ursprünglich von Arbor Software, dann Hyperion, jetzt Oracle). “Essbase” steht für “Extended Spread Sheet dataBASE”. Als Benutzerschnittstelle für Essbase wird häufig Microsoft Excel verwendet (es gibt ein entsprechendes add-in für Excel, und auch Möglichkeiten zum Datenzugriff für andere Office Produkte, z.B. Powerpoint). Andere Systeme zur Verwaltung multidimensionaler Daten sind z.B. Microsoft Analysis Services, IBM Cognos, Oracle DB OLAP Option, MicoStrategy, SAP Business Objects, icCube, Pentaho.

Stern-Schema (1)

- Natürlich kann man eine Funktion wie

  Verkaufszahlen: Produkt $\times$ Filiale $\times$ Datum $\rightarrow$ int

  auch als Tabelle darstellen:

Verkaufszahlen			
Produkt	Filiale	Datum	Stück
10112	HAL1	06.12.2011	5
⋮	⋮	⋮	⋮

- Für die einzelnen Dimensionen gibt weitere Tabellen, die insbesondere die Hierarchie-Information enthalten (Beispiel siehe nächste Folie).

Stern-Schema (2)

- Z.B. könnte eine Tabelle Daten über die Filialen enthalten, die sich für die Zusammenfassung und Analyse der Verkaufszahlen eignen:

Filialen				
Filiale	Stadt	Bundesland	Lage	Größe
HAL1	Halle	Sachsen-Anhalt	Innenstadt	1000
⋮	⋮	⋮	⋮	⋮

Selbst zu Datums-Werten kann man noch weitere Information speichern (Feiertage, Wetter, etc.). Wochentag und Monat muß man nicht speichern, die könnte man berechnen. Vermutlich wäre eine Sicht (virtuelle Tabelle) mit entsprechenden Spalten aber zur Vereinheitlichung nützlich.

Stern-Schema (3)

- Ein Stern-Schema ("star schema") ist im Data Warehouse-Bereich üblich und besteht aus
  - ◇ einer großen Fakten-Tabelle ("fact table"), deren Schlüssel aus mehreren Fremdschlüsseln zusammengesetzt ist, die auf
  - ◇ mehrere kleinere Dimensions-Tabellen ("dimension tables") verweisen.

    Enthalten die Dimensionstabellen noch Fremdschlüssel auf weitere Tabellen (z.B. Informationen über Produktgruppen), so spricht man von einem Schneeflocken-Schema ("snowflake schema").

Inhalt

1. Einführung in Data Warehouses
2. Fortgeschrittene Gruppierung: Cube, Rollup
3. Top-N Anfragen
4. Window Functions

Beispiel-Datenbank (1)

STUDENTEN

SID	VORNAME	NACHNAME	EMAIL
101	Lisa	Weiss	···
102	Michael	Grau	NULL
103	Daniel	Sommer	···
104	Iris	Winter	···

BEWERTUNGEN

SID	ATYP	ANR	PUNKTE
101	H	1	10
101	H	2	8
101	Z	1	12
102	H	1	9
102	H	2	9
102	Z	1	10
103	H	1	5
103	Z	1	7

AUFGABEN

ATYP	ANR	THEMA	MAXPT
H	1	ER	10
H	2	SQL	10
Z	1	SQL	14

Beispiel-Datenbank (2)

- Hier interessieren uns die Prozent der Maximalpunktzahl, die pro Student und Aufgabe erreicht wurden, wobei die Aufgabe durch Typ (Hausaufgabe, Klausur, etc.) und Nummer identifiziert wird:

```
CREATE VIEW ERGEBNISSE(SID, VORNAME, NACHNAME,
                      ATYP, ANR, PCT) AS
SELECT S.SID, S.VORNAME, S.NACHNAME,
       A.ATYP, A.ANR, ROUND(B.PUNKTE*100/A.MAXPT)
FROM   STUDENTEN S, BEWERTUNGEN B, AUFGABEN A
WHERE  B.SID = S.SID
AND    B.ATYP = A.ATYP AND B.ANR = A.ANR
```


Beispiel-Datenbank (3)

ERGEBNISSE

SID	VORNAME	NACHNAME	ATYP	ANR	PCT
101	Lisa	Weiss	H	1	100
101	Lisa	Weiss	H	2	80
101	Lisa	Weiss	Z	1	86
102	Michael	Grau	H	1	90
102	Michael	Grau	H	2	90
102	Michael	Grau	Z	1	71
103	Daniel	Sommer	H	1	50
103	Daniel	Sommer	Z	1	50

Motivation/Beispiel (1)

- Manchmal sollen Daten nach verschiedenen Kriterien gruppiert ausgewertet werden.

  Z.B. durchschnittlich erreichte Prozentzahl nach Student, nach Aufgabentyp, nach Aufgabe, oder über alle Aufgaben und Teilnehmer.

- Bisher braucht man eine Anfrage pro Gruppierung, bzw. kann diese Anfragen auch mit `UNION ALL` zusammensetzen, wenn man die in einem Teil nicht relevanten Gruppierungsattribute z.B. mit Nullwerten auffüllt.

  Sonst hätten die Teilanfragen ja unterschiedliche Schemata (Spaltenmengen).

Motivation/Beispiel (2)

- Durchschnittspunkte pro Aufgabe (`ATYP` und `ANR`), pro Aufgabentyp (`ATYP`), und insgesamt:

ATYP	ANR	AVG(PCT)	
H	1	80	
H	2	85	
H		82	← Durchschnitt über alle H-Aufg.
Z	1	69	
Z		69	← Durchschnitt über alle Z-Aufg.
		77	← Durchschnitt insgesamt

- Der Nullwert bedeutet hier "*" (beliebig).
- Zugehörige SQL-Anfrage auf der nächsten Folie.

Motivation/Beispiel (3)

```
SELECT   ATYP, ANR, AVG(PCT)
FROM     ERGEBNISSE
GROUP BY ATYP, ANR
UNION    ALL
SELECT   ATYP, TO_NUMBER(NULL) AS ANR, AVG(PCT)
FROM     ERGEBNISSE
GROUP BY ATYP
UNION    ALL
SELECT   TO_CHAR(NULL) AS ATYP,
         TO_NUMBER(NULL) AS ANR, AVG(PCT)
FROM     ERGEBNISSE
ORDER BY ATYP NULLS LAST, ANR NULLS LAST
```


Motivation (4)

- Die im folgenden eingeführten Konstrukte (`CUBE`, `ROLLUP`) vereinfachen die Formulierung, bieten aber keine grundsätzlich neuen Möglichkeiten.

  Dies entspricht dem Outer Join: Er erhöht die Ausdrucksfähigkeit auch nicht, ist aber praktisch.

- Die Implementierung ist oft effizienter als bei der Lösung mit `UNION ALL`.

  Z.B. nur einmal sortieren statt ein Mal pro Teilanfrage. Ein intelligenter Optimierer könnte dies aber auch bei der äquivalenten Anfrage mit `UNION ALL` machen, es ist nur schwieriger, dort genau diesen Spezialfall zu erkennen.

Motivation/Beispiel (5)

- Folgende Anfrage liefert das gleiche Ergebnis:

```
SELECT   ATYP, ANR, AVG(PCT)
FROM     ERGEBNISSE
GROUP BY ROLLUP(ATYP, ANR)
```

  Die Sortierung stimmt bei Oracle 10g zufällig, das ist aber nicht allgemein garantiert. Man muß also korrekterweise noch die `ORDER BY`-Klausel wie oben hinzufügen.

- Es werden also drei Gruppierungen ausgeführt:
  - ◇ Nach (`ATYP, ANR`)
  - ◇ Nach (`ATYP`)
  - ◇ Nach (): Aggregation über ganze Menge.

ROLLUP (1)

- Allgemein wirkt

```
GROUP BY ROLLUP(A1, ..., An)
```

wie `n+1` einzelne Gruppierungen, wobei in jedem Schritt ein Attribut mehr rechts weggelassen wird, bis zur Aggregierung ohne `GROUP BY` (über alles).

- Man schreibt also die größten Einheiten der Hierarchie auf die linke Seite und verfeinert nach rechts:

```
GROUP BY ROLLUP(JAHR, MONAT, TAG)
```

Dies liefert die Werte für jeden Tag, Zwischen“summen” für die Monate, für die Jahre, und schließlich insgesamt (Aggregation über alle Zeilen). Die Aggregationsfunktion kann man wählen (nicht nur `SUM`).

ROLLUP (2)

- Wie üblich kann man die Attribute der `GROUP BY`-Klausel (`A1,...,An`) unter `SELECT` außerhalb von Aggregationen verwenden.
- Falls für eine Ergebnis-Zeile nicht wirklich nach dem Attribut `Ai` gruppiert worden ist, hat `Ai` keinen eindeutigen Wert.

  Alle Ergebniszeilen, bei denen nicht nach vollständigen Attributliste `A1,...,An` gruppiert wurde, heißen auch “superaggregate rows”.

- Unter `SELECT` wird dann ein Nullwert für dieses Attribut eingesetzt.

GROUPING (1)

- Falls der Nullwert auch in den Daten vorkommt, wäre die Bedeutung in der Ausgabe nicht eindeutig.
- Deswegen gibt es die Funktion `GROUPING`, die die “beiden” Nullwerte auseinanderhält:
  - ◇ `GROUPING(Ai)=1`, wenn nach dem Attribut `Ai` nicht gruppiert wurde, d.h. der Nullwert für `Ai` die Menge aller Werte repräsentiert.
  - ◇ `GROUPING(Ai)=0`, wenn nach `Ai` gruppiert wurde.

    Falls `Ai` ein Nullwert sein sollte, handelt es sich um einen gewöhnlichen Nullwert, der aus den Daten stammt.

GROUPING (2)

- Wenn man z.B. ein “*” in den Summenzeilen ausgeben will, funktioniert das bei Oracle so:

```
SELECT   DECODE(GROUPING(ATYP), 1, '*', ATYP)
             AS ATYP,
         DECODE(GROUPING(ANR),  1, '*', ANR)
             AS ANR,
         AVG(PCT)
FROM     ERGEBNISSE
GROUP BY ROLLUP(ATYP, ANR)
ORDER BY ATYP, ANR
```

Zur Erinnerung: `DECODE(A,B,C,D)` liefert `C`, falls `A=B`, sonst `D`. Im Beispiel wäre die Verwendung von `GROUPING` nicht nötig, da die Daten keine Nullwerte in `GROUP BY`-Attributen enthalten. Z.B. kürzer: `NVL(ATYP,'*')`.

CUBE (1)

- Mit CUBE kann man über jede Teilmenge einer Menge von Attributen aggregieren:

```
SELECT   SID, ATYP, ANR, AVG(PCT)
FROM     ERGEBNISSE
GROUP BY CUBE(SID, ATYP)
```

- Dies berechnet den durchschnittlichen %-Wert für
  - ◇ jeden Studenten (SID) und Aufgabentyp (ATYP),
  - ◇ jeden Studenten SID (über alle Aufgabentypen),
  - ◇ jeden Aufgabentyp ATYP (über alle Studenten),
  - ◇ insgesamt (alle Studenten und Aufgabentypen)

CUBE (2)

SID	ATYP	AVG(PCT)	
101	H	90	
101	Z	86	
101		89	← Durchschnitt für Studenten 101
102	H	90	
102	Z	71	
102		84	← Durchschnitt für Studenten 102
103	H	50	
103	Z	50	
103		50	← Durchschnitt für Studenten 103
	H	82	← Durchschnitt über alle H-Aufg.
	Z	69	← Durchschnitt über alle Z-Aufg.
		77	← Durchschnitt insgesamt

CUBE (3)

- Übersichtlichere Visualisierung der Daten:

	101	102	103	AVG
H	90	90	50	82
Z	86	71	50	69
AVG	89	84	50	77

Rechter Rand: Gesamtwerte für Aufgabentyp, Unterer Rand: Gesamtwerte für Studenten, Rechts unten: Gesamtwert für ganze Tabelle.

Man kann keine SQL-Anfrage schreiben, die diese Tabelle direkt aus den Daten liefert, weil die Ergebnisspalten jeder SQL-Anfrage fest sind (unabhängig von den Daten). Die Daten sind aber alle im Ergebnis der CUBE-Anfrage enthalten, und ein einfaches Anwendungsprogramm kann die Darstellung entsprechend umformen (man muß hier ORDER BY ATYP NULLS LAST, SID NULLS LAST verwenden, damit die Zahlen von der Datenbank in der richtigen Reihenfolge geliefert werden).

CUBE (4)

- Allgemein wirkt

```
GROUP BY CUBE(A1, ..., An)
```

wie 2^n einzelne Gruppierungen (jede Teilmenge).

- Man kann aber auch Attribute zusammenfassen, so dass sie nur gemeinsam bei der Gruppierung verwendet werden:

```
SELECT   SID, ATYP, ANR, AVG(PCT)
FROM     ERGEBNISSE
GROUP BY CUBE(SID, (ATYP, ANR))
```

Dies bewirkt vier einzelne Gruppierungen (nicht 8). Anfrageergebnis siehe nächste Folie. Solche Gruppierungen gehen auch bei ROLLUP.

CUBE (5)

SID	ATYP	ANR	AVG(PCT)	
101	H	1	100	
101	H	2	80	
101	Z	1	86	
101			89	← Durchschnitt für Student 101
102	H	1	90	
102	H	2	90	
102	Z	1	71	
102			84	← Durchschnitt für Student 102
103	H	1	50	
103	Z	1	50	
103			50	← Durchschnitt für Student 103
	H	1	80	← Durchschnitt für Aufgabe H1
	H	2	85	← Durchschnitt für Aufgabe H2
	Z	1	69	← Durchschnitt für Aufgabe Z1
			77	← Gesamt-Durchschnitt

GROUPING SETS (1)

- Mit `GROUPING SETS` kann man Mengen von Gruppierungsattributen einzeln auswählen.

  Das würde man natürlich nur machen, wenn `ROLLUP` und `CUBE` nicht exakt die gewünschten Aggregationen liefern, denn es ist mehr Tippaufwand. Jedes "Grouping Set" entspricht einem "`GROUP BY`".

- Z.B. Gruppierung einerseits nach `SID` und andererseits nach der Kombination `ATYP`, `ANR`:

```
SELECT   SID, ATYP, ANR, AVG(PCT)
FROM     ERGEBNISSE
GROUP BY GROUPING SETS((SID), (ATYP, ANR))
```

  Anfrageergebnis auf der nächsten Folie.

GROUPING SETS (2)

SID	ATYP	ANR	AVG(PCT)	
101			89	← Durchschnitt für Student 101
102			84	← Durchschnitt für Student 102
103			50	← Durchschnitt für Student 103
	H	1	80	← Durchschnitt für Aufgabe H1
	H	2	85	← Durchschnitt für Aufgabe H2
	Z	1	69	← Durchschnitt für Aufgabe Z1

- Man kann `CUBE` und `ROLLUP` mit `GROUPING SETS` nachbilden, z.B. entspricht `ROLLUP(ATYP, ANR)`:

```
GROUPING SETS((ATYP, ANR), (ATYP), ())
```


Kombinationen

- Man kann unter `GROUP BY` auch mehrere einzelne Gruppierungsspezifikationen angeben.
- Jede solche Spezifikation bestimmt eine Menge von Mengen von Attributen.
- Es wird das kartesische Produkt dieser Mengen gebildet, und die Mengen jedes Tupels vereinigt.
- Beispiel:

```
GROUP BY A, CUBE(B, C)
```

  ist äquivalent zu

```
GROUPING SETS ((A,B,C), (A,B), (A,C), (A))
```

Inhalt

1. Einführung in Data Warehouses
2. Fortgeschrittene Gruppierung: Cube, Rollup
3. Top-N Anfragen
4. Window Functions

Beispiel-Datenbank

STUDENTEN

SID	VORNAME	NACHNAME	EMAIL
101	Lisa	Weiss	···
102	Michael	Grau	NULL
103	Daniel	Sommer	···
104	Iris	Winter	···

AUFGABEN

ATYP	ANR	THEMA	MAXPT
H	1	ER	10
H	2	SQL	10
Z	1	SQL	14

BEWERTUNGEN

SID	ATYP	ANR	PUNKTE
101	H	1	10
101	H	2	8
101	Z	1	12
102	H	1	9
102	H	2	9
102	Z	1	10
103	H	1	5
103	Z	1	7

"Erste n" Anfragen (1)

- Gesucht sind die beiden Studenten mit dem besten Ergebnis für Hausaufgabe 1 (allgemein: die ersten n bezüglich einer Ordnung: "Top-N Queries").
- Solche Anfragen sind mit den Möglichkeiten von SQL-92 recht schwierig zu lösen.

  Man kann zählen, wie viele ein schlechteres Ergebnis haben (s.u.).
- Da solche Anfragen aber relativ häufig vorkommen, haben viele Systeme ein spezielles Konstrukt dafür.

  Die Systeme unterscheiden sich in diesem Punkt aber stark (Portabilitätsproblem). Erst im SQL:2008 Standard wurde die `FETCH FIRST`-Klausel aufgenommen.

"Erste n" Anfragen (2)

- Erste n (hier $n = 2$) Studenten bezüglich der Punkte in Hausaufgabe 1 (bei gleicher Punktzahl: SID):

```
SELECT S.VORNAME, S.NACHNAME, B.PUNKTE
FROM   STUDENTEN S, BEWERTUNGEN B
WHERE  S.SID = B.SID
AND    B.ATYP = 'H' AND B.ANR = 1
AND    2 > (SELECT COUNT(*) FROM BEWERTUNGEN X
            WHERE  X.ATYP = 'H' AND X.ANR = 1
            AND    (X.PUNKTE > B.PUNKTE OR
                    X.PUNKTE = B.PUNKTE AND
                       X.SID < B.SID))
ORDER  BY B.PUNKTE DESC
```

"Erste n" Anfragen (3)

- Wie oben erläutert, kann man bei neueren Datenbanken die Unteranfrage auch links schreiben:

```
(SELECT COUNT(*) ...) < 2
```

- Die Formel

```
(X.PUNKTE > B.PUNKTE OR
 X.PUNKTE = B.PUNKTE AND X.SID < B.SID)
```

  entspricht dem Sortierkriterium

```
ORDER BY PUNKTE DESC, SID
```

  Es werden also die X gezählt, die mehr Punkte als B haben, oder bei gleicher Punktzahl eine kleinere SID.

"Erste n" Anfragen (4)

- Falls man bei gleicher Punktzahl doch alle auflisten will, also ggf. auch mehr als n (hier $n = 2$):

```
SELECT S.VORNAME, S.NACHNAME, B.PUNKTE
FROM   STUDENTEN S, BEWERTUNGEN B
WHERE  S.SID = B.SID
AND    B.ATYP = 'H' AND B.ANR = 1
AND    2 > (SELECT COUNT(*) FROM BEWERTUNGEN X
            WHERE  X.ATYP = 'H' AND X.ANR = 1
            AND    X.PUNKTE > B.PUNKTE)
ORDER  BY B.PUNKTE DESC
```


"Erste n" Anfragen (5)

- Oracle hat die 0-stellige Funktion ROWNUM, die eindeutige Nummern für die Ausgabezeilen erzeugt (beginnend mit 1).

  In der Oracle Dokumentation findet man ROWNUM im Abschnitt "Pseudocolumns", d.h. virtuelle Spalten, die bei Bedarf berechnet werden. In diesem Fall wäre es allerdings eine Spalte der Ausgabetabelle.

- Z.B. kann man damit die Anzahl der Ausgabezeilen begrenzen:

```
SELECT SID, PUNKTE
FROM   BEWERTUNGEN
WHERE  ATYP = 'H' AND ANR = 1
AND    ROWNUM <= 2
```


"Erste n" Anfragen (6)

- Die obige Anfrage gibt zwei Bewertungen für Hausaufgabe 1 aus. Es ist nicht vorhersehbar, welche Bewertungen ausgegeben werden.

  Die Auswahl hängt von internen Algorithmen des Anfrageoptimierers ab und kann sich von einer Oracle-Version zur nächsten ändern.

  Genauer funktioniert ROWNUM folgendermaßen: Es wird ein Zähler verwendet, der mit 1 initialisiert ist. ROWNUM liefert immer den aktuellen Wert des Zählers. Falls eine Tupelkombination die WHERE-Bedingung erfüllt hat, wird der Zähler hochgesetzt.

  ROWNUM bezieht sich nicht eigentlich auf Ausgabezeilen. Der Unterschied wird bei "GROUP BY"-Anfragen deutlich: Hier kann ROWNUM unter WHERE, aber nicht unter SELECT verwendet werden. Bei Anfragen ohne "GROUP BY" kann es natürlich auch unter SELECT verwendet werden.

"Erste n" Anfragen (7)

- Da "ORDER BY" erst nach der WHERE-Bedingung ausgewertet wird, hat es keinen Einfluss auf ROWNUM.

  Möglicherweise wählt der Optimierer aber einen anderen Auswertungsplan, was dann die ausgewählten Zeilen verändern könnte.

- Es geht aber mit einer Unteranfrage:

```
SELECT SID, PUNKTE
FROM   (SELECT SID, PUNKTE
        FROM   BEWERTUNGEN
        WHERE  ATYP = 'H' AND ANR = 1
        ORDER  BY PUNKTE DESC, SID)
WHERE  ROWNUM <= 2
```

ORDER BY in Unteranfragen ist erst seit SQL:2008 standard-konform.

"Erste n" Anfragen (8)

- Beachte: ROWNUM > 1 kann nicht funktionieren!

  Da die Bedingung am Anfang nicht erfüllt ist, wird ROWNUM nicht erhöht, ist also immer 1.

- Man kann ROWNUM aber in einer Unteranfrage generieren, und die Bedingung außen formulieren, z.B.:

```
SELECT SID, PUNKTE
FROM   (SELECT ROWNUM AS NR, SID, PUNKTE
        FROM   (SELECT *
                FROM   BEWERTUNGEN
                WHERE  ATYP = 'H' AND ANR = 1
                ORDER  BY PUNKTE DESC, SID))
WHERE  NR > 3 AND NR <= 5
```


"Erste n" Anfragen (9)

- In DB2 und dem SQL:2008 Standard wird die gleiche Aufgabe (die beiden Studenten mit der besten Punktzahl für Hausaufgabe 1) so gelöst:

```
SELECT SID, PUNKTE
FROM   BEWERTUNGEN
WHERE  ATYP = 'H' AND ANR = 1
ORDER  BY PUNKTE DESC
FETCH  FIRST 2 ROWS ONLY
```

- Wenn man nur das erste Antworttupel will, kann man auch "FETCH FIRST ROW ONLY" schreiben.

  Die FETCH FIRST-Klausel kommt ganz ans Ende der Anfrage.

"Erste n" Anfragen (10)

- In der Wikipedia (Stichwort SELECT_(SQL), 03.11.11) steht, dass Oracle ab Version 8i auch den Standard unterstützt. Das stimmt nicht.

  Ich habe es mit Version 10g ausprobiert: Es funktioniert nicht. Es findet sich auch nicht in der 11g SQL Reference.

- In der Wikipedia steht eine Übersicht mit 11 syntaktischen Varianten für 16 verschiedene Systeme.

  Das zeigt zumindest die Portabilitätsprobleme.

"Erste n" Anfragen (11)

- Microsoft SQL Server:

```
SELECT TOP (2) SID, PUNKTE
FROM   BEWERTUNGEN
WHERE  ATYP = 'H' AND ANR = 1
ORDER  BY PUNKTE DESC
```

Aus Gründen der Abwärtskompatibilität kann man die Klammern um die Anzahl der Zeilen auch weglassen (nicht empfohlen).

- `TOP (10) PERCENT`: erste 10% des Ergebnisses.
- `TOP (2) WITH TIES`: auch mehr als 2 Zeilen, wenn die folgenden Zeilen mit der letzten gewählten Zeile in den `ORDER BY`-Attributen übereinstimmen.

"Erste n" Anfragen (12)

- In MySQL verwendet man die `LIMIT`-Klausel:

```
SELECT SID, PUNKTE
FROM   BEWERTUNGEN
WHERE  ATYP = 'H' AND ANR = 1
ORDER  BY PUNKTE DESC
LIMIT  2
```

- MySQL bietet außerdem die Möglichkeit, nicht bei der ersten Zeile zu beginnen, sondern ab einer vorgegebenen Position im Anfrageergebnis:

```
LIMIT 2 OFFSET 3
```

Überspringt drei Zeilen und gibt dann zwei aus.

"Erste n" Anfragen (13)

- Dieses Konstrukt ist z.B. nützlich, wenn man in einer Web-Schnittstelle eine lange Tabelle auf mehrere Seiten verteilen muss.
- Beachte: Die erste Zeile hat den Offset 0.

```
LIMIT 2 OFFSET 3
```

Dies gibt die vierte und fünfte Zeile des Anfrageergebnisses aus.

- Statt `LIMIT` L `OFFSET` O kann man auch `LIMIT` O, L schreiben (im Beispiel: `LIMIT 3, 2`).

  Es gibt keine Möglichkeit, nur den Offset anzugeben. Das Handbuch erklärt, dass man für `LIMIT` ja einen sehr großen Wert angeben kann.

"Erste n" Anfragen (14)

- Der Anfrageoptimierer kann das Wissen nutzen, dass sich der Benutzer nur für eine oder wenige Ergebniszeilen interessiert.

  Z.B. "nested loop join" statt "merge join" (mit vorgeschaltetem Sortieren), siehe Vorlesung "Datenbanken II".

- Es macht also einen Unterschied,
  - ◇ ob man die spezielle Klausel verwendet,
  - ◇ oder die Anfrage stellt, und dann nur die ersten n Antworten von der Datenbank abholt.

    Das Anwendungsprogramm enthält eine Schleife über den Tupeln im Anfrageergebnis, die man natürlich vorzeitig abbrechen kann. Ggf. spart man mit "`FETCH FIRST`" auch Netzwerkverkehr.

Inhalt

1. Einführung in Data Warehouses

2. Fortgeschrittene Gruppierung: Cube, Rollup

3. Top-N Anfragen

4. Window Functions

Einführung

- Im SQL:2003 Standard wurden "Window Functions" eingeführt, die Aggregation innerhalb eines Fensters ausführen können, das über die sortierten Daten läuft (bei Oracle: "Analytic Functions").
- Damit kann man z.B. Aktienkurse glätten, indem man 3-Tage Durchschnittswerte berechnet.
- Man kann aber auch Positionen innerhalb der geordneten Liste bestimmen.

  Das entspricht einem `count(*)` über einem Fenster, dessen Anfang am Anfang der Liste verankert ist, aber dessen Ende über die Liste hinwegläuft — des Fenster wird also immer größer.

Ranking (1)

- Z.B. alle Abgaben für Hausaufgabe 1 mit
  - Position in der Sortierreihenfolge bzgl. Punkten,
  - laufender Studentennummer, und

    Position bei Sortierung nach `SID`: Nich besonders sinnvoll, soll aber verschiedene Sortierungen in einer Anfrage demonstrieren.
  - die Ausgabe sortiert nach Namen:

```
SELECT S.NACHNAME, S.VORNAME, B.PUNKTE,
       RANK() OVER (ORDER BY B.PUNKTE DESC),
       RANK() OVER (ORDER BY S.SID)
FROM   STUDENTEN S, BEWERTUNGEN B
WHERE  S.SID = B.SID
AND    B.ATYP = 'H' AND B.ANR = 1
ORDER  BY S.NACHNAME, S.VORNAME
```


Ranking (2)

- `RANK()` ist 1 plus die Anzahl der Tupel, die in den `ORDER BY` Attributen einen echt kleineren Wert haben (bzw. echt größer bei `DESC`).

  Es werden also Tupel mit gleichen Werten in den `ORDER BY`-Attributen ("Peers") nicht mitgezählt. Das führt zu Lücken in der Numerierung. Nach der Lücke synchronisiert sich `RANK()` wieder mit `ROW_NUMBER()`.
- `DENSE_RANK()` ist 1 plus die Anzahl von echt kleineren Werten in den `ORDER BY` Attributen.

  So gibt es keine Lücken, aber Werte sind keine Tupelanzahlen mehr.
- `ROW_NUMBER()` numeriert die Zeilen durch.

  Beginnend mit 1. Bei gleichen Werten in den `ORDER BY` Attributen ist die Numerierung nichtdeterministisch.

Ranking (3)

- Beispiel (falls `ORDER BY PUNKTE DESC`):

NAME	PUNKTE	RANK	DENSE_RANK	ROW_NUMBER
Anna	25	1	1	1
Bettina	20	2	2	2
Christine	20	2	2	3
Dorothea	15	4	3	4
Elisabeth	15	4	3	5
Felicia	15	4	3	6
Georgia	10	7	4	7

Ranking (4)

- `PERCENT_RANK()`: Berechnet als `(RANK()-1)/(N-1)`.

  Dabei ist `N` die Gesamtanzahl der Zeilen (in der Partition, s.u.).

- `CUME_DIST()`: Berechnet als `B/N`.

  Dabei ist `B` die Anzahl von Zeilen mit gleichem oder kleineren Wert, `N` wie oben ("cumulative distribution").

NAME	PUNKTE	RANK	PERCENT_RANK	CUME_DIST
Anna	25	1	0.00	0.14
Bettina	20	2	0.17	0.43
Christine	20	2	0.17	0.43
Dorothea	15	4	0.50	0.86
Elisabeth	15	4	0.50	0.86
Felicia	15	4	0.50	0.86
Georgia	10	7	1.00	1.00

Ranking (5)

- Man kann bei der Sortierung auch Aggregationsterme verwenden (alle Terme, die auch sonst unter `SELECT` möglich wären):

```
SELECT S.NACHNAME, S.VORNAME, SUM(B.PUNKTE),
       RANK() OVER (ORDER BY SUM(B.PUNKTE) DESC)
FROM   STUDENTEN S, BEWERTUNGEN B
WHERE  S.SID = B.SID
AND    B.ATYP = 'H'
GROUP  BY S.SID, S.NACHNAME, S.VORNAME
ORDER  BY S.NACHNAME, S.VORNAME
```

  Es ist nicht verlangt, dass die Terme, nach denen in der `OVER`-Klausel sortiert wird, auch unter `SELECT` ausgegeben werden.

Partitionierung (1)

- Man kann das Anfrage-Ergebnis auch erst partitionieren (ähnlich zu `GROUP BY`) und dann die Position innerhalb jeder Partition berechnen:

```
SELECT S.NACHNAME, S.VORNAME, B.ANR, B.PUNKTE,
       RANK() OVER (PARTITION BY B.ANR
                    ORDER BY B.PUNKTE DESC)
FROM   STUDENTEN S, BEWERTUNGEN B
WHERE  S.SID = B.SID
AND    B.ATYP = 'H'
ORDER  BY S.NACHNAME, S.VORNAME, B.ANR
```

- Hier wird die Position (nach Punkten) für jede Aufgabe einzeln bestimmt (Ergebnis: nächste Folie).

Partitionierung (2)

- Ergebnis der Anfrage auf der letzten Folie:

NAME	VORNAME	ANR	PUNKTE	RANK
Grau	Michael	1	9	2
Grau	Michael	2	9	1
Sommer	Daniel	1	5	3
Weiss	Lisa	1	10	1
Weiss	Lisa	2	8	2

Z.B. für Hausaufgabe 1 ist Lisa Weiss mit 10 Punkten Erste, dann kommt Michael Grau mit 9 Punkten und dann Daniel Sommer mit 5 Punkten.

In der anderen Partition (Aufgabe 2) kommt Michael Grau (10 Punkte) vor Lisa Weiss (8 Punkte). Daniel Sommer hat nichts abgegeben.

Partitionierung (3)

- Auch die üblichen Aggregations-Funktionen können im OVER-Konstrukt verwendet werden:

```
SELECT S.NACHNAME, S.VORNAME, B.ANR, B.PUNKTE,
       AVG(PUNKTE) OVER (PARTITION BY B.ANR)
FROM   STUDENTEN S, BEWERTUNGEN B
WHERE  S.SID = B.SID
AND    B.ATYP = 'H'
ORDER  BY S.NACHNAME, S.VORNAME, B.ANR
```

- Auf diese Weise wird in jeder Zeile auch der Punkte-Durchschnitt über alle Abgaben für diese Aufgabe angezeigt (Ergebnis siehe nächste Folie).

Partitionierung (4)

- Die obige Anfrage verwendet implizit ein "Fenster" (Eingabe der Aggregationsfunktion), das sich über alle Tupel der Partition erstreckt.
- Daher ist der berechnete Durchschnitt für alle Bewertungen einer Aufgabe gleich:

NACHNAME	VORNAME	ANR	PUNKTE	AVG
Grau	Michael	1	9	8.0
Grau	Michael	2	9	8.5
Sommer	Daniel	1	5	8.0
Weiss	Lisa	1	10	8.0
Weiss	Lisa	2	8	8.5

Partitionierung (5)

- Hier zeigt sich ein Unterschied zu GROUP BY:
  - ◇ Bei GROUP BY wird nur eine Ausgabe pro Gruppe erzeugt. Es ist nicht mehr möglich, auf Werte von Attributen, nach denen nicht gruppiert wird, einzeln zuzugreifen.
  - ◇ Mit dem OVER-Konstrukt wird dagegen eine Ausgabe pro Tupel erzeugt. Man behält daher den vollen Zugriff auf alle Attribute, kann aber das Tupel in einem Kontext sehen. Man bekommt so aggregierten Zugriff auf andere Tupel.

Fenster/Windows (1)

- Ein `OVER`-Ausdruck, z.B. `OVER (PARTITION BY B.ANR)` definiert zu jeder Zeile (bzw. Variablenbelegung) im Ergebnis des "`FROM...WHERE...GROUP...HAVING...`" ein Fenster (im Beispiel die ganze Partition):

NACHNAME	VORNAME	ANR	PUNKTE	
Weiss	Lisa	1	10	← akt. Zeile
Grau	Michael	1	9	Fenster
Daniel	Sommer	1	5	← Partitionsgrenze
Lisa	Weiss	2	8	
Grau	Michael	2	9	

Fenster/Windows (2)

- Im Beispiel bekommt man für jede der ersten drei Zeilen das gleiche Fenster (die ganze Partition, zu der die aktuelle Zeile gehört).
- Daher liefert z.B.

  `AVG(PUNKTE) OVER (PARTITION BY B.ANR)`

  für diese drei Zeilen den gleichen Wert (8).
- Das ändert sich schon, wenn man zusätzlich eine `ORDER BY` Klausel verwendet, siehe nächste Folie.

Fenster/Windows (3)

- Mit Sortierung, z.B.

  `OVER (PARTITION BY B.ANR ORDER BY B.PUNKTE DESC)`

  erstreckt sich das Fenster vom Anfang der Partition bis zur aktuellen Zeile (plus Peers/Ties, s.u.):

NACHNAME	VORNAME	ANR	PUNKTE	
Weiss	Lisa	1	10	Fenster ← a.Z.
Grau	Michael	1	9	
Daniel	Sommer	1	5	← Partitionsgrenze
Grau	Michael	2	9	
Weiss	Lisa	2	8	

Fenster/Windows (4)

- Fortsetzung des Beispiels:

  `OVER (PARTITION BY B.ANR ORDER BY B.PUNKTE DESC)`

  Fenster für die zweite Zeile:

NACHNAME	VORNAME	ANR	PUNKTE	
Weiss	Lisa	1	10	Fenster
Grau	Michael	1	9	← akt. Zeile
Daniel	Sommer	1	5	← Partitionsgrenze
Grau	Michael	2	9	
Weiss	Lisa	2	8	

Fenster/Windows (5)

- Fortsetzung des Beispiels:

  `OVER (PARTITION BY B.ANR ORDER BY B.PUNKTE DESC)`

  Fenster für die dritte Zeile:

NACHNAME	VORNAME	ANR	PUNKTE	
Weiss	Lisa	1	10	
Grau	Michael	1	9	Fenster
Daniel	Sommer	1	5	← akt. Zeile
				← Partitionsgrenze
Grau	Michael	2	9	
Weiss	Lisa	2	8	

Fenster/Windows (6)

- Fortsetzung des Beispiels:

  `OVER (PARTITION BY B.ANR ORDER BY B.PUNKTE DESC)`

  Fenster für die vierte Zeile:

NACHNAME	VORNAME	ANR	PUNKTE	
Weiss	Lisa	1	10	
Grau	Michael	1	9	
Daniel	Sommer	1	5	
				← Partitionsgrenze
Grau	Michael	2	9	Fenster ← a.Z.
Weiss	Lisa	2	8	

Fenster/Windows (7)

- Fortsetzung des Beispiels:

  `OVER (PARTITION BY B.ANR ORDER BY B.PUNKTE DESC)`

  Fenster für die fünfte Zeile:

NACHNAME	VORNAME	ANR	PUNKTE	
Weiss	Lisa	1	10	
Grau	Michael	1	9	
Daniel	Sommer	1	5	
				← Partitionsgrenze
Grau	Michael	2	9	Fenster
Weiss	Lisa	2	8	← akt. Zeile

Fenster/Windows (8)

- Z.B. bekommt man bei

  ```
  COUNT(*)
  OVER (PARTITION BY B.ANR ORDER BY B.PUNKTE DESC)
  ```

  folgendes Ergebnis:

NACHNAME	VORNAME	ANR	PUNKTE	COUNT
Weiss	Lisa	1	10	1
Grau	Michael	1	9	2
Daniel	Sommer	1	5	3
Grau	Michael	2	9	1
Weiss	Lisa	2	8	2

Fenster/Windows (9)

- Im Beispiel sind zufällig alle Punktzahlen in einer Partition verschieden.
- Sollten zwei Zeilen aber in den `PARTITION BY` und den `ORDER BY` Attributen übereinstimmen, würde das Fenster diese Zeilen immer als Block behandeln.

  Das Fenster kann dann also ggf. über die aktuelle Zeile hinaus reichen. Unten bei der Rahmenspezifikation ist gezeigt, wie man solche “`TIES`” ausschließen kann. Natürlich kann man auch einfach die Liste der `ORDER BY`-Attribute verlängern, z.B. die `SID` noch mit hinzufügen.

Fenster/Windows (10)

- Die Erweiterung des Fensters um jeweils eine Zeile ist z.B. nützlich, wenn man jeweils eine Zwischensumme mit ausgeben will.

```
SELECT S.NACHNAME, S.VORNAME, B.ANR, B.PUNKTE,
       SUM(PUNKTE) OVER (PARTITION BY B.SID
                         ORDER BY B.ANR)
       AS SUMME
FROM   STUDENTEN S, BEWERTUNGEN B
WHERE  S.SID = B.SID
AND    B.ATYP = 'H'
ORDER  BY S.NACHNAME, S.VORNAME, S.SID, B.ANR
```

Hier wird in jeder Zeile die Summe der Hausaufgabenpunkte für den jeweiligen Studenten bis zu der Aufgabe der Zeile angezeigt.

Fenster/Windows (11)

- Man kann sich die Auswertung so vorstellen:
  - ◇ Zunächst wertet das System

    `FROM ... WHERE ... GROUP ... HAVING ...`

    aus mit einer klassischen `SELECT`-Klausel, die alle im Folgenden benötigten Attribute enthält.

    Im Standard heißt dieser Ausdruck (mit noch einer optionalen `WINDOW`-Klausel am Ende) eine “table expression”.
  - ◇ Das Ergebnis wird sortiert nach den `PARTITION BY`-Attributen, gefolgt von den `ORDER BY`-Attributen.

    So stehen die Tupel einer Partition hintereinander, und in richtiger Reihenfolge.

Fenster/Windows (12)

- Auswertung mit `OVER`, Forts.:
  - ◇ Dann wird Ergebnis durchlaufen, und für jede Zeile ihr zugehöriges Fenster duchgegangen, wobei die jeweilige Aggregationsfunktion ausgewertet wird.

    Die Zeilen des Fensters könnte man mit einer geschachtelten Schleife durchlaufen, die je nach Rahmenspezifikation (s.u.) des Fensters auch über die aktuelle Zeile hinaus reichen kann.
  - ◇ Das Ergebnis wird den Zeilen jeweils hinzugefügt.
  - ◇ Gibt es mehrere `OVER`-Ausdrücke, so sortiert man ggf. neu und wiederholt den Vorgang.
  - ◇ Auch für `ORDER BY` am Ende wird neu sortiert.

Rahmen-Spezifikation (1)

- Sei eine Tabelle mit Aktienkursen gegeben:

AKTIENKURSE		
Aktie	Datum	Wert
Santa Claus AG	01.12.2011	100.00
Santa Claus AG	02.12.2011	105.00
Santa Claus AG	03.12.2011	104.00
Santa Claus AG	04.12.2011	106.00
⋮	⋮	⋮
Easter Eggs AG	01.12.2011	50.00
Easter Eggs AG	02.12.2011	40.00
Easter Eggs AG	03.12.2011	45.00
⋮	⋮	⋮

Rahmen-Spezifikation (2)

- Folgende Anfrage berechnet die Durchschnittswerte über jeweils drei Tagen:

```
SELECT AKTIE, DATUM,
       AVG(WERT) OVER (PARTITION BY AKTIE
                       ORDER BY DATUM
                       ROWS BETWEEN
                           1 PRECEDING AND
                           1 FOLLOWING)
FROM   AKTIENKURSE
ORDER  BY AKTIE, DATUM
```

Vereinfachend wird hier zunächst nicht berücksichtigt, dass es nicht an allen Tagen einen Aktienkurs gibt.

Rahmen-Spezifikation (3)

- Anfrage-Ergebnis, falls die Tabelle nur die obigen sieben Zeilen enthält:

AKTIENKURSE		
Aktie	Datum	AVG
Easter Eggs AG	01.12.2011	45.00
Easter Eggs AG	02.12.2011	45.00
Easter Eggs AG	03.12.2011	42.50
Santa Claus AG	01.12.2011	102.50
Santa Claus AG	02.12.2011	103.00
Santa Claus AG	03.12.2011	105.00
Santa Claus AG	04.12.2011	106.00

Bei der ersten Zeile gibt es keine vorangehende Zeile, es wird dann nur der Durchschnitt über dieser Zeile und der nächsten gebildet.

Rahmen-Spezifikation (4)

- Statt n `PRECEDING/FOLLOWING` ist auch möglich:
  - ◇ `UNBOUNDED PRECEDING/FOLLOWING`
  - ◇ `CURRENT ROW`

    Dies ist offenbar nur als Anfang möglich. Man kann allerdings statt `BETWEEN` auch nur den Anfang spezifizieren, dann ist das Ende implizit bei der aktuellen Zeile, z.B. "`ROWS 3 PRECEDING`".
- Statt `ROWS` kann man auch `RANGE` verwenden, um einen Bereich für den Attributwert des `ORDER BY` Attributes anzugeben.

  Es darf in diesem Fall nur ein `ORDER BY` Attribut geben, das von einem numerischen, Datums/Zeit- oder Intervall-Datentyp sein muss.

Rahmen-Spezifikation (5)

- Man kann aus dem Fenster Zeilen ausschließen:
  - `EXCLUDE CURRENT ROW`
  - `EXCLUDE GROUP`

    Damit werden die aktuelle Zeile und alle Zeilen mit den gleichen Werten in den `ORDER BY`-Attributen entfernt.
  - `EXCLUDE TIES`

    Dies entfernt Zeilen mit den gleichen Werten in den `ORDER BY` Attributen, aber nicht die aktuelle Zeile.
  - `EXCLUDE NO OTHERS`

    Dies entfernt gar keine Zeilen (es könnten vorher Zeilen aufgrund von anderen Regeln entfernt worden sein, deswegen diese Bezeichnung).