4. Relationale Datenbanksprachen

Basierend auf dem Tupelkalkül und der relationalen Algebra wurden mit dem Aufkommen relationaler DBMS auch spezielle Sprachen entwickelt.

- SQL ist die derzeit am weitesten verbreitete Anfragesprache
- von praktischer Bedeutung ist auch noch QBE (Query by Example)
- für das DBMS Ingres wurde die Anfragesprache Quel entwickelt, die sich (leider) gegenüber SQL nicht durchgesetzt hat.

Bei Anfragesprachen wird unterschieden zwischen der

Datendefinitionssprache (DDL)

- Anlegen und Ändern der Datenstrukturen für die drei Ebenen einer Datenbank (externe Ebenen, konzeptionelle Ebene, physische Ebene)
- Festlegen von Integritätsbedingungen
- Festlegen der Zugriffsrechte

Datenmanipulationssprache (DML)

- Einfügen, Ändern und Löschen von Datenobjekten
- Formulierung von Anfragen

4.1 SQL

SQL (structured query language) wurde bei IBM als Sprache des relationalen DBMS System R entwickelt (1974, *D.D. Chamberlin et al.*)

heute ist SQL quasi der Standard für Sprachen relationaler DBMS

- SQL1, 1985
- SQL2, 1992 (wird auch als SQL92 bezeichnet)
- SQL3 (bzw. SQL:1999)

SQL wird als interaktive Sprache eingesetzt, kann aber auch durch eine geeignete Kopplung in einer Programmiersprachen wie z. B. C und Java genutzt werden.

SQL kann als eine Mischform aus einer erweiterten relationalen Algebra und dem relationalem Kalkül verstanden werden.

Anmerkung

Das offizielle Dokument, in dem der Standard von SQL beschrieben wird, ist sehr groß, so daß im Rahmen dieser Vorlesung deshalb nur die wichtigsten Konzepte von SQL vorgestellt werden können

Viele Hersteller wie Oracle haben in ihren Systemen Erweiterungen von SQL implementiert. Andererseits wurde in Oracle SQL2 nicht vollständig, sondern nur der sogenannte Entry-Level implementiert.

Wichtige skalare Datentypen

Standardtypen in ORACLE

char(*size*)	Zeichenkette mit konstanter Länge *size*, *size* < 2000.
varchar2(*size*)	variabel lange Zeichenkette mit *size* < 4000
number(*g*,*d*)	*g* = #Gesamtstellen, *d* = #Nachkommastellen

- Weiterhin werden z. B. noch angeboten: decimal, float, real und double precision

date	Datum: Jahr, Monat,…, Sekunde

spezielle Typen

long	variabel lange Zeichenkette mit maximal 2 GB.

- maximale ein Attribut vom Typ long in einer Relation
- eingeschränkte Form der Anfragebearbeitung

blob	variabel lange Bytefolge mit maximal 4 GB
rowid	Typ einer Pseudospalte, die Schlüsselkandidat ist.

4.1.1 DDL

Klarstellung

Im folgenden werden nun einige wichtige Aspekte bei der Datendefinition erläutert ohne jedoch auf viele der angebotenen Optionen einzugehen.

Wir werden beispielhaft die Definition von Datenstrukturen der konzeptionellen Ebene (Relation), der internen Ebene (Index) und der externen Ebene (View) erläutern.

Beispiele basieren auf dem folgenden Datenbankschema:

Kunde (KName, KAdr, Kto)

Auftrag (KName, Ware, Menge)

Lieferant (LName, LAdr, Ware, Preis)

Anlegen eines Relationenschemas

create table <Relationen-Name> (<Relationenkomponente>[,<Relationenkomponente>]*)

<Relationenkomponente> ::= <Spaltendefinition> | <Integritätsbedingung>

<Spaltendefinition>::= <Attributname> <Typ> [<Defaultwert>] [not null | unique]

<Defaultwert>::= default <Literal> | null

Die genaue Behandlung von Integritätsbedingungen wird später erfolgen.

Beispiel:

create table Kunde
(KName char (20),
KAdresse varchar2 (50) unique,
Kto decimal (7) not null,
primary key (KName))

Anmerkungen

unique drückt aus, daß dieses Attribut ein Schlüsselkandidat ist. Wird ein Schlüsselkandidat durch mehrere Attribute A_1, …, A_n gebildet, so wird dies durch die Integritätsbedingung unique (A_1,…,A_n) angegeben.

not null sagt aus, daß das Attribut explizit belegt werden muß. Es dürfen keine Null-Werte auftreten.

Durch primary key (A_1,…,A_n) wird festgelegt, daß die Attributmenge {A_1,…,A_n} der Primärschlüssel der Relation ist.

Durch Angabe eines Defaultwertes wird beim Einfügen eines Tupels dieser Wert zur Initialisierung benutzt, wenn explizit keine Wertzuweisung vorgenommen wurde.

Ändern eines Relationenschema

alter table <Relationen-Name> add <Spaltendefinition>

Beachte:

Hierbei ist not null nur dann erlaubt, wenn auch ein Defaultwert spezifiziert wird.

Durch den alter-Befehl lassen sich noch andere Änderungen des Relationenschemas vornehmen.

Löschen eines Relationenschemas

drop table <Relationen-Name>

Anlegen eines Index

Indexe dienen zur Verbesserung der Anfragezeit

ein Index bezieht sich auf ein Attribut, bzw. eine Folge von Attributen

Maß für die Effizienz ist i. A. die Anzahl der Zugriffe auf die Platte

in kommerziellen Systemen: B-Bäume und Hashverfahren

create [unique] index <Index-Name> on <Relationen-Name>

(<Attributname> [<Ordnung>] [,<Attributname>[<Ordnung>]]*)

[cluster] <Ordnung>::= Asc | Desc

unique: Für alle Attributsnamen keine zwei Tupel mit gleichen Werten erlaubt ⇒ erfüllt Schlüsselbedingung.

cluster:Die Tupel der Relation werden tatsächlich in die Indexstruktur eingefügt und nicht nur Verweise ⇒ nur ein Cluster-Index pro Relation.

Beispiel:

create unique index Kundenindex
on Kunde (KName,KAdresse)

Löschen eines Index

drop index <Index-Name>

Anlegen von Sichten

Sichten entsprechen externen DB-Schemata.

In relationalen Systemen werden Sichten als (abgeleitete) Relationen aufgefaßt, die durch Anfragen definiert werden.

create view <Sicht-Name> [(<Attributname>[,<Attributname>]*)] as <Subquery>

Beispiel:

create view Gute_Kunden as
select * from Kunde where Kto > 100

Beachte: Das SQL Schlüsselwort '*' stellt eine Kurzschreibweise für die gesamte Attributsliste der hinter dem from angegebenen Relationen dar.

Löschen von Sichten

drop view <Sicht-Name>

4.1.2 DML

Anfragen an die Datenbank werden in der DML formuliert

Grundschema:

select < Liste von Attributsnamen >	z.B.	select KName
from < ein oder mehrere Relationennamen >		from Kunde
[where< Bedingung >]		where Kto < 0

Bemerkungen:

Die select-Klausel entspricht der Projektion in der relationalen Algebra (und nicht der Selektion).

Die Bedingung nach der where-Klausel enthält

1. Vergleichsoperatoren *(<, >, = ...)*
2. boolesche Operatoren (and, or , not)
3. Mengenoperatoren (in, not in) und Quantoren(any, some, all)

Reihenfolge der Ausführung wird durch Klammern bestimmt.

Attribute mit gleichen Namen, die zu verschiedenen Relationen gehören, werden mittels des Relationennamen unterschieden.

Algebra-Operationen in SQL

Relation R

```
select *
from R
```

Bei Angabe von "*" inder select-Klausel werden alle Attribute der Relation aus der from-Klausel ausgegeben.

Projektion $\pi_{A,C}(R)$

```
select distinct A, C
from R
```

Ohne das Schlüsselwort distinct wird als Ergebnis eine M-Relation erzeugt.

Selektion $\sigma_{B=b}(R)$

```
select *
from R
where B = b
```

kartesisches Produkt $R \times S$

```
select *
from R, S
```


Theta-Join auf Relationen R(A,B) und S(C,D) $R \underset{B\theta D}{} S$

```
select *
from R, S
where B θ D
```

Vereinigung der Relationen R(A,B) und T(A,B)

```
select * from R
union
select * from T
```

Differenz der Relationen R und T

```
select * from R
minus
select * from T
```

Allgemeine Bedeutung der "select … from … where"-Klausel in der relationalen Algebra:

```
select distinctA,B,C,..
from    R,S,T,...
where   F
```

$\pi_{A,B,C,\ldots}(\sigma_F(R \times S \times T \times \ldots))$

Damit ist insbesondere die Reihenfolge bei der Verarbeitung der Klausel bestimmt.

Bemerkungen zu der Duplikatbeseitigung

Die gewöhnliche select-Klausel beseitigt keine Duplikate in der Ergebnisrelation. Dies ist aber durch Hinzufügen des Schlüsselworts *distinct* möglich:

```
select distinct A, B, C, …
from R,S,T,…
where Bedingung
```

Es wird also dadurch *distinct* als Ausgabe eine Relation erzeugt. Ansonsten wird eine M-Relation ausgegeben.

Die minus-Operation auf zwei Multi-Mengen entspricht der Semantik wie wir sie bereits bei der erweiterten relationalen Algebra kennengelernt hatten. Das Schlüsselwort *minus* wird nur von Oracle benutzt. In SQL92 wird stattdessen das Schlüsselwort *except* benutzt.

Bei der Vereinigung auf Relationen werden automatisch Duplikate beseitigt. Dies gilt auch für M-Relationen. Sollen Duplikate nicht beseitigt werden, muß noch hinter dem Schlüsselwort *union* das Schlüsselwort *all* folgen.

Beispielanfragen

Datenbankschema:

Kunde (KName, KAdr, Kto)

Auftrag (KName, Ware, Menge)

Lieferant (LName, LAdr, Ware, Preis)

Welche Lieferanten liefern Milch oder Mehl?

select distinct LName
from Lieferant
where Ware = ‘Mehl’ or Ware = ‘Milch’

Welche Lieferanten liefern irgendetwas, das Huber bestellt hat ?

select distinct LName
from Lieferant, Auftrag
where Lieferant.Ware = Auftrag.Ware and KName = ‘Huber’

Aggregatfunktionen

Funktionen *count*, *sum*, *avg*, *min* und *max* können auf eine Menge von Zahlen, die als Spalte einer Relation gegeben ist, angewandt werden.

Bei Angebe des Schlüsselworts *distinct* vor dem Aggregatsattribut werden zunächst die Duplikate beseitigt bevor das Aggregat berechnet wird.

Wieviele Lieferanten mit verschiedenen Namen gibt es?

select count (distinct LName)
from Lieferant

Wieviele Liter Milch sind insgesamt bestellt?

select sum (Menge)
from Auftrag
where Ware = ‘Milch’

Gruppieren

Allgemeinere Form der ’select...from...where’-Klausel:

select .....
from.....
[where.....]
[group by <group-by-expression>[,<group-by-expression>]*]
[having < Bedingung>]
[order by <order-expression>]

“group by”-Klausel

Ein group-by-expression ist ein Ausdruck, der sich nur auf die Attribute beziehen, über die nicht das Aggregat gebildet wird. Tupel mit gleichen Werten für die angegebenen Ausdrücke werden in Gruppen zusammengefaßt.

Pro Gruppe erzeugt die Anfrage ein Tupel der Ergebnisrelation. Deshalb sind hinter der select-Klausel nur Attribute mit einem Wert pro Gruppe zugelassen.

“having”-Klausel

Auswahl der Gruppen anhand der Bedingung (es dürfen nur Argumente mit einem Wert pro Gruppe auftreten)

Beispiel:

select LName
from Lieferant
where Preis > 100
group by LName
having count (*) > 5

Beachte:

count(*) zählt die verschiedenen Tupel in einer Gruppe

“order by”-Klausel

order by [Asc|Desc] A_1,…,[Asc|Desc] A_n

Durch diese Klausel wird die Ausgabe des SQL-Befehls sortiert ausgegeben, wobei die Sortierreihenfolge bzgl. den angegebenen Attributen erfolgt (absteigend: *DESC* oder aufsteigend: *ASC*). Statt eines Attributs kann auch ein Ausdruck benutzt werden.

Die order-by Klausel ist die letzte Klausel in einem SQL-Befehl

Unterschied zwischen SQL92 und dem SQL von Oracle

- Bei SQL92 kann nur ein Attribut (Ausdruck) in der order-by Klausel verwendet werden, der auch in der select-Klausel vorzufinden ist.
- Bei Oracle kann auch bzgl. Attributen sortiert werden, die durch die select-Klausel aus der Relation eliminiert werden.

Arithmetische Ausdrücke in der select-Klausel

Gib die Preise der Waren der einzelnen Lieferanten in Dollar an!

```
select LName, Ware, Preis / 1.89 as Dollar
from Lieferant
```

Durch *as* wird nun ein neuer Attributname fuer die Ausgaberealtion definiert. Diese Attributnamen können in Oracle bereits in der order-by Klausel benutzt werden.

Beispiele

Erstelle eine alphabetisch geordnete Liste aller Waren, in der für jede Ware der minmale, maximale und der Durchschnittspreis angegeben ist!

```
select    Ware, min (Preis) as MinP, max (Preis) as MaxP, avg (Preis) as AvgP
from      Lieferant
group by  Ware
order by  Ware
```

Welche Waren werden nur von einem Lieferanten geliefert?

```
select    Ware
from      Lieferant
group by  Ware
having    count(*) = 1
```

Sortiere die Bestellungen nach Waren, für jede Ware die Kunden nach der Größe der Bestellung!

```
select    *
from      Auftrag
order by  Ware, Menge Desc
```


Tupelvariablen

Anfragen sollen auch Relationen mit sich selber verknüpfen können. Dies ist ein Grund für die Einführung von Tupelvariablen.

Beispiele:

Gesucht sind Namen und Adressen aller Kunden, deren Kontostand kleiner als der von Huber ist.

```
select K1.KName, K1.KAdr
from Kunde K1, Kunde K2
where K1.Kto < K2.Kto and K2.KName = 'Huber'
```

Finde alle Paare von Lieferanten, die eine gleiche Ware liefern?

```
select distinct L1.LName, L2.LName
from Lieferant L1, Lieferant L2
where L1.Ware = L2.Ware and L1.LName < L2.LName
```


Geschachtelte Anfragen

In einer SQL-Anweisung kann in der where- und in der from-Klausel wieder SQL-Anweisungen auftreten. Man spricht dann von einer geschachtelten Anfrage.

In der where-Klausel wird dabei noch unterschieden, ob das Resultat der Unteranfrage einen skalaren Wert oder eine Relation zurückliefert.

Skalare Unteranfragen

Welche Lieferanten liefern Schuhe, deren Preis 50% unter dem Durschnittspreis für Schuhe liegen?

```
select LName
from Lieferant
where Ware = 'Schuhe' and Preis < (   select avg(Preis)/2
                                      from Lieferant
                                      where Ware = 'Schuhe')
```

Skalare Unteranfragen können in SQL'92 sogar in der select-Klausel einer Anfrage auftreten. Dies wird aber derzeit nicht in Oracle unterstützt.

Skalare Unteranfragen mit Exists

In der where Klausel werden auch Unteranfragen erlaubt, die einen Booleschen Wert zurückliefern. Diese sind durch das Schlüsselwort exists gekennzeichnet. Dabei ist die Bedingung

– [not] exists <Subquery>

wahr, falls die Subquery nicht leer ist.

Welche Lieferanten liefern irgendetwas, das Huber bestellt hat?

```
select distinct LName
from Lieferant L
where exists   (select Ware
                from Auftrag
                where L.Ware = Ware and KName = 'Huber')
```

Bei dieser Unteranfrage wird nun Bezug genommen auf eine Tupelvariable, die in der äußeren Anfrage definiert wurde. Bei der Auswertung der Anfrage wird nun entsprechend so wie beim Tupelkalkül vorgegangen ("von außen nach innen").

Mengenwertige Unteranfragen

Durch Verwendung vom Schlüsselwort in kann nun auch getestet werden, ob ein Attribut einen Wert einer Menge annimmt.

```
select *
from Kunde
where Kname not in (select Kname from Auftrag)
```

Wie kann die Anfrage formuliert werden, ohne dabei eine Subquery zu benutzen?

Soll nun getestet werden, ob ein Attribut mit allen Elementen einer Menge in einer bestimmten Beziehung stehen, kann das Schlüsselwort all benutzt werden.

Gib für alle Waren die Namen der günstigsten Lieferanten

```
select Name, Ware
from Lieferant L
where Preis <= all(   select Preis
                      from Lieferant
                      where Ware = L.Ware)
```


Anfragen mit Allquantoren

Da $\forall x(\psi(x)) \Leftrightarrow \neg\exists x(\neg\psi(x))$, können alle Anfragen mit einem Allquantor in äquivalente Anfragen umgeformt werden, die nur noch Existenzquantoren benutzen.

Welche Lieferanten liefern alles, was Huber bestellt hat?

```
select distinct LName
from Lieferant L
where not exists( select Ware
                  from Auftrag
                  where KName = 'Huber' and not Ware in (select Ware
                                                         from Lieferant
                                                         where LName =L.LName))
```


Unteranfragen in der from-Klausel

Da eine SQL-Anfrage eine Relation erzeugt spricht formal nichts dagegen statt einer Relation eine Anfrage in der from-Klausel zu benutzen.

Beispiel:

```
select   LName
from     (select LName, count(*) as Anzahl
          from Lieferant
          where Preis > 100
          group by LName)
where    Anzahl > 5
```

In SQL92 wird dies auch noch syntaktisch über die Einführung von neuen Schlüsselwörten in der from-Klausel unterstützt. Z.B.:

– from R natural join S

– from R join S on R.A θ S.B

Nullwerte

Ein spezieller Wert *null* zeigt bei einem Attribut in einer Relation an, daß der Wert nicht bekannt ist.

- In arithmetischen Ausdrücken ist das Ergebnis *null*, wenn einer der Operanden den Wert *null* hat.
- SQL baisert auf einer dreiwertigen Logik mit Werten *true*, *false* und *unknown*. Ein zweistelliges Atom liefert den Wert *unknown*, wenn eine der Argumente *unknown* ist.
- Logische Ausdrücke liefern nun folgendes Ergebnis:

not	
true	false
unknown	unknown
false	true

and	true	unknown	false
true	true	unknown	false
unknown	unknown	unknown	false
false	false	false	false

or	true	unknown	false
true	true	true	true
unknown	true	unknown	unknown
false	true	unknown	false

- In der where-Klausel werden nur Tupel selektiert, die *true* liefern. Zusätzlich kann nun über die Überprüfung “where B is null” alle in B nullwertigen Tupel selektiert werden.
- Beim Gruppieren wird *null* als eigenständiger Wert betrachtet.
- Beim Sortieren wird *null* stets als höchster Wert interpretiert.

Suche nach Teilstrings

Gibt es Kunden in München?

select *
from Kunde
where KAdr like ‘%München%’

Beachte:

Das SQL-Schlüsselzeichen ‘%’ repräsentiert einen beliebigen String, während ‘_ ‘ ein einziges Zeichen markiert.

Rekursive Anfragen

zusätzlich zu unseren bisherigen Relation nehmen wir an, daß es eine Relation Liefert(Lieferant, Kunde) gibt.

- Beachte, daß Lieferanten auch wieder in der Rolle der Kunden auftreten können.

folgende Anfrage:
Gib die Lieferanten, die irgendwas (direkt oder indirekt) an “Müller” liefern.

```
select Lieferant
from Liefert
where Kunde = ‘Müller’
```

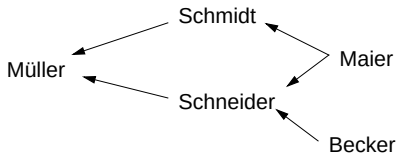
Diese Anfrage liefert aber nur die Lieferanten, die direkt an Müller liefern!

SQL, relationale Algebra oder Relationenkalkül bieten keine Möglichkeiten zur effektiven Berechnung rekursiver Anfragen.

spezielle Lösung von ORACLE

connect By-Klausel

```
select Lieferant
from Liefert
start with Kunde = ‘Müller’
connect by Kunde = prior Lieferant
```



Änderen einer Relation

Einfügen:

insert

into <Relationen-Name> [(<Attributname> [, <Attributname>]*)]

values (<Konstante> [, Konstante]*)

oder

insert

into <Relationen-Name> [(<Attributname> [, <Attributname>]*)]

select ... from ... where ....

Löschen:

delete
from <Relationen-Name>
[where <Bedingung>]

Verändern:

update <Relationen-Name>

set <Attributname> = <Ausdruck> [, <Attributname> = <Ausdruck>]*
[where <Bedingung>]

Beispiele

Füge den Kunden Schmidt mit dem Kontostand 0 DM ein!

insert into Kunde (KName, Kto)

values (‘Schmidt’, 0)

Erhöhe den Kontostand von Schmidt um 200!

update Kunde

set Kto = Kto + 200

where KName = ‘Schmidt’

4.1.3 Anlegen der externen Schemata

Sichten anlegen:

Sichten entsprechen den externen DB-Schemata.

In relationalen Systemen werden Sichten als (abgeleitete) Relationen aufgefaßt, die durch Anfragen definiert werden.

create view <Sichtname> [(<Attributname>[,<Attributname>]*)] as <Subquery> [with check option]

Beispiel:

create view Gute_Kunden as

select * from Kunde where Kto > 100
with check option

Sichten löschen:

drop view <Sicht-Name>

Ändern einer Sichteninstanz

Durch das Schlüsselwort *with check option* können nur Datensätze in eine Sicht eingefügt werden, die bei einer Suche auch wieder gefunden werden können.

Beim Einfügen eines Tupels in einer Sicht müssen die entsprechenden Basisoperationen angepaßt werden.

Die Zuordnung zu den Basisrelationen ist aber nicht immer möglich!

- z. B. wenn ein Attribut einer Sicht durch eine Aggregatsfunktion berechnet wird.

Sichten sind in ORACLE veränderbar, wenn folgende Bedingungen gelten:

- keine Aggregatfunktionen
- keine Anweisungen mit distinct, group by, having, union und minus
- from-Klausel enthält nur eine Relation
- ein Schüssel der Basisrelationen muß in der select-Klausel enthalten sein.

es gibt aber durchaus veränderbare Sichten, die aber nicht alle vier der oben genannten Bedingungen, erfüllen.