Datenbank-Programmierung

Kapitel 5: Data Warehouses und erweiterte Gruppierung

Prof. Dr. Stefan Brass

Martin-Luther-Universität Halle-Wittenberg

Sommersemester 2021

http://www.informatik.uni-halle.de/~brass/dbp21/

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`).

Inhalt

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.

Inhalt

Data Cube (1)

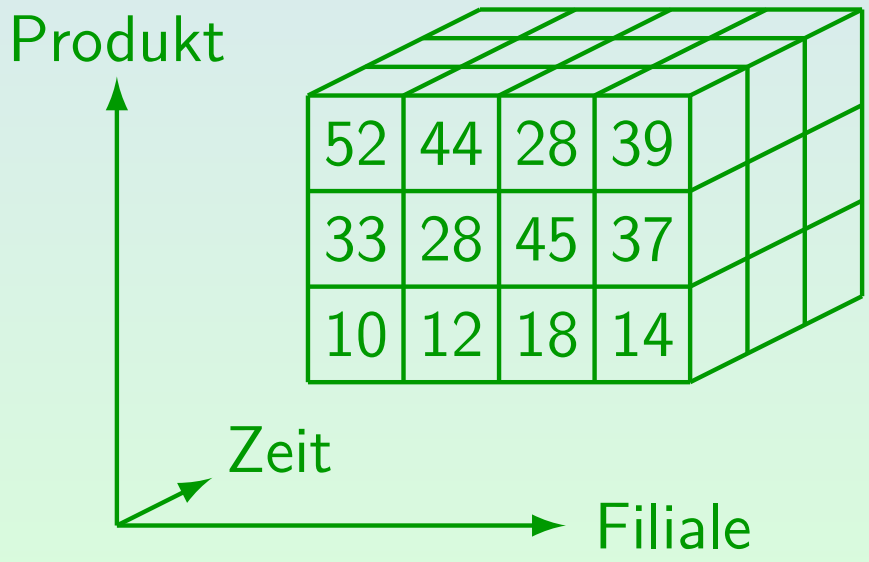
- Im OLAP-Bereich werden die Daten häufig als *n*-dimensionaler Würfel aufgefasst („Hypercube“).

  Man kann das als neues Datenmodell verstehen oder als Visualisierung.

- Ein zweidimensionaler Würfel ist eine Matrix:

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, MicroStrategy, SAP Business Objects, icCube, Pentaho.

Inhalt

Stern-Schema (1)

- Natürlich kann man eine Funktion wie

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

  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“).
- In modernen Datenbanksystemen werden solche Schemata bei der Optimierung teils speziell berücksichtigt.

  Z.B. kann es günstig sein, erst ein kartesisches Produkt von Bedingungen an die Dimensions-Tabellen zu erstellen, und dann einen Index für den Zugriff auf die Fakten-Tabelle zu benutzen. Sonst macht man eher schrittweise Joins.

Inhalt

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

Kurzer Einschub: NULLS FIRST/LAST (1)

- Für eine Summenzeile am Ende einer Tabelle steht in den nicht benötigten Spalten üblicherweise ein Nullwert.
- Beispiel: Hausaufgaben-Punkte von Lisa Weiss:

ANR	PUNKTE
1	10
2	8
(Null)	18

- Bei der Sortierung soll die Summenzeile natürlich zuletzt ausgegeben werden.
- Es ist nicht definiert, ob Nullwerte in der Sortier-Reihenfolge größer oder kleiner als normale Datenwerte sind.

  Ein Vergleich mit einem Nullwert führt zum dritten Wahrheitswert „unbekannt“.

Kurzer Einschub: NULLS FIRST/LAST (2)

- Für die Sortierung mit `ORDER BY` muss aber eine Reihenfolge festgelegt werden.

  PostgreSQL, Oracle, DB2 und MySQL/MariaDB behandeln Nullwerte als größer als jeder Wert, SQLite und MS SQL Server als kleiner.

- Seit SQL-2003 ist möglich: `ORDER BY ANR NULLS LAST`.

  Dies ist ggf. nach `ASC/DESC` anzugeben Der Default für `NULLS FIRST/LAST` ist systemabhängig („implementation-defined“). Das „Null Ordering“ ist Teil des Features T611, „Elementary OLAP operations“, passt also in dieses Kapitel.

- PostgreSQL, Oracle, DB2 verstehen `NULLS FIRST/LAST`.

- MySQL/MariaDB, SQLite und MS SQL Server nicht.

  Wenn man `NULLS LAST` nicht verwenden kann, kann man ein zusätzliches Sortierkriterium mit einem CASE-Ausdruck angeben, der auf Nullwerte testet oder mit `COALESCE` einen Wert einsetzen, der größer als jeder echte Wert ist.

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.

- Man kann diese Anfragen mit `UNION ALL` zusammensetzen, wenn man die in einem Teil jeweils nicht relevanten Gruppierungsattribute z.B. mit Nullwerten auffüllt.

  Sonst hätten die Teilanfragen ja unterschiedliche Schemata (Spaltenmengen): Wenn man nach einer Spalte gruppiert, möchte man den jeweiligen Wert dieser Spalte natürlich unter `SELECT` ausgeben. Wenn man nicht danach gruppiert, fließen alle Werte in die Summe bzw. den Durchschnitt ein, und es gibt keinen speziellen Wert, den man in dieser Spalte ausgeben kann (vielleicht „*“ oder eben einen Nullwert).

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

In manchen Systemen ist es nötig, dem Nullwert explizit den entsprechenden Datentyp zuzuweisen, z.B. `CAST(NULL AS NUMERIC(3,0)) AS ANR`.

Motivation (4)

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

  Wie 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.

- Es gibt sie in PostgreSQL seit Version 9.5 (2016)

  [https://www.postgresql.org/docs/9.5/sql-select.html]

  Oracle hatte `ROLLUP` und `CUBE` seit Version 8i (1998), `GROUPING SETS` seit Version 9i (2001). MS SQL Server versteht diese Konstrukte, SQLite nicht. MySQL hat nur ein Nicht-Standard „`WITH ROLLUP`“.

Motivation/Beispiel (5)

- Folgende Anfrage liefert das gleiche Ergebnis:

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

- 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`, z.B. auch `AVG`).

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. Mit der Funktion `GROUPING` kann man die beiden Nullwerte auseinanderhalten:
  - `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.
- Das ist auch für die Sortierung nützlich (statt `NULLS LAST`):

```
ORDER BY GROUPING(ANR), ANR
```

Wenn `ANR` Null ist, weil nicht nach `ANR` groupiert wurde, liefert `GROUPING(ANR)` den Wert `1`. Damit kommt die Summenzeile ans Ende. Für alle anderen Zeilen (mit einem Wert für `ANR`) ist `GROUPING(ANR)=0`. Sie kommen also nach vorne.

GROUPING (2)

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

```
SELECT CASE GROUPING(ATYP) WHEN 1 THEN '*'
            ELSE ATYP END AS ATYP,
       CASE GROUPING(ANR)  WHEN 1 THEN '*'
            ELSE CAST(ANR AS VARCHAR(3)) END
            AS ANR,
       ROUND(AVG(PCT))
FROM   ERGEBNISSE
GROUP  BY ROLLUP(ATYP, ANR)
ORDER  BY GROUPING(ATYP), ATYP,
          GROUPING(ANR),  ANR;
```

Im Beispiel wäre die Verwendung von `GROUPING` nicht nötig, da die Daten keine Nullwerte in `GROUP BY`-Attributen enthalten. Z.B. kürzer: `COALESCE(ATYP,'*')`.

CUBE (1)

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

```
SELECT   SID, ATYP, ROUND(AVG(PCT))
FROM     ERGEBNISSE
GROUP BY CUBE(SID, ATYP)
ORDER BY SID NULLS LAST, ATYP NULLS LAST
```

- 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))
ORDER BY SID NULLS LAST,
         ATYP NULLS LAST, ANR NULLS LAST
```

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))
ORDER BY SID NULLS LAST,
         ATYP NULLS LAST, ANR NULLS LAST
```

  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))
```

Literatur/Quellen

- 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 (pp. 841–872).
- ISO/IEC 9075-2: Information Technology - Database Languages - SQL - Part 2 (2003).
- Oracle Database SQL Language Reference, 11g Release 2 (11.2), `http://download.oracle.com/docs/cd/E11882_01/server.112/e26088/toc.htm`
- Oracle Database Data Warehousing Guide, 11g Release 2 (11.2). `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`
- Deßloch: Recent Developments for Data Models (Folien, TU Kaiserslautern, 2010). `http://wwwlgis.informatik.uni-kl.de/cms/courses/datenmodelle/`