SQL – Structured Query Language

(Die Sprache der relationalen Datenbanken)

=

DML (Data Manipulation Language) + DDL (Data Definition Language)

- neue Daten einfügen
- bestehende Daten
  - **abfragen (SELECT)**
  - einfügen
  - ändern
  - löschen

- Tabellen
  - anlegen
  - Struktur ändern
  - löschen

Abitur-relevant

- SELECT (DISTINCT) …FROM, WHERE, GROUP BY, ORDER BY, ASC, DESC
- (LEFT/ RIGHT/ INNER) JOIN … ON, UNION, AS, NULL, =, <>, >, <, >=, <=, LIKE, BETWEEN, IN, IS NULL, +, -, *, /, AND, OR, NOT, COUNT, SUM, MAX, MIN, AVG

Die Grund-Abfrage-Befehle

Im Folgenden werden die grundlegenden Abfragebefehle in SQL erläutert. Dabei seien A_1, A_2, ..., A_n Attribute (= Spaltentitel), r_1, r_2, ..., r_m Relationen (= Tabellen) und P_1, P_2 Prädikate (= Bedingungen), die sich ggf. aus mehreren Einzelprädikaten und logischen Operatoren (and/ or/ …) zusammensetzen können. Zur Anschauung seien zunächst in Kurzform zwei mögliche Beispieltabellen abgebildet:

Personen (= r_1):

Name (= A_1)	Vorname (= A_2)	Gehalt (= A_3)	GebDatum (= A_4)
Taylor	Tim	2800	1960-01-01

Arbeitsplatz (= r_2):

Name (= A_1)	Firma (= A_2)
Turner	Harry's Eisenwaren

Eine grundlegende **Abfrage** hat dabei immer die Gestalt

SELECT [**DISTINCT**] A_1, A_2, ..., A_n
FROM r_1, r_2, ..., r_n
[**WHERE** P_1 | Mengenvergleich]
[**GROUP** BY A_1 [**ASC** | **DESC**]]
[**HAVING** P_1]
[**ORDER BY** A_1 [**ASC** | **DESC**]

„**SELECT** Spalten **FROM** Tabellen [**WHERE** …]“

„**wähle** Spalten **aus** Tabellen [**für die gilt** …]“

SELECT A_1, A_2, ..., A_n

wird verwendet, um festgesetzte Spalten ausgewählter Zeilen (, die gewissen Kriterien genügen) aus einer oder mehreren Tabellen abzurufen

[**DISTINCT**]

bewirkt, dass gleiche Datensätze nur einmal ausgegeben werden

FROM r_1, r_2, ..., r_m

listet die notwendigen Tabellen für die Abfrage auf – wird mehr als eine Tabelle angegeben, entspricht die Funktionsweise dem Kartesischen Produkt

[**WHERE** P_1]

definiert die Auswahlbedingung, beispielsweise **WHERE** Name = "Taylor" **AND** Vorname = "Tim" oder bei einem Mengenvergleich **WHERE** Gehalt >= 5000

[**GROUP BY** r_i] [**ASC** | **DESC**]

gruppiert die Ausgabe nach den angegebenen Kriterien, hier r_i; in der Regel erfolgt die Gruppierung nach Feldnamen; mit **DESC** wird eine absteigende, mit **ASC** eine aufsteigende Sortierung erreicht

[**HAVING** P_2]

wird nur in Kombination mit **GROUP BY** verwendet und hat dann dieselbe Funktion wie die **WHERE-**Abfrage

[**ORDER BY** A_j] [**ASC** | **DESC**]

sortiert die Ausgabe nach den angegebenen Kriterien, hier A_j; mit **DESC** wird eine absteigende, mit **ASC** eine aufsteigende Sortierung erreicht

Beispiel:

```
SELECT Firma, Gehalt
FROM Personen, Arbeitsplatz
WHERE Person.Name = Arbeitsplatz.Name
GROUP BY Firma
HAVING Gehalt > 10000
```

Beachte: Es sind auch geschachtelte Abfragen möglich, z.B. der Form

```
SELECT *
FROM (
        SELECT …
        FROM …
) …
WHERE …
```

Weitere Befehle

[Aggregatsfunktionen()]

zur Verfügung stehen **avg**(A_i), **min**(A_i), **max**(Ai), **sum**(A_i) und **count**(A_i)

Beispiel: **SELECT avg**(Gehalt) **FROM** Personen

[AS]

mit dem **AS** – Operator kann man einzelne Tabellen oder Spalten umbenennen

[UNION]

verwendet man um Daten zu vereinigen; Voraussetzung ist, dass die Anzahl der Attribute beider Relationen gleich ist und die Wertebereiche der einander entsprechenden Attribute kompatibel sind; Duplikate werden entfernt

Beispiel:

SELECT * **FROM** Personen **WHERE** Gehalt > 2000
UNION SELECT * **FROM** Personen **WHERE** Gehalt < 1000

Der Join von Relationen

Das Kartesische Produkt (Symbol „×“ = „**JOIN**“ = „**CROSS JOIN**“ – je nach Literatur!) ermöglicht die Kombination der Einträge zweier beteiligter Relationen (Tabellen), indem es jedes Tupel (jede Zeile) der ersten Menge mit jedem Tupel der zweiten Menge verbindet. Wird ein Attributname in beiden Relationen verwendet, so wird zum Zweck der Unterscheidung in der Ergebnisrelation der Attributname mit dem der Relation verknüpft, aus der das Attribut ursprünglich stammt. Für Attribute, die nur in einer Relation vorkommen, verzichtet man auf das Relationennamen-Prefix.

Beispiel:

Relation r

A	B
α	1
β	2

Relation s

A	C	D
α	10	+
β	10	+
β	20	-
γ	10	-

$r \times s$

r.A	B	s.A	C	D
α	1	α	10	+
α	1	β	10	+
α	1	β	20	-
α	1	γ	10	-
β	2	α	10	+
β	2	β	10	+
β	2	β	20	-
β	2	γ	10	-

Der **natürliche Join** (= „**NATURAL JOIN**“) zweier Relationen r und s ist ein spezieller Join und entspricht einer Hintereinanderausführung mehrerer Operationen: Zunächst wird das Kartesische Produkt (= JOIN) aus beiden Mengen gebildet, anschließend diejenigen Tupel ermittelt, die für die in beiden Relationen vorkommenden Attribute die gleichen Werte haben und abschließend alle Duplikate entfernt.

Der **NATURAL JOIN** Kann nur im **FROM** - Teil eingesetzt werden. Die Spalten, bezüglichen derer gejoint wird, bleiben erhalten und werden entsprechend der Relation umbenannt.

Beispiel:

Relation r

A	B	C	D
α	1	α	a
β	2	γ	a
γ	4	β	b
α	1	γ	a
δ	2	β	b

Relation s

B	D	E
1	a	α
3	a	β
1	a	γ
2	b	δ
5	b	ϵ

$r \bowtie s$

A	B	C	D	E
α	1	α	a	α
α	1	α	a	γ
α	1	γ	a	α
α	1	γ	a	γ
δ	2	β	b	δ

Um einen Informationsverlust, wie er beim natürlichen Join auftreten kann, zu vermeiden, existieren so genannte **Outer Join**-Operationen.

Hierbei wird zunächst der **natürliche Join** der beiden an der Operation beteiligten Relationen gebildet und dem Ergebnis anschließend die Tupel einer Relation oder beider Relationen hinzugefügt, die zu keinem Tupel der anderen Relation passen. Nicht vorhandene Informationen werden dabei durch sogenannte ***null***-Werte dargestellt.

Beim **Left-Outer-Join** (= „**LEFT JOIN**“) wird die linke Relation vollständig berücksichtigt, während die fehlenden Werte der rechten Menge entsprechend durch ***null***-Werte ergänzt werden. Analog ist der **Right-Outer-Join** (= „**RIGHT JOIN**“) definiert, während beim **Full-Outer-Join** in beiden beteiligten Relationen nicht vorhandene Einträge durch ***null*** gefüllt werden.
Das Schema der Ergebnisrelation der Outer Join-Operationen entspricht dem des natürlichen Join.

Der **LEFT/ RIGHT [OUTER] JOIN** Kann ebenfalls nur im **FROM** – Teil eingesetzt werden und verknüpft die beteiligten Relationen bezüglich des angegebenen Prädikats. Fehlende Werte werden durch ***null***-Werte ergänzt. Der Vergleich auf null kann nicht durch „=“ erfolgen. Stattdessen ist der Operator **IS** zu verwenden.

Beispiel:

Relation *Schuldner*

KuSozVersNr	KNummer
TT-1960-01-01	kdt0815
WW-1957-04-04	kdt4711
BA-1965-02-02	kdt1234

Relation *Kredit*

KNummer	Betrag
kdt0815	2000
kdt4711	3000
kdt9876	4000

JOIN

Schuldner ⋈ *Kredit*

KuSozVersNr	KNummer	Betrag
TT-1960-01-01	kdt0815	2000
WW-1957-04-04	kdt4711	3000

LEFT JOIN

Schuldner ⟕ *Kredit*

KuSozVersNr	KNummer	Betrag
TT-1960-01-01	kdt0815	2000
WW-1957-04-04	kdt4711	3000
BA-1965-02-02	kdt1234	*null*

RIGHT JOIN

Schuldner ⟖ *Kredit*

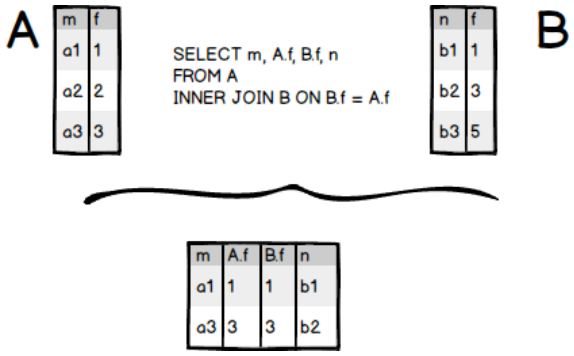
KuSozVersNr	KNummer	Betrag
TT-1960-01-01	kdt0815	2000
WW-1957-04-04	kdt4711	3000
null	kdt9876	4000

Schuldner ⟗ *Kredit*

KuSozVersNr	KNummer	Betrag
TT-1960-01-01	kdt0815	2000
WW-1957-04-04	kdt4711	3000
BA-1965-02-02	kdt1234	*null*
null	kdt9876	4000

FULL OUTER JOIN

Der so genannte **INNER JOIN** (… **ON** Kriterien) führt Datensätze aus zwei beteiligten Relationen genau dann zusammen, wenn die angegebenen Kriterien alle erfüllt sind. Ist eines oder mehrere der Kriterien nicht erfüllt, so entsteht kein Datensatz in der Ergebnismenge. Durch den Einsatz dieses JOINS reduziert sich das Ergebnis des **JOINS** auf ein Minimum.



Zusatz – Modifikationen (nicht Abitur-relevant)

Einfügen am Beispiel:

INSERT INTO Personen
VALUES („Borland“, „Al“, 1800, 1965-02-02)

Löschen am Beispiel:

DELETE FROM Personen
WHERE Gehalt **BETWEEN** 500 **AND** 1300

Ändern am Beispiel:

UPDATE Personen
SET Gehalt = Gehalt * 1,05
WHERE Gehalt <= 1000