Prof. Dr. Gunter Schlageter et. al.

Kurs 01671

Datenbanken I

LESEPROBE

Fakultät für
Mathematik und Informatik

FernUniversität in Hagen

Leseprobe zum Kurs 1671, entnommen aus KE2

3.3 SQL – eine relationale Abfragesprache

In diesem Abschnitt werden wir die Abfragesprache mit der weitesten Verbreitung vorstellen: SQL.

3.3.1 Grundlegende Sprachelemente von SQL

Nachdem E.F. Codd 1970 den bahnbrechenden Artikel „A Relational Model of Data for Large Shared Data Banks“ /COD70/ veröffentlicht hatte, begann man im IBM San Jose Research Laboratory über eine Sprache zu forschen, die dazu geeignet wäre dieses Modell zu implementieren. Die Sprache wurde SEQUEL genannt, ein Akronym für **S**tructured **E**nglish **QUE**ry **L**anguage.

Innerhalb eines Forschungsprojektes mit Namen System R entwickelte man ein Datenbanksystem desselben Namens zu Forschungszwecken. SEQUEL war das Programmierinterface zu diesem Datenbanksystem. Zwischen 1974 und 1975 wurde diese Sprache in einem Prototypen implementiert, der wiederum den Namen SEQUEL erhielt. In den nächsten Jahren wurde eine überarbeitete Version SEQUEL/2 entwickelt Diese Version der Sprache wurde in SQL umbenannt. [1]

Das Projekt System R lief in der Zeit von 1971 bis 1979 und mündete in einem verteilten Datenbankprojekt mit dem Namen System R* (englisch ausgesprochen „are star“).

In den späten 70er Jahren wurde allgemein bekannt, dass IBM dabei war, ein kommerzielles DBMS zu entwickeln, das SQL als Sprache benutzen sollte. Als Konsequenz daraus wurde die Konkurrenz aktiv. Tatsächlich schlug eine kleine Firma mit dem Namen „Relational Software Inc.“ die große IBM in dem Rennen um die erste Markteinführung eines solchen Systems. Diese Firma heißt übrigens inzwischen ORACLE und ist seitdem geringfügig gewachsen. IBM brachte 1981 sein erstes relationales DBMS auf den Markt, damals SQL/DS genannt. Die erste Version des bekannten DB2 erschien 1983.

Einen rechtlichen Status als Standard erhielt SQL 1986 durch die Veröffentlichung eines ANSI Standards der Sprache. Bekannt wurde dieser Standard unter dem Kürzel SQL-86. Eine Überarbeitung erschien 1989 (SQL-89). Größere Änderungen erfolgten 1992 (SQL-92). Viele Schwachstellen von SQL-86 und SQL-89 wurden in diesem Standard beseitigt. SQL:1999, die neueste Revision geht weit über die bisherigen Standards hinaus und ist eine Annäherung an die Möglichkeiten, die neue Datenbanksysteme bieten (können) und die Bedürfnissen ihrer Benutzer.

In der Zeit von 1992 bis 1999 wurden einzelne Aspekte des zukünftigen Standards SQL:1999 gesondert veröffentlicht. Die erste dieser Veröffentlichungen war der Standard zu SQL/CLI (Call Level Interface), bekannt als CLI-95. Wahrscheinlich die bekannteste Implementierung dieses Standards ist ODBC, Open Database Connectivity.

---

1 Aus dieser Umbenennung rührt der Glaube her, SQL stünde für 'Structured Query Language' und würde "sequel" ausgesprochen. Laut Standard ist SQL jedoch kein Akronym.

1996 wurde SQL/PSM (Persistent Stored Modules) veröffentlicht. PSM-96, wie der Standard genannt wurde, legt die Syntax der prozeduralen Logik fest, die auf Serverseite in DBMS implementiert wird. Dieser Standard war nötig geworden, weil viele kommerzielle DBMS „Stored Procedures“ anboten, deren Syntax allerdings voneinander abwich.

1998 erfolgte die Veröffentlichung von SQL/OLB (Object Language Bindings), der beschreibt, wie SQL Statements in JAVA eingebettet werden können. Er basiert auf dem JDBC Modell und wurde bekannt unter dem Namen OLB-98.

1999 wurde der neue Sprachstandard für SQL angenommen und als SQL:1999 veröffentlicht.

Der Bindestrich ist verschwunden und die Jahreszahl wurde um das Jahrhundert und Jahrtausend erweitert. Die Verwendung der vierstelligen Jahreszahl entspringt der Y2K-Umstellung, die Änderung des Bindestriches in einen Doppelpunkt ist von weitreichenderer Bedeutung: Sie bezieht sich auf die Normen der ISO, die die Namen der Standards von den Jahreszahlen der Veröffentlichung durch einen Doppelpunkt trennt, im Gegensatz zur ANSI, die hierzu eben den Bindestrich benutzt. Dies deutet auf die zunehmende Internationalisierung des Standardisierungsprozesses hin und die Marginalisierung der ANSI Norm außerhalb der USA durch das Aufgehen ihrer Normung in der ISO.[2]

SQL basiert auf dem relationalen Modell, ist aber keineswegs eine perfekte Abbildung desselben auf eine Sprache. SQL weicht in einigen Aspekten vom relationalen Modell und relational vollständigen Sprachen, wie in den vorangegangenen Abschnitten erklärt, ab. Die meisten Unterschiede stellen Erweiterungen dar, auf die wir später noch eingehen. Der wichtigste Unterschied betrifft allerdings das grundlegende Objekt relationaler Datenbanken und des relationalen Modells. SQL arbeitet mit Tabellen, die Duplikate von Zeilen enthalten können. Dies entspräche Tupeln, die in allen Attributen übereinstimmen, was in Relationen nicht möglich ist.

Um den Unterschieden zwischen der Sprachregelung in SQL und dem relationalen Modell Rechnung zu tragen, werden wir in diesem Kapitel folgendermaßen unterscheiden:

Relationales Modell	**SQL**	**(SQL englisch)**
Relation	Tabelle	(Table)
Tupel	Zeile	(Row)
Attribut	Spalte	(Column)

An den Stellen, an denen wir uns explizit auf das relationale Modell als Grundlage für SQL beziehen, werden wir die Sprachregelung des Modells anwenden und dort, wo es um die Umsetzung in SQL geht, diejenige, die im SQL:1999 Standard benutzt wird.

Neben den Aggregierungsfunktionen, die wir später in diesem Kapitel behandeln werden, bietet SQL noch eine wichtige Ergänzung gegenüber dem vorgestellten Relationenmodell und „bloß“ relational vollständigen Sprachen an: sogenannte NULL-Werte. Prinzipiell kann jedes

---

2 Eine genaue Darstellung der Geschichte der SQL-Normung (die für Entwickler durchaus interessant sein mag, wenn Sie gezwungen sind mit verschiedenen Implementierungen parallel zu arbeiten) finden Sie in /MELT02/ im Anhang F. Die Darstellung in diesem Kapitel lehnt sich an die in /MELT02/ an.

Attribut für ein Tupel den Wert NULL annehmen, was (ohne weitere Vereinbarungen) bedeutet, dass dieser Wert zwar vorgesehen, aber (noch) nicht festgelegt ist. Die Definition verschiedener Operatoren wurde für NULL-Werte verändert. Es ergibt sich eine ternäre Logik mit den Werte WAHR (TRUE), FALSCH (FALSE) und UNBEKANNT (UNKNOWN) bzw. NULL.

Wir werden mit den folgenden Beispielen einen Einblick in die umfassenden Möglichkeiten der SELECT-Anweisung geben und daran anschließend einige weitere DML- und DDL-Anweisungen in SQL vorstellen.[3]

Die SELECT-Anweisung

Eine SQL-Abfrage (*Query*) hat folgende Grundstruktur:

SELECT A_1, ..., A_n
FROM R_1, ..., R_n
WHERE *Prädikat(R_1, ..., R_n).*

Die Bedeutung einer solchen Query kann man sich so veranschaulichen: selektiere aus dem Kreuzprodukt der Relationen R_1, ..., R_n alle Tupel, die die Bedingung *Prädikat(R_1, ..., R_n)* erfüllen, und projiziere die so entstehende Relation auf die Attribute A_1, ..., A_n.

Dabei ist A_i ein Attribut einer der Relationen R_1, ..., R_n. *Prädikat(R_1, ..., R_n)* ist eine Bedingung, die Attributwerte mit Konstanten oder zwei Attributwerte eines Tupels miteinander vergleicht oder andere Operatoren auf die Attributwerte anwendet; solche Teilbedingungen können durch die Booleschen Operatoren $\wedge$ (AND) und $\vee$ (OR) miteinander verknüpft und mit $\neg$ (NOT) negiert werden. Als Operanden können in den Prädikaten auch Mengen auftreten, insbesondere Ergebnismengen, die über andere SQL-Abfragen konstruiert werden.

Beispiel 3.1: (Gerüst der SELECT-Anweisung)

FINDE DIE BERUFE DER ANGESTELLTEN IN ABTEILUNG 3.

```
SELECT BERUF
FROM ANGEST
WHERE ABTNR = 3
```

Hier werden in der Tabelle ANGEST die Tupel mit ABTNR = 3 ermittelt und von diesen dann die Werte des Attributs BERUF ausgegeben.

Die Ergebnismenge des obigen Beispiels kann doppelte Elemente enthalten.
Zum Aussortieren doppelter Zeilen muss in der SELECT-Klausel das Schlüsselwort DISTINCT angegeben werden. Der Grund hierfür liegt darin, dass es zusätzlichen Rechenaufwand kostet, aus jeder Ergebnistabelle Duplikate zu eliminieren, während viele praktische Anwendungen (insbesondere aber auch die Berechnung von Zwischenergebnissen) diese Anforderung gar nicht stellen.

Duplikate in SQL

---

[3] Traditionell wird SQL häufig in DML (Data Manipulation Language) und DDL (Data Definition Language) aufgeteilt. Diese Aufteilung findet sich so im Standard nicht wieder. Laut Standard müsste man eher von SDL (Schema Definition Language), SML (Schema Manipulation Language) und DML sprechen.

Beispiel 3.2: (Aussortieren doppelter Tupel)

Die Abfrage:

FINDE ALLE WOHNORTE DER ANGESTELLTEN.

... lautet in SQL:

```
SELECT DISTINCT WOHNORT
FROM ANGEST
```

Die WHERE-Klausel kann hier entfallen, da lediglich die Projektion auf die Spalte WOHNORT durchgeführt wird und dazu keine Bedingung an die einzelnen Zeilen der Tabelle ANGEST gestellt werden muss.

Es soll jedoch in diesem Beispiel um den Einsatz des Schlüsselwortes DISTINCT gehen: Das Ergebnis dieser Abfrage enthält erwartungsgemäß drei Zeilen. Ohne die Angabe von DISTINCT jedoch wäre die Zeile (Karlsruhe) zweimal enthalten gewesen, also insgesamt vier Zeilen im Ergebnis (vgl. Beispieldaten aus Beispiel 3.4):

mit DISTINCT	Hagen
	Karlsruhe
	Marburg

ohne DISTINCT	Karlsruhe
	Hagen
	Marburg
	Karlsruhe

Das zweifache Auftauchen von „Karlsruhe“ widerspricht der Frage nach allen Wohnorten, die Query ohne DISTINCT entspricht also auch semantisch nicht der Anforderung.

Um Beziehungen zwischen mehreren Relationen zu betrachten, können in der FROM-Klausel der Abfrage mehrere Tabellen angegeben werden. Über diese Möglichkeit werden in SQL JOINS realisiert. In der WHERE-Klausel wird deren Verknüpfung untereinander bestimmt.

Beispiel 3.3: (Einfache SELECT-Anweisung I)

FINDE FÜR JEDES PROJEKT DIE PROJEKTNUMMER UND DEN NAMEN DES PROJEKTLEITERS

```
SELECT PNR, NAME
FROM PROJEKT, ANGEST
WHERE PROJEKT.P_LEITER = ANGEST.ANGNR
```

In diesem Beispiel werden Werte aus verschiedenen Tabellen verlangt. Die angegebene Anweisung wird folgendermaßen ausgeführt:

- Erst Bildung eines Kreuzproduktes der Tabellen PROJEKT und ANGEST,
- Auswertung der WHERE Bedingung und Selektion der Zeilen, die die Bedingung erfüllen,
- dann Projektion auf die in der SELECT-Klausel angegebenen Spalten.

Man beachte bei diesem Beispiel, dass man die Spalten nicht mit den Namen der Tabellen qualifizieren muss, weil die Namen der Spalten auch bezogen auf zwei Tabellen eindeutig sind. Es ist jedoch sinnvoll, die Namen der Tabellen immer mitzuführen.

Beispiel 3.4: (Einfache SELECT-Anweisung II)

Erinnern wir uns an die letzte Abfrage aus **Beispiel 3.4**:
(siehe auch **Beispiel 3.9**)

> FINDE DIE NAMEN ALLER ANGESTELLTEN, DIE AM PROJEKT MIT DER NUMMER 17 MITARBEITEN

Als SQL-Abfrage lautet sie:

```
SELECT ANGEST.NAME
FROM ANG_PRO, ANGEST
WHERE  ANG_PRO.PNR = 17 AND
       ANG_PRO.ANGNR = ANGEST.ANGNR
```

In der Tabelle, die sich nach Bildung des JOINS aus ANGEST und ANG_PRO ergibt, kommt die Spalte ANGNR zweimal vor, weil sie sowohl in ANGEST als auch in ANG_PRO enthalten ist. In dem WHERE-Teil der Abfrage muss daher jeweils der Tabellenname vorangestellt werden, um das Attribut eindeutig zu qualifizieren.

In komplizierteren Abfragen wird es in SQL unter Umständen notwendig, *Tupelvariablen* (im Standard: 'Correlation Names') einzuführen. Tupelvariablen sind vor allem dann nötig, wenn in einer Abfrage mehrere verschiedene Tupel derselben Relation gleichzeitig betrachtet werden sollen. Oder anders ausgedrückt: Tupelvariablen sind immer dann erforderlich, wenn der JOIN einer Tabelle mit sich selbst gebildet werden soll. Einen solchen Fall demonstriert das folgende Beispiel:

Beispiel 3.5: (Tupelvariablen in SQL)

Die Abfrage (siehe auch **Übung 3.4**) ...

> GIB DIE NAMEN ALLER PERSONEN AUS, DIE MEHRERE PROJEKTE LEITEN.

... kann in SQL folgendermaßen dargestellt werden:

```
SELECT DISTINCT ANGEST.NAME
FROM ANGEST, PROJEKT AS P1, PROJEKT AS P2
WHERE  ANGEST.ANGNR = P1.P_LEITER AND
       ANGEST.ANGNR = P2.P_LEITER AND
       P1.PNR != P2.PNR[4]
```

In dieser Abfrage werden durch FROM ... PROJEKT AS P1, PROJEKT AS P2 zwei Tupelvariablen für die Tabelle PROJEKT definiert[5]. Gemäß der WHERE-Klausel wird ein Angestellter mit zwei Zeilen (P1 und P2) verbunden, die verschiedene Projekte repräsentieren (P1.PNR != P2.PNR), wobei der Angestellte aber in beiden Projektleiter ist (ANGEST.ANGNR= P1.P_LEITER und ANGEST.ANGNR= P2.P1_LEITER).

---

4 '!=' steht in SQL für den mathematischen Vergleichoperator '≠'(„ungleich“) und ist synonym zu '<>'.

5 Das Schlüsselwort AS ist optional.

Übung 3.1:

Formulieren Sie die Abfrage aus Beispiel 3.23 mit nur einer Tupelvariable.

Übung 3.2:

*Formulieren Sie die Abfragen a) bis c) aus **Übung 3.4** als SQL-Anweisungen.*

Übung 3.3:

Geben Sie an, ob die SQL-SELECT-Anweisung eher Ausdrücken der Relationenalgebra oder des Relationenkalküls entspricht und begründen Sie dies.

Der SQL-Standard sieht vor, dass JOIN-Operatoren in der FROM-Klausel angegeben werden. U.a. kann dadurch die Lesbarkeit einer SELECT-Anweisung verbessert werden, weil sich im Idealfall die Angaben in der WHERE-Klausel auf die gewünschten Eigenschaften der einzelnen Entitäten (repräsentiert durch die Tupel der Relationen) konzentrieren.

Beispiel 3.6: (JOIN-Operator)

Die Abfrage aus Beispiel 3.21 lautet mit SQL-JOIN-Operator:

```
SELECT PNR, NAME
FROM PROJEKT JOIN ANGEST
ON P_LEITER = ANGNR
```

Durch diese Schreibweise wird klar, dass nur der JOIN von PROJEKT und ANGEST gefragt ist. Bedingungen an die einzelnen Tabellen (etwa: BERUF = 'PROGRAMMIERER') werden nicht gestellt.

Die Abfrage aus Beispiel 3.22 lautet mit SQL-JOIN-Operator:

```
SELECT NAME
FROM ANGEST NATURAL JOIN ANG_PRO
WHERE PNR = 17
```

Hier wird deutlicher: Alle Angestellten mit Arbeitsanteilen (NATURAL JOIN) am Projekt Nummer 17 (WHERE-Klausel) gesucht. Eine weitere Möglichkeit einen NATURAL JOIN in SQL zu schreiben ist:

```
SELECT NAME
FROM ANGEST JOIN ANG_PRO
USING ANGNR
WHERE PNR = 17
```

Das Schlüsselwort USING kann benutzt werden, um ein Attribut für den NATURAL JOIN explizit anzugeben z.B. wenn mehrere gleichnamige Spalten vorhanden sind. Die ON bzw. USING Klausel wurde eingeführt, um die JOIN Bedingungen syntaktisch von den Selektionsbedingungen in der WHERE Klausel zu trennen[6].

---

6 Wenn Sie ON oder USING benutzen, müssen Sie das Schlüsselwort JOIN zum verbinden der Tabellennamen verwenden.

Die Abfrage aus Beispiel 3.23 lautet mit JOIN...ON:

```
SELECT    DISTINCT NAME
FROM      ANGEST
          JOIN PROJEKT AS P1 ON ANGNR = P1.P_LEITER JOIN PROJEKT
          AS P2 ON ANGNR = P2.P_LEITER
WHERE  P1.PNR != P2.PNR
```

Wiederum werden die verschiedenen JOIN Bedingungen in das JOIN-Statement mit einbezogen (siehe auch Lösung zu **Übung 3.4**). Die WHERE-Klausel wird auf den Vergleich der beiden PNRs reduziert.[7]

Beispiel 3.24a: (OUTER JOINS)
Wir erweitern für dieses Beispiel die Tabelle PROJEKT um eine Zeile:

PROJEKT	P_NAME	PNR	P_BESCHR	P_LEITER
	DATAWAREHOUSE	12	...	205
	INTRANET	18	...	117
	PROJEKT 2000	17	...	198
	VU	33	...	198
	SQL KURS	34	...	NULL

Die Firma in unserem Beispiel hat demnach ein neues Projekt mit dem Namen 'SQL_KURS' initiiert. Das Projekt hat momentan noch keinen Leiter, außerdem sind ihm noch keine Mitarbeiter zugeordnet. Folgende Anfrage kann mit den bislang betrachteten JOINS nicht realisiert werden:

FINDE ALLE PROJEKTNAMEN UND DIE NUMMERN DER ZUGEORDNETEN ANGESTELLTEN

Die QUERY

```
SELECT PROJEKT.P_NAME, ANG_PRO.ANGNR
FROM PROJEKT NATURAL JOIN ANG_PRO
```

liefert das Ergebnis

DATAWAREHOUSE	205
INTRANET	117
INTRANET	198
PROJEKT 2000	112
PROJEKT 2000	198
VU	117

Somit geht die Information über das Projekt SQL KURS nicht in das Ergebnis der Abfrage ein. Um ein korrektes Ergebnis zu erhalten, müssen die Zeilen der Tabelle

---

7 Bei manchen DBMS müssen Sie die einzelnen JOINs verschachteln
...FROM (JOIN PROJEKT AS P1 ON ANGNR =P1.LEITER) JOIN PROJEKT AS P2....

Projekt, die nicht der JOIN Bedingung entsprechen erhalten bleiben. Hierzu dienen die OUTER JOINS[8].

- LEFT OUTER JOIN: Es bleiben die nicht der JOIN-Bedingung entsprechenden Zeilen der Tabelle erhalten, die links (vom Schlüsselbegriff OUTER JOIN) steht.
- RIGHT OUTER JOIN: Es bleiben die nicht der JOIN-Bedingung entsprechenden Zeilen der Tabelle erhalten, die rechts steht.
- FULL OUTER JOIN: Es bleiben die nicht der JOIN-Bedingung entsprechenden Zeilen der Tabellen erhalten, die links und rechts stehen.

Zur Verdeutlichung benutzen wir folgende Tabellen[9]:

EINS	
a	b
a1	b1
a2	b2
a3	NULL

ZWEI	
b	c
b1	c1
b2	c2
b3	NULL

DREI	
c	d
c1	d1
c2	d2
c3	NULL

Ein INNER JOIN über die Tabellen EINS und ZWEI

```
SELECT *
FROM EINS JOIN ZWEI
ON EINS.B = ZWEI.B
```

liefert das Ergebnis:

a1	b1	b1	c1
a2	b2	b2	c2

Ein LEFT OUTER JOIN über die Tabellen EINS und ZWEI

```
SELECT *
FROM EINS LEFT OUTER JOIN ZWEI
ON EINS.B = ZWEI.B
```

liefert das Ergebnis

a1	b1	b1	c1
a2	b2	b2	c2
a3	NULL	NULL	NULL

Entsprechend RIGHT OUTER JOIN und FULL OUTER JOIN:

---

8 Im Gegensatz dazu werden die bislang behandelten JOINS als 'INNER JOIN' bezeichnet. Es ist möglich INNER JOIN explizit in der Query anzugeben (FROM TABELLE1 INNER JOIN TABELLE2).

9 Die Namen der Tabellen und Spalten sind grau, die eigentlichen Inhalte weiß hinterlegt.

RIGHT OUTER JOIN

a1	b1	b1	c1
a2	b2	b2	c2
NULL	NULL	b3	NULL

FULL OUTER JOIN

a1	b1	b1	c1
a2	b2	b2	c2
a3	NULL	NULL	NULL
NULL	NULL	b3	NULL

OUTER JOINS über mehrere Tabellen muss man in Abweichung von der INNER JOIN Syntax verschachteln und klammern:

```
SELECT *
FROM    EINS
            LEFT OUTER JOIN ZWEI ON EINS.B = ZWEI.B
            LEFT OUTER JOIN DREI  ON ZWEI.C = DREI.C
```

Das Ergebnis der Query ist wie folgt:

a1	b1	b1	c1	c1	d1
a2	b2	b2	c2	c2	d2
a3	NULL	NULL	NULL	NULL	NULL

Eine Query, die das richtige Ergebnis zur Frage nach den Projekten und den Angestellten liefert, ist demnach:

```
SELECT PROJEKT.P_NAME, ANG_PRO.ANGNR
FROM PROJEKT
        LEFT OUTER JOIN ANG_PRO
        ON PROJEKT.PNR = ANG_PRO.PNR
```

Sie liefert folgendes Ergebnis:

DATAWAREHOUSE	205
INTRANET	117
INTRANET	198
PROJEKT 2000	112
PROJEKT 2000	198
VU	117
SQL KURS	NULL

Übung 3.11a:
Formulieren Sie eine Query, die die Namen der Projekte und die Namen und Nummern von evtl. zugeordneten Mitarbeitern ausgibt.

Die Grundidee hinter dem Outer Join besteht darin, Informationen zu erhalten, die bei einem Inner Join verloren gehen. Dies passiert weil in einer der beiden Tabellen Zeilen vorhanden sind, die nach der Join-Bedingung keiner Zeile in der anderen Tabelle entsprechen. Ein Outer Join bezieht diese Zeilen mit ein und setzt in die Zeilenpositionen einen NULL-Wert, in der Werte einer passenden Zeile der anderen Tabelle auftauchen würden – wenn es eine passende Zeile gäbe. Ein Outer Join beinhaltet Zeilen, die ein Inner Join herausfiltert. Sie sollten Outer Joins benutzen, wenn Sie meinen, dass Informationen sonst verloren gehen könnten.