HTW Chur
Swiss Institute for
Information Research

Structured Query Language (SQL)

Albert Weichselbraun <albert.weichselbraun@htwchur.ch>

Agenda

1. Einleitung
2. Data Definition Language (DDL)
3. Data Manipulation Language (DML)
   - Abfragen in SQL (SELECT)
   - Mutationen in SQL (INSERT, UPDATE, DELETE)
4. Data Control Language (DCL)
5. Indices und Prozedurale Integritätsregeln

Structured Query Language (SQL)

- Für relationale Datenbanken hat sich der internationale Standard **SQL** (Structured Query Language) durchgesetzt.
- Entwicklungsziele
  - möglichst einfach für den Endbenutzer;
  - möglichst **deskriptiv**, nicht prozedural
- Entwicklung: (im IBM-Labor)
  - basiert auf relationaler Algebra und Tupelkalkül
  - erste Vorschläge 1974; weiterentwickelt 1976-77
  - Implementierungen 1977 im System R (Prototyp eines rel. DBS)
- Erste kommerzielle Implementierungen:
  - Oracle: 1980
  - IBM: SQL/DS (1981), DB2 (1983)

Structured Query Language (SQL)

Hinweise

- SQL ist case-insensitive (Gross-/Kleinschreibung ist irrelevant)
  Achtung: manche Datenbanken (MySQL) unterscheiden Gross-/Kleinschreibung bei Tabellennamen
- Namen, die Sonderzeichen enthalten, müssen unter doppelte Anführungszeichen gesetzt werden.
  Beispiele:
  - Ang-pro → “ang-pro“ oder ANGPRO
  - ANG-NR → “ANG-NR“ oder “AngNr“
- Abhängig von der Datenbank kann es auch notwendig sein, Namen welche Grossbuchstaben enthalten unter Anführungszeichen zu setzen.
- Vermeiden Sie Sonderzeichen und Umlaute in Tabellen- und Attributnamen.
- Zeichenketten (Strings) werden unter einfache Anführungszeichen gesetzt.

```
SELECT name FROM angestellte WHERE wohnort = ‘Chur‘;
```

Data Definition Language

Definition von Tabellenschemata (SQL-DDL)

Datenbank : Menge von Basistabellen
Benutzersichten : Views (Virtuelle Tabellen)

Übersicht: SQL-DDL (vereinfacht)

Befehl	**Bedeutung**
CREATE TABLE	Erstellt eine neue Tabelle
DROP TABLE	Löscht eine Tabelle
ALTER TABLE	Modifiziert eine Tabelle
CREATE / DROP / ALTER VIEW	Erstellt / Löscht / Modifiziert einen View

Definition von Tabellenschemata (SQL-DDL)

Definition einer Relation

```
CREATE TABLE base-table-name (
        {column-definition | table-constraint}
       [, {column-definition | table-constr.}]…
);
```

Spaltendefinition (vereinfacht)

- column-definition:
  column-name data-type [column-constraint …]
- column-constraint:
  NOT NULL | UNIQUE | PRIMARY KEY | reference-constraint
- reference-constraint:
  REFERENCES table-name [(reference-column)]

Definition von Tabellenschemata (SQL-DDL)

Constraints über mehrere Attribute: (table-constraint)

- UNIQUE (column-list)
- PRIMARY KEY (column-list)
- FOREIGN KEY (referencing-columns)
  REFERENCES table-name [(column-list)]

Definition von Tabellenschemata (SQL-DDL)

SQL Datentypen: (Auswahl)

Datentyp	Beschreibung
Zeichenketten	
CHAR	Ein Zeichen
CHAR(n)	Zeichenkette fester Länge (z.B. Signatur)
VARCHAR(n), TEXT	Zeichenkette variabler Länge (z.B. Name)
Zahlen	
DECIMAL	Dezimalzahl
INTEGER	Ganze Zahl
FLOAT	Gleitkommazahl

Definition von Tabellenschemata (SQL-DDL)

SQL Datentypen: (Auswahl)

Datum und Zeit	
DATE	Datum
TIME	Zeit
TIMESTAMP	Zeitpunkt
INTERVAL	Zeitintervall

Constraints

mittels der CHECK Anweisung kann man Einschränkungen für den Datentypen definieren:

```
ProzArbeitsZeit DECIMAL CHECK
                    (ProzArbeitsZeit >=0.0 AND ProzArbeitsZeit <=100.0)
```

Definition von Tabellenschemata (SQL-DDL)

Beispiel

Erstellen Sie die folgende Relation mittels SQL.

angestellte (AngNr, Name, Wohnort, AbtNr | AngNr = PS)

```
CREATE TABLE  angestellte
    ("AngNr"      INTEGER PRIMARY KEY,
     "Name"       VARCHAR (30),
     "Wohnort"    VARCHAR (30),
     "AbtNr"      INTEGER);
```

Definition von Tabellenschemata (SQL-DDL)

Beispiel

Erstellen Sie die folgende Relation mittels SQL.

projekt (PNr, PName, PFiliale, PLeiter | PNr = PS,
projekt.PLeiter ⊆ angestellte.AngNr)

```
CREATE TABLE   projekt
    ("PNr "          INTEGER PRIMARY KEY,
     "PNname"        VARCHAR (15),
     "Pfiliale"      VARCHAR (30),
     "Pleiter"       INTEGER REFERENCES angestellte("AngNr"));
```

Definition von Tabellenschemata (SQL-DDL)

Beispiel

Erstellen Sie die folgende Relation mittels SQL.

angpro (AngNr, PNr, ProzArbZeit | {AngNr, PNr} = PS,
angpro.AngNr ⊆ angestellte.AngNr
angpro.PNr ⊆ projekt.PNr)

```
CREATE TABLE   angpro
    ("PNr"              INTEGER REFERENCES projekt("PNr"),
     "AngNr"            INTEGER REFERENCES angestellte("AngNr"),
     "ProzArbZeit"      DECIMAL CHECK ("ProzArbZeit" >=0.0 AND
                        "ProzArbZeit" <=100.0),
     PRIMARY KEY ("PNr", "AngNr"));
```

Definition von Tabellenschemata (SQL-DDL)

Definition von Sichten (vereinfacht)

CREATE VIEW name AS query-expression;

query-expression … Eine beliebige SQL Abfrage (siehe SQL-DML)

Beispiel

```
CREATE VIEW AngestellteAusChur
      AS  SELECT "AngNr", "Name", "Wohnort", "AbtNr"
            FROM  angestellte
            WHERE "Wohnort" = ‘Chur‘;
```

Exkurs: Installation von PostgreSQL

Exkurs: Installation von PostgreSQL

- **Linux (Ubuntu)**

  apt-get install postgresql pgadmin3

- **MacOS / Windows:**
  - Download Site:
    www.enterprisedb.com/products-services-training/pgdownload
  - Auswahl der korrekten Software Version
  - Manuelle Installation
    nach erfolgter Installation von PostgreSQL wird Ihnen angeboten zusätzlich die StackBuilder Software zu installieren. Lehnen Sie dies bitte ab.

Wichtig: Bitte merken Sie sich Ihr PostgreSQL Admin Passwort!

Exkurs: Installation von PostgreSQL

Download der Testdaten

Laden Sie die Projektdatenbank von

https://weichselbraun.net/dbs/projektdatenbank.sql

und speichern Sie diese auf dem Desktop.

Installation der Testdaten

1. Stellen Sie eine Verbindung zum Datenbankserver her.
2. Erstellen Sie eine neue Datenbank mit dem Namen „student“.
3. Markieren Sie die „student“ Datenbank und starten Sie den SQL Editor.
4. Öffnen Sie die Datei mit der Projektdatenbank (projektdatenbank.sql) und führen Sie diese aus.

Data Manipulation Language

Mutationen in SQL (SQL-DML)

Einfügen von Tupeln (INSERT)

INSERT INTO table [(column [,column] …)] **VALUES** (constant [,constant] …);

oder

INSERT INTO table [(column [,column] …)] query;

Beispiele

1. **INSERT INTO** angestellte ("AngNr", "NAME", "WOHNORT", "AbtNr")
   VALUES (1717, ’Häberle’,’Stuttgart’, 30);
2. **INSERT INTO** angestellte_in_chur
   SELECT * **FROM** angestellte **WHERE** "WOHNORT" **=** ‘Chur‘;

Mutationen in SQL (SQL-DML)

Löschen von Tupeln (DELETE)

DELETE FROM table [WHERE search condition];

Beispiele

1. DELETE FROM "ang-pro" WHERE "AngNr" = 3190;
2. DELETE FROM projekt;

Mutationen in SQL (SQL-DML)

Modifikation von Tupeln (UPDATE)

UPDATE table
    SET column = expression [,column = expression] …
    [WHERE search condition];

Beispiele

1. UPDATE projekt SET "PLeiter" = 3207, "PFiliale" = ’Mannheim’
       WHERE "PNr" = 761235;
2. UPDATE angestellte SET "AbtNr" = 12;

Einfache Abfragen in SQL (SQL-DML)

Tabellen zu den Beispielen (siehe Testdatenbank)

angestellte (#=n)

AngNr	NAME	WOHNORT	AbtNr
3115	Meyer	Karlsruhe	35
3207	Müller	Mannheim	30
2814	Klein	Mannheim	32
3190	Maus	Karlsruhe	30
2314	Groß	Karlsruhe	35
1324	Schmitt	Heidelberg	35
1435	Mayerlein	Bruchsal	32
2412	Müller	Karlsruhe	32
2244	Schulz	Bruchsal	31
1237	Krämer	Ludwigshafen	31
3425	Meier	Pforzheim	30
2454	Schuster	Worms	31

projekt (#=k)

PName	PNr	PFiliale	PLeiter
p-1	761235	Karlsruhe	3115
p-2	770008	Karlsruhe	3115
p-3	770114	Heidelberg	1324
P-4	770231	Mannheim	2814

Einfache Abfragen in SQL (SQL-DML)

ang-pro (#=m)

PNr	AngNr	ProzArbzeit
761235	3207	100
761235	3115	50
761235	3190	50
761235	1435	40
761235	3425	50
770008	2244	20
770008	1237	40
770008	2814	70
770008	2454	40
770114	2814	30
770114	1435	60
770114	1237	60
770114	2454	60
770114	3425	50
770114	2412	100
770231	3190	50
770231	2314	100
770231	2244	80
770231	3115	50
770231	1324	100

Einfache Abfragen in SQL (SQL-DML)

Die SELECT-Abfrage

Vereinfachte Grundform:

SELECT	…	Auswahl von Spalten (π)
FROM	…	Relation(en)/Tabelle(n)
WHERE	…	Selektionsbedingung (σ)

Beispiel:

$\pi_{[angnr,\ name]}\ \sigma_{[wohnort = ‚Mannheim‘]}$ angestellte

```
SELECT angnr, name FROM angestellte WHERE wohnort='Mannheim';
```

Einfache Abfragen in SQL (SQL-DML)

Grundform:

```
SELECT [DISTINCT] derived column(s)
    FROM table(s)
    [WHERE search condition]
    [GROUP BY column(s)
        [HAVING search condition]]
    [ORDER BY column(s)];
```

Mengenoperationen:

SELECT-Abfrage *op* SELECT-Abfrage

op := {EXCEPT | INTERSECT | UNION} [ALL]

Einfache Abfragen in SQL (SQL-DML)

WHERE <search condition>

Die WHERE Bedingung bezieht sich immer auf einzelne Zeilen und entspricht somit der Selektion (σ).

HAVING <search condition>

HAVING bezieht sich immer auf aggregierte Ergebnisse und kann somit in der Relationalen Algebra nicht abgebildet werden.

Die search condition ist ein logischer Ausdruck, der folgendes enthalten kann:

- Vergleichsoperationen: =, <, >, !=, >=, <=, BETWEEN, LIKE, IN
- Boolesche Operatoren: AND, OR, NOT

Einfache Abfragen in SQL (SQL-DML)

Abfragen mit Bedingungen

1. „*Name und Abteilungsnummer aller Angestellten mit Wohnort Karlsruhe*“

```
SELECT "NAME", "AngNr"
FROM   angestellte
WHERE  "WOHNORT" = ‘Karlsruhe‘;
```

angestellte

AngNr	NAME	WOHNORT	AbtNr
3115	Meyer	Karlsruhe	35
3207	Müller	Mannheim	30
2814	Klein	Mannheim	32
3190	Maus	Karlsruhe	30
2314	Groß	Karlsruhe	35
1324	Schmitt	Heidelberg	35
1435	Mayerlein	Bruchsal	32
2412	Müller	Karlsruhe	32
2244	Schulz	Bruchsal	31
1237	Krämer	Ludwigshafen	31
3425	Meier	Pforzheim	30
2454	Schuster	Worms	31

Einfache Abfragen in SQL (SQL-DML)

2. „*Name und Abteilungsnummer aller Angestellten, die in Karlsruhe wohnen oder in Abteilung 30 arbeiten*“

SELECT "NAME", "AngNr"
FROM angestellte **WHERE** "WOHNORT"**=**‘Karlsruhe’ **OR** "AbtNr"**=**30;

angestellte

AngNr	NAME	WOHNORT	AbtNr
3115	Meyer	Karlsruhe	35
3207	Müller	Mannheim	30
2814	Klein	Mannheim	32
3190	Maus	Karlsruhe	30
2314	Groß	Karlsruhe	35
1324	Schmitt	Heidelberg	35
1435	Mayerlein	Bruchsal	32
2412	Müller	Karlsruhe	32
2244	Schulz	Bruchsal	31
1237	Krämer	Ludwigshafen	31
3425	Meier	Pforzheim	30
2454	Schuster	Worms	31

Einfache Abfragen in SQL (SQL-DML)

Abfragen mit Bereichsgrenzen

„Alle Angestellten (Nummer und Name) mit Nummer zwischen 1435 und 2314 (jeweils einschließlich)“

„BETWEEN a AND b“ entspricht „x >= a AND x <= b“

```
SELECT "NAME", "AngNr"
FROM   angestellte
WHERE  "AngNr" BETWEEN 1435 AND 2314;
```

Einfache Abfragen in SQL (SQL-DML)

Abfragen mit Wertaufzählung

4. „Alle Angestellten-Nummern der Abteilungen 30 und 35 aus Karlsruhe und Mannheim“

```
SELECT "NAME", "AngNr"
FROM   angestellte
WHERE  "AbtNr" IN (30, 35) AND
       "WOHNORT" IN ('Karlsruhe', 'Mannheim');
```

Einfache Abfragen in SQL (SQL-DML)

Abfrage mit Teilstring-Suche (Wildcards)

Das Zeichen „**%**“ im Suchstring steht für eine beliebige Folge von Zeichen mit einer Länge von 0 bis ∞, passt also auf jeden String.

Das Muster ‘%ei%‘ passt zum Beispiel auf die Strings ‘**ei**‘, ‘m**ei**n‘, ‘**ei**n‘, ‘b**ei**‘, und ‘Kl**ei**ngartensiedlung‘.

„Nummer und PFiliale aller Projekte, deren P-Filialen in Städten liegen, deren Namen ein „ei“ enthalten.“

```
SELECT "PNr", "PFiliale"
FROM   projekt
WHERE  "PFiliale" LIKE ‘%ei%’;
```

Einfache Abfragen in SQL (SQL-DML)

Der Unterstrich "_" im Suchstring steht für genau ein beliebiges Zeichen.

Das Muster ‘_ei_‘ passt zum Beispiel auf die Strings ‘m**ei**n‘ und ‘k**ei**n‘ nicht aber auf

- ‘b**ei**‘ (fehlendes Zeichen am Ende),
- ‘ei‘ (fehlendes Zeichen am Anfang und Ende) oder
- ‘klein‘ (mehr als ein Zeichen am Anfang).

6. „Nummer und Name aller Angestellten, deren Namen mit ‘Me’ beginnen, mit ‘er’ aufhören und dazwischen nur einen einzelnen Buchstaben beinhalten.“

```
SELECT "NAME", "AngNr"
FROM   angestellte
WHERE  "NAME" LIKE ‘Me_er’;
```

Einfache Abfragen in SQL (SQL-DML)

7. „Alle Angestellten-Namen, deren Anfang wie 'Maier' klingt."

SELECT "NAME", "AngNr" **FROM** angestellte
WHERE "NAME" **LIKE** 'M_ _er%';

angestellte

AngNr	NAME	WOHNORT	AbtNr
3115	Meyer	Karlsruhe	35
3207	Müller	Mannheim	30
2814	Klein	Mannheim	32
3190	Maus	Karlsruhe	30
2314	Groß	Karlsruhe	35
1324	Schmitt	Heidelberg	35
1435	Mayerlein	Bruchsal	32
2412	Müller	Karlsruhe	32
2244	Schulz	Bruchsal	31
1237	Krämer	Ludwigshafen	31
3425	Meier	Pforzheim	30
2454	Schuster	Worms	31

Einfache Abfragen in SQL (SQL-DML)

Retrieval mit Sortierung

- auf / absteigend ([ASC] / DESC)
- nach Wert einer Spalte oder mehrerer Spalten (zusammengesetztes Sortierkriterium, ASC - DESC beliebig mischbar)

8. „*Nummer und Name aller Angestellten, die zur Abteilung 30 gehören, aufsteigend sortiert nach Name und Wohnort.*“

SELECT "NAME", "AngNr" **FROM** angestellte **WHERE** "AbtNr" = 30
ORDER BY "NAME", "WOHNORT" **ASC;**

AngNr	NAME	WOHNORT	AbtNr
3207	Müller	Mannheim	30
3190	Maus	Karlsruhe	30
3425	Meier	Pforzheim	30

AngNr	NAME
3190	Maus
3425	Meier
3207	Müller

Einfache Abfragen in SQL (SQL-DML)

Spezielle Abfragen

1. Selektion aller Zeilen – „Name aller Projekte“
   SELECT "PName" **FROM** projekt

2. Selektion aller Spalten – „alle Projekte in Karlsruhe“
   SELECT * FROM "Projekt" **WHERE** "Pfiliale" = ‘Karlsruhe‘;

3. Ausgabe der gesamten Relation
   SELECT * **FROM** "Projekt";

Einfache Abfragen in SQL (SQL-DML)

Elimination von Duplikaten:

„Die Orte aller P-Filialen ohne Duplikate“

SELECT DISTINCT "PFiliale" **FROM** projekt;

projekt

PName	PNr	PFiliale	PLeiter
p-1	761235	Karlsruhe	3115
p-2	770008	Karlsruhe	3115
p-3	770114	Heidelberg	1324
P-4	770231	Mannheim	2814

PFiliale
Karlsruhe
Heidelberg
Mannheim

Einfache Abfragen in SQL (SQL-DML)

Rechnen mit Attributwerten:

„(Zusammengehörige) Projekt- und Angestelltennummern mit entsprechenden Projektkosten. Eine 100% Stelle verursacht Kosten von 8‘500 CHF.

SELECT "PNr", "AngNr", "ProzArbzeit" * 8500 * 0.01
FROM "ang-pro";

ang-pro

PNr	AngNr	ProzArbzeit * 8500 * 0.01
761235	3207	8500.00
761235	3115	4250.00
761235	3190	4250.00
761235	1435	3400.00
761235	3425	4250.00
770008	2244	1700.00
...	...	...

Aggregatfunktionen in SQL (SQL-DML)

Aggregatfunktionen fassen mehrere Zeilen zu einem neuen Ergebnis zusammen.

Es wird immer **zuerst das ursprüngliche Query** (bis inklusive Selektion) ausgeführt und erst **anschliessend** der Wert der Aggregation berechnet.

Funktion	Bedeutung
COUNT	Anzahl der Werte in Spalte (Name) bzw. Tabelle (*)
SUM	Summe der Werte in Spalte
AVG	Durchschnitt der Werte in Spalte
MAX	Größter Wert in Spalte
MIN	Kleinster Wert in Spalte

Beispiele:

```
SELECT COUNT(DISTINCT "Name") FROM angestellte;
SELECT SUM(gehalt) FROM angestellte WHERE abt=30;
```

Aggregatfunktionen in SQL (SQL-DML)

„Höchster vorkommender Arbeitszeitanteil“

```
SELECT MAX("ProzArbzeit") AS "ProzArbzeitMax"
FROM "ang-pro";
```

ang-pro

PNr	AngNr	ProzArbzeit
761235	3207	100
761235	3115	50
761235	3190	50
761235	1435	40
761235	3425	50
770008	2244	20
770008	1237	40
770008	2814	70
770008	2454	40
770114	2814	30
770114	1435	60
770114	1237	60
770114	2454	60
770114	3425	50
770114	2412	100
770231	3190	50
770231	2314	100
770231	2244	80
770231	3115	50
770231	1324	100

ProzArbzeitMax
100

Aggregatfunktionen in SQL (SQL-DML)

„Durchschnittlicher Arbeitszeitanteil am Projekt Nr. 761235“

SELECT AVG("ProzArbzeit") **AS** "PROZ-ARBEIT-AVG-761235"
FROM "ang-pro" **WHERE** "PNr" **=** 761235;

ang-pro

PNr	AngNr	ProzArbzeit
761235	3207	100
761235	3115	50
761235	3190	50
761235	1435	40
761235	3425	50
770008	2244	20
770008	1237	40
770008	2814	70
770008	2454	40
770114	2814	30
770114	1435	60
770114	1237	60
770114	2454	60
770114	3425	50
770114	2412	100
…	…	…

PROZ-ARBEIT-AVG-761235
58

Aggregatfunktionen in SQL (SQL-DML)

Bildung von Gruppen

Zusammenfassung von Zeilen

- mit demselben Wert in einer oder mehreren vorgegebenen Spalten
- zum Zweck der Anwendung einer Standardfunktion auf diese Gruppe

Aggregatfunktionen in SQL (SQL-DML)

Beispiel: *„Gib den Gesamtarbeitszeitanteil jedes Projekts an.*

```
SELECT "PNr", SUM("ProzArbzeit") AS "PROZ-ARBEIT-PROJEKT"
FROM "ang-pro"
GROUP BY "PNr";
```

Ergebnis:

PNr	ProzArbzeit-PROJEKT
761235	290
770008	170
770114	360
770231	380

	PNr	AngNr	ProzArbzeit	
	761235	3207	100	
Gruppe 761235	761235	3115	50	
	761235	3190	50	
	761235	1435	40	SUM 290
	761235	3425	50	
	770008	2244	20	
Gruppe 770008	770008	1237	40	
	770008	2814	70	SUM 170
	770008	2454	40	
	770114	2814	30	
	770114	1435	60	
Gruppe 770114	770114	1237	60	
	770114	2454	60	SUM 360
	770114	3425	50	
	770114	2412	100	
	770231	3190	50	
Gruppe 770231	770231	2314	100	
	770231	2244	80	SUM 380
	770231	3115	50	
	770231	1324	100	

Aggregatfunktionen in SQL (SQL-DML)

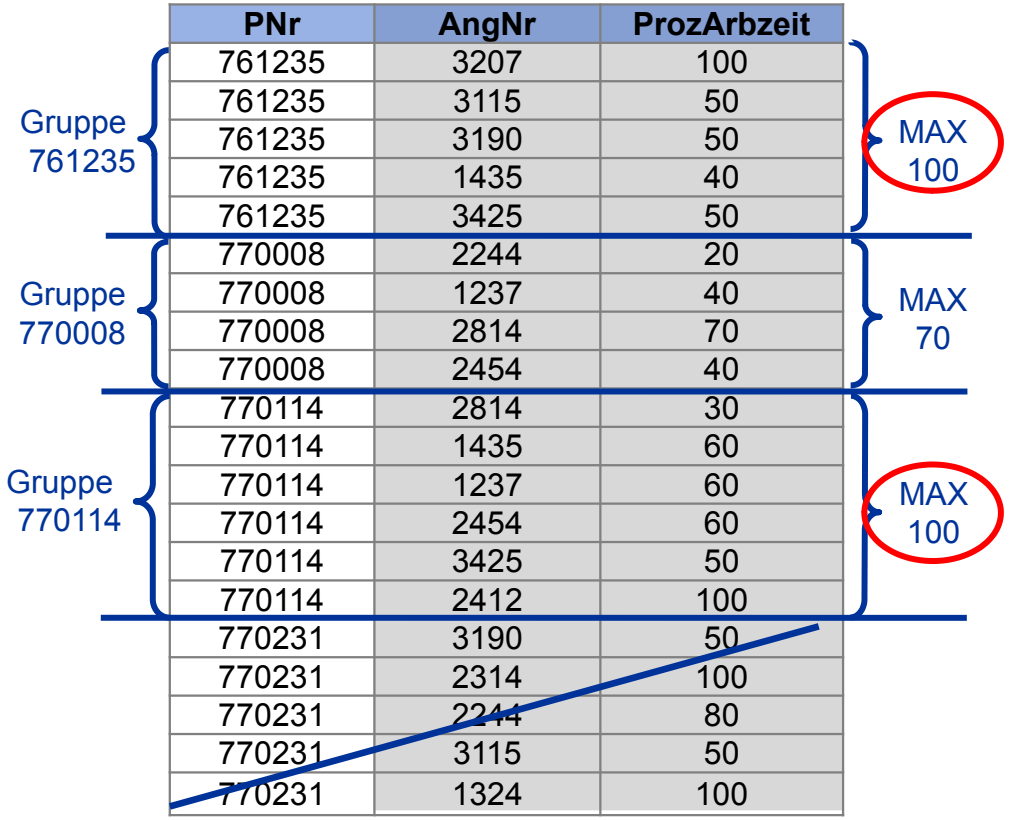
Auswahl von Gruppen mit HAVING-Bedingung

Welche der folgenden Projekte (761235, 770008, 770114) erreichen einen maximalen Arbeitszeitanteil von 100%.“

```
SELECT "PNr",
FROM "ang-pro"
WHERE "PNr"
    IN (761235, 770008, 770114)
GROUP BY "PNr"
HAVING MAX("ProzArbzeit")=100;
```

Ergebnis:

PNr
761235
770114



Aggregatfunktionen in SQL (SQL-DML)

Beispiele

- Geben Sie alle Projekte aus, an denen mindestens fünf Angestellte beteiligt sind.

- Geben Sie eine nach Abteilungsnummer sortierte Liste aus, die zeigt, wie viele Angestellte die einzelnen Abteilungen haben.

- Ermitteln Sie jene Wohnorte, in denen mindestens zwei Angestellte wohnhaft sind und sortieren Sie diese Liste alphabetisch.

Verknüpfung von Relationen (SQL-DML)

Abfragen mit JOIN (natural join)

Alle Projekte mit den Informationen der zugehörigen Angestellten

```
SELECT "PNr", a."AngNr", "NAME", "WOHNORT", "AbtNr",
       "ProzArbzeit"
FROM angestellte a, "ang-pro" ap
WHERE a."AngNr" = ap."AngNr";
```

oder (SQL-92)

```
SELECT "PNr", "AngNr", "NAME", "WOHNORT", "AbtNr",
       "ProzArbzeit"
FROM angestellte JOIN "ang-pro" USING("AngNr");
```

Verknüpfung von Relationen (SQL-DML)

Abfragen mit JOIN (unterschiedliche Attributnamen)

Geben Sie die Namen aller Projektleiter aus

```
SELECT "NAME"
FROM angestellte a, project p
WHERE a."AngNr" = p."PLEITER";
```

oder (SQL-92)

```
SELECT "NAME"
FROM angestellte JOIN projekt
ON(angestellte."AngNr" = projekt."PLEITER");
```

Verknüpfung von Relationen (SQL-DML)

Join mit Joinbedingung

„Alle Projekt-Nummern mit den Namen derjenigen Angestellten, die zu 50% mitarbeiten.“

```
SELECT PNr, a."AngNr", "NAME", "WOHNORT", "AbtNr",
       "ProzArbzeit"
FROM angestellte a, "ang-pro" ap
WHERE a."AngNr" = ap."AngNr" AND "ProzArbzeit"=50;
```

oder (SQL-92)

```
SELECT "PNr", "AngNr", "NAME", "WOHNORT", "AbtNr" ,
       "ProzArbzeit"
FROM angestellte JOIN "ang-pro" USING("AngNr")
WHERE "ProzArbzeit"=50;
```

Verknüpfung von Relationen (SQL-DML)

Geben Sie für jedes Projekt die Projektnummer und die Namen aller Angestellten an, die dort nicht Projektleiter sind (Angestellter muss nicht am Projekt mitarbeiten).

```
SELECT "PNr", "NAME"
FROM angestellte, projekt
WHERE "AngNr" <> "PLeiter";
```

Verknüpfung von Relationen (SQL-DML)

JOIN mit drei Relationen

Geben Sie für jeden Angestellten seinen Namen, sowie die Projekt-Filialen der Projekte an, an denen er mitarbeitet.

SELECT "NAME", "PFiliale"
FROM angestellte a, "ang-pro" ap, projekt p
WHERE a."AngNr" = ap."AngNr" **AND** ap."PNr" = p."PNr"

oder (SQL 92)

SELECT "NAME", "PFiliale"
FROM angestellte **JOIN** "ang-pro" **USING**("AngNr")
JOIN projekt **USING** ("PNr")

Verknüpfung von Relationen (SQL-DML)

Suchen Sie Paare von (unterschiedlichen) Angestellten, die in der selben Stadt wohnen (Das Ergebnis sollte keine redundanten Angaben enthalten).

SELECT a."AngNr" "ANr", b."AngNr" "B-NR"
FROM angestellte a, angestellte b
WHERE a."WOHNORT" = b."WOHNORT" **AND** a."AngNr" < b."AngNr"

a				b			
AngNr	**NAME**	**WOHNORT**	**AbtNr**	**AngNr**	**NAME**	**WOHNORT**	**AbtNr**
3115	Meyer	Karlsruhe	35	3115	Meyer	Karlsruhe	35
3115	Meyer	Karlsruhe	35	3207	Müller	Mannheim	30
3115	Meyer	Karlsruhe	35	2814	Klein	Mannheim	32
3207	Müller	Mannheim	30	3115	Meyer	Karlsruhe	35
3207	Müller	Mannheim	30	3207	Müller	Mannheim	30
3207	Müller	Mannheim	30	2814	Klein	Mannheim	32
2814	Klein	Mannheim	32	3115	Meyer	Karlsruhe	35
2814	Klein	Mannheim	32	3207	Müller	Mannheim	30
2814	Klein	Mannheim	32	2814	Klein	Mannheim	32

ANr	B-NR
2814	3207

Verknüpfung von Relationen (SQL-DML)

Es können sowohl mit SELECT hergeleitete Tabellen in der Form: (SELECT … FROM … WHERE …) *join-op*
(SELECT … FROM … WHERE …)
als auch Basis-Tabellen in der Form:
SELECT …
FROM table *join-op* table *[join-spec]*
WHERE …
miteinander verbunden werden.

Syntax einer Join-Operation:

join-op :=
CROSS JOIN |
[NATURAL] [INNER] JOIN |
[NATURAL] {LEFT | RIGHT | FULL} [OUTER] JOIN |
UNION JOIN

Verknüpfung von Relationen (SQL-DML)

Kartesisches Produkt:

table1 CROSS JOIN table2

berechnet das Kartesische Produkt und entspricht

SELECT * FROM table1, table2

Natural Join:

Die Verwendung des Schlüsselworts NATURAL führt zur Berechnung des Natural Joins.

→ alle gemeinsamen Spalten werden für die Bildung des Joins verwendet.

Verknüpfung von Relationen (SQL-DML)

Subqueries

Ergebnisse von verschiedenen Queries werden miteinander kombiniert.

Namen aller Angestellten, die mit 100 % ihrer Arbeitszeit an einem Projekt mitarbeiten.

Lösungsweg: Aufteilen des Problems in zwei Queries

Schritt: Bestimme aus *ang-pro* alle AngNr, die zu ProzArbzeit=100 gehören.

SELECT "AngNr"
FROM "ang-pro" WHERE "ProzArbzeit" = 100;

AngNr
3207
2412
2314
1324

Verknüpfung von Relationen (SQL-DML)

Schritt: Wähle in *angestellte* alle Tupel aus, deren AngNr in der oben gegebenen Ergebnismenge liegt.

```
SELECT "NAME"
FROM angestellte WHERE "AngNr" IN (3207, 2412, 2314,1324);
```

NAME
Müller
Groß
Schmitt
Müller

Realisierung mittels Subquery:

```
SELECT "NAME"
FROM angestellte WHERE "AngNr" IN (
    SELECT "AngNr"
    FROM "ang-pro" WHERE "ProzArbzeit" = 100);
```

Verknüpfung von Relationen (SQL-DML)

Beispiel

Namen aller Angestellten, die an mindestens einem Projekt in Karlsruhe mitarbeiten.

```
SELECT "NAME" FROM angestellte
WHERE  "AngNr" IN (
      SELECT  "AngNr" FROM "ang-pro"
      WHERE  "PNr" IN (
            SELECT "PNr" FROM  projekt
            WHERE  "PFiliale" ='Karlsruhe'));
```

oder

```
SELECT "NAME" FROM angestellte
   JOIN "ang-pro" USING("AngNr") JOIN projekt USING("PNr")
   WHERE "PFiliale"='Karlsruhe';
```

Verknüpfung von Relationen (SQL-DML)

Abfragen mit Existenz-Quantor

Suche Namen aller Angestellten, die keine Projektleiter sind.

```
SELECT "NAME"
FROM angestellte
WHERE NOT EXISTS (
      SELECT "PLeiter"
      FROM projekt
      WHERE  "PLeiter" = "AngNr");
```

Verknüpfung von Relationen (SQL-DML)

Mengenoperationen

Verknüpfung von Abfragen mit Mengenoperationen: Ergebnismengen müssen „vereinigungskompatibel“ sein (Namen und Wertebereiche korrespondierender Spalten müssen übereinstimmen).

Operationen: {UNION | INTERSECT | EXCEPT} [ALL]

Wird ALL verwendet, dann arbeiten die Operationen mit „uneigentlichen Tabellen“, sonst mit „eigentlichen Tabellen“ (keine Duplikate).

Verknüpfung von Relationen (SQL-DML)

Nummern aller Angestellten, die in Karlsruhe wohnen oder dort Leiter eines Projektes sind.

```
SELECT "AngNr"
FROM   angestellte
WHERE "WOHNORT" = ‘Karlsruhe‘

UNION

SELECT "PLeiter" as "AngNr"
FROM   projekt
WHERE "PFiliale" = ‘Karlsruhe‘;
```

Verknüpfung von Relationen (SQL-DML)

Übungsbeispiele:

1. Nummern aller Angestellten, die an mehr als einem Projekt arbeiten und in Karlsruhe wohnen.

2. Nummern aller Angestellten, deren Name mit ‘M‘ anfängt, die nicht Projektleiter sind.

Data Control Language

Data Control Language (DCL)

Hinzufügen und Entfernen von Berechtigungen

Beispiele

1. GRANT select, update ON angestellte TO wolfgang;
2. REVOKE ALL ON angestellte FROM wolfgang;

Indices und Prozedurale Integritätsregeln

Indices

Beschleunigen das Suchen und Sortieren nach Feldern
(Vergleiche Wörterbuch).

CREATE INDEX name ON table (columns);

Vorteil

Indizierte Felder können um Faktoren schneller durchsucht werden.

Nachteil

- Zusätzlicher Verwaltungsaufwand → Einfügeoperationen langsamer
- Speicherbedarf

Prozedurale Integritätsregeln

- **Deklarative Integritätsregeln** (declarative integrity constraints)
  - Erstellen mit Hilfe der Datendefinitionssprache
  - Bespiele: PRIMARY KEY, NOT NULL, CHECK, FOREIGN KEYS, …)

- **Prozedurale Integritätsregeln** (procedural integrity constraints)
  - Mit Hilfe von Auslösemechanismen (trigger)
  - Kleine Programme, Gruppen von Anweisungen, die beim Zugriff auf eine Spalte / Tabelle ausgeführt werden.

- **Trigger** – eine benutzerdefinierte Prozedur, die automatisch bei Erfüllung einer bestimmten Bedingung eines DBMS gestartet wird.
  Sie kann nicht nur Überprüfungs- sondern auch Berechnungsfunktionen übernehmen.

Prozedurale Integritätsregeln

Beispiel (schematisch):

CREATE TRIGGER "Personalbestand"
ON INSERATION OF "Mitarbeiter"
UPDATE "Abteilung"
SET "BESTAND" = "BESTAND" +1
WHERE "a#" = "A#_Unt"

Auslöser:
Datenbankoperation

Folgeaktion

A#	Bezeichnung	Bestand
A3	Informatik	11
A5	Personal	4
A6	Finanz	9

Prozedurale Integritätsregeln

Vorgangsweise in PostgreSQL (Syntax)

1. Erstellen der Trigger Funktion

CREATE OR REPLACE FUNCTION *FName*() **RETURNS TRIGGER AS**
'
BEGIN
Query
END;
' LANGUAGE 'plpgsql';

2. Verknüpfung der Triggerfunktion mit der zugehörigen Tabelle

CREATE TRIGGER *TriggerName* **AFTER INSERT ON** *Tabelle* **FOR EACH ROW EXECUTE PROCEDURE** *FName();*

Prozedurale Integritätsregeln

1. Erstellen der Triggerfunktion

Die virtuelle Tabelle NEW erlaubt den Zugriff auf das eingefügte Tuple. In der Triggerfunktion können beliebige SQL Anweisungen verwendet werden.

```
CREATE OR REPLACE FUNCTION Aktualisiere_Personalbestand()
RETURNS TRIGGER AS
'
BEGIN
    UPDATE "Abteilung" SET "BESTAND" = "BESTAND" +1
        WHERE "A#" = NEW."ANr";
  RETURN NEW;
END;
' LANGUAGE 'plpgsql';
```

Prozedurale Integritätsregeln

2. Zuordnung des Triggers zur entsprechenden Tabelle

CREATE TRIGGER *Personalbestand* **AFTER INSERT ON** Personal **FOR EACH ROW EXECUTE PROCEDURE** Aktualisiere_Personalbestand();

Quellenangabe

- Foliensatz Prof Stucky (Karlsruhe Institute of Technology – Institut für Angewandte Informatik und Formale Beschreibungsverfahren)
- Foliensatz Prof Panny & Prof Weichselbraun (Wirtschaftsuniversität Wien - Institut für Informationswirtschaft)
- Foliensatz Prof Bischof & Prof Studer (HTW Chur)
- Meier, Andreas (2010): *Relationale und postrelationale Datenbanken*. Berlin / Heidelberg / New York: Springer