we focus on students

Datenbanken 1

Datenbankabfragen (Teil 2)

Fachhochschule Dortmund
University of Applied Sciences

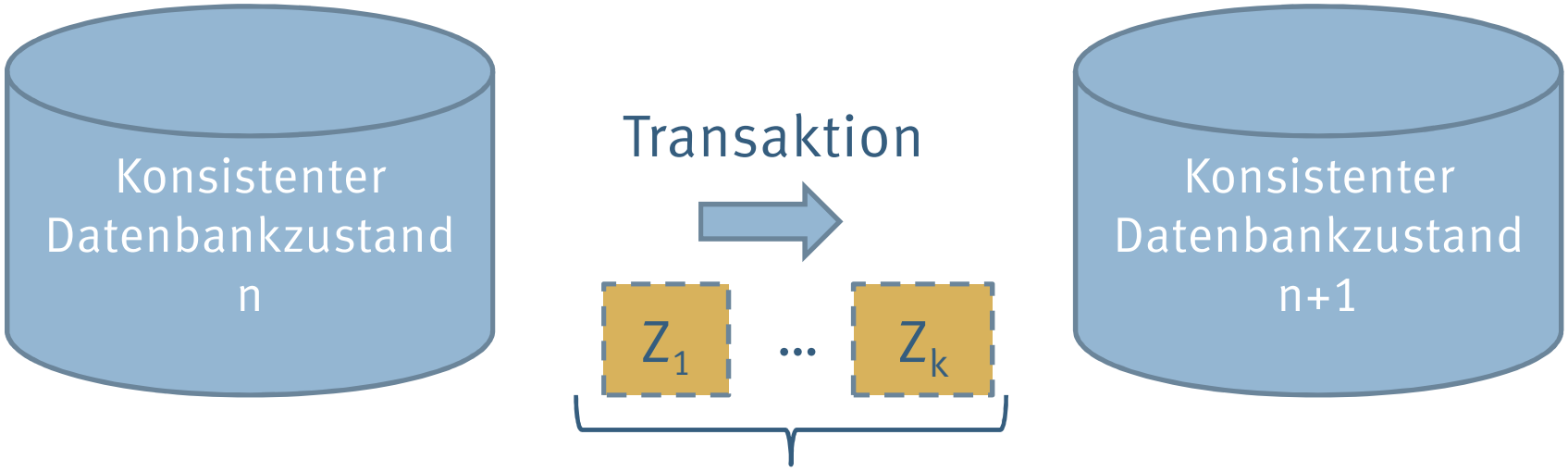
Inhaltsübersicht

Transaktionsbegriff

Eine **Transaktion** ist eine inhaltlich zusammenhängende Menge von Datenbankoperationen, die ganz oder gar nicht ausgeführt werden.

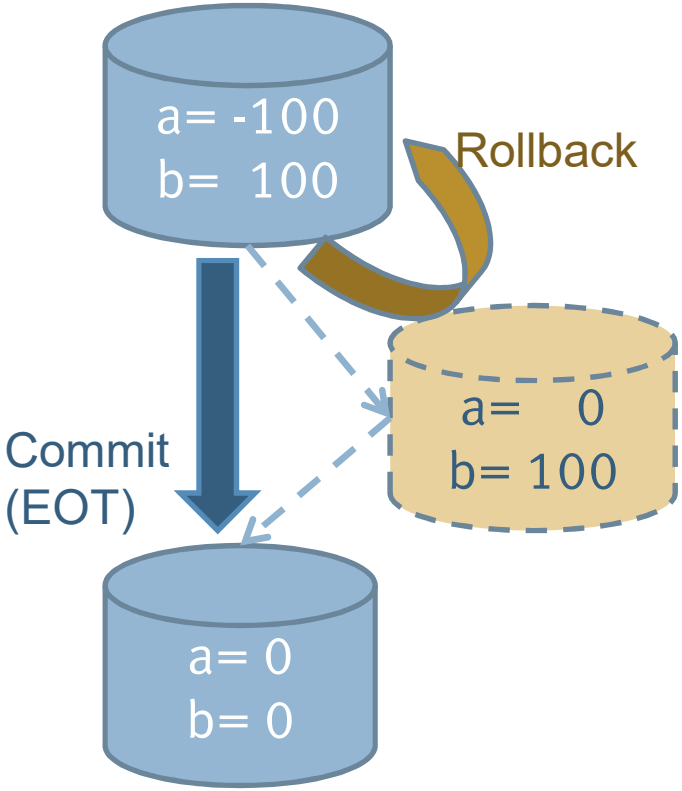
Eine **Transaktion** überführt einen konsistenten Datenbankzustand in einen wiederum konsistenten Datenbankzustand.

Der initiale Zustand (Zustand n=0) der Datenbank ist konsistent.



Commit und Rollback

Wird zum Transaktionsende ein korrekten Datenbankzustand erreicht, d.h. sind die Integritätsbedingungen erfüllt, so wird ein Commit ausgeführt. Bei einem Commit werden die Änderungen der Transaktion gesichert.
Wird kein korrekter Datenbankzustand erreicht oder ist ein anderer Fehlerfall oder einen Transaktionsabbruch aufgetreten, dann wird ein Rollback ausgeführt. Bei einem Rollback erfolgt die Rücksetzung der DB in einen vorherigen konsistenten Zustand.



Begin of Transaction (BOT)
Lese Kontostand a=-100
Lese Zahlungseingang b=100
Schreibe Kontostand a:=a +b
Schreibe Zahlungseingang b:=0
End of Transaction (EOT)

Commit *Rollback*

Regeln für Transaktionen

Atomar
- Entweder ganz oder gar nicht ausgeführt

Consistent
- Ausführung ist Konsistenzerhaltend (Korrekter Inhalt der DB bei EOT)

Isoliert
- Transaktionen laufen isoliert voneinander ab (Simulation des Ein-Benutzer-Betriebs)

Dauerhaft
- Ergebnis einer Transaktion wird dauerhaft gespeichert und gesichert

Inhaltsübersicht

Einfacher Select-Befehl

Was wird gesucht?

In welchen Tabellen?

Auswahlbedingungen?

Sortierung?

SELECT ‹Spalte$_1$›, ..., ‹Spalte$_n$›

Projektion (Festlegung der Ausgabespalten)

FROM ‹Tabelle$_1$›, ..., ‹Tabelle$_m$›

Join (Angabe der Tabellen und Verbundbedingung)

WHERE ‹Bedingung›

Selektionsbedingung (Auswahl der Tupel)
– optional

ORDER BY ‹Attribut-Liste$_s$›

Sortierreihenfolge der Tupel in der Ergebnistabelle
– optional

Beispiel

```
SELECT ANummer, Lagerbestand
FROM   Lager
ORDER BY ANummer DESC
```

Syntax der SELECT-Anweisung

Was wird gesucht?

SELECT ‹Spalte$_1$›, ..., ‹Spalte$_n$›

Projektion (Festlegung der Ausgabespalten)

In welchen Tabellen?

FROM ‹Tabelle$_1$›, ..., **‹Tabelle$_m$›**

Join (Angabe der Tabellen und Verbundbedingung)

Auswahlbedingungen?

WHERE ‹Bedingung›

Selektionsbedingung (Auswahl der Tupel)
– optional

Gruppierung erforderlich?

GROUP BY ‹Spalte$_1$›, ..., ‹Spalte$_n$›

Gruppenbildung mit gleichen Werten
– optional

Gruppierungsbedingung?

HAVING ‹Bedingung›

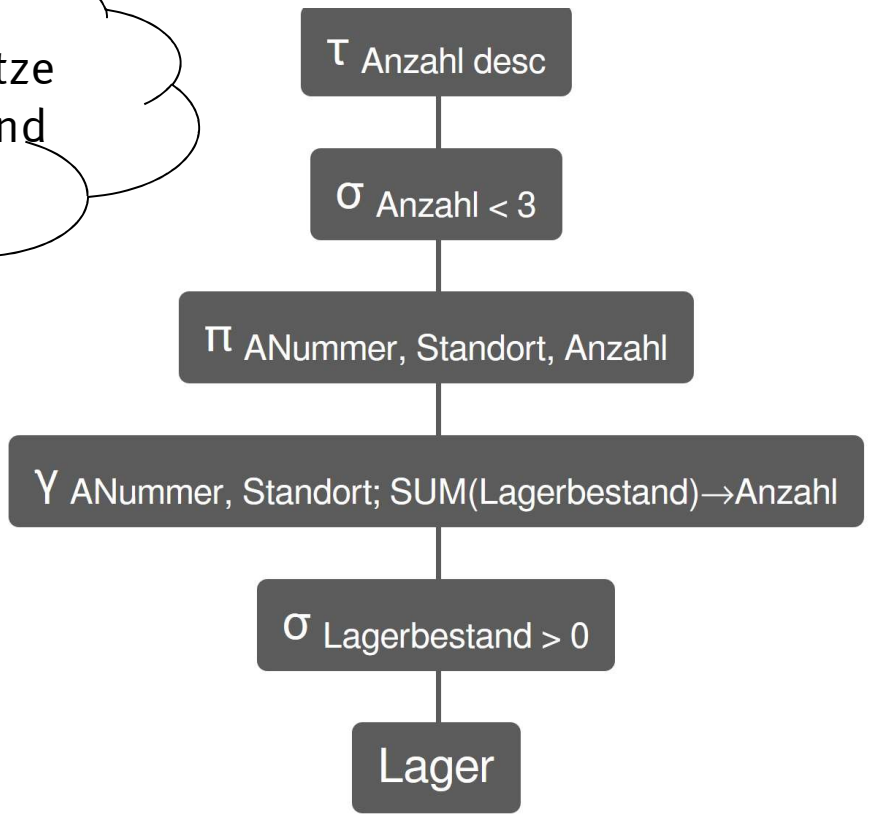
Selektion von Gruppen
– optional unter group by

Sortierung?

ORDER BY ‹Attribut-Liste$_S$›

Sortierreihenfolge der Tupel in der Ergebnistabelle
– optional

Auswahlbedingung für Gruppen



τ Anzahl desc σ Anzahl < 3 π ANummer, Standort, Anzahl γ ANummer, Standort; SUM(Lagerbestand)→Anzahl (σ Lagerbestand > 0 Lager)

Lager.ANummer	Lager.Standort	Anzahl
4811	INF	2
4814	MED	1

Vorsicht Falle!

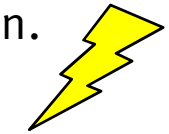
FEHLERHAFT

```
1  SELECT ANummer,Standort, SUM(Lagerbestand) AS Bestand
2  FROM Lager
3  WHERE Lagerbestand >0 AND SUM(Lagerbestand) <3
4  GROUP BY ANummer, Standort
5  ORDER BY Bestand DESC;
```

Weshalb geht das so nicht?

1. Die Gruppierung erfolgt immer nachdem die Auswahlbedingung angewendet wurde.
   → Die Ausführung der Gruppierung erfolgt nach Anwendung der Auswahlbedingung (WHERE-Klausel) der ausgeführt wurde.

2. Damit die WHERE-Bedingung ausgewertet werden kann, muss die Summe über die Lagerbestände vorher berechnet worden sein.
   → D.h. die Gruppierung muss zuerst erfolgen.



Die HAVING-Klausel

Auswahlbedingungen, welche sich auf das Ergebnis einer Gruppierung beziehen, müssen in der HAVING-Klausel angegeben werden.

SQL-Anfrage mit bedingter Gruppierung

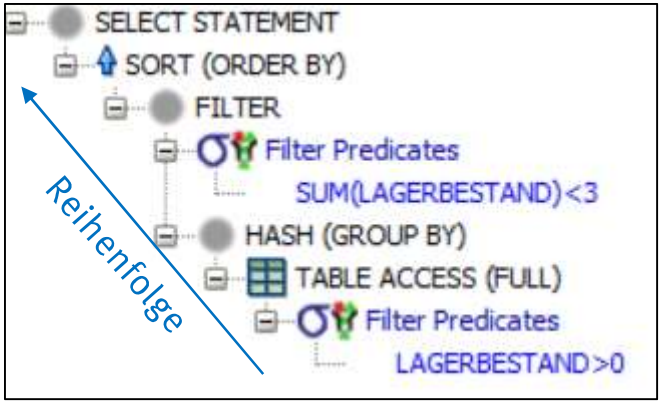
```
1  SELECT ANummer,Standort, SUM(Lagerbestand) AS Bestand
2  FROM Lager
3  WHERE Lagerbestand >0
4  GROUP BY ANummer, Standort
5  HAVING SUM(Lagerbestand) <3
6  ORDER BY Bestand DESC;
```

Abfrageergebnis

SQL | Alle Zeilen abgerufen:2 in 0,008 Sekunden

	ANUMMER	STANDORT	BESTAND
1	4811	INF	2
2	4814	MED	1

Interne Auswertungsreihenfolge



Inhaltsübersicht

Syntax der SELECT-Anweisung

Was wird gesucht?

SELECT ‹Spalte$_1$›, ..., ‹Spalte$_n$›

Projektion (Festlegung der Ausgabespalten)

In welchen Tabellen?

FROM ‹Tabelle$_1$›, ..., **‹Tabelle$_m$›**

Join (Angabe der Tabellen und Verbundbedingung)

Auswahlbedingungen?

WHERE ‹Bedingung›

Selektionsbedingung (Auswahl der Tupel)
– optional

Gruppierung erforderlich?

GROUP BY ‹Spalte$_1$›, ..., ‹Spalte$_n$›

Gruppenbildung mit gleichen Werten
– optional

Gruppierungsbedingung?

HAVING ‹Bedingung›

Selektion von Gruppen
– optional unter group by

Sortierung?

ORDER BY ‹Attribut-Liste$_S$›

Sortierreihenfolge der Tupel in der Ergebnistabelle
– optional

Beispiel: Kartesisches Produkt

Beim Cross-Join wird lediglich das Kreuzprodukt der miteinander verbundenen Tabellen (hier: Artikel und Lager) gebildet.

```
1  SELECT Artikelnummer, Artikelname, ANummer, Lagerbestand
2  FROM Artikel, Lager
3  WHERE Lagerbestand=0;
```

Skriptausgabe | Abfrageergebnis

SQL | Alle Zeilen abgerufen:8 in 0,003 Sekunden

	ARTIKELNUMMER	ARTIKELNAME	ANUMMER	LAGERBESTAND
1	4812	Datenbanksysteme	4813	0
2	4811	Datenbanksysteme	4813	0
3	4813	Märchen von Beedle dem Barden	4813	0
4	4814	Anatomie-interaktiv	4813	0
5	4815	Anatomie	4813	0
6	4816	Anatomie-Atlas	4813	0
7	4820	Datenbank-Skript	4813	0
8	4810	Harry Potter Band 20	4813	0

Spalten aus Artikel

Spalten aus Lager

Beispiel INNER-JOIN

Der **Inner-Join** liefert alle Tupel, welche die Verbundbedingung erfüllen.
Es können nur Tupel gefunden werden, die in **beiden** Tabellen vorkommen.


```
SELECT Artikelname, Autor, Ausgabe
FROM    Artikel a, Lager l
WHERE Lagerbestand=0
AND a.Artikelnummer = l.ANummer
```

semantisch äquivalent

```
SELECT Artikelname, Autor, Ausgabe
FROM    Artikel a JOIN Lager l
      ON a.Artikelnummer = l.ANummer
WHERE Lagerbestand=0
```

INNER-JOIN Arten

- EQUI-JOIN: Überprüft auf Gleichheit von Attributen

```
SELECT a.Artikelnummer, Artikelname, Autor
FROM   Artikel a JOIN Warenkorb w
          ON a.Artikelnummer = w.Artikelnummer
```

- Vereinfachte Schreibweise

```
SELECT w.Artikelnummer, Artikelname, Autor
FROM   Artikel a JOIN Warenkorb w
          USING (Artikelnummer)
```

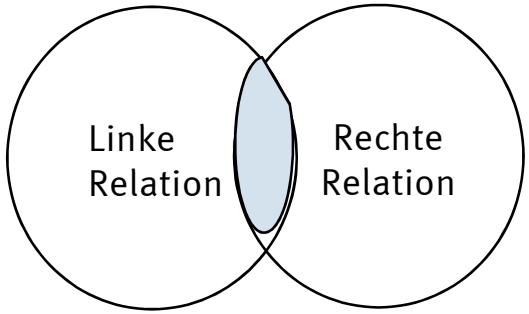
- NATURAL JOIN: EQUI-JOIN über gleichbenannte Attribute

```
SELECT w.Artikelnummer, Artikelname, Autor
FROM   Artikel a Natural JOIN Warenkorb w
```

Join-Arten

Inner Join

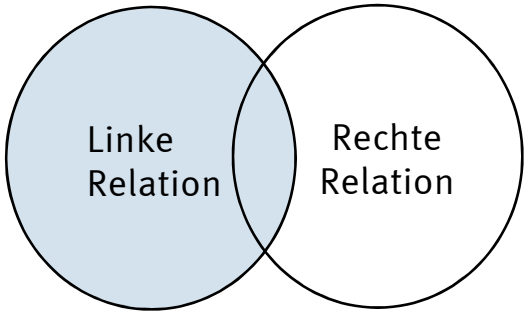
$\bowtie$



Tupel wird nur übernommen, wenn die Verbundbedingung erfüllt wird.

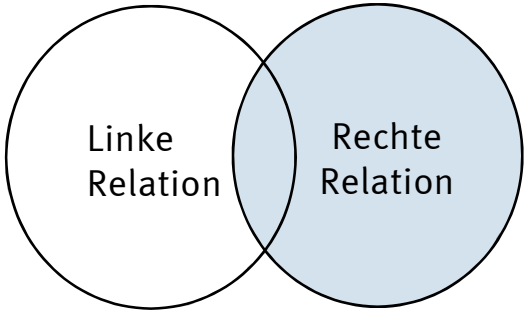
Left Outer Join

$\bowtie$



Right Outer Join

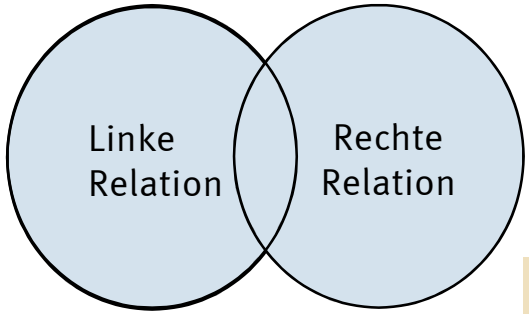
$\bowtie$



Alle Tupel der linken bzw. rechten Relation werden übernommen.

Full Outer Join

$\bowtie$

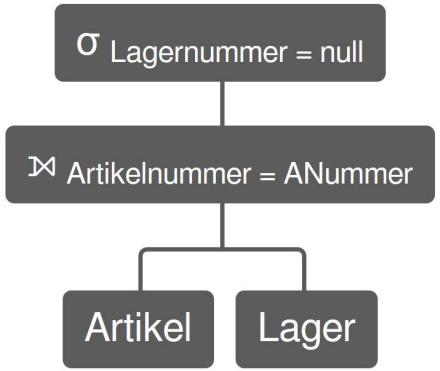


Alle Tupel werden übernommen.

Anwendung des LEFT OUTER JOINs

Beispiel: Ermittle alle Artikel, die keinen Lagerplatz besitzen.

Artikel-nummer	Artikelname	Preis	Ausgabe	Lager-nummer	Standort	ANummer	Lager-bestand
4830	Harry_Potter_20	NULL	NULL	~~27135~~ NULL	~~R235~~ NULL	~~4812~~ NULL	~~0~~ NULL



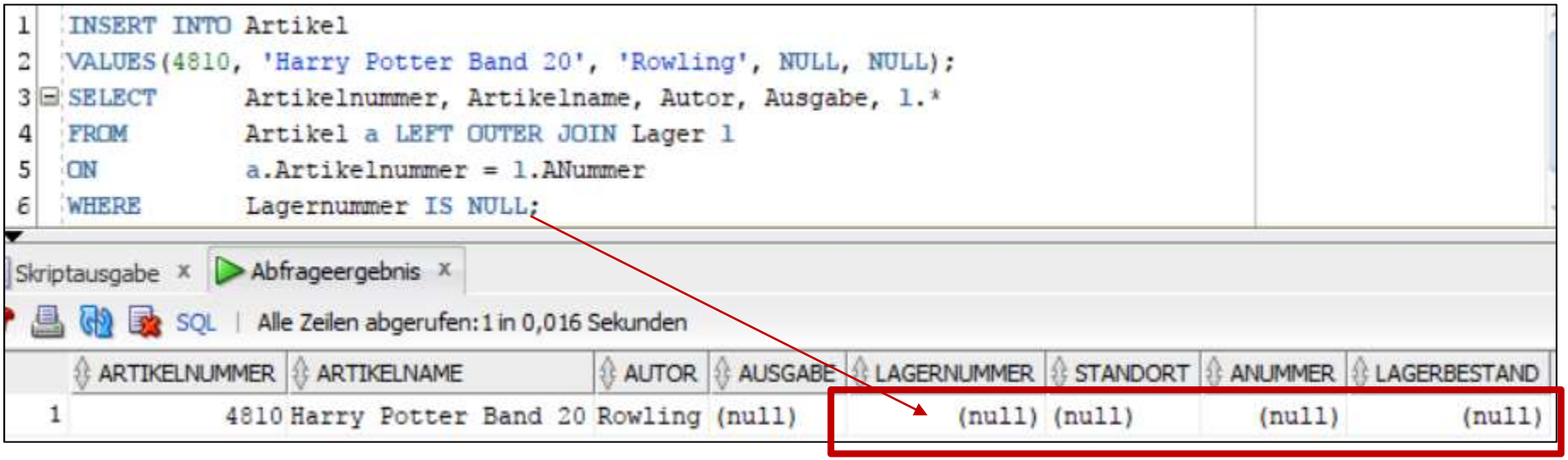
$\sigma_{\text{Lagernummer = null}}(\text{Artikel} \bowtie_{\text{Artikelnummer = ANummer}} \text{Lager})$

Artikel.Artikelnummer	Artikel.Artikelname	Artikel.Autor	Artikel.Preis	Artikel.Ausgabe	Lager.Lagernummer	Lage
4830	Harry_Potter_20	Rowling	*null*	*null*	*null*	*null*

Anwendung des LEFT OUTER JOINs

Bei dem LEFT-OUTER JOIN (LEFT JOIN) werden die Attribute der beim Join links stehenden Relation zu NULL ergänzt, die nicht der Verbundbedingung genügen. Beim RIGHT-OUTER JOIN (RIGHT JOIN) ist es genau anders herum.

Beispiel: Ermittle alle Artikel, die keinen Lagerplatz besitzen.



```
1  INSERT INTO Artikel
2  VALUES(4810, 'Harry Potter Band 20', 'Rowling', NULL, NULL);
3  SELECT      Artikelnummer, Artikelname, Autor, Ausgabe, l.*
4  FROM        Artikel a LEFT OUTER JOIN Lager l
5  ON          a.Artikelnummer = l.ANummer
6  WHERE       Lagernummer IS NULL;
```

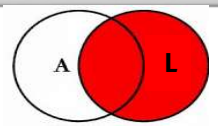
Skriptausgabe | Abfrageergebnis

SQL | Alle Zeilen abgerufen: 1 in 0,016 Sekunden

	ARTIKELNUMMER	ARTIKELNAME	AUTOR	AUSGABE	LAGERNUMMER	STANDORT	ANUMMER	LAGERBESTAND
1	4810	Harry Potter Band 20	Rowling	(null)	(null)	(null)	(null)	(null)

Beispiel: OUTER-JOIN

RIGHT OUTER JOIN ON Artikel.Artikelnummer=Lager.Anummer

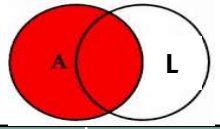


Artikel-nummer	Artikel name	Preis	Ausgabe	Lager-nummer	Standort	ANummer	Lager-bestand
4812	Basiswi	19,95	gebunden	27135	R235	4812	0
4813	Das End	10,00	broschier	27432	R371	4813	0
NULL	NULL	NULL	NULL	27595	INF	NULL	NULL

unbekannter Artikel am leeren Lagerplatz

leerer Lagerplatz

LEFT OUTER JOIN ON Artikel.Artikelnummer=Lager.Anummer



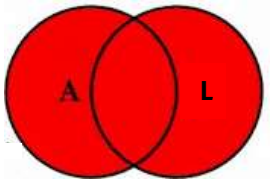
Artikel-nummer	Artikelname	Preis	Ausgabe	Lager-nummer	Standort	ANummer	Lager-bestand
4812	Basiswi	19,95	gebunden	27135	R235	4812	0
4813	Das End	10,00	broschier	27432	R371	4813	0
9999	Gutsch	0,00	broschiert	NULL	NULL	NULL	NULL

Artikel ohne Lagerplatz

unbekannter Lagerplatz

FULL OUTER-JOIN

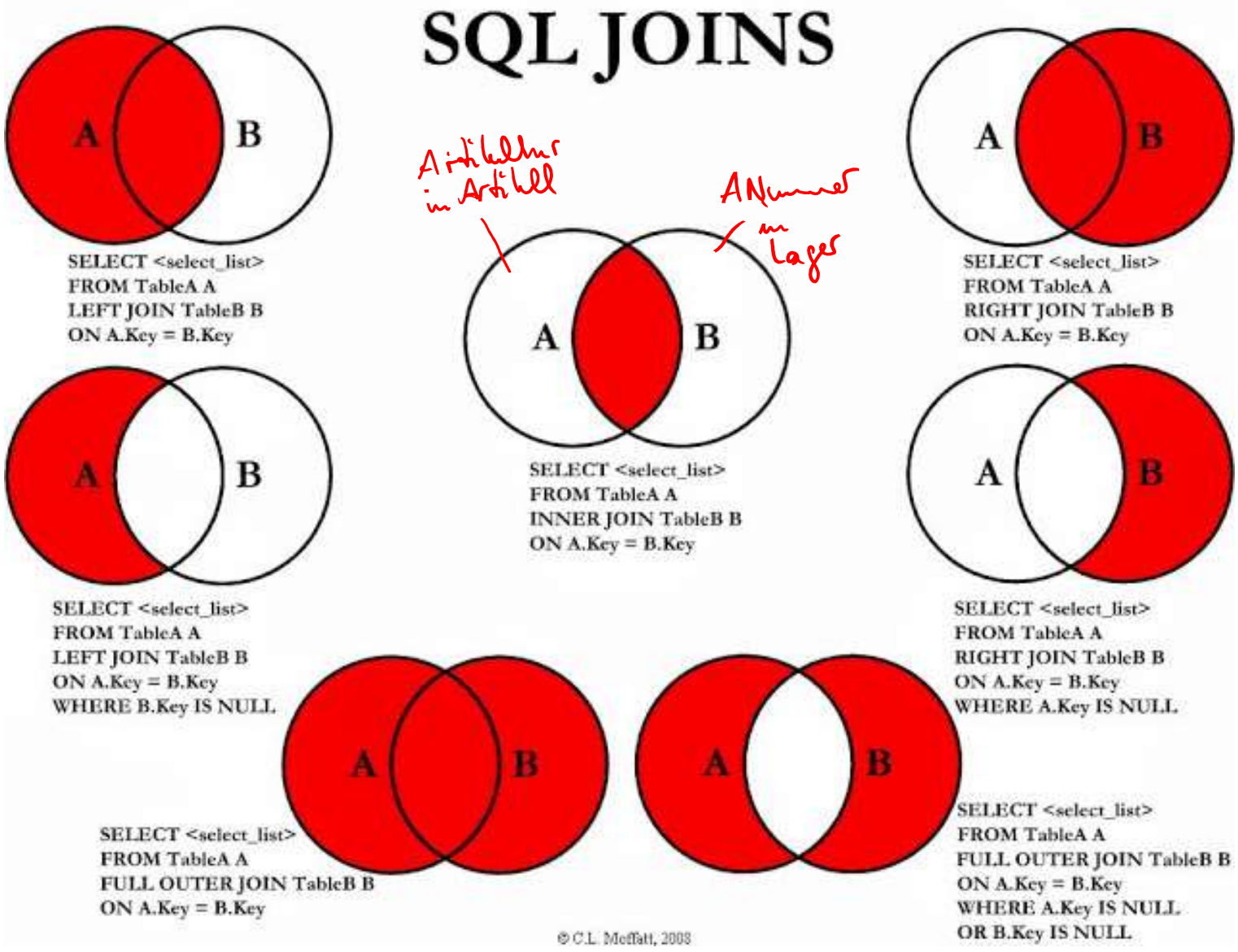
Der FULL Outer-Join liefert alle Tupel des INNER JOINS sowie die durch den RIGHT- und den LEFT-OUTER JOIN zusätzlich gelieferten Tupel.




```
SELECT * FROM Artikel FULL OUTER JOIN Lager ON Artikel.Artikelnummer = Lager.ANummer;
```

Artikel-nummer	Artikelname	Preis	Ausgabe	Lager-nummer	Standort	ANummer	Lager-bestand
Ergebnis des INNER JOINS							
4812	Basiswi	19,95	gebunden	27135	R235	4812	0
4813	Das End	10,00	broschier	27432	R371	4813	0
Zusätzliche Tupel aus dem RIGHT OUTER JOIN							
NULL	NULL	NULL	NULL	27595	INF	NULL	NULL
Zusätzliche Tupel aus dem LEFT OUTER JOIN							
9999	Gutsch	0,00	broschiert	NULL	NULL	NULL	NULL

Übersicht Verbundoperationen



http://javarevisited.blogspot.de/2012/11/how-to-join-three-tables-in-sql-query-mysql-sqlserver.html#more

Inhaltsübersicht

Alternative Vorgehensweisen

- 1. Vorgehensweise: Verbundoperation (Join)
  - Idee: Führe die Inhalte der Tabellen Lager und Artikel zu einer Tabelle im Hauptspeicher zusammen und werte dann die Abfrage aus.

Lager-nummer	Standort	ANummer	Lager-bestand	Artikel-nummer	Artikel-name	Preis	Ausgabe

- 2. Vorgehensweise: Unterabfrage (Subquery)
  - Idee: Ergänze die eigentliche Abfrage um eine Unterabfrage, welche zu jeder ANummer den zugehörenden Artikelnamen aus der Tabelle Artikel liefert.

Hauptabfrage

Lager-nummer	Standort	ANummer	Lager-bestand	Artikelname
12345	FAN	4813	0	

Unterabfrage (Subquery)

Artikel

4813	Märchen von Beedle dem Barden	Rowling	12.90	broschiert

Beispiel 1- Alles in einem Schritt

Gebundene Unterabfrage (Subquery):
- Schachtelung der Anfragen mit Bezug auf äußere Variable
- Eindeutigkeit der Variablenbezüge wird durch Aliasnamen hergestellt

Verschachtelung von SQL-Anfragen

```
SELECT    ANummer,
          (SELECT Artikelname
          FROM    Artikel
          WHERE   Artikelnummer = 4813   ℓ.ANummer
          ),
          Lagerbestand
FROM      Lager  ℓ
WHERE     ANummer =4813
```

Das Attribut ℓ.ANummer ist über den Alias ℓ eine gebundene Variable. Die innere Abfrage referenziert einen Wert der äußeren Abfrage des jeweils bearbeiteten Tupels. Die Abarbeitung der SQL-Abfrage erfolgt von Außen nach innen.

Beispiel – Einwertige gebundene Unterabfrage

Diese Unterabfrage liefert genau ein Tupel zurück.

1. Schritt: Anzahl von Standorten pro Artikel

```
SELECT COUNT(Standort) FROM Lager
WHERE  ANummer=4820
```

2. Schritt: Lagerplätze ermitteln

```
SELECT * FROM Lager l
WHERE (‹Anzahl Standorte›) › 1
```

3. Schritt: 1. Abfrage in 2. Abfrage einschachteln

```
SELECT * FROM Lager l
WHERE (SELECT COUNT(Standort) FROM Lager
        WHERE ANummer=l.ANummer)
      › 1
```

Beispiel – Mehrwertige Unterabfrage

Diese Unterabfrage liefert mehrere Tupel zurück.

1. Schritt: Artikel mit mehr als einem Standort

```
SELECT ANummer
FROM Lager
GROUP BY ANummer
HAVING COUNT(Standort)›1
```

2. Schritt: zugehörende Lagerplätze ermitteln

```
SELECT * FROM Lager
WHERE ANummer IN  ( ‹Artikelliste› )
```

3. Schritt: 1. Abfrage in 2. Abfrage einschachteln

```
SELECT * FROM Lager
WHERE ANUMMER
        IN (SELECT ANummer
            FROM Lager
            GROUP BY ANummer
            HAVING COUNT(Standort)›1)
```

Vorsicht Falle

Wo stecken die Fehler?

```
SELECT DISTINCT Artikelbezeichnung, Preis
FROM        Artikel
WHERE     (Artikelbezeichnung , Preis)
IN  ( SELECT Artikelbezeichnung , Preis
        FROM   Artikel
        WHERE Preis < 10
        AND    Artikelbezeichnung like '%Datenbank%'  )
```

EXISTS, ALL oder ANY?

Operator	Erläuterung
EXISTS 1	Liefert True, wenn die Ergebnismenge der Unterabfrage nicht leer ist, also Tupel vorhanden sind (existieren).
ALL 2	Liefert True, wenn die Bedingung für **alle** Tupel der Unterabfrage erfüllt ist.
ANY 3	Liefert True, wenn die Bedingung für **mindestens ein** Tupel der Unterabfrage erfüllt ist.

Beispiele

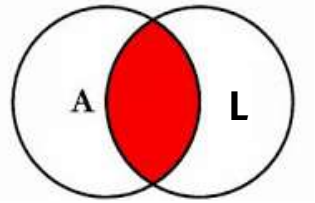
Finden Sie alle Artikel, ...	**Operator**
... die an einem Standort mehr als viermal vorhanden sind.	ANY
... die an jedem Standort mehr als viermal vorhanden sind.	ALL
... die keinen Lagerplatz besitzen	NOT EXISTS

1 Vergleichsoperator - EXISTS

Der EXISTS-Operator überprüft mittels der Exist-Funktion **Exists (‹menge›)**, ob die Ergebnismenge *[einer Subquery]* Elemente enthält.
Die Attributwerte werden ignoriert und nur die Tupel gezählt.

Beispiele:

„Finde alle Artikel, die nicht auf Lager sind."




```
SELECT Artikelnummer FROM  Artikel a
WHERE EXISTS       (SELECT Lagernummer FROM Lager
                    WHERE Lagerbestand = 0 AND ANummer= a.Artikelnummer)
```

```
4813 Märchen von Beedle dem
```

„Finde alle Artikel, die keinen Lagerplatz besitzen."

```
SELECT Artikelnummer FROM  Artikel a
WHERE NOT EXISTS (SELECT Lagernummer FROM Lager
                  WHERE ANummer= a.Artikelnummer)
```

```
4810 Harry Potter Band 20
```

Vergleichsoperator - ALL

Der Vergleichsoperator All liefert true, wenn der Vergleich für **all**e Elemente der Ergebnismenge true liefert. Dies ist insbesondere der Fall, wenn die Ergebnismenge **leer** ist.

Syntax `<Vergleichsoperator>` **ALL**

Beispiel „Finde alle Artikel, die an jedem Standort mehr als viermal vorhanden sind.“

```
SELECT Artikelnummer, Artikelname
FROM Artikel a
WHERE 4 < ALL(SELECT Lagerbestand FROM Lager
              WHERE  ANummer=a.Artikelnummer)
```

Vergleich

ARTIKELNUMMER	ARTIKELNAME
4812	Datenbanksysteme
4816	Anatomie-Atlas
4810	Harry Potter Band 20

Weshalb taucht auch Harry Potter Band 20 auf?

Vergleichsoperator - ALL

Ist die Ergebnismenge der Unterabfrage leer, dann liefert ein Vergleich mit dem ALL-Operator true. Diese Ergebnisse können durch eine zusätzliche Existenzbedingung ausgeschlossen werden.

Beispiel „Finde alle Artikel, die an jedem Standort mehr als viermal vorhanden sind [und hierfür auch Lagerplätze existieren].“

```
SELECT Artikelnummer, Artikelname
FROM Artikel a
WHERE 4 < ALL( SELECT Lagerbestand FROM Lager
                WHERE  ANummer=a.Artikelnummer)
AND    EXISTS( SELECT Lagerbestand FROM Lager
                WHERE  ANummer=a.Artikelnummer)
```

4812	Datenbanksysteme
4816	Anatomie-Atlas

Vergleichsoperator - ANY

Der Vergleichsoperator ANY liefert true, wenn der Vergleich für irgendein (any) Tupel der Ergebnismenge true.

Syntax: ‹Vergleichsoperator› ANY

Beispiel: „Finde alle Artikel, die an [mindestens] einem Standort mehr als viermal vorhanden sind.“

```
SELECT Artikelnummer, Artikelname
FROM Artikel a
WHERE 4 < ANY ( SELECT Lagerbestand FROM Lager
                WHERE ANummer=a.Artikelnummer)
```

4812	Datenbanksysteme
4816	Anatomie-Atlas
4820	Datenbank-Skript

Weshalb taucht Harry Potter Band 20 NICHT auf?

Vorsicht Falle!

Der Vergleich mit einer leeren Menge liefert bei ALL true und bei ANY false.
ALL und ANY liefern daher unterschiedliche Ergebnisse bei leeren Unterabfragen.

ANY

```
SELECT * from Artikel
WHERE 0 < ANY(SELECT Preis from Artikel where Preis <0);
```

iptausgabe x | Abfrageergebnis x

SQL | Alle Zeilen abgerufen:0 in 0,003 Sekunden

ARTIKELN...	ARTIKELN...	AUTOR	PREIS	AUSGABE	KATEGOR...

ALL

```
SELECT * from Artikel
WHERE 0 < ALL(SELECT Preis from Artikel where Preis <0);
```

ausgabe x | Abfrageergebnis x

SQL | Alle Zeilen abgerufen:9 in 0,005 Sekunden

ARTIKELNUMMER	ARTIKELNAME	AUTOR	PREIS	AUSGABE	KATEG
4812	Datenbanksysteme	Elmasri	29,95	gebunden	FBU
4811	Datenbanksysteme	Kemper	28,9	broschiert	DB
4813	Märchen von Beedle dem Barden	Rowling	12,9	broschiert	FAN

ALL liefert alle Tupel der Tabelle Artikel

Hintergrund: Query-Optimierer

Durch den Query-Optimierer werden der ALL- und der ANY-Operator intern umgeformt und auf eine Abfrage mit dem Exists-Operator zurückgeführt.

ANY

0‹ANY (SELECT Preis FROM Artikel WHERE Preis ‹0)

⇩

EXISTS (SELECT Preis FROM Artikel WHERE Preis ‹0 AND 0‹Preis)

leere Menge

⇨ ✗

ALL

0‹ALL (SELECT Preis FROM Artikel WHERE Preis ‹0)

⇩

NOT (0 ›= ANY (SELECT Preis FROM Artikel WHERE Preis ‹0))

⇩

NOT EXISTS (SELECT Preis FROM Artikel WHERE Preis ‹0
AND 0›=Preis)

leere Menge

⇨ ✓

Unterabfragen – Verwendung und Ergebnismengen

Die Unterabfrage liefert

… einen einzelnen Wert

- **SELECT**-Clause
  - als Spaltenangabe
- **WHERE**-Clause
  - einer SELECT-Abfrage
  - einer DELETE-Anweisung
  - einer UPDATE-Anweisung
- **SET**-Clause
  - einer UPDATE-Anweisung

```
UPDATE Kunde
SET Ort = (SELECT Ort FROM Kunde
           WHERE Kundennummer=2310)
WHERE Kundennummer=8536
```

… eine Menge von Tupeln

- **FROM-Clause**
- **WHERE**-Clause
  - einer SELECT-Abfrage
  - einer DELETE-Anweisung
  - einer UPDATE-Anweisung

Inhaltsübersicht

Wochenaufgaben

- Inhalte
  - Lernmodul und Praktikum Datenbankabfragen (Teil 2)
  - Wochentest Datenbankabfragen (Teil 2)
  - Reservierung eines Abnahmetermins für die Projektaufgabe
    → Terminbuchung in ILIAS (Ordner Projektaufgabe)

Vielen Dank
für Ihre aktive Mitarbeit