7. SQL und relationale Algebra

7.1 SQL (Structured Query Language)

SQL ist 'die' Sprache, mit der die meisten relationalen Datenbanken erstellt, manipuliert und abgefragt werden. SQL ist eine sogenannte 4GL (Fourth-Generation Language). Sie ist *nichtprozedural*, d. h. der Fragesteller stellt eine Frage, gibt aber keinen Algorithmus zur Lösung vor. In einer 3GL wie Cobol, Pascal oder C müßte er angeben, wie die gesuchten Informationen gefunden werden können, z. B. vom Öffnen der Datei bis zum schrittweisen Durchgehen der Datensätze.

SQL ist ein ISO- und ANSI-Standard, der mehrfach spezifiziert wurde bzw. noch wird:

- SQL86
  1986 definiert
- SQL 89
  1989 definiert. Zwei mögliche Ebenen des Sprachumfangs:
  - Level 1
  - Level 2
- SQL 92 (SQL2)
  1992 definiert. Vier mögliche Ebenen des Sprachumfangs:
  - Entry Level
  - Transitional
  - Intermediate Level
  - Full Level
- SQL3
  Spezifikation ist gerade in Arbeit. Mehr dazu unter http://www.jcc.com/sql_stnd.html

Neben einem bestimmten SQL-Standard unterstützen Datenbanksysteme meist Teile höherer Standards sowie eigene SQL-Erweiterungen. SQL 89 Level 2 ist auch heute noch Basis des von vielen Datenbanksystemen unterstützen SQL, bei SQL 92 sind die meisten Systeme nur Entry Level-Compliant (z. B. Oracle8).

7.2 Relationale Algebra

Mit einer geeigneten Abfragesprache können gewünschte Daten aus einer relationalen Datenbank herausgesucht werden. Dafür eignet sich z. B. SQL. Dabei können folgende Operationen der Mengenlehre benutzt werden:

7.2.1 Selection (Selektion)

Wählt Zeilen aus einer Tabelle aus, die einer bestimmten Bedingung genügen.

Beispiel:

Mitarbeiter

Nachname	Vorname	Geburtsdatum
Milke	Lise	3.6.1934
Huber	Karl	16.12.1964
Trunstein	Helga	30.7.1956

Das SQL-Statement

```
SELECT * FROM Mitarbeiter WHERE Nachname = 'Trunstein'
```

liefert als Ergebnis alle Zeilen obiger Tabelle, die die angegebene Bedingung erfüllen. Dabei werden in der Ausgabe alle Spalten angezeigt (nach SELECT werden die Spalten aufgelistet, die angezeigt werden sollen. Das Zeichen * steht für 'alle Spalten')

Nachname	Vorname	Geburtsdatum
Trunstein	Helga	30.7.1956

7.2.2 Projection (Projektion)

Wählt bestimmte Spalten einer Tabelle aus.

Beispiel:
Das SQL Statement

```
SELECT Nachname, Geburtsdatum FROM Mitarbeiter
```

liefert z. B. als Ergebnis:

Nachname	Geburtsdatum
Milke	3.6.1934
Huber	16.12.1964
Trunstein	30.7.1956

Natürlich können Selektion und Projektion auch zusammen eingesetzt werden.

7.2.3 Union (Vereinigung)

Fügt die Zeilen zweier Tabellen mit gleicher Spaltenzahl in einer Tabelle zusammen. Die Namen der jeweiligen Spalten der zwei Tabellen müssen nicht identisch sein, lediglich der Datentyp bzw. Wertebereich des Inhalts

Beispiel:

Mitarbeiter

Nachname	Vorname	Geburtsdatum
Milke	Lise	3.6.1934
Huber	Karl	16.12.1964

Trunstein	Helga	30.7.1956

Kunden

Nachname	Vorname	Geburtsdatum
Kelz	Andreas	21.7.1965
Huber	Karl	16.12.1964
Ernsbach	Elli	29.6.1956

Das SQL-Statement

```
SELECT * FROM Mitarbeiter UNION SELECT * FROM Kunden
```

liefert als Ergebnis:

Nachname	Vorname	Geburtsdatum
Milke	Lise	3.6.1934
Huber	Karl	16.12.1964
Trunstein	Helga	30.7.1956
Kelz	Andreas	21.7.65
Ernsbach	Elli	29.6.1956

Doppelte Zeilen werden automatisch unterdrückt. Mit `UNION ALL` werden sie angezeigt. UNION gehört zum SQL 92 Entry Level.

7.2.4 Intersection (Schnittmenge)

Liefert die Zeilen, die in beiden angegebenen Tabellen enthalten sind.

Beispiel mit den zwei Tabellen "Mitarbeiter" und "Kunden":

Das SQL-Statement

```
SELECT * FROM Mitarbeiter INTERSECT SELECT * FROM Kunden
```

liefert als Ergebnis

Nachname	Vorname	Geburtsdatum
Huber	Karl	16.12.1964

INTERSECT steht nicht immer zur Verfügung, da es zum SQL 92 Intermediate Level gehört. Es kann aber bei Bedarf nachgebildet werden, z. B. durch ein geschachteltes SELECT-Statement:

```
SELECT * FROM Mitarbeiter WHERE Nachnahme IN (SELECT Nachname FROM Kunden)
```

7.2.5 Minus (Differenz)

Liefert die Zeilen der ersten Tabelle, die in der zweiten Tabelle nicht enthalten sind.

Beispiel mit den zwei Tabellen **Mitarbeiter** und **Kunden**:

Das SQL-Statement

```
SELECT * FROM Mitarbeiter MINUS SELECT * FROM Kunden
```

liefert als Ergebnis

Nachname	Vorname	Geburtsdatum
Milke	Lise	3.6.1934
Trunstein	Helga	30.7.1956
Kelz	Andreas	21.7.65
Ernsbach	Elli	29.6.1956

MINUS steht nicht immer zur Verfügung, da es zum SQL 92 Intermediate Level gehört. Es kann aber bei Bedarf nachgebildet werden, z. B. durch ein geschachteltes SELECT-Statement:

```
SELECT * FROM Mitarbeiter WHERE Nachnahme NOT IN (SELECT Nachname FROM Kunden)
```

7.2.6 Join (Verbund)

Verbindet die Spalten zweier Tabellen zu einer Tabelle. Es gibt mehrere Varianten des Joins:

Cross Join

Inner Join, Equivalent Join

Natural Join

Left Outer Join, Left Join

Right Outer Join, Right Join

Full Outer Join, Full Join

Union Join

Semi-Join

Theta Join, Non-Equivalent-Join

Self-Join

- **Cross Join, Kartesisches Produkt**
  Verbindet jede Zeile der ersten Tabelle mit jeder Zeile der zweiten Tabelle

  Beispiel:

Mitarbeiter

Nachname	Vorname	Geburtsdatum
Milke	Lise	3.6.1934
Huber	Karl	16.12.1964
Trunstein	Helga	30.7.1956

Projekte

Projekt	Nachname	Vorname
Neubau	Huber	Anna
Werbung	Trunstein	Helga
Design	Kohlmeier	Johann

Das SQL-Statement

```
SELECT * FROM Mitarbeiter CROSS JOIN Projekte
```
(SQL92)

```
SELECT * FROM Mitarbeiter, Projekte
```
(vor SQL92 häufig benutzt)

liefert als Ergebnis

Mitarbeiter. Nachname	Mitarbeiter. Vorname	Geburtsdatum	Projekt	Projekte. Nachname	Projekte. Vorname
Milke	Lise	3.6.1934	Neubau	Huber	Anna
Milke	Lise	3.6.1934	Werbung	Trunstein	Helga
Milke	Lise	3.6.1934	Design	Kohlmeier	Johann
Huber	Karl	16.12.1964	Neubau	Huber	Anna
Huber	Karl	16.12.1964	Werbung	Trunstein	Helga
Huber	Karl	16.12.1964	Design	Kohlmeier	Johann
Trunstein	Helga	30.7.1956	Neubau	Huber	Anna
Trunstein	Helga	30.7.1956	Werbung	Trunstein	Helga
Trunstein	Helga	30.7.1956	Design	Kohlmeier	Johann

Gleichnamige Spalten der zwei Tabellen werden durch Voranstellen des Tabellennamens referenziert, also z. B. "Mitarbeiter.Nachname".

Beachten Sie:
Bei einem Cross Join großer Tabellen wird die Ergebnistabelle sehr groß. Das Ergebnis eines Cross Joins ist häufig nutzlos!

- **Inner Join = Equivalent Join**
  Verbindet Datensätze aus zwei Tabellen, sobald ein gemeinsames Feld dieselben Werte enthält.

  Beispiel mit den zwei Tabellen "Mitarbeiter" und "Projekte":

Das SQL-Statement

```
SELECT * FROM Mitarbeiter INNER JOIN Projekte ON Mitarbeiter.Nachname =
Projekte.Nachname    (SQL92)

SELECT * FROM Mitarbeiter, Projekte WHERE Mitarbeiter.Nachname =
Projekte.Nachname    (vor SQL92 häufig benutzt)
```

vergleicht auf übereinstimmende Nachnamen (`ON ...`) und liefert als Ergebnis

Mitarbeiter. Nachname	Mitarbeiter. Vorname	Geburtsdatum	Projekt	Projekte. Nachname	Projekte. Vorname
Huber	Karl	16.12.1964	Neubau	Huber	Anna
Trunstein	Helga	30.7.1956	Werbung	Trunstein	Helga

Beachten Sie:
Da nur der Nachname verglichen wurde, tauchen in der ersten Zeile zwei verschiedene Personen auf (Huber Karl und Huber Anna). Natürlich kann auch auf Vor- und Nachnamen verglichen werden:

```
SELECT * FROM Mitarbeiter INNER JOIN Projekte ON (Mitarbeiter.Nachname =
Projekte.Nachname AND Mitarbeiter.Vorname = Projekte.Vorname)
```

In der Praxis wird man bei einem Inner Join nicht alle, sondern nur ausgewählte Spalten anzeigen lassen. Das SQL-Statement

```
SELECT Mitarbeiter.*, Projekte.Projekt FROM Mitarbeiter INNER JOIN Projekte
ON (Mitarbeiter.Nachname = Projekte.Nachname AND Mitarbeiter.Vorname =
Projekte.Vorname)
```

liefert z. B.:

Nachname	Vorname	Geburtsdatum	Projekt
Trunstein	Helga	30.7.1956	Werbung

Dieser Join wird als Natural Join bezeichnet (s. u.).

Wird bei einem SQL-Statement nur JOIN statt INNER JOIN angegeben, wird meist ebenfalls ein Inner Join ausgeführt.

- **Natural Join**
  Verknüpft die beiden Tabellen über die Gleichheit aller gleichlautenden Spalten. Gleichlautende Spalten werden im Ergebnis nur einmal angezeigt. Haben die Tabellen keine gleichlautenden Spalten, wird der Natural Join zum Cross Join. Gibt es nur eine gleichlautende Spalte, so ist der Natural Join ein Inner Join mit anschließender Projektion, bei der gleichnamige Spalten ausgeblendet werden.

  Für den Natural Join gibt es keinen speziellen SQL92-Befehl. Er wird bei Bedarf aus einem Inner Join mit anschließender Projektion erzeugt.

- **Left Outer Join = Left Join**

Mit einem Left Join wird eine sogenannte linke Inklusionsverknüpfung erstellen. Linke Inklusionsverknüpfungen schließen alle Datensätze aus der ersten (linken) Tabelle ein, auch wenn keine entsprechenden Werte für Datensätze in der zweiten Tabelle existiert.

Beispiel mit den zwei Tabellen "Mitarbeiter" und "Projekte":

Das SQL-Statement

```
SELECT * FROM Mitarbeiter LEFT JOIN Projekte ON (Mitarbeiter.Nachname =
Projekte.Nachname AND  Mitarbeiter.Vorname = Projekte.Vorname)
```

liefert:

Mitarbeiter. Nachname	Mitarbeiter. Vorname	Geburtsdatum	Projekt	Projekte. Nachname	Projekte. Vorname
Milke	Lise	3.6.1934	NULL	NULL	NULL
Huber	Karl	16.12.1964	NULL	NULL	NULL
Trunstein	Helga	30.7.1956	Werbung	Trunstein	Helga

Hier werden also auch die Mitarbeiter angezeigt, die an keinem Projekt arbeiten. NULL bedeutet, daß das Feld keinen Eintrag enthält.

Häufig sieht man auch die ältere Schreibweise für einen Left Outer Join (z. B. bei Oracle):

```
SELECT * FROM Mitarbeiter, Projekte WHERE (Mitarbeiter.Nachname =
Projekte.Nachname (+) AND Mitarbeiter.Vorname = Projekte.Vorname (+))
```

Das (+) kennzeichnet dabei die Spalten, bei denen Platz für NULL-Werte freigehalten werden muß. Bei einem Left Outer Join also bei den Spalten der rechten(!) Seite.

- **Right Outer Join = Right Join**
  Mit einem Right Join wird eine sogenannte rechte Inklusionsverknüpfung erstellen. Rechte Inklusionsverknüpfungen schließen alle Datensätze aus der zweiten (rechten) Tabelle ein, auch wenn keine entsprechenden Werte für Datensätze in der ersten Tabelle existiert.

  Beispiel mit den zwei Tabellen "Mitarbeiter" und "Projekte":

  Das SQL-Statement

```
SELECT * FROM Mitarbeiter RIGHT JOIN Projekte ON (Mitarbeiter.Nachname =
Projekte.Nachname AND  Mitarbeiter.Vorname = Projekte.Vorname)
```

  liefert:

Mitarbeiter. Nachname	Mitarbeiter. Vorname	Geburtsdatum	Projekt	Projekte. Nachname	Projekte. Vorname
NULL	NULL	NULL	Neubau	Huber	Anna
Trunstein	Helga	30.7.1956	Werbung	Trunstein	Helga
NULL	NULL	NULL	Design	Kohlmeier	Johann

Hier werden also auch die Projekte angezeigt, an denen kein Mitarbeiter der Firma arbeitet (sondern vielleicht externe Arbeitskräfte).

Auch hier sieht man häufig die ältere Schreibweise für einen Right Outer Join:

```
SELECT * FROM Mitarbeiter, Projekte WHERE (Mitarbeiter.Nachname(+) =
Projekte.Nachname AND Mitarbeiter.Vorname(+) = Projekte.Vorname)
```

Das (+) kennzeichnet dabei die Spalten, bei denen Platz für NULL-Werte freigehalten werden muß. Bei einem Right Outer Join also bei den Spalten der linken(!) Seite.

- **Full Outer Join = Full Join**
  Eine Kombination von Left Outer Join und Right Outer Join.

  Beispiel mit den zwei Tabellen "Mitarbeiter" und "Projekte":

  Das SQL-Statement

  ```
  SELECT * FROM Mitarbeiter FULL JOIN Projekte ON (Mitarbeiter.Nachname =
  Projekte.Nachname AND  Mitarbeiter.Vorname = Projekte.Vorname)
  ```

  liefert:

Mitarbeiter. Nachname	Mitarbeiter. Vorname	Geburtsdatum	Projekt	Projekte. Nachname	Projekte. Vorname
Milke	Lise	3.6.1934	NULL	NULL	NULL
Huber	Karl	16.12.1964	NULL	NULL	NULL
Trunstein	Helga	30.7.1956	Werbung	Trunstein	Helga
NULL	NULL	NULL	Neubau	Huber	Anna
NULL	NULL	NULL	Design	Kohlmeier	Johann

  In der älteren Schreibweise kann ein Full Outer Join nicht mit (+)-Zeichen auf beiden Seiten erstellt werden, sondern wird über die Vereinigung (Union) eines Left Outer Join und eines Right Outer Join zusammengesetzt.

- **Union Join**
  Ähnlich dem Full Outer Join werden Datensätze beider Tabellen aufgenommen. Sie werden aber nicht über eine Bedingung verknüpft.

  Beispiel mit den zwei Tabellen "Mitarbeiter" und "Projekte":

  Das SQL-Statement

  ```
  SELECT * FROM Mitarbeiter UNION JOIN Projekte
  ```

  liefert:

Mitarbeiter. Nachname	Mitarbeiter. Vorname	Geburtsdatum	Projekt	Projekte. Nachname	Projekte. Vorname
Milke	Lise	3.6.1934	NULL	NULL	NULL

Huber	Karl	16.12.1964	NULL	NULL	NULL
Trunstein	Helga	30.7.1956	NULL	NULL	NULL
NULL	NULL	NULL	Neubau	Huber	Anna
NULL	NULL	NULL	Werbung	Trunstein	Helga
NULL	NULL	NULL	Design	Kohlmeier	Johann

Der Union Join steht nicht immer zur Verfügung, da er zum SQL 92 Intermediate Level gehört

- **Semi-Join**
  Der Semijoin der Tabellen "Mitarbeiter" und "Projekte" ist ein Natural Join der zwei Tabellen mit anschließender Projektion auf die Attribute der ersten Tabelle:

  Das SQL-Statement

```
SELECT Mitarbeiter.* FROM Mitarbeiter INNER JOIN Projekte
ON (Mitarbeiter.Nachname = Projekte.Nachname AND Mitarbeiter.Vorname =
Projekte.Vorname)
```

  liefert:

Nachname	Vorname	Geburtsdatum
Trunstein	Helga	30.7.1956

- **Theta Join, Non-Equivalent-Join**
  Der Theta Join ist eine Verallgemeinerung des Inner Join. Während beim Inner Join die Gleichheit des Inhalts zweier Attribute verglichen wird, wird beim Theta Join der Inhalt der Attribute i und j mit einer beliebigen Formel Theta(i,j) verglichen, etwa i = j (i gleich j; InnerJoin), i < j (i kleiner j), i <= j (i kleiner oder gleich j), i > j (i größer j) usw.

  Beispiel:

```
SELECT * FROM Mitarbeiter INNER JOIN Projekte ON Mitarbeiter.Nachname <=
Projekte.Nachname
```

  Im lexikalischen Sinne ist z. B. Huber kleiner als Trunstein, da H im Alphabet vor T steht. Das Ergebnis ist folgende Tabelle:

Mitarbeiter. Nachname	Mitarbeiter. Vorname	Geburtsdatum	Projekt	Projekte. Nachname	Projekte. Vorname
Milke	Lise	3.6.1934	Werbung	Trunstein	Helga
Huber	Karl	16.12.1964	Neubau	Huber	Anna
Huber	Karl	16.12.1964	Werbung	Trunstein	Helga
Huber	Karl	16.12.1964	Design	Kohlmeier	Johann
Trunstein	Helga	30.7.1956	Werbung	Trunstein	Helga

- **Self-Join**

Der Self-Join ist ein beliebiger Join, bei dem nicht zwei verschiedene Tabellen benutzt werden, sondern zweimal dieselbe Tabelle.

Beispiel:

Mitarbeiter

Name	Personalnummer	PersonalnummerVorgesetzter
Milke	1	3
Huber	2	1
Trunstein	3	NULL

Es soll zu jedem Mitarbeiter der Name des Vorgesetzten ermittelt werden.

Das SQL-Statement

```
SELECT a.Name, a.Personalnummer, b.Name "Chef" FROM Mitarbeiter a, Mitarbeiter b
WHERE a.PersonalnummerVorgesetzter = b.Personalnummer(+)
```

liefert als Ergebnis

Name	Personalnummer	Chef
Milke	1	Trunstein
Huber	2	Milke
Trunstein	3	NULL

Um zweimal dieselbe Tabelle benutzen zu können, bekommt sie zwei verschiedene Alias-Namen a und b. Die Überschrift der Spalte 'Name' wird für die Ausgabe bei der zweiten Tabelle in 'Chef' geändert. Ohne den Left Outer Join (+) würden diejenigen Mitarbeiter, die keinen Vorgesetzten haben, nicht angezeigt.

7.2.7 Division (Quotient)

Das Konzept der Division ist eng verknüpft mit dem Kartesischen Produk T = R x S zweier Relationen R und S, so daß T/S (T geteilt durch S) die Relation R ergibt. Hat T die Anzahl t Spalten und S die Anzahl s Spalten, so hat T/S die Anzahl t - s Spalten.

Beispiel:

Projektarbeit

Nachname	Vorname	Geburtsdatum	Projekt
Milke	Lise	3.6.1934	Neubau
Milke	Lise	3.6.1934	Werbung
Milke	Lise	3.6.1934	Design
Huber	Karl	16.12.1964	Neubau
Huber	Karl	16.12.1964	Werbung
Huber	Karl	16.12.1964	Design

Trunstein	Helga	30.7.1956	Neubau
Trunstein	Helga	30.7.1956	Werbung
Trunstein	Helga	30.7.1956	Design

Projekte

Projekt
Neubau
Werbung
Design

Dann liefert Projektarbeit/Projekte:

Nachname	Vorname	Geburtsdatum
Milke	Lise	3.6.1934
Huber	Karl	16.12.1964
Trunstein	Helga	30.7.1956

Wichtig:
Im Allgemeinen bedeutet die Division R = T/S nicht, daß T = R x S, da in T zusätzliche Tupel auftreten können. In obiger Tabelle könnte z. B. an beliebiger Stelle die Zeile

Nachname	Vorname	Geburtsdatum	Projekt
Kelz	Andreas	21.7.1965	EDV

stehen, ohne das Ergebnis der Division zu ändern.

↗ Inhaltsverzeichnis

© 1996-98 Andreas Kelz