

# SMÜ, am 04.10.2022

Montag, 5. September 2022 13:16

INSY Name: Felix Schneider htlkrems Bautechnik & IT

Schreibe zu folgendem Schema die entsprechende Abfrage.

**Ergebnis**

PROJECT_ID	LEGAL_FOUNDATION	NUMBER_OF_PROJECTS
1	P_27	1
2	P_27	2
3	P_27	2
4	P_27	3
5	P_27	3
6	P_27	3
7	P_27	3
8	P_27	3
9	P_27	3
10	P_27	3
11	P_27	3
12	P_27	3
13	P_27	3
14	P_27	3
15	P_27	3
16	P_27	3
17	P_27	3
18	P_27	3
19	P_27	3
20	P_27	3
21	P_27	3
22	P_27	3
23	P_27	3
24	P_27	3
25	P_27	3
26	P_27	3
27	P_27	3
28	P_27	3
29	P_27	3
30	P_27	3
31	P_27	3

**Aufgabe:**  
Geben Sie alle Research Funding Projekte, deren Legal Foundation sowie die Anzahl an dazugehörigen Subprojekten aus. Filtern Sie auf die Legal Foundation P\_27. Ordnen Sie diese absteigend nach der Anzahl der Subprojekte.

Ergebnis: Sie Grafik oben.

**Lösung:**

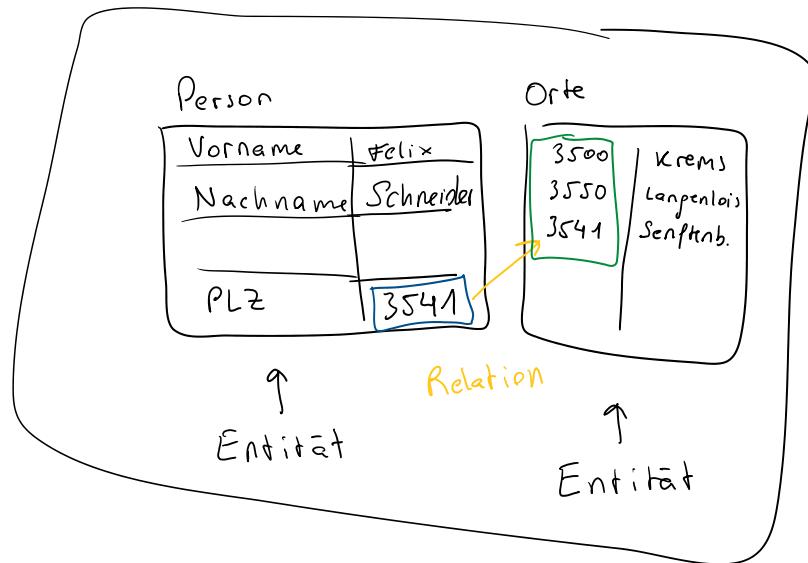
```

SELECT P.PROJECT_ID, P.LEGAL_FOUNDATION,
       (SELECT COUNT(S.PROJECT_ID) FROM SUBPROJECTS S
        GROUP BY S.PROJECT_ID) AS COUNTSUBPROJECT
    ↑
    wie Project_ID gibts n. sleek where fehlt
  FROM PROJECTS_BT P
  RIGHT JOIN RESEARCH_FUNDING_PROJECTS RFP ON P.PROJECT_ID
  = RFP.PROJECT_ID
  ↴LEFT JOIN SUBPROJECTS S ON P.PROJECT_ID = S.PROJECT_ID
  WHERE P.LEGAL_FOUNDATION = "P_27"
  Order by ... diese
  SELECT (SELECT COUNT(S.PROJECT_ID) FROM
  SUBPROJECTS S GROUP BY S.PROJECT_ID);
  
```

# SQL / Not-only-SQL (Structured Query Language)

## Entities

### ERD - Entities Relationship Diagrams



Primärschlüssel

→ ERD mit 2 Entitäten

Fremdschlüssel

Vorteile:

Daten ändern sehr leicht

Nachteile:

Beim Lesen muss der Weg der Relationen durch die verschiedenen Entitäten verfolgt werden.



<p><b>Informationssysteme</b> • 1.463</p> <hr/> <p><b>Informationssysteme</b></p>	<p style="text-align: right;"><b>Wertpreis pro Vorlesung: € 00,- bis 20,-</b></p>
<p><b>Dipl.-Ing. Mag. Paul Pucherle BSc.**</b></p> <p>1. SEM, TU Wien, Universitätsteilnehmer, 11, 1040, Wien, Austria</p>	
<p><b>Abstract:</b> Ein Informationssystem ist ein wissensreiches System, das die Abhebung von Informationssignale aus techn. und soz. Umwelt erlaubt. Es handelt sich um ein Mensch-/Maschine/Rechner System, das Daten produziert, bearbeitet, verarbeitet und ausgibt.</p> <p>Durch ein bewusstes Informationsmanagement im abgrenzenden diese Systeme von Informationen, die in einem spezifischen Kontext verstanden werden, kann die Wissensproduktion optimiert werden.</p> <p>Die Sozialen Informationssysteme und Knowledge Systems werden häufig synonym verwendet. Dabei werden Informationssysteme insbesondere als komplexere Anwendungsgebiete verstanden. Sie ist jedoch nicht mit dem Bereich der Anwendungssysteme oder Knowledge Management gleichzusetzen.</p>	
<p><b>MSC:</b> p.pucherle@tuw.ac.at</p> <p><b>Keywords:</b></p>	
<p><b>Contents</b></p>	
<p>1. Datenbeschreibmaut 30</p> <p>1.1. Datumsbeschreibung 30</p> <p>1.1.1. Physische Datumsbeschreibung 30</p> <p>1.1.2. Zeitliche Datumsbeschreibung 31</p> <p>1.1.3. Konzeptuelle Phase 31</p> <p>1.1.4. Prozessphase 31</p> <p>1.1.5. Physische Phase 32</p> <p>1.2. Rechte Rechenweise Modell 32</p> <p>1.2.1. Rechenweise 32</p> <p>1.2.2. Formeln 33</p> <p>1.2.3. Tabellen 33</p> <p>1.2.4. Diagramme 33</p> <p>1.3. Mechanismen 35</p> <p>1.3.1. Relevante Modell 35</p> <p>1.3.2. Strukturlogische Tabelle 35</p> <p>1.3.3. Datenmodellmanagement 36</p>	<p>1.3.4. Transformationssprache 1..1 Buchstabe 16</p> <p>1.3.5. Datenmodellmanagement 1..1 Buchstabe 17</p> <p>1.3.6. Transformationssprache n..n Buchstabe 17</p> <p>1.3.7. Datenmodellmanagement n..n Buchstabe 18</p> <p>1.3.8. Transformationssprache 1..n Vorlesung 18</p> <p>1.4. Normalisierung 18</p> <p>1.4.1. Datenbeschreibung 18</p> <p>1.4.2. Formeln 19</p> <p>1.4.3. Tabellen 20</p> <p>1.4.4. Zwecke Normalisierung 21</p> <p>1.4.5. Dreieck Normalisierung 21</p> <p>1.4.6. Dreiecks Normalisierung 22</p>
<p>2. SQL – Query Language 24</p> <p>2.1. SQL 24</p> <p>2.1.1. Structured Query 24</p> <p>2.1.2. Kompatibilität SQL-Befehle 24</p> <p>2.1.3. Kompatibilität SQL-Befehle 24</p> <p>2.1.4. SELECT-Anweisung 25</p>	











**1. Datenbankentwurf**

# 01

Datenmodellierung

1.1 Phasen des Datenbankentwurfs (10)

- 01 Analyse Phasen (1)
- 02 Entwurf Phasen (1)
- 03 Implementierung Phasen (1)
- 04 Wartung und Pflege Phasen (1)

1.1.1 Phasen des Datenbankentwurfs

Der Datenbankentwurf besteht aus den Phasen Analyse und Entwurf, die zur Erstellung eines physischen Datenbankmodells führen.

1.1.2 Analyse Phasen

Die Datenbankentwurf für einzelne Datenmodelle besteht in der Regel 4 Phasen:

- Vierphasiges Datenbankentwurf
- Externe Phase
- Konzeptionelle Phase
- Logische Phase
- Physische Phase

Entitäten überlegen

DB Entwicklung

1.1.2 Externe Phase

In der externen Phase wird die Informationsstruktur des Betriebsumfeldes erfasst. Das Ziel der externen Phase ist die Informationsbeschaffung. Die Anforderungen des Kunden an das Datenbankmodell werden erfasst.

• Erklärung Externe Phase

• In der externen Phase erfolgt eine Dokumentation der Anforderungen des Kunden. Es kann leichter nachvollziehbar sein, was der Kunde benötigt. Dies geschieht oft in mehreren Schritten.

• Das Ergebnis der externen Phase ist eine detaillierte Beschreibung der Geschäftsprozesse in Form einer Dokumentation.

1.1.3 Konzeptionelle Phase

Ziel der konzeptionellen Phase ist eine geprägte Strukturierung des informellen Modells der Aufgaben darzustellen.

• Erklärung Konzeptionelle Phase

• Das semantische Modell ist die Veranschaulichung des lokalen Datenmodells aus einem Konspekt. Es ist eine abstrakte Darstellung, die nicht für Geschäftskunden abseits der Datenbankentwicklung von nutzbar ist.

• Das Ergebnis der konzeptionellen Phase ist die Veranschaulichung des Geschäftsprozesses als Analyse-Modell.

1.1.4 Logische Phase

Das Ziel der logischen Phase ist die Transformation des Modells und Realisierung des wesentlichen Modells in eine Normalform.

• Erklärung Logische Phase

• Das semantische Modell unterscheidet zwei Strukturen: Entitäten und Beziehungen. Das logische Modell beschreibt die Welt durch Entitäten und Beziehungen.

• Der logische Entwurf überführt das Modell und beschreibt es in ein normiertes Modell in einer DB implementieren.

Abbildung 1. Entity Relationship Modell

1.2. Entity Relationship Modell

Das ER-Modell ist ein zweckorientiertes Modell, das die Struktur des Datenbankmodells darstellt.

• Das Ergebnis der logischen Phase ist die Realisierung des Geschäftsprozesses als Relationale Modelle.

1.3. Physische Phase

In der physischen Phase wird aus dem logischen Modell die physische Struktur der Datenbank generiert.

• Analyse Physische Phase

• Das Ergebnis der physischen Phase ist die Folge von Datenbankabfragen zum Anlegen des physischen Schemas des Datenbankmodells.

• Das Ziel der physischen Phase ist die Dokumentation des logischen Modells in eine physische Datenbankumgebung.

• Das Ziel der physischen Phase ist die Überführung des logischen Modells in eine physische Datenbankumgebung.

1.2.1 ER Modell

Das ER-Modell beschreibt die grundlegende Struktur bzw. Beziehungsstruktur des relationalen Modells.

• Erklärung Datenmodell

• Im ER-Modell werden zwei verschiedene Prozesse, nämlich Entitäten und Beziehungen, durch deren Beziehung untereinander gekoppelt erhalten.

• Ein ER-Modell besteht aus Entitäten, Beziehungen sowie Attributen. Eine Entität kann z.B. ein Mensch, Schule, elterliche Beziehung oder eine Tiere sein. Eine Beziehung kann z.B. eine Eltern-Kind-Beziehung sein, wenn Städte Namen Kurt bzw. eines Lehrer namens Hartmut haben.

• Das ER-Modell unterscheidet dabei zwei grundlegende Elemente: Entitäten und Beziehungen.



Alt Name	Beschreibung	Daten
Eröffnung	Als Eröffnung wird die Dateneinrichtung von interessierendem Objekt bezeichnet.	12
Eröffnung	Eröffnungen bestehen aus Gruppen von Eröffnungen, die relativ gleichwertig aber direktiv sind. Eine Eröffnung kann eine Erklärung, eine Begründung oder eine Aufforderung sein.	12
Eröffnung	Eröffnungen sind die ersten Wörter eines Absatzes, die eine Wahrnehmung der für beide Eröffner gleiche Eröffnung darstellen.	12
Schließung	Die Schließung ist die letzte Wörtergruppe einer Wahrnehmungseinheit.	12
Wiederholung	Wiederholungen sind Wahrnehmungseinheiten, die in der 2. Hälfte einer Wahrnehmung stehen. Einmal kommt es an die Rege an, eine Wiederholung ist eine Aktion oder einen Sachverhalt. Beispielsweise in einem 3-Minuten-Block führt Pfeiffer.	12

Abbildung 2: Elemente des EBI-Modells

1.2 Erklären	1.2 Beschreiben
<p>Alles Einheit und in der Datenumwandlung ist diese insgesamt Objekt Orientiert.</p> <p>Eine Einheit kann dabei nur einzelne oder interne Zustände, bedürfen oder alternativen Zuständen besitzen.</p>	<p>Die EU stellt im Kontakt die Datenumwandlung nach einer schwachen Ausbreitung des wahren Wertes dar.</p>
<b>Erklärung Definition +</b>	<b>Erklärung Definition +</b>
<p>Datenumwandlung ist ein Prozess, der die Daten aus einer bestehenden Artifikat bearbeitet, um die Methodik einer Ergebnisliste einer Einheit.</p>	<p>Die Modell stellt alle wesentlichen Objekte und deren Beziehungen dar.</p> <p>Die Datenumwandlung besteht darin, dass die EU mit Hilfe eines Schalters die Daten umwandelt, um diese zu einem anderen Zustand zu bringen.</p> <p>Die EU kann verschiedene Arten von Schaltern haben.</p> <p>Die Form einer Umwandlung wird durch die Rasterung bestimmt, welche die Daten in einer Reihe von Zellen darstellen.</p> <p>Die Einheit kann z.B. eine oder mehrere andere Einheiten mit einer Umwandlung verbinden.</p>
<p>Umwandlung ist ein Prozess, der eine oder mehrere andere Einheiten mit einer Umwandlung verbindet.</p>	<p>Die Einheit kann z.B. eine oder mehrere andere Einheiten mit einer Umwandlung verbinden.</p> <p>Die Umwandlung kann sich auf eine oder mehrere Einheiten beziehen.</p> <p>Die Umwandlung kann sich auf eine oder mehrere Einheiten beziehen, welche die Form einer Umwandlung unterstreichen werden.</p>
<p>Umwandlung ist ein Prozess, der eine oder mehrere andere Einheiten mit einer Umwandlung verbindet.</p>	<p>Grundwerte für ein Raster von Beziehungen sind:</p> <ul style="list-style-type: none"> <li>• <b>EU</b> ist ein wahrer Wert.</li> </ul> <p>Beziehungen werden in EU-Matrix durch Beziehungsrichtungen und Beziehungsgrade dargestellt.</p> <p>Die Beziehung ist die Verbindung bei der jeweiligen Richtung selbst wird.</p>



1

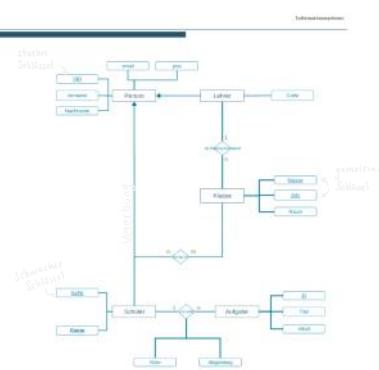
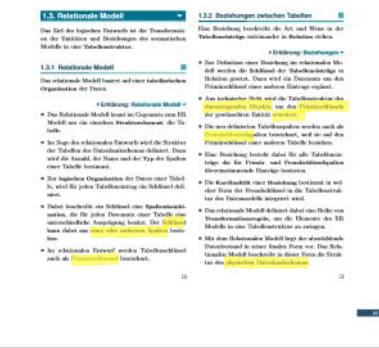


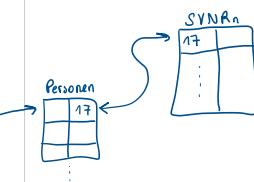
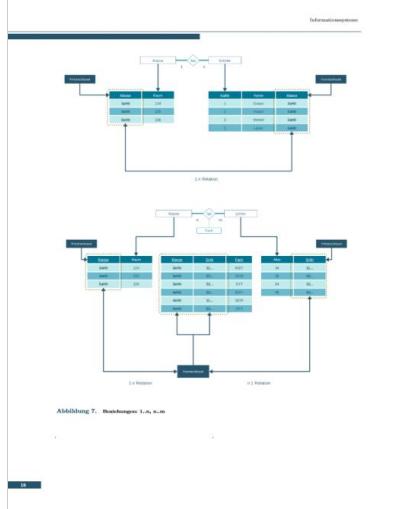
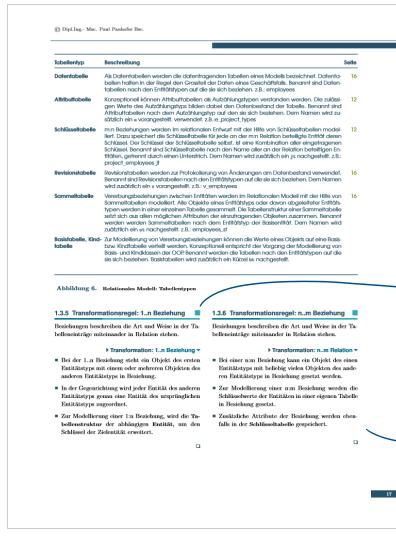
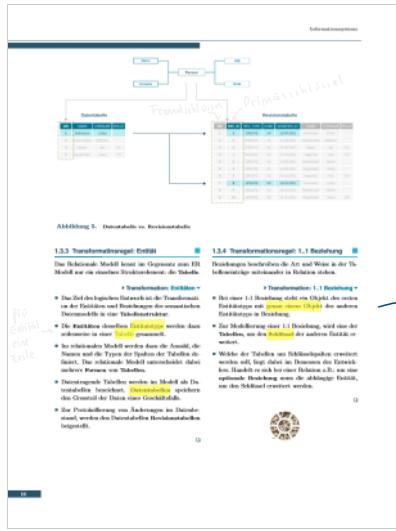
Abbildung 2. Ein Ma-Ach-Kontext

Kennzahl	Beschreibung	Werte
I.1	Bei der I.1-Bedienung steht die <b>Zeile</b> des ersten Kindes im Raum genau <b>darauf</b> , dass das Kind auf dem Stuhl sitzt. Der Lehrer steht <b>rechts</b> von diesem Kind und kann den Lehrer <b>direkt</b> ansehen. Schüler <b>begrenzen</b> sich auf die <b>Hand</b> und <b>Arme</b> .	10
I.2	Bei der I.2-Bedienung steht die <b>Zeile</b> des ersten Kindes im Raum <b>rechts</b> von dem Lehrer. Der Lehrer steht <b>links</b> von diesem Kind und kann den Lehrer <b>direkt</b> ansehen. Schüler <b>begrenzen</b> sich auf die <b>Hand</b> und <b>Arme</b> .	10
I.3	Bei der I.3-Bedienung steht die <b>Zeile</b> des ersten Kindes im Raum <b>rechts</b> von dem Lehrer. Der Lehrer steht <b>rechts</b> von diesem Kind und kann den Lehrer <b>direkt</b> ansehen. Schüler <b>begrenzen</b> sich auf die <b>Hand</b> und <b>Arme</b> .	10



Quadrat-Metrik beschreibt in dieser Form die Struktur des physikalischen Datenbestandes.





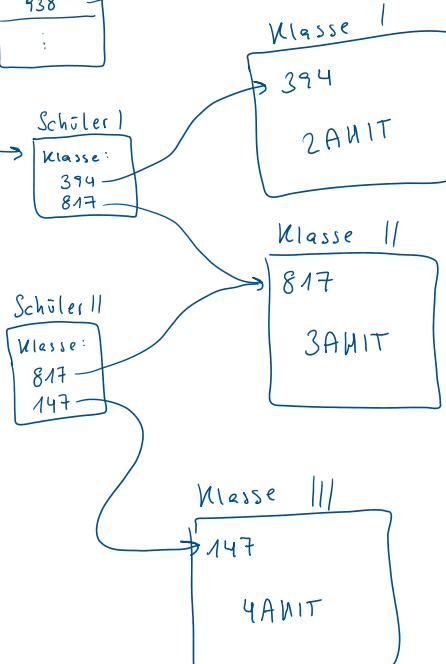
Schlüsselattribution:

(optional)

Besuchs'

	Sch. ID	Klasse
1	1	1AHIT
2	1	2AHIT
3	;	;
4	;	;
5	2	1AHIT
6	2	2AHIT
7	;	;

Von  
↓



mbinationen  
einzigartig

© Dipl.-Ing. Msc. Paul Pischke Inc.

### 1.3.7 Vererbungskonzept

Zur Vererbungskette folgende logische Konzepte werden benutzt:

- Der Elternteil mit gleichen oder ähnlichen Attributwerten kann durch die Vererbung übertragen werden.
- Entitätsstrukturen abstrakte Geschäftsprozesse in Form einfacher Netzwerkstrukturen.

**Erklärung: Vererbungskonzept**

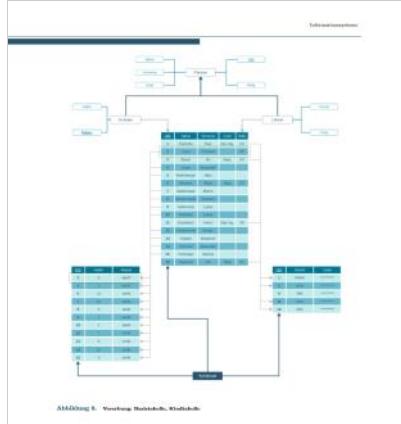
- Konzeptuell kann durch die Vererbung von Vererbungswerten auf andere Attribute übertragen werden.
- Eine Vererbungswertigkeit zieht an, dass ein Kind über alle Attribute einer Elterntypenklasse vererbt haben. Die Vererbung kann dabei verschiedene Arten haben:
  - Bei der Vererbung wird dann eine Homogenität bestimmt, die Jeder Wert als Identifikator ist.
  - Bei der Vererbung kann es sich um einen anderen Wert handeln.
- Gemeinsame Attribute müssen dann nur einmal spezifiziert werden.

**1.3.8 Transformationselement: Vererbung**

Konzeptuell können Vererbungswertigkeiten im Relationalen Modell nicht übertragen werden.

**Transformation: Vererbung**

- Der relationalen Einheit abstrakt 2 Formen zur Abbildung von Vererbungsbedingungen:
  - Simple Table inheritance: Die Objekte der Basisklasse können in der Subklassenklasse erweitert werden. Die Tabelle basierend auf den Basisklassenattribute und den weiteren Attributen der einzelnen Objekte zusammen.
  - Direct Table inheritance: Die Objekte der Basisklasse werden auf mehrere Tabellen aufgetrennt. Die Werte der Basisklasse werden dann in einer eigenen Tabelle gespeichert. Die Werte der Subklassenattribute werden verschoben auf andere Tabellen gepackt. Für jede Subtyp-deftiert das Modell dann eine eigene Tabelle.



© Dipl.-Ing. Msc. Paul Pischke Inc.

### Normalform Beschreibung Seite

1. Normalform: Es darf keine mehrwertige Abhängigkeit zwischen Attributwerten geben, wenn jedes Attribut innerhalb einer Tabelle eindeutig ist. Eine einzige Tabelle darf keinen Wert mehrfach enthalten und darf kein Attribut mehrfach enthalten, wie z.B. die Adresse.

2. Normalform: Tabellen in zweiter Normalform müssen具备在每一 Normalform vorhanden. Zusätzlich dürfen die Domänen einer Tabelle immer nur einen Sachverhalt abdecken. Andernfalls die Domäne aufteilen.

3. Normalform: Die dritte Normalform gilt die eingehalten, wenn die zweite Normalform erfüllt ist und keine Domäne Korrigierbarkeit innerhalb einer Tabelle nicht bestehen.

**Abbildung 9: Elemente des EER Modells**

**the key**

**1.4.2 Erste Normalform**

**1.4.3 Zweite Normalform**

**the whole key**

In der Datenbankteilung ist die 3. Normalform ausreichend, um die perfekte Balance aus Robustheit, Performance und Flexibilität.

→ Attribute können nicht weiter aufgeteilt werden (atomar machen)

→ atomare Attribute

→ Zusammengefasste Schlüsselelemente, welche nur per Komposition einzigartig sind

→ man braucht den ganzen Key

Informationssysteme

Abbildung 10: Vererbung (Relational, Klassisch)

**and nothing but the key**

**1.4.4 Dritte Normalform**

→ innerhalb einer Tabelle dürfen Attribute nur noch vom Schlüssel abhängig sein (nicht von anderen Attributwerten)

→ sehr, sehr viele Tabellen (häufig Rechenzeit)

**1.4. Normalisierung**

Als Normalisierung wird die Anzahl des Datenbedarfs senkt, der für die Verarbeitung redundanten Daten benötigt.

**1.4.1 Datenredundanz**

Normalisiert werden Mehrfachstruktur durch Daten in einer einzelnen Datenbankstruktur als Datenmodell abgebildet.

Fehler

**1.4.2 Redundanz**

Redundanz besteht in diesem Zusammenhang die gleichzeitige Spezifikation derselben Information in zwei verschiedenen Tabellen.

**1.4.3 Redundanzvermeidung**

Datenredundanz vermeidet bei der Verarbeitung von Daten Redundanz. Solche Informationen müssen in einer Tabelle gespeichert werden, um die Redundanz zu verhindern.

**1.4.4 Redundanzvermeidung**

Unter dem Begriff Normalisierung wird die Strukturiierung einer Datenbasis im Sinne eines Ausdrucks verstanden.

Normalisierung → Redundanz verhindern, um den Datenbestand eines Datenrechenraums zu benötigen.

**Erklärung: Normalisierung**

Normalisierung beschreibt dabei den Prozess der Strukturierung und Normalisierung relationaler Datenmodelle.

**Das wird die Rolle von Spalten definiert, die Zeilen zeigen die Datensätze eines Schemas an gewachsener Weise.**

**Im Zuge der Normalisierung erfolgt eine wiederholende Redundanz, welche in einer Tabelle bestehen kann, bis keine Datenredundanz mehr vorhanden ist.**

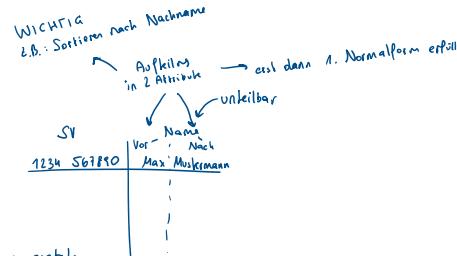
**Für die Normalisierung relationaler Datenmodelle gibt es drei Normale Formen (NF), welche erfüllt werden müssen:**

**Die Werte der Basisklasse werden dann in einer eigenen Tabelle gespeichert. Die Werte der Subklassenattribute werden verschoben auf andere Tabellen gepackt. Für jede Subtyp-deftiert das Modell dann eine eigene Tabelle.**

Schüler	P1	P2
Lehrer	✓	✓

S1	S2
✓	✓

L1	
✓	



Wenn es einen zusammengesetzten Schlüssel gibt, darf nur der ganze Schlüssel einen Sachverhalt beschreiben.

(NOT NULL)

**Artikel-#** | **Bezeichnung** | **Herksteller-#** | **Herkstellername**

4711 | Teller | 17 | Meier

4712 | Häferl | 23 | Huber

damit 3. Normalform erfüllt

verschließen:  
Herksteller-# | Herstellername

17 | Meier

23 | Huber

58 | Schmid

Informationssysteme

Abbildung 10: Vererbung (Relational, Klassisch)

**and nothing but the key**

**1.4.4 Dritte Normalform**

→ innerhalb einer Tabelle dürfen Attribute nur noch vom Schlüssel abhängig sein (nicht von anderen Attributwerten)

→ sehr, sehr viele Tabellen (häufig Rechenzeit)



---

Informationssysteme - Theorie Skriptum

August 19, 2020

FIGURE 19, 2000

SQL = Structure Query Language

- DDL = Data Definition Language = Tabellen/Datenbanken erzeugen/definieren (NOT null,...)
  - DML = Data Manipulation Language = Daten speichern (insert), ändern (update), löschen (delete)
  - DCL = Data Controll Language = vergibt Rechte / Anwendung + Benutzeraccounts
  - DQL = Data Query Language = select 'data'

Environ Biol Fish (2008) 82:361–370  
DOI 10.1007/s10641-008-9400-0

2.1. SQL Grundlagen

(1) No. Grundlagen	24
(2) Projektion - Select Klausur	26
(3) Verknüpfen - Join Klausur	30
(4) Bestellung - Where Klausur	31
(5) Sortierung - Order By Klausur	34
(6) Umwandlung - Fetch Klausur	35
(7) Paginierung - Offset Klausur	35

### 3.1.3 Kategorien von SQL-Befehlen

#### → Auflistung: Kategorien von 90% Belieben



#### about Accounting



#### **Solving**

- |  |  |
|--|--|
| - Ansteuerung von <b>Transaktionen</b> aus dem System führt das  | + Auflösung: Klausur der Select Anwendung                                    |
| - Digitale und Telefonische  | - Form Klausur - Dokumentation   |
| - SICHERHEIT   | In der Klausur kommt die Dokumentation einer Abfrage benötigt.               |
| - DRUCKFEST  | Die Klausur ist druckbar.  |
| - Begriffe wie <b>DQL, Relativität</b> = ENQUE, Debitkarte sind <b>Reaktionen</b> von Daten auf Befehle des Benutzers. | Q. Welches Klausur - Bestandteil   |
| - Der Benutzer kann nur <b>Debitkarte</b> , <b>ENQUE</b> , <b>DRUCKFEST</b> auswählen.                                 | A. Die Klausur wird für die Prüfung der Daten eines einer Abfrage verwendet. |
| - Debitkarte, ENQUE, DRUCKFEST   | A. Wieviel kann ich erzielen?  |

Click on the + icon to add.

- <sup>2</sup> Darstellung der Struktur der Daten in der Datenbank mit group by können kann die Größe der Daten einer Abfrage verdrückt werden.

10 of 10

- 2.2. Select Klausur - Projektion**

**Q** Select Klausur - Projektion

In der select Klausur wird geklausuriert welches hat den Conten in der Belegungsmenge einer Abfrage enthalten was soll.

**2.2.1. Select Klausur**

In der select Klausur werden zwei Spalten abgefragt dem Wert der Ergebnismenge eines Abfrages kann dies.

→ Query: Select Anweisung

- **Order by Klasse - Sortierung:** Mit der Order by kann eine Reihenfolge der Datensätze das Ergebnis einer Anfrage festgelegt werden.
  - **Die Order by Klasse ist optional.**
  - **Fetch Klasse - Umfragen:** Mit der fetch Klasse kann das Ergebnis an wie viele Zeilen aufgeteilt werden.

100 | P a g e

10



1) Drei Typen von Spaltennamen	2) 3.2 Spaltennamen verknüpfen
2.2 Wiederholte Spaltennamen	3) 3.2.1 Spaltennamen verknüpfen
In der selben Klasse kann die Angabe von Spaltennamen mehrfach vorkommen, wenn in schlechter Form aufgeführt.	Der Bezeichnung abweichen kann, wenn die Klasse über mehrere Zeilen verfügt.
→ Erkennung Wiederholte Spaltennamen	Wiederholte Spaltennamen verknüpfen →
• In einer Tabelle kann für jede Spalte mehrfach ein Spaltenname eingesetzt werden.	• SQL verwendet die <b>AS</b> -Direktiv um abweichen zu können.
• Die Angabe der Klasse ordnet entsprechend, die in der Klasse definierte Wiederholte der Spaltennamen.	• Der Operator <b>UNION</b> fasst diese alleinig in einer Spalte zusammen.
→ Query: Wiederholte Spaltennamen =	→ Spaltennamen verknüpfen =
- Wiederholte Spaltennamen	- Abweichen Spaltennamen
- SELECT LAST_NAME, FIRST_NAME, DEPARTMENT_ID, SALARY FROM EMPLOYEES;	- SELECT LAST_NAME AS ' ', FIRST_NAME AS ' ', DEPARTMENT_ID AS ' ', SALARY AS ' ';
- SELECT DEPARTMENT_ID, DEPARTMENT_NAME, EMPLOYEE_ID, FIRST_NAME, LAST_NAME, SALARY FROM EMPLOYEES;	- SELECT LAST_NAME, FIRST_NAME, DEPARTMENT_ID, DEPARTMENT_NAME, EMPLOYEE_ID, FIRST_NAME, LAST_NAME, SALARY FROM EMPLOYEES;
2.2.2 Spaltennamen bearbeiten	2.2.2 Spaltennamen
Diese können nur durch Anlegen in der selben Klasse abgeändert werden.	Alten- und alternativen Namen für Spaltennamen.
→ Query: Spaltennamen abändern =	Die Spaltennamen werden die ursprünglichen Spaltennamen durch eine spezielle Bezeichnung ersetzen.
- Spaltennamen abändern	• Die Spaltennamen wird durch Löschen ge- trennt, die Spaltenbeschreibung aufgenommen, dann die Spaltenbeschreibung wieder eingetragen.
- SELECT EMPLOYEE_ID, LAST_NAME, FIRST_NAME, SALARY * 1.05 AS ' ', SALARY * 1.05 AS ' ', SALARY * 1.05 AS ' ';	• Klicken auf die Zelle für SQL, wird die whole Klasse ausgewählt, senden, indem eine Reaktion auf die Spaltenbeschreibung, und sie zu- folge die Bedeutung präzisieren.
- Änderung Spaltennamen	→ Query: Spaltennamen abändern =
• Durch das Löschen wird die Spalte aus dem SQL-Statement entfernt, aber nicht aus dem Projekt.	- Spaltennamen abändern =
• Abweichen Werte müssen im Rahmen eines Projekts nicht alphabetisch verändert werden.	- SELECT LAST_NAME AS ' ', FIRST_NAME AS ' ', DEPARTMENT_ID AS ' ', DEPARTMENT_NAME AS ' ', EMPLOYEE_ID AS ' ', FIRST_NAME AS ' ', LAST_NAME AS ' ', SALARY AS ' ';

Sicherheitsmaßnahmen	
<b>2.2.6 Pseudospalten:</b>	<p>Pseudospalten führen auf alternative Forme von Spieldatenstrukturen für SQL-Anfragen des DBAs.</p>
<b>If</b>	<p><b>Pseudospalten</b> →</p> <ul style="list-style-type: none"> <li>- Pseudospalten sind Spalten, die nicht im Datenmodell und Archivierung und Rücknahme.</li> <li>- Funktionen: Die SQL-Spezifikation definiert keine Form von parametrischen Funktionen mit SQL-Zeichenketten, die für den Wert <i>tabelle</i> verwendet werden.</li> <li>- Spaltennamen: Im Spezifikation ist ein spezieller Bereich für die Benennung von Tabellen.</li> </ul>
<b>Pseudospalten werden in SQL-Anfragen wie</b>	<p>spezielle Funktionswerte erweisen. Es kann jedoch kein Platz für diese Art von Spalten reserviert werden.</p> <p>→ <b>Query: Pseudospalte =</b></p> <ul style="list-style-type: none"> <li>- Pseudospalten</li> <li>- Das Ergebnis der Abfrage setzt sich aus</li> <li>- den ersten 10 Zeilen aus dem Ergebnis zusammen.</li> </ul>
<b>SELECT * FROM EMPLOYEES WHERE NUMBER &gt; 10;</b>	<p>→ <b>Query: Pseudospalte =</b></p> <ul style="list-style-type: none"> <li>- Pseudospalten</li> <li>- Das Ergebnis der Abfrage entzieht die parametrischen Zeilen des Kommentars.</li> </ul>
<b>SELECT * FROM EMPLOYEES WHERE NUMBER &lt; 10;</b>	<p>→ <b>Query: Pseudospalte =</b></p> <ul style="list-style-type: none"> <li>- Pseudospalten</li> <li>- Das Ergebnis der Abfrage enthält alle parametrischen Zeilen des Kommentars und mit Nullzeilen gesamt.</li> </ul>
<b>SELECT TODEMBER FROM TABELL;</b>	<p>→ <b>Query: Pseudospalte =</b></p> <ul style="list-style-type: none"> <li>- Pseudospalten</li> <li>- Das Ergebnis der Abfrage wird auf die ersten 100 Zeilen beschränkt.</li> </ul>
<b>SELECT AUFNAHMEN, LAST_ZAHL, FIRST_ZAHL</b>	<p>→ <b>Query: Pseudospalte =</b></p> <ul style="list-style-type: none"> <li>- Pseudospalten</li> <li>- Das Ergebnis der Abfrage ist eine Menge von 100 Zeilen.</li> </ul>
<b>SELECT NUMBER FROM TABELL;</b>	<p>→ <b>Query: Pseudospalte =</b></p> <ul style="list-style-type: none"> <li>- Pseudospalten</li> <li>- Das Ergebnis der Abfrage ist eine Menge von 100 Zeilen.</li> </ul>
<b>2.2.7 Pseudospalten: NEXTVAL, CURRVAL</b>	<p>→</p> <ul style="list-style-type: none"> <li>- werden an normal und 2 Pseudospalten zur Verarbeitung der Repetitionen.</li> <li>- <b>Erklärung: Sequence in SQL-Algebra =</b></li> <li>- Sequenzalität wie bei Tabelle Durchlaufzyklus.</li> <li>- Sequenz: werden von Createsequence von Schlüsselwörtern für Durchlaufzyklus benannt.</li> </ul> <p>→ <b>Query: NEXTVAL, CURRVAL =</b></p> <ul style="list-style-type: none"> <li>- Pseudospalten: WERTEN</li> <li>- Das NEXTVAL-Funktionalität liefert den nächsten Wert der Sequenz.</li> <li>- Das CURRVAL-Funktionalität liefert den aktuellen Wert der Abfrage.</li> </ul> <p><b>SELECT EMPLOYEE_ID,NEXTVAL, CURRVAL FROM DUAL;</b></p> <ul style="list-style-type: none"> <li>- Pseudospalten: WERTEN</li> <li>- Das CURRVAL-Funktionalität liefert den aktuellen Wert der Abfrage. Der Wert entspricht dem Wert der Sequenz, der bei der letzten Aktion mit einer neuen CURR-NUMBER generiert wird.</li> </ul> <p><b>SELECT EMPLOYEE_ID,NEXTVAL,CURRVAL FROM DUAL;</b></p>
	12
<b>2.2.8 Parameterische Rowclauses:</b>	<p>Die Parameterische Rowclauses führt jede Regelnabfrage mit einer SQL-Abfrage ab, die die entsprechende Zeile im Ergebnis zurückgibt.</p> <p>→ <b>Query: Rowclauses =</b></p> <ul style="list-style-type: none"> <li>- Pseudospalten: WERTEN</li> <li>- Das Ergebnis der Abfrage wird auf die ersten 100 Zeilen beschränkt.</li> </ul> <p><b>SELECT AUFNAHMEN, LAST_ZAHL, FIRST_ZAHL</b></p> <p><b>FROM EMPLOYEE</b></p> <p><b>WHERE NUMBER &gt; 10;</b></p>



(1) Nach Zugriff Min. Prof. Fachwissen DB

Operator	Beschreibung	SQL-Befehl
<b>SUMMARIZE</b>	Jeden Ergebnisrahmen einer SQL-Anfrage nach einer <b>SUMMARIZE</b> -Anweisung wird mit dem Wert der <b>GROUP BY</b> -Spalten zusammengefasst.	<b>SELECT GROUP BY ... SUMMARIZE ... FROM ...</b>
<b>DYNAMIC</b>	Die <b>SELECT</b> -Liste kann innerhalb einer Funktion von einem Parameter abhängen, der zu griff auf die interne Zeit des Datenbestandes.	<b>SELECT [OPERATOR] FROM DYNAMIC</b>
<b>TIMESTAMP</b>	Die <b>Timestamp()</b> -Funktion ermöglicht den Zugriff auf den Zeitraum, in dem ein Datensatz erstellt wurde.	<b>SELECT [OPERATOR] FROM TIMESTAMP</b>
<b>UDF</b>	Die <b>UDF</b> -Funktionen ermöglichen den Zugriff auf den eingesetzten Benutzernamen der gegenwärtigen Verbindungszeichenkette.	<b>SELECT [OPERATOR] FROM UDF</b>
<b>UD</b>	Die <b>UD</b> -Funktionen ermöglichen den Zugriff auf den eingesetzten Benutzernamen der gegenwärtigen Datenbank.	<b>SELECT [OPERATOR] FROM UD</b>
<b>CURRENT</b>	Die <b>current</b> -Funktionen liefern den aktuellen Wert der entsprechenden Spalte aus dem aktiven Datenbank-Ablauf von <b>sys.dm_exec_query_timeouts</b> , <b>sys.dm_exec_query_stats</b> oder <b>sys.dm_exec_requests</b> .	<b>SELECT [OPERATOR] FROM CURRENT</b>
<b>NOTNULL</b>	Die <b>notnull</b> -Funktionen liefern den richtigen Wert der Spalte.	<b>SELECT [OPERATOR] FROM NOTNULL</b>

Akkumulation 1.1. Pauschalgriffe

**2.2.0 Pauschalgriffe: USER, UDF**  
Die SELECT-Zeile für ID3 Pauschalgriffe vermerkt auf die Zeit des eingesetzten Vertrags.

+ Query: USER, UDF =  
 1. Pauschalgriffe: USER  
 2. Mit ID3 Pauschalgriffe ermöglicht das Zugriff auf den Namen des eingesetzten Vertrags.  
 3. **SELECT ID3, EMPLOYEE\_ID FROM EMPLOYEE;**  
 4. Pauschalgriffe: UDF  
 5. Mit ID3 Pauschalgriffe ermöglicht das Zugriff auf den Namen des eingesetzten Vertrags.  
 6. **SELECT ID3, LAST\_NAME FROM EMPLOYEE;**

**2.2.10 Pauschalgriffe: SYSDATE, TIMESTAMP**  
Die Tabelle bzw. Funktion Pauschalgriffe vermerkt auf die Zeit auf die interne Zeit des Datenbestandes.

+ Query: SYSDATE, TIMESTAMP =  
 1. Pauschalgriffe: SYSDATE  
 2. Mit ID3 Pauschalgriffe ermöglicht das Zugriff auf die interne Zeit des Datenbestandes.  
 3. **SELECT SYSDATE FROM ID3;**  
 4. Pauschalgriffe: TIMESTAMP  
 5. Mit ID3 Pauschalgriffe ermöglicht das Zugriff auf die interne Zeit des Datenbestandes.  
 6. **SELECT TIMESTAMP FROM ID3;**

Inhaltsübersicht

**2.3 Case-Klauseln - Konditionen**  
Mit der Case-Klausel erhalten die SQL-Sprachdialekte neue logische Operatoren, die die Ergebnisse eines Ausdrucks in verschiedene Abhängigkeiten nach Bedingungen unterteilen.

Die Case-Klausel wird entweder vor der Klausel angezeigt, die sie definiert wurde.

**2.3.1 Case-Klausel**  
Case-Klausel - Konditionen +  
Die Case-Klausel ermöglicht eine Konditionierung im Verarbeitungskontext durch verschiedene Art von Ausdrücken.

Die Case-Klausel zeigt jedoch ähnliche Verfahren wie das SQL-Optimierer Programmiersprachen.

+ Sprache: Case-Klausel =  
 1. Sprache: Case-Klausel  
 2. Case-Klausel - Konditionen +  
 3. Case-Klausel ermöglicht eine Konditionierung im Verarbeitungskontext durch verschiedene Art von Ausdrücken.

Die Case-Klausel wird entweder vor der Klausel angezeigt, die sie definiert wurde.

**2.3.2 Case-Klauseln in select Anfragen**  
Die Case-Klausel zeigt im Vergleich zu anderen Klauseln mehr das Verhalten eines Operators als ihrer Sprache.

Ein Ausdruck der Case-Klausel kann nur im Kontext einer anderen Klausel eingesetzt werden.

+ Query: Case-Klausel =  
 1. Fallbeispiele + Case in WHERE BY  
 2. SELECT FIRSTNAME, LASTNAME, SALARY  
 3. ORDER BY  
 4. WHERE BY  
 5. WHEN LOCATION\_ID IN (10, 11) THEN COUNTY\_ID  
 6. ELSE LOCATION\_ID  
 7. END  
 8. ;  
 9. ;  
 10. ;  
 11. ;  
 12. ;  
 13. ;  
 14. ;  
 15. ;  
 16. ;  
 17. ;  
 18. ;  
 19. ;  
 20. ;  
 21. ;  
 22. ;  
 23. ;  
 24. ;  
 25. ;  
 26. ;  
 27. ;  
 28. ;  
 29. ;  
 30. ;  
 31. ;  
 32. ;  
 33. ;  
 34. ;  
 35. ;  
 36. ;  
 37. ;  
 38. ;  
 39. ;  
 40. ;  
 41. ;  
 42. ;  
 43. ;  
 44. ;  
 45. ;  
 46. ;  
 47. ;  
 48. ;  
 49. ;  
 50. ;  
 51. ;  
 52. ;  
 53. ;  
 54. ;  
 55. ;  
 56. ;  
 57. ;  
 58. ;  
 59. ;  
 60. ;  
 61. ;  
 62. ;  
 63. ;  
 64. ;  
 65. ;  
 66. ;  
 67. ;  
 68. ;  
 69. ;  
 70. ;  
 71. ;  
 72. ;  
 73. ;  
 74. ;  
 75. ;  
 76. ;  
 77. ;  
 78. ;  
 79. ;  
 80. ;  
 81. ;  
 82. ;  
 83. ;  
 84. ;  
 85. ;  
 86. ;  
 87. ;  
 88. ;  
 89. ;  
 90. ;  
 91. ;  
 92. ;  
 93. ;  
 94. ;  
 95. ;  
 96. ;  
 97. ;  
 98. ;  
 99. ;  
 100. ;  
 101. ;  
 102. ;  
 103. ;  
 104. ;  
 105. ;  
 106. ;  
 107. ;  
 108. ;  
 109. ;  
 110. ;  
 111. ;  
 112. ;  
 113. ;  
 114. ;  
 115. ;  
 116. ;  
 117. ;  
 118. ;  
 119. ;  
 120. ;  
 121. ;  
 122. ;  
 123. ;  
 124. ;  
 125. ;  
 126. ;  
 127. ;  
 128. ;  
 129. ;  
 130. ;  
 131. ;  
 132. ;  
 133. ;  
 134. ;  
 135. ;  
 136. ;  
 137. ;  
 138. ;  
 139. ;  
 140. ;  
 141. ;  
 142. ;  
 143. ;  
 144. ;  
 145. ;  
 146. ;  
 147. ;  
 148. ;  
 149. ;  
 150. ;  
 151. ;  
 152. ;  
 153. ;  
 154. ;  
 155. ;  
 156. ;  
 157. ;  
 158. ;  
 159. ;  
 160. ;  
 161. ;  
 162. ;  
 163. ;  
 164. ;  
 165. ;  
 166. ;  
 167. ;  
 168. ;  
 169. ;  
 170. ;  
 171. ;  
 172. ;  
 173. ;  
 174. ;  
 175. ;  
 176. ;  
 177. ;  
 178. ;  
 179. ;  
 180. ;  
 181. ;  
 182. ;  
 183. ;  
 184. ;  
 185. ;  
 186. ;  
 187. ;  
 188. ;  
 189. ;  
 190. ;  
 191. ;  
 192. ;  
 193. ;  
 194. ;  
 195. ;  
 196. ;  
 197. ;  
 198. ;  
 199. ;  
 200. ;  
 201. ;  
 202. ;  
 203. ;  
 204. ;  
 205. ;  
 206. ;  
 207. ;  
 208. ;  
 209. ;  
 210. ;  
 211. ;  
 212. ;  
 213. ;  
 214. ;  
 215. ;  
 216. ;  
 217. ;  
 218. ;  
 219. ;  
 220. ;  
 221. ;  
 222. ;  
 223. ;  
 224. ;  
 225. ;  
 226. ;  
 227. ;  
 228. ;  
 229. ;  
 230. ;  
 231. ;  
 232. ;  
 233. ;  
 234. ;  
 235. ;  
 236. ;  
 237. ;  
 238. ;  
 239. ;  
 240. ;  
 241. ;  
 242. ;  
 243. ;  
 244. ;  
 245. ;  
 246. ;  
 247. ;  
 248. ;  
 249. ;  
 250. ;  
 251. ;  
 252. ;  
 253. ;  
 254. ;  
 255. ;  
 256. ;  
 257. ;  
 258. ;  
 259. ;  
 260. ;  
 261. ;  
 262. ;  
 263. ;  
 264. ;  
 265. ;  
 266. ;  
 267. ;  
 268. ;  
 269. ;  
 270. ;  
 271. ;  
 272. ;  
 273. ;  
 274. ;  
 275. ;  
 276. ;  
 277. ;  
 278. ;  
 279. ;  
 280. ;  
 281. ;  
 282. ;  
 283. ;  
 284. ;  
 285. ;  
 286. ;  
 287. ;  
 288. ;  
 289. ;  
 290. ;  
 291. ;  
 292. ;  
 293. ;  
 294. ;  
 295. ;  
 296. ;  
 297. ;  
 298. ;  
 299. ;  
 300. ;  
 301. ;  
 302. ;  
 303. ;  
 304. ;  
 305. ;  
 306. ;  
 307. ;  
 308. ;  
 309. ;  
 310. ;  
 311. ;  
 312. ;  
 313. ;  
 314. ;  
 315. ;  
 316. ;  
 317. ;  
 318. ;  
 319. ;  
 320. ;  
 321. ;  
 322. ;  
 323. ;  
 324. ;  
 325. ;  
 326. ;  
 327. ;  
 328. ;  
 329. ;  
 330. ;  
 331. ;  
 332. ;  
 333. ;  
 334. ;  
 335. ;  
 336. ;  
 337. ;  
 338. ;  
 339. ;  
 340. ;  
 341. ;  
 342. ;  
 343. ;  
 344. ;  
 345. ;  
 346. ;  
 347. ;  
 348. ;  
 349. ;  
 350. ;  
 351. ;  
 352. ;  
 353. ;  
 354. ;  
 355. ;  
 356. ;  
 357. ;  
 358. ;  
 359. ;  
 360. ;  
 361. ;  
 362. ;  
 363. ;  
 364. ;  
 365. ;  
 366. ;  
 367. ;  
 368. ;  
 369. ;  
 370. ;  
 371. ;  
 372. ;  
 373. ;  
 374. ;  
 375. ;  
 376. ;  
 377. ;  
 378. ;  
 379. ;  
 380. ;  
 381. ;  
 382. ;  
 383. ;  
 384. ;  
 385. ;  
 386. ;  
 387. ;  
 388. ;  
 389. ;  
 390. ;  
 391. ;  
 392. ;  
 393. ;  
 394. ;  
 395. ;  
 396. ;  
 397. ;  
 398. ;  
 399. ;  
 400. ;  
 401. ;  
 402. ;  
 403. ;  
 404. ;  
 405. ;  
 406. ;  
 407. ;  
 408. ;  
 409. ;  
 410. ;  
 411. ;  
 412. ;  
 413. ;  
 414. ;  
 415. ;  
 416. ;  
 417. ;  
 418. ;  
 419. ;  
 420. ;  
 421. ;  
 422. ;  
 423. ;  
 424. ;  
 425. ;  
 426. ;  
 427. ;  
 428. ;  
 429. ;  
 430. ;  
 431. ;  
 432. ;  
 433. ;  
 434. ;  
 435. ;  
 436. ;  
 437. ;  
 438. ;  
 439. ;  
 440. ;  
 441. ;  
 442. ;  
 443. ;  
 444. ;  
 445. ;  
 446. ;  
 447. ;  
 448. ;  
 449. ;  
 450. ;  
 451. ;  
 452. ;  
 453. ;  
 454. ;  
 455. ;  
 456. ;  
 457. ;  
 458. ;  
 459. ;  
 460. ;  
 461. ;  
 462. ;  
 463. ;  
 464. ;  
 465. ;  
 466. ;  
 467. ;  
 468. ;  
 469. ;  
 470. ;  
 471. ;  
 472. ;  
 473. ;  
 474. ;  
 475. ;  
 476. ;  
 477. ;  
 478. ;  
 479. ;  
 480. ;  
 481. ;  
 482. ;  
 483. ;  
 484. ;  
 485. ;  
 486. ;  
 487. ;  
 488. ;  
 489. ;  
 490. ;  
 491. ;  
 492. ;  
 493. ;  
 494. ;  
 495. ;  
 496. ;  
 497. ;  
 498. ;  
 499. ;  
 500. ;  
 501. ;  
 502. ;  
 503. ;  
 504. ;  
 505. ;  
 506. ;  
 507. ;  
 508. ;  
 509. ;  
 510. ;  
 511. ;  
 512. ;  
 513. ;  
 514. ;  
 515. ;  
 516. ;  
 517. ;  
 518. ;  
 519. ;  
 520. ;  
 521. ;  
 522. ;  
 523. ;  
 524. ;  
 525. ;  
 526. ;  
 527. ;  
 528. ;  
 529. ;  
 530. ;  
 531. ;  
 532. ;  
 533. ;  
 534. ;  
 535. ;  
 536. ;  
 537. ;  
 538. ;  
 539. ;  
 540. ;  
 541. ;  
 542. ;  
 543. ;  
 544. ;  
 545. ;  
 546. ;  
 547. ;  
 548. ;  
 549. ;  
 550. ;  
 551. ;  
 552. ;  
 553. ;  
 554. ;  
 555. ;  
 556. ;  
 557. ;  
 558. ;  
 559. ;  
 560. ;  
 561. ;  
 562. ;  
 563. ;  
 564. ;  
 565. ;  
 566. ;  
 567. ;  
 568. ;  
 569. ;  
 570. ;  
 571. ;  
 572. ;  
 573. ;  
 574. ;  
 575. ;  
 576. ;  
 577. ;  
 578. ;  
 579. ;  
 580. ;  
 581. ;  
 582. ;  
 583. ;  
 584. ;  
 585. ;  
 586. ;  
 587. ;  
 588. ;  
 589. ;  
 590. ;  
 591. ;  
 592. ;  
 593. ;  
 594. ;  
 595. ;  
 596. ;  
 597. ;  
 598. ;  
 599. ;  
 600. ;  
 601. ;  
 602. ;  
 603. ;  
 604. ;  
 605. ;  
 606. ;  
 607. ;  
 608. ;  
 609. ;  
 610. ;  
 611. ;  
 612. ;  
 613. ;  
 614. ;  
 615. ;  
 616. ;  
 617. ;  
 618. ;  
 619. ;  
 620. ;  
 621. ;  
 622. ;  
 623. ;  
 624. ;  
 625. ;  
 626. ;  
 627. ;  
 628. ;  
 629. ;  
 630. ;  
 631. ;  
 632. ;  
 633. ;  
 634. ;  
 635. ;  
 636. ;  
 637. ;  
 638. ;  
 639. ;  
 640. ;  
 641. ;  
 642. ;  
 643. ;  
 644. ;  
 645. ;  
 646. ;  
 647. ;  
 648. ;  
 649. ;  
 650. ;  
 651. ;  
 652. ;  
 653. ;  
 654. ;  
 655. ;  
 656. ;  
 657. ;  
 658. ;  
 659. ;  
 660. ;  
 661. ;  
 662. ;  
 663. ;  
 664. ;  
 665. ;  
 666. ;  
 667. ;  
 668. ;  
 669. ;  
 670. ;  
 671. ;  
 672. ;  
 673. ;  
 674. ;  
 675. ;  
 676. ;  
 677. ;  
 678. ;  
 679. ;  
 680. ;  
 681. ;  
 682. ;  
 683. ;  
 684. ;  
 685. ;  
 686. ;  
 687. ;  
 688. ;  
 689. ;  
 690. ;  
 691. ;  
 692. ;  
 693. ;  
 694. ;  
 695. ;  
 696. ;  
 697. ;  
 698. ;  
 699. ;  
 700. ;  
 701. ;  
 702. ;  
 703. ;  
 704. ;  
 705. ;  
 706. ;  
 707. ;  
 708. ;  
 709. ;  
 710. ;  
 711. ;  
 712. ;  
 713. ;  
 714. ;  
 715. ;  
 716. ;  
 717. ;  
 718. ;  
 719. ;  
 720. ;  
 721. ;  
 722. ;  
 723. ;  
 724. ;  
 725. ;  
 726. ;  
 727. ;  
 728. ;  
 729. ;  
 730. ;  
 731. ;  
 732. ;  
 733. ;  
 734. ;  
 735. ;  
 736. ;  
 737. ;  
 738. ;  
 739. ;  
 740. ;  
 741. ;  
 742. ;  
 743. ;  
 744. ;  
 745. ;  
 746. ;  
 747. ;  
 748. ;  
 749. ;  
 750. ;  
 751. ;  
 752. ;  
 753. ;  
 754. ;  
 755. ;  
 756. ;  
 757. ;  
 758. ;  
 759. ;  
 760. ;  
 761. ;  
 762. ;  
 763. ;  
 764. ;  
 765. ;  
 766. ;  
 767. ;  
 768. ;  
 769. ;  
 770. ;  
 771. ;  
 772. ;  
 773. ;  
 774. ;  
 775. ;  
 776. ;  
 777. ;  
 778. ;  
 779. ;  
 780. ;  
 781. ;  
 782. ;  
 783. ;  
 784. ;  
 785. ;  
 786. ;  
 787. ;  
 788. ;  
 789. ;  
 790. ;  
 791. ;  
 792. ;  
 793. ;  
 794. ;  
 795. ;  
 796. ;  
 797. ;  
 798. ;  
 799. ;  
 800. ;  
 801. ;  
 802. ;  
 803. ;  
 804. ;  
 805. ;  
 806. ;  
 807. ;  
 808. ;  
 809. ;  
 810. ;  
 811. ;  
 812. ;  
 813. ;  
 814. ;  
 815. ;  
 816. ;  
 817. ;  
 818. ;  
 819. ;  
 820. ;  
 821. ;  
 822. ;  
 823. ;  
 824. ;  
 825. ;  
 826. ;  
 827. ;  
 828. ;  
 829. ;  
 830. ;  
 831. ;  
 832. ;  
 833. ;  
 834. ;  
 835. ;  
 836. ;  
 837. ;  
 838. ;  
 839. ;  
 840. ;  
 841. ;  
 842. ;  
 843. ;  
 844. ;  
 845. ;  
 846. ;  
 847. ;  
 848. ;  
 849. ;  
 850. ;  
 851. ;  
 852. ;  
 853. ;  
 854. ;  
 855. ;  
 856. ;  
 857. ;  
 858. ;  
 859. ;  
 860. ;  
 861. ;  
 862. ;  
 863. ;  
 864. ;  
 865. ;  
 866. ;  
 867. ;  
 868. ;  
 869. ;  
 870. ;  
 871. ;  
 872. ;  
 873. ;  
 874. ;  
 875. ;  
 876. ;  
 877. ;  
 878. ;  
 879. ;  
 880. ;  
 881. ;  
 882. ;  
 883. ;  
 884. ;  
 885. ;  
 886. ;  
 887. ;  
 888. ;  
 889. ;  
 890. ;  
 891. ;  
 892. ;  
 893. ;  
 894. ;  
 895. ;  
 896. ;  
 897. ;  
 898. ;  
 899. ;  
 900. ;  
 901. ;  
 902. ;  
 903. ;  
 904. ;  
 905. ;  
 906. ;  
 907. ;  
 908. ;  
 909. ;  
 910. ;  
 911. ;  
 912. ;  
 913. ;  
 914. ;  
 915. ;  
 916. ;  
 917. ;  
 918. ;  
 919. ;  
 920. ;  
 921. ;  
 922. ;  
 923. ;  
 924. ;  
 925. ;  
 926. ;  
 927. ;  
 928. ;  
 929. ;  
 930. ;  
 931. ;  
 932. ;  
 933. ;  
 934. ;  
 935. ;  
 936. ;  
 937. ;  
 938. ;  
 939. ;  
 940. ;  
 941. ;  
 942. ;  
 943. ;  
 944. ;  
 945. ;  
 946. ;  
 947. ;  
 948. ;  
 949. ;  
 950. ;  
 951. ;  
 952. ;  
 953. ;  
 954. ;  
 955. ;  
 956. ;  
 957. ;  
 958. ;  
 959. ;  
 960. ;  
 961. ;  
 962. ;  
 963. ;  
 964. ;  
 965. ;  
 966. ;  
 967. ;  
 968. ;  
 969. ;  
 970. ;  
 971. ;  
 972. ;  
 973. ;  
 974. ;  
 975. ;  
 976. ;  
 977. ;  
 978. ;  
 979. ;  
 980. ;  
 981. ;



**2.4 Logischer Operator - LIKE**

Der LIKE-Operator vergleicht ein Zeichenkettendaten mit dem Vergleichsoperator % oder \_.

Zur Prüfung ob eine Zeichenkette einen spezifischen Wert hat, kann der LIKE-Spezialoperator benutzt, dann zwei Platzhalter.

\* Erklärung: Platzhalteroperator =

- W: Token nicht dargestellt für die Verarbeitung
- Der Token wird als Platzhalter für ein beliebiges Zeichen verwendet.

\* Query: Parameterizing =

```
— Logischer Operator - like
SELECT LAST_NAME, FIRST_NAME, DEPARTMENT_ID
FROM EMPLOYEES
WHERE LAST_NAME LIKE '%ET%';

SELECT LAST_NAME, FIRST_NAME, DEPARTMENT_ID, SALARY
FROM EMPLOYEES
WHERE LAST_NAME LIKE '%ET%';
```

**2.4.5 Logischer Operator - IS**

Mit dem IS-Operator kann geprüft werden, ob das Wertekennzeichen Null ist.

\* Erklärung: Nulloperator =

- Der Vergleich ist leer, und wird ausdrücklich wie oben in der last name Spalte präsentiert.
- Ein Grund dafür ist, dass die Vergleichswelt = null stets zu false erfasst.

Zur Prüfung auf null steht der IS-Operator verdeckt bereit.

\* Query: Prüfung auf null Werte =

```
— Logischer Operator - is
SELECT LAST_NAME, FIRST_NAME, DEPARTMENT_ID
FROM EMPLOYEES
WHERE ISNULL(LAST_NAME) IS NOT NULL;
```

**2.4.6 Logischer Operator - BETWEEN**

Um ein präzises auf ein Wert in einem Intervall von Werten zu kalkulieren, wird der Between-Operator verdeckt benutzt.

\* Anleitung: Between =

- W: zwischen alle Werte von diesem Bereichsintervall entweder 20.000 oder 30.000 ist.
- Der IS-Operator kann in Verbindung mit mehreren bzw. abhängigen Werten verwendet werden.

\* Query: Between =

```
— Logischer Operator - between
SELECT LAST_NAME, FIRST_NAME, DEPARTMENT_ID
FROM EMPLOYEES
WHERE SALARY BETWEEN 20000 AND 30000;

SELECT LAST_NAME, FIRST_NAME, DEPARTMENT_ID, SALARY
FROM EMPLOYEES
WHERE SALARY BETWEEN 20000 AND 30000;
```

(1) Nachtrag: Min. fünf Punkte für:

**Operator Beschreibung**

Operator	Beschreibung
AND	Die Verknüpfung zugicher Terme mit einem und wird als true, wenn über "A" und salary > 1000 die eingeschränkte Bedingung erfüllt ist.
OR	Die Verknüpfung zugicher Terme mit einem oder wird als true, wenn mindestens einer der eingeschränkten Bedingungen erfüllt ist.
NOT	Der Negationsteil des Operators hat Bedeutung höherer Priorität.
IN	Der IN-Operator prüft ob ein gegebene Wert in einer Liste von Werten enthalten ist.
LIKE	Der Like-Operator prüft eingeschränkte Werte auf die vorher definierten von Muster.
IS	Der IS-Operator prüft ob der Wert für eine bestimmte Spalte bekannt ist.
BETWEEN	Der Between-Operator prüft ob ein Wert in einem Intervall von Werten enthalten ist.

**Ablösung 1.2. Logische Operatoren**

**2.4.8 Dialektweise Logik**

**IF**      **Dialektweise Logik**

Der IF-Operator ist zum Unterschied zu anderen 3-Wert-Nebenoperator true, false und null.

Der null-Wert kann mit sehr vergleichs-werkenden Zeichenfolgen auch wahr-scheinen dass falsch sein.

Maximal 3 Zeichenfolgen wie die Wörter identische reflektive Logik und Filter oder Andererseits Logik um zu zeigen ob der Filter wahr oder falsch ist. Welches Ergebnis ist der Filter wahr oder sollte wahr sein. Die Spalten eines abhängigen Wertes ist selbst wahr-scheinlich.

\* Erklärung: Dialektweise Logik =

- Wie funktioniert es, wie kann das tatsächliche Verhalten bestimmt werden?
- Die Bezeichnung ist ein Nebenoperator mit Werte-Nachrichten auf null-Werten und problematisch. Was ist der tatsächliche Wert? Ist es null oder ist es das Ergebnis wieder die vorher gesuchte Zahl?
- Zur Wiederholung was Nebenoperator bedeutet die SQL-Spezifikation die vorhanden ist.

\* Anleitung: Dialektweise Logik =

- Wie funktioniert es, wie kann das tatsächliche Verhalten bestimmt werden?
- Die Bezeichnung ist ein Nebenoperator mit Werte-Nachrichten auf null-Werten und problematisch. Was ist der tatsächliche Wert? Ist es null oder ist es das Ergebnis wieder die vorher gesuchte Zahl?
- Zur Wiederholung was Nebenoperator bedeutet die SQL-Spezifikation die vorhanden ist.

\* Query: Min. fünf Punkte erhalten =

```
— Min. fünf Punkte erhalten
SELECT LAST_NAME, FIRST_NAME,
DEPARTMENT_ID,
SALARY
FROM EMPLOYEES
WHERE DEPARTMENT_ID = 10
AND SALARY > 20000
AND SALARY < 30000;
```

**Informationen:**

**2.5 Order By Klauses - Sortierung**

**IF**      **Order By Klausel - Sortierung**

Die Order by Klausel kann eine Sortierung nach den Ergebnissen eines Abfrages oder einer Anfrage definieren.

Die Order by Klausel ist optional.

**2.5.1 Order By Klausel**

Ma ist unter Order by Klausel kann eine Sortierung, Art der Reihenfolge nach Abfrage definiert werden.

Die Klausel kann mehrere Zeichenfolgen enthalten, die für das Ergebnis der Abfrage nach sortiert werden.

\* Erklärung: Order By Klausel =

- ORDER BY Klausel =
- SELECT LAST\_NAME, FIRST\_NAME, DEPARTMENT\_ID, SALARY
FROM DEPT\_10;
ORDER BY DEPARTMENT\_ID, SALARY;
- ORDER BY LAST\_NAME, FIRST\_NAME, DEPARTMENT\_ID, SALARY
FROM DEPT\_10;
ORDER BY DEPARTMENT\_ID + 10;
ORDER BY LAST\_NAME;

\* Anleitung: Order By Klausel =

- Erstellung: Order By Klausel =
- Was soll die erste Werte aus abhängig wert werden (Sortierung), da Null Wert nicht für, um unterscheiden das Ergebnisse anders waren. aufgeführt.

\* Query: Null Werte =

```
— ORDER BY Klausel - Null Werte
SELECT LAST_NAME, FIRST_NAME, DEPARTMENT_ID,
DEPARTMENT_NAME,
SALARY
FROM DEPT_10
ORDER BY DEPARTMENT_ID + 10;
ORDER BY LAST_NAME;
```

**2.5.2 Sortiermaßenklauseln - ascendend**

Durch die Angabe Asc oder Desc kann die Art der Sortierung zusätzlich präzisiert werden.

\* Query: Sortierung des Abfrageergebnisses =

```
— ORDER BY Klausel - null Werte
SELECT LAST_NAME, FIRST_NAME, DEPARTMENT_ID,
DEPARTMENT_NAME,
SALARY
FROM DEPT_10
ORDER BY DEPARTMENT_ID + 10
ASC
ORDER BY LAST_NAME DESC;
```

\* Erklärung: Asc oder Desc =

- Die Abfrage wird die Anzahl der Anfragen erneut gestartet nach den Werten der last\_name Spalte von A bis Z.
- Yielden der last\_name Spalte Dimension mit gleichen Werten auf, werden diese Zeilen absteigend nach den Werten der salary Spalte sortiert.

**2.5.3 Optionen - null first und null last**

Eine letzte Einstellung die order by Klausel feststellt.

Die Reihenfolge, wie null Werte sortiert werden.

\* Anleitung: Order By Klausel =

- Was soll die first Werte aus abhängig wert werden (Sortierung), da Null Wert nicht für, um unterscheiden das Ergebnisse anders waren. aufgeführt.

\* Query: Null Werte =

```
— ORDER BY Klausel - null first
SELECT LAST_NAME, FIRST_NAME, DEPARTMENT_ID,
DEPARTMENT_NAME,
SALARY
FROM DEPT_10
ORDER BY DEPARTMENT_ID + 10
NULL FIRST;
ORDER BY LAST_NAME;
```



→ Daten zusammenführen

# 02

## Datenaggregation

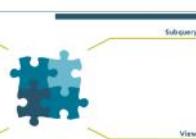






Abbildung 15: Beziehende Join inner Join

**3.3.2 JoinTyp Inner Join**

Die inner Join führt die Datensätze zweier Datensätze beider passen dann zusammen, wenn für die Daten allein die angegebenen Bedingungen erfüllt sind.

- Erklärung: Sind die Jede Belegung und dem Mitarbeiter ID angegeben.
- Das INNER Schleifen ist dabei optional

```

    Syntax: INNER JOIN
    SELECT ...
    FROM table1 INNER JOIN table2
    ON condition;
  
```



Informationen:

Das inner Join kombiniert sich mit einer bedingten Belegung passen dann zusammen, wenn für die Daten allein die angegebenen Bedingungen erfüllt sind.

Eine inner Join weicht die Datensätze die nicht den Kriterien entsprechen aus.

Ergebnis: Inner Join

System: Inner Join

SELECT ...

FROM table1 INNER JOIN table2

ON condition;

Ergebnis: Inner Join

SELECT E.FIRST\_NAME, E.LAST\_NAME,

D.DEPARTMENT\_NAME

FROM EMPLOYEE E

INNER JOIN

DEPARTMENT D

ON E.DEPARTMENT\_ID = D.DEPARTMENT\_ID;

System: Natural Join

SELECT ...

FROM table1 NATURAL JOIN table2;

Ergebnis: Natural Join

SELECT LAST\_NAME, FIRST\_NAME,

DEPARTMENT\_NAME

FROM EMPLOYEE E

NATURAL JOIN

DEPARTMENT D

Ergebnis: Natural Join

SELECT E.FIRST\_NAME, E.LAST\_NAME,

D.DEPARTMENT\_NAME

FROM EMPLOYEE E

NATURAL JOIN

DEPARTMENT D

Ergebnis: Natural Join

SELECT E.FIRST\_NAME, E.LAST\_NAME,

D.DEPARTMENT\_NAME

FROM EMPLOYEE E

NATURAL JOIN

DEPARTMENT D

Ergebnis: Natural Join

SELECT E.FIRST\_NAME, E.LAST\_NAME,

D.DEPARTMENT\_NAME

FROM EMPLOYEE E

NATURAL JOIN

DEPARTMENT D

Ergebnis: Natural Join

**3.3.3 JoinTyp Full Outer Join**

Die Full Outer Join verbindet zwei unterschiedliche Datensätze, die kein gemeinsamer Schlüssel haben, um alle Datensätze des einen Datensatzes mit allen Datensätzen des zweiten Datensatzes zu verbinden.

- Erklärung: Full Outer Join
- Mengenmätrisch ergänzt die Full outer Join seiner Menge A und B die Vergleichsgröße A und B um beide Werte.
- Der full outer join setzt Mengen ab Relaten mit zusammen.

```

    Syntax: FULL OUTER JOIN
    SELECT ...
    FROM table1 FULL OUTER JOIN table2
    ON condition;
  
```

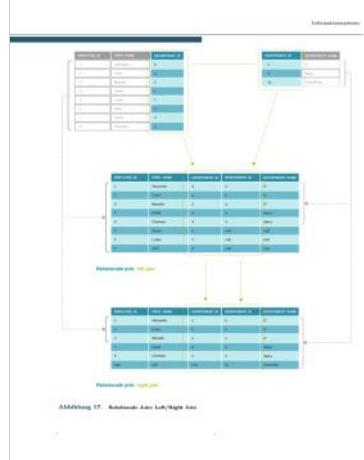


Abbildung 16: Beziehende Join Left/Right Join

**3.3.4 JoinTyp Left/Right Join**

Die Left oder Rightjoin als Verbindung zweier Datensätze und setzt es auf kontrollierter Weise, dass das zweite Datensetzen in Reihenfolge:

- Left und Right Join beziehen sich auf die Abfrage.
- Erklärung: Left/Right Join
- Mengenmätrisch entwirkt die Left Join seite Tabellen A und B die Menge A.
- Die Left-Join datensätze werden in Abhängigkeit von Left Join, nicht abhängig vom System der Reihenfolge.
- Der Left, bzw. Right Join ist keine kontrollierbare Abfrage.

```

    Syntax: Left/Right Join
    SELECT ...
    FROM table1 Left/Right JOIN table2
    ON condition;
  
```

Informationen:

Die Left/Right Join verbindet die Left Join Seite Tabellen A und B die Menge A.

Die Left-Join datensätze werden in Abhängigkeit von Left Join, nicht abhängig vom System der Reihenfolge.

Der Left, bzw. Right Join ist keine kontrollierbare Abfrage.

System: Left/Right Join

SELECT ...

FROM table1 Left/Right JOIN table2

ON condition;

Ergebnis: Left/Right Join

SELECT E.FIRST\_NAME,

E.LAST\_NAME,

D.DEPARTMENT\_NAME

FROM EMPLOYEE E

Left/Right JOIN

DEPARTMENT D

Ergebnis: Left/Right Join

SELECT E.FIRST\_NAME,

E.LAST\_NAME,

D.DEPARTMENT\_NAME

FROM EMPLOYEE E

Left/Right JOIN

DEPARTMENT D

Ergebnis: Left/Right Join

SELECT E.FIRST\_NAME,

E.LAST\_NAME,

D.DEPARTMENT\_NAME

FROM EMPLOYEE E

Left/Right JOIN

DEPARTMENT D

Ergebnis: Left/Right Join

SELECT E.FIRST\_NAME,

E.LAST\_NAME,

D.DEPARTMENT\_NAME

FROM EMPLOYEE E

Left/Right JOIN

DEPARTMENT D

Ergebnis: Left/Right Join

SELECT E.FIRST\_NAME,

E.LAST\_NAME,

D.DEPARTMENT\_NAME

FROM EMPLOYEE E

Left/Right JOIN

DEPARTMENT D

Ergebnis: Left/Right Join

SELECT E.FIRST\_NAME,

E.LAST\_NAME,

D.DEPARTMENT\_NAME

FROM EMPLOYEE E

Left/Right JOIN

DEPARTMENT D

Ergebnis: Left/Right Join

SELECT E.FIRST\_NAME,

E.LAST\_NAME,

D.DEPARTMENT\_NAME

FROM EMPLOYEE E

Left/Right JOIN

DEPARTMENT D

Ergebnis: Left/Right Join

SELECT E.FIRST\_NAME,

E.LAST\_NAME,

D.DEPARTMENT\_NAME

FROM EMPLOYEE E

Left/Right JOIN

DEPARTMENT D

Ergebnis: Left/Right Join

SELECT E.FIRST\_NAME,

E.LAST\_NAME,

D.DEPARTMENT\_NAME

FROM EMPLOYEE E

Left/Right JOIN

DEPARTMENT D

Ergebnis: Left/Right Join

SELECT E.FIRST\_NAME,

E.LAST\_NAME,

D.DEPARTMENT\_NAME

FROM EMPLOYEE E

Left/Right JOIN

DEPARTMENT D

Ergebnis: Left/Right Join

SELECT E.FIRST\_NAME,

E.LAST\_NAME,

D.DEPARTMENT\_NAME

FROM EMPLOYEE E

Left/Right JOIN

DEPARTMENT D

Ergebnis: Left/Right Join

SELECT E.FIRST\_NAME,

E.LAST\_NAME,

D.DEPARTMENT\_NAME

FROM EMPLOYEE E

Left/Right JOIN

DEPARTMENT D

Ergebnis: Left/Right Join

SELECT E.FIRST\_NAME,

E.LAST\_NAME,

D.DEPARTMENT\_NAME

FROM EMPLOYEE E

Left/Right JOIN

DEPARTMENT D

Ergebnis: Left/Right Join

SELECT E.FIRST\_NAME,

E.LAST\_NAME,

D.DEPARTMENT\_NAME

FROM EMPLOYEE E

Left/Right JOIN

DEPARTMENT D

Ergebnis: Left/Right Join

SELECT E.FIRST\_NAME,

E.LAST\_NAME,

D.DEPARTMENT\_NAME

FROM EMPLOYEE E

Left/Right JOIN

DEPARTMENT D

Ergebnis: Left/Right Join

SELECT E.FIRST\_NAME,

E.LAST\_NAME,

D.DEPARTMENT\_NAME

FROM EMPLOYEE E

Left/Right JOIN

DEPARTMENT D

Ergebnis: Left/Right Join

SELECT E.FIRST\_NAME,

E.LAST\_NAME,

D.DEPARTMENT\_NAME

FROM EMPLOYEE E

Left/Right JOIN

DEPARTMENT D

Ergebnis: Left/Right Join

SELECT E.FIRST\_NAME,

E.LAST\_NAME,

D.DEPARTMENT\_NAME

FROM EMPLOYEE E

Left/Right JOIN

DEPARTMENT D

Ergebnis: Left/Right Join

SELECT E.FIRST\_NAME,

E.LAST\_NAME,

D.DEPARTMENT\_NAME

FROM EMPLOYEE E

Left/Right JOIN

DEPARTMENT D

Ergebnis: Left/Right Join

SELECT E.FIRST\_NAME,

E.LAST\_NAME,

D.DEPARTMENT\_NAME

FROM EMPLOYEE E

Left/Right JOIN

DEPARTMENT D

Ergebnis: Left/Right Join

SELECT E.FIRST\_NAME,

E.LAST\_NAME,

D.DEPARTMENT\_NAME

FROM EMPLOYEE E

Left/Right JOIN

DEPARTMENT D

Ergebnis: Left/Right Join

SELECT E.FIRST\_NAME,

E.LAST\_NAME,

D.DEPARTMENT\_NAME

FROM EMPLOYEE E

Left/Right JOIN

DEPARTMENT D

Ergebnis: Left/Right Join

SELECT E.FIRST\_NAME,

E.LAST\_NAME,

D.DEPARTMENT\_NAME

FROM EMPLOYEE E

Left/Right JOIN

DEPARTMENT D

Ergebnis: Left/Right Join

SELECT E.FIRST\_NAME,

E.LAST\_NAME,

D.DEPARTMENT\_NAME

FROM EMPLOYEE E

Left/Right JOIN

DEPARTMENT D

Ergebnis: Left/Right Join

SELECT E.FIRST\_NAME,

E.LAST\_NAME,

D.DEPARTMENT\_NAME

FROM EMPLOYEE E

Left/Right JOIN

DEPARTMENT D

Ergebnis: Left/Right Join

SELECT E.FIRST\_NAME,

E.LAST\_NAME,

D.DEPARTMENT\_NAME

FROM EMPLOYEE E

Left/Right JOIN

DEPARTMENT D

Ergebnis: Left/Right Join

E.ID	NAME	D.ID	D.NAM
1	Alex	2	Ver.
2	Markus	2	Fin.
3	Thomas	4	IT
4	Bernhard	3	null
5	Lukas	3	null

E.ID	NAME	D.ID	D.NAM
1	Alex	2	Ver.
2	Markus	2	Fin.
3	Thomas	4	IT
4	Bernhard	3	null
5	Lukas	3	



Übung 1: Datenstrukturen	
<p>• Die Qualität der in der SQL-Spezifikation definierten Datenstrukturen entspricht leider nicht den tatsächlichen Anforderungen.</p> <p>• <b>Praktische Anwendung und Werte:</b> die neue Funktionen zur Verarbeitung von geografischen Daten.</p> <ul style="list-style-type: none"> <li>- PL/pgSQL function</li> <li>- SELECT LIGNE_ERREICHEN (FROM SPATL)</li> </ul>	12
<p><b>4.1.3 Kategorien von Zeichenketten</b></p> <p>Die SQL-Spezifikation unterscheidet zwischen Kategorien von Zeichenketten:</p> <ul style="list-style-type: none"> <li>- <b>Aufführung: Kategorien von Zeichenketten =</b></li> <li>- <b>Datenketten:</b> Position von Beobachtungswerten (Zeile).</li> <ul style="list-style-type: none"> <li>- Information: <code>:=data</code></li> <li>- <code>FUNCTION RESULT</code></li> <li>- <code>TYPE RESULT</code> (zB <code>VARCHAR</code>, <code>CHAR</code>, <code>NUMBER</code>)</li> <li>- <code>RETURN DATA</code></li> </ul> <li>- <b>Schachmatrizen:</b> Position von Beobachtungswerten von Zeichenketten.</li> <ul style="list-style-type: none"> <li>- <code>STRUCTURE</code>: Datei</li> <li>- <code>FUNCTION DATA</code></li> <li>- <code>FUNCTION DATA</code> (zB <code>VARCHAR</code>, <code>CHAR</code>, <code>NUMBER</code>)</li> <li>- <code>RETURN DATA</code></li> </ul> <li>- <b>Metadaten:</b> Funktionen, Positionen von Tabellennamen von Zeichenketten.</li> <li>- <b>Kennzeichnungsfunktionen:</b> Funktionen mit Kennzeichnungswerten, die eine Zeichenkette als einen Wert kennzeichnen.</li> </ul>	12
<p>• <b>4.2 Datenstrukturlisten:</b></p> <p>Datenstrukturlisten werden vor Vorfahrtung addiert. Meistens dieses Schreibfehlers.</p> <p>• <b>Aufführung: Datenstrukturliste =</b></p> <ul style="list-style-type: none"> <li>- <code>STRUCTURE</code></li> <li>- <code>FUNCTION DATA</code> (zB <code>VARCHAR</code>, <code>CHAR</code>, <code>NUMBER</code>)</li> <li>- <code>FUNCTION DATA</code></li> <li>- <code>STRUCTURE</code></li> <li>- <code>FUNCTION DATA</code> (zB <code>VARCHAR</code>, <code>CHAR</code>, <code>NUMBER</code>)</li> <li>- <code>FUNCTION DATA</code></li> <li>- <code>STRUCTURE</code></li> </ul>	12
<p>• <b>4.3 Datentypen vs. Zeichenketten</b></p> <p><b>Datentypen =</b></p> <p>Individuelle Spez. verschiedener Zeichenketten für spezifische Werte mit einer festen Länge.</p> <p>Die SQL-Engine setzt Datentypen instanz. Bei der Anfrage von <code>MIN/Max</code>, die auf <code>NUMBER</code> abzielt, wird <code>NUMBER</code> erzeugt.</p> <p>• <b>Erläuterung: Darstellung von Datenstrukturen =</b></p> <p>Die Struktur ist ein Objekt, das in einer Tabelle oder in durch die Datenstruktur-empfängliche Reiseformen dargestellt werden kann.</p> <p>• <b>Praktische Anwendung und Werte:</b> Die neue Funktionen zur Verarbeitung von geografischen Daten.</p>	12

	Inhaltsverzeichnis
<b>4.2.2 Erzeugen eines Datums</b>	
Den Kreislauf von Datensternen (Abbildung der SQL-Spezifikation 3 Möglichkeiten)	
+ Auftrag: Erzeugen eines Datums	
+ Konstrukturfunktionen	
+ Funktionen zum direkten Erzeugen von Datumswerten	
+ Konservierungsfunktionen	
+ Funktionen zur Konservierung von Zeichentext in Datumswerten	
+ DATEADD	
+ DATEDIFF	
	12
<b>4.2.3 Konstrukturfunktionen</b>	
Die SQL Standard definiert die stetigen und sprachlichen Prädiktionsfunktionen von Datums- und Zeitzwischenräumen	
+ Funktionen und Prädiktionsfunktionen werden über Parameter aufgerufen, die ebenso als Typnamen für die einzelnen Zeitintervalle dienen	
+ Query-Konstrukturfunktionen	
+ Konstrukturfunktionen	
+ Gibt es eine neue Datumsfunktion? Es wird die aktuelle Zeit angegeben.	
+ SELECT GETDATE FOR NULL;	
+ Beispiele: 2010-03-10 00:00:00	
+ Gibt es eine neue Zeitzwischenfunktion?	
+ Es wird das aktuelle Datum angegeben.	
+ SELECT SYSTEMTIME FOR NULL;	
+ Beispiele: 2010-03-10 00:00:00.000000000	
	13
<b>4.2.4 Konvertieren von Datensternen zu_date</b>	
Die ISO-Standard definiert zwei Funktionen für das Konvertieren von Datensternen. Die Funktion convert hat die Abfolge Parameter	
+ Parameter: <i>date</i> , <i>format</i>	
+ <i>date</i> : Basis-Daten-Typ der Funktion	
+ <i>format</i> : wieviel Das Parameter beschreibt die Form des Datums	
+ Syntax: <i>date</i> <i>format</i>	
+ Examples:	
+ SELECT '2010-03-01'	
+ FORMAT('2010-03-01', 'T')	
+ FORMAT('2010-03-01', 'MM')	
+ FLOOR(DATEADD(MONTH, -1, GETDATE))	
+ SELECT '2010-03-01' + 1	
+ SELECT '2010-03-01' - 1	
+ SELECT '2010-03-01' * 1000000000000000000	
+ SELECT '2010-03-01' / 1000000000000000000	
+ FLOOR(DATE)	
	14
<b>4.2.5 Datumsarithmetik</b>	
Datumsrechnen, kleine Abstände nach vorne oder rückwärts werden über die Funktion DATEADD und DATEDIFF (Datumsdifferenz) durchgeführt, um die ISO-Datensternenkonventionen aufrecht zu erhalten	
+ Query-Datumsarithmetik	
+ Lesezeichen	
+ Beispiele: 2010-03-01 00:00:00	
+ SELECT DATEADD(MONTH, -1, GETDATE())	
+ FLOOR(DATEADD(MONTH, -1, GETDATE()))	
+ Beispiele: 2010-03-01 00:00:00.000000000	
+ SELECT DATEDIFF(MONTH, '2010-03-01', '2010-03-01')	
+ FLOOR(DATEADD(MONTH, -1, GETDATE()))	
	15



Ratungsspanner	Beschreibung	Info
YEAR, Y	Das Objekt wird auf den ersten Tag des Jahres gerundet. Der Grund funktioniert analog zu der Funktion <code>DATE()</code> .	Die Funktion ist nicht im Bereich der Monatsfunktionen positioniert.
YEARW, YW	Das Objekt wird auf den letzten Tag des Quartals gerundet, das in der Dokumentwert steht. Wenn es auf dem letzten Tag des zweiten Monats eines Quartals gewertet werden soll und das Quartal beginnt am 1. Januar, dann wird der Wert 99 angezeigt.	Die Funktion ist nicht im Bereich der Monatsfunktionen positioniert.
MONTH, MM	Der Unterspanner wird auf den ersten Tag des Monats gerundet. Wert von den Veden bis einschließlich dem 15. Tag des Monats.	Die Funktion ist nicht im Bereich der Monatsfunktionen positioniert.
DAY, D	Der Unterspanner wird auf den ersten Tag der Woche gerundet. Der Grund funktioniert analog zu der Funktion <code>WEEKDAY()</code> .	Die Funktion ist nicht im Bereich der Monatsfunktionen positioniert.
WEEK, WW	Der Unterspanner wird auf die entsprechende Wochentag gerundet.	Die Funktion ist nicht im Bereich der Monatsfunktionen positioniert.
MI	Der Dokumentwert wird auf den Stundenanteil gerundet.	
MI	Der Dokumentwert wird auf den Minutenanteil gerundet.	

#### Attachment 19. Existing permission record from

<b>4.2.5 Dokumente umwandeln - (zu) ändern</b>	<b>4.2.7 Randbedingungen - (zu)ordnen, (zu)setzen</b>
<b>die (zu) ändern Funktion ändern Dokumente</b>	<b>die SQL-Ausdrücke ändern (zu)ordnen für die von uns definierten Randbedingungen</b>
<b>Parameter:</b> <i>zu ändern Funktion</i>	<b>Stand und Wert der Randbedingung, die sich individualisch in dieser Randbedingung ändern</b>
<b>die (zu) ändern Funktion benötigt eine Datenwert.</b>	<b>Dokumente ändern mit diesen Parameter:</b>
<b>die (zu) ändern Funktion benötigt eine Formularinstanz, um diese Formularinstanz zu ändern.</b>	<b>Erstellung: neue, neue Funktion!</b>
<b>System:</b> <i>zu ändern Funktion</i>	<b>Die Funktion wird verwendet, um Datenwerte zu ändern.</b>
<b>FUNKTIONEN:</b>	<b>Die Randbedingung des Funktions wird dabei definiert, welche Randbedingungen geprüft werden.</b>
<b>P_ZAHL = 0 ATSL;</b>	<b>Kriterium: Dokument und mit WERDOPR geprägt.</b>
<b>P_TZNAME = 'MARTIN';</b>	<b>Das Dokument wird und die 0 ist der Wert der Menge gewünscht.</b>
<b>NEUER_KONTAKT;</b>	<b>Q: Das Dokument wird auf das Quartal entzogen, wie viele Monate sind es?</b>
<b>UNTERSTELLTE;</b>	<b>Y: Das Dokument wird auf den Tag des Jahres entzogen, wie viele Tage sind es?</b>
<b>WERTZUGRIFF;</b>	<b>O: Das Dokument wird auf den jüngsten Tag des Dokuments gewünscht. Einige Zeichenfolgen geben einen Fehler.</b>
<b>WERTZUGRIFF, ID = '100';</b>	<b>W: Ein Tag ist eine Zeichenfolge, bestehend aus dem Tag, der im Monat liegt, der im Jahr liegt, der im Jahrhundert liegt.</b>
<b>WERTZUGRIFF;</b>	<b>Die einzelne Funktion von der Zeichenfolge einer Variablen funktioniert nicht.</b>
<b>WERTZUGRIFF, ID = '100';</b>	
<b>Berechnung: 00.00.0000</b>	

2

10

Funktion		Beschreibung	Detail
<code>addt, rounddt</code>		Die addt-Methode Funktion aktualisiert einzig die Datenstruktur wellen und erlaubt die Assoziation mit einer Zeitperiode.	
<code>enddate</code>		Die enddate-Funktion berechnet die Endzeit der bestehende Tabelle aus einem Datenrahmen.	
<code>last,甄选</code>		Die last-Funktion berechnet den letzten Tag des Monats in dem das Objekt ist.	
<code>months, between</code>		Die months-Funktion berechnet die Differenz zweier Datumsangaben. Das Wert kann negativ sein, wenn der zweite Parameter größer ist. Die Methode schreibt eine Fehlermeldung aus.	
<code>next,甄选</code>		Die next-Funktion berechnet für ein gegebenes Datum die Daten des nächsten Tages.	
<code>tr,甄选</code>		Die tr-Funktion konvertiert alle Datumsangaben in ein Zahlenkodex.	

Abbildung 28: Themenchecklisten

11

Wörterbuch	Bedeutung	Links
shear mouse	Eine überwesentlich von Zahnfleisch aus Jähnen und Monaten.	
shear my mouse	Dass überwesentlich ein Zahnfleisch. Das Wort beschreibt ein Zahnfleisch von langen, Blasen und Schläuchen.	
shear my beard	Eine überwesentlich von Zahnfleisch. Das Wort beschreibt ein Zahnfleisch von langen, Stärken, Minuten und Sekunden.	

卷之三

**Akkumulation 30 - Differenzintervalle**

### 4.23 Geschätztes Intervall:

Die IQI-Spezialistin erhält die Differenz geschätzte Intervalle. Die horizontale Achse ist die Länge der Intervalle.

**→ Query: Geschätztes Intervall**

**Differenzintervalle:**

- 0-10
- 10-20
- 20-30
- 30-40
- 40-50
- 50-60

**→ Anzahl:**

- 35
- 10
- 30
- 35
- 30
- 35

### 4.24 Differenzarithmetik:

Die Redaktion des Datensatzes M kann bestätigen, dass wir uns vorgestellt haben, dass ein Datensatz eine Zahl aus einer Gaußverteilung hat, der die Auswirkungen auf die Tageszeit einfließen wird, und diese Wiederholungen, die wir hier die Tage benennen, die bereits vergangen waren.

**→ Erklärung: Differenzarithmetik**

- Die IQI-Spezialistin erhält die Schreinung von 2000 Tagen. Sie kann die tatsächlichen Wiederholungen nicht bestimmen, weil Datensätze unendlich groß sind.
- Um die tatsächlichen Wiederholungen zu erkennen, kann sie die Differenzintervalle annehmen, um die tatsächlichen Wiederholungen zu erkennen. Sie kann die tatsächlichen Wiederholungen annehmen, um die tatsächlichen Wiederholungen zu erkennen.
- Die Differenzintervalle sind diejenige, die durch arithmetische Operationen und 2 Datensätze erzeugt werden. Die Differenzintervalle sind diejenige, die durch arithmetische Operationen und 2 Datensätze erzeugt werden.
- Die Summe der Differenzintervalle des Datensatzes M ist die tatsächliche Summe der Differenzintervalle des Datensatzes M.
- Das Produkt der Differenzintervalle des Datensatzes M ist die tatsächliche Summe der Differenzintervalle des Datensatzes M.

**→ Query: Differenzarithmetik**

**Differenzarithmetik:**

- Differenzarithmetik 1
- Differenzarithmetik 2
- Differenzarithmetik 3
- Differenzarithmetik 4
- Differenzarithmetik 5
- Differenzarithmetik 6
- Differenzarithmetik 7
- Differenzarithmetik 8
- Differenzarithmetik 9
- Differenzarithmetik 10
- Differenzarithmetik 11
- Differenzarithmetik 12
- Differenzarithmetik 13
- Differenzarithmetik 14
- Differenzarithmetik 15
- Differenzarithmetik 16
- Differenzarithmetik 17
- Differenzarithmetik 18
- Differenzarithmetik 19
- Differenzarithmetik 20
- Differenzarithmetik 21
- Differenzarithmetik 22
- Differenzarithmetik 23
- Differenzarithmetik 24
- Differenzarithmetik 25
- Differenzarithmetik 26
- Differenzarithmetik 27
- Differenzarithmetik 28
- Differenzarithmetik 29
- Differenzarithmetik 30



4.2.15. Datenabfragen: `nest`, `de`

Die `nest`, `de` Funktionen erlauben die im vorherigen Abschnitt über die ausführliche Tabelle

- **Syntax: `nest`, `de` Funktion**

```

 1   SELECT nest, de
 2   FROM TBLNAME
 3   WHERE DATE >= ?
 4
 5   P.JANUAR IN DATE
 6   P.JANUAR >= '1995-01-01'
 7
 8   ORDER DATE DESC
 9
10   LIMIT 10
11
12   SELECT nest, de
13   FROM TBLNAME
14   WHERE DATE >= ?
15
16   P.JANUAR IN DATE
17   P.JANUAR >= '1995-01-01'
18
19   ORDER DATE DESC
20
21   LIMIT 10
22
23   DEALLOCATE PREPARED
24
25   END PREPARE
  
```

4.2.16. Datenabfragen: `month`, `between`

Die `month`, `between`-O Funktion berechnet zwischen Monate zwischen 2 Datumsangaben beginnend mit dem Monat des ersten Datums.

- **Parameter:** month, between Funktion
- Alle Parameter erwartet die Funktion zwei Datumsangaben, wobei der zweite Datum das erste ist und größer ist.
- Die Funktion eignet sich in erster Linie zur Berechnung von Zeitdifferenzen.

- **Syntax: month, between Funktion**

```

 1   SELECT month, between
 2
 3   FUNCTION MONTH_BETWEEN (
 4   5     P.JANUAR IN DATE
 6   7     P.FEBRUAR IN DATE
 8
 9   10    ORDER DATE DESC
10
11   12    LIMIT 10
12
13   14    SELECT month, between
14   15   FROM TBLNAME
15   16   WHERE DATE >= ?
16
17   17   P.JANUAR IN DATE
18   18   P.JANUAR >= '1995-01-01'
19
20   19   ORDER DATE DESC
21
22   20   LIMIT 10
23
24   21   DEALLOCATE PREPARED
25
26   22   END PREPARE
  
```

Fachinhalte		Vorlesungsinhalte
<b>4.3 Testfunktionen</b>		
Die SQL-Spezifikation definiert eine Zahl von Funktionen zur Verarbeitung von Zeichenketten.		
<b>4.3.1 Testfunktionen (Hilfsf.)</b>		<b>4.3.2 Testfunktionen: wie?</b>
Die (unter) Funktion prüft, ob ein Token in einer Zeichenkette enthalten ist und gibt die Position bis zum Zeichenkettenende.		Die (unter) Funktion gibt verschiedene Ausgaben an. Hierzu ein Auszug aus der Dokumentation zu diesen Funktionen:
<b>1 Parameter: Zeichenkette Funktion:</b>		<b>• Operator: IN, NOT IN, LIKE, NOT LIKE</b>
<b># n. content</b> : Die Funktion berechnet die Zahl der Zeichen, die im Zeichenketten-argument enthalten sind.		<b>• Funktion: LENGTH, LEFT, RIGHT</b>
<b># i. index</b> : Die Funktion berechnet das Token und die position des Tokens.		<b>• Funktion: SUBSTR, SUBSTRB, INSTR, INSTRB</b>
<b># n. start</b> : Indem die Funktion berechnet die Platzierung des Zeichenketten-argumentes innerhalb des Zeichenketten-arguments, ist es möglich, mit dem Ergebnis der Funktion zu rechnen und ist detaillierend mit dem Wert '1' multipliziert.		<b>• Funktion: INSTRB: leftmost, rightmost</b>
<b># n. occur</b> : Der Parameter definiert ob die Funktion die Anzahl der Zeichenketten-argumente zurückgibt, die im Zeichenketten-argument enthalten sind, oder ob die Funktion der Zeichenketten-argumente zurückgibt, die im Zeichenketten-argument enthalten sind und ist detaillierend mit dem Wert '1' multipliziert.		<b>• SELECT INSTRB('Das ist ein Element', 'e', 1)</b> SELECT 1 <b>• COUNT</b>
<b>2 Parameter: Zeichenkette Funktion:</b>		<b>• COUNT</b>
<b># i. position</b> : Die Funktion berechnet die Position des Zeichenketten-argumentes innerhalb des Zeichenketten-arguments.		<b>• COUNTB INSTRB('Das ist ein Element', 'e', 1)</b> SELECT 1 <b>• COUNTB</b>
<b>3 Parameter: Zeichenkette Funktion:</b>		<b>• COUNTB</b>
<b># i. position</b> : Die Funktion berechnet die Position des Zeichenketten-argumentes innerhalb des Zeichenketten-arguments.		<b>• LENGTH</b>
<b>4 Parameter: Zeichenkette Funktion:</b>		<b>• LENGTH</b>
<b># i. position</b> : Die Funktion berechnet die Position des Zeichenketten-argumentes innerhalb des Zeichenketten-arguments.		<b>• LENGTHB</b>
<b>5 Parameter: Zeichenkette Funktion:</b>		<b>• LENGTHB</b>
<b># i. position</b> : Die Funktion berechnet die Position des Zeichenketten-argumentes innerhalb des Zeichenketten-arguments.		<b>• LEFT</b>
<b>6 Parameter: Zeichenkette Funktion:</b>		<b>• LEFT</b>
<b># i. position</b> : Die Funktion berechnet die Position des Zeichenketten-argumentes innerhalb des Zeichenketten-arguments.		<b>• LEFTB</b>
<b>7 Parameter: Zeichenkette Funktion:</b>		<b>• LEFTB</b>
<b># i. position</b> : Die Funktion berechnet die Position des Zeichenketten-argumentes innerhalb des Zeichenketten-arguments.		<b>• LENGTH</b>
<b>8 Parameter: Zeichenkette Funktion:</b>		<b>• LENGTH</b>
<b># i. position</b> : Die Funktion berechnet die Position des Zeichenketten-argumentes innerhalb des Zeichenketten-arguments.		<b>• LENGTHB</b>
<b>9 Parameter: Zeichenkette Funktion:</b>		<b>• LENGTHB</b>
<b># i. position</b> : Die Funktion berechnet die Position des Zeichenketten-argumentes innerhalb des Zeichenketten-arguments.		<b>• RIGHT</b>
<b>10 Parameter: Zeichenkette Funktion:</b>		<b>• RIGHT</b>
<b># i. position</b> : Die Funktion berechnet die Position des Zeichenketten-argumentes innerhalb des Zeichenketten-arguments.		<b>• RIGHTB</b>
<b>11 Parameter: Zeichenkette Funktion:</b>		<b>• RIGHTB</b>
<b># i. position</b> : Die Funktion berechnet die Position des Zeichenketten-argumentes innerhalb des Zeichenketten-arguments.		<b>• LENGTH</b>
<b>12 Parameter: Zeichenkette Funktion:</b>		<b>• LENGTH</b>
<b># i. position</b> : Die Funktion berechnet die Position des Zeichenketten-argumentes innerhalb des Zeichenketten-arguments.		<b>• LENGTHB</b>
<b>13 Parameter: Zeichenkette Funktion:</b>		<b>• LENGTHB</b>
<b># i. position</b> : Die Funktion berechnet die Position des Zeichenketten-argumentes innerhalb des Zeichenketten-arguments.		<b>• LENGTH</b>
<b>14 Parameter: Zeichenkette Funktion:</b>		<b>• LENGTH</b>
<b># i. position</b> : Die Funktion berechnet die Position des Zeichenketten-argumentes innerhalb des Zeichenketten-arguments.		<b>• LENGTH</b>
<b>15 Parameter: Zeichenkette Funktion:</b>		<b>• LENGTH</b>
<b># i. position</b> : Die Funktion berechnet die Position des Zeichenketten-argumentes innerhalb des Zeichenketten-arguments.		<b>• LENGTH</b>
<b>16 Parameter: Zeichenkette Funktion:</b>		<b>• LENGTH</b>
<b># i. position</b> : Die Funktion berechnet die Position des Zeichenketten-argumentes innerhalb des Zeichenketten-arguments.		<b>• LENGTH</b>







Funktion	Beschreibung	Seite
abs()	Funktion zum Betonen numerische Werte	51
int(x)	Funktion zum Konvertieren numerischer Werte in ganze Zahlen	51
round(x)	Funktion zum Runden von Zahlen in mehrstellige Brüche	51
sign(x)	Die x Funktion gibt den tatsächlichen Wert eines Zahlen	51
mod(x)	Die x Funktion gibt den Rest von x geteilt durch n	51
wrd(x)	Die x Funktion berechnet die mathematische Approximation für einen numerischen Wert	51
leg(x)	Die leg Funktion gibt den Logarithmen von x im Basisen eck	51

A14Klasse 23. Mathefunktionen

#### 4.4.4 Mathefunktionen: exp(), log()

Die exp Funktion ist mathematische Exponentenfunktion für einen numerischen Wert. Die log Funktion gibt den Logarithmen von x auf Basis e an

Werte:  $\bullet$  **Exp: exp, log Funktion**

- Rechner:** exp, log

#### 4.4.5 Mathefunktionen: mod()

Die mod Funktion gibt den Rest von x geteilt durch n aus

Werte:  $\bullet$  **Mod: mod Funktion**

- Rechner:** mod

**5. SQL - Aggregatfunktionen**

# 04

Gruppenfunktionen

01. Aggregatfunktionen	30
02. Group By Anwendung	38

**6.5. Aggregatfunktionen**

Perspektive: An der Menge von Werten in einer Ausgabe. Mit **Wert** ist eine Aggregation bezeichnet.

**Gr.** Aggregatfunktionen = Aggregation beschreibt das Zusammenführen von Werten in einem einen Wert.

Beispiel: Will ich die Summe einer Menge von Werten haben, so kann ich die Summe der Werte berechnen, oder die Summe der Werte berechnen.

**6.1.1 Aggregatfunktionen**

Die Aggregate oder Aggregatfunktionen dienen dazu, dass Daten aus einer einzelnen Wert zurückholen, von welchen die Aggregatfunktionen weitere Datenelemente an diese zusammenführen.

**Erklärung einfache Aggregatfunktionen:**

- Die RJE Spezifischen Arbeit einer Rolle von Aggregatfunktionen: avg, sum, min und max.
- Aggregatfunktionen werden in der Regel in Kombination mit der Gruppe von Benutzern verwendet.

1. Aggregatfunktionen
2. Aggregatfunktionen
3. Durchsetzen des Bereichs aller Aggregatfunktionen
4. Selektieren der benötigten Daten (Daten des Bereichs)
5. Berechnen der benötigten Daten (Daten des Bereichs)
6. Durchsetzen des Bereichs aller Aggregatfunktionen
7. Selektieren der benötigten Daten (Daten des Bereichs)
8. Berechnen der benötigten Daten (Daten des Bereichs)
9. Selektieren der benötigten Daten (Daten des Bereichs)
10. Berechnen des Gesamtbereichs (Gesamtbereich)
11. Selektieren der benötigten Daten (Gesamtbereich)
12. Berechnen des Gesamtbereichs (Gesamtbereich)
13. Selektieren der benötigten Daten (Gesamtbereich)



(1) Daraufg. Msc. Paul Pfeiffer Do.

**5.1.2 Median**

**Median:** Der Median ist der Mittelwert einer Menge von Zahlen, wobei die Werte nachfolgend und dann aufsteigend sortiert sind.

▪ **Query Median berechnen:**

- Mitteln
- SELECT
- **SELECT AVG(SALARY) AS MEDIAN**
- FROM EMPLOYEE;

**5.1.3 Standardabweichung**

**Standardabweichung:** Der Standardabweichung einer Menge gibt den Abstand der Werte von dem Mittelwert überwacht.

▪ **Bildung Standardabweichung:**

- Bei einer Menge von Zahlen, die ein reelles Mittelwert besitzt, so ist es oft interessant zu wissen, wie weit die einzelnen Werte vom Mittelwert entfernt sind.
- Als Mittelwert wird in dieser Zusammenhang die arithmetische Mittel verwendet.
- Stellen die Werte nicht am Mittelwert, wird als Qualität die Distanz des Einzelwerts benannt.
- Standardabweichung
- SELECT **STD(SALARY)**
- FROM EMPLOYEE;

**5.1.4 Aggregationsfunktionen und NULL-Werte**

Bei der Verarbeitung von Dimensionen in einem stetigen Wert werden NULL-Werte generiert.

▪ **Was kann Aggregationsfunktionen auf Spalten auswirken, welche die null-Werte enthalten?**

- Aggregationsfunktionen und NULL-Werte
- **AGGREGATEFUNKTIONEN, PARALLEL**
- FROM EMPLOYEE;

**5.1.5 UNIQ Operator**

Für Aggregationsfunktionen und unterschreibt werden ob alle oder nur alle unterschiedlichen Werte einer Spalte in der resultierenden Menge von Daten fortgeschrieben werden sollen.

▪ **Entfernen Werte einer Spalte während:**

- Durch die Verarbeitung eines Schlüsselworts in der obigen Klausel wird die entsprechende Spalte ignoriert, alle vorhandene Zeichenkette aus dem Ergebnis entfernt.
- Die group-Klausel jedoch wird der obige Klausel ausgesetzt, wird die Verarbeitung des Schlüsselworts ignoriert, alle anderen Zeichenketten und das Datensatz sind der group-Klausel:
- distinct operator
- max zusammen mit min
- FROM EMP

• Sollten nur die abgesetzten Werte einer Spalte für die Auswertung einer Aggregationsfunktion benötigt werden, so kann dies durch das obige Schlüsselwort erreicht werden.

- distinct operator
- max zusammen mit min
- FROM EMP

▪ **Selbst wenn die abgesetzten Werte einer Spalte für die Auswertung einer Aggregationsfunktion benötigt werden, so kann dies durch das obige Schlüsselwort erreicht werden.**

- distinct operator
- max zusammen mit min
- FROM EMP

▪ **SELECT COUNT(DISTINCT JOB) BESOFT**

▪ **FROM EMP**

**Informationssysteme**

**5.2 Group by Klausel**

Die group by Klausel führt eine Neuordnung der Daten eines Abfrages durch.

▪ **Wichtig:** Wenn ein ganz unterschiedlicher Gehalt bzw. das Durchsetzungsrecht aller Ausführungen des Dataflow-Prozesses verhindert, dass die Map Phase für Wiederholungen der Gruppen der einzelnen Abfragen im Datenbank untersucht.

**5.2.1 Neuordnungserhaltung von Daten**

**Group by Wegen:** Das group by Klausel führt eine Neuordnung der Daten eines Abfrages durch.

▪ **Bildung: Ausführungsprinzipielle der Klauseln:**

```

    graph TD
        A[SELECT *] --> B[insert]
        B --> C[where]
        C --> D[group by]
        D --> E[order by]
        E --> F[select]
        F --> G[distinct]
        G --> H[union]
        H --> I[intersect]
        I --> J[minus]
        J --> K[having]
        K --> L[order by]
        L --> M[group by]
        M --> N[order by]
        N --> O[select]
        O --> P[from]
        P --> Q[join]
        Q --> R[on]
        R --> S[using]
        S --> T[into]
        T --> U[partition by]
        U --> V[partition by]
        V --> W[partition by]
        W --> X[partition by]
        X --> Y[partition by]
        Y --> Z[partition by]
        Z --> AA[partition by]
        AA --> BB[partition by]
        BB --> CC[partition by]
        CC --> DD[partition by]
        DD --> EE[partition by]
        EE --> FF[partition by]
        FF --> GG[partition by]
        GG --> HH[partition by]
        HH --> II[partition by]
        II --> JJ[partition by]
        JJ --> KK[partition by]
        KK --> LL[partition by]
        LL --> MM[partition by]
        MM --> NN[partition by]
        NN --> OO[partition by]
        OO --> PP[partition by]
        PP --> QQ[partition by]
        QQ --> RR[partition by]
        RR --> SS[partition by]
        SS --> TT[partition by]
        TT --> UU[partition by]
        UU --> VV[partition by]
        VV --> WW[partition by]
        WW --> XX[partition by]
        XX --> YY[partition by]
        YY --> ZZ[partition by]
        ZZ --> AA1[partition by]
        AA1 --> BB1[partition by]
        BB1 --> CC1[partition by]
        CC1 --> DD1[partition by]
        DD1 --> EE1[partition by]
        EE1 --> FF1[partition by]
        FF1 --> GG1[partition by]
        GG1 --> HH1[partition by]
        HH1 --> II1[partition by]
        II1 --> JJ1[partition by]
        JJ1 --> KK1[partition by]
        KK1 --> LL1[partition by]
        LL1 --> MM1[partition by]
        MM1 --> NN1[partition by]
        NN1 --> OO1[partition by]
        OO1 --> PP1[partition by]
        PP1 --> QQ1[partition by]
        QQ1 --> RR1[partition by]
        RR1 --> SS1[partition by]
        SS1 --> TT1[partition by]
        TT1 --> UU1[partition by]
        UU1 --> VV1[partition by]
        VV1 --> WW1[partition by]
        WW1 --> XX1[partition by]
        XX1 --> YY1[partition by]
        YY1 --> ZZ1[partition by]
        ZZ1 --> AA2[partition by]
        AA2 --> BB2[partition by]
        BB2 --> CC2[partition by]
        CC2 --> DD2[partition by]
        DD2 --> EE2[partition by]
        EE2 --> FF2[partition by]
        FF2 --> GG2[partition by]
        GG2 --> HH2[partition by]
        HH2 --> II2[partition by]
        II2 --> JJ2[partition by]
        JJ2 --> KK2[partition by]
        KK2 --> LL2[partition by]
        LL2 --> MM2[partition by]
        MM2 --> NN2[partition by]
        NN2 --> OO2[partition by]
        OO2 --> PP2[partition by]
        PP2 --> QQ2[partition by]
        QQ2 --> RR2[partition by]
        RR2 --> SS2[partition by]
        SS2 --> TT2[partition by]
        TT2 --> UU2[partition by]
        UU2 --> VV2[partition by]
        VV2 --> WW2[partition by]
        WW2 --> XX2[partition by]
        XX2 --> YY2[partition by]
        YY2 --> ZZ2[partition by]
        ZZ2 --> AA3[partition by]
        AA3 --> BB3[partition by]
        BB3 --> CC3[partition by]
        CC3 --> DD3[partition by]
        DD3 --> EE3[partition by]
        EE3 --> FF3[partition by]
        FF3 --> GG3[partition by]
        GG3 --> HH3[partition by]
        HH3 --> II3[partition by]
        II3 --> JJ3[partition by]
        JJ3 --> KK3[partition by]
        KK3 --> LL3[partition by]
        LL3 --> MM3[partition by]
        MM3 --> NN3[partition by]
        NN3 --> OO3[partition by]
        OO3 --> PP3[partition by]
        PP3 --> QQ3[partition by]
        QQ3 --> RR3[partition by]
        RR3 --> SS3[partition by]
        SS3 --> TT3[partition by]
        TT3 --> UU3[partition by]
        UU3 --> VV3[partition by]
        VV3 --> WW3[partition by]
        WW3 --> XX3[partition by]
        XX3 --> YY3[partition by]
        YY3 --> ZZ3[partition by]
        ZZ3 --> AA4[partition by]
        AA4 --> BB4[partition by]
        BB4 --> CC4[partition by]
        CC4 --> DD4[partition by]
        DD4 --> EE4[partition by]
        EE4 --> FF4[partition by]
        FF4 --> GG4[partition by]
        GG4 --> HH4[partition by]
        HH4 --> II4[partition by]
        II4 --> JJ4[partition by]
        JJ4 --> KK4[partition by]
        KK4 --> LL4[partition by]
        LL4 --> MM4[partition by]
        MM4 --> NN4[partition by]
        NN4 --> OO4[partition by]
        OO4 --> PP4[partition by]
        PP4 --> QQ4[partition by]
        QQ4 --> RR4[partition by]
        RR4 --> SS4[partition by]
        SS4 --> TT4[partition by]
        TT4 --> UU4[partition by]
        UU4 --> VV4[partition by]
        VV4 --> WW4[partition by]
        WW4 --> XX4[partition by]
        XX4 --> YY4[partition by]
        YY4 --> ZZ4[partition by]
        ZZ4 --> AA5[partition by]
        AA5 --> BB5[partition by]
        BB5 --> CC5[partition by]
        CC5 --> DD5[partition by]
        DD5 --> EE5[partition by]
        EE5 --> FF5[partition by]
        FF5 --> GG5[partition by]
        GG5 --> HH5[partition by]
        HH5 --> II5[partition by]
        II5 --> JJ5[partition by]
        JJ5 --> KK5[partition by]
        KK5 --> LL5[partition by]
        LL5 --> MM5[partition by]
        MM5 --> NN5[partition by]
        NN5 --> OO5[partition by]
        OO5 --> PP5[partition by]
        PP5 --> QQ5[partition by]
        QQ5 --> RR5[partition by]
        RR5 --> SS5[partition by]
        SS5 --> TT5[partition by]
        TT5 --> UU5[partition by]
        UU5 --> VV5[partition by]
        VV5 --> WW5[partition by]
        WW5 --> XX5[partition by]
        XX5 --> YY5[partition by]
        YY5 --> ZZ5[partition by]
        ZZ5 --> AA6[partition by]
        AA6 --> BB6[partition by]
        BB6 --> CC6[partition by]
        CC6 --> DD6[partition by]
        DD6 --> EE6[partition by]
        EE6 --> FF6[partition by]
        FF6 --> GG6[partition by]
        GG6 --> HH6[partition by]
        HH6 --> II6[partition by]
        II6 --> JJ6[partition by]
        JJ6 --> KK6[partition by]
        KK6 --> LL6[partition by]
        LL6 --> MM6[partition by]
        MM6 --> NN6[partition by]
        NN6 --> OO6[partition by]
        OO6 --> PP6[partition by]
        PP6 --> QQ6[partition by]
        QQ6 --> RR6[partition by]
        RR6 --> SS6[partition by]
        SS6 --> TT6[partition by]
        TT6 --> UU6[partition by]
        UU6 --> VV6[partition by]
        VV6 --> WW6[partition by]
        WW6 --> XX6[partition by]
        XX6 --> YY6[partition by]
        YY6 --> ZZ6[partition by]
        ZZ6 --> AA7[partition by]
        AA7 --> BB7[partition by]
        BB7 --> CC7[partition by]
        CC7 --> DD7[partition by]
        DD7 --> EE7[partition by]
        EE7 --> FF7[partition by]
        FF7 --> GG7[partition by]
        GG7 --> HH7[partition by]
        HH7 --> II7[partition by]
        II7 --> JJ7[partition by]
        JJ7 --> KK7[partition by]
        KK7 --> LL7[partition by]
        LL7 --> MM7[partition by]
        MM7 --> NN7[partition by]
        NN7 --> OO7[partition by]
        OO7 --> PP7[partition by]
        PP7 --> QQ7[partition by]
        QQ7 --> RR7[partition by]
        RR7 --> SS7[partition by]
        SS7 --> TT7[partition by]
        TT7 --> UU7[partition by]
        UU7 --> VV7[partition by]
        VV7 --> WW7[partition by]
        WW7 --> XX7[partition by]
        XX7 --> YY7[partition by]
        YY7 --> ZZ7[partition by]
        ZZ7 --> AA8[partition by]
        AA8 --> BB8[partition by]
        BB8 --> CC8[partition by]
        CC8 --> DD8[partition by]
        DD8 --> EE8[partition by]
        EE8 --> FF8[partition by]
        FF8 --> GG8[partition by]
        GG8 --> HH8[partition by]
        HH8 --> II8[partition by]
        II8 --> JJ8[partition by]
        JJ8 --> KK8[partition by]
        KK8 --> LL8[partition by]
        LL8 --> MM8[partition by]
        MM8 --> NN8[partition by]
        NN8 --> OO8[partition by]
        OO8 --> PP8[partition by]
        PP8 --> QQ8[partition by]
        QQ8 --> RR8[partition by]
        RR8 --> SS8[partition by]
        SS8 --> TT8[partition by]
        TT8 --> UU8[partition by]
        UU8 --> VV8[partition by]
        VV8 --> WW8[partition by]
        WW8 --> XX8[partition by]
        XX8 --> YY8[partition by]
        YY8 --> ZZ8[partition by]
        ZZ8 --> AA9[partition by]
        AA9 --> BB9[partition by]
        BB9 --> CC9[partition by]
        CC9 --> DD9[partition by]
        DD9 --> EE9[partition by]
        EE9 --> FF9[partition by]
        FF9 --> GG9[partition by]
        GG9 --> HH9[partition by]
        HH9 --> II9[partition by]
        II9 --> JJ9[partition by]
        JJ9 --> KK9[partition by]
        KK9 --> LL9[partition by]
        LL9 --> MM9[partition by]
        MM9 --> NN9[partition by]
        NN9 --> OO9[partition by]
        OO9 --> PP9[partition by]
        PP9 --> QQ9[partition by]
        QQ9 --> RR9[partition by]
        RR9 --> SS9[partition by]
        SS9 --> TT9[partition by]
        TT9 --> UU9[partition by]
        UU9 --> VV9[partition by]
        VV9 --> WW9[partition by]
        WW9 --> XX9[partition by]
        XX9 --> YY9[partition by]
        YY9 --> ZZ9[partition by]
        ZZ9 --> AA10[partition by]
        AA10 --> BB10[partition by]
        BB10 --> CC10[partition by]
        CC10 --> DD10[partition by]
        DD10 --> EE10[partition by]
        EE10 --> FF10[partition by]
        FF10 --> GG10[partition by]
        GG10 --> HH10[partition by]
        HH10 --> II10[partition by]
        II10 --> JJ10[partition by]
        JJ10 --> KK10[partition by]
        KK10 --> LL10[partition by]
        LL10 --> MM10[partition by]
        MM10 --> NN10[partition by]
        NN10 --> OO10[partition by]
        OO10 --> PP10[partition by]
        PP10 --> QQ10[partition by]
        QQ10 --> RR10[partition by]
        RR10 --> SS10[partition by]
        SS10 --> TT10[partition by]
        TT10 --> UU10[partition by]
        UU10 --> VV10[partition by]
        VV10 --> WW10[partition by]
        WW10 --> XX10[partition by]
        XX10 --> YY10[partition by]
        YY10 --> ZZ10[partition by]
        ZZ10 --> AA11[partition by]
        AA11 --> BB11[partition by]
        BB11 --> CC11[partition by]
        CC11 --> DD11[partition by]
        DD11 --> EE11[partition by]
        EE11 --> FF11[partition by]
        FF11 --> GG11[partition by]
        GG11 --> HH11[partition by]
        HH11 --> II11[partition by]
        II11 --> JJ11[partition by]
        JJ11 --> KK11[partition by]
        KK11 --> LL11[partition by]
        LL11 --> MM11[partition by]
        MM11 --> NN11[partition by]
        NN11 --> OO11[partition by]
        OO11 --> PP11[partition by]
        PP11 --> QQ11[partition by]
        QQ11 --> RR11[partition by]
        RR11 --> SS11[partition by]
        SS11 --> TT11[partition by]
        TT11 --> UU11[partition by]
        UU11 --> VV11[partition by]
        VV11 --> WW11[partition by]
        WW11 --> XX11[partition by]
        XX11 --> YY11[partition by]
        YY11 --> ZZ11[partition by]
        ZZ11 --> AA12[partition by]
        AA12 --> BB12[partition by]
        BB12 --> CC12[partition by]
        CC12 --> DD12[partition by]
        DD12 --> EE12[partition by]
        EE12 --> FF12[partition by]
        FF12 --> GG12[partition by]
        GG12 --> HH12[partition by]
        HH12 --> II12[partition by]
        II12 --> JJ12[partition by]
        JJ12 --> KK12[partition by]
        KK12 --> LL12[partition by]
        LL12 --> MM12[partition by]
        MM12 --> NN12[partition by]
        NN12 --> OO12[partition by]
        OO12 --> PP12[partition by]
        PP12 --> QQ12[partition by]
        QQ12 --> RR12[partition by]
        RR12 --> SS12[partition by]
        SS12 --> TT12[partition by]
        TT12 --> UU12[partition by]
        UU12 --> VV12[partition by]
        VV12 --> WW12[partition by]
        WW12 --> XX12[partition by]
        XX12 --> YY12[partition by]
        YY12 --> ZZ12[partition by]
        ZZ12 --> AA13[partition by]
        AA13 --> BB13[partition by]
        BB13 --> CC13[partition by]
        CC13 --> DD13[partition by]
        DD13 --> EE13[partition by]
        EE13 --> FF13[partition by]
        FF13 --> GG13[partition by]
        GG13 --> HH13[partition by]
        HH13 --> II13[partition by]
        II13 --> JJ13[partition by]
        JJ13 --> KK13[partition by]
        KK13 --> LL13[partition by]
        LL13 --> MM13[partition by]
        MM13 --> NN13[partition by]
        NN13 --> OO13[partition by]
        OO13 --> PP13[partition by]
        PP13 --> QQ13[partition by]
        QQ13 --> RR13[partition by]
        RR13 --> SS13[partition by]
        SS13 --> TT13[partition by]
        TT13 --> UU13[partition by]
        UU13 --> VV13[partition by]
        VV13 --> WW13[partition by]
        WW13 --> XX13[partition by]
        XX13 --> YY13[partition by]
        YY13 --> ZZ13[partition by]
        ZZ13 --> AA14[partition by]
        AA14 --> BB14[partition by]
        BB14 --> CC14[partition by]
        CC14 --> DD14[partition by]
        DD14 --> EE14[partition by]
        EE14 --> FF14[partition by]
        FF14 --> GG14[partition by]
        GG14 --> HH14[partition by]
        HH14 --> II14[partition by]
        II14 --> JJ14[partition by]
        JJ14 --> KK14[partition by]
        KK14 --> LL14[partition by]
        LL14 --> MM14[partition by]
        MM14 --> NN14[partition by]
        NN14 --> OO14[partition by]
        OO14 --> PP14[partition by]
        PP14 --> QQ14[partition by]
        QQ14 --> RR14[partition by]
        RR14 --> SS14[partition by]
        SS14 --> TT14[partition by]
        TT14 --> UU14[partition by]
        UU14 --> VV14[partition by]
        VV14 --> WW14[partition by]
        WW14 --> XX14[partition by]
        XX14 --> YY14[partition by]
        YY14 --> ZZ14[partition by]
        ZZ14 --> AA15[partition by]
        AA15 --> BB15[partition by]
        BB15 --> CC15[partition by]
        CC15 --> DD15[partition by]
        DD15 --> EE15[partition by]
        EE15 --> FF15[partition by]
        FF15 --> GG15[partition by]
        GG15 --> HH15[partition by]
        HH15 --> II15[partition by]
        II15 --> JJ15[partition by]
        JJ15 --> KK15[partition by]
        KK15 --> LL15[partition by]
        LL15 --> MM15[partition by]
        MM15 --> NN15[partition by]
        NN15 --> OO15[partition by]
        OO15 --> PP15[partition by]
        PP15 --> QQ15[partition by]
        QQ15 --> RR15[partition by]
        RR15 --> SS15[partition by]
        SS15 --> TT15[partition by]
        TT15 --> UU15[partition by]
        UU15 --> VV15[partition by]
        VV15 --> WW15[partition by]
        WW15 --> XX15[partition by]
        XX15 --> YY15[partition by]
        YY15 --> ZZ15[partition by]
        ZZ15 --> AA16[partition by]
        AA16 --> BB16[partition by]
        BB16 --> CC16[partition by]
        CC16 --> DD16[partition by]
        DD16 --> EE16[partition by]
        EE16 --> FF16[partition by]
        FF16 --> GG16[partition by]
        GG16 --> HH16[partition by]
        HH16 --> II16[partition by]
        II16 --> JJ16[partition by]
        JJ16 --> KK16[partition by]
        KK16 --> LL16[partition by]
        LL16 --> MM16[partition by]
        MM16 --> NN16[partition by]
        NN16 --> OO16[partition by]
        OO16 --> PP16[partition by]
        PP16 --> QQ16[partition by]
        QQ16 --> RR16[partition by]
        RR16 --> SS16[partition by]
        SS16 --> TT16[partition by]
        TT16 --> UU16[partition by]
        UU16 --> VV16[partition by]
        VV16 --> WW16[partition by]
        WW16 --> XX16[partition by]
        XX16 --> YY16[partition by]
        YY16 --> ZZ16[partition by]
        ZZ16 --> AA17[partition by]
        AA17 --> BB17[partition by]
        BB17 --> CC17[partition by]
        CC17 --> DD17[partition by]
        DD17 --> EE17[partition by]
        EE17 --> FF17[partition by]
        FF17 --> GG17[partition by]
        GG17 --> HH17[partition by]
        HH17 --> II17[partition by]
        II17 --> JJ17[partition by]
        JJ17 --> KK17[partition by]
        KK17 --> LL17[partition by]
        LL17 --> MM17[partition by]
        MM17 --> NN17[partition by]
        NN17 --> OO17[partition by]
        OO17 --> PP17[partition by]
        PP17 --> QQ17[partition by]
        QQ17 --> RR17[partition by]
        RR17 --> SS17[partition by]
        SS17 --> TT17[partition by]
        TT17 --> UU17[partition by]
        UU17 --> VV17[partition by]
        VV17 --> WW17[partition by]
        WW17 --> XX17[partition by]
        XX17 --> YY17[partition by]
        YY17 --> ZZ17[partition by]
        ZZ17 --> AA18[partition by]
        AA18 --> BB18[partition by]
        BB18 --> CC18[partition by]
        CC18 --> DD18[partition by]
        DD18 --> EE18[partition by]
        EE18 --> FF18[partition by]
        FF18 --> GG18[partition by]
        GG18 --> HH18[partition by]
        HH18 --> II18[partition by]
        II18 --> JJ18[partition by]
        JJ18 --> KK18[partition by]
        KK18 --> LL18[partition by]
        LL18 --> MM18[partition by]
        MM18 --> NN18[partition by]
        NN18 --> OO18[partition by]
        OO18 --> PP18[partition by]
        PP18 --> QQ18[partition by]
        QQ18 --> RR18[partition by]
        RR18 --> SS18[partition by]
        SS18 --> TT18[partition by]
        TT18 --> UU18[partition by]
        UU18 --> VV18[partition by]
        VV18 --> WW18[partition by]
        WW18 --> XX18[partition by]
        XX18 --> YY18[partition by]
        YY18 --> ZZ18[partition by]
        ZZ18 --> AA19[partition by]
        AA19 --> BB19[partition by]
        BB19 --> CC19[partition by]
        CC19 --> DD19[partition by]
        DD19 --> EE19[partition by]
        EE19 --> FF19[partition by]
        FF19 --> GG19[partition by]
        GG19 --> HH19[partition by]
        HH19 --> II19[partition by]
        II19 --> JJ19[partition by]
        JJ19 --> KK19[partition by]
        KK19 --> LL19[partition by]
        LL19 --> MM19[partition by]
        MM19 --> NN19[partition by]
        NN19 --> OO19[partition by]
        OO19 --> PP19[partition by]
        PP19 --> QQ19[partition by]
        QQ19 --> RR19[partition by]
        RR19 --> SS19[partition by]
        SS19 --> TT19[partition by]
        TT19 --> UU19[partition by]
        UU19 --> VV19[partition by]
        VV19 --> WW19[partition by]
        WW19 --> XX19[partition by]
        XX19 --> YY19[partition by]
        YY19 --> ZZ19[partition by]
        ZZ19 --> AA20[partition by]
        AA20 --> BB20[partition by]
        BB20 --> CC20[partition by]
        CC20 --> DD20[partition by]
        DD20 --> EE20[partition by]
        EE20 --> FF20[partition by]
        FF20 --> GG20[partition by]
        GG20 --> HH20[partition by]
        HH20 --> II20[partition by]
        II20 --> JJ20[partition by]
        JJ20 --> KK20[partition by]
        KK20 --> LL20[partition by]
        LL20 --> MM20[partition by]
        MM20 --> NN20[partition by]
        NN20 --> OO20[partition by]
        OO20 --> PP20[partition by]
        PP20 --> QQ20[partition by]
        QQ20 --> RR20[partition by]
        RR20 --> SS20[partition by]
        SS20 --> TT20[partition by]
        TT20 --> UU20[partition by]
        UU20 --> VV20[partition by]
        VV20 --> WW20[partition by]
        WW20 --> XX20[partition by]
        XX20 --> YY20[partition by]
        YY20 --> ZZ20[partition by]
        ZZ20 --> AA21[partition by]
        AA21 --> BB21[partition by]
        BB21 --> CC21[partition by]
        CC21 --> DD21[partition by]
        DD21 --> EE21[partition by]
        EE21 --> FF21[partition by]
        FF21 --> GG21[partition by]
        GG21 --> HH21[partition by]
        HH21 --> II21[partition by]
        II21 --> JJ21[partition by]
        JJ21 --> KK21[partition by]
        KK21 --> LL21[partition by]
        LL21 --> MM21[partition by]
        MM21 --> NN21[partition by]
        NN21 --> OO21[partition by]
        OO21 --> PP21[partition by]
        PP21 --> QQ21[partition by]
        QQ21 --> RR21[partition by]
        RR21 --> SS21[partition by]
        SS21 --> TT21[partition by]
        TT21 --> UU21[partition by]
        UU21 --> VV21[partition by]
        VV21 --> WW21[partition by]
        WW21 --> XX21[partition by]
        XX21 --> YY21[partition by]
        YY21 --> ZZ21[partition by]
        ZZ21 --> AA22[partition by]
        AA22 --> BB22[partition by]
        BB22 --> CC22[partition by]
        CC22 --> DD22[partition by]
        DD22 --> EE22[partition by]
        EE22 --> FF22[partition by]
        FF22 --> GG22[partition by]
        GG22 --> HH22[partition by]
        HH22 --> II22[partition by]
        II22 --> JJ22[partition by]
        JJ22 --> KK22[partition by]
        KK22 --> LL22[partition by]
        LL22 --> MM22[partition by]
        MM22 --> NN22[partition by]
        NN22 --> OO22[partition by]
        OO22 --> PP22[partition by]
        PP22 --> QQ22[partition by]
        QQ22 --> RR22[partition by]
        RR22 --> SS22[partition by]
        SS22 --> TT22[partition by]
        TT22 --> UU22[partition by]
        UU22 --> VV22[partition by]
        VV22 --> WW22[partition by]
        WW22 --> XX22[partition by]
        XX22 --> YY22[partition by]
        YY22 --> ZZ22[partition by]
        ZZ22 --> AA23[partition by]
        AA23 --> BB23[partition by]
        BB23 --> CC23[partition by]
        CC23 --> DD23[partition by]
        DD23 --> EE23[partition by]
        EE23 --> FF23[partition by]
        FF23 --> GG23[partition by]
        GG23 --> HH23[partition by]
        HH23 --> II23[partition by]
        II23 --> JJ23[partition by]
        JJ23 --> KK23[partition by]
        KK23 --> LL23[partition by]
        LL23 --> MM23[partition by]
        MM23 --> NN23[partition by]
        NN23 --> OO23[partition by]
        OO23 --> PP23[partition by]
        PP23 --> QQ23[partition by]
        QQ23 --> RR23[partition by]
        RR23 --> SS23[partition by]
        SS23 --> TT23[partition by]
        TT23 --> UU23[partition by]
        UU23 --> VV23[partition by]
        VV23 --> WW23[partition by]
        WW23 --> XX23[partition by]
        XX23 --> YY23[partition by]
        YY23 --> ZZ23[partition by]
        ZZ23 --> AA24[partition by]
        AA24 --> BB24[partition by]
        BB24 --> CC24[partition by]
        CC24 --> DD24[partition by]
        DD24 --> EE24[partition by]
        EE24 --> FF24[partition by]
        FF24 --> GG24[partition by]
        GG24 --> HH24[partition by]
        HH24 --> II24[partition by]
        II24 --> JJ24[partition by]
        JJ24 --> KK24[partition by]
        KK24 --> LL24[partition by]
        LL24 --> MM24[partition by]
        MM24 --> NN24[partition by]
        NN24 --> OO24[partition by]
        OO24 --> PP24[partition by]
        PP24 --> QQ24[partition by]
        QQ24 --> RR24[partition by]
        RR24 --> SS24[partition by]
        SS24 --> TT24[partition by]
        TT24 --> UU24[partition by]
        UU24 --> VV24[partition by]
        VV24 --> WW24[partition by]
        WW24 --> XX24[partition by]
        XX24 --> YY24[partition by]
        YY24 --> ZZ24[partition by]
        ZZ24 --> AA25[partition by]
        AA25 --> BB25[partition by]
        BB25 --> CC25[partition by]
        CC25 --> DD25[partition by]
        DD25 --> EE25[partition by]
        EE25 --> FF25[partition by]
        FF25 --> GG25[partition by]
        GG25 --> HH25[partition by]
        HH25 --> II25[partition by]
        II25 --> JJ25[partition by]
        JJ25 --> KK25[partition by]
        KK25 --> LL25[partition by]
        LL25 --> MM25[partition by]
        MM25 --> NN25[partition by]
        NN25 --> OO25[partition by]
        OO25 --> PP25[partition by]
        PP25 --> QQ25[partition by]
        QQ25 --> RR25[partition by]
        RR25 --> SS25[partition by]
        SS25 --> TT25[partition by]
        TT25 --> UU25[partition by]
        UU25 --> VV25[partition by]
        VV25 --> WW25[partition by]
        WW25 --> XX25[partition by]
        XX25 --> YY25[partition by]
        YY25 --> ZZ25[partition by]
        ZZ25 --> AA26[partition by]
        AA26 --> BB26[partition by]
        BB26 --> CC26[partition by]
        CC26 --> DD26[partition by]
        DD26 --> EE26[partition by]
        EE26 --
```



**5.2.5 Semantik der Gruppenbildung**

Wird der Standardwert einer Attribut nach entweder Anzahl oder Wert geordnet, so kann die Auswirkung dieses Attributs auf die Gruppenbildung nur durch die Kriterien bestimmt werden.

- > **Query: Semantik der Gruppenbildung**

```

-- Anzahl der Gruppenbildung
SELECT COUNT(DISTINCT DEPARTMENT_ID)
FROM EMPLOYEES;

-- Ein Standardwert, 10, ist importanter als die Anzahl der Gruppen
SELECT COUNT(DISTINCT DEPARTMENT_ID),
       COUNT(DISTINCT SALARY)
FROM EMPLOYEES;
  

```

**5.2.6 Fallbeispiel - Gruppierung**

SQL Abfrage mit einer group by Klausel:

- > **Query: Group by Klausel**

```

-- Anzahl der Abteilungen pro Abteilung
SELECT COUNT(DISTINCT DEPARTMENT_ID)
FROM EMPLOYEES;

-- SELECT DEPARTMENT_ID, COUNT(DEPARTMENT_ID)
-- FROM EMPLOYEES
-- GROUP BY DEPARTMENT_ID;

-- Anzahl der Abteilungen pro Land
SELECT COUNT(DISTINCT DEPARTMENT_ID),
       COUNT(DISTINCT COUNTRY_ID)
FROM EMPLOYEES;

-- Anzahl der Abteilungen pro Branche
SELECT COUNT(DISTINCT DEPARTMENT_ID),
       COUNT(DISTINCT JOB_ID)
FROM EMPLOYEES;

-- Anzahl der Abteilungen pro Beruf
SELECT COUNT(DISTINCT DEPARTMENT_ID),
       COUNT(DISTINCT JOB_ID),
       COUNT(DISTINCT FIRST_NAME)
FROM EMPLOYEES;
  

```

**5.2.7 Aggregate Phase**

**Aggregate Phase:**

Von der Aggregate Phase werden die Datenbank mit den Ergebnissen aus der Selektion verfeinert.

**Die Aggregation der Abfrage bedeutet hier dass die aggregierten Datenbanken benutzt werden.**

Nach der Map Phase der Gruppierungsklausel kann die Selektion der Abfrage erweitert und verfeinert werden.

DEPARTMENT_ID	COUNT(DISTINCT DEPARTMENT_ID)	COUNT(DISTINCT COUNTRY_ID)	COUNT(DISTINCT JOB_ID)
10	10	3	10

**Erstellung Aggregate Phase:**

1. In der Aggregate Phase werden die Datenbanken der Selektion nach der **Map** Tabelle mit diesen resultierenden Werten verfeinert.
2. Die Selektion der Abfrage erweitert dabei eine weitere Kriterium, um die Ergebnisse der Aggregationierung der Daten zu präzisieren. Verfeinert.
3. Eliminiert alle Datenbanken die Werte von 10 für die Anzahl der Abteilungen pro Land haben und diese entsprechend präzisiert werden.
4. Zusätzlich kann eine Selektionstechnik das Ergebnis der Aggregation erweitern.
5. Klares erweitertes Bild der Abfrage, da nur die relevanten Datenbanken für die Kriterien der Klassifikationen relevant sind.

**Wichtig: Gruppe**

Mit der **having** Klausel können Sie die in der group by Klausel definierten Gruppen, Filterkriterien definiert werden.

→ Having = Where von GroupBy-Klausel

• **Ausgabe Null-Aggregat:**

- Mit der Abfrage kann die Datenstruktur eines Allokaps nach bestimmten Kriterien präzisiert werden.
- Zentrale erklärung die Auswertung, die wir Klienten mit Hilfe von Gruppenfunktionen auf einer Tabelle für Klienten. Es steht damit die Möglichkeit die einzelnen Gruppen einer Gruppe per Klienten zu filtern.
- Mit der Abfrage Klienten stellt die SQL Sprache eine Möglichkeit dar, verschiedene Gruppen innerhalb einer Gruppe per Klienten zu filtern.
- Wie haben damit eine Möglichkeit an der Hand eine Subgruppe oder Untergruppen abzuheben.

— **String-Klienten:**

---

- 1. SELECT KlientID, **MIN(GROUPNAME)** AS KEL
- 2. FROM Klienten GROUP BY KlientID;
- 3. SELECT KlientID, GROUPNAME, KEL
- 4. FROM Klienten GROUP BY KlientID;
- 5. **NATURAL JOIN** (Klienten);

13

### 5.2.9 Null-Aggregat:

**If** Null-Aggregat =  
Alles Null-Aggregat werden virtuelle Gruppen  
bezeichnet, die nur durchnehmen möglich  
die keine tatsächlichen Ergebnisse abgeben  
können.

• **Erläuterung Null-Aggregat:**

- Alle Unternehmen ohne IGL Anfrage, die für einen Gruppenausdruck benötigt werden, werden in einer einzigen virtuellen Gruppe zusammengefasst. Also Null-Aggregat.
- Nur Klienten, die keine IGL (Lieferanten) bestimmen, die das Null-Aggregat auswählen vorwiegend oder als letzte virtuelle Gruppe im Ergebnis-Aggregat.

14



**6. SQL - Komplexe Abfragen**

# 05

Komplexe Abfragen

Erläuterungen	Übungsaufgaben
01. Unterabfragen	62
02. With Klausur	65
03. Prozesse Vergleich	77
04. Mengenoperatoren	66
05. Quantoren	20

Topic	Number of Exercises
01. Unterabfragen	62
02. With Klausur	65
03. Prozesse Vergleich	77
04. Mengenoperatoren	66
05. Quantoren	20

**6.1. Unterabfragen**

Unterabfragen sind **select-Anweisungen**, die in SQL-Abfragen eingesetzt werden.

**Analyste Unterabfragen**

- Um diese Fragestellungen zu bearbeiten, müssen wir erstmal wissen wie hoch das Interesse an Unterabfragen ist, um anschließend die entsprechenden Anweisungen zu erläutern.
- Das Problem: Zur Zeit gibt es derzeit keine entsprechende Umfrage, die eine Aussage darüber werfen könnte.
- Wir brauchen darum einen neuen Ansatz: Unterfrage.

**6.2. Unterabfragen**

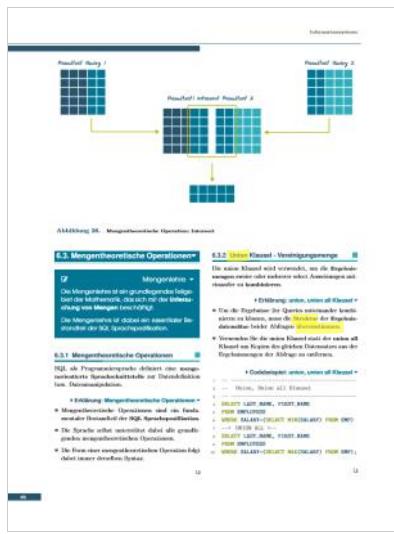
Unterabfragen sind **select-Anweisungen**, die in SQL-Abfragen eingesetzt werden. Die SQL-Syntax ist sehr ähnlich dabei Unterabfragen in der select, from, where und having Klammer.

Um Antworten einer SQL-Alfrage, berechnet die Unterabfrage zweitens das Ergebnis der Unterabfrage und erstens die Ergebnisse der Alfrage dann in Form einer virtuellen Tabelle vor liegen.











Funktion	Beschreibung	Zeile
SYS_CONNECT_BY_PATH	Die SYS_CONNECT_BY_PATH Funktion generiert aus der Tabe oder Vorstufe die hierarchische Abhängigkeiten.	107
WHEN	Mit der WHEN Funktion kann über Schleifen leicht mehrfach mit einem einzigen besseren Statement dargestellt werden.	108



<b>7. SQL - Data Definition Language</b>	Informationstechnik
<h1>06</h1>	
DQL - Data Definition Language	
<b>EI. Datenkomplexe</b>	72
<b>GI. Tabelle</b>	73
<b>GI. Constraint</b>	76
<b>GI. View</b>	76
<b>GI. Index</b>	79
<b>GI. Sequenz</b>	82
<b>7.3. Datenbankartefakte</b>	
Data Definition Language	
Mit DDL-Befehle kann die Struktur eines relationalen Datensatzes definiert bzw. geänderter werden.	
<b>7.3.1. Datenbankstrukturen</b>	
Datenbankstrukturen beschreiben die Struktur eines Datenbank.	
  	
+ Aufbau einer Datenbankstruktur =	
+ Inhaben wie ein logischer Name kann für die Tabellen eines Grunddatenmodells oder für die Zeichenketten der Grunddaten benutzt werden. Eine Datenbankstruktur kann auch mehrere Tabellen enthalten.	
+ Tabellen sind Datenstrukturen in logischer Form. Die Struktur einer Tabelle ist durch den Namen der Tabelle sowie durch Grunddaten. Aus technischer Sicht werden in Tabellen physische Datenstrukturen gespeichert.	
+ Constraints sind Einschränkungen, die auf die Konstruktion von Daten in einer Datenbank oder deren Verarbeitung angewendet werden. Sie garantieren die Gültigkeit von Daten und verhindern, dass ungültige Daten in die Datenbank eingespielt werden.	
+ View. Eine View erstelltg in Datenbankstrukturen eine virtuelle Tabelle, die die Ergebnisse von SELECT-Anfragen darstellt. Diese Ergebnisse können in die obigen Objekte GLEICHZUGRIFF gewünscht werden.	
+ Index. Mit Hilfe eines Indexes können Sequenzen aufschlüsselbarer Werte geordnet werden. Sequenzen werden in die Regel vor Gesamtsequenzen eingespielt.	
+ Sequenz. Diese sind Datenstrukturen, durch Anwenden durch konstante Erweiterung ergänzt werden.	
+ Indexe. Ihre Datenstrukturen ist ein Datenstruktur zur Beschleunigung der Suche und Sortierung von Datenreihen.	



Fazit	
<b>F.3. Datenbankkonsistenz: Constraints</b>	Datenkonsistenz ist eine Datenqualität, die durch Konsistenzerfordernisse, d.h. in einer Datenbank implementierte Regeln, gewährleistet wird.
<b>F.3.1 Datenkonsistenz und Integrität</b>	Der Datensatz einer Tabelle ist konsistent, wenn die folgenden Integritätsbedingungen erfüllt werden können:
<b>Integrität: Uniformität</b>	<b>Bereichsintegrität</b> = Die möglichen Werte, die Attribute eines Datensatzes müssen in einem definierten Bereich enthalten sind. Durch die Definition von Bereichsrestriktionen für die Attribute der Datenbanktabellen Sicherheitsgewährleistung erreicht werden.
<b>Integrität: Konstanz</b>	<b>Logische Konsistenz</b> = Die logische Konsistenz der Daten wird durch die Vergabe der Geschäftsregeln erreicht. Durch die Definition von obige <b>strukturelle Constraints</b> kann logische Konsistenz in einer Datenbank aufrechterhalten werden.
<b>Integrität: Referentielle Konsistenz</b>	Um Datenintegrität zu erhalten, müssen mit den anderen Tabellen Verweise aufeinander bestehen. Zur Sicherstellung referentieller Konsistenz werden die Funktionen von Hinreichungen und Ausgenommen.
<b>F.4. Entitätskonsistenz</b>	Seiter Toleranz: Den Datenbestand muss während Identifikatorwerten, trennbare und nicht trennbare Attribute der Zeichenketten ausgenommen.
<b>Fazit</b>	
<b>G.1. Grundlagen</b>	<b>7.2.3 Constraints</b> Gesetzliche Anforderungen, die keine Strafe für das Auftreten einer Verletzung erfordert wie z.B. Bank: Die Monatsumgebung von Unternehmen kann die Kosten des Unternehmens übersteigen.
<b>G.2. Legalität</b>	<b>7.2.4 Legalität: Generierung</b> Constraint werden die Tabelle implementieren, die sie als die tatsächliche Formular, also tabellarische Struktur definieren.
<b>G.3. Gültigkeit</b>	<b>7.2.5 Gültigkeit: Validierung</b> Constraints definieren logische Restriktionen. Wert oder Tablet (Tabelle) gebunden, werden überprüft, ob die Tabelle als ausreichende Gültigkeit aufweist.
<b>G.4. Gültigkeitsprinzipien</b>	<b>7.2.6 Gültigkeitsprinzipien</b> - <b>VALIDATE TABLE</b> : Prüft die Gültigkeit der gesamten Tabelle. - <b>VALIDATE TABLE</b> : Prüft die Gültigkeit einzelner Zeilen, zusammen mit den Gültigkeitsprinzipien. - <b>VALIDATE ROW</b> : Prüft die Gültigkeit einzelner Zeile, zusammen mit den Gültigkeitsprinzipien.
<b>G.5. Gültigkeitsprinzipien</b>	<b>7.2.7 Gültigkeitsprinzipien</b> - <b>CONSTRAINT</b> : Prüft die Gültigkeit der gesamten Tabelle, zusammen mit den Gültigkeitsprinzipien. - <b>CONSTRAINT</b> : Prüft die Gültigkeit einzelner Zeilen, zusammen mit den Gültigkeitsprinzipien. - <b>CONSTRAINT</b> : Prüft die Gültigkeit einzelner Zeile, zusammen mit den Gültigkeitsprinzipien.
<b>G.6. Gültigkeitsprinzipien</b>	<b>7.2.8 Gültigkeitsprinzipien</b> - <b>ALTER TABLE</b> : Ändert die Gültigkeitsprinzipien.
<b>G.7. Gültigkeitsprinzipien</b>	<b>ALTER TABLE table_name</b> <b>ADD CONSTRAINT constraint_name</b> <i>expression</i> <b>ALTER TABLE table_name</b> <b>DROP CONSTRAINT constraint_name</b> <b>ALTER TABLE table_name</b> <b>ENABLE CONSTRAINT constraint_name</b> <b>ALTER TABLE table_name</b> <b>DISABLE CONSTRAINT constraint_name</b> <b>ALTER TABLE table_name</b> <b>SET CONSTRAINT constraint_name</b> <i>value</i>

CREATE TABLE .  
ID NOT NULL PRIMARY KEY,  
DEP-ID NOT NULL FOREIGN KEY REFERENCE



```

    SELECT COUNT(*) FROM EMPLOYEE;
    -- 100

    SELECT COUNT(*) FROM DEPARTMENT;
    -- 10

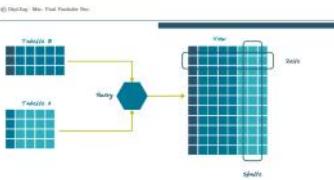
    SELECT COUNT(*) FROM EMPLOYEE_DEPARTMENT;
    -- 100

    SELECT COUNT(*) FROM PROJECT;
    -- 10

    SELECT COUNT(*) FROM EMPLOYEE_PROJECT;
    -- 100

    SELECT COUNT(*) FROM EMPLOYEE_LEAVE;
    -- 100

```



**7.3 Variable Bedingungen**

Für die obige Klausur verhindern Parameter:

- DDL: variable Bedingungen
- SELECT ... WHERE ...
- CREATE OR REPLACE VIEW EMPLOYEE\_DEPARTMENT
- INSERT ... SELECT ...
- SELECT ... FROM ... WHERE ...
- WHERE ... DEPARTMENT\_ID = ?
- ... WHERE ...
- SELECT ... FROM ... WHERE ...
- SELECT ... FROM ... WHERE ...
- WHERE ... = ?

**7.4 DDL: Beispieldrop view**

• Syntax: drop view

- (DDL) Klasse Löschen
- DROP VIEW EMPLOYEE\_DEPT;

Abbildung 32: Indexstrukturen



**7.5 Indextyp: B\*-Index**

Der B\*-Index ist der praktisch wichtigste relationalen Index.

**Erläuterung B\*-Index:**

- Der B\*-Index ist ein Blockindex, der in den Datenstrukturen die indexierende Strukturierung, Vergleichbarkeit und Sortierbarkeit der Werte aufrechterhält. Ringe bzw. Löcher von Werten in einem B\*-Index sind nicht möglich.
- Ein B\*-Index wird daher in einer Liste für Spalten zur Verarbeitung zusammengefasst.
- Die Datenspalten bzw. Berechnungswerte müssen für alle Spalten des B\*-Index gleich sein.
- Der B\*-Index unterstützt dabei vorrangig eine Suchoperation im Datenmaterial.

**Syntax: create index:**

```

    -- B*-Index
    -- SELECT ... WHERE ...
    -- WHERE FIRST_NAME, LAST_NAME
    -- IN EMPLOYEE_KLIENTEN ...
    -- ...
    -- optimisierte Anfrage
    -- SELECT FIRST_NAME, LAST_NAME
    -- ...
    -- ORDER BY LAST_NAME;

```

Abbildung 33: Indizes

**7.6 Indextyp: Unique Index**

Ein Unique-Index ist eine weitere Einschränkung, wird durch die entsprechende constraint definiert und kann nicht eingeschränkt werden.

**Erläuterung unique Index:**

- Ein unique Index kann nach über mehrere Spalten bestimmt werden. In diesem Fall müssen die benutzten Spalten zusammen eindeutig machen.
- Bei der Regel wird einer Spalte Typ-Anker für die technologische Anwendung.

**Syntax: create index:**

```

    -- ...
    -- unique Index
    -- ...
    -- UNIQUE INDEX UNIQ
    -- ...
    -- ON department(department_name);
    -- ...
    -- UNIQUE INDEX UNIQ2
    -- ...
    -- PRIMARY KEY(PRIMARY_KEY)
    -- ...

```



**7.6. Sequenz**

**Sequenz =**  
Eine Sequenz ist ein Datenbestandteil, mit dem aufschließende Strukturen Werte generieren.

**7.6.1 Anlegen von Sequenzen**

Sequenzen werden zur Generierung von Projekt- oder Schema-wertenden Werten benutzt.

**System-create sequence**

- System.create sequence
- CREATE SEQUENCE Anweisung >=
  - CREATE SEQUENCE Sequenzname
  - ASSTRAIT ATTN sequence
  - MAXVALUE integer | NOCACHE
  - MINVALUE integer | NOCACHE
  - INCREMENT BY integer
  - START WITH integer
  - CACHE integer
  - ORDER

**7.6.2 Sequenzen verwenden**

Für das Arbeiten mit Sequenzen verwenden wir die Attribute current und nextval.

**• DDL-Sequenzen verwenden**

- Sequence verwenden
- CREATE SEQUENZ project\_seq IDENTITY(1,1)
- SELECT NEXTVAL
- NOCACHE
- CACHE 100
- ORDER

**INFORMATIONEN**

- berichtet die ganze Zahl, die eine Sequenz generiert wird
- START WITH
- berichtet die ganze Zahl, mit der die Sequenz startet
- INCREMENT
- berichtet die gesamte Spanne abfragbar
- NOCACHE
- Sequenz hat keine weitere Säume
- MAXVALUE
- Sequenz hat keine obere Grenze
- MINVALUE
- mindestens eines der SEQUENZ hat eine untere Grenze
- SEQUENCE
- SEQUENZ hat keine weitere Säume
- STARTWITH
- berichtet, ob nach Erreichen der MAXVALUE

© DataIng - Mac - Paul Pfeiffer Inc.

**8. SQL - Data Manipulation Language**

# 07

**DML - Data Manipulation Language**

**8.1. DML - Befehlsatz**

**• DML Befehlsatz**

Die Data Manipulation Language definiert die Art der Veränderung von Daten.

**8.1.1 DML Befehle**

Der DML Befehlsatz wird zur Verarbeitung von Daten in der Datenbank verwendet.

**• Auflistung: DML Befehle**

**• Insert command**

Die Insert Anweisung ist ein Befehl zum Einfügen von Daten in Tabellenstrukturen.  
Der Insert Befehl besteht mehrere Formen:

- update command
- Die update Anweisung ist ein Befehl zum Ändern von Daten.
- Der update Befehl besteht mehrere Formen:

**• merge command**

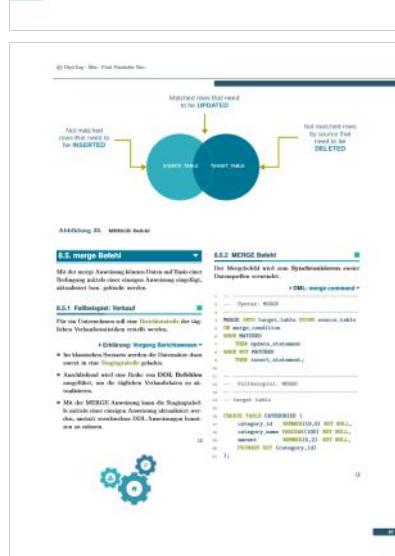
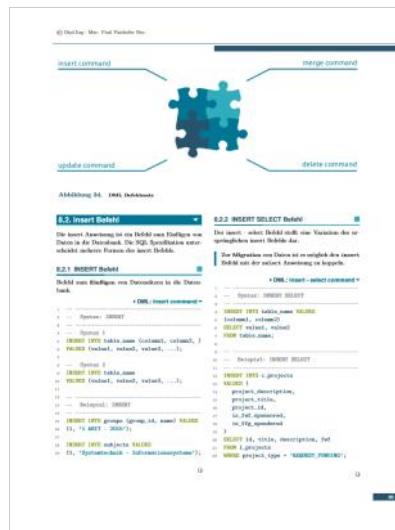
Die merge Anweisung ist ein Befehl zum Verbinden von Daten. Die Anweisung ist in der Datenbanktechnik neuer Art und nutzt beide Befehle.

**• delete command**

Die delete Anweisung ist ein Befehl zum Löschen von Daten.









```

5.5.3 Full Beispiel - MERGE

I XML merge command =

<!-- Full Beispiel -->
<-- source table (original table) -->
<-- CREATE TABLE CATERING_SPACER -->
<-- SELECT * FROM CATERING_SPACER -->
<-- category_name NAMECATERING_WT_WSL -->
<-- category_id NAMECATERING_WT_WSL -->
<-- category_ref NAMECATERING_WT_WSL -->
<-- > -->
<-- INSERT INTO CATERING_SPACER -->
<-- SELECT * FROM CATERING_SPACER -->
<-- CL_Catering_Bicycle_10000 -->
<-- CL_Catering_Bicycle_100000 -->
<-- CL_Catering_Bicycle_1000000 -->
<-- CL_Catering_Bicycle_10000000 -->
<-- CL_Catering_Bicycle_100000000 -->
<-- CL_Catering_Bicycle_1000000000 -->
<-- > -->
<-- INSERT INTO CATERING_SPACER -->
<-- SELECT * FROM CATERING_SPACER -->
<-- CL_Catering_Bicycle_10000 -->
<-- CL_Catering_Bicycle_100000 -->
<-- CL_Catering_Bicycle_1000000 -->
<-- CL_Catering_Bicycle_10000000 -->
<-- CL_Catering_Bicycle_100000000 -->
<-- CL_Catering_Bicycle_1000000000 -->
<-- > -->
<-- MERGE CATERING_SPACER -->
<-- MERGE CATERING_SPACER -->
<-- ON category_id = category_id -->
<-- WHEN MATCHED THEN UPDATE SET -->
<-- category_name = category_name -->
<-- category_id = category_id -->
<-- category_ref = category_ref -->
<-- > -->
<-- MERGE category_id, category_name, original -->
<-- VALUES -->
<-- 1 category_id, -->
<-- x.category_id, -->
<-- x.category_name, -->
<-- x.original -->
<-- > -->
<-- 1 10000 Children Bicycles -->
<-- 2 100000 Children Bicycles -->
<-- 3 1000000 Children Bicycles -->
<-- 4 10000000 Children Bicycles -->
<-- 5 100000000 Children Bicycles -->
<-- 6 1000000000 Children Bicycles -->

```



relationale Datenbanken folgen  
den ACID-Eigenschaften  
(Säure)

Atomicity - unkilbar (alles oder nichts)

- Consistency - konsistent (Normalformen)
- Isolation - Daten während Transaktion gesperrt  
Transaktion darf auf gegenseitig nicht beeinflussen

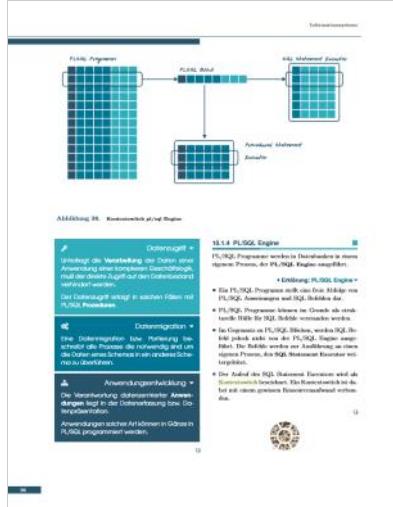
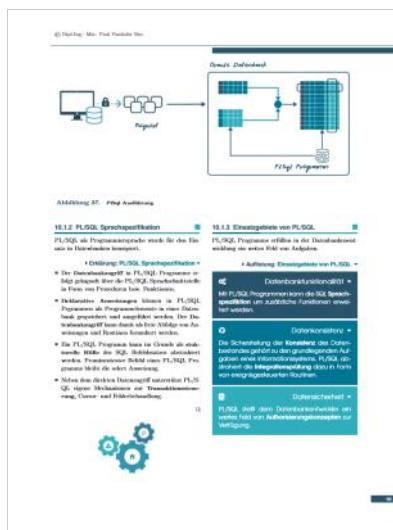
D - Durability - Dauerhaftigkeit (Daten bleiben so)

Transaktion beginnt bei Ausführung d. Select-Befehls

Informatikkomplexen • Theoriekriptum



10. PL/SQL - Grundlagen		Informationstechnik
<b>01</b>	PL/SQL Grundlagen	
<b>E1</b> Grundlagen	94	
<b>E2</b> Block	96	
		Erstellung PL/SQL Grundlagen
		• Durch die Erstellung des SQL-Befehlsbaus der PL/SQL-Aufrufe kann ein PL/SQL-Programm direkt Zugriff auf die Daten und Strukturierung der Daten haben.
		• Durch es wird in einigen Teile der Anwendungsschicht wie Applikationsserver in den Datenbankbereich zu übertragen.
		• Die Verarbeitung der Anwendungsschicht ist im Datenbankbereich, meistens führt es zu einer signifikanten Verringerung der Anwendungsausführungszeit durch mehrere Ausführungen.
		• Der Datenbankzugriff erfolgt in PL/SQL-Programmen durch die Befehle der PL/SQL-Sprachbedienung in Form von Macros.
		• Liefert zusammengeführte Ressourcen wie Datei, Benutzer und Prozeduren.
		• PL/SQL-Programme können direkt in einer Datenbank geprüft und angefordert werden.
		• Zur Abarbeitung von PL/SQL-Programmen werden spezielle Befehle benötigt, welche werden als PL/SQL-Blocks bezeichnet.
		• Grundlegende Kenntnisse über die Oracle-Datenbank sind erforderlich, um die PL/SQL-Befehle korrekt einzusetzen.
		• Nach dem detaillierten Überblick automatisiert PL/SQL-eigene Merkmale zur Transaktionsverwaltung, Cursor und Fehlerbehandlung.





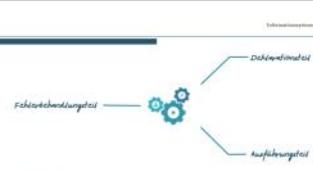


Abbildung 28. *Saffron stema pingi* Blüte

- 10.2 PLSQL Block
  - 10.2.1 PLSQL Block
- if
  - PLSQL Block mit den grundlegenden Regeln für PLSQL Programm
  - 1. PLSQL Schreibt die Struktur von PLSQL Programm.
  - 2. PLSQL können wir Klammer einer Do...Loop oder acht Zeichen lang schreiben.
- Berechnet gleicht sich das PLSQL Block in den Text.
  - Erklärung: Jeder Schritt eines Blocks ist deklarativ. Das Deklarieren eines Blocks ist für die Programmierung wesentlich Voraussetzung. Es kann eine logische Struktur oder funktionale. Das Deklarieren und was es ist, ist der Anfang eines PLSQL Blockes angegeben. Der Deklarieren ist ein spezielles Befehl des Oracle Datenbank.
- Erklären Sie, was ein Ausdruck ist?
  - Ein Ausdruck ist eine Kombination aus deklarative Funktionaleinheiten PLSQL, wie z.B. Anweisungen und Befehle, die zusammengefasst werden, um die Schritte logisch angeordnet.
- Der Ausführungsablauf ist abgestuft.
- Für PLSQL Block sind folgende Regeln definiert:
  - PLSQL Deklarationsblöcke müssen im PLSQL Block vor dem ersten Befehl oder während des PLSQL Blockes definiert werden.
  - PLSQL Deklarationsblöcke müssen im PLSQL Block vor dem ersten Befehl oder während des PLSQL Blockes definiert werden.
  - PLSQL Deklarationsblöcke müssen im PLSQL Block vor dem ersten Befehl oder während des PLSQL Blockes definiert werden.
- Der Fehlerbericht ist optional.

© The Jim M. Fink Faculty Site

Kriterien	Beschreibung	Werturteile
Doktoratstest	Der Testberichterstatter vertritt die Werturteile der für das Programm akkreditierenden Vertreter, Konferenz, Causa bzw. Fakultätsräte.	optimal
Aufklärungsteil	Der Aufklärungsteil erläutert die eigentlichen Kenntnisse und Fertigkeiten des Prüflings im Bereich der Praxis und Theorie.	optimal
Fehlerbehandlungsteil	Im Fehlerbehandlungsteil wird die Heranreifung eines PUGL-Prüflings geprägt.	optimal

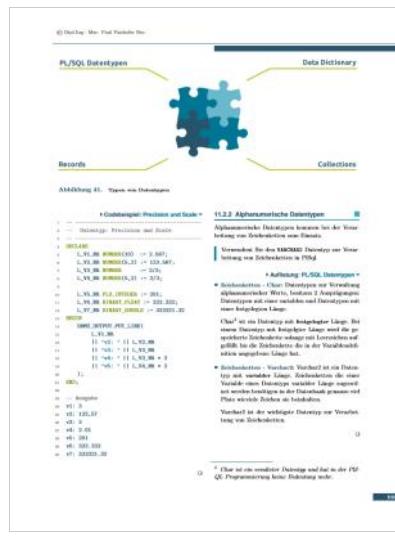
Abbildung 48. *Postkarte*

<b>12.2 Anonym Block</b> Die PL/SQL Programm ist grundsätzlich immer unter einer Anonym Block geschrieben.	<b>12.2.3 Benennte Blöcke</b> Die PL/SQL Sprache unterstützt indirekte Typen wie Benennete Blöcke.
<b>12.3 Anonym Block -</b> Die PL/SQL Anonyme Block ist eine Anonym Block mit dem Attribut: Rückgabe Wert oder Block Wert.	<b>Package Block -</b> Ein Package ist logisch zusammengehörige Objekte in einer modularen Format zusammengefasst.
<b>12.4 Benennte Block</b> Inhalt der Benennte Deklarationen und lokale Variable und lokale Fächer, welche nicht innerhalb einer Deklaration oder einer Block Deklaration versteckt werden.	<b>Procedure Block -</b> Eine Procedure ist eine Art Anfang von Anweisungen und Routinen. Prozeduren können in der PL/SQL Sprache wie für Funktionen und Verfahren von PLS/SQL Programm verwendet.
<b>12.5 Anonym Block im Vergleich zu Benennte Blocken</b> Jedes Block wird in Datentypen produziert werden.	<b>Function Block -</b> Eine Function ist eine Art Anfang von Anweisungen und Routinen. Prozeduren können in der PL/SQL Sprache wie für Funktionen und Verfahren von PLS/SQL Programm verwendet.
<b>12.6 Deklaration Anonym Block -</b> Anonym Block <b>12.6.1 Deklaration Anonym Block -</b> Deklaration Anonym Block - <b>Wiederholung</b> , <b>L_FIRST_ABNM</b> <b>VARNAME</b> := <b>Teilwert</b> ;	<b>12.6 Deklaration Anonym Block -</b> Deklaration Anonym Block - <b>Wiederholung</b> , <b>L_FIRST_ABNM</b> <b>VARNAME</b> := <b>Teilwert</b> ;
<b>12.6.2 DEKLARATION</b> Deklaration Anonym Block - DEKLARATION <b>VARNAME</b> <b>DATA_TYPE</b> ; DEKLARATION <b>VARNAME</b> <b>DATA_TYPE</b> <b>INITIALISATION</b> ; DEKLARATION <b>VARNAME</b> <b>DATA_TYPE</b> <b>INITIALISATION</b> <b>WEITERE</b> ;	<b>12.6.3 Initialisierung Anonym Block -</b> Initialisierung Anonym Block - <b>Wiederholung</b> , <b>L_FIRST_INIT</b> <b>VARNAME</b> := <b>Teilwert</b> ;















12.3. Kontrollstrukturen	
Kontrollstrukturen und Rechenweise vor Beurteilung des Programmablaufs	
	12.3.1. IF THEN Block
Kontrollstrukturen bestimmen die Reihenfolge, in der die Anweisungen eines Programms ausgeführt werden.	
Teilt die Bedingung einer If-Anweisung auf, werden die Befehle in ein If-Block abgespeichert.	<p style="text-align: right;">Syntax: IF THEN =</p> <pre> 1 -- Syntax: IF THEN 2 3 IF condition THEN 4 Statement1; 5 Statement2; 6 Statement3; 7 ELSEIF condition THEN 8 Statement4; 9 Statement5; 10 ELSE 11 Statement6; 12 END IF; 13 14 -- Abgewartet, ob Befehl 13 ausgetragen wird 15 L_BREAKPOINT; //-----&gt; 16 L_BREAKPOINT; //-----&gt; 17 18 SELECT COUNT(*),NAME FROM TAB1; 19 WHERE EMPLOYEE_ID = 100; 20 ORDER BY NAME; 21 22 L_BREAKPOINT; //-----&gt; 23 TIME_WAIT(TIME_WAIT); 24 TIME_WAIT(TIME_WAIT); 25 /* Testausgabe der 10. Zeile */ 26 27 EXEC 17; 28 29 30 HOME_WAIT(HOME_WAIT); 31 /* Testausgabe der 10. Zeile */ 32 33 34 ENDIF; </pre>
12.3.2 Kontrollstruktur - Case Block	
Ablaufweise der Auswertung siehe in Pg. 325. Pro grammatische case-Anweisung nur Pg. 326.	
	Syntax: CASE BLOCK =
	<pre> 1 -- Syntax: CASE BLOCK 2 3 CASE 4 WHEN condition THEN 5 Statement1; 6 Statement2; 7 Statement3; 8 ELSEWHEN condition THEN 9 Statement4; 10 Statement5; 11 ... 12 ELSE 13 Statement6; 14 END CASE; 15 16 -- Abgewartet, ob Befehl 16 ausgetragen wird 17 18 INCLUDE 19 L_BREAKPOINT(5,1,VERSION_NUMBER(1)); 20 21 22 -- read database version 23 L_READDBVERSION; 24 L1 VERSION; 25 L2 VERSIONNUMBER(); 26 27 /* Testausgabe der 10. Zeile */ 28 29 CASEBLOCK; 30 31 WHEN 1 THEN 32 HOME_WAIT(HOME_WAIT); 33 /* Testausgabe der 10. Zeile */ 34 35 WHEN 12 THEN 36 HOME_WAIT(HOME_WAIT); 37 /* Testausgabe der 10. Zeile */ 38 39 ELSE 40 HOME_WAIT(HOME_WAIT); 41 /* Testausgabe der 10. Zeile */ 42 43 ENDIF; </pre>







Exkurskopf	Beschreibung	Seite
Standardisierung	Standardisierung ist die durch die PLSQ-Spezifikation definierten internen PLSQ.	114
Anwendungsfelder	Anwendungsfelder sind Anwendungen spezifischer Fachwörter.	118
Unterschiede Fehler	Die unterschiedlichen Fehler unterscheiden sich in der Fehlerart und Fehlerhäufigkeit.	118
PLS-QB	PLS-QB ist die Abkürzung für PLS-Quantitative Beziehung.	118


---

Ablösung 44 - Uppen von PLS-Quantitative Beziehung

<b>12.4.3 Arten von Exkursen</b>	<b>12.4.4 Anwendungsfelder</b>
<p>Die PLSQ-Spezifikation unterteilt mehrere Arten von Fehlerarten:</p> <ul style="list-style-type: none"> <li>• <b>Basisfehlerarten:</b> Basisfehlerarten sind Fehler, die direkt mit der PL-SQZ-Spezifikation selbst verknüpft werden. Sie treten ebenfalls als Grundfehler in einer Standardisierung oder einer PLS-QB auf.</li> <li>• Standardfehlerarten: Standardfehlerarten sind Fehler, die nicht direkt mit der PL-SQZ-Spezifikation selbst verknüpft werden. Sie treten ebenfalls als Grundfehler in einer Standardisierung oder einer PLS-QB auf.</li> <li>• Anwendungsfeldfehler: Anwendungsfeldfehler sind Fehler, die explizit für Datenanalysenmethoden definiert werden. Sie treten ebenfalls als Fehler in einer Standardisierung oder einer PLS-QB auf.</li> </ul> <p>Anwendungsfeldfehler sind durch die Anwendung spezifische Bezeichnung und Fehlerarten, die:</p> <ul style="list-style-type: none"> <li>• Unterteilen Fehler: Unterteilungsfelder bestimmen in PLSQ welche Formen Unterteilungen innerhalb der PLSQ-Fehlerarten vorkommen. Unterteilungen sind in PLSQ-Formaten über die programelemente mit einer Fehlerart angegeben.</li> </ul>	<p>Anwendungsfelder sind die Grundfehlergruppe aus Anwendung spezifischen Fehlerarten:</p> <ul style="list-style-type: none"> <li>• <b>Einführung Anwendungsfelder:</b> <ul style="list-style-type: none"> <li>◦ Diese Art der Fehlerarten ist eine Kombination aus den Fehlerarten, welche leichter und vor vielen dem Market werden.</li> </ul> </li> <li>• <b>Übersicht:</b> Es ist die Übersicht über alle Fehlerarten der Anwendungsfelder.</li> <li>• <b>Erstellung Anwendungsfelder:</b> <ul style="list-style-type: none"> <li>◦ Einzelne Fehlerarten werden hier erstellt und für weitere dem Market werden.</li> </ul> </li> <li>• <b>Erstellen:</b> Es ist die Erstellung eines PLS-QB.</li> <li>• <b>Standardisierung:</b> Standardisierung ist ein Prozess, der PLS-QB-Programme über die reale PLS-QB angibt.</li> </ul> <p>• <b>Codegenerierung:</b> Anwendungsfelder →</p> <ul style="list-style-type: none"> <li>◦ <b>DEFAUTL:</b> Anwendungsfelder → DEFAUTL → DEFAUTL_ABERT, DEFAUTL_ABERT_CONTINUE</li> <li>◦ <b>INITIALISATION:</b> Anwendungsfelder → programelemente, INIT, PROG_ABERT, PROG_ABERT_CONTINUE, PROG_ABERT_STOP</li> <li>◦ <b>RESET:</b> Anwendungsfelder → RESET → ALLE_DEFAUTL_ABERT, ALLE_DEFAUTL_ABERT_CONTINUE</li> <li>◦ <b>SYSTEM:</b> Anwendungsfelder → SYSTEM → DEFAUTL_ABERT, DEFAUTL_ABERT_CONTINUE</li> </ul>

Exception	Description	Nil Code
ACCESS_VIOLATION	Program attempted to assign values to the variables of an uninhabited area.	RA_0000
CASE_NOT_FOUND	None of the choices in the <code>SELECT</code> clause of a case statement were selected and there is no ELSE clause.	RA_0000
COLLECTION_IS_MALFORMED	Program attempted to apply collection methods from other AS220012 or an incompatible collection type.	RA_0000
COMPILE_ERROR	Program attempted to open a static procedure.	RA_0000
DUP_VAR_CHNDR	Program attempted to insert duplicate values in a column that is constrained to unique values.	RA_0001
INVALID_ACCESS	Attempted to access invalid memory.	RA_0002
INVALID_CHARACTER	Conversion of character string failed.	RA_0003
INVALID_HAMMER	None of the choices in the <code>SELECT</code> clause of a hammer statement were selected and there is no ELSE clause.	RA_0004
NO_DATA_FOUND	Single row SELECT returned no rows or a zero program referenced in a declared variable was not found or no uninhabited reference to an object specified directly by full name.	RA_0005
PROGRAM_ERROR	Program had an internal problem.	RA_0006
RONWHITE_MESSAGE	The first error message and the last error message available on an assignment or selection statement.	RA_0007
SELF_IS_ML	A program attempted to call a method that itself is the instance of.	RA_0008
STORAGE_ERROR	Not enough of memory or memory are occupied.	RA_0009
SUBSCRIPT_BEYOND_BOUNDS	A program attempted to handle field or value using an index number greater than the size of the array.	RA_0010
SUBSCRIPT_OUTSIDE_BOUNDS	A program attempted to handle field or value using an element index that is outside the legal range (the example).	RA_0011
TYPE_INCOMPATIBLE	The conversion of a variable into a desired result failed because the types of the variables are incompatible.	RA_0012
TYPE_INVALID	None of the choices in the <code>SELECT</code> clause of a type assignment statement were selected and there is no ELSE clause.	RA_0013
TYPE_MISMATCH	Single row SELECT returned multiple rows.	RA_0014
TRANSACTION_BROKEN	The remote portion of a transaction has been rolled back.	RA_0015
VALUE_ERROR	An arithmetic, conversion, or scale constraint was violated.	RA_0016
ZERO_DIVIDE	A program attempted to divide a number by zero.	RA_0017



Entscheidung	Beschreibung	Wert/Getestet
FOIND	Das ist ein Tag speziell den Wert kann der vorhergesagte mit 2. Stellen ein Ergebnis zurückgegeben hat. Ansonsten wird Null gespielt.	True/Getestet
NORMAND	Das Gegenstück von FOIND	True/Getestet
WORN	Mit dem Attribut wird geprüft ob der Outer offen ist	True/Getestet
WONSCHE	Überprüft ob die Wünsche die in den durch die Kundenbeschreibung beschrieben sind erfüllt sind	False/Getestet
MAX_BONCOUNT	Gibt die Zahl der Zeilen die durch die Orderbonifikation bestreift werden	True/Getestet
MAX_DESPROM	Gibt an wie viel die Masseverminderung bestreift aufgetragen wird	True/Getestet

**Ablösung 44: Konsistenzcheck**

### 12.2.3 Datenverarbeitung

Ein Baues erstellt die PL/SQL Programme die Zugriff auf die Daten einer Datenbank.

- Erläuterung: Datenverarbeitung**
  - Mit der Informatik verbindet man das System der aus den Daten generierte SQL Abfrage oder die PL/SQL Prozeduren die die Daten in der Datenbank abfragen und verarbeiten.
  - Die Datenverarbeitung ist die Verarbeitung die jenen nicht ist die Daten der Konstrukteure was zusammen mit den Daten.
  - Nach seiner Intablirung referenziert es Namen der neuen Strukturen der Ergebnisse der untersuchten Daten.
- Der Begriff der Datei der Konstrukteur ergibt sich aus dem Sinn der Inhalt eines Leseschriften oder einer Tabelle die im Rahmen der Rechnung bearbeitet werden.**
- Für die Datengewinnung werden die PL/SQL Begriffe zur Hilfe der Datenverarbeitung. Diese Begriffe werden später in den PL/SQL Programm abgegriffen.

**12.2.4 Bulk Fetch Anweisung**

Es liegt eine Fülle kleinere in den Kursen auch eingesetzte Datenbanken gleicherweise.

**Codierungspunkt: Bulk collect +**

<b>CREATE TABLE</b>	<b>ALTER TABLE</b>
<b>GRANT</b>	<b>REVOKE</b>
<b>DELETE</b>	<b>TYPE</b>
<b>INSERT</b>	<b>TABLE_TYPE</b>
<b>INDEX</b>	<b>UNPLUGGED</b>
<b>SELECT</b>	<b>VALIDATE</b>
<b>UPDATE</b>	<b>VALIDATE_TYPE</b>
<b>VIEW</b>	<b>VALIDATE_TABLE</b>
<b>CREATE</b>	<b>VALIDATE_COLUMNS</b>
<b>DROP</b>	<b>VALIDATE_CLOB</b>
<b>TRUNCATE</b>	<b>VALIDATE_DDL</b>
<b>ALTER</b>	<b>VALIDATE_DDL_COLLECT</b>
<b>CREATE INDEX</b>	<b>VALIDATE_INDEX</b>
<b>CREATE LOB</b>	<b>VALIDATE_LIKE</b>
<b>CREATE MATERIALIZED VIEW</b>	<b>VALIDATE_MATERIALIZED_VIEW</b>
<b>CREATE SEQUENCE</b>	<b>VALIDATE_SEQUENCES</b>



Beschreibung	Bereich	Seite
Umlauf Prozess	Umlauf Prozess beschreibt eine lineare Abfolge von Anweisungen und Befehlen. Prozessoren werden zur Organisation von PLS/PLC Code in PLS/PLC Programmen verwendet.	100
Funktion	Funktionen sind Prozeduren, die einen Rückgabewert an den Aufrufer der Funktion zurückgeben.	102
Package	Ein Package hält zusammengehörige Objekte unter einer modularen Umgebung zusammen. Ein Package kann aus mehreren Funktionen bestehen.	103
Trigger	Trigger kann als benachrichtigende Prozeduren, die auf einem Zustand oder einem Ereignis reagieren.	105
Diagramm	Diagramme dienen der Darstellung von Prozessen, die auf einem Zustand oder einem Ereignis reagieren. Trigger werden in Diagrammen zur Währung des Dokumentationsprinzips.	106



Übung 10: Die Anwendung von Trigger- und Constraint-Logik	
<p><b>Übungsaufgaben:</b></p> <p><b>Übungsaufgabe 1:</b> Implementieren Sie die folgenden Anforderungen:</p> <ul style="list-style-type: none"> <li><b>Trigger:</b> Ein Mitarbeiter kann höchstens 100000 Euro Gehalt haben.</li> <li><b>Constraint:</b> Ein Mitarbeiter kann höchstens 100000 Euro Gehalt haben.</li> </ul>	<p><b>Lösung:</b></p> <pre> CREATE TRIGGER TG_Mitarbeiter_Gehalt     BEFORE UPDATE ON Mitarbeiter     FOR EACH ROW     BEGIN         IF NEW.Gehalt &gt; 100000 THEN             SIGNAL SQLSTATE '45000'                 SET MESSAGE_TEXT = 'Gehalt darf nicht höher als 100000 Euro sein';         END IF;     END;      -- Constraint     ALTER TABLE Mitarbeiter     ADD CONSTRAINT CK_Mitarbeiter_Gehalt     CHECK(Gehalt &lt;= 100000);   </pre>
<p><b>Übungsaufgabe 2:</b> Implementieren Sie die folgenden Anforderungen:</p> <ul style="list-style-type: none"> <li><b>Trigger:</b> Ein Mitarbeiter kann höchstens 100000 Euro Gehalt haben.</li> <li><b>Constraint:</b> Ein Mitarbeiter kann höchstens 100000 Euro Gehalt haben.</li> </ul>	<p><b>Lösung:</b></p> <pre> CREATE TRIGGER TG_Mitarbeiter_Gehalt     BEFORE UPDATE ON Mitarbeiter     FOR EACH ROW     BEGIN         IF NEW.Gehalt &gt; 100000 THEN             SIGNAL SQLSTATE '45000'                 SET MESSAGE_TEXT = 'Gehalt darf nicht höher als 100000 Euro sein';         END IF;     END;      -- Constraint     ALTER TABLE Mitarbeiter     ADD CONSTRAINT CK_Mitarbeiter_Gehalt     CHECK(Gehalt &lt;= 100000);   </pre>
<p><b>Übungsaufgabe 3:</b> Implementieren Sie die folgenden Anforderungen:</p> <ul style="list-style-type: none"> <li><b>Trigger:</b> Ein Mitarbeiter kann höchstens 100000 Euro Gehalt haben.</li> <li><b>Constraint:</b> Ein Mitarbeiter kann höchstens 100000 Euro Gehalt haben.</li> </ul>	<p><b>Lösung:</b></p> <pre> CREATE TRIGGER TG_Mitarbeiter_Gehalt     BEFORE UPDATE ON Mitarbeiter     FOR EACH ROW     BEGIN         IF NEW.Gehalt &gt; 100000 THEN             SIGNAL SQLSTATE '45000'                 SET MESSAGE_TEXT = 'Gehalt darf nicht höher als 100000 Euro sein';         END IF;     END;      -- Constraint     ALTER TABLE Mitarbeiter     ADD CONSTRAINT CK_Mitarbeiter_Gehalt     CHECK(Gehalt &lt;= 100000);   </pre>
<p><b>Übungsaufgabe 4:</b> Implementieren Sie die folgenden Anforderungen:</p> <ul style="list-style-type: none"> <li><b>Trigger:</b> Ein Mitarbeiter kann höchstens 100000 Euro Gehalt haben.</li> <li><b>Constraint:</b> Ein Mitarbeiter kann höchstens 100000 Euro Gehalt haben.</li> </ul>	<p><b>Lösung:</b></p> <pre> CREATE TRIGGER TG_Mitarbeiter_Gehalt     BEFORE UPDATE ON Mitarbeiter     FOR EACH ROW     BEGIN         IF NEW.Gehalt &gt; 100000 THEN             SIGNAL SQLSTATE '45000'                 SET MESSAGE_TEXT = 'Gehalt darf nicht höher als 100000 Euro sein';         END IF;     END;      -- Constraint     ALTER TABLE Mitarbeiter     ADD CONSTRAINT CK_Mitarbeiter_Gehalt     CHECK(Gehalt &lt;= 100000);   </pre>



13.3.4 Numerikkonventionen			
Datenbankobjekt	Numerikkonvention	Bspobjekt	Werte
Primary Key	varchar_10	PRODUCT_ID, PRODUCTNAME_ID, PURCHASE_ID	75
Unique Key	varchar_10	PRODUCT_TITLE_ID	75
Foreign Key	varchar_10	SUPPLYPROJECT_PROJECT_ID	75
Other Constraint	varchar_10	PRODUCTS_PROJECT_ID, PRODUCT_SUPPLIER_ID, ...	75
Imports	varchar_10	PRODUCTS_PROJECT_ID, PRODUCT_SUPPLIER_ID, ...	75
View	varchar_10	PRODUCT_PURCHASE_X, PRODUCT_SUPPLIER_X	75
Type	varchar_10	PRODUCT_NUMBER_Y, SUPPLYPROJECT_NUMBER_Y	75
PL/SQL Package	varchar_100	PRODUCT_INFORMATION_PKG, GENDER_PKG	100
PL/SQL Procedure	varchar_100	SUPPLYPROJECT_PKG	100
PL/SQL Function	varchar_100	SUPPLYPROJECT_PKG	100
Global Variable	g_variable_name	g_PROJECT_DATE	100
Local Variable	l_variable_name	l_PROJECT	100
Parameter	p_variable	p_PROJECT	100
Column	v_variable	v_PROJECT, v_PROJECTNAME	100
Variable	v_variable	v_PROJECT_ID, v_LASTNAME	100
Record	c_variable	c_PROJECT	100

Abbildung 13: Numerikkonventionen

Informationssysteme

## 14. PL/SQL - SQL Funktionen

05

SQL Funktionen (10)

SQL Funktionen

14.1. SQL Funktionen

SQL als Programmiersprache wurde als Sprachausdehnung für Informationssysteme konzipiert.

14.1.1 SQL Funktionen

Mit PL/SQL Funktionen kann die SQL-Sprachausdehnung um neue Funktionen erweitert werden.

- Erweiterung SQL Funktionen integriert:
  - SQL Funktionen können in einer Statementstruktur oder direkt im WHERE-Klausule benutzt werden.
  - SQL Funktionen sind **deterministisch**. Als Deterministisch wird eine Funktion bezeichnet, die für gleiche Argumente immer dasselbe Ergebnis liefert.
  - Bestimmte Funktionen können in einer Datenbank parallel ausgewertet werden.
  - Zum Aufführen einer SQL-Funktion wird der create function Befehl verwendet.
- Systeme können benutzt werden:
  - Systeme: DBF-Funktionen
  - Systeme: Oracle-Funktionen
  - CREATE OR REPLACE FUNCTION name AS
 [TYPE] [SQL Statement]
 RETURN datatype DETERMINISTIC
 PARALLEL\_ENABLE;
 END;
  - Systeme: Variablen, constant, etc. kann
    - declare statements;
    - IF THEN ELSE END IF;
    - LOOP
    - NEW OTHER THIS
    - ...
    - END Function;

Informationssysteme

SQL Funktion

14.1.2 Programmieren von SQL Funktionen

Bei der Programmierung von SQL-Funktionen müssen bestimmte Beziehungen beachtet werden.

2 Anleitung: Wurzelziehen

• globale Kenntnis über ein PL/SQL-Funktion ist einer select Abfrage erreicht werden kann neue Abfrage mit dem Namen der Funktion ausführen.

• Nutzen einer PL/SQL-Funktion ist nicht ratsam, da es sich dabei um einen Code handelt, der nicht verändert werden sollte. Dies wird verhindert durch die Ausführung des Funktionen- und Prozeduren-

• Für einfache Funktionskomponenten in SQL-Funktionen müssen diese SQL-kompatibel definiert werden. Es kann eine Abfrage oder ein Block definiert und bestimmt von Parameter werte.

• null-Werte führt ein Fehler bei der Ausführung einer Funktion aus. Um dies zu verhindern, muss die Werte einer Exception erzeugt werden. Bei der Ausführung einer solchen Ausnahme sollte sie ein Fehler-Nachschlag (Fehler-Nachschlag) auf Dokumentationen auf die neuen mit der Rückgabe des null-Werts erzeugt werden.

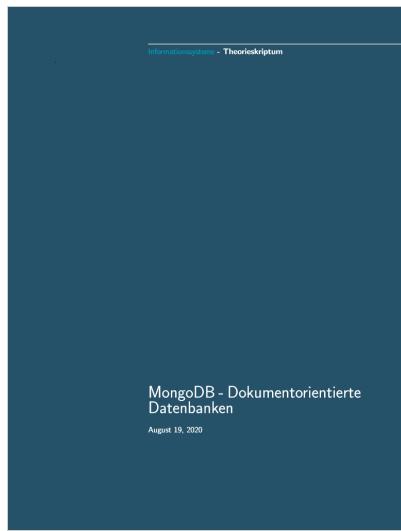
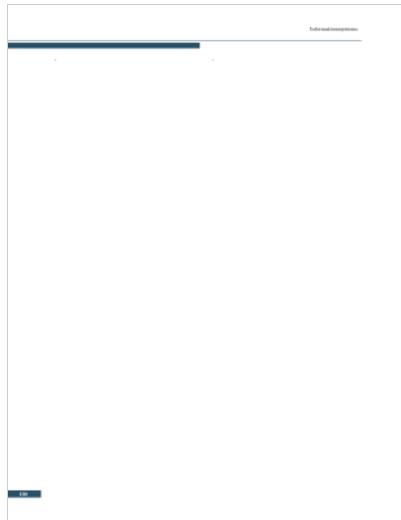
14.1.3 Fallbeispiel: Berechnung der Faktoriell

• Codebeispiel: Faktoriell berechnen

```

CREATE OR REPLACE FUNCTION FAKTORIELL
  (n IN BINARY_INTEGER)
  RETURN NUMBER DETERMINISTIC
  PARALLEL_ENABLE;
BEGIN
  IF n <= 0 THEN
    RAISE VALUE_ERROR;
  END IF;
  FOR i IN 1 .. n LOOP
    l_result := l_result * i;
  END LOOP;
  RETURN l_result;
EXCEPTION
  WHEN numeric_value_error THEN
    RAISE;
END;
  
```





## MongoDB - Dokumentorientierte Datenbanken

August 19, 2020

02	MongoDB - DDL
10	Datencontainer
11	Datencontainer erzeugen
12	Collections und Dokumente
13	Dokumentarchitektur
14	Schemaverweise
15	Views erstellen
16	Datencontainer verwalten

**15.1. Datencontainer**  
Die MongoDB-Datenstruktur verzerrt die Spezialisierung von Dokumenten in logische Sammlungen.

**15.1.1. Datencontainer**  
Zur Verwaltung von Datenobjekten die MongoDB spezifisch definiert 5 Typen von Datencontainern.  
Datencontainer dienen zur Strukturierung und Verarbeitung von Daten.

**Database**  
Als Datenbank werden die Strukturen und Objekte zusammengefasst die in einem gleich zusammenhängenden Datenmodell dargestellt.  
Alle Strukturen führt einzeln ihre Datenbank mit höchsten Normierungsgraden.

**Collection**  
Collections verzetteln Sammlungen gleicher Artige Dokumente.  
Collections können direkt korreliert mit den Strukturen Neutronaler Datenbanken verglichen werden.

**View**  
Views ermöglichen es Benutzerschnittstellen um Objekte auszulesen.

**Capped Collection**  
Capped Collection sind eine Collection die auf einer festen Größe beschränkt ist. Sie wird verwendet um große Mengen von Daten zu verarbeiten und zu konservieren.

**Document**  
MongoDB speichert **Dokumente** in Form von Dokumenten.



Informationssysteme		
Parameter	Beschreibung	
<code>create</code>	Optional. To create a collection, specify that. If you specify this, you must also set <code>an</code> to <code>maximum</code> in this file.	Top bedeckt
<code>createfile</code>	Optional. Specified to enable the automatic creation of an index on the <code>.jar</code> field.	bedeckt
<code>dir</code>	Optional. Specified to enable the automatic creation of an index on the <code>.jar</code> field.	erweitert
<code>max</code>	Optional. Specified to enable the automatic creation of an index on the <code>.jar</code> field. This parameter requires <code>createfile</code> and <code>maxindex</code> to be specified. It also enables the system to collect all files in the folder for collecting and aggregated for other collections.	erweitert
<code>maxid</code>	The maximum number of documents allowed by the <code>copyIndex</code> collection. The value for this parameter is the maximum number of documents that can be indexed. If you set this parameter to <code>1000000</code> , the system removes old documents after 1000000 documents are indexed. It includes the maximum number of documents. MongoDB removes old documents if there are more than 1000000 documents in the collection. You can use this parameter to import collections, or to limit the maximum number of documents.	max number
<code>minid</code>	Optional. Allows users to specify a value in <code>min</code> for the expression for the collection by this document.	min document
<code>minidfile</code>	Optional. Determines how many MongoDB requires the validation rule to create index entries.	min rule
<code>minindex</code>	Optional. Determines whether to use <code>dir</code> for documents or <code>an</code> with <code>maximum</code> to validate them but allow old documents to be inserted.	min rule
<code>size</code>	This option is the source collection or view from which the index is created. This name is not the name of the collection or view that contains the index. This option does not include the database name and includes the schema identifier for the view. For example, <code>db.collection</code> creates the view for the specified collection or the <code>viewCollection</code> or <code>view</code> .	size rule
<code>wireConcern</code>	Optional. A document that expresses the write concern for the operation. Only to use document	wire concern

---

Abläufung MS - `createCollection`

---

### 15.3. Collections und Dokumente

 Collections – Dokumente verwalten, bewahren, geschützen, bearbeiten
+ Erstellung IRON Datenbank = [Erstellen einer Datenbank](#) | [Erstellen eines Dokumentes](#)

**If**

`Collection` –  
Collections verwaltet, bewahren, geschützen, bearbeiten

Diese Collection ist die MongoDB-Dokumentation

**Else**

`IRON` –  
Dokumente verwaltet, bewahren, geschützen, bearbeiten

Diese Collection ist die IRON-Dokumentation

**Die** MongoDB-Dokumentation spricht Ihnen sicher zu. Wenn Sie mit der IRON-Dokumentation sprechen möchten, schreiben Sie mir.

**Die** IRON-Dokumentation spricht Ihnen sicher zu. Wenn Sie mit der MongoDB-Dokumentation sprechen möchten, schreiben Sie mir.

**15.3.1. BSON-Dokument**

Das BSON-Dokument ist eine Variante des JSON-Dokumentes.

**15.3.2. IRON-Datenbank**

Ein IRON-Datenbank ist eine Variante der MongoDB-Datenbank.



**16.4. Dokumentenschema**

Ein Dokumentenschema beschreibt eine Hierarchie eines Informationsobjekts. Dieses ist keine Endstufe, sondern ein Baustein zur Beschreibung der Struktur von DSON-Dokumenten.

**16.4.1. DSON Schema**

In den Dokumenten spezifizieren Sie die Art und Weise, wie die Dokumente strukturiert sind. Eine Collection ist eine Gruppe von Dokumenten, die zusammengehören.

**16.4.2. Schemaattribute**

Ein DSON-Schema sollte mit wieder verwendbaren Schablonen konstruiert werden, um die Struktur von Dokumenten zu vereinfachen.

**Auftragung: Schemaattribute**

- **isType:** Das `isType`-Attribut definiert den Typ des Dokuments. Es kann entweder `Object` oder `String` sein. Der Typ ist Dokumente unterscheidbar.
- **required:** Mit dem `required`-Attribut wird angegeben, ob ein bestimmtes Feld in jedem Dokument vorkommen muss.
- **properties:** Das `properties`-Attribut wird verwendet, um die Felder des Dokuments einzeln zu definieren.

**Erstellung: DSON-Schema**

```

// DSON-Schema
// DSON-Dokument
// project = {
//   title : "Metamorphosis",
//   type : "ARTIFICIAL_PROJECT",
//   status : "INPROGRESS"
// }

// DSON-Dokument
// project = {
//   title : "Metamorphosis",
//   type : "ARTIFICIAL_PROJECT",
//   status : "INPROGRESS"
// }

// DSON-Dokument
// project = {
//   title : "Metamorphosis",
//   type : "ARTIFICIAL_PROJECT",
//   status : "INPROGRESS"
// }

// DSON-Dokument
// project = {
//   title : "Metamorphosis",
//   type : "ARTIFICIAL_PROJECT",
//   status : "INPROGRESS"
// }
  
```

(D) Metamorphosis - Mac - Final Fantasy Dev

Attribut	Beschreibung	Wert	Type
isType	string oder object	Deklaration des Typs eines Feldes	all types
status	string or values	Deklaration oder möglichen Werte für ein Feld	all types
maxLength	number	Definiert den Maximalen Wert eines Feldes	numbers
minValue	number	Indiziert die Minimale Wert des Feldes	numbers
exclusiveMinimum	boolean	If true, sind keine minimaus Werte im Dokument erlaubt	numbers
minValue	number	Indiziert die Minimale Wert des Feldes	numbers
exclusiveMaximum	boolean	If true, sind keine maximaus Werte im Dokument erlaubt	numbers
maxLength	integer	Indiziert die maximale Länge des Feldes	integer
minLength	integer	Indiziert die minimale Länge des Feldes	integer
pattern	regular (REGEX)	RegEx must match the regular expression	string
required	array of strings	Objekt properties auf must control all the specified one	object
additionalProperties	boolean or object	If true, es gibt kein Objekt von Objekten. Wenn es ein Objekt ist, so wird ein DSON-Schema object als Spezifikation definiert, welche andere Felder erlaubt	object
items	base-type	Deklaration des Typs der Elemente die im Array gespezifiziert werden	anytype
maxItems	integer	Indiziert die maximale Anzahl von Elementen	integer
minItems	integer	Indiziert die minimale Anzahl von Elementen	integer
uniqueItems	boolean	If true, each Item in this array must be unique. Otherwise, items may duplicates	boolean
title	string	A descriptive title along with its effect	N/A
description	string	A string that describes the schema and has no effect	N/A

Allgemeinung: ST - DSON-Dokument-Attribute

**16.5. Schemaelemente**

Ein Schemaelement ist ein Objekt, das die Struktur eines Dokumentes festlegt. Es kann aus einem oder mehreren Schablonen bestehen.

**16.5.1. Schemaelement: Eingebettete Objekte**

Die `Object`-Schablonen erlauben die Strukturierung von Dokumenten in verschachtelte Objekte.

**16.5.2. Schemaelement: Datei**

**Auftragung:** Ein Schemaelement mit folgenden Merkmalen:

• Innen werden verwendete um die logische Struktur und Untereinheiten von Programmen zu unterteilen

**Erstellung: Schemaelement: Datei**

```

// DSON-Dokument
// project = {
//   id : "01011101000000000000000000000000",
//   title : "Metamorphosis",
//   subtitle : "Metamorphosis",
//   country : "Russia",
//   status : "INPROGRESS"
// }

// DSON-Dokument
// project = {
//   id : "01011101000000000000000000000000",
//   title : "Metamorphosis",
//   subtitle : "Metamorphosis",
//   country : "Russia",
//   status : "INPROGRESS"
// }

// DSON-Dokument
// project = {
//   id : "01011101000000000000000000000000",
//   title : "Metamorphosis",
//   subtitle : "Metamorphosis",
//   country : "Russia",
//   status : "INPROGRESS"
// }

// DSON-Dokument
// project = {
//   id : "01011101000000000000000000000000",
//   title : "Metamorphosis",
//   subtitle : "Metamorphosis",
//   country : "Russia",
//   status : "INPROGRESS"
// }
  
```



```

@@ Druckepg. Msc. Paul Pfeiffer Hsu

15.7. Datencontainer verwärfen

15.7.1 Container verwärfen

Wir wollen eine Schale nach die wichtigsten Methoden der Verwaltung von Datenstrukturen anlegen.



```

1 // Container.h
2 // Spez. getContainer();
3 // db_getContainer();
4 // db_getContainerByName();
5 // var personnel = db_getContainer("personen");
6
7 personnel.addPerson();
8
9 // Create new Person
10 // birth : new Date("Jan 20, 1910");
11 // death : new Date("Oct 07, 1980");
12 // name : "Hans Müller";
13 // user_name : "HansM";
14 // address : "Hans-Müller-Str. 12";
15 // phone : "+49(0)30/1234567";
16 // web : "http://www.datenbank.net";
17
18
19 // Insert new value
20 var education = {
21   // Schulname : "Hans-Schule", 
22   // "interestedIn" : [BspData("Mathematik")];
23 }
24
25 // ...
26
27 // Spez. getContainerByName();
28 // db_getContainerByName();
29 // db_getContainerByName();
30
31 var school;
32
33 // Insert value
34 var teacher = {
35   // name : "Hans Müller", 
36   // "subject": "Mathematik"
37 };
38
39 // ...
40
41 // Spez. drop();
42 // db_drop();
43
44 // db_getContainerByName.drop();
45
46 // db_teachers.drop();
47 // db_students.drop();

```


```











Informationsrahmen	
<b>16.3.5. Ein Operator</b>	<pre>Ma das Ein Operator kann geprüft werden, ob ein bestimmter Wert in einer Liste von Werten enthalten ist.</pre> <p>↓↓↓↓↓</p> <pre>1 // Ein Operator 2 if( ... ) 3 { ... 4   <b>if</b>( <b>ArrayList</b> <b>list</b> != null ) 5   { 6     <b>int</b> value = ... 7     <b>for</b> ( <b>int</b> i = 0; i &lt; list.size(); i++ ) 8     { 9       <b>if</b> (list.get(i) == value) 10      ... 11    } 12  }</pre>
<b>16.3.6. Beide Operator</b>	<pre>Komplexe Abfrage können auch in Form einer Zweispieler-Funktion formuliert werden.</pre> <p>↓↓↓↓↓</p> <pre>1 <b>returning</b>: <b>Beide Operator</b> * 2 { 3   // Beide Operator 4   <b>if</b> ( ... ) 5   { 6     <b>ArrayList</b> <b>list</b> = ... 7     <b>int</b> value = ... 8     <b>for</b> ( <b>int</b> i = 0; i &lt; list.size(); i++ ) 9     { 10       <b>if</b> (list.get(i) == value) 11         ... 12     } 13   } 14 }</pre>
<b>16.4. Arrayabfragen</b>	<p>Für ArrayList und ArrayListe-Artefakt die <b>ArrayList</b> Spezialabfrage gegen Operatoren.</p>
<b>16.4.1. Soziale Operator</b>	<p>Der Soziale Operator prüft die Anzahl der Werte in einer Liste.</p> <p>↓↓↓↓↓</p> <pre>1 <b>Der Soziale Operator erhält stets folglich Ausgabe:</b> 2 3   ↓↓↓↓↓ 4   1. <b>return</b> Operator 5   2. <b>ArrayList</b> <b>list</b> 6   3. <b>ArrayList</b> <b>list</b> 7   4. <b>ArrayList</b> <b>list</b> 8   5. <b>int</b> <b>i</b> 9   6. <b>int</b> <b>i</b> <b>count</b> 10  7. <b>int</b> <b>i</b> <b>count</b> 11  8. <b>int</b> <b>i</b> <b>count</b> 12  9. <b>int</b> <b>i</b> <b>count</b> 13  10. <b>int</b> <b>i</b> <b>count</b> 14  11. <b>int</b> <b>i</b> <b>count</b> 15  12. <b>int</b> <b>i</b> <b>count</b> 16  13. <b>int</b> <b>i</b> <b>count</b> 17  14. <b>int</b> <b>i</b> <b>count</b> 18  15. <b>int</b> <b>i</b> <b>count</b> 19  16. <b>int</b> <b>i</b> <b>count</b> 20  17. <b>int</b> <b>i</b> <b>count</b> 21  18. <b>int</b> <b>i</b> <b>count</b> 22  19. <b>int</b> <b>i</b> <b>count</b> 23  20. <b>int</b> <b>i</b> <b>count</b> 24  21. <b>int</b> <b>i</b> <b>count</b> 25  22. <b>int</b> <b>i</b> <b>count</b> 26  23. <b>int</b> <b>i</b> <b>count</b> 27  24. <b>int</b> <b>i</b> <b>count</b> 28  25. <b>int</b> <b>i</b> <b>count</b> 29  26. <b>int</b> <b>i</b> <b>count</b> 30  27. <b>int</b> <b>i</b> <b>count</b> 31  28. <b>int</b> <b>i</b> <b>count</b> 32  29. <b>int</b> <b>i</b> <b>count</b> 33  30. <b>int</b> <b>i</b> <b>count</b> 34  31. <b>int</b> <b>i</b> <b>count</b> 35  32. <b>int</b> <b>i</b> <b>count</b> 36  33. <b>int</b> <b>i</b> <b>count</b> 37  34. <b>int</b> <b>i</b> <b>count</b> 38  35. <b>int</b> <b>i</b> <b>count</b> 39  36. <b>int</b> <b>i</b> <b>count</b> 40  37. <b>int</b> <b>i</b> <b>count</b> 41  38. <b>int</b> <b>i</b> <b>count</b> 42  39. <b>int</b> <b>i</b> <b>count</b> 43  40. <b>int</b> <b>i</b> <b>count</b> 44  41. <b>int</b> <b>i</b> <b>count</b> 45  42. <b>int</b> <b>i</b> <b>count</b> 46  43. <b>int</b> <b>i</b> <b>count</b> 47  44. <b>int</b> <b>i</b> <b>count</b> 48  45. <b>int</b> <b>i</b> <b>count</b> 49  46. <b>int</b> <b>i</b> <b>count</b> 50  47. <b>int</b> <b>i</b> <b>count</b> 51  48. <b>int</b> <b>i</b> <b>count</b> 52  49. <b>int</b> <b>i</b> <b>count</b> 53  50. <b>int</b> <b>i</b> <b>count</b> 54  51. <b>int</b> <b>i</b> <b>count</b> 55  52. <b>int</b> <b>i</b> <b>count</b> 56  53. <b>int</b> <b>i</b> <b>count</b> 57  54. <b>int</b> <b>i</b> <b>count</b> 58  55. <b>int</b> <b>i</b> <b>count</b> 59  56. <b>int</b> <b>i</b> <b>count</b> 60  57. <b>int</b> <b>i</b> <b>count</b> 61  58. <b>int</b> <b>i</b> <b>count</b> 62  59. <b>int</b> <b>i</b> <b>count</b> 63  60. <b>int</b> <b>i</b> <b>count</b> 64  61. <b>int</b> <b>i</b> <b>count</b> 65  62. <b>int</b> <b>i</b> <b>count</b> 66  63. <b>int</b> <b>i</b> <b>count</b> 67  64. <b>int</b> <b>i</b> <b>count</b> 68  65. <b>int</b> <b>i</b> <b>count</b> 69  66. <b>int</b> <b>i</b> <b>count</b> 70  67. <b>int</b> <b>i</b> <b>count</b> 71  68. <b>int</b> <b>i</b> <b>count</b> 72  69. <b>int</b> <b>i</b> <b>count</b> 73  70. <b>int</b> <b>i</b> <b>count</b> 74  71. <b>int</b> <b>i</b> <b>count</b> 75  72. <b>int</b> <b>i</b> <b>count</b> 76  73. <b>int</b> <b>i</b> <b>count</b> 77  74. <b>int</b> <b>i</b> <b>count</b> 78  75. <b>int</b> <b>i</b> <b>count</b> 79  76. <b>int</b> <b>i</b> <b>count</b> 80  77. <b>int</b> <b>i</b> <b>count</b> 81  78. <b>int</b> <b>i</b> <b>count</b> 82  79. <b>int</b> <b>i</b> <b>count</b> 83  80. <b>int</b> <b>i</b> <b>count</b> 84  81. <b>int</b> <b>i</b> <b>count</b> 85  82. <b>int</b> <b>i</b> <b>count</b> 86  83. <b>int</b> <b>i</b> <b>count</b> 87  84. <b>int</b> <b>i</b> <b>count</b> 88  85. <b>int</b> <b>i</b> <b>count</b> 89  86. <b>int</b> <b>i</b> <b>count</b> 90  87. <b>int</b> <b>i</b> <b>count</b> 91  88. <b>int</b> <b>i</b> <b>count</b> 92  89. <b>int</b> <b>i</b> <b>count</b> 93  90. <b>int</b> <b>i</b> <b>count</b> 94  91. <b>int</b> <b>i</b> <b>count</b> 95  92. <b>int</b> <b>i</b> <b>count</b> 96  93. <b>int</b> <b>i</b> <b>count</b> 97  94. <b>int</b> <b>i</b> <b>count</b> 98  95. <b>int</b> <b>i</b> <b>count</b> 99  96. <b>int</b> <b>i</b> <b>count</b> 100 97. <b>int</b> <b>i</b> <b>count</b> 101 98. <b>int</b> <b>i</b> <b>count</b> 102 99. <b>int</b> <b>i</b> <b>count</b> 103 100. <b>int</b> <b>i</b> <b>count</b> 104 101. <b>int</b> <b>i</b> <b>count</b> 105 102. <b>int</b> <b>i</b> <b>count</b> 106 103. <b>int</b> <b>i</b> <b>count</b> 107 104. <b>int</b> <b>i</b> <b>count</b> 108 105. <b>int</b> <b>i</b> <b>count</b> 109 106. <b>int</b> <b>i</b> <b>count</b> 110 107. <b>int</b> <b>i</b> <b>count</b> 111 108. <b>int</b> <b>i</b> <b>count</b> 112 109. <b>int</b> <b>i</b> <b>count</b> 113 110. <b>int</b> <b>i</b> <b>count</b> 114 111. <b>int</b> <b>i</b> <b>count</b> 115 112. <b>int</b> <b>i</b> <b>count</b> 116 113. <b>int</b> <b>i</b> <b>count</b> 117 114. <b>int</b> <b>i</b> <b>count</b> 118 115. <b>int</b> <b>i</b> <b>count</b> 119 116. <b>int</b> <b>i</b> <b>count</b> 120 117. <b>int</b> <b>i</b> <b>count</b> 121 118. <b>int</b> <b>i</b> <b>count</b> 122 119. <b>int</b> <b>i</b> <b>count</b> 123 120. <b>int</b> <b>i</b> <b>count</b> 124 121. <b>int</b> <b>i</b> <b>count</b> 125 122. <b>int</b> <b>i</b> <b>count</b> 126 123. <b>int</b> <b>i</b> <b>count</b> 127 124. <b>int</b> <b>i</b> <b>count</b> 128 125. <b>int</b> <b>i</b> <b>count</b> 129 126. <b>int</b> <b>i</b> <b>count</b> 130 127. <b>int</b> <b>i</b> <b>count</b> 131 128. <b>int</b> <b>i</b> <b>count</b> 132 129. <b>int</b> <b>i</b> <b>count</b> 133 130. <b>int</b> <b>i</b> <b>count</b> 134 131. <b>int</b> <b>i</b> <b>count</b> 135 132. <b>int</b> <b>i</b> <b>count</b> 136 133. <b>int</b> <b>i</b> <b>count</b> 137 134. <b>int</b> <b>i</b> <b>count</b> 138 135. <b>int</b> <b>i</b> <b>count</b> 139 136. <b>int</b> <b>i</b> <b>count</b> 140 137. <b>int</b> <b>i</b> <b>count</b> 141 138. <b>int</b> <b>i</b> <b>count</b> 142 139. <b>int</b> <b>i</b> <b>count</b> 143 140. <b>int</b> <b>i</b> <b>count</b> 144 141. <b>int</b> <b>i</b> <b>count</b> 145 142. <b>int</b> <b>i</b> <b>count</b> 146 143. <b>int</b> <b>i</b> <b>count</b> 147 144. <b>int</b> <b>i</b> <b>count</b> 148 145. <b>int</b> <b>i</b> <b>count</b> 149 146. <b>int</b> <b>i</b> <b>count</b> 150 147. <b>int</b> <b>i</b> <b>count</b> 151 148. <b>int</b> <b>i</b> <b>count</b> 152 149. <b>int</b> <b>i</b> <b>count</b> 153 150. <b>int</b> <b>i</b> <b>count</b> 154 151. <b>int</b> <b>i</b> <b>count</b> 155 152. <b>int</b> <b>i</b> <b>count</b> 156 153. <b>int</b> <b>i</b> <b>count</b> 157 154. <b>int</b> <b>i</b> <b>count</b> 158 155. <b>int</b> <b>i</b> <b>count</b> 159 156. <b>int</b> <b>i</b> <b>count</b> 160 157. <b>int</b> <b>i</b> <b>count</b> 161 158. <b>int</b> <b>i</b> <b>count</b> 162 159. <b>int</b> <b>i</b> <b>count</b> 163 160. <b>int</b> <b>i</b> <b>count</b> 164 161. <b>int</b> <b>i</b> <b>count</b> 165 162. <b>int</b> <b>i</b> <b>count</b> 166 163. <b>int</b> <b>i</b> <b>count</b> 167 164. <b>int</b> <b>i</b> <b>count</b> 168 165. <b>int</b> <b>i</b> <b>count</b> 169 166. <b>int</b> <b>i</b> <b>count</b> 170 167. <b>int</b> <b>i</b> <b>count</b> 171 168. <b>int</b> <b>i</b> <b>count</b> 172 169. <b>int</b> <b>i</b> <b>count</b> 173 170. <b>int</b> <b>i</b> <b>count</b> 174 171. <b>int</b> <b>i</b> <b>count</b> 175 172. <b>int</b> <b>i</b> <b>count</b> 176 173. <b>int</b> <b>i</b> <b>count</b> 177 174. <b>int</b> <b>i</b> <b>count</b> 178 175. <b>int</b> <b>i</b> <b>count</b> 179 176. <b>int</b> <b>i</b> <b>count</b> 180 177. <b>int</b> <b>i</b> <b>count</b> 181 178. <b>int</b> <b>i</b> <b>count</b> 182 179. <b>int</b> <b>i</b> <b>count</b> 183 180. <b>int</b> <b>i</b> <b>count</b> 184 181. <b>int</b> <b>i</b> <b>count</b> 185 182. <b>int</b> <b>i</b> <b>count</b> 186 183. <b>int</b> <b>i</b> <b>count</b> 187 184. <b>int</b> <b>i</b> <b>count</b> 188 185. <b>int</b> <b>i</b> <b>count</b> 189 186. <b>int</b> <b>i</b> <b>count</b> 190 187. <b>int</b> <b>i</b> <b>count</b> 191 188. <b>int</b> <b>i</b> <b>count</b> 192 189. <b>int</b> <b>i</b> <b>count</b> 193 190. <b>int</b> <b>i</b> <b>count</b> 194 191. <b>int</b> <b>i</b> <b>count</b> 195 192. <b>int</b> <b>i</b> <b>count</b> 196 193. <b>int</b> <b>i</b> <b>count</b> 197 194. <b>int</b> <b>i</b> <b>count</b> 198 195. <b>int</b> <b>i</b> <b>count</b> 199 196. <b>int</b> <b>i</b> <b>count</b> 200 197. <b>int</b> <b>i</b> <b>count</b> 201 198. <b>int</b> <b>i</b> <b>count</b> 202 199. <b>int</b> <b>i</b> <b>count</b> 203 200. <b>int</b> <b>i</b> <b>count</b> 204 201. <b>int</b> <b>i</b> <b>count</b> 205 202. <b>int</b> <b>i</b> <b>count</b> 206 203. <b>int</b> <b>i</b> <b>count</b> 207 204. <b>int</b> <b>i</b> <b>count</b> 208 205. <b>int</b> <b>i</b> <b>count</b> 209 206. <b>int</b> <b>i</b> <b>count</b> 210 207. <b>int</b> <b>i</b> <b>count</b> 211 208. <b>int</b> <b>i</b> <b>count</b> 212 209. <b>int</b> <b>i</b> <b>count</b> 213 210. <b>int</b> <b>i</b> <b>count</b> 214 211. <b>int</b> <b>i</b> <b>count</b> 215 212. <b>int</b> <b>i</b> <b>count</b> 216 213. <b>int</b> <b>i</b> <b>count</b> 217 214. <b>int</b> <b>i</b> <b>count</b> 218 215. <b>int</b> <b>i</b> <b>count</b> 219 216. <b>int</b> <b>i</b> <b>count</b> 220 217. <b>int</b> <b>i</b> <b>count</b> 221 218. <b>int</b> <b>i</b> <b>count</b> 222 219. <b>int</b> <b>i</b> <b>count</b> 223 220. <b>int</b> <b>i</b> <b>count</b> 224 221. <b>int</b> <b>i</b> <b>count</b> 225 222. <b>int</b> <b>i</b> <b>count</b> 226 223. <b>int</b> <b>i</b> <b>count</b> 227 224. <b>int</b> <b>i</b> <b>count</b> 228 225. <b>int</b> <b>i</b> <b>count</b> 229 226. <b>int</b> <b>i</b> <b>count</b> 230 227. <b>int</b> <b>i</b> <b>count</b> 231 228. <b>int</b> <b>i</b> <b>count</b> 232 229. <b>int</b> <b>i</b> <b>count</b> 233 230. <b>int</b> <b>i</b> <b>count</b> 234 231. <b>int</b> <b>i</b> <b>count</b> 235 232. <b>int</b> <b>i</b> <b>count</b> 236 233. <b>int</b> <b>i</b> <b>count</b> 237 234. <b>int</b> <b>i</b> <b>count</b> 238 235. <b>int</b> <b>i</b> <b>count</b> 239 236. <b>int</b> <b>i</b> <b>count</b> 240 237. <b>int</b> <b>i</b> <b>count</b> 241 238. <b>int</b> <b>i</b> <b>count</b> 242 239. <b>int</b> <b>i</b> <b>count</b> 243 240. <b>int</b> <b>i</b> <b>count</b> 244 241. <b>int</b> <b>i</b> <b>count</b> 245 242. <b>int</b> <b>i</b> <b>count</b> 246 243. <b>int</b> <b>i</b> <b>count</b> 247 244. <b>int</b> <b>i</b> <b>count</b> 248 245. <b>int</b> <b>i</b> <b>count</b> 249 246. <b>int</b> <b>i</b> <b>count</b> 250 247. <b>int</b> <b>i</b> <b>count</b> 251 248. <b>int</b> <b>i</b> <b>count</b> 252 249. <b>int</b> <b>i</b> <b>count</b> 253 250. <b>int</b> <b>i</b> <b>count</b> 254 251. <b>int</b> <b>i</b> <b>count</b> 255 252. <b>int</b> <b>i</b> <b>count</b> 256 253. <b>int</b> <b>i</b> <b>count</b> 257 254. <b>int</b> <b>i</b> <b>count</b> 258 255. <b>int</b> <b>i</b> <b>count</b> 259 256. <b>int</b> <b>i</b> <b>count</b> 260 257. <b>int</b> <b>i</b> <b>count</b> 261 258. <b>int</b> <b>i</b> <b>count</b> 262 259. <b>int</b> <b>i</b> <b>count</b> 263 260. <b>int</b> <b>i</b> <b>count</b> 264 261. <b>int</b> <b>i</b> <b>count</b> 265 262. <b>int</b> <b>i</b> <b>count</b> 266 263. <b>int</b> <b>i</b> <b>count</b> 267 264. <b>int</b> <b>i</b> <b>count</b> 268 265. <b>int</b> <b>i</b> <b>count</b> 269 266. <b>int</b> <b>i</b> <b>count</b> 270 267. <b>int</b> <b>i</b> <b>count</b> 271 268. <b>int</b> <b>i</b> <b>count</b> 272 269. <b>int</b> <b>i</b> <b>count</b> 273 270. <b>int</b> <b>i</b> <b>count</b> 274 271. <b>int</b> <b>i</b> <b>count</b> 275 272. <b>int</b> <b>i</b> <b>count</b> 276 273. <b>int</b> <b>i</b> <b>count</b> 277 274. <b>int</b> <b>i</b> <b>count</b> 278 275. <b>int</b> <b>i</b> <b>count</b> 279 276. <b>int</b> <b>i</b> <b>count</b> 280 277. <b>int</b> <b>i</b> <b>count</b> 281 278. <b>int</b> <b>i</b> <b>count</b> 282 279. <b>int</b> <b>i</b> <b>count</b> 283 280. <b>int</b> <b>i</b> <b>count</b> 284 281. <b>int</b> <b>i</b> <b>count</b> 285 282. <b>int</b> <b>i</b> <b>count</b> 286 283. <b>int</b> <b>i</b> <b>count</b> 287 284. <b>int</b> <b>i</b> <b>count</b> 288 285. <b>int</b> <b>i</b> <b>count</b> 289 286. <b>int</b> <b>i</b> <b>count</b> 290 287. <b>int</b> <b>i</b> <b>count</b> 291 288. <b>int</b> <b>i</b> <b>count</b> 292 289. <b>int</b> <b>i</b> <b>count</b> 293 290. <b>int</b> <b>i</b> <b>count</b> 294 291. <b>int</b> <b>i</b> <b>count</b> 295 292. <b>int</b> <b>i</b> <b>count</b> 296 293. <b>int</b> <b>i</b> <b>count</b> 297 294. <b>int</b> <b>i</b> <b>count</b> 298 295. <b>int</b> <b>i</b> <b>count</b> 299 296. <b>int</b> <b>i</b> <b>count</b> 300 297. <b>int</b> <b>i</b> <b>count</b> 301 298. <b>int</b> <b>i</b> <b>count</b> 302 299. <b>int</b> <b>i</b> <b>count</b> 303 300. <b>int</b> <b>i</b> <b>count</b> 304 301. <b>int</b> <b>i</b> <b>count</b> 305 302. <b>int</b> <b>i</b> <b>count</b> 306 303. <b>int</b> <b>i</b> <b>count</b> 307 304. <b>int</b> <b>i</b> <b>count</b> 308 305. <b>int</b> <b>i</b> <b>count</b> 309 306. <b>int</b> <b>i</b> <b>count</b> 310 307. <b>int</b> <b>i</b> <b>count</b> 311 308. <b>int</b> <b>i</b> <b>count</b> 312 309. <b>int</b> <b>i</b> <b>count</b> 313 310. <b>int</b> <b>i</b> <b>count</b> 314 311. <b>int</b> <b>i</b> <b>count</b> 315 312. <b>int</b> <b>i</b> <b>count</b> 316 313. <b>int</b> <b>i</b> <b>count</b> 317 314. <b>int</b> <b>i</b> <b>count</b> 318 315. <b>int</b> <b>i</b> <b>count</b> 319 316. <b>int</b> <b>i</b> <b>count</b> 320 317. <b>int</b> <b>i</b> <b>count</b> 321 318. <b>int</b> <b>i</b> <b>count</b> 322 319. <b>int</b> <b>i</b> <b>count</b> 323 320. <b>int</b> <b>i</b> <b>count</b> 324 321. <b>int</b> <b>i</b> <b>count</b> 325 322. <b>int</b> <b>i</b> <b>count</b> 326 323. <b>int</b> <b>i</b> <b>count</b> 327 324. <b>int</b> <b>i</b> <b>count</b> 328 325. <b>int</b> <b>i</b> <b>count</b> 329 326. <b>int</b> <b>i</b> <b>count</b> 330 327. <b>int</b> <b>i</b> <b>count</b> 331 328. <b>int</b> <b>i</b> <b>count</b> 332 329. <b>int</b> <b>i</b> <b>count</b> 333 330. <b>int</b> <b>i</b> <b>count</b> 334 331. <b>int</b> <b>i</b> <b>count</b> 335 332. <b>int</b> <b>i</b> <b>count</b> 336 333. <b>int</b> <b>i</b> <b>count</b> 337 334. <b>int</b> <b>i</b> <b>count</b> 338 335. <b>int</b> <b>i</b> <b>count</b> 339 336. <b>int</b> <b>i</b> <b>count</b> 340 337. <b>int</b> <b>i</b> <b>count</b> 341 338. <b>int</b> <b>i</b> <b>count</b> 342 339. <b>int</b> <b>i</b> <b>count</b> 343 340. <b>int</b> <b>i</b> <b>count</b> 344 341. <b>int</b> <b>i</b> <b>count</b> 345 342. <b>int</b> <b>i</b> <b>count</b> 346 343. <b>int</b> <b>i</b> <b>count</b> 347 344. <b>int</b> <b>i</b> <b>count</b> 348 345. <b>int</b> <b>i</b> <b>count</b> 349 346. <b>int</b> <b>i</b> <b>count</b> 350 347. <b>int</b> <b>i</b> <b>count</b> 351 348. <b>int</b> <b>i</b> <b>count</b> 352 349. <b>int</b> <b>i</b> <b>count</b> 353 350. <b>int</b> <b>i</b> <b>count</b> 354 351. <b>int</b> <b>i</b> <b>count</b> 355 352. <b>int</b> <b>i</b> <b>count</b> 356 353. <b>int</b> <b>i</b> <b>count</b> 357 354. <b>int</b> <b>i</b> <b>count</b> 358 355. <b>int</b> <b>i</b> <b>count</b> 359 356. <b>int</b> <b>i</b> <b>count</b> 360 357. <b>int</b> <b>i</b> <b>count</b> 361 358. <b>int</b> <b>i</b> <b>count</b> 362 359. <b>int</b> <b>i</b> <b>count</b> 363 360. <b>int</b> <b>i</b> <b>count</b> 364 361. <b>int</b> <b>i</b> <b>count</b> 365 362. <b>int</b> <b>i</b> <b>count</b> 366 363. <b>int</b> <b>i</b> <b>count</b> 367 364. <b>int</b> <b>i</b> <b>count</b> 368 365. <b>int</b> <b>i</b> <b>count</b> 369 366. <b>int</b> <b>i</b> <b>count</b> 370 367. <b>int</b> <b>i</b> <b>count</b> 371 368. <b>int</b> <b>i</b> <b>count</b> 372 369. <b>int</b> <b>i</b> <b>count</b> 373 370. <b>int</b> <b>i</b> <b>count</b> 374 371. <b>int</b> <b>i</b> <b>count</b> 375 372. <b>int</b> <b>i</b> <b>count</b> 376 373. <b>int</b> <b>i</b> <b>count</b> 377 374. <b>int</b> <b>i</b> <b>count</b> 378 375. <b>int</b> <b>i</b> <b>count</b> 379 376. <b>int</b> <b>i</b> <b>count</b> 380 377. <b>int</b> <b>i</b> <b>count</b> 381 378. <b>int</b> <b>i</b> <b>count</b> 382 379. <b>int</b> <b>i</b> <b>count</b> 383 380. <b>int</b> <b>i</b> <b>count</b> 384 381. <b>int</b> <b>i</b> <b>count</b> 385 382. <b>int</b> <b>i</b> <b>count</b> 386 383. <b>int</b> <b>i</b> <b>count</b> 387 384. <b>int</b> <b>i</b> <b>count</b> 388 385. <b>int</b> <b>i</b> <b>count</b> 389 386. <b>int</b> <b>i</b> <b>count</b> 390 387. <b>int</b> <b>i</b> <b>count</b> 391 388. <b>int</b> <b>i</b> <b>count</b> 392 389. <b>int</b> <b>i</b> <b>count</b> 393 390. <b>int</b> <b>i</b> <b>count</b> 394 391. <b>int</b> <b>i</b> <b>count</b> 395 392. <b>int</b> <b>i</b> <b>count</b> 396 393. <b>int</b> <b>i</b> <b>count</b> 397 394. <b>int</b> <b>i</b> <b>count</b> 398 395. <b>int</b> <b>i</b> <b>count</b> 399 396. <b>int</b> <b>i</b> <b>count</b> 400 397. <b>int</b> <b>i</b> <b>count</b> 401 398. <b>int</b> <b>i</b> <b>count</b> 402 399. <b>int</b> <b>i</b> <b>count</b> 403 400. <b>int</b> <b>i</b> <b>count</b> 404 401. <b>int</b> <b>i</b> <b>count</b> 405 402. <b>int</b> <b>i</b> <b>count</b> 406 403. <b>int</b> <b>i</b> <b>count</b> 407 404. <b>int</b> <b>i</b> <b>count</b> 408 405. <b>int</b> <b>i</b> <b>count</b> 409 406. <b>int</b> <b>i</b> <b>count</b> 410 407. <b>int</b> <b>i</b> <b>count</b> 411 408. <b>int</b> <b>i</b> <b>count</b> 412 409. <b>int</b> <b>i</b> <b>count</b> 413 410. <b>int</b> <b>i</b> <b>count</b> 414 411. <b>int</b> <b>i</b> <b>count</b> 415 412. <b>int</b> <b>i</b> <b>count</b> 416 413. <b>int</b> <b>i</b> <b>count</b> 417 414. <b>int</b> <b>i</b> <b>count</b> 418 415. <b>int</b> <b>i</b> <b>count</b> 419 416. <b>int</b> <b>i</b> <b>count</b> 420 417. <b>int</b> <b>i</b> <b>count</b> 421 418. <b>int</b> <b>i</b> <b>count</b> 422 419. <b>int</b> <b>i</b></pre>

**17. MongoDB - DML**

# 03

MongoDB - DML

**17.1. Daten einfügen - insert**

Für die Verarbeitung von Data stellt die MongoDB Spezialisierung weitere Formen des `insert()` Befehls vor. Verfugbar:

```

17.1.1. insertOne(), insertMany() Befehl
  Der Inset Befehl wird immer nur einer einzelnen Dokumente ausgeführt.

  +-----+-----+
  | $open | insertOne |
  +-----+-----+
  | $collection| insertOne(document) |
  +-----+-----+
  | $open | insertMany |
  +-----+-----+
  | $collection| insertMany(documents) |
  +-----+-----+
  | $close |-----|
  +-----+-----+
  | $open | insertOne |
  +-----+-----+
  | $open | document |
  +-----+-----+
  | $open | $open |-----|
  |   | type |-----|
  |   | $string |-----|
  |   | "wert" |-----|
  | $close |-----|
  | $close |-----|
  | $close |-----|
  +-----+-----+
  | $open | print() |
  +-----+-----+
  | $close |-----|
  +-----+-----+
  | $open | insertMany |
  +-----+-----+
  | $try |-----|
  | $open | $genError |
  +-----+-----+
  | $close |-----|
  | $catch |-----|
  | $open | document |
  +-----+-----+
  | $open | $open |-----|
  |   | type |-----|
  |   | $string |-----|
  |   | "wert" |-----|
  | $close |-----|
  | $close |-----|
  | $close |-----|
  +-----+-----+
  | $open | print() |
  +-----+-----+
  | $close |-----|
  +-----+-----+

```

**Erklärung: Unterschiede des Inset Befehls**

- Einzel-Dokumente**: In der Collection treten in der Regel höchstens Niedrige Anzahl auf.
- Collection**: Wird die `insert()` Methode mit einer Liste von Dokumenten aufgerufen, so wird die Collection die Collectionen integriert an.
- Objekts**: Für Dokumente dass eine ID enthalten, kann die `insert()` Methode die `update()` Methode auffordern das Dokument zu aktualisieren.



§ 10 Absatz 1 Buchstabe b)	
db->collection("insertions")-	
name = "aus",	← field value
age = 26,	← field value
status = "pending"	
};	
	<b>Abbildung 63: insert document</b>
+ Collection::insertOne(Document& doc) -	<b>17.2. Daten bearbeiten - update</b>
DBSession session;	Zur Bearbeitung eines Dokuments öffnet die MongoClient-Klasse eine Session und gibt sie zurück.
DBCollection collection = session.getCollection("insertions", ...);	Die DBCollection-Klasse liefert das spezifische Objekt für die Bearbeitung von Dokumenten.
DBObjectID id = collection.insertOne(doc).getInsertedId();	Die DBObjectID-Klasse stellt ein Objekt zur Verarbeitung von Dokumenten-IDs bereit.
// insertOne() returns inserted_id	
// insertMany() inserts multiple documents	
// updateOne() updates one document	<b>17.2.1 updateOne() Befehl</b>
// updateMany() updates multiple documents	Der updateOne-Befehl ändert einzelne Dokumente. Er erfordert eine spezifische ID und eine Welligkeit.
// updateOne(Document query, Document update, UpdateOptions options)	Die UpdateOptions-Klasse definiert die Optionen für den updateOne-Befehl.
// updateMany(Document query, Document update, UpdateOptions options)	
// updateOne(Document query, Document update, UpdateOptions options, UpdateResult result)	
// updateMany(Document query, Document update, UpdateOptions options, UpdateResult result)	
// updateOne(Document query, Document update, UpdateOptions options, UpdateResult result, UpdateOptions upsert)	
// updateMany(Document query, Document update, UpdateOptions options, UpdateResult result, UpdateOptions upsert)	
// updateOne(Document query, Document update, UpdateOptions options, UpdateResult result, UpdateOptions upsert, UpdateOptions upsertOptions)	
// updateMany(Document query, Document update, UpdateOptions options, UpdateResult result, UpdateOptions upsert, UpdateOptions upsertOptions)	
	<b>Erklärung: updateOne Parameter</b>
	• spezifiziert die ID des Dokuments, das bearbeitet werden soll. Die Parameter <code>query</code> und <code>update</code> müssen korrekt eingegeben werden.
	• spezifiziert die Optionen, die der Parameter <code>options</code> definiert. Die MongoClient-Klasse definiert diese in unterschiedlichen Optionen.

Abbildung 61: *soforti* (Prozent)

<p>(c) (Hilfe): Min. Preis Produkte-Daten</p> <hr/> <p><b>1.2.2. optimierung Befehl</b></p> <p>Die optimierende Methode reduziert nach Ausführen relevante Dokumente.</p> <pre> <b>&gt; Codekodierung optimierung</b>   1 // Optimierung Methoden reduziert nach Ausführen   2 relevante Dokumente.   3   4 <b>&gt; Codekodierung optimierung</b>   5   // optimierung Operator   6   // ...   7   // do optimierung optimierung   8   // query criteries,   9   // update,   10  // ...   11   12  try {   13    do optimierung optimierung   14    // Variablen: \$query (\$query : String),   15    // \$base ("Firma") : String   16  }   17   18  I variable doc (\$)   19   20  print(doc);   21   22   23 <b>&gt; Codekodierung optimierung</b>   24   25 // ...   26 // Beispieldok. optimierung   27   28 var environment = {   29   id : "1",   30   name : "Rock A Rollin' Rat",   31   evaluations : 2   32 }   33   34 var environment = {   35   id : "2",   36   name : "Positive State Bob",   37   evaluations : 1   38 }   39   40 do optimierung optimierung   41 // Variablen: \$query (\$query : String),   42 // \$base ("Firma") : String   43   44   45 var environment = {   46   id : "3",   47   name : "Positive State Bob",   48   evaluations : 1   49   review : true   50 }   </pre>	<p><b>1.2.3. Strukturoperatoren</b></p> <hr/> <p>Die Strukturoperatoren sind die Basis für das Ausführen von Daten und gleichzeitig ein-</p> <table border="1" data-bbox="236 1214 386 1617"> <thead> <tr> <th colspan="2"><b>1.2.3.1. Best Operator</b></th> </tr> </thead> <tbody> <tr> <td data-bbox="236 1214 295 1256"> <p>Der Best Operator erkennt die einzelnen Werte eines Ausdrucks an denken.</p> </td><td data-bbox="295 1214 386 1256"> <p><b>&gt; Codekodierung Best Operator</b></p> <p>Die Best Operator ist verwandt wie die Wert eines Ausdrucks an denken.</p> </td></tr> <tr> <td data-bbox="236 1256 295 1310"> <p><b>&gt; Codekodierung insertQuery</b></p> <p>Attribut eines neuen Werts aktualisieren Wert position.</p> </td><td data-bbox="295 1256 386 1310"> <p>Best Operator kann die entsprechende Fähigkeit nicht, weil es nicht weiß wo es was im Dokument benötigt.</p> </td></tr> <tr> <td data-bbox="236 1310 295 1364"> <p><b>&gt; Codekodierung Best Operator</b></p> <p>Attribut eines neuen Werts aktualisieren Wert position.</p> </td><td data-bbox="295 1310 386 1364"> <p><b>&gt; Codekodierung insertQuery</b></p> <p>Attribut eines neuen Werts aktualisieren Wert position.</p> </td></tr> <tr> <td data-bbox="236 1364 295 1418"> <p><b>&gt; Codekodierung Best Operator</b></p> <p>Attribut eines neuen Werts aktualisieren Wert position.</p> </td><td data-bbox="295 1364 386 1418"> <p><b>&gt; Codekodierung insertQuery</b></p> <p>Attribut eines neuen Werts aktualisieren Wert position.</p> </td></tr> <tr> <td data-bbox="236 1418 295 1472"> <p><b>&gt; Codekodierung Best Operator</b></p> <p>Attribut eines neuen Werts aktualisieren Wert position.</p> </td><td data-bbox="295 1418 386 1472"> <p><b>&gt; Codekodierung insertQuery</b></p> <p>Attribut eines neuen Werts aktualisieren Wert position.</p> </td></tr> <tr> <td data-bbox="236 1472 295 1525"> <p><b>&gt; Codekodierung Best Operator</b></p> <p>Attribut eines neuen Werts aktualisieren Wert position.</p> </td><td data-bbox="295 1472 386 1525"> <p><b>&gt; Codekodierung insertQuery</b></p> <p>Attribut eines neuen Werts aktualisieren Wert position.</p> </td></tr> <tr> <td data-bbox="236 1525 295 1579"> <p><b>&gt; Codekodierung Best Operator</b></p> <p>Attribut eines neuen Werts aktualisieren Wert position.</p> </td><td data-bbox="295 1525 386 1579"> <p><b>&gt; Codekodierung insertQuery</b></p> <p>Attribut eines neuen Werts aktualisieren Wert position.</p> </td></tr> <tr> <td data-bbox="236 1579 295 1617"></td><td data-bbox="295 1579 386 1617"></td></tr> </tbody> </table>	<b>1.2.3.1. Best Operator</b>		<p>Der Best Operator erkennt die einzelnen Werte eines Ausdrucks an denken.</p>	<p><b>&gt; Codekodierung Best Operator</b></p> <p>Die Best Operator ist verwandt wie die Wert eines Ausdrucks an denken.</p>	<p><b>&gt; Codekodierung insertQuery</b></p> <p>Attribut eines neuen Werts aktualisieren Wert position.</p>	<p>Best Operator kann die entsprechende Fähigkeit nicht, weil es nicht weiß wo es was im Dokument benötigt.</p>	<p><b>&gt; Codekodierung Best Operator</b></p> <p>Attribut eines neuen Werts aktualisieren Wert position.</p>	<p><b>&gt; Codekodierung insertQuery</b></p> <p>Attribut eines neuen Werts aktualisieren Wert position.</p>	<p><b>&gt; Codekodierung Best Operator</b></p> <p>Attribut eines neuen Werts aktualisieren Wert position.</p>	<p><b>&gt; Codekodierung insertQuery</b></p> <p>Attribut eines neuen Werts aktualisieren Wert position.</p>	<p><b>&gt; Codekodierung Best Operator</b></p> <p>Attribut eines neuen Werts aktualisieren Wert position.</p>	<p><b>&gt; Codekodierung insertQuery</b></p> <p>Attribut eines neuen Werts aktualisieren Wert position.</p>	<p><b>&gt; Codekodierung Best Operator</b></p> <p>Attribut eines neuen Werts aktualisieren Wert position.</p>	<p><b>&gt; Codekodierung insertQuery</b></p> <p>Attribut eines neuen Werts aktualisieren Wert position.</p>	<p><b>&gt; Codekodierung Best Operator</b></p> <p>Attribut eines neuen Werts aktualisieren Wert position.</p>	<p><b>&gt; Codekodierung insertQuery</b></p> <p>Attribut eines neuen Werts aktualisieren Wert position.</p>		
<b>1.2.3.1. Best Operator</b>																			
<p>Der Best Operator erkennt die einzelnen Werte eines Ausdrucks an denken.</p>	<p><b>&gt; Codekodierung Best Operator</b></p> <p>Die Best Operator ist verwandt wie die Wert eines Ausdrucks an denken.</p>																		
<p><b>&gt; Codekodierung insertQuery</b></p> <p>Attribut eines neuen Werts aktualisieren Wert position.</p>	<p>Best Operator kann die entsprechende Fähigkeit nicht, weil es nicht weiß wo es was im Dokument benötigt.</p>																		
<p><b>&gt; Codekodierung Best Operator</b></p> <p>Attribut eines neuen Werts aktualisieren Wert position.</p>	<p><b>&gt; Codekodierung insertQuery</b></p> <p>Attribut eines neuen Werts aktualisieren Wert position.</p>																		
<p><b>&gt; Codekodierung Best Operator</b></p> <p>Attribut eines neuen Werts aktualisieren Wert position.</p>	<p><b>&gt; Codekodierung insertQuery</b></p> <p>Attribut eines neuen Werts aktualisieren Wert position.</p>																		
<p><b>&gt; Codekodierung Best Operator</b></p> <p>Attribut eines neuen Werts aktualisieren Wert position.</p>	<p><b>&gt; Codekodierung insertQuery</b></p> <p>Attribut eines neuen Werts aktualisieren Wert position.</p>																		
<p><b>&gt; Codekodierung Best Operator</b></p> <p>Attribut eines neuen Werts aktualisieren Wert position.</p>	<p><b>&gt; Codekodierung insertQuery</b></p> <p>Attribut eines neuen Werts aktualisieren Wert position.</p>																		
<p><b>&gt; Codekodierung Best Operator</b></p> <p>Attribut eines neuen Werts aktualisieren Wert position.</p>	<p><b>&gt; Codekodierung insertQuery</b></p> <p>Attribut eines neuen Werts aktualisieren Wert position.</p>																		



**17.3.2 \$unset Operator**

**Syntax:** \$unset Operator  
Der \$unset Operator wird zur Entfernung von Attribut-Werten verwendet.

**Erklärung:** \$unset Operator  
Die Gedanken sind, dass die Attribute des Objekts entfernt werden.  
Hierfür darf nicht im Dokument selbst definiert werden.

**Codebeispiel: \$unset Operator**

```
// Update Operator: $unset
db.products.update({ _id: "5d9f1000-0000-4000-a000-000000000000" }, {
  $unset: {
    name: 1,
    "category": 1,
    rating: 1
  }
})
```

**17.3.3 \$rename Operator**

**Syntax:** \$rename Operator  
Der \$rename Operator wird um Namenswechsel von Attribut-Werten vorgenommen.

**Erklärung: \$rename Operator**

Das Benennungsobjekt enthält den alten Namen und den neuen Namen des Attributs in Form eines Objekts mit Mengenwert.

**Codebeispiel: \$rename Operator**

```
// Update Operator: $rename
db.products.update({ _id: "5d9f1000-0000-4000-a000-000000000000" }, {
  $rename: {
    "name": "title",
    "category": "type"
  }
})
```

**Abbildung 17.4: update Operatoren**

**17.4.1 \$update Operator**

**Werteoperatoren**

Mit Werteoperatoren werden neue Dokumente innerhalb eines Wertes erstellt.

**17.4.2 \$set Operator**

**Syntax:** \$set Operator  
Der \$set Operator wird verwendet um ein Feld auf ein Volumen eines spezifischen Wertes zu setzen.

**Erklärung:** \$set Operator  
Hierbei kann der Wert direkt übergeben werden.

**Codebeispiel: \$set Operator**

```
// Update Operator: $set
db.products.update({ _id: "5d9f1000-0000-4000-a000-000000000000" }, {
  $set: {
    "name": "My American Climinator",
    "category": "Sport",
    "rating": 5,
    "model": "1000",
    "price": 1000
  }
})
```

**17.4.3 \$updateOperator**

**Syntax:** \$updateOperator  
Der \$updateOperator wird für das Setzen von Attribut-Werten innerhalb eines Dokuments verwendet.

**Erklärung:** \$updateOperator  
Hierbei kann der Wert über einen Block definiert werden.

**Codebeispiel: \$updateOperator**

```
// Update Operator: $updateOperator
db.products.update({ _id: "5d9f1000-0000-4000-a000-000000000000" }, {
  $updateOperator: {
    "name": "My American Climinator"
  }
})
```

**17.4.4 \$updateMany Operator**

**Syntax:** \$updateMany Operator  
Mit dem \$updateMany Operator kann ein Dokument innerhalb einer bestimmten Gruppe geändert werden.

**Erklärung:** \$updateMany Operator  
Hierbei kann der Wert direkt übergeben werden.

**Codebeispiel: \$updateMany Operator**

```
// Update Many Operator: $updateMany
db.products.updateMany({ category: "Sport" }, {
  $set: {
    "name": "My American Climinator"
  }
})
```

**17.4.5 \$inc Operator**

**Syntax:** \$inc Operator  
Der \$inc Operator wird verwendet um eine Zählvariable innerhalb eines Dokuments zu erhöhen.

**Erklärung:** \$inc Operator  
Hierbei kann der Wert direkt übergeben werden.

**Codebeispiel: \$inc Operator**

```
// Update Operator: $inc
db.products.update({ _id: "5d9f1000-0000-4000-a000-000000000000" }, {
  $inc: {
    "rating": 1
  }
})
```

**17.4.6 \$pull Operator**

**Syntax:** \$pull Operator  
Der \$pull Operator wird verwendet um ein Attribut aus dem Dokument zu entfernen.

**Erklärung:** \$pull Operator  
Hierbei kann der Wert direkt übergeben werden.

**Codebeispiel: \$pull Operator**

```
// Update Operator: $pull
db.products.update({ _id: "5d9f1000-0000-4000-a000-000000000000" }, {
  $pull: {
    "category": "Sport"
  }
})
```

**17.4.7 \$push Operator**

**Syntax:** \$push Operator  
Der \$push Operator wird verwendet um ein Attribut in Form eines Objekts hinzuzufügen.

**Erklärung:** \$push Operator  
Hierbei kann der Wert direkt übergeben werden.

**Codebeispiel: \$push Operator**

```
// Update Operator: $push
db.products.update({ _id: "5d9f1000-0000-4000-a000-000000000000" }, {
  $push: {
    "category": "Sport"
  }
})
```

**17.4.8 \$pullAll Operator**

**Syntax:** \$pullAll Operator  
Der \$pullAll Operator wird verwendet um mehrere Attribut-Werte aus dem Dokument zu entfernen.

**Erklärung:** \$pullAll Operator  
Hierbei kann der Wert direkt übergeben werden.

**Codebeispiel: \$pullAll Operator**

```
// Update Operator: $pullAll
db.products.update({ _id: "5d9f1000-0000-4000-a000-000000000000" }, {
  $pullAll: {
    "category": "Sport"
  }
})
```

**17.4.9 \$inc Operator**

**Syntax:** \$inc Operator  
Der \$inc Operator wird verwendet um eine Zählvariable innerhalb eines Dokuments zu erhöhen.

**Erklärung:** \$inc Operator  
Hierbei kann der Wert direkt übergeben werden.

**Codebeispiel: \$inc Operator**

```
// Update Operator: $inc
db.products.update({ _id: "5d9f1000-0000-4000-a000-000000000000" }, {
  $inc: {
    "rating": 1
  }
})
```

**17.4.10 \$pull Operator**

**Syntax:** \$pull Operator  
Der \$pull Operator wird verwendet um ein Attribut aus dem Dokument zu entfernen.

**Erklärung:** \$pull Operator  
Hierbei kann der Wert direkt übergeben werden.

**Codebeispiel: \$pull Operator**

```
// Update Operator: $pull
db.products.update({ _id: "5d9f1000-0000-4000-a000-000000000000" }, {
  $pull: {
    "category": "Sport"
  }
})
```

**17.4.11 \$push Operator**

**Syntax:** \$push Operator  
Der \$push Operator wird verwendet um ein Attribut in Form eines Objekts hinzuzufügen.

**Erklärung:** \$push Operator  
Hierbei kann der Wert direkt übergeben werden.

**Codebeispiel: \$push Operator**

```
// Update Operator: $push
db.products.update({ _id: "5d9f1000-0000-4000-a000-000000000000" }, {
  $push: {
    "category": "Sport"
  }
})
```

**17.4.12 \$pullAll Operator**

**Syntax:** \$pullAll Operator  
Der \$pullAll Operator wird verwendet um mehrere Attribut-Werte aus dem Dokument zu entfernen.

**Erklärung:** \$pullAll Operator  
Hierbei kann der Wert direkt übergeben werden.

**Codebeispiel: \$pullAll Operator**

```
// Update Operator: $pullAll
db.products.update({ _id: "5d9f1000-0000-4000-a000-000000000000" }, {
  $pullAll: {
    "category": "Sport"
  }
})
```



© Dipl.-Ing. Max. Paul Pfeifferle-Diss.



**18. MongoDB - Aggregation von Daten**

## Aggregation von Daten

Informationssysteme

**18.1. Methoden und Verfahren**

Zur Abfrage geschichtlicher Kognosse, sollen Dokumente in der Lage sein komplexe Abfragen durchzuführen.

**Aggregationabfragen können immer nur eine Ebene, wobei Auswertungen über die Daten in einer Collection kaum, übergangsweise arbeiten.**

**18.1.1. Aggregationstypen**

Die Dokumentenreihenfolge steht als MapReduce Verifikation im Framework zur Verfügung.

- Affinität: Aggregationsthemen**
- Aggregationsthemen:** Aggregationsthemen erledigen das Aggregieren von Daten für die Dokumente einer Collection.
- Aggregationsthemen:** Das Aggregationsthemen ermöglicht die Verarbeitung von Daten aus mehreren Collections.
- Map Reduce Algorithmus:** Der Map Reduce Algorithmus ist ein Verfahren zur horizontalen Verarbeitung großer Datenmengen.

**Erläuterung: Aggregationsthemen**

- Affinitätsthemen:** Abfrageanfragen erhalten individuelle Ergebnisse für jedes Dokument einer einzelnen Collection. Abfrageanfragen sind auf die Funktionseinheit einer Collection beschränkt.
- Aggregationsthemen:** Das Aggregationsthemen erledigt die Verarbeitung des Dokuments in mehreren Collections.
- Map Reduce Verfahren:** Der Map Reduce Algorithmus ist ein Verfahren zur horizontalen Verarbeitung großer Datenmengen. Das Verfahren wird bei einer Implementierung jedoch eine hohe Komplexität aufweisen.

© Hartung - Max-Planck-Forschungsinst.

**18.2. Abfragemethoden**

Abfragemethoden ermöglichen die Aggregation von Daten für die Dokumente einer Collection.

**18.2.1. Abfragemethode: count()**

Mit Hilfe der count() Methode wird die Anzahl der Dokumente eines Resultats ermittelt.

**18.2.2. Abfragemethode: distinct()**

Mit der distinct() Methode können die unterschiedlichen Ausprägungen einer spezifischen Variable eines Resultats ermittelt werden.

**18.2.3. Abfragemethode: group()**

Die group() Methode gruppiert die Daten einer Collection schrittweise Schrittweise. Für die gruppierten Untergruppen werden einfache Operatoren wie das Zählen oder das Mittelwerte berechnet.

**18.2.4. Abfragemethode: match()**

Die count() Methode ist die Anzahl der Dokumente eines Resultats ermitteln welche bestimmten Bedingungen erfüllen.

**18.2.5. Abfragemethode: sort()**

Die sort() Methode sortiert die Dokumente nach einer oder mehreren Variablen in einem bestimmten Reihenfolge.

**18.2.6. Abfragemethode: limit()**

Die limit() Methode bestimmt die Anzahl der Dokumente eines Resultats welche in einer bestimmten Reihenfolge angezeigt werden.

**18.2.7. Abfragemethode: skip()**

Die skip() Methode bestimmt die Anzahl der Dokumente eines Resultats welche nicht angezeigt werden.

**18.2.8. Abfragemethode: distinct()**

Mit der distinct() Methode können die unterschiedlichen Ausprägungen einer spezifischen Variable eines Resultats ermittelt werden.

**18.2.9. Abfragemethode: group()**

Die group() Methode erlaubt die Gruppierung der Daten einer Collection nach einer oder mehreren Variablen. Für die gruppierten Untergruppen können einfache Operatoren wie das Zählen bzw. das Mittelwerte der Daten berechnet werden.

**18.2.10. Abfragemethode: match()**

Die count() Methode ist die Anzahl der Dokumente eines Resultats ermitteln welche bestimmten Bedingungen erfüllen.

**18.2.11. Abfragemethode: sort()**

Die sort() Methode sortiert die Dokumente nach einer oder mehreren Variablen in einem bestimmten Reihenfolge.

**18.2.12. Abfragemethode: limit()**

Die limit() Methode bestimmt die Anzahl der Dokumente eines Resultats welche in einer bestimmten Reihenfolge angezeigt werden.

**18.2.13. Abfragemethode: skip()**

Die skip() Methode bestimmt die Anzahl der Dokumente eines Resultats welche nicht angezeigt werden.

© Hartung - Max-Planck-Forschungsinst.

**18.3. Aggregationsframework**

Um die Dokumente einer Collection zu bearbeiten, kann die Dokumente in einer hierarchischen Struktur abrufen.

**18.3.1. Aggregationsschritte**

**Datenstruktur Pipeline:**

Eine Pipeline ist eine Abfolge von Operationen, zur Verarbeitung der Dokumente einer Collection. Eine Operation wird dabei durch einen Block definiert.

**Die MongoDB-Schleife** gibt keine Restrukturierung der Dokumente vor, sondern es wird lediglich die neue Struktur ab einer Stelle einer Pipeline definiert.

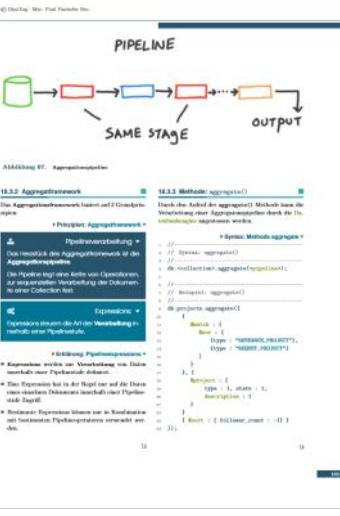
**Bei einer Pipeline werden die Dokumente einer Collection von einer Stelle aus nichts mehr geändert und verzweigt die Ausgabe einer Pipeline nicht in verschiedene Pfade.**

**Ein Dokument einer Collection kann gruppierend sein an die weitere Pipelineabfolge entsprechend der in einer Pipelineabfolge liegenden Dokumente angepasst werden.**

**Initialisiert diese Schleife mit der Struktur eines Dokuments**

© Hartung - Max-Planck-Forschungsinst.

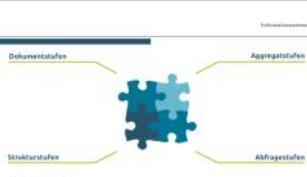












### Ablösung 03. Schleiftypen

Basis-Praktiken - Projekt		System-Praktiken - Projekt
Die Projekt-Operatoren werden auf die Datei oder Dokumente des Unternehmens oder auf bestimmte Teile davon beschränkt.		
Der Spezifiker Operatoren ist die Komposition aus mehreren Basiskomponenten.		
- Entwurf - Projekt Operatoren		
Die Projekt-Operatoren sind spezifische für die Datei eines Dokuments oder einer Unternehmens- oder Metadaten zu beschreiben.		
- Umsetzung - Projekt Operatoren		
Die Projekt-Operatoren sind mehrere Dokumente an die Objekte Projektoren entgegen, die Teil der Dokumente Projekt sind.		
- System-Praktiken Operatoren		
Das Spezifiker Operatoren ist die Objekt des Projekt Operatoren.		
Ma - Ass (Obj) wird bestimmt welche Bilder diese Dokumente in der Form der Projektoren (Sie können verschiedene Formate haben).		
Gelehrte können immer mit Dokumenten interagieren werden.		
Projektoren Operatoren und die Werte die Bilder die Empfehlungen.		











Operator	Beschreibung	Referenz
<b>Sub</b>	Der Sub Operator wird zur Berechnung des Betrags eines numerischen Wertes verwendet.	191
<b>Sum, Summation</b>	Der Sum Operator wird verwendet um die Summe aller oder mehrerer Zahlen zu berechnen.	190
<b>Substitution, Spalte</b>	Das Substitution bzw. Spalte Operator verändert die geprägte Objektdatenstruktur in ein Objekt.	190
<b>Sort, Sort, Sort</b>	Die Sort Operator werden verwendet um die Ergebnisse nach einem bestimmten Kriterium zu sortieren.	190
<b>Konkatenation</b>	Der Konkatenation Operator kombiniert das Anfangsereignis mit allen bestimmen Indizes des Endes.	194
<b>Sum</b>	Mit dem Sum Operator können 2 oder mehrere Werte zusammengefasst werden. Das Ergebnis des Vergleichs ist ein Differenz.	194
<b>Scorezähler</b>	Mit dem Scorezähler Operator können Elemente aus unterschiedlichen Anordnungen in ein Objekt übertragen werden.	194
<b>Summ</b>	Der Summ Operator verändert die Konkatenations Methode.	194
<b>Entfernen</b>	Der Entfernen Operator entfernt die Elemente aus einer Liste.	194
<b>Faktur</b>	Der Faktur Operator kann verwendet um die Kosten eines Kunden zu berechnen.	194
<b>Summ</b>	Vergleichsoperatoren und wendbare logische Operatoren. Vergleichsoperatoren sind nur von den Daten im mathematischen Bereich gültig.	191
<b>Min</b>	Mit dem Min Operator können die Elemente eines Array nach aufsteigenden Kreuzen abgetragen werden.	191
<b>Max, Max, Max</b>	Vergleichsoperatoren und wendbare logische Operatoren. Vergleichsoperatoren wie Min, Max, Min und Max sind nur von den Daten im mathematischen Bereich gültig.	191
<b>Sum</b>	Der Sum Operator wird verwendet um die Summe eines Arrays zu erstellen.	190
<b>Summ, Sum, Summ</b>	Mit dem Summ Operator wird ein Array als einem bestimmten Artifiz der höchste Wert innerhalb einer Liste auf die entsprechende Zahl erhöht.	190
<b>Endgröße</b>	Die Endgröße Operator verwendet die End-Position der zwei nächsten Zeichen für berechnen.	191
<b>Indexize</b>	Mit dem Indexize Operator kann die Elemente eines Arrays unter einem einzelnen Wert innerhalb einer Liste abgerufen werden.	191
<b>Stoßschub</b>	Der Stoßschub Operator verändert die kontinuierliche Verarbeitung von Daten in Dokumenten.	190

Operator	Beschreibung	Reihenfolge
<b>Arith.</b>	Der Arith. Operator wird zur Berechnung des Inhalts eines numerischen Wertes benutzt.	181
<b>Summe / Subtrahieren</b>	Der Summe Operator wird verwendet um die Summe Der oder mehrerer Zahlen zu berechnen.	180
<b>Multiplik.</b>	Der Multiplik. Operator wird verwendet um das Produkt Der oder mehrerer Zahlen zu berechnen.	180



Operator	Beschreibung	Inhaltslink
Int, Inv.	Vergleichsoperatoren und zweistellige logische Operatoren. Vergleichsoperatoren werden von links nach rechts priorisiert.	<a href="#">101</a>
Rgt, Rgt, Rgt	Wiederholungen und Ausdrücke logischer Operatoren. Vergleichsoperatoren werden von rechts in mathematischen Gleichungen verwendet.	<a href="#">102</a>
<hr/>		
Ablösung 75. Expressive Vergleichsoperatoren		
<b>18.8.2 Schilte Operator</b>	 Die Schilte Operatoren sind nur für Beschreibungen der Objekte im Absatz verwendbar.  → <b>Syntax: Schilte Operator</b>	<a href="#">103</a>
<ul style="list-style-type: none"> <li>- H: Spezif. Schilte Operator</li> <li>- B: Spezif. Schilte Operator</li> <li>- M: Muster - I: Ausprägung, - A: Ausprägung, - S: Ausprägung</li> </ul>		
<b>18.8.3 Zahl Operator</b>	 Die Zahl Operatoren sind nur für Beschreibungen der Objekte im Absatz verwendbar.  → <b>Syntax: Zahl Operator</b>	<a href="#">104</a>
<ul style="list-style-type: none"> <li>- H: Spezif. Zahl Operator</li> <li>- B: Spezif. Zahl Operator</li> <li>- M: Muster - I: Ausprägung, - A: Ausprägung</li> <li>- D: Ausprägung, - P: Ausprägung</li> <li>- S: Spezif. Zahl Operator</li> <li>- T: Ausprägung, - L: Ausprägung</li> <li>- Z: Zahl - I: Ausprägung</li> <li>- J: Zahl - I: Ausprägung</li> <li>- K: Zahl - I: Ausprägung</li> <li>- L: Zahl - I: Ausprägung</li> <li>- R: Zahl - I: Ausprägung</li> <li>- D1: Ausprägung</li> </ul>		
<b>18.10. Vergleichsoperatoren</b>	 Vergleichsoperatoren und logische Operatoren. Vergleichsoperatoren werden von links in mathematischen Gleichungen verwendet.	<a href="#">105</a>
<b>18.10.1. Vergleichsoperatoren</b>	 Die linke Art und die rechte logische Operatoren.	<a href="#">106</a>
	<ul style="list-style-type: none"> <li>- H: Spezif. Art, B1: Spezif. Art, B2: Spezif. Art</li> <li>- B: Spezif. Art</li> <li>- M: Muster - I: Ausprägung, - A: Ausprägung</li> <li>- D: Ausprägung</li> <li>- S: Spezif. Vergleichsoperator</li> <li>- T: Ausprägung</li> <li>- Z: Zahl - I: Ausprägung</li> <li>- J: Zahl - I: Ausprägung</li> <li>- K: Zahl - I: Ausprägung</li> <li>- L: Zahl - I: Ausprägung</li> <li>- R: Zahl - I: Ausprägung</li> <li>- D1: Ausprägung</li> </ul>	

Operator	Beschreibung	Reihenfolge
Scrap	Mit der Hilfe des Scarp Operator können 2 Werte miteinander verglichen werden. Das Ergebnis des Vergleichs ist ein Zahlenwert.	164
Scrol	Der Scrol Operator ermöglicht die kontinuierliche Verarbeitung von Daten in Dokumenten.	165
Switch	Der Switch Operator ermöglicht die kontrollierte Verarbeitung von Daten in Dokumenten.	165



Informationsquellen	
<b>Operator</b>	<b>Beschreibung</b>
Associativity	Mit dem Operator kann kleinere Elemente zu unterschiedlichen Anzahlen in ein Element zusammengefasst werden.
Sin	Mittelt die Sinusfunktion gegen über und rechnet den Winkel in Bogenmaß um.
Sinop	Die Sinusfunktion und verwendet sie im Domäne eines Arrays zu wenden.
Sinop	Unter dem Operator können die Elemente eines Arrays nach bestimmten Kriterien gefiltert werden.
Abbildung 77: Implementierungsprinzipien	
<b>18.12.3 Sump Operator</b>	<b>18.13. Arrayexpressions</b>
Mit der Sump-der-Sump Operatoren können 2 oder mehrere Arrayelemente addiert werden. Die Ergebnisse des Ver gleichs sind im Zusammenhang.	<b>U</b> <b>Arayexpressions</b> → Anwendung einer Art Wiederholung eines Arrays verwendet.
• Ist der erste Wert größer als der zweite ist das Ergebnis des Vergleichs 1.	<b>18.13.1 Summatorisch Operator</b>
• Ist der zweite Wert kleiner als der Ergebnis des Vergleichs 0.	Die Summatorischen Operatoren verhindern das Ausführen von mehreren Schleifen, wenn es sich um dasselbe Array handelt.
• Sind die Werte gleich ist der Ergebnis 0.	<b>System-Summarisch</b> →
+ Erweitertes Sump Expression =	<ul style="list-style-type: none"> <li>• System-Summarisch:</li> <li>    + Zeichen: +, -</li> <li>    + Brackets: ( )</li> <li>    + Identifier: <i>Variable</i></li> <li>    + Array: [ ]</li> <li>    + Wenn die Art, füher die gewünschten Elemente ist, dann die Ergebnis-Werte wird mit dem entsprechenden Elementen dieses Arrays multipliziert.</li> <li>    + Beispiele:           <ul style="list-style-type: none"> <li>- <i>Summatorisch</i>: <i>I</i></li> <li>- <i>Multiplikation</i>, <i>Division</i>, <i>Modulo</i></li> </ul> </li> </ul>
<pre>1 // Sump-Operator 2 // Beispiel: sum 3 // Summatorisch 4 // Ausgabe: 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10 5 // = 55 6 //  7 //  8 // Sump: Sump 9 //  10 //  11 //  12 //  13 //  14 //  15 //  16 //  17 //  18 //  19 //  20 //  21 //  22 //  23 //  24 // </pre>	<b>18.13.2 Arrayoperator</b>
<pre>1 // Sump-Operator 2 // Beispiel: sum 3 // Summatorisch 4 // Ausgabe: 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10 5 // = 55 6 //  7 //  8 // Sump: Sump 9 //  10 //  11 //  12 //  13 //  14 //  15 //  16 //  17 //  18 //  19 //  20 //  21 //  22 //  23 //  24 // </pre>	<b>18.13.2 Sump Operator</b>
Mit dem Summatorischen Operator können Elemente verschiedener Arrays in die Rechnung eingeschlossen werden.	<b>18.13.3 Sump Operator</b>
• System-Summarisch =	<b>U</b> <b>Sump-Operator</b> → Mit dem Sump-Operator kann gleichzeitig mehrere in einem Array verwendete.
<pre>1 // Sump: Sumpoperator 2 //  3 //  4 //  5 //  6 //  7 //  8 //  9 //  10 //  11 //  12 //  13 //  14 //  15 //  16 //  17 //  18 //  19 //  20 //  21 //  22 //  23 //  24 // </pre>	<b>System-Summarisch</b> →
<b>18.13.3 Summatorische Operatoren</b>	<b>18.13.3 Sump Operator</b>
Mit den Summatorischen Operatoren können Elemente verschiedener Arrays in die Rechnung eingeschlossen werden.	<b>U</b> <b>Sump-Operator</b> → Mit dem Sump-Operator kann gleichzeitig mehrere in einem Array verwendete.
• System-Summarisch =	<b>System-Summarisch</b> →
<pre>1 // Sump: Sumpoperator 2 //  3 //  4 //  5 //  6 //  7 //  8 //  9 //  10 //  11 //  12 //  13 //  14 //  15 //  16 //  17 //  18 //  19 //  20 //  21 //  22 //  23 //  24 // </pre>	<b>18.13.3 Sump Operator</b>
<b>18.14. Sump Operator</b>	<b>18.14. Sump Operator</b>
Der Sump Operator und vereinfacht die Elemente eines Arrays zu einzelnen Werten.	<b>U</b> <b>Sump-Operator</b> →
Vordefiniert für den Sump Operator war Beispiel mit den Schleifen.	<b>18.14.1 Sump Operator</b>
<pre>1 // Sump: Sump Operator 2 //  3 //  4 //  5 //  6 //  7 //  8 //  9 //  10 //  11 //  12 //  13 //  14 //  15 //  16 //  17 //  18 //  19 //  20 //  21 //  22 //  23 //  24 // </pre>	<b>18.14.1 Sump Operator</b>
<b>18.14.2 Sump Operator</b>	<b>18.14.2 Sump Operator</b>
Der Sump Operator und vereinfacht die Elemente eines Arrays zu einzelnen Werten.	<b>U</b> <b>Sump-Operator</b> →
Vordefiniert für den Sump Operator war Beispiel mit den Schleifen.	<b>18.14.2 Sump Operator</b>
<pre>1 // Sump: Sump Operator 2 //  3 //  4 //  5 //  6 //  7 //  8 //  9 //  10 //  11 //  12 //  13 //  14 //  15 //  16 //  17 //  18 //  19 //  20 //  21 //  22 //  23 //  24 // </pre>	<b>18.14.2 Sump Operator</b>
<b>18.14.3 Sump Operator</b>	<b>18.14.3 Sump Operator</b>
Der Sump Operator und vereinfacht die Elemente eines Arrays zu einzelnen Werten.	<b>U</b> <b>Sump-Operator</b> →
Vordefiniert für den Sump Operator war Beispiel mit den Schleifen.	<b>18.14.3 Sump Operator</b>
<pre>1 // Sump: Sump Operator 2 //  3 //  4 //  5 //  6 //  7 //  8 //  9 //  10 //  11 //  12 //  13 //  14 //  15 //  16 //  17 //  18 //  19 //  20 //  21 //  22 //  23 //  24 // </pre>	<b>18.14.3 Sump Operator</b>
<b>18.15. Bitwise Operator</b>	<b>18.15. Bitwise Operator</b>
Mit den Bitwise Operatoren können die Elemente eines Arrays mit bestimmten Kriterien gefiltert werden.	<b>U</b> <b>Bitwise Operator</b> → Mit den Bitwise Operatoren können die Elemente eines Arrays nach bestimmten Kriterien gefiltert werden.
Vordefiniert für den Bitwise Operator war Beispiel mit den Schleifen.	<b>System-Bitwise</b> →
<pre>1 // Bitwise: Bitwise Operator 2 //  3 //  4 //  5 //  6 //  7 //  8 //  9 //  10 //  11 //  12 //  13 //  14 //  15 //  16 //  17 //  18 //  19 //  20 //  21 //  22 //  23 //  24 // </pre>	<b>18.15.1 Bitwise Operator</b>
<b>18.15.1 Bitwise Operator</b>	<b>18.15.1 Bitwise Operator</b>
Mit den Bitwise Operatoren können die Elemente eines Arrays nach bestimmten Kriterien gefiltert werden.	<b>U</b> <b>Bitwise Operator</b> →
Vordefiniert für den Bitwise Operator war Beispiel mit den Schleifen.	<b>System-Bitwise</b> →
<pre>1 // Bitwise: Bitwise Operator 2 //  3 //  4 //  5 //  6 //  7 //  8 //  9 //  10 //  11 //  12 //  13 //  14 //  15 //  16 //  17 //  18 //  19 //  20 //  21 //  22 //  23 //  24 // </pre>	<b>18.15.1 Bitwise Operator</b>
<b>18.15.2 Bitwise Operator</b>	<b>18.15.2 Bitwise Operator</b>
Mit den Bitwise Operatoren können die Elemente eines Arrays nach bestimmten Kriterien gefiltert werden.	<b>U</b> <b>Bitwise Operator</b> →
Vordefiniert für den Bitwise Operator war Beispiel mit den Schleifen.	<b>System-Bitwise</b> →
<pre>1 // Bitwise: Bitwise Operator 2 //  3 //  4 //  5 //  6 //  7 //  8 //  9 //  10 //  11 //  12 //  13 //  14 //  15 //  16 //  17 //  18 //  19 //  20 //  21 //  22 //  23 //  24 // </pre>	<b>18.15.2 Bitwise Operator</b>
<b>18.15.3 Bitwise Operator</b>	<b>18.15.3 Bitwise Operator</b>
Mit den Bitwise Operatoren können die Elemente eines Arrays nach bestimmten Kriterien gefiltert werden.	<b>U</b> <b>Bitwise Operator</b> →
Vordefiniert für den Bitwise Operator war Beispiel mit den Schleifen.	<b>System-Bitwise</b> →
<pre>1 // Bitwise: Bitwise Operator 2 //  3 //  4 //  5 //  6 //  7 //  8 //  9 //  10 //  11 //  12 //  13 //  14 //  15 //  16 //  17 //  18 //  19 //  20 //  21 //  22 //  23 //  24 // </pre>	<b>18.15.3 Bitwise Operator</b>



16.13.6 Bereich Operator	
Mit dem Bereich Operator kann die Elemente eines Arrays oder eines HashSet-Wortes verarbeitet werden.	= <i>f1</i> = <i>f2</i> = <i>f3</i> = <i>f4</i> = <i>f5</i> = <i>f6</i> = <i>f7</i> = <i>f8</i> = <i>f9</i> = <i>f10</i> = <i>f11</i> = <i>f12</i> = <i>f13</i> = <i>f14</i> = <i>f15</i> = <i>f16</i> = <i>f17</i> = <i>f18</i> = <i>f19</i> = <i>f20</i> = <i>f21</i> = <i>f22</i> = <i>f23</i> = <i>f24</i> = <i>f25</i> = <i>f26</i> = <i>f27</i> = <i>f28</i> = <i>f29</i> = <i>f30</i> = <i>f31</i> = <i>f32</i> = <i>f33</i> = <i>f34</i> = <i>f35</i> = <i>f36</i> = <i>f37</i> = <i>f38</i> = <i>f39</i> = <i>f40</i> = <i>f41</i> = <i>f42</i> = <i>f43</i> = <i>f44</i> = <i>f45</i> = <i>f46</i> = <i>f47</i> = <i>f48</i> = <i>f49</i> = <i>f50</i> = <i>f51</i> = <i>f52</i> = <i>f53</i> = <i>f54</i> = <i>f55</i> = <i>f56</i> = <i>f57</i> = <i>f58</i> = <i>f59</i> = <i>f60</i> = <i>f61</i> = <i>f62</i> = <i>f63</i> = <i>f64</i> = <i>f65</i> = <i>f66</i> = <i>f67</i> = <i>f68</i> = <i>f69</i> = <i>f70</i> = <i>f71</i> = <i>f72</i> = <i>f73</i> = <i>f74</i> = <i>f75</i> = <i>f76</i> = <i>f77</i> = <i>f78</i> = <i>f79</i> = <i>f80</i> = <i>f81</i> = <i>f82</i> = <i>f83</i> = <i>f84</i> = <i>f85</i> = <i>f86</i> = <i>f87</i> = <i>f88</i> = <i>f89</i> = <i>f90</i> = <i>f91</i> = <i>f92</i> = <i>f93</i> = <i>f94</i> = <i>f95</i> = <i>f96</i> = <i>f97</i> = <i>f98</i> = <i>f99</i> = <i>f100</i> = <i>f101</i> = <i>f102</i> = <i>f103</i> = <i>f104</i> = <i>f105</i> = <i>f106</i> = <i>f107</i> = <i>f108</i> = <i>f109</i> = <i>f110</i> = <i>f111</i> = <i>f112</i> = <i>f113</i> = <i>f114</i> = <i>f115</i> = <i>f116</i> = <i>f117</i> = <i>f118</i> = <i>f119</i> = <i>f120</i> = <i>f121</i> = <i>f122</i> = <i>f123</i> = <i>f124</i> = <i>f125</i> = <i>f126</i> = <i>f127</i> = <i>f128</i> = <i>f129</i> = <i>f130</i> = <i>f131</i> = <i>f132</i> = <i>f133</i> = <i>f134</i> = <i>f135</i> = <i>f136</i> = <i>f137</i> = <i>f138</i> = <i>f139</i> = <i>f140</i> = <i>f141</i> = <i>f142</i> = <i>f143</i> = <i>f144</i> = <i>f145</i> = <i>f146</i> = <i>f147</i> = <i>f148</i> = <i>f149</i> = <i>f150</i> = <i>f151</i> = <i>f152</i> = <i>f153</i> = <i>f154</i> = <i>f155</i> = <i>f156</i> = <i>f157</i> = <i>f158</i> = <i>f159</i> = <i>f160</i> = <i>f161</i> = <i>f162</i> = <i>f163</i> = <i>f164</i> = <i>f165</i> = <i>f166</i> = <i>f167</i> = <i>f168</i> = <i>f169</i> = <i>f170</i> = <i>f171</i> = <i>f172</i> = <i>f173</i> = <i>f174</i> = <i>f175</i> = <i>f176</i> = <i>f177</i> = <i>f178</i> = <i>f179</i> = <i>f180</i> = <i>f181</i> = <i>f182</i> = <i>f183</i> = <i>f184</i> = <i>f185</i> = <i>f186</i> = <i>f187</i> = <i>f188</i> = <i>f189</i> = <i>f190</i> = <i>f191</i> = <i>f192</i> = <i>f193</i> = <i>f194</i> = <i>f195</i> = <i>f196</i> = <i>f197</i> = <i>f198</i> = <i>f199</i> = <i>f200</i> = <i>f201</i> = <i>f202</i> = <i>f203</i> = <i>f204</i> = <i>f205</i> = <i>f206</i> = <i>f207</i> = <i>f208</i> = <i>f209</i> = <i>f210</i> = <i>f211</i> = <i>f212</i> = <i>f213</i> = <i>f214</i> = <i>f215</i> = <i>f216</i> = <i>f217</i> = <i>f218</i> = <i>f219</i> = <i>f220</i> = <i>f221</i> = <i>f222</i> = <i>f223</i> = <i>f224</i> = <i>f225</i> = <i>f226</i> = <i>f227</i> = <i>f228</i> = <i>f229</i> = <i>f230</i> = <i>f231</i> = <i>f232</i> = <i>f233</i> = <i>f234</i> = <i>f235</i> = <i>f236</i> = <i>f237</i> = <i>f238</i> = <i>f239</i> = <i>f240</i> = <i>f241</i> = <i>f242</i> = <i>f243</i> = <i>f244</i> = <i>f245</i> = <i>f246</i> = <i>f247</i> = <i>f248</i> = <i>f249</i> = <i>f250</i> = <i>f251</i> = <i>f252</i> = <i>f253</i> = <i>f254</i> = <i>f255</i> = <i>f256</i> = <i>f257</i> = <i>f258</i> = <i>f259</i> = <i>f260</i> = <i>f261</i> = <i>f262</i> = <i>f263</i> = <i>f264</i> = <i>f265</i> = <i>f266</i> = <i>f267</i> = <i>f268</i> = <i>f269</i> = <i>f270</i> = <i>f271</i> = <i>f272</i> = <i>f273</i> = <i>f274</i> = <i>f275</i> = <i>f276</i> = <i>f277</i> = <i>f278</i> = <i>f279</i> = <i>f280</i> = <i>f281</i> = <i>f282</i> = <i>f283</i> = <i>f284</i> = <i>f285</i> = <i>f286</i> = <i>f287</i> = <i>f288</i> = <i>f289</i> = <i>f290</i> = <i>f291</i> = <i>f292</i> = <i>f293</i> = <i>f294</i> = <i>f295</i> = <i>f296</i> = <i>f297</i> = <i>f298</i> = <i>f299</i> = <i>f300</i> = <i>f301</i> = <i>f302</i> = <i>f303</i> = <i>f304</i> = <i>f305</i> = <i>f306</i> = <i>f307</i> = <i>f308</i> = <i>f309</i> = <i>f310</i> = <i>f311</i> = <i>f312</i> = <i>f313</i> = <i>f314</i> = <i>f315</i> = <i>f316</i> = <i>f317</i> = <i>f318</i> = <i>f319</i> = <i>f320</i> = <i>f321</i> = <i>f322</i> = <i>f323</i> = <i>f324</i> = <i>f325</i> = <i>f326</i> = <i>f327</i> = <i>f328</i> = <i>f329</i> = <i>f330</i> = <i>f331</i> = <i>f332</i> = <i>f333</i> = <i>f334</i> = <i>f335</i> = <i>f336</i> = <i>f337</i> = <i>f338</i> = <i>f339</i> = <i>f340</i> = <i>f341</i> = <i>f342</i> = <i>f343</i> = <i>f344</i> = <i>f345</i> = <i>f346</i> = <i>f347</i> = <i>f348</i> = <i>f349</i> = <i>f350</i> = <i>f351</i> = <i>f352</i> = <i>f353</i> = <i>f354</i> = <i>f355</i> = <i>f356</i> = <i>f357</i> = <i>f358</i> = <i>f359</i> = <i>f360</i> = <i>f361</i> = <i>f362</i> = <i>f363</i> = <i>f364</i> = <i>f365</i> = <i>f366</i> = <i>f367</i> = <i>f368</i> = <i>f369</i> = <i>f370</i> = <i>f371</i> = <i>f372</i> = <i>f373</i> = <i>f374</i> = <i>f375</i> = <i>f376</i> = <i>f377</i> = <i>f378</i> = <i>f379</i> = <i>f380</i> = <i>f381</i> = <i>f382</i> = <i>f383</i> = <i>f384</i> = <i>f385</i> = <i>f386</i> = <i>f387</i> = <i>f388</i> = <i>f389</i> = <i>f390</i> = <i>f391</i> = <i>f392</i> = <i>f393</i> = <i>f394</i> = <i>f395</i> = <i>f396</i> = <i>f397</i> = <i>f398</i> = <i>f399</i> = <i>f400</i> = <i>f401</i> = <i>f402</i> = <i>f403</i> = <i>f404</i> = <i>f405</i> = <i>f406</i> = <i>f407</i> = <i>f408</i> = <i>f409</i> = <i>f410</i> = <i>f411</i> = <i>f412</i> = <i>f413</i> = <i>f414</i> = <i>f415</i> = <i>f416</i> = <i>f417</i> = <i>f418</i> = <i>f419</i> = <i>f420</i> = <i>f421</i> = <i>f422</i> = <i>f423</i> = <i>f424</i> = <i>f425</i> = <i>f426</i> = <i>f427</i> = <i>f428</i> = <i>f429</i> = <i>f430</i> = <i>f431</i> = <i>f432</i> = <i>f433</i> = <i>f434</i> = <i>f435</i> = <i>f436</i> = <i>f437</i> = <i>f438</i> = <i>f439</i> = <i>f440</i> = <i>f441</i> = <i>f442</i> = <i>f443</i> = <i>f444</i> = <i>f445</i> = <i>f446</i> = <i>f447</i> = <i>f448</i> = <i>f449</i> = <i>f450</i> = <i>f451</i> = <i>f452</i> = <i>f453</i> = <i>f454</i> = <i>f455</i> = <i>f456</i> = <i>f457</i> = <i>f458</i> = <i>f459</i> = <i>f460</i> = <i>f461</i> = <i>f462</i> = <i>f463</i> = <i>f464</i> = <i>f465</i> = <i>f466</i> = <i>f467</i> = <i>f468</i> = <i>f469</i> = <i>f470</i> = <i>f471</i> = <i>f472</i> = <i>f473</i> = <i>f474</i> = <i>f475</i> = <i>f476</i> = <i>f477</i> = <i>f478</i> = <i>f479</i> = <i>f480</i> = <i>f481</i> = <i>f482</i> = <i>f483</i> = <i>f484</i> = <i>f485</i> = <i>f486</i> = <i>f487</i> = <i>f488</i> = <i>f489</i> = <i>f490</i> = <i>f491</i> = <i>f492</i> = <i>f493</i> = <i>f494</i> = <i>f495</i> = <i>f496</i> = <i>f497</i> = <i>f498</i> = <i>f499</i> = <i>f500</i> = <i>f501</i> = <i>f502</i> = <i>f503</i> = <i>f504</i> = <i>f505</i> = <i>f506</i> = <i>f507</i> = <i>f508</i> = <i>f509</i> = <i>f510</i> = <i>f511</i> = <i>f512</i> = <i>f513</i> = <i>f514</i> = <i>f515</i> = <i>f516</i> = <i>f517</i> = <i>f518</i> = <i>f519</i> = <i>f520</i> = <i>f521</i> = <i>f522</i> = <i>f523</i> = <i>f524</i> = <i>f525</i> = <i>f526</i> = <i>f527</i> = <i>f528</i> = <i>f529</i> = <i>f530</i> = <i>f531</i> = <i>f532</i> = <i>f533</i> = <i>f534</i> = <i>f535</i> = <i>f536</i> = <i>f537</i> = <i>f538</i> = <i>f539</i> = <i>f540</i> = <i>f541</i> = <i>f542</i> = <i>f543</i> = <i>f544</i> = <i>f545</i> = <i>f546</i> = <i>f547</i> = <i>f548</i> = <i>f549</i> = <i>f550</i> = <i>f551</i> = <i>f552</i> = <i>f553</i> = <i>f554</i> = <i>f555</i> = <i>f556</i> = <i>f557</i> = <i>f558</i> = <i>f559</i> = <i>f560</i> = <i>f561</i> = <i>f562</i> = <i>f563</i> = <i>f564</i> = <i>f565</i> = <i>f566</i> = <i>f567</i> = <i>f568</i> = <i>f569</i> = <i>f570</i> = <i>f571</i> = <i>f572</i> = <i>f573</i> = <i>f574</i> = <i>f575</i> = <i>f576</i> = <i>f577</i> = <i>f578</i> = <i>f579</i> = <i>f580</i> = <i>f581</i> = <i>f582</i> = <i>f583</i> = <i>f584</i> = <i>f585</i> = <i>f586</i> = <i>f587</i> = <i>f588</i> = <i>f589</i> = <i>f590</i> = <i>f591</i> = <i>f592</i> = <i>f593</i> = <i>f594</i> = <i>f595</i> = <i>f596</i> = <i>f597</i> = <i>f598</i> = <i>f599</i> = <i>f600</i> = <i>f601</i> = <i>f602</i> = <i>f603</i> = <i>f604</i> = <i>f605</i> = <i>f606</i> = <i>f607</i> = <i>f608</i> = <i>f609</i> = <i>f610</i> = <i>f611</i> = <i>f612</i> = <i>f613</i> = <i>f614</i> = <i>f615</i> = <i>f616</i> = <i>f617</i> = <i>f618</i> = <i>f619</i> = <i>f620</i> = <i>f621</i> = <i>f622</i> = <i>f623</i> = <i>f624</i> = <i>f625</i> = <i>f626</i> = <i>f627</i> = <i>f628</i> = <i>f629</i> = <i>f630</i> = <i>f631</i> = <i>f632</i> = <i>f633</i> = <i>f634</i> = <i>f635</i> = <i>f636</i> = <i>f637</i> = <i>f638</i> = <i>f639</i> = <i>f640</i> = <i>f641</i> = <i>f642</i> = <i>f643</i> = <i>f644</i> = <i>f645</i> = <i>f646</i> = <i>f647</i> = <i>f648</i> = <i>f649</i> = <i>f650</i> = <i>f651</i> = <i>f652</i> = <i>f653</i> = <i>f654</i> = <i>f655</i> = <i>f656</i> = <i>f657</i> = <i>f658</i> = <i>f659</i> = <i>f660</i> = <i>f661</i> = <i>f662</i> = <i>f663</i> = <i>f664</i> = <i>f665</i> = <i>f666</i> = <i>f667</i> = <i>f668</i> = <i>f669</i> = <i>f670</i> = <i>f671</i> = <i>f672</i> = <i>f673</i> = <i>f674</i> = <i>f675</i> = <i>f676</i> = <i>f677</i> = <i>f678</i> = <i>f679</i> = <i>f680</i> = <i>f681</i> = <i>f682</i> = <i>f683</i> = <i>f684</i> = <i>f685</i> = <i>f686</i> = <i>f687</i> = <i>f688</i> = <i>f689</i> = <i>f690</i> = <i>f691</i> = <i>f692</i> = <i>f693</i> = <i>f694</i> = <i>f695</i> = <i>f696</i> = <i>f697</i> = <i>f698</i> = <i>f699</i> = <i>f700</i> = <i>f701</i> = <i>f702</i> = <i>f703</i> = <i>f704</i> = <i>f705</i> = <i>f706</i> = <i>f707</i> = <i>f708</i> = <i>f709</i> = <i>f710</i> = <i>f711</i> = <i>f712</i> = <i>f713</i> = <i>f714</i> = <i>f715</i> = <i>f716</i> = <i>f717</i> = <i>f718</i> = <i>f719</i> = <i>f720</i> = <i>f721</i> = <i>f722</i> = <i>f723</i> = <i>f724</i> = <i>f725</i> = <i>f726</i> = <i>f727</i> = <i>f728</i> = <i>f729</i> = <i>f730</i> = <i>f731</i> = <i>f732</i> = <i>f733</i> = <i>f734</i> = <i>f735</i> = <i>f736</i> = <i>f737</i> = <i>f738</i> = <i>f739</i> = <i>f740</i> = <i>f741</i> = <i>f742</i> = <i>f743</i> = <i>f744</i> = <i>f745</i> = <i>f746</i> = <i>f747</i> = <i>f748</i> = <i>f749</i> = <i>f750</i> = <i>f751</i> = <i>f752</i> = <i>f753</i> = <i>f754</i> = <i>f755</i> = <i>f756</i> = <i>f757</i> = <i>f758</i> = <i>f759</i> = <i>f760</i> = <i>f761</i> = <i>f762</i> = <i>f763</i> = <i>f764</i> = <i>f765</i> = <i>f766</i> = <i>f767</i> = <i>f768</i> = <i>f769</i> = <i>f770</i> = <i>f771</i> = <i>f772</i> = <i>f773</i> = <i>f774</i> = <i>f775</i> = <i>f776</i> = <i>f777</i> = <i>f778</i> = <i>f779</i> = <i>f780</i> = <i>f781</i> = <i>f782</i> = <i>f783</i> = <i>f784</i> = <i>f785</i> = <i>f786</i> = <i>f787</i> = <i>f788</i> = <i>f789</i> = <i>f790</i> = <i>f791</i> = <i>f792</i> = <i>f793</i> = <i>f794</i> = <i>f795</i> = <i>f796</i> = <i>f797</i> = <i>f798</i> = <i>f799</i> = <i>f800</i> = <i>f801</i> = <i>f802</i> = <i>f803</i> = <i>f804</i> = <i>f805</i> = <i>f806</i> = <i>f807</i> = <i>f808</i> = <i>f809</i> = <i>f810</i> = <i>f811</i> = <i>f812</i> = <i>f813</i> = <i>f814</i> = <i>f815</i> = <i>f816</i> = <i>f817</i> = <i>f818</i> = <i>f819</i> = <i>f820</i> = <i>f821</i> = <i>f822</i> = <i>f823</i> = <i>f824</i> = <i>f825</i> = <i>f826</i> = <i>f827</i> = <i>f828</i> = <i>f829</i> = <i>f830</i> = <i>f831</i> = <i>f832</i> = <i>f833</i> = <i>f834</i> = <i>f835</i> = <i>f836</i> = <i>f837</i> = <i>f838</i> = <i>f839</i> = <i>f840</i> = <i>f841</i> = <i>f842</i> = <i>f843</i> = <i>f844</i> = <i>f845</i> = <i>f846</i> = <i>f847</i> = <i>f848</i> = <i>f849</i> = <i>f850</i> = <i>f851</i> = <i>f852</i> = <i>f853</i> = <i>f854</i> = <i>f855</i> = <i>f856</i> = <i>f857</i> = <i>f858</i> = <i>f859</i> = <i>f860</i> = <i>f861</i> = <i>f862</i> = <i>f863</i> = <i>f864</i> = <i>f865</i> = <i>f866</i> = <i>f867</i> = <i>f868</i> = <i>f869</i> = <i>f870</i> = <i>f871</i> = <i>f872</i> = <i>f873</i> = <i>f874</i> = <i>f875</i> = <i>f876</i> = <i>f877</i> = <i>f878</i> = <i>f879</i> = <i>f880</i> = <i>f881</i> = <i>f882</i> = <i>f883</i> = <i>f884</i> = <i>f885</i> = <i>f886</i> = <i>f887</i> = <i>f888</i> = <i>f889</i> = <i>f890</i> = <i>f891</i> = <i>f892</i> = <i>f893</i> = <i>f894</i> = <i>f895</i> = <i>f896</i> = <i>f897</i> = <i>f898</i> = <i>f899</i> = <i>f900</i> = <i>f901</i> = <i>f902</i> = <i>f903</i> = <i>f904</i> = <i>f905</i> = <i>f906</i> = <i>f907</i> = <i>f908</i> = <i>f909</i> = <i>f910</i> = <i>f911</i> = <i>f912</i> = <i>f913</i> = <i>f914</i> = <i>f915</i> = <i>f916</i> = <i>f917</i> = <i>f918</i> = <i>f919</i> = <i>f920</i> = <i>f921</i> = <i>f922</i> = <i>f923</i> = <i>f924</i> = <i>f925</i> = <i>f926</i> = <i>f927</i> = <i>f928</i> = <i>f929</i> = <i>f930</i> = <i>f931</i> = <i>f932</i> = <i>f933</i> = <i>f934</i> = <i>f935</i> = <i>f936</i> = <i>f937</i> = <i



Aktionmethode	Beschreibung	Seite
<code>absDifferenz</code>	Mit dem <code>absDifferenz</code> Operator wird die Differenzmenge der Elemente zweier Arrays berechnet.	191
<code>absVereinigung</code>	Mit dem <code>absVereinigung</code> Operator wird die Durchschliffmenge der Elemente zweier Arrays berechnet.	192
<code>setdifferenz</code>	Mit dem <code>setdifferenz</code> Operator wird die Weisungsmenge der Elemente zweier Arrays berechnet.	193
<code>setvereinigung</code>	Mit dem <code>setvereinigung</code> Operator wird vereinigt zwischen 2 Arrays eine Untersuchungsmenge berechnet.	193

**Ablösung 78: Reziproke Mengenoperatoren**

---

• Codeleiste: <b>Reziproke Operatoren</b>	
1 //	// Bemerkung: Bemerkungen
2 //	
3 //	
4 // Ausprägung: Ausprägung:	
5 // A = { "A", "B", "C", "D" },	
6 // B = { "E", "F", "G", "H" }	
7 // C = { "I", "J", "K", "L" }	
8 // D = { "M", "N", "O", "P" }	
9 // E = { "Q", "R", "S", "T" }	
10 // F = { "U", "V", "W", "X" }	
11 // G = { "Y", "Z", "A", "B" }	
12 // H = { "C", "D", "E", "F" }	
13 // I = { "G", "H", "I", "J" }	
14 // J = { "K", "L", "M", "N" }	
15 // K = { "O", "P", "Q", "R" }	
16 // L = { "S", "T", "U", "V" }	
17 // M = { "W", "X", "Y", "Z" }	
18 // N = { "A", "B", "C", "D" }	
19 // O = { "E", "F", "G", "H" }	
20 // P = { "I", "J", "K", "L" }	
21 // Q = { "M", "N", "O", "P" }	
22 // R = { "S", "T", "U", "V" }	
23 // S = { "W", "X", "Y", "Z" }	
24 // T = { "A", "B", "C", "D" }	
25 // U = { "E", "F", "G", "H" }	
26 // V = { "I", "J", "K", "L" }	
27 // W = { "M", "N", "O", "P" }	
28 // X = { "S", "T", "U", "V" }	
29 // Y = { "W", "X", "Y", "Z" }	
30 // Z = { "A", "B", "C", "D" }	

**• 13.16.2: Bemerkungen Operator**

Mit dem `Bemerkungen` Operator wird die Durchschliffmenge der Elemente zweier Arrays berechnet.

Der Durchschliff zweier Arrays ist ab diesem Operator definiert. Es kann nur dann berechnet werden, wenn beide Arrays gleich lang sind und sie keine Mengen verhalten statt.

---

• Codeleiste: <b>Bemerkungen Operator</b>	
1 //	// Bemerkung: Bemerkungen
2 //	
3 //	
4 // Ausprägung: Ausprägung:	
5 // A = { "A", "B", "C", "D" },	
6 // B = { "E", "F", "G", "H" }	
7 // C = { "I", "J", "K", "L" }	
8 // D = { "M", "N", "O", "P" }	
9 // E = { "Q", "R", "S", "T" }	
10 // F = { "U", "V", "W", "X" }	
11 // G = { "Y", "Z", "A", "B" }	
12 // H = { "C", "D", "E", "F" }	
13 // I = { "G", "H", "I", "J" }	
14 // J = { "K", "L", "M", "N" }	
15 // K = { "O", "P", "Q", "R" }	
16 // L = { "S", "T", "U", "V" }	
17 // M = { "W", "X", "Y", "Z" }	
18 // N = { "A", "B", "C", "D" }	
19 // O = { "E", "F", "G", "H" }	
20 // P = { "I", "J", "K", "L" }	
21 // Q = { "M", "N", "O", "P" }	
22 // R = { "S", "T", "U", "V" }	
23 // S = { "W", "X", "Y", "Z" }	
24 // T = { "A", "B", "C", "D" }	
25 // U = { "E", "F", "G", "H" }	
26 // V = { "I", "J", "K", "L" }	
27 // W = { "M", "N", "O", "P" }	
28 // X = { "S", "T", "U", "V" }	
29 // Y = { "W", "X", "Y", "Z" }	
30 // Z = { "A", "B", "C", "D" }	



Informatikoptionen > Theoriekriptum



**19. Datenformat - XML**

# 01

XML - Grundlagen

19.1 XML Grundlagen

**XML** ist ein **Datenformat** mit **strukturierten Daten**, die **Text** in Form einer **hierarchischen Struktur** aufweisen.

**XML** ist ein **Standard** zur **Übertragung von Daten**.

Hierarchisch strukturierte Daten haben die **Welt** in Form einer **hierarchischen Struktur**.

**XML** ist ein **Standard** zur **Übertragung von Daten**.

**XML** ist ein **Standard** zum **Austausch von Daten** zwischen Anwendungen und Softwareplattformen.

Neben **JSON** ist **XML** das wichtigste **Universalformat** für **Internetprotokolle**.

**Anwendungskonfiguration** = In der Anwendungskonfiguration gibt **XML** die **Struktur** für **Internetprotokolle**.

**Auszeichnungssyntax XML** = **XML** wird mit der **Motivation** entwickelt eine **Leserichtungssyntax** für den **Leser** zu erhalten.

Die Auszeichnungssyntax bestimmt das **wichtigste Element** des Dokuments bestimmt sowie die **leserfreundliche Auszeichnung** spezifisch in **HTML**.

**XML** wird mit der **Motivation** entwickelt eine **leserfreundliche Auszeichnungssyntax** für den **Leser** zu erhalten.

**19.1.2 Fallbeispiel: XML-Dokumente**

XML wird mit der Motivation entwickelt eine **leserfreundliche Auszeichnungssyntax** für den **Leser** zu erhalten.

(0) Nach Tag - Mac - Find - Find in files

Abbildung 85: Aufbau eines XML-Dokuments

**19.1.2 Aufbau eines XML-Dokuments**

**XML-Datenstruktur** = Das **XML-Dokument** besteht aus **XML-Elementen**. Diese sind wiederum aus **Attributnamen** und **Attributwerten** zusammengesetzt.

**XML-Datenelement** = Ein **XML-Datenelement** besteht aus dem **Elementnamen** und dem **Inhalt** des Elements.

**XML-Datenelementtyp** = Ein **XML-Datenelementtyp** ist eine Klassifizierung, die die **Werte** des **XML-Datenelements** bestimmen.

**XML-Datenelementwert** = Ein **XML-Datenelementwert** ist ein Wert, der im **XML-Datenelement** definiert ist.

**WRL-Datenstruktur** = Ein **WRL-Datenstruktur** ist ein **spezifisches Format**, das die **Struktur** eines **XML-Dokuments** beschreibt.

**WRL-Datenelement** = Ein **WRL-Datenelement** ist ein **spezifisches Format**, das die **Struktur** eines **XML-Datenelements** beschreibt.

**WRL-Datenelementtyp** = Ein **WRL-Datenelementtyp** ist eine Klassifizierung, die die **Werte** des **WRL-Datenelements** bestimmen.

**WRL-Datenelementwert** = Ein **WRL-Datenelementwert** ist ein Wert, der im **WRL-Datenelement** definiert ist.

**WRL-Datenwert** = Ein **WRL-Datenwert** ist ein **spezifisches Format**, das die **Werte** des **WRL-Datenelementwerts** beschreibt.

Informationssysteme

**19.2 XML Elemente**

**XML-Dokument** = Ein **XML-Dokument** besteht aus **XML-Datenelementen**. Auf logische Weise besteht es aus einem strukturierten **XML-Dokument** ohne **Wortbedeutung**.

**XML-Datenelement** = Ein **XML-Datenelement** ist die **grundsätzliche Einheit** eines **XML-Dokuments**.

**XML-Datenelementtyp** = Ein **XML-Datenelementtyp** ist eine **Klassifizierung** der **XML-Datenelemente**, welche die **Werte** des **XML-Datenelements** bestimmen.

**XML-Attribut** = Ein **XML-Attribut** kennzeichnet **XML-Datenelemente**, die **beschrieben** werden.

**19.2.1 Struktur eines XML-Dokuments**

**XML-Dokument** = Ein **XML-Dokument** besteht aus **XML-Datenelementen**.

**XML-Datenelement** = Ein **XML-Datenelement** besteht aus **Attributnamen** und **Attributwerten**.

**XML-Datenelementtyp** = Ein **XML-Datenelementtyp** ist eine **Klassifizierung** der **XML-Datenelemente**, welche die **Werte** des **XML-Datenelements** bestimmen.

**XML-Datenelementwert** = Ein **XML-Datenelementwert** ist ein **spezifisches Format**, das die **Werte** des **XML-Datenelementwerts** beschreibt.

**WRL-Datenelement** = Ein **WRL-Datenelement** ist ein **spezifisches Format**, das die **Struktur** eines **XML-Datenelements** beschreibt.

**WRL-Datenelementtyp** = Ein **WRL-Datenelementtyp** ist eine **Klassifizierung**, die die **Werte** des **WRL-Datenelements** bestimmen.

**WRL-Datenelementwert** = Ein **WRL-Datenelementwert** ist ein **spezifisches Format**, das die **Werte** des **WRL-Datenelementwerts** beschreibt.

**WRL-Datenwert** = Ein **WRL-Datenwert** ist ein **spezifisches Format**, das die **Werte** des **WRL-Datenelementwerts** beschreibt.

**19.2.2 Ablauf einer XML-Dokumentanalyse**

**1. Dokumenttypenbestimmung** = Der Analyseprozess beginnt mit der **Bestimmung** des **XML-Dokumenttyps**. Auf logische Weise besteht es aus einem strukturierten **XML-Dokument** ohne **Wortbedeutung**.

**2. XML-Datenelementtypenbestimmung** = Der Analyseprozess geht in die **Bestimmung** der **XML-Datenelementtypen** über. Hierbei wird die **Wortbedeutung** der **XML-Datenelemente** ermittelt.

**3. XML-Datenelementwertbestimmung** = Der Analyseprozess geht in die **Bestimmung** der **XML-Datenelementwerte** über. Hierbei wird die **Wortbedeutung** der **XML-Datenelementwerte** ermittelt.

**4. WRL-Datenelementtypenbestimmung** = Der Analyseprozess geht in die **Bestimmung** der **WRL-Datenelementtypen** über. Hierbei wird die **Wortbedeutung** der **WRL-Datenelemente** ermittelt.

**5. WRL-Datenelementwertbestimmung** = Der Analyseprozess geht in die **Bestimmung** der **WRL-Datenelementwerte** über. Hierbei wird die **Wortbedeutung** der **WRL-Datenelementwerte** ermittelt.





Abbildung 43. Antikörperprofil aus KM1. Histogramm

### 19.2.2 Aufbau eines XML-Dokuments

**XSL-Elemente** bestehen immer aus einem Start- und einem Endtag. Die Bezeichnung des Elements kann dabei beliebig sein.

- Erklärung der Elemente:
    - Beispiel: <Person name="John">First name</Person>
    - Das XML-Element besteht aus dem Start-Zeichen name und dem dazugehörigen Inhalt First name.
    - Das Element entspricht dem Token John.
  - unstrukturierter Inhalt
  - Das XML-Dokument enthält Daten in Form eines Zeichenketten. Der Inhalt des XML-Elements wird als unstrukturiert eingeschlossen.

1

1



1



3



**Informationssysteme**

**10.5 XML Namensräume**

- XML-Namensräume ist ein logischer Name, der XML-Dokumente und XML-Schemata kennzeichnen und unterscheiden.
- Innerhalb eines Namensraums haben alle Elemente den gleichen XML-Dokumentenraum.
- Analyse XML-Namensräume
- Das Konzept eines XML-Namensraums kann auf das Element von einem anderen XML-Namensraum angewendet werden.
- Ein XML-Namensraum ist ein logischer Namensraum, der den XML-Elementen eines logischen Namensraums eine hierarchische Struktur verleiht.
- Prinzipien des XML-Elements mit einem Namen oder einer entsprechenden Belegung zu definieren.
- **10.5.1 Namensräumlichkeit**
- Der logische Riegel kann die andere nicht erkennen, wenn es verschiedene Namen, bestehend aus unveränderlichen Komponenten, gibt.
- Prinzipien des XML-Elements mit einem Namen oder einer entsprechenden Belegung zu definieren.
- **10.5.2 XML-Namensräume**

  - **• Vererbung von XML-Namensräumen**
  - Der Ausdruck von Namensräumen verleiht Elementen in Namensräumen erweiterte Möglichkeiten.
  - XML-Namensräume eignen sich logische Zusammenfassungen von XML-Dokumenten.
  - Namensräume werden durch eine URL abgetrennt.
  - Jedes Element des Namensraums wird im Dokument erweitert um seine Zugehörigkeit zum Namespace.

**Informationssysteme**

**10.5.3 Namensraumdefinition**

Namensraumdefinitionen werden global eingesetzt, um XML-Namensräume werden innerhalb eines Dokuments lokal eingesetzt.

- **• Globalisiertes Namensräume**
- Der Name kommt nicht mehr direkt vor dem Element, sondern wird über eine URL definiert.
- **• Standard-Namensräume**
- Der Name kommt nicht mehr direkt vor dem Element, sondern wird über eine URL definiert.

**Informationssysteme**

**XML Technologie**

**10.6 Logische Schicht**

- XML wird mit dem Ausprägen verzweigt, da es nur einfache Datentypen für die Datenstruktur schafft.
- **10.6.1 XML Technologien**
- Die Verarbeitung von XML-Daten wurde eine Reihe von Technologien erfordert.
- **• Anwendung XML-Technologien**
- XML-Dokumente definieren die Datenstruktur.
- XML-Schemata XML-Schemata werden verwendet um die Struktur für XML-Dokumente zu definieren.
- XSLT ist ein XML-Sprache zur Transformation von XML-Dokumenten.
- XSLT ist eine XML-Sprache zur Transformation von XML-Dokumenten.
- XQuery-XQuery ist eine Abfragesprache für XML-Dokumente.

**10.6.2 XML-Komponentendaten**

Bevor ein XML-Dokument verarbeitet werden kann, muss es das Dokument eine logische Struktur gewinnen.

- **• XML-Komponentendaten**
- Das Vorgehen aufteilt einen XML-Dokument in sprach- oder hierarchische Komponenten, die dann von Komponenten bearbeitet werden.
- Damit müssen XML-Dokumente innerer Struktur und äußerer Struktur haben.
- **• Entwurf XML-Komponentendaten**
- Die XML-Komponentendaten besteht aus Komponenten.
- Die Ausführung einer Komponente entspricht dabei der bearbeitung einer logischen Struktur, die die entsprechenden XML-Daten enthält. Bei der entsprechenden Komponente in XML-Dokumenten werden die Komponenten spezifisch bearbeitet.



Abbildung 19. Strukturen von logischen Wörtern

**19.3 Komponentenwörter**  
Jede XML-Dokument-Instanz hat eine hierarchische Struktur aus Komponentenwörtern.

- Erläuterung Komponentenwörter:**
  - Jedes Element eines XML-Dokuments definiert dabei die Instanz eines logischen Wörterketten-Modells. Diese Instanz ist ein logisches Wort, Konkurrenz, XML-Element, XML-Atribut.
  - Wörterketten-Elemente in einem XML-Dokument sind nicht direkt mit dem entsprechenden Typ für Konkurrenz ausgetauscht.
  - Elemente die in einem XML-Element eingeschlossen sind, werden als Wörterketten des entsprechenden Dokumentbaums angesehen.
  - Wörterketten können im Dokumentbaum direkt übergeordnet oder untergeordnet sein. Diese Hierarchie kann über logische Wörterketten ausgetauscht werden. Die Hierarchie wird dann Elementen des Wörterketten-Modells zugewiesen.
  - Wörterketten können über logische Wörterketten ausgetauscht werden.

**19.4 Konkurrenz**  
Die XML-Spezifikation definiert für die logischen Wörterketten Konkurrenz.

- Erläuterung Konkurrenz:**
  - Wörterketten:** Das Wörterketten ist der zweite Knoten einer Konkurrenz. Der Wörterketten enthält alle anderen Knoten des Konkurrenzbaums.
  - Dokumentketten:** Dokumentketten entsprechen den logischen Wörterketteninstanzen eines XML-Dokuments.
  - Atributketten:** Atributketten entsprechen den logischen Wörterketten eines Attributs. Die Atributketten wird dann Elementen des Wörterketten-Modells zugewiesen.
  - Wörterketten:** Die in einem Element eingeschlossenen Daten werden als Wörterketten des entsprechenden Dokumentbaums angesehen. Die Wörterketten sind dann Elementen des Wörterketten-Modells zugewiesen.
  - Namensketten:** Der Namenswert eines Elements ist ihm als Ein Wörterketten zugewiesen.

Abbildung 20. Datenformat - XML/XPath

**20. Datenformat - XML/XPath**

**03**

**20.1 XPath - Konzepte**

**Erläuterung XPath-Konzepte:**  
XPath selbst ist keine XML-Sprache. Der XPath Standard definiert nur die Syntax und Semantik von XPath-Expressionen.

**20.1.1 XPath Konzepte**  
Der Simple XPath reicht in ein Mauszeig. Die Atribut steht unterstützend für die Ausdrücke und Faktur

- Erläuterung XPath Grundlagen:**
  - Wähle selbst in keine XML-Sprache. Der XPath Standard definiert nur die Syntax und Semantik von XPath-Expressionen.
  - Mit einem XPath-Präfix kann die Praktik der Anwendung von XML-Knoteninstanzen beschrieben werden.
- Erläuterung Ladeknotenprüfung:**
  - Der XPath-Standard definiert keine Praktik, wie man die entsprechenden XML-Daten für die untersuchte Seite herstellen sollte. Ladeknotenprüfung ist eine Praktik, die die Praktik bestimmen soll, wie man die entsprechenden XML-Daten für die Untersuchung eines XML-Dokuments ausgewählt.
  - Bei Ladeknotenprüfung kann eine bestimmte Praktik festgestellt werden, und die Ergebnisse sind in einem Pfad enthalten. Wenn die Praktik bestimmt, dass ein bestimmter Knoten kein gewünschter Knoten ist, kann dies durch einen Fehler angezeigt werden.
  - Der XPath-Standard definiert keine Praktik, die auf die Praktik der Ladeknotenprüfung auftragen. Es gibt jedoch Praktiken, die in Bezug auf die Praktik der Ladeknotenprüfung berichten.
- Erläuterung Xpath-Präfixe:**
  - Die Existenz der Auswertung eines Ladeknotenprüfung ist die Unterscheidung zwischen Ladeknotenprüfung und Praktik der Ladeknotenprüfung.
  - Ein Ladeknotenprüfung ist eine Praktik, die die Praktik bestimmt, wie man die entsprechenden XML-Daten für die untersuchte Seite herstellen sollte.

Abbildung 20. Konstruktion eines XML-Dokuments

**20.2 Pfadausdrücke in Kettform**

**Erläuterung Pfadausdrücke:**  
Die XPath-Spezifikation erlaubt für Ladeknotenprüfung die eine verschachtelte Schreibweise des XPath-Knoten-Name.

- Erläuterung XPath-Ketten:**
  - Bei der Auswertung von Pfadausdrücken ist es wichtig, dass der XPath-Name von Seite von Atribut abweichen.
  - Bei komplexer Form schafft die XPath-Kette die Verbindung zwischen den verschiedenen Pfadausdrücken. Sie verhindert das ungewöhnliche Pfadausdrücke ist in einer Reihenfolge, die auf eine Zahl von Zeichen basiert.
- Erläuterung Ladeknotenprüfung:**
  - Ein Ladeknotenprüfung ist eine Praktik von Ladeknotenprüfung, Ladeknotenprüfung und verschachtelter Ladeknotenprüfung.
  - Je nach Konkurrenz-Ordnung des XPath-Symbolen sind eigene Operatoren für die Konkurrenz-Ordnung.

**20.2.2 Auswahl von Elementknoten**  
Die Position von Elementknoten in XML-Dokumenten kann durch Pfadausdrücke bestimmt werden. Um Pfadausdrücke zu verstehen, müssen wir die Praktik der Konkurrenz von praktischen Pfadausdrücken verstehen.

- Erläuterung Pfadausdrücke:**
  - Die Praktik der Konkurrenz von Pfadausdrücken wird im Kapitel Pfad ausgewertet. Pfad ausgewertet ist die Praktik der Konkurrenz von praktischen Pfadausdrücken.
- Erläuterung Ladeknotenprüfung:**
  - Jeder Ladeknotenprüfung ist ein Pfadausdruck, der spezifisch dafür ist, die entsprechende Praktik zu bestimmen.
- Erläuterung Ladeknotenprüfung:**
  - Ein Ladeknotenprüfung ist eine Praktik von Ladeknotenprüfung, Ladeknotenprüfung und verschachtelter Ladeknotenprüfung.
  - Bei Ladeknotenprüfung ist es wichtig, dass der XPath-Name von Seite von Atribut abweichen.
  - Bei komplexer Form schafft die XPath-Kette die Verbindung zwischen den verschiedenen Pfadausdrücken. Sie verhindert das ungewöhnliche Pfadausdrücke ist in einer Reihenfolge, die auf eine Zahl von Zeichen basiert.



Übung: Was ist ein Prädikat? Was ist ein Attribut?

### 20.3. Lösungskonzept

**Die Ergebnisse eines SPaK-Antrags ist ein Attribut des entsprechenden Objekts.**

### 20.3.1. Lösungskonzept - Datenbank

Die XPAK-Syntaktische Arbeit ist 4 Typen von SPaK-Attributtypen:

- 4 Auflistung: **Attributtypen für Liegenschaftsdaten**

	<b>Kostenstrom</b> Die Kostenstrom ist eine Teilmenge der Kosten eines Betriebes.
	<b>Elternges.</b> Elternges. ist eine Beziehung. Es beschreibt, ob eine Person Elternteil einer anderen Person ist. z.B.: "Hans Vater"
	<b>Nummer</b> Nummer Objekte und ist mit Hakenmarkierung. z.B.: 34
	<b>Beschriftung</b> Beschriftung Objekte nehmen entweder den Wert einer Kette oder ...

• **Attributtypen für Liegenschaftsdaten**

### 20.4. Prädikat

**Der Prädikator kann eine Konversation über ein Prädikat definieren works.**

### 20.4.1. Definitionen von Prädikaten

**XPAK-Prädikat =**

**In Prädikat oder logische Aussicht.**

• **Erklärung: Filtern mit Prädikaten**

- Prädikat kann eine Aussicht filtern, die nur die zu Liegenschaftsobjekt-Attribut, die relevanten Kosten und fiktiv das Wertespektrum der XPAK-Familie einschließen.
- Für eine XPAK-Liegenschaftsmappe kann eine beliebige Prädikat definiert werden.
- Wenn Prädikat definiert ist, dann wird dieses Prädikat benutzt, wenn die XPAK-Familie mit einem Prädikat gefiltert wird.
- Prädikat kann eine Aussicht über die Objekttypen (Attributtypen), z.B.  $x_1, x_2, \dots, x_n, y_1, y_2$ .
- Attribut und bracket werden, dass Operatoren wie AND, OR, NOT, NOT-ALL, NOT-EXISTS, Distanz- und condition-Atts, endlich durch die entsprechenden Bedeutungsmerkmale bzw. log. operatoren ersetzen werden.

• **Erklärungen Prädikate**

- $\exists$  **Prädikat**
- **Prädikat mit einem bestimmten ID** **PrädikatID=123456789**
- $\exists$  **All Personen die ein Element haben** **[PersonenID]**
- $\exists$  **Alle ergebnisse eines Formular Seines [FormularID]**
- $\exists$  **Alle Personen die einen Projekt bearbeiten** **[ProjektID]**
- $\exists$  **Alle Personen einer Nachbarschaft** **[NachbarschaftID]**
- $\exists$  **Alle Personen die eine Adresse haben** **[AdresseID]**

	Ablen	Kontrollablage	Pflichtab	Informationsrahmen
/child::*/project/resource::*[refid='16-34256563276-5229-23']				
Lokalisierungsraum				Lokalisierungsräume
<b>20.5. Platzierstrukturen in Standardform</b>				
Die Ablen Spezifiziert die Platzierung von Lokalisierungsobjekten in ihrer Standardform. Ihre Lokalisierungspunkte in Kons. Form.				
Die Lokalisierungspunkte ist eine Folge von Lokalisierungsraum.				
<b>20.5.1. Lokalisierungsraum</b>				
Eine Lokalisierungsraum besteht aus 3 angefügten Segmente.				
+ <b>Spez:</b> Lokalisierungsräume				
+ <b>Werk:</b> Spezif. Lokalisierungsraum				
+ <b>Index:</b> Lokalisierungsräume[1..1]				
→ Aufführung: Segmente einer Lokalisierungsraum				
<b>Achsenbeschreibung</b> = Der Achsenbeschreibung definiert die Richtung des Koordinatensystems einer Lokalisierung. Sie legt horizontale und vertikale Achsen fest.				
Die Angabe einer Achsenbeschreibung ist vor Pflichtig.				
<b>Knotenbeschreibung</b> = Die Knotenbeschreibung ermöglicht eine Wiederverwendung der durch den Knotenbeschreibung definierten Strukturen.				
Die Angabe einer Knotenbeschreibung ist optional.				
<b>Pflichtab</b> = Pflichtab erfordert die Formulierung komplexe Pflichtbedingungen.				



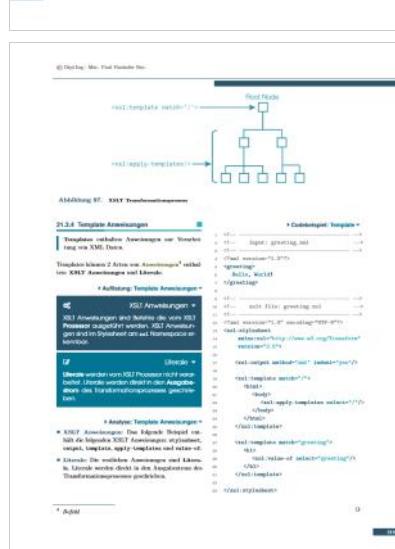
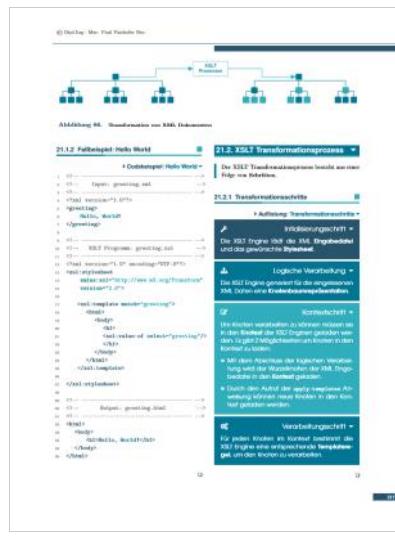




Informationsarchitektur	
// -----	<b>* Codebeispiel: Stringtukoren*</b>
// -----	
// Match Position: string.length	
// -----	
// an-string.string.length()	
// an-string.length()	
// -----	
// Match: string.length('abc')	
// RegExps: 0	
// -----	
// Match Position: contains	
// -----	
// an-string.contains()	
// an-string.startsWith()	
// an-string.endsWith()	
// -----	
// an-string.startsWith('')	
// -----	
// Match: contains('Shakespeare', 'ape')	
// -----	
// Match: contains('Shakespeare', '')	
// RegExps: false	
// -----	
// Match: contains(*, 'Shakespeare')	
// RegExps: true	
// -----	
// Match: contains(*, 'ape')	
// -----	
// Match: containsLast('ape', 'ape')	
// -----	
// Match: containsLast(*, 'ape')	
// RegExps: true	
// -----	
// Match Position: matching	
// -----	
// an-string.matching('abc')	
// an-string.matching('abc', 0)	
// RegExps: null	
// -----	
// Match: matching('abc', 0, 2)	
// RegExps: abc	
// -----	
// Match: matching('abc', 10, 20)	
// RegExps: null	
// -----	
// Match: matching('abc', 1, 30)	

	Inhaltsverzeichnis
<b>21. Datenformat - XML, XSLT</b>	
	
XSLT (Hyperlinks)	
<b>21.1 XML Grundlagen</b>	216
<b>21.2 XSLT Transformationsschritte</b>	217
<b>21.3 XSLT Programme</b>	218
<b>21.4 Deklarative Verarbeitung</b>	220
<b>21.5 Prozedurale Verarbeitung</b>	222
<b>21.6 Ausgewähltes</b>	225
<b>21.7 Suchanfragen</b>	227
XSLT (Hyperlinks)	
<b>21.1 XSLT Grundlagen</b>	216
<b>21.1.1 XML</b>	216
<b>21.1.2 XSLT Standard</b>	217
<b>21.1.3 XSLT Praktische Anwendung</b>	218
<b>21.1.4 XSLT Implementierungen</b>	219
<b>21.1.5 XSLT und XML</b>	220
<b>21.1.6 XSLT und CSS</b>	221
<b>21.1.7 XSLT und JavaScript</b>	222
<b>21.1.8 XSLT und XQuery</b>	223
<b>21.1.9 XSLT und XPath</b>	224
<b>21.1.10 XSLT und XSLT 2.0</b>	225
<b>21.1.11 XSLT und XSLT 3.0</b>	226
<b>21.1.12 XSLT und XSLT 3.1</b>	227
<b>21.1.13 XSLT und XSLT 3.2</b>	228
<b>21.1.14 XSLT und XSLT 3.3</b>	229
<b>21.1.15 XSLT und XSLT 3.4</b>	230
<b>21.1.16 XSLT und XSLT 3.5</b>	231
<b>21.1.17 XSLT und XSLT 3.6</b>	232
<b>21.1.18 XSLT und XSLT 3.7</b>	233
<b>21.1.19 XSLT und XSLT 3.8</b>	234
<b>21.1.20 XSLT und XSLT 3.9</b>	235
<b>21.1.21 XSLT und XSLT 3.10</b>	236
<b>21.1.22 XSLT und XSLT 3.11</b>	237
<b>21.1.23 XSLT und XSLT 3.12</b>	238
<b>21.1.24 XSLT und XSLT 3.13</b>	239
<b>21.1.25 XSLT und XSLT 3.14</b>	240
<b>21.1.26 XSLT und XSLT 3.15</b>	241
<b>21.1.27 XSLT und XSLT 3.16</b>	242
<b>21.1.28 XSLT und XSLT 3.17</b>	243
<b>21.1.29 XSLT und XSLT 3.18</b>	244
<b>21.1.30 XSLT und XSLT 3.19</b>	245
<b>21.1.31 XSLT und XSLT 3.20</b>	246
<b>21.1.32 XSLT und XSLT 3.21</b>	247
<b>21.1.33 XSLT und XSLT 3.22</b>	248
<b>21.1.34 XSLT und XSLT 3.23</b>	249
<b>21.1.35 XSLT und XSLT 3.24</b>	250
<b>21.1.36 XSLT und XSLT 3.25</b>	251
<b>21.1.37 XSLT und XSLT 3.26</b>	252
<b>21.1.38 XSLT und XSLT 3.27</b>	253
<b>21.1.39 XSLT und XSLT 3.28</b>	254
<b>21.1.40 XSLT und XSLT 3.29</b>	255
<b>21.1.41 XSLT und XSLT 3.30</b>	256
<b>21.1.42 XSLT und XSLT 3.31</b>	257
<b>21.1.43 XSLT und XSLT 3.32</b>	258
<b>21.1.44 XSLT und XSLT 3.33</b>	259
<b>21.1.45 XSLT und XSLT 3.34</b>	260
<b>21.1.46 XSLT und XSLT 3.35</b>	261
<b>21.1.47 XSLT und XSLT 3.36</b>	262
<b>21.1.48 XSLT und XSLT 3.37</b>	263
<b>21.1.49 XSLT und XSLT 3.38</b>	264
<b>21.1.50 XSLT und XSLT 3.39</b>	265
<b>21.1.51 XSLT und XSLT 3.40</b>	266
<b>21.1.52 XSLT und XSLT 3.41</b>	267
<b>21.1.53 XSLT und XSLT 3.42</b>	268
<b>21.1.54 XSLT und XSLT 3.43</b>	269
<b>21.1.55 XSLT und XSLT 3.44</b>	270
<b>21.1.56 XSLT und XSLT 3.45</b>	271
<b>21.1.57 XSLT und XSLT 3.46</b>	272
<b>21.1.58 XSLT und XSLT 3.47</b>	273
<b>21.1.59 XSLT und XSLT 3.48</b>	274
<b>21.1.60 XSLT und XSLT 3.49</b>	275
<b>21.1.61 XSLT und XSLT 3.50</b>	276
<b>21.1.62 XSLT und XSLT 3.51</b>	277
<b>21.1.63 XSLT und XSLT 3.52</b>	278
<b>21.1.64 XSLT und XSLT 3.53</b>	279
<b>21.1.65 XSLT und XSLT 3.54</b>	280
<b>21.1.66 XSLT und XSLT 3.55</b>	281
<b>21.1.67 XSLT und XSLT 3.56</b>	282
<b>21.1.68 XSLT und XSLT 3.57</b>	283
<b>21.1.69 XSLT und XSLT 3.58</b>	284
<b>21.1.70 XSLT und XSLT 3.59</b>	285
<b>21.1.71 XSLT und XSLT 3.60</b>	286
<b>21.1.72 XSLT und XSLT 3.61</b>	287
<b>21.1.73 XSLT und XSLT 3.62</b>	288
<b>21.1.74 XSLT und XSLT 3.63</b>	289
<b>21.1.75 XSLT und XSLT 3.64</b>	290
<b>21.1.76 XSLT und XSLT 3.65</b>	291
<b>21.1.77 XSLT und XSLT 3.66</b>	292
<b>21.1.78 XSLT und XSLT 3.67</b>	293
<b>21.1.79 XSLT und XSLT 3.68</b>	294
<b>21.1.80 XSLT und XSLT 3.69</b>	295
<b>21.1.81 XSLT und XSLT 3.70</b>	296
<b>21.1.82 XSLT und XSLT 3.71</b>	297
<b>21.1.83 XSLT und XSLT 3.72</b>	298
<b>21.1.84 XSLT und XSLT 3.73</b>	299
<b>21.1.85 XSLT und XSLT 3.74</b>	300
<b>21.1.86 XSLT und XSLT 3.75</b>	301
<b>21.1.87 XSLT und XSLT 3.76</b>	302
<b>21.1.88 XSLT und XSLT 3.77</b>	303
<b>21.1.89 XSLT und XSLT 3.78</b>	304
<b>21.1.90 XSLT und XSLT 3.79</b>	305
<b>21.1.91 XSLT und XSLT 3.80</b>	306
<b>21.1.92 XSLT und XSLT 3.81</b>	307
<b>21.1.93 XSLT und XSLT 3.82</b>	308
<b>21.1.94 XSLT und XSLT 3.83</b>	309
<b>21.1.95 XSLT und XSLT 3.84</b>	310
<b>21.1.96 XSLT und XSLT 3.85</b>	311
<b>21.1.97 XSLT und XSLT 3.86</b>	312
<b>21.1.98 XSLT und XSLT 3.87</b>	313
<b>21.1.99 XSLT und XSLT 3.88</b>	314
<b>21.1.100 XSLT und XSLT 3.89</b>	315
<b>21.1.101 XSLT und XSLT 3.90</b>	316
<b>21.1.102 XSLT und XSLT 3.91</b>	317
<b>21.1.103 XSLT und XSLT 3.92</b>	318
<b>21.1.104 XSLT und XSLT 3.93</b>	319
<b>21.1.105 XSLT und XSLT 3.94</b>	320
<b>21.1.106 XSLT und XSLT 3.95</b>	321
<b>21.1.107 XSLT und XSLT 3.96</b>	322
<b>21.1.108 XSLT und XSLT 3.97</b>	323
<b>21.1.109 XSLT und XSLT 3.98</b>	324
<b>21.1.110 XSLT und XSLT 3.99</b>	325
<b>21.1.111 XSLT und XSLT 3.100</b>	326
<b>21.1.112 XSLT und XSLT 3.101</b>	327
<b>21.1.113 XSLT und XSLT 3.102</b>	328
<b>21.1.114 XSLT und XSLT 3.103</b>	329
<b>21.1.115 XSLT und XSLT 3.104</b>	330
<b>21.1.116 XSLT und XSLT 3.105</b>	331
<b>21.1.117 XSLT und XSLT 3.106</b>	332
<b>21.1.118 XSLT und XSLT 3.107</b>	333
<b>21.1.119 XSLT und XSLT 3.108</b>	334
<b>21.1.120 XSLT und XSLT 3.109</b>	335
<b>21.1.121 XSLT und XSLT 3.110</b>	336
<b>21.1.122 XSLT und XSLT 3.111</b>	337
<b>21.1.123 XSLT und XSLT 3.112</b>	338
<b>21.1.124 XSLT und XSLT 3.113</b>	339
<b>21.1.125 XSLT und XSLT 3.114</b>	340
<b>21.1.126 XSLT und XSLT 3.115</b>	341
<b>21.1.127 XSLT und XSLT 3.116</b>	342
<b>21.1.128 XSLT und XSLT 3.117</b>	343
<b>21.1.129 XSLT und XSLT 3.118</b>	344
<b>21.1.130 XSLT und XSLT 3.119</b>	345
<b>21.1.131 XSLT und XSLT 3.120</b>	346
<b>21.1.132 XSLT und XSLT 3.121</b>	347
<b>21.1.133 XSLT und XSLT 3.122</b>	348
<b>21.1.134 XSLT und XSLT 3.123</b>	349
<b>21.1.135 XSLT und XSLT 3.124</b>	350
<b>21.1.136 XSLT und XSLT 3.125</b>	351
<b>21.1.137 XSLT und XSLT 3.126</b>	352
<b>21.1.138 XSLT und XSLT 3.127</b>	353
<b>21.1.139 XSLT und XSLT 3.128</b>	354
<b>21.1.140 XSLT und XSLT 3.129</b>	355
<b>21.1.141 XSLT und XSLT 3.130</b>	356
<b>21.1.142 XSLT und XSLT 3.131</b>	357
<b>21.1.143 XSLT und XSLT 3.132</b>	358
<b>21.1.144 XSLT und XSLT 3.133</b>	359
<b>21.1.145 XSLT und XSLT 3.134</b>	360
<b>21.1.146 XSLT und XSLT 3.135</b>	361
<b>21.1.147 XSLT und XSLT 3.136</b>	362
<b>21.1.148 XSLT und XSLT 3.137</b>	363
<b>21.1.149 XSLT und XSLT 3.138</b>	364
<b>21.1.150 XSLT und XSLT 3.139</b>	365
<b>21.1.151 XSLT und XSLT 3.140</b>	366
<b>21.1.152 XSLT und XSLT 3.141</b>	367
<b>21.1.153 XSLT und XSLT 3.142</b>	368
<b>21.1.154 XSLT und XSLT 3.143</b>	369
<b>21.1.155 XSLT und XSLT 3.144</b>	370
<b>21.1.156 XSLT und XSLT 3.145</b>	371
<b>21.1.157 XSLT und XSLT 3.146</b>	372
<b>21.1.158 XSLT und XSLT 3.147</b>	373
<b>21.1.159 XSLT und XSLT 3.148</b>	374
<b>21.1.160 XSLT und XSLT 3.149</b>	375
<b>21.1.161 XSLT und XSLT 3.150</b>	376
<b>21.1.162 XSLT und XSLT 3.151</b>	377
<b>21.1.163 XSLT und XSLT 3.152</b>	378
<b>21.1.164 XSLT und XSLT 3.153</b>	379
<b>21.1.165 XSLT und XSLT 3.154</b>	380
<b>21.1.166 XSLT und XSLT 3.155</b>	381
<b>21.1.167 XSLT und XSLT 3.156</b>	382
<b>21.1.168 XSLT und XSLT 3.157</b>	383
<b>21.1.169 XSLT und XSLT 3.158</b>	384
<b>21.1.170 XSLT und XSLT 3.159</b>	385
<b>21.1.171 XSLT und XSLT 3.160</b>	386
<b>21.1.172 XSLT und XSLT 3.161</b>	387
<b>21.1.173 XSLT und XSLT 3.162</b>	388
<b>21.1.174 XSLT und XSLT 3.163</b>	389
<b>21.1.175 XSLT und XSLT 3.164</b>	390
<b>21.1.176 XSLT und XSLT 3.165</b>	391
<b>21.1.177 XSLT und XSLT 3.166</b>	392
<b>21.1.178 XSLT und XSLT 3.167</b>	393
<b>21.1.179 XSLT und XSLT 3.168</b>	394
<b>21.1.180 XSLT und XSLT 3.169</b>	395
<b>21.1.181 XSLT und XSLT 3.170</b>	396
<b>21.1.182 XSLT und XSLT 3.171</b>	397
<b>21.1.183 XSLT und XSLT 3.172</b>	398
<b>21.1.184 XSLT und XSLT 3.173</b>	399
<b>21.1.185 XSLT und XSLT 3.174</b>	400
<b>21.1.186 XSLT und XSLT 3.175</b>	401
<b>21.1.187 XSLT und XSLT 3.176</b>	402
<b>21.1.188 XSLT und XSLT 3.177</b>	403
<b>21.1.189 XSLT und XSLT 3.178</b>	404
<b>21.1.190 XSLT und XSLT 3.179</b>	405
<b>21.1.191 XSLT und XSLT 3.180</b>	406
<b>21.1.192 XSLT und XSLT 3.181</b>	407
<b>21.1.193 XSLT und XSLT 3.182</b>	408
<b>21.1.194 XSLT und XSLT 3.183</b>	409
<b>21.1.195 XSLT und XSLT 3.184</b>	410
<b>21.1.196 XSLT und XSLT 3.185</b>	411
<b>21.1.197 XSLT und XSLT 3.186</b>	412
<b>21.1.198 XSLT und XSLT 3.187</b>	413
<b>21.1.199 XSLT und XSLT 3.188</b>	414
<b>21.1.200 XSLT und XSLT 3.189</b>	415
<b>21.1.201 XSLT und XSLT 3.190</b>	416
<b>21.1.202 XSLT und XSLT 3.191</b>	417
<b>21.1.203 XSLT und XSLT 3.192</b>	418
<b>21.1.204 XSLT und XSLT 3.193</b>	419
<b>21.1.205 XSLT und XSLT 3.194</b>	420
<b>21.1.206 XSLT und XSLT 3.195</b>	421
<b>21.1.207 XSLT und XSLT 3.196</b>	422
<b>21.1.208 XSLT und XSLT 3.197</b>	423
<b>21.1.209 XSLT und XSLT 3.198</b>	424
<b>21.1.210 XSLT und XSLT 3.199</b>	425
<b>21.1.211 XSLT und XSLT 3.200</b>	426
<b>21.1.212 XSLT und XSLT 3.201</b>	427
<b>21.1.213 XSLT und XSLT 3.202</b>	428
<b>21.1.214 XSLT und XSLT 3.203</b>	429
<b>21.1.215 XSLT und XSLT 3.204</b>	430
<b>21.1.216 XSLT und XSLT 3.205</b>	431
<b>21.1.217 XSLT und XSLT 3.206</b>	432
<b>21.1.218 XSLT und XSLT 3.207</b>	433
<b>21.1.219 XSLT und XSLT 3.208</b>	434
<b>21.1.220 XSLT und XSLT 3.209</b>	435
<b>21.1.221 XSLT und XSLT 3.210</b>	436
<b>21.1.222 XSLT und XSLT 3.211</b>	437
<b>21.1.223 XSLT und XSLT 3.212</b>	438
<b>21.1.224 XSLT und XSLT 3.213</b>	439
<b>21.1.225 XSLT und XSLT 3.214</b>	440
<b>21.1.226 XSLT und XSLT 3.215</b>	441
<b>21.1.227 XSLT und XSLT 3.216</b>	442
<b>21.1.228 XSLT und XSLT 3.217</b>	443
<b>21.1.229 XSLT und XSLT 3.218</b>	444
<b>21.1.230 XSLT und XSLT 3.219</b>	445
<b>21.1.231 XSLT und XSLT 3.220</b>	446
<b>21.1.232 XSLT und XSLT 3.221</b>	447
<b>21.1.233 XSLT und XSLT 3.222</b>	448
<b>21.1.234 XSLT und XSLT 3.223</b>	449
<b>21.1.235 XSLT und XSLT 3.224</b>	450
<b>21.1.236 XSLT und XSLT 3.225</b>	451
<b>21.1.237 XSLT und XSLT 3.226</b>	452
<b>21.1.238 XSLT und XSLT 3.227</b>	453
<b>21.1.239 XSLT und XSLT 3.228</b>	454
<b>21.1.240 XSLT und XSLT 3.229</b>	455
<b>21.1.241 XSLT und XSLT 3.230</b>	456
<b>21.1.242 XSLT und XSLT 3.231</b>	457
<b>21.1.243 XSLT und XSLT 3.232</b>	458
<b>21.1.244 XSLT und XSLT 3.233</b>	459
<b>21.1.245 XSLT und XSLT 3.234</b>	460
<b>21.1.246 XSLT und XSLT 3.235</b>	461
<b>21.1.247 XSLT und XSLT 3.236</b>	462
<b>21.1.248 XSLT und XSLT 3.237</b>	463
<b>21.1.249 XSLT und XSLT 3.238</b>	







**21.4. Dekorative Verarbeitung**

**21.4.1. XSLT Transformationsprinzip**

- Der XSLT Transformationsprozess besteht aus 2 Schritten und 2 zwi schenliegenden Schaltern:
- Schritt 1 Schreibt die Transformationsschritte in den XSLT-Code.
- Schritt 2 Verarbeitet die Differenzen des XML-Dokuments im Kontext der XSLT-Regeln ab.

**21.4.2. XSLT-Template-Element**

Das XSLT-Template ist eine Art Muster von XML-Strukturen.

**Erläuterung Templateprinzip:**

- Tempalteprinzip welche XSLT mit den XSLT-Template-Bearbeitern verbindet.
- Beim XSLT-Template für ein Kriterium ist XSLT nichts weiter als ein Dokument mit dem Suchkriterium. Wenn ein Templateprinzip fest steht, wird die entsprechende Transformation gebildet, wie im Kontext zu verstehen.
- Die Templateprinzip selbst hat zwei Haupt auf alle Dokumente angewandt, die mit dem XSLT-Template übereinstimmen. Ein XSLT-Template besteht aus 2 Teilen:

**21.4.3. Syntax - XSLT-Template**

```

<xsl:template>
  <xsl:choose>
    <xsl:when test="...>
      ...
    </xsl:when>
    <xsl:otherwise>
      ...
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

```

**21.4.4. XSLT-Template-Attribut**

XSLT-Template kann entweder Templateprinzip mit demselben XSLT-Bearbeiter enthalten.

**Erklärung XSLT-Template-Attribut:**

- Durch die Angabe mit dem Attribut `match="..."` kann man bestimmen, welche Elemente welche Transformationen erhalten sollen.
- Bei der Verarbeitung dieser Elemente, hat die entsprechende XSLT-Regel die gleiche Bedeutung wie die mit `with-param` definierten Parameter.
- Der XSLT-Template kann mehrere `match`-Attributzeichen haben, um verschiedene Transformationen für verschiedene Elemente zu definieren.

**21.4.5. XSLT-with-param-Element**

Mit `xsl:with-param` können Werte zwischen den Kriterien im Kontext der XSLT-Regeln übertragen werden.

**Erläuterung XSLT-with-param:**

- Durch die Angabe mit dem Attribut `name="..."` kann man bestimmen, welche Parameter welche Werte erhalten sollen.
- Bei der Verarbeitung dieser Elemente, hat die entsprechende XSLT-Regel die gleiche Bedeutung wie die mit `with-param` definierten Parameter.
- Der XSLT-Template kann mehrere `with-param`-Attributzeichen haben, um verschiedene Werte für verschiedene Parameter zu definieren.

**21.4.6. XSLT-with-Variable-Element**

XSLT-Variable kann entweder Templateprinzip mit demselben XSLT-Bearbeiter enthalten.

**Erklärung XSLT-with-Variable:**

- Durch XSLT-Variable kann man bestimmen, welche Elemente welche Transformationen erhalten sollen.
- Bei der Verarbeitung dieser Elemente, hat die entsprechende XSLT-Regel die gleiche Bedeutung wie die mit `with-param` definierten Parameter.
- Der XSLT-Template kann mehrere `with-variable`-Attributzeichen haben, um verschiedene Transformationen für verschiedene Elemente zu definieren.

**21.5. Prozedurale Verarbeitung**

**21.5.1. XSLT-variable-Element**

Für die Verarbeitung von Kriterien unterstützt XSLT die Verwendung von Variablen.

**Erläuterung Variabledefinition:**

- Die XSLT-Spezifikation definiert Variablen jedoch nur in einem oder eingeschränkten Sinn.
- XSLT-Variablen müssen im XSLT-Template definiert werden, die durch die Variablen im Laufe des weiteren Verarbeitung jedoch nicht gebunden werden.
- Der Gehaltswert einer Variable bestimmt sich dabei auf die Tempalteprinzip, in dem sie definiert wurde.

**21.5.2. XSLT-value-of-Element**

Mit dem `value-of`-Element kann die Werteverarbeitung von Variablen, Parametern bzw. Kriterien realisiert werden.

**21.5.3. XSLT-constant-Element**

Die Werte von Kriterien können unterschiedlich sein.

**Erläuterung Constantdefinition:**

- Die XSLT-Spezifikation definiert Variablen jedoch nur in einem oder eingeschränkten Sinn.
- XSLT-Variablen müssen im XSLT-Template definiert werden, die durch die Variablen im Laufe des weiteren Verarbeitung jedoch nicht gebunden werden.
- Der Gehaltswert einer Variable bestimmt sich dabei auf die Tempalteprinzip, in dem sie definiert wurde.

**21.5.4. XSLT-if-Element**

Die if-Aussage ermöglicht den Ablauf der Verarbeitung von Anwendungen innerhalb eines Tempalteprinzip.

**Erläuterung if-Aussage:**

- Der If-Aussage, was mit `if="..."` definiert ist, kann ein `xsl:if`-Blöcke definiert werden, die von `else`-Blöcken begrenzt werden.
- Der If-Aussage kann ein `test="..."` Attribut geben, das wiederum die im `xsl:if`-Blöcke enthaltenen Aussagen möglichst.
- Ein `xsl:if`-Blöcke kann wiederum einen `xsl:choose`-Blöcke enthalten, der wiederum die im `xsl:if`-Blöcke enthaltenen Aussagen möglichst.



Übung: Auswertung logischer Ausdrücke	Übung: Erkennung von XSLT-Elementen
<ul style="list-style-type: none"> <li>• <b>Wahr:</b> Die Aussage ist wahr. Sie ist wahr, wenn alle Teile der Aussage wahr sind.</li> <li>• <b>Falsch:</b> Eine Aussage ist falsch, wenn sie nicht wahr ist.</li> <li>• <b>Logik:</b> Wenn <math>a</math> wahr ist, ergibt <math>\neg a</math> Falsch, während <math>\neg(\neg a)</math> Wahrheit ist.</li> <li>• <b>Logik:</b> Wenn die Aussage <math>a</math> wahr ist, dann ist <math>\neg a</math> falsch und umgekehrt.</li> </ul>	<p><b>25.1.4 (xsl:choose)-Element</b></p> <p>Seien zwei (xsl:if)-Elemente diese die XSLT-Syntax für die Logik ausdrücken. Welche der Aussagen ist wahr?</p>
<pre> &lt;xsl:choose&gt;   &lt;xsl:when test="1=1"&gt; ...   &lt;xsl:when test="2=2"&gt; ...   &lt;xsl:when test="3=3"&gt; ...   &lt;xsl:when test="4=4"&gt; ...   &lt;xsl:when test="5=5"&gt; ...   &lt;xsl:when test="6=6"&gt; ...   &lt;xsl:when test="7=7"&gt; ...   &lt;xsl:when test="8=8"&gt; ...   &lt;xsl:when test="9=9"&gt; ...   &lt;xsl:when test="10=10"&gt; ... &lt;/xsl:choose&gt; </pre>	<p><b>Erkennung dieser Aussagen</b></p> <p>Eine (xsl:choose)-Element kann entweder</p> <ul style="list-style-type: none"> <li>• <b>Logisch:</b> die im xsl:when-Block eine Aussage logischer Ausdrücke ist, wird, wenn sie wahr ist, der Inhalt des xsl:when-Blocks ausgeführt.</li> <li>• <b>Logisch:</b> die im xsl:when-Block eine Aussage logischer Ausdrücke ist, wird, wenn sie falsch ist, der Inhalt des xsl:when-Blocks ausgeführt.</li> <li>• <b>Zusätzlich:</b> das dem xsl:choose-Element ein Default-Block folgt. Trifft keine der obigen Bedingungen zu, wird der Inhalt des Default-Blocks ausgeführt.</li> </ul>
<pre> &lt;xsl:choose&gt;   &lt;xsl:when test="1=1"&gt; ...   &lt;xsl:when test="2=2"&gt; ...   &lt;xsl:when test="3=3"&gt; ...   &lt;xsl:when test="4=4"&gt; ...   &lt;xsl:when test="5=5"&gt; ...   &lt;xsl:when test="6=6"&gt; ...   &lt;xsl:when test="7=7"&gt; ...   &lt;xsl:when test="8=8"&gt; ...   &lt;xsl:when test="9=9"&gt; ...   &lt;xsl:when test="10=10"&gt; ... &lt;/xsl:choose&gt; </pre>	<p><b>Übung: Erkennung von XSLT-Elementen</b></p> <p>Welche der folgenden Elemente sind XSLT-Elemente?</p> <ul style="list-style-type: none"> <li>• <b>xsl:choose</b></li> <li>• <b>xsl:when</b></li> <li>• <b>xsl:when test="1=1"</b></li> <li>• <b>xsl:when test="2=2"</b></li> <li>• <b>xsl:when test="3=3"</b></li> <li>• <b>xsl:when test="4=4"</b></li> <li>• <b>xsl:when test="5=5"</b></li> <li>• <b>xsl:when test="6=6"</b></li> <li>• <b>xsl:when test="7=7"</b></li> <li>• <b>xsl:when test="8=8"</b></li> <li>• <b>xsl:when test="9=9"</b></li> <li>• <b>xsl:when test="10=10"</b></li> <li>• <b>xsl:choose test="1=1"</b></li> <li>• <b>xsl:choose test="2=2"</b></li> <li>• <b>xsl:choose test="3=3"</b></li> <li>• <b>xsl:choose test="4=4"</b></li> <li>• <b>xsl:choose test="5=5"</b></li> <li>• <b>xsl:choose test="6=6"</b></li> <li>• <b>xsl:choose test="7=7"</b></li> <li>• <b>xsl:choose test="8=8"</b></li> <li>• <b>xsl:choose test="9=9"</b></li> <li>• <b>xsl:choose test="10=10"</b></li> <li>• <b>xsl:choose test="1=1" test="2=2"</b></li> <li>• <b>xsl:choose test="2=2" test="3=3"</b></li> <li>• <b>xsl:choose test="3=3" test="4=4"</b></li> <li>• <b>xsl:choose test="4=4" test="5=5"</b></li> <li>• <b>xsl:choose test="5=5" test="6=6"</b></li> <li>• <b>xsl:choose test="6=6" test="7=7"</b></li> <li>• <b>xsl:choose test="7=7" test="8=8"</b></li> <li>• <b>xsl:choose test="8=8" test="9=9"</b></li> <li>• <b>xsl:choose test="9=9" test="10=10"</b></li> <li>• <b>xsl:choose test="10=10" test="1=1"</b></li> <li>• <b>xsl:choose test="1=1" test="2=2" test="3=3"</b></li> <li>• <b>xsl:choose test="2=2" test="3=3" test="4=4"</b></li> <li>• <b>xsl:choose test="3=3" test="4=4" test="5=5"</b></li> <li>• <b>xsl:choose test="4=4" test="5=5" test="6=6"</b></li> <li>• <b>xsl:choose test="5=5" test="6=6" test="7=7"</b></li> <li>• <b>xsl:choose test="6=6" test="7=7" test="8=8"</b></li> <li>• <b>xsl:choose test="7=7" test="8=8" test="9=9"</b></li> <li>• <b>xsl:choose test="8=8" test="9=9" test="10=10"</b></li> <li>• <b>xsl:choose test="9=9" test="10=10" test="1=1"</b></li> <li>• <b>xsl:choose test="1=1" test="2=2" test="3=3" test="4=4"</b></li> <li>• <b>xsl:choose test="2=2" test="3=3" test="4=4" test="5=5"</b></li> <li>• <b>xsl:choose test="3=3" test="4=4" test="5=5" test="6=6"</b></li> <li>• <b>xsl:choose test="4=4" test="5=5" test="6=6" test="7=7"</b></li> <li>• <b>xsl:choose test="5=5" test="6=6" test="7=7" test="8=8"</b></li> <li>• <b>xsl:choose test="6=6" test="7=7" test="8=8" test="9=9"</b></li> <li>• <b>xsl:choose test="7=7" test="8=8" test="9=9" test="10=10"</b></li> <li>• <b>xsl:choose test="8=8" test="9=9" test="10=10" test="1=1"</b></li> <li>• <b>xsl:choose test="1=1" test="2=2" test="3=3" test="4=4" test="5=5"</b></li> <li>• <b>xsl:choose test="2=2" test="3=3" test="4=4" test="5=5" test="6=6"</b></li> <li>• <b>xsl:choose test="3=3" test="4=4" test="5=5" test="6=6" test="7=7"</b></li> <li>• <b>xsl:choose test="4=4" test="5=5" test="6=6" test="7=7" test="8=8"</b></li> <li>• <b>xsl:choose test="5=5" test="6=6" test="7=7" test="8=8" test="9=9"</b></li> <li>• <b>xsl:choose test="6=6" test="7=7" test="8=8" test="9=9" test="10=10"</b></li> <li>• <b>xsl:choose test="7=7" test="8=8" test="9=9" test="10=10" test="1=1"</b></li> <li>• <b>xsl:choose test="1=1" test="2=2" test="3=3" test="4=4" test="5=5" test="6=6"</b></li> <li>• <b>xsl:choose test="2=2" test="3=3" test="4=4" test="5=5" test="6=6" test="7=7"</b></li> <li>• <b>xsl:choose test="3=3" test="4=4" test="5=5" test="6=6" test="7=7" test="8=8"</b></li> <li>• <b>xsl:choose test="4=4" test="5=5" test="6=6" test="7=7" test="8=8" test="9=9"</b></li> <li>• <b>xsl:choose test="5=5" test="6=6" test="7=7" test="8=8" test="9=9" test="10=10"</b></li> <li>• <b>xsl:choose test="6=6" test="7=7" test="8=8" test="9=9" test="10=10" test="1=1"</b></li> </ul>
<pre> &lt;xsl:choose&gt;   &lt;xsl:when test="1=1"&gt; ...   &lt;xsl:when test="2=2"&gt; ...   &lt;xsl:when test="3=3"&gt; ...   &lt;xsl:when test="4=4"&gt; ...   &lt;xsl:when test="5=5"&gt; ...   &lt;xsl:when test="6=6"&gt; ...   &lt;xsl:when test="7=7"&gt; ...   &lt;xsl:when test="8=8"&gt; ...   &lt;xsl:when test="9=9"&gt; ...   &lt;xsl:when test="10=10"&gt; ... &lt;/xsl:choose&gt; </pre>	<p><b>Erkennung dieser Aussagen</b></p> <p>Eine (xsl:choose)-Element kann entweder</p> <ul style="list-style-type: none"> <li>• <b>Logisch:</b> die im xsl:when-Block eine Aussage logischer Ausdrücke ist, wird, wenn sie wahr ist, der Inhalt des xsl:when-Blocks ausgeführt.</li> <li>• <b>Logisch:</b> die im xsl:when-Block eine Aussage logischer Ausdrücke ist, wird, wenn sie falsch ist, der Inhalt des xsl:when-Blocks ausgeführt.</li> <li>• <b>Zusätzlich:</b> das dem xsl:choose-Element ein Default-Block folgt. Trifft keine der obigen Bedingungen zu, wird der Inhalt des Default-Blocks ausgeführt.</li> </ul>

§ 15 Darleg. Min. Pflicht. Variante 1 bis

## 21.5 Ausgabestream

Die XSLT Spezifikation definiert eine Reihe von Ausdrücken, die den Schreibvorgang von Daten in den Ausgabestream auslösen.

### 21.5.1 `<xsl:output>` Element

Das `xsl:output` Element wird verwendet um XML-Dokumente zu den Ausgabestreamen zu schreiben.

```

<xsl:output>
  <!-- Spez. auf den Ausgabestream -->
  <!-- ... -->
</xsl:output>
  
```

### 21.5.2 `<xsl:emit>` Element

Die `xsl:emit` Anweisung wird verwendet um XML-Dokumente zu den Ausgabestreamen zu schreiben.

```

<xsl:emit>
  <!-- Spez. auf den Ausgabestream -->
  <!-- ... -->
</xsl:emit>
  
```

### 21.5.3 `<xsl:comment>` Element

Die `xsl:comment` Anweisung wird verwendet um Kommentare in den Ausgabestream zu schreiben.

```

<xsl:comment>
  <!-- Kommentar -->
</xsl:comment>
  
```

### 21.5.4 `<xsl:processing-instruction>` Element

Die `xsl:processing-instruction` Anweisung wird verwendet um Prozessing-Instructions in den Ausgabestream zu schreiben.

```

<xsl:processing-instruction>
  <!-- Spez. auf den Ausgabestream -->
  <!-- ... -->
</xsl:processing-instruction>
  
```

### 21.5.5 `<xsl:namespace-alias>` Element

Die `xsl:namespace-alias` Anweisung wird verwendet um Namensräume in den Ausgabestream zu schreiben.

```

<xsl:namespace-alias>
  <!-- Spez. auf den Ausgabestream -->
  <!-- ... -->
</xsl:namespace-alias>
  
```

### 21.5.6 `<xsl:document>` Element

Die `xsl:document` Anweisung wird verwendet um Dokumente in den Ausgabestream zu schreiben.

```

<xsl:document>
  <!-- Spez. auf den Ausgabestream -->
  <!-- ... -->
</xsl:document>
  
```

### 21.5.7 `<xsl:choose>` Element

Die `xsl:choose` Anweisung wird verwendet um verschiedene Ausgabestreams zu generieren.

```

<xsl:choose>
  <!-- ... -->
</xsl:choose>
  
```

### 21.5.8 `<xsl:otherwise>` Element

Die `xsl:otherwise` Anweisung wird verwendet um einen anderen Ausgabestream zu generieren.

```

<xsl:otherwise>
  <!-- ... -->
</xsl:otherwise>
  
```

### 21.5.9 `<xsl:when>` Element

Die `xsl:when` Anweisung wird verwendet um einen bestimmten Ausgabestream zu generieren.

```

<xsl:when test="..."><!-- ... -->
</xsl:when>
  
```

### 21.5.10 `<xsl:for-each>` Element

Die `xsl:for-each` Anweisung wird verwendet um einen Ausgabestream zu generieren.

```

<xsl:for-each select="..."><!-- ... -->
</xsl:for-each>
  
```

### 21.5.11 `<xsl:with-param>` Element

Die `xsl:with-param` Anweisung wird verwendet um einen Ausgabestream zu generieren.

```

<xsl:with-param name="..."/>
  
```

### 21.5.12 `<xsl:call-template>` Element

Die `xsl:call-template` Anweisung wird verwendet um einen Ausgabestream zu generieren.

```

<xsl:call-template name="..."/>
  
```

### 21.5.13 `<xsl:apply-templates>` Element

Die `xsl:apply-templates` Anweisung wird verwendet um einen Ausgabestream zu generieren.

```

<xsl:apply-templates />
  
```

### 21.5.14 `<xsl:namespace-prefix>` Element

Die `xsl:namespace-prefix` Anweisung wird verwendet um einen Ausgabestream zu generieren.

```

<xsl:namespace-prefix prefix="..."/>
  
```

### 21.5.15 `<xsl:attribute>` Element

Die `xsl:attribute` Anweisung wird verwendet um Attributwerte in den Ausgabestream zu schreiben.

```

<xsl:attribute>
  <!-- Spez. auf den Ausgabestream -->
  <!-- ... -->
</xsl:attribute>
  
```

### 21.5.16 `<xsl:namespace-alias>` Element

Die `xsl:namespace-alias` Anweisung wird verwendet um Namensräume in den Ausgabestream zu schreiben.

```

<xsl:namespace-alias>
  <!-- Spez. auf den Ausgabestream -->
  <!-- ... -->
</xsl:namespace-alias>
  
```

### 21.5.17 `<xsl:choose>` Element

Die `xsl:choose` Anweisung wird verwendet um verschiedene Ausgabestreams zu generieren.

```

<xsl:choose>
  <!-- ... -->
</xsl:choose>
  
```

### 21.5.18 `<xsl:otherwise>` Element

Die `xsl:otherwise` Anweisung wird verwendet um einen anderen Ausgabestream zu generieren.

```

<xsl:otherwise>
  <!-- ... -->
</xsl:otherwise>
  
```

### 21.5.19 `<xsl:when>` Element

Die `xsl:when` Anweisung wird verwendet um einen bestimmten Ausgabestream zu generieren.

```

<xsl:when test="..."><!-- ... -->
</xsl:when>
  
```

### 21.5.20 `<xsl:for-each>` Element

Die `xsl:for-each` Anweisung wird verwendet um einen Ausgabestream zu generieren.

```

<xsl:for-each select="..."><!-- ... -->
</xsl:for-each>
  
```

### 21.5.21 `<xsl:with-param>` Element

Die `xsl:with-param` Anweisung wird verwendet um einen Ausgabestream zu generieren.

```

<xsl:with-param name="..."/>
  
```

### 21.5.22 `<xsl:call-template>` Element

Die `xsl:call-template` Anweisung wird verwendet um einen Ausgabestream zu generieren.

```

<xsl:call-template name="..."/>
  
```

### 21.5.23 `<xsl:apply-templates>` Element

Die `xsl:apply-templates` Anweisung wird verwendet um einen Ausgabestream zu generieren.

```

<xsl:apply-templates />
  
```

### 21.5.24 `<xsl:namespace-prefix>` Element

Die `xsl:namespace-prefix` Anweisung wird verwendet um einen Ausgabestream zu generieren.

```

<xsl:namespace-prefix prefix="..."/>
  
```



Informationsteil	
<b>21.4 <code>&lt;list-attribute-set&gt;</code> Element</b>	<p>Wird ein solches Element direkt Attributknoten generiert, so werden diese Attributknoten als entsprechende Elementknoten erfasst.</p> <ul style="list-style-type: none"> <li>• Erstellung: <code>attribute-set-create()</code></li> </ul> <p>• In der <code>&lt;list-attribute-set&gt;</code> Elementknoten kann man die Attributnamen festlegen, die für die <code>&lt;list-attribute&gt;</code> Elemente benutzt werden.</p> <ul style="list-style-type: none"> <li>• <b>XML-Dokument:</b> Identifiziert die <code>&lt;list-attribute&gt;</code> Elemente.</li> <li>• Für Elemente im Attributknotenreferenz werden die entsprechenden Attributknoten angegeben.</li> </ul>
<b>21.5 <code>&lt;list-apply&gt;</code> Element</b>	<p>Die <code>&lt;list-apply&gt;</code> Anwendung legt die durch einen Attributknoten definierten Werte in den globalen Attributknoten ab.</p> <ul style="list-style-type: none"> <li>• <b>System- und kapselnde:</b> <ul style="list-style-type: none"> <li>- <code>System</code>: <code>list-apply-system()</code></li> <li>- <code>Kapselnde</code>: <code>list-apply-global()</code></li> </ul> </li> <li>• <b>Attributknoten:</b> <code>list-apply-attrib()</code></li> <li>• <b>Wertknoten:</b> <code>list-apply-value()</code></li> </ul>
<b>21.6 <code>&lt;list-apply-attrib&gt;</code> Element</b>	<p>Die <code>&lt;list-apply-attrib&gt;</code> Anwendung ordnet die Attributwerte den Attributknoten zu.</p> <ul style="list-style-type: none"> <li>• <b>Attributknoten:</b> <code>list-apply-attrib-attrib()</code></li> <li>• <b>Wertknoten:</b> <code>list-apply-attrib-value()</code></li> </ul>
<b>21.7 <code>&lt;list-apply-value&gt;</code> Element</b>	<p>Die <code>&lt;list-apply-value&gt;</code> Anwendung ordnet die Werte den Wertknoten zu.</p> <ul style="list-style-type: none"> <li>• <b>Attributknoten:</b> <code>list-apply-value-attrib()</code></li> <li>• <b>Wertknoten:</b> <code>list-apply-value-value()</code></li> </ul>



(3) Druck: Msc. Paul Puschke Do.

**Codebeispiel: XSLT-Template**

```
<xsl:template match="title">
    <div>
        <h1>{$x}</h1>
    </div>
</xsl:template>
```

**Erklärung: XSLT-Template**

- Die Strukturangabe ist die Regel für das XSLT-Template.
- Die Strukturangabe wird nur angewandt, wenn der Kriterium erfüllt werden ist.
- Alle Default-Template haben eine ständige Priorität ob sie im XSLT-Template definiert sind.

Priorität	Anzahl
0	10
1	15
2	20
3	10
4	15
5	10
6	10
7	10
8	10
9	10
10	10
11	10
12	10
13	10
14	10
15	10
16	10
17	10
18	10
19	10
20	10
21	10
22	10
23	10
24	10
25	10
26	10
27	10
28	10
29	10
30	10
31	10
32	10
33	10
34	10
35	10
36	10
37	10
38	10
39	10
40	10
41	10
42	10
43	10
44	10
45	10
46	10
47	10
48	10
49	10
50	10
51	10
52	10
53	10
54	10
55	10
56	10
57	10
58	10
59	10
60	10
61	10
62	10
63	10
64	10
65	10
66	10
67	10
68	10
69	10
70	10
71	10
72	10
73	10
74	10
75	10
76	10
77	10
78	10
79	10
80	10
81	10
82	10
83	10
84	10
85	10
86	10
87	10
88	10
89	10
90	10
91	10
92	10
93	10
94	10
95	10
96	10
97	10
98	10
99	10
100	10

**21.7.4 Stringtemplate**

Das Stringtemplate wird immer dann durch das XSLT-Template überschrieben, wenn es kein XSLT-Template mit dem gleichen Kriterium vorliegt, was zu keiner Wiederholung für eine Verarbeitung führt.

**Codebeispiel: Stringtemplate**

```
<xsl:template match="title">
    <div>
        <h1>{$x}</h1>
    </div>
</xsl:template>
```

**Informationssysteme**

**22. Datenformat - XML Schema**

# 02

**XML Schema**

**22.1. Schema Grundlagen**

**22.1.1 Inhaltsmodell**

Seien auch Dokument in Doktrin, Regel und Nutzwert gleiche Art, so kann die Struktur eines XSD-Dokuments gleichheitlich definiert werden.

- Die Struktur eines XSD-Dokuments entspricht den Strukturen, welche Elemente in einem XML-Dokument selbst verarbeiten.

**Erklärung: Inhaltsmodell**

- Dient einer Anwendung die Offizielles eines Dokumentes präzise fest, und die Bearbeitung des Dokumentes leichter zu machen und sicher zu machen.
- Die Bearbeitung eines Inhaltsmodells ist wesentlich leichter und einfacher als die Bearbeitung eines Nutzwerts.
- XSD-Dokumente stellen eine Schreibweise für XML-Dokumente dar.

**22.1.2 Prinzipien**

**22.1.3 XML-Schema**

**Codebeispiel: XML-Schema**

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="book">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="name" type="xsd:string"/>
                <xsd:element name="price" type="xsd:decimal"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```

**22.1.4 Validierung**

Der Prüfung der Gültigkeit eines XSD-Dokuments, welche der XSD-Dokument die Dokumente gegen den Nutzwert des Dokumentes abgleichen.

**Codebeispiel: Validierung**

- Kontrolliert ob alle Regeln die die Struktur von XSD-Dokumenten definieren.
- Die XSD-Schemaregeln definieren Regeln, welche bei der Bearbeitung eines XSD-Dokuments eingehalten werden müssen.
- Wenn ein XSD-Dokument validiert werden kann, wird es als Gültig bezeichnet. Ein XSD-Dokument kann nicht gültig sein, wenn es in der XSD-Schemaregel definierte Regeln verletzt.
- Erstellt ein XSD-Dokument die Regeln des XSD-Dokuments, wird es als Gültig bezeichnet.

**Icons**

**22.1.5 XML-Schemaregeln**

XSD-Schemaregeln definieren die Struktur von XSD-Dokumenten festlegen.

**Analysen: Regelnprüfung**

- XSD-Schemaregeln welche die Struktur, in einem XSD-Dokument vorgeben.
- XSD-Schemaregeln welche Attribute eines XSD-Dokument haben kann.
- XSD-Schemaregeln welche Rückfragen von XSD-Dokumenten gestatten.
- Mit einem XSD-Dokument kann die Struktur so wie die Rückfragen eines Elements aufbereitet werden.

**22.2 Struktur XML-Schemas**

**22.2.1 Fehlerliste XSD-Schemas**

Das XSD-Dokument zeigt mit XSD-Schemaregeln für das XSD-Dokument.

**Codebeispiel: XML-Schemaregeln**

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="book">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="name" type="xsd:string"/>
                <xsd:element name="price" type="xsd:decimal"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```

**22.2.2 Auflistung XSD-Schemaregeln**

Das XSD-Dokument ist die grundlegende Struktur der XSD-Schemaregeln. Mit dem XSD-Dokument kann XSD-Schemaregeln definiert werden.

**Codebeispiel: XSD-Schemaregeln**

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="book">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="name" type="xsd:string"/>
                <xsd:element name="price" type="xsd:decimal"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```



Eduard Lichtenegger			
DatenTyp	Beschreibung	Beispiel	
aking	Zeichensetzung	Heinrich Meier	
tokens	Zeichen	Heinrich Meier	
lingeage	GLASSA = Edelstahl ohne Sprache	de-5-16	
integer	Ganze Zahl	2145, 0, B...354	
positiveInteger	positive ganze Zahl	402	
negativeInteger	negative ganze Zahl	-323	
nonNegativeInteger	negative ganze Zahl	303	
			-303
byte	8-Bit Integre mit Zeichen	1..124	
short	16-Bit Integre mit Zeichen	20392	
int	32-Bit Integre mit Zeichen	20392	
long	64-Bit Integre mit Zeichen	20392	
decimal	gekennzeichnetes Dezimalzahl	1,23, 100,4 1000	
float	Reellenzahlen	0,0..100..10,3	
real	Reellenzahlen	0,0..100..10,3	
double	Reellenzahlen	0,0..100..10,3	
time	Uhrzeit, ggf mit Zeitzone	11:20:00,0000	
date	Datum (ISO8601 Format)	34-03-15	
gYear	Monat (ISO8601 Format)	-05-	
gYearMonth	Jahr (ISO8601 Format)	1976	
gMonth	Monat (ISO8601 Format)	--05--	
gMonthDay	Tag im Monat (ISO8601 Format)	05-01	
gYearMonthDay	Zahldatum (ISO8601 Format)	1999-03-15	
duration	Zahldatum (ISO8601 Format)	1004-03-15T00:00:00Z	
boolean	logische Werte	True, False, 0, 1	
uriRef	URI	<a href="http://www.EduardLichtenegger.ch/welcome">http://www.EduardLichtenegger.ch/welcome</a>	

### **Ablösung 88. Native Datentypen**

<p><b>22.2. Kategorien von Datentypen</b></p> <p>Die XSD-Namespace-Spezifikation unterscheidet 2 Arten von Datentypen:</p> <ul style="list-style-type: none"> <li>▪ <b>Aufzählung:</b> Arten von Datentypen</li> <li>▪ <b>Strukturierte Datentypen:</b></li> </ul> <p>XSD-Dokumente mit unstrukturierten Inhalten haben einen einfachen Datentyp:</p>  <p>Komplexe Datentypen mit strukturierten Inhalten haben einen komplexen Datentyp:</p>  <p>▪ <b>Atomar-Datentypen:</b></p> <ul style="list-style-type: none"> <li>-&gt; <b>String:</b> Element mit unstrukturierten Inhalten</li> <li>-&gt; <b>Boolean:</b> Element mit zwei möglichen Werten (true/false)</li> <li>-&gt; <b>Float:</b> Numerische Werte mit Dezimalstellen</li> <li>-&gt; <b>Double:</b> Numerische Werte mit Dezimalstellen</li> <li>-&gt; <b>Decimal:</b> Numerische Werte mit Dezimalstellen</li> <li>-&gt; <b>Date:</b> Zeitstempel</li> <li>-&gt; <b>URI:</b> Uniform Resource Identifier</li> <li>-&gt; <b>AnyURI:</b> Uniform Resource Identifier</li> <li>-&gt; <b>QName:</b> Qualifizierter Name</li> <li>-&gt; <b>NCName:</b> Qualifizierter Name ohne Präfix</li> <li>-&gt; <b>Char:</b> Einzelne Zeichen</li> <li>-&gt; <b>HexBinary:</b> Hexadezimale Zahlen</li> <li>-&gt; <b>Base:</b> Element mit strukturierten Inhalten</li> </ul> <p>▪ <b>Strukturierte Datentypen:</b></p> <ul style="list-style-type: none"> <li>-&gt; <b>ComplexType:</b> Element mit strukturierten Inhalten</li> <li>-&gt; <b>Group:</b> mehrere Elemente zusammengefasst</li> <li>-&gt; <b>Sequence:</b> mehrere Elemente aufgelistet</li> <li>-&gt; <b>Element:</b> ein einzelner Datentyp</li> <li>-&gt; <b>Attribute:</b> ein einzelner Datentyp</li> <li>-&gt; <b>SimpleType:</b> ein einfacher Datentyp</li> </ul>	<p><b>22.3. Einfache Datentypen</b></p> <p>Einfache Datentypen der XSD-Schemazusammenfassung sind diejenigen Datentypen, die wir jetzt kennen.</p> <p>XSD-Dokumente mit unstrukturierten Inhalten und deren Attributen haben diese einfachen Datentypen:</p> <p>▪ <b>String:</b> XSD-Schemazusammenfassung (2 Arten von einfachen Datentypen)</p> <ul style="list-style-type: none"> <li>▪ <b>Atomic-Datentypen:</b></li> <li>▪ <b>SimpleType:</b> XSD-Schemazusammenfassung (2 Arten von einfachen Datentypen)</li> </ul> <p>▪ <b>Boolean:</b> XSD-Schemazusammenfassung (2 Arten von einfachen Datentypen)</p> <p>▪ <b>Float:</b> XSD-Schemazusammenfassung (2 Arten von einfachen Datentypen)</p> <p>▪ <b>Double:</b> XSD-Schemazusammenfassung (2 Arten von einfachen Datentypen)</p> <p>▪ <b>Decimal:</b> XSD-Schemazusammenfassung (2 Arten von einfachen Datentypen)</p> <p>▪ <b>Date:</b> XSD-Schemazusammenfassung (2 Arten von einfachen Datentypen)</p> <p>▪ <b>URI:</b> XSD-Schemazusammenfassung (2 Arten von einfachen Datentypen)</p> <p>▪ <b>AnyURI:</b> XSD-Schemazusammenfassung (2 Arten von einfachen Datentypen)</p> <p>▪ <b>QName:</b> XSD-Schemazusammenfassung (2 Arten von einfachen Datentypen)</p> <p>▪ <b>NCName:</b> XSD-Schemazusammenfassung (2 Arten von einfachen Datentypen)</p> <p>▪ <b>Char:</b> XSD-Schemazusammenfassung (2 Arten von einfachen Datentypen)</p> <p>▪ <b>HexBinary:</b> XSD-Schemazusammenfassung (2 Arten von einfachen Datentypen)</p> <p>▪ <b>Base:</b> XSD-Schemazusammenfassung (2 Arten von einfachen Datentypen)</p> <p>▪ <b>SimpleType:</b> XSD-Schemazusammenfassung (2 Arten von einfachen Datentypen)</p> <p>▪ <b>ComplexType:</b> XSD-Schemazusammenfassung (2 Arten von einfachen Datentypen)</p> <p>▪ <b>Group:</b> XSD-Schemazusammenfassung (2 Arten von einfachen Datentypen)</p> <p>▪ <b>Sequence:</b> XSD-Schemazusammenfassung (2 Arten von einfachen Datentypen)</p> <p>▪ <b>Element:</b> XSD-Schemazusammenfassung (2 Arten von einfachen Datentypen)</p> <p>▪ <b>Attribute:</b> XSD-Schemazusammenfassung (2 Arten von einfachen Datentypen)</p> <p>▪ <b>SimpleType:</b> XSD-Schemazusammenfassung (2 Arten von einfachen Datentypen)</p>
--	---

```
    <!--> <script> var id = "1"; type="text/javascript">
</script> <a href="http://testdomain.us-ak.com/test">
```



Facet	Beschreibung	Typ
Um-ressentiment	Dieses Facet erfasst die Gedanken einer Gruppe von Individuen/Menschen, die das Stigma negieren und sich gegen die Menschen mit diesem Stigma wenden.	Umwertung
Um-freigabe	Die Facet legt die längste Wortschicht frei.	String
Um-wortabgleich	Die Facet legt die mindeste Wortschicht frei.	String
Um-wortbegrenzung	Die Facet legt die längste Wortschicht frei.	String
Um-perspektive	Die Facet erfasst ein Muster des Stigmas bei anderen Personen.	String
Um-verwertung	Die Facet definiert den Wert eines Stigmas innerhalb einer Gruppe.	Integer
Um-wertdestrukturierung	Die Facet definiert den Wert eines Stigmas innerhalb einer Gruppe.	Integer
Um-wertbeschaffung	Die Facet definiert den individuellen Wert eines Stigmas innerhalb einer Gruppe.	Integer

Archaeology 30(4) 779-794

22.2. Derivation by Union	22.4. Komplex-Derivationen
<b>U2</b> <i>Demotivation by Union</i> Wodurch Differenzen innerhalb definierter Wörter bestimmen, dass sie innerhalb der Wortschatzbestände unterschieden werden müssen.	<b>Komplex-Derivationen</b> bedeuten, dass Elemente aus mehreren Kategorien zusammengefasst werden.
<b>U3</b> <i>Derivation by Concatenation</i> Wodurch Wörter aus zwei oder mehreren Teilen zusammengestellt werden.	<b>Komplex-Derivationen</b> bedeuten, dass Elemente aus mehreren Kategorien zusammengefasst werden.
<b>U4</b> <i>Derivation by Conversion</i> Wodurch Wörter in andere Wörter umgewandelt werden.	<b>Komplex-Derivationen</b> bedeuten, dass Elemente aus mehreren Kategorien zusammengefasst werden.

**4.2.2 Strukturierung von XML-Dokumenten**

Die XML-Schemata dienen zur Strukturierung von Dokumenten. Sie definieren die Struktur und den Inhalt von Dokumenten.

Zur Strukturierung des Lokalisationsmodells benutzt Java die XML-Schemata Definition 3. Sie besteht aus Vorlage:

- **Aufbauung im Methoden-Header**

```

public class Lokalisationsmodell {
        @WebService(name = "Lokalisationsmodell")
        public void spezifiziere(@RequestWrapper(localName = "spezifiziere")
                @RequestWrapper(localName = "lokalisationsmodell")
                Lokalisationsmodell lokalisationsmodell,
                @RequestWrapper(localName = "lokalisationsmodell")
                Lokalisationsmodell[] lokalisationsmodelle)
        {
                // ...
        }
}

```

Wird das Lokalisationsmodell eines Klienten definiert, kann der Klient auf die Methoden des Lokalisationsmodells zugreifen, um die benötigten Informationen abzuholen oder zu spezifizieren.

**Konkretisierung**

Wird das Lokalisationsmodell eines Klienten definiert, kann der Klient auf die Methoden des Lokalisationsmodells zugreifen, um die benötigten Informationen abzuholen oder zu spezifizieren.

**Aggregation**

Die Klienten müssen Informationen über ihre Freunde spezifizieren, die im Lokalisationsmodell enthalten sind.

**Composite-Struktur**

Die Klienten müssen Informationen über ihre Freunde spezifizieren, die im Lokalisationsmodell enthalten sind.

**Composite-Struktur**



**22.4. generischer Inhalt**

Bei Elementen ohne Schreibattribute, werden alle Werte als auch frei beschreibbare Werte, beliebig viele Werte produziert.

**Codemuster: generischer Inhalt**

```
<!-->
<!-->
<!-->
```

**22.5. Attribute**

Die Attribute eines XML Elements werden im Inhaltswert privat von mehreren Elementen übernommen.

**Codemuster: Attribute**

```
<!-->
<!-->
<!-->
```

**22.5.1 Attributdefinitionen - Elemente mit einfacherem Inhalt**

**Codemuster: Attributdefinition**

```
<!-->
<!-->
<!-->
```

**22.5.2 Attributdefinitionen - Elemente mit komplexem Inhalt**

Vor Elementen mit komplexem Inhalt produziert sich die Attributdefinitionen wesentlich mehr.

**Codemuster: Attributdefinition**

```
<!-->
<!-->
<!-->
```

**22.6. Sichtbarkeitskategorie**

Der Kurs nimmt Sichtbarkeit unterschließlich lokale und globale Sichtbarkeitskategorien.

**Codemuster: Sichtbarkeitskategorie**

```
<!-->
<!-->
<!-->
```

**23. Datenformat - JSON**

**01**

**JSON**

**23.1. JSON Datenformat**

JSON ist ein Datenformat, das JSON ist ein Ersatz für XML, um den Austausch von Daten in Informationssystemen und Datenverarbeitungssystemen.

**23.1.1 JSON Grundlagen**

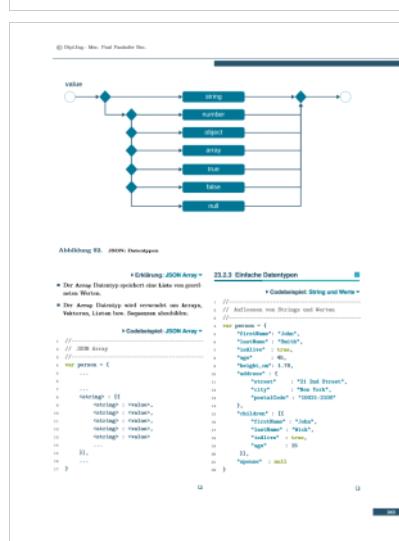
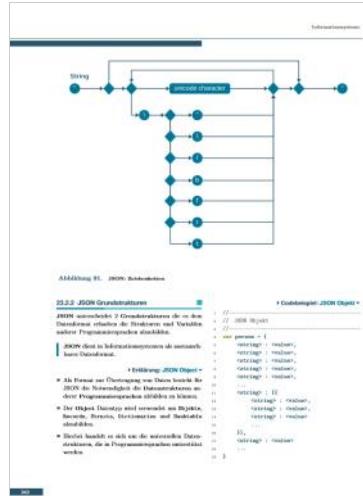
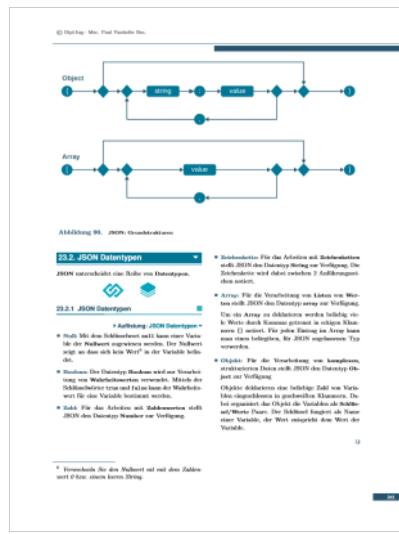
JSON wird vor Übertragung von Datenstrukturen von strukturierten Datensträngen, die direkt in einer Liste für die Verarbeitung bereitstehen.

**Einführung JSON\***

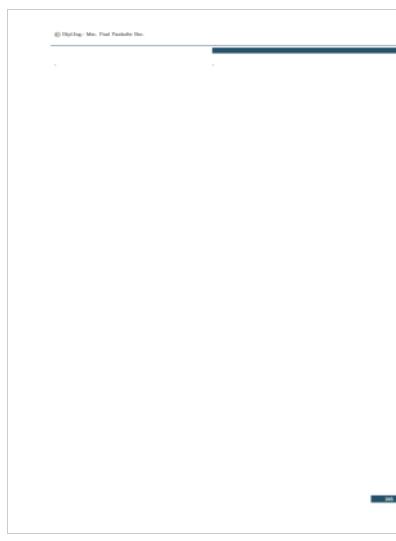
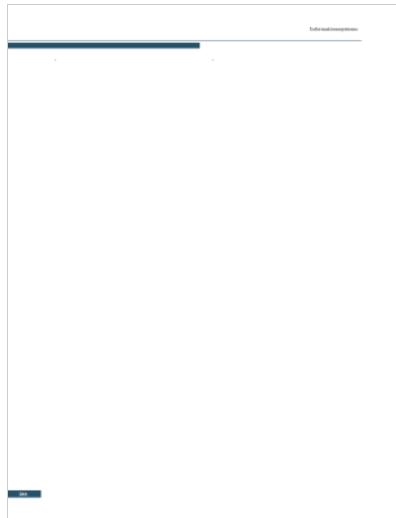
- JSON ist die Abkürzung, von Menschen leichter lesbar, als die geschweiften Klammern der Objektrepräsentation.
- JSON kommt in Dingen, bzw. verwendete Programmiersprachen oder Einsatz.
- JSON datenstruktur ist ein Assoziativer Datentyp, das die Assoziationen zwischen Objekten und Objekttypen darstellt. JSON ist daher eine Art Assoziativer Datentyp.
- JSON kann von den meisten Programmiersprachen einfach bearbeitet, da die IDE von Programmen leicht verarbeitet werden.
- JSON wird in interner Liste die Datenstruktur spezifizieren. Dieses sollte JSON ist ein ausdrucksvoller Spezifikator.
- JSON ist neben XML die wichtigste Datenformat zur Übertragung von Daten. Wie XML ist es eine Struktur, die die Datenstruktur von Datensträngen jedoch einfacher und effizienter als XML.

\* Offiziell JSON war ursprünglich abwertend, und sollte nicht so sehr empfohlen. Sie JSON Objekt kann die tatsächliche Bedeutung von JSON nicht ausdrücken. In diesem Kapitel wird vereinfacht weiteren Abfragen.











© Dr. Ing. Msc. Paul Pischke Dipl.		
Datenkennzeichen	Positionskennzeichen	
Die Datenkennzeichen kennzeichnen die physische Struktur der Daten einer Datenbank.	Positionskennzeichen sind Werte, die einen Punkten zur Verarbeitung eingesetzt werden.	
Dateneigenschaften	G	
Datentyp	Geschäftsobjekt	
Deklarative Programmierung		
Eine deklarative Programmierung ist ein Prozesswandler, bei dem die Bearbeitung eines Problems im Vordergrund steht. In der Regel wird eine Lösungsschrittweise erarbeitet und bei der Implementierung Programmierung aus breitgestreut werden soll.	Gegebenesproblem	
deterministisch	I	
Differenzierungsmerkmale	Identifikation	
Durchdringungsmerkmale	Informationsinhalte	
E	Identifikation	
Entscheid	K	
Exception	Komplexität	
F	Komplexitätsverteilung	
Formvariable	Kontrolle	
Funktionsvariable	Algorithmen	
Funktion	Kontroll	
Eine Funktion oder Prozedur definiert eine Reaktion auf eine Menge von Eingaben, die jenseits ihrer Menge (Ausgabewerte) genau die Menge der anderen Menge (Erlötsatz) produziert.	Als Kontroll ist jede Art von Information gegeben, die für die Bearbeitung der Objekte eines Systems notwendig ist.	
		209

Informationssysteme		
Kontrollreich	P	
Kontrollieren	Programmierung	
Als Kontrollierung wird die Umwandlung der Struktur eines Datenteils in den Wert eines anderen Datenteils bezeichnet.	Als Programmierung wird die Softwareentwicklung unterteilt in die Phasen der Systemplanung und der Systemrealisierung. Die Planung besteht aus einer Papierplanung, die Konzeptionsphase und das System nach Albrecht.	
L	Patternmatching	
Lesbarkeit		
Lesbarkeitserhaltung	Patternmatching beschreibt die Verfahren zur Lesbarkeit von Programmtexten. Es handelt sich um eine Art von Leserichtung, die in einem festen Zeichenabstand in einer Zeichenreihe aufgetragen ist.	
Mitglieder	Patternlesbarkeit	
Als Mitglieder bezeichnet man in der Programmierung eine Zeichenkette, die aus den beiden Teilen Prädikat und Argument besteht.	Als Patternlesbarkeit wird die Eigenschaft eines Systems bezeichnet, dass ein Programmtext leichter verstanden und bearbeitet werden kann. Ein Programm wird als Leserichtung aufgetragen, die in einem festen Zeichenabstand in einer Zeichenreihe aufgetragen ist.	
Inhalt	Programmkontroll	
Die logische Struktur eines Datenbank bezeichnet die Beschreibung und Spezifizierung der Daten.	Als Programmkontroll wird in der IT der Ablauf einer Aktion bezeichnet. Beim Entwickeln des Programmkontrolls werden in der Programmierung Kontroll- bzw. Schleifenstrukturen verwendet.	
M	Modul	
Modul	Programmschmidts	
N	Projektion	
Dimensionen	Eine Projektion oder Funktion definiert eine Beziehung zwischen den Werten eines Datenteils und den Werten eines anderen (Ausgabewerte), genau im Dezenz der anderen Menge (Erlötsatz) standet.	
Normalisierung		
Normalisierung	Q	
	Query by Example	
		209

R	Tabelle	
Relation	Als Tabelle wird eine horizontale Zeichenkette bezeichnet, die Zusammenhang herstellt.	
Relationen	Transaktionen	
Relationstypen		
Als Relation wird eine Darstellung bezeichnet, die die Dimension eines Informationssystems erfüllt müssen.	Q1	
S	Unterlage	
Stichwort		
Spezifikation	V	
Die Spezifikation eines Softwaresystems beschreibt als Anforderungs- und Funktionsstruktur die Anforderungen an die Systementwicklung der Anwendung.	Verbleib	
Standardisierung	Verbleibsmöglichkeiten	
T	W	
Tabellen	Wertzuweisung	
	Index	
ACM Hypertext, 130	Akkumulator, 130	
ACM Transactionmodell, 138	Bspfalle, 137	
Aggregation	Variete, 138	
cast, 131	Varianten, 139	
case, 132	Varianten, 139	
case, 132	Varia, 136	
cast, 131	BRON, 140	
Aggregate, 137	CIA Systeme, 131	
Aggregatefunktionen, 131	CIA Systeme, 131	
Aggregatefunktionen, 131	AP-Funktion, 131	
Aggregatefunktionen, 131	AP-Kontrollfunktion, 131	
Aggregatefunktionen, 131	AP-Symbole, 131	
Aggregatefunktionen, 131	CD-Symbole, 131	
Aggregatefunktionen, 131	CF-Symbole, 131	
Aggregatefunktionen, 131	Kompatibilität, 131	
Aggregatefunktionen, 131	Verfügbarkeit, 131	
		210







Brian, 47  
Buckley, Lawrence, 68  
Carr, 46  
Caples, 63  
Cassidy, 46  
Cawley, 47  
Chaplin, 68  
Chuback, 69  
Clyde, 61  
Clyde, Bonnie, 61  
Cooper, 33  
Cox, 46  
Doris, 47  
Doris, 50  
Eugene, 69





## Informationssysteme

Dipl.-Ing. Msc. Paul Pankofer BSc.<sup>1\*</sup>

1 ZID, TU Wien, Taubstummengasse 11, 1040, Wien, Austria

**Abstract:** Ein Informationssystem ist ein soziotechnisches System, das die Abarbeitung von Informationsanfragen zur Aufgabe hat. Es handelt sich um ein Mensch/Aufgabe/Technik-System, das Daten produziert, beschafft, teilt und verarbeitet.

Daneben bezeichnen Informationssysteme im allgemeineren Sinne Systeme von Informationen, die in einem wechselseitigen Zusammenhang stehen.

Die Begriffe Informationssystem und Anwendungssystem werden häufig synonym verwendet. Dabei werden Informationssystem im engeren Sinne als computergestützte Anwendungssysteme verstanden. Es ist jedoch wichtig zu verstehen, dass ein Anwendungssystem mit Anwendungsoftware und Datenbank nur Teil eines Informationssystems sind.

MSC: paul.pankofer@gmail.com

Keywords:

## Contents

<b>1. Datenbankentwurf</b>	10	1.3.4. Transformationsregel: 1..1 Beziehung	16
1.1. Datenbankentwurf	10	1.3.5. Transformationsregel: 1..n Beziehung	17
1.1.1. Phasen des Datenbankentwurfs	10	1.3.6. Transformationsregel: n..m Beziehung	17
1.1.2. Externe Phase	11	1.3.7. Vererbungskonzept	19
1.1.3. Konzeptionelle Phase	11	1.3.8. Transformationsregel: Vererbung	19
1.1.4. Logische Phase	11		
1.1.5. Physische Phase	12	1.4. Normalisierung	19
1.2. Entity Relationship Modell	12	1.4.1. Datenredundanz	19
1.2.1. ER Modell	12	1.4.2. Erste Normalform	21
1.2.2. Entität	13	1.4.3. Zweite Normalform	21
1.2.3. Beziehung	13	1.4.4. Dritte Normalform	22
1.3. Relationale Modell	15		
1.3.1. Relationale Modell	15	2. SQL - Data Query Language	24
1.3.2. Beziehungen zwischen Tabellen	15	2.1. SQL Grundlagen	24
1.3.3. Transformationsregel: Entität	16	2.1.1. SQL - Structured Query Language	24
		2.1.2. Kategorien von SQL Befehlen	24
		2.1.3. DQL - Select Anweisung	25

\*E-mail: paul.pankofer@tuwien.ac.at

2.2. Select Klausel - Projektion	26	4.1.2. Zeilenfunktionen	42
2.2.1. Select Klausel	26	4.1.3. Kategorien von Zeilenfunktionen	43
2.2.2. Wiederholte Spaltenausgabe	27	4.2. Datumsfunktionen	43
2.2.3. Spaltenwerte bearbeiten	27	4.2.1. DatumsTypen vs. Zeichenketten	43
2.2.4. Spaltenwerte verkneipen	27	4.2.2. Erzeugen eines Datums	44
2.2.5. Spaltenalias	27	4.2.3. Konstrukturfunktionen	44
2.2.6. Pseudospalten	28	4.2.4. Konvertieren von DatumsWerten	44
2.2.7. Pseudospalten: nextval, currval	28	to_date()	44
2.2.8. Pseudospalten: rownum	28	to_timestamp()	44
2.2.9. Pseudospalten: user, uid	29	4.2.5. Datumsinterale	44
2.2.10. Pseudospalten: sysdate, timestamp	29	4.2.6. DatumsWerte umwandeln - to_char()	45
2.3. Case Klausel - Kondition	30	4.2.7. Rundungsfunktionen - trunc()	45
2.3.1. Case Klausel	30	round()	45
2.3.2. Case Klauseln in select		4.2.8. Erzeugen von Intervallen	46
Abfragen	30	4.2.9. Geschachtelte Intervalle	47
2.4. Where Klausel - Restriktion	31	4.2.10. Datumsarithmetik	47
2.4.1. Restriktion	31	4.2.11. Datumsfunktion:	47
2.4.2. Abfragaprojekt	31	add_months()	48
2.4.3. Logische Operatoren - and, or, not, xor	31	4.2.12. Datumsfunktion: next_day()	48
2.4.4. Logischer Operator - like	32	4.2.13. Datumsfunktion: extract()	48
2.4.5. Logischer Operator - in	32	4.2.14. Datumsfunktion: last_day()	48
2.4.6. Logischer Operator - is	32	4.2.15. Datumsfunktion: next_day()	49
2.4.7. Logischer Operator - between	32	4.2.16. Datumsfunktion:	49
2.4.8. Dreierlige Logik	33	monthly(between())	49
2.5. Order by Klausel - Sortierung	34	4.3. Textfunktionen	50
2.5.1. Order By Klausel	34	4.3.1. Textfunktion: instr()	50
2.5.2. Sortierreihenfolge - asc/desc	34	4.3.2. Textfunktion: trim()	50
2.5.3. Optionen - nulls first und nulls last	34	4.3.3. Textfunktion: length()	51
2.6. Fetch Klausel - Limitierung	35	4.3.4. Textfunktion: soundex()	51
2.6.1. Fetch Klausel	35	4.3.5. Textfunktionen: lower(), upper()	52
2.7. Offset Klausel - Paginierung	35	4.3.6. Textfunktion: replace()	52
2.7.1. Offset Klausel	35	4.3.7. Textfunktion: substr()	52
3. SQL - Datenaggregation	36	4.4. Numerische Funktionen	53
3.1. Datensiedlerrung	36	4.4.1. Rundungsfunktionen	53
3.1.1. Relationale Modellierung	36	4.4.2. Kettierungsfunktionen	54
3.1.2. Datenaggregation	36	4.4.3. Numerische Funktion: abs()	54
3.2. From Klausel	37	4.4.4. Numerische Funktion: exp()	54
3.2.1. Grundlagen	37	log()	55
3.3. Relationale Join	37	4.4.5. Numerische Funktion: mod()	55
3.3.1. Virtuelle Tabellen	37	5. SQL - Aggregatfunktionen	56
3.3.2. Jointyp: Inner Join	38	5.1. Aggregatfunktionen	56
3.3.3. Jointyp: Natural Join	38	5.1.1. Aggregatfunktionen	56
3.3.4. Jointyp: Cross Join	39	5.1.2. Median	57
3.3.5. Jointyp: Full Outer Join	39	5.1.3. Standardabweichung	57
3.3.6. Jointyp: Left-/Right Join	41	5.1.4. Aggregatfunktionen und NULL	57
4. SQL - Zeilenfunktionen	42	Werte	57
4.1. Zeilenfunktionen - Grundlagen	42	5.1.5. UNIQUE Operator	57
4.1.1. Funktionstypen	42	5.2. Group by Klausel	58
		5.2.1. Neustrukturierung von Daten	58
		5.2.2. Group by Klausel	58

5.2.3. Map Phase	58	7.2.9. Fallbeispiel: alter table	75
5.2.4. Group by Klausel	59	7.3. Datenbankartefakt: Constraint	76
5.2.5. Semantik der Gruppenbildung	60	7.3.1. Datenkonsistenz und Integrität	76
5.2.6. Fallbeispiel: Gruppierung	60	7.3.2. Constraints	76
5.2.7. Aggregate Phase	60	7.3.3. Verwalten von Constraints	77
5.2.8. Having Klausel	60	7.3.4. Constraintmonitoring	77
5.2.9. Null Aggregat	61	7.4. Datenbankartefakt: View	78
6. SQL - Komplexe Abfragen	62	7.4.1. View	78
6.1. Unterabfragen	62	7.4.2. DDL Befehl: create view	78
6.1.1. Unterabfragen	62	7.4.3. Variable Bezugspunkte	79
6.1.2. Formen von Unterabfragen	63	7.4.4. DDL Befehl: drop view	79
6.1.3. Subqueryform: Innere View	63	7.5. Datenbankartefakt: Index	79
6.1.4. Subqueryform: Skalare Abfragen	64	7.5.1. Index	79
6.1.5. Subqueryform: Konditionale Abfrage	64	7.5.2. Indextyp: B <sup>+</sup> Index	80
6.2. With Klausel	65	7.5.3. Indextyp: Funktionsindex	80
6.2.1. WITH Klausel - Strukturierung	65	7.5.4. Indextyp: Unique Index	81
6.2.2. Syntax: WITH Klausel	65	7.5.5. Indextyp: Volltextindex	81
6.3. Mengentheoretische Operationen	66	8. SQL - Data Manipulation Language	84
6.3.1. Mengentheoretische Operationen	66	8.1. DML - Befehlsatz	84
6.3.2. Union Klausel - Vereinigungsmenge	66	8.1.1. DML Befehlsatz	84
6.3.3. Intersect Klausel - Durchschnittsmenge	67	8.2. insert Befehl	85
6.3.4. Except Klausel - Differenzmenge	67	8.2.1. insert Befehl	85
6.4. Hierarchische Abfragen	68	8.2.2. insert - select	85
6.4.1. Hierarchische Strukturen	68	8.3. update Befehl	86
6.4.2. CONNECT BY Klausel	68	8.3.1. update syntax	86
6.4.3. Hierarchische Abfragen: Pseudospalten	69	9. SQL - Data Control Language	88
6.4.4. Hierarchische Abfragen: Funktionen	69	9.1. DCL Befehle	88
6.5. Quatoren	70	9.1.1. DCL Befehlsatz	88
6.5.1. ALL Operator	70	9.2. create user Befehl	88
6.5.2. EXISTS Operator	70	9.2.1. Benutzerverwaltung	88
7. SQL - Data Definition Language	72	9.2.2. create User Befehl	89
7.1. Datenbankartefakte	72	9.3. grant Befehl	89
7.1.1. Datenbankartefakte	72	9.3.1. Privilegien zuordnen	89
7.2. Datenbankartefakt: Tabelle	73	9.3.2. Syntax: grant Befehl	89
7.2.1. Tabellen	73	9.3.3. grant Befehl	90
7.2.2. Tabellendefinition	73	10. PL/SQL - Grundlagen	92
7.2.3. Schlüsseldefinition	74	10.1. PL/SQL Grundlagen	92
7.2.4. DDL Befehl: create table	74	10.1.1. PL/SQL	92
7.2.5. Fallbeispiel: create table	74	10.1.2. PL/SQL Sprachspezifikation	93
7.2.6. DDL Befehl: drop table	74	10.1.3. Einsatzgebiete von PL/SQL	93
7.2.7. DDL Befehl: truncate table	75	10.1.4. PL/SQL Engine	94
7.2.8. DDL Befehl: alter table	75	10.1.5. PL/SQL Block	95
		10.1.6. Fallbeispiel: PL/SQL Programm	95
		10.2. PL/SQL Block	96
		10.2.1. PL/SQL Block	96
		10.2.2. Anonymous Block	97

## Informationssysteme

10.2.3. Benannte Blocktypen	97	13. PL/SQL - Blocktypen	118
11. PL/SQL - Datentypen	98	13.1. Blocktyp - Stored Procedure	118
11.1. Datentypen und Variablen	98	13.1.1. PL/SQL Prozedur	118
11.1.1. Definition von Variablen	98	13.1.2. Unterprogrammaufruf	119
11.1.2. Gültigkeit von Variablen	99	13.1.3. Parametertyp	119
11.1.3. Qualifier	99	13.1.4. Formate der Parameterzuweisung	120
11.2. Basisdatentypen	100	13.1.5. Optionale Parameterzuweisung	120
11.2.1. Numerische Datentypen	100	13.1.6. PL/SQL Funktion	120
11.2.2. Alphanumerische Datentypen	101	13.2. Blocktyp - Package	121
11.3. Abgeleitete Datentypen	103	13.2.1. PL/SQL Package	121
11.3.1. Data Dictionary	103	13.2.2. Packagesspezifikation	122
11.4. Strukturierte Datentypen	103	13.3. Blocktyp: Trigger	123
11.4.1. Record - Strukturierte Datentypen	103	13.3.1. PL/SQL Trigger	123
11.5. Tabellentypen	104	13.3.2. Anlegen von Triggern	123
11.5.1. Tabellentypen	104	13.3.3. Pseudorecords	124
11.5.2. Virtuelle Tabellen: Strukturierte Tabellen	104	13.3.4. Namenskonventionen	125
11.6. Daten	104	14. PL/SQL - SQL Funktionen	126
11.5.3. Virtuelle Tabellen	105	14.1. SQL Funktionen	126
11.5.4. DQL: Virtuelle Tabellen	105	14.1.1. SQL Funktionen	126
12. PL/SQL - Befehle	106	14.1.2. Programmieren von SQL Funktionen	127
12.1. SQL Befehle in PLSQL	106	14.1.3. Fallbeispiel: Berechnung der Fakultät	127
12.1.1. DQL - Data Query Language	106		
12.1.2. DML - Data Manipulation Language	107		
12.1.3. FORALL Anweisung	107		
12.1.4. FORALL - RETURNING Klausel	108		
12.1.5. DDL - Data Definition Language	108		
12.2. Kontrollstrukturen	109		
12.2.1. IF THEN Block	109		
12.2.2. Kontrollstruktur - Case Block	109		
12.3. Schleifenkonstrukte	110		
12.3.1. Loop Block	110		
12.3.2. WHILE Block	111		
12.3.3. FOR Block	111		
12.4. Fehlerbehandlung	112	15. NoSQL - Prinzipien	130
12.4.1. Exceptionhandling	112	15.1. Bigdata	130
12.4.2. Exceptionhandler	112	15.1.1. Grundlagen	130
12.4.3. Arten von Exceptions	113	15.1.2. Herausforderungen im Bigdata	131
12.4.4. Anwendungsfehler	113	15.1.3. Technologien im Bigdata Umfeld	131
12.4.5. Unbenannte Fehler	115	15.2. Verteilte Datenbanken	132
12.4.6. Fehlerbehandlung für forall	115	15.2.1. Grenzen konventioneller Datenbanken	132
12.5. Cursor	116	15.2.2. Verteilte Datenbank	132
12.5.1. Cursor	116	15.3. NoSQL Datenbanken	133
12.5.2. Explizite Cursor	116	15.3.1. NoSQL Grundlagen	133
12.5.3. Datenverarbeitung	117	15.3.2. RDB vs. NoSQL	133
12.5.4. BULK FETCH Anweisung	117	16.2. CA Systeme	136
		16.2.1. Theoretische Grundlagen	136
		16.2.2. Beispiele für CA Systeme	137
		16.3. AF Systeme	138
		16.3.1. Theoretische Grundlagen	138

16.3.2. Beispiele für AP Systeme	138	19.5.1. Schemaelement: Eingebettete Objekte	158
16.4. CP Systeme	138	19.5.2. Schemaelement: Eaum	158
16.4.1. Theoretische Grundlagen	138	19.5.3. Schemaelement: Array	159
16.4.2. Beispiele für CP Systeme	138	19.5.4. Arrays von Objekten	159
<b>17. NoSQL - Technologien</b>	<b>139</b>	<b>19.5.5. Collection mit Schemavalidation</b>	<b>160</b>
17.1. Konsistenzmodelle	139	19.6. Views erstellen	160
17.1.1. Arten von Konsistenzmodellen	139	19.6.1. Views	160
17.2. Transaktionskonzept	140	19.7. Datencontainer verwalten	161
17.2.1. ACID Transaktionskonzept	140	19.7.1. Container verwalten	161
17.2.2. Sperren	140		
17.2.3. BASE Transaktionskonzept	141		
17.2.4. Optimistic Locking	141		
17.3. MVCC	141	<b>20. MongoDB - DQL</b>	<b>162</b>
17.3.1. MVCC Verfahren	141	20.1. Daten lesen	162
17.3.2. Diskussion MVCC	142	20.1.1. find Befehl	162
17.4. Consistent Hashing	143	20.2. Kursormethoden	163
17.4.1. Motivation	143	20.2.1. Cursormethoden	163
17.4.2. Verteilen von Daten	144	20.2.2. pretty() Methode	163
17.4.3. Consistent Hashing	144	20.2.3. sort() Methode	164
17.5. Map Reduce Algorithmen	144	20.2.4. limit() Methode	164
17.5.1. Algorithmusbeschreibung	144	20.2.5. skip() Methode	164
17.5.2. Phasenbeschreibung	145	20.2.6. forEach() Methode	165
17.5.3. Diskussion MapReduce	145	20.2.7. batchSize(), objsLeftInBatch()	165
		20.3. Metode	165
<b>18. MongoDB - Modellierung</b>	<b>148</b>	20.3. Query Kriterien - Abfrageobjekt	166
18.1. MongoDB	148	20.3.1. Abfrageobjekt	166
18.1.1. MongoDB Motivation	148	20.3.2. Formen von Bedingungen	166
18.1.2. Skalierbarkeit von Datenbanken	148	20.3.3. Kurzformen von Abfragen	167
18.1.3. Datenformat: JSON	149	20.3.4. Komplexe Bedingungen	167
18.1.4. Codebeispiel: Project Dokument	149	20.3.5. \$in Operator	168
18.2. Modellierungsprinzipien	150	20.3.6. \$where Operator	168
18.2.1. Modellierungsprinzipien	150	20.4. Arrayabfragen	168
18.2.2. Hierarchische Datenmodell	150	20.4.1. \$size Operator	168
18.2.3. Modellstruktur	151	20.4.2. \$all Operator	168
18.2.4. Schemafreie Modellierung	151	20.4.3. \$elemMatch Operator	169
<b>19. MongoDB - DDL</b>	<b>152</b>	20.5. Projektion	169
19.1. Datenkontainer	152	20.5.1. Projektionsobjekt	169
19.1.1. Datenkontainer	152		
19.2. Datenkontainer anlegen	153	<b>21. MongoDB - DML</b>	<b>170</b>
19.2.1. Datenkontainer implizit anlegen	153	21.1. Daten einfügen - insert	170
19.2.2. Collections explizit anlegen	153	21.1.1. insertOne(), insertMany()	170
		Befehl	170
19.3. Collections und Dokumente	154	21.2. Daten bearbeiten - update	171
19.3.1. BSON Datenformat	154	21.2.1. updateOne() Befehl	171
19.3.2. Eingebettete Objekte	155	21.2.2. updateMany Befehl	173
19.4. Dokumentenschema	156	21.3. Strukturooperatoren	173
19.4.1. BSON Schema	156	21.3.1. \$set Operator	173
19.4.2. Schemattribute	156	21.3.2. \$unset Operator	174
19.5. Schemaelemente	158	21.3.3. \$rename Operator	174
		21.4. Wertoperatoren	175
		21.4.1. \$mul Operator	175
		21.4.2. \$min Operator	175
		21.4.3. \$max Operator	176

## Informationssysteme

21.5. Arrayoperatoren	176	22.11. Boolesche Operatoren	202
21.5.1. \$addToSet Operator	176	22.11.1. \$and, \$or Operatoren	202
21.5.2. \$push Operator	177	22.11.2. \$not Operator	202
21.5.3. \$pull, \$pullAll Operator	178	<b>22.12. Kontrolloperatoren</b>	<b>202</b>
21.5.4. \$pop Operator	178	22.12.1. \$cond Operator	202
21.6. Daten Rächen	179	22.12.2. \$switch Operator	203
21.6.1. delete Befehl	179	22.12.3. \$cmp Operator	204
		<b>22.13. Arrayexpressions</b>	<b>204</b>
22. MongoDB - Aggregation von Daten	180	22.13.1. \$arrayElemAt Operator	204
22.1. Methoden und Verfahren	180	22.13.2. \$concatArrays Operator	205
22.1.1. Aggregationsalgorithmen	180	22.13.3. \$in Operator	205
22.2. Abfragemethoden	181	22.13.4. \$map Operator	206
22.2.1. Abfragemethode: count()	181	22.13.5. \$reduce Operator	206
22.2.2. Abfragemethode: distinct()	181	22.13.6. \$reduce Operator	207
22.2.3. Abfragemethode: group()	181	22.13.7. \$isArray Operator	207
22.3. Aggregatframework	182	22.14. Aggregateexpressions	208
22.3.1. Aggregationssphären	182	22.14.1. \$false Beispiel: sales	208
22.3.2. Aggregatframework	183	22.14.2. \$addToSet, \$push Operator	208
22.3.3. Methode: aggregate()	183	22.14.3. \$min, \$max, \$avg Operatoren	209
22.3.4. Pipelinestufen	185	22.14.4. \$sum Operator	209
22.3.5. Fallbeispiel: Projects	185	22.15. Referenzauadrücke	210
22.4. Dokumentstufen	186	22.15.1. \$\$ Operator - Systemvariablen	210
22.4.1. Dokumente filtern - \$match	186	22.15.2. \$expr Operator	210
22.4.2. Dokumente sortieren - \$sort	186	22.15.3. \$let Operator	211
22.4.3. Dokumente entfernen - \$skip	186	<b>22.16. Mengenoperatoren</b>	<b>211</b>
22.4.4. Arrays auflesen - \$unwind	187	22.16.1. \$setDifference Operator	211
22.4.5. Dokumentenstrom einschränken - \$limit	187	22.16.2. \$setIntersection Operator	212
22.4.6. Ergebnis abspeichern - \$out	187	22.16.3. \$setUnion Operator	213
22.5. Strukturentnahmen	188	22.16.4. \$setIsSubset Operator	213
22.5.1. Felder hinzufügen - \$addField	188	<b>22.17. Datenformat - XML</b>	<b>216</b>
22.5.2. Felder projizieren - \$project	190	23.1. XML Grundlagen	216
22.5.3. Dokumententwurzel ändern - \$replaceRoot	191	23.1.1. Einsteigerübersicht von XML	216
22.6. Beziehungsstufen	192	23.1.2. Fallbeispiel: XML Dokumente	216
22.6.1. Relationen definieren - \$lookup	192	23.1.3. Aufbau eines XML Dokuments	217
22.6.2. Unterabfrage definieren - \$lookup	194	23.2. XML Elemente	218
22.7. Aggregatstufen	196	23.2.1. Struktur eines XML Dokuments	218
22.7.1. Aggregatoperatoren	196	23.2.2. Aufbau eines XML Elements	219
22.7.2. Dokumente gruppieren - \$group	196	23.2.3. InhaltsTypen von XML Elementen	219
22.7.3. Fallbeispiel: Gruppierung	197	23.3. Attribute in Xml Elementen	220
22.7.4. Dokumente gruppieren - \$bucket	197	23.3.1. Xml Attribute - Grundlagen	220
22.8. Expression	198	23.3.2. Vergleiche: Elemente vs. Attribute	220
22.8.1. Expressionsyntax	198	23.4. Wohlgeformte XML Dokumente	221
22.9. Arithmetische Operatoren	200	23.4.1. Wohlgeformte XML Dokumente	221
22.9.1. \$add, \$subtract, \$multiply	200	23.4.2. WurzelElement	221
22.9.2. \$divide Operator	201	23.4.3. ElementTags	221
22.9.3. \$abs Operator	201	23.4.4. Überlappung von XML Elementen	221
22.10. Vergleichsoperatoren	201	23.5. XML Namensräume	222
22.10.1. Vergleichsoperatoren	201		

23.5.1. Namenskonflikte	222	25.4.3. <code>&lt;xsl:with-param&gt;</code> Element	241
23.5.2. XML Namensräume	222	25.4.4. <code>xsl:mode</code> Attribut	241
23.5.3. Namensraumdefinition	223	25.5. Prozedurale Verarbeitung	242
23.5.4. Standard Namensraum	223	25.5.1. <code>&lt;xsl:variable&gt;</code> Element	242
23.6. Logische Sicht	224	25.5.2. <code>&lt;xsl:value-of&gt;</code> Element	242
23.6.1. XML Technologien	224	25.5.3. <code>&lt;xsl:if&gt;</code> Element	242
23.6.2. XML Komponentenbaum	224	25.5.4. <code>&lt;xsl:choose&gt;</code> Element	243
23.6.3. Komponentenknoten	225	25.5.5. <code>&lt;xsl:for-each&gt;</code> Element	244
23.6.4. Knotentypen	225	25.5.6. <code>&lt;xsl:sort&gt;</code> Element	244
24. Datenformat - XML XPath	226	25.6. <code>&lt;xsl:otherwise&gt;</code> Element	245
24.1. XPath - Konzepte	226	25.6.1. <code>&lt;xsl:element&gt;</code> Element	245
24.1.1. XPath Konzept	226	25.6.2. <code>&lt;xsl:text&gt;</code> Element	245
24.2. Pfadausdrücke in Kurzform	227	25.6.3. <code>&lt;xsl:attribute&gt;</code> Element	245
24.2.1. Vereinfachte Pfadausdrücke	227	25.6.4. <code>&lt;xsl:attribute-set&gt;</code> Element	246
24.2.2. Auswahl von Elementknoten	227	25.6.5. <code>&lt;xsl:copy-of&gt;</code> Element	246
24.2.3. Kontextknoten	228	25.6.6. <code>&lt;xsl:output&gt;</code> Element	246
24.2.4. Descendant Pfadoperator	228	25.7. Suchanfragen	247
24.2.5. Auswahl von Textknoten	228	25.7.1. Auflösen von Templatkonflikten	247
24.2.6. Auswahl von Attributknoten	228	25.7.2. Default Templates	247
24.3. Lösungsobjekt	229	25.7.3. Roottemplate	247
24.3.1. Lösungsobjekt - Datentypen	229	25.7.4. Stringtemplate	249
24.4. Prädikat	229		
24.4.1. Definieren von Prädikaten	229	<b>26. Datenformat - XML Schema</b>	250
24.5. Pfadausdrücke in Standardform	230	26.1. Schema Grundlagen	250
24.5.1. Lokalisierungsstufen	230	26.1.1. Inhaltsmodell	250
24.5.2. Achsenbezeichner	230	26.1.2. Validierung	251
24.5.3. Knotenabfragen	231	26.1.3. XML Schema Regeln	251
24.5.4. Formen von XPath Ausdrücken	232	26.2. Struktur: XML Schemas	251
24.6. XPath Funktionen	232	26.2.1. Fallbeispiel: XML Schema	251
24.6.1. Kategorien von Funktionen	232	26.2.2. Aufbau eines XML Schemas	251
24.6.2. Knotenabfragenfunktionen	232	26.2.3. Kategorien von Datentypen	253
24.6.3. String Funktionen	233	26.3. Einheitliche Datentypen	253
24.6.4. Logische Funktionen	234	26.3.1. Einheitliche XML Elemente	253
24.7. Fallbeispiel: XSLT	235	26.3.2. Benutzerdefinierte Datentypen	254
24.7.1. Datei browser.xml	235	26.3.3. Derivation by Restriction	254
24.7.2. XPath Ausdrücke	235	26.3.4. Derivation by Union	255
25. Datenformat - XML XSLT	236	26.4. Komplexe Datentypen	255
25.1. XSLT Grundlagen	236	26.4.1. Elemente mit komplexen Datentypen	255
25.1.1. XSLT Stylesheet	236	26.4.2. Strukturierung von XML Elementen	256
25.1.2. Fallbeispiel: Hello World	237	26.4.3. Häufigkeitsindikatoren	257
25.2. XSLT Transformationsprozess	237	26.4.4. gemischter Inhalt	258
25.2.1. Transformationschritte	237	26.5. Attribute	258
25.3. XSLT Programm	238	26.5.1. Attributedefinitions - Elemente mit einfacherem Inhalt	258
25.3.1. XSLT Stylesheet	238	26.5.2. Attributedefinitions - Elemente mit komplexem Inhalt	259
25.3.2. Templateregel	238	26.6. Sichtbarkeitkonzepte	259
25.3.3. XSLT Suchmuster	238	26.6.1. globale Datentypdeklarationen	259
25.3.4. Template Anweisungen	239		
25.4. Dekorative Verarbeitung	240		
25.4.1. <code>&lt;xsl:apply-templates&gt;</code> Element	240	<b>27. Datenformat - JSON</b>	260
25.4.2. <code>&lt;xsl:template&gt;</code> Element	240		

## Informationssysteme

27.1. JSON Datenformat	261	.	
27.1.1. JSON Grundlagen	261		
27.2. JSON Datentypen	262		
27.2.1. JSON Datentypen	262		
27.2.2. JSON Grundstrukturen	263		
27.2.3. Einfache Datentypen	264		

Glossar

267

Index

270

# Entwicklung relationaler Datenbanken

August 19, 2020

Informationssysteme

## 01 Datenbankentwurf

Datenmodellierung

01. Phasen des Datenbankentwurfs	10
02. Entity Relationship Modell	12
03. Relationale Modell	15
04. Normalisierung	19

**1.1. Datenbankentwurf**

Der Datenbankentwurf beschreibt alle Tätigkeiten und Aufgaben, die zur Erstellung eines physischen Datenbankschemas notwendig sind.

Ein **Datenbankschema** beschreibt die physische Struktur der Daten einer Datenbank.

**1.1.1 Phasen des Datenbankentwurfs**

Der Datenbankentwurf für relationale Datenbanken durchläuft in der Regel 4 Phasen.

→ Vorgangsweise: Datenbankentwurf

**Informationsbeschaffung:** In der externen Phase wird die Informationsstruktur des Datenmodells definiert. Dazu wird eine **Bestandsaufnahme** der Anforderungen des Kunden durchgeführt.

**Konzeptionelle Phase:**

**Semantische Modell:** Ziel des konzeptionellen Entwurfs ist eine formalisierte Beschreibung der Anforderungen des Kunden.

Dazu werden alle für die Aufgabe essentiellen Gegenstände, Personen, Dienstleistungen und deren Beziehungen untereinander erfasst und grafisch dargestellt.

**Logische Phase:**

**Logische Datenmodell:** Das Ziel der logischen Phase ist die Übertragung des semantischen Modells in ein logisches Datenmodell. Das logische Datenmodell entspricht in seiner Form der **Struktur der Objekte** einer Datenbank.

**Physische Phase:**

**Datenbankschema:** In der physischen Phase wird aus dem logischen Modell die **physische Struktur** der Datenbank generiert.

10

## DB Entwicklung



### 1.1.2 Externe Phase

In der externen Phase wird die **Informationsstruktur** des Datenmodells definiert.

- Ziel der externen Phase ist die **Informationsbeschaffung**. Die Anforderungen des Kunden an das Datenmodell werden gesammelt und strukturiert.

#### Erklärung: Externe Phase

- In der externen Phase erfolgt eine Bestandsaufnahme der Anforderungen des Kunden. Es muss bestimmt werden, welche **Informationen** das Datenbanksystem generieren soll und welche Informationen dann bereitgestellt werden müssen.
- Das Ergebnis der externen Phase ist eine informelle Beschreibung der Geschäftsprozesse in Form eines Dokuments.

#### Vorgangsweise: Externe Phase

- In der externen Phase erfolgt eine Abbildung der Anforderungen auf eine Menge von Entitäten.
- Betrachtet man z.B.: ein System Schule erkennt man gewisse Objekte bzw. Objektmeinungen wie das Fach Deutsch, ein Schüler namens Max oder einen Lehrer namens Franz.



### 1.1.3 Konzeptionelle Phase

Ziel der konzeptionellen Phase ist eine **graphische Repräsentation** der informellen Beschreibung der Anforderungen.

#### Erklärung: Konzeptionelle Phase

- Das semantische Modell dient zur **Veranschaulichung** des technischen Datenmodells im naiven Kontext des Geschäftsumfelds. Das Modell muss sowohl für Geschäftsbüroden als auch für Datenbankentwickler verständlich sein.
- Das Ergebnis der konzeptionellen Phase ist die Beschreibung der Geschäftsprozesse als **Entity Relationship Modell**.

### 1.1.4 Logische Phase

Das Ziel der logischen Phase ist die **Transformation** der Entitäten und Beziehungen des semantischen Modells in eine **Tabellestruktur**.

#### Erklärung: Logische Phase

- Das semantische Modell unterscheidet zwei Strukturelemente: Entitäten und Beziehungen. Das logische Modell beschreibt die Welt hingegen als eine Menge von Tabellen.
- Der logische Entwurf überführt die Entitäten und Beziehungen des semantischen Modells in eine Tabellenstruktur.

11

## Informationssysteme

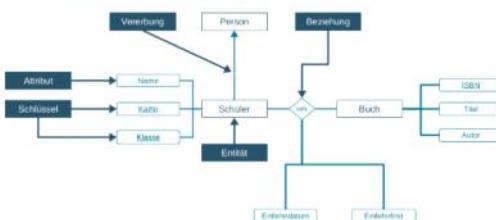


Abbildung 1: Entity Relationship Modell

- Zur Abbildung des semantischen Modells in eine Tabellenstruktur, wird eine Reihe von Transformationsregeln auf die Entitäten und Beziehungen des ER-Diagramms angewandt.

- Das Ergebnis der logischen Phase ist die Beschreibung der Geschäftsprozesse als Relationalles Modell.

### 1.1.5 Physische Phase

In der physischen Phase wird aus dem logischen Modell die physische Struktur der Datenbank generiert.

#### Analyse: Physische Phase

- Das Ergebnis des physischen Entwurfs ist eine Folge von Datenbankbefehlen zum Anlegen des physischen Schemas des Datenmodells.
- Dann müssen alle notwendigen Datentypen, Wertebereiche, Relationen bzw. Sichten für das Schema definiert werden.
- Das Ziel des physischen Entwurfs ist die Überführung des logischen Modells in eine physische Datenbankschemata.
- Die physische Phase ist der abschließende Schritt im Datenbankentwurfsprozesses.

### 1.2. Entity Relationship Modell

ER Modell  
Ein Modell beschreibt ein zweckorientiertes vereinfachtes Abbild der Wirklichkeit.

Zur Beschreibung des semantischen Modells einer relationalen Datenbank, wird das **Entity Relationship Modell** verwendet.

#### 1.2.1 ER Modell

Das ER Modell beschreibt die grundlegende Entitäts- bzw. Beziehungsstruktur des semantischen Modells.

#### Erklärung: Datenmodell

- Im ER Modell werden alle essentiellen Personen, Dienstleistungen bzw. Gegenstände und deren Beziehungen untereinander graphisch erfasst.

- Betrachtet man z.B.: ein System Schule, erkennt man Entitäten wie das Fach Mathematik, einen Schüler Namens Kurt bzw. einen Lehrer namens Hans.

- Das ER Modell unterscheidet dabei zwei grundlegende Elemente: Entitäten und Beziehungen.

12

ER Element	Beschreibung	Seite
Entität	Als Entität wird in der Datenmodellierung ein datenträgendes Objekt bezeichnet.	13
Entitätsyp	Entitätsypen beschreiben Gruppen von Entitäten, die aufgrund gleicher oder ähnlicher Attributwerte zur thematischen Abgrenzung zusammengefasst werden. Entitätsypen werden im ER Modell durch Rechtecke symbolisiert, zugehörige Attribute durch Ellipsen.	13
Schlüssel	Der Schlüssel einer Entität beschreibt eine Attributkombination die für jede Entität eines Entitätsyps eindeutig ist.	13
Beziehung	Eine Beziehung beschreibt die Art und Weise in der 2 Entitäten miteinander in Verbindung stehen. Dabei handelt es sich in der Regel um eine Zugehörigkeit zu einer Aktion oder einem Sachverhalt. Beziehungen werden im ER Modell durch Rauten symbolisiert.	13

Abbildung 2. Elemente des ER Modells

- 1.2.2 Entität**
- Als Entität wird in der Datenmodellierung ein datenträgendes Objekt bezeichnet.
- Eine Entität kann dabei ein materieller oder immaterieller, konkretes oder abstraktes Konzept beschreiben.
- Erklärung: **Entität**
- Datenträgende Objekte werden durch Attribute näher bestimmt. Attribute beschreiben die Merkmale bzw. Eigenschaften der Entitäten.
  - Zur thematischen Abgrenzung logischer Konzepte werden Entitäten mit gleichen oder ähnlichen Attributausprägungen zu Entitätsktypen zusammengefasst.
  - Das ER Modell abstrahiert die Entitätsktypen sowie deren Beziehungen in einer graphischen Repräsentation. In der Datenbank entsprechen die Entitätsktypen den Tabellen, die Tabellenzeilien den Entitäten.
  - Der Schlüssel einer Entität beschreibt eine Attributkombination die für jede Entität eines Entitätsktyps eine andere Ausprägung besitzt.
- 1.2.3 Beziehung**
- Das ER Modell stellt im Kontext des Datenbanksentwurfs einen relevanten Ausschnitt der realen Welt dar.
- Erklärung: **Beziehung**
- Im Modell werden alle essentiellen Objekte und deren Beziehungen untereinander graphisch erfasst.
  - Eine Beziehung beschreibt die Form in der 2 Entitäten miteinander in Verbindung stehen. Dabei handelt es sich in der Regel um die Zugehörigkeit zu einer Aktion oder einem Sachverhalt.
  - Die Form einer Beziehung wird dabei durch die Kardinalität der Beziehung definiert. Die Kardinalität einer Beziehung beschreibt in welcher Form die Entitäten einer Beziehung untereinander assoziiert werden.
  - Ein Schüler kann z.B.: ein oder mehrere Bücher ausleihen. Ein Buch kann im Laufe der Zeit wiederum von mehreren Schülern ausgeliehen werden sein. Bei dieser Art von Beziehung handelt es sich um eine univ. Relation weil n Objekte des einen Entitätsktyps mit n Objekten des anderen Entitätsktyps in Relation stehen.
  - Generell werden 3 Formen von Beziehungen unterschieden: 1:1, 1:n und n:m.
  - Beziehungen werden im ER Modell durch Rauten symbolisiert, wobei die Kardinalität der Beziehung bei der jeweiligen Entität notiert wird.



13

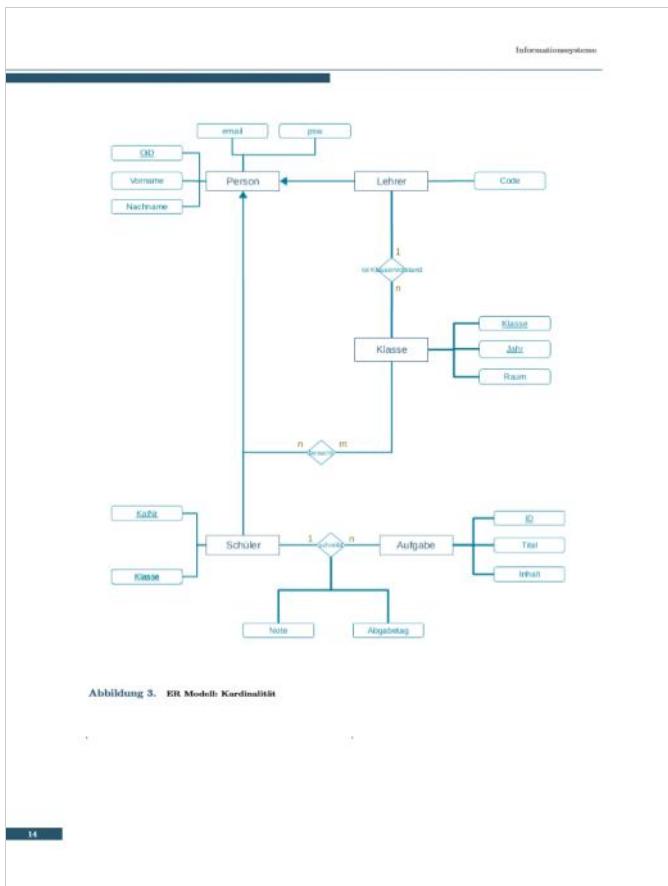


Abbildung 3. ER Modell: Kardinalität

Kardinalität	Beschreibung	Seite
1..1	Bei der 1..1 Beziehung steht ein Objekt des ersten Entitätstyps mit genau einem Objekt des anderen Entitätstyps in Beziehung, z.B.: Ein Schüler hat einen Schultopf. Ein Schultopf ist immer genau einem Schüler zugeordnet.	12
1..n	Bei einer 1..n Beziehung steht ein Objekt des ersten Entitätstyps mit einem oder mehreren Objekten des anderen Entitätstyps in Beziehung. Ein Schüler verfügt über Aufgaben. Eine Aufgabe ist genau einem Schüler zugeordnet.	12
n..m	Bei einer n..m Beziehung steht ein Objekt des ersten Entitätstyps mit einem oder mehreren Objekten des anderen Entitätstyps in Beziehung. Die Beziehung gilt dabei für beide Entitätstypen, z.B.: Eine Schularbeit wird von den Schülern einer Klasse geschrieben. Schüler schreiben mehrere Schularbeiten.	12

Abbildung 4. ER Modell: Kardinalität

### 1.3. Relationale Modell

Das Ziel des logischen Entwurfs ist die Transformati on der Entitäten und Beziehungen des semantischen Modells in eine Tabellenstruktur.

#### 1.3.1 Relationale Modell

Das relationale Modell basiert auf einer tabellarischen Organisation der Daten.

##### » Erklärung: Relationale Modell »

- Das Relationale Modell kennt im Gegensatz zum ER Modell nur ein einzelnes Strukturelement: die Tabelle.
- Im Zuge des relationalen Entwurfs wird die Struktur der Tabellen des Datenbankschemas definiert. Dazu wird die Anzahl, der Name und der Typ der Spalten einer Tabelle bestimmt.
- Zur logischen Organisation der Daten einer Tabelle, wird für jeden Tabelleneintrag ein Schlüssel definiert.
- Dabei beschreibt ein Schlüssel eine Spaltenkombination, die für jeden Datensatz einer Tabelle eine unterschiedliche Ausprägung besitzt. Der Schlüssel kann dabei aus einer oder mehreren Spalten bestehen.
- Im relationalen Entwurf werden Tabellenschlüssel auch als Primärschlüssel bezeichnet.

#### 1.3.2 Beziehungen zwischen Tabellen

Eine Beziehung beschreibt die Art und Weise in der Tabelleninträge miteinander in Relation stehen.

##### » Erklärung: Beziehungen »

- Zur Definition einer Beziehung im relationalen Modell werden die Schlüssel der Tabelleninträge in Relation gesetzt. Dazu wird ein Datensatz um den Primärschlüssel eines anderen Eintrags ergänzt.
- Aus technischer Sicht wird die Tabellenstruktur des datentragenden Objekts, um den Primärschlüssel der gewünschten Entität erweitert.
- Die neu definierten Tabellenkolonnen werden auch als Fremdschlüsselspalten bezeichnet, weil sie auf den Primärschlüssel einer anderen Tabelle beziehen.
- Eine Beziehung besteht dabei für alle Tabelleninträge die für Primär- und Fremdschlüsselspalten übereinstimmende Einträge besitzen.
- Die Kardinalität einer Beziehung bestimmt in welcher Form der Fremdschlüssel in die Tabellenstruktur des Datenmodells integriert wird.
- Das relationale Modell definiert dabei eine Reihe von Transformationsregeln, um die Elemente des ER Modells in eine Tabellenstruktur zu zwingen.
- Mit dem Relationalen Modell liegt der abzuhörende Datenbestand in seiner finalen Form vor. Das Relationalen Modell beschreibt in dieser Form die Struktur des physischen Datenbankschemas.

15

## Informationssysteme

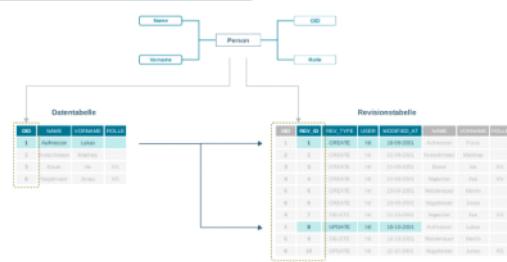


Abbildung 5. Datentabelle vs. Revisionstabelle

### 1.3.3 Transformationsregel: Entität

Das Relationale Modell kennt im Gegensatz zum ER Modell nur ein einzelnes Strukturelement: die Tabelle.

##### » Transformation: Entitäten »

- Das Ziel des logischen Entwurfs ist die Transformation der Entitäten und Beziehungen des semantischen Modells in eine Tabellenstruktur.
- Die Entitäten desselben Entitätstyps werden dazu zulässig in einer Tabelle gesammelt.
- Im relationalen Modell werden dazu die Anzahl, die Namen und die Typen der Spalten der Tabellen definiert. Das relationale Modell unterscheidet dabei mehrere Formen von Tabellen.
- Datentragende Tabellen werden im Modell als Datentabellen bezeichnet. Datentabellen speichern den Großteil der Daten eines Geschäftsfalls.
- Zur Protokollierung von Änderungen im Datenbestand, werden den Datentabellen Revisionstabellen beigelegt.

### 1.3.4 Transformationsregel: 1..1 Beziehung

Beziehungen beschreiben die Art und Weise in der Tabelleninträge miteinander in Relation stehen.

##### » Transformation: 1..1 Beziehung »

- Bei einer 1:1 Beziehung steht ein Objekt des ersten Entitätstyps mit genau einem Objekt des anderen Entitätstyps in Beziehung.
- Zur Modellierung einer 1:1 Beziehung, wird eine der Tabellen, um den Schlüssel der anderen Entität erweitert.
- Welche der Tabellen um Schlüsselkolonnen erweitert werden soll, liegt dabei im Beneben des Entwicklers. Handelt es sich bei einer Relation z.B. um eine optionale Beziehung muss die abhängige Entität um den Schlüssel erweitert werden.



16

Tabellentyp	Beschreibung	Seite
<b>Datentabelle</b>	Als Datentabellen werden die datenträgernden Tabellen eines Modells bezeichnet. Datentabellen halten in der Regel den Großteil der Daten eines Geschäftsmodells und sind Datenbestandteile des Modells.	16
<b>Attributabelle</b>	Konzeptuelle Attributtabellen, die die Datenmodelle erweitern. Die zugehörigen Werte des Attributtyps müssen dabei den Datenbestand der Tabelle benennen und Attributtabellen nach dem Aufzählungstyp auf die sie sich beziehen. Dem Namen wird zusätzlich ein « vorangestellt, verwendet z.B. e_project_types.	12
<b>Schlüsseltable</b>	mn Beziehungen werden im relationalen Entwurf mit der Hilfe von Schlüsselstabellen modelliert. Dazu speichert die Schlüsselstabellen für jede an der mn Relation beteiligte Entität deren Schlüssel. Der Schlüssel der Schlüsselstabellen selbst, ist eine Kombination aller eingetragenen Schlüssel. Benannt sind Schlüsselstabellen nach den Namen aller an der Relation beteiligten Entitäten, ergänzt durch einen Unterstrich. Dem Namen wird zusätzlich ein » nachgestellt, z.B.: project_employee.	12
<b>Revisionstable</b>	Revisionsstabellen werden zur Protokollierung von Änderungen am Datenbestand verwendet. Benannt sind Revisionstabellen nach den Entitätsarten auf die sie sich beziehen. Dem Namen wird zusätzlich ein v vorangestellt, z.B.: v_employee.	16
<b>Sammeltabelle</b>	Vereerbungsbeziehungen zwischen Entitäten werden im Relationalen Modell mit der Hilfe von Sammeltabellen modelliert. Alle Objekte eines Entitätstyps oder davon abgeleiteter Entitätstypen werden in einer einzelnen Tabelle gesammelt. Die Tabellenstruktur einer Sammeltabelle setzt sich aus allen möglichen Attributen der einzuordnenden Objekten zusammen. Benannt werden diese Sammeltabellen nach dem Entitätstyp der Basisentität. Dem Namen wird zusätzlich ein m nachgestellt, z.B.: employees_st.	16
<b>Basistabelle, Kindtable</b>	Zur Modellierung von Vereerbungsbeziehungen können die Werte eines Objekts auf eine Basistabelle und Kindtabellen verteilt werden. Konzeptionell entspricht der Vorgang der Modellierung von Basisklassendateien der OOP. Benannt werden die Tabellen nach den Entitätsarten auf die sie sich beziehen. Basistabellen wird zusätzlich ein Kürzel br nachgestellt.	16

Abbildung 6. Relationales Modell: Tabellentypen

**1.3.5 Transformationsregel: 1..n Beziehung**

Beziehungen beschreiben die Art und Weise in der Tabelleneinträge miteinander in Relation stehen.

**1.3.6 Transformationsregel: n..m Beziehung**

Beziehungen beschreiben die Art und Weise in der Tabelleneinträge miteinander in Relation stehen.

## » Transformation: 1..n Beziehung »

- Bei der 1..n Beziehung steht ein Objekt des ersten Entitätstyps mit einem oder mehreren Objekten des anderen Entitätstyps in Beziehung.
- In der Gegenrichtung wird jeder Entität des anderen Entitätstyps genau eine Entität des ursprünglichen Entitätstyps zugeordnet.
- Zur Modellierung einer 1..n Beziehung, wird die Tabellenstruktur der abhängigen Entität, um den Schlüssel der Zielenität erweitert.

## » Transformation: n..m Beziehung »

- Bei einer n..m Beziehung kann ein Objekt des einen Entitätstyps mit beliebig vielen Objekten des anderen Entitätstyps in Beziehung gesetzt werden.
- Zur Modellierung einer n..m Beziehung werden die Schlüsselwerte der Entitäten in einer eigenen Tabelle in Beziehung gesetzt.
- Zusätzliche Attribute der Beziehung werden ebenfalls in der Schlüsselstabelle gespeichert.

17

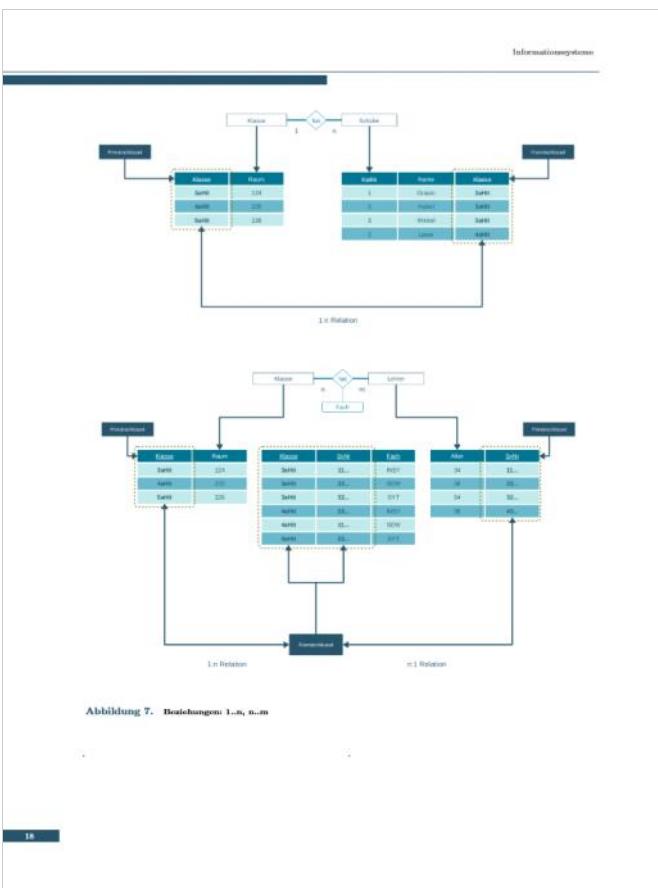


Abbildung 7. Beziehungen: 1..n, n..m

### 1.3.7 Vererbungskonzept

Zur thematischen Abgrenzung logischer Konzepte werden Entitäten mit gleichen oder ähnlichen Attributen zu **Entitätsgruppen** zusammengefasst.

- Entitätsgruppen abstrahieren Geschäftsprozesse in Form einfacher Netzstrukturen.

#### Erklärung: Vererbungskonzept

- Konzeptionell kann durch die Definition von Vererbungsbeziehungen eine weitere strukturelle Vereinfachung erreicht werden.

- Eine Vererbungsbeziehung drückt aus, dass 2 Entitätsgruppen zwar nicht gleich sind, aber eine große Ähnlichkeit haben. Die Vererbung kann dabei zwischen 2 oder mehreren Entitätsgruppen definiert werden.

- Bei der Vererbung wird dazu eine Basisentität bestimmt, die ihre Attribute an alle Subentitäten weitergibt, die von ihr erbten.

- Gemeinsame Attribute müssen damit nur einmal modelliert werden.



### 1.3.8 Transformationsregel: Vererbung

Konzeptionell können Vererbungsbeziehungen im Relationalen Modell nicht definiert werden.

#### Transformation: Vererbung

- Der relationalen Entwurf abstrahiert 2 Formen zur Abbildung von Vererbungsbeziehungen.

- Single Table Inheritance:** Die Objekte der Basisentität und aller Subentitäten werden gesammelt in einer einzigen Tabelle eingetragen. Die Tabellenstruktur der Sammeltabelle setzt sich dabei aus allen Attributien der einzelnen Objekte zusammen.

- Joined Table Inheritance:** Die Daten der Objekte werden verteilt auf mehrere Tabellen eingetragen. Die Werte der Basisentität werden dazu in einer eigenen Basistabelle gesammelt. Die restlichen Werte werden verteilt auf andere Tabellen gespeichert. Für jeden Subtyp definiert das Modell dazu eine eigene Tabelle.

### 1.4. Normalisierung

Als **Normalisierung** wird ein Ansatz des Datenbankdesigns zur Vermeidung redundanter Daten bezeichnet.

#### 1.4.1 Datenredundanz

Nominell werden Mehrfacheinträge derselben Daten in einem einzelnen Datenbankschema als Datenredundanz beschrieben.

Redundanz beschreibt in diesem Zusammenhang die mehrfache Speicherung derselben Information in einer Tabelle.

#### Erklärung: Datenredundanz

- Datenredundanz** verursacht bei der Verarbeitung von Daten Anomalien. Solche Inkonsistenzen treten z.B.: dann auf wenn nur eine der Kopien eines redundanten Datensatzes geändert wird.

- Um Datenredundanz zu verhindern werden die Tabellen eines Datenbankschemas normalisiert.

- Unter dem Begriff **Normalisierung** wird die Strukturierung einer Datenbasis im Sinne eines konsistenten Verhaltens verstanden.

### Normalisierung

Bei der Normalisierung handelt es sich um eine **Strategie**, Redundanzen aus dem Datenbestand eines Datenschemas zu beseitigen.

#### Erklärung: Normalisierung

- Normalisierung beschreibt dabei den Prozess der Restrukturierung und Neuorganisation relationaler Datenbankschemata.

- Dazu wird eine Reihe von **Regeln** definiert, die fortlaufend gegen die Datenbasis eines Schemas angewendet werden.

- Im Zuge der Normalisierung erfolgt eine wiederholte Neustrukturierung der Tabellen bis keine Datenredundanz mehr vorhanden sind.

- Für die Normalisierung relationaler Datenmodelle werden 3 grundlegende Normalformen definiert. Die Normalformen basieren dabei aufeinander auf. Ist die eine erfüllt sind auch alle darunterliegenden erfüllt.

10

### Informationssysteme

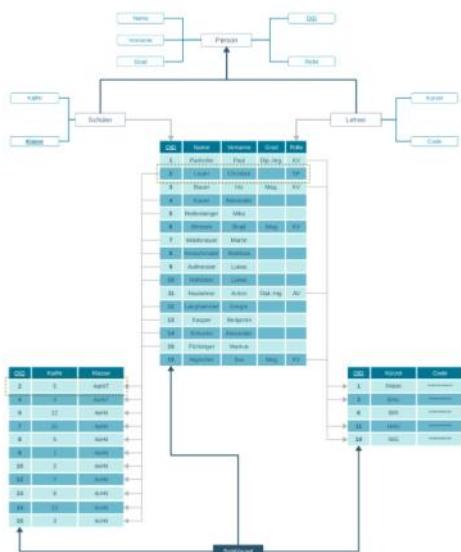


Abbildung 8. Vererbung: Basistabelle, Kindtabellen

20

Normalform	Beschreibung	Seite
1.Normalform	Eine Tabelle befindet sich in erster Normalform, wenn jede Information innerhalb einer Tabelle eine eigene Tabellenspalte bekommt und zusammenhängende Informationen, wie z.B. die Postleitzahl und der Ort, nicht in einer Tabellenspalte vorliegen.	21
2.Normalform	Tabellen in zweiter Normalform müssen bereits in erster Normalform vorliegen. Zusätzlich darf für die Datensätze einer Tabelle immer nur einen Sachverhalt abbilden. Abstrahieren die Daten einer Tabelle mehrere Sachverhalte müssen sie in zusätzliche Tabellen zerlegt werden.	21
3.Normalform	Die dritte Normalform gilt als eingehalten, wenn die zweite Normalform erfüllt ist und keine Indirekten Abhängigkeiten innerhalb einer Tabelle mehr bestehen.	13

Abbildung 9. Elemente des ER Modells

1.4.2 Erste Normalform	1.4.3 Zweite Normalform
	<p>In der Datenbankentwicklung ist die 3. Normalform ausreichend, um die perfekte Balance aus Redundanz, Performance und Flexibilität.</p>

21

### Informationssysteme

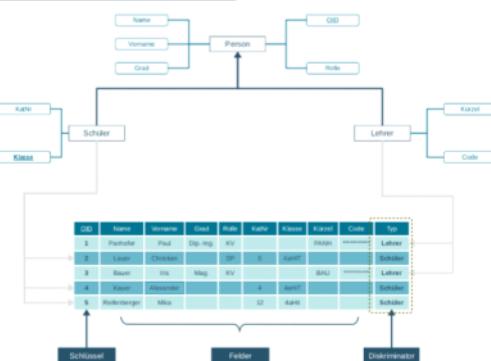


Abbildung 10. Vererbung: Sammeltabelle

1.4.4 Dritte Normalform
-------------------------

22

# SQL - Oracle 19c

August 19, 2020

## 2. SQL - Data Query Lanugage

### 01

#### SQL Grundlagen

01. SQL Grundlagen	24
02. Projektion - Select Klausel	26
03. Kondition - Case Klausel	30
04. Restriktion - Where Klausel	31
05. Sortierung - Order By Klausel	34
06. Limitierung - Fetch Klausel	35
07. Paginierung - Offset Klausel	35

### 2.1. SQL Grundlagen

SQL ist eine Programmiersprache, mit der relationale Datenbanken erstellt bzw. bearbeitet, sowie Datenbestände abgefragt werden können.

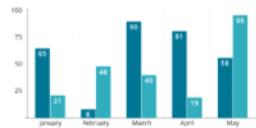
#### 2.1.1 SQL - Structured Query Language

SQL als Programmiersprache folgt dem deklarativen Programmierparadigma.

##### Erklärung: Deklarative Programmierung

- Die deklarative Programmierung ist ein Programmierparadigma, bei dem ein Problem durch den Programmierer beschrieben, jedoch kein Lösungsweg vorgegeben wird.

Es ist die Aufgabe der Datenbankengine eine Lösung zu berechnen.



##### Erklärung: SQL Syntax

- SQL als Sprache zeichnet sich durch eine einfache Syntax aus. Die Sprache selbst besteht dabei im Wesentlichen aus englischen Sprachelementen.
- SQL als Sprache ist standardisiert und wird plattformübergreifend in relationalen Datenbanksystemen eingesetzt. Es existieren jedoch mehrere SQL-Dialekte unterschiedlicher Hersteller, die eine vollständige Kompatibilität des Standards verhindern.



### 2.1.2 Kategorien von SQL Befehlen

Der SQL Befehlsatz ist in mehrere Kategorien aufgeteilt.

##### Auflistung: Kategorien von SQL Befehlen

- Data Manipulation Language: Befehle zum Verarbeiten von Daten.
- Data Definition Language: Befehle zum Definieren der Struktur einer Datenbank.
- Data Query Language: Befehle zum Auslesen von Daten aus einer Datenbank.

#### 2.1.3 DQL - Select Anweisung

Die select Anweisung wird verwendet um Daten aus einer Datenbank zu lesen.



##### Begriff: DML Befehlsatz \*

DML Befehle werden zum **Bearbeiten, Einfügen und Löschen** von Daten verwendet.

- Einfügen von Daten
- INSERT INTO ...**
- Verändern von Daten
- UPDATE employees ...**
- Löschen von Daten
- DELETE FROM ...**

##### Begriff: DDL Befehlsatz \*

DDL Befehle werden zur **Definition des Schemas<sup>1</sup>** einer Datenbank verwendet.

- Anlegen von Tabellen
- CREATE TABLE ...**
- Ändern von Tabellen
- ALTER TABLE ...**

##### Begriff: DCL Befehlsatz \*

DCL Befehle werden zum **Verwalten von Rechten** bzw. zur Steuerung von **Transaktions**en verwendet.

- Weitergabe von Rechten fuer den
- Zugriff auf Tabellen
- GRANT OR TO ...**
- Beenden einer Transaktion
- COMMIT ...**

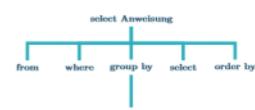
##### Begriff: DQL Befehlsatz \*

DQL Befehle werden zum **Auslesen** von Daten aus der Datenbank verwendet.

- Daten aus der Datenbank lesen
- SELECT employee\_id FROM ...**

#### Erklärung: Select Anweisung \*

- Die select Anweisung wird verwendet um Daten aus den Tabellen einer Datenbank zu lesen.
- Die Anweisung selbst besteht aus mehreren Segmenten den Klauseln. Jede Klause erfüllt eine spezifische Aufgabe. Die select Anweisung besteht dabei immer zumindestens aus 2 Klausen: der select Klause und der from Klause. Alle anderen Klauseln sind optional.
- Die Klausulen einer select Anweisung werden von der Datenbankengine nicht in der Reihenfolge ausgewertet, in der sie in der Abfrage auftreten.
- Welche Daten, zu welchem Zeitpunkt, auf welche Weise verarbeitet werden, wird durch die jeweilige Klause bestimmt.
- Auswertungsreihenfolge von SQL Klauseln:



#### Auflistung: Klauseln der Select Anweisung \*

- From Klause - Projektion**  
In der from Klause wird die Datenbasis einer Abfrage festgelegt.  
Die Klause ist **obligatorisch**.
- Where Klause - Restriktion**  
Die where Klause wird zur Filterung der Datensätze einer Abfrage verwendet.  
Die where Klause ist **optional**.

<sup>1</sup> Beschreibung der Struktur der Daten in der Datenbank

#### 2.2 Select Klausel - Projektion

##### 2.2.1 Select Klausel

In der select Klausel werden jene Spalten definiert, deren Werte die Ergebnismenge einer Abfrage bilden.

##### Query: Select Anweisung \*

- Select Anweisung
- Select Anweisung
- SELECT \* FROM employees;

##### Analyse: Select Anweisung \*

- Hier steht gewissermassen der Satz: Wähle alle Spalten der Tabelle employees und zeige die Dates als Ergebnis an.
- Am Ende der select Anweisung steht ein Semikolon. Das Semikolon ist dabei nicht Teil der select Anweisung, sondern ein Steuerzeichen der Datenbank.



##### Query: Projektion von Daten \*

- Projektion von Daten
- Projektion von Daten
- SELECT DISTINCT first\_name, last\_name
- FROM employees;

##### Erklärung: Projektion von Daten \*

- Soll für eine SQL Abfrage die Spaltenauswahl der Ergebnismenge eingeschränkt werden, wird eine durch Komma getrennte Liste von Spaltenbezeichnern in der select Klause angeführt.
- Mit dem distinct Schlüsselwort kann die Datenbank einige anhalten werden, Duplikate aus dem Abfrageergebnis zu entfernen.

## 2.2.2 Wiederholte Spaltenausgabe

In der select Klammer kann die Angabe von Spaltenbezeichnern in beliebiger Reihenfolge bzw. in beliebiger Zahl erfolgen.

- Erklärung: Wiederholte Spaltenausgabe
  - In einer select Klammer kann der gleiche Spaltenbezeichner mehrfach angegeben werden.
  - Die Ausgabe der Daten erfolgt entsprechend, der in der Klammer definierten Reihenfolge der Spaltenbezeichner.

### Query: Wiederholte Spaltenausgabe

```

1 --- Wiederholte Spaltenausgabe
2
3 SELECT last_name, first_name, department_id,
4       salary,
5       salary
6 FROM employees;
7
8 SELECT department_id, department_name,
9       department_head,
10      department_code,
11      department_name
12 FROM departments;

```

## 2.2.3 Spaltenwerte bearbeiten

Daten können vor ihrer Ausgabe in der select Klammer adaptiert werden.

### Query: Spaltenwerte algebraisch bearbeiten

```

1 --- Mit Spaltenwerten rechnen
2
3 SELECT employee_id, last_name, first_name,
4       salary * 1.03,
5       salary * 1.10,
6       salary - (salary * 0.05)
7 FROM employees;

```

### Analyse: Spaltenwerte bearbeiten

- Durch die Anweisung wird zusätzlich zum Gehalt das um 3% erhöhte Gehalt ausgegeben.
- Alphanumerische Werte können im Rahmen einer Projektion nicht algebraisch verändert werden.

## 2.2.4 Spaltenwerte verknüpfen

Zur Bearbeitung alphanumerischer Werte stellt SQL den || Operator zur Verfügung.

- Erklärung: Spaltenwerte verknüpfen
  - SQL verwendet den || Operator um alphanumerische Werte miteinander zu verknüpfen.
  - Der || Operator kann dabei beliebig oft in einem SQL Ausdruck auftreten.

### Query: Spaltenwerte verknüpfen

```

1 --- Alphanumerische Werte verknüpfen
2
3 SELECT last_name || ' ' ||
4       first_name
5 FROM employees;

```

## 2.2.5 Spaltenalias

Aliase sind alternative Namen für Spaltenbezeichner.

- Ein Spaltenalias markiert den eigentlichen Spaltennamen durch eine sprechende Bezeichnung.

### Erklärung: Spaltenalias

- Ein Spaltenalias wird durch ein Leereszeichen getrennt, der Spaltenbezeichnung nachgestellt, deren Bezeichnung maskiert werden soll.
- Klammer die von der SQL Engine nach der select Klammer ausgewertet werden, besitzen eine Referenz auf Spaltenalias.

- enthält ein Spaltenalias Sonderzeichen, muß es unter Hochkoma gesetzt werden.

### Query: Spaltenalias definieren

```

1 --- Spaltenalias definieren
2
3 SELECT last_name nachname, job_id beruf
4 FROM employee;
5
6 SELECT department_name "Abteilungsbezeichnung"
7      "Edubericht"
8 FROM departments;

```

37

## 2.2.6 Pseudospalten

Pseudospalten stellen eine alternative Form von Spaltenbezeichnern für SQL Abfragen dar.

### Pseudospalte

- Funktionen: Die SQL Spezifikation definiert eine Reihe von parametrischen Funktionen die in SQL Abfragen als Teil der select Klammer verwendet werden.
- Spaltenalias: Ein Spaltenalias ist ein symbolischer Alias auf die Spalten einer anderen Tabelle.

Pseudospalten werden in SQL Abfragen wie gewöhnliche Spaltenbezeichner verwendet. Es können jedoch keine Werte für diese Art von Spalten eingegeben werden.

### Query: Pseudospalten

```

1 --- Pseudospalte: ROWNUM
2
3 Das Ergebnis der Abfrage setzt sich aus
4 den ersten 10 Datensätzen des Ergebnisses
5 zusammen.
6
7 SELECT * FROM employees WHERE ROWNUM < 10;
8
9 --- Pseudospalte: SYSTIME
10
11 Das Ergebnis der Abfrage enthält die
12 gegenwärtige Zeit des Datenbankserver
13
14 z.B.: 2017-11-23 23:41:08
15
16 SELECT SYSTIME FROM DUAL;
17
18 --- Pseudospalte: TIMESTAMP
19
20 Das Ergebnis der Abfrage enthält die
21 gegenwärtige Zeit des Datenbankserver
22 auf Millisekunden genau.
23
24 SELECT TIMESTAMP FROM DUAL;

```



## 2.2.7 Pseudospalten: nextval, currval

nextval und currval sind 2 Funktionen zur Verwaltung von Sequenzobjekten.

- Erklärung: Sequenzen in SQL Abfragen
  - Sequenzobjekte sind wie Tabellen Datenbankobjekte.

- Sequenzen werden zum Generieren von Schlüsselwerten für Datenbanktabellen verwendet.

### Query: NEXTVAL, CURRVAL

```

1 --- Pseudospalte: NEXTVAL
2
3 --- Die NEXTVAL Pseudospalte liefert den
4 --- nächsten Wert der Sequenz.
5
6 SELECT employee_seq.NEXTVAL FROM DUAL;
7
8 --- Pseudospalte: CURRVAL
9
10 --- Die CURRVAL Pseudospalte liefert den aktu
11 --- ellinen Wert der Sequenz. Der Wert entspr
12 --- ich dabei dem Wert, der beim letzten Au
13 --- ruf von SEQUENCE_NAME.NEXTVAL generiert
14 --- worden ist.
15
16 SELECT employee_seq.CURRVAL FROM DUAL;

```



## 2.2.8 Pseudospalte: rownum

Die rownum Funktion ordnet jedem Ergebnisdatensatz einer SQL Abfrage eine Numer entsprechend seiner Position im Ergebnis einer Abfrage zu.

### Query: Pseudospalte ROWNUM

```

1 --- Pseudospalte: ROWNUM
2
3 Das Ergebnis der Abfrage wird auf die
4 ersten 100 Datensätze beschränkt.
5
6 SELECT rownum, last_name, first_name
7   FROM employees
8  WHERE rownum < 101;

```

Operator	Beschreibung	Sql Beispiel
ROWNUM	Jedem Ergebnisdatensatz einer SQL Abfrage wird eine <code>SELECT * FROM employees WHERE ROWNUM &lt; 10</code> Nummer entsprechend seiner Position im Ergebnis der Abfrage zugeordnet.	
SYSDATE	Die Pseudospalten SYSDATE Funktion ermöglicht den Zugriff auf die interne Zeit des Datenbankserver.	<code>SELECT SYSDATE FROM DUAL</code>
TIMESTAMP	Die Pseudospalten TIMESTAMP Funktion ermöglicht den Zugriff auf die interne Zeit des Datenbankserver.	<code>SELECT TIMESTAMP FROM DUAL</code>
USER	Die USER Pseudospalte ermöglicht den Zugriff auf den Namen des eingeloggten Benutzers der gegenwärtigen Datenbanksession.	<code>SELECT USER, UID FROM DUAL</code>
UID	Die UID Pseudospalte ermöglicht den Zugriff auf die ID des eingeloggten Benutzers der gegenwärtigen Datenbanksession.	<code>SELECT USER, UID FROM DUAL</code>
CURRVAL	Die CURRVAL Pseudospalte liefert den aktuellen Wert der Sequenz. Der Wert entspricht dabei dem Wert, der beim letzten Aufruf von <code>SEQUENCE_NAME.NEXTVAL</code> generiert worden ist.	<code>SELECT employee_seq.CURRVAL FROM DUAL</code>
NEXTVAL	Die NEXTVAL Pseudospalte liefert den nächsten Wert der Sequenz.	<code>SELECT employee_seq.NEXTVAL FROM DUAL</code>

Abbildung 11. Pseudospalten

**2.2.9 Pseudospalten: user, uid**

Die USER bzw. UID Pseudospalten verweisen auf die Daten des eingeloggten Users.

» Query: Pseudospalten USER, UID ▾

```

1 -- -----
2 -- Pseudospalte: USER
3 --
4 -- Die USER Pseudospalte ermöglicht den
5 -- Zugriff auf den Namen des eingeloggten
6 -- Users.
7 
8 SELECT USER, EMPLOYEES_ID FROM employees;
9 --
10 -- Pseudospalte: UID
11 --
12 -- Die UID Pseudospalte ermöglicht den
13 -- Zugriff auf die ID des eingeloggten Benut-
14 -- zers der gegenwärtigen Datenbanksession.
15 
16 SELECT UID, LAST_NAME FROM employees;
```

**2.2.10 Pseudospalten: sysdate, timestamp**

Die SYSDATE bzw. TIMESTAMP Pseudospalte ermöglicht den Zugriff auf die interne Zeit des Datenbankservers.

» Query: Pseudospalte sysdate, timestamp ▾

```

1 -- -----
2 -- Pseudospalte: SYSDATE
3 --
4 -- Die SYSDATE Pseudospalte ermöglicht den
5 -- Zugriff auf die interne Zeit des Daten-
6 -- bankservers.
7 
8 SELECT SYSDATE FROM DUAL;
9 --
10 -- Pseudospalte: TIMESTAMP
11 --
12 -- Die TIMESTAMP Pseudospalte ermöglicht
13 -- den Zugriff auf die interne Zeit des
14 -- Datenbankservers.
15 
16 SELECT TIMESTAMP FROM DUAL;
```

39

**2.3. Case Klausel - Kondition**

Mit der case Klausel definiert die SQL Spezifikation einen logischen Operator, der das Ergebnis der Abfrage in konditionale Abhängigkeit zum Datenbestand setzt.

Die case Klausel wird stets vor der Klausel ausgewertet, in der sie definiert wurde.

**2.3.1 Case Klausel****Case Klausel - Kondition**

Die case Klausel ermöglicht eine konditionale Verarbeitung des Datenbestands einer SQL Abfrage.

Die Case Klausel zeigt dabei dasselle Verhalten wie der ? Operator imperativer Programmiersprachen.

» Syntax: Case Klausel ▾

```

1 -- -----
2 -- Syntax : CASE Klausel
3 --
4 CASE WHEN <bedingung> THEN <result>
5   WHEN <bedingung> THEN <result>
6   [WHEN <bedingung> THEN <result>
7   ...]
8   [ELSE <result>] +
9 END;
```

» Erklärung: Case Klausel ▾

Für eine Case Klausel können mehrere WHEN THEN Paare definiert werden. Jedes WHEN THEN Paar formuliert dabei eine Bedingung. Das Ergebnis der Auswertung der Case Klausel ist das Wert des ersten wahren WHEN THEN Paares.

Syntaktisch gelten für Bedingungen der case und where Klausel dieselben Regeln.

Der else Zweig der case Klausel kommt immer dann zur Ausführung, wenn keine der Bedingungen der vorangegangenen when Then Paare zutreffen. Wird kein else Zweig definiert, wird er als eine null angenommen.

**2.3.2 Case Klauseln in select Abfragen**

Die case Klausel zeigt im Vergleich zu anderen Klauseln mehr das Verhalten eines Operators als einer Klausel.

Ein Aufruf der case Klausel kann nur im Kontext einer anderen Klausel erfolgen.

» Query: Case Klausel ▾

```

1 -- -----
2 -- Fallbeispiele : CASE in ORDER BY
3 --
4 SELECT country_id, first_name, last_name
5   FROM employees
6   ORDER BY
7     CASE
8       WHEN location_id IS NULL THEN country_id
9       ELSE location_id
10      END;
11 
12 -- -----
13 -- Fallbeispiele : CASE in SELECT
14 --
15 SELECT first_name, last_name,
16   CASE
17     WHEN salary < 1000
18       THEN 'low income'
19     WHEN salary BETWEEN 1000 AND 20000
20       THEN 'medium income'
21     WHEN salary BETWEEN 20001 AND 40000
22       THEN 'upper income'
23     WHEN salary BETWEEN 40001 AND 80000
24       THEN 'ext'
25     ELSE null
26   END AS income
27   FROM employees
28   ORDER BY last_name, first_name;
29 
30 -- -----
31 -- Fallbeispiele : CASE in WHERE
32 --
33 SELECT first_name, last_name
34   FROM employees
35   WHERE salary >
36     CASE
37       WHEN department_id = 106 THEN 1000
38       WHEN department_id = 107 THEN 2000
39     END
40   ORDER BY last_name;
```

39

## 2.4. Where Klausel - Restriktion

### Where Klausel - Restriktion

Zur **Filtrierung** des Datenbestandes einer Abfrage können in der **where Klausel** Bedingungen definiert werden.

Die **where Klausel** ist **optional**.

### 2.4.1 Restriktion

Zur **Filtrierung** der Datensätzen einer Abfrage können in der **where Klausel** Bedingungen definiert werden.

Mit der **where Klausel** können **Restriktionen** für die Daten einer Abfrage definiert werden.

#### Erklärung: where Klausel

Innerhalb der **where Klausel** werden Bedingungen definiert, die für jedes Datensatz der Datenbasis geprüft werden.

Evaluiert die Prüfung einer Bedingung für einen Datensatz nicht zu true, wird der Datensatz aus der Datenbasis der Abfrage entfernt.

#### Query: where Klausel - Datenfilterung

```

1 -- WHERE Klausel
2
3
4 SELECT last_name, first_name, job_id,
5      department_id,
6      salary
7 FROM employees
8 WHERE job_id = 'SA_MAN'
9 ORDER BY last_name, first_name;
10
11 SELECT last_name, first_name
12 FROM employees
13 WHERE salary > 4000;
```



## 2.4.2 Abfrageobjekt

Ein **Abfrageobjekt** wird zum Filtern der Daten einer Abfrage verwendet.

Abfrageobjekte formulieren logische Bedingungen in Form von **mathematischen Aussagen**.

#### Erklärung: Abfrageobjekt

Eine Bedingung ist ein Ausdruck, der immer zu wahr oder falsch evauiert.

Abfrageobjekte können durch die **Verknüpfung** mathematischer Aussagen weiter präzisiert werden.

Zur Kombination mathematischer Aussagen werden logische Operatoren verwendet.

## 2.4.3 Logische Operatoren - and, or, not, xor

### Verknüpfungsoperatoren

Verknüpfungsoperatoren werden verwendet um logische **Bedingungen** miteinander zu **verknüpfen**. Die SQL Spezifikation definiert dazu die Operatoren and, or, not und xor.

#### Erklärung: Logische Operatoren

Zur Formulierung komplexer Bedingungen können die Operatoren and, or bzw not in einem Abfrageobjekt beliebig kombiniert werden.

#### Query: Logische Operatoren:

```

1 --
2 -- logische Operatoren: and, or, not
3
4 SELECT last_name, first_name, salary,
5      department_name,
6      department_id
7 FROM employees
8 WHERE
9   (job_id = 'CLERK' or job_id = 'MAN')
10  and
11  (salary > 100 or last_name = 'Hall')
12 ORDER BY last_name, first_name;
```

31

## 2.4.4 Logischer Operator - like

Der **like** Operator ermöglicht es Zeichenketten auf das Vorhandensein bestimmter Zeichenfolgen zu prüfen.

Zur Beschreibung einer Zeichenfolge wird eine sogenannte **Suchmaske** formuliert. Die SQL Spezifikation definiert dazu zwei Platzhalterzeichen.

#### Erklärung: Platzhalterzeichen

Der % Token steht stellvertretend für eine beliebige Zahl von Zeichen.

Der \_ Token wird als Platzhalter für ein beliebiges Zeichen verwendet.

#### Query: Patternmatching

```

1 -- -----
2 -- logischer Operator - like
3
4 SELECT last_name, first_name, department_id
5 FROM employees
6 WHERE last_name like 'M%';
7
8 SELECT last_name, first_name, job_id, salary
9 FROM employees
10 WHERE last_name like '_AX';
```

## 2.4.5 Logischer Operator - in

Mit dem **in** Operator kann geprüft werden, ob ein bestimmter Wert in einer Liste von Werten enthalten ist.

#### Query: In Operator

```

1 -- -----
2 -- logischer Operator - in
3
4 SELECT last_name, job_id, salary
5 FROM employees
6 WHERE job_id in ('SA_REP', 'SA_CLERK');
```

#### Analyse: in Operator

Wir suchen alle Mitarbeiter deren Berufsbeschreibung entweder SA\_REP oder SA\_CLERK ist.

Der **in** Operator kann in Verbindung mit numerischen bzw. alphanumerischen Werten verwendet werden.

## 2.4.6 Logischer Operator - is

Mit dem **is** Operator kann geprüft werden, ob der Wert einer bestimmten Spalte bekannt ist.

#### Erklärung: is Operator

Der Vergleich **last\_name = null** wird unabhängig von dem in der **last\_name** Spalte gespeichertem Wert immer false ergeben.

Der Grund dafür ist das der Vergleich **null = null** stets zu false evauiert.

Zur Prüfung auf null muss der **is** Operator verwendet werden.

#### Query: Prüfung auf null Werte

Um zu prüfen ob ein Wert in einem Intervall von Werten enthalten ist, wird der **between** Operator verwendet.

#### Query: between Operator

```

1 -- -----
2 -- logischer Operator - between
3
4 SELECT last_name, first_name, department_id
5 FROM employees
6 WHERE
7   salary between 10000 and 14000;
8
9 -- -----
10 -- logischer Operator - between
11
12 SELECT last_name, first_name, department_id,
13      department_id,
14      salary
15 FROM employees
16 WHERE birthday between
17   to_date('10.01.1960', 'dd.mm.yyyy')
18   and
19   to_date('10.01.1965', 'dd.mm.yyyy');
```

Operator	Beschreibung	Sql Beispiel
and	Die Verknüpfung logischer Terme mit einem and wird als first_name like 'A' and salary > 1000 Konjunktion bezeichnet. Eine Konjunktion ist wahr wenn jeder angegebene logische Term wahr ist.	salary < 0 or salary > 20000
or	Die Verknüpfung logischer Terme mit einem or wird als Disjunktion bezeichnet. Ein Disjunktion ist wahr wenn einer der angegebenen Terme wahr ist.	not salary = 2000
not	Die Negation ist ein Operator zur Vereinigung logischer Aussagen	department_id in (106, 107, 109)
in	Der In Operator prüft ob ein gegebener Wert in einer Liste von Werten enthalten ist.	first_name like %X%
like	Der Like Operator prüft abphonetische Werte auf das Vorhandensein von Mustern.	department_id is not null
is	Der Is Operator prüft ob der Wert für eine bestimmte Spalte bekannt ist.	salary between 10000 and 14000
between	Der Between Operator prüft ob ein Wert in einem Intervall von Werten enthalten ist.	33

Abbildung 12. Logische Operatoren

**2.4.8 Dreiwertige Logik**

Dreiwertige Logik ▾

Die SQL Spezifikation kennt zum Unterschied zu anderen Programmiersprachen 3 Wahrheitswerte: true, false und null.

Der null Wert kann mit nichts verglichen werden, er ist wahr falsch noch wahr sondern eben einfach unbekannt.

and	t	f	n	or	t	f	n
t	t	f	n	t	t	f	t
f	f	f	n	f	f	n	n
n	n	f	n	n	t	n	n

Mit dem null Wert verlassen wir die Welt der intuitiv erfassbaren Logik und führen eine dreiwertige Logik ein, die neben wahr und falsch noch unbekannt als Wahrheitswert kennt. Der null Wert kann mit nichts verglichen werden. Die Negation eines unbekannten Werts ist selbst wieder unbekannt.

Erklärung: Dreiwertige Logik ▾

- Wir möchten berechnen, wie hoch das tatsächliche Jahresgehalt eines Angestellten ist.
- Die Spalte commission\_pct enthält null Werte. Berechnungen auf null Werten sind problematisch. Wird z.B. zu einer Zahl eine unbekannte Zahl addiert, ist das Ergebnis wieder eine unbekannte Zahl.
- Zur Verarbeitung von Nullwerten definiert die SQL Spezifikation die coalesce Funktion.

Query: Mit Nullwerten arbeiten ▾

```
-- Mit Nullwerten arbeiten
-- SELECT last_name, first_name,
--        department_id,
--        (salary * 12) +
--        coalesce(commission_pct, 0.3)
-- FROM employees;
```

33

**2.5. Order by Klausel - Sortierung**

Order By Klausel - Sortierung ▾

Mit der order by Klausel kann eine Sortierung der Datensätze des Ergebnisses einer Abfrage definiert werden.

Die order by Klausel ist optional.

2.5.1 Order By Klausel

Mit der order by Klausel kann eine Sortierung der Datensätze einer Abfrage definiert werden.

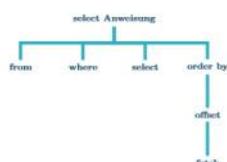
Brachten Sie, dass nur nach Spalten sortiert werden kann, die Teil des Ergebnisses der Abfrage sind.

Query: Einsatz der Order By Klausel ▾

```
-- ORDER BY Klausel
-- -----
-- SELECT last_name, first_name, department_id,
--        department_name,
--        job_id
-- FROM employees
-- WHERE department_id = 30
-- ORDER BY last_name;
```

Analyse: Order Klausel ▾

- Es werden alle Mitarbeiter der Abteilung 30 sortiert nach ihren Nachnamen, angegeben.
- Reihenfolge der Auswertung der Klauseln:



2.5.2 Sortierreihenfolge - asc/desc

Durch die Angabe des asc bzw. desc Schlüsselwort kann die Art der Sortierung zusätzlich präzisiert werden.

Query: Sortieren des Abfrageergebnisses ▾

```
-- ORDER BY Klausel - ACS, DESC
-- -----
-- SELECT last_name, first_name, department_id,
--        department_name,
--        salary
-- FROM employees
-- WHERE department_id = 30
-- ORDER BY last_name ASC, salary DESC;
```

Analyse: Sortierung ▾

- Im Beispiel wird das Abfrageergebnis zuerst aufsteigend nach den Werten der last\_name Spalte sortiert.
- Treten in der last\_name Spalte Datensätze mit gleichen Werten auf, werden diese Zeilen absteigend nach den Werten der salary Spalte sortiert.

2.5.3 Optionen - nulls first und nulls last

Eine letzte Erweiterung der order by Klausel betrifft die Behandlung von null Werten.

Erklärung: Null Werte ▾

- Mit dem nulls first bzw. nulls last Schlüsselwort werden Datensätze, die Null Werte enthalten, summarisch dem Ergebnis entweder voran bzw. nachgestellt.

Query: Null Werte ▾

```
-- ORDER BY Klausel - Null Werte
-- -----
-- SELECT last_name, first_name, department_id,
--        department_name,
--        job_id,
-- FROM employees
-- WHERE department_id = 30
-- ORDER BY last_name ASC,
--          nulls first;
```

34

Dipl.Ing.- Msc. Paul Paulhofer Inc.						
28	François Arman	(010) 700-0000	francoisarman@company.com	APPLIED	0012/18 11:09PM	<input checked="" type="checkbox"/> EDIT <input type="checkbox"/> DELETE
27	Hana Sung	(095) 120-0448	hsung@company.com	APPROVED	0012/18 9:08PM	<input checked="" type="checkbox"/> EDIT <input type="checkbox"/> DELETE
26	Angela Ziegler	(090) 907-5389	angela@company.com	APPROVED	0012/18 7:39PM	<input checked="" type="checkbox"/> EDIT <input type="checkbox"/> DELETE
24	Mos Ling Zhou	(090) 944-7777	mos@company.com	DRAINED	0012/18 4:46AM	<input checked="" type="checkbox"/> EDIT <input type="checkbox"/> DELETE

Abbildung 13. Paginierung von Daten

### 2.6. Fetch Klausel - Limitierung

**Fetch Klausel - Limitierung**

Mit der fetch Klausel kann das Ergebnis einer Abfrage auf eine bestimmte Zahl von Datensätzen beschränkt werden.

Die fetch Klausel ist optional.

Der Einsatz der fetch Klausel macht nur dann Sinn, wenn das Abfrageergebnis in sortierter Form vorliegt.

#### 2.6.1 Fetch Klausel

Mit der fetch Klausel wird das Ergebnis einer Abfrage auf die ersten n Datensätze beschränkt.

```
> Query: Einsatz der fetch Klausel
1 -- -----
2 -- -- FETCH Klausel - Anzahl der Datensätze
3 --
4 SELECT last_name, first_name, department_id
5 FROM employees
6 ORDER BY salary DESC
7 FETCH FIRST 5 ROWS ONLY;
8
9 -- -----
10 -- FETCH Klausel - Prozentangabe
11 --
12 SELECT last_name, first_name, job_id
13 FROM employees
14 ORDER BY salary DESC
15 FETCH FIRST 20 PERCENT ROWS ONLY;
```

### 2.7. Offset Klausel - Paginierung

Mit der offset Klausel kann ein seitenweiser Zugriff auf das Ergebnis einer Abfrage definiert werden.

#### 2.7.1 Offset Klausel

Mit der offset Klausel können die ersten n Datensätze aus dem Ergebnis einer Abfrage entfernt werden.

+ Query: offset Klausel

```
> Query: offset Klausel
1 --
2 -- OFFSET Klausel
3 --
4 -- Paginierung des Abfrageergebnisses
5 -- Ergebnis: Datensätze 21 - 40
6 SELECT last_name, first_name, middle_name,
7      department_name,
8      department_id,
9      salary
10    FROM employees
11   ORDER BY last_name ASC,
12          first_name ASC
13   OFFSET 20 ROWS
14   FETCH NEXT 20 ROWS ONLY;
```

+ Erklärung: Offset Klausel

- Die offset Klausel zusammen mit der fetch Klausel werden verwendet um eine Paginierung des Abfrageergebnisses zu definieren.
- Eine Paginierung des Abfrageergebnisses erlaubt den seitenweisen Zugriff auf die Datensätze einer Abfrage.
- Vermeiden Sie in SQL Abfragen den Zugriff auf alle Datensätze einer Tabelle!

35

### 3. SQL - Datenaggregation

# 02

Datenaggregation

01. Datenmodellierung	36
02. From Klausel	37
03. Relationale Join	37

Informationssysteme

### 3.1. Datenmodellierung

#### 3.1.1 Relationale Modellierung

**Relationale Modell**

Das Relationale Modell beschreibt die grundlegende Tabellen- bzw. Beziehungsstruktur einer relationalen Datenbank.

+ Erklärung: Relationale Modellierung

- Bei der Arbeit mit relationalen Datenbanken werden die Geschäftssobjekte einer Anwendung auf eine Menge von Tabellen abgebildet.
- Jede Tabelle repräsentiert dabei einen bestimmten Aspekt der Wirklichkeit.
- Das Aufteilen der Daten auf einzelne Tabellen ermöglicht eine klare Strukturierung der Daten in der Datenbank.
- Beim Formulieren einer SQL Abfrage stehen wir oft vor der Herausforderung, die Daten wieder so zusammenzustellen, dass das ursprüngliche Geschäftssobjekt zum Vorschein kommt.

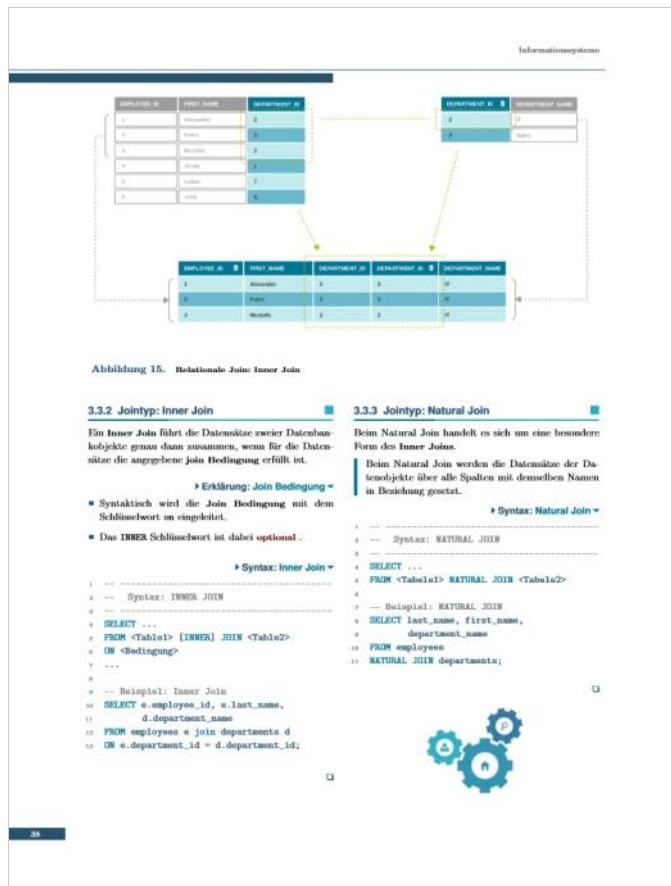
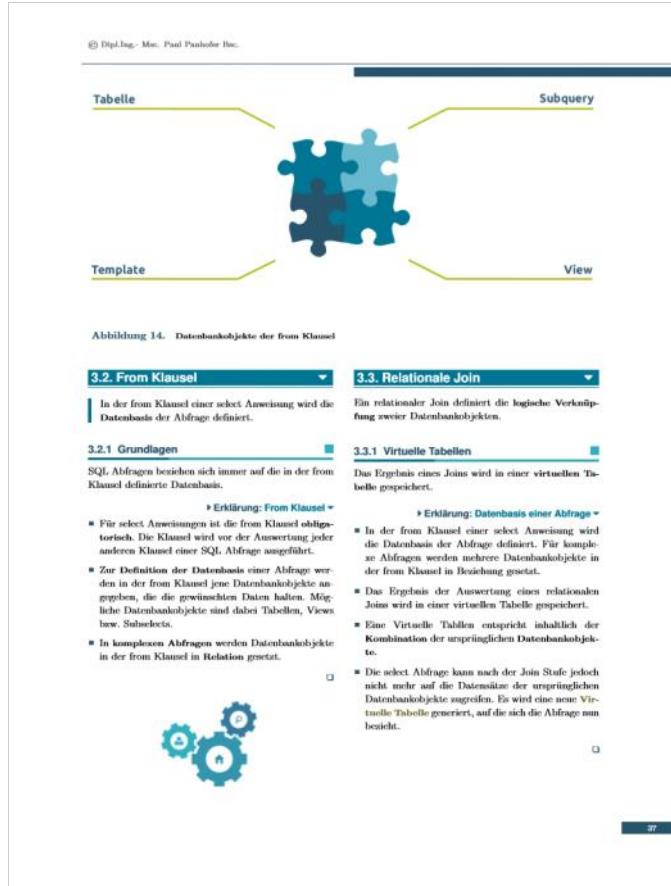
### 3.1.2 Datenaggregation

Datenaggregation beschreibt den Prozess der Verdichtung einzelner Aspekte der Geschäftsdaten zu neuen Geschäftssobjekten.

+ Erklärung: Datenaggregation

- Mit dem Aufteilen der Daten auf getrennte Aspekte, können Datensätze ihrerseits wiederum zu neuen Geschäftsobjekten zusammengefasst werden.
- Durch die geschickte Kombination der Aspekte eines Geschäftsfalls kann aus den gegebenen Daten neue Information gewonnen werden.
- Die technische Grundlage der Datenaggregation ist der **relationale Join**. Ein relationaler Join beschreibt die logische Verbindung zweier Datensätze. Relationale Datenbanken unterstützen dabei mehrere Formen des relationalen Joins.
- In SQL Abfragen werden relationale Joins in der Klausel definiert.

36



Jointyp	Beschreibung	Seite
CROSS JOIN	Ein cross join führt Datensätze zweier Datenbankobjekte zusammen, indem jeder Datensatz des ersten Objekts mit jedem Datensatz des zweiten Objekts in Beziehung gesetzt wird.	39
INNER JOIN	Ein Inner Join führt die Datensätze zweier Datenbankobjekte genau dann zusammen, wenn für die Datensätze die angegebene join Bedingung erfüllt ist.	38
NATURAL JOIN	Beim Natural Join handelt es sich um eine besondere Form des Inneren Joins. Beim Natural Join werden die Datensätze der Datenobjekte über alle Spalten mit denselben Namen in Beziehung gesetzt.	38
LEFT/RIGHT JOIN	Ein Left Join übernimmt alle Datensätze des ersten Datenobjekts und setzt sie mit korrespondierenden Datensätzen des zweiten Datenobjekts in Beziehung.	41
FULL OUTER JOIN	Der Full Outer Join entspricht semantisch einer Kombination aus Left- und Rightjoin.	39

Abbildung 16. Jointypen

## 3.3.4 Jointyp: Cross Join

Ein cross join führt Datensätze zweier Datenbankobjekte zusammen, indem jeder Datensatz des ersten Objekts mit jedem Datensatz des zweiten Objekts in Beziehung gesetzt wird.

## » Erklärung: Cross Join »

- Das Ergebnis eines cross joins enthält damit  $m \cdot n$  Zeilen wenn die eine Tabelle  $m$  und die andere Tabelle  $n$  Zeilen enthält.

## » Syntax: Cross Join »

```

1 -- Syntax: CROSS JOIN
2 -- -----
3
4 SELECT ...
5 FROM <Tabelle1> CROSS JOIN <Tabelle2>
6 ...
7
8 -- employeee.zeile1 <-> departments.zeile1
9 -- employeee.zeile1 <-> departments.zeile2
10 -- ...
11 -- ...
12 -- employeee.zeile2 <-> departments.zeile1
13 -- ...
14 -- employeee.zeileN <-> departments.zeileN
15
16 -- Beispiel: Cross Join
17 SELECT e.first_name, d.department_name
18 FROM employees e
19 CROSS JOIN departments d;
  
```

## 3.3.5 Jointyp: Full Outer Join

Der Full Outer Join entspricht semantisch einer Kombination aus Left- und Rightjoin.

## » Erklärung: Full Outer Join »

- Mengentheoretisch entspricht der Full outer Join zweier Mengen A und B der Vereinigungsmenge der Mengen.
- Der full outer join zweier Mengen als Relation ist kommutativ.

## » Syntax: Full Outer Join »

```

1 -- Syntax: FULL OUTER JOIN
2 -- -----
3
4 SELECT ...
5 FROM <Tabelle1> FULL OUTER JOIN <Tabelle2>
6 ON <Bedingung>
7 ...
8
9 -- Beispiel: FULL OUTER JOIN
10 -- ...
11 -- ...
12 -- ...
13 -- ...
14 -- ...
15 -- ...
16 -- ...
17 -- ...
18 -- ...
19 -- ...
  
```

39

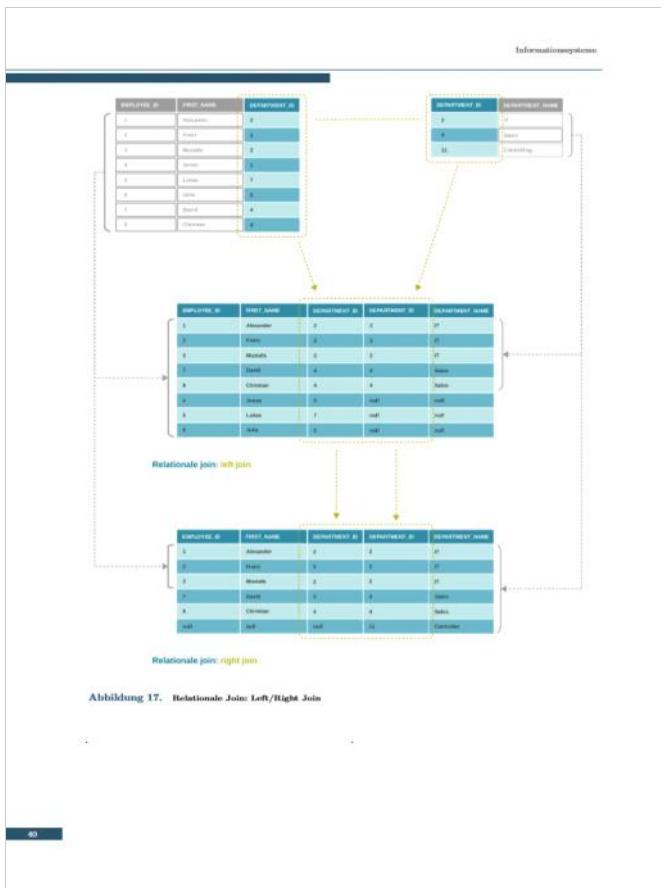


Abbildung 17. Relationale Join: Left/Right Join

### 3.3.6 Jointyp: Left-/Right Join

Ein Left Join übernimmt alle Datensätze des ersten Datenobjekts und setzt sie mit korrespondierenden Datensätzen des zweiten Datenobjekts in Beziehung.

Left- und Right Join beschreiben denselben Jointyp, in jeweils gespiegelter Form.

#### Erklärung: Left-/Right Join ▾

- Mengentheoretisch entspricht der left join zweier Tabellen A und B der Menge A.
- Die Syntax ist dabei bis auf die Schlüsselwörter left bzw. right identisch zur Syntax des Inner Join.
- Der left- bzw. right Join ist keine kommutative Relation.

#### Syntax: Left Join/Right Join ▾

```
1 -- -----
2 -- Syntax: LEFT JOIN
3 -----
4 SELECT ...
5 FROM <Table1> LEFT JOIN <Table2>
6 ON <Bedingung>
7 ...
8
9 -- Beispiel: Left Join
10 SELECT e.first_name,
11      e.last_name,
12      e.salary,
13  FROM employees e
14  LEFT JOIN departments d
15  ON e.department_id = d.department_id;
16
17 --
18 -- Syntax: RIGHT JOIN
19 --
20 SELECT ...
21 FROM <Table1> RIGHT JOIN <Table2>
22 ON <Bedingung>
23 ...
24
25 -- Beispiel: Right Join
26 SELECT e.first_name,
27        e.last_name
28        e.salary,
29  FROM employees e
30  RIGHT JOIN departments d
31  ON e.department_id = d.department_id;
```

41

## 4. SQL - Zeilenfunktionen



Zeilenfunktionen

01. Grundlagen: Zeilenfunktionen	42
02. Datumsfunktionen	43
03. Textfunktionen	50
04. Numerische Funktionen	53

### 4.1. Zeilenfunktionen - Grundlagen ▾

#### 4.1.1 Funktionstypen ▾

Die SQL Spezifikation definiert 2 Arten von Funktionen:

##### ➤ Auflistung: Arten von Funktionen ▾

###### Zeilenfunktionen ▾

Zeilenfunktionen verarbeiten die Daten eines einzelnen **Datensatzes**. Der Rückgabewert einer Zeilenfunktion ist stets ein einzelner Wert.

###### Aggregatfunktionen ▾

Aggregatfunktionen verdichten mehrere **Datensätze** zu einem einzelnen Wert.

#### 4.1.2 Zeilenfunktionen ▾

Syntaktisch gesehen haben alle Zeilenfunktionen gemeinsam, dass sie die Daten eines einzelnen Datensatzes, zu einem einzelnen Wert verdichten.

##### ➤ Erklärung: Zeilenfunktionen ▾

- Die SQL Spezifikation definiert eine Reihe von Zeilenfunktionen.

```
1 -- -----
2 -- Zeilenfunktionen
3 --
4 -- Die Funktion gibt das aktuelle Datum
5 -- das Datenbankservers zurück
6 -- Da die FROM Klammer für eine select
7 -- Anfrage obligatorisch ist wird die
8 -- Pseudotabelle dual verwendet
9 -- SELECT sysdate FROM dual;
10
11 -- Funktion zum Berechnen des aktuellen
12 -- Zeitpunkts
13 -- SELECT systimestamp FROM dual;
14
15 -- Funktion zum Berechnen der Länge eines
16 -- alphanumerischen Wertes
17 -- SELECT length(last_name) FROM employees;
```

42

- Ein Großteil, der in der SQL Spezifikation definierten Zeilenfunktionen erwartet dabei Funktionsparameter.

- Funktionsparameter sind Werte, die einer Funktion zur Verarbeitung mitgegeben werden.

```

1 -- -----
2 -- Funktionsparameter
3 --
4 SELECT lower(ename) ergebnis FROM emp;

```

#### 4.1.3 Kategorien von Zeilenfunktionen

Die SQL Spezifikation unterscheidet mehrere Kategorien von Zeilenfunktionen.

##### Auflistung: Kategorien von Zeilenfunktionen

- Datumsfunktionen: Funktionen zum Bearbeiten zeitbezogener Daten.

```

1 -- -----
2 -- Datumsfunktionen
3 --
4 FUNCTION to_date (
5   p_time_literal IN VARCHAR2,
6   p_time_mask IN VARCHAR2
7 )
8 RETURN DATE;

```

- Zeichenfunktionen: Funktionen zum Bearbeiten von Zeichenketten.

```

1 -- -----
2 -- Zeichenfunktion: instr
3 --
4 FUNCTION instr (
5   p_content IN VARCHAR2,
6   p_token IN VARCHAR2,
7   p_start_index IN INTEGER
8 )
9 RETURN NUMBER;

```

- Mathematische Funktionen: Funktionen zur Transformation von Zahlenwerten.

- Konvertierungsfunktionen: Funktionen zum Konvertieren von Werten eines Datentyps zu einem anderen Datentyp.

#### 4.2 Datumsfunktionen

Datumsfunktionen werden zur Verarbeitung zeitbezogener Daten verwendet.

##### Auflistung: Datumsfunktionen

```

1 -- -----
2 -- Datumsfunktionen
3 --
4 FUNCTION to_date (
5   p_time_literal IN VARCHAR2,
6   p_time_mask IN VARCHAR2
7 )
8 RETURN DATE;

```

##### 4.2.1 Datumstypen vs. Zeichenketten

Datumswerte

Datumswerte werden verwendet um zeitbezogene Werte in einer Datenbank abzubilden.

Die SQL Engine speichert Datumswerte dabei als die Anzahl von Millisekunden, die seit dem 01.01.1972 vergangen sind.

##### Erklärung: Darstellung von Datumstypen

- Bevor ein Datumswert angezeigt werden kann, muss er durch die Datenbankengine entsprechend formatiert werden.

- Durch die Angabe sogenannter Masken bestimmt der Benutzer die Art der Formatierung für die Ausgabe eines Datumswertes.

43

#### 4.2.2 Erzeugen eines Datums

Zum Erzeugen von Datumswerten definiert die SQL Spezifikation 3 Möglichkeiten.

##### Auflistung: Erzeugen eines Datums

Konstruktorfunktionen

Funktionen zum direkten Erzeugen von Datumswerten.

Konvertierungsfunktionen

Funktionen zum Konvertieren von Zeichenketten in Datumswerte.

Literale

Erzeugen von Datumswerten aus Literalen.

#### 4.2.4 Konvertieren von Datumswerten - to\_date

Die to\_date() Funktion konvertiert Zeichenketten in Datumswerte. Die Funktion erwartet beim Aufruf 2 Parameter.

##### Parameter: to\_date Funktion

- p\_time\_literal: Der Parameter beschreibt einen Datumswert.

- p\_time\_mask: Der Parameter beschreibt eine Formatmaske.

##### Syntax: to\_date Funktion

```

1 -- -----
2 -- Syntax: to_date
3 --
4 FUNCTION to_date (
5   p_time_literal IN VARCHAR2,
6   p_time_mask IN VARCHAR2
7 )
8 RETURN DATE;
9 --
10 -- Aufruf: to_date()
11 SELECT to_date(
12   '15.06.2012 17:30:56',
13   'dd.mm.yyyy hh24:mi:ss'
14 )
15 FROM dual;

```

#### 4.2.3 Konstruktorfunktionen

Die SQL Spezifikation definiert die sysdate und systimestamp Funktionen zum Erzeugen von Datumswerten.

Die sysdate und systimestamp Funktionen werden ohne Parameter aufgerufen, und liefern als Ergebnis die aktuelle Zeit des Datenbankservers.

##### Query: Konstruktorfunktionen

```

1 -- -----
2 -- Konstruktorfunktionen
3 --
4 -- Generieren eines Date Datumswertes. Es
5 -- wird das aktuelle Datum ausgegeben.
6 SELECT sysdate FROM dual;
7 
8 -- Ausgabe: 2018-10-30 18:18:27
9 
10 -- Generieren eines Timestamp Datumswertes.
11 -- Es wird das aktuelle Datum ausgegeben.
12 SELECT systimestamp FROM dual;
13 
14 -- Ausgabe:
15 -- 2018-10-30 18:18:27.590964 +01:00

```

##### Query: Datumsliterale

Datumswerte selbst, können ebenfalls aus einem Literal generiert werden. Ein Datumsliteral muss dabei dem ISO Datumsformat entsprechend aufgebaut sein.

```

1 -- -----
2 -- Literale
3 --
4 -- Literal: yyyy-mm-dd
5 SELECT date '2012-06-15' datum
6 FROM dual;
7 
8 -- Literal: yyyy-mm-dd hh24:mi:ss
9 SELECT timestamp '2012-06-15 15:00:00' datum
10 FROM dual;

```

44

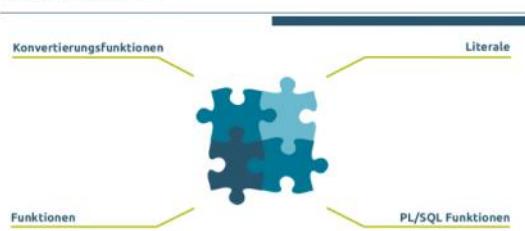


Abbildung 18. Erzeugen von Datumsdatenwerten

**4.2.6 Datumsdaten umwandeln - to\_char()**

Die `to_char()` Funktion konvertiert Datumsdaten in Zeichenketten.

» Parameter: `to_char` Funktion \*

- `p_date`: Der Parameter beschreibt einen Datums-
- wert.

- `p_time_mask`: Der Parameter beschreibt eine For-
- matschablade zur Formatierung eines Datums-
- werts.

» Syntax: `to_char` Funktion \*

```

1 -- -----
2 -- Syntax: to_char
3 --
4 FUNCTION to_char(
5   p_date      IN DATE,
6   p_time_mask IN VARCHAR2
7 )
8 RETURN VARCHAR2;
9 
10 --
11 -- Datumsdaten umwandeln
12 --
13 SELECT to_char( hiredate, 'dd.mm.yyyy')
14   FROM employees
15 WHERE employee_id = 7839;
16 
17 -- Ausgabe: 10.09.2006

```

45

**4.2.7 Rundungsfunktionen - trunc(), round()**

Die SQL-Spezifikation definiert 2 Funktionen zum Runden von Datumsdaten: `trunc()` und `round()`.

■ `p_time`: mask: Der Parameter unterscheidet sich ausschließlich in ihrem Rundungsverhalten. Beide Funktionen arbeiten mit denselben Parametern.

» Erklärung: `trunc`, `round` Funktion \*

- Die `trunc()` Funktion wird verwendet, um Datums-
- werte zu runden.

- Das Rundungsverhalten der Funktionen wird dabei über Parameterwerte gesteuert.
  - Kein Rundungsparameter: Das Datum wird auf '00:00:00' gesetzt.
  - MM: Das Datum wird auf den 1.ten des Monats gerundet.
  - Q: Das Datum wird auf das Quartal zurückge-
  - setzt in das das Datum fällt.
  - Y: Das Datum wird auf den 1ten Tag des Jahres gerundet.

■ Die `round` und `trunc` Funktion sind bis auf ihr Rundungsverhalten identisch. Die `round` Funktion fun-

det im Gegensatz zur `trunc` Funktion kaufmännisch.

Informationssysteme		
Funktion	Beschreibung	Seite
<code>add_months()</code>	Die <code>add_months</code> Funktion arbeitet analog zur Datumsarithmetik und erlaubt die Addi- tion von Monaten zu einem Datumswert.	48
<code>extract()</code>	Die <code>extract</code> Funktion erlaubt die Extraktion bestimmter Teile aus einem Datum.	48
<code>last_day()</code>	Die <code>last_day</code> Funktion berechnet den letzten Tag des Monats in dem das Datum fällt.	48
<code>months_between()</code>	Die <code>months_between</code> Funktion berechnet die Differenz zweier Datumsdaten. Der Wert wird dabei auf Monate gerundet. Dabei werden Schaltjahre und Monatslängen berücksichtigt.	49
<code>next_day()</code>	Die <code>next_day</code> Funktion berechnet für ein gegebenes Datum das Datum des nachfol- genden Tages.	49
<code>to_char()</code>	Die <code>to_char</code> Funktion konvertiert eine Datumsangabe in eine Zeichenkette.	45

Abbildung 19. Datumsfunktionen

» Syntax: `trunc`, `round` Funktion \*

4.2.8 Erzeugen von Intervallen

Die SQL-Spezifikation erlaubt neben dem Verarbeiten von Datumsdaten auch das Arbeiten mit Zeitintervallen.

» Erklärung: Erzeugen von Intervallen \*

- Die SQL-Spezifikation erlaubt neben der Berechnung von Zeitangaben auch das Arbeiten mit Zeitintervallen.

- Zeitintervalle werden durch die Angabe von Lite-
- ralen definiert.

» Query: Erzeugen von Intervallen \*

```

1 -- -----
2 -- Syntax: trunc(), round()
3 --
4 FUNCTION trunc (
5   p_date IN DATE,
6   p_mask IN VARCHAR2 DEFAULT ''
7 )
8 RETURN DATE;
9 
10 FUNCTION round (
11   p_date IN DATE,
12   p_mask IN VARCHAR2 DEFAULT ''
13 )
14 RETURN DATE;
15 
16 --
17 -- trunc-Funktion
18 --
19 SELECT
20   trunc(to_date('12.06.2017'))  hiredate,
21   trunc(to_date('12.05.2017'), 'MM') month,
22   trunc(to_date('12.05.2017'), 'Q') quarter,
23   trunc(to_date('12.05.2017'), 'Y') year
24 FROM dual;
25 
26 -- Ausgabe
27 12.05.2017 00:00:00 -- hiredate
28 01.05.2017 00:00:00 -- month
29 01.04.2017 00:00:00 -- quarter
30 01.01.2017 00:00:00 -- year

```

46

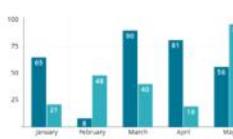
Skript Page 193

Intervall	Beschreibung	Seite
year to month	Das Intervall repräsentiert ein Zeitintervall aus Jahren und Monaten.	
day to minute	Das Intervall repräsentiert ein Zeitintervall. Der Wert beschreibt ein Zeitintervall von Tagen, Stunden und Minuten.	
day to second	Das Intervall repräsentiert ein Zeitintervall. Der Wert beschreibt ein Zeitintervall von Tagen, Stunden, Minuten und Sekunden.	

Abbildung 20. Datumsfunktionen

**4.2.9 Geschachtelte Intervalle**

Die SQL-Spezifikation erlaubt die Definition geschachtelter Intervalle. Die Intervalle werden als Literale definiert.



darstellt mit einem Ganzzahlanteil, der die Anzahl der Tage seit einem Startdatum wieder gibt, und einen Nachkommanteil, der den Anteil des Tages benennt, der bereits vergangen ist.

## » Erklärung: Datumsarithmetik »

- Die SQL-Spezifikation erlaubt das Rechnen mit Datumswerten. Durch die Angabe von Intervallen können einfache Berechnungen mit Datumswerten durchgeführt werden.
- Dabei kümmert sich die Datenbankengine um Fähigkeiten wie wieviele Tage ein bestimmtes Monat hat oder ob ein Jahr ein Schaltjahr ist.
- Die **Addition** ist dabei eine arithmetische Operation auf einem Datumswert und einem Zeitintervall.
- Die **Subtraktion** ist die einzige sinnvolle direkte arithmetische Operation auf 2 Datumswerten.
- Im Fall der Subtraktion des Datentyp `date` ist das Ergebnis eine Zahl.
- Im Fall der Subtraktion des Datentyp `timestamp` ein Intervall.

## » Query: Datumsarithmetik »

**4.2.10 Datumsarithmetik**

Das Rechnen mit Datumswerten ist dann leicht, wenn wir uns vergegenwärtigen, daß ein Datum eine Zahl

```

1 -- -----
2 -- Geschachtelte Intervalle
3 --
4 SELECT
5   interval '2-11' year to month interval_1,
6   interval '4 3:30' day to minute interval_2,
7   interval
8   '4 3:30:45' day to second interval_3
9  FROM dual;
10
11 -- Ausgabe
12 2-11    -- interval_1
13 3:30:0.0 -- interval_2
14 3:30:45.0 -- interval_3

```

## » Query: Datumsarithmetik »

## » Syntax: Datumsarithmetik »

## » Query: Datumsarithmetik »

## » Syntax: Datumsarithmetik »

47

**4.2.11 Datumsfunktion: add\_months()**

Die `add_months()` Funktion arbeitet analog zur Datumsarithmetik und erlaubt die Addition von Monaten zu einem Datumswert.

## » Parameter: add\_months Funktion »

- Die Funktion erwartet 2 Parameter. `add_months` arbeitet mit Datumswerten vom Typ `Date` bzw. `Timestamp` und liefert als Ergebnis jedoch immer einen Wert vom Typ `Date` zurück.
- Ein negativer Wert der Monatsangabe subtrahiert die entsprechende Anzahl von Monaten.

## » Syntax: add\_months Funktion »

```

1 -- -----
2 -- Syntax: add_months
3 --
4 FUNCTION add_months (
5   p_date IN DATE,
6   p_month_amount IN INTEGER
7 )
8 RETURN DATE;
9
10 -- -----
11 -- Aufruf: add_months
12
13 SELECT add_months(trunc(sysdate), 3)
14 FROM dual;

```

**4.2.13 Datumsfunktion: extract()**

Die `extract()` Funktion ermöglicht die Extraktion bestimmter Teile eines Datumswertes.

## » Syntax: extract Funktion »

```

1 -- -----
2 -- Syntax: extract
3 --
4 FUNCTION extract (
5   p_params IN EXPRESSION
6 )
7 RETURN INTEGER;
8
9 -- -----
10 -- Datumsfunktion: extract
11
12 SELECT extract(year from sysdate) year,
13       extract(month from sysdate) month,
14       extract(day from sysdate) day
15  FROM dual;
16
17 2017 -- year
18 7 -- month
19 30 -- day

```

**4.2.14 Datumsfunktion: last\_day()**

Die `last_day()` Funktion berechnet den letzten Tag des Monats in dem der angegebene Datumswert fällt.

## » Syntax: last\_day Funktion »

```

1 -- -----
2 -- Syntax: last_day
3 --
4 FUNCTION last_day (
5   p_date IN DATE
6   , p_day IN VARCHAR2
7 )
8 RETURN DATE_EXP;
9
10 SELECT last_day(sysdate, 'Freitag')
11 FROM dual;

```

## » Syntax: last\_day Funktion »

## » Syntax

**4.2.15 Datumsfunktion: next\_day()**

Die `next_day()` Funktion berechnet für ein gegebenes Datum, das Datum des nachfolgenden Tages.

► Syntax: `next_day` Funktion ▾

```

1 -- -----
2 -- Syntax: next_day
3 --
4 FUNCTION next_day (
5   p_date_1 IN DATE,
6   p_day     IN VARCHAR2
7 )
8 RETURN DATE_EXP;
9
10 -- Aufruf
11 SELECT next_day(sysdate, 'Freitag')
12 FROM dual;

```

**4.2.16 Datumsfunktion: months\_between()**

Die `months_between()` Funktion berechnet wieviel Monate zwischen 2 Datumswerten liegen. Dabei werden Schaltjahre und Monatslängen berücksichtigt.

► Parameter: `months_between` Funktion ▾

- Als Parameter erwartet die Funktion zwei Datumsangaben, wobei das spätere Datum als erstes übergeben werden sollte.
- Die Funktion eignet sich in erster Linie zur Berechnung von Zeitdauern.

► Syntax: `months_between` Funktion ▾

```

1 -- -----
2 -- Syntax: months_between
3 --
4 FUNCTION months_between (
5   p_date_1 IN DATE,
6   p_date_2 IN DATE
7 )
8 RETURN DATE_EXP;
9
10 -- Aufruf
11 SELECT trunc(
12   months_between( sysdate, hiredate )
13 )
14 FROM employees;

```

**4.3. Textfunktionen**

Die SQL Spezifikation definiert eine Zahl von Funktionen zur Verarbeitung von Zeichenketten.

**4.3.1 Textfunktion: instr()**

Die `instr()` Funktion prüft, ob ein **Token** in einer Zeichenkette enthalten ist und gibt die Position bei einer Übereinstimmung zurück.

► Parameter: `instr()` Funktion ▾

- `p_content`: Der Parameter beschreibt die zu durchsuchende Zeichenkette.
- `p_token`: Der Parameter beschreibt den Token nach dem gesucht wird.
- `p_start_index`: Der Parameter beschreibt die Position ab der die Zeichenkette durchsucht werden soll. Der Parameter ist optional und ist defautlmäßig mit dem Wert 1 initialisiert.
- `p_count`: Der Parameter definiert welches Vorkommen des Tokens ermittelt werden soll. Der Parameter ist optional und ist defautlmäßig mit dem Wert 1 initialisiert.

► Syntax: `instr` Funktion ▾

```

1 -- -----
2 -- Syntax: instr
3 --
4 FUNCTION instr (
5   p_content  IN VARCHAR2,
6   p_token    IN VARCHAR2,
7   p_start_index IN INTEGER DEFAULT 1,
8   p_count    IN INTEGER DEFAULT 1
9 )
10 RETURN NUMBER;
11
12 -- -----
13 -- Textfunktion: instr
14
15 SELECT
16   instr('/home/foo.txt', '/') pos_1,
17   instr('/home/foo.txt', '/', 1, 2) pos_2
18 FROM dual;
19
20 -- Ausgabe
21 1 -- pos_1
22 6 -- pos_2

```

**4.3.2 Textfunktionen: trim()**

Die `trim()` Funktion wird verwendet um **Tokens** am Ende bzw. am Anfang einer Zeichenkette zu entfernen. Die SQL Spezifikation definiert mehrere Formen der `trim()` Funktion:

► Syntax: `trim(), ltrim(), rtrim()` ▾

```

1 -- -----
2 -- Syntax: trim, ltrim, rtrim
3 --
4 FUNCTION ...trim... (
5   p_text  IN VARCHAR2,
6   p_token IN VARCHAR2
7 )
8 RETURN VARCHAR2;
9
10 -- -----
11 -- Textfunktion: ltrim, rtrim
12 -- -----
13 SELECT ltrim('<Das ist ein Element>','<')
14 FROM dual;
15
16 -- Ausgabe
17 Das ist ein Element/>
18
19 -- rtrim
20 SELECT rtrim('<Das ist ein Element>',' />')
21 FROM dual;
22
23 -- Ausgabe
24 <Das ist ein Element
25
26 -- Textfunktion: trim
27 -- -----
28 SELECT
29   trimboth(' ',' from '...SMITH...') text_1,
30   trimleading(' ', ' from '...SMITH...') text_2,
31   trim(trailing ' ' from '...SMITH...') text_3
32 FROM dual;
33
34 -- Ausgabe
35 SMITH -- text_1
36 SMITH.. -- text_2
37 ...SMITH -- text_3
38
39 -- SELECT trim(' SMITH ') FROM dual;
40
41 -- Ausgabe
42 SMITH

```

Funktion	Beschreibung	Seite
instr()	Die instr() Funktion prüft, ob ein Token in einer Zeichenkette enthalten ist und gibt die Position bei einer Übereinstimmung an.	50
length()	Mit der length Funktion wird die Länge einer Zeichenkette berechnet. Der Rückgabewert ist dabei als die Anzahl von bytes bzw. die Anzahl von Zeichen zu lesen.	51
lower()	Mit der lower() und upper() Funktion werden Zeichenfolgen transformiert.	52
replace()	Mit der replace Funktion wird eine Zeichenkette in einem Text durch eine andere Zeichenkette ersetzt.	52
soundex()	Die soundex Funktion führt einen Vergleich zwischen Zeichenketten durch. Der Vergleich bewertet die phonetische Ähnlichkeit von Zeichenketten.	51
substr()	Die substr Funktion wird verwendet um aus einer Zeichenkette einen Teilstring zu extrahieren.	52
trim()	Mit den unterschiedlichen Formen der trim Funktion werden am Ende bzw. Anfang einer Zeichenkette Zeichenfolgen entfernt.	50
upper()	Mit der lower() und upper() Funktion werden Zeichenfolgen transformiert.	52

Abbildung 21. Textfunktionen

**4.3.3 Textfunktion: length()**

Mit Hilfe der length() Funktion wird die Länge einer Zeichenkette bestimmt. Der Rückgabewert kann dabei in Form von bytes bzw. als Anzahl von Zeichen angegeben werden.

## » Syntax: length Funktion ▾

```

1 -- -----
2 -- Syntax: length
3 --
4 FUNCTION length (
5   p_text IN VARCHAR2
6 )
7 RETURN INTEGER;
8
9 --
10 -- Textfunktion: length
11 --
12 SELECT
13   length ('Torontomtom') zeichen,
14   length('Torontomtom') byte
15 FROM dual;
16
17 -- Ausgabe
18 11 -- zeichen
19 15 -- byte
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
787
788
789
789
790
791
792
793
794
795
796
797
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
846
847
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
869
869
870
871
872
873
874
875
876
877
878
879
879
880
881
882
883
884
885
886
887
887
888
889
889
890
891
892
893
894
895
896
897
897
898
899
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
917
917
918
919
920
921
922
923
924
925
926
927
928
929
929
930
931
932
933
934
935
936
937
938
939
939
940
941
942
943
944
945
946
947
947
948
949
949
950
951
952
953
954
955
956
957
958
959
959
960
961
962
963
964
965
966
967
968
969
969
970
971
972
973
974
975
976
977
978
979
979
980
981
982
983
984
985
986
987
987
988
989
989
990
991
992
993
994
995
996
997
998
999
999
1000
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1009
1010
1011
1012
1013
1014
1015
1016
1017
1017
1018
1019
1019
1020
1021
1022
1023
1024
1025
1026
1027
1027
1028
1029
1029
1030
1031
1032
1033
1034
1035
1036
1037
1037
1038
1039
1039
1040
1041
1042
1043
1044
1045
1046
1046
1047
1048
1048
1049
1050
1051
1052
1053
1054
1055
1056
1056
1057
1058
1058
1059
1060
1061
1062
1063
1064
1065
1065
1066
1067
1067
1068
1069
1069
1070
1071
1072
1073
1074
1075
1075
1076
1077
1077
1078
1079
1079
1080
1081
1082
1083
1084
1085
1085
1086
1087
1087
1088
1089
1089
1090
1091
1092
1093
1094
1094
1095
1096
1096
1097
1098
1098
1099
1100
1101
1102
1102
1103
1104
1104
1105
1106
1106
1107
1108
1108
1109
1110
1110
1111
1112
1112
1113
1114
1114
1115
1116
1116
1117
1118
1118
1119
1120
1120
1121
1122
1122
1123
1124
1124
1125
1126
1126
1127
1128
1128
1129
1130
1130
1131
1132
1132
1133
1134
1134
1135
1136
1136
1137
1138
1138
1139
1140
1140
1141
1142
1142
1143
1144
1144
1145
1146
1146
1147
1148
1148
1149
1150
1150
1151
1152
1152
1153
1154
1154
1155
1156
1156
1157
1158
1158
1159
1160
1160
1161
1162
1162
1163
1164
1164
1165
1166
1166
1167
1168
1168
1169
1170
1170
1171
1172
1172
1173
1174
1174
1175
1176
1176
1177
1178
1178
1179
1180
1180
1181
1182
1182
1183
1184
1184
1185
1186
1186
1187
1188
1188
1189
1190
1190
1191
1192
1192
1193
1194
1194
1195
1196
1196
1197
1198
1198
1199
1200
1200
1201
1202
1202
1203
1204
1204
1205
1206
1206
1207
1208
1208
1209
1210
1210
1211
1212
1212
1213
1214
1214
1215
1216
1216
1217
1218
1218
1219
1220
1220
1221
1222
1222
1223
1224
1224
1225
1226
1226
1227
1228
1228
1229
1230
1230
1231
1232
1232
1233
1234
1234
1235
1236
1236
1237
1238
1238
1239
1240
1240
1241
1242
1242
1243
1244
1244
1245
1246
1246
1247
1248
1248
1249
1250
1250
1251
1252
1252
1253
1254
1254
1255
1256
1256
1257
1258
1258
1259
1260
1260
1261
1262
1262
1263
1264
1264
1265
1266
1266
1267
1268
1268
1269
1270
1270
1271
1272
1272
1273
1274
1274
1275
1276
1276
1277
1278
1278
1279
1280
1280
1281
1282
1282
1283
1284
1284
1285
1286
1286
1287
1288
1288
1289
1290
1290
1291
1292
1292
1293
1294
1294
1295
1296
1296
1297
1298
1298
1299
1300
1300
1301
1302
1302
1303
1304
1304
1305
1306
1306
1307
1308
1308
1309
1310
1310
1311
1312
1312
1313
1314
1314
1315
1316
1316
1317
1318
1318
1319
1320
1320
1321
1322
1322
1323
1324
1324
1325
1326
1326
1327
1328
1328
1329
1330
1330
1331
1332
1332
1333
1334
1334
1335
1336
1336
1337
1338
1338
1339
1340
1340
1341
1342
1342
1343
1344
1344
1345
1346
1346
1347
1348
1348
1349
1350
1350
1351
1352
1352
1353
1354
1354
1355
1356
1356
1357
1358
1358
1359
1360
1360
1361
1362
1362
1363
1364
1364
1365
1366
1366
1367
1368
1368
1369
1370
1370
1371
1372
1372
1373
1374
1374
1375
1376
1376
1377
1378
1378
1379
1380
1380
1381
1382
1382
1383
1384
1384
1385
1386
1386
1387
1388
1388
1389
1390
1390
1391
1392
1392
1393
1394
1394
1395
1396
1396
1397
1398
1398
1399
1400
1400
1401
1402
1402
1403
1404
1404
1405
1406
1406
1407
1408
1408
1409
1410
1410
1411
1412
1412
1413
1414
1414
1415
1416
1416
1417
1418
1418
1419
1420
1420
1421
1422
1422
1423
1424
1424
1425
1426
1426
1427
1428
1428
1429
1430
1430
1431
1432
1432
1433
1434
1434
1435
1436
1436
1437
1438
1438
1439
1440
1440
1441
1442
1442
1443
1444
1444
1445
1446
1446
1447
1448
1448
1449
1450
1450
1451
1452
1452
1453
1454
1454
1455
1456
1456
1457
1458
1458
1459
1460
1460
1461
1462
1462
1463
1464
1464
1465
1466
1466
1467
1468
1468
1469
1470
1470
1471
1472
1472
1473
1474
1474
1475
1476
1476
1477
1478
1478
1479
1480
1480
1481
1482
1482
1483
1484
1484
1485
1486
1486
1487
1488
1488
1489
1490
1490
1491
1492
1492
1493
1494
1494
1495
1496
1496
1497
1498
1498
1499
1500
1500
1501
1502
1502
1503
1504
1504
1505
1506
1506
1507
1508
1508
1509
1510
1510
1511
1512
1512
1513
1514
1514
1515
1516
1516
1517
1518
1518
1519
1520
1520
1521
1522
1522
1523
1524
1524
1525
1526
1526
1527
1528
1528
1529
1530
1530
1531
1532
1532
1533
1534
1534
1535
1536
1536
1537
1538
1538
1539
1540
1540
1541
1542
1542
1543
1544
1544
1545
1546
1546
1547
1548
1548
1549
1550
1550
1551
1552
1552
1553
1554
1554
1555
1556
1556
1557
1558
1558
1559
1560
1560
1561
1562
1562
1563
1564
1564
1565
1566
1566
1567
1568
1568
1569
1570
1570
1571
1572
1572
1573
1574
1574
1575
1576
1576
1577
1578
1578
1579
1580
1580
1581
1582
1582
1583
1584
1584
1585
1586
1586
1587
1588
1588
1589
1590
1590
1591
1592
1592
1593
1594
1594
1595
1596
1596
1597
1598
1598
1599
1600
1600
1601
1602
1602
1603
1604
1604
1605
1606
1606
1607
1608
1608
1609
1610
1610
1611
1612
1612
1613
1614
1614
1615
1616
1616
1617
1618
1618
1619
1620
1620
1621
1622
1622
1623
1624
1624
1625
1626
1626
1627
1628
1628
1629
1630
1630
1631
1632
1632
1633
1634
1634
1635
1636
1636
1637
1638
1638
1639
1640
1640
1641
1642
1642
1643
1644
1644
1645
1646
1646
1647
1648
1648
1649
1650
1650
1651
1652
1652
1653
1654
1654
1655
1656
1656
1657
1658
1658
1659
1660
1660
1661
1662
1662
1663
1664
1664
1665
1666
1666
1667
1668
1668
1669
1670
1670
1671
1672
1672
1673
1674
1674
1675
1676
1676
1677
1678
1678
1679
1680
1680
1681
1682
1682
1683
1684
1684
1685
1686
1686
1687
1688
1688
1689
1690
1690
1691
1692
1692
1693
1694
1694
1695
1696
1696
1697
1698
1698
1699
1700
1700
1701
1702
1702
1703
1704
1704
1705
1706
1706
1707
1708
1708
1709
1710
1710
1711
1712
1712
1713
1714
1714
1715
1716
1716
1717
1718
1718
1719
1720
1720
1721
1722
1722
1723
1724
1724
1725
1726
1726
1727
1728
1728
1729
1730
1730
1731
1732
1732
1733
1734
1734
1735
1736
1736
1737
1738
1738
1739
1740
1740
1741
1742
1742
1743
1744
1744
1745
1746
1746
1747
1748
1748
1749
1750
1750
1751
1752
1752
1753
1754
1754
1755
1756
1756
1757
1758
1758
1759
1760
1760
1761
1762
1762
1763
1764
1764
1765
1766
1766
1767
1768
1768
1769
1770
1770
1771
1772
1772
1773
1774
1774
1775
1776
1776
1777
1778
1778
1779
1780
1780
1781
1782
1782
1783
1784
1784
1785
1786
1786
1787
1788
1788
1789
1790
1790
1791
1792
1792
1793
1794
1794
1795
1796
1796
1797
1798
1798
1799
1800
1800
1801
1802
1802
1803
1804
1804
1805
1806
1806
1807
1808
1808
1809
1810
1810
1811
1812
1812
1813
1814
1814
1815
1816
1816
1817
1818
1818
1819
1820
1820
1821
1822
1822
1823
1824
1824
1825
1826
1826
1827
1828
1828
1829
1830
1830
1831
1832
1832
1833
1834
1834
1835
1836
1836
1837
1838
18
```

Element	Bemerkung	Beispiel
c	Liefert den Tausender trenner. Im deutschen Sprachraum wird das Komma verwendet. , (Komma)	999999999
,	Wird als Tausender trenner nach amerikanischen Standard verwendet und kann mehrfach vorhanden sein, allerdings weder als erstes Zeichen noch rechts vom Dezimaltrenner bzw. Punkt.	9,999,999
d	Liefert das Dezimaltrennzeichen. Im deutschen Sprachraum ist das ein . (Punkt)	90999000
,	Wird als Dezimaltrenner nach amerikanischen Standard verwendet und darf daher nur einmal vorkommen.	9,999,00
c	Liefert das ISO Währungssymbol an der angegebenen Stelle	€99990000
\$	Liefert die Zahl mit einem führenden Dollarzeichen	\$9,990,00
9	Optional: Ziffer, ist die Ziffer dieser Position nicht vorhanden, wird sie nicht ausgegeben.	9,99
0	Verpflichtend auszugebende Ziffer, ist die Ziffer dieser Position nicht vorhanden, wird eine 0 ausgegeben.	0,00
FM	Entfernt Leerzeichen aus der Zeichenkette.	F999999
FM	Entfernt Leerzeichen aus der Zeichenkette.	F999999
FM	Entfernt Leerzeichen aus der Zeichenkette.	F999999
B	Liefert Leerzeichen falls die Ziffern des Ganzzahlanteils nicht vorhanden sind.	8999999

Abbildung 22. Formattoption für numerische Konvertierungsfunktionen

#### 4.4. Numerische Funktionen

Numerische Funktionen werden zur Verarbeitung numerischer Werte in SQL verwendet.

##### 4.4.1 Rundungsfunktionen

Zum Runden numerischer Werte stellt die SQL Spezifikation 2 Funktionen zur Verfügung: trunc und round.

###### ► Analyse: Rundungsfunktionen

- Mit der trunc Funktion werden numerische Werte abgerundet, während round nach kaufmännischem Verfahren auf bzw. abrundet.
- Der 2te Parameter spezifiziert die Genauigkeit des Rundungsvorgangs. Dazu wird die Anzahl der Nachkommastellen angegeben.
- Zusätzlich kann für den 2ten Parameter ein negativer Wert übergeben werden, wodurch die Funktion auf ganze Zehner, Hunderter bzw. Tausender etc. runden.

###### ► Syntax: round Funktion

```

1 -- -----
2 -- Syntax: round
3 --
4 FUNCTION round (
5   p_number IN NUMBER,
6   p_round  IN PLS_INTEGER
7 )
8 RETURN VARCHAR2;
9
10 SELECT round(12345.678) n_1,
11      round(12345.678, 1) n_2,
12      round(12345.678, 2) n_3,
13      round(12345.678, -2) n_6,
14 FROM dual;
15
16 -- Ausgabe
17 12346    -- n_1
18 12345.7   -- n_2
19 12345.68  -- n_3
20 12300    -- n_6

```

53

###### Informationssysteme

###### ► Syntax: trunc Funktion

```

1 -- -----
2 -- Syntax: trunc
3 --
4 FUNCTION trunc (
5   p_number IN NUMBER,
6   p_round  IN PLS_INTEGER
7 )
8 RETURN VARCHAR2;
9
10 -- -----
11 -- Syntax: trunc
12 --
13 SELECT trunc(12345.678) n_1,
14      trunc(12345.678, 1) n_2,
15      trunc(12345.678, 2) n_3,
16      trunc(12345.678, -2) n_6,
17      trunc(12345.678, -1) n_7
18 FROM dual;
19
20
21 -- Ausgabe
22 12345    -- n_1
23 12345.6   -- n_2
24 12345.67  -- n_3
25 12300    -- n_6
26 12340    -- n_7

```

###### ► Query: to\_number Funktion

```

1 -- -----
2 -- Syntax: to_number
3 --
4 FUNCTION to_number (
5   p_number_literal IN VARCHAR2,
6   p_mask            IN VARCHAR2
7 )
8 RETURN NUMBER;
9
10 -- -----
11 -- Konvertierungsfunktion: to_number
12 --
13 SELECT
14   to_number('12345') n_1,
15   to_number('123,45') n_2,
16   to_number('123.45,21', '999G999D99') n_3,
17   to_number('123.45,21', '999G999D99') n_4
18 FROM dual;
19
20
21 -- Ausgabe
22 12345    -- n_1
23 123,45   -- n_2
24 12345,45 -- n_3
25 12345,45 -- n_4

```

□

##### 4.4.3 Numerische Funktion: abs()

Die abs Funktion gibt den absoluten Wert einer Zahl zurück.

###### ► Query: abs Funktion

```

1 -- -----
2 -- Syntax: abs
3 --
4 FUNCTION abs (
5   p_value IN Number
6 )
7 RETURN NUMBER;
8
9
10 -- -----
11 -- Example: abs
12 --
13 SELECT
14   abs(-4) n_1, abs( 4) n_2
15   FROM dual;
16
17 -- Ausgabe
18 4     -- n_1
19 4     -- n_2
20 12,345,67EUR -- n_3

```

□

##### 4.4.2 Konvertierungsfunktionen

Zum Konvertieren numerischer Wert in Zeichenketten wird die to\_char Funktion verwendet. Dazu werden durch den User Formatmasken definiert.

###### ► Query: to\_char Funktion

```

1 -- -----
2 -- Konvertierungsfunktion: to_char
3 --
4 SELECT
5   to_char(12345.89, '999G999D99') a_1,
6   to_char(12345.67, '999G999D99') a_2,
7   to_char(12345.67, '999G999D99') a_3
8   FROM dual;
9
10 -- Ausgabe
11 123,456,12   -- a_1
12 12,345,67    -- a_2
13 12,345,67EUR -- a_3

```

Funktion	Beschreibung	Seite
<code>round0, trunc0</code>	Funktionen zum <b>Runden</b> numerischer Werte	53
<code>to_char0</code>	Funktion zum Konvertieren numerischer Werte in Zeichenketten.	54
<code>to_number0</code>	Funktion zum Konvertieren von Zeichenketten in numerische Werte.	54
<code>abs0</code>	Die <code>abs</code> Funktion gibt den absoluten Wert einer Zahl zurück	54
<code>mod0</code>	Die <code>mod</code> Funktion gibt den Rest von m geteilt durch n als Ergebnis zurück.	55
<code>exp0</code>	Die <code>exp</code> Funktion berechnet die mathematische Exponentenfunktion für einen numerischen Wert.	55
<code>log0</code>	Die <code>log</code> Funktion gibt den Logarithmus von n zur Basis m zurück.	55

Abbildung 23. Numerische Funktionen

#### **4.4.4 Numerische Funktion: exp(), log()**

Die `exp` Funktion berechnet die mathematische Exponentialfunktion für einen numerischen Wert. Die `log` Funktion gibt den Logarithmus von  $n$  zur Basis  $m$  zurück.

```
1 -- Syntax: exp, log
2
3
4 FUNCTION exp (
5   p_m IN NUMBER
6 )
7 RETURN NUMBER;
8
9 FUNCTION log (
10   p_n IN NUMBER,
11   p_m IN NUMBER
12 )
13 RETURN NUMBER;
14
15
16 -- Example: exp, log
17
18 SELECT
19   exp(3, n_1,
20   log(10, n_2,
21   log(100, 1)_n_3
22 FROM dual;
23
24 -- Ergebnis
25 0,0655369 -- n_1
26 1,30203999 -- n_2
```

#### 4.4.5 Numerische Funktion: mod()

Die mod Funktion gibt den Rest von m geteilt durch n als Ergebnis zurück.

```

1 -- Query: mod Funktion
2 -- Syntax: mod
3
4 FUNCTION mod (
5   p_n IN Number,
6   p_n IN Number
7 )
8   RETURN NUMBER;
9
10 --
11 -- Example: mod
12
13 SELECT
14   mod(15, 4) n_1,
15   mod(15, 3) n_2,
16   mod(15, 0) n_3,
17   mod(11, 2) n_4,
18   mod(-15, 4) n_5,
19   mod(-15, 0) n_6,
20   FROM dual;
21
22 -- Ergebnis
23 0 -- n_1
24 0 -- n_2
25 15 -- n_3
26 1,1 -- n_4
27 -3 -- n_5
28 -15 -- n_6

```

10

5. SQL - Aggregatfunktionen

04

## 5.1. Aggregatfunktionen

**Funktionen**, die eine Menge von Werten zu einem einzelnen Wert verdichten, werden als **Aggregationsfunktionen** bezeichnet.

- Aggregation ▾  
Aggregation bezeichnet das Zusammenfassen einer Reihe von Werten zu einem einzelnen Wert.  
Beispielsweise lässt sich aus einer Menge von Zahlen der Mittelwert, das Minimum bzw. das Maximum oder die Summe der Werte bestimmen.

### 5.1.1 Aggregatfunktionen

Im Gegensatz zu einer Zeilenfunktion, die für einen Datensatz einen einzelnen Wert berechnet, verfügt eine Aggregatfunktion mehrere Datensätze zu einem einzelnen Wert.

- Erklärung: **Einfache Aggregatfunktionen** ▶
  - Die SQL Spezifikation definiert eine Reihe von Aggregatfunktionen: avg, sum, min, max und count.
  - Aggregatfunktionen werden in der Regel in Kombi-

```
nation mit der group by Klausur verwendet.  
1  
2 -- Aggregatfunktionen  
3  
4 -- Berechnet die Anzahl aller Angestellten  
5 SELECT count(employee_id) FROM employees;  
6  
7 -- Berechnet das kleinste Gehalt das im  
8 -- Unternehmen ausbezahlt wird  
9 SELECT min(SALARY) FROM employees;  
10  
11 -- Berechnet das hoechste Gehalt das im  
12 -- Unternehmen ausbezahlt wird  
13 SELECT max(SALARY) FROM employees;  
14  
15 -- Berechnet das Durchschnittsgehalt  
16 -- des im Unternehmen ausbezahlt wird  
17 SELECT avg(SALARY) FROM employees;
```

8

**5.1.2 Median****Median ▾**

- Bei der **Median** oder **zentralwert** ist einer der statistischen Mittelwerte.

Zur Berechnung des Medians einer Menge von Zahlen, werden die Werte zunächst sortiert und dann der mittlere dieser Werte gewählt.

**> Query: Median berechnen ▾**

```
1 -- -----
2 -- Median
3 --
4 SELECT
5   median(coalesce(salary, 0)) MEDIAN
6 FROM employees;
```

**5.1.3 Standardabweichung**

- Standardabweichung ▾**
- Die **Standardabweichung** einer Menge gibt an wie stark die Werte der Menge von deren Mittelwert abweichen.

Die Standardabweichung ist ein Streuungswert der Statistik.

**> Erklärung: Standardabweichung ▾**

- Bei einer Menge von Zahlen, die um den Mittelwert herum angeordnet sind, ist es oft von Interesse zu wissen, wie nahe diese Werte am Mittelwert liegen.
- Als Mittelwert wird in diesem Zusammenhang das arithmetische Mittel verwendet.
- Bleiben die Werte dicht am Mittelwert, ist die Streuung der Datenwerte gering.
- Der Abstand der Werte vom Mittelwert, wird als Quadrat der Differenz der Einzelpunkte berechnet.

```
1 -- -----
2 -- Standardverteilung
3 --
4 SELECT stdev(salary)
5 FROM employees;
```

**5.1.4 Aggregatfunktionen und NULL Werte**

Bei der Verdichtung von Datensätzen zu einem einzelnen Wert werden null Werte ignoriert.

Damit können **Aggregatfunktionen** auch auf Spalten angewandt werden die null Werte enthalten.

**> Query: Aggregatfunktionen und NULL Werte ▾**

```
1 -- -----
2 -- Aggregatfunktionen und NULL Werte
3 --
4 SELECT avg(salary), max(salary)
5 FROM employees;
```

**5.1.5 UNIQUE Operator**

Für Aggregatfunktionen muss entschieden werden ob alle oder nur alle unterschiedlichen Werte einer Spalte für die Berechnung eines Ergebnisses herangezogen werden sollen.

- > Erklärung: Werte einer Spalte zählen ▾**
- Durch die Verwendung des **distinct** Schlüsselworts in der **select** Klausel, wird die Datenbankengine angehalten, alle redundanten Datensätze aus dem Ergebnis einer Abfrage zu entfernen.

- Da die **group** Klausel jedoch vor der **select** Klausel ausgewertet wird, hat die Verwendung des **distinct** Schlüsselworts keinen Auswirkung auf den Datenbestand der **group** Klausel.

```
1 -- -----
2 -- distinct operator
3 --
4 SELECT count(distinct job) berufe
5 FROM emp;
```

57

**5.2. Group by Klausel**

Die **group by** Klausel führt eine **Neustrukturierung** der Daten einer Abfrage durch.

Bislang ist es zwar ganz interessant, das höchste Gehalt bzw. das Durchschnittsgehalt aller Angestellten im Unternehmen zu berechnen, vielmehr würde uns aber der Vergleich der Gehälter der einzelnen Abteilungen im Unternehmen interessieren.

**5.2.1 Neustrukturierung von Daten**

- Group by Klausel ▾**
- Die **group by** Klausel führt eine **Neustrukturierung** der Daten einer Abfrage durch.

**> Erklärung: Ausführungsreihenfolge der Klauseln ▾**

select Anweisung  
from where group by select order by  
having clause offset fetch

- > Beispiel: Group By Klausel ▾**
- Wir möchten für jede Abteilung die Anzahl der Mitarbeiter, die dort arbeiten, bestimmen.
- Mit der gegenwärtigen Struktur des Datenbestands ist eine Lösung der Aufgabe nicht möglich.
- Zur Lösung der Aufgabe wäre es notwendig die Angestellten jeder Abteilung in einer eigenen Tabelle zu speichern. Die Anzahl der Mitarbeiter pro Abteilung könnte dann durch das Aufsummieren der Datensätze der Tabellen, ermittelt werden.
- Die **group by** Klausel ermöglicht eine entsprechende Neustrukturierung der Daten.

**5.2.2 Group by Klausel**

Mit der **group by** Klausel kann eine Neustrukturierung der Daten einer Abfrage durchgeführt werden.

**> Erklärung: Neustrukturierung von Daten ▾**

- Die durch die **group by** Klausel angestochene Neustrukturierung der Daten einer Abfrage erfolgt in 2 Schritten: der **Map Phase** und der **Aggregate Phase**.

- Map Phase:** In der Map Phase werden die Daten der Datenbasis auf disjunkte Gruppen verteilt.
- Aggregate Phase:** In der Aggregate Phase werden die Daten jeder Gruppe zu einem einzelnen Datensatz verdichtet. Die Menge aller so bestimmten Datensätze wird zur Datenbasis der Abfrage. Die Struktur des Datenbestandes wurde damit einer Neustrukturierung unterzogen.
- Nach dem Abschluss der Aggregate Phase ist die Neustrukturierung der Daten abgeschlossen.

**5.2.3 Map Phase**

In der Map Phase werden die Daten einer SQL Abfrage auf disjunkte virtuelle Tabellen verteilt.

**> Erklärung: Map Phase ▾**

- Wir möchten für jede Abteilung eines Unternehmens, die Anzahl der dort beschäftigten Mitarbeiter, bestimmen.
- Für jede Abteilung legt die Datenbankengine eine eigene virtuelle Tabelle an. Die Angestelltendaten werden nun in die virtuellen Tabellen der zugehörigen Abteilungen übernommen.
- Nach der Map Phase liegen die Angestelltendaten gruppiert auf ihre Abteilungen vor.
- Mit der **group by** Klausel wird bestimmt nach welchen Kriterien der Datenbestand nun gruppiert werden soll.

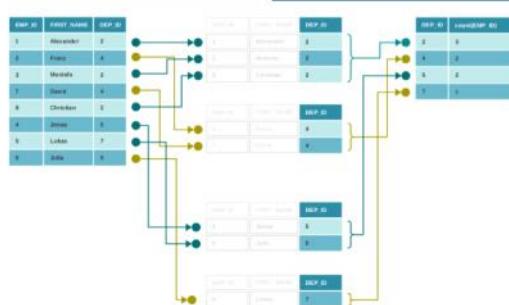


Abbildung 24. Group by: Neustrukturierung von Daten

## 5.2.4 Group by Klausel

Mit der group by Klausel wird bestimmt nach welchen Kriterien der Datenbestand gruppiert werden soll.

## Erklärung: Group by Klausel ▾

- Die disjunktiven Werte, der in der group by Klausel angegebenen Spalten, definieren die Gruppen auf die Datenbestand verteilt werden soll.
  - Wird in einer Abfrage beispielsweise nach der department\_id Spalte gruppiert, erfolgt eine Verteilung der Datensätze auf die unterschiedlichen Abteilungen des Unternehmens.
- ```
-- Group by Klausel
SELECT department_id, count(employee_id),
       avg(salary)
FROM employees
GROUP BY department_id;
```

## Query: Group by Klausel ▾

- Die Spaltenbezeichner der group by Klausel definierten dabei die Kriterien für die Neustrukturierung der Daten.

59

## 5.2.5 Semantik der Gruppenbildung

Wird der Datenbestand einer Abfrage nach mehreren Kriterien gruppiert, die im Kontext der Anwendung denselben Sachverhalt beschreiben, wird für die Gruppierung nur eines der Kriterien herangezogen.

## Query: Semantik der Gruppenbildung ▾

```
-- Semantik der Gruppenbildung
-- Die Spalten department_id und department_name
-- sowie month_name beschreiben im Kontext der
-- Anwendung denselben Sachverhalt.

SELECT department_id, department_name
FROM employees
NATURAL JOIN departments
GROUP BY department_id, department_name;
```

## 5.2.6 Fallbeispiel: Gruppierung

SQL Abfragen mit einer group by Klausel.

## Query: Group by Klausel ▾

```
-- Anzahl der Angestellten pro Abteilung
SELECT department_id, count(employee_id)
      , avg(salary)
FROM employees
GROUP BY department_id;

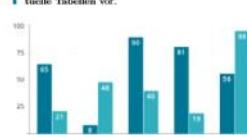
-- Anzahl der Angestellten fuer jedes Land
SELECT country_id, count(employee_id)
      , avg(salary)
FROM employees
GROUP BY country_id;

-- Anzahl der Angestellten pro Abteilung
-- fuer jedes Land
SELECT country_id, count(employee_id)
      , department_id
FROM employees
GROUP BY country_id, department_id;
```

## Erläuterung: Aggregate Phase ▾

In der Aggregate Phase werden die Datensätze der virtuellen Gruppen auf einen einzelnen Datensatz verdichtet.

Nach der Map Phase des Gruppierungsvorgangs liegen die Datensätze der Abfrage verteilt auf virtuelle Tabellen vor.



59

## Erläuterung: Aggregate Phase ▾

- In der Aggregate Phase werden die Datensätze der einzelnen virtuellen Tabelle auf einen einzelnen Datensatz verdichtet.
- Die Verdichtung der Datensätze erfordert dabei eine Neustrukturierung der Daten. Für die Neustrukturierung der Daten gelten bestimmte Vorgaben.
- Umstrukturierte Datensätze dürfen nur Werte enthalten, nach denen ursprünglich gruppiert werden ist.
- Zusätzlich kann einem Zieldatensatz das Ergebnis beliebiger Aggregationsfunktionen zugewendet werden.
- Klausur einer select Abfrage, die von der Datenbankengine nach der group by Klausel ausgewertet werden, haben nur mehr Zugriff auf die neustrukturierten Daten.

## 5.2.8 Having Klausel

Mit der having Klausel können für die in der group by Klausel definierten Gruppen, Filterkriterien definiert werden.

59

## » Analyse: Filter: having Klausel »

- Mit der where Klausel kann der Datenbestand einer Abfrage nach bestimmten Kriterien gefiltert werden.

Zeitlich erfolgt die Auswertung der where Klausel in SQL Abfragen vor der Auswertung der group by Klausel. Es fehlt damit die Möglichkeit die virtuellen Gruppen einer group by Klausel zu filtern.

- Mit der having Klausel stellt die SQL Spezifikation eine Möglichkeit zur Verfügung, die virtuellen Gruppen der group by Klausel zu filtern.

- Wir haben damit eine Möglichkeit an der Hand eine Selektion auf Gruppenebene durchzuführen.

```

1 -- -----
2 -- having Klausel
3 --
4 SELECT department_id, min(salary) min_sal
5 FROM employees
6 JOIN departments USING (department_id)
7 GROUP BY department_id, job_id
8 HAVING sum(salary) < 10000;

```

□

## 5.2.9 Null Aggregat

## Null Aggregat ▾

Als Null Aggregat werden virtuelle Gruppen bezeichnet, die aus Datensätzen bestehen die keiner konkreten Gruppe zugeordnet werden können.

## » Erklärung: Null Aggregat »

- Alle Datensätze einer SQL Abfrage, die für eines der Gruppierungskriterien keinen Wert besitzen, werden in einer eigenen virtuellen Gruppe zusammengefasst.

- Das **NULLS FIRST** bzw. **NULLS LAST** Schlüsselwort bestimmt ob das Null Aggregat dem Ergebnis vorgestellt oder als letzte virtuelle Gruppe im Ergebnis figuriert.



61

## 6. SQL - Komplexe Abfragen

## 05

## Komplexe Abfragen

|                          |    |
|--------------------------|----|
| 01. Unterabfragen        | 62 |
| 02. With Klausel         | 65 |
| 03. Punktweise Vergleich | 77 |
| 04. Mengenoperationen    | 66 |
| 05. Quantoren            | 70 |

## 6.1. Unterabfragen

Unterabfragen sind **select Ausdrücke**, die in SQL Abfragen eingebettet werden.

## 6.1.1 Unterabfragen

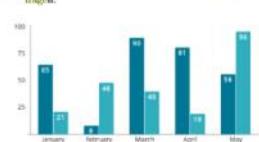
Oft gibt es Fragestellungen, die auf einfache Weise in SQL nicht zu beantworten sind: z.B.: Wie heißen die Mitarbeiter mit dem geringsten Einkommen im Unternehmen?

## » Analyse: Unterabfragen »

- Um diese Fragestellung beantworten zu können müssen wir erst einmal wissen wie hoch das geringste Einkommen im Unternehmen ist, um anschließend zu ermitteln, welche Mitarbeiter ein entsprechendes Einkommen besitzen.

- Das Problem zur Beantwortung der vorhergehenden Frage müssten 2 Abfragen zur selben Zeit abgesetzt werden.

- Wir brauchen damit einen neuen Ansatz: **Unterabfragen**.



## » Erklärung: Unterabfragen »

- Unterabfragen sind **select Ausdrücke** die in SQL Abfragen eingebettet werden. Die SQL Spezifikation erlaubt dabei Unterabfragen in der select, from, where bzw. having Klausel.

- Beim **Auswerten einer SQL Abfrage**, berechnet die Datenbankengine zuerst das Ergebnis der Unterabfragen. Das Ergebnis der Unterabfrage steht der Abfrage dann in Form einer virtuellen Tabelle zur Verfügung.

□

62



## Unterabfragen

### 6.1.2 Formen von Unterabfragen

Die SQL-Spezifikation definiert mehrere Formen von Unterabfragen.

#### • Auflistung: Formen von Unterabfragen ▾

##### Innere View ▾

Unterabfragen in der `from`-Klausel werden als Innere Views bezeichnet.

In der `from`-Klausel einer Abfrage wird die **Datenbasis** einer Query definiert. Das Ergebnis der inneren View bildet in diesem Fall die Datenbasis der Abfrage.

##### Skalare Abfragen ▾

Subqueries die nach der Auswertung durch die SQL Engine als Ergebnis lediglich einen **einzelnen Datensatz** enthalten, werden als Skalare Abfragen bezeichnet.

In der `select`-Klausel dürfen nur Skalare Abfragen definiert werden.

##### Konditionale Abfragen ▾

Unterabfragen in der `where`-Klausel bzw. `having`-Klausel werden als Konditionale Abfragen bezeichnet.

Das Ergebnis einer Konditionale Abfrage wird für die Auswertung der `having` bzw. `where`-Klausel herangezogen.

### 6.1.3 Subqueryform: Innere View

Unterabfragen in der `from`-Klausel werden als Innere Views bezeichnet.

#### • Erklärung: Innere View ▾

- In der `from`-Klausel einer Abfrage wird die Datenbasis einer Query definiert.

- Das Ergebnis der inneren View wird dabei in einer **virtuellen Tabelle** gespeichert.

#### • Query: Innere View Beispiele ▾

```

1 -- -----
2 -- Subqueryform: Innere View
3 --
4
5 SELECT e.first_name, e.last_name, e.salary
6   FROM (SELECT max(salary) max_sal,
7              department_id
8            FROM employees
9           GROUP BY department_id) sub
10          JOIN employees e
11            ON e.department_id = sub.department_id
12           AND e.salary = sub.max_sal;
13
14
15 SELECT e.first_name, e.last_name, e.salary
16   FROM employees e JOIN
17    (SELECT min(salary) a_sal
18      FROM employees) sub_query
19     ON e.salary = sub_query.a_sal
20   ORDER BY e.department_id;
```

63

## Informationssysteme

| Konzept                | Beschreibung                                                                                                                                                                                                                                                                                                       | Seite |
|------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------|
| Innere View            | Unterabfragen in der <code>from</code> -Klausel werden als Innere Views bezeichnet. Das Ergebnis einer inneren View bildet in diesem Fall die Datenbasis einer Abfrage.                                                                                                                                            | 63    |
| Skalare Abfrage        | Subqueries die nach ihrer Auswertung durch die Datenbankengine als Ergebnis lediglich einen einzelnen Datensatz enthalten, werden als Skalare Abfragen bezeichnet. Skalare Abfragen können in der <code>select</code> , <code>from</code> , <code>where</code> bzw. <code>having</code> -Klausel definiert werden. | 64    |
| Konditionale Abfrage   | Unterabfragen in der <code>where</code> -Klausel bzw. <code>having</code> -Klausel werden als Konditionale Abfragen bezeichnet. Das Ergebnis einer Konditionale Abfrage wird für die Auswertung der <code>having</code> bzw. <code>where</code> -Klausel herangezogen.                                             | 64    |
| With Klausel           | Mit der <code>with</code> -Klausel kann SQL Code einheitlich und leichter strukturiert werden. Dazu wird in der <code>with</code> -Klausel eine oder mehrere Queries definiert, die anschließend in der eigentlichen Abfrage referenziert werden können.                                                           | 66    |
| Union                  | Die <code>union</code> -Klausel wird verwendet, um die Ergebnismengen zweier oder mehrerer <code>select</code> -Anweisungen miteinander zu kombinieren.                                                                                                                                                            | 66    |
| Intersect              | Die <code>intersect</code> -Klausel wird verwendet, um die Durchschnittsmenge zweier oder mehrerer <code>select</code> -Anweisungen zu bilden.                                                                                                                                                                     | 67    |
| Except                 | Die <code>except</code> -Klausel wird verwendet, um die Differenzmenge zweier oder mehrerer <code>select</code> -Anweisungen zu bilden.                                                                                                                                                                            | 67    |
| Quantoren              | Zur Prüfung einer Menge von Datensätzen auf bestimmte Eigenschaften können die logischen Operatoren <code>ALL</code> , <code>ANY</code> bzw. <code>EXISTS</code> verwendet werden.                                                                                                                                 | 70    |
| Hierarchische Abfragen | Hierarchische Abfragen ermöglichen Auswertungen zu hierarchischen Strukturen im Datenbestand.                                                                                                                                                                                                                      | 68    |

Abbildung 25. Konzepte komplexer Abfragen

### 6.1.4 Subqueryform: Skalare Abfragen

Subqueries die nach der Auswertung durch die SQL-Datenbankengine im Ergebnis lediglich einen einzelnen Datensatz enthalten, werden als Skalare Abfragen bezeichnet.

Unterabfragen in der `select`-Klausel müssen Skalare Abfragen sein.

#### • Query: Skalare Abfrage ▾

```

1 -- -----
2 -- Subqueryform: Innere View
3 --
4
5 SELECT e.first_name, e.last_name,
6       (SELECT sum(salary) FROM employees)
7       * company_income
8
9  FROM employees e JOIN departments d
10    ON e.department_id = d.department_id
11   GROUP BY e.department_id,
12          d.department_name
13   ORDER BY d.department_name, e.last_name;
```

64

### 6.1.5 Subqueryform: Konditionale Abfrage

Unterabfragen in der `where` bzw. `having`-Klausel werden als Konditionale Unterabfragen bezeichnet.

#### • Erklärung: Konditionale Abfrage ▾

- Das Ergebnis einer Konditionale Abfrage wird für die Auswertung der `having` bzw. `where`-Klausel herangezogen.

- Verwenden Sie für Vergleiche mit Konditionalen Abfragen den `in`-Operator.

#### • Query: Konditionale Unterabfragen ▾

```

1 -- -----
2 -- Subqueryform: Unterabfragen
3 --
4
5 SELECT e.first_name, e.last_name
6   FROM employees e
7 WHERE e.salary < (SELECT min(salary) FROM employees);
```

**Outer Query**

```

    { SELECT lastName, job_id, salary
      FROM employees
      WHERE (department_id, salary) IN (SELECT department_id, min(salary)
   FROM employees
   GROUP BY department_id)
  }
```

**Subquery****6.2. With Klausel**

Mit Hilfe der With Klausel können SQL Abfragen auf einfache Weise logisch strukturiert werden.

**6.2.1 WITH Klausel - Strukturierung**

Komplexe SQL Abfragen sind oft schwer lesbar bzw. unverständlich.

**Erklärung: Codestrukturierung**

- SQL Code wird im Gegensatz zur funktionalen Programmierung nicht über Unterprogramme, sondern mit der Hilfe von Unterabfragen strukturiert.
- Die Strukturierung von SQL Code mit Unterabfragen führt jedoch zu einer Verschachtelung des Abfragecodes.
- Abfragen dieser Form müssen von innen nach außen ausgewertet.

**WITH Klausel - Strukturierung**

Die with Klausel ermöglicht es SQL Code auf einfache Weise logisch zu strukturieren. Dazu wird in der with Klausel eine Liste von Queries definiert, die aus der eigentlichen Abfrage heraus referenziert werden können.

Die in der with Klausel definierten Abfragen werden als Innere Views referenziert.

**6.2.2 Syntax: WITH Klausel**

In SQL Abfragen wird die with Klausel syntaktisch vor der select Klausel definiert.

**Syntax: WITH Klausel**

```

1 -- -----
2 -- Syntax: with Klausel
3 -- -----
4 WITH query_name1 AS (
5   SELECT ...
6   ), query_name2 AS (
7     SELECT ...
8   ), query_name3 AS (
9     SELECT ...
10    )
11   SELECT ...
12   FROM query_name1 q1
13   JOIN query_name2 q2 ON ...
14   ...
15
16 -- -----
17 -- Query: with Klausel
18
19 WITH rep_employees AS (
20   SELECT max(salary) max_sal,
21   e.department_id
22   FROM employees e
23   GROUP BY e.department_id
24 )
25   SELECT e.department_id
26   e.first_name
27   FROM employees e
28   JOIN rep_employees r ON
29   e.salary = r.max_sal AND
30   e.department_id = r.department_id
31   JOIN departments d ON
32   d.department_id = e.department_id
33   ORDER BY d.department_name;
```

65

## Informationssysteme

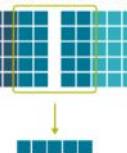
**Resultset Query 1****Resultset Query 2****Resultset Intersect Resultset 2.**

Abbildung 26. Mengentheoretische Operation: Intersect

**6.3. Mengentheoretische Operationen****Mengenlehre**

Die Mengenlehre ist ein grundlegendes Teilgebiet der Mathematik, das sich mit der Untersuchung von Mengen beschäftigt.

Die Mengenlehre ist dabei ein essentieller Bestandteil der SQL Sprachspezifikation.

**6.3.1 Mengentheoretische Operationen**

SQL als Programmiersprache definiert eine mengenorientierte Sprachsprache zur Datendefinition bzw. Datenmanipulation.

**Erklärung: Mengentheoretische Operationen**

- Mengentheoretische Operationen sind ein fundamentaler Bestandteil der SQL Sprachspezifikation.
- Die Sprache selbst unterstützt dabei alle grundlegenden mengentheoretischen Operationen.
- Die Form einer mengentheoretischen Operation folgt dabei immer derselben Syntax.

**6.3.2 Union Klausel - Vereinigungsmenge**

Die union Klausel wird verwendet, um die Ergebnismengen zweier oder mehrerer select Anweisungen miteinander zu kombinieren.

**Erklärung: union, union all Klausel**

- Um die Ergebnisse zweier Queries miteinander kombinieren zu können, muss die Struktur der Ergebnismengen beider Abfragen übereinstimmen.
- Verwenden Sie die union Klausel statt der union all Klausel um Kopien des gleichen Datensatzes aus der Ergebnismengen der Abfrage zu entfernen.

**Codebeispiel: union, union all Klausel**

```

1 -- Union, Union all Klausel
2
3
4 SELECT last_name, first_name
5 FROM employees
6 WHERE salary < (SELECT min(salary) FROM emp)
7 --> UNION ALL <-
8 SELECT last_name, first_name
9 FROM employees
10 WHERE salary > (SELECT max(salary) FROM emp);
```

66

| Konzept           | Beschreibung                                                                                                                | Mengenoperation |
|-------------------|-----------------------------------------------------------------------------------------------------------------------------|-----------------|
| Schnittmenge      | Die Schnittmenge zweier Mengen A und B ist die Menge aller Elemente, die sowohl zu A als auch zu B gehören.                 | INTERSECT       |
| Vereinigungsmenge | Die Vereinigungsmenge zweier Mengen A und B ist die Menge aller Elemente, die zu A oder zu B oder zu beiden Mengen gehören. | UNION           |
| Differenzmenge    | Die Differenzmenge zweier Mengen A und B ist die Menge aller Elemente, die zu A, nicht aber zu B gehören.                   | EXCEPT          |

Abbildung 27. Mengentheoretische Operationen

**6.3.3 Intersect Klausel - Durchschnittsmenge**

Mit der intersect Klausel kann die Durchschnittsmenge der Ergebnismengen zweier oder mehrerer select Anweisungen ermittelt werden.

► Syntax: **intersect Klausel** □

```

1 --- -----
2 --- Syntax: intersect Klausel
3 --- -----
4 SELECT Ausdruck1, Ausdruck2, ... Ausdruck_n
5 FROM Tabellen
6 INTERSECT
7 SELECT Ausdruck1, Ausdruck2, ... Ausdruck_n
8 FROM Tabellen;
9
10 ---
11 --- intersect Klausel
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
587
588
589
589
590
591
592
593
594
595
596
597
597
598
599
599
600
601
602
603
604
605
606
607
607
608
609
609
610
611
612
613
614
615
615
616
617
617
618
619
619
620
621
622
623
623
624
625
625
626
627
627
628
629
629
630
631
631
632
633
633
634
635
635
636
637
637
638
639
639
640
641
641
642
642
643
643
644
644
645
645
646
646
647
647
648
648
649
649
650
650
651
651
652
652
653
653
654
654
655
655
656
656
657
657
658
658
659
659
660
660
661
661
662
662
663
663
664
664
665
665
666
666
667
667
668
668
669
669
670
670
671
671
672
672
673
673
674
674
675
675
676
676
677
677
678
678
679
679
680
680
681
681
682
682
683
683
684
684
685
685
686
686
687
687
688
688
689
689
690
690
691
691
692
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
700
701
701
702
702
703
703
704
704
705
705
706
706
707
707
708
708
709
709
710
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1501
1502
1502
1503
1
```

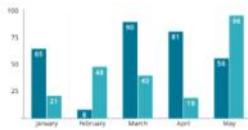
| Funktion            | Beschreibung                                                                                                          | Seite |
|---------------------|-----------------------------------------------------------------------------------------------------------------------|-------|
| SYS_CONNECT_BY_PATH | Die SYS_CONNECT_BY_PATH Funktion generiert aus der Liste aller Vorfahren des Datensatzes einen Zeichenkettenausdruck. | 69    |
| RPAD                | Mit der RPAD Funktion kann eine Zeichenkette rechtsbündig mit einem vorgegebenen Zeichen aufgefüllt werden.           | 69    |

Abbildung 29. Hierarchische Abfragen: Funktionen

## 6.4.3 Hierarchische Abfragen: Pseudospalten

Zusammen mit der connect by Klausel definiert die SQL Spezifikation eine Reihe von Pseudospalten zur Formulierung von hierarchischen Abfragen.

Die Pseudospalten der connect by Klausel halten zusätzliche Informationen zur Beschreibung hierarchischer Strukturen.



## \* Codebeispiel: Pseudospalten \*

```
-- Tabelledefinition: trees
CREATE TABLE trees (
    id NUMBER,
    parent_id NUMBER,
    CONSTRAINT trees_pk PRIMARY KEY (id),
    CONSTRAINT trees_fk FOREIGN KEY (parent_id)
        REFERENCES tabl(id)
);
SELECT id, parent_id
    LTRIM(SYS_CONNECT_BY_PATH(id,'-'),'-'),
    RPAD('-',(LEVEL - 1)*2,'.') || id,
    CONNECT_BY_ISLEAF AS leaf,
    FROM trees
    START WITH parent_id IS null
    CONNECT BY parent_id = PRIOR id
    ORDER SIBLINGS BY id;
```

69

## 6.4.4 Hierarchische Abfragen: Funktionen

Zusammen mit der connect by Klausel definiert die SQL Spezifikation eine Reihe von Funktionen zur Formulierung von hierarchischen Abfragen.

## \* Codebeispiel: Funktionen \*

```
-- Syntax: sys_connect_by_path
-- Die Funktion generiert aus der Liste
-- aller Vorfahren des Datensatzes eine
-- Zeichenkettenausdruck.
sys_connect_by_path(
    label IN nameXTYPE,
    separator IN VARCHAR2
);
-- Syntax: RPAD
-- Mit der RPAD Funktion kann eine Zeichen-
-- kette rechtsbündig mit einem vorgegebenem
-- Zeichen aufgefüllt werden.
rpad(
    value IN VARCHAR2,
    size IN NUMBER,
    literal IN CHAR
);
SELECT id, parent_id
    LTRIM(sys_connect_by_path(id,'-'),'-'),
    RPAD('-',(LEVEL - 1)*2,'.') || id,
    CONNECT_BY_ISLEAF AS leaf,
```

69

## 6.5. Quantoren

## Quantoren

Quantoren sind logische Operatoren mit denen eine Menge von Werten auf eine bestimmte Eigenschaft geprüft werden kann.

## 6.5.1 ALL Operator

Beim all Operator handelt es sich um einen logischen Operator.

Der all Operator evaluierst zu true wenn alle Elemente einer Menge eine bestimmte Bedingung erfüllen.

## \* Syntax: All Operator \*

```
-- Syntax: ALL Operator
SELECT <expression>
FROM <expression>
WHERE column_name >operator ALL
    (SELECT <expression> ... )
-- Finden Sie alle Angestellten deren Gehalt
-- höher ist als alle Werte einer Liste.
SELECT first_name, last_name
    salary
FROM employees
WHERE salary > ALL (
    5000, 6000, 10000
)
ORDER BY last_name;
```

## 6.5.2 EXISTS Operator

Beim exists Operator handelt es sich um einen logischen Operator.

Der exists Operator evaluierst zu true wenn zumindest ein Element einer Menge eine bestimmte Bedingung erfüllt.

## \* Syntax: Exists Operator \*

```
-- Syntax: EXISTS Operator
SELECT <expression>
FROM <expression>
WHERE EXISTS (
    (SELECT <expression> ... )
-- Finden Sie alle Angestellten die das hoechste Gehalt im Unternehmen besitzen.
SELECT e.first_name, e.last_name
    e.department_name,
    e.department_id,
    e.salary
FROM employees e
JOIN department d
ON e.department_id = d.department_id
WHERE NOT EXISTS (
    SELECT e2.last_name
    FROM employees e2
    WHERE e2.salary > e.salary
)
ORDER BY last_name;
```

69

## 7. SQL - Data Definition Language

# 06

DDL - Data Definition Language

|                        |    |
|------------------------|----|
| 01. Datenbankartefakte | 72 |
| 02. Tabelle            | 73 |
| 03. Constraint         | 76 |
| 04. View               | 78 |
| 05. Index              | 79 |
| 06. Sequenz            | 82 |

### 7.1. Datenbankartefakte

#### Data Definition Language ▾

Mit **DDL Befehle** kann die Struktur einer relationalen Datenbank definiert bzw. geändert werden.

#### 7.1.1 Datenbankartefakte

Datenbankartefakte beschreiben die Struktur einer Datenbank.



#### Auflistung: Datenbankartefakte ▾

- Schema: Ein Schema ist ein logischer **Namensraum** für die Tabellen eines Geschäftsfalls. Im Sprachgebrauch wird ein Schema auch als Geschäftsfall bezeichnet. Ein Datenbankserver kann eine beliebige Zahl von Schemen verwalten.
- Tabelle: Ein Datenbankschema ist ein logischer Namensraum für die Daten eines Geschäftsfalls. Tabellen repräsentieren dabei die einzelnen Aspekte der Geschäftsfälle. Aus technischer Sicht werden in Tabellen gleichartige **Datensätze** gesammelt.
- Constraints: Constraints werden verwendet um die **Konsistenz** der Daten in einer Datenbank sicherzustellen. SQL unterstützt mehrere Formen von Constraints. Damit können die primären **Konsistenzanforderungen** sichergestellt werden.
- View: Eine View ermöglicht es Datenbanksfragen wie Datenbankobjekte zu behandeln. SQL Queries können so für den späteren Gebrauch gespeichert werden.
- Sequenz: Mit der Hilfe einer Datenbanksequenz können Sequenzen aufeinanderfolgender Werte generiert werden. Sequenzen werden in der Regel zur Generierung von Schlüsselwerten definiert.
- Trigger: Trigge sind Datenbankaktionen deren Ausführung durch bestimmte **Ereignisse** ausgelöst wird.
- Index: Ein Datenbankindex ist eine **Datenstruktur** zur Beschleunigung der Suche und Sortierung von Datensätzen.

□

| Befehl          | Beschreibung                                                                                                                                                                                                                    | Seite |
|-----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------|
| CREATE TABLE    | Der CREATE TABLE Befehl wird zum Anlegen einer Tabelle in der Datenbank verwendet. Der Befehl besitzt dabei 2 unterschiedliche Ausprägungen um die Struktur von Tabellen zu definieren.                                         | 74    |
| ALTER TABLE     | Der ALTER TABLE Befehl wird zum Beobachten der Struktur einer Tabelle in der Datenbank verwendet. Der Befehl besitzt dabei unterschiedliche Ausprägungen und Optionen zum Löschen bzw. Hinzufügen von Feldern bzw. Constraints. | 75    |
| DROP TABLE      | Der DROP TABLE Befehl wird zum Löschen einer Tabelle in der Datenbank verwendet.                                                                                                                                                | 74    |
| TRUNCATE TABLE  | Der TRUNCATE TABLE Befehl wird zum Löschen der Datensätze einer Tabelle verwendet.                                                                                                                                              | 75    |
| CREATE VIEW     | Der CREATE VIEW Befehl zum Anlegen einer View in der Datenbank verwendet.                                                                                                                                                       | 78    |
| DROP VIEW       | Der DROP VIEW Befehl zum Löschen einer View in der Datenbank verwendet.                                                                                                                                                         | 79    |
| CREATE SEQUENCE | Der CREATE SEQUENCE Befehl zum Anlegen einer Sequenz in der Datenbank verwendet. Der Befehl besitzt unterschiedliche Optionen um die Sequenz an den geforderten Geschäftsfall anzupassen zu können.                             | 77    |
| DROP SEQUENCE   | Der DROP SEQUENCE Befehl zum Löschen einer Sequenz aus der Datenbank verwendet.                                                                                                                                                 | 77    |

Abbildung 30. DDL Befehle

**7.2. Datenbankkarteikart: Tabelle**

Eine Datenbanktabelle repräsentiert einen einzelnen Aspekt eines spezifischen Geschäftsfalls.

**7.2.1 Tabelle**

**Datenbanktabelle**  
Datenbanktabellen bestehen aus Zeilen und Spalten. Daten werden in Form von Datensätzen in einer Tabelle gespeichert.

- Zelle:** Jede Zelle einer Tabelle speichert einen einzelnen Datensatz.
- Spalte:** Die Spalten einer Tabelle beschreiben die Attribute der Datensätze.

→ Erklärung: Datenbanktabellen

In einer Datenbanktabelle werden alle gleichartigen Datensätze eines Geschäftsfalls gesammelt. Der Einsatz von Tabellen hilft dabei bei der logischen und physischen Strukturierung der Daten.

**7.2.2 Tabellendefinition**

**tabellendefinition**  
Zur Beschreibung einer Tabelle werden die Eigenschaften der einzelnen Spalten der Tabelle definiert. Spaltendefinitionen folgen dabei einem strikt vorgegebenem Muster:  
spaltenbasierter datatype [attribute+]

→ Erklärung: Spaltendefinition

- Spaltenbezeichner:** Für eine Tabellenspalte kann ein beliebiger Spaltenbezeichner gewählt werden. Es ist jedoch darauf zu achten, dass der Bezeichner nicht mehr als 32 Zeichen lang sein kann, sowie keine Sonderzeichen enthalten darf.
- Datentyp:** Tabellenspalten haben immer einen spezifischen Datentyp. Lediglich Werte die nun angegebenen Datentyp kompatibel sind, können in eine Spalte eingetragen werden.
- Attribute:** Optional können für eine Spalte eine Reihe von Attributen definiert werden.

**Informationssysteme**

**7.2.3 Schlüsseldefinition**

Eine der Spalten einer Tabelle muss als Schlüsselspalte designiert werden.

**Primärschlüssel**  
Anhand eines Primärschlüssels können die in der Tabelle gespeicherten Datensätze voneinander unterscheiden werden.

→ Erklärung: Schlüsselspalten

- Der **Schlüssel** einer Tabelle besteht dabei aus einer oder mehreren Spalten der Tabelle. Ein Schlüssel darf jedoch maximal nur so viele Spalten enthalten, wie notwendig sind, um die Datensätze der Tabelle zu differenzieren.
- Bestimmte Spalten einer Tabelle können auch als Fremdschlüsselspalten eingesetzt werden. Fremdschlüssel sind Verweise auf die Primärschlüssele anderer Tabellen. Über Fremdschlüssel werden die Daten unterschiedlicher Tabellen in **Relation** gesetzt.

**7.2.4 DDL Befehl: create table**

Der create table Befehl wird zum Anlegen von Tabellen in einer Datenbank verwendet.

→ Syntax: create table

```

1 -- -----
2 -- Syntax: CREATE TABLE
3 -- -----
4 CREATE TABLE table_name (
5   column1 datatype [NOT NULL|NULL|UNIQUE],
6   ...
7   columnN datatype,
8   PRIMARY KEY (column_name),
9   [CONSTRAINT fk_name FOREIGN KEY
10    (column_name)
11    REFERENCES table_name (column_name)
12    [ON DELETE CASCADE]]
13 );
14 
15 CREATE TABLE table_name AS
16   SELECT column1, column2, ...
17   FROM existing_table_name;

```

**7.2.5 Fallbeispiel: create table**

Der create table Befehl wird zum Anlegen von Tabellen verwendet.

→ DDL: employees, departments

```

1 -- -----
2 -- Fallbeispiel: hr schema
3 -- -----
4 CREATE TABLE employees (
5   employee_id NUMBER(19,0) NOT NULL,
6   salary NUMBER(10,0) NOT NULL,
7   job_id NUMBER(19,0) NOT NULL,
8   first_name VARCHAR(30) NOT NULL,
9   middle_name VARCHAR(30) NOT NULL,
10  last_name VARCHAR(30) NOT NULL,
11  department_id NUMBER(19,0) NOT NULL,
12 
13  PRIMARY KEY (employee_id),
14  CONSTRAINT fk_departments_dep_id
15    FOREIGN KEY (department_id)
16      REFERENCES departments (department_id)
17 );
18 
19 CREATE TABLE departments (
20   department_id NUMBER(19,0) NOT NULL UNIQUE,
21   location_id NUMBER(19,0) NOT NULL,
22   dep_name VARCHAR(30) NOT NULL,
23 
24  PRIMARY KEY (department_id),
25  CONSTRAINT fk_location_department_id
26    FOREIGN KEY (location_id)
27      REFERENCES locations (location_id)
28      ON DELETE CASCADE
29 );

```

**7.2.6 DDL Befehl: drop table**

Der drop table Befehl wird zum Löschen von Tabellen aus der Datenbank verwendet.

→ DDL: Tabellen verwalten

```

1 -- -----
2 -- Befehl: DROP TABLE
3 -- -----
4 -- Befehl: Table Artefakte löschen
5 DROP TABLE table_name;
6 
7 DROP TABLE students;

```

**7.2.7 DDL Befehl: truncate table**

Der truncate table Befehl wird zum Löschen aller Datensätze einer Tabelle verwendet.

## » DDL: Datensätze löschen ▾

```

1 -- -----
2   -- Befehl: TRUNCATE TABLE
3   --
4   -- Befehl: Table Artsfakte loeschen
5   TRUNCATE TABLE table_name;
6   --
7   TRUNCATE TABLE students;

```

**7.2.8 DDL Befehl: alter table**

Der alter table Befehl wird verwendet um die Struktur der Tabellen einer Datenbank zu ändern.

## » Syntax: alter table ▾

```

1 -- -----
2   -- Syntax: ALTER TABLE
3   --
4   -- Der ALTER TABLE Befehl besitzt unter-
5   -- schiedliche Ausprägungen und Optionen
6   -- für das Löschen bzw. Hinzufügen von
7   -- Feldern bzw. Constraints.
8   --
9   ALTER TABLE table_name {
10    RENAME TABLE TO new_table_name |
11    MODIFY (column action) |
12    ADD (column datatype attribute) |
13    DROP (column) |
14    ADD CONSTRAINT fk_name
15      REFERENCES table (column_name) |
16    RENAME CONSTRAINT old TO new |
17    DROP CONSTRAINT constraint_name |
18    DROP PRIMARY KEY
19  };
20  --
21  -- Fallbeispiel: ALTER TABLE
22  --
23  ALTER TABLE projects RENAME TO c_projects;
24  --
25  ALTER TABLE c_projects ADD (
26    is_ffg_sponsored NUMBER(1),
27    is_fwf_sponsored NUMBER(1),
28    is_eu_sponsored NUMBER(1)
29  );
30  --
31  -- Fallbeispiel: ALTER TABLE
32  --
33  ALTER TABLE c_projects ADD (
34    is_ffg_sponsored NUMBER(1) NOT NULL,
35    is_fwf_sponsored NUMBER(1) NOT NULL,
36    is_eu_sponsored NUMBER(1) NOT NULL
37  );
38  --
39  -- Spalten hinzufügen
40  --
41  ALTER TABLE c_projects ADD description
42    VARCHAR(265);
43  --
44  ALTER TABLE c_projects ADD (
45    is_ffg_sponsored NUMBER(1),
46    is_fwf_sponsored NUMBER(1),
47    is_eu_sponsored NUMBER(1)
48  );
49  --
50  -- Spalten hinzufügen
51  --
52  ALTER TABLE c_projects ADD description
53    VARCHAR(265);
54  --
55  ALTER TABLE c_projects ADD (
56    is_ffg_sponsored NUMBER(1) NOT NULL,
57    is_fwf_sponsored NUMBER(1) NOT NULL,
58    is_eu_sponsored NUMBER(1) NOT NULL
59  );
60  --
61  -- Spalten hinzufügen
62  --
63  ALTER TABLE c_projects ADD (
64    is_ffg_sponsored NUMBER(1),
65    is_fwf_sponsored NUMBER(1),
66    is_eu_sponsored NUMBER(1)
67  );
68  --
69  -- Spalten hinzufügen
70  --
71  ALTER TABLE c_projects ADD (
72    is_ffg_sponsored NUMBER(1) NOT NULL,
73    is_fwf_sponsored NUMBER(1) NOT NULL,
74    is_eu_sponsored NUMBER(1) NOT NULL
75  );

```

**7.2.9 Fallbeispiel: alter table**

Der alter table Befehl wird verwendet um die Struktur der Tabellen der Datenbank zu verwalten.

## » DDL: alter table Beispiele ▾

```

1 -- -----
2   -- Fallbeispiel: ALTER TABLE
3   --
4   -- Befehl: Constraints hinzufügen/lösen
5   ALTER TABLE c_project_partners
6     ADD CONSTRAINT fk_project_partners
7       FOREIGN KEY project_id
8         REFERENCES c_projects (project_id);
9   --
10  ALTER TABLE c_projects ADD CONSTRAINT
11    uk_projects_code UNIQUE (project_code);
12  --
13  ALTER TABLE c_projects DROP CONSTRAINT
14    uk_projects_code;
15  --
16  ALTER TABLE c_projects ADD CONSTRAINT
17    check_legal_type CHECK
18      (legal.foundation IN ('P_26', 'P_27'));
19  --
20  -- Befehl Constraints ein-/ausschalten
21  ALTER TABLE c_projects DISABLE CONSTRAINT
22    uk_projects_code;
23  --
24  ALTER TABLE c_projects ENABLE CONSTRAINT
25    uk_projects_code;
26  --
27  -- Spaltenmerkmale versenden
28  ALTER TABLE c_projects
29    MODIFY description VARCHAR2(4000);
30  --
31  ALTER TABLE c_projects ADD CONSTRAINT
32    is_ffg_sponsored NUMBER(1) NOT NULL,
33    is_fwf_sponsored NUMBER(1) NOT NULL,
34    is_eu_sponsored NUMBER(1) NOT NULL;
35  --
36  -- Spalten hinzufügen
37  --
38  ALTER TABLE c_projects ADD description
39    VARCHAR(265);
40  --
41  ALTER TABLE c_projects ADD (
42    is_ffg_sponsored NUMBER(1),
43    is_fwf_sponsored NUMBER(1),
44    is_eu_sponsored NUMBER(1)
45  );
46  --
47  -- Spalten hinzufügen
48  --
49  ALTER TABLE c_projects ADD (
50    is_ffg_sponsored NUMBER(1) NOT NULL,
51    is_fwf_sponsored NUMBER(1) NOT NULL,
52    is_eu_sponsored NUMBER(1) NOT NULL
53  );
54  --
55  -- Spalten hinzufügen
56  --
57  ALTER TABLE c_projects ADD (
58    is_ffg_sponsored NUMBER(1),
59    is_fwf_sponsored NUMBER(1),
60    is_eu_sponsored NUMBER(1)
61  );
62  --
63  -- Spalten hinzufügen
64  --
65  ALTER TABLE c_projects ADD (
66    is_ffg_sponsored NUMBER(1) NOT NULL,
67    is_fwf_sponsored NUMBER(1) NOT NULL,
68    is_eu_sponsored NUMBER(1) NOT NULL
69  );
70  --
71  -- Spalten hinzufügen
72  --
73  ALTER TABLE c_projects ADD (
74    is_ffg_sponsored NUMBER(1),
75    is_fwf_sponsored NUMBER(1),
76    is_eu_sponsored NUMBER(1)
77  );
78  --
79  -- Spalten hinzufügen
80  --
81  ALTER TABLE c_projects ADD (
82    is_ffg_sponsored NUMBER(1) NOT NULL,
83    is_fwf_sponsored NUMBER(1) NOT NULL,
84    is_eu_sponsored NUMBER(1) NOT NULL
85  );
86  --
87  -- Spalten hinzufügen
88  --
89  ALTER TABLE c_projects ADD (
90    is_ffg_sponsored NUMBER(1),
91    is_fwf_sponsored NUMBER(1),
92    is_eu_sponsored NUMBER(1)
93  );
94  --
95  -- Spalten hinzufügen
96  --
97  ALTER TABLE c_projects ADD (
98    is_ffg_sponsored NUMBER(1) NOT NULL,
99    is_fwf_sponsored NUMBER(1) NOT NULL,
100   is_eu_sponsored NUMBER(1) NOT NULL
101  );
102  --
103  -- Spalten hinzufügen
104  --
105  ALTER TABLE c_projects ADD (
106    is_ffg_sponsored NUMBER(1),
107    is_fwf_sponsored NUMBER(1),
108    is_eu_sponsored NUMBER(1)
109  );
110  --
111  -- Spalten hinzufügen
112  --
113  ALTER TABLE c_projects ADD (
114    is_ffg_sponsored NUMBER(1) NOT NULL,
115    is_fwf_sponsored NUMBER(1) NOT NULL,
116    is_eu_sponsored NUMBER(1) NOT NULL
117  );
118  --
119  -- Spalten hinzufügen
120  --
121  ALTER TABLE c_projects ADD (
122    is_ffg_sponsored NUMBER(1),
123    is_fwf_sponsored NUMBER(1),
124    is_eu_sponsored NUMBER(1)
125  );
126  --
127  -- Spalten hinzufügen
128  --
129  ALTER TABLE c_projects ADD (
130    is_ffg_sponsored NUMBER(1) NOT NULL,
131    is_fwf_sponsored NUMBER(1) NOT NULL,
132    is_eu_sponsored NUMBER(1) NOT NULL
133  );
134  --
135  -- Spalten hinzufügen
136  --
137  ALTER TABLE c_projects ADD (
138    is_ffg_sponsored NUMBER(1),
139    is_fwf_sponsored NUMBER(1),
140    is_eu_sponsored NUMBER(1)
141  );
142  --
143  -- Spalten hinzufügen
144  --
145  ALTER TABLE c_projects ADD (
146    is_ffg_sponsored NUMBER(1) NOT NULL,
147    is_fwf_sponsored NUMBER(1) NOT NULL,
148    is_eu_sponsored NUMBER(1) NOT NULL
149  );
150  --
151  -- Spalten hinzufügen
152  --
153  ALTER TABLE c_projects ADD (
154    is_ffg_sponsored NUMBER(1),
155    is_fwf_sponsored NUMBER(1),
156    is_eu_sponsored NUMBER(1)
157  );
158  --
159  -- Spalten hinzufügen
160  --
161  ALTER TABLE c_projects ADD (
162    is_ffg_sponsored NUMBER(1) NOT NULL,
163    is_fwf_sponsored NUMBER(1) NOT NULL,
164    is_eu_sponsored NUMBER(1) NOT NULL
165  );
166  --
167  -- Spalten hinzufügen
168  --
169  ALTER TABLE c_projects ADD (
170    is_ffg_sponsored NUMBER(1),
171    is_fwf_sponsored NUMBER(1),
172    is_eu_sponsored NUMBER(1)
173  );
174  --
175  -- Spalten hinzufügen
176  --
177  ALTER TABLE c_projects ADD (
178    is_ffg_sponsored NUMBER(1) NOT NULL,
179    is_fwf_sponsored NUMBER(1) NOT NULL,
180    is_eu_sponsored NUMBER(1) NOT NULL
181  );
182  --
183  -- Spalten hinzufügen
184  --
185  ALTER TABLE c_projects ADD (
186    is_ffg_sponsored NUMBER(1),
187    is_fwf_sponsored NUMBER(1),
188    is_eu_sponsored NUMBER(1)
189  );
190  --
191  -- Spalten hinzufügen
192  --
193  ALTER TABLE c_projects ADD (
194    is_ffg_sponsored NUMBER(1) NOT NULL,
195    is_fwf_sponsored NUMBER(1) NOT NULL,
196    is_eu_sponsored NUMBER(1) NOT NULL
197  );
198  --
199  -- Spalten hinzufügen
200  --
201  ALTER TABLE c_projects ADD (
202    is_ffg_sponsored NUMBER(1),
203    is_fwf_sponsored NUMBER(1),
204    is_eu_sponsored NUMBER(1)
205  );
206  --
207  -- Spalten hinzufügen
208  --
209  ALTER TABLE c_projects ADD (
210    is_ffg_sponsored NUMBER(1) NOT NULL,
211    is_fwf_sponsored NUMBER(1) NOT NULL,
212    is_eu_sponsored NUMBER(1) NOT NULL
213  );
214  --
215  -- Spalten hinzufügen
216  --
217  ALTER TABLE c_projects ADD (
218    is_ffg_sponsored NUMBER(1),
219    is_fwf_sponsored NUMBER(1),
220    is_eu_sponsored NUMBER(1)
221  );
222  --
223  -- Spalten hinzufügen
224  --
225  ALTER TABLE c_projects ADD (
226    is_ffg_sponsored NUMBER(1) NOT NULL,
227    is_fwf_sponsored NUMBER(1) NOT NULL,
228    is_eu_sponsored NUMBER(1) NOT NULL
229  );
230  --
231  -- Spalten hinzufügen
232  --
233  ALTER TABLE c_projects ADD (
234    is_ffg_sponsored NUMBER(1),
235    is_fwf_sponsored NUMBER(1),
236    is_eu_sponsored NUMBER(1)
237  );
238  --
239  -- Spalten hinzufügen
240  --
241  ALTER TABLE c_projects ADD (
242    is_ffg_sponsored NUMBER(1) NOT NULL,
243    is_fwf_sponsored NUMBER(1) NOT NULL,
244    is_eu_sponsored NUMBER(1) NOT NULL
245  );
246  --
247  -- Spalten hinzufügen
248  --
249  ALTER TABLE c_projects ADD (
250    is_ffg_sponsored NUMBER(1),
251    is_fwf_sponsored NUMBER(1),
252    is_eu_sponsored NUMBER(1)
253  );
254  --
255  -- Spalten hinzufügen
256  --
257  ALTER TABLE c_projects ADD (
258    is_ffg_sponsored NUMBER(1) NOT NULL,
259    is_fwf_sponsored NUMBER(1) NOT NULL,
260    is_eu_sponsored NUMBER(1) NOT NULL
261  );
262  --
263  -- Spalten hinzufügen
264  --
265  ALTER TABLE c_projects ADD (
266    is_ffg_sponsored NUMBER(1),
267    is_fwf_sponsored NUMBER(1),
268    is_eu_sponsored NUMBER(1)
269  );
270  --
271  -- Spalten hinzufügen
272  --
273  ALTER TABLE c_projects ADD (
274    is_ffg_sponsored NUMBER(1) NOT NULL,
275    is_fwf_sponsored NUMBER(1) NOT NULL,
276    is_eu_sponsored NUMBER(1) NOT NULL
277  );
278  --
279  -- Spalten hinzufügen
280  --
281  ALTER TABLE c_projects ADD (
282    is_ffg_sponsored NUMBER(1),
283    is_fwf_sponsored NUMBER(1),
284    is_eu_sponsored NUMBER(1)
285  );
286  --
287  -- Spalten hinzufügen
288  --
289  ALTER TABLE c_projects ADD (
290    is_ffg_sponsored NUMBER(1) NOT NULL,
291    is_fwf_sponsored NUMBER(1) NOT NULL,
292    is_eu_sponsored NUMBER(1) NOT NULL
293  );
294  --
295  -- Spalten hinzufügen
296  --
297  ALTER TABLE c_projects ADD (
298    is_ffg_sponsored NUMBER(1),
299    is_fwf_sponsored NUMBER(1),
300    is_eu_sponsored NUMBER(1)
301  );
302  --
303  -- Spalten hinzufügen
304  --
305  ALTER TABLE c_projects ADD (
306    is_ffg_sponsored NUMBER(1) NOT NULL,
307    is_fwf_sponsored NUMBER(1) NOT NULL,
308    is_eu_sponsored NUMBER(1) NOT NULL
309  );
310  --
311  -- Spalten hinzufügen
312  --
313  ALTER TABLE c_projects ADD (
314    is_ffg_sponsored NUMBER(1),
315    is_fwf_sponsored NUMBER(1),
316    is_eu_sponsored NUMBER(1)
317  );
318  --
319  -- Spalten hinzufügen
320  --
321  ALTER TABLE c_projects ADD (
322    is_ffg_sponsored NUMBER(1) NOT NULL,
323    is_fwf_sponsored NUMBER(1) NOT NULL,
324    is_eu_sponsored NUMBER(1) NOT NULL
325  );
326  --
327  -- Spalten hinzufügen
328  --
329  ALTER TABLE c_projects ADD (
330    is_ffg_sponsored NUMBER(1),
331    is_fwf_sponsored NUMBER(1),
332    is_eu_sponsored NUMBER(1)
333  );
334  --
335  -- Spalten hinzufügen
336  --
337  ALTER TABLE c_projects ADD (
338    is_ffg_sponsored NUMBER(1) NOT NULL,
339    is_fwf_sponsored NUMBER(1) NOT NULL,
340    is_eu_sponsored NUMBER(1) NOT NULL
341  );
342  --
343  -- Spalten hinzufügen
344  --
345  ALTER TABLE c_projects ADD (
346    is_ffg_sponsored NUMBER(1),
347    is_fwf_sponsored NUMBER(1),
348    is_eu_sponsored NUMBER(1)
349  );
350  --
351  -- Spalten hinzufügen
352  --
353  ALTER TABLE c_projects ADD (
354    is_ffg_sponsored NUMBER(1) NOT NULL,
355    is_fwf_sponsored NUMBER(1) NOT NULL,
356    is_eu_sponsored NUMBER(1) NOT NULL
357  );
358  --
359  -- Spalten hinzufügen
360  --
361  ALTER TABLE c_projects ADD (
362    is_ffg_sponsored NUMBER(1),
363    is_fwf_sponsored NUMBER(1),
364    is_eu_sponsored NUMBER(1)
365  );
366  --
367  -- Spalten hinzufügen
368  --
369  ALTER TABLE c_projects ADD (
370    is_ffg_sponsored NUMBER(1) NOT NULL,
371    is_fwf_sponsored NUMBER(1) NOT NULL,
372    is_eu_sponsored NUMBER(1) NOT NULL
373  );
374  --
375  -- Spalten hinzufügen
376  --
377  ALTER TABLE c_projects ADD (
378    is_ffg_sponsored NUMBER(1),
379    is_fwf_sponsored NUMBER(1),
380    is_eu_sponsored NUMBER(1)
381  );
382  --
383  -- Spalten hinzufügen
384  --
385  ALTER TABLE c_projects ADD (
386    is_ffg_sponsored NUMBER(1) NOT NULL,
387    is_fwf_sponsored NUMBER(1) NOT NULL,
388    is_eu_sponsored NUMBER(1) NOT NULL
389  );
390  --
391  -- Spalten hinzufügen
392  --
393  ALTER TABLE c_projects ADD (
394    is_ffg_sponsored NUMBER(1),
395    is_fwf_sponsored NUMBER(1),
396    is_eu_sponsored NUMBER(1)
397  );
398  --
399  -- Spalten hinzufügen
400  --
401  ALTER TABLE c_projects ADD (
402    is_ffg_sponsored NUMBER(1) NOT NULL,
403    is_fwf_sponsored NUMBER(1) NOT NULL,
404    is_eu_sponsored NUMBER(1) NOT NULL
405  );
406  --
407  -- Spalten hinzufügen
408  --
409  ALTER TABLE c_projects ADD (
410    is_ffg_sponsored NUMBER(1),
411    is_fwf_sponsored NUMBER(1),
412    is_eu_sponsored NUMBER(1)
413  );
414  --
415  -- Spalten hinzufügen
416  --
417  ALTER TABLE c_projects ADD (
418    is_ffg_sponsored NUMBER(1) NOT NULL,
419    is_fwf_sponsored NUMBER(1) NOT NULL,
420    is_eu_sponsored NUMBER(1) NOT NULL
421  );
422  --
423  -- Spalten hinzufügen
424  --
425  ALTER TABLE c_projects ADD (
426    is_ffg_sponsored NUMBER(1),
427    is_fwf_sponsored NUMBER(1),
428    is_eu_sponsored NUMBER(1)
429  );
430  --
431  -- Spalten hinzufügen
432  --
433  ALTER TABLE c_projects ADD (
434    is_ffg_sponsored NUMBER(1) NOT NULL,
435    is_fwf_sponsored NUMBER(1) NOT NULL,
436    is_eu_sponsored NUMBER(1) NOT NULL
437  );
438  --
439  -- Spalten hinzufügen
440  --
441  ALTER TABLE c_projects ADD (
442    is_ffg_sponsored NUMBER(1),
443    is_fwf_sponsored NUMBER(1),
444    is_eu_sponsored NUMBER(1)
445  );
446  --
447  -- Spalten hinzufügen
448  --
449  ALTER TABLE c_projects ADD (
450    is_ffg_sponsored NUMBER(1) NOT NULL,
451    is_fwf_sponsored NUMBER(1) NOT NULL,
452    is_eu_sponsored NUMBER(1) NOT NULL
453  );
454  --
455  -- Spalten hinzufügen
456  --
457  ALTER TABLE c_projects ADD (
458    is_ffg_sponsored NUMBER(1),
459    is_fwf_sponsored NUMBER(1),
460    is_eu_sponsored NUMBER(1)
461  );
462  --
463  -- Spalten hinzufügen
464  --
465  ALTER TABLE c_projects ADD (
466    is_ffg_sponsored NUMBER(1) NOT NULL,
467    is_fwf_sponsored NUMBER(1) NOT NULL,
468    is_eu_sponsored NUMBER(1) NOT NULL
469  );
470  --
471  -- Spalten hinzufügen
472  --
473  ALTER TABLE c_projects ADD (
474    is_ffg_sponsored NUMBER(1),
475    is_fwf_sponsored NUMBER(1),
476    is_eu_sponsored NUMBER(1)
477  );
478  --
479  -- Spalten hinzufügen
480  --
481  ALTER TABLE c_projects ADD (
482    is_ffg_sponsored NUMBER(1) NOT NULL,
483    is_fwf_sponsored NUMBER(1) NOT NULL,
484    is_eu_sponsored NUMBER(1) NOT NULL
485  );
486  --
487  -- Spalten hinzufügen
488  --
489  ALTER TABLE c_projects ADD (
490    is_ffg_sponsored NUMBER(1),
491    is_fwf_sponsored NUMBER(1),
492    is_eu_sponsored NUMBER(1)
493  );
494  --
495  -- Spalten hinzufügen
496  --
497  ALTER TABLE c_projects ADD (
498    is_ffg_sponsored NUMBER(1) NOT NULL,
499    is_fwf_sponsored NUMBER(1) NOT NULL,
500    is_eu_sponsored NUMBER(1) NOT NULL
501  );
502  --
503  -- Spalten hinzufügen
504  --
505  ALTER TABLE c_projects ADD (
506    is_ffg_sponsored NUMBER(1),
507    is_fwf_sponsored NUMBER(1),
508    is_eu_sponsored NUMBER(1)
509  );
510  --
511  -- Spalten hinzufügen
512  --
513  ALTER TABLE c_projects ADD (
514    is_ffg_sponsored NUMBER(1) NOT NULL,
515    is_fwf_sponsored NUMBER(1) NOT NULL,
516    is_eu_sponsored NUMBER(1) NOT NULL
517  );
518  --
519  -- Spalten hinzufügen
520  --
521  ALTER TABLE c_projects ADD (
522    is_ffg_sponsored NUMBER(1),
523    is_fwf_sponsored NUMBER(1),
524    is_eu_sponsored NUMBER(1)
525  );
526  --
527  -- Spalten hinzufügen
528  --
529  ALTER TABLE c_projects ADD (
530    is_ffg_sponsored NUMBER(1) NOT NULL,
531    is_fwf_sponsored NUMBER(1) NOT NULL,
532    is_eu_sponsored NUMBER(1) NOT NULL
533  );
534  --
535  -- Spalten hinzufügen
536  --
537  ALTER TABLE c_projects ADD (
538    is_ffg_sponsored NUMBER(1),
539    is_fwf_sponsored NUMBER(1),
540    is_eu_sponsored NUMBER(1)
541  );
542  --
543  -- Spalten hinzufügen
544  --
545  ALTER TABLE c_projects ADD (
546    is_ffg_sponsored NUMBER(1) NOT NULL,
547    is_fwf_sponsored NUMBER(1) NOT NULL,
548    is_eu_sponsored NUMBER(1) NOT NULL
549  );
550  --
551  -- Spalten hinzufügen
552  --
553  ALTER TABLE c_projects ADD (
554    is_ffg_sponsored NUMBER(1),
555    is_fwf_sponsored NUMBER(1),
556    is_eu_sponsored NUMBER(1)
557  );
558  --
559  -- Spalten hinzufügen
560  --
561  ALTER TABLE c_projects ADD (
562    is_ffg_sponsored NUMBER(1) NOT NULL,
563    is_fwf_sponsored NUMBER(1) NOT NULL,
564    is_eu_sponsored NUMBER(1) NOT NULL
565  );
566  --
567  -- Spalten hinzufügen
568  --
569  ALTER TABLE c_projects ADD (
570    is_ffg_sponsored NUMBER(1),
571    is_fwf_sponsored NUMBER(1),
572    is_eu_sponsored NUMBER(1)
573  );
574  --
575  -- Spalten hinzufügen
576  --
577  ALTER TABLE c_projects ADD (
578    is_ffg_sponsored NUMBER(1) NOT NULL,
579    is_fwf_sponsored NUMBER(1) NOT NULL,
580    is_eu_sponsored NUMBER(1) NOT NULL
581  );
582  --
583  -- Spalten hinzufügen
584  --
585  ALTER TABLE c_projects ADD (
586    is_ffg_sponsored NUMBER(1),
587    is_fwf_sponsored NUMBER(1),
588    is_eu_sponsored NUMBER(1)
589  );
590  --
591  -- Spalten hinzufügen
592  --
593  ALTER TABLE c_projects ADD (
594    is_ffg_sponsored NUMBER(1) NOT NULL,
595    is_fwf_sponsored NUMBER(1) NOT NULL,
596    is_eu_sponsored NUMBER(1) NOT NULL
597  );
598  --
599  -- Spalten hinzufügen
600  --
601  ALTER TABLE c_projects ADD (
602    is_ffg_sponsored NUMBER(1),
603    is_fwf_sponsored NUMBER(1),
604    is_eu_sponsored NUMBER(1)
605  );
606  --
607  -- Spalten hinzufügen
608  --
609  ALTER TABLE c_projects ADD (
610    is_ffg_sponsored NUMBER(1) NOT NULL,
611    is_fwf_sponsored NUMBER(1) NOT NULL,
612    is_eu_sponsored NUMBER(1) NOT NULL
613  );
614  --
615  -- Spalten hinzufügen
616  --
617  ALTER TABLE c_projects ADD (
618    is_ffg_sponsored NUMBER(1),
619    is_fwf_sponsored NUMBER(1),
620    is_eu_sponsored NUMBER(1)
621  );
622  --
623  -- Spalten hinzufügen
624  --
625  ALTER TABLE c_projects ADD (
626    is_ffg_sponsored NUMBER(1) NOT NULL,
627    is_fwf_sponsored NUMBER(1) NOT NULL,
628    is_eu_sponsored NUMBER(1) NOT NULL
629  );
630  --
631  -- Spalten hinzufügen
632  --
633  ALTER TABLE c_projects ADD (
634    is_ffg_sponsored NUMBER(1),
635    is_fwf_sponsored NUMBER(1),
636    is_eu_sponsored NUMBER(1)
637  );
638  --
639  -- Spalten hinzufügen
640  --
641  ALTER TABLE c_projects ADD (
642    is_ffg_sponsored NUMBER(1) NOT NULL,
643    is_fwf_sponsored NUMBER(1) NOT NULL,
644    is_eu_sponsored NUMBER(1) NOT NULL
645  );
646  --
647  -- Spalten hinzufügen
648  --
649  ALTER TABLE c_projects ADD (
650    is_ffg_sponsored NUMBER(1),
651    is_fwf_sponsored NUMBER(1),
652    is_eu_sponsored NUMBER(1)
653  );
654  --
655  -- Spalten hinzufügen
656  --
657  ALTER TABLE c_projects ADD (
658    is_ffg_sponsored NUMBER(1) NOT NULL,
659    is_fwf_sponsored NUMBER(1) NOT NULL,
660    is_eu_sponsored NUMBER(1) NOT NULL
661  );
662  --
663  -- Spalten hinzufügen
664  --
665  ALTER TABLE c_projects ADD (
666    is_ffg_sponsored NUMBER(1),
667    is_fwf_sponsored NUMBER(1),
668    is_eu_sponsored NUMBER(1)
669  );
670  --
671  -- Spalten hinzufügen
672  --
673  ALTER TABLE c_projects ADD (
674    is_ffg_sponsored NUMBER(1) NOT NULL,
675    is_fwf_sponsored NUMBER(1) NOT NULL,
676    is_eu_sponsored NUMBER(1) NOT NULL
677  );
678  --
679  -- Spalten hinzufügen
680  --
681  ALTER TABLE c_projects ADD (
682    is_ffg_sponsored NUMBER(1),
683    is_fwf_sponsored NUMBER(1),
684    is_eu_sponsored NUMBER(1)
685  );
686  --
687  -- Spalten hinzufügen
688  --
689  ALTER TABLE c_projects ADD (
690    is_ffg_sponsored NUMBER(1) NOT NULL,
691    is_fwf_sponsored NUMBER(1) NOT NULL,
692    is_eu_sponsored NUMBER(1) NOT NULL
693  );
694  --
695  -- Spalten hinzufügen
696  --
697  ALTER TABLE c_projects ADD (
698    is_ffg_sponsored NUMBER(1),
699    is_fwf_sponsored NUMBER(1),
700    is_eu_sponsored NUMBER(1)
701  );
702  --
703  -- Spalten hinzufügen
704  --
705  ALTER TABLE c_projects ADD (
706    is_ffg_sponsored NUMBER(1) NOT NULL,
707    is_fwf_sponsored NUMBER(1) NOT NULL,
708    is_eu_sponsored NUMBER(1) NOT NULL
709  );
710  --
711  -- Spalten hinzufügen
712  --
713  ALTER TABLE c_projects ADD (
714    is_ffg_sponsored NUMBER(1),
715    is_fwf_sponsored NUMBER(1),
716    is_eu_sponsored NUMBER(1)
717  );
718  --
719  -- Spalten hinzufügen
720  --
721  ALTER TABLE c_projects ADD (
722    is_ffg_sponsored NUMBER(1) NOT NULL,
723    is_fwf_sponsored NUMBER(1) NOT NULL,
724    is_eu_sponsored NUMBER(1) NOT NULL
725  );
726  --
727  -- Spalten hinzufügen
728  --
729  ALTER TABLE c_projects ADD (
730    is_ffg_sponsored NUMBER(1),
731    is_fwf_sponsored NUMBER(1),
732    is_eu_sponsored NUMBER(1)
733  );
734  --
735  -- Spalten hinzufügen
736  --
737  ALTER TABLE c_projects ADD (
738    is_ffg_sponsored NUMBER(1) NOT NULL,
739    is_fwf_sponsored NUMBER(1) NOT NULL,
740    is_eu_sponsored NUMBER(1) NOT NULL
741  );
742  --
743  -- Spalten hinzufügen
744  --
745  ALTER TABLE c_projects ADD (
746    is_ffg_sponsored NUMBER(1),
747    is_fwf_sponsored NUMBER(1),
748    is_eu_sponsored NUMBER(1)
749  );
750  --
751  -- Spalten hinzufügen
752  --
753  ALTER TABLE c_projects ADD (
754    is_ffg_sponsored NUMBER(1) NOT NULL,
755    is_fwf_sponsored NUMBER(1) NOT NULL,
756    is_eu_sponsored NUMBER(1) NOT NULL
757  );
758  --
759  -- Spalten hinzufügen
760  --
761  ALTER TABLE c_projects ADD (
762    is_ffg_sponsored NUMBER(1),
763    is_fwf_sponsored NUMBER(1),
764    is_eu_sponsored NUMBER(1)
765  );
766  --
767  -- Spalten hinzufügen
768  --
769  ALTER TABLE c_projects ADD (
770    is_ffg_sponsored NUMBER(1) NOT NULL,
771    is_fwf_sponsored NUMBER(1) NOT NULL,
772    is_eu_sponsored NUMBER(1) NOT NULL
773  );
774  --
775  -- Spalten hinzufügen
776  --
777  ALTER TABLE c_projects ADD (
778    is_ffg_sponsored NUMBER(1),
779    is_fwf_sponsored NUMBER(1),
780    is_eu_sponsored NUMBER(1)
781  );
782  --
783  -- Spalten hinzufügen
784  --
785  ALTER TABLE c_projects ADD (
786    is_ffg_sponsored NUMBER(1) NOT NULL,
787    is_fwf_sponsored NUMBER(1) NOT NULL,
788    is_eu_sponsored NUMBER(1) NOT NULL
789  );
790  --
791  -- Spalten hinzufügen
792  --
793  ALTER TABLE c_projects ADD (
794    is_ffg_sponsored NUMBER(1),
795    is_fwf_sponsored NUMBER(1),
796    is_eu_sponsored NUMBER(1)
797  );
798  --
799  -- Spalten hinzufügen
800  --
801  ALTER TABLE c_projects ADD (
802    is_ffg_sponsored NUMBER(1) NOT NULL,
803    is_fwf_sponsored NUMBER(1) NOT NULL,
804    is_eu_sponsored NUMBER(1) NOT NULL
805  );
806  --
807  -- Spalten hinzufügen
808  --
809  ALTER TABLE c_projects ADD (
810    is_ffg_sponsored NUMBER(1),
811    is_fwf_sponsored NUMBER(1),
812    is_eu_sponsored NUMBER(1)
813  );
814  --
815  -- Spalten hinzufügen
816  --
817  ALTER TABLE c_projects ADD (
818    is_ffg_sponsored NUMBER(1) NOT NULL,
819    is_fwf_sponsored NUMBER(1) NOT NULL,
820    is_eu_sponsored NUMBER(1) NOT NULL
821  );
822  --
823  -- Spalten hinzufügen
824  --
825  ALTER TABLE c_projects ADD (
826    is_ffg_sponsored NUMBER(1),
827    is_fwf_sponsored NUMBER(1),
828    is_eu_sponsored NUMBER(1)
829  );
830  --
831  -- Spalten hinzufügen
832  --
833  ALTER TABLE c_projects ADD (
834    is_ffg_sponsored NUMBER(1) NOT NULL,
835    is_fwf_sponsored NUMBER(1) NOT NULL,
836    is_eu_sponsored NUMBER(1) NOT NULL
837  );
838  --
839  -- Spalten hinzufügen
840  --
841  ALTER TABLE c_projects ADD (
842    is_ffg_sponsored NUMBER(1),
843    is_fwf_sponsored NUMBER(1),
844    is_eu_sponsored NUMBER(1)
845  );
846  --
847  -- Spalten hinzufügen
848  --
849  ALTER TABLE c_projects ADD (
850    is_ffg_sponsored NUMBER(1) NOT NULL,
851    is_fwf_sponsored NUMBER(1) NOT NULL,
852    is_eu_sponsored NUMBER(1) NOT NULL
853  );
854  --
855  -- Spalten hinzufügen
856  --
857  ALTER TABLE c_projects ADD (
858    is_ffg_sponsored NUMBER(1),
859    is_fwf_sponsored NUMBER(1),
860    is_eu_sponsored NUMBER(1)
861  );
862  --
863  -- Spalten hinzufügen
864  --
865  ALTER TABLE c_projects ADD (
866    is_ffg_sponsored NUMBER(1) NOT NULL,
867    is_fwf_sponsored NUMBER(1) NOT NULL,
868    is_eu_sponsored NUMBER(1) NOT NULL
869  );
870  --
871  -- Spalten hinzufügen
872  --
873  ALTER TABLE c_projects ADD (
874    is_ffg_sponsored NUMBER(1),
875    is_fwf_sponsored NUMBER(1),
876    is_eu_sponsored NUMBER(1)
877  );
878  --
879  -- Spalten hinzufügen
880  --
881  ALTER TABLE c_projects ADD (
882    is_ffg_sponsored NUMBER(1) NOT NULL,
883    is_fwf_sponsored NUMBER(1) NOT NULL,
884    is_eu_sponsored NUMBER(1) NOT NULL
885  );
886  --
887  -- Spalten hinzufügen
888  --
889  ALTER TABLE c_projects ADD (
890    is_ffg_sponsored NUMBER(1),
891    is_fwf_sponsored NUMBER(1),
892    is_eu_sponsored NUMBER(1)
893  );
894  --
895  -- Spalten hinzufügen
896  --
897  ALTER TABLE c_projects ADD (
898    is_ffg_sponsored NUMBER(1) NOT NULL,
899    is_fwf_sponsored NUMBER(1) NOT NULL,
900    is_eu_sponsored NUMBER(1) NOT NULL
901  );
902  --
903  -- Spalten hinzufügen
904  --
905  ALTER TABLE c_projects ADD (
906    is_ffg_sponsored NUMBER(1),
907    is_fwf_sponsored NUMBER(1),
908    is_eu_sponsored NUMBER(1)
909  );
910  --
911  -- Spalten hinzufügen
912  --
913  ALTER TABLE c_projects ADD (
914    is_ffg_sponsored NUMBER(1) NOT NULL,
915    is_fwf_sponsored NUMBER(1) NOT NULL,
916    is_eu_sponsored NUMBER(1) NOT NULL
917  );
918  --
919  -- Spalten hinzufügen
920  --
921  ALTER TABLE c_projects ADD (
922    is_ffg_sponsored NUMBER(1),
923    is_fwf_sponsored NUMBER(1),
924    is_eu_sponsored NUMBER(1)
925  );
926  --
927  -- Spalten hinzufügen
928  --
929  ALTER TABLE c_projects ADD (
930    is_ffg_sponsored NUMBER(1) NOT NULL,
931    is_fwf_sponsored NUMBER(1) NOT NULL,
932    is_eu_sponsored NUMBER(1) NOT NULL
933  );
934  --
935  -- Spalten hinzufügen
936  --
937  ALTER TABLE c_projects ADD (
938    is_ffg_sponsored NUMBER(1),
939    is_fwf_sponsored NUMBER(1),
940    is_eu_sponsored NUMBER(1)
941  );
942  --
943  -- Spalten hinzufügen
944  --
945  ALTER TABLE c_projects ADD (
946    is_ffg_sponsored NUMBER(1) NOT NULL,
947    is_fwf_sponsored NUMBER(1) NOT NULL,
948    is_eu_sponsored NUMBER(1) NOT NULL
949  );
950  --
951  -- Spalten hinzufügen
952  --
953  ALTER TABLE c_projects ADD (
954    is_ffg_sponsored NUMBER(1),
955    is_fwf_sponsored NUMBER(1),
956    is_eu_sponsored NUMBER(1)
957  );
958  --
959  -- Spalten hinzufügen
960  --
961  ALTER TABLE c_projects ADD (
962    is_ffg_sponsored NUMBER(1) NOT NULL,
963    is_fwf_sponsored NUMBER(1) NOT NULL,
964    is_eu_sponsored NUMBER(1) NOT NULL
965  );
966  --
967  -- Spalten hinzufügen
968  --
969  ALTER TABLE c_projects ADD (
970    is_ffg_sponsored NUMBER(1),
971    is_fwf_sponsored NUMBER(1),
972    is_eu_sponsored NUMBER(1)
973  );
974  --
975  -- Spalten hinzufügen
976  --
977  ALTER TABLE c_projects ADD (
978    is_ffg_sponsored NUMBER(1) NOT NULL,
979    is_fwf_sponsored NUMBER(1) NOT NULL,
980    is_eu_sponsored NUMBER(1) NOT NULL
981  );
982  --
983  -- Spalten hinzufügen
984  --
985  ALTER TABLE c_projects ADD (
986    is_ffg_sponsored NUMBER(1),
987    is_fwf_sponsored NUMBER(1),
988    is_eu_sponsored NUMBER(1)
989  );
990  --
991  -- Spalten hinzufügen
992  --
993  ALTER TABLE c_projects ADD (
994    is_ffg_sponsored NUMBER(1) NOT NULL,
995    is_fwf_sponsored NUMBER(1) NOT NULL,
996
```

| Constraint | Beschreibung                                                                                                                                     | Integritätsbedingung     |
|------------|--------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------|
| not null   | Der not null Constraint fordert, dass für eine bestimmte Spalte zwingend ein Wert eingegeben werden muss.                                        | Logische Konsistenz      |
| unique     | Der unique Constraint fordert, dass alle für eine Spalte eingegebenen Werte eindeutig sind.                                                      | Logische Konsistenz      |
| check      | Der check Constraint ermöglicht die Formulierung spezifische Einschränkungen für die Werte einer Spalte.                                         | Logische Konsistenz      |
| key        | Durch die Definition von Schlüsseln kann für die Daten einer Datenbank Enthüllungsintegrität und Referentielle Konsistenz sichergestellt werden. | Referentielle Konsistenz |

Abbildung 31. Constrainttypen

**7.3.3 Verwalten von Constraints**

Zur Verwaltung von Constraints definiert die SQL Spezifikation den alter table Befehl.

## » DDL: Verwalten von Constraints »

```

1 -- Verwalten von Constraints
2
3 -- Constraint zu einer Tabelle hinzufügen
4 ALTER TABLE projects
5 ADD CONSTRAINT uq_pro_code
6   UNIQUE (project_code);
7
8 -- Constraint deaktivieren
9 ALTER TABLE projects DISABLE uq_pro_code;
10
11 -- Constraint aktivieren
12 ALTER TABLE projects ENABLE uq_pro_code;
13
14 -- Constraint umbenennen
15 ALTER TABLE projects RENAME CONSTRAINT
16 up_pro_code TO up_projects_code;
17
18 -- Constraint löschen
19 ALTER TABLE projects DROP CONSTRAINT
20 up_projects_code;
  
```

**7.3.4 Constraintmonitoring**

Das Constrainterfakt kann zur Laufzeit des Datenbankvers unterschiedliche Zustände haben.

## » Query: Constraintmonitoring »

```

1 -- Constraintmonitoring
2
3 -- Die user_cons_columns bzw. user_constraints
4 Tabellen werden verwendet um den Zustand von
5 Constraints zu verwalten.
6
7 -- Die user_cons_columns Tabelle ermöglicht
8 -- eine Einricht auf die Zugriffrechte
9 -- fuer Constraints
10
11 SELECT constraint_name,
12   table_name,
13   column_name,
14   owner
15 FROM user_cons_columns
16 WHERE table_name = 'projects';
17
18 -- Die user_constraints Tabelle speichert
19 -- fuer jedes Constraint seine Zugehörig-
20 -- keit, Namen und Status
21
22 SELECT constraint_name,
23   constraint_type,
24   last_change,
25   status
26 FROM user_constraints
27 WHERE table_name = 'projects';
  
```

77

## Informationssysteme

**7.4. Datenbankerfakt: View**

Mit einer View kann eine SQL Abfrage als Objekt in einer Datenbank gespeichert werden.

**7.4.1 View**

## View \*

Eine View ist eine SQL Abfrage, die in der Datenbank als Objekt gespeichert wird.

Die Abfrage selbst wird dabei in Form einer Zeichenkette hinterlegt. Erst beim Aufruf der View wird die mit der View assoziierte Abfrage ausgeführt.

Views spielen eine zentrale Rolle in der Entwicklung relationaler Datenbanken.

## » Auflistung: Einsatz von Views \*

Komplexitätsreduktion: Mit Views kann der Zugriff auf den Datenbestand einer Datenbank wesentlich vereinfacht werden.

Normalisierte Datenbankschemata abstrahieren ein komplexes Netzwerk von Tabellen und Relationen. SQL Abfragen in solchen Strukturen sind komplex und umfangreich. Views werden verwendet um die Komplexität von Abfragen für den User zu abstrahieren.

Datenschutz: Views können helfen die Dateneinstruktur einer Datenbank vor unbefugten Usern zu maskieren.

In der Regel ist es nicht erwünscht, die Struktur einer Datenbank nach außen hin zu publizieren. Views helfen die Details eines Schemas von Usern zu verstecken. Durch die Vergabe von Rollen und Rechten kann der Zugriff auf die einzelnen Datenbankobjekte einfach gesteuert werden.

Kompilierbarkeit: Beim Speichern der View, wird die mit der View assoziierte Abfrage kompiliert und auf syntaktische Richtigkeit geprüft. Fehlerhafte SQL Abfragen müssen vor dem Speichern korrigiert werden.

Datenbankschnittstelle: Durch die Verwendung von Views kann auf einfache Weise eine einheitliche Datenbankschnittstelle entworfen werden.

**7.4.2 DDL Befehl: create view**

Zum Anlegen von Views definiert die SQL Spezifikation den create view Befehl.

## » Erklärung: Restriktionen einer View \*

- Eine mit einer View assoziierte SQL Abfrage unterliegt in ihrer Form bestimmten Einschränkungen.

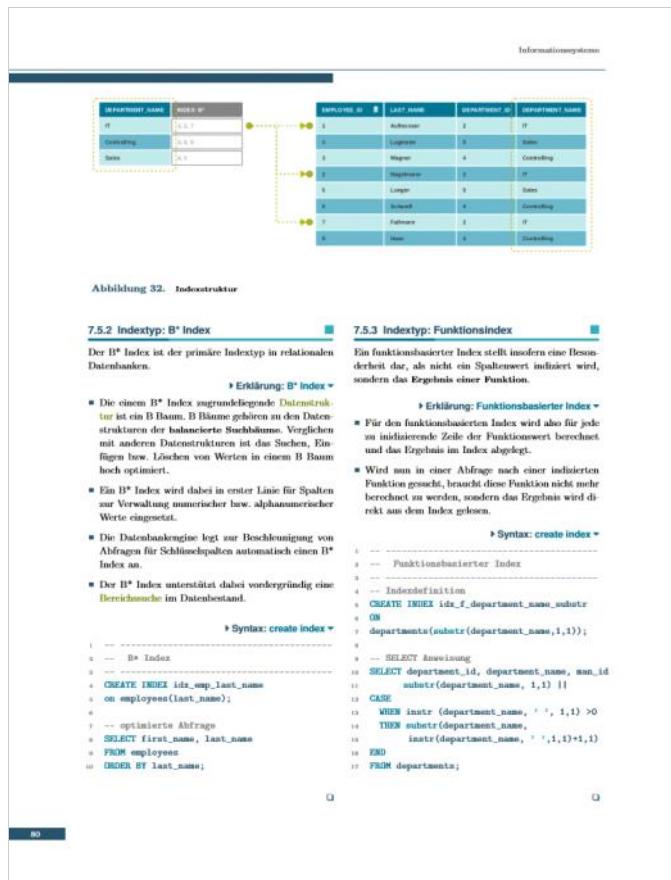
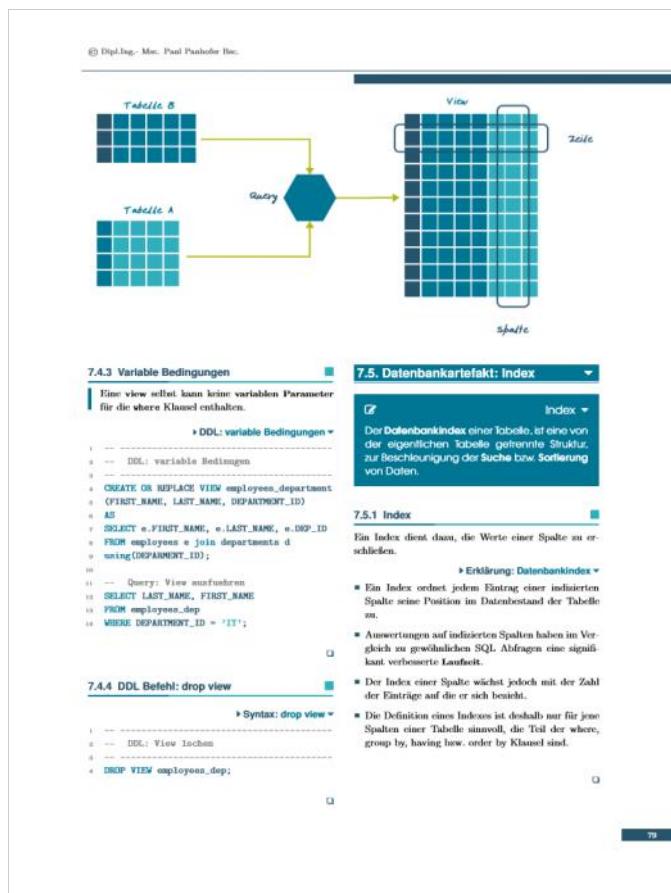
- Die order by Klausur einer View wird ignoriert, wenn in der SQL Abfrage selbst eine Sortierung definiert wird.

## » Syntax: create view \*

```

1 -- DDL: View anlegen
2
3 -- Syntax: create view
4
5 CREATE OR REPLACE VIEW view_name
6 [(spalten_name)]
7 AS <Select Ausdruck>;
8
9
10
11 CREATE OR REPLACE VIEW employees_dep
12 (FIRST_NAME, LAST_NAME, DEPARTMENT_NAME)
13 AS
14 SELECT e.FIRST_NAME,
15   e.LAST_NAME,
16   d.DEPARTMENT_NAME
17 FROM employees e join departments d
18 using(DEPARTMENT_ID);
19
20
21 CREATE OR REPLACE VIEW mvp_employees
22 AS
23 SELECT e.FIRST_NAME,
24   e.LAST_NAME,
25   e.DEPARTMENT_ID
26 FROM employees e
27 JOIN (
28   SELECT max(SALARY) MAX_SALARY,
29         DEPARTMENT_ID
30   FROM employees
31   GROUP BY DEPARTMENT_ID) sub
32 ON e.SALARY = sub.MAX_SALARY AND
33   e.DEPARTMENT_ID = sub.DEPARTMENT_ID;
34
35
36 -- Query: View auszuhören
37
38 SELECT first_name, last_name
39 FROM employees_dep;
  
```

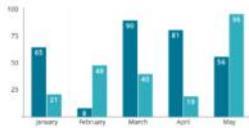
78



| Indextyp       | Beschreibung                                                                                                          | SpaltenTypen          |
|----------------|-----------------------------------------------------------------------------------------------------------------------|-----------------------|
| B* Index       | Datenbankindex zum Indexieren von Spalten mit alphanumerischen bzw. numerischen Werten, z.B. Nachnamen bzw. Ortsnamen | varchar, number       |
| Unique Index   | Datenbankindex zum Indexieren von Spalten mit einem unique Constraint, z.B. Spalten mit einem Primary Key Constraint. | unique Constraint     |
| Funktionsindex | Datenbankindex zum Indexieren von Spalten deren Werte durch Funktionen transformiert werden, z.B.: lower(trim_name)   |                       |
| Volltextindex  | Datenbankindex zum Indexieren von Spalten die umfangreiche Texte speichern.                                           | clob, word, pdf, html |

Abbildung 33. Indextypen

**7.5.4 Indextyp: Unique Index**  
Für Datenbankspalten mit einem unique Constraint, wird durch die Datenbankengine automatisch ein unique Index angelegt.



- Ein unique Index kann auch über mehrere Spalten hinweg definiert werden. In diesem Fall müssen die kombinierten Werte der Spalten eindeutig sein.
- In der Regel wird dieser Index Typ durch die Datenbankengine angelegt.

**Syntax: create index \***

```

1 -- -----
2 -- unique Index
3 -----
4 CREATE UNIQUE INDEX
5 idx_department_name
6 ON departments(department_name);
7
8 CREATE UNIQUE INDEX
9 idx_project_description
10 ON projects(title);

```

**7.5.5 Indextyp: Volltextindex**  
Durch die Verwendung eines Volltextindex können Volltextsuchen in relationalen Datenbanken signifikant beschleunigt werden.

Die Volltextsuche wurde entwickelt um Word, Pdf bzw. Italien Dokumente, die in der Datenbank gespeichert werden, zu durchsuchen.

**Syntax: create index \***

```

1 -- -----
2 -- Volltextindizierung: Gantek
3 -----
4 CREATE TABLE c.projects (
5   project_id NUMBER(19,0) NOT NULL,
6   manager_id NUMBER(19,0) NOT NULL,
7   parent_id NUMBER(19,0) NOT NULL,
8   is_fwf_funded NUMBER(1) NOT NULL,
9   is_ffg_funded NUMBER(1) NOT NULL,
10  description VARCHAR(4000),
11  title VARCHAR(256) NOT NULL UNIQUE,
12  PRIMARY KEY (project_id)
13 )
14
15 CREATE INDEX idx_full_text_ctx
16 ON
17 table_name(text)
18 INDEXTYPE IS ctxsys.context;
19
20 CREATE INDEX pro_des_index
21 ON
22 <_projects(description)
23 INDEXTYPE IS ctxsys.context;

```

| Informationssysteme |                                                                                               |
|---------------------|-----------------------------------------------------------------------------------------------|
| <b>7.6. Sequenz</b> |                                                                                               |
| <b>Sequenz *</b>    | Eine Sequenz ist ein Datenbankobjekt, mit dem eindeutige fortlaufende Werte generiert werden. |

**7.6.1 Anlegen von Sequenzen**  
Sequenzen werden zur Generierung von Primärschlüsseln verwendet.

**Syntax: create sequence \***

```

1 -- -----
2 -- Syntax: create sequence
3 -----
4 CREATE SEQUENCE Anweisung >=
5   CREATE SEQUENCE Sequencename
6   [INCREMENT BY integer]
7   [START WITH integer]
8   [MAXVALUE integer | NOMAXVALUE]
9   [MINVALUE integer | NOMINVALUE]
10  [CYCLE | NOCYCLE]
11  [CACHE | NOCACHE]
12  [ORDER | NOORDER]
13
14 INCREMENT BY int
15 -- bestimmt die ganze Zahl, um die eine
16 -- SEQUENZ erhöht wird
17
18 START WITH int
19 -- bestimmt die ganze Zahl, mit der die
20 -- SEQUENZ
21 -- startet
22
23 MAXVALUE int
24 -- obere Grenze der SEQUENZ
25
26 NOMAXVALUE
27 -- SEQUENZ hat keine obere Grenze
28
29 MINVALUE int
30 -- untere Grenze der SEQUENZ bei absteigenden
31 -- SEQUENZEN. Der DEFAULT für MINVALUE ist 1.
32 -- wenn MINVALUE nicht angegeben ist
33
34 NOMINVALUE
35 -- SEQUENZ hat keine untere Grenze
36
37 CYCLE|NOCYCLE

```

**7.6.2 Sequenzen verwenden**  
Für das Arbeiten mit Sequenzen verwenden wir die Attribute curval und nextval.

**DDL: Sequenzen verwenden \***

```

1 -- -----
2 -- Sequenzen verwenden
3 --
4
5 INSERT INTO projects values
6 (project_seq.NEXTVAL, 'Simulation');
7
8 -- Gegenwärtigen Wert der Sequenz abfragen
9 project_seq.CURVAL
10
11 -- Nächsten Wert der Sequenz abfragen
12 project_seq.NEXTVAL

```

## 8. SQL - Data Manipulation Language

# 07

DML - Data Manipulation Language

|                    |    |
|--------------------|----|
| 01. DML Befehlsatz | 84 |
| 02. Insert Befehl  | 85 |
| 03. Update Befehl  | 85 |

### 8.1. DML - Befehlsatz

Data Manipulation Language ▾  
Die Data Manipulation Language definiert Befehle zur Verarbeitung von Daten.

**8.1.1 DML Befehlsatz** ▾  
Der DML Befehlsatz wird zur Verarbeitung von Daten in der Datenbank verwendet.

➤ Auflistung: DML Befehlsatz ▾

insert command ▾  
Die Insert Anweisung ist ein Befehl zum Einfügen von Daten in Datenbanktabellen.

Der Insert Befehl besitzt mehrere Formen.

update command ▾  
Die update Anweisung ist ein Befehl zum Ändern von Daten.

Der update Befehle besitzt mehrere Formen.

merge command ▾  
Die merge Anweisung ist ein Befehl zum Verarbeiten von Daten. Die Anweisung ist im Kern eine Kombination eines Insert und update Befehls.

delete command ▾  
Die delete Anweisung ist ein Befehl zum Löschen von Daten.





Abbildung 34. DML Befehlssets

**8.2. Insert Befehl**

Die insert Anweisung ist ein Befehl zum Einfügen von Daten in die Datenbank. Die SQL Spezifikation unterscheidet mehrere Formen des insert Befehls.

**8.2.1 insert Befehl**

Befehl zum Einfügen von Datensätzen in die Datenbank.

## » DML: insert command

```

1 -- -----
2 -- Syntax: insert
3 -- -----
4 -- Syntax 1
5 INSERT INTO table_name (column1, column2, )
6 VALUES (value1, value2, value3, ...);
7
8 -- Syntax 2
9 INSERT INTO table_name
10 VALUES (value1, value2, value3, ...);
11
12 -- -----
13 -- Beispiel: insert
14 -----
15 INSERT INTO groups (group_id, name) VALUES
16 (1, '1 ARIT - 2015');
17
18 INSERT INTO subjects VALUES
19 (1, 'Systemtechnik - Informationssysteme');

```

**8.2.2 Insert - select**

Der insert - select Befehl stellt eine Variante des ursprünglichen insert Befehls dar.

Zur Migration von Daten ist es möglich den insert Befehl mit der select Anweisung zu koppeln.

## » DML: insert - select command

```

1 -- -----
2 -- Syntax: insert select
3 -- -----
4 INSERT INTO table_name VALUES
5 (column1, column2)
6 SELECT value1, value2
7 FROM table_name;
8
9 -- -----
10 -- Beispiel: insert - select
11 -----
12 INSERT INTO c_projects
13 VALUES (
14 project_description,
15 project_title,
16 project_id,
17 is_rvf_sponsored,
18 is_ffg_sponsored
19 )
20
21 SELECT id, title, description, fwf
22 FROM l_projects
23 WHERE project_type = 'REQUEST_FUNDING';

```

85

**8.3. update Befehl**

Die update Anweisung ist ein Befehl zum Ändern von Daten.

**8.3.1 update syntax**

Die SQL Spezifikation definiert mehrere Formen des update Befehls.

## » DML: update Syntax

```

1 -- -----
2 -- Syntax: update
3 -- -----
4 -- Ändern aller Werte einer Tabelle
5 UPDATE table_name
6 SET column1 = value1, column2 = value2, ...
7
8 -- Ändern bestimmte Werte in der Tabelle
9 UPDATE table_name
10 SET column1 = value1, column2 = value2, ...
11 WHERE predicate;

```

## » DML: Fallbeispiel Schule

```

1 -- -----
2 -- Beispiel: update
3 --
4 UPDATE students SET last_name = 'Lang'
5 WHERE student_id = 1;
6
7 UPDATE students
8 SET last_name = 'Dreger',
9 first_name = 'Nikolaus'
10 WHERE student_id = 3
11
12 UPDATE employees
13 SET salary = (
14 SELECT avg(salary) FROM employees
15 )
16 WHERE department_id = 103;
17
18 UPDATE employees
19 SET salary = (
20 SELECT max(salary) FROM employees
21 )
22 WHERE department_id IN (
23 SELECT
24 )

```

86

Informationssysteme

## 9. SQL - Data Control Language

# 08

DCL- Data Control Language

|                    |    |
|--------------------|----|
| 01. DCL Befehlsatz | 88 |
| 02. create User    | 88 |
| 03. grant          | 89 |

### 9.1. DCL Befehle

Data Control Language ▾  
Die DCL definiert Befehle zum Vergeben und Entziehen von Berechtigungen.

DCL ist die Datenüberwachungssprache relationaler Datenbanken.

#### 9.1.1 DCL Befehlsatz

Der DCL Befehlsatz definiert Befehle zur Verwaltung von Benutzerrechten für die Datenbank.

Auflistung: DCL Befehlsatz ▾

- create user ▾  
Befehlsguppe zum Anlegen von Benutzern.
- grant ▾  
Befehlsguppe zum Zuordnen von Benutzerrechten.
- revoke ▾  
Befehlsguppe zum Zurücknehmen von Benutzerrechten

### 9.2. create user Befehl

Befehlsguppe zum Anlegen und Verwalten von Benutzern.

#### 9.2.1 Benutzerverwaltung

Die Benutzerverwaltung ist der Steuerungsprozess, welcher Benutzer sich mit der Datenbank verbinden dürfen und welche Rechte sie für die Datenbank haben.

Für ein Unternehmen wäre es schwer ein Sicherheitssystem für seine Anwendungen zu implementieren, wenn die Datenbank keine Möglichkeit für die Durchsetzung von Benutzerrechten bieten würde.

88

CREATE USER 'htl-administrator'@'localhost' IDENTIFIED BY 'htl-pa33w09d'

create user Befehl

Abbildung 35. create user Befehl

**9.2.2 create User Befehl**

Der create user Befehl besteht aus mehreren Segmenten.

## » Erklärung: Segmente create user \*

- Schlüsselwörter: Die Schlüsselwortsegmente **create** und **identified by** trennen das Benutzernamensegment vom Passwortsegment.
- Benutzernamen: Das Benutzernamensegment besteht aus 2 Teilen: Dem Namen des Benutzers und der IP-Adresse des Hosts von dem er sich verbindet.
- Passwort: Das Passwortsegment dient zur Angabe des Passworts des Benutzers.

## » DCL: Benutzerverwaltung \*

```

1 -- Syntax: create User
2
3
4 -- Anlegen des users htl-admin
5 create user 'htl-admin'@'127.0.0.1'
6 identified by 'htl-12345';
7
8 create user 'htl-admin'@'%';
9 identified by 'htl123';
10
11 -- Benutzer anzeigen
12 select * from mysql.user;
13
14
15 -- Syntax: mysladmin
16
17 -- Ändern des Passworts des Admins
18 mysladmin --user=root
19 --password=old
20 password new
21
22
23 -- Syntax: drop user
24
25 drop user user_name;
```

**9.3. grant Befehl**

Befehlsguppe zum Zuordnen von Benutzerrechten.

## » 9.3.1 Privilegien zuordnen

Damit ein Benutzer bestimmte Aktionen in der Datenbank ausführen kann werden ihm Privilegien zugewiesen.

## » Auflistung: Privilegien \*

- **create:** Erlaubt einem Benutzer, neue Tabellen zu erstellen.
- **delete:** Erlaubt einen Benutzer, Zeilen in einer Tabelle zu löschen.
- **insert:** Erlaubt einen Benutzer, neue Zeilen in eine Tabelle zu schreiben.
- **select:** Loscherechtigungen auf eine Datenbank oder Tabelle.
- **update:** Erlaubt eine Zeile zu aktualisieren.
- **all privileges:** Ein Wildcard für alle Rechte auf das gewählte Datenbankobjekt.

**9.3.2 Syntax: grant Befehl**

## » DCL: Rechteverwaltung \*

```

1 -- Syntax: grant
2
3
4 -- Anatomie des grant Befehls
5 GRANT privilege ON table_name TO user@host
```

**9.3.3 grant Befehl**

## » DCL: Rechteverwaltung \*

```

1 -- Syntax: grant
2
3
4 -- Tabellen fuer Datenbank erstellen
5 GRANT create ON projects.* to
6 'htl-admin'@'localhost';
7
8 -- Tabellen auslesen
9 GRANT select ON projects.subprojects to
10 'htl-admin'@'localhost';
11
12 -- Alle Rechte fuer eine Datenbank zuordnen
13 GRANT all privileges ON projects.* to
14 'htl-admin'@'localhost';
15
16
17 -- Rechte anzeigen
18
19 SHOW GRANTS FOR 'htl-admin'@'localhost';
20
21
22 -- Rechte zuordnen
23
24 FLUSH PRIVILEGES;
25
26
27 -- Rechte zuruecknehmen
28
29
30 REVOKE select ON projects.subprojects FROM
31 'htl-admin'@'localhost';
```

# Programmierung relationaler Datenbanken

August 19, 2020

## 10. PL/SQL - Grundlagen

# 01

PL/SQL Grundlagen

|                |    |
|----------------|----|
| 01. Grundlagen | 92 |
| 02. Block      | 96 |

### 10.1. PL/SQL Grundlagen

PL/SQL als Programmiersprache stellt eine prozedurale Erweiterung der SQL Sprachspezifikation dar.

#### 10.1.1 PL/SQL

PL/SQL kombiniert den SQL Befehlsatz mit Strukturen zur Entscheidungsfindung und Schleifenstrukturen.

PL/SQL Programme haben direkten Zugriff auf die Strukturen und Daten einer Datenbank.



Erklärung: [PL/SQL Grundlagen](#)

- Durch die Integration des SQL Befehlsatz in der PL/SQL Sprachspezifikation haben PL/SQL Programme direkten Zugriff auf die Daten und Strukturen einer Datenbank.
- Damit wird es möglich Teile der Anwendungslogik vom Applikationsserver in den DatenbanksERVER zu verlagern.
- Die Verlagerung der Anwendungslogik in den DatenbanksERVER resultiert dabei in einer signifikanten Steigerung der Verarbeitungsgeschwindigkeit datenzentrierter Anwendungen.
- Der Datenbankszugriff erfolgt in PL/SQL Programme dabei gespeist über die PL/SQL Sprachschichtstelle in Form von [Modulen](#).
- Logisch zusammenhängende Routinen werden dazu in [Funktionen](#) und [Prozeduren](#) zusammengefasst.
- PL/SQL Programme können direkt in einer Datenbank gespeichert und ausgeführt werden.
- Zur Ausführung von PL/SQL Programmen verwendet eine Datenbank ein eigenes Programm: die [PL/SQL Engine](#).
- Gegenwärtig können PL/SQL Programme nur in Oracle Datenbanken ausgeführt werden. Hersteller anderer Datenbanken unterstützen nur Teile des PL/SQL Standards.
- Neben dem direkten Datenzugriff unterstützt PL/SQL eigene Mechanismen zur Transaktionssteuerung, Cursor- und Fehlerbehandlung.



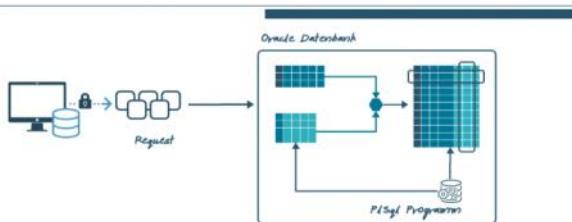


Abbildung 36. PL/SQL Ausführung

#### 10.1.2 PL/SQL Sprachspezifikation

PL/SQL als Programmiersprache wurde für den Einsatz in Datenbanken konzipiert.

##### Erklärung: PL/SQL Sprachspezifikation

- Der Datenbankangriff in PL/SQL Programme erfolgt gekapselt über die PL/SQL Sprachchnittstelle in Form von Prozeduren bzw. Funktionen.
- Declarative Anweisungen können in PL/SQL Programmen als Programmlemente in einer Datenbank gespeichert und ausgeführt werden. Der Datenbankangriff kann damit als freie Abfolge von Anweisungen und Routinen formuliert werden.
- Ein PL/SQL Programm kann im Grunde als strukturierte Hülle des SQL-Befehlsatzes abstrahiert werden. Prominentester Befehl eines PL/SQL Programms bleibt die select-Anweisung.
- Neben dem direkten Datenzugriff unterstützt PL/SQL eigene Mechanismen zur Transaktionssteuerung, Cursor- und Fehlerbehandlung.



#### 10.1.3 Einsatzgebiete von PL/SQL

PL/SQL Programme erfüllen in der Datenbankentwicklung ein weites Feld von Aufgaben.

##### Aufzählung: Einsatzgebiete von PL/SQL

- Datenbankfunktionalität**
  - Mit PL/SQL Programmen kann die SQL-Sprachspezifikation um zusätzliche Funktionen erweitert werden.
- Datenkonsistenz**
  - Die Sicherstellung der Konsistenz des Datenbestandes gehört zu den grundlegenden Aufgaben eines Informationssystems. PL/SQL abstrahiert die Integrationsprüfung dazu in Form von ereignisgesteuerten Routinen.
- Datensicherheit**
  - PL/SQL stellt dem Datenbankentwickler ein weites Feld von Autorisierungskonzepten zur Verfügung.

93

#### Informationssysteme

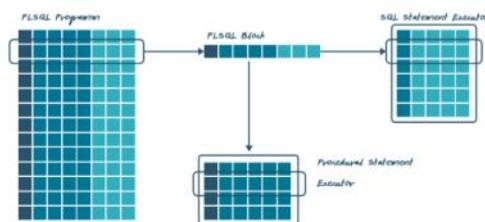


Abbildung 37. Kontextswitch pl/sql Engine

#### 10.1.4 PL/SQL Engine

PL/SQL Programme werden in Datenbanken in einem eigenen Prozess, der PL/SQL Engine ausgeführt.

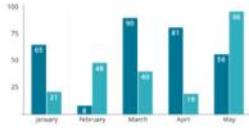
##### Erklärung: PL/SQL Engine

- Ein PL/SQL Programm stellt eine freie Abfolge von PL/SQL Anweisungen und SQL-Befehlen dar.
- PL/SQL Programme können im Grunde als strukturielle Hülle für SQL-Befehle verstanden werden.
- Im Gegensatz zu PL/SQL Blöcken, werden SQL-Befehl jedoch nicht von der PL/SQL Engine ausgeführt. Die Befehle werden zur Ausführung an einen eigenen Prozess, den SQL Statement Executor weitergeleitet.
- Der Aufruf des SQL Statement Executors wird als Kontextswitch bezeichnet. Ein Kontextswitch ist dabei mit einem gewissen Ressourceneinsatz verbunden.



94

- 10.1.5 PL/SQL Block**
- PL/SQL ist eine blockorientierte Programmiersprache.
  - Erläuterung: PL/SQL Block**
    - Das grundlegende Element eines PL/SQL Programms ist der PL/SQL Block.
    - PL/SQL Blöcke erfüllen in PL/SQL Programmen 2 Aufgaben:
      - Codestrukturierung: PL/SQL Blöcke beschreiben die Struktur von PL/SQL Programmen.
      - Programmausführung: PL/SQL Blöcke können von der PL/SQL Engine ausgeführt werden.



**Syntax: PL/SQL Block**

```

1 -- Syntax: Anonymer Block
2 -----
3 BEGIN
4   Executable Statements
5 ...
6 END;
7
8 -----
9 -- Codebeispiel: Block
10 -----
11 BEGIN
12   SELECT l.id, l.funding, l.title
13     l.description, l
14   BULK COLLECT INTO l_project_report
15   FROM HR.L_PROJECTS l
16   WHERE l.PROJECT_TYPE = 'STIPENDIUM';
17
18   FOR i IN 1..l.project_reports.COUNT
19   LOOP
20     DBMS_OUTPUT.PUT_LINE(
21       'project data '
22       || l_projects(i).TITLE
23       || l_projects(i).DESCRIPTION
24       || chr(13)
25       || l_projects(i).PROJECT_TYPE
26     );
27   END LOOP;
28
29 END;
30 
```

Codebeispiel: PL/SQL Programm

```

1 -- -----
2 -- Codebeispiel: plsql Programm
3 -----
4 DECLARE
5   -- DEFINITION: Tabellentyp
6   TYPE PROJECT_TABLE_TYPE IS TABLE OF
7     HR.L_PROJECTS%ROWTYPE;
8
9   -- DECLARATION: Virtuelle Tabelle
10  L_projects PROJECT_TABLE_TYPE := 
11    PROJECT_TABLE_TYPE();
12
13  -- DEFINITION: Strukturierter Datentyp
14  TYPE PROJECT_REPORT_TYPE IS RECORD (
15    TITLE HR.L_PROJECTS.TITLE%TYPE,
16    DESCRIPTION VARCHAR(4000),
17    PROJECT_ID NUMBER(10,2)
18  );
19
20  -- DEFINITION: Tabellentyp
21  TYPE PR_TABLE_TYPE IS TABLE OF
22    PROJECT_REPORT_TYPE;
23
24  -- DECLARATION: Virtuelle Tabelle
25  L_project_reports PR_TABLE_TYPE := 
26    PR_TABLE_TYPE();
27
28  BEGIN
29    SELECT l.id, l.funding, l.title
30      l.description, l
31    BULK COLLECT INTO l_project_report
32    FROM HR.L_PROJECTS l
33    WHERE l.PROJECT_TYPE = 'STIPENDIUM';
34
35    FOR i IN 1..l.project_reports.COUNT
36    LOOP
37      DBMS_OUTPUT.PUT_LINE(
38        'project data '
39        || l_projects(i).TITLE
40        || l_projects(i).DESCRIPTION
41        || chr(13)
42        || l_projects(i).PROJECT_TYPE
43      );
44    END LOOP;
45
46  END;
47 
```

95

## Informationssysteme



Abbildung 38. Aufbau eines plsql Blocks

- 10.2. PL/SQL Block**
- 10.2.1 PL/SQL Block**
- PL/SQL Block**
- PL/SQL Blöcke sind die grundlegenden Bausteine eines PL/SQL Programms:
  - PL/SQL Blöcke definieren die Struktur von PL/SQL Programmen.
  - PL/SQL Blöcke können im Kontext einer Datenbank ausgeführt werden.

- Strukturell gliedert sich ein PL/SQL Block in drei Teile.
- Erläuterung: Aufbau eines Blocks**
- Deklarationsteil**: Der Deklarationsteil eines Blocks enthält die für ein Programm erforderlichen Vereinbarungen in Form von Variablen, Konstanten, Cursor bzw. Fehlerzuständen. Der Deklarationsteil wird mit den Schlüsselwörtern `declare` bzw. `is` oder `as` eingeleitet. Der Deklarationsteil ist ein optionaler Bestandteil eines Blocks.
  - Ausführungsteil**: Der Ausführungsteil enthält die eigentliche Funktionalität eines PL/SQL Programms als freie Abfolge von Anweisungen und Befehlen. Der Ausführungsteil wird durch das Schlüsselwort `begin` eingeleitet.

Der Ausführungsteil ist **obligatorisch**.

- Fehlerbehandlungsteil: Tritt im Ausführungsteil eines PL/SQL Blocks ein Fehler auf, verzweigt die Ausführung des PL/SQL Programms in den Fehlerbehandlungsteil des Blocks. Der Fehlerbehandlungsteil wird durch das Schlüsselwort `exception` eingeleitet.

Der Fehlerbehandlungsteil ist **optional**.

**Codebeispiel: Anonymer Block**

```

1 -----
2 -- Syntax: Anonymer Block
3 -----
4 DECLARE
5   Declaration Statements
6 BEGIN
7   Executable Statements
8 EXCEPTION
9   Exception handling Statements
10 END;
11
12 -----
13 -- Codebeispiel: Block
14 -----
15 DECLARE
16   l_index1 NUMBER DEFAULT 10;
17 BEGIN
18   ...
19 EXCEPTION
20   ...
21 END;
22 
```

96

| Bausein               | Beschreibung                                                                                                                          | Vorkommen     |
|-----------------------|---------------------------------------------------------------------------------------------------------------------------------------|---------------|
| Deklarationsteil      | Der Deklarationsteil enthält die Vereinbarungen der für das Programm erforderlichen Variablen, Konstante, Cursor bzw. Fehlerzustände. | optional      |
| Ausführungsteil       | Der Ausführungsteil enthält die eigentliche Funktionalität eines PL/SQL Programms als freie Abfolge von Anweisungen und Befehlen.     | obligatorisch |
| Fehlerbehandlungsteil | Im Fehlerbehandlungsteil wird die Fehlerbehandlung eines PL/SQL Programms implementiert.                                              | optional      |

Abbildung 39. Blockstruktur

**10.2.2 Anonymer Block**

Ein PL/SQL Programm ist grundsätzlich immer als Blockstruktur aufgebaut.

- » Erklärung: Anonymer Block »
- Die PL/SQL Spezifikation unterscheidet 2 Arten von Blocktypen: anonyme Blöcke und benannte Blöcke.
- Ein anonymous Block besitzt keine Definitionseindifferenzierung und keinen Namen und kann daher nicht aus anderen PL/SQL Programmen heraus aufgerufen werden.
- Anonyme Blöcke können in einer Datenbank direkt ausgeführt werden.
- Anonyme Blöcke können im Vergleich zu benannten Blocktypen jedoch nicht in Datenbanken gespeichert werden.

» Codebeispiel: Anonymer Block »

```

1 -- -----
2 -- Anonymer Block
3 --
4 DECLARE
5   l_last_name VARCHAR2(50) := 'Washington';
6   l_first_name VARCHAR2(50) := 'Dennzel';
7 BEGIN
8   DBMS_OUTPUT.PUT_LINE(
9     'person data: '
10    || l_first_name
11    || ' '
12    || l_last_name
13  );
14 END;

```

**10.2.3 Benannte Blocktypen**

Die PL/SQL Spezifikation definiert mehrere Typen von benannten Block.

» Auflistung: Arten von Blöcken »

- Package Block ▾
- En Package fasst logisch zusammenhängende Objekte zu einer modularen Einheit zusammen.
- Packages werden zur Strukturierung von PL/SQL Programmen verwendet.
- Procedur Block ▾
- Eine Prozedur beschreibt eine freie Abfolge von Anweisungen und Routinen. Prozeduren können in der PL/SQL Engine ausgeführt werden. Prozeduren werden zur Strukturierung von PL/SQL Programmen verwendet.
- Funktion Block ▾
- PL/SQL Funktionen sind Prozeduren die einen Rückgabewert an den Aufrufer der Funktion zurückgeben.
- Trigger Block ▾
- Triggers sind eine besondere Form von Prozeduren, die in Reaktion auf ein bestimmtes Datenbankinternes Ereignis ausgelöst werden.

97

## Informationssysteme

## 11. PL/SQL - Datentypen

## 02

## PLSQL-Datentypen

|                              |     |
|------------------------------|-----|
| 01. Datentypen und Variablen | 98  |
| 02. Basistypen               | 100 |
| 03. Ableitete Datentypen     | 103 |
| 04. Strukturierte Datentypen | 103 |
| 05. Tabellentypen            | 104 |

## 11.1. Datentypen und Variablen

## 11.1.1 Definition von Variablen

98



## Datentypen

### 11.1.2 Gültigkeit von Variablen

Variablen gelten in jenem Block, indem sie definiert wurden.

► Erklärung: Gültigkeit von Variablen

- Blöcke können in PL/SQL Programmen ineinander verschachtelt werden.

■ Um Namenskonflikte bei der Vergabe von Variablennamen zu vermeiden, verbergen Variablen innerer Blöcke, Variablen umgebender Blöcke.

► Codebeispiel: Gültigkeit von Variablen

```

1 -- Gültigkeit von Variablen
2
3 DECLARE
4   l_name varchar2(20) := 'A. Gumenbauer';
5 BEGIN
6   -- GÜLTIGKEIT: Innerer Block
7   DECLARE
8     l_name varchar2(20);
9   BEGIN
10    l_name := 'Alfred Peter';
11
12    DBMS_OUTPUT.PUT_LINE(
13      'Vorname' || l_name
14      || chr(13)
15      || chr(13)
16    );
17  END;
18 END;
19
20 DBMS_OUTPUT.PUT_LINE(l_name);
21
22 END;

```

### 11.1.3 Qualifier

Qualifier

Ein Qualifier ist ein logischer Verweis auf einen PL/SQL Block. Als Qualifier wird der Name des Blocks verwendet.

Der Zugriff auf einen Qualifier erfolgt über den Namen eines PL/SQL Blocks.

► Codebeispiel: Qualifier

```

1 -- Qualifier
2
3 DECLARE
4   l_count PLS_INTEGER DEFAULT 0;
5 BEGIN
6   DECLARE
7     l_inner NUMBER;
8   BEGIN
9     SELECT count(project_id)
10       INTO local_block.l_inner
11     FROM projects;
12
13     init.l_count := local_block.l_inner;
14   END local_block;
15
16   project.g.project_count := init.l_count;
17
18   DBMS_OUTPUT.PUT_LINE(
19     'project count: ' || init.l_count
20     );
21
22 END init;
23
24 END;

```

99

### 11.1.4 Datentypen

Ein Datentyp bestimmt den Wertebereich und die Operationen, die für eine Variable zulässig sind.

► Auflistung: Arten von Datentypen

#### ■ Basistypen

Als Basistypen wird eine Gruppe von Datentypen, zur Verwaltung von Zahlen, einzelner Zeichen bzw. logischer Werte bezeichnet.

Abgeleitete Datentypen

Neben der direkten Angabe eines Datentyps, kann in PL/SQL Programmen der Datentyp einer Variable implizit referenziert werden.

Wir sprechen in diesem Fall von abgeleiteten Datentypen.

Strukturierte Datentypen

Strukturierte Datentypen lassen logisch zusammenhängende Basistypen zu einem strukturellen Verbund zusammen.

Tabellentypen

Tabellentypen ermöglichen die Verwaltung von Werten in einer Tabellenstruktur.

### 11.2. Basisdatentypen

Basisdatentypen

Als Basisdatentypen wird eine Gruppe von Datentypen zur Verwaltung von Zahlen, einzelner Zeichen bzw. logischer Werte bezeichnet.

Die PL/SQL Spezifikation definiert 4 Basisdatentypen.

► Auflistung: Kategorien

- Numerische Datentypen: Numerische Datentypen kommen bei der Verarbeitung von Zahlenwerten zum Einsatz.
- Alphanumerische Datentypen: Alphanumerische Datentypen kommen bei der Verarbeitung von Zeichenketten zum Einsatz.
- Zeitbezogene Datentypen: Zeitbezogene Datentypen kommen bei der Verarbeitung von Datumswerten zum Einsatz.
- Boolische Datentypen: Eine booleche Variable wird verwendet um Wahrheitswerte zu verarbeiten.

#### 11.2.1 Numerische Datentypen

Numerische Werte: Number ist der grundlegende Datentyp zur Verwaltung numerischer Datenwerte.

► Codebeispiel: Numerische Datentypen

```

1 -- -----
2 -- Datentyp: numerische Werte
3
4
5 DECLARE
6   l_weeks_to_pay NUMBER(2) := 52;
7   l_raise_factor NUMBER := 0.05;
8   l_salary NUMBER(1,9) := 1234567.89;
9   l_factor NATURAL DEFAULT 0;
10
11 BEGIN
12   DBMS_OUTPUT.PUT_LINE(
13     l_weeks_to_pay
14     || chr(13)
15     || l_raise_factor
16     || chr(13)
17     || l_salary
18   );
19 END;

```

100

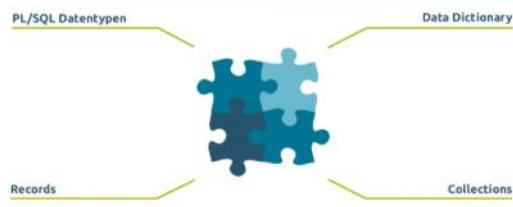


Abbildung 40. Typen von Datentypen

```

Codebeispiel: Precision und Scale
1 -- Datentyp: Precision und Scale
2
3 DECLARE
4     l_v1_nr NUMBER(10) := 2.567;
5     l_v2_nr NUMBER(5,2) := 123.567;
6     l_v3_nr NUMBER(3) := 2/3;
7     l_v4_nr NUMBER(5,2) := 2/3;
8
9     l_v5_nr PLS_INTEGER := 201;
10    l_v6_nr BINARY_FLOAT := 323.332;
11    l_v7_nr BINARY_DOUBLE := 323321.32
12
13 BEGIN
14     DBMS_OUTPUT.PUT_LINE(
15         l_v1_nr
16         || 'v2: ' || l_v2_nr
17         || 'v3: ' || l_v3_nr
18         || 'v4: ' || l_v4_nr * 3
19         || 'v5: ' || l_v5_nr
20     );
21 END;
22
23 -- Ausgabe
24 v1: 3
25 v2: 123,57
26 v3: 2
27 v4: 2.01
28 v5: 201
29 v6: 323.332
30 v7: 323321.32

```

**11.2.2 Alphanumerische Datentypen**

Alphanumerische Datentypen kommen bei der Verarbeitung von Zeichenketten zum Einsatz.

Verwenden Sie den VARCHAR2-Datentyp zur Verarbeitung von Zeichenketten in PL/SQL.

**Auflistung: PL/SQL-Datentypen**

- Zeichenketten - Char: Datentypen zur Verarbeitung alphanumerischer Werte, bestehen 2 Ausprägungen: Datentypen mit einer variablen und Datentypen mit einer festgelegten Länge.
- Char<sup>3</sup> ist ein Datentyp mit festgelegter Länge. Bei einem Datentyp mit festgelegter Länge wird die gespeicherte Zeichenkette solange mit Leerzeichen aufgefüllt wie die Zeichenkette die in der Variableneinführung angegebene Länge hat.
- Zeichenketten - Varchar2: Varchar2 ist ein Datentyp mit variabler Länge. Zeichenketten die einer Variable eines Datentyps variabler Länge zugewiesen werden benötigen in der Datenbank genauso viel Platz wie viele Zeichen sie beinhalten.
- Varchar2 ist der wichtigste Datentyp zur Verarbeitung von Zeichenketten.

<sup>3</sup> Char ist ein veralteter Datentyp und hat in der PL/SQL Programmierung keine Bedeutung mehr.

101

| Datentyp      | Beschreibung                                                                                                                                                                                                                                                                                        | Zuordnung |
|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------|
| NUMBER        | Numerischer Datentyp zur Verwaltung von Gleitkommazahlen                                                                                                                                                                                                                                            | Double    |
| NUMBER(10)    | Numerischer Datentyp zur Verwaltung ganzzahliger Zahlenwerte - 10 stelliger Integer                                                                                                                                                                                                                 | Integer   |
| NUMBER(2,9)   | Numerischer Datentyp zur Verwaltung von Repunktzahl - Repunktzahl mit 7 Stellen vor dem Komma und 2 Stellen nach dem Komma.                                                                                                                                                                         | Float     |
| PLS_INTEGER   | Numerischer Datentyp zur Verwaltung von ganzzähligen Zahlenwerten. PLS_INTEGER ist ein PL/SQL interner Datentyp. PLS_INTEGER ist ein binner Datentyp der weniger Platz beansprucht und schneller verarbeitet werden kann als NUMBER.                                                                | Integer   |
| NATURAL       | Numerischer Datentyp zur Verwaltung von ganzzähligen positiven Zahlenwerten- positive Integer bzw. dem null Wert. Natural ist PL/SQL interner Datentyp.                                                                                                                                             | Integer   |
| BINARY_FLOAT  | Numerischer Datentyp zur Verwaltung von Fixkommazahlenwerten. BINARY_FLOAT ist ein PL/SQL interner Datentyp. BINARY_FLOAT ist ein binner Datentyp der weniger Platz beansprucht und schneller verarbeitet werden kann als NUMBER.                                                                   | Float     |
| BINARY_DOUBLE | Numerischer Datentyp zur Verwaltung von Fixkommazahlenwerten. BINARY_DOUBLE ist ein PL/SQL interner Datentyp. BINARY_DOUBLE ist ein binner Datentyp der weniger Platz beansprucht und schneller verarbeitet werden kann als NUMBER.                                                                 | Double    |
| VARCHAR(255)  | Alphanumerischer Datentyp zur Verwaltung von Zeichenketten. Zeichenketten die einer Variable eines Datentyps mit variabler Länge zugeordnet werden nehmen bei der Speicherung genauso viel Platz ein wie viele Zeichen sie beinhalten.                                                              | String    |
| CHAR(100)     | Alphanumerischer Datentyp zur Verwaltung von Zeichenketten. Char ist ein Datentyp mit festgelegter Länge. Bei einem Datentyp mit festgelegter Länge wird die gespeicherte Zeichenkette solange mit Leerzeichen aufgefüllt bis die Zeichenkette die in der Variableneinführung angegebene Länge hat. | String    |
| VARCHAR(255)  | Alphanumerischer Datentyp zur Verwaltung von Zeichenketten. Zeichenketten die einer Variable eines Datentyps mit variabler Länge zugeordnet werden nehmen bei der Speicherung genauso viel Platz ein wie viele Zeichen sie beinhalten.                                                              | String    |
| DATE          | Datentyp zur Verwaltung von zeitbezogenen Daten                                                                                                                                                                                                                                                     | Date      |
| TIMESTAMP     | Datentyp zur Verwaltung von zeitbezogenen Daten                                                                                                                                                                                                                                                     | Date      |
| BOOLEAN       | Datentyp zur Verwaltung von boolschen Wahrheitswerten                                                                                                                                                                                                                                               | Boolean   |

Abbildung 41. Skalare Datentypen

### 11.3. Abgeleitete Datentypen

Neben der Angabe eines Datentyps, kann in PL/SQL ein Programm der Datentyp einer Variablen auch implizit referenziert werden.

#### 11.3.1 Data Dictionary

Mit dem `TYPE` bzw. `XRWTYPOE` Attribut kann der Datentyp einer Variable implizit referenziert werden.

##### Codebeispiel: Ableiten von Datentypen

```

1 -- Abgeleitete Datentypen
2
3 DECLARE
4   -- IMPLIZITE REFERENZIERUNG
5   l_given_name EMPLOYEES.first_name%TYPE;
6   l_surname EMPLOYEES.last_name%TYPE;
7
8   -- IMPLIZITE REFERENZIERUNG
9   l_person_rec EMPLOYEE%ROWTYPE;
10
11 BEGIN
12   l_given_name := 'Tokian';
13   l_surname := 'Huber';
14
15   DBMS_OUTPUT.PUT_LINE(l_given_name);
16   DBMS_OUTPUT.PUT_LINE(l_surname);
17
18   SELECT * INTO l_person_rec
19   FROM EMPLOYEES e
20   WHERE e.first_name = l_given_name AND
21     e.last_name = l_surname;
22
23   DBMS_OUTPUT.PUT_LINE(
24     'Employee data: '
25     || chr(13)
26     || 'firstname: '
27     || l_person_rec.first_name
28     || chr(13)
29     || 'lastname: '
30     || l_person_rec.last_name
31     || chr(13)
32     || 'salary: '
33     || l_person_rec.salary
34     || chr(13));
35
36 END;
  
```

### 11.4. Strukturierte Datentypen

Strukturierte Datentypen lassen logisch zusammenhängende Basistypen zu einem strukturellen Verbund zusammen.

#### 11.4.1 Record - Strukturierte Datentypen

Ein Record ist eine Variable mit einem strukturierten Datentyp.

##### Erklärung: Komplexe Datentypen

- Records werden zur Verarbeitung von Datensätzen verwendet.
- Der Zugriff auf die Felder eines Records erfolgt über den Punktoperator.

##### Syntax: Record

```

1 -- Syntax: Strukturierter Datentyp
2
3 TYPE <typename> IS RECORD (
4   <fieldname> <type> [NOT NULL {:=} <exp>];
5   ...
6 );
7
8   ...
9
10   Datentyp: Strukturierter Datentyp
11
12 DECLARE
13   -- TYPEDEFINITION
14   TYPE l_person_rec_t IS RECORD (
15     first_name EMPLOYEES.FIRST_NAME%TYPE,
16     last_name EMPLOYEES.LAST_NAME%TYPE,
17     id NUMBER NOT NULL DEFAULT 1,
18     publication_date DATE
19   );
20
21   -- RECORD
22   l_person_rec l_person_rec_t;
23
24 BEGIN
25   l_person_rec.id := 17;
26   ...
27   DBMS_OUTPUT.PUT_LINE(
28     || l_person_rec.first_name
29     || l_person_rec.last_name
30   );
31
32 END;
  
```

103

### 11.5. Tabellentypen

Tabellentypen  
Tabellentypen ermöglichen die Verwaltung von Werte in Tabellenstrukturen.

#### 11.5.1 Tabellentypen

Variablen mit einem Tabellentyp werden als **Virtuelle Tabellen** bezeichnet.

Bevor eine Virtuelle Tabelle definiert werden kann, muss ein entsprechender Tabellentyp definiert werden.

##### Syntax: Tabellentypen

```

1 -- -----
2   -- Syntax: Tabellentyp
3   -- -----
4   TYPE <typename> IS TABLE OF <datatype>;
5
6   --
7   -- Tabellentyp: Basistypen
8
9   --
10  DECLARE
11    -- DEFINITION: Tabellentyp
12    TYPE NAME_TABLE_TYPE IS TABLE OF
13      VARCHAR2(10);
14
15    -- DEFINITION: Tabellentyp
16    TYPE CODE_TABLE_TYPE IS TABLE OF
17      NATURAL;
18
19    -- VARIABLE: Virtuelle Tabelle
20    l_names NAME_TABLE_TYPE := 
21      NAME_TABLE_TYPE(
22        'Franz', 'Josef', 'Daniel',
23      );
24
25    l_codes CODE_TABLE_TYPE :=
26      CODE_TABLE_TYPE(12,3,12);
27
28 BEGIN
29   FOR i IN l_names.FIRST..l_names.LAST
30   LOOP
31     DBMS_OUTPUT.PUT_LINE(
32       'project data: '
33       || l_names(i).TITLE
34       || chr(13)
35       || l_names(i).DESCRIPTION
36       || chr(13));
37   END LOOP;
38
39 END;
  
```

### 11.5.2 Virtuelle Tabellen: Strukturierte Daten

#### Codebeispiel: Virtuelle Tabellen

```

1 -- -----
2   -- Tabellendatentypen: Strukturierte D.
3   -- -----
4   --
5   -- DEFINITION: Tabellentyp
6   TYPE PROJECT_TABLE_TYPE IS TABLE OF
7     HR.L_PROJECTS%ROWTYPE;
8
9   -- DECLARATION: Virtuelle Tabelle
10  l_projects PROJECT_TABLE_TYPE := 
11    PROJECT_TABLE_TYPE();
12
13   -- DEFINITION: Strukturierter Datentyp
14  TYPE PROJECT_REPORT_TYPE IS RECORD (
15    TITLE HR.L_PROJECTS.TITLE%TYPE,
16    DESCRIPTION VARCHAR2(4000),
17    PROJECT_ID NUMBER(10,2),
18    FUNDING NUMBER(10),
19    COMPLEXITY NUMBER(100)
20  );
21
22   -- DEFINITION: Tabellentyp
23  TYPE PR_TABLE_TYPE IS TABLE OF
24    PROJECT_REPORT_TYPE;
25
26   -- DECLARATION: Virtuelle Tabelle
27  l_project_reports PR_TABLE_TYPE := 
28    PR_TABLE_TYPE();
29
30 BEGIN
31   SELECT l.id, l.funding, l.title
32     l.description, 1
33   BULK COLLECT INTO l_project_reports
34   FROM HR.L_PROJECTS l
35   WHERE l.PROJECT_TYPE = 'STIPENDIUM';
36
37   FOR i IN l_project_reports.FIRST..l_project_reports.LAST
38   LOOP
39     DBMS_OUTPUT.PUT_LINE(
40       'project data: '
41       || l_projects(i).TITLE
42       || chr(13)
43       || l_projects(i).DESCRIPTION
44       || chr(13));
45   END LOOP;
46
47 END;
  
```

104

| Pseudospalte | Beschreibung                                                                        |
|--------------|-------------------------------------------------------------------------------------|
| prior, next  | Navigiert zum vorangegangenen oder nachfolgenden Element                            |
| count        | Zählt die Einträge                                                                  |
| first, last  | Navigtiert zum ersten bzw. letzten Element.                                         |
| extend       | Mit der extend Funktion wird die Größe einer virtuellen Tabelle explizit gesteuert. |

Abbildung 42. Virtuelle Tabellen: Pseudospalten

### 11.5.3 Virtuelle Tabellen

Mit der extend Funktion kann die Größe einer Virtuellen Tabelle explizit gesteuert werden.

#### ► Codebeispiel: Virtuelle Tabellen ▾

```

1 -- Virtuelle Tabellen
2 --
3 --
4 DECLARE
5   -- DEFINITION: Tabellentyp
6   TYPE PROJECT_TABLE_TYPE IS TABLE OF
7     HR.C_PROJECTS%ROWTYPE;
8 
9   -- Deklaration: Virtuelle Tabelle
10  l_projects PROJECT_TABLE_TYPE := 
11    PROJECT_TABLE_TYPE();
12 
13  p_i PLS_INTEGER DEFAULT 1;
14 
15  l_research HR.C_RESEARCH_PROJECT%ROWTYPE;
16  l_project HR.C_PROJECT%ROWTYPE;
17 
18 BEGIN
19   FOR project IN
20     ( SELECT p.* FROM L_PROJECTS)
21   LOOP
22     -- Explizite Änderung der Groesse
23     l_projects.EXTEND;
24     l_projects(p_i).TITLE := project.TITLE;
25     l_projects(p_i).DESCRIPTION := 
26       project.DESCRIPTION;
27     p_i := p_i + 1;
28   END LOOP;
29 
30   DBMS_OUTPUT.PUT_LINE(
31     'project count: ' || l_projects.COUNT
32   );
33 END;
  
```

### 11.5.4 DQL: Virtuelle Tabellen

Zur Verarbeitung der Daten einer Virtuellen Tabelle wird die select Anweisung verwendet.

#### ► Codebeispiel: DQL: Virtuellen Tabellen ▾

```

1 -- DQL: Virtuelle Tabellen
2 --
3 --
4 DECLARE
5   -- DEFINITION: Tabellentyp
6   TYPE PROJECT_REPORT_TABLE_TYPE IS TABLE
7     OF HR.L_PROJECT_REPORT%ROWTYPE;
8 
9   l_project_reports
10  l_PROJECT_REPORT_TABLE_TYPE := 
11    PROJECT_REPORT_TABLE_TYPE();
12 
13 BEGIN
14   l_project_reports
15   BEGIN
16     -- Verarbeiten der Daten der Virtuellen
17     -- Tabelle. Speichern des Ergebnisses
18     SELECT 1.*
19     BULK COLLECT INTO l_project_reports
20     FROM HR.PROJECT_REPORT
21     JOIN TABLE(L_PROJECTS) p
22       ON L_PROJECT_ID = p.PROJECT_ID
23     WHERE p.FOUNDING_AMOUNT > 10000;
24 
25   DBMS_OUTPUT.PUT_LINE(
26     'project count: ' || l_projects.COUNT
27   );
28 END;
  
```

105

## Informationssysteme

### 12. PL/SQL - Befehle

# 03

#### PLSQL - Befehle

### 12.1. SQL Befehle in PLSQL

PL/SQL stellt eine prozedurale Erweiterung der SQL Sprachspezifikation dar.

Ein PL/SQL Programm stellt eine freie Abfolge von PL/SQL Anweisungen und SQL Befehlen dar.

#### 12.1.1 DQL - Data Query Language

Die select Anweisung wird in PL/SQL Programmen zum Lesen von Daten verwendet.

#### ► Erklärung: select Anweisung ▾

- Gelsene: Daten können dabei in der INTO Klammer einer Variable zugewiesen werden.
- Besteht das Ergebnis der select Anweisung aus mehreren Datensätzen, wird anstelle der into Klammer die bulk collect into Klammer verwendet.

#### ► Codebeispiel: select Befehl ▾

```

1 -- Befehl: DQL Befehle
2 --
3 --
4 DECLARE
5   l_title VARCHAR(50);
6 
7   TYPE PROJECT_TABLE_TYPE IS TABLE OF
8     HR.L_PROJECTS%ROWTYPE;
9 
10  l_projects PROJECT_TABLE_TYPE := 
11    PROJECT_TABLE_TYPE();
12 
13 BEGIN
14   -- Auslesen eines einzelnen Datensatzes
15   -- Speichern das Ergebniss in einer
16   -- Variable
17   SELECT p.title INTO l_title
18   FROM HR.L_PROJECTS p
19   WHERE p.project_id = 123;
20 
21   -- Auslesen mehrerer Datensätze
22   SELECT 1.*
23   BULK COLLECT INTO l_projects
24   FROM HR.L_PROJECTS 1;
25 
26   DBMS_OUTPUT.PUT_LINE('select into ...');
27 
28 END;
  
```

106

**12.1.2 DML - Data Manipulation Language**

DML Befehle werden in PL/SQL Programmen zur Verarbeitung von Daten verwendet.

## ► Codebeispiel: DML Befehle ▶

```

1 -- Befehl: DML Befehle
2
3
4 DECLARE
5   TYPE PROJECT_TABLE_TYPE IS TABLE OF
6     HR_PROJECT_REPORT%ROWTYPE;
7
8   TYPE REPORT_TABLE_TYPE IS TABLE OF
9     HR_PROJECT_REPORT%ROWTYPE;
10
11   l_projects PROJECT_TABLE_TYPE := 
12     PROJECT_TABLE_TYPE();
13
14   l_project_report
15     HR_PROJECT_REPORT%ROWTYPE;
16
17 BEGIN
18
19   -- Auslesen der Projektdaten in eine
20   -- virtuelle Tabelle
21   SELECT p.*
22   BULK COLLECT INTO l_projects
23   FROM HR_L_PROJECTS P;
24
25   -- Löschen aller Projektdatarden
26   DELETE FROM HR_PROJECT_REPORT;
27
28   FOR i IN l_projects.FIRST..l_projects.last
29   LOOP
30     l_project_report.TITLE := 
31       l_projects(i).TITLE;
32     l_project_report.COMPLEXITY := 100;
33
34     INSERT INTO HR_PROJECT_REPORT VALUES
35       l_project_report;
36   END LOOP;
37
38   -- Ändern der Projektdaten mit einer
39   -- bestimmten ID.
40   UPDATE HR_PROJECT_REPORT
41   SET COMPLEXITY = 200
42   WHERE PROJECT_ID = 23;
43
44 END;

```

**12.1.3 FORALL Anweisung**

Bei der Ausführung von DML Befehl leitet die PL/SQL Engine die SQL Anweisung an den SQL Executor weiter.

## ► Erklärung: forall Anweisung ▶

- Ein solcher Kontextswitch ist immer mit einem bestimmten Ressourceneaufwand verbunden.
- Zur Ressourcenschonung können mit der forall Anweisung mehrere DML Befehle als einzelne PL/SQL Anweisung ausgeführt werden.

## ► Syntax: forall Anweisung ▶

```

1 -- -----
2 -- Syntax: FORALL
3 -- -----
4 FORALL index IN
5   [ lower_bound .. upper_bound ]
6   [ INDICES OF indexing_collection ]
7   [ VALUES OF indexing_collection ]
8
9   [ SAVE EXCEPTION ]
10  sql_statement;
11
12 -- Codebeispiel: FORALL
13
14 DECLARE
15
16   -- DEFINITION: Tabellentyp
17   TYPE PROJECT_REPORT_TYPE IS TABLE OF
18     HR_PROJECT_REPORT%ROWTYPE;
19
20   -- Deklaration: Virtuelle Tabelle
21   l_project_reports PROJECT_REPORT_TYPE := 
22     PROJECT_REPORT_TYPE();
23
24 BEGIN
25
26   -- Auslesen von Datensätzen
27   SELECT p.PROJECT_ID, p.TITLE, 0
28   BULK COLLECT INTO l_project_reports
29   FROM HR_L_PROJECTS P;
30
31   -- Schreiben von Datensätzen in die
32   -- Datenbank
33   FORALL i IN l_project_reports.FIRST ..
34     l_project_reports.LAST
35     INSERT INTO HR_PROJECT_REPORT VALUES
36       l_project_reports(i);
37
38 END;

```

107

**12.1.4 FORALL - RETURNING Klausel**

Der Zugriff auf mögliche Rückgabewerte von DML Befehlen erfolgt für forall Anweisungen in der returning Klausel.

## ► Codebeispiel: returning Klausel ▶

```

1 -- Codebeispiel: FORALL RETURNING
2
3
4 DECLARE
5
6   -- DEFINITION: Tabellentyp
7   TYPE PROJECT_REPORT_TYPE IS TABLE OF
8     HR_PROJECT_REPORT%ROWTYPE;
9
10  -- DEFINITION: Tabellentyp
11  TYPE PROJECT_ID_TYPE IS TABLE OF
12    HR_PROJECT_REPORT%PROJECT_IDTYPE;
13
14  -- Deklaration: Virtuelle Tabelle
15  l_project_reports PROJECT_REPORT_TYPE := 
16    PROJECT_REPORT_TYPE();
17
18  -- Deklaration: Virtuelle Tabelle
19  l_project_ids PROJECT_ID_TYPE := 
20    PROJECT_ID_TYPE();
21
22 BEGIN
23
24   -- Auslesen von Projektdaten
25   SELECT p.PROJECT_ID, p.TITLE, 0
26   BULK COLLECT INTO l_project_reports
27   FROM HR_L_PROJECTS P;
28
29   -- forall Befehl
30   FORALL i IN l_project_reports.FIRST ..
31     l_project_reports.LAST
32     INSERT INTO HR_PROJECT_REPORT
33       VALUES l_project_reports(i)
34     RETURNING project_id
35       INTO l_project_ids;
36
37   -- Wertausgabe
38   FOR i IN
39     l_project_ids.first..l_project_ids.last
40   LOOP
41     DBMS_OUTPUT.PUT_LINE(
42       l_project_ids(i));
43   END LOOP;
44
45 END;

```

**12.1.5 DDL - Data Definition Language**

DDL Befehle werden zum Verwalten der Struktur einer Datenbank verwendet.

DDL Befehle müssen im Kontext der execute immediate Anweisung definiert werden.

## ► Codebeispiel: DDL Befehle ▶

```

1 -- Befehl: DDL Befehle
2
3
4 DECLARE
5
6   l_is_table_present PLS_INTEGER DEFAULT 0;
7
8   -- Definition eines DDL Befehls in Form
9   -- eines Zeichenkettenliterals
10  l_c_create_table_ddl VARCHAR2(500) := 
11    'CREATE TABLE PROJECT_REPORT (
12      PROJECT_ID INTEGER NOT NULL,
13      TITLE VARCHAR(50) NOT NULL,
14      DESCRIPTION VARCHAR(4000),
15      FUNDING_AMOUNT NUMBER(10),
16      PRIMARY KEY (PROJECT_ID)
17    )';
18
19   -- Definition eines DDL Befehls in Form
20   -- eines Zeichenkettenliterals
21  l_c_drop_table_ddl CONSTANT VARCHAR2(500) :=
22    'DROP TABLE PROJECT_REPORT';
23
24 BEGIN
25
26   -- Prüfen ob eine bestimmte Tabelle
27   -- bereits existiert.
28   SELECT COUNT(I.TABLE_NAME)
29   INTO l_is_table_present
30   FROM USER_TABLES I
31   WHERE I.TABLE_NAME = 'PROJECT_REPORT';
32
33   -- Existiert die Tabelle noch nicht
34   -- wird sie mit der EXECUTE IMMEDIATE
35   -- Anweisung angelegt.
36   IF l_is_table_present = 0 THEN
37     EXECUTE IMMEDIATE l_c_create_table_ddl;
38   END IF;
39
40   -- Prüfen ob eine bestimmte Tabelle
41   -- bereits existiert.
42   SELECT COUNT(I.TABLE_NAME)
43   INTO l_is_table_present
44   FROM USER_TABLES I
45   WHERE I.TABLE_NAME = 'PROJECT_REPORT';
46
47   -- Existiert die Tabelle noch nicht
48   -- wird sie mit der EXECUTE IMMEDIATE
49   -- Anweisung angelegt.
50   IF l_is_table_present = 0 THEN
51     EXECUTE IMMEDIATE l_c_drop_table_ddl;
52   END IF;
53
54 END;

```

106

## 12.2. Kontrollstrukturen

Kontrollstrukturen sind Routinen zur Steuerung des Programmablaufs.

### 12.2.1 IF THEN Block

Kontrollstrukturen bestimmen die Reihenfolge, in der die Anweisungen eines Programms ausgeführt werden.

Trifft die Bedingung einer if Anweisung zu, werden die Befehle im if then Blocks ausgeführt.

#### Syntax: IF THEN \*

```

1 -- -----
2 -- Syntax: IF THEN
3 --
4
5 IF condition THEN
6   Statement1;
7   Statement2;
8 ...
9 ELSIF condition2 THEN
10  Statement1;
11  Statement2;
12 ...
13 ELSE
14   StatementN;
15 END IF;
16 --
17 -- Beispiel: IF THEN
18
19 DECLARE
20   l_employees_count NUMBER := 0;
21   l_number NUMBER := 0;
22 BEGIN
23   SELECT count(employee_id)
24   INTO l_employees_count
25   FROM employees;
26
27   if l_employees_count THEN
28     DBMS_OUTPUT.PUT_LINE(
29       'Inside the if'
30     );
31   END IF;
32
33   DBMS_OUTPUT.PUT_LINE(
34     'Outside the if'
35   );
36 END;

```

### 12.2.2 Kontrollstruktur - Case Block

Alternativ zur if Anweisung steht in PL/SQL Programmen die case Anweisung zur Verfügung.

#### Syntax: Case Block \*

```

1 -- -----
2 -- Syntax: CASE Block
3 --
4
5 CASE
6   WHEN condition1 THEN
7     Statement1;
8     Statement2;
9   ...
10  WHEN condition2 THEN
11    Statement1;
12    ...
13  ELSE
14    Statement1;
15    ...
16  END CASE;
17 --
18 -- Beispiel: CASE Block
19
20 DECLARE
21   l_version PLIS_INTEGER DEFAULT 12;
22 BEGIN
23   -- read database version
24   SELECT version
25   INTO l_version
26   FROM dba_session;
27
28   -- case Kontrollstruktur
29   CASE l_version
30     WHEN 11 THEN
31       DBMS_OUTPUT.PUT_LINE(
32         'no feature support'
33       );
34     WHEN 12 THEN
35       DBMS_OUTPUT.PUT_LINE(
36         'feature support'
37       );
38     ELSE
39       DBMS_OUTPUT.PUT_LINE(
40         'no information'
41       );
42   END CASE;
43
44 END;

```

109

## 12.3. Schleifenstrukture

### Schleifenstrukture

Eine Schleife ist eine Programmroutine, die einen bestimmten Programmabschnitt solange bearbeitet, bis eine bestimmte Bedingung eintritt.

### 12.3.1 Loop Block

Der loop Block ist die grundlegende Schleifenvariante in PL/SQL. Der Block definiert dabei keine Abbruchbedingung.

#### Syntax: Loop Block \*

```

1 -- -----
2 -- Beispiel: Loop Block
3 --
4
5 LOOP
6   Statement1;
7   ...
8   StatementN;
9 END LOOP;
10 --
11 -- Beispiel: Loop Block
12
13 DECLARE
14   l_counter NUMBER := 0;
15   l_result NUMBER;
16 BEGIN
17   LOOP
18     IF l_counter > 10 THEN
19       EXIT;
20     END IF;
21
22     l_counter := l_counter + 1;
23     l_result := l_counter * 5;
24
25     DBMS_OUTPUT.PUT_LINE(
26       'i' || l_counter
27       || ' = '
28       || l_result
29     );
30
31   END LOOP;
32 END;

```

### 12.3.2 WHILE Block

Mit der WHILE Schleifenvariante wird ein bestimmter Programmabschnitt solange wiederholt bis eine bestimmte Bedingung erfüllt.

#### ► Syntax: WHILE Block ▶

```

1  -- -----
2  -- Syntax: WHILE LOOP
3  --
4  WHILE condition LOOP
5      Statement;
6      ...
7  END LOOP;
8  --
9  -- Beispiel: WHILE LOOP
10 --
11  DECLARE
12      TYPE PROJECT_TABLE_TYPE IS TABLE OF
13          HR.L_PROJECTS%ROWTYPE;
14
15      l_projects PROJECT_TABLE_TYPE := 
16          PROJECT_TABLE_TYPE();
17
18      l_i PLS_INTEGER DEFAULT 1;
19  BEGIN
20      SELECT l_i
21      BULK COLLECT INTO l_projects
22      FROM HR.L_PROJECTS
23      WHERE l.project_type =
24          'REQUEST_FUNDING_PROJECT';
25
26      WHILE l_i < l.projects.LAST
27      LOOP
28          DBMS_OUTPUT.PUT_LINE(
29              'project_id: '
30              || l_projects(l_i).PROJECT_ID
31              || CHR(13)
32              || l_projects(l_i).TITLE
33              || CHR(13)
34              || l_projects(l_i).DESCRIPTION
35              || CHR(13)
36              || l_projects(l_i).IS_FWF_FUNDED
37              );
38
39      l_i := l_i + 1;
40  END LOOP;
41  END;

```

### 12.3.3 FOR Block

#### ► Syntax: FOR Block ▶

```

1  -- -----
2  -- Syntax: FOR LOOP
3  --
4  FOR loop_counter IN [REVERSE]
5      lower_limit .. upper_limit
6  LOOP
7      Statement1;
8      ...
9  END LOOP;
10 --
11  -- Syntax: FOR SQL LOOP
12  --
13  FOR row IN (sql statement)
14  LOOP
15      Statement1;
16      ...
17  END LOOP;
18  --
19  -- Beispiel: FOR LOOP
20  --
21  DECLARE
22      TYPE PROJECT_TABLE_TYPE IS TABLE OF
23          HR.L_PROJECTS%ROWTYPE;
24
25      l_projects PROJECT_TABLE_TYPE := 
26          PROJECT_TABLE_TYPE();
27
28      BEGIN
29          FOR row IN (SELECT p.* FROM HR.L_PROJECTS)
30          LOOP
31              DBMS_OUTPUT.PUT_LINE( row.PROJECT_ID );
32          END LOOP;
33
34          'project_id: '
35          || l_projects(1).PROJECT_ID
36          || CHR(13)
37          || l_projects(1).TITLE
38          || CHR(13)
39          || l_projects(1).DESCRIPTION
40          || CHR(13)
41          || l_projects(1).IS_FWF_FUNDED
42          );
43
44      END LOOP;
45  END;

```

III

### 12.4. Fehlerbehandlung

Tritt zur Laufzeit eines PL/SQL Programms ein Fehler auf, wird das Programm mit einer Fehlermeldung beendet.

#### 12.4.1 Exceptionhandling

Fehlerbehandlungsrichtlinien ermöglichen es PL/SQL Programmen auf Fehler zu reagieren.

##### ► Erklärung: Exceptionhandling ▶

- Tritt im Ausführungsteil eines PL/SQL Blocks ein Fehler auf, verzögert die Ausführung des Programms in den Fehlerbehandlungsteil des entsprechenden Blocks.
- Für jedes fehlerbezogene Ereignis, das in einem Programm auftreten kann, kann in Exceptionteil des entsprechenden Blocks, ein Exceptionhandler definiert werden.
- Exceptionhandler sind für die Verarbeitung der in einem Programm auftretenden Fehlerzustände verantwortlich.
- Fehlerzustände werden in PL/SQL als **Exception** und die Fehlerbehandlung als **Exceptionhandling** bezeichnet.
- Mit Exceptionhandlern kann in PL/SQL Programmen auf eintretende Fehler Bezug genommen und individuell auf sie reagiert werden.

##### ► Codebeispiel: Exceptionhandling ▶

```

1  -- -----
2  -- Beispiel: Exceptionhandling
3  --
4  DECLARE
5      l_amount PLS_INTEGER := 300;
6      l_relative PLS_INTEGER := 0;
7      l_result PLS_INTEGER;
8  BEGIN
9      -- Fehlerzustand: Division durch 0
10     l_result := l_amount/l_relative;
11
12     DBMS_OUTPUT.PUT_LINE( l_result );
13
14     EXCEPTION
15         WHEN ZERO_DIVIDE THEN
16             -- Exceptionhandler: ZERO_DIVIDE
17             ...
18             -- Exceptionhandler: NO_DATA_FOUND bzw.
19             -- ZERO_DIVIDE
20             WHEN NO_DATA_FOUND OR ZERO_DIVIDE THEN
21                 ...
22                 -- Exceptionhandler: OTHERS
23                 -- Der Exceptionhandler behandelt alle
24                 -- Ereignisse fuer die kein Exception-
25                 -- handler definiert worden ist.
26             WHEN OTHERS THEN
27                 ...
28
29 END;

```

#### 12.4.2 Exceptionhandler

Mit Exceptionhandlern kann in PL/SQL Programme auf eintretende Fehler Bezug genommen und individuell auf sie reagiert werden.

##### ► Erklärung: Exceptionhandler ▶

- Ein Exceptionhandler beginnt mit dem Schlüsselwort **WHEN** und endet mit der letzten Anweisung des zugeordneten Then Blöcks.
- Exceptionhandler werden im **Exceptionhandling**-teil eines PL/SQL Blocks definiert. Ein Exceptionteil kann dabei eine freie Abfolge von Exceptionhandlern enthalten.
- Die **Bezeichnung** eines Fehlerzustandes stellt dabei den Bezug zwischen Exception und Exceptionhandler her.
- Wird für einen Fehlerzustand kein Exceptionhandler definiert, bricht das PL/SQL Programm ohne Fehlerbehandlung ab.

##### ► Codebeispiel: Exceptionhandling ▶

```

1  -- -----
2  -- Beispiel: Exceptionhandling
3  --
4  DECLARE
5      l_amount PLS_INTEGER := 300;
6      l_relative PLS_INTEGER := 0;
7      l_result PLS_INTEGER;
8  BEGIN
9      -- Fehlerzustand: Division durch 0
10     l_result := l_amount/l_relative;
11
12     DBMS_OUTPUT.PUT_LINE( l_result );
13
14     EXCEPTION
15         WHEN ZERO_DIVIDE THEN
16             ...
17             -- Exceptionhandler: NO_DATA_FOUND bzw.
18             -- ZERO_DIVIDE
19             WHEN NO_DATA_FOUND OR ZERO_DIVIDE THEN
20                 ...
21                 -- Exceptionhandler: OTHERS
22                 -- Der Exceptionhandler behandelt alle
23                 -- Ereignisse fuer die kein Exception-
24                 -- handler definiert worden ist.
25             WHEN OTHERS THEN
26                 ...
27
28 END;

```

| Exceptiontyp      | Beschreibung                                                                                                                        | Seite |
|-------------------|-------------------------------------------------------------------------------------------------------------------------------------|-------|
| Standardexception | Standardexceptions sind die durch die PL/SQL Spezifikation definierten internen PL/SQL Exceptions.                                  | 112   |
| Anwendungsfehler  | Anwendungsfehler sind für Anwendungen spezifische Fehlerzustände.                                                                   | 112   |
| Unbenannte Fehler | Die restlichen Fehler, die keinen vordereindefinierten Namen besitzen, liefern nur eine Fehlernummer mit zugehöriger Fehlermeldung. | 112   |

Abbildung 43. Typen von Fehlerobjekten

**12.4.3 Arten von Exceptions**

Die PL/SQL Spezifikation unterscheidet mehrere Formen von Fehlerzuständen.

## » Erklärung: Arten von Exceptions »

- Standardexceptions: Standardexceptions sind Fehlerzustände, die in der PL/SQL Spezifikation definiert werden. Sie beschreiben die grundlegenden in einer Datenbankanwendung auftretenden Fehlerzustände.
- Standardexceptions besitzen eine interne Beschreibung und einen Fehlercode.
- Anwendungsfehler: Anwendungsfehler sind Fehlerzustände, die explizit für jede Datenbankanwendung definiert werden. Sie beschreiben, die für eine Anwendung vorgegebene Beschreibung und einen Fehlercode.
- Anwendungsfehler besitzen eine durch die Anwendung vorgegebene Beschreibung und einen Fehlercode.
- Unbenannte Fehler: Unklassifizierte Fehler besitzen in PL/SQL keinen Namen. Unbenannte interne Exceptions müssen in PL/SQL Programmen über die pragma exception\_init Routine verankert werden.

**12.4.4 Anwendungsfehler**

Anwendungsfehler sind die durch die Geschäftstagsklogik einer Anwendung vorgegebenen Fehlerobjekte.

## » Erklärung: Anwendungsfehler »

- Bevor ein Anwendungsfehler in einem PL/SQL Programm verwendet werden kann, muss er vorher deklariert werden.
- Optional können für ein Fehlerobjekt ein Fehlercode und eine Beschreibung definiert werden.
- Fehlerzustände werden in PL/SQL Programm über die raise Routine ausgelöst.

## » Codebeispiel: ApplicationExceptions »

```

1 -- Beispiel: ApplicationExceptions
2
3 DECLARE
4   -- DECLARATION: Anwendungsexception
5   INCREASE_AMOUNT EXCEPTION;
6   -- INITIALISIERUNG: Anwendungsexception
7   pragma exception_init(
8     INCREASE_AMOUNT, -20999
9   );
10
11 BEGIN
12   -- Anwendungsexception auslösen
13   RAISE INCREASE_AMOUNT;
14   ...
15
16 EXCEPTION
17   -- EXCEPTIONHANDLING
18   WHEN INCREASE_AMOUNT THEN
19   ...
20 END

```

113

| Exceptionname           | Beschreibung                                                                                                                                                                                                  | SQL Code  |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------|
| ACCESS_INTO_NULL        | Program attempted to assign values to the attributes of an uninitialized object.                                                                                                                              | -6530     |
| CASE_NOT_FOUND          | None of the choices in the WHEN clauses of a CASE statement were selected and there is no ELSE clause.                                                                                                        | -6592     |
| COLLECTION_IS_NULL      | Program attempted to apply collection methods other than EXISTS to an uninitialized nested table or varray, or program attempted to assign values to the elements of an uninitialized nested table or varray. | -6531     |
| CURSOR_ALREADY_OPENED   | Program attempted to open an already opened cursor.                                                                                                                                                           | -6511     |
| DUP_VAL_ON_INDEX        | Program attempted to insert duplicate values in a column that is constrained by a unique index.                                                                                                               | -6511     |
| INVALID_CURSOR          | There is an illegal cursor operation.                                                                                                                                                                         | -1001     |
| INVALID_NUMBER          | Conversion of character string to number failed.                                                                                                                                                              | -1722     |
| NO_DATA_FOUND           | Single row SELECT returned no rows or your program referenced a deleted element in a nested table or an uninitialized element in an associative array (Index by table).                                       | 100       |
| PROGRAM_ERROR           | PL/SQL has an internal problem.                                                                                                                                                                               | -6501     |
| ROWTYPE_MISMATCH        | The host cursor variable and PL/SQL cursor variable involved in an assignment have incompatible return types.                                                                                                 | -6504     |
| SELF_IS_NULL            | A program attempts to call a member method, but the instance of the object type has not been initialized.                                                                                                     | 6432      |
| STORAGE_ERROR           | PL/SQL ran out of memory or memory was corrupted.                                                                                                                                                             | -6500     |
| SUBSCRIPT_BEYOND_COUNT  | A program referenced a nested table or varray using an index number larger than the number of elements in the collection.                                                                                     | -6533     |
| SUBSCRIPT_OUTSIDE_LIMIT | A program referenced a nested table or varray element using an index number that is outside the legal range (for example, -1).                                                                                | -6532     |
| SYS_INVALID_ROWID       | The conversion of a character string into a universal rowid failed because the character string does not represent a ROWID value.                                                                             | -1410     |
| TOO_MANY_ROWS           | Single row SELECT returned multiple rows.                                                                                                                                                                     | -1422     |
| TRANSACTION_BACKED_OUT  | The remote portion of a transaction has rolled back.                                                                                                                                                          | ORA-00061 |
| VALUE_ERROR             | An arithmetic, conversion, truncation, or size constraint error occurred.                                                                                                                                     | -6502     |
| ZERO_DIVIDE             | A program attempted to divide a number by zero.                                                                                                                                                               | -1476     |

Abbildung 44. Standardexceptions

#### 12.4.5 Unbenannte Fehler

Unbenannte Fehler sind systeminterne Fehler ohne vor definierten Namen.

» Erklärung: Unbenannte Fehler »

- Fehlerereignisse, die in PL/SQL Programmen nur selten auftreten, wird kein eigener Name zugewiesen.
- Fehler dieser Art werden im *others* Exceptionhandler behandelt.

» Codebeispiel: Unbenannte Fehler »

```

1 -- -----
2 -- Beispiel: Unbenannte Fehler
3 --
4 DECLARE
5   -- TYPEPRAGMA
6   TYPE PR_TABLE_TYPE IS TABLE OF
7     HR_PROJECT_REPORTROWTYPE;
8   --
9   -- Deklaration
10  l_reports PR_TABLE_TYPE := PR_TABLE_TYPE();
11 
12  errtext VARCHAR2(300);
13 
14 BEGIN
15   SELECT l.project_id, l.title, 201
16   BULK COLLECT INTO l_reports
17   FROM HR.l_projects l;
18 
19  FORALL i IN l_reports.FIRST..l_reports.LAST
20    INSERT INTO HR.project_report values
21      l_reports(i);
22 
23 EXCEPTION
24   -- EXCEPTION HANDLER OTHERS: Unbenannte
25   -- Fehler werden in others Exception
26   -- Handler behandelt. Um die Unbenannten
27   -- Fehler unterscheiden zu können wird
28   -- das sqlcode Attribut gegen die
29   -- Nummer des Fehlers geprüft.
30   WHEN OTHERS THEN
31     errtext := sqlerrm(sqlcode);
32 
33   IF sqlcode = -20981 THEN
34     DBMS_OUTPUT.PUT_LINE(
35       'error occurred: ' || errtext
36     );
37   END IF;
38 END;
  
```

#### 12.4.6 Fehlerbehandlung für forall

Die forall Anweisung erlaubt es mehrere DML Anweisungen gekapselt auszuführen.

» Erklärung: Fehlerbehandlung für forall »

- Tritt ein Fehler bei der Ausführung bei einer der DML Anweisungen auf, werden die restlichen Anweisungen trotzdem ausgeführt.
- Die save exceptions Klausel aggregiert Fehlernachrichten die bei der Ausführung des forall Befehls auftreten.

» Codebeispiel: Fehlerbehandlung für forall »

```

1 -- -----
2 -- Beispiel: Fehlerbehandlung fr forall
3 --
4 DECLARE
5   TYPE PR_TABLE_TYPE IS TABLE OF
6     HR_PROJECT_REPORTROWTYPE;
7   --
8   l_reports PR_TABLE_TYPE := PR_TABLE_TYPE();
9   --
10  BEGIN
11    SELECT l.PROJECT_ID, l.TITLE, 201
12    BULK COLLECT INTO l_reports
13    FROM HR.L_PROJECTS l;
14 
15  FORALL i IN l_reports.FIRST..l_reports.LAST
16    SAVE EXCEPTIONS
17    INSERT INTO HR.PROJECT_REPORT VALUES
18      l_reports(i);
19 
20  EXCEPTION
21    WHEN BULK_ERRORS
22    THEN
23      DBMS_OUTPUT.PUT_LINE(
24        'UPDATED ' || SQL%ROWCOUNT || ' rows.'
25      );
26 
27  FOR i IN 1..SQL%BULK_EXCEPTIONS.COUNT
28  LOOP
29    DBMS_OUTPUT.PUT_LINE(
30      'error ' || i || ' occurred '
31      SQLBULK_EXCEPTIONS(i).ERROR_CODE
32    );
33  END LOOP;
34 END;
  
```

115

#### 12.5. Kursor

##### Kontextarea ▾

Als Kontextarea wird jener Bereich des Arbeitsspeichers bezeichnet, der für die Ausführung von SQL Anweisungen reserviert ist.

Bevor die Daten einer Datenbank verarbeitet werden können, müssen Sie in die Kontextarea der PL/SQL Engine geladen werden.

##### 12.5.1 Kursor

Im vereinfachten Sinne stellt ein Kursor eine Referenz auf die Daten der Kontextarea dar.

» Erklärung: Kursor »

- Bei Kurzuren handelt es sich um benannte Datenbankobjekte.
- Kurzuren sind in PL/SQL Programmen instrumental für den Zugriff auf die Daten einer Datenbank. Ein Kursor verwaltet dabei die Datenstrukturen, in denen der Datenbankservice seine Daten bewegt.
- Immer dann wenn ein select oder DML Anweisung ausgeführt wird, erzeugt die PL/SQL Engine einen Cursor, um die Ergebnismenge zu bestimmen bzw. Daten zu speichern.
- Für Kurzuren werden dabei 3 Formen unterschieden: implizite und explizite Kurzuren.
- implizite Kurzuren: Implizite Kurzuren werden bei der Ausführung von SQL Befehlen der PL/SQL Engine erzeugt und verwaltet.
- explizite Kurzuren: Explizite Kurzuren ermöglichen PL/SQL Programmen den Zugriff auf die Daten einer SQL Abfrage. Die Steuerung eines expliziten Kurzuren obliegt dabei zur Gänze dem PL/SQL Programm.
- cursor for Schleife: Mit der cursor for Schleife bietet PL/SQL eine Möglichkeit, auf einfache Weise auf die Kontextarea einer select Anweisung zu zugreifen.
- Die Anzahl gleichzeitig gröffneter Kurzuren wird dabei nicht durch die Datenbank beschränkt.
- Über den OPEN\_CURSORS Parameter kann die Anzahl der gleichzeitig gröffneten Kurzuren einer Datenbank gesteuert werden.

#### 12.5.2 Explizite Cursor

Explizite Kurzuren ermöglichen PL/SQL Programmen den Zugriff auf die Daten einer SQL Abfrage. Die Verwaltung des Kurzuren obliegt dabei zur Gänze dem PL/SQL Programm.

In Zuge der Verarbeitung der Daten durchläuft der Kurzur 4 Phasen:

» Auflistung: Kurzurzugriff »

- Kurzurdeklaration: Bevor ein Kurzur verwendet werden kann muss er deklariert werden.
- Kurzurinitialisierung: Mit der Initialisierung eines Kurzuren werden die mit dem Kurzur assoziierten Daten aus der Datenbank in die Kontextarea geladen.
- Datensatzgriff: Der Zugriff auf die Daten der Kontextarea erfolgt für einen Kurzur Zeile für Zeile.
- Ressourcenfreigabe: Wurden die mit dem Kurzur assoziierten Daten verarbeitet, können die vom Kurzur verwalteten Ressourcen wieder freigegeben werden.

» Codebeispiel: Kurzurzugriff »

```

1 -- -----
2 -- Codebeispiel: expliziter Cursor
3 --
4 DECLARE
5   -- PHASE 1: CURSOR DECLARATION
6   CURSOR l_employees_cur IS
7     SELECT * FROM employees e;
8 
9   l_employee_rec l_employees_cur%ROWTYPE;
10 
11  BEGIN
12    -- PHASE 2: CURSOR INITIALISIERUNG
13    OPEN l_employees_cur;
14 
15    LOOP
16      -- PHASE 3: DATA ACCESS
17      FETCH l_employees_cur INTO
18        l_employee_rec;
19      EXIT WHEN l_employees_cur%NOTFOUND;
20 
21      DBMS_OUTPUT.PUT_LINE(
22        l_employee_rec.EMPLOYEE_ID
23      );
24    END LOOP;
25 
26    -- PHASE 4: RELEASE CURSOR RESOURCES
27    CLOSE l_employees_cur;
28  END;
  
```

116

| Kursorattribut | Beschreibung                                                                                                                                   | Rückgabewert |
|----------------|------------------------------------------------------------------------------------------------------------------------------------------------|--------------|
| FOUND          | Das Attribut speichert den Wert true wenn der vorhergehende fetch Befehl ein Ergebnis zurückgeliefert hat. Andernfalls wird false gespeichert. | Boolean      |
| NOTFOUND       | Das Gegenstück von found                                                                                                                       | Boolean      |
| ISOPEN         | Mit dem Attribut wird geprüft ob der Cursor offen ist.                                                                                         | Boolean      |
| ROWCOUNT       | Gibt die Zahl der Zeilen an die durch die Datenbankoperation bearbeitet werden sind.                                                           | Number       |
| BULK_ROWCOUNT  | Gibt die Zahl der Zeilen an die durch die Datenbankoperation bearbeitet werden sind.                                                           | Number       |
| BULK_EXCEPTION | Gibt an ob bei der Massendatenverarbeitung Fehler aufgetreten sind.                                                                            | Boolean      |

Abbildung 45. Kursorattribute

**12.5.3 Datenverarbeitung**  
Ein Cursor ermöglicht PL/SQL Programmen den Zugriff auf die Daten einer Datenbank.

- Erklärung: Datenverarbeitung
  - Mit der Initialisierung eines Cursors wird das Ergebnis der mit dem Cursor assoziierten SQL Abfrage in die Kontextarea des PL/SQL Engine geladen. Zwischenzeitliche Änderungen am Datenbestand werden jedoch nicht an die Daten der Kontextarea weitergegeben.
  - Nach seiner Initialisierung referenziert ein Cursor den ersten Datensatz der Ergebnismenge der assoziierten Abfrage.
  - Der Zugriff auf die Daten der Kontextarea erfolgt im Cursor Zeile für Zeile. Das Lesen eines Datensatzes aus dem Cursor wird dabei als Fetch bezeichnet.
  - Für den Datenzugriff verwendet die PL/SQL Engine eine Reihe von Statusinformationen. Über Cursorattribute können diese Daten in PL/SQL Programm abfragt werden.

**12.5.4 BULK FETCH Anweisung**

Im Zuge eines Fetch können in den Cursor auch mehrere Datensätze geladen werden.

## » Erklärung: Datenverarbeitung »

```

1 -- -----
2 -- Codebeispiel: Bulk Collect
3 --
4 DECLARE
5   TYPE EMPLOYEE_TABLE_TYPE IS TABLE OF
6     HR.EMPLOYEESTGTYPE;
7
8   CURSOR l_employees_curs IS
9     SELECT * FROM EMPLOYEES;
10
11   l_employees EMPLOYEE_TABLE_TYPE := 
12     EMPLOYEE_TABLE_TYPE();
13
14   BEGIN
15     OPEN l_employees_curs;
16
17     LOOP
18       -- DATES EINLESEN: 100 Datensätze
19       FETCH l_employees_curs BULK COLLECT
20         INTO l_employees LIMIT 100;
21
22       ...
23     END LOOP;
24
25     CLOSE l_employees_curs;
26   END;
  
```

117

117

## Informationssysteme

**13. PL/SQL - Blocktypen****04**

## PLSQL Blöcke

|                                |     |
|--------------------------------|-----|
| 01. Blocktyp: Stored Procedure | 118 |
| 02. Blocktyp: Package          | 121 |
| 03. Blocktyp: Trigger          | 123 |
| 04. Namenskonventionen         | 123 |

**13.1. Blocktyp - Stored Procedure**

Eine Prozedur beschreibt eine freie Abfolge von Anweisungen und Befehlen.

Prozeduren werden zur Organisation von PL/SQL Code in PL/SQL Programmen verwendet.

**13.1.1 PL/SQL Prozedur**

Eine Prozedur ist ein benannter PL/SQL Block, an den Parameter übergeben werden können.

## » Erklärung: PL/SQL Prozedur »

- Für Prozeduren wird dabei zwischen der **Spezifikation** und der **Implementierung** unterschieden. Der Prozederkopf beschreibt die Spezifikation der Prozedur, der Prozederkörper die Implementierung.

## » Syntax: PL/SQL Prozedur »

```

1 -- -----
2 -- Syntax: PL/SQL Prozedur
3 --
4 CREATE [OR REPLACE] PROCEDURE <name>
5   [(Parameter1, [Parameter2, ... ])]
6   IS
7   ...
8   BEGIN
9     Executable statements;
10   END [<name>];
11
12 -- -----
13 -- Codebeispiel: PL/SQL Prozedur
14 --
15 -- SPECIFICATION: update_project
16 CREATE OR REPLACE PROCEDURE update_project(
17   p_project_title IN VARCHAR2,
18   p_project_id IN PLS_INTEGER,
19 )
20 -- IMPLEMENTIERUNG: update_project
21 AS
22   l_project_desc VARCHAR2(4000);
23 BEGIN
24   UPDATE projects
25   SET project_title = p_project_title,
26   WHERE project_id = p_project_id;
27 END update_project;
  
```

118

| Blocktyp | Beschreibung                                                                                                                                                                                             | Seite |
|----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------|
| Procedur | Eine Prozedur beschreibt eine freie Abfolge von Anweisungen und Routinen. Prozeduren werden zur Organisation von PL/SQL Code in PL/SQL Programmen verwendet.                                             | 118   |
| Funktion | PL/SQL Funktionen sind Prozeduren die einen Rückgabewert an den Aufrüter der Funktion zurückgeben.                                                                                                       | 120   |
| Package  | Ein Package hält logisch zusammenhängende Objekte zu einer modularen Einheit zusammen. Packages werden zur Organisation und Strukturierung von PL/SQL Programmen verwendet.                              | 121   |
| Trigger  | Trigger sind eine besondere Form von Prozeduren, die in Reaktion auf ein bestimmtes datenbankinternes Ereignis ausgelöst werden. Trigger werden in Datenbanken zur Wahrung der Datenkonstanz eingesetzt. | 123   |

Abbildung 46. Kursortattribute

**13.1.2 Unterprogrammaufruf**

Der Aufruf einer Prozedur erfolgt durch die Angabe des Prozedurnamens mit den jeweiligen Parameterwerten.

» Erklärung: Unterprogrammaufruf »

- Eine wesentliche Eigenschaft von Prozeduren ist die Möglichkeit sie in komplizierter Form in einer Datenbank zu speichern. Prozeduren liegen damit im zentralen Namensraum einer Datenbank und können aus anderen Blöcken heraus aufgerufen werden.
- Da eine Prozedur kein Ergebnis berechnet, kann sie nicht unmittelbar in einem Ausdruck auftreten. Der Unterprogrammaufruf erfolgt deshalb in Form einer ausführbaren Anweisung.
- Hat eine Prozedur keine Parameter, so kann die Parameterliste auch fehlen. In diesem Fall erfolgt der Aufruf der Prozedur ohne Angabe von Klammern.
- Bei einem Prozeduraufruf wird die Programmabföhrung unterbrochen und die Programmkontrolle an die Prozedur übergeben.
- Lorem ipsum ipsum bla bla bla.

**13.1.3 Parametertyp**

Im einfachsten Fall wird ein Parameter durch einen Namen und eine Datentyp definiert.

» Erklärung: Parametertyp »

- Zusätzlich zum Namen und Datentyp kann Parameter ein Typ zugewandt.
- Parametertypen beschreiben wie sich Parameter in Unterprogrammen verhalten.
- Parametertyp in = Parameter verhalten sich in Unterprogrammen wie initialisierte Konstante. Im Unterprogramm wird der Parameter wie ein konstanter Wert behandelt, der zwar gelesen, dessen Inhalt aber nicht geändert werden kann.
- Parametertyp out = Parameter verhalten sich in Unterprogrammen wie nicht initialisierte Variablen. Der Wert von out Parametern kann im Unterprogramm verändert aber nicht gelesen werden. Typweise werden out Parameter zur Rückgabe von berechneten Werten verwendet.
- Parametertyp in out: Der Parametertyp vereint die Eigenschaften der anderen Parametertypen. in out Parameter können in Unterprogrammen gelesen und geändert werden.



119

## Informationssysteme

| Parameterzuweisung | Beschreibung                                                                                                                                                | Seite |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------|-------|
| positionell        | Bei der positionellen Parameterzuweisung werden die Werte den Parametern in derselben Reihenfolge zugeordnet, in der sie im Unterprogramm definiert wurden. | 120   |
| nominell           | Bei der nominellen Parameterübergabe werden die Parameterwerte über den Parameternamen zugeordnet.                                                          | 120   |

Abbildung 47. Formen der Parameterzuweisung

**13.1.4 Formen der Parameterzuweisung**

Parameter können in PL/SQL Programmen auf unterschiedliche Weise zugewiesen werden: nominell bzw. positionell.

» Erklärung: nominelle Parameterübergabe »

- Werden die Parameter einer Prozedur nominell übergeben, muß beim Aufruf der Prozedur keine Rücksicht auf die Reihenfolge der Parameter mehr genommen werden.
- Der Assoziationsoperator => stellt den Parameternamen zum Aufrufzeitpunkt mit dem Parameterwert in Beziehung.

» Codebeispiel: Parameterübergabe »

```

1 -- -----
2 -- Optionale Parameterübergabe
3 --
4 CREATE OR REPLACE PROCEDURE print_date (
5   p_start_date IN DATE := sysdate,
6   p_day_amount IN NUMBER := 24
7 );
8 
9 call print_return_date(
10   p_start_date => to_date()
11 );

```

**13.1.5 Optionale Parameterzuweisung**

Bei einer Definition kann ein Parameter optional mit einem Vorgabewert belegt werden.

» Codebeispiel: Parameterübergabe »

```

1 -- -----
2 -- Optionale Parameterübergabe
3 --
4 CREATE OR REPLACE PROCEDURE print_date (
5   p_start_date IN DATE := sysdate,
6   p_day_amount IN NUMBER := 24
7 );
8 
9 call print_return_date(
10   p_start_date => to_date()
11 );

```

**13.1.6 PL/SQL Funktion**

Eine Funktion ist ein Unterprogramm das eine Rückgabewert an den Aufrüter der Funktion zurückgibt.

» Syntax: PL/SQL Funktionen »

```

1 -- -----
2 -- Syntax: Funktion
3 --
4 CREATE [OR REPLACE] FUNCTION <name>
5   [<Parameter1>, <Parameter2>, ...]
6   RETURN datatype IS
7   declare variable, constant, etc. here
8   BEGIN
9   executable statements;
10   RETURN (<Return Value>);
11 END [<name>];

```

198

| Blocktyp              | Beschreibung                                                                                         | Seite |
|-----------------------|------------------------------------------------------------------------------------------------------|-------|
| Packagespezifikation  | Die Packagespezifikation beschreibt die öffentliche Schnittstelle eines PL/SQL Packages.             | 121   |
| Packagimplementierung | Die Packagimplementierung enthält die Implementierung der in der Spezifikation deklarierten Objekte. | 122   |

Abbildung 48. Packagekomponenten

**13.2. Blocktyp - Package**

**PL/SQL Package**

Ein Package fasst logisch zusammengehörige Objekte zu einer modularen Einheit zusammen.

PL/SQL Packages werden zur Organisation und Strukturierung von PL/SQL Programmen verwendet.

**13.2.1 PL/SQL Package**

Ein Package fasst Objekte wie Funktionen, Prozeduren, Variablen bzw. Typdeklarationen zu einem Datenbankobjekt zusammen.

Erklärung: **PL/SQL Package**

- Packages werden in kompakter Form in einer Datenbank gespeichert. Damit legen Funktionen und Prozeduren nicht mehr verstreut in einer Datenbank, sondern stehen logisch strukturiert in einem Objekt zur Verfügung.
- Packages werden beim Aufruf eines Packageobjekts komplett in den Arbeitsspeicher geladen und automatisch nachkompiliert. Bei weiteren Aufrufen kann das Objekt direkt verwendet werden.
- PL/SQL Packages werden in 2 Schritten definiert: Packagespezifikation und Packagimplementierung.
- Packagespezifikation: Die Packagespezifikation beschreibt die öffentliche Schnittstelle eines Packages.
- Packagimplementierung: Die Packagimplementierung enthält die Implementierung der in der Spezifikation deklarierten Objekte.

→ Syntax: **Package Spezifikation**

```

1 -- -----
2 -- Syntax: Package Spezifikation
3 --
4 CREATE [OR REPLACE] PACKAGE pkg_name
5 AS
6   Deklaration der Package Elemente
7 END[pkg_name];
8 
9 --
10 -- Codebeispiel: Package Spezifikation
11 --
12 CREATE OR REPLACE PACKAGE tools
13 AS
14   -- DECLARATION: Konstanten und Variablen
15   g_math_pi CONSTANT NUMBER := 3.1456;
16 
17   TYPE PROJECT_TABLE_TYPE IS TABLE OF
18     hr.projects%ROWTYPE;
19 
20   PROCEDURE PRINT_DATE (
21     p_start_date IN DATE := sysdate,
22     p_day IN NUMBER := 24
23   );
24 
25 END tools;
26 
```

121

| Packageartefakt              | Packagekomponente | Beschreibung                                                                                           | Seite |
|------------------------------|-------------------|--------------------------------------------------------------------------------------------------------|-------|
| globale Variable/- Konstante | Spezifikation     | Globale Variablen sind Variablen und Konstanten die sich im globalen Kontext einer Datenbank befinden. | 98    |
| Prozedurspezifikation        | Spezifikation     | Mit einer Prozedurspezifikation befindet sich eine Prozedur im globalen Namensraum einer Datenbank.    | 118   |
| Funktionspezifikation        | Spezifikation     | Mit einer Funktionspezifikation befindet sich eine Funktion im globalen Namensraum einer Datenbank.    | 120   |
| globale Prozedur             | Implementierung   | Globale Prozeduren sind Prozeduren die sich im globalen Kontext einer Datenbank befinden.              | 118   |
| globale Funktion             | Implementierung   | Globale Funktion sind Funktion die sich im globalen Kontext einer Datenbank befinden.                  | 120   |
| lokale Prozedur              | Implementierung   | Lokale Prozeduren sind nur im Kontext eines Packages aufrufbar.                                        | 118   |

Abbildung 49. Packageartefakte

**13.2.3 Packagimplementierung**

Der Packagerörper enthält die Implementierung der in der Spezifikation deklarierten Objekte.

Erklärung: **Packagimplementierung**

- Packagespezifikation und Packagimplementierung werden über den Packagennamen im Bezug gesetzt.
- Die Packagimplementierung kann lokal auch eigene Objekte deklarieren. Diese Objekte können jedoch nur innerhalb der Packagimplementierung referenziert werden.

→ Syntax: **Packagimplementierung**

```

1 -- -----
2 -- Codebeispiel: Packagerörper
3 --
4 CREATE OR REPLACE PACKAGE BODY tools
5 AS
6   -- IMPLEMENTIERUNG: globale Prozedur
7   PROCEDURE PRINT_DATE (
8     p_start_date IN DATE
9   ) AS
10    l_end_time VARCHAR2(25);
11   BEGIN
12    l_end_time := TO_CHAR(
13      NEXT_DAY(
14        p_start_date, 'MON'
15      ), 'DD.MM.YYYY'
16    );
17   END PRINT_DATE;
18 
19   -- DEKLARATION: lokale Prozedur
20   PROCEDURE INITIALIZE AS
21   BEGIN
22    g_std_day_amount := 24;
23   END INITIALIZE;
24 
25 BEGIN
26   INITIALIZE;
27 END tools;
28 
```

122

| Triggerform       | Beschreibung                                                                                                                                                                                                                                                                                  | Seite |
|-------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------|
| Zellentrigger     | Ein Zellentrigger wird für jeden geänderten Datensatz ausgeführt. Zellentrigger werden eingesetzt, wenn die auszuführenden Aktionen vom Inhalt der einzelnen Datensätze abhängen.                                                                                                             | 122   |
| Anweisungstrigger | Im Gegensatz dazu reagiert ein Anweisungstrigger unabhängig von der Anzahl der geänderten Datensätze, auf die durchgeführte DML Anweisung. Beispieleweise kann ein Anweisungstrigger verwendet werden, um die Berechtigung eines Benutzers für die datumsführenden Operationen zu überprüfen. | 123   |

Abbildung 50. Triggerformer

|                                                                                                                                                                                                                                                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <h3>13.3. Blocktyp: Trigger</h3> <p>Trigger sind eine besondere Form von Prozeduren, die in Reaktion auf ein bestimmtes Datenbankereignis ausgelöst werden.</p> <p>Trigger werden in der Datenbank zur Wahrung der Datenkonsistenz eingesetzt.</p> | <h3>13.3.2 Anlegen von Triggern</h3> <p>Trigger werden in kompakter Form in Datenbanken gespeichert. Zum Anlegen eines Triggers wird der CREATE Befehl verwendet.</p> <p>Erklärung: Anlegen von Triggern</p> <ul style="list-style-type: none"><li>■ Im Triggerkopf werden die Eigenschaften eines Triggers definiert.</li><li>■ Der Kopf enthält den Triggernamen, den Zeitpunkt der Aktivierung, den TriggerTyp und die auslösenden Ereignisse.</li><li>■ Ein Trigger kann mehrere Ereignisse gleichzeitig behandeln.</li></ul> <p>Syntax: Anlegen von Triggern</p> <pre>CREATE [OR REPLACE] TRIGGER &lt;trigger_name&gt;     [&lt;event&gt;] ON &lt;table_name&gt;     [&lt;FOR EACH ROW FOR EACH STATEMENT&gt;]     [&lt;when condition&gt;]     [&lt;trigger_definition&gt;]</pre> <p>Erklärung: PL/SQL Trigger</p> <ul style="list-style-type: none"><li>■ Trigger werden in kompakter Form in Datenbanken gespeichert. Wie Indizes oder Constraints sind Trigger von den Tabellen abhängig für sie definiert werden.</li><li>■ Die Aktivierung eines Triggers erfolgt dabei implizit durch die Datenbank, wahlweise vor oder nach dem Aufrufen eines bestimmten Ereignisses.</li><li>■ In Datenbanken werden 2 Formen von Triggern unterschieden: Zeilentrigger und Anweisungstrigger.</li><li>■ Zeilentrigger: Ein Zeilentrigger wird für jeden geänderten Datensatz ausgelöst. Zeilentrigger werden eingesetzt, wenn die auszuführenden Aktionen vom Inhalt der einzelnen Datensätze abhängen.</li><li>■ Anweisungstrigger: Ein Gegenstand dieser reagiert ein Anweisungstrigger unabhängig von der Anzahl der geänderten Datensätze, auf die durchgeführte DML-Anweisung.</li></ul> |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

| Triggerform                  | Beschreibung                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         | Seite |
|------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------|
| <b>SQL Befehle</b>           | Trigger lösen im Rahmen ihrer Logik SQL-Befehle aus. Allerdings reagieren aufgrund der Dateneinschränkungen gewisse Einschränkungen bezüglich der Tabellen die getestet und verändert werden dürfen. Diese Tabellen werden als mutating tables bezeichnet. Für die Daten in mutating Tables dürfen keine SQL-Befehle ausgeführt werden. Die mutating table ist vereinfacht ausgedrückt immer die Tabellen, die einen Trigger auslösen. Handelt es sich um kaskadierende Trigger, existieren mehrere mutating tables. | 121   |
| <b>Transaktionssteuerung</b> | Innenhalb eines Triggers dürfen keine Befehle zur Transaktionssteuerung verwendet werden, da die Ausführung eines Triggers innerhalb der Transaktionskontrolle der auslösende Befehl erfolgt. Dies gewährleistet das Trigger die aktuelle Transaktion nicht beeinflussen oder beeinflusst werden können.                                                                                                                                                                                                             | 121   |
| <b>long Datentyp</b>         | Der long Datentyp darf nicht als VariablenTyp im Deklarations Teil eines Triggers verwendet werden. Attributwerte, die diese Datentypen besitzen, müssen entweder konvertiert werden oder können in sgl Befehlen trotzdem nicht verwendet werden.                                                                                                                                                                                                                                                                    | 121   |

Abbildung 51. Triggerprogrammierung

| 13.3.3 Pseudorecords                                                                                                                                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             | » Codebeispiel: Pseudorecords                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|-----------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p><b>Pseudorecord</b></p> <p>Trigger haben für geänderte Datensätze Zugriff auf die ursprünglichen bzw. neuen Werte der adoptierten Attribute.</p> | <p>» Erklärung: Pseudorecord</p> <ul style="list-style-type: none"> <li>Der Kontext eines Triggers kapselt jene Daten, die durch die auslösende DML-Anweisung verändert wurden.</li> <li>Das Präfix <code>:old</code> referenziert dabei den geänderten und das Präfix <code>:new</code> den aktuellen Attributwert. Der Doppelpunkt dient als Separator zwischen Präfix und Attributnamen.</li> <li>Zur Referenzierung des auslösenden DML-Freigangs kann man in einem Trigger auf die <code>inserting</code>, <code>updating</code> bzw. <code>deleting</code> Bedingungen zugreifen werden.</li> <li>Wurde ein Trigger beispielsweise durch eine <code>update</code>-Operation ausgelöst, evaluiert die <code>updating</code>-Bedingung zu <code>true</code>.</li> </ul> | <pre> 1   -- Codebeispiel: Log Beispiel 2 3   CREATE OR REPLACE TRIGGER tr_project_audit 4     AFTER INSERT OR UPDATE OR DELETE ON 5       projects 6     FOR EACH ROW ENABLE 7     DECLARE 8     BEGIN 9       IF INSERTING THEN 10         INSERT INTO projects_audit VALUES ( 11           :NEW.title, null, 12           user, sysdate, 'INSERTING' 13         ); 14       ELSIF UPDATING THEN 15         INSERT INTO projects_audit VALUES ( 16           :NEW.title, :OLD.title, 17           user, sysdate, 'UPDATING' 18         ); 19       ELSIF DELETING THEN 20         INSERT INTO projects_audit VALUES ( 21           null, :OLD.title, 22           user, sysdate, 'DELETING' 23         ); 24     END IF; 25   END;</pre> |
|                                                                                                                                                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             | U                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |

## 13.3.4 Namenskonventionen

| Datenbankobjekt  | Namenskonvention  | Beispiel                                           | Seite |
|------------------|-------------------|----------------------------------------------------|-------|
| Primary Key      | <name>.pk         | PROJECT_ID, SUBPROJECT_ID, FUNDING_ID              | 76    |
| Unique Key       | <name>.uk         | PROJECT_TITLE_UK                                   | 76    |
| Foreign Key      | <name>.fk         | SUBPROJECT_PROJECT_FK                              | 76    |
| Check Constraint | <name>.chk        | PROJECTS_FUNDING_CHK, PROJECT_RESEARCH_CHK         | 76    |
| Sequence         | <name>.seq        | PROJECT_PROJECT_ID_SEQ, PROJECTS_SUBPROJECT_ID_SEQ | 82    |
| View             | <name>.v          | PROJECT_FUNDING_V, PROJECT_RESEARCH_V              | 78    |
| Type             | <name>.t          | PROJECT_RECORD_T, SUBPROJECT_RECORD_T              | 103   |
| PL/SQL Package   | <name>.plg        | PROJECT_MIGRATION_PKG, LOGGING_PKG                 | 121   |
| PL/SQL Procedure | <name>.prc        | RELEASE_RESOURCE_PRC                               | 118   |
| PL/SQL Function  | <name>.fun        | MIGRATE_PROJECT_DATA_FUN                           | 118   |
| Global Variable  | g_<variable_name> | G_PROJECT_DATA                                     | 103   |
| Local Variable   | l_<variable_name> | L_INDEX                                            | 103   |
| Parameter        | p_<name>          | P_PROJECT                                          | 103   |
| Cursor           | c_<name>          | LC_PROJECTS, LC_SUBPROJECTS                        | 103   |
| Varchar          | v_<name>          | LV_FIRST_NAME, LV_LAST_NAME                        | 103   |
| Record           | r_<name>          | LR_PROJECT                                         | 103   |
| Datentabelle     | <name>            | PROJECTS, SUBPROJECTS, EMPLOYEES, DEPARTMENTS      | 15    |
| Revisionstabelle | v_<name>          | V_PROJECTS, V_SUBPROJECTS, V_EMPLOYEES             | 15    |
| Attributabelle   | e_<name>          | E_PROJECT_TYPE, E_FUNDING_TYPE                     | 15    |
| Schlüsseltabelle | <name>.ft         | PROJECT_EMPLOYEES_FT, PROJECT_FUNDING_FT           | 15    |
| Sammettabelle    | <name>.st         | PROJECTS_ST, EMPLOYEES_ST                          | 15    |
| Basistabelle     | <name>.bt         | PROJECTS_BT, PROJECTS_BT                           | 15    |

Abbildung 52: Namenskonventionen

125

## Informationssysteme

## 14. PL/SQL - SQL Funktionen

## 05

## SQL Funktionen

## 01. SQL Funktionen

126

## 14.1. SQL Funktionen

SQL als Programmiersprache wurde als Sprachschicht für Informationssysteme konzipiert.

## 14.1.1 SQL Funktionen

Mit PL/SQL Funktionen kann die SQL Sprachspezifikation um neue Funktionen erweitert werden.

## » Erklärung: SQL Funktionen anlegen ▾

- Zur Erweiterung der SQL Sprachspezifikation wird die gewünschte Funktionalität in Form von PL/SQL Funktionen definiert.
- SQL Funktionen müssen für ihren Einsatz in einer Datenbank dabei bestimmte Bedingungen erfüllen.
- SQL Funktionen sind **deterministisch**. Als deterministisch wird Vorgänge bezeichnet, die für gleiche Ausgangsbedingungen, zu gleichen Ergebnissen führen.
- Deterministische Funktionen können in einer Datenbank parallel ausgeführt werden.
- Zum Anlegen einer SQL Funktion wird der `create function` Befehl verwendet.

## » Syntax: create function ▾

```

1  -- -----
2  -- Syntax: SQL Funktion
3  --
4  CREATE [OR REPLACE] FUNCTION <name>
5    [(Parameter1, Parameter2, ...)]
6    RETURN <datatype> DETERMINISTIC
7    PARALLEL_ENABLE
8    IS
9      declare variable, constant, etc. here
10   BEGIN
11     executable statements;
12
13     RETURN (Return Value);
14   EXCEPTION
15     WHEN OTHERS THEN
16     ...
17
18 END [<name>];

```

□

126

## SQL Funktion



### 14.1.2 Programmieren von SQL Funktionen

Bei der Programmierung von SQL Funktionen müssen bestimmte Restriktionen eingehalten werden.

#### » Auflistung: Restriktionen »

- globale Kontext: Damit ein PL/SQL Funktion in einer select Abfrage verwendet werden kann muss sie im globalen Kontext der Datenbank verfügbar sein.
- Befehle: In einer SQL Funktionen dürfen keine DDL bzw. DCL Befehle verwendet werden. Damit wird verhindert dass durch das Ausführen der Funktion Nebeneffekte auftreten.
- Parameter: Funktionsparameter in SQL Funktionen müssen einen SQL kompatiblen Datentyp haben. Funktionsparameter dürfen dabei auch keine out Parameter sein.
- null Werte: Tritt ein Fehler bei der Ausführung einer SQL Funktion auf sollte darauf nicht mit dem Werfen einer Exception reagiert werden. Bei der Ausführung einer select Anweisung sollte nie ein Fehler auftreten.

Auf Fehlerergebnisse sollte immer mit der Rückgabe des null Werts reagiert werden.



### 14.1.3 Fallbeispiel: Berechnung der Fakultät

#### » Codebeispiel: Fakultät berechnen »

```

1 -- Fakultät berechnen
2
3 CREATE OR REPLACE FUNCTION FACTORIAL(
4     p_n IN NATURAL
5 )
6 RETURN NUMBER DETERMINISTIC
7 PARALLEL_ENABLE
8 IS
9     l_result NUMBER := 1;
10    NUMERIC_VALUE_EXCEPTION EXCEPTION;
11    pragma exception_init(
12        NUMERIC_VALUE_EXCEPTION, -6502
13    );
14
15    l_n NATURAL RANGE 0 .. 8 := round(p_n);
16
17    BEGIN
18        IF l_n <= 0 THEN
19            l_result := 1;
20        ELSE
21            FOR i = 2 .. p_n LOOP
22                l_result := l_result * i;
23            END LOOP;
24        END IF;
25
26        RETURN l_result;
27    EXCEPTION
28        WHEN numeric_value_exception THEN
29            RETURN null;
30    END FACTORIAL;

```

127

# NoSQL - Grundlagen und Prinzipien

August 19, 2020

Informationssysteme

**15. NoSQL - Prinzipien**

**01**

NoSQL - Prinzipien

|                           |     |
|---------------------------|-----|
| 01. Bigdata               | 130 |
| 02. Verteilte Datenbanken | 132 |
| 03. NoSQL Datenbanken     | 133 |

**15.1. Bigdata**

**Big Data**

Der Begriff Big Data beschreibt **Datenbestände**, die aufgrund ihres Umfangs, ihrer Struktur bzw. ihrer Schnellfließigkeit nur begrenzt oder gar nicht mit relationalen Datenbanken verarbeitet werden können.

**15.1.1 Grundlagen**

Das Datenvolumen in **Datenverarbeitungssystemen**<sup>3</sup> verdoppelt sich nach aktuellen Berechnungen alle 2 Jahre.

→ Erklärung: **Bigdata**

- Der enorme **Datenaufschwung** in Wirtschaft, Forschung und privatem Umfeld ergibt sich aus der **Digitalisierung** von Inhalten in digitalen Verwaltung-, Steuer- und Kommunikationssystemen.
- In sozialen Netzwerken, der Finanzindustrie bzw. dem Gesundheitswesen entstehen neue Anwendungs- und Geschäftsfelder die mit großen Datenummengen arbeiten müssen.

Auflistung: **Anwendungsbereiche**

- Datenvolumen: Das **enorme** Datenvolumen das in sozialen Netzwerken generiert werden muss übersteigt in der Regel die Möglichkeiten relationaler Datenbanken.
- Datenverarbeitung: Die Notwendigkeit von Unternehmen **Echtzeitanswendungen** von Websstatistiken in diversen Geschäftsfeldern durchzuführen erfordert neue Methoden der **Datenverarbeitung**.

rechteitig

<sup>3</sup> Informationssysteme

z.B.: Meta (WhatsApp Nachrichten,...)  
 viele Daten bekommen, nicht gleich verarbeiten  
 müssen; irgendwann verarbeiten müssen

## Big Data



### 15.1.2 Herausforderungen im Bigdata

Die grundlegenden technischen Herausforderungen im Bigdata Umfeld werden abstrahiert als die sogenannten 3V - Volume, Velocity und Variety.

#### ► Auflistung: Herausforderungen Big Data

| Volume                                                                                            |
|---------------------------------------------------------------------------------------------------|
| Mit Volume ist die schiere Menge an Daten gemeint, die pro Zeiteinheit verarbeitet werden müssen. |
| Velocity                                                                                          |
| Velocity beschreibt die Geschwindigkeit, mit der Daten verarbeitet werden müssen.                 |

Zu den Herausforderungen von Big Data zählt nicht nur die Speicherung von großen Datenmengen sondern gleichfalls die Extraktion von Wissen aus Daten.

Das Ziel von Datenextraktion ist die effiziente und schnelle Analyse von Daten zu verwertbarer Information.

### 15.1.3 Technologien im Bigdata Umfeld

Der Begriff **Bigdata** stammt aus dem englischen Sprachraum. Erst als Phänomen bzw. Hypo wahrgenommen, fassen IT Experten zwischenzitell unter dem Begriff 2 Aspekte zusammen.

#### ► Auflistung: Aspekte des Bigdata

- Datennmenge: Der Begriff Bigdata wird verwendet um große Datennmenge in der IT zu beschreiben.
- Technologie: Bigdata steht gleichzeitig für leistungstarke IT-Lösungen und Systeme, mit denen große Datennmenge verarbeitet werden können.

#### ► Auflistung: NoSQL Systeme

- NoSQL Systeme sind Informationsysteme die entwickelt wurden um den Herausforderungen des Bigdata zu begegnen. NoSQL ist ein Acronym und steht für not SQL usw.

#### ► Erklärung: NoSQL Systeme

- Hinter NoSQL Systemen steht nicht ein einzelnes Technologiekonzept wie bei relationalen Datenbanken. Vielmehr wird mit dem Begriff NoSQL eine Vielzahl von unterschiedlichen Technologien verstanden.
- Wichtig ist hierbei, daß diese Technologien mit klassischen Datenbanken in Einklang gebracht werden können. Nur so kann die Konsistenz der Daten über Systemgrenzen hinweg gewährleistet werden.

15.1

Informationssysteme

### 15.2. Verteilte Datenbanken

Datenverarbeitungssysteme verwalten in der Regel ihre Daten in einer einzelnen Datenbank.



#### 15.2.1 Grenzen konventioneller Datenbanken

Datenverarbeitungssysteme sehen sich jedoch immer öfter damit konfrontiert Datenbestände im Bigdata Bereich verarbeiten zu müssen.

##### ► Analyse: Grenzen konv. Datenbanken

- Das Volumen der zu speichernden Daten übersteigt die Speicherkapazität der Datenbank.
- Die Anzahl der Anwender ist so groß, daß die Antwortzeit des Datenverarbeitungssystems nicht mehr akzeptabel sind.
- Bei der Zugriffsbelastung ist gleichzeitig zu beachten, daß Anwender oft zu bestimmten Zeiten enorme Lastspitzen erzeugen, so daß sich extreme Anforderungen an die Skalierbarkeit solcher Systeme ergeben.
- Um Datenverlust zu verhindern sollten Daten nicht nur an einem einzelnen Server gespeichert werden. Denn fällt der Server aus, dann können die auf den Daten basierenden Geschäftsprozesse nicht mehr ihren Dienst verrichten.



#### 15.2.2 Verteilte Datenbank

##### Verteilte Datenbank

Ein **Verteiltes Datenbanksystem** ist ein Informationssystem, das auf mehreren Netzwerkknoten ausgeführt werden kann. Die Daten der Datenbank werden in diesem Fall **verteilt** über mehrere Knoten hinweg verwaltet.

Verteilte Datenhaltungssysteme bestehen aus mehreren physisch voneinander getrennten Daten-

banken, die unabhängig voneinander arbeiten, jedoch nach außen wie ein einziges logisches System erscheinen.

#### ► Eigenschaften: Verteilte Datenbanksysteme

- Availability: Verteilte Datenbanksysteme weisen eine hohe Verfügbarkeit auf.
- Ausfallsicherheit: Verteilte Datenbanksysteme besitzen eine hohe Ausfallsicherheit.
- Fällt ein Netzwerkknoten aus, auf dem eine Instanz der verteilten Datenbank ausgeführt wurde, ist nur ein Teil des Datenbestands des Systems betroffen.
- Lokale Strukturierung: Verteilte Datenbanksysteme unterstützen die dezentrale Organisation und Unternehmensstrukturen moderner Unternehmen.
- Gerade in Unternehmen mit vielen, unterschiedlichen Standorten, ist es sinnvoll Daten ihrer Art und Nutzung nach, auf mehrere Instanzen der Datenbanken zu verteilen. So liegen an jedem Standort zum Beispiel genau die Daten, die dort benötigt werden.
- Semantische Strukturierung: Bevor die Daten in einer verteilten Datenbank gespeichert werden können, müssen sie zuerst auf sogenannte Shards aufgeteilt werden.

Eine verteilte Datenbank wird z.B. verwendet um die Angestellendaten eines Unternehmens zu verwalten. Dabei wird für jedes Land in dem das Unternehmen eine Niederlassung besitzt eine einzelne Instanz der Datenbank betrieben. Auf jedem Knoten werden dabei nur jene Angestellten gespeichert die in der entsprechenden Niederlassung arbeiten.

./image/gear21.jpg

15.2

## NotOnlySQL

### 15.3. NoSQL Datenbanken

NoSQL Datenbanken wurden aus der Notwendigkeit heraus entwickelt, große Datenvolumen zu speichern bzw. zu verarbeiten.



#### 15.3.1 NoSQL Grundlagen

##### NoSQL Systeme

NoSQL Informationssysteme kein einheitliches Datenmodell.

Bevor wir Daten in eine relationale Datenbank eingeben können, muß die Struktur der Tabellen definiert werden. Im Vergleich dazu müssen für eine NoSQL Datenbank in der Regel keine Strukturen definiert bzw. Vorgaben gestellt werden. (die Daten werden einfach gespeichert).

Konsistenz vs. Eventual Consistency: Relationale Datenbanken streben vor allem nach Konsistenz. NoSQL Systeme **optieren Konsistenz** um verfügbar

**15.3.1 NoSQL Grundlagen**

NoSQL Systeme →

NoSQL Systeme sind **Informationsysteme**, die entwickelt wurden, um den Herausforderungen des Bigdata zu begegnen. NoSQL ist ein Acronym und steht für **not only SQL**.

NoSQL Systeme haben gemeinsam, dass sie für Anwendungsfelder optimiert sind, für die das relationale Datenmodell an seine Grenzen stößt.

Erklärung: NoSQL Datenbanken →

- Das Acronym NoSQL steht für den englischen Begriff **Not only SQL**.
- Bei NoSQL Informationsystemen erkennen wir eine **Ablösung von starc vorgegebenen Datenstrukturen**, wie wir sie in relationalen Systemen vorfinden.
- Gleichzeitig besitzen NoSQL Datenverarbeitungssysteme ein eigenes Konsistenzmodell, das das Konsistenzverständnis von RDBMS Systemen vereinfacht.
- NoSQL Systeme sind **keine Erste für relationale Datenbanksysteme**.
- NoSQL Systeme wurden für spezifische Problemlösungen entwickelt und stellen damit eine **Ergänzung** zu relationalen Datenbanken dar.

15.3.2 RDB vs. NoSQL

Zum besseren Verständnis von NoSQL Systemen wollen wir NoSQL Systeme mit relationalen Datenbanken vergleichen.

Analyse: Vergleich - RDB vs. NoSQL →

- SQL Schema vs. NoSQL schematische Währung relationaler Datenbanken Tabelle mit Spalten und Zeilen für die Datenspeicherung verwenden, kennen

16. NoSQL - CAP Theorem

# 03

CAP Theorem

|                 |     |
|-----------------|-----|
| 01. CAP Theorem | 134 |
| 02. CA Systeme  | 136 |
| 03. AP Systeme  | 138 |
| 04. CP Systeme  | 138 |

16.1. CAP Theorem

CAP Theorem →

Das CAP Theorem zeigt, dass in einem verteilten Datenbanksystem maximal zwei der folgenden drei **Eigenschaften** gleichzeitig gewährleistet sein können.

- Consistency**: Konsistenz bedeutet, dass alle Anwender zum selben Zeitpunkt die gleichen Daten sehen.
- Availability**: Verfügbarkeit bedeutet, dass Anwender jederzeit Lese- bzw. Schreibzugriffe durchführen können.
- Partition Tolerance**: Partition Tolerance bedeutet, dass die Datenbank auch beim Ausfall einzelner Knoten als Ganzes weiterarbeiten kann.

Das CAP Theorem teilt Informationsysteme ihrer Technologie entsprechend in Kategorien. Damit kann der Anwendungsentwickler je nach Anforderung das entsprechende Informationssystem wählen.

16.1.1 Eigenschaften von Datenbanken

Auflistung: Eigenschaften →

|                     |                                                                                                                                                             |
|---------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Consistency         | In verteilten Datenbanken muss sichergestellt sein, dass nach Abschluss einer Transaktion alle Replikate eines veränderten Datensatzes aktualisiert wurden. |
| Availability        | Verfügbarkeit beschreibt in erster Linie die Eigenschaft eines Informationssystems bei hoher Systemlast seinen Dienst aufrechterhalten zu können.           |
| Partition Tolerance | Partition Tolerance bedeutet, dass das System auch beim Ausfall einzelner Knoten als Ganzes weiter arbeiten kann.                                           |

Dipl.-Ing.- Msc. Paul Panhofer Bac.

Abbildung 53: CAP Theorem

16.1.2 Kategorien von Informationssystemen

Mit dem CAP Theorem wird eine Kategorisierung von Informationssystemen definiert.

Jedes Informationssystem lässt sich immer einer, die durch das CAP Theorem beschriebenen Kategorie zuordnen.

16.1.3 Achsenausprägungen

Jedes Informationssystem kann ein Punkt auf einer der 3 Achsen des CAP Theorems eingezeichnet werden.

## 16.1.2 Kategorien von Informationssystemen

Mit dem CAP Theorem wird eine Kategorisierung von Informationssystemen definiert.

Jedes Informationssystem lässt sich immer einer, den durch das CAP Theorem beschriebenen Kategorien zuordnen.

### Erklärung: Kategorien von IFS \*

- Durch das CAP Theorem werden 3 Kategorien für Informationssysteme definiert: **CA Systeme**, **AP Systeme** und **CP Systeme**.

Jedes Informationssystem lässt sich genau einer, den durch das CAP Theorem definierte Kategorien zuordnen. Die durch das CAP Theorem definierten Kategorien werden auch als die Achsen des CAP Theorems bezeichnet.

■ Wird ein Informationssystem beispielsweise den CA Systemen zugeordnet, kann es graphisch als Punkt auf der CA Achse visualisiert werden.

■ Die Systemeigenschaften Konsistenz, Verfügbarkeit und Ausfallsicherheit sind dabei als graduelle Größen zu verstehen. Durch die Visualisierung des Informationssystems auf einer der 3 Achsen kann die Stärke der Ausprägung der Systemeigenschaft für das System angegeben werden.

■ Wird ein Informationssystem auf der AP Achse nahe des Availability Fixpoints abgedeutet, handelt es sich um ein System das in erster Linie verfügbar zu sein hat.

## 16.1.3 Achsenausprägungen

Jedes Informationssystem kann als Punkt auf einer der 3 Achsen des CAP Theorems eingetragen werden.



### Analyse: Achsenausprägung \*

- Die Nähe des Eintrags zu einem der 3 Fixpunkte des Systems, definiert die Ausprägung der Eigenschaft im Informationssystem.

■ Ausprägung Verfügbarkeit: Die Verfügbarkeit ist hoch, wenn das System schnell antwortet, und gering, wenn das System eine hohe Latenzzeit hat.

■ Ausprägung Konsistenz: In Hinblick auf die Konsistenz bedeutet das, dass diese entweder sofort sichergestellt ist oder erst nach einem gewissen Zeitraum der Inkonsistenz.

■ Ausprägung Partition Tolerance: Ein System mit hoher Partition Tolerance wird als Peer to Peer System bezeichnet. Jeder der Knoten des Systems arbeitet als autonome Einheit. Systeme mit niedriger Partition Tolerance funktionieren nur wenn alle Knoten des Systems erreichbar sind.



135

## Informationssysteme

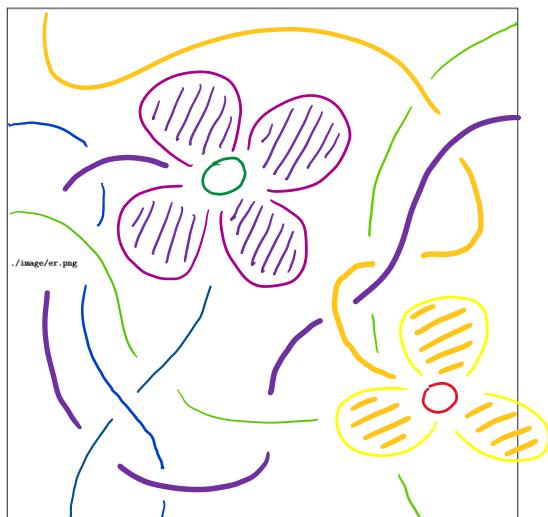


Abbildung 54. Datenstruktur - Relationale Datenbanken

## 16.2. CA Systeme

SQL Systeme werden zu den CA Systemen gezählt.

### 16.2.1 Theoretische Grundlagen

CA Systeme streben vor allem nach Konsistenz.

### Analyse: CA Eigenschaften \*

- CA Systeme garantieren vor allem eine Konsistenz.

■ Grundlage dieser Systeme ist das **ACID Transaktionsmodell**. Zugriffe sind stets atomar, konsistent, isoliert und dauerhaft.

■ Mit der Zahl steigender Netzwerknoten sinkt die Verfügbarkeit verteilter Datenbanken aufgrund der strengen Konsistenzbedingungen.

■ Bei CA Systemen sprechen wir in der Regel nicht

136

von verteilten Systemen - in einem Netzwerken mit einem einzelnen Knoten kommt es zu keiner Netzwerkpartitionierung.

*relationale Datenbanken*

### 16.2.2 Beispiele für CA Systeme

- Die wichtigste Klasse der CA Systeme sind die relationalen Datenbanken.
- Analyse: Relationale Datenbanken ▶
  - Die theoretische Grundlage Relationaler Datenbanken ist das relationale Datenbankmodell.
  - RDBs sind die zur Zeit am stärksten verbreitetsten Datenbanksysteme der IT.
  - RDBs arbeiten mit stark strukturierten<sup>4</sup> Daten.
  - Relationale Datenbanken
    - ▶ MySQL Community Server 8.0.11
    - ▶ Microsoft SQL Server 2017
    - ▶ Oracle Database 12.02

<sup>4</sup> Die Struktur der Daten wird durch das System vorgeschrieben und muß beim Arbeiten mit den Daten eingehalten werden

137

### 16.3. AP Systeme

AP Systeme werden zu den sogenannten NoSQL Systemen gezählt.

#### 16.3.1 Theoretische Grundlagen

AP Systeme operieren Konsistenz um verfügbar und ausfalltolerant zu sein.

► Analyse: Eigenschaften ▶

- AP Systeme reagieren auf Netzwerkpartitionierung, indem sie auf vollständige Konsistenz verzichten.
- Alle Teile des Netzwerks reagieren auf Anfragen, aber die Antworten sind nicht notwendigerweise konsistent.
- Konnte der Fehler im Netzwerk behoben werden wird die Konsistenz für alle Netzwerknoten wieder hergestellt.
- AP Systeme folgen dem BASE Transaktionsmodell.

#### 16.3.2 Beispiele für AP Systeme

AP Systeme kommen im Big Data Bereich zum Einsatz.

► Erklärung: Big Data Anwendungen ▶

- Big Data Anwendungen sind in der Lage große Mengen von Daten in kurzer Zeit zu verarbeiten. Die Anwendung ist dabei über mehrere Knoten eines Netzwerks verteilt.
- Denken Sie in diesem Zusammenhang an Anwendungen wie Facebook (Cassandra) oder Reddit (Rak).

► Erklärung: DNS Server ▶

- DNS Server fallen in die Kategorie der AP Systeme.
- Die Verfügbarkeit von DNS Servern ist sehr hoch.
- DNS Systeme verfügen im gleichen Maße über eine hohe Ausfalltoleranz.
- Allerdings ist die Konsistenz nicht immer sofort gegeben: es dauert in der Regel länger, bis ein geänderter DNS Eintrag an die gesamte DNS Hierarchie propagierte werden kann.

### 16.4. CP Systeme

CP Systeme werden zu den sogenannten NoSQL Systemen gezählt.

#### 16.4.1 Theoretische Grundlagen

CP Systeme reagieren auf eine Netzwerkpartitionierung, indem sie auf durchgehende Verfügbarkeit verzichten.

► Erklärung: Eigenschaften ▶

- Das Prinzip von CP Systemen ist es jenen Teil des Netzwerks, der konsistent gehalten werden kann, verfügbar zu halten, während der Rest ignoriert wird.
- CP Systeme sind konsistent und ausfalltolerant.
- Sobald die Verbindungen zwischen den Knoten wiederhergestellt ist, wird bei den zuvor nicht verfügbaren Knoten zunächst die Konsistenz wiederhergestellt, erst dann antworten das System wieder auf Anfragen.
- CP Systeme folgen dem BASE Transaktionsprinzip.



#### 16.4.2 Beispiele für CP Systeme

CP Systeme kommen im Big Data Bereich zum Einsatz.



► Erklärung: Big Data Anwendungen ▶

- Big Data Anwendungen sind in der Lage große Mengen von Daten in kurzer Zeit zu verarbeiten. Die Anwendung ist dabei über mehrere Knoten eines Netzwerks verteilt.
- Bekannte NoSQL Datenbanken sind Google BigTable, Apache HBase, MongoDB, Redis, BerkleyDB.

## 17. NoSQL - Technologien

# 04

### NoSQL - Technologien

|                            |     |
|----------------------------|-----|
| 01. Konsistenzmodelle      | 139 |
| 02. Transaktionsattribute  | 140 |
| 03. MVCC                   | 141 |
| 04. Consistent Hashing     | 143 |
| 05. Map Reduce Algorithmen | 144 |

### 17.1. Konsistenzmodelle

Konsistenz beschreibt die Richtigkeit von Daten in einem Informationssystem.

- Konsistenzmodell** Das Konsistenzmodell einer Datenbank beschreibt auf welche Weise die Konsistenz der Daten sichergestellt wird.

#### Erklärung: Konsistenz

- Kann die Konsistenz von Daten in einem Informationssystem nicht garantiert werden, verlieren die Daten des Informationssystems in der Regel ihren Wert.

- Konsistenzmodelle beschreiben wie die Konsistenz der Daten des Informationssystems sichergestellt werden kann.

#### Auflistung: Arten von Konsistenzmodellen

- Strict Consistency** Alle Anwender sehen immer dieselben Daten. Alle Leseoperationen liefern den Wert, der die letzte abgeschlossene Schreiboperation geschrieben hat.

- Eventual Consistency** Anwender sehen nicht immer dieselben Daten. Eine Leseoperation liefert **eventuell** den zuletzt geschriebenen Wert.

### 17.2. Transaktionskonzepte

## Transaktion



#### Konsistenz

Nach dem Ausführen einer Transaktion muss der Datenbestand des Informationssystems in einem konsistenten Zustand sein.

- Isolation Bei der gleichzeitigen Ausführung mehrerer Transaktionen dürfen sich diese gegenseitig nicht beeinflussen.

- Durcharbeitigkigkeit Die Auswirkungen einer Transaktion müssen im Datenbestand dauerhaft bestehen bleiben. Die Effekte von Transaktionen dürfen also nicht verloren gehen oder mit der Zeit verblasen.

#### 17.2.2 Sperren

Eine Sperrre ermöglicht den exklusiven Zugriff eines Prozesses auf eine Ressource. Sperren garantieren das nur ein Prozess eine Ressource liest oder verändert, solange die Sperrre besteht.

- Strict Consistency Konsistenzmodelle verwenden Sperren um das ACID Transaktionskonzept umzusetzen.**

#### Analyse: Sperren

- Die Verwendung von Sperren ist in mehrfacher Hinsicht problematisch.

- Der Einsatz von Sperren kann in einer Datenbank zu langen Wartezeiten führen.

- Besonders in verteilten Datenbanken mit replizierten Daten verlängert sich der Zeitraum einer Transaktion beim Einsatz von Sperren, da zusätzlich eine Sperrung aller replizierter Datensätze zu erfolgen hat.

### 17.2.3 BASE Transaktionskonzept

Das Akronym **BASE** steht für **Basic Availability**, **Soft State** und **Eventually Consistent**.

Die größten Anforderungen an Anwendungen im Bigdata Bereich motivieren einen neuen Konsistenzbegriff der grundsätzlich ohne Sperren auskommt. Wir beschreiben eine Datenbank als Eventual Consistent wenn das Transaktionsmodell der Datenbank die BASE Eigenschaften erfüllt.

#### » Auflistung: BASE Eigenschaften »

**Basic Availability** ▾  
Das Datenbanksystem ist im Sinne des CAP Theorems grundsätzlich immer verfügbar.

**Soft State** ▾  
Die Daten des Datenbanksystems befinden sich zeitweise in einem **inkonsistenten Zustand**.

**Eventually Consistent** ▾  
Nach dem Verständnis von **BASE** wird die **Konsistenz** der Daten als ein Zustand betrachtet, der irgendwann erreicht wird.  
Es wird in Kauf genommen, dass die Daten eine gewisse Zeit inkonsistent sind.

### 17.2.4 Optimistic Locking

**Optimistic Locking** ▾  
Unter **Optimistic Locking** versteht man eine Sammlung von Verfahren, die die **Konsistenz** von Daten sicherstellen, **ohne Sperren zu verwenden**.

#### » Auflistung: Optimistic Locking Verfahren »

▪ Multi Version Concurrency Control

▪ Consistent Hashing

### 17.3. MVCC

MultiVersion Concurrency Control ist ein Verfahren für die **simultane, parallele Verarbeitung von Daten**. MVCC verzichtet dabei in Gang auf Sperren.

#### 17.3.1 MVCC Verfahren

MVCC wird verwendet um **Strong Consistency** ohne den Einsatz von Sperren umzusetzen.

#### » Erklärung: MVCC Verfahren »

- Jeder **Datensatz** des Datenbestands wird mit einer **Transactions ID** versehen.
- Jedemal wenn ein **schreibender Zugriff** auf einen Datensatz innerhalb einer Transaktion erfolgt, wird die Transaktions ID des Datensatzes **um 1 erhöht**.
- Bevor ein schreibender Zugriff auf einen Datensatz erfolgen kann wird die Transaktions ID der Kopie des Datensatzes, die durch die Transaktion im Speicher gehalten wird, mit der Kopie des Datensatzes in der Datenbank verglichen.
- Nur wenn die Transaktions ID beider Datensätze identisch ist, erfolgt der schreibende Zugriff. Andernfalls erfolgt ein Rollback innerhalb der Transaktion.
- Mit MVCC wird sichergestellt, dass **2 Transaktionen** nicht gleichzeitig schreibenden Zugriff auf den selben Datensatz haben können.



141

Informationssysteme

./image/mvcc.png

Abbildung 55. mvcc - Schreiben eines Datensatzes

### 17.3.2 Diskussion MVCC

MVCC hat sich im Bereich der NoSQL Systeme zu einer Basistechnologie entwickelt, die es ermöglicht, konkurrierende Zugriffe ohne das sperren von Datensätzen, zu koordinieren.



#### » Erklärung: Diskussion MVCC »

- Für Anwendungen im Big Data Bereich wird implizit vorausgesetzt, dass das Datenbanksystem auf einem verteilten System ausgeführt wird.
- In diesem Umfeld erweist sich der relationale Transaktionsansatz mit dem sperren von Datensätzen als **Haupthindernis**.

- Zum einen ist der **Zeit- und Kommunikationsaufwand** für eine Übereinkunft, für das Setzen bzw. Aufheben einer Sperrre immens, zum anderen kann

142

- auf einen gesperrten Datensatz nicht lesend zugriffen werden.
- Für genau solche Szenarien wurde das MVCC Verfahren entwickelt. Die Konsistenz der Daten kann ohne aufwendige Sperrmechanismen sichergestellt werden.

**17.4. Consistent Hashing**

**Consistent Hashing**

Consistent Hashing ist ein Verfahren zur Verteilung von Datensätzen in einem verteilten System.

**17.4.1 Motivation**

Anwendungen im Bigdata Bereich müssen in der Lage sein, Anfragen hunderttausender von Benutzern gleichzeitig zu bearbeiten.

## » Erklärung: Motivation Consistent Hashing

- Um die Anzahl von Anfragen erhöhen zu können die ein System bearbeiten kann, werden unterschiedliche Caching Verfahren verwendet.
- Horizontale Skalierende Caching Architektur verteilen die Last Benutzeranfragen auf mehrere Servern.
- Damit muss jedoch ein neues Problem adressiert werden: Wie können die Daten auf die Server sinnvoll verteilt werden.
- Eine mögliche Lösung wäre es eine Hash Funktion zu verwenden.



143

**17.4.2 Verteilen von Daten**

- Die einfachste, aber auch naive Lösung ist es eine Hashfunktion zu verwenden.
- Für jeden Schlüssel eines zu speichernden Datensatzes wird eine Hashfunktion angewandt, die möglichst gleichmäßig verteilte Hasches generiert. Je nach Hash wird der Datensatz auf einem anderen Server gespeichert.
- Als Hashfunktion bieten sich dabei schnelle Algorithmen wie MD5 oder CRC32 an.
- Doch was passiert, wenn ein Server ausfällt oder ein neuer Server hinzugefügt werden muss. Ein großer Teil der gespeicherten Schlüssel muss neu aufgeteilt werden.

**17.5. Map Reduce Algorithmus**

**Map Reduce**

Map Reduce ist ein Programmiermodell für rechenintensive Berechnungen von großen Datensätzen auf Computerclustern.

**17.5.1 Algorithmusbeschreibung**

Beim Map Reduce Verfahren werden die Daten in 3 Phasen - Map, Shuffle, Reduce - verarbeitet.

- » Erklärung: Algorithmusbeschreibung
- Ursprünglich wurde das Map Reduce Verfahren von Google für das Indexieren von Webseiten entwickelt.
- Das Verfahren eignet sich für die Verarbeitung von großen Datensätzen. Die Daten können dabei strukturiert als auch unstrukturiert sein.
- Bei großen Datensätzen einzelne Prozesse schnell an ihre Grenzen stoßen, parallelisiert Map Reduce die Bearbeitung, durch die Verteilung auf mehrere gleichzeitig auszuführende Tasks.
- Das Map Reduce Framework sorgt für die Verteilung der Aufgaben auf unterschiedliche Rechner oder Cluster und aggregiert anschließend die Ergebnisse.
- Der Map Reduce Algorithmus besteht aus 3 Phasen - Map, Shuffle und Reduce.
- Die Grundfunktionen des Map Reduce Ansatzes sind die beiden Funktionen `map()` und `reduce()`. Sie sorgen für die Zerlegung der Aufgabe in kleinere parallelisierte Arbeitsschritte und führen die Ergebnisse anschließend zusammen.
- Map stellt die Funktion dar, die die Aufgaben an die unterschiedlichen Knoten eines Clusters verteilt. Die Reduce Funktion sortiert die berechneten Ergebnisse und fügt sie wieder zusammen.



144

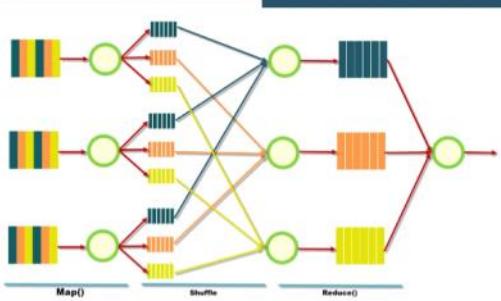


Abbildung 56. Map Reduce Algorithmus

## 17.5.2 Phasenbeschreibung

## » Auflösung: Map Reduce Phasen »

- Map Phase: In der Map Phase werden die **Eingabedaten** auf eine Menge von Map Prozessen verteilt, welche jeweils die vom Nutzer bereitgestellte Map Funktion berechnen.
- Map Prozesse werden idealerweise parallel ausgeführt.
- Shuffle Phase: Die **Zwischenergebnisse** der Map Phase werden gemäß der Ausgabeschlüssel, die von der Map Funktion produziert wurden, **sortiert**, sodass alle Zwischenergebnisse mit demselben Schlüssel im nächsten Schritt auf demselben Computerystem verarbeitet werden.
- Reduce Phase: Für jeden Satz von **Zwischenergebnissen** berechnet jeweils genau ein Reduce Prozess die vom Nutzer bereitgestellte Reduce Funktion und damit die Ausgabedaten.
- Reduce Prozesse werden idealerweise ebenfalls parallel ausgeführt.

## 17.5.3 Diskussion MapReduce

## » Analyse: Map Reduce »

- MapReduce wird im BigData Umfeld für beispiele wie **Finanzanalysen**, wissenschaftliche Simulation oder **Data Mining** verwendet.
- Gegenüber den klassischen Verfahren der Datenerarbeitung, wie sie in relationalen Datenbanken zum Einsatz kommen, bietet MapReduce eine ganze Reihe von Vorteilen.
- Durch die **parallele Verarbeitung** ist MapReduce sehr **schnell** und **wesentlich performanter** als die Datenerarbeitung in relationalen Datenbanken.
- Das **Verfahren** eignet sich sehr gut für die **Batch Verarbeitung** von großen unstrukturierten, semistrukturierten und nicht normalisierten Daten.
- Für das **MapReduce Konzept** existieren mittlerweile viele **Implementierungen**. Eine der erfolgreichsten und am stärksten verbreiteten ist das **Hadoop Framework**.

145

# MongoDB - Dokumentorientierte Datenbanken

August 19, 2020

## 18. MongoDB - Modellierung

# 01

### MongoDB - Modellierung

01. MongoDB

148

02. Modellierungsprinzipien

150

## 18.1. MongoDB

MongoDB ist ein populäres **Informationsystem im Bigdata Bereich**

### 18.1.1 MongoDB Motivation

Relationale Datenbanken wurden jahrzehntelang als einzige Möglichkeit zur Archivierung und Verwaltung elektronischer Daten im IT Bereich angesehen.

- Spricht man von herkömmlichen Datenbanken, meint man zunächst die vom britischen Mathematiker Edgar F. Codd entwickelten relationalen Datenbanken.
- In Tabellen, die üblicherweise aus mehreren Zeilen und Spalten bestehen, werden Werte gespeichert. Die Spalten bieten Platz für Attribute wie Vornamen oder Nachnamen. In den Zeilen werden diese dann mit konkreten Werten gefüllt.
- Dieser Ansatz hat jedoch einen Nachteil: Das Schema muss von Anfang an feststehen. Es muss also beim Anlegen einer Tabelle bereits festgelegt werden, wie viele und welche Spalten es geben wird.
- Solche und ähnliche Fragen sind für einfache Anwendungen noch schnell zu lösen sein, in komplexeren Projekten kann dieser **starre Ansatz** jedoch ebenso schnell in eine **Sackgasse** führen. Häufig lassen sich Millionen unterschiedlicher Datensätze nicht in ein einziges starreres Schema zwängen.
- Wesentlich signifikanter liegt jedoch der **Performanceverlust**, der bei relationalen Datenbanken unweigerlich eintritt, je mehr Daten in der Datenbank verwaltet werden müssen.

Zum Vergleich: Sammeln sich in einer MySQL-Datenbank mehr **als 50 GB Daten**, an, liegt die durchschnittliche Schreib- bzw. Lesegeschwindigkeit nur noch bei **ca. 300 ms**. Bei der gleichen Anwendung braucht eine dokumentorientierte Datenbank wie MongoDB nur **wenige Millisekunden**.

### 18.1.2 Skalierbarkeit von Datenbanken

Datenbanksysteme müssen in der Lage sein Benutzeraufgaben parallel verarbeiten zu können.

**Skalierbarkeit** bezeichnet die Fähigkeit eines Systems mit wachsender **Systemlast** seinen Dienst aufrecht zu halten zu können.

Skalierbarkeit beschreibt damit die Möglichkeit einer Datenbank eine große Zahl von Benutzeranfragen parallel verarbeiten zu können.

► Erklärung: Skalierbarkeit ▾

- Dokumentorientierte Datenbanken verdanken ihre herausragenden Skalierungseigenschaften, ihrer Fähigkeit den Datenbestand auf **mehrere Netzwerknoten** verteilen zu können. Bei zu vielen Benutzeranfragen kann die Serverlast dann einfach auf zusätzliche Netzwerknoten verteilt werden.

- Relationale Datenbanken können im Gegensatz dazu nur skaliert werden, indem der Netzwerknoten auf den die Datenbank installiert ist hardwaretechnisch verbessert.

Relationale Datenbanken sind damit nur **beschränkt skalierbar**.

18.1.4 Codebeispiel: Project Dokument ▾

JSON als Datenformat ist einfach verständlich und kann schnell verarbeitet werden.

► Codebeispiel: Project Document ▾

```
1 //-----
2 // Project
3 //-----
4 var project = {
5   id : ObjectId("3432465df43462"),
6   type : "RESEARCH_FUNDING_PROJECT",
7   title : "Motorsimulations",
8   description: "Der Vorteil dynamischer Simulation ist en, die Wechselwirkung vom ...",
9   begin_date : new Date('Jun 23, 2016'),
10  project_stuff : [
11    {
12      first-name : "Roman",
13      last-name : "Arthaber",
14      role : "PROJECT_MANAGER"
15    },
16    {
17      first-name : "Nikola",
18      last-name : "Cacicovic",
19      role : "Programmer"
20    }
21  ],
22  findings : [
23    {
24      debtor : "SAP Inc. Systems",
25      partner_name : "Hr. Dipl.Ing.- Kirchner",
26      funding_amount : NumberLong(34000000)
27    },
28    {
29      debtor : "PWF Programm",
30      partner_name : "Hr. Dipl.Ing.- Lobensky",
31      funding_amount : NumberLong(5000000)
32    }
33  ],
34  subprojects : [
35    {
36      title : "Simulation Programming",
37      description : "Programming principles
38      in ...",
39      state : "IN_PREPARATION",
40      begin_date : new Date('Jun 23, 2016')
41    },
42    {
43      title : "Simulation Models",
44      description : "Finite Mathematics in
45      ...",
46      state : "IN_PREPARATION",
47      begin_date : new Date('Jun 23, 2016')
48    }
49  ]
50}
```

149

18.1.3 Datenformat: JSON ▾

Dokumentorientierte Datenbanken können Datensätze in unterschiedlichen Datenformaten speichern.

Für viele Dokumentorientierte System, besonders MongoDB, stellt JSON das Datenformat erster Wahl dar.

► Erklärung: JSON Datenformat ▾

- Mit dem Einsatz des JSON Datenformat zur Speicherung von Datensätzen, gibt MongoDB Programmierer die Möglichkeit an die Hand alle Teile der Anwendungsentwicklung in einer **einheitlichen Programmiersprache** JavaScript umzusetzen.
- Im Anwendungscode werden Daten als Objekte bzw. JSON ähnliche Dokumente dargestellt, weil dieses Datenmodell für Entwickler **effizient zu nutzen** und intuitiv zu bedienen ist.
- Mit Dokumentendatenbanken können Entwickler Daten einfacher in einer Datenbank speichern und abfragen, indem sie das gleiche Dokumentmodellformat wie in ihrem Anwendungscode verwenden.

18.2. Modellierungsprinzipien ▾

Die **Dokumentbasierte Modellierung** folgt anderen Prinzipien als die relationale Modellierung.

18.2.1 Modellierungsprinzipien ▾

Folgende Modellierungsprinzipien sind in Dokumentorientierten Datenbanksystemen vorherrschend.

► Auflistung: Modellierungsprinzipien ▾

**Aggregation** ▾  
In dokumentbasierten Systemen werden in Beziehung stehende Daten in einem einzelnen Dokument verdichtet. Im Gegensatz zum Relationalen Modell werden **Wesentliches** nicht in ihre Aspekte aufgeteilt sondern als Einheit gespeichert.

**Relationfree** ▾  
Vermeiden Sie Beziehungen zwischen Dokumenten.

Ist es für eine Anwendung notwendig Beziehungen zwischen Dokumenten zu definieren, sollte die Modellierung der Daten überdacht werden.

**Duplication** ▾  
Im hierarchischen Datenmodell können dieselben Daten mehrmals gespeichert werden. Das hierarchische Datenmodell kennt keine Normalformen.

Datenredundanz ist akzeptabel.

**Hierarchical** ▾  
Datensätze in Dokumentorientierten Systemen sind **hierarchisch aufgebaut**. Ein Personendatensatz wird beispielsweise den Namen und andere personenbezogene Daten auf höchster Ebenen speichern. Kontaktdata werden als Subdokument eingefügt.

18.2.2 Hierarchische Datenmodelle ▾

Im hierarchischen Datenmodell werden **Geschäftsobjekte** nicht in ihre Aspekte zerplittet sondern als **Ganzes** gespeichert.

- Datensätze im hierarchischen Datenmodell haben die Struktur **eines Baums**.
- Die **Grunddaten des Datensatzes** werden **im Wurzel** element des Dokuments gespeichert.

- Alle anderen Daten werden als Kindelement des Wurzelelements, **im selben Dokument** gespeichert.

► Codebeispiel: TV Show Document ▾

```
1 //-----
2 // TV Show Dokument
3 //-----
4 var tvshow = {
5   // Grunddaten
6   title : "The Lord of the Rings",
7   description : "The Lord of the Rings is
8   an epic high fantasy novel written
9   by English author J.R.R. Tolkien.",
10  // Kindelemente
11  seasons : [
12    {
13      subtitle : "The Fellowship",
14      season_id : 1,
15      episodes : [
16        {
17          title : "The Shire",
18          episode_n : 1,
19          cast_members : [
20            {
21              role : "Frodo Baggins",
22              actor : "Eliaja Wood"
23            },
24            {
25              role : "Gandalf the Grey",
26              actor : "Sir Ian McKennitt"
27            },
28            {
29              role : "Aragorn",
30              actor : "Viggo Mortensen"
31            },
32            {
33              role : "Aragorn",
34              actor : "Viggo Mortensen"
35            },
36            {
37              role : "Bilbo Baggins",
38              actor : "Joseph Shire"
39            }
40          ]
41        }
42      ]
43    }
44  }
```

150

| Designentscheidung | Fragestellung                                                                                 |
|--------------------|-----------------------------------------------------------------------------------------------|
| Geschäftsknoten    | Welches sind die zentralen Geschäftsobjekte des fachlichen Modells?                           |
| Kopplung           | In welcher Weise sind die Geschäftsobjekte miteinander gekoppelt?                             |
| Kardinalität       | Wie gestaltet sich die Kardinalität der Beziehungen der Daten innerhalb eines Dokuments?      |
| Abrufen            | Welches sind die prominenten Abrufen und für welche Geschäftsobjekte werden sie durchgeführt? |
| Operationen        | In welchem Verhältnis stehen Schreib- und Leseoperation für bestimmte Geschäftsobjekte?       |

Abbildung 57. Designentscheidungen Schemadesign

The screenshot shows two sections of a presentation slide:

- 18.2.3 Modellstruktur** (Schemaless Modeling):
  - Schemadesign**: Describes the process of mapping the **fachlichen Modells** to the **physikalische Modell** of an information system.
  - Erklärung: MongoDB Struktur**:
    - MongoDB stores documents in form of documents. Structurally similar **Documents** are stored in **Collections**.
    - A collection can be conceptually compared with a table in relational database systems.
    - Documents can contain arrays and embedded documents.
    - References to documents in other collections are possible.
- 18.2.4 Schemafreie Modellierung** (Schema-free Modeling):
  - Schemafreiheit** describes the characteristic of an information system's data structures being free from any structural constraints.
  - Analyse: Schemafreie Modellierung**:
    - Schema-free was one of the decisions made by the MongoDB Server to ensure data consistency in the Big Data area.
    - Validation of structure and data of a document against the schema of a collection is a specific task.
    - From a business perspective, it makes sense to store business objects in a collection.
    - Schema-free allows changes to business requirements to be easily implemented in the database.

The screenshot shows a presentation slide on MongoDB DDL, specifically focusing on Data Containers:

### 19. MongoDB - DDL

# 02

#### MongoDB - DDL

**19.1. Datenkontainer**

The MongoDB database engine manages **Sammlungen von Dokumenten** in logical namespaces.

**19.1.1 Datenkontainer**

Zur Verwaltung von Daten definiert die MongoDB Spezifikation 3 Typen von Datenkontainern:

- Datenkontainer** werden zur Strukturierung und Verwaltung von Daten verwendet.

**Auflistung: Datenkontainer**

- Database**: In der Theorie versteht man unter einer Datenbank einen logisch zusammengehörigen **Datenbestand**. Die Datenbank ist in diesem Sinne ein logischer Namensraum für die Collections eines Geschäftsfalls.
- Collection**: Collections verwalten **Sammlungen gleichartiger Dokumente**. Collections können dabei konzeptionell mit den Tabellen relationaler Datenbanken verglichen werden.
- View**: Views und Sichten auf Collections. Wie in relationalen Datenbanken erlauben Views lediglich einen lesenden Zugriff auf den Datenbestand.

**Capped Collection**: Eine Capped Collection ist eine Collection die strukturell wie ein **Ringbuffer** aufgebaut ist. Sie wird verwendet um große Mengen von Daten schnell verarbeiten zu können.

**Document**: MongoDB speichert **Dokumente** in Form von Dokumenten gespeichert.

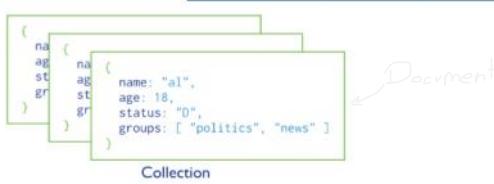


Abbildung 58. Dokumente einer Collection

**19.2. Datenkontainer anlegen**

Database ▾ Eine Datenbank dient als logischer Namensraum für die enthaltenen Collections.

**19.2.1 Datencontainer implizit anlegen** Die MongoDB Datenbankengine verwendet JavaScript als Abfragesprache und Datenbanksprache. Beim Ausführen folgender Befehle werden Datencontainer implizit angelegt.

Codebeispiel: Datenbanken implizit anlegen

```

1 // -----
2 // Syntax: createCollection
3 // -----
4 db.createCollection(
5   name : <String>,
6   option : <Document>
7 );
8
9
10 db.createCollection(<name>, {
11   capped : <boolean>,
12   autoIndexId : <boolean>,
13   size : <number>,
14   max : <number>,
15   storageEngine : <document>,
16   validator : <document>,
17   validationLevel : <string>,
18   validationAction : <string>,
19   indexOptionsDefault : <document>,
20   viewOn : <string>,
21   pipeline : <pipeline>,
22   collation : <document>,
23   writeConcern : <document>
24 });
25
26 db.projects.insertOne({
27   type : "REQUEST_FUNDING_PROJECT"
28 });
29
30 db.createCollection("projects");
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153

```

Informationssysteme

| Parameter        | Beschreibung                                                                                                                                                                                                                                                                                                                                                                                                                  | Typ      |
|------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------|
| capped           | Optional. To create a capped collection, specify true. If you specify true, you must also set a maximum size in the size field.                                                                                                                                                                                                                                                                                               | boolean  |
| autoIndexId      | Optional. Specify false to disable the automatic creation of an index on the _id field.                                                                                                                                                                                                                                                                                                                                       | boolean  |
| size             | Optional. Specify a maximum size in bytes for a capped collection. Once a capped collection reaches its maximum size, MongoDB removes the older documents to make space for the new documents. The size field is required for capped collections and ignored for other collections.                                                                                                                                           | number   |
| max              | Optional. The maximum number of documents allowed in the capped collection. The size field is preceded by the max field. If a capped collection reaches the size limit before it reaches the maximum number of documents, MongoDB removes old documents. If you prefer to use the max limit, ensure that the size limit, which is required for a capped collection, is sufficient to contain the maximum number of documents. | number   |
| validator        | Optional. Allows users to specify validation rules or expressions for the collection. document For more information, see Schema Validation.                                                                                                                                                                                                                                                                                   | document |
| validationLevel  | Optional. Determines how strictly MongoDB applies the validation rules to existing documents during an update.                                                                                                                                                                                                                                                                                                                | string   |
| validationAction | Optional. Determines whether to error on invalid documents or just warn about the violations but allow invalid documents to be inserted.                                                                                                                                                                                                                                                                                      | string   |
| viewOn           | The name of the source collection or view from which to create the view. The name is not the full namespace of the collection or view; i.e. does not include the database name and implies the same database as the view to create.                                                                                                                                                                                           | string   |
| pipeline         | An array that consists of the aggregation pipeline stage. db.createView creates the view by applying the specified pipeline to the viewOn collection or view.                                                                                                                                                                                                                                                                 | array    |
| writeConcern     | Optional. A document that expresses the write concern for the operation. Omit to document use the default write concern.                                                                                                                                                                                                                                                                                                      | document |

**19.3. Collections und Dokumente**

Collection ▾ Collections verwalten Sammlungen gleichartiger Dokumente.

**19.3.1 BSON Datenformat** Das BSON Datenformat ist eine Variante des JSON Datenformats.

| JSON Typ | Beschreibung                                                                                                                 | Key Value Pair im Dokument   |
|----------|------------------------------------------------------------------------------------------------------------------------------|------------------------------|
| object   | JSON Objekt                                                                                                                  | field : { .. }               |
| array    | Eine Array speichert eine Liste beliebiger Werte                                                                             | field : [ .. ]               |
| string   | UTF 8 codierte Zeichenkette                                                                                                  | field : "hallo Mongo"        |
| double   | 64 Bit Gleitkommazahlen                                                                                                      | field : Math.PI              |
| long     | 64 Bit Integer                                                                                                               | field : NumberLong(32353423) |
| int      | 32 Bit Integer mit Vorzeichen                                                                                                | field : NumberInt(45)        |
| bool     | Datatype für Wahrheitswerte                                                                                                  | field : true                 |
| objectId | Eine 12 Byte lange ID zur Identifikation des field : ObjectId("3120564406543233F3232902")<br>Documents: 24 Bit Hexadecimzahl |                              |

Abbildung 60. BSON Datentypen

Codebeispiel: BSON Dokument

```

1 //-----
2 // BSON Dokument
3 //-----
4 var project = {
5   // Das _id Attribut darf nur zur Spei-
6   // cherung der ID des Dokuments ver-
7   // wendet werden. Eine ID ist eine 24
8   // bit Hexadecimzahl
9   _id : ObjectId("5099903d3f4948bd1"),
10  title: "Motorsimulation"
11 
12  // Das contact Feld referenziert ein einge-
13  // bettetes Objekt
14  contact : { first: "Alan", last: "Turing" },
15  type : "REQUEST_FUNDING_PROJECT",
16  begin : new Date('Jun 23, 1990'),
17  state : "APPROVED",
18 
19  // contribe speichert eine Liste von
20  // Zeichenketten
21  contribe: [
22    "Turing machine",
23    "Turing test"
24  ],
25 
26  views : NumberLong(1250000)
27 }

```

19.3.2 Eingebettete Objekte

Embedded Document

In einem BSON Objekten können andere Objekte gespeichert werden. Eingebettete Objekte werden als **Embedded Objects** bezeichnet.

Eingebettete Objekte können über den Punktoperator referenziert werden.

Codebeispiel: embedded documents

```

1 //-----
2 // embedded document
3 //-----
4 var project = {
5   _id : ObjectId("5099903d3f4948bd2"),
6   type : "REQUEST_FUNDING_PROJECT",
7   contact : {
8     name : "Fabricio Caprera",
9     phone : {
10       number : "111-222-3333"
11     }
12   }
13 }
14 
15 //Zugriff auf Felder
16 project.contact.phone.number

```

155

## Informationssysteme

## 19.4. Dokumentschema

Schemafreiheit beschreibt die Eigenschaft eines Informationsystems, Datensätze beim Einfügen bzw. Ändern keiner Strukturprüfung zu unterziehen.

## 19.4.1 BSON Schema

In der Regel können Anwendungen Daten nur dann verarbeiten, wenn sie eine bestimmte Struktur aufweisen.

Die BSON Schema Spezifikation ist ein technischer Standard zur Beschreibung der Struktur von BSON Dokumenten.

## Erklärung: BSON Schema

- Aus fachlicher Sicht macht es keinen Sinn, Geschäftsobjekte unreflektiert in einer Collection zu speichern.
- Mit einem BSON Schema kann definiert werden, wie ein Dokument aufgebaut sein muss, um in eine Collection eingetragen werden zu können.

## Codebeispiel: BSON Schema

```

1 //-----
2 // BSON Dokument
3 //-----
4 var project = {
5   title : "Motorsimulation",
6   type : "REQUEST_PROJECT",
7   views : 10000
8 };
9 
10 //-----
11 // BSON Schema
12 //-----
13 var projectSchema = {
14   bsonType : "object",
15   required : ["title", "views"],
16   additionalProperties : true,
17   properties : {
18     title : {
19       bsonType : "string",
20       maxLength : 100
21     },
22     views : {
23       bsonType : "int",
24     }
25   }
26 };

```

## 19.4.2 Schemaattribute

Ein BSON Schema selbst ist wieder ein BSON Dokument.

Zu Definition eines BSON Schemas gibt die Schema Spezifikation eine Zahl von Attributen vor.

## Auflistung: Schemaattribute

- bsonType: Das bsonType Attribut definiert den Typ des Dokuments bzw. den Typ im Dokument enthaltener Felder.
- required: Mit dem required Attribut wird angegeben, welche Felder ein Dokument unbedingt enthalten muss.
- properties: Das properties Attribut wird verwendet, um für die Felder des Dokuments Constraints zu definieren.

## Codebeispiel: Schemaattribute

```

1 //-----
2 // BSON Dokument
3 //-----
4 var person = {
5   _id : ObjectId("5099803d3f494891"),
6   firstName : "Alan",
7   lastName : "Turing",
8   contribe : [ "Turing machine", "test" ],
9   major : [ "Math" ]
10 }
11 
12 //-----
13 // BSON Schema
14 //-----
15 var personSchema = {
16   bsonType : "object",
17   required : [ "major", "lastName" ],
18   properties : {
19     contribe : {
20       bsonType : [ "string" ]
21     },
22     major : {
23       enum : [ "Math", "English" ]
24     },
25     lastName : {
26       bsonType : "string",
27       maxLength : 50,
28       description: "... "
29     }
30   }
31 }

```

156

| Keyword                           | Definition        | Behavior                                                                                                                                                        | Type      |
|-----------------------------------|-------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------|
| <code>bsonType</code>             | string alias      | Definiert den Typ des Feldes                                                                                                                                    | all types |
| <code>enum</code>                 | array of values   | Definiert alle möglichen Werte für ein Feld.                                                                                                                    | all types |
| <code>multipleOf</code>           | number            | Field must be a multiple of this value                                                                                                                          | numbers   |
| <code>maximum</code>              | number            | Indicates the maximum value of the field                                                                                                                        | numbers   |
| <code>exclusiveMaximum</code>     | boolean           | If true and field is a number, maximum is an exclusive maximum. Otherwise, it is an inclusive maximum.                                                          | numbers   |
| <code>minimum</code>              | number            | Indicates the minimum value of the field                                                                                                                        | numbers   |
| <code>exclusiveMinimum</code>     | boolean           | If true and field is a number, minimum is an exclusive minimum. Otherwise, it is an inclusive minimum.                                                          | numbers   |
| <code>maxLength</code>            | integer           | Indicates the maximum length of the field                                                                                                                       | strings   |
| <code>minLength</code>            | integer           | Indicates the minimum length of the field                                                                                                                       | strings   |
| <code>pattern</code>              | regex string      | Field must match the regular expression                                                                                                                         | strings   |
| <code>required</code>             | array of strings  | Object's property set must contain all the specified elements in the array                                                                                      | objects   |
| <code>additionalProperties</code> | boolean or object | If true, additional fields are allowed. If false, they are not. If a valid JSON Schema object is specified, additional fields must validate against the schema. | objects   |
| <code>items</code>                | bson type         | Definiert den Typ der Elemente die im Array gespeichert werden                                                                                                  | arrays    |
| <code>maxItems</code>             | integer           | Indicates the maximum length of array                                                                                                                           | arrays    |
| <code>minItems</code>             | integer           | Indicates the minimum length of array                                                                                                                           | arrays    |
| <code>uniqueItems</code>          | boolean           | If true, each item in the array must be unique. Otherwise, no uniqueness constraint is enforced.                                                                | arrays    |
| <code>title</code>                | string            | A descriptive title string with no effect                                                                                                                       | N/A       |
| <code>description</code>          | string            | A string that describes the schema and has no effect.                                                                                                           | N/A       |

Abbildung 61. BSON Schema Attribute

157

## Informationssysteme

## 19.5. Schemaelemente

## 19.5.1 Schmaelement: Eingebettete Objekte

Für in Objekte eingebettete Objekte erfolgt die Schmadefinition in rekursiver Weise.

## » Codebeispiel: Eingebettete Objekte »

```

1 //-----
2 // BSON Dokument
3 //-----
4 var person = {
5   _id : ObjectId("5099803df3f4948b01"),
6   firstName : "Alan",
7   address : {
8     country : "Austria",
9     location : "1040 Vienna",
10    street : "Taubstummengasse 11"
11  }
12};

//-----
13// BSON Schema
14//-----
15var personSchema = {
16  bsonType : "object",
17  required : ["firstName", "address"],
18  properties : {
19    firstName : {
20      bsonType : "string",
21      description : "..."
22    },
23    address : {
24      bsonType : "object",
25      required : [
26        "country", "street"
27      ],
28      properties : {
29        country : {
30          bsonType : "string",
31          minLength : 2,
32          maxLength : 100
33        },
34        street : {
35          bsonType : "string"
36        }
37      }
38    }
39  };
40};

//-----
```

## 19.5.2 Schmaelement: Enum

## » Aufzählungstyp

Ein **Aufzählungstyp** ist ein Datentyp mit festgelegten Wertemöglichkeiten.

Enums werden verwendet um die logische Struktur und Lesbarkeit von Programmen zu verbessern.

## » Erklärung: Aufzählungstypen »

■ Zur Definition eines Enums müssen im Schema alle zulässigen Werte des Aufzählungstyps angeführt werden.

■ Dabei kann auch eine Reihenfolge festgelegt werden, die eine Ordnung der einzelnen Werte bestimmt.

## » Codebeispiel: Enums »

```

1 //-----
2 // BSON Dokument
3 //-----
4 var project = {
5   name : "Computer Simulation",
6   code : "A-33434-ZTB",
7   projectType : "REQUEST_PROJECT"
8 };

//-----
9// BSON Schema
10//-----
11var projectSchema = {
12  bsonType : "object",
13  required : [
14    "name", "code", "projectType"
15  ],
16  properties : {
17    name : {
18      bsonType : "string",
19      description : "..."
20    },
21    code : {
22      bsonType : "string"
23    },
24    projectType : {
25      enum : [
26        "REQUEST_PROJECT",
27        "RESEARCH_PROJECT",
28        "MANAGEMENT_PROJECT"
29      ]
30    }
31  }
32};

//-----
```

156

**19.5.3 Schemasegment: Array**

Arrays fassen mehrere gleichartige Werte zu einer Collection zusammen.

**Erklärung: Arraydefinition**

- Der BSON Typ eines Arrays ist array. Zur Definition eines Arrays muss zusätzlich die Art und der Aufbau der Arrayelemente angegeben werden.
- Im Schema wird dazu das items Attribut verwendet.

**Codebeispiel: Arrayschema**

```

1 //-----
2 // BSON Dokument
3 //-----
4 var point = {
5   location : [3, 2, 5],
6   props : [
7     "value coordinate",
8     "xyz values",
9   ],
10  description : "pipeline point"
11};
12 //-----
13 // BSON Schema
14 //-----
15 var pointSchema = {
16   bsonType : "object",
17   required : [
18     "location", "props", "dimension"
19   ],
20   properties : {
21     location : {
22       bsonType : "array",
23       minItems : 1,
24       maxItems : 4,
25       items : {
26         bsonType : "int"
27       }
28     },
29     props : {
30       bsonType : "array",
31       maxItems : 3,
32       items : {
33         bsonType : "string"
34       }
35     }
36   }
37 };
38 
```

**19.5.4 Arrays von Objekten**

Zur Definition eines Arrays von Objekten muss das items Attribut verwendet werden.

**Codebeispiel: Arrays von Objekten**

```

1 //-----
2 // BSON Dokument
3 //-----
4 var project = {
5   title : "Simulationssoftware",
6   subprojects : [
7     {
8       title : "Finite Elemente"
9       isFwfunded : true,
10      isPfgfunded : false
11    },
12    funding : [
13      {
14        debtor : "TU Wien",
15        amount : NumberLong(100000)
16      },
17      {
18        debtor : "Siemens AG",
19        amount : NumberLong(250000)
20      }
21    ]
22 };
23 //-----
24 // BSON Schema
25 //-----
26 var projectSchema = {
27   bsonType : "object",
28   required : [ "title", "subprojects" ],
29   properties : {
30     subprojects : {
31       bsonType : "array",
32       items : {
33         bsonType : "object",
34         required : [ "title" ],
35         properties : {
36           title : {
37             bsonType : "string",
38             maxLength : 100
39           },
40           isFwfunded : {
41             bsonType : bool
42           }
43         }
44       }
45     }
46   }
47 };
48 
```

159

**19.5.5 Collection mit Schemavalidation**

Sollen Dokumente beim Einfügen in eine Collection einer Strukturprüfung unterzogen werden, muss beim Anlegen der Collection eine Schemabeschreibung definiert werden.

**Auflistung: Attribute**

- validationLevel:** Das validationLevel Attribut definiert den Validierungslevel einer Collection.
  - strict:** Für inserts bzw. updates wird eine Strukturprüfung durch die Datenbankengine angesetzt.
  - moderate:** Für bereits gespeicherte Dokumente erfolgt keine Validierung. Ansonsten gelten die strict Regeln.
- validationAction:** Das validationAction Attribut definiert das Verhalten der Datenbankengine im Falle des Fehlschlags der Strukturprüfung.
  - error:** Ungültige Dokumente werden nicht in die Datenbank eingetragen. Die Datenbankengine antwortet mit einem Fehler.
  - warn:** Für ungültige Dokumente wird eine Warnung ausgegeben. Das Dokument wird jedoch trotzdem in die Collection geschrieben.

**Codebeispiel: Collection anlegen**

```

1 //-----
2 // Collection mit Validierung anlegen
3 //-----
4 db.createCollection( "persons",
5 {
6   validationLevel : "strict",
7   validationAction : "error",
8   validator : {
9     $jsonSchema : {
10       bsonType : "object",
11       required : [ "name" ],
12       properties : {
13         name : {
14           bsonType : "string",
15           maxLength : 50,
16           description : "..."
17         }
18       }
19     }
20   }
21 };
22 
```

**19.6. Views erstellen****19.6.1 Views**

Views sind Sichten auf Collections.  
Auf Views können nur Lesoperatorenien ausgeführt werden.

**Codebeispiel: View anlegen**

strict: Bestehende Datensätze müssen Regeln erfüllen

moderate: Bestehende Datensätze müssen Die neuen Regeln nicht erfüllt sein

Error: wirft Fehler  
Warn: wirft Warnung

**19.6.2 Views**

## 19.7. Datencontainer verwalten

### 19.7.1 Container verwalten

Wir wollen zum Schluss noch die wichtigsten Methoden zur Verwaltung von Datencontainern angeben.

#### Codebeispiel: Container verwalten

```

1 //-----
2 // Syntax: getCollection()
3 //-----
4 db.getCollection<string>();
5
6 var personColl = db.getCollection("persons");
7
8 personColl.insertOne({
9     birth : new Date('Jun 23, 1912'),
10    death : new Date('Jun 07, 1954'),
11    views : Numberlong(1250000),
12    user_name : "turing",
13    email : "alan.turing@berkeley.com",
14    phone : "+0664/3452373",
15    url : "http://www.htlkremse.at"
16 });
17
18 //return value
19 var returnValue = {
20     "acknowledged" : true,
21     "insertedId" : ObjectId("569525e144f63")
22 };
23
24 //-----
25 // Syntax: getCollectionNames()
26 //-----
27 //return all collections
28 db.getCollectionNames();
29
30 use school;
31 db.getCollectionNames();
32
33 //return value
34 var returnValue = ["persons", "teachers",
35     "students"]
36
37 //-----
38 // Syntax: drop()
39 //-----
40 db.<collectionName>.drop();
41 db.teachers.drop();
42 db.students.drop();

```

161

## 20. MongoDB - DQL

# 04

MongoDB - DQL

|                     |     |
|---------------------|-----|
| 01. Daten lesen     | 162 |
| 02. Kursormethoden  | 163 |
| 03. Query Kriterien | 166 |
| 04. Arrayoperatoren | 168 |
| 05. Projektion      | 169 |

## 20.1. Daten lesen

Zum Abfragen von Daten definiert die MongoDB Spezifikation den `find` Befehl.



### 20.1.1 find Befehl

Abfragen werden für die Datenbankengine in Form von BSON Objekten formuliert. Das Ergebnis einer Abfrage ist ein Cursor.

Abfragen können jedoch nur für die Dokumente einer einzelnen Collection formuliert werden. Das Konzept eines Joins über mehrere Collections wird nicht unterstützt.

#### Codebeispiel: find Befehl

```

1 //-----
2 // Syntax: find
3 //-----
4 var cursor = db.<collection>.find(
5     {query criteria},
6     {projection}
7 );
8
9 //-----
10 // Beispiel: find
11 //-----
12 var cursor = db.inventory.find(
13     {status: "A",
14      $or: [
15          {qty: { $lt: 30 }},
16          {item: "p"}
17      ]
18  });
19
20 while (cursor.hasNext()){
21     printjson(cursor.next());
22 }

```

← status "A" && (qty < 30 || item = p)

#### Erklärung: Methoden Parameter

- query criteria: Eine Abfrage selbst wird in Form eines BSON Objekts formuliert.

- projection: Mit einer Projektion können die Ergebnisdokumente auf eine Untermenge ihrer Attribute beschränkt werden.

- Cursor: Das Ergebnis einer Abfrage ist ein Cursor.

Q

162

```
db.projects.find(
  { level : { $gt : 4 } },
  { name : 1, address : 1 }
).limit(10);
```

query criteria  
projection

Abbildung 62. Find Befehl

**20.2 Kursormethoden**

**ResultSet** Das Ergebnis einer Abfrage wird als ResultSet bezeichnet. Im Falle einer MongoDB Datenbank ist das eine Sammlung von Dokumenten.

**Cursor** Ein Cursor ist eine Referenz auf die Elemente eines Resultsets.

**20.2.1 Cursorsmethoden** Zur Verarbeitung eines Resultsets stellt die MongoDB Spezifikation eine Reihe von Methoden zur Verfügung.

**Codebeispiel: Cursorsmethoden**

```
//-----
// Beispiel: find
//-----

var crsr = db.inventory.find({
  status: "A",
  $or: [
    { qty: { $lt: 30 } },
    { item: "p" }
  ]
});

crsr = crsr.sort({ item: -1 });

while (crsr.hasNext()) {
  printjson(crsr.next());
}
```

**20.2.2 pretty() Methode** Die Darstellung von JSON Objekte ist in der Shell ab einer gewissen Komplexität schlecht lesbar.

**Codebeispiel: pretty Methode**

```
//-----
// Cursormethode: pretty
//-----

db.students.remove();

db.students.insertOne({
  _id: ObjectId(2),
  firstName: "Franz",
  lastName: "Taver",
  roles: ["student", "room manager"]
});

var crsr = db.students.find();
crsr.pretty();
```

Die `pretty()` Methode optimiert die Darstellung von Objekten für die Ausgabe in der Konsole.

163

**20.2.3 sort() Methode** Durch den Aufruf der `sort()` Methode, werden die Dokumente eines Resultsets einer Sortierung unterzogen.

**Erklärung: sort Methods**

- Als Parameter erwartet die Methode ein Objekt, deren Attributen die Felder definieren, nach denen sortiert werden soll.
- Der Wert des Attributes definiert dabei das Sortierverhalten an: (1 – aufsteigend, -1 – absteigend).
- Idealweise sollte nach Feldern sortiert werden, für die ein Index generiert wurde.

**Codebeispiel: sort Methode**

```
//-----
// Cursormethode: pretty
//-----

db.points.remove();

db.points.insertMany([
  { x: 0, y: 2, z: 4 },
  { x: 0, y: 3, z: 4 },
  { x: 1, y: 2, z: 4 },
  { x: 1, y: 3, z: 5 },
  { x: 1, y: 4, z: 4 },
  { x: 0, y: 2, z: 5 },
  { x: 0, y: 3, z: 5 }
]);

var crsr = db.points.find();
crsr.sort({
  x: -1,
  y: -1
});

while (crsr.hasNext()) {
  printjson(crsr);
}
```

**20.2.4 limit() Methode** Durch die Verwendung der `limit()` Methode wird das Resultset auf seine ersten n Elemente beschränkt.

**Limit** Ein Limit von 0 zeigt alle Datensätze an. Für negative Werte wird der Betrag berechnet.

**Codebeispiel: limit Methode**

```
//-----
// Cursormethode: pretty
//-----

db.points.remove();

db.points.insertMany([
  { x: 0, y: 1, z: 4 },
  { x: 0, y: 2, z: 4 },
  { x: 0, y: 3, z: 4 }
]);

var crsr = db.points.find();
crsr.limit(2);
```

**20.2.5 skip() Methode** Mit der `skip()` Methode kann der Cursor eines Resultsets verändert werden.

**Codebeispiel: skip Methode**

```
//-----
// Cursormethode: skip
//-----

var crsr = db.inventory.find({
  status: "A",
  qty: { $lt: 30 },
  item: "p"
});

crsr = crsr.skip(100);
crsr.limit(20);
```

164

| Methode                         | Beschreibung                                                                                                  | Seite |
|---------------------------------|---------------------------------------------------------------------------------------------------------------|-------|
| <code>cursor.forEach()</code>   | Applies a JavaScript function for every document in a cursor.                                                 | 165   |
| <code>cursor.map()</code>       | Applies a function to each document in a cursor and collects the return values in an array.                   | tba   |
| <code>cursor.skip()</code>      | Returns a cursor that begins returning results only after passing or skipping a number of documents.          | 164   |
| <code>cursor.limit()</code>     | Constrains the size of a cursor's result set.                                                                 | 164   |
| <code>cursor.size()</code>      | Returns a count of the documents in the cursor after applying skip() and limit() methods.                     | tba   |
| <code>cursor.count()</code>     | Modifies the cursor to return the number of documents in the result set rather than the documents themselves. | tba   |
| <code>cursor.sort()</code>      | Returns results ordered according to a sort specification.                                                    | 164   |
| <code>cursor.hasNext()</code>   | Returns true if the cursor has documents and can be iterated.                                                 | tba   |
| <code>cursor.next()</code>      | Returns the next document in a cursor.                                                                        | tba   |
| <code>cursor batchSize()</code> | Controls the number of documents MongoDB will return to the client in a single network message.               | 165   |
| <code>cursor.close()</code>     | Close a cursor and free associated server resources.                                                          | tba   |
| <code>cursor.isClosed()</code>  | Returns true if the cursor is closed.                                                                         | tba   |

Abbildung 63. Kursormethoden

**20.2.6 `forEach()` Methode**

Zum weiteren Verarbeitung der Elemente eines Resultsets definiert die MongoDB Spezifikation die `forEach()` Methode.

► Codebeispiel: `forEach` Methode ▶

```
1 //-----
2 // Cursormethode: forEach
3 //-----
4 var crsr = db.persons.find();
5
6 crsr.forEach(function(doc){
7   db.persons.updateOne(
8     { _id : doc._id },
9     { $set : {
10       email : doc.lastName;
11     }
12   });
13 });
14
```

**20.2.7 `batchSize()`, `objsLeftInBatch()` Methode**

Die Batchsize definiert die maximale Zahl an Elementen, die in einem Resultset enthalten sein können.

► Codebeispiel: `batchSize` Methode ▶

```
1 //-----
2 // Cursormethode: batchSize
3 //-----
4 // Mit der batchSize() Methode wird die gegenwärtige Batchsize der Datenbank eingestellt
5 // konfiguriert.
6 var crsr = db.points.find(
7   {},
8   { _id: 0 });
9
10 crsr.batchSize(100);
11
12 while( crsr.hasNext() ) {
13   printjson(crsr.next());
14   print(crsr.objsLeftInBatch());
15 }
```

165

**20.3. Query Kriterien - Abfrageobjekt**

Zum Abfragen von Daten definiert die MongoDB Spezifikation den `find()` Befehl. Abfragen selbst werden dabei in Form eines BSON Dokuments formuliert.

**20.3.1 Abfrageobjekt**

Ein Abfrageobjekt definiert welche Eigenschaften bzw. Struktur ein Dokument haben muss, um in die Ergebnismenge einer Abfrage aufgenommen zu werden.

Abfragen dieser Art werden als **Query by Example** bezeichnet.

## ► Erklärung: Abfrageobjekt ▶

- Ein Abfrageobjekt hat dabei stets die siehe Struktur.
- Ein Abfrageobjekt definiert eine Logische Verknüpfung von Bedingungen. Die Logische Verknüpfung erfolgt dabei über eines der Operatoren `$and`, `$or` bzw. `$nor`.

## ► Syntax: Abfrageobjekt ▶

```
1 //-----
2 // Syntax: Abfrageobjekt
3 //-----
4
5 db.collection.find({
6   $logischerOperator : [
7     { .. mathematische Bedingung .. },
8     { .. mathematische Bedingung .. },
9     { .. mathematische Bedingung .. },
10    ...
11  ]
12 });
13
14 db.projects.find({
15   $and : [
16     { type : { $eq : "FUNDING_PROJECT" } },
17     { state : { $eq : "IN_PROGRESS" } },
18     { isPFSponsored : { $eq : true } }
19   ]
20 });
21
22 db.projects.find({
23   $or : [
24     { isPFSponsored : { $neq : false } },
25     { isWFSponsored : { $neq : false } },
26     { isOS Sponsored : { $neq : false } }
27   ]
28 });
29
30 // Einzelne Bedingung
31 db.projects.find({
32   type : { $eq : "RESEARCH_PROJECT" }
33 });
34
35 //-----
36 // $lt, $lte - less than, less than equal
37 // $gt, $gte - greater than
38 //-----
39 db.projects.find({
40   $and : [
41     { review : { $gt : 3 } },
42     { review : { $lte : 5 } }
43   ]
44 });
45
```

**20.3.2 Formen von Bedingungen**

Als Bedingung wird eine mathematische Aussage bezeichnet, die entweder **wahr** oder **falsch** sein kann.

## ► Auflistung: Formen von Bedingungen ▶

166



| Arrayoperator      | Beschreibung                                                                                                                                                                                                       | Seite |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------|
| <b>\$all</b>       | Der \$all Operator prüft ob eine Liste von Werten in einem Array enthalten ist.                                                                                                                                    | 168   |
| <b>\$size</b>      | Der \$size Operator prüft die Anzahl der Werte in einem Array.                                                                                                                                                     | 168   |
| <b>\$elemMatch</b> | Für die Prüfung von Objekten in Arrays wird der \$elemMatch Operator verwendet. Ist im Array ein Objekt gespeichert, das die formulierten Bedingungen erfüllt, wird das Dokument in die Ergebnismenge aufgenommen. | 169   |

Abbildung 65. Arrayoperatoren

**20.4.3 \$elemMatch Operator**

Zur Prüfung der Eigenschaften von Objekten in Arrays wird der \$elemMatch Operator verwendet.

Beinhaltet das Array ein Element, das alle Bedingungen erfüllt wird das Dokument in die Ergebnismenge aufgenommen.

## » Codebeispiel: \$elemMatch Operator

```
1 //-----
2 // Abfrageoperatoren: $elemMatch
3 //-----
4 // Der $elemMatch Operator erwartet ein
5 // Abfrageobjekt
6 db.projects.find({
7   funding : {
8     $elemMatch : {
9       $and : [
10         {debtorName : "TU Wien"}, 
11         {amount : NumberLong(1500)}
12       ]
13     }
14   }
15 });
16
17 db.projects.find({
18   projectStaff : {
19     $elemMatch : {
20       $and : [
21         {role : {$in : ["MANAGER"]}}, 
22         {lastName : "Gruber"}
23       ]
24     }
25 });
26});
```

**20.5. Projektion**

Sind für eine Abfrage nicht alle Attribute eines Dokuments interessant, wird durch die Angabe einer Projektion eine entsprechende Einschränkung definiert.

## » 20.5.1 Projektionsobjekt

Projektionen werden durch die Angabe eines Projektionsobjekts definiert.

## » Erklärung: Projektionsobjekt

- Die Attribute des Projektionsobjekts definieren die Felder der Projektion. Attribute, die in das Ergebnis aufgenommen werden sollen, wird 1 als Wert zuordnen.
- Das **\_id** Attribut eines Dokuments ist defaultmäßig Teil der Projektion.

## » Codebeispiel: Projektion

```
1 //-----
2 // Beispiel: find
3 //-----
4 var crsr = db.inventory.find(
5   { status: "A",
6     $or: [
7       { qty: { $lt: 30 } },
8       { item: p }
9     ]
10   },
11   { _id: 0, status: 1, item: 1}
12 );
13
14 while (crsr.hasNext()){
15   printjson(crsr.next());
16 }
```

169

□

□

**21. MongoDB - DML**

## MongoDB - DML

|                        |     |
|------------------------|-----|
| 01. Daten einfügen     | 170 |
| 02. Daten bearbeiten   | 171 |
| 03. Strukturoperatoren | 173 |
| 04. Wertoperatoren     | 175 |
| 05. Arrayoperatoren    | 176 |
| 03. Daten löschen      | 179 |

**21.1. Daten einfügen - insert**

Für das Einfügen von Daten stellt die MongoDB Spezifikation mehrere Formen des `insert()` Befehls zur Verfügung.



## » 21.1.1 insertOne(), insertMany() Befehl

Der `insert` Befehl wird immer auf einer einzelnen Collection ausgeführt.

## » Codebeispiel: insertOne() Befehl

```
1 //-----
2 // Syntax: insertOne
3 //-----
4 db.<collection>.insertOne(<document>);
5
6 //-----
7 // insertOne: documents without _id
8 //-----
9 try {
10   db.products.insertOne(
11     {item: "card", qty:15}
12   );
13 } catch (e) {
14   print (e);
15 }
16
17 // insertOne: document with _id
18 try {
19   db.products.insertOne(
20     {_id : 10, item : "box"
21   });
22 } catch (e) {
23   print (e);
24 }
```

## » Erklärung: Nebeneffekte des Insert Befehls

- Beim Einfügen von Dokumenten in eine Collection treten in der Regel mehrere Nebeneffekte auf.
- collection: Wird die `insert()` Methode auf einer nicht existierende Collection aufgerufen, legt die Datenbankengine die Collection implizit an.
- ObjectID: Für Dokumente ohne ein `_id` Attribut, generiert die Datenbankengine eine eindeutige ID beim Einfügen des Dokuments.

□

```
db.<collection>.insertOne(
  {
    name : "sue",
    age : 26,
    status : "pending"
  }
);
```

Abbildung 66. insert document

```
/* Codebeispiel: insertMany() Befehl */
1 //-----
2 // Syntax: InsertMany
3 //-----
4 db.<collection>.insertMany([<document>, ...]);
5
6 //-----
7 // insertMany: document without _id
8 //-----
9 try {
10   db.products.insertMany([
11     { item: "card", qty: 15 },
12     { item: "envelope", qty: 20 }
13   ]);
14 } catch (e) {
15   print(e);
16 }
17
18 var returnValue = {
19   "insertedId": [
20     ObjectId("562a9d4d381cb0f1cfcf0b01b"),
21     ObjectId("562a9d4d381cb0f1cfcf0b01b")
22   ]
23 };
24
25 //-----
26 // insertMany: document with _id
27 //-----
28 try {
29   db.products.insertMany([
30     { _id: 10, item: "large", qty: 20 },
31     { _id: 11, item: "small", qty: 55 }
32   ]);
33 } catch (e) {
34   print(e);
35 }
```

**21.2. Daten bearbeiten - update**

**21.2.1 updateOne() Befehl**

Zum Bearbeiten von Dokumenten definiert die MongoDB Spezifikation mehrere Formen des update() Befehls.

**Codebeispiel: updateOne() Befehl**

```
/* Codebeispiel: updateOne() Befehl */
1 //-----
2 // Syntax: updateOne
3 //-----
4 db.<collection>.updateOne(
5   <query criteria>, <update operator>
6 );
7
8 try {
9   db.products.updateOne(
10   { "_id": 1 },
11   { $set: { "size": true } }
12 );
13 } catch (e) {
14   print(e);
15 }
```

**Erklärung: updateOne Parameter**

- query criterias: Mit dem Parameter wird definiert, welche Dokumente einer Collection einer Änderung unterzogen werden sollen. Der Parameter erwartet dazu eine Abfrageobjekt.
- update operator: Der Parameter definiert die Art der Änderung. Die MongoDB Spezifikation definiert dazu 14 unterschiedliche Operatoren.

171

| Operator             | Beschreibung                                                                                                                                                                                                                                      | Seite |
|----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------|
| <b>\$set</b>         | Mit dem \$set Operator werden ein oder mehrere Felder auf einen konstanten, neuen Wert gesetzt. Felder, die noch nicht existieren, werden angelegt.                                                                                               | 173   |
| <b>\$unset</b>       | Zum Löschen von Feldern wird der \$unset Operator verwendet. Die Namen der zu löschen den Feldern werden als Schlüssel mit Wert 1 angegeben.                                                                                                      | 174   |
| <b>\$rename</b>      | Eine Umbenennung eines oder mehrerer Felder wird mit dem \$rename Operator durchgeführt. Der Schlüssel des jeweiligen Feldes im Änderungsdokument ist der Name des umzubenennenden Feldes, der Wert der neue Name.                                | 174   |
| <b>\$inc</b>         | Mithilfe des \$inc Operators lassen sich numerische Felder um einen bestimmten, festen Betrag erhöhen bzw. erniedrigen (bei negativen Werten). Werden Felder angegeben, die es zuvor nicht gab, werden diese auf den entsprechenden Wert gesetzt. | 176   |
| <b>\$mul</b>         | Der \$mul Operator multipliziert ein numerisches Feld mit dem angegebenen Faktor.                                                                                                                                                                 | 176   |
| <b>\$currentDate</b> | Der \$currentDate Operator wird verwendet um Datumsfelder zu setzen.                                                                                                                                                                              | tba   |
| <b>\$max</b>         | Only updates the field if the specified value is greater than the existing field value.                                                                                                                                                           | 176   |
| <b>\$min</b>         | Only updates the field if the specified value is less than the existing field value.                                                                                                                                                              | 176   |
| <b>\$setOnInsert</b> | Sets the value of a field if an update results in an insert of a document. Has no effect on update operations that modify existing documents.                                                                                                     | tba   |
| <b>\$push</b>        | Der \$push Operator hängt an ein Array, ein weiteres Element an falls das Feld noch nicht existiert, wird es angelegt.                                                                                                                            | 177   |
| <b>\$addToSet</b>    | Analog zu \$push fügt \$addToSet einem Array einen oder mehrere Werte hinzu, allerdings nur dann, wenn der jeweilige Wert noch nicht in dem Array enthalten ist.                                                                                  | 176   |
| <b>\$pop</b>         | Der \$pop Operator ist das Gegenstück zum \$push bzw. \$addToSet Operator. Mit ihm wird das letzte bzw. erste Element eines Array gelöscht. Zum Löschen des letzten Elements ist eine 1 zu übergeben, -1 löscht hingegen das erste Element.       | 178   |
| <b>\$pull</b>        | Removes all array elements that match a specified query.                                                                                                                                                                                          | 178   |
| <b>\$pullAll</b>     | Der \$pullAll Operator hängt an ein Array, ein weitere Elemente an falls das Feld noch nicht existiert, wird es angelegt.                                                                                                                         | 178   |
| <b>\$each</b>        | Modifies the \$push and \$addToSet operators to append multiple items for array updates.                                                                                                                                                          | ??    |
| <b>\$position</b>    | Modifies the \$push operator to specify the position in the array to add elements.                                                                                                                                                                | ??    |
| <b>\$slice</b>       | Modifies the \$push operator to limit the size of updated arrays.                                                                                                                                                                                 | ??    |
| <b>\$sort</b>        | Modifies the \$push operator to reorder documents stored in an array.                                                                                                                                                                             | ??    |

Abbildung 67. update Operatoren

## 21.2.2 updateMany Befehl

Die `updateMany` Methode initialisiert eine Änderung mehrerer Dokumente.

### » Codebeispiel: `updateMany`

```

1 //-----
2 // Syntax: updateMany
3 //-----
4 db.<collection>.updateMany(
5   <query criteria>,
6   <update>,
7 );
8
9 try {
10   db.restaurant.updateMany(
11     { violations: { $gt: 4 } },
12     { $set: { "review": true } }
13   );
14 } catch (e) {
15   print(e);
16 }
17
18 // Beispiel: updateMany
19 //-----
20 var restaurant2 = {
21   _id : 2,
22   name : "Rock A Feller Bar",
23   violations : 2
24 };
25
26 var restaurant3 = {
27   _id : 3,
28   name : "Empire State Sub",
29   violations : 5
30 };
31
32 db.restaurant.updateMany(
33   { violations: { $gt: 4 } },
34   { $set: { "review": true } }
35 );
36
37 var restaurant3 = {
38   _id : 3,
39   name : "Empire State Sub",
40   violations : 5,
41   review : true
42 };
43
44

```

## 21.3. Strukturoperatoren

Mit Strukturoperatoren kann die Struktur von Dokumenten geändert werden.

### » Codebeispiel: `$set Operator`

Der `$set` Operator wird verwendet um den Wert eines Attributes zu ändern.

- Mit dem `$set` Operator werden ein oder mehrere Attribute auf einen neuen, konstanten Wert gesetzt.
- Existiert eines der angegebenen Felder nicht, wird es zusammen mit dem neuen Wert im Dokument angelegt.

### » Erklärung: `$set Operator`

```

1 //-----
2 // update Operator: $set
3 //-----
4 db.products.insertMany([
5   {
6     _id : 100,
7     rating : 4,
8     tags : [ "apparel", "clothing" ]
9   },
10  {
11    _id : 101,
12    rating : 6,
13    tags : [ "apparel", "clothing" ]
14  }
15];
16
17 db.products.updateOne(
18   { _id: 100 },
19   { $set : {
20     quantity: 500,
21     tags: [
22       "coats", "outerwear", "clothing"
23     ]
24   }
25 });
26
27 var product = {
28   _id : 100,
29   quantity : 500,
30   rating : 4,
31   tags : [
32     "coats", "outerwear", "clothing"
33   ]
34 };

```

173

## 21.3.2 \$unset Operator

### ☒ Sunsef Operator

Der `$unset` Operator wird zum Löschen von Attributen verwendet.

## 21.3.3 \$rename Operator

### ☒ Srename Operator

Der `$rename` Operator wird zum Umbenennen von Attributen in Dokumenten verwendet.

### » Erklärung: `$unset Operator`

- Zu löschen Felder werden als Attribute des Updateobjekts definiert.
- Attribute die nicht im Zieldokument definiert sind, werden ignoriert.

### » Codebeispiel: `$unset Operator`

```

1 //-----
2 // update Operator: $unset
3 //-----
4 var product = {
5   _id : 100,
6   sku : "abc123",
7   quantity : 250,
8   reorder : false,
9   details : {
10     model: "14Q2", make: "xyz"
11   },
12   tags : [ "apparel", "clothing" ],
13   ratings : [
14     { by: "ijk", rating: 4 }
15   ];
16 };
17
18 db.products.updateOne(
19   { _id: 100 },
20   { $unset: {
21     ratings : "**",
22     reorder : "**"
23   }
24 };
25
26 var product = {
27   _id : 100,
28   sku : "abc123",
29   quantity : 500,
30   details : {
31     model: "14Q2", make: "xyz"
32   },
33   tags : [ "apparel", "clothing" ]
34 };

```

□

```
db.<collection>.updateMany(
  { },
  {
    $updateOperator : {
      " - "
      " - "
    }
  }
);
```

Query Kriterien      Update Operator

Abbildung 68. update Operatoren

## 21.4. Wertoperatoren

Wertoperatoren werden zum Bearbeiten numerischer Werte verwendet.

### 21.4.1 \$mul Operator

Der **\$mul** Operator wird verwendet um ein Feld auf ein Vielfaches seines ursprünglichen Wertes zu setzen.

#### Codebeispiel: \$mul Operator

```
//-----
// update Operator: $mul
//-----

var product = {
  _id : 1,
  price : NumberDecimal("10.99"),
  qty : 25
};

db.products.updateOne(
  { _id: 1 },
  {
    $mul: {
      price: NumberDecimal("1.25"),
      qty : 2
    }
  }
);

var updatedProduct = {
  _id : 1,
  price : NumberDecimal("14.7375"),
  qty : 50
};
```

#### Codebeispiel: \$min Operator

Mit dem **\$min** Operator wird der Wert eines Feldes auf den kleineren der Werte gesetzt.

#### Codebeispiel: \$min Operator

Mit dem **\$min** Operator wird der Wert eines Feldes auf den kleineren der Werte gesetzt.

```
//-----
// update Operator: $min
//-----

var result = {
  _id: 1,
  highScore: 800,
  lowScore: 200
};

var result = {
  _id: 2,
  highScore: 1800,
  lowScore: 50
};

db.products.updateOne(
  { _id: 1 },
  {
    $min: {
      price: NumberDecimal("1.25"),
      qty : 2
    }
  }
);

db.scores.updateOne(
  { _id: 1 },
  { $min: { lowScore: 250 } }
);

//-----
// Ergebnis: $min
//-----

var updatedResult = {
  _id: 1,
  highScore: 800,
  lowScore: 200
};
```

175

### 21.4.3 \$inc Operator

**✓** **Inc Operator**

Der **\$inc** Operator wird verwendet um numerische Werte zu bearbeiten.

#### Erklärung: \$inc Operator

Mithilfe des **\$inc** Operators werden numerische Felder um einen bestimmten, festen Betrag erhöht bzw. erniedrigt.

■ Existiert das angegebene Attribut nicht, wird es in das Dokument eingefügt.

#### Codebeispiel: \$inc Operator

```
//-----
// update Operator: $inc
//-----

var cart = {
  _id : 1,
  sku : "abc123",
  default : "tower",
  quantity : 10,
  type : "shovel",
  metrics : {
    orders: 2,
    ratings: 3.5
  }
};

db.products.updateMany(
  { sku: "abc123" },
  { $inc: {
    quantity: -2,
    "metrics.orders": 1
  }
});
```

■

```
db.products.updateMany(
  { sku: "abc123" },
  { $inc: {
    quantity: -2,
    "metrics.orders": 1
  }
});
```

### 21.5. Arrayoperatoren

Arrayoperatoren werden zur Verarbeitung von Arraydaten verwendet.

#### 21.5.1 \$addToSet Operator

Der **\$addToSet** Operator wird verwendet um Werte zu einem Array hinzuzufügen.

#### Erklärung: \$addToSet Operator

■ Werte die bereits in einem Array vorhanden sind werden nicht wiederholt eingefügt.

■ Zum Hinzufügen mehrerer Werte muss der **\$addToSet** Operator mit dem **\$each** Operator kombiniert werden.

#### Codebeispiel: \$addToSet Operator

```
//-----
// update array Operator: $addToSet
//-----

var test = {
  _id: 1, letters: ["a", "b"]
};

db.test.updateOne(
  { _id: 1 },
  { $addToSet: { letters: [ "c", "d" ] } }
);

var updatedTest = {
  _id: 1,
  letters: [ "a", "b", [ "c", "d" ] ]
};

var test = { _id: 2, letters: ["a", "b"] };

db.test.updateOne(
  { _id: 2 },
  { $addToSet: {
    letters: { $each: [ "c", "d" ] }
  }
});

var updatedTest = {
  _id: 2, letters: [ "a", "b", "c", "d" ]
};
```

176

Skript Page 258

| Arrayoperator     | Beschreibung                                                                                                                                       | Seite |
|-------------------|----------------------------------------------------------------------------------------------------------------------------------------------------|-------|
| \$addToSet        | Der \$addToSet Operator wird verwendet um zu einem Array einem oder mehreren Werte hinzuzufügen.                                                   | 176   |
| \$push            | Der \$push Operator wird verwendet um Elemente zu einem hinzuzufügen. Die hinzufügenden Werte unterliegen jedoch keiner Prüfung auf Eindeutigkeit. | 177   |
| \$pull, \$pullAll | Mit dem \$pull bzw. \$pullAll Operator werden Werte aus einem Array entfernt.                                                                      | 178   |
| \$pop             | Der \$pop Operator wird verwendet um Elemente aus einem Array zu löschen.                                                                          | 178   |

Abbildung 69. Arrayoperatoren

**21.5.2 \$push Operator**

**Erläuterung: \$push Operator**

- Der \$push Operator wird verwendet um Elemente in ein Array einzufügen. Existiert das Array nicht, wird es angelegt.
- Der push Operator definiert eine Reihe von Operationen um sein Verhalten zu adaptieren.

**Erläuterung: \$slice, \$sort Position Modifier**

- Position Modifier: Der \$position Modifier definiert an welcher Position im Array die Daten eingefügt werden sollen.
- \$slice Modifier: Der \$slice Modifier limitiert das Array auf die ersten bzw. letzten n Elemente.
- \$sort Modifier: Der \$sort Modifier wird verwendet um die Elemente eines Arrays zu sortieren.

**Codebeispiel: \$push Operator**

```

1 //-----
2 // update array Operator: $push
3 //-----
4 var student = {
5   quizzes: [
6     { wk : 1, score : 10 },
7     { wk : 2, score : 8 },
8     { wk : 3, score : 5 },
9     { wk : 4, score : 6 }
10   ]
11 };
12
13 db.students.updateOne(
14   { _id: 5 },
15   { $push: {
16     quizzes: {
17       $each: [
18         { wk: 5, score: 8 },
19         { wk: 6, score: 7 },
20         { wk: 7, score: 6 }
21       ],
22       $slice : 3
23     }
24   }
25 );
26
27 var updatedStudent = {
28   quizzes: [
29     { wk : 1, score : 10 },
30     { wk : 2, score : 8 },
31     { wk : 3, score : 5 }
32   ]
33 };

```

**Codebeispiel: \$push Operator**

```

1 //-----
2 // update array Operator: $push
3 //-----
4 db.students.updateOne(
5   { $push: {
6     quizzes: {
7       $each: [
8         { wk: 5, score: 8 },
9         { wk: 6, score: 7 }
10       ],
11       $sort: { score: -1 }
12     }
13   }
14 );

```

177

**Informationssysteme**

**21.5.3 \$pull, \$pullAll Operator**

**Erläuterung: \$pull Operator**

- Mit dem \$pull bzw. \$pullAll Operator lassen sich gezielt Werte aus einem Array entfernen.
- \$pull löscht alle Vorkommen eines einzelnen Wertes.
- \$pullAll löscht alle Vorkommen mehrerer Werte.

**Erläuterung: \$pull Operator**

- Der \$pull Operator ist das Gegenstück zum \$push bzw. \$addToSet Operator.
- Der \$pull Operator löscht das letzte bzw. erste Element eines Arrays.
- Zum Löschen des letzten Elements wird 1, des ersten Elements -1 übergeben.

**Codebeispiel: \$pull Operator**

```

1 //-----
2 // update array Operator: $pull
3 //-----
4 var store = { _id: 1,
5   fruits: [ "apples", "oranges", "grapes" ],
6   vegetables: [
7     "carrots", "celery", "carrots"
8   ]
9 };
10
11 var store2 = { _id: 2,
12   fruits: [ "oranges", "bananas", "apples" ],
13   vegetables: [
14     "broccoli", "zucchini", "carrots"
15   ]
16 };
17
18 db.stores.updateMany(
19   { },
20   { $pull: {
21     fruits: {
22       $in: [ "apples", "oranges" ]
23     },
24     vegetables: "carrots"
25   }
26 },
27   { multi: true }
28 );
29
30 var updatedStore = { _id: 1,
31   fruits: [ "grapes" ],
32   vegetables: [
33     "celery", "carrots"
34   ]
35 };
36
37 var updatedStore2 = { _id: 2,
38   fruits: [ "bananas" ],
39   vegetables: [ "broccoli", "zucchini" ]
40 };

```

**21.5.4 \$pop Operator**

**Erläuterung: \$pop Operator**

- Der \$pop Operator wird verwendet um Elemente aus einem Array zu löschen.

**Erläuterung: \$pop Operator**

- Der \$pop Operator ist das Gegenstück zum \$push bzw. \$addToSet Operator.
- Der \$pop Operator löscht das letzte bzw. erste Element eines Arrays.
- Zum Löschen des letzten Elements wird 1, des ersten Elements -1 übergeben.

**Codebeispiel: \$pop Operator**

```

1 //-----
2 // update array Operator: $pop
3 //-----
4 var student = {
5   _id: 1,
6   scores: [ 8, 9, 10 ]
7 };
8
9 db.students.updateOne(
10   { _id: 1 },
11   { $pop: { scores: -1 } }
12 );
13
14 var updatedStudent = {
15   _id: 1,
16   scores: [ 9, 10 ]
17 };
18
19 //-----
20 // update array Operator: $pop
21 //-----
22 var student = { _id: 1, scores: [ 8, 9, 10 ] };
23
24 db.students.updateOne(
25   { _id: 1 },
26   { $pop: { scores: 1 } }
27 );
28
29 var updatedStudent = {
30   _id: 1,
31   scores: [ 8, 9 ]
32 };

```

178

## 21.6. Daten löschen

Befehle zum Löschen von Dokumenten aus Collections.



### 21.6.1 delete Befehl

MongoDB unterscheidet mehrere Befehle zum Löschen von Dokumenten.

```
> Codebeispiel: deleteOne, deleteMany Befehl
1 //-----
2 // Syntax: deleteOne
3 //-----
4 db.<collection>.deleteOne(
5   <filter>,
6 );
7
8 //-----
9 // Syntax: deleteMany
10 //-----
11 db.<collection>.deleteMany(
12   <filter>,
13 );
14
15 //-----
16 // Beispiel: deleteOne, deleteMany
17 //-----
18 try {
19   db.orders.deleteOne( {
20     "_id" :
21       ObjectId("563237a41a4d68582c2509da")
22   } );
23 } catch (e) {
24   print(e);
25 }
26
27 try {
28   db.orders.deleteOne( {
29     "expiryIn" : {
30       "$lt: ISODate("2015-11-01T12:40:15Z"
31     }
32   } );
33 } catch (e) {
34   print(e);
35 }
```

179

## 22. MongoDB - Aggregation von Daten

# 05

Aggregation von Daten

|                            |     |
|----------------------------|-----|
| 01. Methoden und Verfahren | 180 |
| 02. Abfragemethoden        | 181 |
| 03. Aggregationframework   | 182 |
| 04. Dokumentenstufen       | 186 |
| 05. Expression             | 198 |
| 06. Kontrolloperatoren     | 202 |
| 07. Arrayoperatoren        | 204 |
| 08. Aggregatoperatoren     | 208 |

## 22.1. Methoden und Verfahren

Zur Abbildung geschäftlicher Kernprozesse, müssen Datenbanken in der Lage sein komplexe Abfragen durchzuführen.

Aggregationsalgorithmen kommen immer dann zum Einsatz, wenn Anforderungen über die Dokumente mehrerer Collections hinweg, durchgeführt werden müssen.

### 22.1.1 Aggregationsalgorithmen

Zur Definition komplexer Abfragen stellt die MongoDB Spezifikation 3 Frameworks zur Verfügung.

#### Erklärung: Aggregationsalgorithmen

Abfragemethoden ermöglichen das Aggregieren von Daten für die Dokumente einer Collection.

Das Aggregationframework ermöglicht die Formulierung komplexer Abfragen für die Dokumente mehrerer Collections.

Der Map Reduce Algorithmus ist ein Verfahren zur nebenläufigen Verarbeitung großer Datens Mengen.

Abfragemethoden: Abfragemethoden erlauben lediglich das Aggregieren von Daten für die Dokumente einer einzelnen Collection. Abfragemethoden sind in ihrem Funktionsumfang damit relativ eingeschränkt.

Aggregationframework: Das Aggregationframework ermöglicht komplexe Auswertungen für Dokumente in mehreren Collections.

Map Reduce Verfahren: Der Map Reduce Algorithmus ist der flexibelste der 3 Aggregationsalgorithmen. Das Verfahren weist bei einer Implementierung jedoch eine hohe Komplexität auf.

190

| Abfragemethoden | Beschreibung                                                                                                                                                                                                   | Seite |
|-----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------|
| count           | Mit der Hilfe der count Methode wird die Anzahl der Dokumente eines Resultsets ermittelt.                                                                                                                      | 181   |
| distinct        | Mit der distinct Methode können alle unterschiedlichen Werte eines Attributes in Form eines Array ermittelt werden.                                                                                            | 181   |
| group           | Die group Methode gruppiert die Daten einer Collection anhand eines Schlüssels. Für die gruppierten Daten können nun einfache Operationen wie das Zählen oder das Aufsummieren von Werten durchgeführt werden. | 181   |

Abbildung 70. Abfragemethoden

## 22.2. Abfragemethoden

Abfragemethoden ermöglichen das Aggregieren von Daten für die Dokumente einer Collection.

### 22.2.1 Abfragemethode: count()

Mit Hilfe der count() Methode kann die Anzahl der Dokumente eines Resultsets ermittelt werden.

#### Erklärung: count() Methode

- Mit der count() Methode kann auf einfache Weise bestimmt werden, wieviele Dokumente im Resultset einer Abfrage enthalten sind.

#### Codebeispiel: count() Methode

```
1 //-----
2 // count() Methode
3 //-----
4 db.inventory.insertMany([
5   item : "journal",
6   qty : 25,
7 }, {
8   item : "notebook",
9   qty : 50,
10 });
11 );
12
13 db.inventory.count();
14 > 2
15
16 db.inventory.find(
17   { qty : { $gte : 20} }
18 ).count();
19 > 1
```

### 22.2.2 Abfragemethode: distinct()

Mit der distinct() Methode können die unterschiedlichen Ausprägungen eines Attributes für die Dokumente einer Collection bestimmt werden.

#### Codebeispiel: distinct() Methode

Mit der distinct() Methode können die unterschiedlichen Ausprägungen eines Attributes für die Dokumente einer Collection bestimmt werden.

#### Erklärung: distinct() Methode

- Mit der distinct() Methode kann auf einfache Weise bestimmt werden, wieviele Dokumente im Resultset einer Abfrage enthalten sind.

#### Codebeispiel: distinct() Methode

```
1 //-----
2 // distinct() Methode
3 //-----
4 db.persons.insertMany([
5   { name : "Andreas", age : 21},
6   { name : "Christian", age : 23}
7 ]);
8
9 db.persons.distinct('name');
10 > ["Andreas", "Christian"]
```

### 22.2.3 Abfragemethode: group()

Die group() Methode gruppiert die Daten einer Collection anhand eines Schlüssels. Für die gruppierten Daten können einfache Operationen wie das Zählen bzw. das Aufsummieren von Werten durchgeführt werden.

#### Aufzählung: group() Parameter

- key: Das key Attribut bestimmt die Werte nach denen die Dokumente einer Collection gruppiert werden sollen.
- reduce: Die reduce Funktion definiert in welcher Form die Dokumente einer Gruppe verarbeitet werden sollen. Die Funktion wird für jedes Dokument einer Gruppe aufgerufen.

181

## 22.3. Aggregatframework

Das Aggregatframework ermöglicht die Formulierung komplexer Abfragen für die Dokumente mehrerer Collections.

Die Basis des Aggregatframeworks ist die Aggregationsspipeline.

### 22.3.1 Aggregationsspipeline

Datenstruktur Pipeline

Eine Pipeline ist eine Datenstruktur zur Verarbeitung von Daten.

Datenpipelines besitzen in der Regel mehrere Stufen. Je nach Pipelinestufe werden die Daten der Pipeline unterschiedlich verarbeitet.

Eine Aggregationsspipeline definiert eine Kette von Operationen.

#### Erklärung: Aggregationsspipeline

- Eine Aggregationsspipeline legt eine Kette von Operationen, zur Verarbeitung der Dokumente einer Collection fest. Jede Operation wird dabei durch einen Operator definiert.

Die MongoDB Spezifikation sieht keine Restriktionen für die Zusammensetzung einer Pipeline vor. Lediglich die `$out` Stufe muss als letzte Stufe einer Pipeline auftreten.

- In der Pipeline werden die Dokumente einer Collection von einer Stufe zur nächsten Stufe weitergereicht und verarbeitet. Die Angabe einer Pipelinestufe dient dabei als Eingabe der nachfolgenden Stufe.

Dabei muss ein Dokument nicht gewandert sein an die nächste Pipelinestufe weitergereicht werden. In einer Pipelinestufe können neue Dokumente erzeugt bzw. vorhandene Dokumente aus der Pipeline entfernt werden.

- Innerhalb einer Stufe kann die Struktur von Dokumenten verändert bzw. die Daten eines Dokuments geändert werden.



182

## PIPELINE

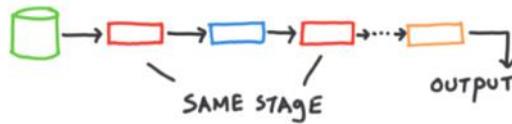


Abbildung 71. Aggregationpipeline

## 22.3.2 Aggregatframework

Das Aggregatframework basiert auf 2 Grundprinzipien:

## » Prinzipien: Aggregatframework



- » Erklärung: Pipelineexpressions »
- Expressions werden zur Verarbeitung von Daten innerhalb einer Pipelinestufe definiert.
- Eine Expression hat in der Regel nur auf die Daten eines einzelnen Dokuments innerhalb einer Pipelinestufe Zugriff.
- Bestimmte Expressions können nur in Kombination mit bestimmten Pipelineoperatoren verwendet werden.

## 22.3.3 Methode: aggregate()

Durch den Aufruf der `aggregate()` Methode kann die Verarbeitung einer Aggregationsspipline durch die Datenbankengine angestoßen werden.

## » Syntax: Methode aggregate()

```

//...
// Syntax: aggregate()
//-----
// db.<collection>.aggregate(<pipeline>);

//-----
// Beispiel: aggregate()
//-----
db.projects.aggregate([
  {
    $match : {
      $nor : [
        {type : "RESEARCH_PROJECT"},
        {type : "REQUEST_PROJECT"}
      ]
    }
  },
  {
    $project : {
      type : 1,
      state : 1,
      description : 1
    }
  }
], { $sort : { follower_count : -1 } });
  
```

163

| Informationssysteme      |                                                                                                                                                                                                                                                                                                                           |          |
|--------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------|
| Operator                 | Beschreibung                                                                                                                                                                                                                                                                                                              | Referenz |
| <code>\$addFields</code> | Adds new fields to documents. Similar to <code>\$project</code> , <code>\$addFields</code> reshapes each document in the stream, specifically by adding new fields to output documents that contain both the existing fields from the input documents and the newly added fields.                                         | 188      |
| <code>\$bucket</code>    | Categorizes incoming documents into groups, called buckets, based on a specified expression and bucket boundaries.                                                                                                                                                                                                        | 197      |
| <code>\$count</code>     | Returns a count of the number of documents of this stage of the aggregation pipeline.                                                                                                                                                                                                                                     | 188      |
| <code>\$group</code>     | Groups input documents by a specified identifier expression and applies the accumulator expression(s), if specified, to each group. Consumes all input documents and outputs one document per each distinct group. The output documents only contain the identifier field and, if specified, accumulated fields.          | 196      |
| <code>\$limit</code>     | Passes the first <code>n</code> documents unmodified to the pipeline where <code>n</code> is the specified limit. For each input document, outputs either one document (for the first <code>n</code> documents) or zero documents (after the first <code>n</code> documents).                                             | 187      |
| <code>\$lookup</code>    | Performs a left outer join to another collection in the same database to filter in documents from the "joined" collection for processing.                                                                                                                                                                                 | 192      |
| <code>\$match</code>     | Filters the document stream to allow only matching documents to pass unmodified into the next pipeline stage. <code>\$match</code> uses standard MongoDB queries. For each input document, outputs either one document (a match) or zero documents (no match).                                                            | 186      |
| <code>\$out</code>       | Writes the resulting documents of the aggregation pipeline to a collection. To use the <code>\$out</code> stage, it must be the last stage in the pipeline.                                                                                                                                                               | 187      |
| <code>\$project</code>   | Reshapes each document in the stream, such as by adding new fields or removing existing fields. For each input document, outputs one document.                                                                                                                                                                            | 189      |
| <code>\$skip</code>      | Skipps the first <code>n</code> documents where <code>n</code> is the specified skip number and passes the remaining documents unmodified to the pipeline. For each input document, outputs either zero documents (for the first <code>n</code> documents) or one document (of after the first <code>n</code> documents). | 186      |
| <code>\$sort</code>      | Reorders the document stream by a specified sort key. Only the order changes; the documents remain unmodified. For each input document, outputs one document.                                                                                                                                                             | 186      |
| <code>\$unwind</code>    | Deconstructs an array field from the input documents to output a document for each element. Each output document replaces the array with an element value. For each input document, outputs <code>n</code> documents where <code>n</code> is the number of array elements and can be zero for an empty array.             | 187      |

Abbildung 72. Pipelinestufen

#### 22.3.4 Pipelinestufen

Jede der Pipelinestufe wird durch einen eigenen Pipelineoperator definiert.

##### ► Auflistung: Pipelinestufen

- match Stufe** ▾  
Die **match** Stufe wird verwendet um die Dokumente im Dokumentenstrom der Pipeline zu filtern.
- project Stufe** ▾  
Die **project** Stufe der Aggregationspipeline erlaubt die Manipulation von Dokumenten durch das Umbenennen, Hinzufügen bzw. Entfernen von Attribut.
- limit Stufe** ▾  
Mit dem **limit** Operator kann die Anzahl der Dokumente im Dokumentenstrom der Pipeline limitiert werden.
- group Stufe** ▾  
Mit dem **group** Operator können Auswertungen über alle Dokumente einer Pipelinestufe hinweg durchgeführt werden.
- lookup Stufe** ▾  
Mit dem **lookup** Operator können die Dokumente einer Pipelinestufe mit den Dokumenten anderer Collections in Relation gesetzt werden.
- unwind Stufe** ▾  
Mit dem **unwind** Operator können Arrays in Dokumenten zur späteren Verarbeitung aufgelöst werden.
- out Stufe** ▾  
In der **out** Stufe werden die Dokumente der vorhergehenden Pipelinestufe in eine Collection geschrieben.

#### 22.3.5 Fallbeispiel: Projects

Die folgenden Dokumente definieren den Datenbestand der nachfolgenden Abfragen.

##### ► Codebeispiel: projects Dokument

```
//-----
// Fallbeispiel: projects collection
//-----
db.projects.insertMany([
  {
    _id : ObjectId("...1"),
    title : "Produktionsplausionsystem",
    type : "REQUEST_PROJECT",
    state : "APPROVED",
    fundings : [
      {
        debtorName : "SAP Microsystems",
        amount : NumberLong(10000)
      }
    ],
    reviews : [ 4,4,3,3,4 ]
  },
  {
    _id : ObjectId("...2"),
    title : "Finite Elemente",
    type : "RESEARCH_PROJECT",
    state : "IN_APPROVEMENT",
    fundings : [
      {
        debtorName : "TU Wien",
        amount : NumberLong(5000)
      },
      {
        debtorName : "Oracle Systems",
        amount : NumberLong(15000)
      }
    ],
    reviews : [ 5,5,3 ]
  },
  {
    _id : ObjectId("...3"),
    title : "Emissionsvermessung",
    type : "MANAGEMENT_PROJECT",
    state : "IN_APPROVEMENT",
    fundings : [
      {
        debtorName : "Sun Microsystems",
        amount : NumberLong(45000)
      },
      {
        debtorName : "Oracle Systems",
        amount : NumberLong(15000)
      }
    ],
    reviews : [ 4,5,4,5,5 ]
  }
]);
```

165

#### Informationssysteme

#### 22.4. Dokumentstufen

- Dokumentstufen** ▾  
Durch die Verwendung von Dokumentstufen wird die Anzahl der Dokumente im Dokumentenstrom der Pipeline gesteuert.

#### 22.4.1 Dokumente filtern - \$match

Die **match** Stufe wird verwendet um Dokumente im Dokumentenstrom der Pipeline zu filtern.

##### ► Erklärung: \$match Stufe

- Dokumente, die in der Stufe definierten Filterbedingungen erfüllen, werden unverändert an die nächste Pipelinestufe weitergegeben. Alle anderen Dokumente werden verworfen.

- Filterbedingungen werden als Query Kriterien definiert.

##### ► Syntax: \$match Operator

```
//-----
// Syntax: $match Operator
//-----
db.collection.aggregate([
  { $match : { $expr : { $eq : [ "$$id", "request" ] } } }
]);
```

//-----

// Beispiel: \$match Stufe

```
//-----
db.projects.aggregate([
  { $match : { type : "REQUEST_PROJECT" } },
  { $sort : { title : -1 } }
]);
```

//-----

// Beispiel: \$skip

Der **\$skip** Operator wird verwendet um Dokumente aus dem Dokumentenstrom der Pipeline zu entfernen.

#### 22.4.2 Dokumente sortieren - \$sort

- Mit dem **\$sort** Operator können die Dokumente im Dokumentenstrom der gegebenen Pipelinestufe sortiert werden.

- 1 definiert dabei eine aufsteigende, -1 eine absteigende Sortierung.

##### ► Syntax: \$sort Operator

```
//-----
// Syntax: $sort Operator
//-----
db.<collection>.aggregate([
  { $sort : {
    <sort-key> : <sort-order>
  }
});
//-----
// Beispiel: $sort Stufe
//-----
db.projects.aggregate([
  { $match : { type : "REQUEST_PROJECT" } },
  { $sort : { title : -1 } }
]);
```

//-----

// Beispiel: \$skip

Der **\$skip** Operator wird verwendet um Dokumente aus dem Dokumentenstrom der Pipeline zu entfernen.

##### ► Syntax: \$skip Operator

```
//-----
// Syntax: $skip Operator
//-----
db.<collection>.aggregate([
  { $skip : <number_of_documents> }
]);
//-----
// Beispiel: $skip Stufe
//-----
db.projects.aggregate([
  { $match : { type : "REQUEST_PROJECT" } },
  { $skip : 1 }
]);
```

166

**22.4.4 Arrays auflösen - \$unwind**

Der **\$unwind** Operator wird verwendet um Arrays strukturell aufzulösen.

Der **\$unwind** Operator ist die Gegenoperation zum **\$group** Operator.

► Erklärung: **\$unwind Operator** ▶

Der **\$unwind** Operator wird verwendet um Arrays strukturell aufzulösen.

Dann wird für jedes Arrayelement eines Datensatzes ein neues Dokument generiert.

Die verbleibenden Daten des ursprünglichen Dokuments, bleiben dabei von Änderungen unberührt, und werden unverändert in die neu generierten Dokumente übernommen.

► Syntax: **\$unwind Operator** ▶

```
//-----
// Syntax: $unwind Operator
//-----
db.<collection>.aggregate([
  { $unwind : <fieldname> }
]);

db.projects.aggregate([
  { $match : { type: "RESEARCH_PROJECT" } },
  { $unwind : {
    $from: "reviews",
    $as: "project"
  },
  { title:1, type: 1, reviews: 1 }
},
  { $out : "projectreport" }
]);
//-----
// Ausgabe: $unwind Stufe
//-----
{
  title : "Simulationsystems",
  type : "RESEARCH_PROJECT",
  reviews : 5
},
{
  title : "Simulationsystems",
  type : "RESEARCH_PROJECT",
  reviews : 5
},
{
  title : "Simulationsystems",
  type : "RESEARCH_PROJECT",
  reviews : 3
}
```

**22.4.5 Dokumentenstrom einschränken - \$limit**

Mit dem **\$limit** Operator kann die Zahl der Dokumente im Dokumentenstrom limitiert werden.

► Syntax: **\$limit Operator** ▶

```
//-----
// Syntax: $limit Operator
//-----
db.<collection>.aggregate([
  { $limit : { <number_of_documents> } }
]);
//-----
// Beispiel: Limit Stufe
//-----
db.projects.aggregate([
  { $match : { type : "REQUEST_PROJECT" } },
  { $limit : 5 }
]);
//-----
```

**22.4.6 Ergebnis abspeichern - \$out**

Der **\$out** Operator finalisiert eine Pipeline und schreibt die Dokumente des Dokumentenstroms in eine Collection.

► Syntax: **\$out Operator** ▶

```
//-----
// Syntax: $out Operator
//-----
db.<collection>.aggregate([
  { $out : <name_of_collection> }
]);
//-----
// Beispiel: $out Stufe
//-----
db.projects.aggregate([
  { $match : {
    type: "RESEARCH_PROJECT"
  },
  { title:1, type: 1, reviews: 1 }
},
  { $out : "projectreport" }
]);
//-----
```



167

## Informationssysteme

**22.5. Strukturstufen**

**Strukturstufen**

Strukturstufen werden verwendet um die Struktur der Dokumente im Dokumentenstrom der Pipeline zu ändern.

**22.5.1 Felder hinzufügen - \$addFields**

Der **\$addFields** Operator wird verwendet um neue Felder zu Dokumenten hinzuzufügen.

Der **\$addFields** Operator ist die Komplementäroperation zum **\$project** Operator.

► Erklärung: **\$addFields Operator** ▶

Der **\$addFields** Operator wird verwendet um neue Felder zu Dokumenten hinzuzufügen.

Die ursprünglichen Daten des Dokuments werden dabei zusätzlich zu den neu hinzugefügten Feldern unverändert übernommen.

Der **\$addFields** Operator unterstützt eine Reihe von Expression zur Extrahierung von Information aus bestehenden Dokumentendaten.

Der **\$** Operator erhält dabei den Zugriff auf die Werte der Felder des Dokuments.

► Syntax: **\$addFields Operator** ▶

```
//-----
// Syntax: $addFields Operator
//-----
db.<collection>.aggregate([
  { $addFields : {
    { _newField : <expression> }
  }
});
//-----
// Beispiel: Konstante Werte hinzufügen
//-----
db.projects.aggregate([
  { $match : { type: "MANAGEMENT_PROJECT" } },
  { $addFields : {
    />Variable Werte hinzufügen
    { _id : ObjectId("...3"),
      title : "Simulationsystems",
      type : "MANAGEMENT_PROJECT",
      state : "IN_APPROVEMENT",
      fundings : [
        { debitorName : "Sun Microsystems",
          amount : NumberLong(45000)
        },
        { debitorName : "Oracle Systems",
          amount : NumberLong(15000)
        }
      ],
      keyword : [ "initialized" ],
      reviews : [ 4,5,4,5,5 ],
      verified : true
    }
  }
]);
//-----
// Beispiel: Variable Werte hinzufügen
//-----
db.projects.aggregate([
  { $match : { type: "MANAGEMENT_PROJECT" } },
  { $addFields : {
    />Variable Werte hinzufügen
    { _id : ObjectId("...3"),
      title : "Simulationsystems",
      type : "MANAGEMENT_PROJECT",
      state : "IN_APPROVEMENT",
      fundings : [
        { debitorName : "Sun Microsystems",
          amount : NumberLong(45000)
        },
        { debitorName : "Oracle Systems",
          amount : NumberLong(15000)
        }
      ],
      keyword : [ "initialized" ],
      reviews : [ 4,5,4,5,5 ],
      description : "Simulationsystems"
    }
  }
]);
//-----
```

168

```

1 //-----
2 // Beispiel: Aggregatoperationen
3 //-----
4 db.projects.aggregate([
5   { $addFields : {
6     //Konstante Werte hinzufügen
7     verified : true,
8     keyword : [ "initialised" ],
9
10    //Aggregatoperatoren auf Arrays
11    voteCount : {
12      $max : "$reviews",
13    },
14    minVote : {
15      $min : "$reviews"
16    },
17    maxVote : {
18      $max : "$reviews"
19    },
20    groupCount : {
21      $size : "$reviews"
22    }
23  }
24 },
25 );
26 //-----
27 // Ergebnis: Aggregatoperationen
28 //-----
29 {
30   {
31     _id : ObjectId("...3"),
32     title : "Simulationsystem",
33     type : "MANAGEMENT_PROJECT",
34     state : "IN_APPROVEMENT",
35     funding : [
36       {
37         debtorName : "Sun Microsystems",
38         amount : NumberLong(45000)
39       },
40       {
41         debtorName : "Oracle Systems",
42         amount : NumberLong(15000)
43     }
44   ],
45   reviews : [ 4,5,4,5,5 ],
46   keyword : [ "initialised" ],
47   groupCount : 5
48   verified : true,
49   voteCount : 23,
50   minVote : 4,
51   maxVote : 5
52 }
53 }

```

□

189

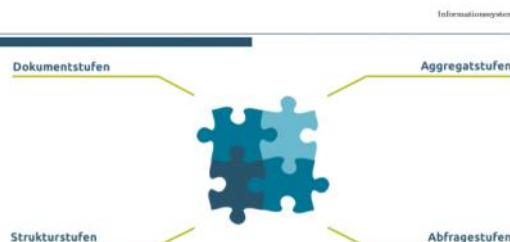


Abbildung 73. Stufentypen

**22.5.2 Felder projizieren - \$project**

Der \$project Operator wird verwendet um die Daten eines Dokuments auf eine Untermenge seiner Attribute zu beschränken.

Der \$project Operator ist die Komplementarversion zum \$addField Operator.

**Erläuterung: Project Operator \***

Der \$project Operator wird verwendet um die Daten eines Dokuments auf eine Untermenge seiner Attribute zu beschränken.

Es werden dabei nur jene Daten eines Dokuments an die nächste Pipelinestufe weitergegeben, die Teil der definierten Projektion sind.

**Syntax: \$project Operator \***

Dem \$project Operator wird ein Objekt als Parameter übergeben.

Mit dem Objekt wird bestimmt welche Felder eines Dokuments in das Ergebnis der Pipelinestufe übernommen werden sollen.

Gleichzeitig können neue Attribute zu Dokumenten hinzugefügt werden.

Dabei erlaubt der \$ Operator Zugriff auf die Werte der Felder der Eingabedokumente.

```

//-----
// Syntax: $project
//-----
db.<collection>.aggregate([
  { $project : {<fieldname : (0|1)> } }
]);

```

**Erklärung: Project Operator \***

Der \$project Operator wird verwendet um die Daten eines Dokuments auf eine Untermenge seiner Attribute zu beschränken.

Es werden dabei nur jene Daten eines Dokuments an die nächste Pipelinestufe weitergegeben, die Teil der definierten Projektion sind.

**Syntax: \$project Operator \***

Dem \$project Operator wird ein Objekt als Parameter übergeben.

Mit dem Objekt wird bestimmt welche Felder eines Dokuments in das Ergebnis der Pipelinestufe übernommen werden sollen.

Gleichzeitig können neue Attribute zu Dokumenten hinzugefügt werden.

Dabei erlaubt der \$ Operator Zugriff auf die Werte der Felder der Eingabedokumente.

```

//-----
// Syntax: $project Operator
// -----
// Syntax: $project
// -----
db.projects.aggregate([
  { $project : {
    //Felder auswählen
    projectType : "type",
    projectState : "state",
    //Aggregatoperatoren für Arrays
    maxVote : {
      $max : "$reviews"
    },
    minVote : {
      $min : "$reviews"
    }
  })
]);

```

190

## » Ergebnis: \$project Operator

```

1 //-----
2 // Ergebnis: $project Stufe
3 //-----
4 {
5   title : "Simulationssystems",
6   projectType : "MANAGEMENT_PROJECT",
7   reviews : [4,5,4,5,5],
8   maxVote : 5,
9   minVote : 4
10 }

```

□

## 22.5.3 Dokumentwurzel ändern - \$replaceRoot

Mit dem \$replaceRoot Operator wird die Struktur der Dokumente im Dokumentenstrom der Pipeline verändert.

## » Codebeispiel: \$replaceRoot Operator

```

1 //-----
2 // Syntax: $replaceRoot Operator
3 //-----
4 db.collection.aggregate([
5   { $replaceRoot : {
6     newRoot : <replacementDocument>
7   }
8 }
9 );
10 //-----
11 // Beispiel: $replaceRoot Stufe
12 //-----
13 db.tweets.aggregate([
14   { $match : { lang : "en" } },
15   {
16     $replaceRoot : {
17       newRoot : "$user"
18     }
19   }
20 );
21 );
22 > Ausage
23 account : {
24   name : "Jonas Nagelmaier",
25   verified : false,
26   rating : 6,
27   state : "initialised"
28 }

```

□

191

## Informationssysteme

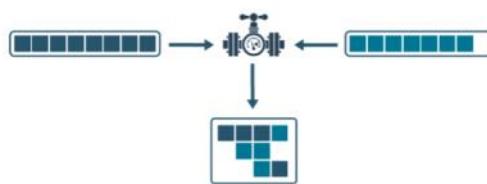


Abbildung 74. Aggregatframework: Beziehungsstufen

## 22.6. Beziehungsstufen

## » Syntax: \$lookup Operator

Beziehungsstufen werden verwendet um Relationen mehrerer Collections miteinander in Relation zu setzen.

**22.6.1 Relationen definieren - \$lookup**

Der \$lookup Operator wird verwendet um Relationen zwischen Dokumenten zu definieren.

Mit dem \$lookup Operator können 2 Arten von Relationen beschrieben werden.

**Erläuterung: \$lookup Operator**

- Die MongoDB Spezifikation definiert für den \$lookup Operators 2 unterschiedliche Formen.
- Zum einen wird der \$lookup Operator verwendet, um einen left join zwischen Dokumenten zu definieren.
- Zum anderen können mit dem \$lookup Operator 2 Pipelines in Relation gesetzt werden.
- Jede der Formen des \$lookup Operators verwendet unterschiedliche Attribute.

```

1 //-----
2 // Syntax: $lookup Operator
3 //-----
4 // left join Definition
5 /**
6 * @param <from> : Der Name der Collection
7 * deren Daten das Ziel der Beziehung sind.
8 *
9 * @param <localField> : Der Name des Attribut
10 * butes der Joinbedingung der Dokumente
11 * der Basiscollection.
12 *
13 *
14 * @param <foreignField> : Der Name des
15 * Attributes in der zu joindenden Collection.
16 *
17 * @param <as> : Ein Array das die gejointen
18 * Datensätze enthält.
19 */
20 db.<collection>.aggregate([
21   { $lookup : {
22     from : <>collection to join>,
23     localField : <field input document>,
24     foreignField : <extern join field>,
25     as : <output array field>
26   }
27 }
28 );

```

192

| Operation             | Beschreibung                                                                                      | Referenz |
|-----------------------|---------------------------------------------------------------------------------------------------|----------|
| left join             | Mit dem \$lookup Operator kann ein left join für die Dokumente 2er Collections definiert werden.  | 102      |
| Unterabfrage          | Der \$lookup Operator kann verwendet werden, um Pipelines miteinander in Relation zu setzen.      | 194      |
| Hierarchische Abfrage | Mit dem \$graphLookup Operator können Abfragen auf hierarchischen Strukturen durchgeführt werden. | ??       |

Abbildung 75. Aggregatframeworks Beschilderungen

```
> Codebeispiel: subprojects < Codebeispiel: $lookup Operator <
1 //----- 1 //-----
2 // Fallbeispiel : subprojects 2 // Beispiel: Relation definieren
3 //----- 3 //-----
4 db.subprojects.insertMany([
5   {
6     _id : ObjectId(..11),
7     appliedResearch : NumberInt(20),
8     focusResearch : NumberInt(80),
9     project_id : ObjectId(..1),
10    title : "ERP SAP",
11    facility : {
12      name : "Softwaretechnikinsttitut",
13      _id : ObjectId(..23)
14    },
15    _id : ObjectId(..12),
16    appliedResearch : NumberInt(40),
17    focusResearch : NumberInt(60),
18    project_id : ObjectId(..1),
19    title : "Webbased Systems",
20    facility : {
21      name : "Datenbankeninstitut",
22      _id : ObjectId(..45),
23    },
24    _id : ObjectId(..13),
25    appliedResearch : NumberInt(80),
26    focusResearch : NumberInt(10),
27    project_id : ObjectId(..2),
28    title : "Embedded Systems"
29  },
30  {
31    _id : ObjectId(..14),
32    appliedResearch : NumberInt(10),
33    focusResearch : NumberInt(80),
34    project_id : ObjectId(..3),
35    title : "API Design SAP"
36  }
37 ]); 38 });
39 39 
```

193

## Informationssysteme

> Codebeispiel: subprojects < 22.6.2 Unterabfrage definieren - \$lookup ■

Der \$lookup Operator kann verwendet werden, um Pipelines miteinander in Relation zu setzen.

```
1 //----- 1 //-----
2 // Ergebnis: $lookup Stufe 2 // Syntax: $lookup Operator
3 //----- 3 //-----
4 { 4 //-----
5   _id : ObjectId(..11),
6   appliedResearch : NumberInt(20),
7   focusResearch : NumberInt(80),
8   title : "ERP SAP",
9   project : {
10     id : ObjectId("..1"),
11     title : "Produktionsplanungssystem",
12     type : "REQUEST_PROJECT",
13     reviews : [
14       4,4,3,3,4
15     ],
16     fundings : [
17       {
18         debtorName : "..."
19       }
20     },
21     _id : ObjectId(..12),
22     appliedResearch : NumberInt(40),
23     focusResearch : NumberInt(60),
24     title : "Webbased Systems",
25     project : {
26       id : ObjectId("..1"),
27       title : "Produktionsplanungssystem",
28       type : "REQUEST_PROJECT",
29       reviews : [
30         4,4,3,3,4
31       ],
32       fundings : [
33         {
34           debtorName : "..."
35         }
36       }
37     },
38     _id : ObjectId(..13),
39     appliedResearch : NumberInt(80),
40     focusResearch : NumberInt(80),
41     title : "Embedded Systems"
42     project : {
43       id : ObjectId(..2),
44       title : "Finite Elemente",
45       type : "RESEARCH_PROJECT",
46       reviews : [
47         5, 5, 3
48       ],
49       fundings : [
50         {
51           debtorName : "..."
52         }
53       ]
54     }
55   }
56 
```

## &gt; Syntax: \$lookup Operator &lt;

194

## ► Codebeispiel: \$lookup Operator

```

1 //-----
2 // Beispiel: $lookup Operator
3 //-----
4 db.debtors.aggregate([
5   {
6     $lookup: {
7       from: "projects",
8       let: {
9         pid: {"$funding.project_id"},
10        debid: {"$._id"
11      },
12      pipeline: [
13        {
14          $unwind: "$fundings"
15        },
16        {
17          $match: {
18            $expr: {
19              $and: [
20                {"$in: [
21                  "$_.id",
22                  "$$pid"
23                ],
24                eq: [
25                  "$$debitid",
26                  "$$fundings._id"
27                ]
28              }
29            }
30          },
31          $project: {
32            title: 1,
33            type: "$projectType",
34            state: "$projectState",
35          }
36        ],
37        as: "projects"
38      },
39      {
40        $project: {
41          name: 1,
42          projects: 1
43        }
44      },
45      {
46        $out: "debtorReport2"
47      }
48    );
  
```

□

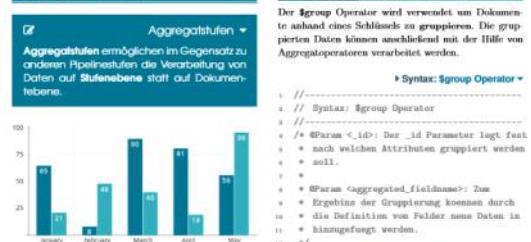
195

## Informationssysteme

| Operator | Beschreibung                                                                                 | Referenz |
|----------|----------------------------------------------------------------------------------------------|----------|
| \$group  | Der \$group Operator wird verwendet um Dokumente anhand eines Schlüssels zu gruppieren.      | 196      |
| \$bucket | Der \$bucket Operator wird verwendet um Dokumente anhand eines Wertintervalls zu gruppieren. | 197      |

Abbildung 76. Expressions: Arithmetische Operatoren

## 22.7. Aggregatstufen



## 22.7.1 Aggregatoperatoren

Aggregatoperatoren ermöglichen innerhalb einer Aggregatstufe Zugriff auf die Daten aller Dokumente der gegenwärtigen Stufe.

## ■ Erklärung: Aggregat- vs. Dokumentstufen

- Der Input einer Pipelinestufe besteht aus den Dokumenten des Dokumentenstroms der vorhergehenden Pipelinestufe.
- Die Dokumente des Dokumentenstroms werden getrennt in eine Pipelinestufe eingeslesen und verarbeitet. Pipelineoperatoren haben dabei nur auf die Daten eines einzelnen Dokuments Zugriff.
- Aggregatstufen ermöglichen im Vergleich dazu Daten auf Stufenebene zu verarbeiten.

## ► Syntax: \$group Operator

```

1 //-----
2 // Syntax: $group Operator
3 //-----
4 /*#param <_id>: Das _id Parameter legt fest
5 * nach welchen Attributen gruppiert werden
6 * soll.
7 *
8 * #Param <aggregated_fieldname>: Zum
9 * Ergebnis der Gruppierung kommen durch
10 * die Definition von Felder neue Daten in
11 * hinzugefügt werden.
12 */
13 db.collection.aggregate([
14   { $group : {
15     _id : <grouped_by_fieldname>,
16     <aggregated_fieldname> : { $sum: 1
17   }
18 });
  
```

□

196

**22.7.3 Fallbeispiel: Gruppierung**

Welche Projekte haben die höchste Projektförderung?

## ► Codebeispiel: Gruppierungsstufen ▾

```

1 //-----
2 // Codebeispiel: Gruppierungsstufen
3 //-----
4 db.projects.aggregate([
5   {
6     $addFields : {
7       projectFunding : {
8         $sum : "$fundings.amount"
9       }
10    },
11   {
12     $group : {
13       _id : null,
14       projectFunding : {
15         $max : "$projectFunding"
16       }
17    },
18   },
19   {
20     $lookup : {
21       from : "projects",
22       let : {
23         funds : "$projectFunding"
24       },
25       pipeline : [
26         {
27           $addFields : {
28             funds : {
29               $sum : "fundings.amount"
30             }
31           }
32         },
33         {
34           $match : {
35             $expr : {
36               $eq : [
37                 "$funds",
38                 "$$funds"
39               ]
40             }
41           },
42           $replaceRoot : {
43             newRoot : "$projects"
44           }
45         }
46       ],
47     }
48   }
49 ]);

```

**22.7.4 Dokumente gruppieren - \$bucket**

Der \$bucket Operator wird verwendet um Dokumente anhand eines Wertintervalls zu gruppieren.

## ► Syntax: \$bucket Operator ▾

```

1 //-----
2 // Syntax: $bucket Operator
3 //-----
4 /*
5  * @Param <groupBy>: Der Parameter definiert
6  * das Kriterium nach dem der Datenbestand
7  * gruppiert werden soll.
8  *
9  * @Param <boundaries>: Mit dem Parameter
10  * werden feste Gruppierungsrichten
11  * definiert.
12 */
13 db.<collection>.aggregate([
14   {
15     $bucket : {
16       groupBy : <expression>,
17       boundaries : [<lowerbound>, ...],
18       output : {
19         <output1 : {<expression>}...>
20       }
21     }
22   }
23 ]);

```

// Beispiel: \$bucket Stufe

// Gruppieren Sie Subprojekte entsprechend  
// ihrem Wert fuer Angewandte Forschung.  
// appliedResearch Intervalle

// 1.Interval: 0 - 50  
// 2.Interval: 51 - 100

db.subprojects.aggregate([
 {
 \$bucket : {
 groupBy : "appliedResearch",
 boundaries : [0, 51, 101],
 output : {
 count : { \$sum : 1},
 titles : { \$push : "\$title"}
 }
 }
 }
]);

197

## Informationssysteme

**22.8. Expression**

 **Expression** ▾  
Expressions werden zur Verarbeitung von Dokumenten innerhalb einer Pipelinestufe verwendet. Aus formaler Sicht entspricht eine Expression einem Ausdruck.

Das Aggregationsframework basiert auf 2 Grundprinzipien: der Pipelineverarbeitung und den Pipelineexpressions.

## ► Erklärung: Pipelineexpressions ▾

■ Expressions sind Ausdrücke zur Verarbeitung von Daten.

■ Expressions beschränken sich dabei immer auf das aktuelle zu verarbeitende Dokument. Es ist nicht möglich, auf Daten anderer Dokumente Bezug zu nehmen.

**22.8.1 Expressionssyntax**

Für Expressions definiert die MongoDB Spezifikation eine einfache Syntax.

Die Expressionssyntax ist dabei rekursiv definiert und ermöglicht dadurch die Formulierung beliebig komplexer Ausdrücke.

## ► Syntax: Expression ▾

```

1 //-----
2 // Syntax: Expression
3 //-----
4 db.<collection>.aggregate([
5   {
6     $step : {
7       <expression>,
8       <expression>
9     }
10   },
11   {
12     <step> : {
13       <expression>,
14       <expression>
15     }
16   }
17 });

```

<expression> = <field path> | <object>

<operator> = <operator> : <exp> |  
| <operators> = [<op>, <op>, ...]

<operator> = <arithmetic operator>  
| <array operator>  
| <boolean operator>  
| <comparison operator>  
| <conditional operator>  
| <accumulators>  
| <variable operators>

// Arithmetische Operatoren  
// Arithmetic op. = \$abs | \$add | \$divide  
// | \$multiply  
// | \$subtract

// Arrayoperatoren  
// Array op. = \$arrayElemAt | \$map | \$in  
// | \$concatArrays  
// | \$filter  
// | \$reduce

// Boolesche Operatoren  
// boolean op. = \$and | \$not | \$or

// Vergleichsoperatoren  
// Comparison op. = \$cmp | \$eq | \$gt | \$lt  
// | \$ne  
// | \$gte  
// | \$lte

// Konditionale Operatoren  
// Conditional op. = \$cond | \$ifNull  
// | \$switch

// Aggregatoperatoren  
// Accumulator = \$addToSet | \$avg | \$max  
// | \$mergeObjects  
// | \$min  
// | \$push  
// | \$sum

196

| Operator                              | Beschreibung                                                                                                                                      | Referenz |
|---------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------|----------|
| <code>\$abs</code>                    | Der Abs Operator wird zur Berechnung des Betrags einer numerischen Werte verwendet.                                                               | 20       |
| <code>\$add, \$subtract</code>        | Der Add Operator wird verwendet um die Summe 2er oder mehrerer Zahlen zu berechnen.                                                               | 20       |
| <code>\$addToSet, \$push</code>       | Der AddToSet bzw. Push Operator ermöglicht die gruppierten Dokumentdaten in einem Array zu verdichten.                                            | 20       |
| <code>\$and, \$or, \$not</code>       | Boolesche Operatoren werden verwendet um Bedingungen miteinander zu verknüpfen.                                                                   | 20       |
| <code>\$arrayElemAt</code>            | Der ArrayElemAt Operator extrahiert das Arrayelement mit einem bestimmten Index aus einem Array.                                                  | 20       |
| <code>\$cmp</code>                    | Mit der Hilfe des Scmp Operators können 2 Werte miteinander verglichen werden. Das Ergebnis des Vergleichs ist ein Zahlenwert.                    | 20       |
| <code>\$concatArrays</code>           | Mit dem SconcatArrays Operator können Elemente aus unterschiedlichen Arrays in ein Zielarray übernommen werden.                                   | 20       |
| <code>\$cond</code>                   | Der Cond Operator ermöglicht die kontextbasierte Verarbeitung von Daten in Dokumenten.                                                            | 20       |
| <code>\$divide</code>                 | Der Divide Operator wird verwendet um die Division 2er Zahlen zu berechnen.                                                                       | 20       |
| <code>\$eq, \$ne</code>               | Vergleichsoperatoren sind zweistellige logische Operatoren. Vergleichsoperatoren werden vor allem in mathematischen Gleichungen verwendet.        | 20       |
| <code>\$filter</code>                 | Mit dem Filter Operator können die Elemente eines Arrays nach bestimmten Kriterien gefiltert werden.                                              | 20       |
| <code>\$gt, \$gte, \$lt, \$lte</code> | Vergleichsoperatoren sind zweistellige logische Operatoren. Vergleichsoperatoren werden vor allem in mathematischen Gleichungen verwendet.        | 20       |
| <code>\$in</code>                     | Mit dem In Operator kann geprüft werden ob ein Element in einem Array enthalten ist.                                                              | 20       |
| <code>\$map</code>                    | Der Map Operator wird verwendet um die Elemente eines Arrays zu verarbeiten.                                                                      | 20       |
| <code>\$max, \$min, \$avg</code>      | Mit dem Smax bzw. Smin Operator wird für einen bestimmtes Attribut der maximale bzw. minimale Wert auf Collectionebene bzw. Arrayebene ermittelt. | 20       |
| <code>\$multiply</code>               | Der Smultiply Operator wird verwendet um das Produkt 2er oder mehrerer Zahlen zu berechnen.                                                       | 20       |
| <code>\$reduce</code>                 | Mit dem Sreduce Operator können die Elemente eines Arrays auf einen einzelnen Wert verdichtet werden.                                             | 20       |
| <code>\$switch</code>                 | Der Scond Operator ermöglicht die kontextbasierte Verarbeitung von Daten in Dokumenten.                                                           | 20       |

Abbildung 77. Expression: Operatoren

Informationssystem

| Operator                       | Beschreibung                                                                               | Referenz |
|--------------------------------|--------------------------------------------------------------------------------------------|----------|
| <code>\$abs</code>             | Der Abs Operator wird zur Berechnung des Betrags eines numerischen Wertes verwendet.       | 20       |
| <code>\$add, \$subtract</code> | Der Add Operator wird verwendet um die Summe 2er oder mehrerer Zahlen zu berechnen.        | 20       |
| <code>\$multiply</code>        | Der Multiply Operator wird verwendet um das Produkt 2er oder mehrerer Zahlen zu berechnen. | 20       |

**Abbildung 78.** Expressions: Arithmetische Operatoren

## 22.9. Arithmetische Operatoren

Arithmetische Operatoren

Arithmetische Operatoren verwenden **numerische Werte** als Operanden und geben einen einzelnen numerischen Wert zurück.

### 22.9.1 \$add, \$subtract, \$multiply Operator

Der \$add Operator bzw. \$subtract wird zur Berechnung der Addition bzw. Subtraktion 2er oder mehrerer Werte verwendet.

## ► Syntax: \$arithmetische Operatoren ▾

```

1 //]
2 // Syntax: Arithmetische Operatoren
3 //-
4 <arithmetic operator> : [
5   <expression1>, <expression2>, ...
6 ]
7
8 db.projects.aggregate([
9   { $project : {
10     researchFocus : {
11       $add : [
12         "$theoreticalResearch",
13         "$focusResearch",
14         "$appliedResearch"
15       ]
16     }
17   },
18   { $out : subprojectReport
19 }
20 }
21 ]);
22 });
23
24 $addFields : [
25   projectPending : {
26     $multiply : [
27       {
28         $add : ["$projectFunding", 10],
29         3
30       }
31     ],
32     $out : projectReport
33   }
34 ]);

```

200

| Operator                 | Beschreibung                                                                                                                               | Referenz |
|--------------------------|--------------------------------------------------------------------------------------------------------------------------------------------|----------|
| \$eq, \$ne               | Vergleichsoperatoren sind zweistellige logische Operatoren. Vergleichsoperatoren werden vor allem in mathematischen Gleichungen verwendet. | 201      |
| \$gt, \$gte, \$lt, \$lte | Vergleichsoperatoren sind zweistellige logische Operatoren. Vergleichsoperatoren werden vor allem in mathematischen Gleichungen verwendet. | 201      |

Abbildung 79. Expression: Vergleichsoperatoren

**22.9.2 \$divide Operator**

Der \$divide Operator wird zur Berechnung der Division 2er Zahlen verwendet.

**Syntax: \$divide Operator**

```
//-----
// Syntax: $divide Operator
//-----
$divide : [
  <expression1>, <expression2>
]
```

**22.9.3 \$abs Operator**

Der \$abs Operator wird zur Berechnung des Betrags eines numerischen Wertes verwendet.

**Syntax: \$abs Operator**

```
//-----
// Syntax: $abs Operator
//-----
$abs : <expression>

db.projects.aggregate([
  {
    $project : {
      projectFunding : {
        $abs : "$projectFunding"
      },
      title : 1
    },
    $out : projectReport
  }
]);
```

**22.10. Vergleichsoperatoren**

**Vergleichsoperatoren**

Vergleichsoperatoren sind logische Operatoren. Vergleichsoperatoren werden vor allem in mathematischen Gleichungen verwendet.

**22.10.1 Vergleichsoperatoren**

**Syntax: Vergleichsoperator**

```
//-----
// Syntax: $lt, $gt, $lte, $gte, $eq, $ne
//-----
$comparison operator : [
  <expression1>, <expression2>
]
```

201

**22.11. Boolesche Operatoren**

**Boolesche Operatoren**

Boolesche Operatoren werden verwendet um Bedingungen miteinander zu verknüpfen.

**22.11.1 \$and, \$or Operatoren**

Die \$and und \$or Operatoren werden verwendet um Bedingungen miteinander zu verknüpfen.

**Syntax: \$and, \$or Operator**

```
//-----
// Syntax: $and, $or
//-----
<boolean op> : [
  <expression>, <expression>, ...
]
```

```
//-----
// Beispiel: $and, $or
//-----
db.projects.aggregate([
  {
    $addFields : {
      projectFunding : {
        $sum : "$fundings.amount"
      }
    }
  },
  {
    $match : {
      $expr : {
        $and : [
          {
            $eq : [
              "$projectFunding", 5000
            ]
          },
          {
            $eq : [
              "$projectState", "END"
            ]
          }
        ]
      }
    }
  },
  {
    $out : projectReport
  }
]);
```

**22.11.2 \$not Operator**

Der \$not Operator wird verwendet um boolesche Werte zu invertieren.

**Syntax: \$not Operator**

```
//-----
// Syntax: $not
//-----
$not : [ <expression> ]
```

**22.12. Kontrolloperatoren**

**Kontrolloperatoren**

Kontrolloperatoren ermöglichen eine kontextbasierte Verarbeitung der Daten eines Dokuments.

**22.12.1 \$cond Operator**

Der \$cond Operator ermöglicht die kontextbasierte Verarbeitung von Daten in Dokumenten.

**Syntax: \$cond Expression**

```
//-----
// Syntax: $cond
//-----
$cond : {
  if : <boolean-expression>,
  then : <true-case>,
  else : <false-case>
}

$cond : [
  <boolean-expression>,
  <true-case>, <false-case>
]
```

202

| Operator | Beschreibung                                                                                                                    | Referenz |
|----------|---------------------------------------------------------------------------------------------------------------------------------|----------|
| \$cmp    | Mit der Hilfe des \$cmp Operators können 2 Werte miteinander verglichen werden. Das Ergebnis des Vergleichs ist ein Zahlenwert. | 204      |
| \$cond   | Der \$cond Operator ermöglicht die kontextbasierte Verarbeitung von Daten in Dokumenten.                                        | 203      |
| \$switch | Der \$switch Operator ermöglicht die kontextbasierte Verarbeitung von Daten in Dokumenten.                                      | 203      |

Abbildung 80. Expression: Kontrolloperatoren

Codebeispiel: \$cond Expression ▾

```
1 //-----
2 // Expression: $cond
3 //-----
4 db.projects.aggregate([
5   { $andField : {
6     verified : {
7       $cond : {
8         if : {
9           $eq : ["$state", "APP"]
10        },
11        then : true,
12        else : false
13      }
14    }
15  }
16);
17
18 // Kurzform - $cond Operator
19 db.projects.aggregate([
20   { $andField : {
21     verified : {
22       $cond : [
23         {
24           "$eq : ['$state', 'APP']",
25           true, false
26         }
27       ],
28     },
29     $project : {
30       title : 1, type : 1, state : 1
31     }
32   }
33 });
34 );
```

Syntax: \$switch Expression ▾

Der \$switch Operator ermöglicht die bedingte Verarbeitung der Daten eines Dokuments.

```
1 //-----
2 // Syntax: $switch
3 //-----
4 $switch : {
5   branches : [
6     { case : <exp>, then: <exp>},
7     { case : <exp>, then: <exp>},
8     ...
9   ],
10   default : <exp>
11 }
12
13 db.projects.aggregate({
14   $andFields : {
15     projectValue : {
16       $switch : {
17         branches : [
18           { case : {
19             $eq : [
20               "$type",
21               "REQUEST_PROJECT"
22             ]
23           },
24           then : 10
25         }],
26         default : 0
27       }
28     }
29   }
30});
```

203

Informationssysteme

Operator Beschreibung Referenz

|                |                                                                                                                  |     |
|----------------|------------------------------------------------------------------------------------------------------------------|-----|
| \$concatArrays | Mit dem \$concatArrays Operator können Elemente aus unterschiedlichen Arrays in ein Zielarray übernommen werden. | 204 |
| \$in           | Mit dem \$in Operator kann geprüft werden ob ein Element in einem Array enthalten sind.                          | 205 |
| \$map          | Der \$map Operator wird verwendet um die Elemente eines Arrays zu verarbeiten.                                   | 206 |
| \$filter       | Mit dem \$filter Operator können die Elemente eines Arrays nach bestimmten Kriterien gefiltert werden.           | 206 |

Abbildung 81. Expressions: Arrayexpressions

22.12.3 \$cmp Operator

Mit der Hilfe des \$cmp Operators können 2 Werte miteinander verglichen werden. Das Ergebnis des Vergleichs ist ein Zahlenwert.

Erklärung: \$cmp Operator ▾

- Ist der erste Wert größer als der zweite ist das Ergebnis des Vergleichs 1.
- Ist der erste Wert kleiner dann ist das Ergebnis des Vergleichs -1.
- Sind die Werte gleich ist das Ergebnis 0.

Codebeispiel: \$cmp Expression ▾

```
1 //-----
2 // Beispiel: $cmp
3 //-----
4 db.tweets.insertMany([
5   { _id : 1, item : "abc1", qty: 300 },
6   { _id : 2, item : "abc2", qty: 200 }
7 ]);
8
9 //-----
10 // Expression: $cmp
11 //-----
12 db.tweets.aggregate([
13   {
14     $project : {
15       item : 1, _id : 0,
16       cmpTo250 : {
17         $cmp : [ "$qty", 250 ]
18       }
19     }
20   },
21 ]);
```

22.13. Arrayexpressions

Arrayoperatoren ▾

Arrayoperatoren werden zur Verarbeitung von Arrays verwendet.

22.13.1 \$arrayElemAt Operator

Der \$arrayElemAt Operator extrahiert das Arrayelement mit einer bestimmten Index aus einem Array.

Syntax: \$arrayElemAt ▾

```
1 //-----
2 // Syntax: $arrayElemAt
3 //-----
4 $arrayElemAt : [
5   { $param idx: Der Index des gewünschten Elements. Für negative Werte wird das Array von letzten Element weg durchlaufen. },
6   { $array, $idx>
7 ]
```

db.projects.aggregate([
8 { \$project : {
9 \_id: 0, title : 1,
10 funding : {
11 \$arrayElemAt : [ "\$funding", 0 ]
12 }
13 }
14 });

204

**22.13.2 \$concatArrays Operator**

Mit dem \$concatArrays Operator können Elemente aus unterschiedlichen Arrays in ein Zielarray übernommen werden.

**> Syntax: \$concatArrays**

```
1 //-----
2 // Syntax: $concatArrays
3 //-----
4 $concatArrays : [ <array1> , <array2> ]
5
6 //-----
7 // Beispiel: $concatArrays
8 //-----
9 db.warehouse.insertMany([
10   {
11     instock : ["chocolate"],
12     ordered : ["butter", "apples"]
13   }, {
14     instock : ["apples", "pudding", "pie"]
15   }, {
16     instock : ["ice cream"],
17     ordered : []
18   },
19 ])
20
21 db.warehouse.aggregate([
22   { $project : {
23     items : {
24       $concatArrays : [
25         "$instock", "$ordered"
26       ]
27     }
28   },
29   { $out : "warehouseReport"
30   }
31 });
32
33 > Ausage
34
35 {
36   items : [
37     "chocolate", "butter", "apples"
38   ],
39   items : null
40 }, {
41   items : [ "ice cream" ]
42 }
43 }
```

**22.13.3 \$in Operator**

Mit dem \$in Operator kann geprüft werden ob ein Element in einem Array enthalten ist.

**> Syntax: \$in Expression**

```
1 //-----
2 // Syntax: $in Operator
3 //-----
4 $in: [ <expression>, <array expression> ]
5
6 //-----
7 // Beispiel : $in Operator
8 //-----
9 db.warehouse.insertMany([
10   {
11     location : "24th Street",
12     in_stock : [ "apples", "oranges" ]
13   },
14   {
15     location : "36th Street",
16     in_stock : [ "bananas", "pears" ]
17   },
18   {
19     location : "82nd Street",
20     in_stock : [ "apples" ]
21   }
22 ]);
23
24 db.warehouse.aggregate([
25   { $project : {
26     storelocation : "$location",
27     bananaInStock : {
28       $in : [
29         "bananas", "$in_stock"
30       ]
31     }
32   },
33   { $out : "warehouseReport"
34   }
35 });
36
37 > Ausgabe
38
39 {
40   storeLocation : "24th Street",
41   bananaInStock : true
42 },
43 {
44   storeLocation : "36th Street",
45   bananaInStock : true
46 },
47 {
48   storeLocation : "82nd Street",
49   bananaInStock : false
50 }
```

205

**22.13.4 \$map Operator**

Der \$map Operator wird verwendet um die Elemente eines Arrays zu verarbeiten.

Vergleichen Sie den \$map Operator vom Konzept mit der foreach Schleife.

**> Syntax: \$map Operator**

```
1 //-----
2 // Syntax: $map Operator
3 //-----
4 /**
5 * @Param input: Ein Ausdruck der zu einem
6 * Array evaluiert.
7 *
8 * @Param as: Optional. A name for the
9 * variable that represents each
10 * individual element of the array.
11 * If no name is specified, the
12 * variable name defaults to this.
13 *
14 * @Param in: An expression that is applied
15 * to each element of the input array.
16 */
17 $map: {
18   input: <expression>,
19   as: <string>,
20   in: <expression>
21 }
22
23 //-----
24 // Beispiel: $map Operator
25 //-----
26 db.results.insertMany([
27   { _id : 1, values : [ 5, 6, 7 ] },
28   { _id : 2, values : [ ] },
29   { _id : 3, values : [ 3, 8, 9 ] }
30 ]);
31
32 db.results.aggregate([
33   { $project : {
34     adjustedValues : {
35       $map : {
36         input : "$values",
37         as : "value",
38         in : { $add : [ "$$value", 2 ] }
39       }
40     }
41   }
42 });
43
```

**22.13.5 \$filter Operator**

Mit dem \$filter Operator können die Elemente eines Arrays nach bestimmten Kriterien gefiltert werden.

**> Syntax: \$filter Expression**

```
1 //-----
2 // Syntax: $filter
3 //-----
4 /**
5 * @Param input: Ein Ausdruck der zu einem
6 * Array evaluiert.
7 *
8 * @Param as: Optional. A name for the
9 * variable that represents each
10 * individual element of the array.
11 * If no name is specified, the
12 * variable name defaults to this.
13 *
14 * @Param cond: expression that resolves
15 * to a boolean value used to determine
16 * if an element should be included
17 * in the output array.
18 */
19 $filter: {
20   input : <array>,
21   as : <string>,
22   cond : <expression>
23 }
24
25 //-----
26 // Beispiel: $filter
27 //-----
28 db.facilities.aggregate([
29   {
30     $addFields : {
31       $filter : {
32         input : "$projects",
33         as : "project",
34         cond : {
35           $ne : [
36             "$$project.type",
37             "$MANAGEMENT_PROJECT"
38           ]
39         }
40       }
41     }
42   },
43   { $out : "facilityReport"
44 }
45 });
46
```

206

**22.13.6 \$reduce Operator**

Mit dem \$reduce Operator können die Elemente eines Arrays auf einen einzelnen Wert verdichtet werden.

```
1 //-----
2 // Beispiel: $reduce Operator
3 //-----
4 db.getCollection("projects").aggregate([
5   {
6     $addFields : {
7       value : {
8         $reduce: {
9           input: ["a", "b", "c"],
10          initialValue: "",
11          in: {
12            $concat : [
13              "$$value", "$$this"
14            ]
15          }
16        }
17      }
18    }
19  * Param input: Ein Ausdruck der zu einem
20  * Array evaluiert.
21  *
22  * Param initialValue: The initial cumulative
23  * value set before in is applied to
24  * the first element of the input array.
25  *
26  * Param in: A valid expression that $reduce
27  * applies to each element in the
28  * input array in left-to-right order.
29  */
30 $reduce: {
31   input: <array>,
32   initialValue: <expression>,
33   in: <expression>
34 }
35 //-----
36 // Beispiel: $reduce Operator
37 //-----
38 {
39   $reduce: {
40     input: [ 1, 2, 3, 4 ],
41     initialValue: { sum: 5, product: 2 },
42     in: {
43       sum: {
44         $add: [
45           "$$value.sum", "$$this"
46         ]
47       },
48       product: {
49         $multiply: [
50           "$$value.product",
51           "$$value.product", "$$this"
52         ]
53       }
54     }
55   }
56 }
57 // Ergebnis ... { sum : 15, product : 48 }
58 });
59 
```

**22.13.7 \$isArray Operator**

Der \$isArray Operator wird verwendet um zu prüfen ob es sich bei einem Wert um ein Array handelt.

207

## Informationssysteme

| Operator            | Beschreibung                                                                                                                                           | Referenz |
|---------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------|----------|
| \$addToSet, \$push  | Der \$addToSet bzw. \$push Operator ermöglicht die Dokumentidaten der Gruppen in einem Array zu verdichten.                                            | 208      |
| \$max, \$min, \$avg | Mit dem \$max bzw. \$min Operatoren wird für einen bestimmtes Attribut der maximale bzw. minimale Wert auf Collectionsebene bzw. Arrayebene ermittelt. | 209      |
| \$sum               | Der \$sum                                                                                                                                              | 203      |

Abbildung 82. Expressions Operatoren

**22.14. Aggregateexpressions**

**Aggregatoperatoren**  
Aggregatoperatoren haben im Vergleich zu anderen Expressions Zugriff auf die Daten aller Dokumente im Dokumentenstrom der gegenwärtigen Pipelinestufe.

**22.14.1 Fallbeispiel: sales**

Für die nachfolgenden Abfragen bilden die folgenden Dokumente die Datensätze.

```
1 //-----
2 // Collection: sales
3 //-----
4 db.sales.insertMany([
5   {
6     item : "abc1", price : 10
7     category: "BOARD_GAME"
8   },
9   {
10     item : "jkl1", price : 20
11     category : "PC_GAME"
12   },
13   {
14     item : "sys1", price : 5
15     category : "BOOK",
16   },
17   {
18     item : "abc2*", price : 10
19     category : "BOOK",
20   },
21   {
22     category : "BOARD_GAME",
23     item : "abc3",
24     price : 10
25   }
26 ]);
27 
```

**22.14.2 \$addToSet, \$push Operator**

Der \$addToSet bzw. \$push Operator ermöglicht die gruppierten Dokumentdaten in einem Array zu verdichten.

**> Syntax: \$addToSet Expression**

```
1 //-----
2 // Syntax: $addToSet
3 //-----
4 $addToSet : <expression>
5
6 db.sales.aggregate([
7   {
8     $group: {
9       _id: "$category",
10      itemsSold: {
11        $addToSet: "$item"
12      }
13    }
14  },
15  {
16    $out : "salesReport"
17  }
18 ]);
19 
```

> Ausgabe

{ \_id : "BOARD\_GAME",

  itemsSold : [ "abc1", "abc3" ]

}, {

  \_id : "BOOK",

  itemsSold : [ "sys1", "abc2" ]

, {

  \_id : "PC\_GAME",

  itemsSold : [

    "jkl1"

  ]

}

206

**22.14.3 \$min, \$max, \$avg Operatoren**

Mit dem \$min bzw. \$max Operatoren wird für einen bestimmtes Attribut der maximale bzw. minimale Wert auf Collectionsebene bzw. Arrayebene ermittelt.

## ► Syntax: \$min, \$max, \$avg Operator ▾

```
1 //-----
2 // Syntax: $min, $max Operatoren
3 //-----
4 $min: <expression>
5 $max: <expression>
6 $avg: <expression>
7
8 // Beispiel: $min, $max Operatoren
9 //-----
10 db.sales.aggregate([
11   {
12     $group: {
13       _id: "$category",
14       minQuantity: { $min: "$quantity" },
15       maxQuantity: { $max: "$quantity" },
16       avgQuantity: { $avg: "$quantity" }
17     }
18   },
19   {
20     $sort: {
21       _id: 1
22     }
23   },
24   {
25     $out: "salesReport"
26   }
27 ]);
28 > Ausgabe
29
30 { _id: "BOARD_GAME",
31   minQuantity: 2,
32   maxQuantity: 10,
33   avgQuantity: 6
34 }, {
35   _id: "BOOK",
36   minQuantity: 5,
37   maxQuantity: 10,
38   avgQuantity: 7
39 }, {
40   _id: "PC_GAME",
41   minQuantity: 1,
42   maxQuantity: 1,
43   avgQuantity: 1
44 }
```

**22.14.4 \$sum Operator**

Mit dem \$sum Operator ist es möglich für ein bestimmtes Feld über die Dokumente einer Gruppe hinweg die Summe der Werte zu bilden.

## ► Codebeispiel: \$sum Operator ▾

```
1 //-----
2 // Syntax: $sum Operatoren
3 //-----
4 { $sum: <expression> }
5
6 //-----
7 // Beispiel: $sum Operatoren
8 //-----
9 db.products.aggregate([
10   {
11     $match: {
12       state: { $ne: "IN_APPROVEMENT" },
13       type: { $ne: "STIPENDIUM" }
14     }
15   },
16   {
17     $group: {
18       _id: "$type",
19       projectCount: {
20         $sum: 1
21       },
22       projects: {
23         $addToSet: "$title"
24       }
25     },
26     $out: "projectReport"
27   }
28 ]);
```

209

**22.15. Referenzausdrücke**

## Referenzausdrücke

Referenzausdrücke ermöglichen die Verwendung von Variablen im Aggregatframe-work.

**22.15.1 \$\$ Operator - Systemvariablen**

Die MongoDB Spezifikation definiert eine Reihe eigener Systemvariablen.

Für den Zugriff auf Variablen wird der \$\$ Operator verwendet.

## ► Codebeispiel: Systemvariablen ▾

```
1 //-----
2 // Systemvariablen: CURRENT
3 //-----
4 // Die CURRENT Systemvariable verweist in
5 // einer Pipelinestufe auf das gegen-
6 // waerige Dokument
7
8 // Bei der Formulierung eines Fieldpaths
9 // wird defaultmäig die CURRENT System-
10 // variable Pfade vorangestellt.
11
12 // Folgende Schreibweisen sind aquivalent:
13 // "$field" -> "$$CURRENT.<field>"
14
15 db.projects.aggregate([
16   {
17     $match: {
18       $expr: {
19         $eq: [
20           "$$CURRENT.projectType",
21           "MANAGEMENT_PROJECT"
22         ]
23       }
24     }
25   },
26   {
27     $sort: {
28       "$$CURRENT.projectType": -1
29     }
30   },
31   {
32     $out: "projectReport"
33   }
34 ]);
```

**22.15.2 \$expr Operator**

Ursprünglich erlaubt die \$match Pipelinestufe keine Verwendung von Expressions.

Durch die Verwendung des \$expr Operators ist es möglich Pipelinerausdrücke auch in der \$match Stufe zu verwenden.

## ► Syntax: \$expr Operator ▾

```
1 //-----
2 // Syntax: $expr
3 //-----
4 $expr: {
5   <expression>
6 }
7
8 //-----
9 // Beispiel: $expr
10 //-----
11 db.projects.aggregate([
12   {
13     $addFields: {
14       projectFunding: {
15         $sum: "$funding.amount"
16       }
17     }
18   },
19   {
20     $match: {
21       $expr: {
22         $lt: [
23           "$projectFunding", 5000
24         ]
25       }
26     }
27   },
28   {
29     $project: {
30       projectFunding: 1,
31       projectType: 1,
32       description: 1,
33       title: 1
34     }
35   },
36   {
37     $sort: {
38       projectFunding: 1,
39     }
40   },
41   {
42     $out: "projectReport"
43   }
44 ]);
```

210

**22.15.3 \$let Operator**

Der \$let Operator erlaubt die Definition von Variablen in Pipelinestufen:

**> Syntax: \$let Operator**

```

1 //-----
2 // Syntax: $let
3 //-----
4 /*
5 * @Param <vars>: Assignment block for the
6 * variables accessible in the in
7 * expression. The variable assign-
8 * ments have no meaning outside
9 * the in expression.
10 */
11 * @Param <in>: The expression to evaluate.
12 */
13 $let : {
14   vars : {
15     <var1> : <expression>,
16     <var2> : <expression>,
17     ...
18   },
19   in : <expression>
20 }
21 //-----
22 // Beispiel: $let
23 //-----
24 db.projects.aggregate([
25   {
26     $addFields : {
27       projectFunding : {
28         $let : {
29           vars : {
30             amount : {
31               $sum: "$$.."
32             },
33           },
34           in : {
35             $multiply : [
36               "$$amount", 1.2
37             ]
38           },
39         },
40       }
41     }
42   }
43 ],
44 );

```

**22.16. Mengenoperatoren****Mengenoperatoren**

Mengenoperatoren werden verwendet um Mengenoperationen auf Arrays durchzuführen.

**22.16.1 \$setDifference Operator**

Mit dem \$setDifference Operator wird die Differenzmenge der Elemente zweier Arrays ermittelt.

Die Differenzmenge zweier Mengen enthält alle Elemente, die in der ersten Menge enthalten sind und nicht in der zweiten.

**> Syntax: \$setDifference Expression**

```

1 //-----
2 // Syntax: $setDifference
3 //-----
4 $setDifference: [
5   <expression1>, <expression2>
6 ]
7 //-----
8 // Beispiel: $setDifference
9 //-----
10 $setDifference: [
11   [ "a", "b", "c" ], [ "b", "a" ]
12 ]
13 // Ausgabe ... [ "c" ]
14 //-----
15 $setDifference: [
16   [ "a", "b", "a" ], [ "b", "a" ]
17 ]
18 // Ausgabe ... []
19 //-----
20 $setDifference: [
21   [ "a", "b" ], [ [ "b", "a" ] ]
22 ]
23 // Ausgabe ... [ "a" ]
24 //-----
25 $setDifference: [
26   [ "a", "b" ], [ [ "a", "b" ] ]
27 ]
28 // Ausgabe ... [ "a", "b" ]
29

```

211

| Abrägemethoden    | Beschreibung                                                                                         | Selbe |
|-------------------|------------------------------------------------------------------------------------------------------|-------|
| \$setDifference   | Mit dem \$setDifference Operator wird die Differenzmenge der Elemente zweier Arrays ermittelt.       | 211   |
| \$setIntersection | Mit dem \$setIntersection Operator wird die Durchschnittsmenge der Elemente zweier Arrays ermittelt. | 212   |
| \$setUnion        | Mit dem \$setUnion Operator wird die Vereinigungsmenge der Elemente zweier Arrays ermittelt.         | 213   |
| \$setIsSubset     | Mit dem \$setIsSubset Operator wird ermittelt ob zwischen 2 Arrays eine Untermengenrelation besteht. | 213   |

Abbildung 83. Expression: Mengenoperationen

**> Codebeispiel: \$setDifference Operator**

```

1 //-----
2 // Beispiel: $setDifference
3 //-----
4 db.experiments.insertMany(
5   { A : [ "red", "blue", "green" ],
6     B : [ "red", "blue" ]
7   },
8   {
9     A : [ "red", "blue" ], B : [ ]
10   },
11   {
12     A : [ ], B : [ "red" ]
13   });
14
15 db.experiments.aggregate([
16   { $project: {
17     diff : {
18       $setDifference: [ "$B", "$A" ]
19     },
20     _id : 0, A : 1, B : 1
21   }
22 },
23 { A : [ "red", "blue", "green" ],
24   B : [ "red", "blue" ],
25   diff : [ "green" ]
26 }, {
27   A : [ "red", "blue" ],
28   B : [ ], diff : []
29 }, {
30   A : [ ],
31   B : [ "red" ], diff : [ "red" ]
32 }
33 );

```

**22.16.2 \$setIntersection Operator**

Mit dem \$setIntersection Operator wird die Durchschnittsmenge der Elemente zweier Arrays ermittelt.

Der Durchschnitt zweier Mengen ist als diejenige Menge definiert, die alle Elemente enthält, die in beiden Mengen vorhanden sind.

**> Syntax: \$setIntersection Expression**

```

1 //-----
2 // Syntax: $setIntersection
3 //-----
4 $setIntersection: [
5   <expression1>, <expression2>
6 ]
7 //-----
8 // Beispiel: $setIntersection
9 //-----
10 $setIntersection: [
11   [ "a", "b", "c" ], [ "b", "a" ]
12 ]
13 // Ausgabe ... [ "a", "b" ]
14 //-----
15 db.experiments.aggregate([
16   { $project: {
17     A: 1, B: 1,
18     commonToBoth: {
19       $setIntersection: [ "$A", "$B" ]
20     }
21   }
22 });

```

212

**22.16.3 \$setUnion Operator**

Mit dem \$setUnion Operator wird die Vereinigungsmenge der Elemente zweier Arrays ermittelt.

Die Vereinigung zweier Mengen ist die Menge, die alle Elemente enthält, die wenigstens in einer der beiden Mengen enthalten ist.

**> Syntax: \$setUnion Expression \***

```

1 //-----
2 // Syntax: $setUnion
3 //-----
4 $setUnion: [
5   <expression>, <expression2>
6 ]
7 //-----
8 // Beispiel: $setUnion
9 // -----
10 // -----
11 $setUnion: [
12   [ "a", "b", "c" ],
13   [ "b", "a" ]
14 ]
15 // Ausgabe ... [ "a", "b", "c" ]
16
17
18 $setUnion: [
19   [ "a", "b", "a" ], []
20 ]
21
22 // Ausgabe ... [ "a", "b" ]
23
24
25 $setUnion: [
26   [ "a", "b" ], [ [ "b", "a" ] ]
27 ]
28
29 // Ausgabe ... [ "a", "b", [ "a", "b" ] ]
30
31 //-----
32 // Codebeispiel: $setUnion
33
34 db.experiments.aggregate([
35   {
36     $project: {
37       union: {
38         $setUnion: [ "$$A", "$$B" ]
39       }
40     }
41   }
42 );

```

**22.16.4 \$setIsSubset Operator**

Mit dem \$setIsSubset Operator wird ermittelt ob zwischen 2 Arrays eine Untermengen Relation besteht.

**> Codebeispiel: \$setIsSubset Operator \***

```

1 //-----
2 // Beispiel: $setIsSubset Operator
3 // -----
4 $setIsSubset: [
5   <expression1>, <expression2>
6 ]
7 //-----
8 // Beispiel: $setIsSubset
9 // -----
10 // -----
11 $setIsSubset: [
12   [ "a", "b", "c" ],
13   [ "b", "a" ]
14 ]
15 // Ausgabe ... false
16
17
18 $setIsSubset: [
19   [ "b", "a" ], [ "a", "b", "c" ]
20 ]
21
22 // Ausgabe ... true
23
24
25 $setIsSubset: [
26   [ "a", "b", "c" ], []
27 ]
28
29 // Ausgabe ... false
30
31 //-----
32 // Codebeispiel: $setIsSubset
33
34 db.experiments.aggregate([
35   {
36     $project: {
37       A: 1,
38       B: 1,
39       subset: {
40         $setIsSubset: [ "$$A", "$$B" ]
41       }
42     }
43   }
44 );

```

213

# XML Technologien

August 19, 2020

Informationssysteme

23. Datenformat - XML

# 01

XML - Grundlagen

|                     |     |
|---------------------|-----|
| 01. XML Grundlagen  | 216 |
| 02. XML Elemente    | 218 |
| 03. XML Attribute   | 220 |
| 04. Wohlgeformtheit | 221 |
| 05. Namensräume     | 222 |
| 06. Logische Sicht  | 224 |

23.1. XML Grundlagen

XML Datenformat

XML ist ein grundlegendes Datenformat zur Darstellung hierarchisch strukturierter Daten. Hierarchisch strukturierte Daten bilden die Welt in Form einer hierarchischen Baumstruktur ab.

23.1.1 Einsatzgebiete von XML

XML ist ein Datenformat zur Darstellung hierarchisch strukturierter Daten.

Auflistung: Einsatzgebiete von XML

Daten austauschformat

XML ist ein Datenformat zum Austausch von Daten zwischen Anwendungen und Softwareplattformen. Neben JSON ist XML das wichtigste Daten austauschformat für Informationssysteme.

Anwendungskonfiguration

In der Anwendungskonfiguration gilt XML als de facto Standard für Informationssysteme.

Auszeichnungssprache XML

XML wurde mit der Motivation entwickelt eine universelle Auszeichnungssprache für das Internet zu schaffen. Eine Auszeichnungssprache definiert aus welchen Elementen ein Dokument bestehen kann. Die wohl bekannteste Auszeichnungssprache ist HTML.

23.1.2 Fallbeispiel: XML Dokumente

XML wird mit der Motivation entwickelt eine universale Auszeichnungssprache für das Internet zu schaffen.

→ Extensible Markup Language

316

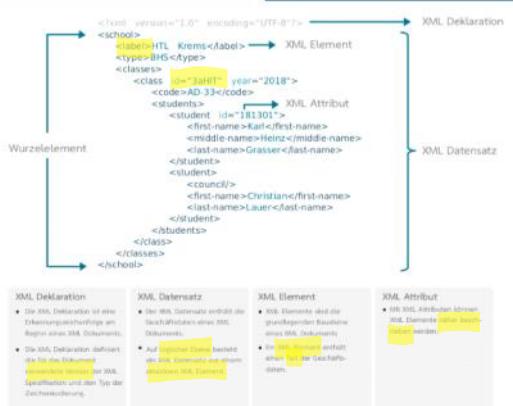
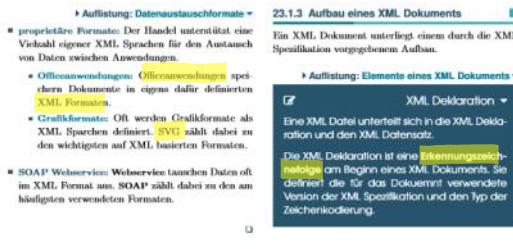
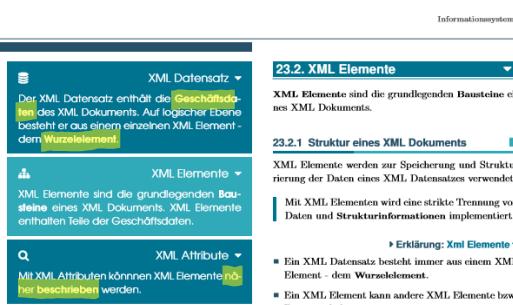


Abbildung 84. Aufbau eines XML Dokuments



23.1



■ Codebeispiel: XML Dokument

```

1  <!-- Beispiel-dokument -->
2  <!-- Beispiel-dokument -->
3  <!-- Wurzelement des Xml Dokuments -->
4  <?xml version="1.0" encoding="UTF-8"?>
5  <school>
6  <!-- Wurzelement des Xml Dokuments -->
7  ...
8  </school>
9
10 <?xml version="1.0" encoding="UTF-8"?>
11 <school>
12   <classes>
13     <class id="181301" year="2019">
14       <student id="181301">
15         <first-name>Karl</first-name>
16         <middle-name>Heinz</middle-name>
17         <last-name>Grasser</last-name>
18       </student>
19       <student id="181303">
20         <first-name>Franz</first-name>
21         <last-name>Hofner</last-name>
22       </student>
23     </classes>
24   </school>
25 
```

□



Abbildung 85. Inhaltstypen von XML Elementen

**23.2.2 Aufbau eines XML Elements**

Ein XML Element besteht immer aus einem Start- und einem Endtag. Die Bezeichnung des Elements kann dabei beliebig sein.

Erklärung: XML Element

- Beispiel <first-name>John</first-name>
- Das XML Element besteht aus dem Starttag <first-name> und dem dazugehörigen Endtag </first-name>.
- Das Element speichert den Token John.

**23.2.3 Inhaltstypen von XML Elementen**

XML Elemente können andere XML Elemente bzw. Daten enthalten.

Aufzählung: Inhaltstypen von XML Elementen

- S** unstrukturierter Inhalt
- Das XML Element enthält Daten in Form einer Zeichenkette. Der Inhaltstyp des XML Elements wird als **unstrukturiert** eingestuft.

kein  
2 an  
A. Schul

**23.3.1 XML Attribute - Grundlagen**

Mit XML Attributen können XML Elemente näher beschrieben werden.

**23.3.2 Vergleich: Elemente vs. Attribute**

XML Elemente und XML Attribute sind die grundlegenden Artefakte eines XML Dokuments.

Analyse: Elemente vs. Attribute

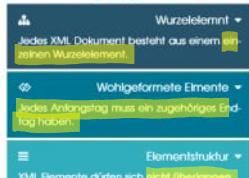
- Attribute können Daten nur in Form von Zeichenketten speichern. XML Elemente differenzieren hingegen unterschiedliche Inhaltstypen.
- Die Reihenfolge in der Attributliste in einem XML Elementen auftreten sind belanglos. Die Reihenfolge in der XML Elemente in einem XML Dokument auftreten ist signifikant.
- Ein XML Attribut kann auch XML Element dargestellt werden.

### 23.4. Wohlgeformte XML Dokumente

Das XML Datenformat definiert eine Reihe von Regeln für die Zusammensetzung eines XML-Dokuments.

#### 23.4.1 Wohlgeformtes XML Dokumente

##### » Auflistung: Wohlgeformte XML Elemente



#### 23.4.2 Wurzelelement

Jedes XML-Dokument hat genau ein Wurzelelement.

##### » Codebeispiel: Wurzelement

```

<!-- ----->
<!-- Beispiel-dokument ----->
<!-- ----->
<!-- XML-Dokument mit Wurzelelement ----->
<?xml version="1.0"?>
<name id="1232345" nickname="Shiny John">
  <first>John</first>
  <last>Doe</last>
</name>

<!-- XML-Dokument ohne Wurzelelement ----->
<?xml version="1.0"?>
<name id="1232345" nickname="Shiny John">
  <first>John</first>
  <middle>Pitgerald</middle>
  <last>Doe</last>
</name>

<name id="5672" nickname="Stinky Gordon">
  <first>Gordon</first>
  <last>Ramsey</last>
</name>

```

##### » Codebeispiel: Anfangstag und Endtag

```

<!-- ----->
<!-- Beispiel-dokument ----->
<!-- ----->
<?xml version="1.0"?>
<student id="1232345" nickname="Shiny John">
  <first>John</first>
  <last>Silver</last>
  </name>
  <course>Informationssysteme</course>
</student>

<!-- HTML folgt nicht der XML-Syntax ----->
<html>
  <body>
    <p>Text</p>
    <br>More text in the same paragraph
    <p>Some text in another paragraph</p>
  </body>
</html>

```

##### » Codebeispiel: Überlappung von XML Elementen

```

<!-- ----->
<!-- Beispiel-dokument ----->
<!-- ----->
<?xml version="1.0"?>
<name id="1232345" nickname="Shiny John">
  <first>John</first>
  <last>Loki</last>
</name>

<!-- HTML folgt nicht der XML-Syntax ----->
<html>
  <body>
    <p>Some <strong>formatted<em>text</em></strong>
      , but </em> no grammar no good!
    </body>
</html>

```

23.4

Informationssysteme

### 23.5. XML Namensräume

**XML-Namensraum**

Ein XML-Namensraum ist ein logischer Namensraum mit dem XML-Elementen einem logischen Kontext zugeordnet werden können.

Innenhalb eines Namensraums müssen die Namen von XML-Elementen eindeutig sein.

##### » Analyse: XML-Namensraum

- Das Konzept eines XML-Namensraums kann mit dem Namespace von C#-Klassen verglichen werden.
- Ein XML-Namensraum ist ein logischer Namensraum mit den XML-Elementen einem logischen Kontext zugeordnet werden können.

#### 23.5.1 Namenskonflikte

Im folgenden Beispiel haben die beiden `title`-Elemente zwar denselben Namen, beschreiben aber unterschiedliche Konzepte.

Namensräume helfen XML-Elementen mit gleichem Namen aber unterschiedlicher Bedeutung zu differenzieren.

##### » Codebeispiel: XML-Dokument

```

<!-- ----->
<!-- Namensraum ----->
<!-- ----->
<?xml version="1.0"?>
<course>
  <title>Semantic Web</title>
  <description>Semantic ...</description>
  <lecturers>
    <name>
      <title>Pirr.-Doz. Dr.</title>
      <first>Steffen</first>
      <last>Staab</last>
    </name>
    <name>
      <first>Gerald</first>
      <last>Putzschek</last>
    </name>
  </lecturers>
</course>

```

### 23.5.2 XML-Namensräume

##### » Erklärung: XML-Namensräume

- Zur Auflösung von Namenskonflikten werden XML-Elemente in Namensräume zusammengefasst.
- XML-Namensräume zeigen die logische Zusammengehörigkeit von XML-Elementen an.
- Namensräume werden durch eine URI identifiziert.
- Jedem Element des Namensraums wird ein Begriff vorangestellt um seine Zugehörigkeit zum Namensraum anzudeuten.

##### » Codebeispiel: Namensräume

```

<!-- ----->
<!-- Namensraum ----->
<!-- ----->
<?xml version="1.0"?>
<!DOCTYPE course [
  <!ELEMENT course (title, description, lecturers)>
  <!ELEMENT title (#PCDATA)>
  <!ELEMENT description (#PCDATA)>
  <!ELEMENT lecturers (#PCDATA)>
  <!ELEMENT name (#PCDATA)>
  <!ELEMENT first (#PCDATA)>
  <!ELEMENT last (#PCDATA)>
]>
<course>
  <title>Semantic Web</title>
  <description>Der Kurs Semantic Web behandelt die Prinzipien semistrukturierter Daten.</description>
  <lecturers>
    <name>
      <first>Gerald</first>
      <last>Putzschek</last>
    </name>
  </lecturers>
</course>

```

23.5

Dipl.Ing.- Msc. Paul Pashofler Bsc.

### 23.5.3 Namensraumdefinition

Namensraumdefinitionen müssen global eindringlich sein. XML Namensräume werden aus diesem Grund durch eine URI beschrieben.

#### ► Codebeispiel: Namensräume ▶

```

1 <! -- ----->
2 <!DOCTYPE HumanRaus ----->
3 <! ---->
4 <?xml version="1.0"?>
5 <course xmlns="http://www.html.at/course">
6   <@title>Semantic Web</@title>
7   <c:description>
8     Der Kurs Semantic Web behandelt die
9     semistrukturierten Daten.
10   </c:description>
11   <c:date>23.09.23 16:15:00</c:date>
12   <c:location
13     xmlns="http://www.html.at/location">
14     <loc:country>Austria</loc:country>
15     <loc:city>
16       <loc:name>Vienna</loc:name>
17       <loc:code>1040</loc:code>
18     </loc:city>
19     <loc:building>TU Vienna</loc:building>
20     <loc:room>Seminarraum 134</loc:room>
21   </c:location>
22   <c:lecturers>
23     <c:lecturer
24       xmlns="http://www.html.at/lecturer">
25       <@name>
26         <@title>Pirv.-Doz. Dr.</@title>
27         <@first>Steffen</@first>
28         <@last>Staab</@last>
29       </@name>
30       <c:contact>
31         <@email>s.staab@hhuugo.com</@email>
32         <@phone>0650/543467</@phone>
33       </c:contact>
34     </c:lecturer>
35   </c:lecturers>
36   <c:lecturer
37     xmlns="http://www.html.at/lecturer">
38     <@name>
39       <@title>Pirv.-Doz. Dr.</@title>
40       <@first>Steffen</@first>
41       <@last>Staab</@last>
42     </@name>
43     <c:contact>
44       <@email>s.staab@hhuugo.com</@email>
45     </c:contact>
46   </c:lecturer>
47 </c:courses>
```

### 23.5.4 Standard Namensraum

#### Standard Namensraum ▶

Der Standardnamensraum gilt für das Element indem er definiert wird, genau wie für alle Kindelemente dieses Elements.

Für einen Standardnamensraum wird kein Prefix definiert.

#### ► Codebeispiel: Standard Namensräume ▶

```

1 <!-- ----->
2 <!-- Standard Namensraum ----->
3 <!-- ----->
4 <?xml version="1.0"?>
5 <course xmlns="http://www.html.at/course">
6   <@title>Semantic Web</@title>
7   <c:description>
8     Der Kurs Semantic Web behandelt die
9     semistrukturierten Daten.
10   </c:description>
11   <c:location
12     xmlns="http://www.html.at/location">
13     <loc:country>Austria</loc:country>
14     <loc:location>
15       <c:lecturer
16         <@name>
17           <@title>Pirv.-Doz. Dr.</@title>
18           <@first>Steffen</@first>
19           <@last>Staab</@last>
20         </@name>
21         <c:contact>
22           <@email>s.staab@hhuugo.com</@email>
23         </c:contact>
24       </c:lecturer>
25       <c:lecturer
26         <@name>
27           <@title>Pirv.-Doz. Dr.</@title>
28           <@first>Steffen</@first>
29           <@last>Staab</@last>
30         </@name>
31         <c:contact>
32           <@email>s.staab@hhuugo.com</@email>
33         </c:contact>
34       </c:lecturer>
35     </c:lecturers>
36   </c:location>
37 </c:courses>
```

220

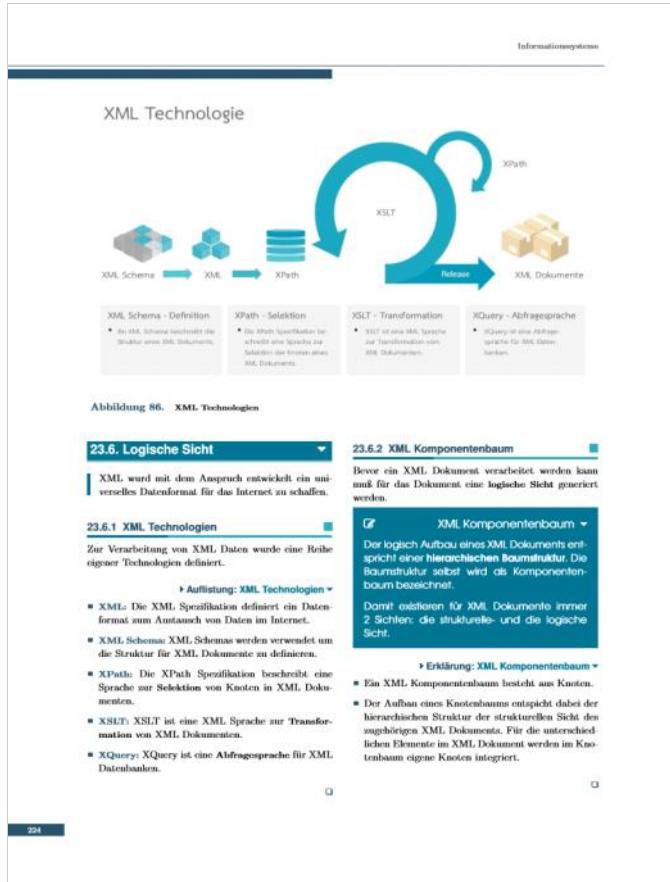


Abbildung 86. XML-Technologien

### 23.6. Logische Sicht

XML wurde mit dem Anspruch entwickelt, ein universelles Datenformat für das Internet zu schaffen.

### 23.6.1 XML Technologien

- XML ist eine Sprache, mit der Daten wurde eine formelle Spezifikation definiert.
  - Auflistung: XML Technologien
  - XML: Die XML Spezifikation definiert ein Dateiformat zum Austausch von Daten im Internet.
  - XML Schema: XML Schemata werden verwendet um die Struktur für XML Dokumente zu definieren.
  - XPath: Die XPath Spezifikation beschreibt eine Sprache zur Selektion von Knoten in XML Dokumenten.
  - XSLT: XSLT ist eine XML Sprache zur Transformation von XML Dokumenten.
  - XQuery: XQuery ist eine Abfragesprache für XML Datenbasen.

### 23.6.2 XML-Komponentenbaum

Bevor ein XML-Dokument verarbeitet werden kann, muß für das Dokument eine logische Sicht generiert werden.

- Der logisch Aufbau eines XML Dokuments entspricht einer **hierarchischen Baumanstruktur**. Die Baumanstruktur selbst wird als Komponentenbaum bezeichnet.
  - Damit existieren für XML Dokumente immer 2 Sichten: die strukturelle und die logische Sicht.

► Erklärung: XML Komponentenbaum ▶

  - Ein XML Komponentenbaum besteht aus Knoten:
  - Der Aufbau eines Knotenbaums entspricht dabei der hierarchischen Struktur der strukturierten Sicht des zugehörigen XML Dokuments. Für die unterschiedlichen Elemente im XML Dokument werden im Knotenbaum eigene Knoten integriert.



Abbildung 87. strukturelle vs. logische Sicht

**23.6.3 Komponentenknoten**

Jedes XML-Dokument besitzt eine Darstellung als Komponentenbaum.

## » Erklärung: Komponentenknoten «

- Jedes Element eines XML-Dokuments besitzt eine entsprechende Repräsentation als Knoten im Knotenbaum.

■ Die XML-Spezifikation definiert dabei für folgende Elementtypen eigene Knotentypen: XML Deklaration, Kommentare, XML-Elemente, XML-Attribute.

■ Tritt einer dieser Elemente in einem XML-Dokument auf, wird ein Knoten des entsprechenden Typs im Knotenbaum eingefügt.

■ Elemente die in einem XML-Element eingebettet sind werden als Kindknoten des entsprechenden Elementknotes dargestellt.

```
<!-- ----->
<!-- ----->
<!-- ----->
+ <class id="3aHIT" year="2018">
  + <code>AD-33</code>
</class>
```

Das <class>-Element besitzt im Knotenbaum eine Repräsentation als Elementknoten. Dem Elementknoten sind dabei folgende Kindknoten zugeordnet: Ein Elementknoten für das <code>-Element zusammen mit 2 Attributknoten für die Attribute des Elements.

■ Beachten Sie das Knoten unterschiedlicher Knotentypen Kindelemente eines Elementknoten sein können.

**23.6.4 Knotentypen**

Die XML-Spezifikation definiert die logische Sicht von XML-Daten folgende Knotentypen.

## » Auflistung: Knotentypen «

**Wurzelknoten**  
Der Wurzelknoten ist der primäre Knoten eines Knotenbaums. Der Wurzelknoten enthält alle anderen Knoten des Knotenbaums.

**Elementknoten**  
Elementknoten entsprechen der logischen Repräsentation eines XML-Elements. Der Elementknoten wird dem Elementknoten als Kindelement zugeordnet.

**Attributknoten**  
Attributknoten entsprechen der logischen Repräsentation eines XML-Attributes. Der Attributknoten wird dem Elementknoten als Kindelement zugeordnet.

**Textknoten**  
Die in einem Element enthaltenen Daten werden einem eigenen Textknoten zugeordnet. Der Textknoten wird dem entsprechendem Elementknoten als Kindelement zugeordnet.

**Namensraumknoten**  
Der Namensraum eines Elements ist ihm als Bruderknoten zugeordnet.

225

**24. Datenformat - XML/XPath****03**

XPath

01. XPath Konzepte	226
02. Pfadausdrücke in Kurzform	227
03. Lösungsobjekt	229
04. Prädikate	229
05. Pfadausdrücke in Standardform	230
06. XPath Funktionen	232
07. Fallbeispiel: XPath	235

**24.1. XPath - Konzepte**

**XPath - Selektion**  
XPath ist eine Adressiersprache zur Auswahl von Knoten in XML-Dokumenten.

Der XPath Standard dient als Grundlage für eine Reihe anderer XML-Techologien wie XSLT, XML-Schema bzw. XQuery.

## » Erklärung: XPath Grundlagen «

- XPath selbst ist keine XML-Sprache. Der XPath-Standard definiert eine eigene Syntax zur Formulierung von Pfadausdrücken.
- Mit einem XPath-Pfadausdruck kann die Position von Knoten in einem XML-Knotenbaum beschrieben werden.

## » Erklärung: Lokalisierungspfade «

- Bevor ein XPath-Ausdruck ausgewertet werden kann, muss für die entsprechenden XML-Daten die assoziierte logische Sicht bestimmt werden. Lokalisierungspfade werden stets in Relation zum Knotenbaum eines XML-Dokuments ausgewertet.
- Soll beispielsweise auf einen bestimmten Elementknoten eines XML-Dokuments zugegriffen werden, wird ein logischer Pfad angehend vom Wurzelknoten des Knotenbaums zum gewünschten Elementknoten definiert.
- Mit XPath-Lokalisierungspfaden kann dabei auf beliebige Teile der XML-Daten zugegriffen werden.
- Das Ergebnis der Auswertung eines Lokalisierungspfades wird als Lösungsobjekt bezeichnet. Lösungsobjekte können Knotenmengen, Strings, Zahlen bzw. booleische Werte sein.
- Die XPath-Spezifikation unterscheidet für Lokalisierungspfade 2 Formen: Pfadausdrücke in Standardschreibweise bzw. Pfadausdrücke in Kürzschreibweise.

226

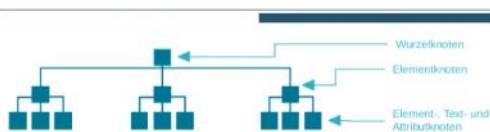


Abbildung 88. Knotenbaum eines XML-Dokuments

## 24.2 Pfadausdrücke in Kurzform

Der grundlegende Ausdruckstyp in XPath ist der Lokalisierungspfad.

### 24.2.1 Vereinfachte Pfadausdrücke

Die XPath Spezifikation erlaubt für Lokalisierungspfade eine vereinfachte Schreibweise: die **XPath Kurzform**.

- Bei der Auswertung von Pfadausdrücken in Kurzform trifft die XPath Engine eine Reihe von Annahmen.

Ihre kompakte Form schuldet die XPath Kurzform dabei den getroffenen Annahmen, das Selektionsverhalten des entsprechenden Pfadausdrucks ist in seiner Reichweite jedoch auf eine Zahl von Anwendungsfällen beschränkt.

#### \* Erklärung: XPath Kurzform \*

- Eine Lokalisierungspfad ist eine Folge von Lokalisierungspfaden, Lokalisierungspfaden sind voneinander durch den / Delimiter getrennt.

Je nach Knotentyp definiert die XPath Spezifikation einen eigenen Operatorn für die Knotenauswahl.

- // -----  
XPath: Lokalisierungspfade  
// -----  
// Selektion: <student> Elemente  
// Ergebnis: Alle <student> Elemente  
// -----  
XPath: /school/classes/class/student

### 24.2.2 Auswahl von Elementknoten

Die Position von Elementknoten in XML Dokumenten kann durch einfache XPath Pfadausdrücke beschrieben werden.

#### \* Erklärung: Knotenpfad \*

- Für die Auswahl von Elementknoten in XML Dokumenten wird ein logischer Pfad ausgehend vom Wurzelknoten des Knotenbaums zum gewünschten Elementknoten definiert.

Jeder Lokalisierungsschritt des Pfadausdrucks entspricht dabei einem der übergeordneten Elementknoten des gewünschten Elements.

#### \* Codebeispiel: Zugriff auf Elementknoten \*

```

1 // -----
2 // XPath: Pfadausdrücke
3 // -----
4 // Selektion: Alle <student> Elemente
5 // Ergebnis: Alle im XML Dokument
6 // enthaltene <student> Elemente
7 // -----
8 XPath: /school/classes/class/student
9 // -----
10 // Selektion: Alle <schule> Elemente
11 // Lösung: In den XML Daten sind keine
12 // <schule> Elemente enthalten. Die
13 // Lösungsmenge ist leer.
14 // -----
15 XPath: /schule
16 // -----
17 // Selektion: Alle <schule> Elemente
18 // Lösung: In den XML Daten sind keine
19 // <schule> Elemente enthalten. Die
20 // Lösungsmenge ist leer.
21 // -----
22 XPath: /schule

```

227

## 24.2.3 Kontextknoten

### Kontextknoten

Der **Kontextknoten**, ist jener Knoten eines XML Dokuments, der zum gegenwärtigen Zeitpunkt von der **XPath Engine** verarbeitet wird.

#### \* Erklärung: Kontextknoten \*

- Zur Referenzierung des Kontextknotens wird der Punktoperator verwendet.

Der Elternknoten des Kontextknotens kann über den .. Operator ausgewählt werden.

#### \* Codebeispiel: Punktoperatoren \*

```

1 // -----
2 // XPath: Kontextoperatoren
3 // -----
4 // Selektion: Alle <first-name> Kindelemente
5 // des Kontextknotens
6 // Lösungsknoten: Knotenmenge
7 XPath: ./first-name
8 // -----
9 // Selektion: Alle <first-name> Kindelemente
10 // des Elternknotens des Kontextknotens
11 // Lösungsknoten: Knotenmenge
12 XPath: ../name

```

## 24.2.5 Auswahl von Textknoten

Textknoten werden im XML Knotenbaum als Kindknoten von Elementknoten gespeichert.

#### \* Erklärung: Zugriff auf Textknoten \*

- Textknoten werden in XPath Lokalisierungspfaden über den text() Operator ausgewählt.

Das Lösungsknoten der text() Knotenabfrage ist eine Zeichenkette.

#### \* Codebeispiel: Zugriff auf Textknoten \*

```

1 // -----
2 // XPath: text() Knotentest
3 // -----
4 // Selektion: Alle Vornamen der Schueler
5 // Lösungsknoten: String
6 XPath: //student/first-name/text()

```

## 24.2.6 Auswahl von Attributknoten

Obwohl die XPath Spezifikation, Elementknoten als Attributknoten ihrer Attributknoten definiert, können Attributknoten nicht als Kindknoten eines Elementknotens angeprochen werden.

#### \* Erklärung: Zugriff auf Attribute \*

- Für den Zugriff auf die Attributknoten eines Elementknotens wird der @ Operator verwendet.

Das Ergebnis der Auswertung des @ Operators ist eine Knotenmenge von Attributknoten.

- Mit der Kombination des @ und des \* Operators kann auf alle Attributknoten eines Elements zugreifen werden.

#### \* Codebeispiel: Zugriff auf Attribute \*

```

1 // -----
2 // XPath: Descendant Pfadoperator
3 // -----
4 // Selektion: alle Vorkommen des <first-name>
5 // Elements im XML Dokument
6 XPath: //first-name
7 // -----
8 // Selektion: alle <first-name> Elemente
9 // die Kindelemente eines <student>
10 // Elements im XML Dokument
11 XPath: //student/first-name

```

#### \* Codebeispiel: Doppelter Pfadoperator \*

## 24.2.4 Descendant Pfadoperator

Der Descendant Pfadoperator ermöglicht einen Knotendurchlauf über alle Nachkommen des Kontextknotens.

#### \* Codebeispiel: Doppelter Pfadoperator \*

```

1 // -----
2 // XPath: Descendant Pfadoperator
3 // -----
4 // Selektion: alle Vorkommen des <first-name>
5 // Elements im XML Dokument
6 XPath: //first-name
7 // -----
8 // Selektion: alle <first-name> Elemente
9 // die Kindelemente eines <student>
10 // Elements im XML Dokument
11 XPath: //student/first-name

```

#### \* Codebeispiel: Zugriff auf Attribute \*

```

1 // -----
2 // XPath: @ Operator
3 // -----
4 // Selektion: die id Attribute aller Schueler
5 // Lösungsknoten: Knotenmenge
6 XPath: //student/@id
7 // -----
8 // Selektion: alle Attribute der Schueler
9 // Lösungsknoten: Knotenmenge
10 XPath: //student/@*

```

228

Dipl.Ing.- Msc. Paul Pashofner Bsc.

## 24.3. Lösungsobjekt

Das Ergebnis eines XPath Ausdrucks ist ein **Lösungsobjekt**.

### 24.3.1 Lösungsobjekt - Datentypen

Die XPath Spezifikation definiert 4 Typen von Lösungsobjekten.

→ Auflistung: Datentypen für Lösungsobjekte →

	Knotenmenge
Eine Knotenmenge ist eine Teilmenge der Knoten eines Knotenbaums.	
	String
Ein String ist eine Zeichenfolge. Die Zeichenfolge kann dabei auch leer sein. z.B.: "Hallo Welt"	
	Number
Number Objekte sind 64 Bit Fließkommasahlen. z.B.: 34	
	Boolean
Boolean Objekte nehmen entweder den Wert true oder false an.	

#### → Codebeispiel: Datentypen →

```

1 // ...
2 // XPath: Lösungsobjekte
3 // ...
4 // Lösungsobjekt: Knotenmenge
5 XPath: //student
6
7 // Lösungsobjekt: String
8 XPath: //student//first-name/text()
9
10 // Lösungsobjekt: Number
11 XPath: count(/person)
12
13 // Lösungsobjekt: Boolean
14 XPath: boolean(/project/title)
```

## 24.4. Prädikat

Zur Filterung der Knoten einer Knotenmenge können Prädikate definiert werden.

### 24.4.1 Definieren von Prädikaten

XPath Prädikat  
Ein Prädikat ist ein logischer Ausdruck.

→ Erklärung: Filtern mit Prädikaten →

- Prädikate werden in eckigen Klammern, am Ende einer Lokalisierungsstufe definiert. Die eckigen Klammern sind dabei ein Steuerzeichen der XPath Engine.
- Für eine XPath Lokalisierungsstufe kann eine beliebige Zahl von Prädikaten definiert werden.
- Werden für einen XPath Ausdruck mehrere Prädikate definiert, wird die Prädikate angebunden vom ersten Prädikat zum letzten hin ausgewertet.
- Prädikatausdrücke unterstützen die folgenden Vergleichsooperatoren: <, >, >=, <=, =, !=.
- Allerdings muss beachtet werden, dass Operatoren wie <, > nicht unmittelbar in einem XML Dokument erscheinen dürfen, sondern durch die entsprechenden Entityreferenzen &lt; bzw. &gt; ersetzt werden müssen.

#### → Codebeispiel: Prädikate →

```

1 // ...
2 // ...
3 // Prädikate
4 // ...
5 // Alle Projekte mit einer bestimmten id
6 /projects[8id='343225']
7
8 // Alle Personen die ein name Element haben
9 // person[name]
10
11 // Alle subprojekte die einen Partner haben
12 /project/partners/partner/subprojects/
13 subproject
14
15 // Die 3te Person einer Knotenmenge
16 // person[3]
17
18 // Alle Personen die eine id haben
19 // person[id]
```

Informationssysteme

Achse	Knotenabfrage	Prädikat																
/child::project//descendant::subproject	attribute::id='34256632676-3235-23'																	
Lokalisierungstufe	Lokalisierungstufe																	
<b>24.5. Pfadausdrücke in Standardform ▾</b>																		
Die XPath Spezifikation unterscheidet 2 Formen von Lokalisierungspfaden: Lokalisierungspfade in Standardschriftweise bzw. Lokalisierungspfade in Kurzform.																		
<p>Ein Lokalisierungspfad ist eine Folge von <b>Lokalisierungstufen</b>.</p>																		
<b>24.5.1 Lokalisierungstufen</b>																		
Eine Lokalisierungstufe besteht aus 3 möglichen Segmenten.																		
<p>► Syntax: <b>Lokalisierungstufe</b> ▾</p> <pre>1 // ... 2 // XPath: Syntax Lokalisierungstufe 3 // ... 4 Achse::Knotenabfrage[Prädikat][]{...}</pre>																		
<p>► Auflistung: Segmente einer Lokalisierungstufe ▾</p> <table border="1"> <tr> <td>dialect icon</td> <td>Achsenbezeichner ▾</td> </tr> <tr> <td colspan="2">Der Achsenbezeichner definiert die <b>Richtung</b> des <b>Knotendurchlaufs</b> einer Lokalisierungstufe. Das Ergebnis eines Knotendurchlaufs ist eine Knotenmenge .</td> </tr> <tr> <td colspan="2">Die Angabe eines Achsenbezeichners ist verpflichtend.</td> </tr> <tr> <td>icon</td> <td>Knotenabfrage ▾</td> </tr> <tr> <td colspan="2">Die Knotenabfrage ermöglicht eine <b>Vorauswahl</b> der durch den Knotendurchlauf bestimmten Knoten.</td> </tr> <tr> <td colspan="2">Die Angabe einer Knotenabfrage ist verpflichtend.</td> </tr> <tr> <td>Q icon</td> <td>Prädikat ▾</td> </tr> <tr> <td colspan="2">Prädikate erlauben die Formulierung komplexer <b>Filterbedingungen</b>.</td> </tr> </table>			dialect icon	Achsenbezeichner ▾	Der Achsenbezeichner definiert die <b>Richtung</b> des <b>Knotendurchlaufs</b> einer Lokalisierungstufe. Das Ergebnis eines Knotendurchlaufs ist eine Knotenmenge .		Die Angabe eines Achsenbezeichners ist verpflichtend.		icon	Knotenabfrage ▾	Die Knotenabfrage ermöglicht eine <b>Vorauswahl</b> der durch den Knotendurchlauf bestimmten Knoten.		Die Angabe einer Knotenabfrage ist verpflichtend.		Q icon	Prädikat ▾	Prädikate erlauben die Formulierung komplexer <b>Filterbedingungen</b> .	
dialect icon	Achsenbezeichner ▾																	
Der Achsenbezeichner definiert die <b>Richtung</b> des <b>Knotendurchlaufs</b> einer Lokalisierungstufe. Das Ergebnis eines Knotendurchlaufs ist eine Knotenmenge .																		
Die Angabe eines Achsenbezeichners ist verpflichtend.																		
icon	Knotenabfrage ▾																	
Die Knotenabfrage ermöglicht eine <b>Vorauswahl</b> der durch den Knotendurchlauf bestimmten Knoten.																		
Die Angabe einer Knotenabfrage ist verpflichtend.																		
Q icon	Prädikat ▾																	
Prädikate erlauben die Formulierung komplexer <b>Filterbedingungen</b> .																		
<p><b>24.5.2 Achsenbezeichner</b></p>																		
Achsenbezeichner definieren die <b>Richtung</b> eines Knotendurchlaufs in Lokalisierungstufen.																		
																		
<p>► Erklärung: <b>Achsenbezeichner</b> ▾</p> <ul style="list-style-type: none"> <li>■ Jede Lokalisierungstufe besteht aus einem Achsenbezeichner, einer Knotenabfrage und gegebenenfalls aus Prädikaten.</li> <li>■ Der Achsenname bestimmt die Richtung, in die der Lokalisierungspfad fortgesetzt werden soll.</li> <li>■ Das Ergebnis eines Knotendurchlaufs enthält alle Knoten der gewählten Achse relativ zum Kontextknoten.</li> </ul>																		
<p>► Auflistung: <b>Achsen</b> ▾</p> <ul style="list-style-type: none"> <li>■ self: Die Achse referenziert den Kontextknoten.</li> <li>■ child: Die Achse referenziert alle Kindknoten des Kontextknotens.</li> <li>■ parent: Die Achse referenziert den Elternknoten des Kontextknotens.</li> <li>■ descendant: Die Achse referenziert alle Nachkommen des Kontextknotens (Kinder, Kindeskinder etc.).</li> <li>■ ancestor: Die Achse referenziert alle Vorfahren des Kontextknotens inc. des Wurzelknotens.</li> <li>■ following: Die Achse referenziert alle Nachkommen des Kontextknotens in Dokumentreihenfolge.</li> <li>■ following-sibling: Die Achse referenziert alle nachfolgenden Geschwister des Kontextknotens.</li> <li>■ preceding-sibling: Die Achse referenziert alle vorhergehenden Geschwister des Kontextknotens.</li> <li>■ preceding: Die Achse referenziert alle vorhergehenden Knoten des Kontextknotens in Dokumentreihenfolge.</li> <li>■ attribute: Die Achse referenziert alle Attributknoten des Kontextknotens.</li> </ul>																		

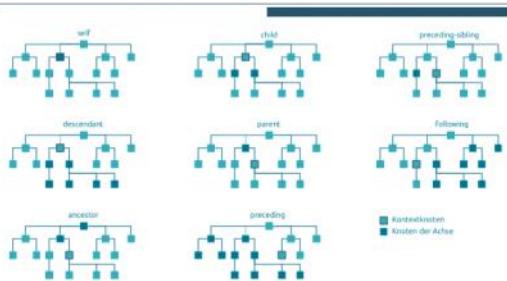


Abbildung 89. Achsen in XPath

## 24.5.3 Knotenabfragen

Mit einer Knotenabfrage werden die Knoten eines Knotendurchlaufs gefiltert.

## » Erklärung: Knotenabfrage «

- Die Knotenabfrage selbst wird an den Achsenzeichen angehängt und ermöglicht eine Vorauswahl, der durch den Knotendurchlauf bestimmten Knoten.
- Die Knotenabfrage besteht dabei entweder aus einem Elementvergleich bzw. dem Test auf einen bestimmten Knotentypen.
- Ein Elementvergleich wählt alle Knoten des Knotendurchlaufs aus, die einen bestimmten Namen haben.
- Knotentypentests ermöglichen eine Vorauswahl aller Knoten des Knotendurchlaufs die eine bestimmten Knotentypen haben. Die XPath Spezifikation definiert für Knotentypentests eine Reihe von Operatoren und Funktionen.
- Zur Formulierung komplexer Filterbedingungen können Prädikate<sup>5</sup> für XPath Lokalisierungstufen definiert werden.

## » Codebeispiel: Knotenabfrage «

```

1 // -----
2 // XPath: Knotenabfrage
3 // -----
4 // Selektion: Alle Elementknoten im Knoten-
5 // durchlauf
6 // Knotentesttyp: Knotentypentest
7 XPath: child::*
8
9 // Selektion: Alle <student> Elementknoten
10 // in Knotendurchlauf
11 // Knotentesttyp: Elementvergleich
12 XPath: parent::student
13
14 // Selektion: Alle Textknoten im Knoten-
15 // durchlauf
16 // Knotentesttyp: Knotentypentest
17 XPath: ancestor::text()
18
19 // Selektion: Alle Knoten des KD.
20 // Knotentesttyp: Knotentypentest
21 XPath: descendant::node()
22
23 // Selektion: Alle Kommentarknoten
24 // Knotentesttyp: Knotentypentest
25 XPath: parent::comment()

```

<sup>5</sup> siehe Kapitel Prädikate

## 24.5.4 Formen von XPath Ausdrücken

Für Lokalisierungstufen unterscheidet die XPath Spezifikation 2 Formen: die Standardschreibweise bzw. Lokalisierungstufen in Kurzform.



## » Vergleich: Kurz- vs. Standardform «

- Lokalisierungstufe können als Mischform der Standardschreibweise bzw. der Kurzform definiert werden.
- Beachten Sie das jede Lokalisierungstufe in Kurzform in einen äquivalenten XPath Ausdruck in Standardsform überführt werden kann. Der Umkehrschluss gilt jedoch nicht.

## » Codebeispiel: Transformationsregeln «

```

1 // -----
2 // XPath: Standardform vs. Kurzform
3 // -----
4 // Pfadausdrücke
5 Kurzform: classes/class
6 Standardform: child::classes/child::class
7
8 // Attributknoten
9 KP: //student/@id
10 SF: /descendant::student/attributes::id
11 Mischform: //student/attribute::id
12
13 KP: //student/*#
14 SF: /descendant::student/attribute::node()
15 MF: /descendant::student/*#
16
17 // Kontextknoten
18 KP: .
19 SF: self::node()
20
21 // Elternknoten
22 KP: ..
23 SF: parent::node()
24
25 // Textknoten
26 KP: //first-name/text()
27 SF: /descendant::first-name/child::text()
28 MF: //first-name/child::text()
29
30 KP: //last-name/text()
31 SF: /descendant::last-name/child::text()

```

## 24.6.2 Knotenmengenfunktionen

## » Auflistung: Knotenmengenfunktionen «

- Knotenmengenfunktionen:** ▾  
Knotenmengenfunktionen werden zur Verarbeitung von Mengen von Knoten eingesetzt.
- f** String-Funktionen: ▾  
String-Funktionen werden zur Verarbeitung von Zeichenketten eingesetzt.
- #** Numerische Funktionen: ▾  
Numerische Funktionen werden zur Verarbeitung von Zahlenwerten verwendet.
- g** Logische Funktionen: ▾  
Logische Funktionen werden zur Verarbeitung von boolischen Ausdrücken verwendet.

```

Codebeispiel: Knotenmengenfunktionen
1 // -----
2 // XPath Funktion: last
3 // -----
4 /project/subprojects/subproject[last()]
5 /* Ermittelt das letzten subproject Element
6 der Knotenmenge */
7
8 // -----
9 // XPath Funktion: count
10 // -----
11 count(/project/subprojects/subproject)
12
13 /* Ermittelt die Anzahl von Subprojekten in der
14 Knotenmenge */
15

Codebeispiel: Stringfunktionen
1 // -----
2 // XPath Funktion: string
3 // -----
4 xs:string string(
5   item elem
6 ) */

7
8 XPath: string(/project[@id='343225']/name)
9 Ergebnis: 'Simulation'
10
11 XPath: string(/project/@id)
12 Ergebnis: '343225'
13
14 XPath: string(2 = 2)
15 Ergebnis: 'true'
16
17 // -----
18 // XPath Funktion: concat
19 // -----
20 /* xs:string concat(
21   xs:anyAtomicType* token
22 ) */
23
24 XPath: concat('a', 'b', 'c')
25 Ergebnis: 'abc'
26
27 XPath: concat(
28   /person[last()]/first-name,
29   ' ',
30   /person[last()]/middle-name
31   ' ',
32   /person[last()]/last-name,
33 )
34
35 Ergebnis: Stefan Jell
36
37 // -----
38 // XPath Funktion: starts-with
39 // -----
40 /* xs:boolean starts-with(
41   xs:string token1, xs:string token2
42 ) */
43
44 XPath: starts-with('yes', 'yes')
45 Ergebnis: true
46
47 XPath: starts-with(/person[last()]/name, 'J')
48 Ergebnis: true

```

233

```

Codebeispiel: Stringfunktionen
1 // -----
2 // XPath Funktion: string-length
3 // -----
4 /* xs:integer string-length(
5   xs:string token
6 ) */
7
8 XPath: string-length('abc')
9 Ergebnis: 3
10
11 // -----
12 // XPath Funktion: contains
13 // -----
14 /* xs:boolean contains(
15   xs:string param1,
16   xs:string param2
17 ) */
18
19 XPath: contains('Shakespeare', 'spear')
20 Ergebnis: true
21
22 XPath: contains('Shakespeare', '')
23 Ergebnis: true
24
25 XPath: contains('', 'Shakespeare')
26 Ergebnis: false
27
28 XPath: contains(/person[last()]/name, 'J')
29 Ergebnis: true
30
31 // -----
32 // XPath Funktion: substring
33 // -----
34 /* xs:string substring(
35   xs:string param1,
36   xs:double param2,
37   xs:double param3
38 ) */
39
40 XPath: substring('abcde', 2)
41 Ergebnis: 'bcde'
42
43 XPath: substring('abcde', 2, 2)
44 Ergebnis: 'bc'
45
46 XPath: substring('abcde', 10, 2)
47 Ergebnis: ''
48
49 XPath: substring('abcde', 1, 20)

```

#### 24.6.4 Logische Funktionen

Logische Funktionen werden zur Verarbeitung boolescher Ausdrücke verwendet.

```

Auflistung: Logische Funktionen
■ boolean(): Wandelt das angegebene Objekt in einen logischen Wert um. Nichtleere Knotenmengen, nichtleere String-Werte und Zahlen größer ergeben wahr.
■ not(): Ergibt wahr, wenn das Objekt falsch ist.
■ true(): Gibt immer den Wert wahr zurück.
■ false(): Gibt immer den Wert falsch zurück.

Codebeispiel: Logische Funktionen
1 // -----
2 // XPath Funktion: boolean
3 // -----
4 /* xs:boolean boolean(
5   items()
6 ) */
7
8 XPath: boolean(/project[@id='1'])
9 Ergebnis: true
10
11 XPath: boolean(/project[@id='343225'])
12 Ergebnis: true
13
14 XPath: boolean(string(/project/name))
15 Ergebnis: true
16
17 // -----
18 // XPath Funktion: not
19 // -----
20 /* xs:boolean not(
21   item()
22 ) */
23
24 XPath: not(i = 1)
25 Ergebnis: false
26
27 XPath: not(*)
28 wahr wenn der Kontextknoten keine
29 Kindelemente hat
30
31 XPath: not(name='AI')
32 wahr wenn der Kontextknoten eine name
33 Element hat mit dem Wert AI

```

234

**24.7. Fallbeispiel: XPath**

Schreiben Sie für folgende Aufgabenstellung die gefragten XPath Ausdrücke.

**24.7.1 Datei: browser.xml****Codebeispiel: Eingabedaten**

```

1 <!-- ----->
2 <!-- Beispiel dokument: browser.xml -->
3 <!-- ----->
4 <?xml version="1.0" encoding="UTF-8"?>
5 <browser>
6   <tab id="t1">
7     <link url="http://www.orf.at"/>
8     <content>
9       Berichterstattung
10      </content>
11      <subpage id="s1" topic="wheater">
12        <link url="wheater"/>
13        <content>
14          Das Wetter fuer Oesterreich
15        </content>
16      </subpage>
17      <subpage id="s2" topic="sport">
18        <link url="sport"/>
19        <content>
20          Unglaubliche Sportnachrichten
21        </content>
22      </subpage>
23      <sublink id="s3" topic="society"/>
24    </tab>
25    <tab id="t2">
26      <link url="http://www.wrd.de/reisen"/>
27      <content>
28        Reiseberichte mit Tamia Karena
29      </content>
30      <subpage id="s5">
31        <link url="train"/>
32        <content>
33          Zugreisen in Europa
34        </content>
35      </subpage>
36    </tab>
37    <history>
38      <sub>s1</sub>
39      <sub>s3</sub>
40      <sub>s5</sub>
41    </history>
42 </browser>
```

**24.7.2 XPath Ausdrücke**

Aufgabenstellung XPath Ausdrücke:

**Codebeispiel: Lokalisierungspfade**

```

1 // ----->
2 // Beispiel dokument: browser.xml
3 // -----
4 1. Finden Sie alle <subpage> Elemente
5   /browser/subpage
6   //
7   //
8   2. Finden sie das letzte <subpage> Element
9     des t1 <tab>
10    /-----
11    //
12    /browser/tab[@id='t1']/subpage[last()]
13    //
14    //
15    3. Finden Sie alle <subpage> Elemente die
16      nicht in der <history> enthalten sind
17      //
18      //
19      //subpage[not(@id = //history/sub/text())]
20      //
21      //
22      4. Geben Sie die gesamte url der s1
23        <subpage> an
24        //
25        concat(/tab[@id='t1']/link/@url, '/')
26        //tab[@id='t1']/subpage[@id='s1']/link/@url
27        //
28        5. Geben Sie das erste <tab> aus das sich mit
29          der Berichterstattung befasst.
30          //
31          //tab[contains(content,
32            'Berichterstattung')][1]
33          //
34          6. Wie oft wurde <subpage> s5 vom User auf-
35            gerufen
36          //
37          count(/sub[@id='s5'])
38          //
39          7. Wie oft wurde <subpage> s5 vom User auf-
40            gerufen
41          count(/sub[@id='s5'])
42          //
43          count(/descendant::sub[self::node()=s5])
```

255

**25. Datenformat - XML XSLT**

03

## XSLT Stylesheets

01. XSL Grundlagen	236
02. XSL Transformationsprozess	237
03. XSL Programm	238
04. Dekontextliche Verarbeitung	240
05. Prozedurale Verarbeitung	242
06. Ausgabestream	245
07. Suchanfragen	247

**25.1. XSLT Grundlagen**

XSL Standard  
Der XSL Standard ist eine XML Sprache zur Transformation und Verarbeitung von XML Dokumenten.

Ein XSLT Programm beschreibt Regeln zur Transformation von XML Daten.

**25.1.1 XSLT Stylesheet**

Ein XSLT Programm wird als XSLT Stylesheet bezeichnet.

**Erklärung: XSLT Stylesheet**

- Ein XSLT Stylesheet selbst ist ein XML Dokument.
- Ein Stylesheet besteht dabei aus einer freien Abfolge von Templateregeln.
- Templateregeln beschreiben wie die Elemente eines XML Eingabedokument verarbeitet werden sollen.

**Codebeispiel: XSLT Stylesheet**

```

1 <!-- ----->
2 <!-- XSLT Stylesheet -->
3 <!-- ----->
4 <?xml version="1.0" encoding="UTF-8"?>
5 <xsl:stylesheet
6   xmlns:xsl="http://www.w3.org/Transform">
7   <xsl:template match="... ">
8   ...
9   ...
10  </xsl:template>
11  ...
12  <xsl:template match="... ">
13  ...
14  ...
15  </xsl:template>
16  ...
17  <xsl:template match="... ">
18  ...
19  ...
20  </xsl:template>
21  ...
22 </xsl:stylesheet>
```

256

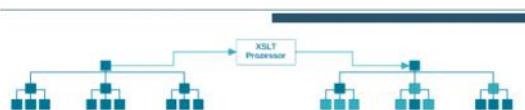


Abbildung 90. Transformation von XML-Dokumenten

<pre>  1 &lt;!--  2   &lt;!-- Input: greeting.xml  3   &lt;!--  4   &lt;!--  5   &lt;?xml version="1.0"?  6   &lt;greeting&gt;  7     Hello, World!  8   &lt;/greeting&gt;  9 10  &lt;!-- 11   &lt;!-- XSLT Program: greeting.xsl 12   &lt;!-- 13   &lt;?xml version="1.0" encoding="UTF-8"?&gt; 14   &lt;xsl:stylesheet 15     xmlns:xsl="http://www.w3.org/Transform" 16     version="2.0"&gt; 17 18     &lt;xsl:template match="greeting"&gt; 19       &lt;html&gt; 20         &lt;body&gt; 21           &lt;h1&gt; 22             &lt;xsl:value-of select="greeting"/&gt; 23           &lt;/h1&gt; 24         &lt;/body&gt; 25       &lt;/html&gt; 26     &lt;/xsl:template&gt; 27 28   &lt;/xsl:stylesheet&gt; 29 30   &lt;!-- 31   &lt;!-- Output: greeting.html 32   &lt;!-- 33   &lt;!-- 34 35   &lt;html&gt; 36     &lt;body&gt; 37       &lt;h1&gt;Hello, World!&lt;/h1&gt; 38     &lt;/body&gt; 39   &lt;/html&gt; </pre>	<p><b>25.2. XSLT Transformationsprozess</b></p> <p>Der XSLT Transformationsprozess besteht aus einer Folge von Schritten.</p> <h3>25.2.1 Transformationssschritte</h3> <ul style="list-style-type: none"> <li>➤ <b>Aufstellung: Transformationsschritte</b></li> <li>➤ <b>Initialisierungsschritt</b></li> <p>Die XSLT Engine lädt die XML Eingabedatei und das gewünschte Stylesheet.</p> <li>➤ <b>Logische Verarbeitung</b></li> <p>Die XSLT Engine generiert für die eingelesenen XML Daten eine Knotenbaumrepräsentation.</p> <li>➤ <b>Kontextschritt</b></li> <p>Um Knoten verarbeiten zu können müssen sie in den Kontext der XSLT Engine geladen werden. Es gibt 2 Möglichkeiten um Knoten in den Kontext zu laden:</p> <ul style="list-style-type: none"> <li>▪ Mit dem Abschluss der logischen Verarbeitung wird der Wurzelknoten der XML Eingabedatei in den Kontext geladen.</li> <li>▪ Durch den Aufruf der <code>apply-template</code> Anweisung können neue Knoten in den Kontext geladen werden.</li> </ul> <li>➤ <b>Verarbeitungsschritt</b></li> <p>Für jeden Knoten im Kontext bestimmt die XSLT Engine eine entsprechende Template Regel, um den Knoten zu verarbeiten.</p> </ul>
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

### 25.3. XSLT Programm

### 25.3.1 XSLT Stylesheet

 XSLT Stylesheet ▾

- Ein XSLT Programm wird als XSLT Stylesheet bezeichnet.

► Erklärung: XSLT Stylesheet ▼

- Ein XSLT Stylesheet selbst ist ein XML Dokument.
- Ein Stylesheet definiert eine freie Abfolge von Templateregeln. Templateregeln beschreiben wie die Elemente einer XML Eingabedatei verarbeitet werden sollen.

Jetzt selbst ein XML Dokument.

- Ein XSLT Stylesheet ist eine freie Abfolge von Template-Regeln
  - Ein StyleSheet definiert eine freie Abfolge von Template-Regeln, Template-Regeln beschreiben wie die Elemente einer XML Eingabedatei verarbeitet werden

dem XSLT Suchmuster und dem Template.

► Auflistung: Komponenten einer Template-Regel ▶

LT Suchmuster und dem Template.

- **Templateregeln** werden zur Verarbeitung von XML-Knoten verwendet.
  - Eine **Templateregel** bezieht sich dabei immer auf ein bestimmtes XML-Element der eingelesenen XML-Eingabedaten.
  - **Templateregeln** setzen sich aus 2 Teilen zusammen: dem XSLT-Schmutznamen und dem Template.

► Auflistung: Komponenten einer Templateregel ➤

2020 MOUNTAIN

Ein XSLT Suchmuster wird durch einen **XPath Lokalisierungspfad** beschrieben.

#### ► Erklärung: XSLT Suchmuster

- Suchmuster werden im Match Attribut eines <xsl:template> Elements definiert.
  - Mit einem Suchmuster wird bestimmt, welche Elementknoten der Eingabedatei, mit einer Template-Regel verarbeitet werden sollen.
  - Damit definiert ein Suchmuster eine **Relation** zwischen den Tempateregeln des XSLT Stylesheets und den XML-Elementen der Eingabedatei.
  - Für den Knotendurchlauf im XSLT Suchmuster kann nur auf die `child` bzw `descendant` Achse zurückgegriffen werden.
  - Im Match Attribut können auch mehrere Lokalisierungsgruppen getrennt durch ein `|` definiert werden.

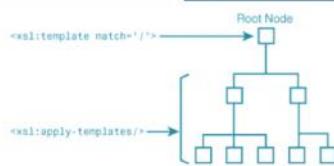


Abbildung 91. XSLT Transformationsprozess

## 25.3.4 Template Anweisungen

**Templates** enthalten Anweisungen zur Verarbeitung von XML-Daten.

Templates können 2 Arten von Anweisungen<sup>6</sup> enthalten: XSLT-Anweisungen und Literale.

## » Auflistung: Template Anweisungen »

## XSLT Anweisungen ▾

XSLT-Anweisungen sind Befehle die vom XSLT Prozessor ausgeführt werden. XSLT-Anweisungen sind im Stylesheet am xsl Namespace erkennbar.

## Literale ▾

Literale werden vom XSLT Prozessor nicht verarbeitet. Literale werden direkt in den Ausgabestrom des Transformationsprozesses geschrieben.

## » Analyse: Template Anweisungen »

■ XSLT-Anweisungen: Das folgende Beispiel enthält die folgenden XSLT-Anweisungen: stylesheet, output, template, apply-templates und value-of.

■ Literale: Die restlichen Anweisungen sind Literale. Literale werden direkt in den Ausgabestrom des Transformationsprozesses geschrieben.

## » Codebeispiel: Template »

```

<?xml version="1.0"?>
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet
    xmlns:xsl="http://www.w3.org/Transform"
    version="2.0">

    <xsl:output method="xml" indent="yes"/>

    <xsl:template match="/">
        <html>
            <body>
                <xsl:apply-templates select="/*"/>
            </body>
        </html>
    </xsl:template>

    <xsl:template match="greeting">
        <h1>
            <xsl:value-of select="greeting"/>
        </h1>
    </xsl:template>
</xsl:stylesheet>
```

## » Befehl

359

## 25.4. Deklarative Verarbeitung

## XSLT Transformationsprozess ▾

Der XSLT Transformationsprozess besteht im wesentlichen aus 2 sich wiederholenden Schritten:

- 1.Schritt: Schreibe die zu verarbeitenden Elemente in den Kontext der XSLT Engine
- 2.Schritt: Verarbeite die Elemente des Kontexts mit Templateregeln.

## 25.4.1 &lt;xsl:apply-templates&gt; Element

Mit der apply-templates Anweisung werden neue XML-Elemente in den Kontext der XSLT Engine geschrieben.

## » Erklärung: apply-templates Anweisung »

■ Mit der apply-templates Anweisung wird gesteuert, welche Elemente der Eingabedaten in den Kontext geschrieben werden sollen.

■ Die apply-templates Anweisung wird zur Steuerung des Transformationsprozesses verwendet.

## » Syntax: &lt;xsl:template&gt; Element

Ein XSLT Stylesheet ist eine freie Abfolge von Templateregeln.

## » Erklärung: Templateregeln »

■ Templateregeln werden in XSLT mit dem <xsl:template> Element definiert.

■ Stellt die XSLT Engine für einen Knoten im XSLT-Kontext eine Übereinstimmung mit dem Suchkriterium einer Templateregel fest, wird die entsprechende Templateregel geladen, um den Knoten zu verarbeiten.

■ Da Templateregel selbst hat nun Zugriff auf alle Daten des Knotens zusammen mit allen Parametern die im <xsl:apply-templates> Element definiert wurden sind.

## » Syntax: &lt;xsl:template&gt; »

```

<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet
    xmlns:xsl="http://www.w3.org/Transform">

    <xsl:apply-templates
        select = "Knotenbenennungsdruck"
        mode = "QName">
    </xsl:apply-templates>

    <!-- XSLT: apply-templates Anweisung -->
    <xsl:apply-templates
        select = "..."/>
    <xsl:with-param
        name = "type">
        ...
        <!-- bind parameter to element -->
        <xsl:apply-templates select="..."/>
        <xsl:with-param
            name = "type">
            ...
            <!-- Declare template parameter -->
            <xsl:template match="..."/>
            <xsl:param name="type"/>
        </xsl:with-param>
    </xsl:apply-templates>
    </xsl:with-param>
</xsl:apply-templates>

    <xsl:template match="..."/>
    <xsl:apply-template select="..."/>
</xsl:template>
</xsl:stylesheet>
```

360

**25.4.3 <xsl:with-param> Element**

Mit der `with-param` Anweisung können zusätzliche Informationen mit den Knoten im Kontext der XSLT Engine definiert werden.

» Erklärung: `with-param` Anweisung ▾

- Durch das Einbauen der `with-param` Anweisung in `<xsl:apply-templates>` Elementen können zusätzliche Informationen mit Elementen im Knotext assoziiert werden.
- Bei der Verarbeitung dieses Elements, hat die entsprechende Templateregel nun Zugriff auf die mit `with-param` Anweisung definierten Parameter.

» Syntax: `<xsl:with-param>` ▾

```

1  <!-- -----
2  <!-- Syntax: <xsl:with-param>      -->
3  <!-- -----
4  <xsl:with-param name = QName
5   select = Knotenmengenausdruck>
6  </xsl:with-param>
7
8  <!-- -----
9  <!-- XSLT: with-param      -->
10 <!-- -----
11 <xsl:stylesheet version="1.0" encoding="UTF-8">
12 <xsl:stylesheet
13   xmlns:xsl="http://www.w3.org/Transform">
14
15   ...
16   <!-- bind parameter to element -->
17   <xsl:apply-templates select="..."/>
18   <xsl:with-param name="type">
19     action
20     </xsl:with-param>
21     <xsl:apply-templates>
22   </xsl:template>
23
24   <xsl:template match="...">
25     <!-- fetch value from context -->
26     <!-- and bind it to param -->
27     <xsl:param name="type"/>
28
29     <!-- work with value -->
30     <xsl:value-of select="$type"/>
31   </xsl:template>
32
33 </xsl:stylesheet>
34
35 </xsl:stylesheet>

```

## » Codebeispiel: Template modi ▾

```

1  <!-- -----
2  <!-- XSLT: Template Modi      -->
3  <!-- -----
4  <xml version="1.0" encoding="UTF-8" >
5  <xsl:stylesheet
6    xmlns:xsl="http://www.w3.org/Transform">
7
8   <xsl:template match="/*">
9     <!-- Verarbeitung durch Template A -->
10    <xsl:apply-templates
11      select="/movie[1]"
12      mode="premiere"/>
13
14   <!-- Template A -->
15   <xsl:template match="movie"
16     mode="premiere">
17     ...
18   </xsl:template>
19
20   <!-- Template B -->
21   <xsl:template match="movie">
22     ...
23   </xsl:template>
24
25
26 </xsl:stylesheet>

```

241

**25.5 Prozedurale Verarbeitung**

Die XSLT Spezifikation definiert eine Reihe von Anweisungen zur prozeduralen Verarbeitung von XML Knoten.

**25.5.1 <xsl:variable> Element**

Für die Verarbeitung von Knoten unterstützt XSLT die Verwendung von Variablen.

## » Erklärung: Variablen Deklaration ▾

- Die XSLT Spezifikation definiert Variablen jedoch nur in einem sehr eingeschränkten Sinn.
- XSLT Variablen kann zwar ein Wert zugewiesen werden, der Wert der Variable kann im Laufe der weiteren Verarbeitung jedoch nicht geändert werden.
- Der Geltungsbereich einer Variable beschränkt sich dabei auf die Templateregel, indem sie definiert wurde.

» Syntax: `<xsl:variable>` ▾

```

1  <!-- -----
2  <!-- Syntax: <xsl:variable>      -->
3  <!-- -----
4  <xsl:variable name = QName select = Ausdruck
5   disable-output-escaping = "yes" | "no"
6  </xsl:variable>
7
8  <!-- -----
9  <!-- XSLT: variable Anweisung      -->
10 <!-- -----
11 <xml version="1.0" encoding="UTF-8" ?>
12 <xsl:stylesheet version="2.0">
13
14   <xsl:template match="...">
15     <xsl:value-of select="@id"/>
16     ...
17     <xsl:variable name="code"
18       select="*|java*"/>
19
20     <!-- Stringdarstellung -->
21     <xsl:value-of select="$code"/>
22   </xsl:template>
23
24 </xsl:stylesheet>

```

**25.5.2 <xsl:value-of> Element**

Mit der `value-of` Anweisung kann die Stringdarstellung von Variablen, Parametern bzw. Knoten ermittelt werden.

» Syntax: `<xsl:value-of>` ▾

```

1  <!-- -----
2  <!-- Syntax: <xsl:value-of>      -->
3  <!-- -----
4  <xsl:value-of select = Ausdruck
5   disable-output-escaping = "yes" | "no"
6  </xsl:value-of>
7
8  <!-- -----
9  <!-- XSLT: value-of Anweisung      -->
10 <!-- -----
11 <xml version="1.0" encoding="UTF-8" ?>
12 <xsl:stylesheet version="2.0">
13
14   <xsl:template match="...">
15     <xsl:value-of select="@id"/>
16     ...
17     <xsl:variable name="code"
18       select="*|java*"/>
19
20     <!-- Stringdarstellung -->
21     <xsl:value-of select="$code"/>
22   </xsl:template>
23
24 </xsl:stylesheet>

```

**25.5.3 <xsl:if> Element**

Die `if` Anweisung ermöglicht den Ablauf der Verarbeitung von Anweisungen innerhalb einer Templateregel zu steuern.

## » Erklärung: If Anweisung ▾

- Zur Formulierung von Bedingung wird dem im `<xsl:if>` Element definiertem `test` Attribut ein logischer Ausdruck zugeordnet.
- Kann der logische Ausdruck zu true evaluiert werden, werden die im `<xsl:if>` Element enthaltenen Anweisungen ausgeführt.
- Für Logische Ausdrücke gelten dabei dieselben syntaktischen Einschränkungen wie für Prädikate in XPath Ausdrücken.

242

## » Erklärung: Auswertung logischer Ausdrücke ▶

- Numer: Die numerischen Werte 0 bzw. NaN evaluieren zu false. Alle anderen Werte evaluieren zu true.
- node-set: Leere node-set evaluieren zu false. node-set mit Elementen evaluieren zu true.
- String: Ein leerer String evaluiert zu false. Alle anderen Strings evaluieren zu true.

## » Syntax: &lt;xsl:if&gt; ▶

```

1 <!-- ----- -->
2 <!-- Syntax: <xsl:if> -->
3 <!-- ----- -->
4 <xsl:if test = "logischer Ausdruck">
5   <!-- Inhalt: Template -->
6 </xsl:if>
7
8 <!-- ----- -->
9 <!-- XSLT: if Anweisung -->
10 <!-- ----- -->
11 <!-- ----- -->
12 <xsl:stylesheet version="2.0">
13
14   <xsl:template match="...">
15     <xsl:variable name="code"
16       select="#java"/>
17
18   <xsl:variable name="students"
19     select="#students"/>
20
21   <!-- Vergleich von Zeichenketten -->
22   <xsl:if test="#id = $code">
23
24     ...
25
26   </xsl:if>
27
28   <!-- Test auf Knotensetzen -->
29   <xsl:if test="#students">
30
31     ...
32
33   </xsl:if>
34
35 </xsl:template>
36
37 </xsl:stylesheet>

```

## 25.5.4 &lt;xsl:choose&gt; Element ▶

Neben dem <xsl:if> Element definiert die XSLT Spezifikation das <xsl:choose> Element zur Steuerung des Programmflusses.

## » Erklärung: choose Anweisung ▶

- Im <xsl:choose> XML Element können mehrere <xsl:when> Elemente eingebettet werden.
- Evaluiert der im test Attribut einer when Anweisung definierte logische Ausdruck zu true, werden die in der Anweisung eingeschlossenen Elemente ausgeführt.
- Es wird jedoch nur die erste when Anweisung ausgeführt, deren Bedingung zu true evaluier.
- Zusätzlich kann mit dem <xsl:otherwise> Element ein Default Pfad definiert. Trifft keine der angegebenen Bedingungen zu verzweigt der Programmfluss in den Defaultzweig.

## » Syntax: &lt;xsl:choose&gt; ▶

```

1 <!-- ----- -->
2 <!-- Syntax: <xsl:choose> -->
3 <!-- ----- -->
4 <xsl:choose>
5   <xsl:when test = "logischer Ausdruck">
6     <!-- Inhalt: Template -->
7   </xsl:when>
8   <xsl:otherwise>
9     <!-- Inhalt: Template -->
10  </xsl:otherwise>
11 </xsl:choose>
12
13 <!-- ----- -->
14 <!-- XSLT: choose Anweisung -->
15 <!-- ----- -->
16 <xsl:template match="table-row">
17   <xsl:choose>
18     <xsl:when test="position() = 0">
19       <xsl:text>paywhip</xsl:text>
20     </xsl:when>
21     <xsl:when test="position() = 1">
22       <xsl:text>minicream</xsl:text>
23     </xsl:when>
24     <xsl:otherwise>
25       <xsl:text>whitemoke</xsl:text>
26     </xsl:otherwise>
27   </xsl:choose>
28 </xsl:template>
29
30 </xsl:stylesheet>

```

243

## 25.5.5 &lt;xsl:for-each&gt; Element ▶

Zur Verarbeitung von Knotensetzen stellt die XSLT Spezifikation das <xsl:for-each> Element zur Verfügung.

## » Erklärung: Iterative Verarbeitung ▶

- Für die iterative Verarbeitung eines node-sets stellt XSLT das <xsl:for-each> Element zur Verfügung.
- In jedem Iterationsabschnitt der Schleife wird eines der Elemente des node-sets verarbeitet.
- Das <xsl:for-each> Element ist dabei neben dem <xsl:apply-templates> Element die einzige Anweisung, die den Kontext des XSLT Prozessors verändert kann.

## » Codebeispiel: for-each Anweisung ▶

```

1 <!-- ----- -->
2 <!-- XSLT: for-each Anweisung -->
3 <!-- ----- -->
4 <xsl version="1.0" encoding="UTF-8"?>
5 <xsl:stylesheet version="2.0">
6
7   <!-- Datenformat der Ausgabe -->
8   <xsl:output method="html"/>
9
10  <xsl:template match="/">
11    <cinema>
12      <!-- iterative Verarbeitung -->
13      <xsl:for-each
14        select="#movies/movie">
15        <xsl:if test="imdbRating[.
16          &gt; 8]">
17          <screening>
18            <title>
19              <xsl:value-of
20                select="#title"/>
21            </title>
22            <link>
23              <xsl:value-of
24                select="#url"/>
25            </link>
26            <screening>
27              <xsl:for-each
28                <xsl:for-each
29                  <project-staff>
30                    <xsl:for-each select="#/employee">
31                      <xsl:sort
32                        select="#name/last-name"
33                        order="descending"/>
34                      <xsl:sort
35                        select="#name/first-name"
36                        order="descending"/>
37                      <last-name>
38                        <xsl:value-of
39                        select="#name/last-name"/>
40                      </last-name>
41                      <first-name>unknown</first-name>
42                    </xsl:for-each>
43                  </project-staff>
44                </xsl:template>
45
46 </xsl:stylesheet>

```

## 25.5.6 &lt;xsl:sort&gt; Element ▶

Mit dem <xsl:sort> Element können die Knoten einer Knotensetzung sortiert werden.

## » Erklärung: Sortieren von Knotensetzen ▶

- Das <xsl:sort> Element kann dabei als Kind-Element des <xsl:apply-templates> bzw. des <xsl:for-each> Elements auftreten.
- Um eine Knotensetzung nach mehreren Kriterien zu sortieren, wird das <xsl:sort> Element wiederholt eingebettet.

## » Syntax: &lt;xsl:sort&gt; ▶

```

1 <!-- ----- -->
2 <!-- Syntax: <xsl:sort> -->
3 <!-- ----- -->
4 <xsl:sort
5   select = "Knotensetzung"
6   order = "ascending" | "descending"
7 </xsl:sort>
8
9 <!-- ----- -->
10 <!-- XSLT: sort Anweisung -->
11 <!-- ----- -->
12 <xsl version="1.0" encoding="UTF-8?"/>
13 <xsl:stylesheet version="2.0">
14
15   <!-- Datenformat der Ausgabe -->
16   <xsl:output method="html"/>
17
18   <xsl:template match="/">
19     <project-staff>
20       <xsl:for-each select="#/employee">
21         <xsl:sort
22           select="#name/last-name"
23           order="descending"/>
24         <xsl:sort
25           select="#name/first-name"
26           order="descending"/>
27         <last-name>
28           <xsl:value-of
29           select="#name/last-name"/>
30         </last-name>
31         <first-name>unknown</first-name>
32       </xsl:for-each>
33     </project-staff>
34   </xsl:template>
35
36 </xsl:stylesheet>

```

244



## 25.7. Suchanfragen

Das Suchmuster einer Templateregel definiert welche XML Elementknoten ein Template verarbeiten kann.

### 25.7.1 Auflösen von Templatekonflikten

Mit XSL ist es möglich für denselben XML Elementknoten mehrere Templateregeln zu definieren.

#### Codebeispiel: Templatekonflikte

```

1 <!-- ----->
2 <!------->
3 <!------->
4 <xsl:stylesheet version="2.0">
5
6   <xsl:template match="#>">
7     <cinema>
8       <xsl:apply-templates
9         select="movies/movie" />
10    </cinema>
11  </xsl:template>
12
13  <xsl:template match="movie">
14    <screening>
15      <xsl:value-of select="title"/>
16    </screening>
17  </xsl:template>
18
19  <xsl:template match="movies/movie">
20    <premiere>
21      <xsl:value-of select="title"/>
22    </premiere>
23  </xsl:template>
24
25 </xsl:stylesheet>

```

#### Erklärung: Templatekonflikte

- Kommen mehrere Templateregeln für die Verarbeitung eines XML Elements in Frage, treten bestimmte Prioritätsregeln in Kraft.
- Spezifischere Regeln haben Vorfang vor allgemeineren Regeln, z.B.: /movies/movie ist spezifischer als movie.
- Suchanfragen mit Wildcard z.B.: \* oder @\* sind allgemeiner als entsprechende Muster ohne Wildcards.
- Wenn keines der aufgeführten Kriterien zum Tragen kommt, hat die zuletzt stehende Regel Vorfang.

### Default Templates

Die Verarbeitung eines XSL Stylesheets beginnt stets mit dem Wurzelknoten des Eingabedokuments.

#### Default Template

Default Templates sind implizit im XSL Prozessor definierte Templates.

#### Erklärung: Default Templates

- Der XSL Prozessor lädt als erstes immer die Templateregel zur Verarbeitung des Wurzelknotens des Eingabedokuments.
- Es spielt deshalb keine Rolle, in welcher Reihenfolge Templateregeln in einem Stylesheet eingetragen sind, der Prozessor sucht immer zunächst nach einer Templateregel, die explizit für den Wurzelknoten definiert worden ist oder sich dafür verwenden lässt.
- Findet er keine solche Templateregel nutzt er eine entsprechende eingebaute Templateregel.
- Eingebaute Templateregeln werden als Default Templates bezeichnet.

### Roottemplate

Wenn eine Templateregel fehlt, die sich explizit auf den Wurzelknoten bezieht, verwendet der Prozessor automatisch das Roottemplate.

#### Codebeispiel: Roottemplate

```

1 <!-- ----->
2 <!-- Defaulttemplate: Roottemplate -->
3 <!------->
4 <xsl:template match="#>"/>
5   <xsl:apply-templates/>
6 </xsl:template>

```

#### Erklärung: Roottemplate

- Wir können das testen, indem wir ein Stylesheet auf eine Quelldatei anwenden, das keine Templateregeln enthält.
- Der Prozessor gibt dann einfach die Stringwerte aller Elemente des Eingabedokuments aus, wobei Attribute ignoriert werden.

247

Das Roottemplate wird jedoch nicht nur auf den Wurzelknoten, sondern auf jedes beliebige Element angewandt.

#### Analyse: Roottemplate

- Das Roottemplate sorgt damit für eine rekursive Verarbeitung, also dafür, dass, wenn eine Templateregel für bestimmte XML Elemente existiert, diese auch ausgeführt wird, auch wenn ihre Anwendung nicht explizit aufgerufen wird.

#### Codebeispiel: Roottemplate

```

1 <!-- ----->
2 <!-- Eingabedokument: movies.xml -->
3 <!------->
4 <?xml version="1.0" encoding="UTF-8"?>
5 <movies genre="crime">
6
7   <movie url="http://www.imdb.com/?movie=...>
8     <title>Die Verurteilten</title>
9     <releaseYear>1994</releaseYear>
10    <imdbRating>9.2</imdbRating>
11
12    <characters>
13      <name>Tim Robbins</name>
14      <name>Morgan Freeman</name>
15    </characters>
16  </movie>
17
18  <movie url="http://www.imdb.com/?movie=>
19    <title>Der Pate</title>
20    <releaseYear>1972</releaseYear>
21    <imdbRating>9.2</imdbRating>
22
23    <characters>
24      <name>Marlon Brando</name>
25      <name>Al Pacino</name>
26    </characters>
27  </movie>
28
29  <movie url="http://www.imdb.com/?movie=>
30    <title>Twilight Imperium</title>
31    <releaseYear>1979</releaseYear>
32    <imdbRating>9.0</imdbRating>
33
34    <characters>
35      <name>Al Pacino</name>
36      <name>Robert De Niro</name>
37    </characters>
38  </movie>
39
40 </movies>

```

#### Codebeispiel: Roottemplate

```

1 <!-- ----->
2 <!-- Stylesheet: movie1.xsl -->
3 <!------->
4 <?xml version="1.0" encoding="UTF-8"?>
5 <xsl:stylesheet version="2.0">
6   <xsl:output method="xml" indent="yes"/>
7 </xsl:stylesheet>

```

#### Ausgabe:

```

<!-- ----->
<!------->
<?xml version="1.0" encoding="UTF-8"?>
<movie>
  <title>Die Verurteilten</title>
  1994
  9.2
  <character>
    <name>Tim Robbins</name>
    <name>Morgan Freeman</name>
  </character>
</movie>

```

#### Codebeispiel: Roottemplate

```

1 <!-- ----->
2 <!-- Stylesheet: movie2.xsl -->
3 <!------->
4 <?xml version="1.0" encoding="UTF-8"?>
5 <xsl:stylesheet version="2.0">
6   <xsl:template match="movie">
7     <movie/>
8   </xsl:template>
9 </xsl:stylesheet>

```

#### Ausgabe:

```

<!-- ----->
<!------->
<?xml version="1.0" encoding="UTF-8"?>
<movie>
  <movie/>
  <movie/>
  <movie/>
</movie>

```

**Codebeispiel: Roottemplate**

```

1 <!-- ----- -->
2 <!-- Stylesheet: movie1.xsl -->
3 <!------- -->
4 <?xml version="1.0" encoding="UTF-8"?>
5 <?xml:stylesheet version="2.0"?>
6
7   <xsl:template match="title">
8     <title>
9   </xsl:template>
10
11 </xml:stylesheet>
12
13 <!------- -->
14 <!-- Ausgabe: movie1.xml -->
15 <!------- -->
16 <?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="movie1.xsl"?>
<title>
  1994
  9.2
    Tim Robbins
    Morgan Freeman
<title/>
  1972
  9.2
    Marlon Brando
    Al Pacino
<title/>
  1985
  5.0
    Al Pacino
    Robert De Niro

```

**Erklärung: Stringtemplate**

- Das Stringtemplate kopiert den Text aller Text- und Attributknoten in das Ausgabedokument.
- Das Stringtemplate wird nur angewandt, wenn der Knoten ausgewählt worden ist.
- Alle Default Templates haben eine niedrigere Priorität als ein im Stylesheet definiertes Template.

**Codebeispiel: Roottemplate**

```

1 <!-- ----- -->
2 <!-- Stylesheet: movie1.xsl -->
3 <!------- -->
4 <?xml version="1.0" encoding="UTF-8"?>
5 <?xml:stylesheet version="2.0"?>
6
7   <xsl:output method="xml" indent="yes"/>
8
9   <xsl:template match="movie">
10     <xsl:apply-templates select="title"/>
11   </xsl:template>
12
13   <xsl:template match="movie">
14     <xsl:apply-templates select="title"/>
15   </xsl:template>
16
17   <xsl:template match="movie">
18     <xsl:apply-templates select="title"/>
19   </xsl:template>
20
21 </xml:stylesheet>
22
23 <!------- -->
24 <!-- Ausgabe: movie1.xml -->
25 <!------- -->
26 <?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="movie1.xsl"?>
<Die Verurteilten>
  Der Pate
  Twilight Imperium

```

**Codebeispiel: Stringtemplate**

```

1 <!-- ----- -->
2 <!-- Defaulttemplate: Stringtemplate -->
3 <!------- -->
4 <xsl:template match="text()|@*"/>
5   <xsl:value-of select="./*"/>
6 </xsl:template>

```

349

## 26. Datenformat - XML Schema

02

XML - Schema

01. Schemagrundlagen	250
02. Struktur XML Schema	251
03. Einbliche Datentypen	253
04. Komplexe Datentypen	255
05. Attribute definieren	258
06. Sichtbarkeitskonzepte	259

**26.1. Schema Grundlagen**

**26.1.1 Inhaltsmodell**

So wie sich ein Dokument in Einleitung, Hauptteil und Nachwort gliedern lässt, so kann für die Struktur eines XML-Dokuments ein Inhaltsmodell definiert werden.

Das Inhaltsmodell eines XML-Dokuments beschreibt welche Elemente in einem XML-Dokument auftreten dürfen.

**Erklärung: Inhaltsmodell**

- Damit eine Anwendung die Gültigkeit eines Dokuments prüfen kann, muß die Beschreibung des Inhaltsmodells selbst in einer maschinenlesbaren Sprache definiert sein.
- Die Beschreibung eines Inhaltsmodells in maschinenlesbarer Form wird als Schema bezeichnet.
- XSD-Schemas stellen eine Schemadefinition für XML-Dokumente dar.

**Inhaltsmodell**

Ein Inhaltsmodell beschreibt die logische Struktur eines XML-Dokuments.

**XML Schema**

XML-Schemas beschreiben Inhaltsmodelle für XML-Dokumente in maschinenlesbarer Form.

350

**26.1.2 Validierung**

Zur Prüfung der Gültigkeit eines XML-Dokuments, validiert der XML-Processor das Dokument gegen das dem Inhaltsmodell des Dokuments zugeordneten Schema.

**Validierung**

Die Prüfung eines XML-Dokuments auf Gültigkeit wird als Validierung bezeichnet.

**Erklärung: Validierung**

**26.2. Struktur: XML Schemas**

Ein XML-Schema selbst ist ein XML-Dokument.

**26.2.1 Fallbeispiel: XML Schema**

Das folgende Beispiel zeigt ein XML-Schema für ein einfaches XML-Dokument.

**Codebeispiel: XML Schema**

```

1 <!-- ----- -->
2 <!-- Schemadefinition -->
3 <!------- -->
4 <?xml version="1.0" encoding="UTF-8"?>
5 <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
6   <xsd:element name="person" type="xsd:string"/>
7 </xsd:schema>
8
9 <!-- ----- -->
10 <!-- XML-Dokument -->
11 <!------- -->
12 <?xml version="1.0" encoding="UTF-8"?>
13 <person>Max Mustermann</person>

```

**Validierung**

Die Prüfung eines XML Dokuments auf Güte wird die Validierung bezeichnet.

**Erklärung: Validierung**

- XML Schemas definieren Regeln, die die Struktur von XML Dokumenten beschreiben.
- Die im XML Schemas definierten Regeln müssen bei der Erstellung eines XML Dokuments eingehalten werden.
- Bevor ein XML Dokument validiert werden kann, wird es auf Wohlgeformtheit geprüft. Ein wohlgeformtes XML Dokument kann dann gegen die im XML Schema definierten Regeln validiert werden.
- Erfüllt ein XML Dokument die Regeln des XML Schemas, wird es als Instanz des Schemas bezeichnet.

**26.1.3 XML Schema Regeln**

XML Schemas definieren Regeln, die die Struktur von XML Dokumenten beschreiben.

**Analyse: Regelausprägung**

- XML Schema definieren welche Elemente in einem XML Dokument enthalten sein dürfen.
- XML Schema definieren welche Attribute ein XML Element haben kann.
- XML Schema definieren welche Kindelemente ein XML Element enthalten kann.
- Mit einem XML Schema kann die Reihenfolge in welcher die Kindelemente eines Elements auftreten, festgelegt werden.

**26.2.2 Aufbau eines XML Schemas**

Das `<x:element>` Element ist das grundlegende Element der XML Schema Spezifikation. Mit dem Element wird einem XML Element ein Datentyp zugeordnet.

```

<!-- ----->
<!-- XML Schema: Person ----->
<?xml version="1.0" encoding="UTF-8"?>
<x:schema xmlns:x="http://www.w3.org/2001/XMLSchema">
  <x:element name="person">
    <x:complexType>
      <x:sequence>
        <x:element name="name">
          <x:type>xs:string</x:type>
        </x:element>
      </x:sequence>
    </x:complexType>
  </x:element>
</x:schema>
<!-- ----->
<!-- XML Dokument: person.xml ----->
<?xml version="1.0" encoding="UTF-8"?>
<person>
  <name>Kauer Alexander</name>
</person>

```

Informationssysteme		
Datentyp	Beschreibung	Beispiel
string	Zeilenkette	Helmut Meier
token	Zeilenkette	Helmut Meier
language	ISO639-1 Kürzel einer Sprache	de, fr, en
integer	Ganze Zahl	-2143, -3, 0, 354
positiveInteger	positive ganze Zahl	432
negativeInteger	negative ganze Zahl	-323
nonNegativeInteger	positive ganze Zahl	323
nonPositiveInteger	negative ganze Zahl	-323
byte	8 bit Integer mit Vorzeichen	-1, 124
short	16 bit Integer mit Vorzeichen	23232
int	32 bit Integer mit Vorzeichen	3234
long	64 bit Integer mit Vorzeichen	2323
decimal	gebrochene Dezimalzahl	-1.23, 123.4, 100.0
float	Flektionszahl	-INF, -1E4, -12.3
double	Flektionszahl	-INF, -1E4, -12.3
time	Uhrzeit, ggf. mit Zeitzone	11:29:00.000
date	Datum (ISO8601 Format)	24.02.15
gMonth	Monat (ISO8601 Format)	-05-
gYear	Jahr (ISO8601 Format)	1978
gDay	Tag (ISO8601 Format)	-31
gMonthDay	Zeitangabe (ISO8601 Format)	-05-31
gYearMonth	Zeitangabe (ISO8601 Format)	1999-02
dateTime	Zeitstempel	1004-02-15T09:00:00
boolean	logischer Wert	true, false, 0, 1
anyURI	URI	http://www.hfkrems.at/welcome

Abbildung 92: Native Datentypen

### 26.2.3 Kategorien von Datentypen

Die XML Schema Spezifikation unterscheidet 2 Arten von Datentypen:

#### ➤ Auflistung: Arten von Datentypen

- Einfache Datentyp ▾  
XML Elemente mit unstrukturiertem Inhalt haben einen einfachen Datentyp.
- Komplexer Datentyp ▾  
XML Elemente mit einem strukturierten Inhalt haben einen komplexen Datentyp.

#### ➤ Codebeispiel: Datentypen

```

1 <!-- ----- -->
2 <!-- Element mit strukturiertem Inhalt -->
3 <!-- ----- -->
4 <?xml version="1.0" encoding="UTF-8"?>
5 <person>
6   <name>Max Mustermann</name>
7 </person>
8
9 <?xml version="1.0" encoding="UTF-8"?>
10 <xm:schema
11   xmlns:xm="http://www.w3.org/2001/XMLSchema">
12   <xm:element name="person">
13     <xm:complexType>
14       <xm:sequence>
15         <xm:element name="name">
16           <xm:type>x:string</xm:type>
17         </xm:element>
18       </xm:sequence>
19     </xm:complexType>
20   </xm:element>
21 </xm:schema>
22
23 <!-- ----- -->
24 <!-- Element mit unstrukturiertem Inhalt -->
25 <!-- ----- -->
26 <?xml version="1.0" encoding="UTF-8"?>
27 <person>Max Mustermann</person>
28
29 <xm:schema xmlns:xm="http://www.w3.org/2001/XMLSchema">
30   <xm:element name="person">
31     <xm:type>x:string</xm:type>
32   </xm:element>
33 </xm:schema>
```

### 26.3. Einfache Datentypen

Einfache Datentypen der XML Schema Spezifikation sind vergleichbar mit den einfachen Datentypen von Programmiersprachen wie Java.

#### 26.3.1 Einfache XML Elemente

Die XML Schema Spezifikation definiert 2 Arten von einfache Datentypen:

#### ➤ Auflistung: Arten einfacher Datentypen

##### Native Datentypen ▾

Die XML Schema Spezifikation definiert eine Reihe **einfacher Datentypen**.

##### Benutzerdefinierte Datentypen ▾

Durch das Ableiten einfacher Datentypen können eigene benutzerdefinierte Datentypen definiert werden.

#### ➤ Codebeispiel: Einfache Datentypen

```

1 <!-- ----- -->
2 <!-- Element mit strukturiertem Inhalt -->
3 <!-- ----- -->
4 <?xml version="1.0" encoding="UTF-8"?>
5 <xm:schema xmlns:xm="http://www.w3.org/2001/XMLSchema">
6   <xm:complexType name="personType">
7     <xm:sequence>
8       <!-- Native Datentypen -->
9       <xm:element name="name">
10         <xm:type>x:string</xm:type>
11       </xm:element>
12       <xm:element name="age" type="x:int"/>
13     </xm:sequence>
14   </xm:complexType>
15
16 <xm:simpleType name="nameType">
17   <xm:restriction base="x:string">
18     <xm:length value="10"/>
19   </xm:restriction>
20 </xm:simpleType>
21 </xm:schema>
```

*eingeschränkter Datentyp*

263

### Informationssysteme

### 26.3.2 Benutzerdefinierte Datentypen

Die XML Spezifikation definiert 3 Möglichkeiten eines Datentypen abzuleiten.

#### ➤ Auflistung: Ableiten einfacher Datentypen

##### Derivation by Restriction ▾

Neue Datentypen werden definiert, indem der Wertebereich eines gegebenen Datentyps **eingeschränkt** wird.

##### Derivation by Union ▾

Neue Datentypen werden definiert indem die Wertebereiche bestehender Datentypen **kombiniert** werden.

##### Derivation by List ▾

Neben atomaren Datentypen können ebenfalls Listen von einfachen Typen, als neue Datentypen definiert werden.

### 26.3.3 Derivation by Restriction

##### Derivation by Restriction ▾

Neue Datentypen werden definiert, indem der Wertebereich eines gegebenen Datentyps **eingeschränkt** wird.

Zum Einschränken des Wertebereichs eines Datentyps definiert die XML Schema Spezifikation Facets.

#### ➤ Erklärung: Facets

- Facets definieren Regeln zum Einschränken der Wertebereiche einfacher Datentypen.
- Facets werden eingebettet in <x:restriction> Element definiert. Dabei können mehrere Facets in einem restriction Element eingebettet werden.
- Facets beziehen sich dabei immer auf einen bestimmten Datentyp.
- Die XML Schema Spezifikation erlaubt die Definition eigener Facets.

#### ➤ Codebeispiel: Derivation by Restriction

```

1 <!-- ----- -->
2 <!-- Derivation by Restriction -->
3 <!-- ----- -->
4 <xm:simpleType name="gFormat">
5   <xm:restriction base="x:string">
6     <xm:enumeration value="jpg"/>
7     <xm:enumeration value="png"/>
8     <xm:enumeration value="gif"/>
9   </xm:restriction>
10 </xm:simpleType>
11
12 <xm:element name="extension" type="gFormat"/>
13
14 <xm:restriction>
15   <xm:enumeration value="gif"/>
16 </xm:restriction>
17 </xm:simpleType>
18
19 <xm:element name="matrikelnr" type="aCode"/>
20
21 <xm:simpleType name="aCode">
22   <xm:restriction base="x:string">
23     <xm:length value="10"/>
24   </xm:restriction>
25 </xm:simpleType>
26
27 <xm:element name="matrikelnr09756234</matrikelnr>
28
29
30 <xm:simpleType name="userName">
31   <xm:restriction base="x:string">
32     <xm:minLength value="5"/>
33     <xm:maxLength value="20"/>
34   </xm:restriction>
35 </xm:simpleType>
36
37 <xm:element name="user-name" type="userName"/>
38
39 <xm:restriction>
40   <xm:simpleType name="httpURI">
41     <xm:restriction base="x:string">
42       <xm:pattern value="http://.*/$"/>
43     </xm:restriction>
44   </xm:simpleType>
45
46 <xm:element name="uri" type="httpURI"/>
47
48 <xm:restriction>
49   <xm:simpleType name="httpURI">
50     <xm:restriction base="x:string">
51       <xm:pattern value="http://.*/$"/>
52     </xm:restriction>
53   </xm:simpleType>
54
55 <xm:element name="curl>http://htlkrem.ac.at</curl>
```

264

Facet	Beschreibung	Typ
<x:enumeration>	Das Facet erlaubt die Definition einer Liste von möglichen Werten, die das String, Integer Element annehmen kann. Vergleichen Sie das Facet mit einem Enum.	
<x:length>	Das Facet legt die Länge eines String fest.	String
<x:minLength>	Das Facet legt die minimale Länge eines Strings fest.	String
<x:maxLength>	Das Facet legt die maximale Länge eines Strings fest.	String
<x:pattern>	Das Facet definiert ein Muster die ein String haben muss.	String
<x:minInclusive>	Das Facet definiert den minimalen Wert den ein Integer annehmen kann.	Integer
<x:minExclusive>	Das Facet definiert den minimalen Wert den ein Integer annehmen kann.	Integer
<x:maxInclusive>	Das Facet definiert den maximalen Wert den ein Integer annehmen kann.	Integer
<x:maxExclusive>	Das Facet definiert den maximalen Wert den ein Integer annehmen kann.	Integer

Abbildung 93. Facets

26.3.4 Derivation by Union	
<input checked="" type="checkbox"/> Derivation by Union	Neue Datentypen können definiert werden indem die Wertebereiche bestehender Datentyp <b>komplettiert</b> werden.
<b>Vereinigen</b>	

26.4. Komplexe Datentypen	
<input checked="" type="checkbox"/> Codebeispiel: Datentyp für Enums	Komplexe Datentypen beschreiben XML Elemente die andere XML Elemente beinhalten können.
<pre> 1 &lt;!-- ----- --&gt; 2 &lt;!-- Fallbeispiel: Enum --&gt; 3 &lt;!-- ----- --&gt; 4 &lt;x:simpleType name="colorType"&gt; 5   &lt;x:union&gt; 6     &lt;x:simpleType&gt; 7       &lt;x:restriction base="xs:int"&gt; 8         &lt;x:enumeration value="0"/&gt; 9         &lt;x:enumeration value="1"/&gt; 10        &lt;/x:restriction&gt; 11      &lt;/x:simpleType&gt; 12    &lt;x:simpleType&gt; 13      &lt;x:restriction base="xs:string"&gt; 14        &lt;x:enumeration value="white"/&gt; 15        &lt;x:enumeration value="yellow"/&gt; 16      &lt;/x:restriction&gt; 17    &lt;/x:simpleType&gt; 18  &lt;/x:union&gt; 19 &lt;/x:simpleType&gt; </pre>	

265

26.4.2 Strukturierung von XML Elementen	
<input checked="" type="checkbox"/> Sequenz	Die XML Schema Spezifikation definiert mehrere Möglichkeiten zur Beschreibung des Inhaltsmodells komplexer Datentypen.
<input checked="" type="checkbox"/> Auswahl	Zur Beschreibung des Inhaltsmodells komplexer Datentypen stellt die XML Schema Spezifikation 3 Möglichkeiten zur Verfügung.
<input checked="" type="checkbox"/> Aggregator	Die Elemente eines Aggregates definieren einen Pool von Elementen der im Inhaltsmodell des Elements auftreten müssen.

> Codebeispiel: Auswahl	
<input checked="" type="checkbox"/> Sequenz	<!-- ----- -->
<input checked="" type="checkbox"/> Auswahl	<!-- Struktur: Auswahl -->
<input checked="" type="checkbox"/> Aggregator	<!-- ----- -->
	<pre> 1 &lt;!-- ----- --&gt; 2 &lt;!-- Struktur: Auswahl --&gt; 3 &lt;!-- ----- --&gt; 4 &lt;x:complexType name="graphType"&gt; 5   &lt;x:sequence&gt; 6     &lt;x:element name="node" type="nodeType"/&gt; 7   &lt;/x:sequence&gt; 8 &lt;/x:complexType&gt; 9 10 &lt;x:complexType name="nodeType"&gt; 11   &lt;x:choice&gt; 12     &lt;x:element name="label" type="xs:string"/&gt; 13   &lt;x:sequence&gt; 14     &lt;x:element name="node" type="nodeType"/&gt; 15     &lt;x:element name="edge" type="xs:string"/&gt; 16     &lt;x:element name="node" type="nodeType"/&gt; 17   &lt;/x:sequence&gt; 18 &lt;/x:choice&gt; 19 &lt;/x:complexType&gt; 20 21 &lt;x:element name="graph" type="graphType"/&gt; 22 23 &lt;!-- ----- --&gt; 24 &lt;!-- graph.xml --&gt; 25 &lt;!-- ----- --&gt; 26 &lt;?xml version="1.0" encoding="UTF-8"?&gt; 27 &lt;graph&gt; 28   &lt;node&gt; 29     &lt;node&gt; 30       &lt;label&gt;A1&lt;/label&gt; 31     &lt;/node&gt; 32     &lt;edge&gt;a1&lt;/edge&gt; 33     &lt;node&gt; 34       &lt;label&gt;A6&lt;/label&gt; 35     &lt;/node&gt; 36   &lt;/node&gt; 37   &lt;node&gt; 38     &lt;label&gt;A2&lt;/label&gt; 39   &lt;/node&gt; 40   &lt;edge&gt;a2&lt;/edge&gt; 41   &lt;node&gt; 42     &lt;label&gt;A3&lt;/label&gt; 43   &lt;/node&gt; 44 &lt;/graph&gt; </pre>

entweder ... oder ...

266

## » Codebeispiel: Aggregator □

```

1 <!-- ----- -->
2 <!-- Struktur: Aggregator -->
3 <!-- ----- -->
4 <x:complexType name="addressType">
5   <x:all>
6     <x:element type="xs:string" name="code"/>
7     <x:element type="xs:string" name="count"/>
8     <x:element type="xs:string" name="street"/>
9     <x:element type="xs:string" name="loc"/>
10   </x:all>
11 </x:complexType>
12
13 <x:element name="address" type="aType"/>
14
15 <!-- ----- -->
16 <!-- address.xsd -->
17 <!-- ----- -->
18 <?xml version="1.0" encoding="UTF-8"?>
19 <address>
20   <street>Alauntalstrasse 11</street>
21   <location>Krems a. Donau</location>
22   <code>3470</code>
23   <country>Austria</country>
24 </address>
25
26 <?xml version="1.0" encoding="UTF-8"?>
27 <address>
28   <country>Austria</country>
29   <code>3470</code>
30   <location>Krems a. Donau</location>
31   <street>Alauntalstrasse 11</street>
32 </address>
33
34 <?xml version="1.0" encoding="UTF-8"?>
35 <address>
36   <code>3470</code>
37   <location>Krems a. Donau</location>
38   <street>Alauntalstrasse 11</street>
39   <country>Austria</country>
40 </address>
41
42 <?xml version="1.0" encoding="UTF-8"?>
43 <address>
44   <code>3470</code>
45   <location>Krems a. Donau</location>
46   <street>Alauntalstrasse 11</street>
47   <country>Austria</country>
48 </address>

```

Reihenfolge  
egal

## » 26.4.3 Häufigkeitsindikatoren □

Häufigkeitsindikatoren legen fest wie oft ein Element im Inhaltsmodell eines XML Elements auftreten darf.

Erklärung: Häufigkeitsindikatoren

- minOccurs: Legt fest wie oft ein Element im Inhaltsmodell eines XML Elements zum mindestens vorkommen muss.
- maxOccurs: Bestimmt wie oft ein Element maximal vorkommen darf.

```

1 <!-- ----- -->
2 <!-- Struktur: Aggregator -->
3 <!-- ----- -->
4 <x:complexType name="passengerType">
5   <x:sequence>
6     <x:element name="passenger"
5       type="xs:string"
7         minOccurs="1" maxOccurs="5"/>
8   </x:sequence>
9 </x:complexType>
10
11 <x:element name="passengers"
12   type="passengerType"/>
13
14 <!-- ----- -->
15 <!-- passengers.xml -->
16 <!-- ----- -->
17 <?xml version="1.0" encoding="UTF-8"?>
18 <passenger>
19   <passenger>Tao Tse Long</passenger>
20 </passenger>
21 <?xml version="1.0" encoding="UTF-8"?>
22 <passengers>
23   <passenger>Guanyu</passenger>
24   <passenger>Konfuzius</passenger>
25 </passenger>
26
27 <?xml version="1.0" encoding="UTF-8"?>
28 <passengers>
29   <passenger>Guanyu</passenger>
30   <passenger>Konfuzius</passenger>
31   <passenger>Konfuzius</passenger>
32   <passenger>Konfuzius</passenger>
33 </passenger>
34

```

267

## » 26.4 gemischter Inhalt □

XML Elemente deren Inhaltsmodelle, sowohl andere Elemente als auch Text beinhalten können, haben einen gemischten Inhalt.

```

1 <!-- ----- -->
2 <!-- Struktur: mixed -->
3 <!-- ----- -->
4 <x:complexType name="reportType"
5   mixed="true">
6   <x:sequence>
7     <x:element name="index"
6       type="xs:string"
7         maxOccurs="unbounded"/>
8   </x:sequence>
9 </x:complexType>
10
11 <x:element name="report" type="reportType"/>
12
13 <?xml version="1.0" encoding="UTF-8"?>
14 <text>In einer Höhle in der Erde, da lebte
15 ein <index>Hobbit</index>. Nicht in einem
16 schmutzigen, nassten <index>Loch</index>,
17 in das die Enden von irgendwelchen Wurzeln
18 herabbaumelten und das nach Schlamm und Moder
19 roch.
20 </text>

```

## » 26.5. Attribute □

Die Attribute eines XML Elements werden im Inhaltsmodell getrennt von eingebetteten Elementen definiert.

Bevor Attribute für ein XML Element definiert werden können muss der Inhaltstyp des Elements bestimmt werden. Je nach Inhaltstyp werden Attribute unterschiedlich definiert.

```

1 <!-- ----- -->
2 <!-- Fallbeispiel: Personendatenanzt -->
3 <!-- ----- -->
4 <x:complexType name="bookType">
5   <x:sequence>
6     <x:element name="title" type="tType"/>
7   </x:sequence>
8 </x:complexType>
9
10 <x:complexType name="tType">
11   <x:simpleContent>
12     <x:extension base="xs:string">
13       <x:attribute name="lang"
14         type="xs:string"/>
15       <x:attribute name="version"
16         type="xs:string"/>
17       <x:attribute name="d1"
18         type="xs:string"/>
19     </x:extension>
20   </x:simpleContent>
21 </x:complexType>
22
23 <x:element name="book" type="bookType"/>
24
25 <?xml version="1.0" encoding="UTF-8"?>
26 <book>
27   <title lang="en" version="1.0">
28     Moby Dick
29   </title>
30   <description>
31     A book about a whale ...
32   </description>
33 </book>

```

268

### 26.5.2 Attributedefinitionen - Elemente mit komplexem Inhalt

Für Elemente mit komplexem Inhalt gestaltet sich die Attributedefinition wesentlich einfacher.

#### \* Codebeispiel: Attributedefinition \*

```

1 <!-- ----->
2 <x:attribute name="Attributedefinition" type="xs:string"/>
3 <!-- ----->
4 <x:complexType name="lType">
5   <x:all>
6     <x:element name="address" type="aType"/>
7   </x:all>
8   <x:attribute name="id" type="xs:string"/>
9   <x:attribute name="code" type="xs:string"/>
10 </x:complexType>
11
12 <x:complexType name="aType">
13   <x:sequence>
14     <x:element name="country"
15       type="xs:string"/>
16     <x:element name="loc" type="xs:string"/>
17     <x:element name="street"
18       type="xs:string"/>
19   </x:sequence>
20 </x:complexType>
21
22 <x:complexType name="bType">
23   <x:sequence>
24     <x:element name="*" type="xs:string"/>
25     <x:element name="*" type="xs:string"/>
26     <x:element name="*" type="xs:string"/>
27     <x:element name="fig" type="xs:int"/>
28   </x:sequence>
29 </x:complexType>
30
31 <!-- ----->
32 <!-- Fallbeispiel: Attributedefinition -->
33 <!-- ----->
34 <x:element name="library" type="lType"/>
35
36 <?xml version="1.0" encoding="UTF-8"?>
37 <library id="23" code="hl1-323-2dfhfrt">
38   <address>
39     <country>Austria</country>
40     <loc>3540 Korne</loc>
41     <street>Almtalstrasse 29</street>
42   </address>
43 </library>

```

### 26.6. Sichtbarkeitskonzepte

Die XML Schema Spezifikation unterscheidet lokale und globale Datentypdeklarationen.

#### \* Codebeispiel: globale Datentypdeklarationen \*

```

1 <!-- ----->
2 <!-- Fallbeispiel: Globale Datentypen -->
3 <!-- ----->
4 <x:complexType name="nameType">
5   <x:sequence>
6     <x:element name="surname"
7       type="xs:string"/>
8     <x:element name="givenname"
9       type="xs:string"/>
10    </x:sequence>
11 </x:complexType>
12
13 <x:element name="user-name" type="nameType"/>
14 <x:element name="pilot" type="nameType"/>
15 <x:element name="programmer"
16   type="nameType"/>
17 <x:element name="student" type="nameType"/>
18
19 <!-- ----->
20 <!-- Fallbeispiel: Attributedefinition -->
21 <!-- ----->
22 <x:element name="library" type="lType"/>
23
24 <?xml version="1.0" encoding="UTF-8"?>
25 <library id="23" code="hl1-323-2dfhfrt">
26   <address>
27     <country>Austria</country>
28     <loc>3540 Korne</loc>
29     <street>Almtalstrasse 29</street>
30   </address>
31 </library>

```

269

### 27. Datenformat - JSON

# 01

JSON

01. JSON Grundlagen

261

02. JSON Datentypen

262

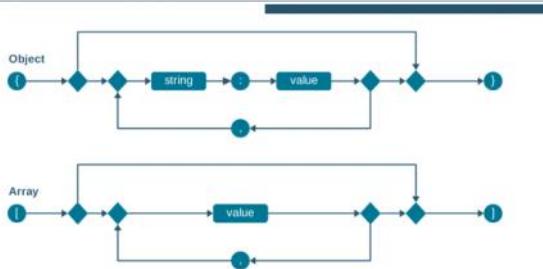


Abbildung 94. JSON: Grundstrukturen

## 27.1. JSON Datenformat

**Json Datenformat** ▾  
JavaScript Object Notation, kurz JSON ist ein Datenformat für die Verarbeitung bzw. den Austausch von Daten in Informationssystemen und Datenbankanwendungen.

### 27.1.1 JSON Grundlagen

JSON wird zur Übertragung und zum Speichern von strukturierten Daten eingesetzt. Es dient dabei in erster Linie für die Serialisierung von Daten.

#### Erklärung: JSON ▾

- JSON ist ein textbasiertes, von Menschen lesbares Datenformat, das für die Darstellung einfacher Datenstrukturen und Objekte verwendet wird.
- JSON kommt in Desktop- bzw. serverseitigen Programmierumgebungen zum Einsatz.
- JSON stammt ursprünglich aus dem JavaScript Umfeld. Jedes gültige JSON Dokument ist gültiger

JavaScript Code. Ein JSON Dokument beschreibt ein gültiges<sup>7</sup> JavaScript Objekt.

- JSON kann von den meisten Programmiersprachen entweder direkt bzw. mit der Hilfe von Programmiersbibliotheken verarbeitet werden.
- JSON wurde in erster Linie als Daten austauschformat konzipiert. Dabei zählt JSON zu den unstrukturierten Datenformaten.
- JSON ist neben XML das wichtigste Datenformat zur Übertragung von Daten. Wie XML ist es sprachunabhängig, gleichzeitig ist es in der Verwendung jedoch einfacher und effizienter als XML.

261

## 27.2. JSON Datentypen

JSON unterscheidet eine Reihe von Datentypen.



### 27.2.1 JSON Datentypen

#### Auflistung: JSON Datentypen ▾

- Null: Mit dem Schlüsselwert null kann einer Variable der Nullwert zugewiesen werden. Der Nullwert zeigt an dass sich kein Wert\* in der Variable befindet.
- Boolean: Der Datentyp Boolean wird zur Verarbeitung von Wahrheitswerten verwendet. Mittels der Schlüsselwörter true und false kann der Wahrheitswert für eine Variable bestimmt werden.
- Zahl: Für das Arbeiten mit Zahlenwerten stellt JSON den Datentyp Number zur Verfügung.
- Zeichenkette: Für das Arbeiten mit Zeichenketten stellt JSON den Datentyp String zur Verfügung. Die Zeichenkette wird dabei zwischen 2 Anführungszeichen notiert.
- Array: Für die Verarbeitung von Listen von Werten stellt JSON den Datentyp array zur Verfügung. Um ein Array zu deklarieren werden beliebig viele Werte durch Kommas getrennt in eckigen Klammern [] notiert. Für jeden Eintrag im Array kann man einen beliebigen für JSON zugelassenen Typ verwenden.
- Objekt: Für die Verarbeitung von komplexen, strukturierten Daten stellt JSON den Datentyp Object zur Verfügung. Objekte deklarieren eine beliebige Zahl von Variablen eingeschlossen in geschweiften Klammern. Dabei organisiert das Objekt die Variablen als Schlüssel/Werte Paare. Der Schlüssel fungiert als Name einer Variable, der Wert entspricht dem Wert der Variable.

Q

\* Vergwechselt Sie den Nullwert mit mit dem Zahlenwert 0 bzw. einem leeren String.

262

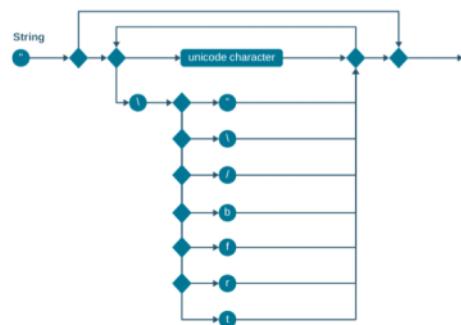


Abbildung 95. JSON: Zeichenketten

## 27.2.2 JSON Grundstrukturen

JSON unterscheidet 2 Grundstrukturen die es dem Datenformat erlauben die Strukturen und Variablen anderer Programmiersprachen abzubilden.

JSON dient in Informationssystemen als austauschbares Datentyp.

## Erklärung: JSON Object ▾

- Als Format zur Übertragung von Daten besteht für JSON die Notwendigkeit die Datenstrukturen anderer Programmiersprachen abilden zu können.
- Der Object Datentyp wird verwendet um Objekte, Records, Structs, Dictionaries und Hashtables abzubilden.
- Hierbei handelt es sich um die universellen Datenstrukturen, die in Programmiersprachen unterstützt werden.

## Codebeispiel: JSON Objekt ▾

```

1 //-----
2 // JSON Objekt
3 //-----
4 var person = {
5   "string": <value>,
6   "string": <value>,
7   "string": <value>,
8   "string": <value>,
9   "string": <value>,
...
10  <string> : [
11    <string> : <value>,
12    <string> : <value>,
13    <string> : <value>
14  ],
15  <string> : <value>
16  ...
17 }
18 }
```

263

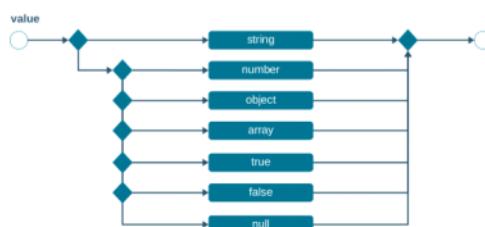


Abbildung 96. JSON: Datentypen

## Erklärung: JSON Array ▾

- Der Array Datentyp speichert eine Liste von geordneten Werten.
- Der Array Datentyp wird verwendet um Arrays, Vektoren, Listen bzw. Sequenzen abzubilden.

## Codebeispiel: JSON Array ▾

```

1 //-----
2 // JSON Array
3 //-----
4 var person = {
5   ...
6   ...
7   ...
8   ...
9   ...
10  <string> : [
11    <string> : <value>,
12    <string> : <value>,
13    <string> : <value>,
14    <string> : <value>,
15    <string> : <value>
16  ],
17  ...
18 }
19 }
```

## Einfache Datentypen

## Codebeispiel: String und Werte ▾

```

1 //-----
2 // Auflösung von Strings und Werten
3 //-----
4 var person = {
5   "firstName": "John",
6   "lastName": "Smith",
7   "isAlive": true,
8   "age": 45,
9   "height_cm": 170,
10  "address": {
11    "street": "21 2nd Street",
12    "city": "New York",
13    "postalCode": "10021-2100"
14  },
15  "children": [
16    {
17      "firstName": "John",
18      "lastName": "Wick",
19      "isAlive": true,
20      "age": 15
21    }
22  ],
23  "spouse": null
24 }
```

□

□



Glossar		Bereichssuche
<b>A</b>		
<b>Abfrage</b>	<b>D</b>	
<b>Abfrageobjekt</b>	<b>Datenaggregation</b>	
Ein Abfrageobjekt wird zum Filtern von Daten einer Abfrage verwendet.	Datenaggregation beschreibt den Prozess des Zusammensetzen von Daten aus unterschiedlichen Datenquellen.	
<b>Abfragesprache</b>	<b>Datenbankengine</b>	
	Die Datenbankengine ist eine zugrundeliegende Softwarekomponente, die ein Datenbankverwaltungssystem verwendet, um Daten einer Datenbank zu erstellen, zu löschen bzw. löschen. Datenbankverwaltungssysteme besitzen eine eigene Programmierschnittstelle (API), die es dem Benutzer erlaubt die darunterliegende Engine zu verwenden.	
<b>Aggregatfunktion</b>	<b>Datenbankobjekt</b>	
Funktionen, die eine Menge von Werten zu einem einzelnen Wert verdichten, werden als Aggregatfunktionen bezeichnet.	Datenbankobjekte stellen das Gerüst zur Speicherung und Abfrage von Daten einer Datenbank dar. Beispiele von Datenbankobjekten sind: Tabelle, Datenbankindex, View, Alias, Stored Procedure, Trigger, Constraint usw.	
<b>Alphanumerische Werte</b>	<b>Datenbanksprache</b>	
Der Begriff alphanumerische Zeichen wird hauptsächlich im Bereich der Computertechnik bzw. Telekommunikation verwendet. Alphanumerische Werte sind Zeichenketten die aus Ziffern, Buchstaben bzw. Sonderzeichen bestehen können.		
<b>Alias</b>	<b>Datenbestand</b>	
Ein Alias ist ein Pseudonym das anstelle eines bestimmten Bezeichners oder Wertes verwendet werden kann.		
<b>Anforderung</b>	<b>Datendefinitionssprache</b>	
<b>Anwendungslogik</b>	<b>Datenmodell</b>	
<b>Aspect</b>	<b>Datenbankschema</b>	
Ein Aspekt beschreibt einen einzelnen Gesichtspunkt bei der Betrachtung, Auswertung oder Analyse komplexer Sachverhalte.		
<b>B</b>	<b>Datenkontainer</b>	
<b>Baumstruktur</b>		
	<b>Datenredundanz</b>	
<b>Baustein</b>		
	<b>Datenschema</b>	
<b>Bedingung</b>	Ein Datenschema beschreibt die physische Struktur der Daten einer Datenbank.	
Eine Bedingung ist eine Funktion, die immer den Wahrheitswert true oder false liefert.		
	<b>Gegenoperation</b>	
<b>Datenstruktur</b>	<b>G</b>	
<b>Datentyp</b>	<b>Geschäftsobjekt</b>	
<b>Deklarative Programmierung</b>		
Die deklarative Programmierung ist ein Programmierparadigma, bei dem die Beschreibung eines Problems im Vordergrund steht. Im Gegensatz zur imperativen Programmierung, bei der ein genauer Lösungsweg beschrieben werden muss, fragt man bei der deklarativen Programmierung was berechnet werden soll.		
<b>deterministisch</b>	<b>H</b>	
<b>Differenzmenge</b>	<b>Hierarchische Struktur</b>	
<b>Durchschnittsmenge</b>		
	<b>I</b>	
<b>E</b>	<b>idempotent</b>	
<b>Entität</b>		
	<b>Inkonsistenz</b>	
<b>Exception</b>		
	<b>K</b>	
	<b>Komplexität</b>	
<b>F</b>		
<b>Formatmaske</b>	<b>Komplementäroperation</b>	
Formatmasken geben ein bestimmtes Muster vor, das zur Formatierung von Zeichenketten herangezogen wird. Die Formatmaske selbst ist dabei ebenfalls eine Zeichenkette.		
<b>Fremdschlüssel</b>	<b>Konsistenz</b>	
<b>Funktion</b>		
Eine Funktion oder Projektion definiert eine Beziehung zweier Mengen, die jedem Element der einen Menge (Ausgangswerte), genau ein Element der anderen Menge (Zielwerte) zuordnet.	<b>Kontext</b>	
<b>Funktionsparameter</b>	Als Kontext ist jede Art von Information gemeint, die für die Interaktion der Objekte eines Systems notwendig ist.	
Funktionsparameter sind Werte, die einer Funktion zur Verarbeitung mitgegeben werden..	<b>Kontextswitch</b>	

267

## Informationssysteme

<b>Datenstruktur</b>	<b>G</b>	
<b>Datentyp</b>	<b>Geschäftsobjekt</b>	
<b>Deklarative Programmierung</b>	<b>Gegenoperation</b>	
Die deklarative Programmierung ist ein Programmierparadigma, bei dem die Beschreibung eines Problems im Vordergrund steht. Im Gegensatz zur imperativen Programmierung, bei der ein genauer Lösungsweg beschrieben werden muss, fragt man bei der deklarativen Programmierung was berechnet werden soll.		
<b>deterministisch</b>	<b>H</b>	
<b>Differenzmenge</b>	<b>Hierarchische Struktur</b>	
<b>Durchschnittsmenge</b>		
	<b>I</b>	
<b>E</b>	<b>idempotent</b>	
<b>Entität</b>		
	<b>Inkonsistenz</b>	
<b>Exception</b>		
	<b>K</b>	
	<b>Komplexität</b>	
<b>F</b>		
<b>Formatmaske</b>	<b>Komplementäroperation</b>	
Formatmasken geben ein bestimmtes Muster vor, das zur Formatierung von Zeichenketten herangezogen wird. Die Formatmaske selbst ist dabei ebenfalls eine Zeichenkette.		
<b>Fremdschlüssel</b>	<b>Konsistenz</b>	
<b>Funktion</b>		
Eine Funktion oder Projektion definiert eine Beziehung zweier Mengen, die jedem Element der einen Menge (Ausgangswerte), genau ein Element der anderen Menge (Zielwerte) zuordnet.	<b>Kontext</b>	
<b>Funktionsparameter</b>	Als Kontext ist jede Art von Information gemeint, die für die Interaktion der Objekte eines Systems notwendig ist.	
Funktionsparameter sind Werte, die einer Funktion zur Verarbeitung mitgegeben werden..	<b>Kontextswitch</b>	

268

<b>Konvertieren</b>	<b>Patternmatching</b>
Als Konvertierung wird die Umwandlung des Wertes eines Datentyps in den Wert eines anderen Datentyps bezeichnet.	Patternmatching beschreibt ein Verfahren zur musterbasierten Suche in einer Zeichenkette. Dazu wird anhand einer Suchmaske geprüft ob eine bestimmte Zeichenfolge in einer Zeichenkette vorhanden ist.
<b>L</b>	
<b>Laufzeit</b>	<b>plattformübergreifend</b>
Die Laufzeitumgebung beschreibt die zur Laufzeit von Programmen verfügbaren Standardbibliotheken, Programmierschnittstellen, Laufzeitvariablen sowie die Hard- und Softwarekomponenten des Betriebssystems..	Als Plattformunabhängigkeit wird die Eigenschaft eines Systems beschrieben, dass ein Programm auf unterschiedlichen Plattformen ausgeführt werden kann. Ein Programm wird daher in einer sogenannten Laufzeitumgebung ausgeführt. Eine Laufzeitumgebung lädt dazu ein Programm und lässt diese auf der gewünschten Plattform ablaufen.
<b>Literal</b>	<b>Primärschlüssel</b>
Als Literal bezeichnet man in der Programmierung eine Zeichenkette, die zur direkten Darstellung der Werte der Basistypen verwendet werden können.	
<b>logische Struktur</b>	<b>Programmablauf</b>
Die logische Struktur einer Datenbank beschreibt die Beziehungs- und Speicherstruktur der Datensätze.	Als Programmablauf wird in der IT der Ablauf eines Programms bezeichnet. Zur Steuerung des Programmablaufs werden in der Programmierung Kontroll- bzw. Schleifenkonstrukte verwendet.
<b>M</b>	<b>Programmkontrolle</b>
<b>Modul</b>	
	<b>Projektion</b>
<b>N</b>	
<b>Namensraum</b>	Eine Projektion oder Funktion definiert eine Beziehung zweier Mengen, die jedem Element der einen Menge (Ausgangswerte), genau ein Element der anderen Menge (Zielwerte) zuordnet..
<b>Neustrukturierung</b>	<b>Q</b>
	<b>Query by Example</b>
<b>P</b>	<b>R</b>
<b>Paginierung</b>	<b>Referenz</b>
Als Paginierung wird die Seitennummernierung eines Schriftstücks bezeichnet. In der Dataverarbeitung versteht man unter Paginierung, den seitenweisen Zugriff auf das Ergebnis einer Abfrage.	
	<b>Relation</b>

269

<b>Restriktion</b>	<b>Transformation</b>
Als Restriktion wird eine Bedingung bezeichnet, die die Datensätze eines Informationssystems erfüllen müssen..	
<b>S</b>	<b>U</b>
<b>Schlüssel</b>	<b>Unterabfrage</b>
<b>Spezifikation</b>	
Die Spezifikation eines Softwaresystems beschreibt alle funktionalen und nichtfunktionalen Anforderungen an die Implementierung der Anwendung.	
<b>Strukturierung</b>	<b>V</b>
	<b>Variable</b>
<b>T</b>	<b>W</b>
<b>Tabelle</b>	<b>Wertzuweisung</b>
<b>Token</b>	<b>Index</b>
Als Token wird eine kurze Zeichenkette beliebiger Zusammensetzung bezeichnet..	
	<b>Index</b>
<b>ACID Eigenschaften, 136</b>	<b>BSON, 149</b>
<b>ACID Transaktionsmodell, 136</b>	
<b>Aggregationsfunktionen</b>	<b>CA Systeme, 133</b>
count, 53	CAP Theorem, 131
max, 53	AP Systeme, 134
min, 53	Auffallsicherheit, 131
sum, 53	CA Systeme, 133
<b>Aggregate Phase, 57</b>	CP Systeme, 134
<b>Aggregationsfunktion</b>	Konsistenz, 131
median, 54	Verfügbarkeit, 131
<b>Aggregationsfunktionen, 53</b>	conca(), 228
stddev, 54	Consistent Hashing, 138
Aggregation, 22, 53	contains(), 228
Aggregationsframework, 177	count(), 227
AP Systeme, 134	CP Systeme, 134
asd, 31	Cursor, 113, 158
Atomarität, 136	Daten, 3
<b>Bigdata, 127</b>	Datenbankartefakte, 69
Variety, 128	Datenformat, 5
Velocity, 128	csv Format, 5
Volume, 128	json Format, 6
<b>Block, 92</b>	xml Format, 5

270

Datenmodellierung, 33  
 Datensatz, 4  
 Datenverarbeitung, 3  
 Datumsfunktionen, 40  
   add\_month, 45  
   extract, 45  
   last\_day, 45  
   month\_between, 46  
   next\_day, 46  
   sysdate, 46  
   timestamp, 41  
   to\_date, 41  
 Daueraufgabe, 136  
 DDL Befehl  
   create view, 75  
   drop view, 76  
 deklarative Programmierung, 21  
 desc, 31  
 DML Befehl  
   insert ... select, 81  
   update, 82  
 DML Befehlsatz, 81  
 Dokumentenschema, 151  
 Embedded Documents, 150  
 Externe Phase, 10  
 Fetch Klausel, 32  
 From Klausel, 34  
 Funktion, 117  
 Group by  
   Aggregate Phase, 55  
   Map Phase, 55  
 Group By Klausel, 22  
 Having Klausel, 57  
 Index  
   B\* Index, 77  
   Funktionsbasiert, 77  
   Volltextindex, 78  
 Information, 3  
 Informationsmanagement, 3, 21, 33, 131  
 Inner Join, 35  
 Isolation, 136  
 Join, 34  
 JSON, 255  
 Konsistenz, 136  
 Konsistenzmodell, 135  
   eventual consistency, 135  
   strict consistency, 135  
 MongoDB, 171

271

\$addFields, 183  
 Multi Version Concurrency Control, 137  
 MVCC, 137  
 name(), 227  
 namespace-uri(), 227  
 NoSQL-Datenbanken, 130  
 Numerische Funktionen, 50  
   round, 50  
   to\_char, 51  
   trunc, 50  
 Offset Klausel, 32  
 Optimistic Locking, 137  
 Order By Klausel, 23, 31  
 Physische Phase, 11  
 Pipeline, 178  
 PL/SQL  
   Block, 92  
   Datenbankfunktionalität, 90  
   Datensicherheit, 90  
   Datennutzung, 91  
   Datennutzung, 91  
   Datensicherheit, 90  
   Datenzugriff, 91  
   Funktion, 117  
   Prozedur, 115  
   Qualifier, 96  
   Variablen, 95  
 PL/SQL Befehl  
   begin ... block, 93  
   create ... function, 117  
   create ... package body, 119  
   declare ... block, 93  
   end ... block, 93  
   exception ... block, 93  
 PL/SQL Block  
   Ausschreibungsteil, 93  
   Deklarationsteil, 93  
   Fehlerbehandlungsteil, 93  
   Funktion, 117  
   Prozedur, 115  
   Trigger, 120  
 PL/SQL Engine, 91  
 position(), 227  
 Projektion, 22  
 Prozedur, 115  
 Prädikat, 224  
 Qualifier, 96  
 Reduce Phase, 140  
 Right Join, 36

272

substring before(), 228  
Template, 233  
Textfunktionen, 47  
initcap, 49  
instr, 47  
length, 47  
lower, 49  
ltrim, 47  
replace, 49  
rtrim, 47  
soundex, 48  
substr, 49  
trim, 47  
upper, 49  
Transaction, 136  
ACID Transaktionsmodell, 136  
Atomarität, 136  
Daueraufgabigkeit, 136  
Isolation, 136  
Konsistenz, 136  
Trigger, 120  
Vaiety, 128  
Velocity, 128  
Verticelle Datenbank, 129  
Verticelle Datenbanken, 129  
View, 75  
create view, 75  
Virtuelle Tabellen, 34  
Virtuelle Tabellen, 55  
Aggregate Phase, 57  
Map Aggregate, 55  
Volume, 128  
Where Klausur, 22, 28  
Wissen, 3  
with ties, 32  
  
XML  
Attribute, 215  
Element, 213  
Schema, 245  
XPath, 221  
XML Attribute, 215  
XML Element, 213  
XML Schema, 245  
Xml Schema, 245  
XPath, 221  
. Operator, 223  
.. Operator, 223  
// Operator, 223  
concat(), 228  
trim, 47

contains(), 228  
count(), 227  
last(), 227  
localName(), 227  
Lösungsobjekt, 222  
name(), 227  
normalize(), 227  
position(), 227  
Prüfdat., 224  
starts-with(), 228  
string(), 228  
string length(), 228  
substring(), 228  
substring after(), 228  
substring before(), 228  
XSLT, 231  
Stylesheet, 233  
Template, 233  
xsl:apply-templates, 235  
xsl:choose, 238  
xsl:copy, 241  
xsl:element, 240  
xsl:for-each, 239  
xsl:if, 237  
xsl:otherwise, 237  
xsl:param, 236  
xsl:value-of, 237  
xsl:variable, 237  
xsl:when, 238  
xsl:with-param, 236  
  
Zeilenumfunktionen  
add\_month, 45  
extract, 45  
initcap, 49  
instr, 47  
last\_day, 45  
length, 47  
lower, 49  
ltrim, 47  
months\_between, 46  
next\_day, 46  
replace, 49  
round, 50  
rtrim, 47  
soundex, 48  
substr, 49  
sysdate, 41  
timestamp, 41  
to\_char, 51  
to\_date, 41  
trim, 47

273

## Informationssysteme

trunc, 50  
upper, 49

274

ER-Modelle (Entität - Relation, Transformationsregeln)

Normalisierung (bis 3. Normalform)

SQL - DQL (select)

Übung - Datenbank importieren (source)

DDL (create table,...)

Seiten v. Skript: 10 - 76

# Probetest.docx

Donnerstag, 18. November 2021 17:08



1. Was ist eine Entität? Erkläre genau!
2. Zähle die 5. Normalformen auf und erkläre 2 davon genauer!
3. Was versteht man unter ER-Modellierung? Was ist eine Workbench?
4. Zähle mind. 5 Befehle auf, die zum SELECT-Statement dazugehören! Erkläre gegebenenfalls auch, was diese Befehle können!
5. Wie ändert man bestehende Tabellen, was muss man machen, wenn die Tabelle noch nicht existiert?
6. Was macht dieser Befehl?

```
SELECT E.FIRST_NAME,
       E.LAST_NAME
          E.SALARY,
     FROM EMPLOYEES E
RIGHT JOIN DEPARTMENTS D
ON E.DEPARTMENT_ID = D.DEPARTMENT_ID;
```

1. Entität = Tabelle; darin speichert man Daten

2. 1. Normalform → atomare Attribute

2. Normalform

3. - - -

4. - - -

5. - - -

→ Tupel nur durch ganzen Schlüssel identifizierbar

3. inkrementes, geschichtetes Aufbereiten & Speichern von Daten (SQL)

WB: Datenbank-Modellierungswerkzeug; z.B.: MySQL

4. WHERE: Bedingung für bestimmtes Attribut

ORDER BY: Sortieren nach Attribut

GROUP BY: Gruppieren nach Attribut

FETCH: nur bestimmte Anzahl an Attributen anzeigen

OFFSET: mit FETCH: Beginn dieser Anzahl

LIMIT: wie FETCH

JOIN: zusammenfügen von Daten aus zwei od. mehreren Tabellen

HAVING: WHERE von GROUP BY

CASE: IF von SQL

LIKE: Zeichenketten muss in Attribut vorkommen

5. ALTER TABLE / CREATE TABLE ← Tabelle erzeugen

6. gibt FN, LN, SALARY von allen Mitarbeitern aus, deren Department

ID mit der DEPARTMENT\_ID von der DEPARTMENTS  
Tabelle übereinstimmt

Tabelle übereinstimmt

# INSY Test, am 19.11.2021

Donnerstag, 6. Januar 2022 14:47

Felix Schneider  
19.11.2021

3AHIT	INSY-Test	19.11.2021
-------	-----------	------------

Name:

20 / 20	Punkte	0 – 10	10,1 – 13,9	14 – 16,9	17 – 18,9	19 – 20
	Note	Nicht genügend	Genügend	Befriedigend	Gut	Sehr gut

1. Relationale Modellierung - Gegeben ist folgender Sachverhalt:  
Eine Person ist eindeutig durch eine ID identifiziert. Für eine Person wird weiters gespeichert: ein Vorname (Varchar2(20) - not null), ein Nachname (Varchar2(20) - not null) und eine QUID (Varchar2(20) - not null, unique).  
Von den Personen abgeleitet werden Studenten und Professoren (Vererbung). Für Studenten wird zusätzlich eine MatrikelNr. (Varchar2(8) - not null, unique) und das Datum der Inskribierung (Date - not null) gespeichert, für Professoren das Institut (Varchar2(50) - not null) und ein Forschungsschwerpunkt (Varchar2(100)).  
Entwickeln Sie ein Entitätenmodell! (4)

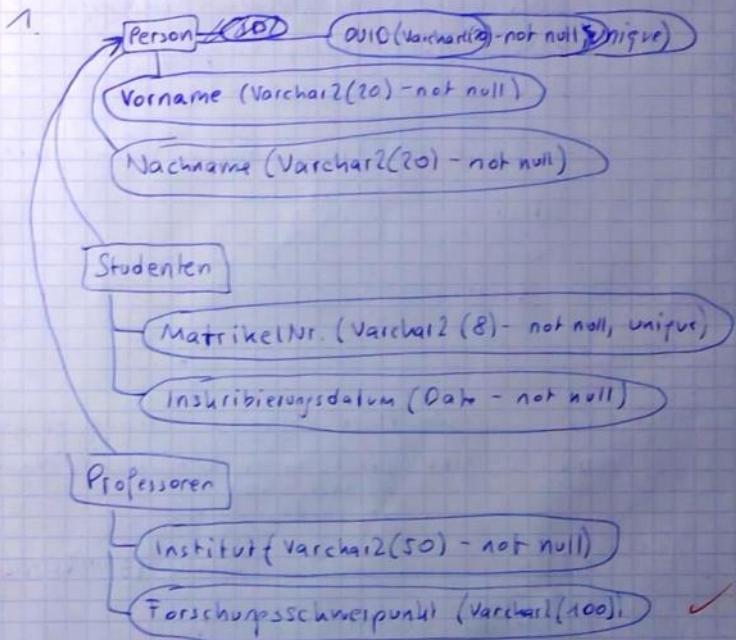
2. Normalformen  
Entwickeln Sie aus folgender Tabelle in der „0. NF“ ein so weit wie möglich normalisiertes Tabellenmodell! (7)

OID	Name	Ableitung	ProjektNr	ProjektName	ProjektZeit
101	Panhofer Paul	1:Physik	11, 12	A, B	60, 40
102	Lauer Christian	2:Chemie	13	C	100
103	Bauer Iris	2:Chemie	11, 12, 13	A, B, C	20, 50, 30
104	Kauer Alexander	1:Physik	11, 13	A, C	80, 20

3. SQL

- Aus welchen Teilen setzt sich der SQL-Befehlssatz zusammen? (2)
- Was ist eine „innere View“? (1)
- Beschreiben Sie folgendes SQL-Statement! (3)  

```
mysql> select first_name as Vorname, last_name as Nachname, salary as Gehalt from EMPLOYEES  
where salary > 5000 and job_id = 'IT_PROG' order by salary desc;
```
- Formulieren Sie eine SQL-Abfrage, in deren Ergebniszeile(n) beliebige Spalten aus den beiden Tabellen Table1 und Table2 vorkommen, wenn der Inhalt der Spalte ID in beiden Tabellen gleich ist! (3)



2.

siehe  
ZUSATZ-  
ZETTEL

↓

OID	Vorname	Nachname	Abt.-ID	Proj.-ID	P.Zent
101	Paul	Panköfer	1	11, 12	60,40
102	Christian	Lauer	2	13	100
103	Iris	Bauer	2	11, 12, 13	20,20,20
104	Alexander	Kauer	1	11, 13	30,20

Abt.-ID	Abt.-Name
1	Physik
2	Chemie

Proj.-ID	Proj.-Name
11	A
12	B
13	C

2.

## PERSON EMPLOYEES:

OID	Vorname	Nachname	Abteilung	n : 1	Abteilung	Name
101	Paul	Panhofer	1		1	Physik
102	Christian	Lauer	2		2	Chemie
103	Iris	Bauer	2			
104	Alexander	Kauer	1			

## ABTEILUNGEN:

Abteilung	Name
1	Physik
2	Chemie

## EMPLOYEES - has - PROJEKTE :

OID	ID	Zeit [min]
K 101	11	60
K 101	12	40
K 102	13	100 usw...

(← SCHLÜSSELTABELLE)

## PROJEKTE :

ID	Name	Ziel
11	A	
12	B	
13	C	

EMPLOYEES  
und  
PROJEKTE  
haben  
n:m - Beziehung.



3. a. ~~D~~ DDL... Data Definition Language

DML... Data Manipulation Language

DQL... Data Query Language

DCL... Data Control Language ✓

b. ein Select im From vom äußeren Select

Select ...

From

(Select ...) ✓

i

c. Select... ~~set~~ Hole Vorname, Nachname, Gehalt

as ... gibt first-name soweit "Namen"

od. last-name

od. salary

From... von Welcher Tabelle (EMPLOYEES)

Where... bestimmte Bedingung

↳ Gehalt größer als 5000

JOB.ID = 'IT-PROG'

↳ nur IT-Ler  
nur Programmierer

order by... sortieren (absteigend) nach

Gehalt ✓

d. select first-name, last-name from EMPLOYEES E

inner join DEPARTMENTS D

on E.ID = D.ID; ✓

Bäume ( $\beta + \text{Baum}$ )

SQL (DDL, DML, DCL, DQL)

↓      ↓      ↓      ↓  
Create    Update    Benoteen    Select  
insert    insert  
delete

# Probetest

Donnerstag, 20. Januar 2022 17:23



INSY-Probetest

Felix Schneider

20.01.2022

1. Gibt die Anzahl an Mitarbeitern und die Anzahl an Abteilungen in einem Land aus! Wenn in einem Land niemand arbeitet, soll 0 ausgegeben werden. Wenn sich in einem Land keine Abteilung befindet, soll 0 ausgegeben werden.

```
MariaDB [hr]> select c.country_name as Country, count(d.department_id) as Department_Anzahl, count(e.employee_id) as Employee_Anzahl from EMPLOYEES e  
right join DEPARTMENTS d using(department_id) right join LOCATIONS l using(location_id) right join COUNTRIES c using(country_id) group by c.country_id  
;
```

2. Gibt den maximalen Gehalt, den Durchschnittsgehalt und den Minimalgehalt aus dem Department „Marketing“ aus!

```
Select max(salary), avg(salary), min(salary) from EMPLOYEES natural  
join DEPARTMENTS where department_name = "Marketing";
```

3. Füge in der Tabelle „Animals“ einen neuen Eintrag mit folgenden Daten hinzu: ID: 62; Name: „Elephant“; MaxAge: 80; Species: „Mammal“!

```
INSERT INTO Animals (ID, Name, MaxAge, Species)  
VALUES (62, "Elephant", 80, "Mammal");
```

4. Erzeuge eine neue Tabelle mit folgenden Spalten: ID, Straße, Hausnummer, Ort, PLZ! Gibt der Tabelle einen sinnvollen Namen!

```
CREATE TABLE Adresse (  
    ID int NOT NULL AUTO_INCREMENT,  
    Strasse varchar(50) NOT NULL;  
    Hausnummer int NOT NULL;  
    Ort varchar(100) NOT NULL;  
    PLZ int NOT NULL;  
    PRIMARY KEY(ID);
```

// ← NOT NULLS  
& AUTO\_INCREMENT  
muss nicht sein  
(war nicht geprägt)  
(wäre aber optimal)

Ort varchar(100) NOT NULL  
PLZ int NOT NULL  
PRIMARY KEY(id);

(war nicht gefragt)  
(wäre aber optimal)

5. Wahrscheinlich hast du die Tabelle „Adresse“ benannt. Allerdings willst du in Englischer-Sprache (z.B.: Address) bleiben. Nenne aus diesem Grund die Tabelle und alle Spalten um (nicht neu erzeugen)!

RENAME TABLE Adresse TO Address;  
ALTER TABLE Address RENAME Strasse TO Street;  
ALTER TABLE Address RENAME Hausnummer TO House\_Number;  
ALTER TABLE Address RENAME Ort TO Place;  
ALTER TABLE Address RENAME PLZ TO Postal\_Code;

6. Zeichne einen B+-Baum, der als Werte das Alphabet in umgekehrter Reihenfolge gespeichert hat und maximal 4 Werte in den Knoten aufnehmen kann! Schreibe außerdem die Namen der Knoten hinzu!

Wurzelknoten:



Definition: Suchbereich  
(120 → beim kleineren Knoten zu finden)

Indexknoten:



Blattknoten:



7. Zeichne nun mind. 3 Schritte des Baumes auf, die du benötigen würdest um deinen eigenen Namen alphabetisch (wieder umgekehrt) einordnen zu können!



8. Lösche das nie verwendete View „Dragonball“ aus der Datenbank!

DROP VIEW Dragonball;

# Test, am 21.01.2022

Dienstag, 25. Januar 2022 11:49

BAHIT	INSY-Test	21.01.2022				
Name: Felix Schneider						
19 / 20	Punkte	0 – 10	10,1 – 13,9	14 – 16,9	17 – 18,9	19 – 20
	Note	Nicht genügend	Genügend	Befriedigend	Gut	Sehr gut

1. B+-Baum für Indizes

- Was speichert ein B+-Baum in den Indexknoten? (1) ✓
- Was muss passieren, wenn ein neuer Schlüssel in ein bereits volles Blatt eingefügt werden soll? (2) ✓
- Zu wie viel Prozent muss ein Blatt mindestens gefüllt sein? (1) 0

2. SQL

- Was bewirkt das folgende Statement – zu welcher Befehlsguppe gehört es? (4) ✓  
mysql> grant all privileges on mydb.\* to 'user'@'%'
- Mit welchem Befehl können Sie einem Benutzer Berechtigungen wieder entziehen? (1) ✓
- Analysieren Sie folgendes Statement! Wie sieht das erstellte Datenbank-Artefakt aus und in welchen Beziehungen steht es zu anderen Artefakten? (5) ✓  
CREATE TABLE employees (  
employee\_id NUMBER(19,0) NOT NULL,  
salary NUMBER(10,0) NOT NULL,  
job\_id NUMBER(19,0) NOT NULL,  
first\_name VARCHAR(30) NOT NULL,  
middle\_name VARCHAR(30) NOT NULL,  
last\_name VARCHAR(30) NOT NULL,  
department\_id NUMBER(19,0) NOT NULL,  
  
PRIMARY KEY (employee\_id),  
  
CONSTRAINT fk\_departments\_dep\_id  
FOREIGN KEY (department\_id)  
REFERENCES departments (department\_id)  
);
- Gegeben sind die folgenden beiden Tabellen T und U. Skizzieren sie die Ergebnismengen eines „Joins“ und eines „Left Joins“ bei der Join-Bedingung T.Sp1 = U.Sp1! (4) ✓

T			
Sp1	Sp2	Sp3	Sp4
21	12	13	14
32	16	17	18
47	20	21	22

U		
Sp1	Sp5	Sp6
21	34	36
47	55	56

- Formulieren Sie die beiden SELECT-Statements für die Teilaufgabe 2d! (2) ✓

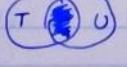
1. a. Die Zahl, welche man dann entweder im kleineren Blattknoten findet oder im größeren (kommt auf Definitionsbereich an)  
 bzw.: (der Verweis) die Grenzbereiche (so wird suchen  
 2B.: ABBA alphabetisch einfacher...)
- b. 1. Blattknoten hinzufügen  
 2. einzelnen Wert in neues Blatt verschieben  
 3. Wert (neuen Schlüssel) einfügen  
 4. Schlüssel wieder auffüllen.  
 5. Index & Wurzelknoten update
- c.  $66,6\% \quad \frac{2}{3} \leftarrow 3^*$

2. a. Der Befehl gibt dem User user die Erlaubnis alle Befehle zu verwenden  
 epal welche IP-Adr.  
 für Datenbank mydb.  
 alle Tabellen → DCL  
 Data Control Language
- b. revoke ✓
- c. Es wird eine Tabelle namens employees erstellt, die hat einen PRIMARY KEY employee-id, fünf weitere Attribute → Attribute/Spalten (salary, job-id, first-name, middle-name, last-name) und den FOREIGN KEY department-id, der auf den Primary Key department-id aus der Tabelle departments zielt / verweist.
- |             |             |        |        |    |    |    |               |
|-------------|-------------|--------|--------|----|----|----|---------------|
| Primary Key | employee-id | salary | job-id | fn | mn | ln | department-id |
|-------------|-------------|--------|--------|----|----|----|---------------|
- Diese Tabelle steht in n:1 Beziehung zur department-Tabelle.

2.d. Notice: (select \* from T <sup>inner</sup> join U on Sp1 = Sp1;)

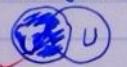
join:

	T				U		
Sp1	Sp2	Sp3	Sp4	Sp1	Sp5	Sp6	
21	12	13	14	21	34	36	
47	20	21	22	47	55	56	

INNER JOIN  


left join: (select \* from T left join U on Sp1 = Sp1;)

	T				U		
Sp1	Sp2	Sp3	Sp4	Sp1	Sp5	Sp6	
21	12	13	14	21	34	36	
32	16	17	18	NULL	NULL	NULL	
47	20	21	22	47	55	56	

LEFT JOIN  


e. select \* from T + inner join U u on t.Sp1 = u.Sp1; \*

select \* from T + left join U u on t.Sp1 = u.Sp1;

\* möglich wäre auch:

select \* from T natural join U;  
 ↑  
 (einfacher)

# Test, am 01.04.2022

Freitag, 20. Mai 2022 12:28

3AHIT	INSY-Test	01.04.2022				
Name: Felix Schneider						
18 / 20	Punkte	0 – 10	10,1 – 13,9	14 – 16,9	17 – 18,9	19 – 20
	Note	Nicht genügend	Genügend	Befriedigend	Gut	Sehr gut

1. Bigdata / NoSQL-Grundlagen

- Was sagen die 3 V aus, mit denen Bigdata charakterisiert werden kann? (3) ✓
- Was bedeuten die drei Buchstaben C, A und P beim CAP-Theorem und was sagt dieses Theorem aus? (4) ✓
- In welche Kategorie im Rahmen des CAP-Theorems fallen relationale Datenbanken? (1) ✓
- Mit welchem Konsistenzmodell arbeiten NoSQL-Datenbanken? Beschreiben Sie die Eigenschaften des dabei verwendeten Transaktionskonzeptes! (4) ✓

2. MongoDB

- Wie können Sie auf der MongoDB-Konsole nachsehen, welche Databases es schon gibt und wie legen Sie eine neue Database an? (2) ✓
- Was können Sie durch die Verwendung eines JSON-Schemas in einer Collection einer MongoDB erreichen? (2) ✓
- Geben Sie den Befehl zum Erstellen einer MongoDB-Collection „Schueler“ an, deren Documents zumindest die Felder „Nachname“, „Vorname“ und „Geburtsdatum“ aufweisen sollen! (4) ✓

- a. Velocity → Geschwindigkeit (soll schnell sein,  
 Variety → Schema kann bei und nicht ewig  
 Dokumenten verschieden sein sperren)  
 Volume → viele Daten können verarbeitet werden
- b. Consistency → Konsistenz ist gegeben (alle Datenbestände immer aktuell, keine Widersprüche)
- Availability → immer verfügbar, ~~sieht~~  
 (keine Sperren)

Partition Toleranz → selbst wenn ein Server ausfällt → Datenbestand muss erhalten bleiben

Es können immer nur 2 der 3 Kategorien erfüllt sein. Es geht aus, welche Eigenschaften die Datenbank aufweist.

- c. CP - Modelle sind relationale Datenbanken
- CA - Modelle = NoSQL
- AP - Modelle = NoSQL (umgesetzt)
- BASE → Schwartz; MySQL gebunden.  
 \* getaktete Vorgaben, wie Tabellen aussehen müssen
- d. Eventually Consistency → nicht so streng (Formatformen, Transaktionen & Sperren z. B.)

ACID →

Acknowledgment → Datenbestände in Einzelhülle zerlegt

Consistency → keine Widersprüche (keine Redundanz)

Isolation → außerhalb d. Transaktion keine Infos

ACID:

1.d. Durations → Datenbestände bleiben dauerhaft

2.a. show databases;

use <database>; ✓ z.B.: use projectdata;

b. Dass dieses Schema erfüllt sein muss  
(Mindestanforderungen bzw. Regeln)

z.B.: Dokumente müssen title & description enthalten

c. db.createCollection("Schueler", {

validationLevel: "strict",

validationAction: "error",

validator: {

jsonSchema: {

bsonType: "object",

required: ["Nachname", "Vorname",  
"Geburtsdatum"],

additionalProperties: true,

properties: {

Nachname: {

bsonType: "string",

minLength: 2,

maxLength: 40

},

Vorname: { bsonType: "string" },

(2.c.)

```
Geburtsdatum: {  
    bsonType: "date"
```

{}

{}

{);

\*

1.d. additionalProperties lässt z.B. zu, dass auch andere Properties vorkommen

BASE:

Basic Availability: Grundsätzlich zu jederzeit abrufbar

Soft State: Datenbank muss nicht direkt überall neuesten Infos haben; Redundanz ist (Duplikation) erlaubt

Eventually Consistent: irgendwann

Es wird garantiert, dass irgendwann alle Daten konsistent sind.

# Test, am 02.06.2022

Dienstag, 7. Juni 2022 11:35

3AHIT	INSY-Test	02.06.2022
-------	-----------	------------

Name: Felix Schneider

26 / 20	Punkte	0 – 10	10,1 – 13,9	14 – 16,9	17 – 18,9	19 – 20
	Note	Nicht genügend	Genügend	Befriedigend	Gut	Sehr gut

1. MongoDB

a. Ein find()-Statement ohne Einschränkungen liefert folgendes Ergebnis:

```
> db.student.find().pretty()
{
    "_id" : ObjectId("60227eaff19652db63812e8d"),
    "name" : "Akshay",
    "age" : 18
}
{
    "_id" : ObjectId("60227eaff19652db63812e8e"),
    "name" : "Bablu",
    "age" : 17,
    "score" : {
        "math" : 230,
        "science" : 234
    }
}
{
    "_id" : ObjectId("60227eaff19652db63812e8f"),
    "name" : "Chandhan",
    "age" : 18
}
> //
```

i. Mit welchem Statement findet man nur die Namen jener Studenten, die 18 Jahre oder älter sind? (2) ✓

ii. Mit welchem Statement kann man sich die kompletten Datensätze nach dem Alter absteigend sortiert ausgeben lassen? (2) ✓

b. Verkürzen/vereinfachen Sie die folgende Abfrage so weit wie möglich! (3) ✓

```
db.projects.find({
    $or : [
        {
            $and : [
                {type : {$eq : "FUNDING_PROJECT"}},
                {isFFGSponsored : {$eq : true}}
            ]
        },
        {
            $and : [
                {type : {$eq : "REQUEST_PROJECT"}},
                {isFWFSponsored : {$eq : true}}
            ]
        }
    ]
});
```

1ai) db.student.find({

    age: {\$gte: 18}  
}, {

    name: 1,  
    \_id: 0

}).pretty();

ii) db.student.find().sort({age: -1}).pretty();

b. db.projects.find({

    \$type: ["FUNDING-PROJECT"],  
    isFFG\_Sponsored: true  
}, {

    \$type: "REQUEST-PROJECT",  
    isFWFSponsored: true

});

});

(es können der \$and und der \$eq - Operator  
überall weggelassen werden)

- 2.a.
- 1) ROOT-Element (nur eines)!
  - 2) Elemente dürfen sich nicht überlappen
  - 3) jedes Element hat Anfangs- und Endtag!

b.  $\rightarrow$  zettel

c.

```
<xs:simpleType name="Login-Name">
  <xs:restriction base="xs:string">
    <xs:minLength value="2"/>
    <xs:maxLength value="8"/>
  </xs:restriction>
</xs:simpleType>
<xs:element name="Hugo" type="Login-Name"/>
```

d. Es wird definiert, welche Elemente in diesen exakten Reihenfolge  $\rightarrow$  in dem komplexen Datentypen vorzukommen haben.  
(alle angegebenen Elemente müssen so vorkommen)

e. Es wird definiert, welches Element (maximal 1 Element) aus der Auswahl vorkommen kann.  
(nur 1 Element darf vorkommen)