Datenbanken und SQL

Kapitel 4

Die Datenbankzugriffssprache SQL

Die Datenbankzugriffssprache SQL

- Der Select-Befehl
  - Der Hauptteil des Select-Befehls
    - From-, Where-, Select-Klausel
    - Group By und Having
    - Join
  - Union, Except, Intersect
  - Order By
  - Die Arbeitsweise des Select-Befehls
- Delete, Update, Insert
- Transaktionsbetrieb
- Relationale Algebra

Überblick (1)

- SELECT * FROM Personal ;

Persnr	Name	Ort	Vorgesetzt	Gehalt
1	Maria Forster	Regensburg	NULL	4800.00
2	Anna Kraus	Regensburg	1	2300.00
3	Ursula Rank	Frankfurt	6	2700.00
4	Heinz Rolle	Nürnberg	1	3300.00
5	Johanna Köster	Nürnberg	1	2100.00
6	Marianne Lambert	Landshut	NULL	4100.00
7	Thomas Noster	Regensburg	6	2500.00
8	Renate Wolters	Augsburg	1	3300.00
9	Ernst Pach	Stuttgart	6	800.00

Überblick (2)

```
SELECT  Name, Ort
FROM    Personal ;
```

Name	Ort
Maria Forster	Regensburg
Anna Kraus	Regensburg
Ursula Rank	Frankfurt
Heinz Rolle	Nürnberg
Johanna Köster	Nürnberg
Marianne Lambert	Landshut
Thomas Noster	Regensburg
Renate Wolters	Augsburg
Ernst Pach	Stuttgart

```
SeleCT name
, orT        fROM
                Personal ;
```

Die Syntax des SELECT Befehls

Select-Befehl:

Select-Hauptteil

[{ UNION [ALL] | EXCEPT | INTERSECT }

Select-Hauptteil

]

[…]

[ORDER BY Ordnungsliste]

SELECT Befehl: Hauptteil

Select-Hauptteil:

```
SELECT     [ ALL | DISTINCT ] Spaltenauswahlliste
 FROM      Tabellenliste
 [ WHERE           Bedingung ]
 [ GROUP BY        Spaltenliste
    [ HAVING       Bedingung ] ]
```

Hinweise zur Syntax

- GROSSBUCHSTABEN : Reservierte Bezeichner
- [xyz] : Wahlfrei
- { xx | yy | zz } : Auswahlliste
- [xx | yy | zz] : Wahlfreie Auswahlliste
- a2, a_a, aaa, ab___33 : Erlaubte Bezeichner
  Erstes Zeichen ist Buchstabe
  Weitere Zeichen: Buchstabe / Ziffer / ‚_‘
- … : Wiederholzeichen

Vergleich: SQL – Relationale Algebra

	Algebra	SQL
Vereinigung	$R1 \cup R2$	UNION
Schnitt	$R1 \cap R2$	INTERSECT
Differenz	$R1 \setminus R2$	EXCEPT
Kreuzprodukt	$R1 \times R2$	Tabellenliste
Restriktion	$\sigma_{Bedingung}(R)$	WHERE - Klausel
Projektion	$\pi_{Auswahl}(R)$	SELECT - Klausel
Verbund	$R1 \bowtie R2$	Tabellenliste
Division	$R1 \div R2$	---
Umbenennung	$\rho_{Rneu}(R)$	Spaltenauswahlliste, Tabellenliste

Die FROM Klausel

Tabellenliste:

Tabellenreferenz [, ...]

Tabellenreferenz:

Tabellenname [[AS] Aliasname]
| (Select-Hauptteil) [[AS] Aliasname]
| (Tabellenreferenz) [[AS] Aliasname]
| Joinausdruck [[AS] Aliasname]

Die FROM Klausel: Beispiele

- SELECT * FROM Personal AS P ;
- SELECT * FROM (SELECT * FROM Personal) AS P2 ;
- SELECT * FROM (Personal) ;
- SELECT * FROM Personal, Auftrag ;

Füllwort AS in Oracle nicht erlaubt

In SQL Server: Klammern hier nicht erlaubt

Aliasname hier in MySQL und SQL Server zwingend

Kreuzprodukt

Kreuzprodukt

SELECT * FROM Personal, Auftrag ;

Persnr	Name	Ort	Vorg.	Gehalt	A.Nr	Datum	K.nr	Persnr
1	Maria Forster	Regensburg	NULL	4800.00	1	04.01.13	1	2
1	Maria Forster	Regensburg	NULL	4800.00	2	06.01.13	3	5
1	Maria Forster	Regensburg	NULL	4800.00	3	07.01.13	4	2
1	Maria Forster	Regensburg	NULL	4800.00	4	18.01.13	6	5
1	Maria Forster	Regensburg	NULL	4800.00	5	06.02.13	1	2
2	Anna Kraus	Regensburg	1	2300.00	1	04.01.13	1	2
2	Anna Kraus	Regensburg	1	2300.00	2	06.01.13	3	5
2	Anna Kraus	Regensburg	1	2300.00	3	07.01.13	4	2
2	Anna Kraus	Regensburg	1	2300.00	4	18.01.13	6	5
2	Anna Kraus	Regensburg	1	2300.00	5	06.02.13	1	2
3	Ursula Rank	Frankfurt	6	2700.00	1	04.01.13	1	2
3	Ursula Rank	Frankfurt	6	2700.00	2	06.01.13	3	5
3	Ursula Rank	Frankfurt	6	2700.00	3	07.01.13	4	2
…	…	…	…	…	…	…	…	…

Die SELECT Klausel

SELECT Klausel:

SELECT [ALL | DISTINCT] Spaltenauswahlliste

Spaltenauswahlliste:

Spaltenausdruck [[AS] Aliasname] [, ...]

Füllwort AS hier auch in Oracle erlaubt

```
SELECT Name, 12 * Gehalt AS Jahresgehalt
 FROM  Personal ;
```

$\rho_{12*Gehalt->Jahresgehalt}(\pi_{Name,12*Gehalt}(Personal))$

Vergleich: SQL – Rel. Algebra

Qualifizieren von Attributen

```
SELECT  *
 FROM   Personal, Auftrag ;
```

Alle Attribute, Reihenfolge abhängig von From-Klausel

```
SELECT  Personal.*, Auftrag.*
 FROM   Personal, Auftrag ;
```

Erst alle Personalattribute, dann alle Auftragsattribute

```
SELECT  Personal.Persnr, Name, Ort,  Vorgesetzt, Gehalt,
          AuftrNr, Datum, Kundnr, Auftrag.Persnr
 FROM   Personal, Auftrag ;
```

Reihenfolge der Attribute angegeben

Skalare Funktionen (Auswahl)

UPPER	Wandelt Kleinbuchstaben in Großbuchstaben um. Andere Zeichen bleiben unverändert.
LOWER	Wandelt Großbuchstaben in Kleinbuchstaben um. Andere Zeichen bleiben unverändert.
TRIM	Führende und schließende Leerzeichen werden entfernt
RTRIM	Schließende Leerzeichen werden entfernt
SUBSTRING	Aus einer Zeichenkette wird eine Teilzeichenkette extrahiert

Syntax: UPPER(String) analog LOWER…

SUBSTRING(String FROM Pos FOR Anzahl)

In SQL Server: String, Pos, Anzahl

In Oracle: SUBSTR(String, Pos, Anzahl)

Skalare Funktionen (Beispiele)

```
SELECT UPPER(Name), LOWER(Name), Name
FROM Personal ;
```

Ausgabe des Namens in Klein- und Großbuchstaben

```
SELECT Name || '.' , TRIM(Name) || '.'
FROM Personal ;
```

In ANSI SQL: Konstante Zeichenketten in Hochkommata

In Oracle und ANSI SQL: Konkatenierungsoperator ||

Ausgabe: Leerzeichen zwischen Name und Punkt beachten!

```
SELECT SUBSTRING(TRIM(Name) FROM 1 FOR
        POSITION(' ' IN TRIM(Name)))
FROM Personal ;
```

In Oracle: SUBSTR(…,…,…)

Position des ersten Vorkommens eines Zeichens in einer Zeichenkette, in Oracle: InStr(Trim(Name),' ')

Ausgabe: Nur Vornamen!

Aggregatfunktionen in SQL

AVG	Average	Mittelwert, ermittelt über alle Zeilen
COUNT	Count	Anzahl aller Zeilen
MAX	Maximum	Maximalwert aller Zeilen
MIN	Minimum	Minimalwert aller Zeilen
SUM	Sum	Summenwert, summiert über alle Zeilen

```
SELECT  Persnr, Name, 12*Gehalt + 1000 *(6 - Beurteilung) AS Jahresgehalt
 FROM   Personal ;
```

```
SELECT  SUM (12*Gehalt +1000 *(6- Beurteilung)) AS Jahrespersonalkosten
 FROM   Personal ;
```

In MySQL: hier kein Leerzeichen

Der Bezeichner DISTINCT

```
SELECT Ort FROM Personal ;
SELECT DISTINCT Ort FROM Personal ;
SELECT COUNT ( Ort ) FROM Personal ;
SELECT COUNT ( DISTINCT Ort ) FROM Personal ;

SELECT COUNT(*) FROM Personal ;
SELECT COUNT( Vorgesetzt ) FROM Personal ;
SELECT COUNT( DISTINCT Vorgesetzt ) FROM Personal ;
```

9 Orte

6 unterschiedliche Orte

9

6

9, da 9 Tupel

7, da 2 Nullwerte

2, da nur 2 unterschiedliche Werte

Die WHERE Klausel

```
SELECT   MIN( Gehalt )
 FROM    Personal
 WHERE   Gehalt > 3000 ;
```

Kleinstes Gehalt größer 3000

nicht: !=

Boolesche Operatoren	NOT , AND , OR
Vergleichsoperatoren	< , <= , > , >= , = , <>
Intervalloperator	[NOT] BETWEEN ... AND
Enthaltenoperator	[NOT] IN
Ähnlichkeitsoperator	[NOT] LIKE
Nulloperator	IS [NOT] NULL
Auswahloperatoren	ALL , ANY , SOME
Existenzoperator	EXISTS

Intervalloperator

A BETWEEN B AND C  A >= B AND A <= C

A NOT BETWEEN B AND C  NOT A BETWEEN B AND C

- **Beispiel:**

```
SELECT   *
 FROM    Personal
 WHERE   Gehalt BETWEEN 2000 AND 3000 ;
```

Enthaltenoperator

A IN (B_1, B_2, …, B_n) ⟺ A=B_1 OR A=B_2 OR … OR A=B_n

A NOT IN (B_1, B_2, …, B_n) ⟺ NOT A IN (B_1, B_2, …, B_n)

- **Beispiel:**

```
SELECT   *
 FROM    Personal
WHERE    Ort IN ('Regensburg', 'Nürnberg', 'Passau') ;
```

Ähnlichkeitsoperator LIKE

Wildcardsymbole	in SQL	in Windows in Unix
Beliebig viele Zeichen (0..n)	%	*
Genau ein Zeichen (1..1)	_	?

- Beispiele:

```
SELECT    *
 FROM     Personal
 WHERE    Name LIKE '%Heinz%' ;

SELECT     *
 FROM      Personal
 WHERE     Upper(Name) LIKE  '%A_E%'  ;
```

Nulloperator

A IS NULL — true, falls A gleich Null ist

A IS NOT NULL  NOT A IS NULL

- Beispiel:

```
SELECT   *
 FROM    Personal
WHERE   Vorgesetzt IS NULL ;
```

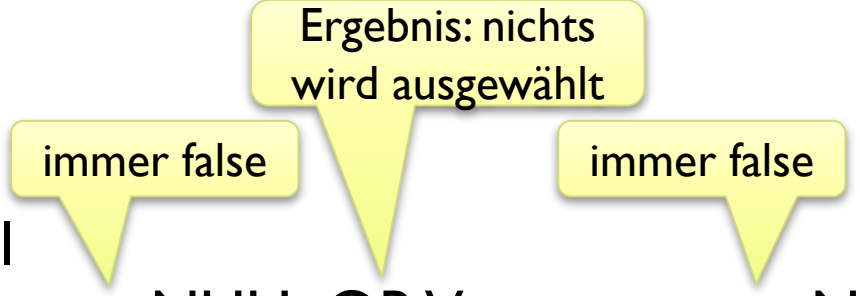
- nicht: WHERE Vorgesetzt = NULL

Einschub: NULL

- **Wichtig:** Alle Nullwerte unterscheiden sich voneinander
- Informell gilt also:

  NULL ist ungleich NULL

- Jeder Vergleich mit Nullwerten liefert FALSE zurück!
- Folgerung: Wir verwenden den Nulloperator **Is Null**
- **Beispiel:**



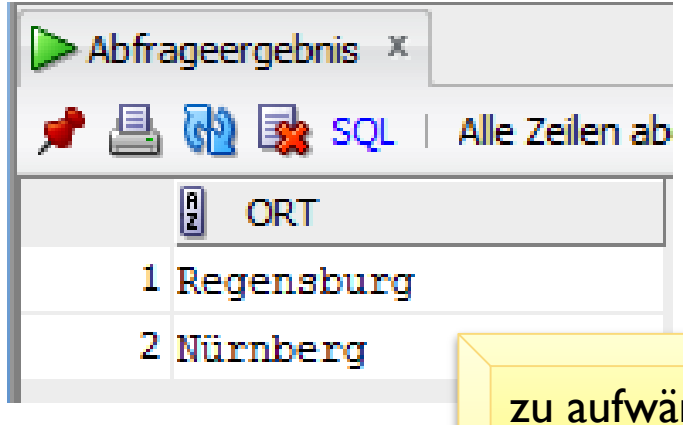

```
SELECT   *
FROM     Personal
WHERE    Vorgesetzt = NULL OR Vorgesetzt <> NULL
```

Unterabfragen (1)

- **Anfrage:** Gesucht sind alle Mitarbeiter, die in den gleichen Orten wie Mitarbeiterinnen 2 und 5 wohnen.
- **Lösung 1:**

```
SELECT  Ort
 FROM   Personal
 WHERE Persnr IN (2, 5) ;

SELECT  *
 FROM   Personal
 WHERE Ort IN ( 'Nürnberg', 'Regensburg' );
```



zu aufwändig bei großen Zwischen-ergebnissen

Unterabfragen (2)

- **Anfrage:** Gesucht sind alle Mitarbeiter, die in den gleichen Orten wie Mitarbeiterinnen 2 und 5 wohnen.
- **Lösung 2:**

```
SELECT  *
 FROM   Personal
 WHERE Ort IN ( SELECT  Ort
                FROM    Personal
                WHERE  Persnr IN ( 2, 5 ) ) ;
```

Unterabfrage: liefert alle Orte von Mitarbeiter 2 und 5

Unterabfrage steht in Klammern

Globaler Attributname bezieht sich auf globale Relation

Lokale Attributnamen beziehen sich auf lokale Relation

Auswahloperatoren

Ausdruck op ANY (Unterabfrage)
Ausdruck op SOME (Unterabfrage)
Ausdruck op ALL (Unterabfrage)

Any ≙ Some:
True, falls mindestens eine Übereinstimmung

All:
True, falls alle übereinstimmen

Vergleichsoperator:
<, <=, >, >=, =, <>

- Beispiel:

```
SELECT  *
 FROM   Personal
 WHERE Ort = ANY ( SELECT  Ort
                   FROM    Personal
                   WHERE  Persnr IN ( 2, 5 ) ) ;
```

„=ANY“ ≙ „IN“

Auswahloperatoren: Beispiele

```
SELECT  *
 FROM   Personal
WHERE   Gehalt >= ALL ( SELECT Gehalt
                        FROM Personal  ) ;
```

Ergebnis: Mitarbeiter mit maximalem Gehalt

```
SELECT  *
 FROM   Personal
WHERE   Gehalt = MAX( Gehalt ) ;
```

Fehler: Aggregatfunktion direkt in Where Klausel nicht möglich!

```
SELECT  *
 FROM   Personal
WHERE   Gehalt IN ( SELECT MAX(Gehalt)
                    FROM Personal  ) ;
```

Hier ist auch „=“ erlaubt

Korrekt: Aggregatfunktion in Unterabfrage

Unterabfragen oder Kreuzprodukt?

- **Gesucht:** Alle Mitarbeiter, die weniger als Mitarbeiter 3 verdienen

```
SELECT  *
 FROM   Personal
WHERE   Gehalt < ( SELECT Gehalt
                   FROM Personal
                   WHERE Persnr = 3 ) ;
```

Hier erlaubt: Unterabfrage liefert nur einen Wert

Gehalt von Mitarbeiter 3

```
SELECT  P1.*
 FROM   Personal AS P1, Personal AS P2
WHERE   P1.Gehalt < P2.Gehalt
  AND   P2.Persnr = 3 ;
```

Kreuzprodukt enthält 9 Tupel

P2.Gehalt: Gehalt von Mitarbeiter 3

Restriktion: P2 enthält nur ein Tupel!

Existenzoperator

- **EXISTS** liefert True, wenn die folgende Unterabfrage mindestens ein Ergebnis liefert.
- **Beispiel** (siehe vorherige Folie):

```
SELECT   *
 FROM    Personal  AS P1
 WHERE  EXISTS (  SELECT  *
                  FROM    Personal
                  WHERE   Persnr = 3
                    AND   Gehalt > P1.Gehalt ) ;
```

Wenn P1.Gehalt < 2700, dann liefert Unterabfrage Ergebnis

Ändert sich von Zeile zu Zeile: 4800, 2300, 2700, …

Nur Mitarbeiter 3

Also: Gehalt von Mitarbeiter 3: 2700

Die GROUP BY Klausel

- GROUP BY Spaltenliste
- Spaltenliste:
  - Liste von Spaltennamen (Ausdrücke sind nicht erlaubt!)
- Beispiel:
  - Gesucht: Alle Wohnorte der Mitarbeiter, jeder Wohnort soll nur einmal aufgeführt werden

```
SELECT DISTINCT Ort
FROM   Personal ;
```

```
SELECT  Ort
FROM    Personal
GROUP BY Ort ;
```

Aggregatfunktionen im GROUP BY

```
SELECT      Ort, COUNT (*) AS Anzahl
 FROM       Personal
GROUP  BY   Ort ;
```

Ort	Anzahl
Regensburg	3
Frankfurt	1
Nürnberg	2
Landshut	1
Augsburg	1
Stuttgart	1

Zu beachten im GROUP BY

- In Select-Klausel gleichzeitig erlaubt:
  - Attributsnamen und Aggregatfunktionen
- Die Aggregatfunktionen wirken auf die gruppierten Tupel
- Group-By-Klausel enthält mindestens:
  - alle Attribute, die in Select-Klausel außerhalb der Aggregat-funktionen vorkommen.
- Fehler:

```
SELECT Name, Ort, Count(*) AS Anzahl
FROM Personal
GROUP BY Ort ;
```

Ein Ort kann viele Mitarbeiter haben.
Welcher wird hier angegeben???

Die HAVING Klausel

- **Where Klausel:** Restriktion vor der Gruppierung
- **Having Klausel:** Restriktion nach der Gruppierung
- **Beispiel:** Alle Wohnorte mit mehreren Mitarbeitern

```
SELECT    Ort, COUNT (*) As Anzahl,
          SUM(12*Gehalt) AS Jahresgehalt,
          12 * MAX(Gehalt) AS MaxJahresgehalt
FROM      Personal
GROUP BY  Ort
HAVING    COUNT(*) > 1 ;
```

SQL | Alle Zeilen abgerufen: 2 in 0 Sekunden

ORT	ANZAHL	JAHRESGEHALT	MAXJAHRESGEHALT
Nürnberg	2	64800	39600
Regensburg	3	115200	57600

HAVING ist Restriktion nach Group By

- **Alternative Lösung zum letzten Beispiel:**

```
SELECT   *
 FROM    ( SELECT  Ort, COUNT (*) As Anzahl,
                   SUM (12*Gehalt) As Jahresgehalt,
                   12 * MAX (Gehalt) As MaxJahresgehalt
             FROM   Personal
             GROUP BY Ort        )     AS Zwischentabelle
WHERE Anzahl > 1 ;
```

Restriktion nach Group By: Entspricht Having

In SQL Server, MySQL: Aliasname ist zwingend

Zwischentabelle liefert alle Orte

Beispiel mit SELECT in FROM Klausel

- **Gesucht:**
  - Mittleres Auftragsvolumen über alle Aufträge

Hauptabfrage bildet daraus den Mittelwert

```
SELECT   'Mittleres Auftragsvolumen: ', AVG( Auftragsvolumen )
 FROM    ( SELECT     SUM( Gesamtpreis ) As Auftragsvolumen
           FROM       Auftragsposten
           GROUP BY Auftrnr              )   AS Auftragspreis;
```

Unterabfrage berechnet das Auftragsvolumen jedes einzelnen Auftrags aus der Summe der Auftragspositionen

Der UNION Operator

- SELECT-Hauptteil UNION SELECT-Hauptteil
- R1 $\cup$ R2
- Beide Hauptteile müssen zueinander kompatibel sein:
  - Gleiche Anzahl der Attribute
  - Die einzelnen Attribute müssen typverträglich sein
- Beispiel (Alle Wohnorte von Kunden und Mitarbeiter):

```
SELECT Ort  FROM Personal
UNION
SELECT Ort  FROM Kunde;
```

$\pi_{Ort}(Personal) \cup \pi_{Ort}(Kunde)$

UNION ALL

- Union liefert eindeutige Ergebnisse:

```
SELECT Ort  FROM Personal
UNION
SELECT Ort  FROM Kunde;
```

9 Mitarbeter

10 Tupel, da doppelte Einträge

6 Kunden

- Union All fasst doppelte Tupel nicht zusammen:

```
SELECT Ort  FROM Personal
UNION ALL
SELECT Ort  FROM Kunde;
```

9+6=15 Tupel

Der Verbund (Join)

Joinausdruck:

Tabellenreferenz
{ [NATURAL] [INNER]
| [NATURAL] { LEFT | RIGHT | FULL } [OUTER]
} JOIN Tabellenreferenz
[ON Bedingung | USING (Spaltenliste)]

- Genau eine der folgenden 3 Angaben muss vorkommen:
  NATURAL | ON | USING
- **Einfaches Beispiel:** R1 NATURAL JOIN R2

Der JOIN Operator

- R1 $\bowtie$ R2 :

  R1 NATURAL INNER JOIN R2

- R1 $\bowtie_{\text{Bedingung}}$ R2 :

  R1 INNER JOIN R2 ON Bedingung

- Weitere Möglichkeit:
- R1 INNER JOIN R2 USING (Spaltenliste)

Natürlicher Verbund: Ein Beispiel

Auftrnr	Datum	Kundnr	Persnr
1	04.01.2013	1	2
2	06.01.2013	3	5
3	07.01.2013	4	2
4	18.01.2013	6	5
5	03.02.2013	1	2

- Verbund über Persnr
- Nur Persnr 2 und 5 bleiben übrig

Persnr	Name	Ort	Vorgesetzt	Gehalt
1	Maria Forster	Regensburg	NULL	4800.00
2	Anna Kraus	Regensburg	1	2300.00
3	Ursula Rank	Frankfurt	6	2700.00
4	Heinz Rolle	Nürnberg	1	3300.00
5	Johanna Köster	Nürnberg	1	2100.00
6	Marianne Lambert	Landshut	NULL	4100.00
7	Thomas Noster	Regensburg	6	2500.00
8	Renate Wolters	Augsburg	1	3300.00
9	Ernst Pach	Stuttgart	6	800.00

Natürlicher Verbund (Auftrag⋈Personal)

```
SELECT  *
 FROM   Auftrag NATURAL INNER JOIN Personal ;
```

AuftrNr	Datum	Kundnr	Persnr	Name	Vorgesetzt	Gehalt	Ort
1	04.01.13	1	2	Anna Kraus	1	3400.00	Regensburg
2	06.01.13	3	5	Joh. Köster	1	3200.00	Nürnberg
3	07.01.13	4	2	Anna Kraus	1	3400.00	Regensburg
4	18.01.13	6	5	Joh. Köster	1	3200.00	Nürnberg
5	06.02.13	1	2	Anna Kraus	1	3400.00	Regensburg

Vergleich der verschiedenen Joins

```
SELECT  *
 FROM   Auftrag NATURAL INNER JOIN Personal ;
```

kein Qualifizieren möglich!
z.B. Auftrag.Datum ist Fehler

```
SELECT  *
 FROM   Auftrag INNER JOIN Personal USING ( Persnr ) ;
```

Nur Qualifizieren der Persnr ist verboten

```
SELECT  Auftrnr, Datum, Kundnr, Personal.*
FROM    Auftrag INNER JOIN Personal
                ON Auftrag.Persnr = Personal.Persnr ;
```

Qualifizieren ist beliebig möglich

Alle drei Befehle liefern die gleiche Ausgabe

Nachbilden des Verbunds

- **Aus der relationalen Algebra:**
- $R1 \bowtie R2 = \pi_{R1.X,R1.Y,R2.Z}(\sigma_{R1.Y=R2.Y}(R1 \times R2))$
  - mit $R1 = R1(X,Y)$, $R2 = R2(Y,Z)$
  - und Y ist gemeinsames Attribut
- $Auftrag \bowtie Personal =$

$$\pi_{\cancel{Auftrag.Persnr}}(\sigma_{Auftrag.Persnr=Personal.Persnr}(Auftrag \times Personal))$$

```
SELECT   Auftrnr, Datum, Kundnr, Personal.*
 FROM    Personal, Auftrag
 WHERE   Personal.Persnr = Auftrag.Persnr ;
```

Gleiche Ausgabe wie die drei vorherigen Befehle

Beispiel zum Verbund (1)

- Gesucht: Alle Mitarbeiter und die Anzahl der Verkäufe
- 1. Versuch:

```
SELECT     Persnr, Name, COUNT (*) As AnzahlAuftrag
 FROM      Personal NATURAL INNER JOIN Auftrag
 GROUP BY  Persnr, Name ;
```

Persnr	Name	AnzahlAuftrag
2	Anna Kraus	3
5	Johanna Köster	2

- Aber: 7 Mitarbeiter fehlen!

Beispiel zum Verbund (2)

- **Versuch 2:**

```
SELECT    Persnr, Name, COUNT(AuftrNr) AS AnzahlAuftrag
 FROM     Personal NATURAL LEFT OUTER JOIN Auftrag
 GROUP BY Persnr, Name ;
```

Count(*) wäre falsch

Persnr	Name	AnzahlAuftrag
2	Anna Kraus	3
5	Johanna Köster	2
1	Maria Forster	0
3	Ursula Rank	0
4	Heinz Rolle	0
6	Marianne Lambert	0
7	Thomas Noster	0
8	Renate Wolters	0
9	Ernst Pach	0

Bei Count(*): 1

Beispiel zum Verbund (3)

- **Lösung ohne Outer Join:**

Count(*) korrekt

```
SELECT      Persnr, Name, COUNT (*) AS AnzahlAuftrag
 FROM       Personal NATURAL INNER JOIN Auftrag
 GROUP BY   Persnr, Name
UNION
SELECT      Persnr, Name, 0
 FROM       Personal
 WHERE      Persnr NOT IN ( SELECT  Persnr
                            FROM    Auftrag  ) ;
```

Alle anderen haben nichts verkauft

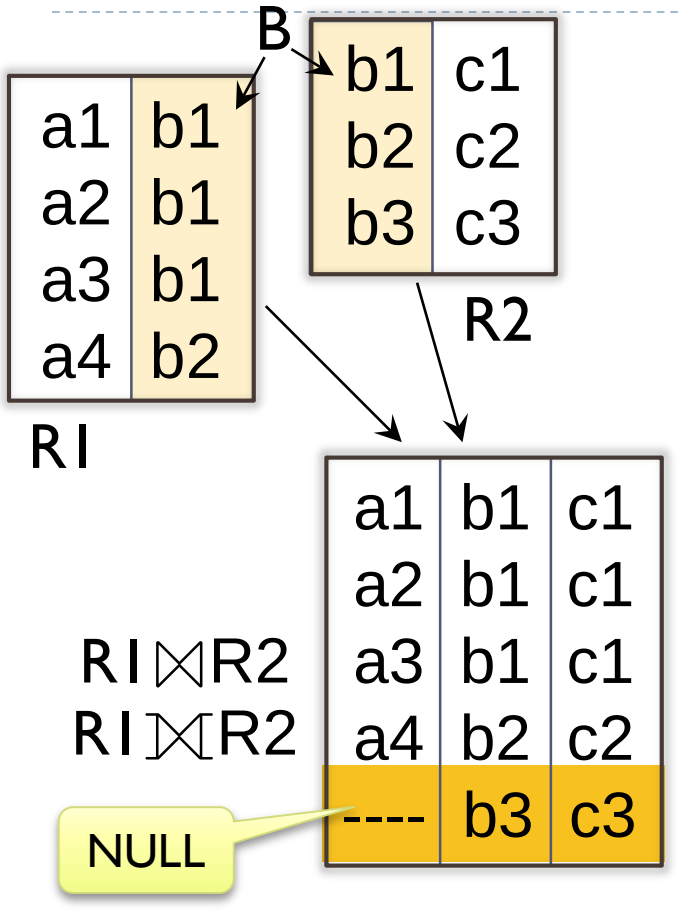
Beispiel zum Verbund (4)

- In der Praxis:
- Der Bezeichner NATURAL sollte vermieden werden, da
  - Fehler im Optimizer von Oracle
  - nicht unterstützt in SQL Server
  - das spätere Hinzufügen von Attributen falsche Ergebnisse liefern kann
- Lösung ohne Bezeichner NATURAL:

```
SELECT    P.Persnr, Name, COUNT(AuftrNr) AS AnzahlAuftrag
FROM      Personal AS P LEFT OUTER JOIN Auftrag As A
                        ON P.Persnr = A.Persnr
GROUP BY P.Persnr, Name ;
```

ohne Natural: Aliasname möglich

Der äußere Verbund



- Der innere Verbund:
  - Enthält alle Elemente, deren Elemente aus B in beiden Relationen vorkommen (Schnitt)
- Der äußere Verbund:
  - Enthält alle weiteren Elemente
  - R1 Left Outer Join R2: + R1
  - R1 Right Outer Join R2: + R2
  - R1 Full Outer Join R2: + R1 + R2
- Nicht vorhandene Werte mit Null auffüllen

Der äußere Verbund: OUTER JOIN

- R1 Left Outer Join R2 $R1 ⟕ R2$
  - R1 Inner Join R2 vereinigt mit allen weiteren Tupel von R1
- R1 Right Outer Join R2 $R1 ⟖ R2$
  - R1 Inner Join R2 vereinigt mit allen weiteren Tupel von R2
- R1 Full Outer Join R2 $R1 ⟗ R2$
  - R1 Left Outer Join R2 Union R1 Right Outer Join R2

- Nicht vorhandene Werte werden mit NULL aufgefüllt.

Die ORDER BY Klausel

- **Sortieren der Tupel eines Select-Befehls**

```
SELECT    Ort, COUNT (*) AS Anzahl
FROM      Personal
GROUP BY  Ort
ORDER BY  Anzahl DESC, Ort ;
```

Absteigendes Sortieren

Nur Namen sind erlaubt …

- **oder:**

```
SELECT    Ort, COUNT (*) AS Anzahl
FROM      Personal
GROUP BY  Ort
ORDER BY  2 DESC, 1 ;
```

… oder Zahlen

Nullwerte sind unterschiedlich

- Ein Vergleich mit Nullwerten liefert immer FALSE !
- Dies ist auch vernünftig. Beispiel:
  - Gesucht: Alle Mitarbeiter, die den gleichen Vorgesetzten haben wie Mitarbeiter 1 (oder 2..9)

```
SELECT   Name, Gehalt, Vorgesetzt
FROM     Personal
WHERE    Vorgesetzt = ( SELECT Vorgesetzt FROM Personal
                        WHERE Persnr = 1 ) ;
```

Unterabfrage liefert NULL

Vergleich liefert false:
Kein Mitarbeiter (auch nicht 6) hat den gleichen Vorgesetzten, und das ist korrekt!

Befehl liefert für 1..9 immer korrektes Ergebnis!

Funktion Coalesce

- Nullwert in einem Ausdruck liefert als Ergebnis NULL
- Nullwerte werden in Aggregatfunktionen nicht mitgezählt
- Falsch:

```
SELECT  SUM (12 * Gehalt +1000 * ( 6- Beurteilung))
                                        AS Jahrespersonalkosten
 FROM   Personal ;
```

Mitarbeiter 9 hat Beurteilung NULL

Also wird auch sein Gehalt verworfen

- Korrekt:

```
SELECT  SUM(12*Gehalt+COALESCE(1000 *( 6- Beurteilung),
                                1000) ) AS Jahrespersonalkosten
 FROM   Personal ;
```

falls NULL …

… dann Ersatzwert

Funktion Coalesce: Komplexes Beispiel

- Beim äußeren Verbund ist COALESCE sehr wichtig!
- Gesucht: Alle Mitarbeiter mit Verkaufsumsatz
- Lösung:

```
SELECT      Persnr, Name,
              COALESCE( SUM(Gesamtpreis), 0) AS Summe
 FROM       Personal NATURAL LEFT OUTER JOIN
              (Auftrag NATURAL INNER JOIN Auftragsposten)
GROUP BY    Persnr, Name ;
```

Vermeidet Null-Ausgaben

… dann Outer Join

Inner Join über Auftrnr …

Arbeitsweise des SELECT Befehls

1	Kreuzprodukt, Verbund: Alle in der Tabellenliste angegebenen Relationen werden miteinander verknüpft.
2	Restriktion: Aus dieser verknüpften Relation werden die Tupel ausgewählt, die die angegebene WHERE-Bedingung erfüllen.
3	Projektion: Mittels der Spaltenauswahlliste werden die angegebenen Spalten ausgewählt.
4	Gruppierung: Jeweils gleiche Tupel werden zusammengefasst. Angegebene Aggregierungen werden durchgeführt.
5	2. Restriktion: Nach der Gruppierung werden die Tupel gewählt, die die angegebene HAVING-Bedingung erfüllen.
6	Vereinigung, Schnitt, Differenz: Alle in Schritt 1 bis 5 erstellten Hauptteile des Select-Befehls werden miteinander verknüpft.
7	Sortierung: Die Ergebnisrelation wird gemäß der Order-By-Klausel sortiert.

Mutationsbefehle in SQL

UPDATE	Ändert bestehende Einträge
INSERT	Fügt neue Tupel (Zeilen) ein
DELETE	Löscht bestehende Tupel (Zeilen)

Der DELETE Befehl

- Syntax:

DELETE FROM Tabellenname [[AS] Aliasname]
[WHERE Bedingung]

- Beispiele:

```
DELETE FROM  Personal
WHERE        Name = 'Ursula Rank' ;
```

Entfernt Mitarbeiterin Ursula Rank

```
DELETE FROM  Personal ;
```

Entfernt alle Mitarbeiter!

Der UPDATE Befehl

- Syntax:

```
UPDATE   Tabellenname  [ [ AS ]  Aliasname ]
 SET          Spalte  =  Spaltenausdruck  [ , ... ]
 [ WHERE  Bedingung ]
```

- Beispiel:

```
UPDATE  Personal
 SET        Gehalt = 1.05 * Gehalt
 WHERE   Gehalt < 3000 ;
```

Zuweisung erfolgt immer zuletzt!

Alle Mitarbeiter mit weniger als 3000 Euro Gehalt erhalten 5% Gehaltserhöhung

Bisheriges Gehalt

Der INSERT Befehl

- **Syntax:**

```
INSERT INTO Tabellenname [ ( Spaltenliste ) ]
  { VALUES ( Auswahlliste ) [, … ]
  | Select-Befehl }
```

- **Beispiel:**

In Oracle: nur eine Auswahlliste möglich

Alle anderen Attribute werden mit NULL gefüllt

```
INSERT INTO Personal (Persnr, Name, GebDatum, Ort)
 VALUES  (10, 'Lars Anger', DATE '1980-01-13', 'Augsburg') ,
         (11, 'Karl Meier', DATE '1983-05-15', 'Darmstadt') ;
```

Cast-Operator DATE

Der INSERT Befehl: SELECT Auswahl

- **Beispiel:**

```
INSERT INTO  Personal (Persnr, Name, GebDatum,
                                        Ort, Vorgesetzt, Gehalt)
 SELECT  10, 'Lars Anger', DATE '1980-01-13',
                                        'Augsburg', Vorgesetzt, Gehalt
 FROM     Personal
WHERE   Persnr = 7 ;
```

In SELECT Klausel sind auch konstante Werte erlaubt

- **Empfehlung:** Select-Auswahl dann, wenn mindestens ein Wert aus Datenbank gelesen werden soll

Transaktionsbetrieb (1)

- Transaktionen mit Mutationsbefehlen enorm wichtig
- Ablauf:

[BEGIN TRANSACTION ;]

Nur in SQL Server
Sonst: automatischer Transaktionsstart

Innerhalb einer Transaktion

COMMIT ; / ROLLBACK ;

Ende einer Transaktion

[BEGIN TRANSACTION ;]

Nur in SQL Server

Innerhalb einer Transaktion

COMMIT ; / ROLLBACK ;

Alle geänderten Daten werden zurückgesetzt

…

Alle geänderten Daten werden dauerhaft gespeichert

Transaktionsbetrieb (2)

- Transaktionsbetrieb ist wichtig für Konsistenz der Daten
- Beispiel: Überweisung von 1000 € von Konto A nach B

```
UPDATE Bank  SET Saldo = Saldo – 1000  WHERE Konto = 'A' ;

UPDATE Bank  SET Saldo = Saldo + 1000  WHERE Konto = 'B' ;

COMMIT ;
```

Hier: inkonsistenter Datenstand

Hier: konsistenter Datenstand

Commit nur, wenn Datenstand konsistent

Vergleich: Relationale Algebra – SQL

Operator	Algebra	SQL-2
Vereinigung	$R1 \cup R2$	SELECT * FROM R1 UNION SELECT * FROM R2
Kreuzprodukt	$R1 \times R2$	SELECT * FROM R1, R2
Restriktion	$\sigma_p(R1)$	SELECT * FROM R1 WHERE p
Projektion	$\pi_{x1,\ldots xn}(R1)$	SELECT x1, ... , xn FROM R1
Differenz	$R1 \setminus R2$	SELECT * FROM R1 EXCEPT SELECT * FROM R2
Verbund	$R1 \bowtie R2$	SELECT * FROM R1 NATURAL INNER JOIN R2
Schnitt	$R1 \cap R2$	SELECT * FROM R1 INTERSECT SELECT * FROM R2

Die Division in SQL

- Gesucht: Alle Lieferanten, die mindestens die gleichen Artikel wie Lieferant 3 liefern.
- Lösung 1: Nachbilden der Division (siehe rel. Algebra)
- Lösung 2: Idee
  - Bestimme alle Artikel des Lieferanten 3: Artikelliste3
  - Bestimme zu jedem Lieferanten alle Artikel aus Artikelliste3
  - Gib alle Lieferanten aus, deren Artikel aus Artikelliste3 genau so groß ist wie die Artikelliste3

Die Division in SQL: Lösung

```
SELECT    Liefnr
 FROM     Lieferung
 WHERE    Anr IN ( SELECT   Anr
                   FROM     Lieferung
                   WHERE    Liefnr = 3  )
 GROUP BY  Liefnr
 HAVING    COUNT(*) = ( SELECT   COUNT(*)
                        FROM     Lieferung
                        WHERE    Liefnr = 3  ) ;
```

Alle Tupel mit gleicher Artikelnr wie Lieferant 3

Nach Lieferanten gruppiert, also Artikel zusammengefasst

Nur die Lieferanten ausgewählt, die gleich viele gleiche Artikel haben

Oracle, SQL Server, MySQL

	Abweichungen zur SQL-Norm
Oracle	Bezeichner As ist in From-Klausel nicht erlaubt Fehlerhafte Ausgabe bei geschachteltem Natural Join Der Operator Except ist durch Minus zu ersetzen Eigene Syntax für die Funktion Substring: Substr(str,pos,anz) Im Insert-Befehl ist nur eine Values-Angabe erlaubt
SQL Server	Ein Select-Befehl in der From-Klausel erfordert einen Aliasnamen Der Natural Join und Join…Using werden nicht unterstützt Eigene Syntax für die Funktion Substring: Substring(str,pos,anz) Der Cast-Operator Date ist nicht erlaubt Eine Transaktion muss mit Begin Transaction eingeleitet werden
MySQL	Ein Select-Befehl in der From-Klausel erfordert einen Aliasnamen Den Aggregatfunktionsnamen darf kein Leerzeichen folgen Beim Natural Inner Join ist der Bezeichner Inner nicht erlaubt Der Full Outer Join wird nicht unterstützt Die Operatoren Except und Intersect werden nicht unterstützt Der Transaktionsbetrieb wird nur in der Engine InnoDB unterstützt