4.5 Struktur der Syntax

SFW-Ausdruck

SELECT $A_1, \ldots, A_n$	Liste der Attribute
FROM $R_1, \ldots, R_m$	Liste der Relationen
WHERE F	Bedingung
GROUP BY $B_1, \ldots, B_k$	Liste der Gruppierungsattribute
HAVING G	Gruppierungsbedingung
ORDER BY H	Sortierordnung

Orthogonalität der Syntax

- Ein *tabellenwertiger* Ausdruck ist überall dort zulässig ist, wo eine Tabelle stehen darf.
- Ein *skalarer* Ausdruck ist überall dort zulässig, wo ein skalarer Wert stehen darf.
- Ein *bedingter* Ausdruck ist überall dort zulässig, wo ein Wahrheitswert stehen darf.

(1) tabellenwertiger Ausdruck

Tabellen in SQL

Ein Anfrageausdruck in SQL definiert im Allgemeinen eine Tabelle.

- Jeder Tabellenbezeichner ist ein Anfrageausdruck.
- Jeder SFW-Ausdruck ist ein Anfrageausdruck.
- Ein Verbundausdruck ist ebenfalls ein Anfrageausdruck.
- Die üblichen Mengenoperatoren können verwendet werden, um Anfrageausdrücke zu bilden.
- Ein *Tabellen-Konstruktor* der Form
  `VALUES ('a1',...,'an'), ('b1',...,'bn'), ...` ist ein Anfrageausdruck.

Liste die Namen, die für Städte und Länder verwendet werden.

```
SELECT Name
FROM ( SELECT SName AS Name FROM Stadt
       UNION
       SELECT LName AS Name FROM Land ) T
```

Berechne die Anzahl der Menschen aller Länder, die in der größten Stadt ihres Landes leben.

```
SELECT SUM(GroßStädter)
FROM ( SELECT LCode, MAX(Einwohner) AS Großstädter
       FROM Stadt
       GROUP BY LCode ) T
```

Division à la Algebra:

Seien X_1, X_2 Formate, $X_2 \subset X_1$, $Z = X_1 - X_2$
und weiter $r_1 \subseteq Tup(X_1)$ (Dividend), $r_2 \subseteq Tup(X_2)$ (Divisor), wobei $r_2 \neq \emptyset$.

$$r_1 \div r_2 = \pi[Z]r_1 - \pi[Z](((\pi[Z]r_1) \times r_2) - r_1)$$

Welche Länder sind in denselben Organisationen wie Österreich vertreten?

$$\pi[LCode, Organisation]Mitglied \div \pi[Organisation](\sigma[LCode =' A']Mitglied)$$

```
SELECT DISTINCT LCode FROM Mitglied
MINUS /* EXCEPT */
SELECT LCode FROM (
   SELECT LCode, Organisation FROM (
      (SELECT LCode FROM Mitglied)
      CROSS JOIN
      (SELECT Organisation FROM Mitglied WHERE LCode = 'A') )
   MINUS /* EXCEPT */
   SELECT LCode, Organisation FROM Mitglied
)
```

(2) skalarer Anfrageausdruck

Anstelle eines Wertes, bzw. Spaltenbezeichners, ist auch ein geklammerter Tabellenausdruck zulässig, sofern er *skalar* ist, d.h. genau einen Wert definiert.

Bestimme zu jeder Stadt den Mittelwert der Einwohnerzahl aller Städte, die weniger Einwohner haben als sie selbst.

```
SELECT SName, Einwohner,
   ( SELECT AVG(Einwohner) FROM Stadt S2
     WHERE S2.Einwohner < S1.Einwohner )
   AS kleinerMittelwert
FROM Stadt S1
```

Bestimme diejenigen asiatischen Länder, deren Flächenanteil in Asien kleiner ist als der Anteil der Türkei in Asien.

```
SELECT LCode, Prozent FROM Lage
WHERE Kontinent = 'Asia' AND
      Prozent <
      ( SELECT Prozent FROM Lage
        WHERE LCode = 'TR' AND Kontinent = 'Asia' )
```

(3) bedingter Ausdruck

Welche Länder befinden sich im gleichen Kontinent wie Russland?

```
SELECT DISTINCT L2.LCode
   FROM Lage L1, Lage L2
   WHERE L1.Kontinent = L2.Kontinent AND L1.LCode = 'RU'

SELECT DISTINCT LCode FROM Lage
   WHERE Kontinent IN
      (SELECT Kontinent FROM Lage WHERE LCode = 'RU')
```

- Ein Ausdruck, dem ein Wahrheitswert zugeordnet werden kann, ist ein *bedingter* Ausdruck.
- Bedingte Ausdrücke sind Teil einer `WHERE`-, `HAVING`- oder `ON`-Klausel.
- Wesentlich für die korrekte Verwendung bedingter Ausdrücke ist die Miteinbeziehung des Auftretens von Nullwerten.

Probleme mit Nullwerten.

Vergleiche unter Annahme Tabelle `Land` enthält `Wunderland` mit Nullwert für `HStadt`:

Welche Städte sind keine Hauptstadt, d.h., heißen nicht so, wie die Hauptstadt irgendeines Landes?

```
SELECT SName FROM Stadt S
WHERE S.SName NOT IN ( SELECT HStadt FROM Land )
```

Resultat: leere Tabelle

```
SELECT SName FROM Stadt S
WHERE NOT EXISTS (
   SELECT HStadt FROM Land
   WHERE HStadt = S.SName )
```

Resultat: `Freiburg, Munich, Nuremberg, Karlsruhe`

Welche Städte sind keine Hauptstadt?

```
INSERT INTO Land VALUES ('Wunderland', 'WU', null, 0)

SELECT SName FROM Stadt S
WHERE S.SName NOT IN ( SELECT HStadt FROM Land )
```

SELECT HStadt FROM Land

Tabelle:

Vienna
Bern
Berlin
Cairo
Paris
Rome
Moscow
Ankara
null

'Freiburg' IN (SELECT HStadt FROM Land) $\equiv$ UNKNOWN
$\Rightarrow$ 'Freiburg' NOT IN (SELECT HStadt FROM Land) $\equiv$ UNKNOWN

da der Vergleich 'Freiburg' = null den Wahrheitswert UNKNOWN liefert.

4.6 Datentypen

Basis-Datentypen

SQL bietet eine Fülle von unterschiedlichen Datentypen an, mittels derer die Wertebereiche der Spalten einer Tabelle festgelegt werden können.

- `INTEGER`, `SMALLINT`
- `NUMERIC`, `DECIMAL`
  Angabe Anzahl Ziffern insgesamt und Anzahl Kommastellen.
- `REAL`, `DOUBLE PRECISION`, `FLOAT`
- `CHARACTER`, `CHARACTER VARYING`, `CHARACTER LARGE OBJECT`
- `BIT`, `BIT VARYING`, `BINARY LARGE OBJECT`
- `BOOLEAN`
- `DATE`, `TIME`, `TIMESTAMP`, `INTERVALL`, ...

Definition einer Tabelle am Beispiel

```
CREATE TABLE Stadt (
   SName        VARCHAR(50),
   PName        VARCHAR(50),
   LCode        CHAR(4),
   Einwohner    INTEGER,
   LGrad        NUMBER,
   BGrad        NUMBER,
   PRIMARY KEY (SName,PName,LCode) )
```

DESCRIBE liefert die Attribute einer Tabelle inklusive Typen.

```
DESCRIBE Stadt;
```

Objektrelationale Datenbanken: Konstruierte Datentypen

Ein Datentyp heißt *konstruiert*, sofern seine Werte aus Werten anderen Typen, sogenannter *Element-Typen*, zusammengesetzt sind.

- Der Datentyp `ARRAY` fasst mehrere Werte seines Element-Typs geordnet zusammen, die über einen Index referenziert werden können.
- Der Typ `ROW` lässt hingegen zu, dass Werte unterschiedlicher Element-Typen geordnet zusammengefasst werden und das der Zugriff auf die einzelnen Komponenten über Bezeichner, analog zu Spaltenbezeichern, ermöglicht wird.
- Der Datentyp `MULTISET` fasst mehrere Werte eines Element-Typs, möglicherweise mit Duplikaten, zu einer ungeordneten Menge zusammen.

```
CREATE TABLE Land (
    ⋮
    Provinzen VARCHAR(50) ARRAY[20]
    Organisationen
        ROW(Organiation VARCHAR(50), Art VARCHAR(20)) MULTISET
);

CREATE TABLE Stadt (
    ⋮
    Koordinaten ROW(LGrad NUMBER, BGrad NUMBER )
);
```

- Die fünfte Provinz eines Landes wird mittels `Provinzen[5]` referenziert.
- Der Längengrad innerhalb der Koordinaten einer Stadt kann mittels eines Pfadausdrucks der Form `Koordinaten.LGrad` angesprochen werden.
- Mittels `('EU','member') ELEMENT Organisationen` wird getestet, ob die Mitgliedschaften eines Landes bezüglich der EU den *member*-Status hat.

4.7 Einfügen, Löschen und Ändern

Einfügen

Aufnahme eines neuen Mitgliedes in die EU.

```
INSERT INTO Mitglied (LCode, Organisation, Art)
VALUES ('PL', 'EU', 'member')
```

Alle Länder, die in irgendwelchen Organisationen vertreten sind aber noch nicht in der Relation Land auftreten, werden in diese Relation übernommen.

```
INSERT INTO Land (LCode)
   SELECT DISTINCT M.LCode
   FROM Mitglied M
   WHERE NOT EXISTS (
      SELECT L.LCode
      FROM Land L
      WHERE L.LCode = M.LCode )
```

- Mittels `INSERT` kann eine neue Zeile, oder eine Menge von neuen Zeilen in eine Tabelle `T` eingefügt werden.
- Werden nicht zu allen Attributen von `T` Werte gegeben, so werden für die fehlenden Werte möglicherweise vorgesehene Default-Werte, bzw. der Nullwert `null` genommen.
- Die Angabe der Spaltennamen kann entfallen, wenn die Werte in der Reihenfolge der Spaltennamen in der `CREATE`-Anweisung definiert werden.

Sequenznummern

Beim Einfügen von Zeilen muss die Eindeutigkeit des Primärschlüssels gewährleistet sein.

Beispiel Identitätsspalte

```
CREATE TABLE Land (
   LandNr INTEGER GENERATED ALWAYS AS IDENTITY (
      START WITH 1
      INCREMENT BY 1
      MINVALUE 1
      MAXVALUE 100000
      NO CYCLE ),
   ⋮
)

INSERT INTO Land (LName, HStadt, Fläche)
VALUES ('Bavaria', 'Munich', 70)
```

generierte Spalten

Der Wert eines Attributes ergibt sich automatisch aus den Werten anderer Attribute desselben Tupels.

Beispiel

```
CREATE TABLE Land (
   LandNr INTEGER GENERATED ALWAYS AS IDENTITY ( ... ),
   ⋮
   Fläche      NUMBER,
   Einwohner   NUMBER,
   Dichte GENERATED ALWAYS AS (Einwohner / Fläche)
)
```

Löschen

- `DELETE FROM T WHERE P`
- Es werden alle Tupel aus `T`, für die der bedingte Ausdruck `P` wahr ist, markiert und anschließend aus `T` entfernt.

Löschen des gesamten Inhalts der Tabelle `Stadt`.

```
DELETE FROM Stadt
```

Löschen von ausgewählten Zeilen.

```
DELETE FROM Stadt
WHERE Einwohner < (
   SELECT AVG(Einwohner) FROM Stadt )
```

Ändern

- ```
  UPDATE T
     SET A_1 = val_1,..., A_n = val_n
     WHERE P
  ```
- Anstelle eines direkten Wertes innerhalb einer Zuweisung kann auch ein skalarer Ausdruck stehen.

Im Zuge der Euro-Umstellung werden die Angaben des Bruttosozialprodukts angepasst.

```
UPDATE Land
   SET BruttoSP =
      CASE BruttoSP
         WHEN LCode = 'D' THEN BruttoSP * 0,5
         WHEN LCode = 'F' THEN BruttoSP * 0,16
         ELSE NULL
      END
```
```

4.8 Sichten

- Eine Sicht `V` ist eine durch einen Anfrageausdruck `E` definierte Tabelle:
- ```
  CREATE VIEW V AS
      <E>
  ```
- Im Unterschied zu den als Sicht definierten Tabellen bezeichnen wir die mittels `CREATE TABLE` definierten Tabellen als *Basistabellen*.
- Bezeichner von Sichten dürfen in SQL überall stehen, wo ein Tabellenbezeichner stehen darf.
  ```

Definiere zu der Tabelle Benachbart eine bezüglich Symmetrie abgeschlossene Tabelle symBenachbart in Form einer Sicht.

Benachbart

LCode1	LCode2
CH	D
CH	F
CH	I
D	F
I	F

```
CREATE VIEW symBenachbart AS
   ( SELECT LCode1 AS Von, LCode2 AS Nach
     FROM Benachbart )
   UNION
   ( SELECT LCode2 AS Von, LCode1 AS Nach
     FROM Benachbart )
```

Welche Länder sind zu Deutschland benachbart?

```
SELECT Nach FROM symBenachbart
WHERE Von = 'D'
```

Materialisierte und virtuelle Sichten

- Ein Datenbanksystem kann Sichten entweder bei Bedarf jeweils neu berechnen, oder eine einmal berechnete Sicht für weitere Bearbeitungen permanent speichern. Im ersten Fall redet man von einer *virtuellen* Sicht, im zweiten Fall von einer *materialisierten* Sicht.
- Soll eine Anfrage bearbeitet werden, die sich auf eine virtuelle Sicht bezieht, so wird vor Ausführung der Anfrage der Name der Sicht durch den sie definierenden Ausdruck ersetzt (*Anfrage-Modifizierung*).
- Gegenüber einer materialisierten Sicht hat eine virtuelle Sicht den Vorteil, dass ihr Inhalt garantiert dem aktuellen Zustand der Datenbank entspricht.
- Standardmäßig ist eine Sicht einer Datenbank virtuell. Materialiserte Sichten werden typischerweise für sogenannte *Datenlager* (engl. *Data-Warehouses*) eingesetzt; zu ihrer Aktualisierung ist häufig ein erheblicher organisatorischer und systemtechnischer Aufwand vonnöten.
- Im Folgenden betrachten wir ausschließlich virtuelle Sichten.
- Eine interessante Frage ist, ob in einer virtuellen Sicht Einfügen, Löschen und Ändern von Zeilen erlaubt sein kann, oder nicht.

Ändern von Sichten

Sei $\mathcal{R}$ ein Datenbank-Schema.

- Eine *Datenbankänderung* ist eine Funktion t von der Menge der Instanzen zu $\mathcal{R}$ auf sich selbst. $t : \mathcal{I}^{\mathcal{R}} \to \mathcal{I}^{\mathcal{R}}$
- Eine *Sicht* ist eine Funktion f von der Menge aller Instanzen zu $\mathcal{R}$ in die Menge aller Instanzen zu S, wobei S das durch die Sicht definierte Relationsschema ist. $f : \mathcal{I}^{\mathcal{R}} \to \mathcal{I}^{S}$
- Eine *Sichtänderung* ist eine Funktion u von der Menge aller Instanzen zu S auf sich selbst. $u : \mathcal{I}^{S} \to \mathcal{I}^{S}$
- Sei u eine Sichtänderung und t eine Datenbankänderung, so dass für jede Datenbank-Instanz $\mathcal{I}^{\mathcal{R}}$ gilt:

$$u(f(\mathcal{I}^{\mathcal{R}})) = f(t(\mathcal{I}^{\mathcal{R}})),$$

  dann nennen wir t eine *Transformation* von u.
- Auf einer Sicht sind grundsätzlich nur solche Änderungen zulässig, zu denen eine Transformation existiert.

Beispiel

$$r = \begin{array}{cc} A & B \\ \hline a & b \\ x & b \end{array} \qquad s = \begin{array}{cc} B & C \\ \hline b & c \\ b & z \end{array} \qquad v = r \bowtie s = \begin{array}{ccc} A & B & C \\ \hline a & b & c \\ a & b & z \\ x & b & c \\ x & b & z \end{array}$$

INSERT (y,b,c) in v

Es existiert keine Transformation.

DELETE (a,b,c) in v

Es existiert keine Transformation.

UPDATE (a,b,c) zu (y,b,c) in v

Es existiert keine Transformation.

Projektionssicht

Einfügen, Löschen und Ändern ist nur dann erlaubt, wenn der Schlüssel der Basistabelle komplett in der Sicht vorhanden ist.

Informationen über Städte sind nur anonymisiert erlaubt.

```
CREATE VIEW StadtInfo AS
   SELECT PName, Einwohner
   FROM Stadt

INSERT INTO StadtInfo VALUES (Baden,90)
```

INSERT nicht zulässig!

Selektionssicht

- Aufgrund von Einfügungen und Änderungen können als unerwünschter Seiteneffekt Zeilen aus der Sicht herausfallen und damit in anderen Sichten erkennbar werden.
- Dieser Seiteneffekt kann durch Hinzunahme der Klausel `WITH CHECK OPTION` zu der Sichtdefinition verhindert werden.

Beschränkung auf Großstädte.

```
CREATE VIEW Großstadt AS
   SELECT *
   FROM Stadt
   WHERE Einwohner >= 1000

UPDATE Großstadt SET Einwohner = Einwohner * 0.9
```

Verbundsicht

In SQL-92 und in SQL:1999 werden eine Reihe von Regeln diskutiert, deren Erfülltsein die Existenz einer Transformation sichern. Diese Regeln garantieren im Wesentlichen ein eindeutiges Rückverfolgen der Sichtänderung zu einzelnen Zeilen in den Basistabellen.

Ordne jeder Stadt ihren Kontinent zu.

```
CREATE VIEW StadtInfo AS
   SELECT S.SName, L.Kontinent
   FROM Stadt S, Lage L
   WHERE S.LCode = L.LCode

INSERT INTO StadtInfo VALUES ('Freiburg', 'DreiLänderEck')
```

INSERT nicht zulässig!

empfohlene Lektüre

Efficiently Updating Materialized Views*

José A. Blakeley, Per-Åke Larson, Frank Wm. Tompa

Data Structuring Group,
Department of Computer Science,
University of Waterloo,
Waterloo, Ontario, N2L 3G1

Abstract

Query processing can be sped up by keeping frequently accessed users' views materialized. However, the need to access base relations in response to queries can be avoided only if the materialized view is adequately maintained. We propose a method in which all database updates to base relations are first filtered to remove from consideration those that cannot possibly affect the view. The conditions given for the detection of updates of this type, called *irrelevant updates*, are necessary and sufficient and are independent of the database state. For the remaining database updates, a *differential* algorithm can be applied to re-evaluate the view expression. The algorithm proposed exploits the knowledge provided by both the view definition expression and the database update operations.

rived relation–or *view*–is defined by a relational expression (i.e., a query evaluated over the base relations). A derived relation may be *virtual*, which corresponds to the traditional concept of a view; or *materialized*, which means that the resulting relation is actually stored. As the database changes because of updates applied to the base relations, the materialized views may also require change. A materialized view can always be brought up to date by re-evaluating the relational expression that defines it. However, complete re-evaluation is often wasteful, and the cost involved may be unacceptable.

The need for a mechanism to update materialized views efficiently has been expressed by several authors. Gardarin et al. [GSV84] consider *concrete views* (i.e., materialized views) as a candidate approach for the support of real time queries. However, they discard this approach because of the lack [1]

[1] In: Proceedings of the 1986 ACM SIGMOD International Conference on Management of Data.