4. Relationale Datenbanksprache SQL

1. Data Definition Language (DDL)
2. Data Manipulation Language (DML)
3. DQL – Data Query Language
4. DCL – Data Control Language
5. TCL – Transaction Control Language

Relationale Datenbanksprachen

Relationenalgebra und Relationenkalkül sind formale Spezialsprachen für Anfragen

SQL (Structured Query Language) als praxisnähere Sprache 1974 im IBM Almaden Research Centre entwickelt.

SQL ist Mischung von relationalen Algebra und relationalem Kalkül, plus Zusätze (z.B. arithmetische Operatoren oder Aggregatfunktionen).

SQL Tabellen sind Multimengen von Tupeln – d.h. diese können Duplikate enthalten

Teilsprachen, die wir behandeln werden:

- **Data Manipulation Language (DML)**: Einfügen, Ändern, Löschen und Anfragen von Daten
- **Data Definition Language (DDL)**: Schemadefinitionen
- **Data Query Language (DQL)**: Abfragen
- **Data Control Language (DCL)**: Rechtebehandlung
- **Transaction Control Language (TCL)**: Transaktionsbehandlung

SQL Teilsprachen Beispiel

Alle wesentlichen Teilsprachen in einem Beispiel

- DDL: CREATE TABLE country ...
- DML: INSERT INTO country VALUES ...
- DQL: SELECT ... FROM country LIMIT
- DCL: GRANT ...
- TCL: COMMIT

Wichtige SQL Standards

1986: SQL1 (ANSI Standard)

1992: SQL2 oder SQL-92 (ISO Standard) (wichtigster Standard!)

1999: SQL3 oder SQL:1999: Trigger, rekursive Abfragen, OO

2003: SQL:2003: Window functions, Sequences

2006: SQL/XML:2006: XML, XQuery

2008: SQL:2008 bzw. ISO/IEC 9075:2008: INSTEAD OF-Trigger, TRUNCATE-Statement und FETCH Klausel.

2011: SQL:2011 bzw. ISO/IEC 9075:2011: „Zeitbezogene Daten“ (PERIOD FOR)

2016: SQL:2016 bzw. ISO/IEC 9075:2016: JSON und „row pattern matching”

2019: SQL/MDA:2019. Datentyp „mehrdimensionales Feld“

Wichtiger Hinweis zu SQL für Übungsblätter und Klausuren

Wichtig:

- **Für die Übungsblätter und die Klausuren ist nur standard-konformes SQL zulässig, so wie es in dieser Vorlesung vorgestellt wird!**
- PostgreSQL ist standard-konform.
- **Nicht zulässig** sind SQL Erweiterungen wie z.B. in MySQL oder MS SQL, SAP, oder von anderen Herstellern.

SQL Beispiele

Für die Beispiele zu SQL wird das folgende Datenbankschema benutzt (Schlüssel sind unterstrichen):

- Kunde (KName, KAdr, Saldo)
- Lieferant (LName, Ware, LAdr, Preis)
- Auftrag (KName, LName, Ware, Menge)

SQL Wertebereiche

Die von SQL unterstützten Wertebereiche (domains) sind endliche Unterbereiche der ganzen und der reellen Zahlen

(`NUMBER`, `DECIMAL`, `INTEGER`, `SMALLINT`, `FLOAT`, `REAL`, `DOUBLE PRECISION`),

sowie Zeichenketten mit einer festen bzw. wählbaren, maximalen Länge

(`CHAR(n)` bzw. `VARCHAR(n)`), `DATE`, `BOOLEAN`, `CURRENCY`, ...

4. Relationale Datenbanksprache SQL

1. **Data Definition Language (DDL)**
2. Data Manipulation Language (DML)
3. DQL – Reporting und Änderungsoperationen
4. DCL – Data Control Language
5. TCL – Transaction Control Language

Die DQL von SQL

- Anfragen an die Datenbank und Operationen auf der Datenbank werden in der **Data Query Language (DQL)** formuliert.
- Grundform:

```
SELECT <Liste von Attributsnamen>
FROM <ein oder mehrere Relationennamen>
[WHERE <Bedingung>]
```

Projektion ⟸
Kreuzprodukt ⟸
Selektion ⟸

- Beispiel:

```
SELECT KName
FROM Kunde
WHERE Saldo < 0
```

- Die `SELECT`-Klausel entspricht der *Projektion* der relationalen Algebra.
- Die `FROM`-Klausel beschreibt das *Kreuzprodukt* der beteiligten Relationen.
- Die `WHERE`-Klausel enthält die Bedingung zur *Selektion.*

Eine kleine Beispieldatenbank für unsere DML-Beispiele

Kunde

KName	KAdr	Saldo
Müller	Melatenerstr. 32 Aachen	120.52
Russel	Jülicherstr. 61 Aachen	259.50
Kant	Aachenerstr. 5 Köln	651.14
Huber	Kölnerstr. 124 Düsseldorf	525.01

Auftrag

KName	LName	Ware	Menge
Russel	Mertens AG	Mehl	10
Huber	Glaser AG	Milch	5
Kant	Milan AG	Mehl	22
Müller	Milan AG	Milch	30
Russel	Trieste AG	Salz	15
Huber	Mertens AG	Mehl	3
Müller	Milan AG	Zucker	12

Lieferant

LName	Ware	LAdr	Preis
Mertens AG	Mehl	Saagengasse 13 Köln	1.97
Glaser AG	Milch	Kalossenweg 8 Köln	1.49
Müller AG	Salz	Hinterhausen 12 Köln	0.99
Milan AG	Milch	Grüner Weg 5 Köln	1.19
Trieste AG	Salz	Kirchenstrasse 7 Köln	0.53
Milan AG	Mehl	Grüner Weg 5 Köln	0.93
Milan AG	Zucker	Grüner Weg 5 Köln	0.69

Die DQL von SQL: WHERE-Klausel

Die Bedingung nach dem `WHERE` kann Vergleichsoperatoren

- {<, >, =, `BETWEEN`, `LIKE`, ...}, `IS NULL`,
- logische Verknüpfungen (`AND`, `OR`, `NOT`),
- Mengenoperatoren (`IN`, `NOT IN`, `ANY`, `ALL`, `EXISTS`) ⟸ Siehe später
- arithmetische Operatoren (+, –, *, /)

enthalten.

Durch die Punktnotation <Relation>.<Attribut> können Mehrdeutigkeiten vermieden werden.

Einige praktische Erweiterungen gegenüber Algebra/Kalkül

Arithmetische Ausdrücke in der `SELECT`-Klausel

- Beispiel: *Gib die Preise der Waren der einzelnen Lieferanten inkl. Mwst an.*

  `SELECT` LName, Ware, Preis * 1.19 `FROM` Lieferant

Suche nach Teilstrings

- Das SQL-Schlüsselzeichen ‘%’ repräsentiert einen beliebigen String.
- Beispiel: *Gibt es Kunden in Aachen?*

  `SELECT` * `FROM` Kunde `WHERE` KAdr `LIKE` ’%Aachen%’

Test auf Null-Wert: <Attributname> `IS [NOT] NULL`

- Beispiel: *Gib alle Kundennamen aus, deren Adresse nicht vorhanden ist.*

  `SELECT` KName `FROM` Kunde `WHERE` KAdr `IS NULL`
- Die Bedingung kann nicht als “`KAdr = NULL`” geschrieben werden.

SQL vs. Algebra/Kalkül: SELECT FROM Klausel

```
SELECT DISTINCT A, B, C, …
FROM R, S, T, …
WHERE Bedingung
```

Relationale Algebra:

Tupelkalkül:

- Die Tupelvariablen sind passend aus gewählt, sie sind nicht notwendig verschieden.
- Wenn ein Attribut (z.B. A) in zwei Relationen vorkommt (z.B. in R und S), dann muss man in SQL den Relationsnamen als „Tupelvariable“ angeben:

```
SELECT DISTINCT R.A, S.A, C, …  FROM R, S, T, …
WHERE Bedingung
```

SQL für Basisoperationen der relationalen Algebra

Im folgenden gelte *R*(A, B, C, D), *S*(E, F, G) und *T*(A, B, C, D)

- Vereinigung *RT*
```
SELECT * FROM R UNION SELECT * FROM T
```
- Differenz *R – T*
```
SELECT * FROM R EXCEPT SELECT * FROM T
```
- Kreuzprodukt *R T*
```
SELECT * FROM R CROSS JOIN T
```
- Selektion *(R)*
```
SELECT * FROM R WHERE B = 'b'
```
- Projektion *(R)*
```
SELECT DISTINCT A, C FROM R
```
- Umbenennung
```
SELECT V.A FROM R AS V
```
- Umbenennung
```
SELECT A, B, C AS K, D FROM R
```

SQL für weitere Operationen der relationalen Algebra

Im folgenden gelte *R*(A, B, C, D), *S*(E, F, G) und *T*(A, B, C, D), U (A, E)

- Relation *R*

```
SELECT * FROM R
```

- Wenn keine Duplikate auftreten, gilt auch (ansonsten mit `DISTINCT`):

(R) = `SELECT A, C FROM R`

- Theta-Join

= `SELECT * FROM R, S  WHERE` *B F*

- Natürlicher Verbund (Natural Join)

= `SELECT * FROM R NATURAL JOIN U`

SQL-Beispiel: Selektion

LName	Ware	LAdr	Preis
Mertens AG	Mehl	Saagengasse 13 Köln	1.97
Glaser AG	Milch	Kalossenweg 8 Köln	1.49
Müller AG	Salz	Hinterhausen 12 Köln	0.99
Milan AG	Milch	Grüner Weg 5 Köln	1.19
Trieste AG	Salz	Kirchenstrasse 7 Köln	0.53
Milan AG	Mehl	Grüner Weg 5 Köln	0.93
Milan AG	Zucker	Grüner Weg 5 Köln	0.69

Beispiel:

- Welche Lieferanten liefern Milch oder Mehl?
- SQL:

```
SELECT DISTINCT LName
FROM Lieferant
WHERE Ware = 'Mehl' OR Ware = 'Milch'
```

- Relationale Algebra:

- Tupelkalkül:

SQL-Beispiel mit natürlichem Join

Kunde (KName, KAdr, Saldo)
Lieferant (LName, Ware, LAdr, Preis)
Auftrag (KName, LName, Ware, Menge)

Beispiel:

- Welche Lieferanten liefern irgendetwas, das der Kunde Huber bestellt hat?
- SQL:

```
SELECT DISTINCT Lieferant.LName
FROM Lieferant
NATURAL JOIN Auftrag
WHERE Auftrag.KName = 'Huber'
```

- Relationale Algebra:

- Tupelkalkül:

Quantorensimulation in SQL: Geschachtelte Anfragen

- In WHERE-Klausel sind unterschiedliche Arten von Unterabfragen (Subqueries) möglich:
  - **`EXISTS` (<Subquery>):** Wahr, wenn Subquery mindestens ein Ergebnis liefert
  - **`X IN` (<Subquery>):** Wahr, wenn X in der Ergebnismenge der Subquery ist
  - **X <op> `ALL` (<Subquery>):** Wahr, wenn der Vergleichsoperator <op> für alle Resultate der Subquery erfüllt ist
  - **X <op> `ANY` (<Subquery>):** Wahr, wenn der Vergleichsoperator <op> für irgendein Resultat der Subquery erfüllt ist

Beispiel: Geschachtelte Anfrage

- Beispiel: *Welche Lieferanten liefern irgendetwas, das Huber bestellt hat?*

```
SELECT DISTINCT LName
FROM Lieferant
WHERE Ware IN
        (SELECT Ware
         FROM Auftrag
         WHERE KName = 'Huber')
```

LName	Ware	LAdr	Preis
Mertens AG	Mehl	Saagengasse 13 Köln	1.97
Glaser AG	Milch	Kalossenweg 8 Köln	1.49
Müller AG	Salz	Hinterhausen 12 Köln	0.99
Milan AG	Milch	Grüner Weg 5 Köln	1.19
Trieste AG	Salz	Kirchenstrasse 7 Köln	0.53
Milan AG	Mehl	Grüner Weg 5 Köln	0.93
Milan AG	Zucker	Grüner Weg 5 Köln	0.69

KName	KAdr	Saldo
Müller	Melatenerstr. 32 Aachen	120.52
Russel	Jülicherstr. 61 Aachen	259.50
Kant	Aachenerstr. 5 Köln	651.14
Huber	Kölnerstr. 124 Düsseldorf	525.01

KName	LName	Ware	Menge
Russel	Mertens AG	Mehl	10
Huber	Glaser AG	Milch	5
Kant	Milan AG	Mehl	22
Müller	Milan AG	Milch	30
Russel	Trieste AG	Salz	15
Huber	Mertens AG	Mehl	3
Müller	Milan AG	Zucker	12

- Beispiel:

```
SELECT * FROM Kunde WHERE Saldo >= ALL (SELECT Saldo FROM Kunde)
```

Beispiel: Tupelkalkül für geschachtelte Anfrage

- Beispiel: *Welche Lieferanten liefern alles, was Huber bestellt hat ?*

Tupelkalkül:

Aber: Allquantor ‘’ kann man in SQL so nicht verwenden!

⇒ Also müssen wir die Transformation durchführen:

Tupelkalkül:

Nach der Ersetzung der Implikation erhalten wir:

Tupelkalkül:

Kunde (KName, KAdr, Saldo)
Lieferant (LName, Ware, LAdr, Preis)
Auftrag (KName, LName, Ware, Menge)

Beispiel: SQL für geschachtelte Anfrage

(Beispiel weitergeführt)

- *Welche Lieferanten liefern alles, was Huber bestellt hat ?*

Tupelkalkül:

In SQL:

```
SELECT DISTINCT LName
FROM Lieferant L
WHERE NOT EXISTS
                (SELECT *
                FROM Auftrag
                WHERE KName = 'Huber' AND NOT Ware IN
                               (SELECT Ware
                               FROM Lieferant
                               WHERE LName = L.LName))
```

KName	LName	Ware	Menge
Russel	Mertens AG	Mehl	10
Huber	Glaser AG	Milch	5
Kant	Milan AG	Mehl	22
Müller	Milan AG	Milch	30
Russel	Trieste AG	Salz	15
Huber	Mertens AG	Mehl	3
Müller	Milan AG	Zucker	12

LName	Ware	LAdr	Preis
Mertens AG	Mehl	Saagengasse 13 Köln	1.97
Glaser AG	Milch	Kalossenweg 8 Köln	1.49
Müller AG	Salz	Hinterhausen 12 Köln	0.99
Milan AG	Milch	Grüner Weg 5 Köln	1.19
Trieste AG	Salz	Kirchenstrasse 7 Köln	0.53
Milan AG	Mehl	Grüner Weg 5 Köln	0.93
Milan AG	Zucker	Grüner Weg 5 Köln	0.69

4. Relationale Datenbanksprache SQL

1. Data Definition Language (DDL)
2. **Data Manipulation Language (DML) – einfache Anfragen**
3. DQL – Reporting und Änderungsoperationen
4. DCL – Data Control Language
5. TCL – Transaction Control Language

Die DDL von SQL: CREATE TABLE

- Das konzeptionelle Schema einer Datenbank wird durch die Data Definition Language (**DDL**) spezifiziert.

Relation anlegen

`CREATE TABLE` <Relationenname> (<Spaltendefinition>{, <Spaltendefinition>})
mit <Spaltendefinition> ::= <Attributname> <Typ> {<Option>}
und <Option> ::= `DEFAULT` <Ausdruck> | `NOT NULL` | `UNIQUE` | `PRIMARY KEY` | …

- Beispiel: `Kunde (KName, KAdr, Saldo)`

```
CREATE TABLE Kunde
( KName CHAR (20) NOT NULL,
  KAdresse VARCHAR (50),
  Saldo DECIMAL (7))
```

Keine NULL-Werte erlaubt

PRIMARY KEY

- Die `PRIMARY KEY` Einschränkung ist äquivalent zu „`NOT NULL UNIQUE`“
- Verschiedene Möglichkeiten zur Festlegung eines Primary Keys

```
CREATE TABLE Kunde
( KName CHAR (20) NOT NULL UNIQUE, KAdr VARCHAR (50),
  Saldo DECIMAL (7) ,
  PRIMARY KEY (KName)  )
```

oder

```
CREATE TABLE Kunde
(KName CHAR (20) NOT NULL UNIQUE PRIMARY KEY, KAdr VARCHAR (50),
 Saldo DECIMAL (7) )
```

oder

```
CREATE TABLE Kunde
( KName CHAR (20) NOT NULL UNIQUE, KAdr VARCHAR (50), Saldo DECIMAL (7),
  CONSTRAINT pk_kunde PRIMARY KEY (KName) )
```

FOREIGN KEY

Kunde (KName, KAdr, Saldo)
Lieferant (LName, Ware, LAdr, Preis)
Auftrag (KName, LName, Ware, Menge)

Lieferant-Tabelle:

```
CREATE TABLE Lieferant
( LName CHAR (20) NOT NULL, Ware  VARCHAR (20) NOT NULL,
  LAdr VARCHAR (50), Preis DECIMAL (7), PRIMARY KEY (LName, Ware) )
```

Auftrag-Tabelle:

```
            CREATE TABLE Auftrag
( KName CHAR (20) NOT NULL REFERENCES Kunde(KName),
        Lname CHAR(20) NOT NULL,
        Ware  VARCHAR (20) NOT NULL,
        Menge INTEGER,
        PRIMARY KEY (KName, Lname, Ware),
        FOREIGN KEY (Lname,Ware) REFERENCES Lieferant(Lname,Ware)  )
```

Wichtig: `FOREIGN KEY` impliziert **NICHT** `PRIMARY KEY`.

ALTER TABLE und DROP TABLE

Relation ändern

- Spalte hinzufügen, wobei `NOT NULL` nicht erlaubt ist
- Neues Attribut wird mit Nullwerten belegt.

  `ALTER TABLE` <Relationenname>

  `ADD` <Attributname> <Typ> [`DEFAULT` <Ausdruck>]

- Spalte entfernen

  `ALTER TABLE` <Relationen-Name>

  `DROP COLUMN` <Attributname>

Relation löschen

`DROP TABLE` <Relationenname>

Index anlegen

Index anlegen

- Indexe werden verwendet, um den Zugriff auf eine Relation zu beschleunigen.
- Dabei wird ein schneller Direktzugriff auf Tupel über indizierte Feldwerte ermöglicht.
- Je nach DBMS und Datentyp unterschiedliche Realisierung, Häufig durch B^+ Bäume realisiert (Details dazu in Kapitel 5 der Vorlesung).

```
    CREATE [UNIQUE] INDEX <Indexname>
ON <Relationenname> (<Attributname> [<Ordnung>]{, <Attributname> [<Ordnung>]})
    wobei  <Ordnung> ::= ASC | DESC
```

Index anlegen und löschen

Beispiel:

```
CREATE UNIQUE INDEX Kundenindex
ON Kunde (KName) [CLUSTER]
```

KName	KAdr	Saldo
Müller	Melatenerstr. 32 Aachen	120.52
Russel	Jülicherstr. 61 Aachen	259.50
Kant	Aachenerstr. 5 Köln	651.14
Huber	Kölnerstr. 124 Düsseldorf	525.01

`UNIQUE`: Für alle Attributsnamen keine zwei Tupel mit gleichen Werten erlaubt

➞ erfüllt Schlüsselbedingung.

`CLUSTER`: Die Speicher-Reihenfolge der Tupel der Relation entsprechen genau der Reihenfolge der Werte/Verweise im Index

➞ nur ein Clusterindex pro Relation.

Index löschen

```
DROP INDEX <Indexname>
```

Sichten (Views) anlegen und löschen

Sichten anlegen

- Sichten entsprechen externen DB-Schemata der Standardarchitektur. In relationalen Systemen werden Sichten als abgeleitete Relationen aufgefasst, die über Anfragen definiert sind.

  `CREATE VIEW` <Sichtname> [(<Attributname>{, <Attributname>})] `AS` <Subquery>

Beispiel:

```
CREATE VIEW  Gute_Kunden AS
        SELECT * FROM Kunde WHERE Saldo > 100
```

- Das SQL Schlüsselwort `*´ stellt eine Kurzschreibweise für die gesamte, konkatenierte Attributsliste der hinter dem FROM angegebenen Relationen dar.

Sichten löschen

```
DROP VIEW <Sicht-Name>
```

Sichten ausführen: Jede Anfrage auf eine Sicht führt diese auch aus.

4. Relationale Datenbanksprache SQL

1. Data Definition Language (DDL)
2. Data Manipulation Language (DML) – einfache Anfragen
3. **DQL – Reporting und Änderungsoperationen**
4. DCL – Data Control Language
5. TCL – Transaction Control Language

Aggregatfunktionen in SQL

- Die Funktionen `COUNT`, `MIN`, `MAX`, `SUM`, und `AVG` können auf eine Menge von Werten angewandt werden, die als Spalte einer Relation oder Teilrelation gegeben ist. Nur bei Verwendung von `DISTINCT` werden vorher Duplikate eliminiert (normalerweise ist das bei Statistik-Anwendungen nicht erwünscht).

- Zusätzlich werden `STDDEV` (Standardabweichung) und `VARIANCE` angeboten.

- Beispiel:

```
SELECT SUM ([DISTINCT] <Attributname>)
FROM ...WHERE ...
```

Beachte:

- `COUNT(*)` oder `COUNT`(<Attributname>) liefert die Anzahl der Tupel in einer Relation.

Aggregatfunktionen in SQL

- Nur um die Anzahl unterschiedlicher Attributwerte in einer Spalte (in Algebra: Größe der Projektion) zu bestimmen, muss `DISTINCT` verwendet werden.
- `NULL`-Werte werden bei allen Funktionen vorher aus der Argumentmenge eliminiert.
- Falls die Argumentmenge leer ist, liefert `COUNT` den Wert 0, die anderen Funktionen `NULL`.

Beispiele:

- *Wie viele Lieferanten gibt es?*

```
SELECT COUNT (DISTINCT LName)
FROM Lieferant
```

- *Wie viele Liter Milch sind insgesamt bestellt ?*

```
SELECT SUM (Menge)
FROM Auftrag
WHERE Ware = 'Milch'
```

LName	Ware	LAdr	Preis
Mertens AG	Mehl	Saagengasse 13 Köln	1.97
Glaser AG	Milch	Kalossenweg 8 Köln	1.49
Müller AG	Salz	Hinterhausen 12 Köln	0.99
Milan AG	Milch	Grüner Weg 5 Köln	1.19
Trieste AG	Salz	Kirchenstrasse 7 Köln	0.53
Milan AG	Mehl	Grüner Weg 5 Köln	0.93
Milan AG	Zucker	Grüner Weg 5 Köln	0.69

KName	LName	Ware	Menge
Russel	Mertens AG	Mehl	10
Huber	Glaser AG	Milch	5
Kant	Milan AG	Mehl	22
Müller	Milan AG	Milch	30
Russel	Trieste AG	Salz	15
Huber	Mertens AG	Mehl	3
Müller	Milan AG	Zucker	12

Reportgenerierung in SQL: Gruppieren und Sortieren

- Allgemeinere Form der `SELECT … FROM … WHERE-Klausel:`

  `SELECT … FROM … [JOIN …][WHERE …]`
  `[GROUP BY` <Liste von Attributsnamen>, `[HAVING` <Bedingung>`]]`
  `[ORDER BY` <Liste von Attributsnamen>`]`

`GROUP BY:`

- Mengen von Tupeln mit gleichen Werten der angegebenen Attribute werden zu Gruppen zusammengefasst.
- Die Ergebnisrelation enthält ein Tupel für jede Gruppe.
- Hinter `SELECT` nur Ausdrücke, die einen Wert pro Gruppe annehmen (Aggregatfunktionen oder Gruppenattribute).
- Die nach `GROUP BY` angegebenen Attribute müssen auch in `SELECT`-Klausel auftreten (gilt in neueren Versionen von PostgreSQL und MySQL nur für den Primärschlüssel)

Gruppieren und Sortieren

`HAVING`:

- Gruppen werden anhand der <Bedingung> ausgewählt.
- In der Bedingung dürfen nur Argumente mit einem Wert pro Gruppe auftreten.
- Unterschied `WHERE/HAVING`:
  - `WHERE` filtert auf Tupelebene
  - `HAVING` filtert auf Gruppenebene
    - nur Gruppierungsattribute oder aggregierte Attribute erlaubt!

`ORDER BY`:

- Die Ergebnisrelation wird nach einem oder mehreren Attributen sortiert (`DESC` | `ASC`).

Beispiel: GROUP BY und HAVING

Beispiel:

- *Gib die Namen aller Lieferanten aus, die mehr als zwei Teile liefern.*

```
SELECT LName
FROM Lieferant
GROUP BY LName
HAVING COUNT (*) > 2
```

LName	Ware	LAdr	Preis
Mertens AG	Mehl	Saagengasse 13 Köln	1.97
Glaser AG	Milch	Kalossenweg 8 Köln	1.49
Müller AG	Salz	Hinterhausen 12 Köln	0.99
Milan AG	Milch	Grüner Weg 5 Köln	1.19
Trieste AG	Salz	Kirchenstrasse 7 Köln	0.53
Milan AG	Mehl	Grüner Weg 5 Köln	0.93
Milan AG	Zucker	Grüner Weg 5 Köln	0.69

- '*' steht für alle Attribute einer Relation, auf die Bezug genommen wird.
- Im Beispiel bezieht es sich auf LName in Lieferant.
- `COUNT(*)` zählt also die verschiedenen Tupel.
- In der `HAVING`-Klausel wird die Bedingung für jede Gruppe unabhängig ausgewertet.

Beispiele: GROUP BY, ORDER BY, und HAVING

Beispiele:

- *Erstelle eine alphabetisch sortierte Liste aller Waren, in der für jede Ware der minimale, maximale und der Durchschnittspreis angegeben ist:*

```
SELECT Ware, MIN (Preis), MAX (Preis), AVG (Preis)
FROM Lieferant
GROUP BY Ware
ORDER BY Ware ASC
```

- *Welche Waren werden nur von einem Lieferanten geliefert?*

```
SELECT Ware FROM Lieferant
GROUP BY Ware
HAVING COUNT(*) = 1
```

LName	Ware	LAdr	Preis
Mertens AG	Mehl	Saagengasse 13 Köln	1.97
Glaser AG	Milch	Kalossenweg 8 Köln	1.49
Müller AG	Salz	Hinterhausen 12 Köln	0.99
Milan AG	Milch	Grüner Weg 5 Köln	1.19
Trieste AG	Salz	Kirchenstrasse 7 Köln	0.53
Milan AG	Mehl	Grüner Weg 5 Köln	0.93
Milan AG	Zucker	Grüner Weg 5 Köln	0.69

Beispiel: ORDER BY

Beispiel:

- *Sortiere die Bestellungen nach Waren, für jede Ware die Kunden absteigend nach der Größe der Bestellung:*

```
SELECT *  FROM Auftrag
ORDER BY Ware ASC, Menge DESC
```

KName	LName	Ware	Menge
Russel	Mertens AG	Mehl	10
Huber	Glaser AG	Milch	5
Kant	Milan AG	Mehl	22
Müller	Milan AG	Milch	30
Russel	Trieste AG	Salz	15
Huber	Mertens AG	Mehl	3
Müller	Milan AG	Zucker	12

Selfjoin

Tupelvariablen:

- Anfragen können sich auch auf zwei oder mehrere Tupel in einer Relation beziehen. Das entspricht einem *Selfjoin*, d.h. einem Join der Relation mit sich selbst.
- Um Selfjoins in SQL formulieren zu können, werden Tupelvariablen für die Relationen in der FROM-Klausel benannt, um die Partner des Selfjoins unterscheiden zu können.

Beispiel:

- *Gib Namen und Adressen aller Kunden aus, deren Kontostand kleiner als der von Huber ist.*

```
SELECT K1.KName, K1.KAdr
FROM Kunde K1, Kunde K2
WHERE K1.Saldo < K2.Saldo AND K2.KName = 'Huber'
```

- Bemerkung für Fortgeschrittene: Selfjoins spielen in *rekursiven* Anfragen eine große Rolle, wenn man etwa auf Basis einer „Eltern“-Relation sämtliche aus der Datenbank ableitbaren Vorfahren ermitteln möchte (*deduktive Datenbanken*)

Beispiel: Selfjoin

Beispiel:

- *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.LNa
```

Verhindert, dass Namenpaare im Ergebnis doppelt auftauchen

L1	L2
Mertens AG	Milan AG
Milan AG	Mertens AG
…	…

LName	Ware	LAdr	Preis
Mertens AG	Mehl	Saagengasse 13 Köln	1.97
Glaser AG	Milch	Kalossenweg 8 Köln	1.49
Müller AG	Salz	Hinterhausen 12 Köln	0.99
Milan AG	Milch	Grüner Weg 5 Köln	1.19
Trieste AG	Salz	Kirchenstrasse 7 Köln	0.53
Milan AG	Mehl	Grüner Weg 5 Köln	0.93
Milan AG	Zucker	Grüner Weg 5 Köln	0.69

Joins in SQL-Anfragen: Formulierungsvarianten

- Klassischer Stil (nicht zu empfehlen, da nicht sofort als Join zu erkennen):

```
SELECT K.KName, Ware
FROM Kunde K, Auftrag
WHERE K.KName = Auftrag.KName
```

- Seit SQL92 ("ANSI-Stil"): Natürlicher Verbund (implizit über gleiche Attribute)

```
SELECT KName, Ware
FROM Kunde NATURAL JOIN Auftrag
```

- Theta-Join mit expliziter Join-Bedingung: ON-Klausel

```
SELECT Kunde.KName, Ware
FROM Kunde JOIN Auftrag
ON Kunde.KName = Auftrag.KName
```

- Analog auch andere Join-Arten

```
LEFT (OUTER) JOIN
RIGHT (OUTER) JOIN
FULL (OUTER) JOIN
```

Kunde (KName, KAdr, Saldo)
Lieferant (LName, Ware, LAdr, Preis)
Auftrag (KName, LName, Ware, Menge)

SQL-Änderungsoperationen nutzen SQL-Anfragen, um mengenorientiert zu definieren, wo zu ändern ist

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>]

- Beispiel: *Füge den Kunden Schmidt mit dem Kontostand 0 EURO ein.*

```
INSERT INTO Kunde (KName, Saldo)
VALUES ('Schmidt', 0)
```

Beispiele für Änderungen

- Beispiel: *Huber bestellt von allem, was auch Müller bestellt hat, die doppelte Menge.*

```
INSERT INTO Auftrag
  SELECT 'Huber', LName, Ware, Menge*2
  FROM Auftrag
  WHERE KName = 'Müller'
```

- Beispiel: *Lösche alle Aufträge für Mehl.*

```
DELETE FROM Auftrag
WHERE Ware = 'Mehl'
```

- Beispiel: *Erhöhe den Kontostand von Kant um 200.*

```
UPDATE Kunde
SET Saldo = Saldo + 200
WHERE KName = 'Kant'
```

Zusammenfassung:
Anatomie und Formulierungsstrategie von SQL-Anfragen

Klausel	Reihenfolge	Semantik (was passiert?)
SELECT (DISTINCT)	5	**Projektion**: Übernehme nur die genannten Spalten, streiche die restlichen; wende Funktionen (sum, avg, ...) an Streiche Duplikate aus Ergebnis-Tabelle
FROM	1	Bilde das **kartesische Produkt** (... **,** ...) oder den **Join** (... JOIN ... ON ...) über die angegebenen Tabellen
WHERE	2	**Selektion**: streiche alle Tupel des Joins/kart. Produkts, die die WHERE-Bedingung nicht erfüllen
GROUP BY	3	**Gruppiere** Tupel der selektierten Eingabedatenkombinationen
HAVING	4	Streiche alle Tupel der **Gruppierung**, die die **HAVING-Bedingung** nicht erfüllen
ORDER BY	6	**Sortiere** das Ergebnis

4. Relationale Datenbanksprache SQL

1. Data Definition Language (DDL)
2. Data Manipulation Language (DML) – einfache Anfragen
3. DQL – Reporting und Änderungsoperationen
4. **DCL – Data Control Language**
5. TCL – Transaction Control Language

Festlegung der Zugriffsrechte

- SQLite hat keine GRANT und Revoke da es keine Client/Server Implementierung von SQL ist.
- Oracle dagegen hat als Beispiel-User einen User „Scott“ mit Passwort „Tiger“

4. Relationale Datenbanksprache SQL

1. Data Definition Language (DDL)
2. Data Manipulation Language (DML) – einfache Anfragen
3. DQL – Reporting und Änderungsoperationen
4. DCL – Data Control Language
5. **TCL – Transaction Control Language**

Beispiele für COMMIT and ROLLBACK

- Werden verwendet, um Modifikationen innerhalb einer Transaktion in die Datenbank zu übertragen