DHBW Stuttgart

Datenbanken I

Kapitel 6.2 – SQL (DQL und View)

Modul: T2INF2004

Hinweis

DHBW
Duale Hochschule
Baden-Württemberg

Nutzungshinweis:

**Diese Unterlagen dürfen ausschließlich von Mitgliedern
(das sind Studierende, Bedienstete)
der Dualen Hochschule Baden-Württemberg Stuttgart eingesetzt werden.
Eine Weitergabe an andere Personen oder Institutionen ist untersagt.**

Inhalte Vorlesung

1. Grundlagen und Begriffsdefinitionen
2. Der konzeptionelle Datenbankentwurf (ER-Modell)
3. Der relationale Entwurf (logischer Entwurf)
4. Die relationale Entwurfstheorie (Normalformen)
5. Einführung zum Datenbankentwurf

6.1 Die Sprache SQL (Teile DDL und DML)

7. Relationale Algebra (eine formale Sprache)

6.2 Die Sprache SQL (Teile DQL und Views)

8. Transaktion und Mehrbenutzersysteme

9. Interne Speicherorganisation (Indexstrukturen)

Einfaches SELECT

```
SELECT Attribut1, Attribut2, …, Attribut_n
FROM Relation1, Relation2, …, Relation_n
WHERE Prädikat

SELECT *
FROM Kunde, Auftrag
;
```

- Beim FROM wird das Kreuzprodukt zwischen den beiden Eingabe-Schemata gebildet.
- Mit einer WHERE-Klausel wird dann das Ergebnis noch eingeschränkt indem die gesetzte Bedingung (Prädikat) erfüllt wird.
- Bei SELECT * wird eine Projektion auf das Ergebnis aus der FROM-Klausel ausgeführt.
- Das * in der Selektion sagt aus, dass das Ergebnisschema aus den Kreuzprodukt 1:1 übernommen wird.

Einfaches SELECT

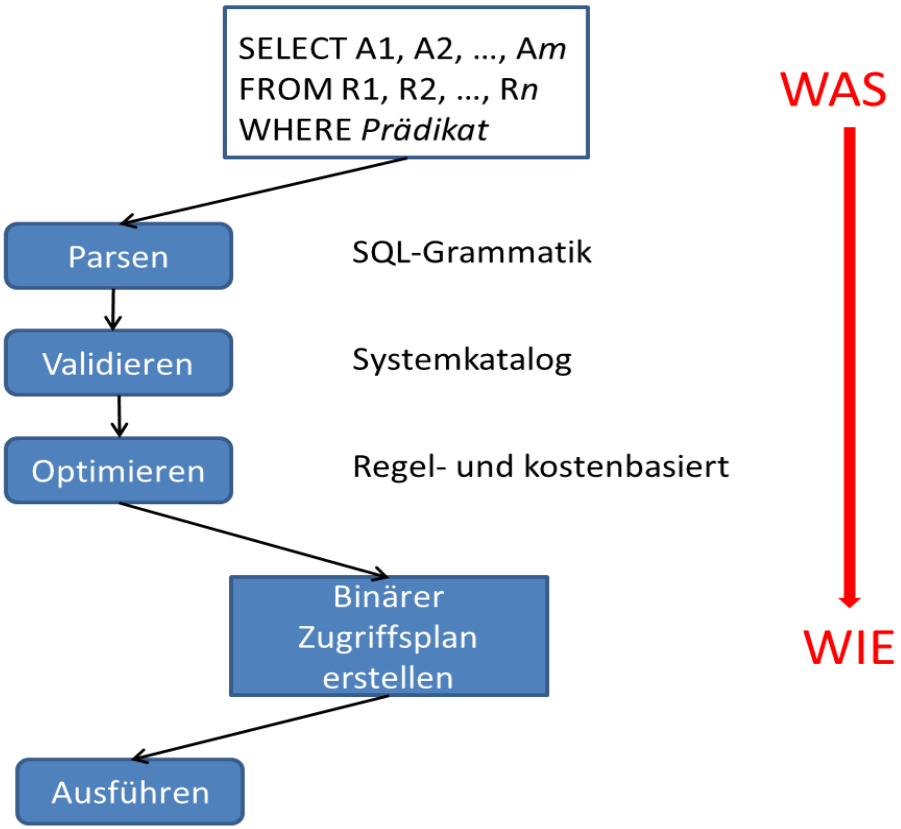
```
SELECT *
FROM Mitarbeiter
WHERE Nachname = 'Brecht'
;
```

- Man erhält nur noch das Tupel welches die Bedingung `Nachname = 'Brecht,` erfüllt.

Welche Ergebnis erhält man auf nachfolgende Abfrage und warum (testen)?

```
Select 42
from Mitarbeiter
;
Select 42 AS Die_Antwort_auf_alle_Fragen
from Mitarbeiter
;
```

Abfrage im DBMS verarbeiten



Weiter Angaben beim SELECT

- In der Relationalen Algebra gibt es keine doppelten Tupel, in SQL aber sehr wohl.
- Doppelte Werte müssen …….

```
SELECT ......... Vorname
FROM Mitarbeiter
;
```

Aufgabe:

1. Geben Sie die Vornamen alle Tupel der Tabelle *Mitarbeiter* aus.
2. Geben Sie die Vornamen alle Tupel der Tabelle *Mitarbeiter aus*, aber jeder Vorname darf nur einmal erscheinen.
3. Vergleichen Sie die Ergebnisse. Was fällt Ihnen bei der Reihenfolge auf?

Weiter Angaben beim SELECT

- Um eine bestimmte Reihenfolge zu erzeugen, müssen die Ausgabewerte gezielt geordnet werden.

```
SELECT DISTINCT Vorname
FROM Mitarbeiter
ORDER BY Vorname
;
```

Einschränkung beim SELECT

Nachfolgend zwei weitere Möglichkeiten die Suche einzuschränken.

1. Suche von Tupel aus einer Werteliste

```
SELECT * from Mitarbeiter
WHERE Nachname in ('Schulze', 'Kunze', 'Brecht')
;
```

Die Umkehrung dazu lautet NOT IN.

2. Mit BETWEEN und AND können wir bei der Abfrage einen Wertbereich betrachten (NOT ist auch möglich).

```
… WHERE Eintrittsdatum Between '01.01.2010' and
'01.01.2015'
```

Einschränkung beim SELECT

Aufgabe:
Geben Sie alle Mitarbeiter aus (Vorname, Nachname, Gehalt):
1. deren Gehalt zwischen 40000 und 60000 Euro liegt
2. und die Programmierer oder Tester sind
3. und deren Nachname auf ‚z' endet.

sortiert nach Gehalt

Funktionen beim SELECT

- Anzahl der Mitarbeiter

```
SELECT count (*) from Mitarbeiter
;
```

- Wer hat das kleinste und größte Gehalt im Softwarehaus?

```
SELECT  Min (Gehalt), Max (Gehalt)
from Mitarbeiter
;
```

Funktionen beim SELECT

In SQL gibt es auch eine große Anzahl von Funktionen.

- Z.B. interessiert uns das Durchschnittsgehalt aller Mitarbeiter unseres Softwarehauses:

```
SELECT AVG (Gehalt)
from Mitarbeiter
;
```

Frage: Wie hoch ist die Summe der geplanten Std. der noch offenen Leistungen (aus Aufträgen) ?

Funktionen beim SELECT

- Anzahl der Mitarbeiter

```
SELECT count (*) from Mitarbeiter
;
```

- Welches ist das kleinste und größte Gehalt im Softwarehaus?

Gruppen bilden (Group By)

- Was macht eigentlich unser DBMS, wenn wir COUNT(*).., sagen?
- Das DBMS bildet aus allen Tupeln der Tabelle (z.B. Mitarbeiter) eine Gruppe und berechnet dann innerhalb dieser Gruppe die Anzahl.
- Funktionen liefern aber immer nur ein Ergebnis.
- Wir wollen aber auch Ergebnisse über mehrere Spalten hinweg ermitteln.
- Hierzu dient uns die **GROUP BY**-Klausel.
- Das Gruppenkriterium kann aus einer oder mehreren Spalten bestehen.

```
SELECT  Skill, Count(*), AVG(Gehalt)
from Mitarbeiter
Group By Skill
Order By AVG(Gehalt)DESC
;
```

Frage: Was passiert, wenn wir bei der Sortierung anstatt AVG(Gehalt) nur Gehalt schreiben und warum ?

Gruppen bilden (Group By)

- Wir wollen nun auch die Ergebniswerte der Gruppierungen einschränken.
- Wir möchte z.B. alle Abteilungen angezeigt bekommen, bei denen das Durchschnittsgehalt > 50000€ beträgt.
- In diesem Fall müssten wir nach Abteilungen gruppieren und dann mit der **HAVING**-Klausel die Ergebnisse noch einschränken.

Aufgabe: Erstellen Sie die oben angegebene Abfrage (Abteilungen und Durchschnittsgehalt).

- Auch in der HAVING-Klausel kann nur auf die Werte eingeschränkt werden, welche in den Zeilengruppen gleich sind.

Zusammenfassung des SELECTs

- Wir wollen nun die komplette SELECT-Anweisung an dieser Stelle nochmals kurz zusammenfassen.

```
SELECT …
from …
Where …
Group By …
Having …
Order By …
;
```

- Zwingend notwendig ist beim SELECT nur die Angabe ……., alle anderen Klauseln sind optional.

1. Zuerst wird das FROM ausgeführt und eventuell ein Kreuzprodukt gebildet. Hier sind noch die Spalten aller beteiligten Tabellen vorhanden.
2. Nun werden über die ……. Klausel alle Zeilen aus dieser „Zwischentabelle“ entfernt, welche der Suchbedingung nicht entsprechen.

Zusammenfassung des SELECTs

```
SELECT …
from …
Where …
Group By …
Having …
Order By …
;
```

3. Über die ……. Klausel werden nun Zeilengruppen gebildet, welche die Anzahl der Zeilen aber noch nicht verringern.
4. Durch die ……. Bedingung werden nun alle Zeilengruppen entfernt, welche die Bedingung nicht erfüllen.
5. Im SELECT werden dann die Zeilen innerhalb der Gruppe zu einer Zeile zusammengefasst und durch die Projektion nur noch die gewünschten Spalten ins Ergebnis übernommen.
6. Zum Schluss werden die Zeilen nach der Angabe in der ORDER BY-Klausel sortiert und ausgegeben.

Zusammenfassung des SELECTs

Aufgabe:
Wir wollen das Durchschnittsgehalt der Mitarbeiter wissen, welche nach dem 1.1.2014 im Softwarehaus eingestellt wurden, bezogen auf die Skill-Gruppen (NULL ist keine Skill-Gruppe) und alles nach Durchschnittsgehalt sortiert.

Unterabfragen

- Was tun wir, wenn wir wissen wollen, welche Mitarbeiter in unserer Firma über dem Durchschnittsgehalt aller Mitarbeiter verdienen?
- Dies müssen wir mit dem derzeitigen Wissen in zwei Stufen abhandeln.

```
1. Select …… from Mitarbeiter;

2. Select Vorname, Nachname, Gehalt
   from Mitarbeiter
   Where Gehalt …….
   Order By Gehalt;
```

- In der WHERE Klausel gibt es nicht nur die Möglichkeit mit einem festen Wert zu vergleichen, sondern auch eine variable Abfrage mit SELECT einzubauen.
- Dies nennt man eine Unterabfrage oder subquery.

Unterabfragen

- Anstatt der Abfrage mit zwei SELECTs, lässt sich Aufgabe mit einer Unterabfrage viel eleganter lösen.

```
Select Vorname, Nachname, Gehalt
from Mitarbeiter
Where Gehalt > (Select Avg(Gehalt) from Mitarbeiter)
Order By Gehalt
;
```

- Man spricht hier nun von einer Hauptabfrage und einer Unterabfrage, welche durch Klammern getrennt sind.
- Unterabfragen sind auch in der HAVING-Klausel möglich.

- Bei der Unterabfrage muss das Ergebnis einem Vergleichsoperator entsprechen.
- Normalerweise ist das Ergebnis eines SELECTs wieder eine Tabelle und dies ist nicht zulässig.

Unterabfragen

Beispiel:

```
…
Where Gehalt >
(Select Gehalt from Mitarbeiter
where Pers_Nr=1)…
```

- Diese Unterabfrage wäre zulässig, da sie nur einen Wert liefert.
- Ändern wir aber nur eine Kleinigkeit und schreiben:

```
…
Where Gehalt >
(Select Gehalt from Mitarbeiter
where Pers_Nr>1)…
```

ist dies nicht mehr zulässig, da wir mehr als einen Ergebniswert aus der Unterabfrage erhalten.

- Hierbei würde ein Ergebniswert (Gehalt) mit dem Inhalt einer Tabelle verglichen werden und dies ist natürlich nicht möglich.

Unterabfragen

- Die Unterabfrage darf als Attributliste nur ein Element zurückliefern.
- Dieses kann aus Attributname, Funktion oder arithmetischer Ausdruck bestehen.
- Die WHERE-Klausel in der Unterabfrage muss so formuliert werden, dass diese nur eine Ergebniszeile zurück liefert.
- SQL bietet aber auch die Möglichkeit Unterabfragen mit mehreren Ergebniszeilen zu verarbeiten.
- Der Unterabfrage muss dazu einer der nachfolgenden Ausdrücke vorangestellt werden.

  ……….…

Unterabfragen

Aufgabe:
Wir suchen alle Mitarbeiter und deren Skill am Standort „Leinfelden“.
Ist kein Skill angegeben (NULL) wird der Mitarbeiter nicht ausgegeben. Wir sortieren nach Nachname.

Unterabfragen

- Wie verarbeitet das DBMS nun diese Anfrage?
  1. Die Unterabfrage, welche auch nur ein Attribut im Ergebniswert enthalten darf, liefert eine Anzahl Zeilen zurück.
  2. Der Ergebniswert jeder Zeile wird nun mit dem linken Wert der WHERE-Klausel verglichen.
  3. Das Ergebnis des Vergleichs ist „wahr“(true), wenn mindestens ein Ergebniswert der Unterabfrage die Vergleichsbedingung erfüllt.
  4. Erfüllt kein Ergebniswert die Bedingung, dann liefert der Vergleich „unwahr“(false) zurück.

Unterabfragen im INSERT

- Unterabfragen sind nicht nur beim SELECT, sondern auch in anderen SQL-Befehlen der DML, beispielsweise beim INSERT möglich.
- Beim CREATE-Statement hatten wir einen wichtigen Datentyp bewusst nur am Rande kurz erwähnt.
- Dies war der Datentyp …….

Frage: Was mussten wir seither tun, wenn wir einen Datensatz z.B. in die Tabelle *Mitarbeiter* einfügen wollten?

- Wir mussten wissen, welches …….. ist, um diese dann beim Einfügen des neuen Datensatzes um …… zu erhöhen.
- Durch die Unterabfrage ist es nun möglich, den höchsten Wert aus der Tabelle auszulesen und diesen um einen Wert zu erhöhen und beim INSERT direkt einzufügen.

Unterabfragen im INSERT

Aufgabe:

1. Fügen Sie nachfolgenden Datensatz in die Tabelle Mitarbeiter ein
   ```
   'Horst','Mahler',NULL,'01.04.1975','m',
   '15.3.2013',default,49000
   ```
2. Lesen Sie zuvor die höchste Personalnummer aus und erhöhen Sie diese um einen Wert.

Autoincrement (SERIAL)

- `Select …… From Mitarbeiter`
- Dies können wir dadurch ersetzen, dass wir bereits beim Anlegen der Tabelle (oder später durch ALTER) das sogenannte Autoincrement mit dem Constraint …….. setzen.

```
CREATE TABLE Mitarbeiter
 (
    Pers_Nr ……,
    Vorname varchar,
        ….
```

Aufgabe:

Fügen Sie nachträglich ein Autoincrement bei der Tabelle Mitarbeiter ein. Beim Einfügen eines neuen Datensatzes soll dann automatisch die nächsthöhere Personalnummer verwendet werden.

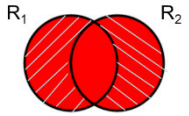
Autoincrement (SERIAL)

PostgreSQL provides three serial pseudo-types `SMALLSERIAL`, `SERIAL`, and `BIGSERIAL` with the following characteristics:

Name	Storage Size	Range
SMALLSERIAL	2 bytes	1 to 32,767
SERIAL	4 bytes	1 to 2,147,483,647
BIGSERIAL	8 bytes	1 to 9,223,372,036,854,775,807

UNION oder Vereinigung

- Mit `UNION` oder Vereinigung (∪) werden die Ergebnismengen zweier **Abfragen** miteinander vereinigt.



$R' := R_1 \cup R_2$

$[R'] := [R_1] = [R_2]$

$\Pi_{Vorname}(Mitarbeiter) \cup \Pi_{Vorname}(Kunden)$

Schreiben Sie dies in SQL

```
Select Vorname
from Mitarbeiter
.....
.....
.....

;
```

```
Select Nachname
from Mitarbeiter
.....
.....
.....

;
```

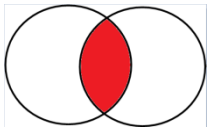
vorname character v
Paul
Rita
Claudia
Karin
Werner
Klaus
Florian
Edith
Manfred
Paul
Horst

vorname character varying
Manfred
Claudia
Klaus
Jan
Dominik

- `ALL` schaltet die Duplikateliminierung aus!

INTERSECT (Schnitt)

- Der Durchschnitt (∩) `INTERSECT` gibt an, welche Attribute zweier Abfragen im Schnitt der beiden (Mengen) liegen.



$R' := R_1 \cap R_2$
$[R'] := [R_1] = [R_2]$

$\Pi_{Vorname}$ (Mitarbeiter) ∩ $\Pi_{Vorname}$ (Kunden)

Schreiben Sie dies in SQL

```
Select Vorname
from Mitarbeiter
.....
Select Vorname
.....
;
```

vorname character v
Paul
Rita
Claudia
Karin
Werner
Klaus
Florian
Edith
Manfred
Paul
Horst

vorname character varying
Manfred
Claudia
Klaus
Jan
Dominik

INTERSECT (Schnitt)

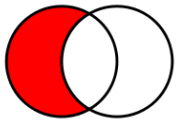
- Testen Sie INTERSECT ALL und überlegen Sie im Vorfeld was das Ergebnis sein wird und warum.
- Erweitern Sie (temporär) die beiden Tabellen mit dem gerade erworbenen Wissen sodass ein INTERSECT ALL ein anderes Ergebnis liefert.

vorname character v
Paul
Rita
Claudia
Karin
Werner
Klaus
Florian
Edith
Manfred
Paul
Horst

vorname character varying
Manfred
Claudia
Klaus
Jan
Dominik

EXCEPT (Differenz oder Minus)

- Die Differenz (Minus) zweier Mengen oder Abfragen wird in SQL `EXCEPT` genannt.



$R' := R_1 - R_2$
$[R'] := [R_1] = [R_2]$

$\Pi_{Vorname}$ (Mitarbeiter) - $\Pi_{Vorname}$ (Kunden)

```
Select Vorname
from Mitarbeiter
EXCEPT (ALL)
Select Vorname
from Kunde
;
```

Was passiert bei `EXCEPT ALL`?

vorname character v
Paul
Rita
Claudia
Karin
Werner
Klaus
Florian
Edith
Manfred
Paul
Horst

vorname character varying
Manfred
Claudia
Klaus
Jan
Dominik

Abfragen auf mehrere Tabellen

Frage: Welches ist die einfachste Verknüpfung zweier Tabellen?

.....

.....

- Das Schema der neuen Tabelle ist die Vereinigung der Schemata beider Ausgangstabellen.

Frage: Wie erhalten wir alle Kunden mit den von Ihnen erteilten Aufträgen?

σ ..... (Kunde x Auftrag)

```
Select *
.....
.....
;
```

Abfragen auf mehrere Tabellen

Aufgabe:

1. Setzen Sie bei einem Mitarbeiter des Softwarehauses das Gehalt auf den Wert eines anderen Mitarbeiters.
2. Erstellen Sie nun eine Abfrage deren Ergebnis die beiden Mitarbeiter sind welche dasselbe Gehalt haben.

Theta JOIN

- Erst mit SQL-92 wurde die Möglichkeit des JOINS geschaffen.

TJ:=R1 ⋈p R2
[TJ]:= [R1] ∪ [R2]

```
Select *
from Kunde JOIN Auftrag
On Kunden_Nr = erteilt_von
;
```

Frage: Was ist nun der Unterschied der beiden Statements?

- Durch den JOIN Operator sagen wir dem Optimizer, dass eine ..... existiert, an derer er entlang die beiden Tabellen verbinden soll.

Equi JOIN

- Der Equi-Join verbindet die Tabellen über die Werte zweier Tabellen, bei denen ein Teilschema der linken Relation vor dem Joinsymbol gleich einem Teilschema der rechten Relation sein muss.

$EJ:=R_1 \bowtie_{[L],[R]} R_2$

```
Select *
from Tabelle1 JOIN Tabelle2
Using (Attribut,…);
```

Aufgabe:
Geben Sie nach Nachname sortiert alle Mitarbeiter mit Vorname, Nachname und die Aufträge aus, an denen Sie arbeiten (gearbeitet haben).

Natural JOIN

Beim Natural-Join (als Spezialfall des Equi-Joins) werden zwei Tabellen über die Spalten miteinander verknüpft welche denselben Namen und Inhalt haben.

```
Select *
from Tabelle1 Natural JOIN Tabelle2
;
```

Wir wollen uns nun die kompletten Daten der Projektleiter anzeigen lassen.

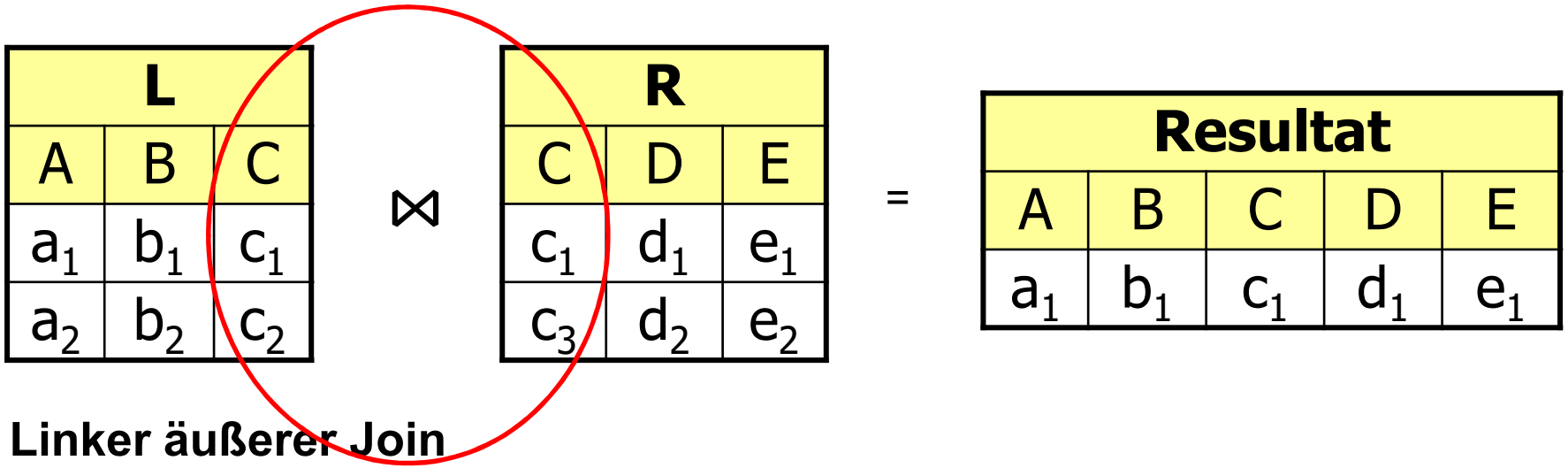
```
Select *
from ……
;
```

- Diese Joins (Theta, Equi, Natural), welche jeweils einen Join-Partner haben, werden auch **Innere Joins** oder INNER JOIN genannt.

```
Select *
from ……  Natural ……..JOIN ……
;
```

Ergebnis verschiedener Joins

Natürlicher Join



L		
A	B	C
a_1	b_1	c_1
a_2	b_2	c_2

$\bowtie$

R		
C	D	E
c_1	d_1	e_1
c_3	d_2	e_2

=

Resultat				
A	B	C	D	E
a_1	b_1	c_1	d_1	e_1

Linker äußerer Join

L		
A	B	C
a_1	b_1	c_1
a_2	b_2	c_2

$\bowtie$

R		
C	D	E
c_1	d_1	e_1
c_3	d_2	e_2

=

Resultat				
A	B	C	D	E
a_1	b_1	c_1	d_1	e_1
a_2	b_2	c_2	-	-

Ergebnisse verschiedener Joins

Rechter äußerer Join

L		
A	B	C
a_1	b_1	c_1
a_2	b_2	c_2

$\bowtie$

R		
C	D	E
c_1	d_1	e_1
c_3	d_2	e_2

=

Resultat				
A	B	C	D	E
a_1	b_1	c_1	d_1	e_1
-	-	c_3	d_2	e_2

Äußerer Join (voller)

L		
A	B	C
a_1	b_1	c_1
a_2	b_2	c_2

$\bowtie$

R		
C	D	E
c_1	d_1	e_1
c_3	d_2	e_2

=

Resultat				
A	B	C	D	E
a_1	b_1	c_1	d_1	e_1
a_2	b_2	c_2	-	-
-	-	c_3	d_2	e_2

ÄUSSERE JOINs

- Beim Äußeren Join (**OUTER JOIN**) werden jeweils die Tupel mit angezeigt für die es keinen Join-Partner gibt.

Aufgabe:
Wir wollen nun wissen, welche unserer Projekt-Mitarbeiter bisher noch an keiner Leistung gearbeitet haben.

```
Select ……..
from ……..
……..
……..
……..
……..
;
```

Freie Mitarbeiter

Aufgabe:
Wir erweitern die Aufgabe nun etwas und wollen nun wissen, welche Mitarbeiter gerade für neue Aufgaben frei sind. D.h. derzeit an keiner Leistung arbeiten und wie lange sie bereits Projekterfahrung haben.

```
select pers_nr, vorname, nachname from mitarbeiter
where Pers_nr ….
(
……
……
……
……
));
```

Was müssen wir tun, wenn wir die Projekterfahrung noch mit ausgeben wollen?

Freie Mitarbeiter

Eine alternative Abfrage generiert eine temporäre Tabelle mit dem SQL-Statement .....

```
..... Keine_Arbeit .....
(select pers_nr, projekterfahrung from
mitarbeiter_projekt
where Pers_Nr not in
(
Select Pers_nr
.....
.....
.....
))
Select pers_nr, Nachname, Vorname, Projekterfahrung
from Keine_Arbeit natural Join Mitarbeiter
;
```

Semi JOIN

L		
A	B	C
a_1	b_1	c_1
a_2	b_2	c_2

$\ltimes$

R		
C	D	E
c_1	d_1	e_1
c_3	d_2	e_2

=

Resultat		
A	B	C
a_1	b_1	c_1

- Beim Semi Join (hier L mit R) bleiben alle Tupel von links übrig, welche einen Join-Partner haben.
- Für den Semi Join gibt es keinen Befehl in Postgres. Wir können diesen aber mit uns bereits bekannten Befehlen nachbauen.

Aufgabe:

Erstellen Sie einen Left Semi JOIN von Mitarbeiter und Projekte?

```
Select * from
mitarbeiter
.....
.....
```

Views / Sichten

- Die Spezialisierung der Mitarbeiter wurde in verschiedenen Tabellen durchgeführt.
- Tabelle Projektmitarbeiter enthält nur noch die Attribute, welche die jeweilige Spezialisierung anzeigen (Std.-Satz und Projekterfahrung).
- **Anforderung:** Die Daten eines Projektmitarbeiters sollen wieder komplett anzeigt werden.
- `JOIN` Tabellen *Mitarbeiter* und *Mitarbeiter_Projekt*

```
Select * from Mitarbeiter

.....

;
```

Einfache Sichten

- Auf der externen Ebene wollen wir die kompletten Daten eines Projektmitarbeiters verarbeiten können.
- Es müsste immer dieser JOIN durchgeführt werden.
- Es gibt die Möglichkeit, definierte Abfragen zu einer sogenannten View (Sicht) zusammen zu fassen und als solche abzuspeichern.

```
Create View Projektmitarbeiter As
Select * from Mitarbeiter
natural Join Mitarbeiter_Projekt
;
```

Diese Views können wir nun wie eine „normale“ Tabelle verwenden.

```
Select * from Projektmitarbeiter
;
```

Eine View in Postgres

- Diese View wird nicht wie eine Tabelle in Postgres behandelt.
- Die View wird im Unterpunkt „Views“ angelegt.
- Das `SELECT` wird komplett ausgerollt hinterlegt.
- Wenn die View *Projektmitarbeiter* aufgerufen wird, wird an deren Stelle das `SELECT` ausgeführt.
- Zu der View gibt es auch eine Regel (Rules).
- Dies Regel besagt:
  Wird ein `SELECT` auf die View durchgeführt, so wird an deren Stelle das nachfolgende `SELECT` ausgeführt.
- Mit `ALTER` und `DROP` können Views auch verändert oder gelöscht werden.

Datenunabhängigkeit durch Views

- Ein Mitarbeiter im Softwarehaus ist für das Kursmanagement zuständig.
- Dieser Mitarbeiter (Herr Klein) braucht für seine Aufgaben verschiedene Informationen aus der Datenbank.
- Wir generieren ihm dazu folgende Views:
  - Welcher Mitarbeiter hat welche Kurse besucht.
  - Welche Kurse bietet welches Trainingsinstitut an.
  - Welcher Mitarbeiter hat in diesem Jahr noch keinen Kurs besucht.

Aufgabe:
Legen Sie exemplarisch die erste View an für den Mitarbeiter an.

```
Create ……
Select pers_nr, vorname, nachname, ……
……..
from …… natural Join …….
……;
```

Daten ändern über eine View

- Wir wollen noch eine View generieren, welche die Tabelle Kurs genau abbildet.

Aufgabe:
Legen diese View mit dem Namen V-Kurs an.

.....

.....

Einfügen Daten mit INSERT in die Tabelle Kurs über die View.

```
Insert into V_kurs
Values (2223, 'Testkurs', 'Testinstitut')
```

Daten sind über eine View auch änderbar.

Datenunabhängigkeit durch Views

- Eine organisatorische Änderung erfordert nun die Anpassung des konzeptionellen Datenbankentwurfs des Softwarehauses.
- Eine neue Tabelle *Institut* muss angelegt werden.
- *Inst-Nr* als künstlichen Primärschlüssel.
- *Institutsname, Ort*.. sollen in der neuen Tabelle gespeichert werden.
- Fremdschlüssel in die Tabelle Kurs anstatt der Spalte *Institut* einfügen.

Aufgabe:

Legen Sie die neue Tabelle „training_institut“ mit Verbindung an.

Der Mitarbeiter soll mit:

```
SELECT * from Mitarbeiter_besucht_kurs
```

dieselben Daten wie bisher erhalten. Schreiben Sie die neue View.

```
Create ……..
Select pers_nr, vorname, ……..
……
from ……. natural Join ……..
natural Join ……..
……
;
```

Daten ändern über eine View

Aufgabe:
Verändern Sie die View **V-Kurs** damit diese wieder dieselben Daten liefert.

```
CREATE VIEW V_kurs AS
.....
.....
;
```

Fügen Sie nun erneut Daten mit INSERT ein.

```
Insert into V_kurs
Values (2224, 'Testkurs', 'Testinstitut')
```

Die Daten sind über die View .....

Daten ändern über eine View

Eine Datenunabhängigkeit von Sichten kann nur bis zu einem gewissen Grad gewährleitstet werden.

- Werden Views aus mehreren Tabellen (JOIN) gebildet, so wird aus der View eine .....
- Welche Veränderungen machen eine View noch „read only“:
  - Wenn in der View eine Aggregation (SUM, COUNT..) stattfindet.
  - Auch bei DISTINCT, UNION und Unterabfragen ist eine Datenveränderung über eine Sicht nicht mehr möglich.

Aufgabe:
Prüfen Sie die Fehlermeldung welche beim INSERT entsteht und versuchen Sie diese zu interpretieren.

Ende Kapitel 6.2