



Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

Vorlesung Datenbanken und Informationssysteme I Sommersemester 2017

Prof. Dr. rer. nat. Oliver P. Waldhorst

Termine

Vorlesung mit integrierten Übungen:

- Mittwoch, 09:50 – 11:20 Uhr in Raum M304
- Donnerstag, 09:50 – 11:20 Uhr in Raum M304

Übungen mit integrierter Vorlesung:

- Dienstag, 14:10 – 15:40 Uhr in Raum MU11 – Gruppe a bzw.
- Dienstag, 15:40 – 17:10 Uhr in Raum MU11 – Gruppe b

Tutorium (Details folgen)

Prüfungen

Eine Prüfungsklausur

- schriftlich - 90 min
- Ende des Semesters
- Fragen und Aufgaben zum Inhalt der Vorlesung
- Durch drei Hausaufgaben während des Semesters können Punkte für die Klausur gesammelt werden

Hausaufgaben

- Gruppenarbeit 3-4 Studierende
- Aufgaben werden eine Woche vor Abgabe online gestellt

Punkteverteilung:

- Abgabe 1 – 33%
- Abgabe 2 – 33%
- Abgabe 3 – 33%

- Bis zu 10% der Abschlussnote

Vorlesungsorganisation über ILIAS

Bitte melden Sie sich für die Vorlesung an!

- Tragen Sie sich in <http://ilias.hs-karlsruhe.de> für die Lehrveranstaltung Datenbank und Informationssysteme 1 ein
- Sie benötigen das Password: DBSYS

Wichtig: in ILIAS navigieren sie über Dozenten zu der Lehrveranstaltung

Literatur (1)

Frank Geisler:
Datenbanken - Grundlagen
und Design
5. Auflage, mitp, 2014

- Einführend, leicht verständlich

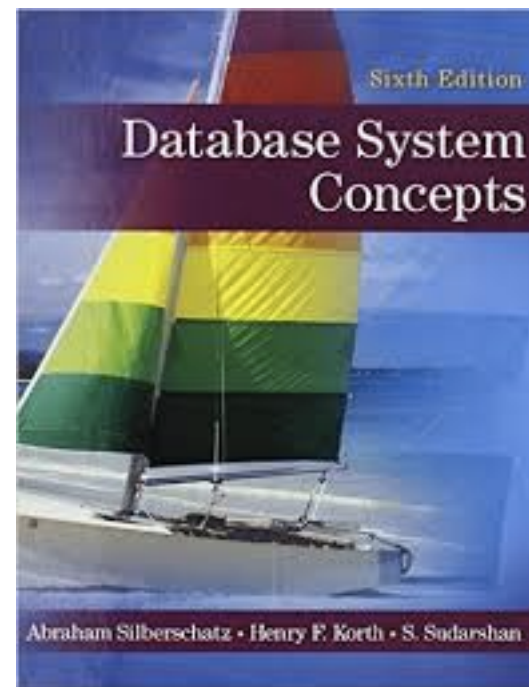


Literatur (2)

Abraham Silberschatz, Henry F. Korth, S. Sudarshan:
Database System
Concepts (Englisch)
6. Auflage, Mcgraw-Hill, 2010

- Weiterführend, behandelt SQL-Teil besser

Weitere Hinweise in
Vorlesung / ILIAS



Gliederung

1. Einführung
2. Datenbanksysteme und Anwendungen
3. Das Relationale Datenbankmodell
4. Datenbanknormalisierung
5. Relationale Algebra
6. Einführung in SQL
7. Grundlegende SQL-Konzepte
8. Fortgeschrittenes SQL
9. Datenbankmodellierung



Kontakt

Prof. Dr. rer. nat. Oliver Waldhorst

Tel.: +49 721 925-1474

<mailto:oliver.waldhorst@hs-karlsruhe.de>

Sprechstunde: Dienstag, 10:00 – 11:00, Gebäude E, Raum 308



Viel Spaß bei Datenbanken und Informationssysteme 1!



Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

Kapitel 1

Einführung in Datenbanken

Vorlesung Datenbanken und Informationssysteme I
Sommersemester 2017

Prof. Dr. rer. nat. Oliver P. Waldhorst

Inhalte dieses Kapitels

- Daten und Informationen
- Konzept der Datenbanken und Datenbankmanagementsysteme
- Ablage von Daten in Datendateien und Dateisystem
- Herausforderungen bei der Datenablage

Daten und Informationen

Überall sind Daten und Informationen!

- Informationsflut, „Information Overkill“
- Für gute Entscheidungen müssen wir viele Informationen auswerten, zueinander in Beziehung stellen

Computer-gestützte **Datenbanken** helfen dabei!

Daten vs. Informationen

- **Daten** repräsentieren Fakten
- **Informationen** sind Daten im Zusammenhang

Beispiel: Die Amazon-Rechnung

amazon.de

RECHNUNG

Datum
Rechnungsnummer ist
ohne Zusammenhang
nutzlos!

Amazon Media EU S.à r.l.
5 rue Plaetis
L-2338
Luxemburg
UST-ID: LU20944528

Bestelldatum: 11.08.2016
Lieferdatum: 11.08.2016
Bestellnummer: D01-2381335-7325103

Ausgestellt für:
Oliver Waldhorst

Rechnungsdatum: 11.08.2016

Rechnungsnummer: AMEU-INV-DE-2016-28935484

Menge	Beschreibung	Stückpreis (ohne USt.)	USt. %	Stückpreis (inkl. USt.)	Gesamtpreis (inkl. USt.)
1	Datenbanken: Grundlagen und Design (mitp Professional)	21,84 €	19%	25,99 €	25,99 €
GESAMT:					25,99 €

USt Detail:

Zwischensumme (ohne USt.)	USt. %	Umsatzsteuer betrag
21,84 €	19%	4,15 €
GESAMT:		4,15 €

Beispiel: Die Amazon-Rechnung

Für Service-Mitarbeiter ohne Computer-Zugang sagt „AMEU-INV-DE-2016-28935484“ nichts aus

- Kann noch nicht einmal sagen, ob Rechnung existiert!

Für Service-Mitarbeiter mit Computer-Zugang (und Datenbank im Hintergrund) ist z.B. ersichtlich, dass

- *Datenbanken: Grundlagen und Design* bestellt wurde
- Rechnungsbetrag 25,99 € ist
- Vom selben Kunden Bücher über Rechnernetze gekauft wurden (→ Angebote, Werbung)
- Vom selben Kunden andere Rechnungen häufig verspätet gezahlt wurden (→ Anpassung Zahlungsmodalitäten auf Vorkasse)

! Nennen Sie weitere Informationen, die evtl. abgeleitet werden können!

Zusammengefasst

- Informationen setzen sich aus Daten zusammen
- Durch Datenverarbeitung werden aus Daten Informationen
- Informationsgehalt von Daten hängt vom Zusammenhang ab
- Gute, zeitnah vorliegende Informationen helfen gute Entscheidungen zu treffen

Datenmanagement

Daten sind Ausgangspunkt aller weiteren Aktivitäten und müssen sorgfältig behandelt werden

- Datenfehler führen zu fehlerhaften Informationen und so letztendlich zu falschen Entscheidungen

Aufgaben des **Datenmanagements**:

- Erzeugung von Daten
- Speicherung von Daten
- Wiedergabe von Daten

Datenmanagement und Datenbanken

Datenmanagement ist kein Erfindung der IT-Zeitalters

- Früher wurden Daten in Stein geritzt
- Heute noch werden Aktenschränke meterweise gefüllt

Im modernen Datenmanagement spielen **Datenbanken** eine zentrale Rolle

Datenbank – Definitionen [Geisler 2014]

- Technische Definition:
Eine **Datenbank** ist ein verteiltes, integriertes Computersystem, dass Nutzdaten und Metadaten enthält. **Nutzdaten** sind Daten, die Benutzer in der Datenbank ablegen und aus denen Informationen gewonnen werden. **Metadaten** werden oft auch als Daten über Daten bezeichnet und helfen, die Nutzdaten in der Datenbank zu strukturieren.
- Theoretische Definition:
Eine **Datenbank** ist eine geordnete, selbstbeschreibende Sammlung von Daten, die miteinander in Beziehung stehen.

Beispiel Mitarbeiter- und Abteilungsdatenbank

Personalnr	Nachname	Vorname	Funktion	Abteilung
1	Zeldberg	Günter	Geschäftsführer	1
	alter	Benno	Abteilungsleiter	4
		Fritz	Programmierer	3
		Hans	Chefp	
5	Nieda	Frieda	Chefsekret	1
6	Gründlich	Gerda	Sachbearbeiterin	4
7	Klugscher	Karl	Azubi	2
8	Lueger	Ludwig	Abteilungsleiter	
801	Tutnix	Toni	Praktikant	
...

Selbstbeschreibung
durch Metadaten
(Bedeutung der Spalte)

Instanzen des Objekts
Mitarbeiter (Entität)

Beziehung „gehört zu
Abteilung“

Beziehung
„Abteilungsleiter ist“

Abteilungsnummer	Abteilungsname	Abteilungsleiter
	Geschäftsführung	1
	Vertrieb	8
3	Programmierung	4
4	Verwaltung	2

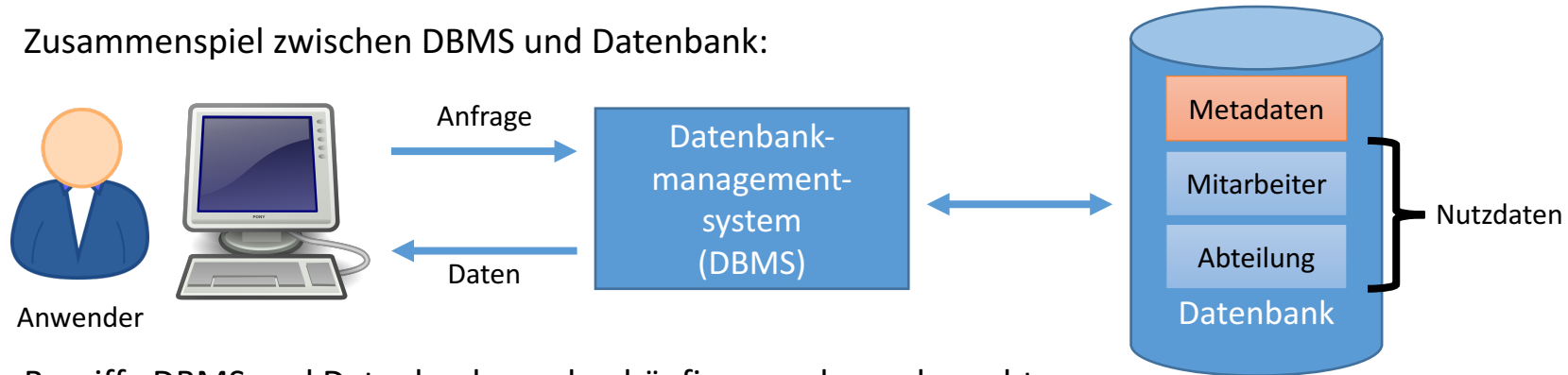
Datenbankmanagementsystem (DBMS)

DBMS übernimmt die Verwaltung der Daten in einer Datenbank

- Datenorganisation
- Zugriffsregelung

DBMS kann aus einem Programm auf einem Rechner oder vielen verteilten Programmen auf unterschiedlichen Rechnern (z.B. Servern bestehen)

Zusammenspiel zwischen DBMS und Datenbank:



Quelle: [Geisler 2014]

Begriffe DBMS und Datenbank werden häufig unsauber gebraucht

- Z.B. ist Oracle keine Datenbank, sondern ein DBMS, das Datenbanken verwaltet

Ablage von Daten - Dateisystem

Im Dateisystem können Daten hierarchisch organisiert werden

- Ähnlich zur Ordnung in Hängeregistratur
 - Schrank für „Rechnungen“, „Schriftverkehr“, ...
 - Ordner im Schrank für Rechnungen z.B. geordnet nach Kundennummer
 - Was ist, wenn wir Rechnungen suchen, die von einem bestimmten Sachbearbeiter erstellt wurden?

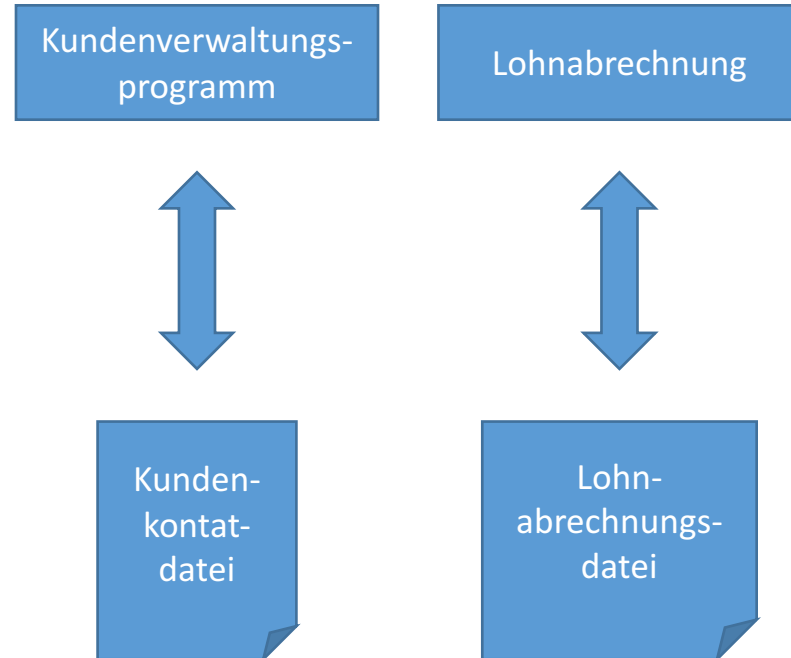
Dateisysteme arbeiten nach diesem Prinzip (mit größerer Schachtelungstiefe)

Ablage von Daten – Organisation von Datendateien

Historisch hat häufig jedes Programm eigene Dateien für Daten abgelegt

Probleme:

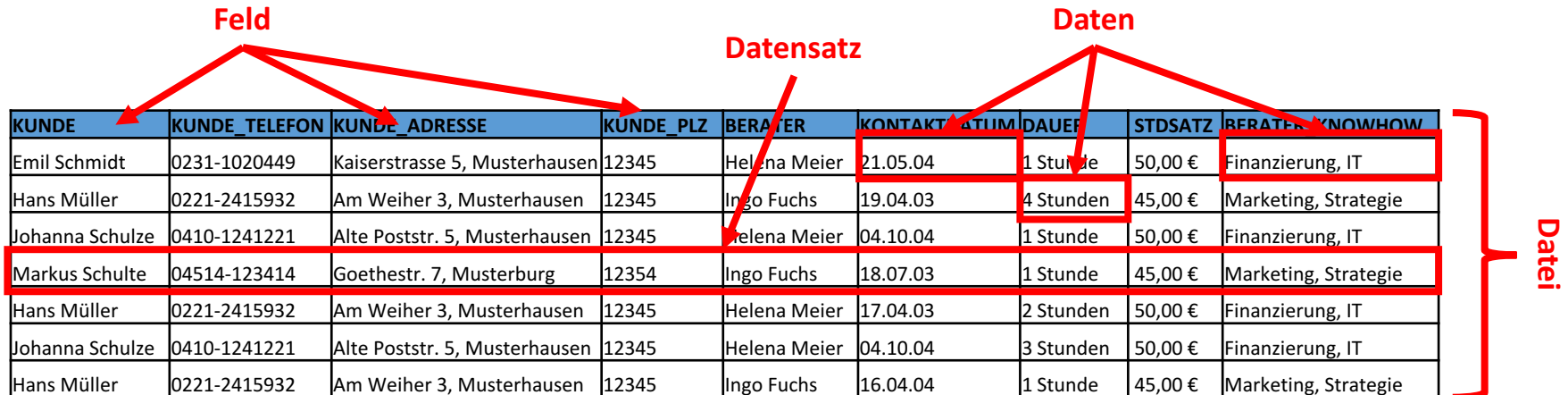
- Daten können doppelt vorkommen
 - Bsp.: Beratungsfirma ordnet Berater einem Kunden zu (Beratername in Kundendatei) und verwaltet Löhne der Berater (Beratername in Lohndatei)
- Datenverknüpfung schwierig
 - Beispiel: Lohn des Beraters hängt von geleisteten Stunden in Kundenprojekten ab



Quelle: [Geisler 2014]

Logischer Aufbau von Datendateien

- **Daten:** Alle Fakten, die in der Datei gespeichert sind
- **Feld:** Benannte Einheit, die immer Daten des selben Typs aufnimmt
- **Datensatz:** Verknüpfte Felder zu einem Objekt des täglichen Lebens (hier: Kundenkontakt)
- **Datei:** Menge von zusammengehörigen Datensätzen



KUNDE	KUNDE_TELEFON	KUNDE_ADRESSE	KUNDE_PLZ	BERATER	KONTAKT_DATUM	DAUER	STDSATZ	BERATER_KNOWLEDGE
Emil Schmidt	0231-1020449	Kaiserstrasse 5, Musterhausen	12345	Helena Meier	21.05.04	1 Stunde	50,00 €	Finanzierung, IT
Hans Müller	0221-2415932	Am Weiher 3, Musterhausen	12345	Ingo Fuchs	19.04.03	4 Stunden	45,00 €	Marketing, Strategie
Johanna Schulze	0410-1241221	Alte Poststr. 5, Musterhausen	12345	Helena Meier	04.10.04	1 Stunde	50,00 €	Finanzierung, IT
Markus Schulte	04514-123414	Goethestr. 7, Musterburg	12354	Ingo Fuchs	18.07.03	1 Stunde	45,00 €	Marketing, Strategie
Hans Müller	0221-2415932	Am Weiher 3, Musterhausen	12345	Helena Meier	17.04.03	2 Stunden	50,00 €	Finanzierung, IT
Johanna Schulze	0410-1241221	Alte Poststr. 5, Musterhausen	12345	Helena Meier	04.10.04	3 Stunden	50,00 €	Finanzierung, IT
Hans Müller	0221-2415932	Am Weiher 3, Musterhausen	12345	Ingo Fuchs	16.04.04	1 Stunde	45,00 €	Marketing, Strategie

Physischer Aufbau von Datendateien

Flache Datei: Besitzt nur ein Minimum an Struktur

- Häufig im **Comma Separated Values (CSV)** Format gespeichert
 - Felder durch Trennzeichen (hier: Semikolon ;) getrennt
 - Jeder Datensatz in neuer Zeile
 - Keine Informationen über Bedeutung der Felder, Datentypen, ...

```
Emil Schmidt;0231-1020449;Kaiserstrasse 5, Musterhausen;12345;Helena Meier; ...  
Hans Müller;0221-2415932;Am Weiher 3, Musterhausen;12345;Ingo Fuchs; ...  
Johanna Schulze;0410-1241221;Alte Poststr. 5, Musterhausen;12345;Helena Meier; ...  
Markus Schulte;04514-123414;Goethestr. 7, Musterburg;12354;Ingo Fuchs; ...  
Hans Müller;0221-2415932;Am Weiher 3, Musterhausen;12345;Helena Meier; ...  
Johanna Schulze;0410-1241221;Alte Poststr. 5, Musterhausen;12345;Helena Meier; ...  
Hans Müller;0221-2415932;Am Weiher 3, Musterhausen;12345;Ingo Fuchs; ...
```

Quelle: [Geisler 2014]

Verwaltung von Datendateien

Jedes Programm, dass mit flacher Datei umgeht (z.B. Lohnabrechnungsprogramm) muss **Dateiverwaltung** für folgende Aufgaben besitzen:

- Datei anlegen
- Daten zur Datei hinzufügen
- Daten aus Datei löschen
- Daten in der Datei ändern
- Daten aus der Datei laden
- ...

Anfragen von Datendateien

Dateiverwaltung hängt von Datei-Struktur ab

- Problematisch für Daten-Anfragen: Jede Anfrage muss speziell für die Datei geschrieben werden
- „Ad-hoc Anfragen“ sind quasi nicht möglich
- Damit stehen Informationen nicht zeitnah zur Verfügung, was Entscheidungsqualität beeinflussen kann

Änderung der Struktur von Datendateien

Dateiverwaltung hängt von Dateistruktur ab

- Problematisch bei Strukturänderungen
 - z.B. Auftrennung von KUNDE_ADRESSE in KUNDE_STRASSE und KUNDE_ORT
 - Erfordert Konvertierungsprogramm zur Strukturumstellung (machbar)
 - Erfordert Umstellung der Dateiverwaltung aller Programme, die auf die Daten zugreifen (kann sehr komplex sein)

KUNDE	KUNDE_TELEFON	KUNDE_ADRESSE	KUNDE_PLZ	BERATER	KONTAKTDATUM	DAUER	STDSATZ	BERATER_KNOWLEDGE
Emil Schmidt	0231-1020449	Kaiserstrasse 5, Musterhausen	12345	Helena Meier	21.05.04	1 Stunde	50,00 €	Finanzierung, IT
Hans Müller	0221-2415932	Am Weiher 3, Musterhausen	12345	Ingo Fuchs	19.04.03	4 Stunden	45,00 €	Marketing, Strategie
Johanna Schulze	0410-1241221	Alte Poststr. 5, Musterhausen	12345	Helena Meier	04.10.04	1 Stunde	50,00 €	Finanzierung, IT
Markus Schulte	04514-123414	Goethestr. 7, Musterburg	12354	Ingo Fuchs	18.07.03	1 Stunde	45,00 €	Marketing, Strategie
Hans Müller	0221-2415932	Am Weiher 3, Musterhausen	12345	Helena Meier	17.04.03	2 Stunden	50,00 €	Finanzierung, IT
Johanna Schulze	0410-1241221	Alte Poststr. 5, Musterhausen	12345	Helena Meier	04.10.04	3 Stunden	50,00 €	Finanzierung, IT
Hans Müller	0221-2415932	Am Weiher 3, Musterhausen	12345	Ingo Fuchs	16.04.04	1 Stunde	45,00 €	Marketing, Strategie

Quelle: [Geisler 2014]

Programme, die von der Struktur der Daten abhängen heißen **strukturell abhängig**

Flachen Dateien vs. Datentypen

Häufig unterscheidet sich **logische Darstellung** (Format, das Menschen verstehen) von **physischer Darstellung** (Format, in dem Daten maschinell gespeichert sind)

Dateiverwaltung hängt von physischer Darstellung der Daten ab

- Z.B. Darstellung von Zahl als Integer, Festkomma-Darstellung oder Exponent/Mantisse
- Änderung der physischen Darstellung erfordert Änderung der Dateiverwaltung aller Programme, die eine Datei benutzen

Programme, die von der physischen Darstellung eine bestimmten Feldes abhängen, heißen **datenabhängig**

Sicherheit von Datendateien

Text-Charakter von flachen Dateien ermöglicht einfaches aushebeln von Sicherheitsmaßnahmen

- Z.B. Öffnen und Verändern in Texteditor

Dateicharakter und fehlender Zugriffsschutz ermöglicht auch Aufbau von vielen kleinen Datenbestände („Dateninseln“, „Informationsinseln“)


- Verschiedene inkonsistente Versionen derselben Datei

Datenredundanzen und Anomalien

Dateninkonsistenzen können auch in einer einzelnen Datei auftreten, wenn Datenredundanzen vorliegen

- **Datenredundanzen:** Dieselbe Information über ein Objekt (Person, Gegenstand, ...) wird an verschiedenen Orten gehalten
- **Dateninkonsistenzen:** Es existieren verschiedene Versionen derselben Daten, die zueinander in Konflikt stehen
- Beispiel: Unvollständige Änderung der Telefonnummer einer Kundin

Nennen sie andere Möglichkeiten für Inkonsistenzen!



KUNDE	KUNDE_TELEFON	KUNDE_ADRESSE	KUNDE_PLZ	BERATER	KONTAKTDATUM	DAUER	STDSATZ	BERATER_KNOWHOW
Emil Schmidt	0231-1020449	Kaiserstrasse 5, Musterhausen	12345	Helena Meier	21.05.04	1 Stunde	50,00 €	Finanzierung, IT
Hans Müller	0221-2415932	Am Weiher 3, Musterhausen	12345	Ingo Fuchs	19.04.03	4 Stunden	45,00 €	Marketing, Strategie
Johanna Schulze	0410-1241335	Am Weiher 3, Musterhausen	12345	Helena Meier	04.10.04	1 Stunde	50,00 €	Finanzierung, IT
Markus Schulte	04514-123414	Goethestr. 7, Musterburg	12354	Ingo Fuchs	18.07.03	1 Stunde	45,00 €	Marketing, Strategie
Hans Müller	0221-2415932	Am Weiher 3, Musterhausen	12345	Helena Meier	17.04.03	2 Stunden	50,00 €	Finanzierung, IT
Johanna Schulze	0410-1241221	Am Weiher 3, Musterhausen	12345	Helena Meier	04.10.04	3 Stunden	50,00 €	Finanzierung, IT
Hans Müller	0221-2415932	Am Weiher 3, Musterhausen	12345	Ingo Fuchs	16.04.04	1 Stunde	45,00 €	Marketing, Strategie

Dateninkonsistenz

Quelle: [Geisler 2014]

Arten von Anomalien

Änderungs-Anomalie (Update-Anomalie)

- Daten werden nicht an jeder Stelle geändert, an der sie gespeichert werden (Bsp. siehe oben)

Einfüge-Anomalie (Insert-Anomalie)

- Daten können nicht erfasst werden, ohne andere Daten gleichzeitig erfassen zu müssen

! Beispiel?

Lösch-Anomalie (Delete-Anomalie)

- Löschen bestimmter Daten löscht auch andere Daten, die eigentlich erhalten bleiben sollten

! Beispiel?

Zusammenfassung

Informationen sind heutzutage unverzichtbar!

- Informationen = Daten im Zusammenhang
- Gute Entscheidungen basieren auf guten, zeitnah vorliegenden Informationen
- Hierarchische abgelegte oder flache Dateien helfen dabei nicht
- Datenbankmanagementsystem und gutes Datenbank-Design können helfen (und sind Gegenstand dieser Vorlesung)

Weiterer Fahrplan:

- Übersicht über Datenbankmanagementsysteme und –anwendungen
- Das relationale Datenbankmodell
- Datenbanknormalisierung
- Relationen-Algebra
- Structured Query Language SQL
- Datenbankmodellierung



Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

Kapitel 2

Datenbanksysteme und Datenbankanwendungen

Vorlesung Datenbanken und Informationssysteme I
Sommersemester 2017

Prof. Dr. rer. nat. Oliver P. Waldhorst



Inhalte dieses Kapitels

- Datenbank Anwendungen: Komponenten und Strukturmodelle
- Funktionen von Datenbankmanagementsystemen (DBMS)
- Middleware
- Organisationsebenen in einem DBMS
- Logische Datenmodelle
 - Hierarchisches Modell
 - Netzwerkmodell
 - Relationales Modell
 - Entity-Relationship-Modell und –Diagramme
 - Objektorientiertes Modell, NOSQL

Motivation

Informationen sind Daten im Zusammenhang

- Gute Entscheidungen basieren auf guten, zeitnah vorliegenden Informationen
- Die Datenablage im Dateisystem und in flachen Dateien erfüllt diese Anforderungen nicht!

Datenbanksysteme helfen, Daten und Zusammenhänge zu verwalten

Definition Datenbankanwendung

Zur Wiederholung:

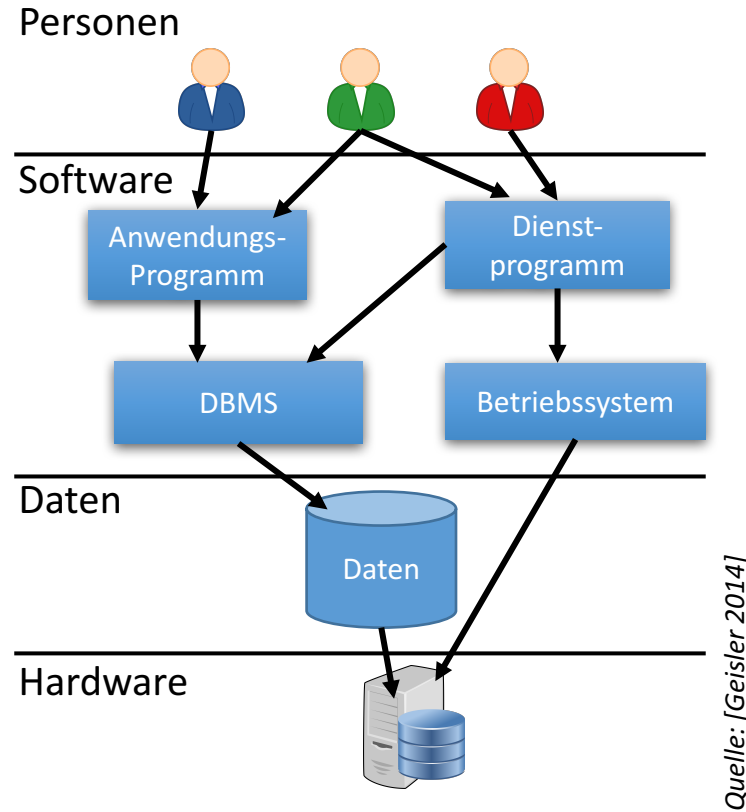
Datenbankmanagementsystem (DBMS):

- Verwaltet Daten in einer Datenbank
- regelt Organisation der Daten und Datenzugriff

Datenbankanwendung (Datenbanksystem oder Anwendungssystem mit Datenbank):

- Komplettes, zur Datenverarbeitung genutztes System
- besteht aus DBMS und zahlreichen anderen Komponenten (weitere Software, Hardware, Anwender, ...)

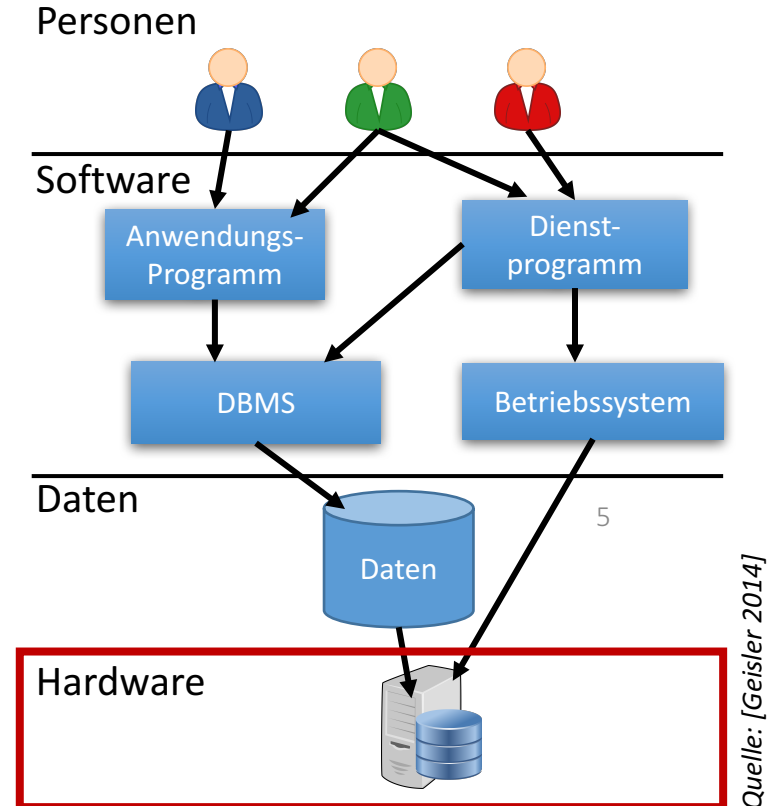
Komponenten einer Datenbankanwendung



Datenbankanwendung: Hardware

Alle physischen Geräte

- Datenbankserver
- Clientrechner
- Vernetzung: Kabel, Hubs, Switches, Firewalls etc. zur Verbindung der Client-Rechner
- Drucker, Magnetbänder, CD-/DVD-Roms
- ...

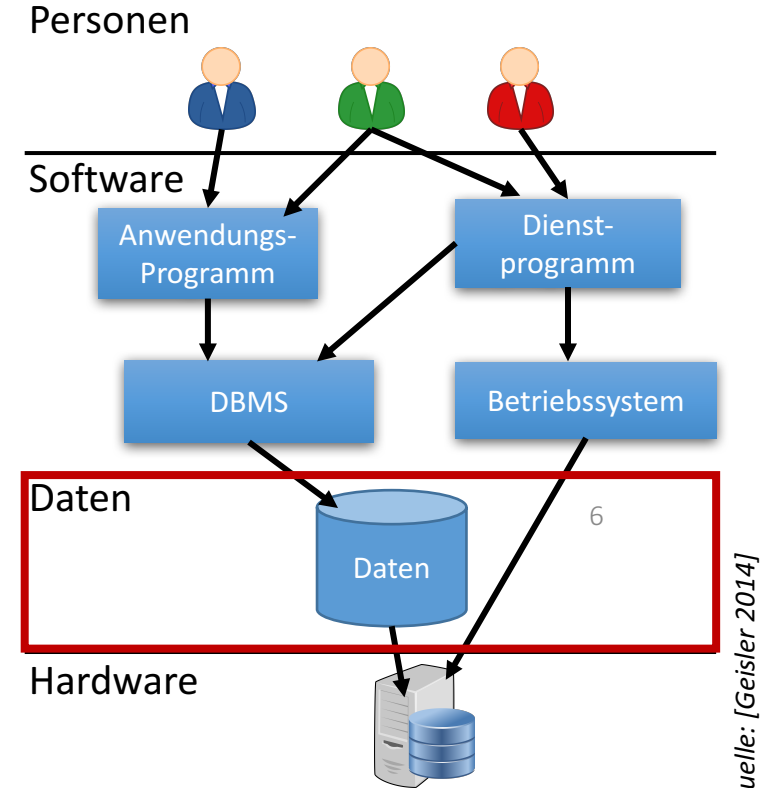


Quelle: [Geisler 2014]

Datenbankanwendung: Daten

Im Datenbanksystem
gespeicherte Fakten

- „Rohstoff“ für Informationen
- Schutz der Daten ist entscheidend
→ keine andere Komponente außer DBMS besitzt Zugriff auf die Daten



Quelle: [Geisler 2014]

Datenbankanwendung: Software

Datenbankmanagement-System (DBMS)

- Verwaltung der Daten (Speicherung, Wiederbeschaffung) auf komfortable Weise

Anwendungsprogramm(e)

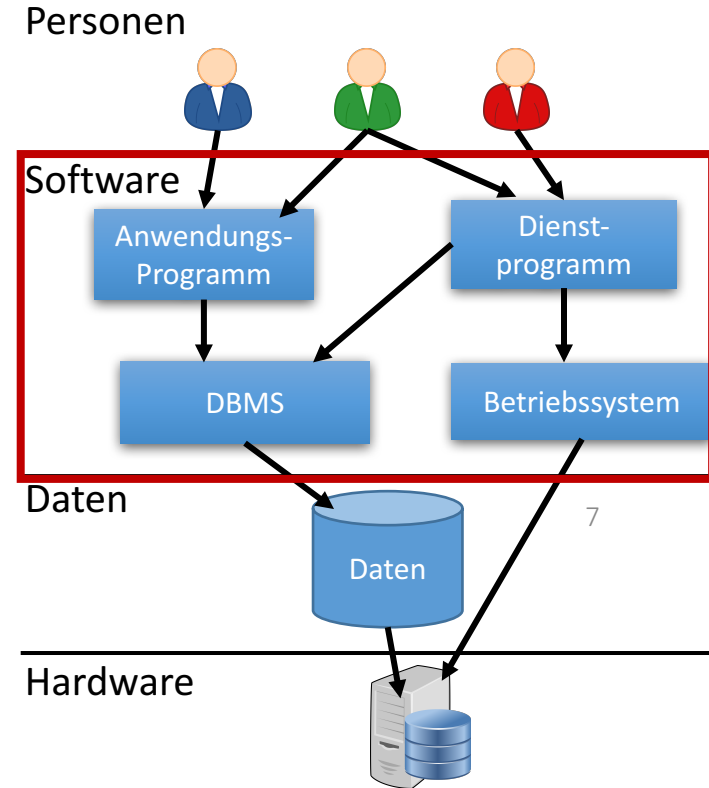
- Primäre Schnittstelle für Nutzer
- PC-Anwendung, Web-Anwendung in Internet-Browser, App, ...

Dienstprogramme (Tools, Utilities)

- Unterstützen Wartungsarbeiten (Datensicherung, Überprüfung, ...)
- I.d.R. von Administratoren, Designern, Programmierern verwendet

Betriebssystem(e)

- U.u. unterschiedlich auf Server und Clients (z.B. Unix-Oracle-Server und iPhone-App)

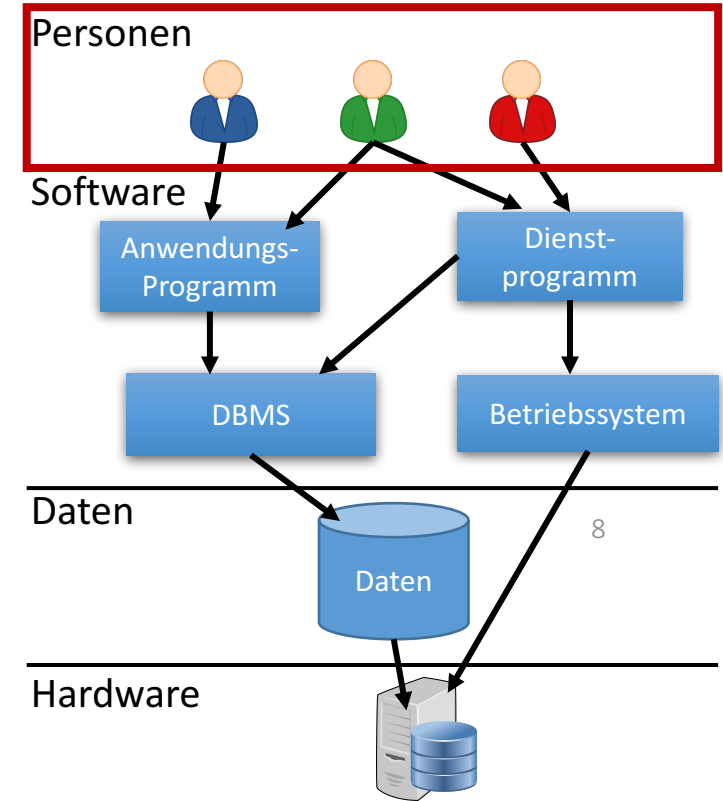


Quelle: [Geisler 2014]

Datenbankanwendung: Personen

Kann in funktionale Gruppen unterteilt werden

- Anwender: Nutzen die Daten über Anwendungsprogramm
- Administratoren: Führen Verwaltungsfunktionen aus (Anlegen von Datenbanken, Benutzern, Speicherverwaltung)
- Designer und Programmierer: Legen Datenbankstruktur fest und ermöglichen komfortablen Zugriff
- Gruppen spiegeln sich meist in Berechtigungsgruppen oder Rollen

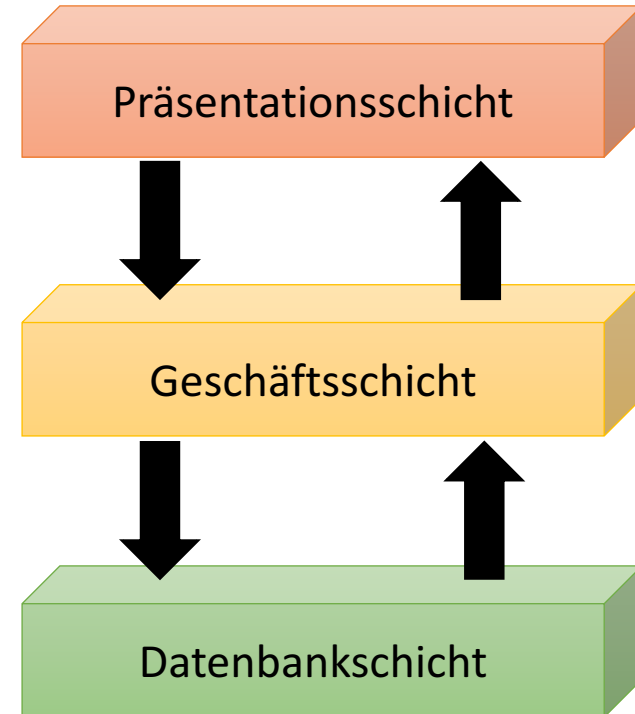


Quelle: [Geisler 2014]

Strukturmodelle für Datenbankanwendungen

Datenbankanwendung kann in drei logische Schichten aufgeteilt werden

- **Präsentationsschicht**
 - Darstellung und Eingabe von Daten
 - Z.B. Benutzer-Interfaces, Berichte
- **Geschäftsschicht**
 - Auch als **Business-Logik** bezeichnet
 - Regeln für Datenbank, z.B. Überprüfung von Wertebereichen, Gültigkeiten
- **Datenbankschicht**
 - Speicherung, Suche, Integrität
 - Wird von DBMS vollständig implementiert

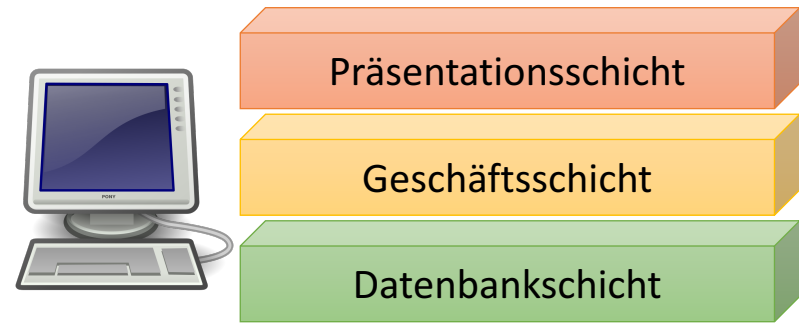


Quelle: [Geisler 2014]

Einschichtige Datenbankanwendungen

Alle Schichten auf einem Computer realisiert

- Direkter Zugriff auf Datendateien auf lokaler Festplatte
- Häufig verwendet mit Desktop-Datenbanken



Quelle: [Geisler 2014]

Zweischichtige Datenbankanwendungen

Client-Server-Architektur

- Zentraler **Server**
- Zugriff von **Clients** über Netzwerk
- Server koordiniert Zugriffe mehrerer Clients und aktualisiert Daten

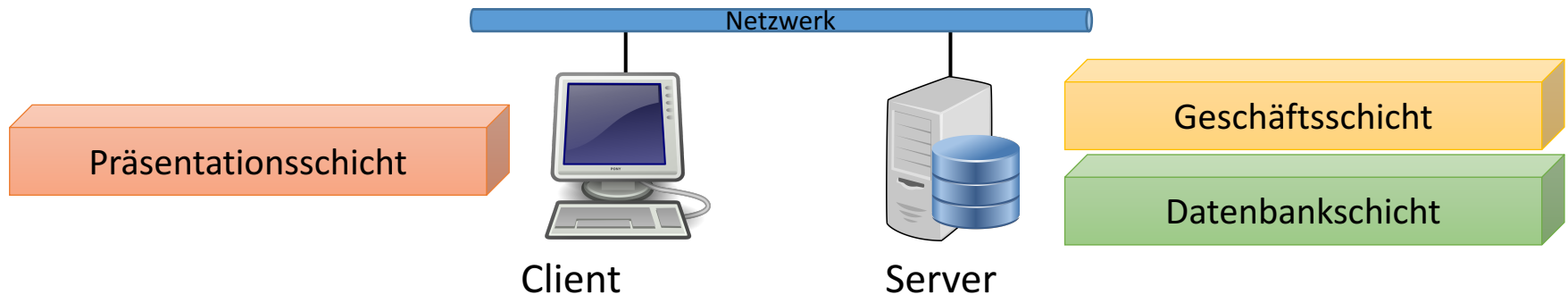
Ausprägungen nach Verortung der Geschäftsschicht:

- **Intelligenter Server / Thin Client**
- **Intelligenter Client / Fat Client**

Intelligenter Server / Thin Client

Server implementiert die Geschäftsschicht

- Umfangreiche Berechnungen direkt auf dem Server möglich
 - Nachteil: Server kann zum Engpass werden
- Daten müssen für jeden Verarbeitungsschritt zwischen Client und Server übertragen werden

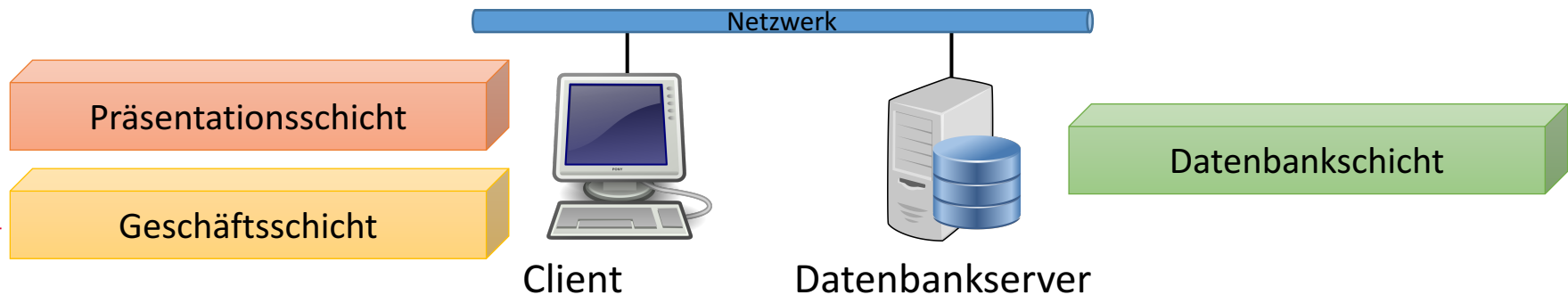


Intelligenter Client / Fat Client

Client implementiert die Geschäftsschicht

- Umfangreiche Berechnungen direkt auf Client
- Ggf. Konsistenzprobleme

In Praxis meist keine strikte Trennung der Modelle



N-Schichtige (n-tier) Datenbankanwendungen

Schichten (meist drei) strikt getrennt auf verschiedenen Rechnern ausgeführt

- Client: Präsentationsschicht
- Anwendungsserver: Geschäftsschicht
- Datenbankserver: Datenbankschicht

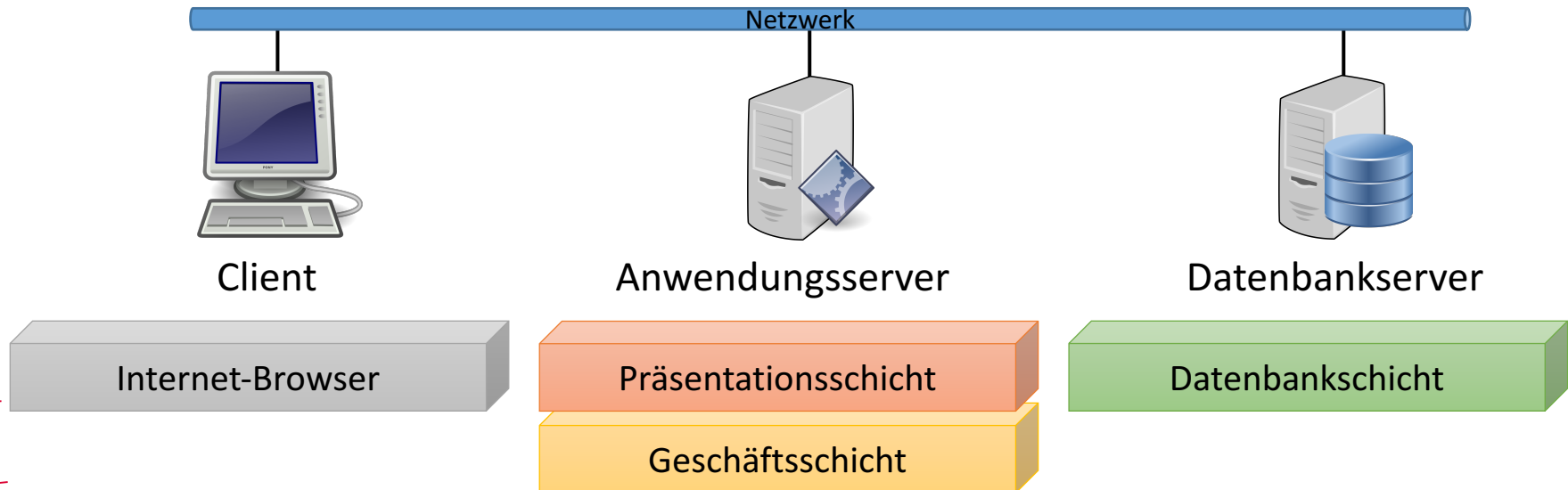


Quelle: [Geisler 2014]

Internet-Datenbankanwendung

Spezialfall der n-schichtigen Datenbankanwendung

- Lediglich Internet-Browser läuft auf Client
- Keine Software-Installation notwendig



Klassifizierung von Datenbank Anwendungen

Einbenutzer-Datenbank vs. Mehrbenutzer-Datenbank

- Können mehrere Benutzer gleichzeitig (!) mit der Datenbank arbeiten?

Desktop-Datenbank vs. Server-Datenbank

- Wo ist die Datenbank gespeichert?

Desktop-Datenbank vs. Workgroup-/Abteilungsdatenbank vs. Unternehmens-/Enterprise-Datenbank

- Wie viele Benutzer greifen auf die Datenbank zu (<10, 10-50, >50)?

Zentrales/monolithisches Datenbanksystem vs. verteiltes Datenbanksystem

- Auf wie vielen Rechnern ist die Datenbank gespeichert (zentraler Server vs. viele Rechner)?

Transaktionales Datenbanksystem/Produktivsystem vs. Decision Support System/Data Warehouse

- Wird die Datenbank direkt in einer Anwendung eingesetzt oder unterstützt sie bei Bedarf die Entscheidungsfindung?

DBMS-Funktionen (nach [Geisler, 2014])

Ein DBMS bietet in der Regel folgende Funktionalität:

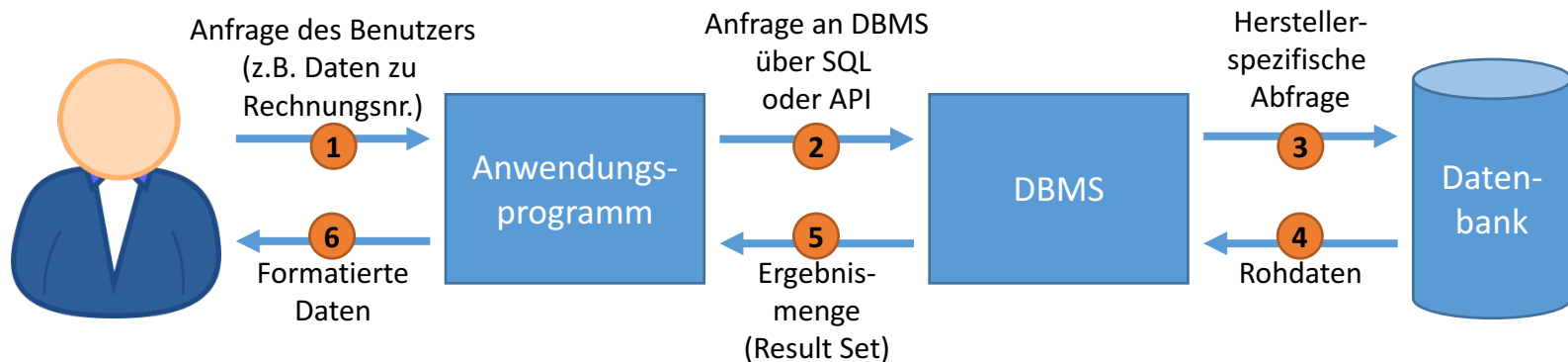
- Datenbankzugriff
- Datenumwandlung / Präsentation
- Wahrung der Datenintegrität
- Metadatenverwaltung
- Bereitstellung von Kommunikationswegen
- Datenträgerverwaltung
- Verwaltung der Sicherheit
- Anwenderverwaltung
- Datensicherung / Wiederherstellung

DBMS-Funktionen

Datenbankzugriff

- **Datenmanipulationssprachen** (DML) ermöglicht einfachen Weg für Zugriff auf gespeicherte Daten
 - Als Standardsprache hat sich ANSI-Standard **SQL** („**Structured Query Language**“) etabliert
 - Alle Kommunikation mit DBMS findet i.d.R. mit SQL statt
 - Einige Hersteller bieten auch zusätzlich Application Programming Interfaces (APIs)

Ablauf des Zugriffs einer Anwendung auf die Datenbank:



DBMS-Funktionen

Datenumwandlung/Präsentationen

- Trennt zwischen Eingabe- und Speicherformat von Daten
- Z.B. wird Datum logisch als TT.MM.JJJJ eingegeben, aber physisch als Julianisches Datum (in Tagen seit dem 1. Januar –4712 12:00 Uhr) gespeichert
- Trennung von logischen und physischen Datentypen gewährleistet
Datenunabhängigkeit
 - Anwendungsprogramm nur von logischen Daten abhängig

Wahrung der Datenintegrität

- Definition von Regeln, die **Datenintegrität** gewährleisten (z.B. referentielle Integrität durch Prüfung von Verweisen zwischen Daten)
- Regeln werden in Katalog (**Data Dictionary**) gespeichert, in dem das DBMS Metainformationen ablegt

DBMS-Funktionen

Metadatenverwaltung

- Beschreibung der gespeicherten Daten durch Metadaten
- I.d.R. im Katalog (Data Dictionary), auch in Tabellen gespeichert
- Können gelesen, aber nicht direkt sondern nur über spezielle Funktionen verändert werden

Bereitstellung von Kommunikationswegen

- Möglichkeiten für Client-Rechner zur Kommunikation mit dem Datenbank-Server
- Häufig Kommunikationsprotokolle, die auf Netzwerk-Protokolle (z.B. TCP/IP) aufsetzen
- Middleware für Abstraktion des Anwendungszugriff auf Datenbank (Unabhängigkeit von speziellem Datenbankmanagementsystem, s.u.)

DBMS-Funktionen

Datenträgerverwaltung

- Effiziente und performante Speicherung von Daten durch Aufbau von logischen Strukturen
- Anwender muss sich nicht um Datenablage kümmern

Verwaltung der Sicherheit

- Beschränkung des Zugriffs auf Datenobjekte für bestimmte Benutzergruppen (z.B. für eine Tabelle nur Lesen vs. Einfügen, Verändern, Löschen)

DBMS-Funktionen

Verwaltung mehrerer Anwender

- Verwaltung von konkurrierenden Zugriffen auf die selben Datensätze
- Vermeidung von Race-Conditions („Wer zuletzt speichert, gewinnt“) durch sperren von Datensätzen, z.B. beim ersten Zugriff

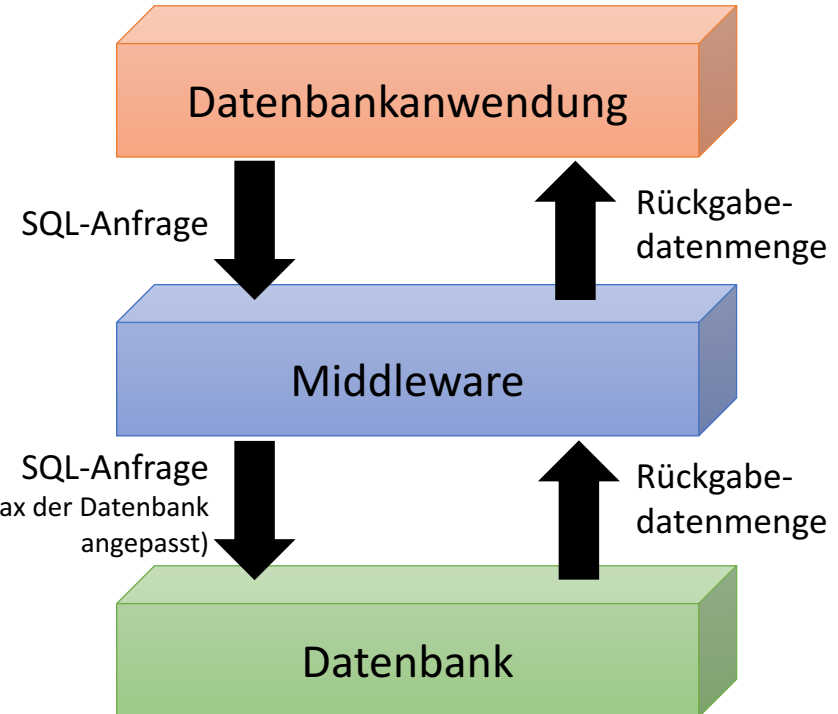
Datensicherung/Wiederherstellung

- Mögliche Gefahren sind Hardware-Ausfälle und Benutzerfehler (z.B. unbeabsichtigtes löschen)
- Unterstützt durch Dienstprogramme für Administratoren

Middleware

Zwischenschicht für Kopplung zwischen Anwendungsprogramm und Datenbank

- Datenbank- und DBMS-unabhängige Schnittstelle für Programmierung
- Änderungen (anderes DBMS, anderer Speicherort für Daten, ...) für Programmierer transparent (ggf. an Syntax der Datenbank angepasst)
- Beispiele:
 - Open Database Connectivity (ODBC)
 - Java Database Connectivity (JDBC)
 - ...



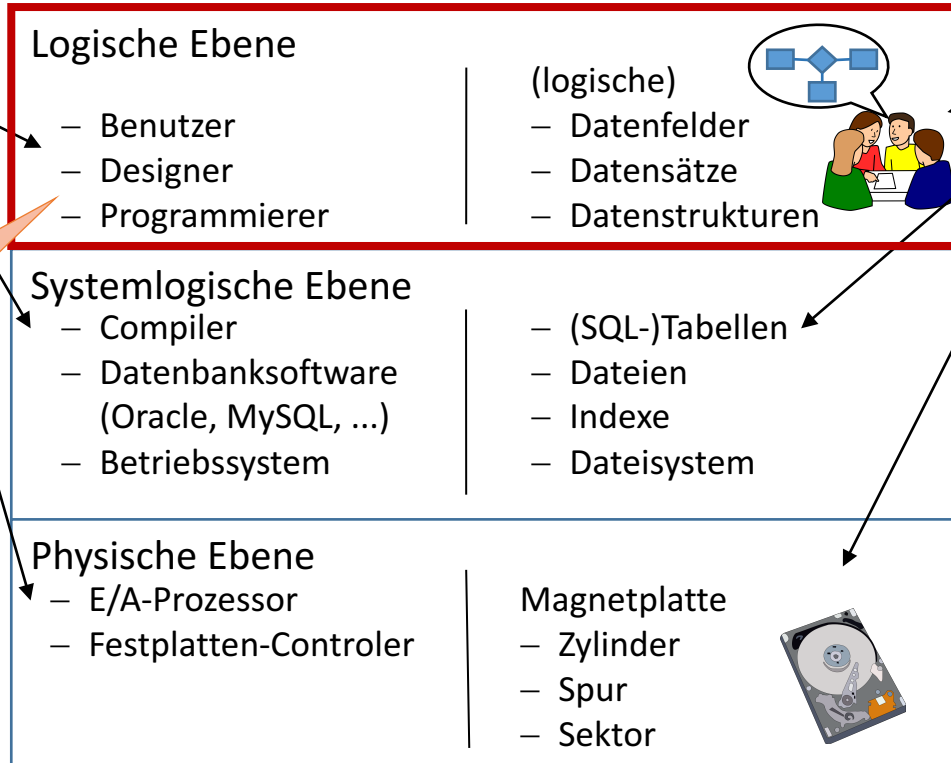
Quelle: [Geisler 2014]

Organisationsebenen in einem DBMS

Aktive Komponenten zur
Datenverwaltung

Datenorganisationseinheiten
(Beispiele)

Die Vorlesung
beschäftigt sich
(hauptsächlich)
mit der logischen
Ebene



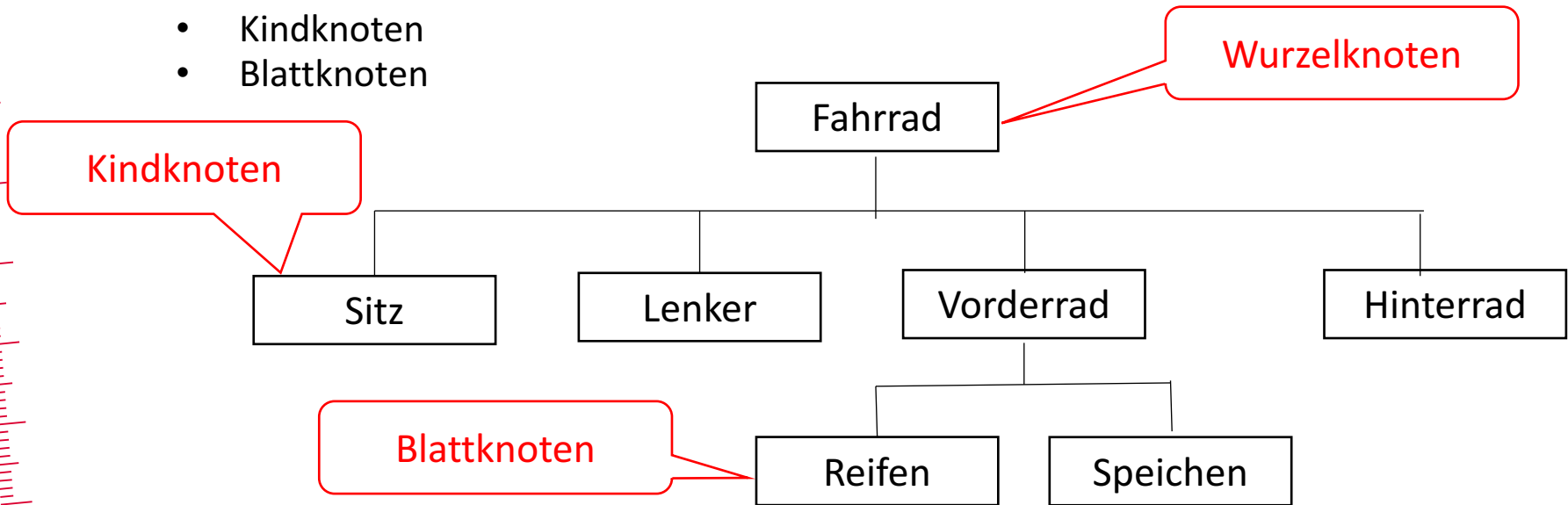
Evolution der logischen Datenmodelle

Zeitraum	Datenmodell	Beispiele
1960-1970	Dateisysteme	VMS/VSAM
1970er Jahre	Hierarchische und Netzwerk Datenmodell	IMS, ADABAS
Mitte der 1970er	Relationale DB	DB2, Oracle, MS SQL, MySQL
Bis 1980er	OO DB oder Objektrelationale DBs	DB2 UDB, Oracle 11g
Bis 1990er	XML, Hybride DB (Objekt Frontend+ Rel.DB)	dbXML, Tamino, Oracle 11g, MS SQL,
Ende 2000 -jetzt	Verteilte, hochskalierbare DBs	SimpleDB, BigTable, Cassandra

Quelle: Vorlesungsfolien Prof. I. Stengel

Das Hierarchische Modell

- Idee aus Apollo-Programm: Größere Fertigungseinheiten durch kleinere aufbauen
- Strukturiert hierarchisch mit unterschiedlichen Knoten
 - Wurzelknoten
 - Kindknoten
 - Blattknoten



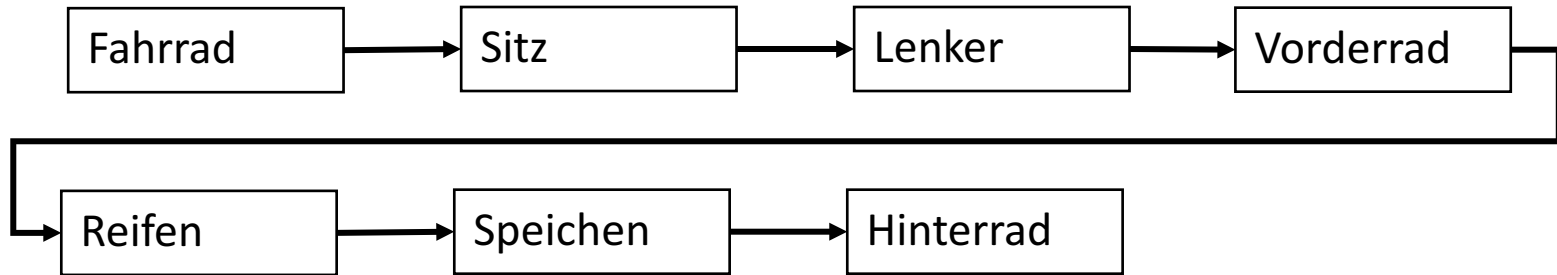
! Wie kann hier der Reifen gefunden werden?

Quelle: Vorlesungsfolien Prof. I. Stengel

Das hierarchische Modell

Speicherung einer hierarchischen Struktur nur sequenziell möglich, da der Computer einen linearen Arbeitsraum besitzt

- Traversieren eines Baumes in Pre-Order: zuerst links dann rechts, wenn es nicht weiter geht dann wird eine Stufe höher gesprungen
- Baum in einer linearen Form dargestellt:



! Wie kann jetzt der Reifen gefunden werden?

Vor-/Nachteile hierarchisches Modell

Vorteile

- Datenintegrität wird erzwungen – jedes Kind hat genau einen Elternknoten
- Eignet sich gut für 1:N Beziehungen
 - Bsp.: Ein Fahrrad besteht aus vier Teilen (Sitz, Lenker, Vorderrad, Hinterrad)
- Einfache Suche, wenn Struktur bekannt (man muss durch die Daten navigieren)

Nachteile

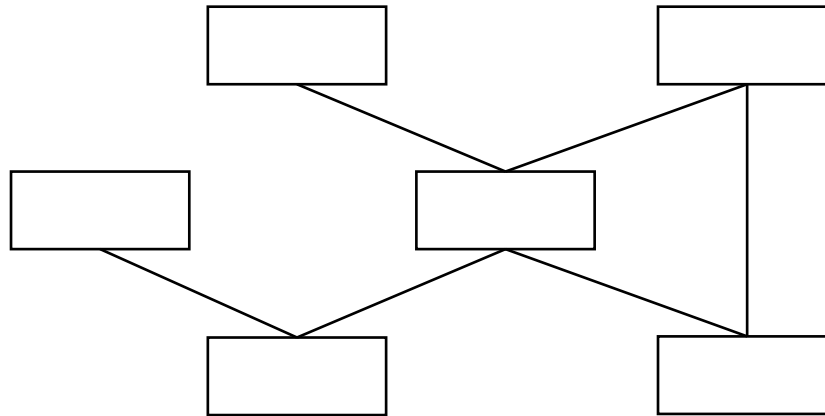
- Keine strukturelle Unabhängigkeit → Struktur darf sich nicht verändern
- Ineffiziente Suche, wenn Struktur unbekannt (Traversieren des Baumes)
- Begünstigt Delete-Anomalie bei löschen eines Unterbaums
- Keine Darstellung von N:M Beziehungen
- Komplexe Implementierung
- Fehlende Standards

! Übungsaufgabe

Modellieren sie die Hochschule Karlsruhe mit Ihren Fakultäten und ihren Professoren mit Hilfe des hierarchischen Modells!

Das Netzwerkmodell

- Adressiert Nachteile des hierarchischen Modells. (N-zu-M Beziehung nicht darstellbar)
- Ähnlich zu hierarchischen DB – Unterschied ein Kindknoten kann mehrere Elternknoten haben



Quelle: Wikipedia

- Hat die Vorteile des hierarchischen Modells übernommen
- Integrität wird verbessert
- Nachteil: einige Schwächen der hierarchischen DB (auch hier muss man durch die Daten navigieren, ...)

Quelle: Vorlesungsfolien Prof. I. Stengel

Das Relationale Modell

1970 von E.F. Codd vorgestellt

- Basiert auf dem mathematischen Konstrukt **Relation**
- Damals wegen der fehlenden Leistung der Computer als nicht umsetzbar eingestuft

Relationale DBMS (RDBMS) stellt Relation als **Tabellen** dar

- Matrix, die aus verschiedenen Zeilen/Spalten besteht

Beziehungen zwischen Relationen / Tabellen über **Schlüssel**

- Tabellen enthalten **Primärschlüssel** (Personalnummer, Abteilungsnummer)
- Verweis auf andere Tabelle durch Verwendung von Primärschlüssel als **Fremdschlüssel**
- **Beispiel: Beziehung zwischen PERSON und ABTEILUNG.**

Abteilungsnummer	Abteilungsname	Abteilungsleiter
1	Geschäftsführung	1
2	Vertrieb	8
3	Programmierung	4
4	Verwaltung	2

Personalnr	Nachname	Vorname	Funktion	Abteilung
1	Geldberg	Günter	Geschäftsführer	1
2	Buchhalter	Benno	Abteilungsleiter	4
3	Fehler	Fritz	Programmierer	3
4	Hacker	Hans	Chefprogrammierer	3
5	Nieda	Frieda	Chefsekretärin	1
6	Gründlich	Gerda	Sachbearbeiterin	4
7	Klugscher	Karl	Azubi	2
8	Lueger	Ludwig	Abteilungsleiter	2
801	Tutnix	Toni	Praktikant	3
...

Quelle: Vorlesungsfolien Prof. I. Stengel

! Übungsaufgabe

Modellieren sie die Hochschule Karlsruhe mit Ihren Fakultäten und ihren Professoren mit Hilfe des relationalen Modells



Das Entity-Relationship-Modell (ER-Modell)

Entität (Entity)

- Kann fast alles sein: eine Person, ein Ort, ein Ding, ein Event – alles worüber Daten gesammelt werden können
- Stellt ein Objekt in der realen Welt dar
- Bsp.: KUNDEN, PRODUKTE oder abstraktere Objekte wie ROUTEN

Attribut (Attribute)

- Charakterisiert eine Entität
- Beispiel: Ein Produkt hat einen Namen, Preis, etc.
- Attribute werden als Datenfelder umgesetzt

Quelle: Vorlesungsfolien Prof. I. Stengel

Elemente des ER-Datenmodells

Beziehung (Relationship)

Beschreibt eine Assoziation verschiedener Entitäten

- Bsp.: Die Beziehung zwischen einem Buch und der Bibliothek

Beziehungstypen:

- **1-zu-N Beziehung**
 - Bsp.: Ein AUTOR schreibt mehrere GEDICHTe, wobei ein GEDICHT immer nur von einem AUTOR geschrieben wird.
- **N-zu-M Beziehung**
 - Bsp.: Ein STUDENT kann mehrere MODULen belegen, wobei ein MODUL von mehreren STUDENTEN besucht werden kann
- **1-zu-1 Beziehung**
 - Bsp.: Eine PERSON hat einen einzigen PERSONALAUSWEIS, wobei ein PERSONALAUSWEIS immer nur von einer PERSON besessen wird.
- Eine Beziehung ist bidirektional

!

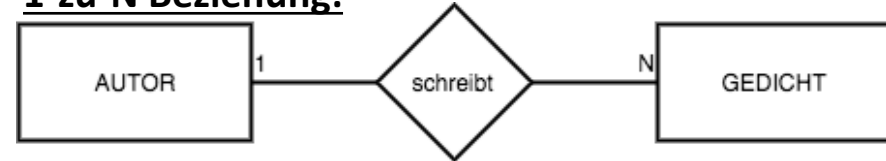
In welchem Beziehungstyp stehen BIBLIOTHEK und BUCH?

Quelle: Vorlesungsfolien Prof. I. Stengel

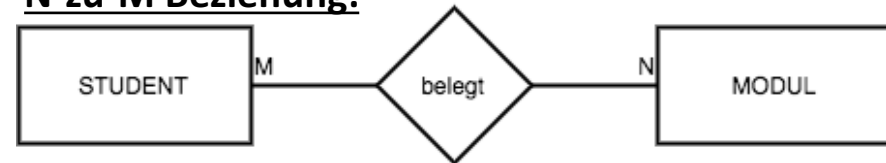
Entity Relationship Diagramme (ERD)

- Ziel: Grafische Darstellung für Übersicht, Reduktion der Komplexität
- Stellt Entitäten (Entities) und Ihre Beziehungen (Relationships) dar
- Wurde 1976 von Peter Chen eingeführt
- Folgende Beziehungen werden verwendet: 1-zu-1, 1-zu-N, M-zu-N
- Verschiedene Notationen, z.B:
 - Chen (Beispiel rechts)
 - Crow`s Foot (Krähenfuß),
 - UML (Unified Modeling Language)
- Details in Kapitel 9

1-zu-N Beziehung:



N-zu-M Beziehung:



1-zu-1 Beziehung:



Quelle: Vorlesungsfolien Prof. I. Stengel

! Übungsaufgabe

Modellieren sie die Hochschule Karlsruhe mit Ihren Fakultäten und ihren Professoren als ER-Diagramm

Objektorientiertes Modell

Anwendungsprogramm wird häufig in objektorientierte Programmiersprache geschrieben

- Nutzt Objekte mit Eigenschaften, Vererbung, Methoden

Relationale Datenbank arbeitet im Hintergrund

- Basiert Tabellen, Felder, Beziehungen, ...

Tabellen werden objektorientiert abgebildet

- Viel Code notwendig
- Methoden existieren nur in der objektorientierten Welt

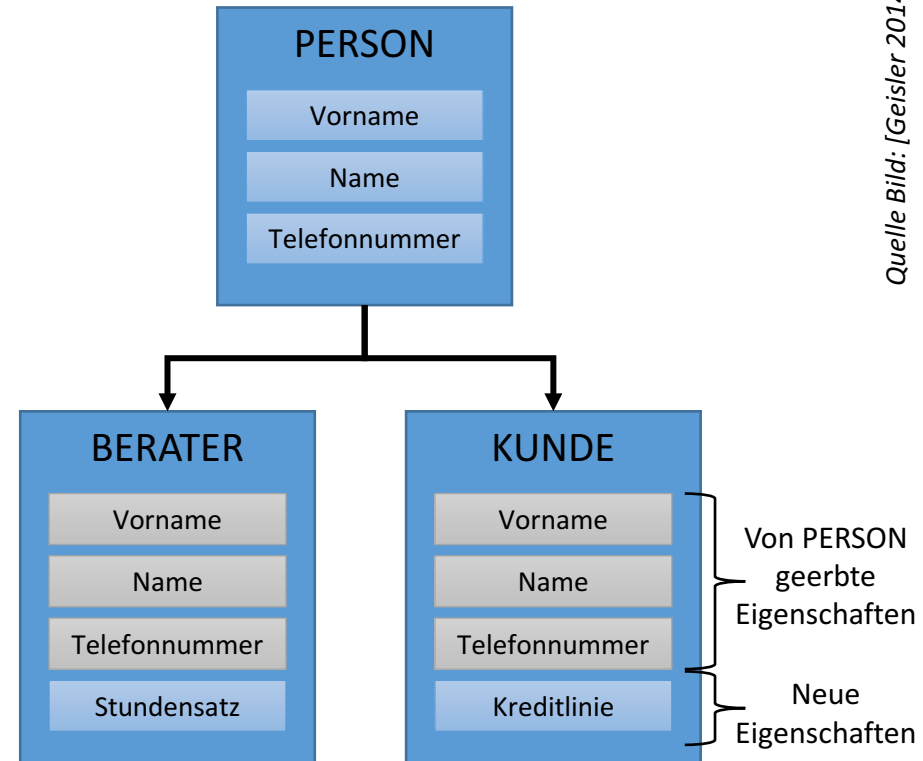
! Warum ist das ineffizient?

Objektorientiertes Modell

Object Oriented Data Modelling (OODM)

– Daten und Beziehungen sind in einer Struktur – dem Objekt – enthalten.

- Unterschied zwischen Entität und Objekt: Entität enthält keine Info zu Beziehungen; Ein Objekt enthält Beziehungen zwischen seinen Daten
- Komponenten des OODM:
 - **Objekt** – Abstraktion einer Entität aus der realen Welt. Wird manchmal als äquivalent zu einer Entität angesehen
 - **Attribute** sind Eigenschaften des Objektes
 - Objekte mit gleichen Eigenschaften sind in **Klassen** gruppiert
 - **Methoden** sind Teil der Klasse und werden zum bearbeiten der Daten eines Objektes benötigt
 - **Vererbung** und **Generalisierung**



Quelle Bild: [Geisler 2014]

Quelle: Vorlesungsfolien Prof. I. Stengel

Andere Datenmodelle

Extended Relational Data Model

- Eigenschaften einer OO Datenbank wurden zum relationalen Modell hinzugefügt
 - Relationale DB die OO Features unterstützen (Kapselung Daten und Methoden)
- **Extensible Data Type** – basierend auf Klassen und Vererbung
- Entstehen einer neuen Kategorie von DBMS – **Objektrelationale Datenbanken (ORDBMS)**
- Vorteile: Einfachheit, Daten Integrität wird gewährleistet, einfache Sprache zur Abfrage, gute Performance, hohe Verfügbarkeit, Sicherheit, gute Skalierbarkeit

XML (Extensible Markup Language)

- Ideal für den Austausch von strukturierter, semi-strukturierter und unstrukturierter Daten
- ORDBMS fügen eine Schnittstelle für XML-Daten hinzu

Big Data

Neue Herausforderungen für DBMS:

- Riesige Mengen von Daten aus dem Web
- Unterschiedliche Datenarten von strukturiert bis unstrukturiert
- Neue Technologien, mehr Ressourcen und Rechnerleistung notwendig

Big Data: Stichwort um massiv steigende Datenmengen, Performance, Skalierbarkeit und Kosten zu adressieren:

- Unstrukturierte Daten (Social Media, ...) können nicht immer mit relationalen Ansätzen bearbeitet werden
- Mehr Daten führen zu schlechter strukturierten Daten und zu mehr Bedarf an Speicherplatz, Rechenleistung und neue Algorithmen um Daten effektiv zu bearbeiten
- Relationale Ansätze in diesem Umfeld bringen große Kosten mit sich
- Es gibt keine Lösung für all diese Anforderungen



NOSQL

NOSQL-Datenbanken adressieren Anforderungen von Big Data

- Verarbeiten von großen nicht-strukturierten Daten
- Nutzen kein relationales Modell
- Unterstützen verteilte Architekturen
- Hohe Skalierbarkeit
- Hohe Verfügbarkeit
- Fehlertoleranz
- Nachteil: Es gibt kein Standard, nur viele Ansätze

NOSQL: Key-Value Modell (2)

Struktur besteht aus einem **Schlüssel (key)** und einem **Wert (value)**

- Schlüssel kann beliebig sein (ggf. mit Einschränkungen auf Text-Zeichen)
- Wert kann beliebig sein (ggf. mit Größenbeschränkungen): Text, Bild, Video, ...

Color	Red
Age	18
Size	Large
Name	Smith
Title	The Brown Dog

Quelle: <http://dba.stackexchange.com/questions/607/what-is-a-key-value-store-database>

NOSQL: Key-Value Modell (2)

- Jeder Zeileneintrag kann als ein Attribut einer Entität mit seinem Wert aufgefasst werden

```
user1923_color Red
user1923_age 18
user3371_color Blue
user4344_color Brackish
user1923_height 6' 0"
user3371_age 34
```

- Im relationalen Modell wird beim Hinzufügen eines Attributes Tabelle geändert, hier wird einfach Attribute als Zeile hinzugefügt
- Es werden kein Relationen zwischen Entitäten gespeichert! Dies ist die Aufgabe des Programmierers
- Indexieren und Suchen ist schwierig, wenn nicht nach Key gesucht wird

Quelle: <http://dba.stackexchange.com/questions/607/what-is-a-key-value-store-database>



Übungsaufgabe

Bilden Sie zwei Professoren der Hochschule Karlsruhe mit Hilfe des Key-Value-Modells ab

Zusammenfassung Kapitel 2

Datenbanksysteme helfen bei der Verarbeitung von Informationen

DBMS spielen dabei eine zentrale Rolle

- Können auf verschiedene Arten in Datenbankanwendungen eingebunden sein (ein-/zwei-/n-schichtig)
- DBMS können auf verschiedenen Datenmodellen basieren
 - Hier im Fokus: Das relationale Datenmodell



Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

Kapitel 3

Das relationale Datenbankmodell

Vorlesung Datenbanken und Informationssysteme I
Sommersemester 2017

Prof. Dr. rer. nat. Oliver P. Waldhorst

Basierend auf Vorlesungsfolien von Prof. I. Stengel

Inhalte dieses Kapitels

- Entitäten und Attribute vs. Tabellen und Spalten
- Datentypen und Domänen
- Primär- und Fremdschlüssel
- Funktionale Abhängigkeiten

Motivation **Relationale DBMS (RDBMS)**

Die meisten heute eingesetzten Datenbankmanagementsysteme (DBMS) sind relationale Datenbankmanagementsysteme RDBMS

- Große Vorteile für Datenbankdesigner
 - Insbesondere strukturelle Unabhängigkeit und Datenunabhängigkeit
- Nachteil: Komplexität des DBMS selbst
 - I.d.R. vor Datenbankdesigner verborgen, der sich nur mit Strukturen der logischen Ebene beschäftigt
 - „Geheimnis“ der Hersteller

Entitäten und Attribute

Entität = Objekt des täglichen Lebens

- Z.B. Berater Ingo Fuchs, Kundin Johanna Schulze

Gleichartige Entitäten entsprechen **Entitätstypen**

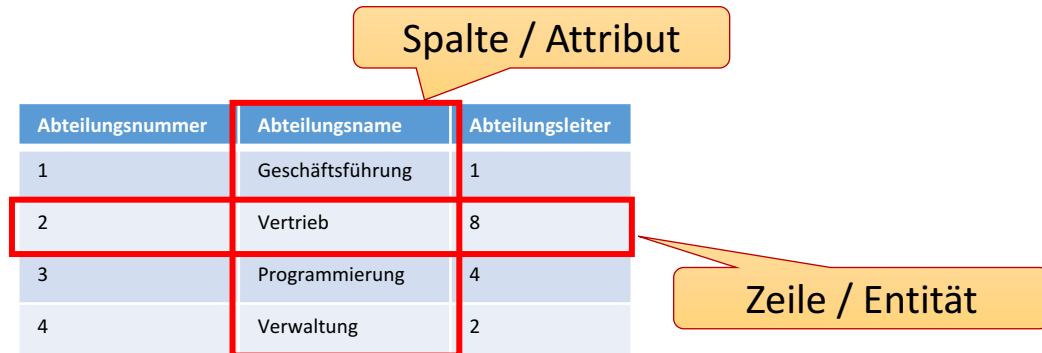
- Alle Entitäten eines Entitätstypen haben dieselben Charakteristika
 - Z.B. Berater Ingo Fuchs und Helena Meier haben
 - *Namen*
 - *Vornamen*
 - *Stundensatz*
- Es kann Entitätstyp BERATER definiert werden
- Namen, Vornamen, Stundensatz sind **Attribute** des Entitätstypen BERATER
- Jeder weitere Berater wird dem Entitätstyp BERATER zugeordnet

Entitäten vs. Tabellen

Zentrale Struktur in einem RDBMS ist die Tabelle

- In Tabellen werden Entitäten gespeichert
 - Tabelle ist Abbildung eines Entitätstyps in ein RDBMS
 - Begriffe Tabelle und Entitätstyp sind synonym

Beispiel:

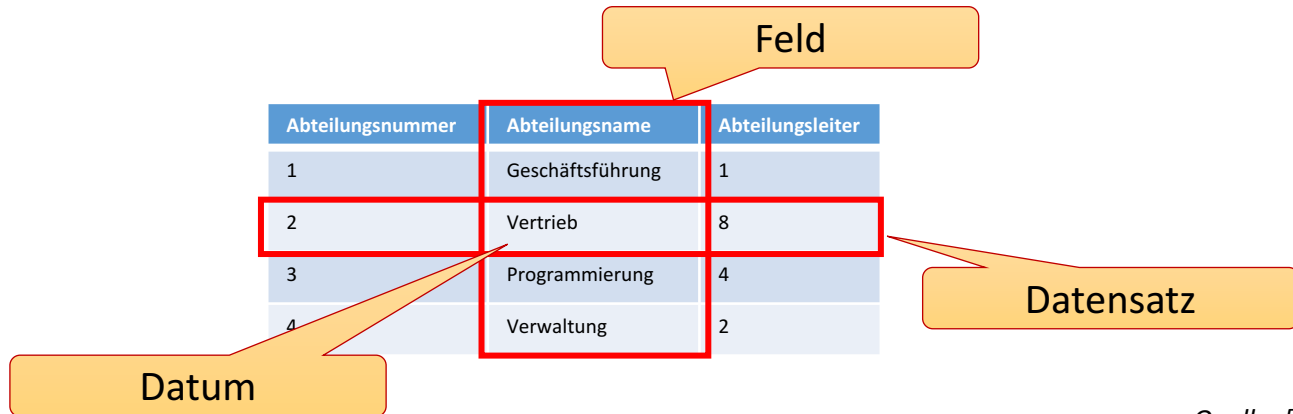


Abteilungsnummer	Abteilungsname	Abteilungsleiter
1	Geschäftsführung	1
2	Vertrieb	8
3	Programmierung	4
4	Verwaltung	2

Quelle: [Geisler 2014]

Wording

In der Praxis werden häufig die Begriffe Feld, Datensatz, Datum verwendet



Abteilungsnummer	Abteilungsname	Abteilungsleiter
1	Geschäftsführung	1
2	Vertrieb	8
3	Programmierung	4
4	Verwaltung	2

Quelle: [Geisler 2014]

Wertebereiche und Domänen

Jedes Attribut besitzt einen bestimmten **Datentyp**

- Alle Einträge einer Tabellenspalte haben diesen Datentyp
- Datentyp ist so zu wählen, dass alle möglichen Werte eines Attributs abgebildet werden können

Jedes Attribut besitzt außerdem eine **Domäne**

- Menge aller Werte, die ein Attribut annehmen kann
- Ist i.d.R. nicht der Wertebereich des Datentyps

Beispiel: Schulnoten

- Lassen sich mit dem Datentyp **FLOAT (2)** darstellen
- Haben die Domäne {„1,0“, „1,3“, „1,7“, „2,0“, ..., „5,0“}

! Übungen

Definieren Sie die Domäne für das *Alter* einer Person

- Mit welchem Datentyp könnten Sie dieses abbilden?
- Welchen Wertebereich hat das Attribut?

Definition Schlüssel

Ein **Schlüssel** ist ein Attribut (oder eine minimale Kombination von Attributen), deren Werte für die eindeutige Identifizierung der Datensätzen der Datenbanktabellen verwendet werden können

- Man nennt diesen auch **Primärschlüssel** (engl.: **Primary Key**)

Personalnr	Nachname	Vorname	Funktion	Abteilung
1	Geldberg	Günter	Geschäftsführer	1
2	Buchhalter	Benno	Abteilungsleiter	4
3	Fehler	Fritz	Programmierer	3
4	Hacker	Hans	Chefprogrammierer	3
...

Primärschlüssel

Beobachtungen zu Schlüsseln

Schlüssel besteht nur aus einem Attribut

→ Jeder Datensatz hat einen anderen Wert in diesem Feld

Schlüssel besteht aus mehreren Attributen (**Schlüsselattributen**)

→ Jede Wertekombination der Attribute darf in max. einem Datensatz angenommen werden

In der Theorie muss jede Tabelle einen Primärschlüssel aufweisen

- In der Praxis erfordert nicht jedes DBMS Definition eines Primärschlüssels
- Falls Primärschlüssel definiert ist, muss beim Hinzufügen eines Datensatzes mindestens der Schlüssel gegeben sein, alle anderen Werte können unbekannt (“Nullwerte”) sein

Einschub: Nullwerte

Einem Feld einer Tabelle kann (u.U. vorübergehend) kein Wert zugeordnet werden

- Wird mit speziellem **Nullwert** (engl.: **Null Value**) belegt
- **NULL** bedeutet „nicht existent“, „nicht bekannt“, „noch nicht bekannt“ oder „an dieser Stelle ohne Bedeutung“
- **Darf nicht mit dem numerischen Wert 0 (engl.: zero) verwechselt werden!**

Nullwerte in Tabellen führen zu gewissen Eigenheiten bei Auswertung von Abfragen durch das Datenbanksystem

- **Prüfung von Bedingungen, die Null-werte enthalten, ist nicht entscheidbar**
- In solchen Fällen liefert SQL als Ergebnis einer Anfrage stets den **Wert NULL** zurück.
 - Kann auf den ersten Blick zu unerwarteten Ergebnissen führen!

Funktionale Abhängigkeit

Bsp. vollst. funkt. Abh.:
(Name, ProjNr) \rightarrow Projekt

Definition: Gegeben sei eine Tabelle

- Eine Menge von Attributen B der Tabelle ist **funktional abhängig** von einer Menge von Attributen A der Tabelle, wenn es zu jeder konkreten Belegung von A nur maximal eine konkrete Belegung von B geben kann (Schreibweise $A \rightarrow B$)
- Ist B von keiner Teilmenge von A funktional abhängig, so heißt B **vollständig funktional abhängig** von A

Beispiel: Hier gilt $MitNr \rightarrow Name$

! Unter welcher Voraussetzung gilt das? Vollständig funktional abhängig?

ProjNr \rightarrow Abteilung
gilt nicht

MitNr	Name	AbtNr	Abteilung	ProjNr	Projekt
1	Egon	42	DB	1	Infra
1	Egon	42	DB	2	Portal
2	Erna	42	DB	2	Portal
2	Erna	42	DB	3	Frame
3	Uwe	43	GUI	1	Infra
3	Uwe	43	GUI	3	Portal

Funktionale Abhängigkeit

Anmerkung: Im konkreten Fall kann nicht auf eine funktionale Abhängigkeit geschlossen werden

- Dazu ist Wissen über den Anwendungsbereich notwendig
- Man kann meistens nur schlussfolgern das eine Abhängigkeit nicht gegeben ist

Beispiel: *Name* \rightarrow *MitNr*

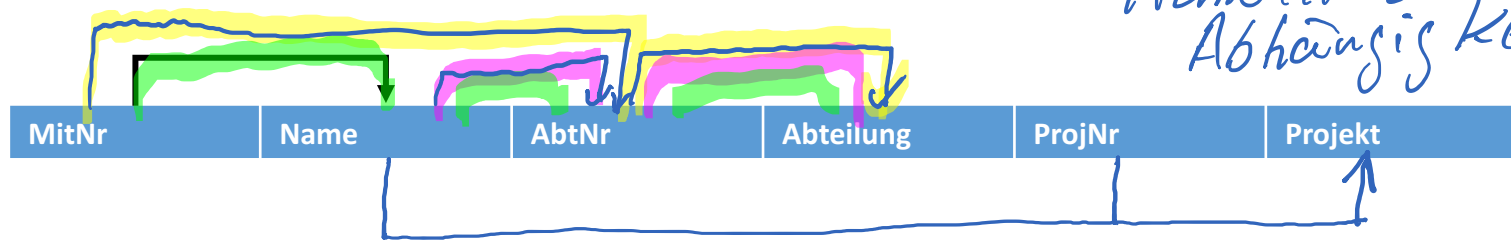
- *MitNr* ist funktional abhängig von *Name*, d.h. wir können vom Namen auf die *MitNr* schließen
- Allerdings nur so lange bis ein zweiter Mitarbeiter mit dem gleichen Namen hinzugefügt wird!

! Wie sieht es für *ProjNr* \rightarrow *Abteilung* aus?

Graphische Darstellung von Abhängigkeiten

Funktionale Abhängigkeiten können durch
Abhängigkeitsdiagramm dargestellt werden

- Beispiel: $MitNr \rightarrow Name$




! Zeichnen Sie weitere funktionale Abhängigkeiten ein!

Transitive funktionale Abhängigkeiten

Transitivität:

$MitNr \rightarrow AbtNr$ und $AbtNr \rightarrow Abteilung$

→ Dann kann man $MitNr \rightarrow Abteilung$ schlussfolgern



MitNr	Name	AbtNr	Abteilung	ProjNr	Projekt
1	Egon	42	DB	1	Infra
1	Egon	42	DB	2	Portal
2	Erna	42	DB	2	Portal
2	Erna	42	DB	3	Frame
3	Uwe	43	GUI	1	Infra
3	Uwe	43	GUI	3	Portal

Transitivität im Abhängigkeitsdiagramm

Transitivität ist durch „Ketten von Pfeilen“ zu erkennen

- Beispiel: $MitNr \rightarrow AbtNr \rightarrow Abteilung$



Kann vereinfacht dargestellt werden durch:



! Gibt es weitere transitive Abhängigkeiten?

→ s. Folie 13



Primärschlüssel und funktionale Abhängigkeit

Für einen **Primärschlüssel** gilt für alle Attribute $A1, A2, \dots$ der Tabelle:

$$P \rightarrow A1, A2, \dots$$

Für Primärschlüssel mit mehreren Attributen $P1, P2$ gilt:

$$P1, P2 \rightarrow A1, A2, \dots$$

D.h. alle Attribute sind funktional vom Primärschlüssel abhängig!

Frage: Wie wählen wir den Primärschlüssel aus?

Superschlüssel

Für unsere Beispieltabelle gilt:

(MitNr, ProjNr)

→ *(MitNr, ProjNr, Name, AbtNr, Abteilung, Projekt)*

Ein möglicher Schlüssel ist damit =
(MitNr, ProjNr)

Unsere Tabelle hat noch weitere mögliche Schlüssel:

(MitNr, ProjNr, Name)

(MitNr, ProjNr, Abteilung)

(MitNr, ProjNr, Projekt)

(MitNr, ProjNr, AbtNr)

(MitNr, ProjNr, Name, Abteilung)

...

All diese sind **Superschlüssel**

MitNr	Name	AbtNr	Abteilung	ProjNr	Projekt
1	Egon	42	DB	1	Infra
1	Egon	42	DB	2	Portal
2	Erna	42	DB	2	Portal
2	Erna	42	DB	3	Frame
3	Uwe	43	GUI	1	Infra
3	Uwe	43	GUI	3	Portal

Schlüsselkandidaten

Es gibt viele Superschlüssel (auch triviale - welche?)

- Wir sind allerdings an minimalen Schlüsseln interessiert!

Ein Schlüssel P_1, P_2, \dots , von dem alle Attribute A_1, A_2, \dots vollständig funktional abhängen, heißt **Schlüsselkandidat**

P_1, P_2, \dots ist minimal, wir können kein Attribut weglassen!

! Nennen sie mögliche Schlüsselkandidaten!

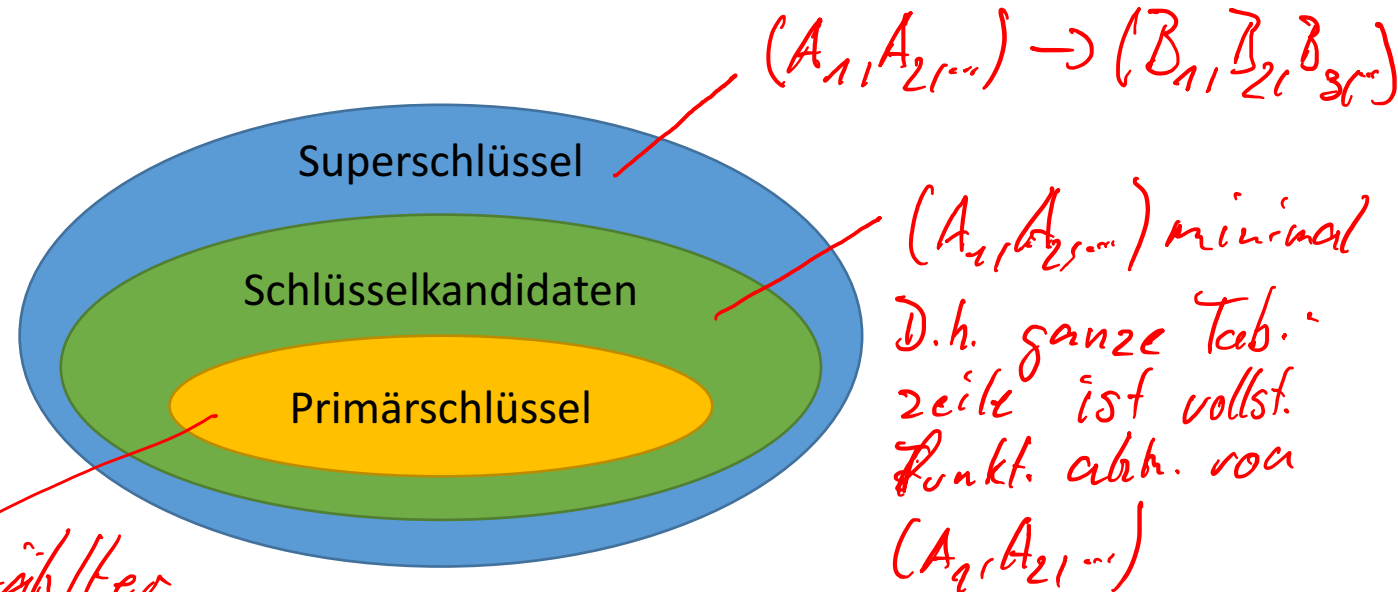
MitNr	Name	AbtNr	Abteilung	ProiNr	Projekt
1	Egon	42	DB	1	Infra
1	Egon	42	DB	2	Portal
2	Erna	42	DB	2	Portal
2	Erna	42	DB	3	Frame
3	Uwe	43	GUI	1	Infra
3	Uwe	43	GUI	3	Portal



Zusammenhang der Schlüssel

Tabelle $B_1, B_2, B_3 \dots$

Primärschlüssel sollte aus den Schlüsselkandidaten gewählt sein!



Quelle: Wikipedia

Künstliche und natürliche Primärschlüssel

Beispiel Projekt: *ProjID*, *ProjName*, *Datum*,...

- *ProjID* ist ein künstlicher Primärschlüssel

Was wäre ein natürlicher Primärschlüssel?

- Keine generierten Identifier!

Beispiel

- *Personalausweisnr* in
Person: *Name*, *Geburtstag*, *Geburtsort*, *Adresse*, *Personalausweisnr*,...

Warum?

Natürliche Primärschlüssel müssen gut gewählt sein!

Arten von Schlüsseln

Man beachte, dass mit dem Begriff Schlüssel also u. U. ganz unterschiedliche Sachverhalte bezeichnet werden. So verwendet man ein und denselben Begriff insbesondere auch dann, wenn eigentlich präziser von

- Primärschlüssel
- Fremdschlüssel
- Sekundärschlüssel
- Zugriffsschlüssel
- Sortierschlüssel

Der Primärschlüssel ist lediglich ein Identifikationsschlüssel

- **Kann keinerlei Aussagen bezüglich der Zugriffsmöglichkeiten auf die Daten oder der Sortierung der Tupel implizieren**

Fremd- und Sekundärschlüssel

Fremdschlüssel (engl.: **Foreign Key**) wird zur Repräsentation von Beziehungen zwischen Objekten im relationalen Modell genutzt

- Attribut hat die Funktion eines "Fremdschlüssels", wenn es auf einen Datensatz in anderer Relation (über deren Primärschlüssel bzw. einen Schlüsselkandidaten) verweist

Sekundärschlüssel dienen dazu, Gruppen von Objekten mit gleichen Eigenschaften einzugrenzen

- Beispiel: „Die Gruppe aller Mitarbeiter, die in Karlsruhe wohnen“
 - Hier dient Attribut Wohnort als Sekundärschlüssel

! Übung Primär-/Fremdschlüssel



- Was sind Schlüsselkandidaten in beiden Tabellen?
- Welches ist jeweils ein guter Primärschlüssel?
- Was wird jeweils als Fremdschlüssel verwendet?
- Was wäre ein möglicher Sekundärschlüssel?

Primärschlüssel

Fremdschlüssel

Kein Schlüsselkandidat

Abteilungsnummer	Abteilungsname	Abteilungsleiter
1	Geschäftsführung	1
2	Vertrieb	8
3	Programmierung	4
4	Verwaltung	2

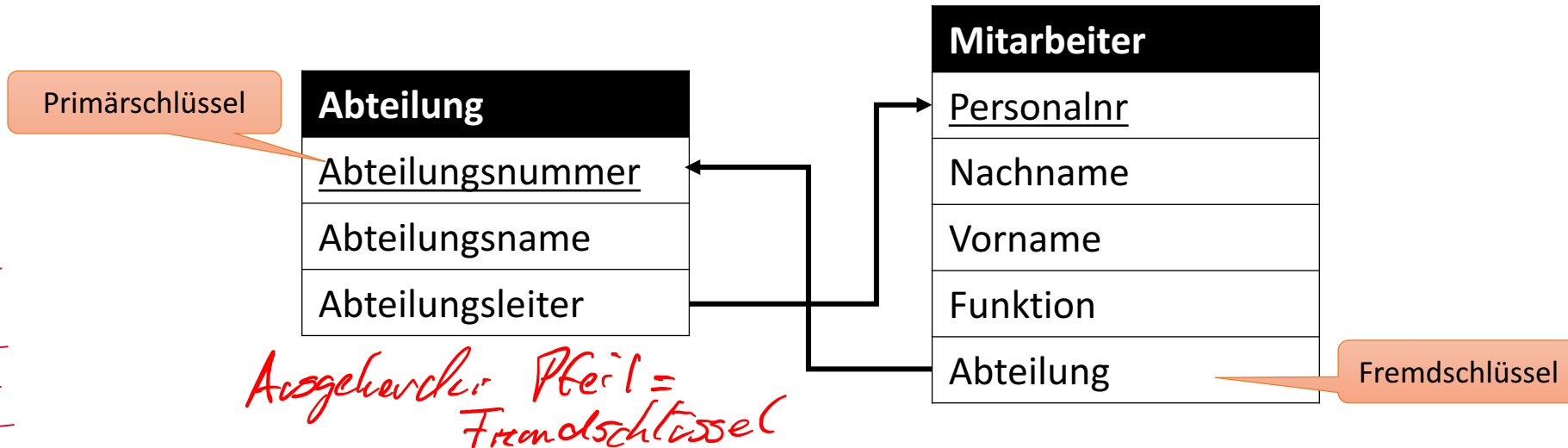
Personaln.	Nachname	Vorname	Funktion	Abteilung
1	Geldberg	Günter	Geschäftsführer	1
2	Buchhalter	Benno	Abteilungsleiter	4
3	Fehler	Fritz	Programmierer	3
4	Hacker	Hans	Chefprogrammierer	3
5	Nieda	Frieda	Chefsekretärin	1
6	Gründlich	Gerda	Sachbearbeiterin	4
7	Klugscher	Karl	Azubi	2
8	Lueger	Ludwig	Abteilungsleiter	2
801	Tutnix	Toni	Praktikant	3
...

Schema-Diagramme

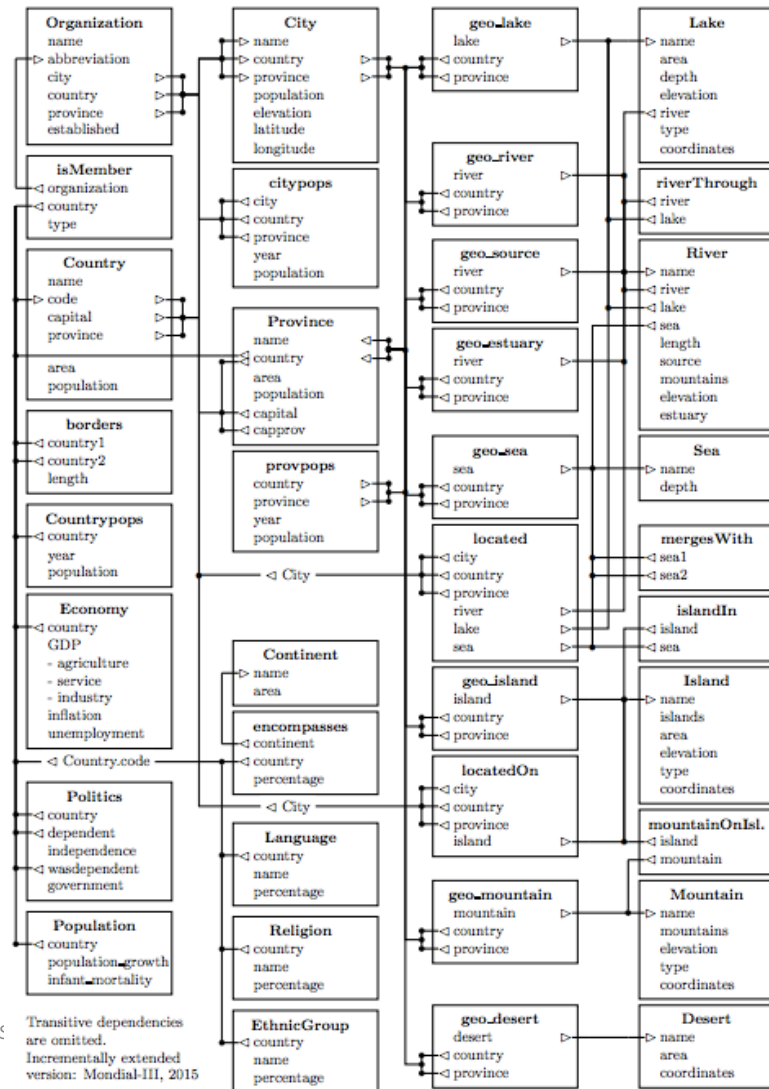
Häufig ist grafische Darstellung des Datenbankschemas mit Primär und Fremdschlüsseln gegeben

Beispiel:

*Unterstrichen =
Primärschlüssel*



Beispiel: Mondial- Datenbank (vgl. Übung)



Auf der Webseite
<https://www.dbis.informatik.uni-goettingen.de/Mondial/>
zu finden als
„Referential
Dependency Diagram“

Zusammenfassung Kapitel 3

Relationale Datenbanken speichern Daten in Tabellen

- Tabellenstruktur repräsentiert Entitätstypen
- Tabellenspalten repräsentieren Attribute mit Datentyp und Domäne
- Tabellenzeilen repräsentieren Entitäten

Beziehungen werden durch Primär-/Fremdschlüssel abgebildet

- Primärschlüssel basiert auf Konzept der funktionalen Abhängigkeit



Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

Kapitel 4

Datenbanknormalisierung

Vorlesung Datenbanken und Informationssysteme I
Sommersemester 2017

Prof. Dr. rer. nat. Oliver P. Waldhorst

Inhalte dieses Kapitels

- Motivation Normalisierung
- 1. Normalform
- 2. Normalform
- 3. Normalform
- Nachteile der Normalisierung

Normalisierung

Relationale Datenbanken enthalten häufig **Redundanzen**

- Gleiche Information wird mehrfach gespeichert
- Folgen:
 - Erhöhter Speicherplatzbedarf
 - Gefahr von Inkonsistenzen, wenn Daten nur teilweise oder unvollständig geändert werden (**Anomalien**)

Die **Normalisierung** ist ein Vorgehen zur Vermeidung von Redundanzen

- Ergebnis ist eine Menge von Tabellen in **Normalform(en)**
 - Gleichen Semantik wie Ursprungstabelle(n)
 - Keine vermeidbaren Redundanzen
 - Die Menge der Tabellen nimmt durch den Normalisierungsprozess zu

1. Beispiel

Titel können (mit einfachen Mitteln) nur alle gleichzeitig als Titelliste oder gar nicht dargestellt werden

Tabelle CD_Lied

CD_ID	Album	Gründungsjahr	Titelliste
4711	Anastacia – Not That Kind	1999	{1. Not That Kind, 2. I'm Outta Love, 3. Cowboys & Kisses}
4712	Pink Floyd – Wish You Were Here	1965	{1. Shine On You Crazy Diamond}
4713	Anastacia – Freak of Nature	1999	{1. Paid my Dues}

Zur Sortierung nach Albumtitel muss Attribut **Album** in **Interpret** und **Albumtitel** aufgeteilt werden

1. Normalform - Definition

Eine Tabelle R ist in **1. Normalform (1NF)**, wenn die Wertebereiche aller Attribute von R **atomar** sind

Atomar bedeutet:

- Nur einfache, unstrukturierte Attribute sind erlaubt
- Listenartige, mengenwertige oder aufzählungsartige Attribute sind nicht erlaubt
- Erlaubte Datentypen: *integer, real, string (CHAR(), VARCHAR()), enum*
- Nicht erlaubte Datentypen: *array, record, list*



1. Beispiel: 1NF verletzt

Attribut **Titelliste** enthält
eine Menge von Titeln

<i>CD_ID</i>	Album	Gründungsjahr	Titelliste
4711	Anastacia – Not That Kind	1999	{1. Not That Kind, 2. I'm Outta Love, 3. Cowboys & Kisses}
4712	Pink Floyd – Wish You Were Here	1965	{1. Shine On You Crazy Diamond}
4713	Anastacia – Freak of Nature	1999	{1. Paid my Dues}

Attribut **Album** enthält
Attributwertebereiche
Interpret und *Albumtitel*

! Übung: Bringen Sie diese Tabelle in die 1NF!

Tabelle <i>CD_Lied</i>			
<i>CD_ID</i>	Album	Gründungsjahr	Titelliste
4711	Anastacia – Not That Kind	1999	{1. Not That Kind, 2. I'm Outta Love, 3. Cowboys & Kisses}
4712	Pink Floyd – Wish You Were Here	1965	{1. Shine On You Crazy Diamond}
4713	Anastacia – Freak of Nature	1999	{1. Paid my Dues}

Mögliche Lösung

Tabelle *CD_Lied*

<i>CD_ID</i>	Albumtitel	Interpret	Gründungsjahr	<i>Track</i>	Titel
4711	Not That Kind	Anastacia	1999	1	Not That Kind
4711	Not That Kind	Anastacia	1999	2	I'm Outta Love
4711	Not That Kind	Anastacia	1999	3	Cowboys & Kisses
4712	Wish You Were Here	Pink Floyd	1965	1	Shine On You Crazy Diamond
4713	Freak of Nature	Anastacia	1999	1	Paid my Dues

2. Beispiel

Tabelle *CD_Lied*

<i>CD_ID</i>	Albumtitel	Interpret	Gründungsjahr	<i>Track</i>	Titel
4711	I Don't Mind	Anastacia	1999	1	Not That Kind
4711	Not That Kind	Anastacia	1999	2	I'm Outta Love
4711	Not That Kind	Anastacia	1999	3	Cowboys & Kisses
4712	Wish You Were Here	Pink Floyd	1965	1	Shine On You Crazy Diamond
4713	Freak of Nature	Anastacia	1999	1	Paid my Dues

Es werden redundante Informationen gespeichert

2. Normalform - Definition

Ein Tabelle R ist in **2. Normalform (2NF)**, wenn

- R in 1. Normalform ist und
- jedes Nichtschlüsselattribut vollständig funktional von jedem Schlüssel abhängt (und nicht nur von einem Teil des Schlüssels)

Wiederholung Definition **funktionale Abhängigkeit**:

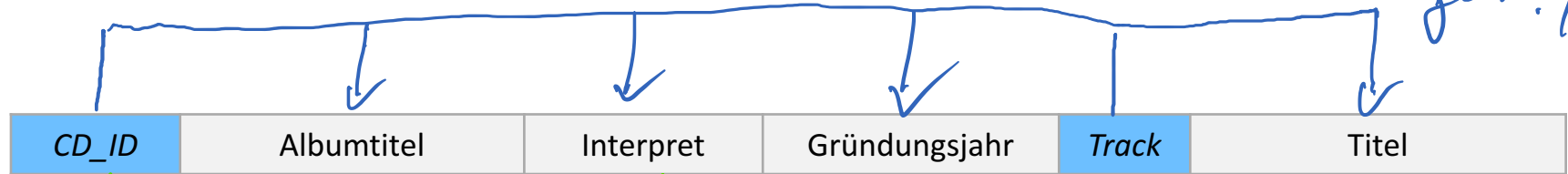
- Eine Menge von Attributen Y von R ist **funktional abhängig** von einer Menge von Attributen X von R, wenn es zu jeder konkreten Belegung von X nur maximal eine konkrete Belegung von Y geben kann (Schreibweise $X \rightarrow Y$)
- Ist Y von keiner Teilmenge von X funktional abhängig, so heißt Y **vollständig funktional abhängig** von X

! Übung: Abhängigkeitsdiagramm

Zeichnen Sie in einem Abhängigkeitsdiagramm alle funktionalen Abhängigkeiten

- vom gesamten Primärschlüssel ($CD_ID, Track$)
- von den einzelnen Schlüsselattributen CD_ID und $Track$

Alle Attr. hängen vom Primärschlüssel ab (das ist gut!)



Manche Attribute hängen nur von Teilschlüssel ab (Das ist schlecht) → 2. NF verletzt

2. Beispiel: 2NF verletzt

Tabelle CD_Lied

<i>CD_ID</i>	Albumtitel	Interpret	Gründungsjahr	<i>Track</i>	Titel
4711	Not That Kind	Anastacia	1999	1	Not That Kind
4711	Not That Kind	Anastacia	1999	2	I'm Outta Love
4711	Not That Kind	Anastacia	1999	3	Cowboys & Kisses
4712	Wish You Were Here	Pink Floyd	1965	1	Shine On You Crazy Diamond
4713	Freak of Nature	Anastacia	1999	1	Paid my Dues

Das Attribut **Albumtitel** hängt funktional vom Teil-Schlüssel *CD_ID* ab (analog **Interpret**, **Gründungsjahr**)

! Übung: Bringen Sie diese Tabelle in 2NF!

Tabelle CD_Lied					
CD_ID	Albumtitel	Interpret	Gründungsjahr	Track	Titel
4711	Not That Kind	Anastacia	1999	1	Not That Kind
4711	Not That Kind	Anastacia	1999	2	I'm Outta Love
4711	Not That Kind	Anastacia	1999	3	Cowboys & Kisses
4712	Wish You Were Here	Pink Floyd	1965	1	Shine On You Crazy Diamond
4713	Freak of Nature	Anastacia	1999	1	Paid my Dues

Für 2. NF: Attribute, die von Teilschlüssel abhängen in eigene Tabelle auslagern (inkl. Teilschlüssel)

Mögliche Lösung

Tabelle <i>CD</i>			
<i>CD_ID</i>	Albumtitel	Interpret	Gründungsjahr
4711	Not That Kind	Anastacia	1999
4712	Wish You Were Here	Pink Floyd	1965
4713	Freak of Nature	Anastacia	1999

Tabelle <i>Lied</i>		
<i>CD_ID</i>	<i>Track</i>	Titel
4711	1	Not That Kind
4711	2	I'm Outta Love
4711	3	Cowboys & Kisses
4712	1	Shine On You Crazy Diamond
4713	1	Paid my Dues

3. Beispiel

Tabelle CD			
CD_ID	Albumtitel	Interpret	Gründungsjahr
4711	Not That Kind	Anastacia	1999
4712	Wish You Were Here	Pink Floyd	1965
4713	Freak of Nature	Anastacia	1999

Hier besteht immer noch Redundanz!

3. Normalform - Definition

Ein Tabelle R ist in **3. Normalform (3NF)**, wenn

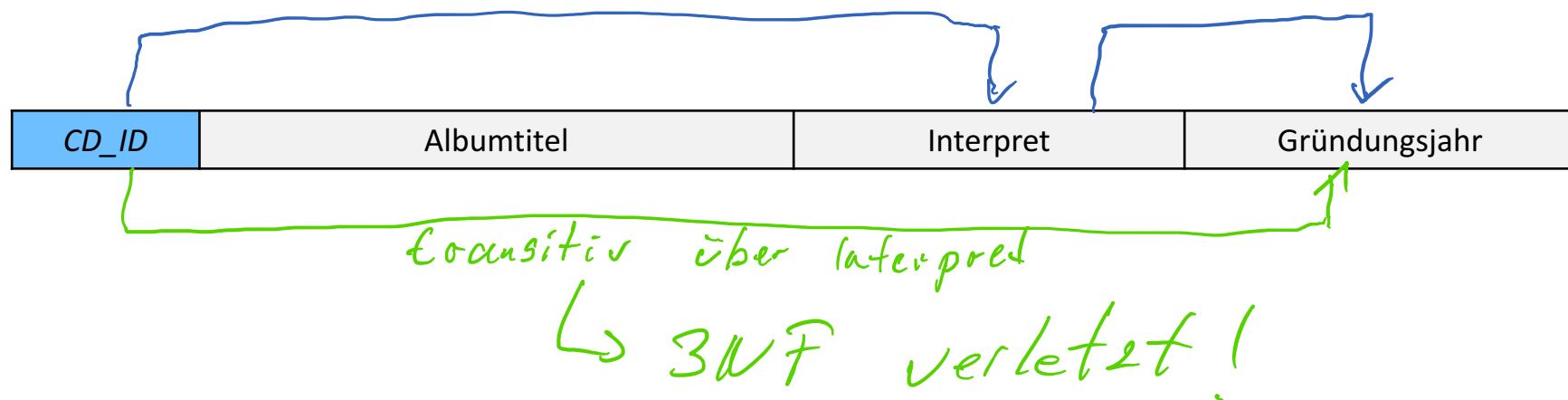
- R in 2. Normalform ist (und somit auch in 1. Normalform) und
- wenn kein Nichtschlüsselattribut transitiv über ein anderes Nichtschlüsselattribut von einem Schlüsselkandidaten abhängt

Transitive Abhängigkeit bedeutet:

- Es gibt Abhängigkeiten über andere Attribute
- Beispiel: Attribut C hängt transitiv von Attribut A ab, wenn es ein Attribut B gibt, so dass gilt: $A \rightarrow B$ und $B \rightarrow C$

! Übung: Abhängigkeitsdiagramm

Zeichnen Sie in einem Abhängigkeitsdiagramm alle transitiven Abhängigkeiten vom Primärschlüssel CD_ID



Beispiel: 3NF verletzt

Tabelle CD

CD ID	Albumtitel	Interpret	Gründungsjahr
4711	Not That Kind	Anastacia	1999
4712	Wish You Were Here	Pink Floyd	1965
4713	Freak of Nature	Anastacia	1999

Offensichtlich lässt sich *Interpret* einer CD aus *CD_ID* bestimmen

Gründungsjahr von Band/Interpreten hängt wiederum vom *Interpreten* und damit transitiv von *CD_ID* ab

Quelle Beispiel: Wikipedia

! Übung: Bringen Sie diese Tabelle in die 3NF!

Tabelle CD			
CD_ID	Albumtitel	Interpret	Gründungsjahr
4711	Not That Kind	Anastacia	1999
4712	Wish You Were Here	Pink Floyd	1965
4713	Freak of Nature	Anastacia	1999

Für 3NF Attribute mit trans. Abhängigkeit
in neue Tabelle auslagern

Mögliche Lösung → bessere Lösung
siehe Tafelbild

Tabelle CD		
CD_ID	Albumtitel	Interpret_ID
4711	Not That Kind	311
4712	Wish You Were Here	312
4713	Freak of Nature	311

Tabelle Künstler		
Interpret_ID	Interpret	Gründungsjahr
311	Anastacia	1999
312	Pink Floyd	1965

- Die Tabellen sind nun frei von Redundanzen!

Nachteile der 1. Normalform

Tabelle <i>CD_Lied</i>					
<i>CD_ID</i>	Albumtitel	Interpret	Gründungsjahr	<i>Track</i>	Titel
4711	Not That Kind	Anastacia	1999	1	Not That Kind
4711	Not That Kind	Anastacia	1999	2	I'm Outta Love
4711	Not That Kind	Anastacia	1999	3	Cowboys & Kisses
4712	Wish You Were Here	Pink Floyd	1965	1	Shine On You Crazy Diamond
4713	Freak of Nature	Anastacia	1999	1	Paid my Dues

Unnötig viele Attribute

- Z.B. *Album* und *Interpret*

Ggf. mehrere Zugriffe notwendig

- Z.B. zum Anzeigen aller Titel eines Albums

Rücknahme der 1. Normalform kann sinnvoll sein!

Nachteile der 2. und 3. Normalform

Beispiel: SQL-Anfrage zur Ausgabe aller Albumtitel von Interpreten, deren Namen mit „A“ beginnt, nach Albumtitel aufsteigend sortiert

Tabelle CD		
CD_ID	Albumtitel	Interpret_ID
4711	Not That Kind	311
4712	Wish You Were Here	312
4713	Freak of Nature	311

Tabelle Künstler		
Interpret_ID	Interpret	Gründungsjahr
311	Anastacia	1999
312	Pink Floyd	1965

SELECT Interpret, Albumtitel
FROM CD **INNER JOIN** Künstler **USING** (Interpret_ID)
WHERE Interpret **LIKE** 'A%'
ORDER BY Albumtitel **ASC**

Handwritten red note: aufwändig!

Nachteile der 2. und 3. Normalform

Vereinfacht sich stark, wenn Tabelle CD auch Interpreten enthält!

Tabelle CD			
CD_ID	Albumtitel	Interpret	Gründungsjahr
4711	Not That Kind	Anastacia	1999
4712	Wish You Were Here	Pink Floyd	1965
4713	Freak of Nature	Anastacia	1999

```
SELECT Interpret, Albumtitel
FROM CD
WHERE Interpret LIKE 'A%'
ORDER BY Albumtitel ASC
```

→ nur eine Tabelle!

Rücknahme der zweiten Normalform kann sinnvoll sein!

Lohnt sich die Normalisierung?

Auswirkungen der Normalisierung [Lee95]

1. Weniger Anomalien – Kostenersparnis ϕ
2. Weniger Speicherplatzbedarf – Kostenersparnis ψ
3. Verlängerte Antwortzeiten durch JOINS – Kosten Ω

Normalisierung lohnt sich, wenn $\phi + \psi \geq \Omega$

Zusammenfassung Kapitel 4

Normalisierung ist ein geeignetes Werkzeug, um Redundanzen (und damit Anomalien) formal zu vermeiden

- Vorteile bei schreibenden Zugriffen!

Normalisierte Tabellen haben allerdings Nachteile bei lesenden Zugriffen

- Aufwändige / mehrfache Zugriffe oder JOINS notwendig

In bestimmten Anwendungsfällen kann auf Normalisierung verzichtet werden!



Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

Kapitel 5 Die relationale Algebra

Vorlesung Datenbanken und Informationssysteme I
Sommersemester 2017

Prof. Dr. rer. nat. Oliver P. Waldhorst

Inhalte dieses Kapitels

- Darstellung von Tabellen als Relationen
- Grundoperatoren der relationalen Algebra
 - Selektion, Projektion, Umbenennung, Vereinigung, Komplement, kartesisches Produkt
- Zusätzliche Operatoren
 - Zuweisung, Schnittmenge, Verbund (Join)
- Aggregatfunktionen und Gruppierung

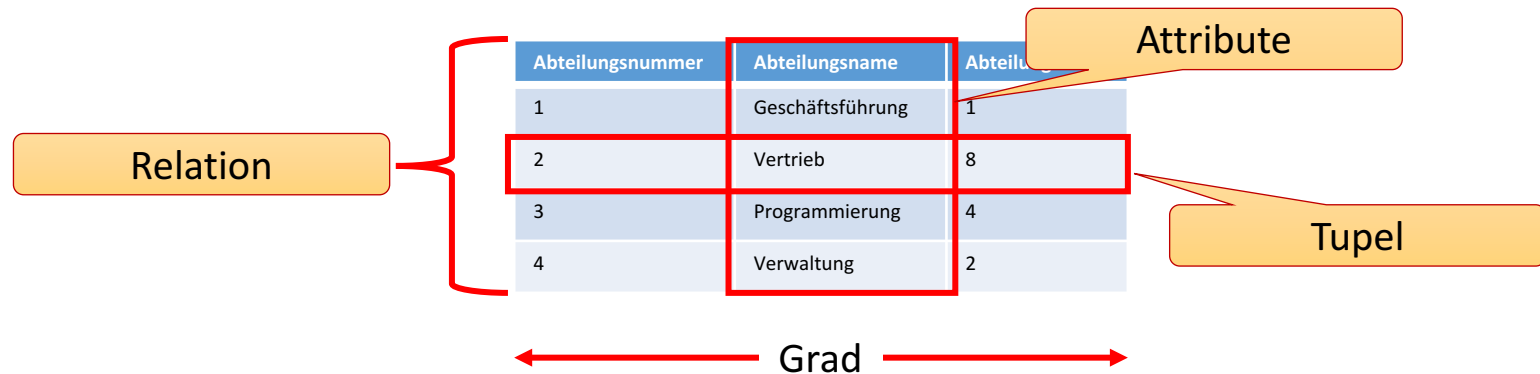
Eine andere Sichtweise: Relationen

Sei (A_1, A_2, \dots, A_n) eine Menge von Attributen mit Wertebereichen M_i

- Eine **n -stellige Relation R** ist eine Teilmenge des kartesischen Produkts $M_1 \times M_2 \times \dots \times M_n$
- Man schreibt auch: $R(A_1, A_2, \dots, A_n)$
- Der **Grad n der Relation R** ist die Anzahl der Attribute der Relation

Anschaulich entspricht eine Relation R wiederum einer Tabelle

- Attributnamen A_i sind die Spaltenüberschriften
- Tupel (a_1, a_2, \dots, a_n) mit Attributwerten a_i entsprechen Zeilen und beschreiben einzelne Datensätze



Beispiel Relationen

Attribut i	Wertebereich Mi
Abteilungsnummer	{1, 2, 3, 4}
Abteilungsname	{'Geschäftsführung', 'Vertrieb', 'Programmierung', 'Verwaltung'}
Abteilungsleiter	{1, 2, 4, 8}

Abteilungsnummer	Abteilungsname	Abteilungsleiter
1	Geschäftsführung	1
2	Vertrieb	8
3	Programmierung	4
4	Verwaltung	2

Die relationale Algebra

Menge von Operatoren auf Relationen

- Dienen zur Formulierung von Anfragen auf Relationen
- Eingabe: Relation(en)
 - **Unärer Operator**: Eingabe ist eine Relation
 - **Binärer Operator**: Eingabe sind zwei Relationen
- Ausgabe: Eine (neue) Relation

Operatoren der relationalen Algebra lassen sich beliebig schachteln!

- Vgl. arithmetische Ausdrücke, z.B. $(1 + 2) * 3$

Übersicht Operatoren

Unäre Operatoren

- Selektion
- Projektion
- Umbenennung
- Zuweisung

Binäre Operatoren

- Vereinigung
- Komplement
- Kartesisches Produkt
- Schnittmenge
- Verbund (Join)

Schwarz: Grundoperatoren
Blau: zusätzliche Operatoren

Die Selektion σ

Gegeben

- Relation $R(A_1, A_2, \dots, A_n)$
- **Prädikat** β
 - logischer Ausdruck auf Attributen A_1, A_1, \dots, A_n
 - Ergebnis: *wahr* oder *falsch*

$$\sigma_{\beta}(R) := \{t | t \in R \wedge t \text{ erfüllt } \beta\}$$

Anschaulich:

- Die Menge aller Tupel aus R , die das Prädikat β erfüllen

Beispiele Selektion

Predikat β



Relation R :

A	B	C
1	2	4
4	6	7
1	6	7
8	6	1

$\sigma_{A=1}(R)$:

A	B	C
1	2	4
1	6	7

Ergebnis
ist
Relation
(Tabelle!)

$\sigma_{C>6}(R)$:

A	B	C
4	6	7
1	6	7

Anmerkungen zum Prädikat β

Logischer Ausdruck auf A_1, A_2, \dots, A_n

- Kann Vergleiche mit $=, \neq, <, \leq, \geq, >$ enthalten

- Beispiel: $A = 1$

$$\sigma_{A=1}(R)$$

- Verkettung von Vergleichen mit \wedge, \vee, \neg möglich

- Beispiel: $B = 6 \wedge C = 7$

$$\sigma_{B=6 \wedge C=7}(R)$$

- Auch Vergleiche zwischen Attributen möglich

- Beispiel: $A = B$

$$\sigma_{A=B}(R)$$

! Übung: Selektion (1)

$$\sigma_{B=6 \wedge C=7}(R):$$

Relation R:

A	B	C
1	2	4
4	6	7
1	6	7
8	6	1

A	B	C
4	6	7
1	6	7

$$\sigma_{A>B}(R)$$

A	B	C
8	6	1

Bestimmen Sie die Ergebnisse der folgenden Anfragen:

- $\sigma_{B=6 \wedge C=7}(R)$
- $\sigma_{A>B}(R)$

! Übung: Selektion (2)

Relation *Dozent*:

ID	Name	Abteilung	Gehalt
1010	Srinivasan	Informatik	65000
12121	Wu	Wirtschaft	90000
15151	Mozart	Musik	40000
22222	Einstein	Physik	95000
32343	El Said	Geschichte	60000
33456	Gold	Physik	87000
45565	Katz	Informatik	75000
58583	Califieri	Geschichte	62000
76543	Singh	Wirtschaft	80000
76766	Crick	Biologie	72000
83821	Brandt	Informatik	92000
98345	Kim	Elektrotechnik	80000

Geben Sie relationale
Anfragen an, die die
folgenden
Informationen liefern:

- Daten aller Dozenten
der Abteilung „Physik“
- Daten aller Dozenten
mit einem Gehalt von
90000 und mehr

Selektion in SQL

$\sigma_{\beta}(R)$ lässt sich in SQL darstellen als

SELECT * FROM R WHERE β ;

Beispiel $\sigma_{C>6}(R)$:

SELECT * FROM R WHERE C > 6;

6 Abteilung = 'Physik' (Dozent)

→ SELECT * FROM Dozent WHERE Abteilung = 'Physik'

Die Projektion π

Gegeben

- Relation $R(A_1, A_1, \dots, A_n)$
- Liste von Attributen γ
 - Z.B. A_1, A_4

$$\pi_{\gamma}(R) := \{t_{\gamma} | t \in R\}$$

- t_{γ} : Tupel enthalten nur Attribute aus Liste γ

Anschaulich:

- Die Menge aller Tupel aus R , beschränkt auf Attribute aus γ

„auf die Spalten aus γ “

Beispiele Projektion

Relation R :

A	B	C
1	2	3
4	5	6
1	3	8

$\pi_{A,B}(R)$:

A	B
1	2
4	5
1	3

$\pi_A(R)$:

A
1
4

'1' kommt nur 1x vor!

Achtung: Relationen sind Mengen von Tupeln, d.h. es gibt keine doppelten Tupel!

→ anders als in SQL!

! Übung: Projektion

Relation *Dozent*:

ID	Name	Abteilung	Gehalt
10101	Srinivasan	Informatik	65000
12121	Wu	Wirtschaft	90000
15151	Mozart	Musik	40000
22222	Einstein	Physik	95000
32343	El Said	Geschichte	60000
33456	Gold	Physik	87000
45565	Katz	Informatik	75000
58583	Califieri	Geschichte	62000
76543	Singh	Wirtschaft	80000
76766	Crick	Biologie	72000
83821	Brandt	Informatik	92000
98345	Kim	Elektrotechnik	80000

Geben Sie eine relationale Anfrage an, die die folgenden Informationen liefert:

- ID, Name, Gehalt aller Dozenten

$\pi_{ID, Name, Gehalt}(Dozent)$

ID	Name	Gehalt
...

Projektion in SQL

$\pi_{\gamma}(R)$ lässt sich in SQL darstellen als

SELECT γ FROM R ;

Beispiel $\pi_{A,B}(R)$:

SELECT A,B FROM R;

Beispiel Folie 16:

SELECT ID, Name, Gehalt FROM Dozent;

Komposition von relationalen Operatoren

Beispiel:

- Finden Sie die Namen aller Dozenten der Abteilung Physik

$$\pi_{\text{Name}}(\sigma_{\text{Abteilung}=\text{'Physik'}}(\text{Dozent}))$$

ID	Name	Abteilung	Gehalt
...	Gold	Physik	..
...	Einstein	Physik	..

Name
Gold
Einstein

Da das Ergebnis wieder eine Relation ist, können sehr komplexe Ausdrücke erstellt werden!

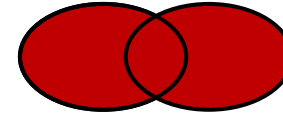
Komposition relationaler Operatoren in SQL

Komposition kann in SQL teilweise in einem Statement ausgedrückt werden.

Beispiel $\pi_{Name}(\sigma_{Abteilung='Physik'}(Dozent))$:

$\swarrow \quad \searrow$
SELECT Name FROM Dozent
WHERE Abteilung = 'Physik';

Die Vereinigung \cup



Gegeben:

- Vereinigungsverträgliche Relationen R, S
 - **Vereinigungsverträgliche** Relationen haben das gleiche Relationsschema
 - Gleicher Grad (Anzahl von Attributen)
 - Gleiche Wertebereiche

$$R \cup S := \{t \mid t \in R \vee t \in S\}$$

Anschaulich:

- Die Menge aller Tupel, die in R oder in S (oder in beiden) sind

Beispiel Vereinigung

Relation R:

A	B	C
1	2	3
4	5	6

Relation S:

A	B	C
7	8	9
4	5	6



$R \cup S$:

A	B	C
1	2	3
4	5	6
7	8	9

Achtung: Keine
Doppelten, weil
Relationen Mengen
sind!

! Übung: Vereinigung

Relation
Kurs:

KursID	GruppenID	Semester	Jahr	Gebäude	Raum	Block
BIO-1001	1	Sommer	2009	Painter	514	B
BIO-301	1	Sommer	2010	Painter	514	A
INF-101	1	Herbst	2009	Packard	101	H
INF-101	1	Frühling	2010	Packard	101	F
CS-190	1	Frühling	2009	Taylor	3128	E
CS-190	2	Frühling	2009	Taylor	3128	A
CS-315	1	Frühling	2010	Watson	120	D
CS-319	1	Frühling	2010	Watson	100	B
CS-319	2	Frühling	2010	Taylor	3128	C
CS-347	1	Herbst	2009	Taylor	3128	A
EE-181	1	Frühling	2009	Taylor	3128	C
FIN-201	1	Frühling	2010	Packard	101	B
HIS-351	1	Frühling	2010	Painter	541	C
MU-199	1	Frühling	2010	Packard	101	D
PHY-101	1	Herbst	2009	Watson	100	A

Geben Sie eine relationale Anfrage an, die die folgenden Informationen liefert:

- KursID aller Kurse, die im Herbst 2009 oder Frühling 2010 stattfanden

(Hinweis: Nutzen Sie dazu Selektion, Projektion und Vereinigung)

Vereinigung in SQL

R U S lässt sich in SQL darstellen als

Anmerkung: Jahr
als CHAR(4)
gespeichert

SELECT * FROM R UNION SELECT * FROM S;

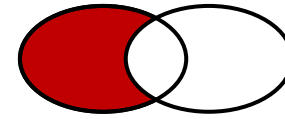
Beispiel Übung Folie 21:

SELECT KursID FROM Kurs
WHERE Jahr = '2009' AND Semester = 'Herbst'

UNION

SELECT KursID FROM Kurs
WHERE Jahr = '2010' AND Semester = 'Frühling';

Die (Mengen-)Differenz –



Gegeben:

- Vereinigungsverträgliche Relationen R, S

$$R - S := \{t | t \in R \wedge t \notin S\}$$

Anschaulich:

- Die Menge aller Tupel, die in R und nicht in S sind

Beispiel Differenz

Relation R:

A	B	C
1	2	3
4	5	6

Relation S:

A	B	C
7 1	8	9
4	5	6

← Nicht
Sum!

$R - S$:

A	B	C
1	2	3

A	B	C
1	2	3

! Übung: Differenz

Relation
Kurs:

KursID	GruppenID	Semester	Jahr	Gebäude	Raum	Block
BIO-1001	1	Sommer	2009	Painter	514	B
BIO-301	1	Sommer	2010	Painter	514	A
INF-101	1	Herbst	2009	Packard	101	H
INF-101	1	Frühling	2010	Packard	101	F
CS-190	1	Frühling	2009	Taylor	3128	E
CS-190	2	Frühling	2009	Taylor	3128	A
CS-315	1	Frühling	2010	Watson	120	D
CS-319	1	Frühling	2010	Watson	100	B
CS-319	2	Frühling	2010	Taylor	3128	C
CS-347	1	Herbst	2009	Taylor	3128	A
EE-181	1	Frühling	2009	Taylor	3128	C
FIN-201	1	Frühling	2010	Packard	101	B
HIS-351	1	Frühling	2010	Painter	541	C
MU-199	1	Frühling	2010	Packard	101	D
PHY-101	1	Herbst	2009	Watson	100	A

Geben Sie eine relationale Anfrage an, die die folgenden Informationen liefert:

- KursID aller Kurse, die im Herbst 2009 aber nicht im Frühling 2010 stattfanden

(Hinweis: Nutzen Sie dazu Selektion, Projektion und Differenz)

Differenz in SQL

$R - S$ lässt sich in SQL darstellen als

SELECT * FROM R EXCEPT SELECT * FROM S;

bzw. in Oracle als

SELECT * FROM R MINUS SELECT * FROM S;

Beispiel Übung Folie 21:

*SELECT KursID FROM Kurs
WHERE Jahr='2009' AND Semester='Herbst'
MINUS
SELECT KursID FROM Kurs
WHERE Jahr='2010' AND Semester='Frühling';*

Das kartesische Produkt \times

Gegeben:

- Zwei Relationen $R(A_1, A_2, \dots, A_n), S(B_1, B_2, \dots, B_m)$
 - nicht notwendigerweise vereinigungsverträglich!

$$R \times S := \{(a_1, a_2, \dots, a_n, b_1, b_2, \dots, b_m) \mid (a_1, a_2, \dots, a_n) \in R \wedge (b_1, b_2, \dots, b_m) \in S\}$$

Anschaulich:

- Die Menge aller möglichen Kombinationen von je einem Tupel aus R mit je einem Tupel aus S
- Im Allgemeinen $n \cdot m$ Tupel!

Berechnung kartesisches Produkt

Für jedes Tupel t_R aus R {

 Für jedes Tupel t_S aus S {

 Verbinde t_R, t_S zu einem neuen Tupel t

 Füge t zur Ergebnisrelation hinzu

 }

}

Beispiel kartesisches Produkt

Relation R :

A	B	C	D
1	2	3	4
4	5	6	7
7	8	9	0

Relation S :

E	F	G
1	2	3
7	8	9

$R \times S$:

A	B	C	D	E	F	G
1	2	3	4	1	2	3
1	2	3	4	7	8	9
4	5	6	7	1	2	3
4	5	6	7	7	8	9
7	8	9	0	1	2	3
7	8	9	0	7	8	9

Attributsnamen beim kartesisches Produkt

Streng genommen müssen zur Berechnung von $R \times S$ die Namen der Attribute in R und S verschieden sein

- Man behilft sich durch Voranstellen der Relationsnamen, z.B. $R.a_1, S.b_2, \dots$
 - Konvention: Wenn ein Attributsnamen eindeutig ist, kann Relationsnamen weggelassen werden
- Schwierig bei $R \times R$, komplexeren Ausdrücken
 - Lösung siehe unten!

z.B. $R: \begin{array}{|c|c|c|c|} \hline A & B & C & D \\ \hline \end{array}$ $S: \begin{array}{|c|c|c|} \hline A & B & C \\ \hline \end{array}$

$R \times S:$

$R.A$	$R.B$	$R.C$	$R.D$	$S.A$	$S.B$

D
↓

! Übung: Kartesisches Produkt (1)

Relation *Unterrichtet*:

ID	KursID	GruppenID	Semester	Jahr
10101	CS-101	1	Herbst	2009
10101	CS-315	1	Frühling	2010
10101	CS-347	1	Herbst	2009
12121	FIN-201	1	Frühling	2010
15151	MU-199	1	Frühling	2010
22222	PHY-101	1	Herbst	2009
32343	HIS-351	1	Frühling	2010
45565	CS-101	1	Frühling	2010
45565	CS-319	1	Frühling	2010
76766	BIO-101	1	Sommer	2009
76766	BIO-301	1	Sommer	2010
76766	BIO-301	1	Sommer	2010

Relation *Dozent*:

ID	Name	Abteilung	Gehalt
10101	Srinivasan	Informatik	65000
12121	Wu	Wirtschaft	90000
15151	Mozart	Musik	40000
22222	Einstein	Physik	95000
32343	El Said	Geschichte	60000
33456	Gold	Physik	87000
45565	Katz	Informatik	75000
58583	Califieri	Geschichte	62000
76543	Singh	Wirtschaft	80000
76766	Crick	Biologie	72000
83821	Brandt	Informatik	92000
98345	Kim	Elektrotechnik	80000

Bestimmen Sie *Unterrichtet* × *Dozent*- Was sagt das Ergebnis aus?

! Übung: Kartesisches Produkt (2)

Geben Sie eine relationale Anfrage an, die die folgenden Informationen liefert:

- Namen aller ~~Physik~~ Dozenten zusammen mit den KursIDs der Kurse, die sie jeweils unterrichten

(Hinweis: Nutzen Sie dazu kartesisches Produkt, Selektion und Projektion)

$$\pi_{\text{Name, KursID}} \left(\sigma_{\text{Dozent.ID} = \text{Unterrichtf.ID}} \left(\text{Unterrichtf} \times \text{Dozent} \right) \right)$$

Kartesisches Produkt in SQL

$R \cup S$ lässt sich in SQL darstellen als

SELECT * FROM R, S ;

Beispiel Übung Folie 32:

*SELECT Name, KursID FROM Unterrichtet, Dozent
WHERE Unterrichtet.ID = Dozent.ID;*

Die Umbenennung ρ

Gegeben:

- Eine Relation $R(A_1, A_2, \dots, A_n)$

Umbenennung der Relation

- $\rho_S(R)$ ist die Umbenennung der Relation R in S

Umbenennung der Relation und ihrer Attribute

- $\rho_{S(B_1, B_2, \dots, B_n)}(R)$ ist die Umbenennung der Relation $R(A_1, A_2, \dots, A_n)$ in $S(B_1, B_2, \dots, B_n)$

Beispiel Umbenennung

Relation R :

A	B	C	D
1	2	3	4
4	5	6	7
7	8	9	0

$\rho_S(R)$:

Relation S :

A	B	C	D
1	2	3	4

$\rho_{S(E,F,G,D)}(R)$:

Relation S :

E	F	G	D
1	2	3	4
4	5	6	7
7	8	9	0

! Übung: Umbenennung (1)

Relation
Kurs:

KursID	GruppenID	Semester	Jahr	Gebäude	Raum	Block
BIO-1001	1	Sommer	2009	Painter	514	B
BIO-301	1	Sommer	2010	Painter	514	A
INF-101	1	Herbst	2009	Packard	101	H
INF-101	1	Frühling	2010	Packard	101	F
CS-190	1	Frühling	2009	Taylor	3128	E
CS-190	2	Frühling	2009	Taylor	3128	A
CS-315	1	Frühling	2010	Watson	120	D
CS-319	1	Frühling	2010	Watson	100	B
CS-319	2	Frühling	2010	Taylor	3128	C
CS-347	1	Herbst	2009	Taylor	3128	A
EE-181	1	Frühling	2009	Taylor	3128	C
FIN-201	1	Frühling	2010	Packard	101	B
HIS-351	1	Frühling	2010	Painter	541	C
MU-199	1	Frühling	2010	Packard	101	D
PHY-101	1	Herbst	2009	Watson	100	A

Geben Sie eine relationale Anfrage an, die die folgenden Informationen liefert:

- KursID aller Kurse, die im Herbst 2009 und im Frühling 2010 stattfanden

(Hinweis: Nutzen Sie dazu Umbenennung, kartesisches Produkt, Selektion, und Projektion)

! Übung: Umbenennung (2)

Relation *Dozent*:

ID	Name	Abteilung	Gehalt
10101	Srinivasan	Informatik	65000
12121	Wu	Wirtschaft	90000
15151	Mozart	Musik	40000
22222	Einstein	Physik	95000
32343	El Said	Geschichte	60000
33456	Gold	Physik	87000
45565	Katz	Informatik	75000
58583	Califieri	Geschichte	62000
76543	Singh	Wirtschaft	80000
76766	Crick	Biologie	72000
83821	Brandt	Informatik	92000
98345	Kim	Elektrotechnik	80000

Geben Sie eine relationale Anfrage an, die die folgenden Informationen liefert:

- Höchstes Gehalt aller Dozenten der Hochschule
(Hinweis: Nutzen Sie dazu *Umbenennung*, *kartesisches Produkt*, *Selektion*, *Projektion* und *Differenz*)

Umbenennung in SQL

$\rho_{S(B_1, B_2, \dots, B_n)}(R)$ lässt sich in SQL darstellen als

```
SELECT  A1 AS B1, A2 AS B2, ..., An AS Bn  
        FROM R AS S;
```

Bzw. in Oracle

```
SELECT  A1 AS B1, A2 AS B2, ..., An AS Bn  
        FROM R S;
```

Beispiel $\rho_{S(E, F, G, D)}(R)$:

```
SELECT A AS E, B AS F, C AS G, D  
FROM R S;
```



Formale Definition der relationalen Algebra

Basis-Ausdrücke

- Relation (in Datenbank)
- Konstante Relation, z.B. $\{(22222, 'Einstein', 'Physik', 95000), \dots\}$

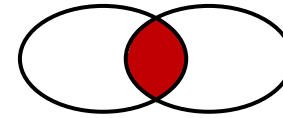
Für relationale Ausdrücke E_1, E_2 sind folgendes ebenfalls relationale Ausdrücke:

- Vereinigung $E_1 \cup E_2$
- Differenz $E_1 - E_2$
- Kartesisches Produkt $E_1 \times E_2$
- Selektion $\sigma_P(E_1)$ mit P Prädikate auf Attributen von E_1
- Projektion $\pi_L(E_1)$ mit L Liste von Attributen von E_1
- Umbenennung $\rho_S(E_1)$ mit S neuer Name von E_1

Zusätzliche Operatoren

- Mit den bisher gelernten Grundoperatoren lässt sich jede Anfrage der relationalen Algebra ausdrücken
- Manche Anfragen sind aber etwas „sperrig“
- Zusätzliche Operatoren vereinfachen die Darstellung von Anfragen

Die Schnittmenge \cap



Gegeben:

- Vereinigungsverträgliche Relationen R, S

$$R \cap S := \{t | t \in R \wedge t \in S\}$$

Anschaulich:

- Die Menge aller Tupel, die in R und in S sind

Beispiel Schnittmenge

Relation R :

A	B	C
1	2	3
4	5	6

Relation S :

A	B	C
7	8	9
4	5	6

$R \cap S$:

A	B	C
4	5	6

! Übung: Schnittmenge (1)

Relation
Kurs:

KursID	GruppenID	Semester	Jahr	Gebäude	Raum	Block
BIO-1001	1	Sommer	2009	Painter	514	B
BIO-301	1	Sommer	2010	Painter	514	A
INF-101	1	Herbst	2009	Packard	101	H
INF-101	1	Frühling	2010	Packard	101	F
CS-190	1	Frühling	2009	Taylor	3128	E
CS-190	2	Frühling	2009	Taylor	3128	A
CS-315	1	Frühling	2010	Watson	120	D
CS-319	1	Frühling	2010	Watson	100	B
CS-319	2	Frühling	2010	Taylor	3128	C
CS-347	1	Herbst	2009	Taylor	3128	A
EE-181	1	Frühling	2009	Taylor	3128	C
FIN-201	1	Frühling	2010	Packard	101	B
HIS-351	1	Frühling	2010	Painter	541	C
MU-199	1	Frühling	2010	Packard	101	D
PHY-101	1	Herbst	2009	Watson	100	A

Geben Sie eine relationale Anfrage an, die die folgenden Informationen liefert:

- KursID aller Kurse, die im Herbst 2009 und Frühling 2010 stattfanden

(Hinweis: Nutzen Sie dazu Selektion, Projektion und Schnittmenge)

Übung: Schnittmenge (2)

Warum ist die Schnittmenge ein zusätzlicher Operator, d.h. wie lässt sie sich mit den Grundoperatoren darstellen?

Schnittmenge in SQL

$R \cap S$ lässt sich in SQL darstellen als

```
SELECT * FROM R INTERSECT SELECT * FROM S;
```

Beispiel Übung Folie 43:

```
SELECT KursID FROM Kurs
WHERE Semester = 'Herbst' AND Jahr = '2009'
INTERSECT
SELECT KursID FROM Kurs
WHERE Semester = 'Frühling' AND Jahr = '2010'
```

Verbund-Operatoren (Join-Operatoren)

Anfragen in Zusammenhang mit kartesischem Produkt \times erfordern häufig Kombinationen mit Selektion σ

- Kann „sperrig“ werden

Beispiel Folie 32: $\text{Dozent.ID} = \text{Unterrichtet.ID}$ durch Selektion sichergestellt

$\pi_{\text{Name, Kurs ID}} (\sigma_{\text{Dozent.ID} = \text{Unterrichtet.ID}} (\text{Unterrichtet} \times \text{Dozent}))$

- Vereinfachung benötigt!

Der natürliche Verbund \bowtie (Natural Join)

Kombination aus

- Kartesischem Produkt von zwei Relationen
- Selektion der Tupel in Ergebnis, die gleiche Werte in gemeinsamen Attributen beider Relationen haben
- Löschung doppelter Attribute

Definition Natural Join \bowtie

Gegeben:

- Relationen $R(A_1, \dots, A_n, C_1, \dots, C_l)$ und $S(B_1, \dots, B_m, C_1, \dots, C_l)$
- C_1, \dots, C_l sind gemeinsame Attribute

$$R \bowtie S := \pi_{A_1, \dots, A_n, C_1, \dots, C_l, B_1, \dots, B_m} \left(\sigma_{R.C_1=S.C_1 \wedge \dots \wedge R.C_l=S.C_l} (R \times S) \right)$$

Konvention: Reihenfolge der Attribute

- Erst Attribute, die nur in R sind
- Dann Attribute, die in R und in S sind
- Zuletzt Attribute, die nur in S sind

Beispiel Natural Join

Relation R:

A	B	C	D
1	2	3	4
4	5	6	7
7	8	9	0

Relation S:

A	F	G
1	2	3
7	8	9

$R \bowtie S$:

B	C	D	A	F	G
2	3	4	1	2	3
8	9	0	7	8	9

! Übung: Natural Join (1)

Relation *Unterrichtet*:

ID	KursID	GruppenID	Semester	Jahr
10101	CS-101	1	Herbst	2009
10101	CS-315	1	Frühling	2010
10101	CS-347	1	Herbst	2009
12121	FIN-201	1	Frühling	2010
15151	MU-199	1	Frühling	2010
22222	PHY-101	1	Herbst	2009
32343	HIS-351	1	Frühling	2010
45565	CS-101	1	Frühling	2010
45565	CS-319	1	Frühling	2010
76766	BIO-101	1	Sommer	2009
76766	BIO-301	1	Sommer	2010
76766	BIO-301	1	Sommer	2010

Relation *Dozent*:

ID	Name	Abteilung	Gehalt
10101	Srinivasan	Informatik	65000
12121	Wu	Wirtschaft	90000
15151	Mozart	Musik	40000
22222	Einstein	Physik	95000
32343	El Said	Geschichte	60000
33456	Gold	Physik	87000
45565	Katz	Informatik	75000
58583	Califieri	Geschichte	62000
76543	Singh	Wirtschaft	80000
76766	Crick	Biologie	72000
83821	Brandt	Informatik	92000
98345	Kim	Elektrotechnik	80000

Übung: Natural Join (1, Fortsetzung)

Geben Sie eine relationale Anfrage an, die die folgenden Informationen liefert:

- Namen aller Dozenten mit den Kurs-IDs, die sie unterrichten

(Hinweis: Nutzen Sie dazu Natural Join und Projektion)

Anmerkung Natural Join

Natural Join ist **assoziativ**, d.h. für Relationen R, S, T gilt

$$\begin{aligned} R \bowtie (S \bowtie T) \\ &= (R \bowtie S) \bowtie T \\ &= R \bowtie S \bowtie T \end{aligned}$$

! Übung: Natural Join (2)

Relation *Kursinhalt*:

KursID	Titel	Abteilung	Creditpoints
BIO-101	Einführung in die Biologie	Biologie	4
BIO-301	Genetik	Biologie	4
BIO-399	Computational Biology	Biologie	3
CS-101	Einführung in die Informatik	Informatik	4
CS-190	Spieleentwurf	Informatik	4
CS-315	Robotik	Informatik	3
CS-319	Bildverarbeitung	Informatik	3
CS-347	Datenbanken	Informatik	3
EE-181	Digitale Systeme	Elektrotechnik	3
FIN-201	Investment Banking	Wirtschaft	3
HIS-351	Weltgeschichte	Geschichte	3
MU-199	Musikvideoproduktion	Musik	3
PHY-101	Physikalische Prinzipien	Physik	4

Übung: Natural Join (2, Fortsetzung)

Geben Sie eine relationale Anfrage an, die die folgenden Informationen liefert:

- Namen aller Dozenten mit Kursnamen ihrer Kurse

(Hinweis: Nutzen Sie dazu Natural Join und Projektion)

Natural Join in SQL

$R \bowtie S$ lässt sich in SQL darstellen als

```
SELECT * FROM R NATURAL INNER JOIN S;  
SELECT * FROM R NATURAL JOIN S;
```

Beispiel Übung Folie 51:

```
SELECT Name, KursID FROM Unterricht  
NATURAL INNER JOIN Dozent;
```

Der Theta Join (θ -Join)

Gegeben:

- Relationen R, S
- Prädikat Θ auf den Attributen von R, S

$$R \bowtie_{\Theta} S := \sigma_{\Theta}(R \times S)$$

Anschaulich:

- Verallgemeinerung des Natural Joins für beliebige Prädikate Θ

Beispiel Theta Join

Relation R:

A	B	C	D
1	2	3	4
4	5	6	7
7	8	9	0

Relation S:

E	F	G
1	2	3
7	8	9

$R \bowtie_{R.A \neq S.E} S$:

R.A	R.B	R.C	R.D	S.E	S.F	S.G
1	2	3	4	7	8	9
4	5	6	7	1	2	3
4	5	6	7	7	8	9
7	8	9	0	1	2	3

Übung: Theta Join

Geben Sie eine relationale Anfrage an, die die folgenden Informationen liefert:

- Namen aller Dozenten mit KursIDs der Kurse, die sie nicht unterrichten!

(Hinweis: Nutzen Sie dazu Theta Join und Projektion; die Relationen finden Sie auf Folie 50)

Theta Join in SQL

$R \bowtie_{\Theta} S$ lässt sich in SQL darstellen als

SELECT * FROM R INNER JOIN S ON Θ ;
SELECT * FROM R JOIN S ON Θ ;

Beispiel $R \bowtie_{R.A \neq S.E} S$:

*SELECT * FROM R INNER JOIN S
ON R.A <> S.E;*

Outer Join Operatoren

Bei Join Operationen gehen Informationen verloren

- **Beispiel:** In $Dozent \bowtie Unterrichtet$ die Dozenten, die keinen Kurs unterrichten

Outer Join Operationen nehmen zusätzliche Tupel in Ergebnisse auf:

- **Left Outer Join** $R \bowtie_{\leftarrow} S$: Tupel aus R , die Join-Bedingung nicht erfüllen
- **Right Outer Join** $R \bowtie_{\rightarrow} S$: Tupel aus S , die Join-Bedingung nicht erfüllen
- **Full Outer Join** $R \bowtie_{\leftarrow\rightarrow} S$: Tupel aus R und S , die Join-Bedingung nicht erfüllen

Outer Join Operationen gibt es sowohl als **Natural Outer Join** (z.B. $R \bowtie S$) als auch als **Theta Outer Join** (z.B. $R \bowtie_{\theta} S$)

- Zur Abgrenzung spricht auch von **Inner Join**, wenn kein Outer Join gemeint ist

Join-Beid: Natural

Beispiele Outer Join (1)

Relation R:

A	B	C	D
1	2	3	4
4	5	6	7
7	8	9	0

Relation S:

A	F	G
1	2	3
7	8	9
2	3	4

$R \bowtie S$: (Natural Left Outer Join)

B	C	D	A	F	G
2	3	4	1	2	3
8	9	0	7	8	9
5	6	7	4	Null	Null

$R \bowtie S$: (Natural Right Outer Join)

B	C	D	A	F	G
2	3	4	1	2	3
8	9	0	7	8	9
Null	Null	Null	2	3	4

Beispiele Outer Join (2)

Relation R:

A	B	C	D
1	2	3	4
4	5	6	7
7	8	9	0

Relation S:

A	F	G
1	2	3
7	8	9
2	3	4

$R \bowtie S$:

B	C	D	A	F	G
2	3	4	1	2	3
3	9	0	7	8	9
5	6	7	4	Null	Null
Null	Null	Null	2	3	4

Untersch. Kart. Produkt: Auffüllen mit Null statt alle Kombinationen!

! Übung: Outer Join (1)

Relation *Unterrichtet*:

ID	KursID	GruppenID	Semester	Jahr
10101	CS-101	1	Herbst	2009
10101	CS-315	1	Frühling	2010
10101	CS-347	1	Herbst	2009
12121	FIN-201	1	Frühling	2010
15151	MU-199	1	Frühling	2010
22222	PHY-101	1	Herbst	2009
32343	HIS-351	1	Frühling	2010
45565	CS-101	1	Frühling	2010
45565	CS-319	1	Frühling	2010
76766	BIO-101	1	Sommer	2009
76766	BIO-301	1	Sommer	2010
76766	BIO-301	1	Sommer	2010

Relation *Dozent*:

ID	Name	Abteilung	Gehalt
10101	Srinivasan	Informatik	65000
12121	Wu	Wirtschaft	90000
15151	Mozart	Musik	40000
22222	Einstein	Physik	95000
32343	El Said	Geschichte	60000
33456	Gold	Physik	87000
45565	Katz	Informatik	75000
58583	Califieri	Geschichte	62000
76543	Singh	Wirtschaft	80000
76766	Crick	Biologie	72000
83821	Brandt	Informatik	92000
98345	Kim	Elektrotechnik	80000

Übung: Outer Join (1, Fortsetzung)

Geben Sie eine relationale Anfrage an, die die folgenden Informationen liefert:

- Namen aller Dozenten und falls vorhanden die KursIDs ihrer Kurse

(Hinweis: Nutzen Sie dazu Natural Outer Join und Projektion)

! Übung: Outer Join (2)

Wie kann der Natural Left Outer Join mit Hilfe von Natural Join \bowtie und den Grundoperatoren ausdrücken?

Outer Join in SQL (1)

$R \bowtie S$, $R \ltimes S$, $R \rhd S$ lässt sich in SQL darstellen als

```
SELECT * FROM R NATURAL LEFT OUTER JOIN S;  
SELECT * FROM R NATURAL RIGHT OUTER JOIN S;  
SELECT * FROM R NATURAL FULL OUTER JOIN S;
```

Beispiel Folie 64:

```
SELECT Name, KursID FROM Unterrichtet NATURAL RIGHT  
OUTER JOIN Dozent;
```

Outer Join in SQL (2)

$R \bowtie_{\theta} S$, $R \ltimes_{\theta} S$, $R \rhd_{\theta} S$ lässt sich in SQL darstellen als

SELECT * FROM R LEFT OUTER JOIN S ON θ ;

SELECT * FROM R RIGHT OUTER JOIN S ON θ ;

SELECT * FROM R FULL OUTER JOIN S ON θ ;

*SELECT Name, KursID FROM Unterrichtet RIGHT OUTER
JOIN Dozent ON Unterrichtet.ID = Dozent.ID;*

Die Zuweisung \leftarrow

Relationale Anfragen können sehr komplex sein

- Manchmal sind Zwischenschritte hilfreich
- Zuweisung \leftarrow ermöglicht das Speichern von Zwischenergebnissen

$\equiv \sigma_{R.A \neq S.E} (R \times S)$

Beispiel: Wir können $R \bowtie_{R.A \neq S.E} S$ (s. Folie 48) schreiben als

CREATE TABLE
TEMP AS ...

$Temp \leftarrow R \times S$

$Result \leftarrow \sigma_{R.A \neq S.E}(Temp)$ WITH TEMP AS ...
(temporär)

Aggregatsfunktionen

Aggregatsfunktionen berechnen einen einzelnen Wert aus einer Menge von Werten

Beispiel: Um z.B. das durchschnittliche Gehalt aller Dozenten zu bestimmen, kann man schreiben:

$$G_{\text{avg}(\text{Gehalt})}(\text{Dozent})$$

Mögliche Aggregatsfunktionen sind *avg*, *min*, *max*, *sum*, *count*, *avg-distinct*, *sum-distinct*, *count-distinct*

- **-distinct* bezeichnet die Anwendung der Aggregatsfunktion auf verschiedene Elemente

$\text{avg}(\text{Gehalt})$
45000


Achtung: Ergebnis ist eine Relation!

Beispiel Aggregatsfunktionen

Relation *Dozent*:

ID	Name	Abteilung	Gehalt
10101	Srinivasan	Informatik	65000
12121	Wu	Wirtschaft	90000
15151	Mozart	Musik	40000
22222	Einstein	Physik	95000
32343	El Said	Geschichte	60000
33456	Gold	Physik	87000
45565	Katz	Informatik	75000
58583	Califieri	Geschichte	62000
76543	Singh	Wirtschaft	80000
76766	Crick	Biologie	72000
83821	Brandt	Informatik	92000
98345	Kim	Elektrotechnik	80000

$G_{\max(\text{Gehalt})}(\text{Dozent})$:



max(Gehalt)
95000

! Übung: Aggregatsfunktion

Relation *Dozent*:

ID	Name	Abteilung	Gehalt
10101	Srinivasan	Informatik	65000
12121	Wu	Wirtschaft	90000
15151	Mozart	Musik	40000
22222	Einstein	Physik	95000
32343	El Said	Geschichte	60000
33456	Gold	Physik	87000
45565	Katz	Informatik	75000
58583	Califieri	Geschichte	62000
76543	Singh	Wirtschaft	80000
76766	Crick	Biologie	72000
83821	Brandt	Informatik	92000
98345	Kim	Elektrotechnik	80000

Geben Sie eine relationale Anfrage an, die die folgenden Informationen liefert:

- Summer der Gehälter aller Dozenten der Hochschule
(Hinweis: Nutzen Sie dazu die Aggregatsfunktion *sum*)

G_{sum(Gehalt)} (Dozent)

Aggregatfunktionen in SQL

$G_{F(A)}(R)$ für Aggregatfunktion F lässt sich in SQL darstellen als

SELECT $F(A)$ **FROM** R ;

Beispiel $G_{\max(\text{Gehalt})}(\text{Dozent})$:

SELECT max(Gehalt) FROM Dozent,

Aggregatsfunktionen und Gruppierung

Aggregatsfunktionen können auf **Gruppen** von Tupeln angewendet werden

Beispiel: Um das durchschnittliche Gehalt pro Abteilung zu bestimmen (also Durchschnittsgehälter **gruppiert** nach gleichen Werten für Attribut *Abteilung*), kann man schreiben:

Abteilung **G** avg(Gehalt)(*Dozent*)

Beispiel Gruppierung

Relation *Dozent*:

ID	Name	Abteilung	Gehalt
10101	Srinivasan	Informatik	65000
12121	Wu	Wirtschaft	90000
15151	Mozart	Musik	40000
22222	Einstein	Physik	95000
32343	El Said	Geschichte	60000
33456	Gold	Physik	87000
45565	Katz	Informatik	75000
58583	Califieri	Geschichte	62000
76543	Singh	Wirtschaft	80000
76766	Crick	Biologie	72000
83821	Brandt	Informatik	92000
98345	Kim	Elektrotechnik	80000

Abteilung **G** avg(Gehalt) (*Dozent*) :

Abteilung	avg(Gehalt)
Informatik	< Ø blau >
Wirtschaft	< Ø grün >
Musik	40000
Physik	< Ø rot >
⋮	⋮

! Übung: Gruppierung

Relation *Kursinhalt*:

KursID	Titel	Abteilung	Creditpoints
BIO-101	Einführung in die Biologie	Biologie	4
BIO-301	Genetik	Biologie	4
BIO-399	Computational Biology	Biologie	3
CS-101	Einführung in die Informatik	Informatik	4
CS-190	Spieleentwurf	Informatik	4
CS-315	Robotik	Informatik	3
CS-319	Bildverarbeitung	Informatik	3
CS-347	Datenbanken	Informatik	3
EE-181	Digitale Systeme	Elektrotechnik	3
FIN-201	Investment Banking	Wirtschaft	3
HIS-351	Weltgeschichte	Geschichte	3
MU-199	Musikvideoproduktion	Musik	3
PHY-101	Physikalische Prinzipien	Physik	4

Geben Sie eine relationale Anfrage an, die die folgenden Informationen liefert:

- Wenigste Creditpoints aller Kurse gruppiert nach Abteilungen

(Hinweis: Nutzen Sie dazu Gruppierung und die Aggregatsfunktion min)

Abteilungen ^G min (Creditp.) (Kursinhalt)

Verallgemeinerung Aggregatsfunktionen (1)

Im Allgemeinen ist die Schreibweise:

$$G_1, G_2, \dots, G_n \mathbf{G}_{F_1(A_1), F_2(A_2), \dots, F_m(A_m)}(R)$$

Dabei ist

R	eine Relation
G_1, G_2, \dots, G_n	eine Liste von Attributnamen
F_i	Aggregationsfunktionen aus <i>avg</i> , <i>min</i> , <i>max</i> , <i>sum</i> , <i>count</i> , <i>avg-distinct</i> , <i>sum-distinct</i> , <i>count-distinct</i>
A_i	Attributnamen

Verallgemeinerung Aggregatsfunktionen (2)

Aggregatfunktion

$$G_1, G_1, \dots, G_n \mathbf{G}_{F_1(A_1), F_2(A_2), \dots, F_m(A_m)}(R)$$

wird jeweils über alle Attribute ausgewertet, die gleiche Werte (g_1, g_2, \dots, g_n) für G_1, G_2, \dots, G_n haben

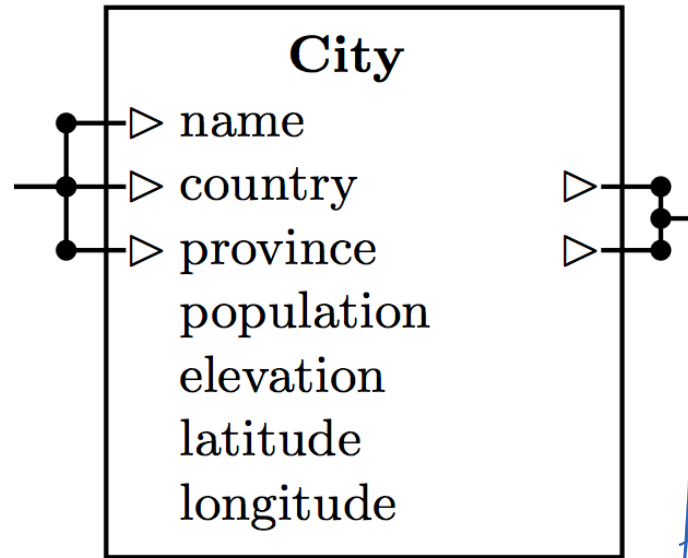
- Man spricht hier von **Gruppieren** der Attribute
- (g_1, g_2, \dots, g_n) wird als **Gruppe** bezeichnet
- Ergebnis für jede Gruppe (g_1, g_2, \dots, g_n) ist $(g_1, g_2, \dots, g_n, a_1, a_2, \dots, a_m)$, wobei $a_i := F_i(A_i)$ Ergebnis der Auswertung der Aggregatsfunktion F_i auf einem Attribut für alle Tupel der Gruppe ist



Beispiel

Verallgemeinerung Aggregatsfunktionen

Relation *City*:



Relationaler Ausdruck für
„Durchschnittliche
Bevölkerung und maximale
Höhe aller Städte gruppiert
nach Land und Provinz“:

Country, Province \mathcal{G} $\text{avg}(\text{Population}), \text{max}(\text{Elevation})$ (*City*)

Country	Province	avg(Population)	max(Elevation)
USA	Texas	400000	3500
USA	California	500000	4200
D	Bay	300000	1200



Aggregatfunktionen und Gruppierung in SQL

G_1, G_2, \dots, G_n $\mathbf{G}_{F_1(A_1), F_2(A_2), \dots, F_m(A_m)}(R)$ für Aggregatfunktionen F_1, F_2, \dots, F_m lässt sich in SQL darstellen als

SELECT $F_1(A_1), F_2(A_2), \dots, F_m(A_m)$ **FROM** R
GROUP BY G_1, G_2, \dots, G_n ;

Beispiel Folie 76:

*SELECT AVG(Population), Max(ELEVATION)
FROM City GROUP BY Country, Province;*

Zusammenfassung Kapitel 5

Die relationale Algebra ermöglicht die Formulierung von Anfragen auf Relationen (Tabellen)

- Mit Grundoperatoren lassen sich alle auf Relationen möglichen Anfragen formulieren
- Zusätzliche Operatoren „sparen“ Schreibarbeit

Relationale Operatoren liefern immer Relationen zurück

- Lassen sich beliebig schachteln

SQL bildet den Umfang der relationalen Algebra ab



Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

Kapitel 6 Einführung in SQL

Vorlesung Datenbanken und Informationssysteme I
Sommersemester 2017

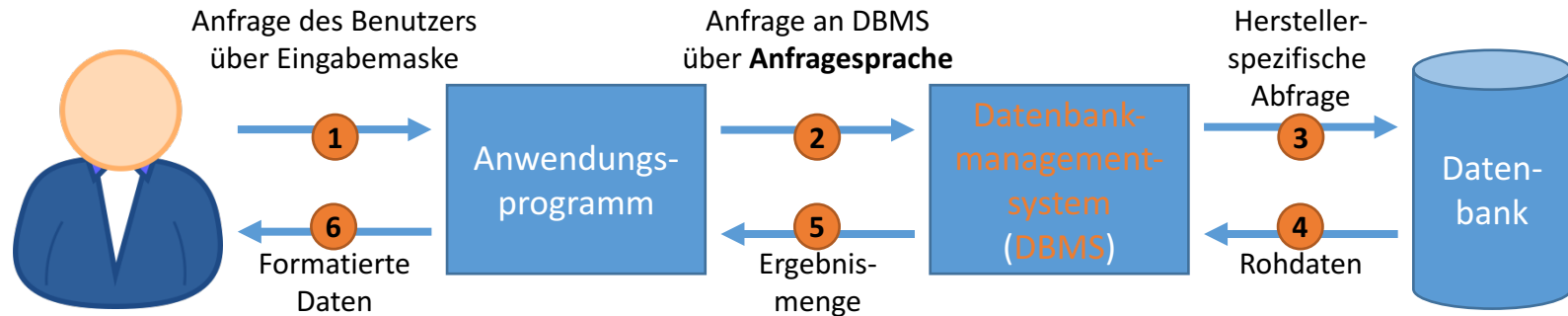
Prof. Dr. rer. nat. Oliver P. Waldhorst

Inhalte dieses Kapitels

- Übersicht über Geschichte & Funktionalität von SQL
- Einführung in Datenbank-Management-System ORACLE und SQL-Developer
- Datentypen in SQL
- Anlegen / modifizieren / löschen von Tabellen
- Einfügen / aktualisieren / löschen von Datensätzen
- Grundlegende Beschränkungen / Constraints für Tabellen

Diese Vorlesung beschäftigt sich mit Datenbanken (vgl. Kapitel 1)

- Definition: Eine **Datenbank** ist eine geordnete, selbstbeschreibende Sammlung von Daten, die miteinander in Beziehung stehen
- Ablauf des Zugriffs eines Anwenders auf die Datenbank:



- Als Anfragesprache hat sich ANSI-Standard **SQL** („Structured Query Language“) etabliert
- Alle Kommunikation mit DBMS findet i.d.R. mit Hilfe von SQL statt
 - Einige Hersteller bieten auch zusätzlich Application Programming Interfaces (APIs)

SQL-Übersicht

SQL ist die gebräuchlichste unter den Datenbank-Anfragesprachen

- ... eigentlich viel mehr als eine **Anfrage**-Sprache (Strukturierung, Anlage, Modifikation von Daten, Zugriffsrechten, ...)

Auch wenn SQL ein Standard ist, unterscheiden sich die Umsetzungen in einzelnen DBMS-Produkten

- Oracle, MySQL, DB2, PostgreSQL, MSSQLServer, ...

Ziel der Vorlesung:

- Verständnis der grundlegenden Konzepte und Konstrukte
- Keine Vollständige Referenz!
 - Gute Übersicht findet sich z.B. unter <https://www.w3schools.com/sql/default.asp>



Geschichte von SQL [Silberschatz et al., 2010]

- Ursprüngliche Version als Sequel von IBM im System R Projekt in den 1970er Jahren eingeführt
 - Name wurde später auf **SQL (Structured Query Language)** geändert
- 1986 wurde von *American Standards Institute (ANSI)* und *International Organization for Standardization (ISO)* ein SQL-Standard veröffentlicht, Bezeichnung SQL-86
- Später folgten SQL-89, SQL-92, SQL:1999, ..., SQL:2011
- Umsetzung der Standards hängt von Produkt ab
 - Die Vorlesung behandelt Konzepte \geq SQL-92
 - Ggf. in Anleitung des jeweiligen Produkts schauen

Tabellen in SQL

SQL legt Daten in **Tabellen (Tables)** ab

- Feste Anzahl von Spalten (Attributen)
- Spalten haben eindeutige Namen
- Spalten haben eindeutige Datentypen

Beispiel:

Abteilungsnummer	Abteilungsname	Abteilungsleiter
1	Geschäftsführung	Meier
2	Vertrieb	Müller
3	Programmierung	Schneider
4	Verwaltung	Schuster

Spalte / Feld / Attribut

Zeile / Datensatz

! Was sind hier die Datentypen?

Teile von SQL

Data Definition Language (DDL)

- Anlegen von und modifizieren von **Tabellen (Tables)**
- Spezifikation für **Bedingungen (Constraints)** für die in den Tabellen gespeicherten Daten

Data Manipulation Language (DML)

- Einfügen, Modifizieren, Verändern von **Datensätzen** in Tabellen
- Auslesen von **Informationen** (Daten im Zusammenhang, vgl. Kapitel 1) aus den Tabellen

Teile von SQL (2)

Data Control Language (DCL)

- Zugriffsrechte für Daten verwalten

Außerdem:

- Definition von angepassten **Sichten (Views)** auf Tabellen
- Zusammenfassen von Operationen zu **Transaktionen**
- *Einbettung in Programmiersprachen wie C, C++, Java, ...*

Schreibweise von SQL-Befehlen

SQL unterscheidet keine Groß- und Kleinschreibung

- Konventionen
 - Schlüsselworte werden **GROSS** geschrieben
 - Bezeichner für Tabelle, Attribute, ... werden im `CamelCase` geschrieben

Kommentare

- Einzeilige Kommentare beginnen mit „--“
- Mehrzeilige Kommentare beginnen mit „/*“ und enden mit „*/“

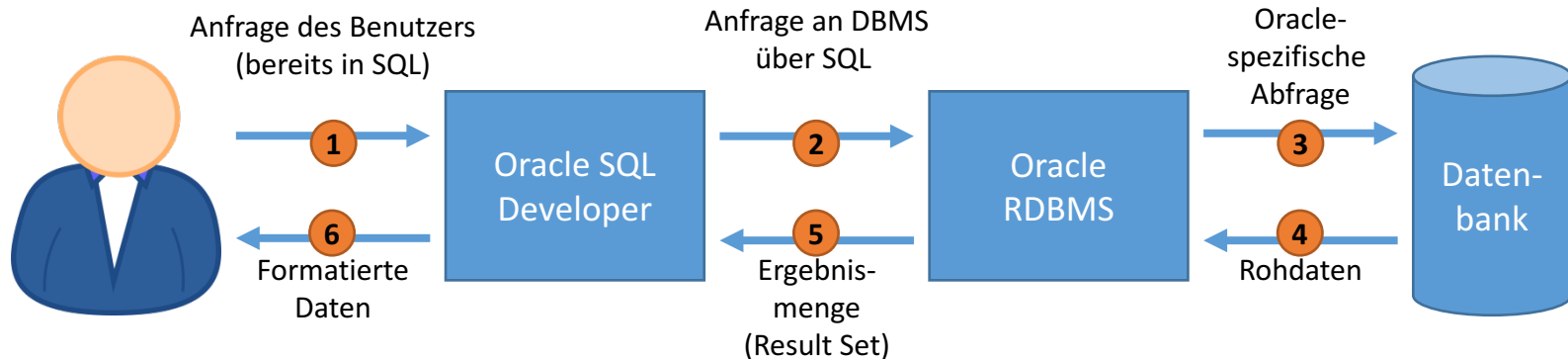
Unser DBMS: Oracle

Für DB1 Übung (und andere Lehrveranstaltungen) steht das RDBMS **Oracle** zur Verfügung

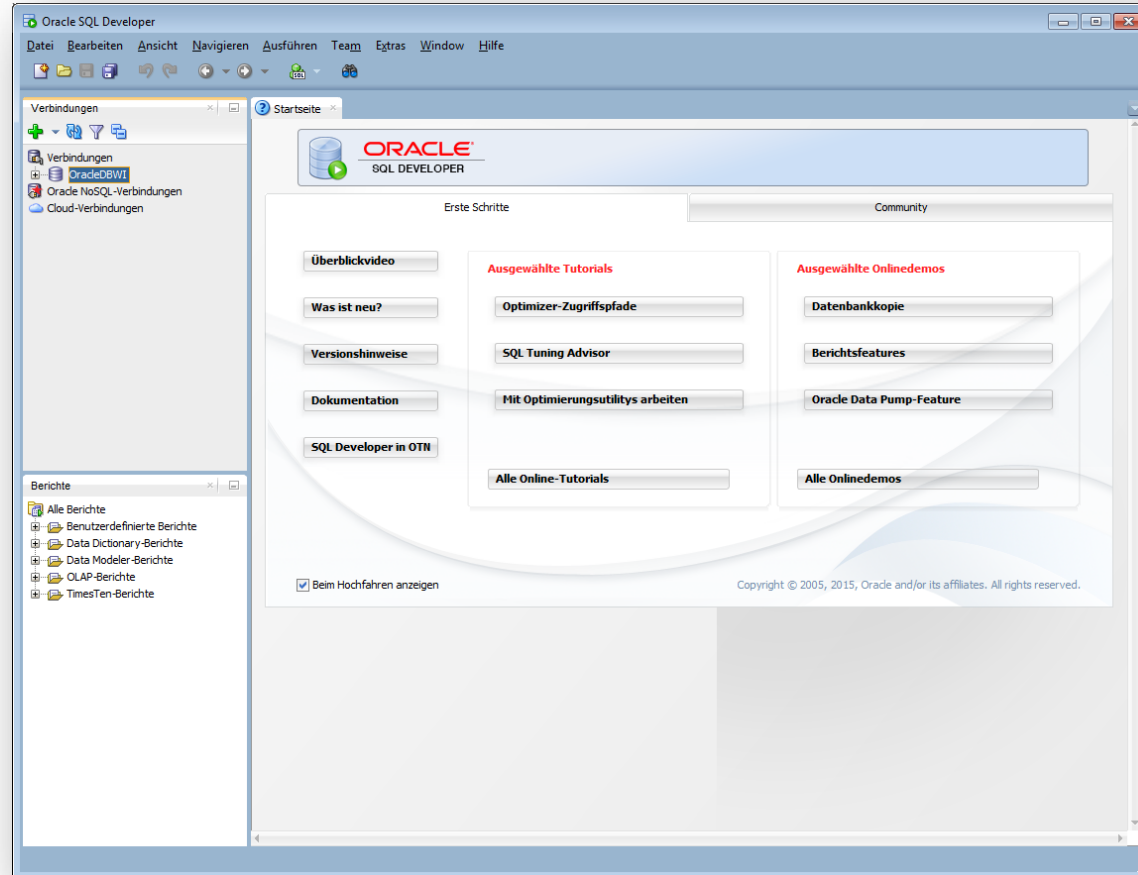
- Das „R“ in „RDBMS“ erklären wir in den nächsten Vorlesungen

Wir empfehlen als Anwendungsprogramm **Oracle SQL Developer**

- Ist auf den Pool-Rechnern installiert



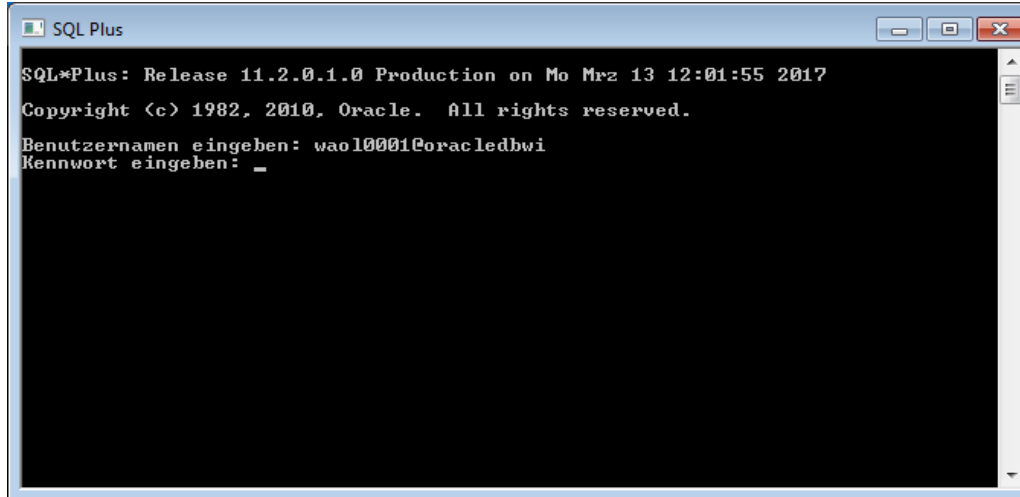
Oracle SQL Developer



Oracle-Konto und Passwort

Jeder regulär immatrikulierte Studierende sollte einen Konto für den Oracle-Server der Fakultät besitzen

- Initiales Passwort ist „**kein**“ – vor erster Benutzung mit im Pool installiertem Tool „SQL Plus“ ändern

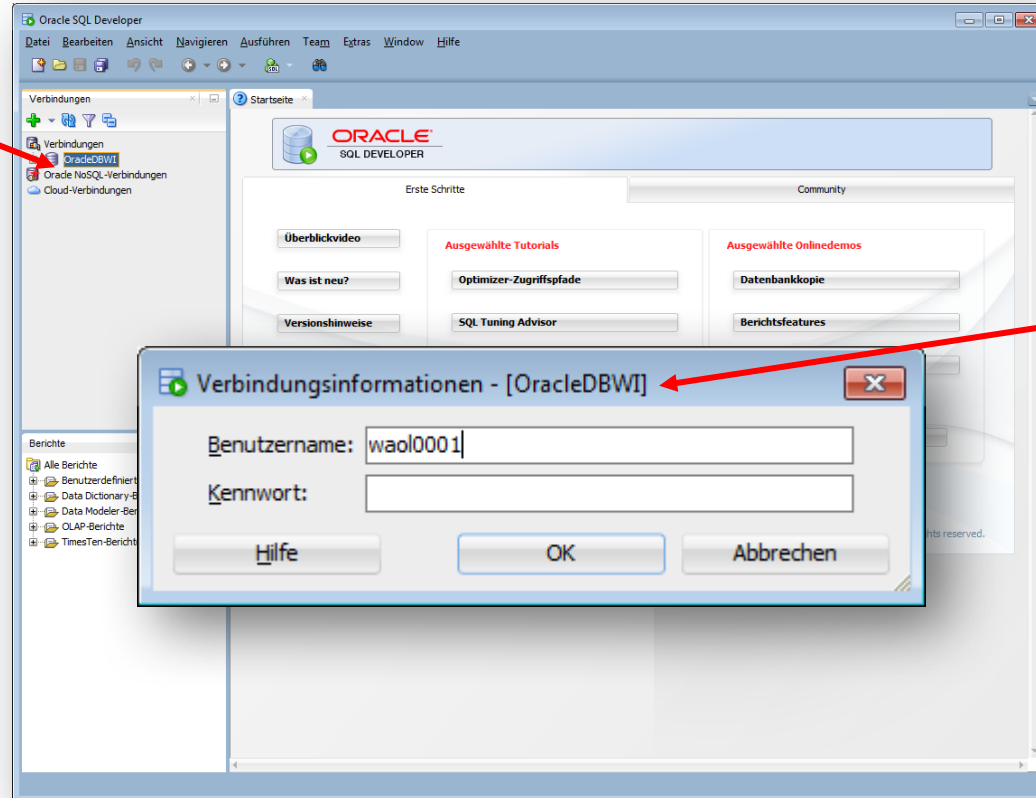


```
SQL*Plus: Release 11.2.0.1.0 Production on Mo Mrz 13 12:01:55 2017
Copyright (c) 1982, 2010, Oracle. All rights reserved.
Benutzernamen eingeben: wao10001@oracledbwi
Kennwort eingeben: _
```

Bei Problemen Datenbankadministrator kontaktieren (Holger.Bechtold@hs-karlsruhe.de)

Datenbankverbindung herstellen

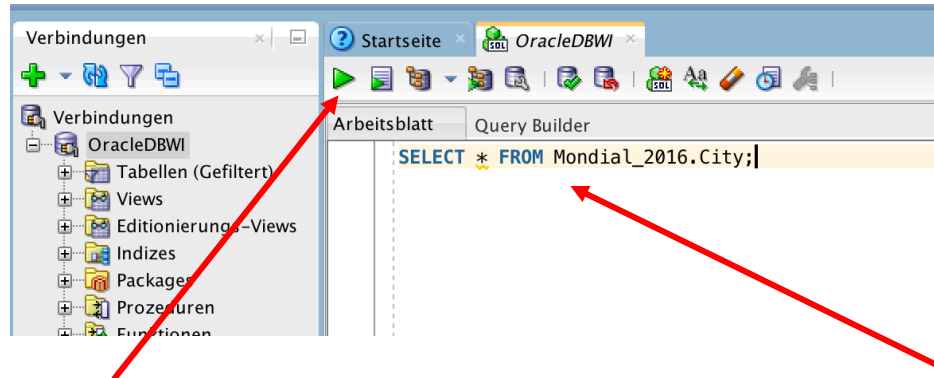
1. Doppelklick



2. Konto- informationen eingeben + OK drücken

Verbindung herstellen und Abfrage starten

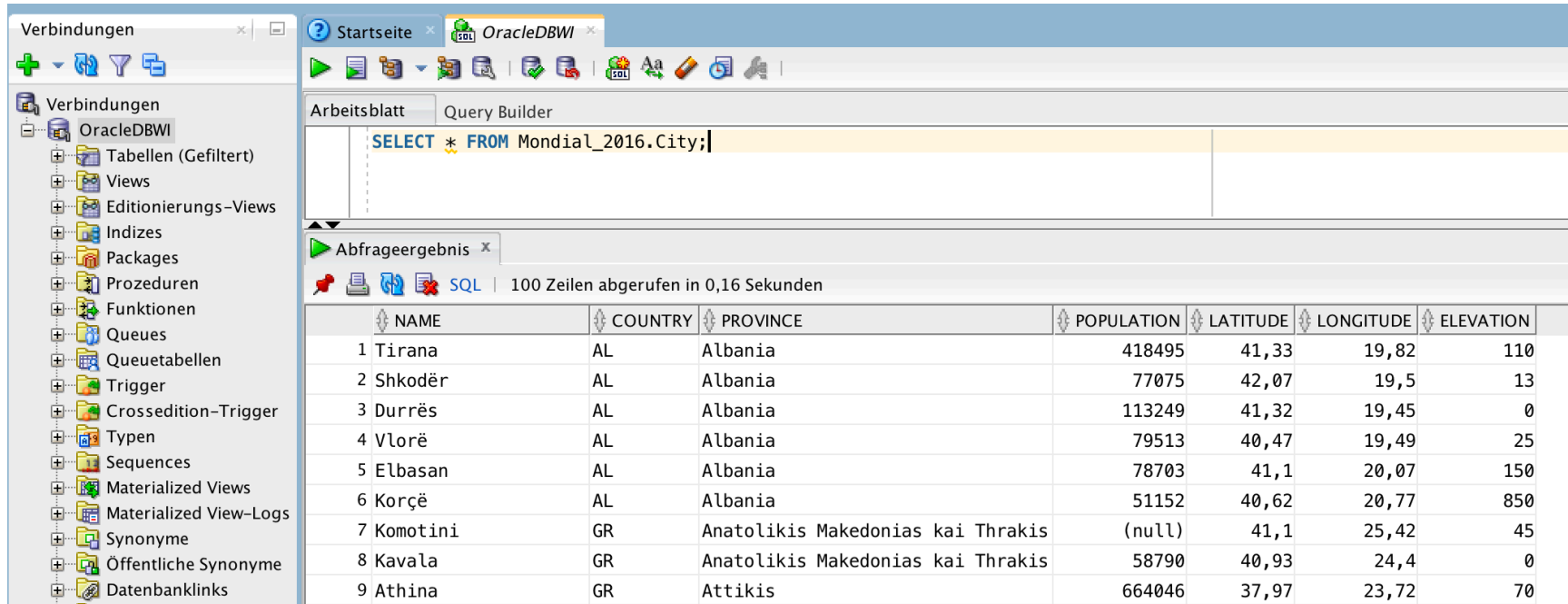
- Danach kann z.B. eine Tabelle einer Beispiel-Datenbank abgefragt werden:
 - **SELECT** * **FROM** Mondial_2016.City;



2. Auf Symbol „Anweisung ausführen“ klicken

1. Abfrage in Arbeitsblatt eingeben

Beispiel-Ausgabe



The screenshot shows the OracleDBWI Query Builder interface. The left pane displays the database structure under 'Verbindungen' (Connections) for 'OracleDBWI', including tables, views, indexes, and other database objects. The main workspace shows a SQL query in the 'Arbeitsblatt' (Worksheet) tab:

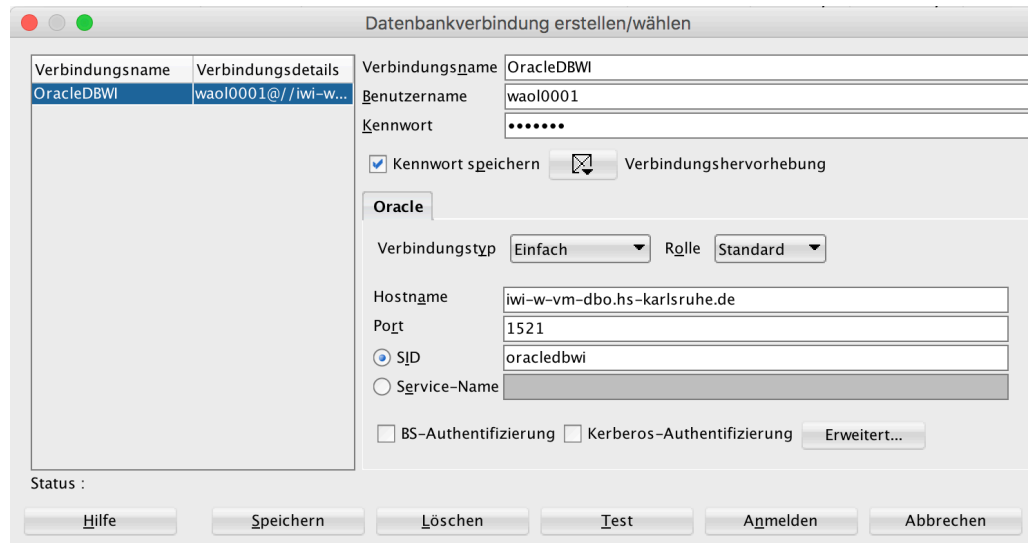
```
SELECT * FROM Mondial_2016.City;
```

The 'Abfrageergebnis' (Query Result) tab shows the results of the query, indicating that 100 rows were retrieved in 0.16 seconds. The results are displayed in a table with the following columns: NAME, COUNTRY, PROVINCE, POPULATION, LATITUDE, LONGITUDE, and ELEVATION.

	NAME	COUNTRY	PROVINCE	POPULATION	LATITUDE	LONGITUDE	ELEVATION
1	Tirana	AL	Albania	418495	41,33	19,82	110
2	Shkodër	AL	Albania	77075	42,07	19,5	13
3	Durrës	AL	Albania	113249	41,32	19,45	0
4	Vlorë	AL	Albania	79513	40,47	19,49	25
5	Elbasan	AL	Albania	78703	41,1	20,07	150
6	Korçë	AL	Albania	51152	40,62	20,77	850
7	Komotini	GR	Anatolikos Makedonias kai Thrakis	(null)	41,1	25,42	45
8	Kavala	GR	Anatolikos Makedonias kai Thrakis	58790	40,93	24,4	0
9	Athina	GR	Attikis	664046	37,97	23,72	70

Einrichten von SQL Developer zu Hause

- SQL Developer kann für den eigenen Rechner von den Oracle-Webseiten heruntergeladen werden:
<http://www.oracle.com/technetwork/developer-tools/sql-developer/downloads/index.html>
- Mit dem eigenen Konto für den Oracle-Datenbankserver kann im SQL Developer eine neue Verbindung hinzugefügt werden:



Verbindungsname	Verbindungsdetails
OracleDBWI	waol0001@//iwi-w-...

Verbindungsname: OracleDBWI
Benutzername: waol0001
Kennwort:
☒ Kennwort speichern ☐ Verbindungshervorhebung

Oracle

Verbindungstyp: Einfach Rolle: Standard

Hostname: iwi-w-vm-dbo.hs-karlsruhe.de
Port: 1521
☒ SID: oracledbwi
☐ Service-Name:
☐ BS-Authentifizierung ☐ Kerberos-Authentifizierung

Status :

Übung: SQL-Developer einrichten

- Ändern Sie Ihr Passwort (Folie 12)
- Starten Sie SQL Developer und stellen Sie ein Verbindung mit dem Oracle-Server her (Folie 13)
- Stellen Sie eine Beispiel-SQL-Anfrage (Folie 14)

Weitere Anfragen zum Ausprobieren

- **SELECT** * **FROM** Mondial_2016.Country;
- **SELECT** * **FROM** Mondial_2016.City **WHERE** Country='D';
- **DESCRIBE** Mondial_2016.City;
- **SELECT** Name, Province **FROM** Mondial_2016.City
WHERE Country='D';

! Übung: Probieren Sie die Befehle aus! Was macht jeder einzelne Befehl?

Anlegen von Tabellen

Syntax:

```
CREATE TABLE R (  
    A1 D1,  
    A2 D2,  
    ...,  
    An Dn,  
    <integrity-constraint1>,  
    ...,  
    <integrity-constraintn> );
```

- D_i ist der Datentyp von Attribut (Spalte) A_i
- Mit $\langle \text{integrity-constraint}_i \rangle$ beschäftigen wir uns später

Beispiel: Anlegen von Tabellen

Tabelle von Folie 6:

```
CREATE TABLE Abteilung (  
    Abteilungsnummer INTEGER,  
    Abteilungsname CHARACTER(30),  
    Abteilungsleiter CHARACTER(30) );
```

! Übung: Legen Sie diese Tabelle an!

Grundlegende Datentypen (1)

CHAR (n) / **CHARACTER** (n)

- Zeichenkette mit genau n Zeichen
- Aufgefüllt mit Leerzeichen

VARCHAR (n) / **CHARACTER VARYING** (n)

- Zeichenkette min max. n Zeichen

INT / **INTEGER**

- Ganze Zahl, maschinenabhängig

SMALLINT

- Teilmenge der ganzen Zahlen, maschinenabhängig

Grundlegende Datentypen (2)

NUMERIC (d, p)

- Festkommazahl, d Ziffern, davon p hinter dem Dezimalpunkt

REAL / DOUBLE PRECISION

- Fließkommazahl, Genauigkeit maschinenabhängig

FLOAT (n)

- Fließkommazahl, mit Genauigkeit (mind.) n Stellen

! Übung: Datentypen in Oracle

- Legen Sie die folgende Tabelle an:

```
CREATE TABLE DatentypenBeispiel (  
  Zeichenkette CHARACTER(5),  
  VariableZeichenkette VARCHAR(10),  
  GanzeZahl INTEGER,  
  KleineGanzeZahl SMALLINT,  
  Festkommazahl NUMERIC(3,1),  
  ReelleZahl REAL,  
  DoppelteGenauigkeit DOUBLE PRECISION,  
  Fließkommazahl FLOAT(6) );
```

- Prüfen Sie dann die Tabellenstruktur

```
DESCRIBE DatentypenBeispiel;
```

- Welche Datentypen verwendet Oracle intern?

Weitere Datentypen

DATE

- Datum

TIME

- Zeit mit und ohne Zeitzone

INTERVALL

- Zeitdauer

BLOB

- Beliebige binäre Zeichenketten, Größe meist Gigabyte

BOOLEAN

- **TRUE** oder **FALSE**

Tabellen modifizieren oder löschen

- Komplette Tabelle R löschen:

DROP TABLE R ;

- Attribut A mit Datentyp D zu Tabelle R hinzufügen:

ALTER TABLE R **ADD** A D ;

- Attribut A aus Relationsschema R löschen:

ALTER TABLE R **DROP COLUMN** A ;

Beispiel: Tabellen modifizieren / löschen

-- Spalte für Anzahl Mitarbeiter einfügen
ALTER TABLE Abteilung **ADD** AnzMitarbeiter **INTEGER**;

-- Spalte löschen
ALTER TABLE Abteilung **DROP COLUMN** AnzMitarbeiter;

-- komplette Tabelle löschen
DROP TABLE Abteilung;

Einfügen von Datensätzen mit **INSERT**

- Beispieltabelle MyRiver

```
CREATE TABLE MyRiver (  
    Name VARCHAR(50),  
    Country VARCHAR(4),  
    Length NUMBER );
```

- Beispiel: 3x einfügen einzelner Datensätze

```
INSERT INTO MyRiver VALUES ('Ruhr', 'D', 219 );
```

```
INSERT INTO MyRiver(Length, Country, Name)  
VALUES (93, 'D', 'Kinzig');
```

```
INSERT INTO MyRiver(Name, Country)  
VALUES ('Alb', 'D');
```

Aktualisieren von Datensätzen mit **UPDATE**

Im Allgemeinen **UPDATE** *R* **SET** *A* **WHERE** *P*

- Aktualisiert Tabelle *R*
- Prädikat *p* in **WHERE**-Klausel wählt aus, welche Daten
- Ausdruck *A* in **SET**-Klausel gibt an, was geändert werden soll

Beispiel: Länge der Alp einfügen

```
UPDATE MyRiver  
  SET Length=51  
  WHERE Name='Alb';
```

Löschen von Datensätzen mit DELETE

Löschen von Tupeln, die Prädikat in **WHERE**-Klausel erfüllen

Beispiel: Datensatz für Alp löschen

```
DELETE FROM MyRiver  
WHERE Name='Alp';
```

Beschränkungen / Constraints (1)

NOT NULL

- Attribut muss einen Wert zugewiesen bekommen

... ,
 A_{j1} D_{j1} **NOT NULL** ,
...

UNIQUE

- Jeder Wert eines Attributs / einer Menge von Attributen darf nur einmal vorkommen

UNIQUE (A_{j1} , A_{j2} , ..., A_{jn})



Beispiele UNIQUE / NOT NULL Constraints

```
CREATE TABLE MyRiver (  
    Name VARCHAR(50),  
    Country VARCHAR(4),  
    Length NUMBER NOT NULL,  
    UNIQUE(Name, Country) );
```

```
INSERT INTO MyRiver (Name, Country)  
VALUES ('Rhein', 'D'); -- funktioniert nicht
```

```
INSERT INTO MyRiver (Name, Country, Length)  
VALUES ('Rhein', 'D', 1234); -- funktioniert
```

```
INSERT INTO MyRiver (Name, Country, Length)  
VALUES ('Rhein', 'D', 1233); -- funktioniert nicht
```

```
INSERT INTO MyRiver (Name, Country, Length)  
VALUES ('Rhein', 'NL', 1234); -- funktioniert
```

Beschränkungen / Constraints (2)

Primärschlüssel

PRIMARY KEY ($A_{j1}, A_{j2}, \dots, A_{jn}$)

- Nicht NULL (**NOT NULL**) und eindeutig (**UNIQUE**)
- Es ist immer eine gute Idee einen Primärschlüssel zu haben!

Fremdschlüssel

FOREIGN KEY ($A_{k1}, A_{k2}, \dots, A_{kn}$) **REFERENCES** S

- Die Attribute müssen auf den Primärschlüssel von Tabelle S referenzieren
- Fremdschlüssel von R kann in keinem Datensatz Werte annehmen, die in dieser Kombination in S nicht existieren

Beispiel Primär- / Fremdschlüssel (1)

```
CREATE TABLE Abteilung (  
    Abteilungsnummer NUMERIC(3),  
    Abteilungsname VARCHAR(30),  
    Abteilungsleiter VARCHAR(40),  
    PRIMARY KEY (Abteilungsnummer) );
```

```
CREATE TABLE Mitarbeiter (  
    Personalnummer NUMERIC(4),  
    Nachname VARCHAR(40),  
    Vorname VARCHAR(30),  
    Funktion VARCHAR(30),  
    Abteilung NUMERIC(3),  
    PRIMARY KEY (Personalnummer),  
    FOREIGN KEY (Abteilung) REFERENCES Abteilung );
```


Beispiel Primär- / Fremdschlüssel (2)

```
INSERT INTO Abteilung (Abteilungsnummer, Abteilungsname, Abteilungsleiter)
VALUES (1, 'Buchhaltung', 'Meier'); -- funktioniert
```

```
INSERT INTO Abteilung (Abteilungsnummer, Abteilungsname, Abteilungsleiter)
VALUES (1, 'Vertrieb', 'Mueller'); -- funktioniert nicht
```

```
INSERT INTO Abteilung (Abteilungsname, Abteilungsleiter)
VALUES ('Vertrieb', 'Mueller'); -- funktioniert nicht
```

```
INSERT INTO Mitarbeiter
VALUES (22, 'Waldhorst', 'Oliver', 'Professor', 2 ); -- funktioniert nicht
```

```
INSERT INTO Mitarbeiter
VALUES (22, 'Waldhorst', 'Oliver', 'Professor', 1 ); -- funktioniert
```

Zusammenfassung Kapitel 6

SQL ist eine der gebräuchlichsten Anfragesprachen für relationale Datenbankmanagementsysteme (RDBMS)

- Besteht aus Data Definition Language (DDL), Data Manipulation Language (DML) und Data Control Language (DCL)
- Standard, aber in jedem RDBMS etwas unterschiedlich implementiert

Wir verwenden

- Oracle als RDBMS
- SqlDeveloper als Anwendungsprogramm

SQL-Grundlagen

- Anlegen und verändern von Tabellen
- Einfügen, verändern und löschen von Datensätzen



Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

Kapitel 7

Grundlegende SQL-Konzepte

Vorlesung Datenbanken und Informationssysteme I
Sommersemester 2017

Prof. Dr. rer. nat. Oliver P. Waldhorst

Inhalte dieses Kapitels

- SQL vs. relationale Algebra
- **SELECT**-Statements
 - Anfragen auf einer Relation / mehreren Relationen
 - Mengen-Operationen
 - Null-Werte
 - Aggregation und Gruppierung
 - Geschachtelte Anfragen
- **INSERT**-Statements
- **UPDATE**-Statements
- **DELETE**-Statements

SQL vs. relationale Algebra (1)

SQL bildet Konzepte der relationalen Algebra in **Data Manipulation Language (DML)** ab

- **SELECT** kann leicht mit Selektion verwechselt werden; dabei entspricht die Selektion eher dem Teil hinter **WHERE**
 - Z.B. „... **WHERE** MatrikelNr = 12345“

Achtung: SQL-Anfragen können doppelte Tabellenzeilen liefern!

- Relationen können – da sie Mengen sind – keine doppelten Tupel enthalten

SQL vs. relationale Algebra (2)

Ein **SELECT**-Statement in SQL fasst mehrere relationale Operatoren zusammen:

$$\pi_{A_1, A_2, \dots, A_n} \left(\sigma_{\beta} (R_1 \times R_2 \times \dots \times R_m) \right)$$

lässt sich schreiben als

SELECT A_1, A_2, \dots, A_n **FROM** R_1, R_2, \dots, R_m **WHERE** β

Mehr zur Data Manipulation Language (DML)

SQL-Anfragen bestehen aus drei **Klauseln** (engl. **Clauses**)

SELECT A_1, A_2, \dots, A_n **FROM** R_1, R_2, \dots, R_m **WHERE** β

SELECT-Klausel **FROM-Klausel** **WHERE-Klausel**

SELECT -Klausel:	Wählt / modifiziert Spalten/Attribute
FROM -Klausel:	Wählt Tabellen/Relationen
WHERE -Klausel:	Spezifiziert Bedingungen

Beispiele für Anfragen auf einer Relation

```
SELECT * FROM City;
```

```
SELECT Name, Province FROM City;
```

```
SELECT Province FROM City; -- 3381
```

```
SELECT DISTINCT Province FROM City; -- 1570  
-- in Oracle auch UNIQUE
```

```
SELECT ALL Province FROM City; -- 3381
```


Operatoren in der **SELECT**-Klausel

In der **SELECT**-Klausel können die Operatoren $+$, $-$, $*$, $/$ auf Konstanten und Attribute angewendet werden

Beispiel: Für Relation $R(A, B, C, D)$:

```
SELECT A+3, B*2, C-D FROM R;
```

 Übung: Geben Sie alle Flüsse mit ihrer Länge in Meilen aus

```
SELECT Name, Length/1.6 FROM River;
```

Auswahl mit der **WHERE**-Klausel

WHERE-Klausel wählt Tupel/Zeilen, die bestimmte Bedingungen (**Prädikat**) erfüllen

- Kann Attribute und Konstanten erhalten
 - Arithmetische Operationen sind erlaubt
- Vergleichsoperatoren: $<$, $<=$, $=$, $>$, $>=$, $<>$, \neq
- Verknüpfungen: **AND**, **OR**, **NOT**

Beispiel: Für Relation $R(A, B, C, D)$:

```
SELECT * FROM R WHERE A  $\leq$  10;
```

 Übung: Geben Sie die Name der Städte mit mehr als 50.000 Einwohnern aus!

```
SELECT * FROM CITY WHERE Population  $>$  50000;
```

Vereinfachende Schreibweisen

Für häufig genutzte Vergleiche gibt es Vereinfachungen

Beispiel **BETWEEN**:

```
SELECT Name FROM City  
WHERE Population>100000 AND Population<500000;
```

kann geschrieben werden als:

```
SELECT Name FROM City  
WHERE Population BETWEEN 100000 AND 500000;
```

String-Funktionen

- **Strings** sind Zeichenketten und werden in SQL in einfachen Anführungszeichen (z.B. 'Zeichenkette') geschrieben
- Abhängig von DBMS gibt es viele Funktionen auf Strings
 - Z.B. **UPPER**(string), **LOWER**(string), **TRIM**(string), ...
- String-Vergleiche unterscheiden standardmäßig Groß- und Kleinschreibung

Beispiel: 'Waldhorst' != 'waldhorst' ist wahr!

Beispiel Zeichenkettenvergleiche

```
CREATE TABLE ZeichenkettenBeispiel (  
    KurzeZeichenkette CHAR(6),  
    LangeZeichenkette CHAR(10),  
    VariableZeichenkette VARCHAR(10) );
```

```
INSERT INTO ZeichenkettenBeispiel  
VALUES      ('Oliver', 'Oliver', 'Oliver' );
```

```
SELECT * FROM ZeichenkettenBeispiel  
  WHERE KurzeZeichenkette=LangeZeichenkette; -- liefer ein Tupel, weil  
CHAR(6) auf 10 Zeichen aufgefüllt wird
```

```
SELECT * FROM ZeichenkettenBeispiel  
  WHERE KurzeZeichenkette=VariableZeichenkette; -- liefer ein Tupel
```

```
SELECT * FROM ZeichenkettenBeispiel  
  WHERE LangeZeichenkette=VariableZeichenkette; -- liefer KEIN Tupel,  
VARCHAR(6) wird NICHT auf 10 Zeichen aufgefüllt!!!
```

String-Matching mit **LIKE**

Operator **LIKE** ermöglicht String-Vergleiche mit Platzhaltern

- **%** steht für Substring beliebiger Länge (auch keinem Zeichen!)
- **_** steht für Substring der Länge 1 (d.h. genau einem Zeichen)

Beispiele:

```
SELECT Name FROM Country  
WHERE Name LIKE 'B%';
```

```
SELECT Name FROM Country  
WHERE Name LIKE '____land';
```

```
SELECT Name FROM Country  
WHERE Name LIKE '%land%';
```

```
SELECT Name FROM Country  
WHERE Name LIKE '_____ %';
```

Übung: String-Funktionen

1. Geben Sie alle Flüsse aus, deren Namen sechs Buchstaben hat, wobei der vierte Buchstabe ein „a“ ist!

```
SELECT Name FROM River  
WHERE Name LIKE '___a__';
```

2. Geben Sie alle Flüsse aus, deren Namen als vierten Buchstaben ein „a“ und als siebten Buchstaben ein B enthält!

```
SELECT Name FROM River  
WHERE Name LIKE '___a__B%';
```

3. Geben Sie alle Flussnamen in GROSSBUCHSTABEN aus!

```
SELECT UPPER(Name) FROM River;
```

Sortieren ausgegebener Tupel

ORDER BY-Klausel ermöglicht Sortierung der ausgegebenen Tupel nach Attributen

- Richtung der Sortierung
 - **ASC** - aufsteigend (ascending)
 - **DESC** - absteigend (descending)
- Auch für mehrere Attribute möglich
 - Semantik: Sortierung zuerst nach erstgenanntem Attribut, bei Gleichheit nach zweitgenanntem, ...

Beispiel: Für Relation $R(A, B, C, D)$:

```
SELECT * FROM R ORDER BY A ASC, B DESC;
```

!

Übung: Geben Sie alle Städtenamen mit Land und Provinz aus, geordnet nach

- Land, aufsteigend
- dann Provinz, absteigend
- dann Städtenamen, aufsteigend

```
SELECT Name, Country, Province FROM City  
ORDER BY Country ASC, Province DESC, Name ASC;
```


Anfragen auf mehreren Relationen (1)

Manchmal brauchen wir mehrere Relationen, um Anfragen zu beantworten

- Diese können in **FROM**-Klausel mit Komma getrennt aufgezählt werden
- Semantik entspricht kartesischem Produkt (vgl. Kapitel 5)

Beispiel: Für Relationen R, S:

SELECT * **FROM** R, S;

 Übung: Zu welchem Kontinent gehört eine Insel?

Anfragen auf mehreren Relationen (2)

Anmerkungen:

- Beim Zugriff auf Attribute muss Name der Relation vorangestellt werden, wenn Attributname in mehreren Relationen verwendet wird

Beispiel: Name aller Länder mit ihren Hauptstädten

```
SELECT DISTINCT Country.Name, City.Name  
FROM Country, City  
WHERE Country.Capital = City.Name;
```

- Wenn eindeutig ist Nennung der Relation nicht notwendig

Beispiel: Zusätzlich Ländercode zu Ausgabe oben

```
SELECT DISTINCT Country.Name, City.Name, Code  
FROM Country, City  
WHERE Country.Capital = City.Name;
```

Auswertung des **SELECT**-Statements (1)

SELECT A_1, A_2, \dots, A_n **FROM** R_1, R_2, \dots, R_m **WHERE** P ;

- 1) **FROM**-Klausel: kartesisches Produkt von R_1, R_2, \dots, R_m
- nicht alle Tupel sind sinnvoll!

Beispiel: GeoIsland x Encompasses:

Auswertung des **SELECT**-Statements (2)

2) **WHERE**-Klausel: Einschränkung auf Tupel, die aussagekräftig für die gesuchte Antwort sind

Beispiel: Gleiches Country in Geo_Island und Encompasses

Auswertung des **SELECT**-Statements (3)

- 3) Für alle nach 2) übriggebliebenen Tupel Attribute (oder Ausdrücke) aus **SELECT**-Klausel ausgeben
- Achtung: **SELECT**-Klausel wird erst in Schritt 3 ausgewertet, d.h. wir können in **WHERE**-Klausel Attribute verwenden, die in **SELECT**-Klausel nicht vorkommen!

Beispiel: Name von Insel und Kontinent

```
SELECT Island, Continent FROM Geo_Island, Encompasses  
WHERE Geo_Island.Country = Encompasses.Country;
```

DBMS macht Auswertung geschickter (z.B. Tupel früh verwerfen)

Natural Join

Analog zur relationalen Algebra: Vereinfachung des Vergleichs von Attributen mit gleichem Namen

Beispiel von vorheriger Folie

```
SELECT Island, Continent  
FROM Geo_Island NATURAL JOIN Encompasses;
```

Natural Join (2)

Konzeptionell wird Ausdruck in **FROM**-Klausel ersetzt durch das Ergebnis der Natural Joins (ist wieder eine Relation)!

- Ggf. kann dann nicht mehr über Relationsnamen auf Attribute zugegriffen werden

Beispiel: Einschränkung auf deutsche Inseln

```
SELECT Geo_Island.Country, Encompasses.Country
FROM Geo_Island, Encompasses
WHERE Geo_Island.Country = Encompasses.Country;

SELECT Geo_Island.Country, Encompasses.Country
FROM Geo_Island NATURAL JOIN Encompasses;
-- geht nicht!!!

SELECT Country
FROM Geo_Island NATURAL JOIN Encompasses;
```

Natural Join (3)

Um irrtümliches gleichsetzen zu verhindern, können Attribute mit **USING**-Bedingung angegeben werden

Beispiel: Gleichheit nur für Attribut Country

```
SELECT Country  
FROM Geo_Island JOIN Encompasses  
USING(Country);
```


NATURAL JOIN vs. JOIN USING (1)

A	B	C
1	2	3
4	5	6

B	C	D
2	3	4
5	6	2

SELECT * **FROM** R **NATURAL JOIN** S;

Folie 22

A	B	C	D
1	2	3	4
4	5	6	2

NATURAL JOIN vs. JOIN USING (2)

A	B	C
1	2	3
4	5	6

B	C	D
2	3	4
5	6	2

SELECT * **FROM** R **JOIN** S **USING** (B) ;

Folie 63

R.A	R.C	B	S.C	S.D
1	3	2	3	4
4	6	5	6	2

NATURAL JOIN vs. JOIN USING (3)

```
CREATE TABLE R (A INT, B INT, C INT);  
INSERT INTO R VALUES (1,2,3);  
INSERT INTO R VALUES (4,5,6);
```

```
CREATE TABLE S (B INT, C INT, D INT);  
INSERT INTO S VALUES (2,8,9);  
INSERT INTO S VALUES (5,6,2);
```

```
SELECT * FROM R NATURAL JOIN S;
```

```
SELECT * FROM R JOIN S USING (B);
```

```
SELECT * FROM R JOIN S USING (B) WHERE C=3;
```

```
SELECT * FROM R JOIN S USING (B) WHERE R.C=3;
```

```
SELECT * FROM R JOIN S USING (B) WHERE S.C=8;
```

Natural Join (4)

Generelle Form des Natural Joins

- Verkettung ist möglich

```
SELECT A1, A2, ..., An
FROM R1 NATURAL JOIN R2 NATURAL JOIN ... NATURAL JOIN Rm
WHERE P;
```

- Noch allgemeinerer

```
... FROM E1, E2, ..., En ...
```

wobei ein Relation oder ein Ausdruck mit Natural Joins ist

Beispiel: Ausgabe von Inselname, Kontinent und Typ mit Kombination von NATURAL JOIN und kartesischem Produkt

```
SELECT Island, Continent, Type
FROM Geo_Island NATURAL JOIN Encompasses, Island
WHERE Name=Island;
```

Join-Bedingungen (vgl. θ -Join)

Join-Bedingung kann mit Schlüsselwort **ON** durch beliebiges Prädikat definiert werden

Beispiel: Ausgabe von Inselname, Kontinent und Typ mit Kombination von **NATURAL JOIN** und **JOIN ... ON**

```
SELECT Island, Continent, Type
FROM Geo_Island NATURAL JOIN Encompasses
JOIN I_Sland ON Name=Island;
```

JOIN ... ON vs. kartesisches Produkt + **WHERE...**

- Unterschied bei OUTER JOIN (s. unten)
- Außerdem besser lesbar!

! Übung: JOIN ... ON

Listen Sie alle Länder mit den Flüssen, die durch Sie fließen, auf, aufsteigend geordnet nach Länder und dann Flussnamen.

```
SELECT DISTINCT Country.Name, Geo_River.River  
FROM Country JOIN geo_river  
ON Country.Code=Geo_River.Country  
ORDER BY Country.Name, Geo_River.River;
```

Outer Join

Vermeidung des Verlustes von Informationen wegen fehlender Zuordnung

LEFT OUTER JOIN: Tupel aus Relation links von Operation

RIGHT OUTER JOIN: Tupel aus Relation rechts von Operation

FULL OUTER JOIN: Tupel aus beiden Relationen


Anmerkungen:

- Bedingungen für Outer Join können durch **NATURAL**, **USING**, **ON** spezifiziert werden
- Zur besseren Unterscheidung kann „normaler“ Join durch Schlüsselwort **INNER** gekennzeichnet werden:

```
SELECT Island, Continent  
FROM Geo_Island NATURAL INNER JOIN Encompasses;
```

Übung: Outer Join

Geben Sie alle Länder aus, falls vorhanden mit Flüssen, die durch sie fließen!

```
 SELECT DISTINCT Country.Name, Geo_River.River  
FROM Country LEFT OUTER JOIN geo_river  
ON Country.Code=Geo_River.Country  
ORDER BY Country.Name, Geo_River.River;
```


OUTER JOIN ... ON vs. kartesisches Produkt + WHERE

Beispiel: Umbau des Fluss-Beispiels von vorheriger Folie

```
SELECT DISTINCT Country.Name, Geo_River.River
FROM Country LEFT OUTER JOIN geo_river ON 1=1
WHERE Country.Code=Geo_River.Country
ORDER BY Country.Name, Geo_River.River;
```

Hier werden Tupel, die Join-Bedingung nicht erfüllen, nicht in Ergebnis übernommen!

Die Umbenennungsoperation (1)

Beispiel:

```
SELECT Island, Continent  
FROM Geo_Island, Encompasses  
WHERE Geo_Island.Country = Encompasses.Country;
```

Ergebnis ist eine Relation mit Attributen `Island`, `Continent`

- Abgeleitet aus der **FROM**-Klausel

Ableitung in manchen Fällen nicht möglich

- Doppelte Attributnamen
- Arithmetische Ausdrücke in **SELECT**-Klausel
- Vielleicht sind andere Attributnamen gewünscht

Die Umbenennungsoperation (2)

Umbenennung der Attribute durch Schlüsselwort **AS**:

```
SELECT AlterName AS NeuerName ...
```

! Übung: Geben Sie für alle Inseln den Inselname als „InselName“ und den Kontinent als „ContinentName“ aus

```
SELECT Island AS IslandName, Continent AS ContinentName  
FROM Geo_Island, Encompasses  
WHERE Geo_Island.Country = Encompasses.Country;
```

Die Umbenennungsoperation (3)

Umbenennung der Relationen

- ... zum Beispiel für einfachere Schreibweise (SQL-Standard)

```
SELECT G.Island, E.Continent  
  FROM Geo_Island AS G JOIN Encompasses AS E  
    ON G.Country = E.Country;
```

Achtung: Oracle erlaubt kein AS in FROM-Klausel!

```
SELECT G.Island, E.Continent  
  FROM Geo_Island G JOIN Encompasses E  
    ON G.Country = E.Country;
```

Die Umbenennungsoperation (4)

Im Beispiel oben sind G und E „Aliase“, d.h. alternative Namen

- Sie können sich diese als „Kopien“ der Tabellen vorstellen
- Im Standard heißen G und E **Correlation Name**

Übung: Umbenennung

Finden Sie die Namen aller Flüsse, die länger sind als mindestens ein Fluss, der in die Nordsee mündet. Verwenden Sie dazu das kartesische Produkt und die Umbenennung!

```
SELECT DISTINCT R1.Name  
  FROM River R1, River R2  
 WHERE R2.Sea='North Sea'  
       AND R1.Length > R2.Length;
```

Mengen-Operationen

... wie aus Mengenlehre bekannt

Im Folgenden als Beispiel verwendet: alle Länder(-codes), die einen See oder einen Fluss beinhalten

```
SELECT Country FROM Geo_Lake;
```

```
SELECT Country FROM Geo_River;
```

Der **UNION**-Operator

Vereinigung - entspricht U aus der relationalen Algebra

Beispiel:

```
SELECT Country FROM Geo_Lake  
UNION  
SELECT Country FROM Geo_River;
```

Achtung: Mengen-Operation! Keine doppelten Tupel!

Der **UNION ALL**-Operator

Manchmal sind Duplikate gewünscht!

Beispiel:

```
SELECT Country FROM Geo_Lake  
UNION ALL  
SELECT Country FROM Geo_River;
```

Der **INTERSECT**- / **INTERSECT ALL**-Operator

Schnittmenge – entspricht \cap aus relationaler Algebra

Beispiele:

```
SELECT Country FROM Geo_Lake  
INTERSECT  
SELECT Country FROM Geo_River;
```

```
SELECT Country FROM Geo_Lake  
INTERSECT ALL  
SELECT Country FROM Geo_River;
```

Achtung: **INTERSECT ALL** in Oracle nicht vorhanden!

Der **EXCEPT**- / **EXCEPT-ALL** Operator

Mengendifferenz - entspricht — aus relationaler Algebra

Beispiel:

```
SELECT Country FROM Geo_Lake  
EXCEPT  
SELECT Country FROM Geo_River;
```

```
SELECT Country FROM Geo_Lake  
EXCEPT ALL  
SELECT Country FROM Geo_River;
```

Achtung:

- In Oracle heißt der Operator **MINUS**
- In Oracle ist **MINUS ALL** nicht vorhanden!

Null-Werte (1)

Wiederholung:

- NULL bedeutet „nicht existent“, „nicht bekannt“, „noch nicht bekannt“ oder „an dieser Stelle ohne Bedeutung“
- **Darf nicht mit dem numerischen Wert 0 (engl.: zero) verwechselt werden!**

Umgang mit NULL-Werten

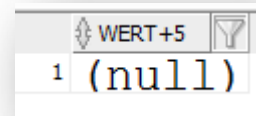
- Arithmetische Ausdrücke mit NULL sind immer Null
z.B. $5 + \text{NULL} = \text{NULL}$
- Jeder Vergleich mit NULL liefert UNKNOWN (außer **IS NULL**, **IS NOT NULL**, s.u.)
z.B. $1 < \text{NULL} = \text{UNKNOWN}$
- *UNKNOWN* ist 3. Vergleichsergebnis neben *TRUE*, *FALSE*

Beispiel Null-Werte

```
CREATE TABLE NullBsp ( Wert INT );
INSERT INTO NullBsp VALUES ( NULL );
SELECT * FROM NullBsp;
```

-- gibt Null aus

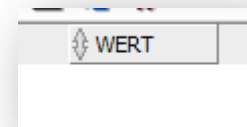
```
SELECT Wert+5 FROM NullBsp;
```



	WERT+5
1	(null)

-- gibt keinen Wert aus

```
SELECT * FROM NullBsp
WHERE 1 < Wert OR 1 >= Wert;
```



	WERT
--	------

Null-Werte (2)

In **WHERE**-Klausel kann *UNKNOWN* in Booleschen Ausdrücken auftauchen – Auswertung:

<i>TRUE</i>	AND	<i>UNKNOWN</i>	= <i>UNKNOWN</i>
<i>FALSE</i>	AND	<i>UNKNOWN</i>	= <i>FALSE</i>
<i>UNKNOWN</i>	AND	<i>UNKNOWN</i>	= <i>UNKNOWN</i>

<i>TRUE</i>	OR	<i>UNKNOWN</i>	= <i>TRUE</i>
<i>FALSE</i>	OR	<i>UNKNOWN</i>	= <i>UNKNOWN</i>
<i>UNKNOWN</i>	OR	<i>UNKNOWN</i>	= <i>UNKNOWN</i>

NOT	<i>UNKNOWN</i>	= <i>UNKNOWN</i>
------------	----------------	------------------

Null-Werte (3)

Auswertung für ein Tupel

- Wenn Prädikat in **WHERE**-Klausel *FALSE* oder *UNKNOWN* ist, kommt Tupel nicht ins Ergebnis (s. Beispiel oben)
- Explizite Abfrage von NULL Werten mit **IS NULL** / **IS NOT NULL**

Beispiel:

```
SELECT * FROM NullBsp WHERE Wert IS NULL;
```

Null-Werte (4)

- Bei **SELECT DISTINCT** sind zwei Tupel gleich, die für alle Attribute gleiche oder NULL-Werte besitzen → nicht im Ergebnis

Beispiele:

```
INSERT INTO NullBsp VALUES ( NULL );
```

```
-- liefert 2 Zeilen
```

```
SELECT * FROM NullBsp;
```

```
-- liefert 1 Zeile
```

```
SELECT DISTINCT * FROM NullBsp;
```


Übung: Null-Werte

1. Geben Sie alle Flüsse aus, die in einen See (Lake) münden!

```
SELECT Name, Lake  
FROM River  
WHERE Lake IS NOT NULL;
```

2. Geben Sie alle Flüsse aus, die nicht in einen See (Lake) münden!

```
SELECT Name, Lake  
FROM River  
WHERE Lake IS NULL;
```

Aggregation

Aggregatsfunktionen nehmen eine Sammlung (Menge oder Multimenge) von Werten und liefern einen einzigen Wert zurück

- Anwendung in **SELECT**-Klausel
- **Ergebnis ist immer eine Relation (d.h. Tabelle)!**

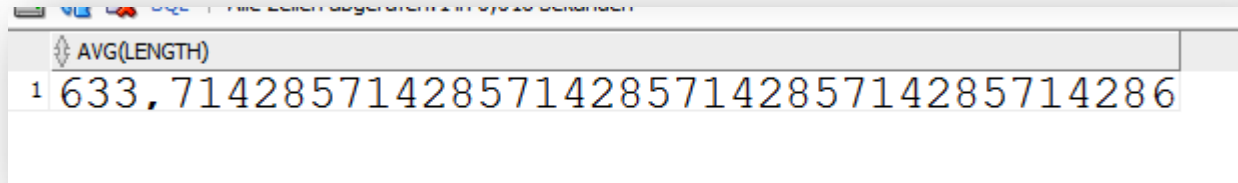
Verfügbare Aggregatsfunktionen

- Durchschnitt: **AVG**
- Minimum: **MIN**
- Maximum: **MAX**
- Summe: **SUM**
- Anzahl: **COUNT**

! Übung: Aggregation

Bestimmen Sie die durchschnittliche Länge aller Flüsse, die in die Nordsee (North Sea) fließen!

```
SELECT AVG(Length) FROM River  
WHERE Sea = 'North Sea';
```



The screenshot shows a database query result window. The title bar indicates the file path 'C:\Programme\Microsoft SQL Server\Enterprise Edition\sqlcmd.exe'. The window contains a table with one column named 'AVG(LENGTH)' and one row with the value '633,714285714285714285714285714286'.

AVG(LENGTH)
633,714285714285714285714285714286

Aggregation mit **DISTINCT**

In Sammlung vorhandene Duplikate spielen für manche Aggregatsfunktionen eine Rolle

! Übung: Welche?

Mit **DISTINCT** können Duplikate vor Anwendung der Aggregatsfunktion eliminiert werden

- Kann auf alle Aggregatsfunktionen angewendet werden, ändert aber teilweise am Ergebnis nichts!

Beispiele **DISTINCT**

```
SELECT AVG(Population) FROM City;
```

```
SELECT AVG(DISTINCT Population) FROM City;
```

```
SELECT COUNT(Population) FROM City;
```

```
SELECT COUNT(DISTINCT Population) FROM City;
```

```
SELECT SUM(Population) FROM City;
```

```
SELECT SUM(DISTINCT Population) FROM City;
```

```
SELECT MAX(Population) FROM City;
```

```
SELECT MAX(DISTINCT Population) FROM City;
```

Übung DISTINCT

Multimenge: $S = \{1,1,2,3\}$

Berechnen Sie:

$$\text{COUNT}(S) = 4$$

$$\text{COUNT}(\text{DISTINCT } S) = 3$$

$$\text{SUM}(S) = 7$$

$$\text{SUM}(\text{DISTINCT } S) = 6$$

$$\text{AVG}(S) = 7/4$$

$$\text{AVG}(\text{DISTINCT } S) = 6/3 = 2$$

$$\text{MIN}(S) = 1$$

$$\text{MAX}(S) = 3$$

Aggregatsfunktionen mit Null

- NULL-Werte in Sammlungen werden von allen Aggregatsfunktionen außer **COUNT** ignoriert

Beispiel:

```
SELECT * FROM NullBsp; -- mit Werten von oben
SELECT COUNT(*) FROM NullBsp;
SELECT MAX(Wert) FROM NullBsp;
```

- Für leere Sammlungen liefert **COUNT** den Wert 0, alle anderen Aggregatsfunktionen geben NULL zurück

Beispiel:

```
DELETE FROM NullBsp;
SELECT COUNT(*) FROM NullBsp;
SELECT MAX(Wert) FROM NullBsp;
```

Aggregation mit Gruppierung (1)

Manchmal wollen wir Aggregation nicht auf die ganze Tupel-Menge anwenden, sondern nur auf Gruppen von Tupeln

- Hierzu wird **GROUP BY**-Klausel verwendet

Beispiel: Für Relation $R(A, B, C, D)$:

```
SELECT MAX(C), AVG(D)
FROM R GROUP BY A, B;
```

Ohne explizite Gruppierung ist gesamte Relation eine Gruppe!

 Übung: Bestimmen Sie die durchschnittliche Einwohnerzahl der Städte pro Land!

```
SELECT Country, AVG(Population)
FROM City
GROUP BY Country;
```


Aggregation mit Gruppierung (2)

Gruppierung ist auch auf komplexeren Anfragen möglich

- Z.B. kartesisches Produkt oder **JOIN** in **FROM**-Klausel, Prädikate in **WHERE**-Klausel

! Übung: Bestimmen Sie jeweils die durchschnittliche Einwohnerzahl der Städte in Europa und Amerika

```
SELECT Encompasses.Continent, AVG(Population) AS Average
FROM City NATURAL JOIN Encompasses
WHERE Encompasses.Continent = 'Europe'
      OR Continent = 'America'
GROUP BY Encompasses.Continent;
```

Aggregation mit Gruppierung (3)

Bei Gruppierung dürfen in **SELECT**-Klausel nur Attribute außerhalb von Aggregatsfunktionen verwendet werden, die auch in **GROUP BY**-Klausel stehen

Beispiel: Gruppierung nach Kontinent und Land

```
SELECT Continent, Country, AVG(Population)
FROM City NATURAL JOIN Encompasses
GROUP BY Continent, Country;
```

Die **HAVING**-Klausel

Bedingungen können für Gruppen anstatt für einzelne Tupel definiert werden

- Hierzu wird **HAVING**-Klausel verwendet
- Diese wird ausgewertet, nachdem die Gruppen gebildet wurden, daher können Aggregatsfunktionen verwendet werden

Beispiel: Durchschnittliche Einwohnerzahl von Städten in Ländern, die mehr als 10 Städte in der Mondial-DB verzeichnet haben

```
SELECT Country, AVG (Population) AS Average  
FROM City GROUP BY Country  
HAVING COUNT (*) > 10;
```

Übung: **HAVING**-Klausel

Bestimmen Sie die durchschnittliche Einwohnerzahl aller Städte pro Land für Länder, in denen eine Stadt mit mehr als 1.000.000 Einwohnern liegt!

```
SELECT Country, AVG(Population) AS Average  
FROM City GROUP BY Country  
HAVING MAX(Population) > 1000000;
```

Auswertungsreihenfolge

1. **FROM**-Klausel bestimmt die Relation (ggf. mit kartesischem Produkt, Join)
2. **WHERE**-Klausel filtert die Tupel der Relation aus 1)
3. **GROUP BY**-Klausel bildet Gruppen
4. **HAVING**-Klausel filtert Gruppen
5. **SELECT**-Klausel gibt Attribute der Gruppen aus bzw. berechnet Aggregatsfunktionen

Geschachtelte Abfragen / Unterabfragen

Es ist möglich, dass innerhalb einer Abfrage

SELECT ... **FROM** ... **WHERE** ...

weitere **SELECT**-Statements als **Unterabfragen** vorkommen:

- In **WHERE**-Klausel
- In **FROM**-Klausel
- Als **skalare Unterabfragen**

Mitgliedschaft in Mengen (1)

In **WHERE**-Klausel erlaubt Operator **IN** Test einer Mengen-Mitgliedschaft

- Menge kann dabei durch ein **SELECT**-Statement erzeugt werden

Beispiel: Durchschnittliche Einwohnerzahl aller Städte in Afrika und Europa

1) Menge der betroffenen Städte in Europa oder Afrika:

```
SELECT City.Name  
  FROM City NATURAL JOIN Encompasses  
  WHERE Continent = 'Europe' OR Continent = 'Africa';
```

2) Berechnung der durchschnittlichen Einwohnerzahl

```
SELECT AVG(Population) FROM City  
  WHERE Name IN  
    (SELECT City.Name  
      FROM City NATURAL JOIN Encompasses  
      WHERE Continent = 'Europe' OR Continent = 'Africa');
```

(wir vernachlässigen, dass Städtenamen auch auf anderen Kontinenten vorkommen können)

Mitgliedschaft in Mengen (2)

- Analog zu **IN** gibt es **NOT IN**
- Eingabemengen für **IN** bzw. **NOT IN** können auch „per Hand“ anstatt mit **SELECT**-Statement gebildet werden

Beispiel: Städte in Europa oder Afrika

```
SELECT City.Name  
  FROM City NATURAL JOIN Encompasses  
  WHERE Continent IN ('Europe', 'Africa');
```


! Übung: Mitgliedschaft in Mengen

Bestimmen Sie den Namen aller Flüsse, die in ein Meer (Sea) münden, das tiefer als 10.000m ist. Verwenden Sie dazu den **IN**-Operator!

```
SELECT Name FROM River
WHERE Sea IN (SELECT Name FROM Sea WHERE Depth > 10000);
```

Mengenvergleiche (1)

Manchmal, ist es notwendig, einen Wert mit einer Menge von Werten zu vergleichen.

- Beispiel: Ist ein Fluss länger als alle Flüsse, die in die Nordsee fließen?

Hier nutzt man den Operator **>ALL**

Beispiel: Namen der Flüsse, die länger sind als alle Flüsse, die in die Nordsee münden

- 1) Längen aller Flüsse, die in die Nordsee münden (Menge):

SELECT Length **FROM** River **WHERE** SEA = 'North Sea';

- 2) Ergebnis durch Mengenvergleich

SELECT Name **FROM** River
WHERE Length **>ALL** (**SELECT** Length **FROM** River
WHERE SEA = 'North Sea');

Folie 63

River (vereinfacht):

Name	Length	Sea
Rhein	1000	North Sea
Elbe	1200	North Sea
Amazonas	2000	Atlantic Ocean
Nordseeufer	300	Baltic Sea

Ergebnis von 1)

Length
1000
1200

2) lässt sich schreiben als:

SELECT Name **FROM** River
WHERE LENGTH
>ALL (1000, 1200)

> SOME

Mengenvergleiche (2)

Es gibt folgende Operatoren:

<ALL	<SOME
<=ALL	<=SOME
	=SOME
>=ALL	>=SOME
>ALL	>SOME
<>ALL	<>SOME

Beispiel >SOME:

Flüsse die Länger sind als
einer oder mehrere, die in
die Nordsee fließen:

```
SELECT Name
FROM River
WHERE Length >SOME
      (SELECT Length
       FROM River
       WHERE SEA =
         'North Sea');
```

! Übung: Mengenvergleiche

Finden Sie die Namen aller Berge, die niedriger sind als ein Berg in Großbritannien (Ländercode ,GB')!

Relation Mountain:

Name	Null	Typ
-----	-----	-----
NAME	NOT NULL	VARCHAR2 (50)
MOUNTAINS		VARCHAR2 (50)
ELEVATION		NUMBER
TYPE		VARCHAR2 (10)
COORDINATES		GEOCOORD

Relation Geo_Mountain:

Name	Null	Typ
-----	-----	-----
MOUNTAIN	NOT NULL	VARCHAR2 (50)
COUNTRY	NOT NULL	VARCHAR2 (4)
PROVINCE	NOT NULL	VARCHAR2 (50)

```

SELECT Elevation
FROM Mountain JOIN Geo_Mountain
ON Mountain.Name = Geo_Mountain.Mountain
WHERE Country='GB';

SELECT Name, Elevation
FROM Mountain
WHERE Elevation < SOME
(SELECT Elevation
FROM Mountain JOIN Geo_Mountain
ON Mountain.Name = Geo_Mountain.Mountain
WHERE Country='GB');

```

Test auf leere Mengen / Relationen

Operator **EXISTS** prüft, ob Ergebnis einer Unterabfrage leer ist (oder mindestens ein Tupel enthält)

Beispiel: Insel und Ländercode für alle vulkanischen Inseln

```
SELECT Island, Country
FROM Geo_Island
WHERE EXISTS (SELECT * FROM Island
               WHERE Name = Island
               AND Type = 'volcanic');
```

Analog gibt es auch **NOT EXISTS**

! Übung: Test auf leere Mengen / Relationen

Finden Sie die Namen aller Länder, in denen Deutsch gesprochen wird, mit Hilfe des **EXISTS**-Operators!

Relation Country:

Name	Null	Typ
-----	-----	-----
NAME	NOT NULL	VARCHAR2 (50)
CODE	NOT NULL	VARCHAR2 (4)
CAPITAL		VARCHAR2 (50)
PROVINCE		VARCHAR2 (50)
AREA		NUMBER
POPULATION		NUMBER

Relation Language:

Name	Null	Typ
-----	-----	-----
COUNTRY	NOT NULL	VARCHAR2 (4)
NAME	NOT NULL	VARCHAR2 (50)
PERCENTAGE		NUMBER

```
SELECT Name
FROM Country
WHERE EXISTS ( SELECT * FROM Language
                WHERE Name = 'German'
                AND Country = Code );
```

Unterabfragen in der **FROM**-Klausel

Ergebnis eines **SELECT**-Statements ist eine Relation
→ kann also in **FROM**-Klausel verwendet werden!

Beispiel: Maximum der durchschnittlichen Stadtbevölkerung aller Länder

1) Tabelle mit durchschnittlicher Stadtbevölkerung:

```
SELECT Country, AVG(Population) AS Average  
FROM City GROUP BY Country;
```

2) Maximum der Tabelle aus 1)

```
SELECT MAX(Average)  
FROM( SELECT Country, AVG(Population) AS Average  
FROM City GROUP BY Country );
```

! Übung: Unterabfragen in der **FROM**-Klausel

1. Bestimmen Sie die Höhe des jeweils niedrigsten Berges pro Land!
2. Bestimmen Sie das Maximum über das Ergebnis von 1)!

Relation Mountain:

Name	Null	Typ
NAME	NOT NULL	VARCHAR2 (50)
MOUNTAINS		VARCHAR2 (50)
ELEVATION		NUMBER
TYPE		VARCHAR2 (10)
COORDINATES		GEOCOORD

Relation Geo_Mountain:

Name	Null	Typ
MOUNTAIN	NOT NULL	VARCHAR2 (50)
COUNTRY	NOT NULL	VARCHAR2 (4)
PROVINCE	NOT NULL	VARCHAR2 (50)

```
SELECT Country, MIN(Elevation) AS MinElevation
FROM Mountain JOIN Geo_Mountain
ON Mountain.Name = Geo_Mountain.Mountain
GROUP BY Country;

SELECT MAX(MinElevation)
FROM ( SELECT Country, MIN(Elevation) AS MinElevation
FROM Mountain JOIN Geo_Mountain
ON Mountain.Name = Geo_Mountain.Mountain
GROUP BY Country );
```


Die **WITH**-Klausel

Temporäre Relationen können mit **WITH**-Klausel definiert werden

- Nur gültig innerhalb des **SELECT**-Statements, dem **WITH**-Klausel vorangestellt wurde

Beispiel: Andere Darstellung Beispiel Folie 68

```
WITH AvgTab AS  
    (SELECT Country, AVG(Population) AS Average  
     FROM City GROUP BY Country)  
SELECT MAX(Average) FROM AvgTab;
```

! Übung: **WITH**-Klausel

Bestimmen Sie die Lösung der Übung von Folie 70 mit Hilfe der **WITH**-Klausel!

```
WITH MinElevation AS
( SELECT Country, MIN(Elevation) AS MinE
  FROM Mountain JOIN Geo_Mountain
    ON Mountain.Name = Geo_Mountain.Mountain
  GROUP BY Country )
SELECT MAX(MinE)
FROM MinElevation;
```

Skalare Unterabfragen

Ergebnisse von Unterabfragen können als skalarer Wert in Prädikat verwendet werden, wenn Unterabfrage nur eine einziges Tupel mit einem einzigen Attribut zurück liefert

Beispiel: Insel mit höchster Erhebung (vgl. Übung)

```
SELECT MAX(Elevation) FROM Island;
```

```
SELECT Name, Elevation  
  FROM Island  
 WHERE Elevation=  
        (SELECT MAX(Elevation) FROM Island);
```

! Übung: Skalare Unterabfragen

Welche Länder(-codes) haben den kleinsten Anteil an katholischer ('Roman Catholic') Bevölkerung?

Relation Religion:

Name	Null	Typ
-----	-----	-----
COUNTRY	NOT NULL	VARCHAR2 (4)
NAME	NOT NULL	VARCHAR2 (50)
PERCENTAGE		NUMBER

```
SELECT MIN(Percentage) FROM Religion
WHERE Name = 'Roman Catholic';

SELECT Country, Name, Percentage
FROM Religion
WHERE Percentage = ( SELECT MIN(Percentage) FROM Religion
                     WHERE Name = 'Roman Catholic' )
AND Name = 'Roman Catholic';
```

Datenbankmodifikationen

Datensätze können über folgende Statements modifiziert werden:

- **INSERT**-Statement: Einfügen von Datensätzen
- **UPDATE**-Statement: Aktualisieren von Datensätzen
- **DELETE**-Statement: Löschen von Datensätzen

INSERT-Statement (1)

Im Folgenden verwendete Beispieeltabelle:

```
CREATE TABLE MyRiver (  
    Name VARCHAR(50),  
    Country VARCHAR(4),  
    Length NUMBER );
```

Beispiele für Insert-Statements (vgl. Kapitel 6):

-- Alle Werte, Reihenfolge wie in CREATE TABLE Statement

```
INSERT INTO MyRiver VALUES ('Ruhr', 'D', 219 );
```

-- Werte in Reihen abweichend von CREATE TABLE Statement

```
INSERT INTO MyRiver(Length, Country, Name)  
VALUES (93, 'D', 'Kinzig');
```

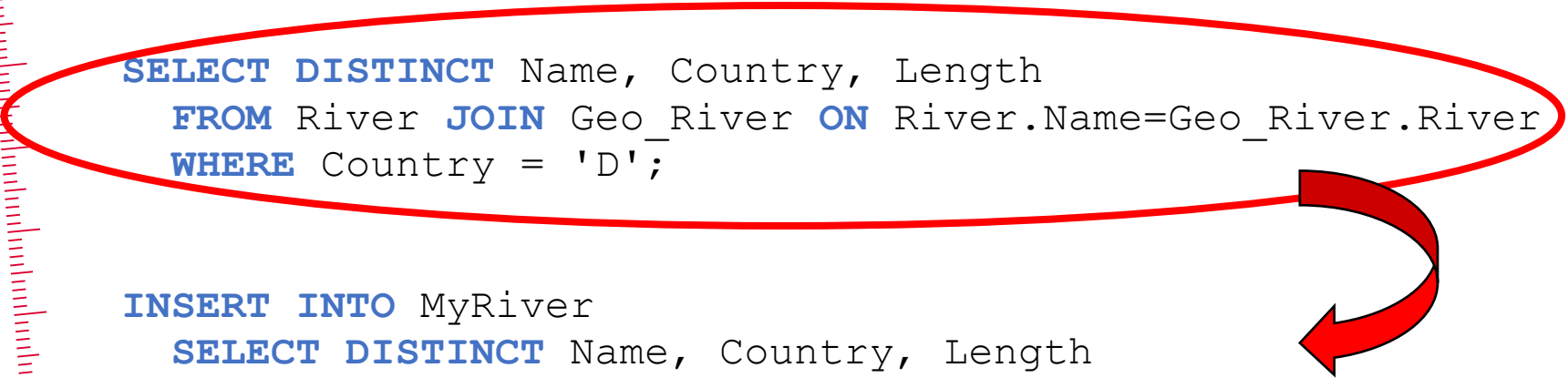
-- Nur ausgewählte Werte

```
INSERT INTO MyRiver(Name, Country)  
VALUES ('Alb', 'D');
```

INSERT-Statement (2)

Durch **INSERT**-Statement eingesetzte Tupel können auch Ergebnis einer **SELECT**-Abfrage sein

Beispiel: Alle deutschen Flüsse aus Mondial-Datenbank in Tabelle `MyRiver` einfügen:



```
SELECT DISTINCT Name, Country, Length  
FROM River JOIN Geo_River ON River.Name=Geo_River.River  
WHERE Country = 'D';
```

```
INSERT INTO MyRiver  
SELECT DISTINCT Name, Country, Length  
FROM River JOIN Geo_River ON River.Name=Geo_River.River  
WHERE Country = 'D';
```

! Übung: **INSERT**-Statement

Fügen sie alle Flüsse in Großbritannien (Country-Code 'GB') zur Tabelle MyRiver hinzu!

Relation River:

Name	Null	Typ
-----	-----	-----
NAME	NOT NULL	VARCHAR2 (50)
RIVER		VARCHAR2 (50)
LAKE		VARCHAR2 (50)
SEA		VARCHAR2 (50)
LENGTH		NUMBER
AREA		NUMBER
SOURCE		GEOCOORD
MOUNTAINS		VARCHAR2 (50)
SOURCEELEVATION		NUMBER
ESTUARY		GEOCOORD

```

INSERT INTO MyRiver
SELECT DISTINCT Name, Country, Length
FROM River JOIN Geo_River
ON River.Name=Geo_River.River
WHERE Country = 'GB';
  
```


Ausführungsreihenfolge **INSERT**-Statement

Im Allgemeinen wird **SELECT**-Statement ausgeführt, bevor **INSERT**-Statement ausgeführt wird

- Wichtig, wenn **INSERT** und **SELECT** auf gleicher Relation ausgeführt wird!

Beispiel: Ausführungsreihenfolge

```
INSERT INTO MyRiver SELECT * FROM MyRiver;
```

! Was passiert?

UPDATE-Statement (1)

Allgemeine Form **UPDATE**-Statement:



UPDATE-Klausel

Gibt betroffene Relation an

SET-Klausel

Gibt an, was aktualisiert werden soll

WHERE-Klausel

Wählt zu aktualisierende Tupel

UPDATE-Statement (2)

SET-Klausel kann komplexe Berechnungen enthalten

Beispiel: Umrechnung aller Flusslängen in Meilen

```
UPDATE MyRiver  
  SET Length=Length/1.6
```

Beispiel: Umrechnung nur für Flüsse in Großbritannien (mit WHERE)

```
UPDATE MyRiver  
  SET Length=Length/1.61  
  WHERE Country='GB';
```

Übung: Update-Statement

Machen Sie die Umrechnung in Meilen für Flüsse in Großbritannien rückgängig!

```
UPDATE MyRiver  
  SET Length=Length*1.61  
 WHERE Country='GB';
```

UPDATE und skalare Unterabfragen

Analog zu **SELECT**-Statement können in **WHERE**-Klausel skalare Unterabfragen verwendet werden

Beispiel: Namen aller Flüsse in Großbuchstaben umwandeln, wenn die Länge des Flusses größer als der Durchschnitt ist

```
UPDATE MyRiver
  SET Name=UPPER(Name)
 WHERE Length >=
    (SELECT AVG(Length) FROM MyRiver);
```

Übung: Update und skalare Unterabfragen

Wandeln Sie den Namen des kürzesten Flusses / der kürzesten Flüsse in Kleinbuchstaben um!

```
UPDATE MyRiver  
  SET Name=LOWER(Name)  
 WHERE Length =  
   (SELECT MIN(Length) FROM MyRiver);
```

Aktualisierungen kombinieren mit **CASE**

Um Probleme mit Reihenfolge von Aktualisierungen zu vermeiden, können mehrere Aktualisierungen mit **CASE**-Schlüsselwort kombiniert werden

Beispiel: Verlängerung von Flüssen über 440 km

```
UPDATE MyRiver  
  SET Length = CASE  
    WHEN Length <= 440 THEN Length * 1.1  
    ELSE Length * 1.3  
  END;
```

! Warum nicht das gleiche Ergebnis wie 2x **UPDATE**?

Übung: **CASE**-Schlüsselwort


Stellen Sie allen Flussnamen, die mit einem 'A' beginnen, ein 'B' voran, allen anderen Flussnamen ein 'A' . Verwenden Sie dabei das **CASE**-Schlüsselwort!

```
UPDATE MyRiver
SET Name = CASE
    WHEN Name LIKE 'A%' THEN CONCAT('B', Name)
    ELSE CONCAT('A', Name)
END;
```


DELETE-Statement (1)

Allgemeine Form **DELETE**-Statement:

DELETE **FROM** R **WHERE** β



FROM-Klausel **WHERE-Klausel**

FROM-Klausel

Gibt betroffene Relation an

WHERE-Klausel

Wählt zu löschende Tupel

DELETE-Statement (2)

Beispiel: Löschen aller Flüsse mit a als letztem Buchstaben

```
DELETE FROM MyRiver  
WHERE Name LIKE '%a';
```

Beispiel: Löschen aller Flüsse mit einer Länge zwischen 80 und 100 Kilometer

```
DELETE FROM MyRiver  
WHERE Length BETWEEN 50 AND 100;
```

DELETE-Statement und Unterabfragen

Analog **SELECT**-Statement: Unterabfragen in **WHERE**-Klausel sind möglich

- Auch über andere Relationen, als die, aus der gelöscht wird!

Beispiel: Bayrische Flüsse löschen

```
SELECT River FROM Geo_River  
WHERE Province='Bayern';
```

```
DELETE FROM MyRiver  
WHERE Name IN  
  (SELECT River FROM Geo_River  
   WHERE Province='Bayern');
```



Ausführungsreihenfolge **DELETE**-Statement

1. Zuerst bestimmt **WHERE**-Klausel Tupel
2. Dann wird gelöscht

Ansonsten könnte sich das Ergebnis des Prädikats verändern!

Beispiel: Löschen aller Flüsse mit einer Länge über Durchschnitt

```
DELETE FROM MyRiver  
  WHERE Length >  
    (SELECT AVG (Length) FROM MyRiver);
```

Zusammenfassung Kapitel 7

SQL bietet in **SELECT**-Statement mehr als nur Abbildung der relationalen Algebra

- Operatoren in **SELECT**-Klausel
- String-Funktionen
- Sortieren der Ausgabe
- Bedingungen für Gruppen mit **HAVING**-Klausel
- Geschachtelte Unterabfragen
- ...

Datenbankmodifikationen mit **INSERT**, **UPDATE**, **DELETE** verwenden aus **SELECT**-Statement bekannte Konzepte



Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

Kapitel 5 Fortgeschrittenes SQL

Vorlesung Datenbanken und Informationssysteme I
Sommersemester 2017

Prof. Dr. rer. nat. Oliver P. Waldhorst

Inhalte dieses Kapitels

- Sichten (Views)
- Transaktionen
- Erweiterte Integritätsbedingungen
- Default-Werte
- Index-Erzeugung
- Sequenzen
- Benutzerdefinierte Datentypen
- Data Control Language

Sichten (Views)

Bisher haben wir immer auf logischem Datenbank-Modell gearbeitet. Das ist nicht immer wünschenswert!

Beispiele:

- Benutzer darf nicht alle Informationen sehen
- Benutzer benötigt nur gefilterte Informationen
- Benutzer benötigt nur aggregierte Informationen, die nicht direkt im Datenbank-Modell zu finden sind

Anwendungsfall Sichten / Views

Zur Illustration nutzen wir die folgenden Relationen:

```
CREATE Table Sekretariat (  
    Abteilung VARCHAR(30),  
    Gebaeude VARCHAR(30),  
    PRIMARY KEY (Abteilung) );
```

```
CREATE TABLE Dozent (  
    Name VARCHAR(30),  
    Abteilung VARCHAR(30),  
    Gehalt NUMBER,  
    PRIMARY KEY (Name),  
    FOREIGN KEY (Abteilung) REFERENCES Sekretariat );
```

Anwendungsfall Sichten / Views

-- *Hochschul-Verwaltung Sekretariat*

```
SELECT Name, Abteilung FROM Dozent;
```

-- *Personalverwaltung Informatik*

```
SELECT * FROM Dozent WHERE Abteilung='Informatik';
```

-- *Hauspost*

```
SELECT Name, Gebaeude  
      FROM Dozent NATURAL INNER JOIN Sekretariat;
```

-- *Rektorat*

```
SELECT Abteilung, SUM(Gehalt)  
      FROM Dozent GROUP BY Abteilung;
```

Sichten (Views)

Warum nicht eine Kopie der Daten mit **INSERT** erstellen?

- Problem bei Updates!

SQL definiert das Konzept von **Sichten (Views)**:

- Virtuelle Relation
- Nicht gespeichert, sondern immer bei Bedarf neu berechnet

Syntax: **CREATE VIEW** *<View>* **AS** *<Anfrage>*

- *<View>* Name des Views (kann wie Relation verwendet werden)
- *<Anfrage>* **SELECT**... Anfrage, die View definiert, s.o.

Beispiele: Zielgruppenorientierte Views

Syntax: **CREATE VIEW** <View> **AS** <Anfrage>

View für Sekretariat der Hochschulverwaltung:

```
CREATE VIEW Professoren AS  
  SELECT Name, Abteilung FROM Dozent;
```

View für Personalverwaltung Informatik:

```
CREATE VIEW InformatikDozenten AS  
  SELECT * FROM Dozent WHERE Abteilung='Informatik';
```

View für Hauspost:

```
CREATE VIEW Postadressen AS  
  SELECT Name, Gebaeude  
    FROM Dozent NATURAL INNER JOIN Sekretariat;
```

Views und Aggregatsfunktionen

Achtung: Bei Verwendung von Views zusammen mit Aggregatsfunktionen ist Umbenennung der Attribute notwendig!

- Kann entweder in **SELECT**-Anfrage oder bei der **VIEW**-Definition vorgenommen werden

Beispiel: View für Rektorat

```
CREATE VIEW Abteilungsgehaelter AS
  SELECT Abteilung, SUM(Gehalt)
    FROM Dozent GROUP BY Abteilung; -- funktioniert nicht!
```

```
CREATE VIEW Abteilungsgehaelter AS
  SELECT Abteilung, SUM(Gehalt) AS GehaltsSumme
    FROM Dozent GROUP BY Abteilung; -- funktioniert
```

```
CREATE VIEW Abteilungsgehaelter(Abteilungsname, Gehaltssumme) AS
  SELECT Abteilung, SUM(Gehalt)
    FROM Dozent GROUP BY Abteilung; -- funktioniert
```

Übung: Views

Erstellen Sie eine View für die Dozenten der Abteilung Physik mit den Gebäuden ihrer Sekretariate!

```
CREATE VIEW Physikdozenten AS  
  SELECT * FROM Dozent NATURAL INNER JOIN Sekretariat  
  WHERE Abteilung = 'Physik';
```

Materialisierte Sichten / Materialized Views

On-Demand Berechnung von Views kann Leistung bei Abfragen beeinflussen

- Gegenmaßnahme: Manche DBMS berechnen Sichten vor und speichern diese
- Dieses Konzept wird als **Materialisierte Sichten / Materialized View** bezeichnet
- In Oracle: **CREATE MATERIALIZED VIEW . . .**



Materialisierte Sichten / Materialized Views

Herausforderung für DBMS

- View aktualisieren, wenn sich zugrundeliegende Tabellen ändern

Ansätze:

- Direkt: Bei jeder Änderung der Tabellen wird der View neu berechnet
- Bei Zugriff („lazy“): Bei jedem Zugriff wird der View (falls nötig) neu berechnet
- Periodisch: Es findet eine regelmäßige Neuberechnung statt

Aktualisierungen vs. Views (1)

View kann prinzipiell wie Relation verwendet werden

- **UPDATE** möglich?
- Schwierig, da u.U.
 - nicht alle Attribute einer Relation im View
 - mehr als eine Relation betroffen

SQL erlaubt Updates unter bestimmten Bedingungen

- Nur eine Relation in **FROM**-Klausel
- Nur Attributsnamen in **SELECT**-Klausel (und keine Ausdrücke, Aggregatsfunktionen, **DISTINCT**-Statement)
- Alle Attribute außerhalb von SELECT können NULL sein (insbesondere nicht Teil des Primärschlüssels)
- Keine **GROUP BY** / **HAVING** Klauseln

Übung: Welche Views sind aktualisierbar?

```
CREATE VIEW Professoren AS
  SELECT Name, Abteilung FROM Dozent;
```

```
CREATE VIEW InformatikDozenten AS
  SELECT * FROM Dozent WHERE Abteilung='Informatik';
```

```
CREATE VIEW Postadressen AS
  SELECT Name, Gebaeude
  FROM Dozent NATURAL INNER JOIN Sekretariat;
```

```
CREATE VIEW Abteilungsgehaelter(Abteilungsname, Gehaltssumme) AS
  SELECT Abteilung, SUM(Gehalt)
  FROM Dozent GROUP BY Abteilung;
```

Aktualisierungen vs. Views (2)

Ggf. ist die **WHERE**-Klausel des zugrundeliegenden **SELECT**-Statements für eingefügtes Tupel nicht erfüllt

- Tupel erscheint nicht im View

Beispiel: Physik-Professor in View InformatikDozenten einfügen

```
INSERT INTO InformatikDozenten VALUES  
('Planck', 'Physik', 54000);
```

WITH CHECK OPTION erzwingt, dass eingefügte Tupel **WHERE**-Klausel erfüllen

Beispiel: View InformatikDozenten mit Überprüfung

```
CREATE VIEW InformatikDozenten AS  
SELECT * FROM Dozent WHERE Abteilung='Informatik'  
WITH CHECK OPTION;
```

Erweiterte Tabellenerzeugung

Tabellen lassen sich direkt aus einer Suchanfrage erstellen

- Spart ein Statement gegenüber
CREATE TABLE...; INSERT INTO...;

Syntax: **CREATE TABLE** *<Tabelle>* **AS** *<Anfrage>*

- *<Tabelle>* Name der Tabelle
- *<Anfrage>* **SELECT**... Anfrage, die Tabelle definiert

Syntax ist bewusst analog zur Erzeugung von Views gehalten!

Transaktionen

SQL definiert **Transaktion** als Reihe von zusammengehörigen DML-Kommandos (**SELECT**, **UPDATE**, **INSERT**, **DELETE**)

Transaktion wird beendet durch

- **COMMIT WORK** → Festschreiben der Kommando-Ergebnisse
- **ROLLBACK WORK** → Rückgängigmachen der Kommandos
- **“WORK”** ist optional

Transaktionen sind **atomar**, d.h. nicht zerteilbar

- Es werden entweder alle Kommandos ausgeführt oder keins

Beispiel: Bank-Transaktion Konto-Transfer

- Abheben von einem Konto, Gutschreiben auf einem anderen Konto
- Es sollten nur beide Operationen durchgeführt werden oder keine!

Transaktionen

Wenn Datenbankanwendung abstürzt, werden entweder alle Kommandos einer ausstehenden Transaktion ausgeführt oder alle zurückgerollt

Transaktionen in Oracle...

- ... beginnen implizit durch DML- oder DDL-Kommando oder explizit durch **SET TRANSACTION** Statement
- ... enden durch DDL-Kommando oder **COMMIT** oder **ROLLBACK** Statement

Integritätsbedingungen, zum Zweiten

Integritätsbedingungen (Integrity Constraints) verhindern Beschädigung der Datenbank-Integrität durch Benutzereingaben

Beispiel: Relationen Sekretariat, Dozent von oben

- Dozentenname kann nicht NULL sein
- Keine zwei Dozenten mit selben Namen
- Jeder Wert von Dozent.Abteilung muss auf existierendes Tupel in Sekretariat verweisen

SQL definiert folgende Integritätsbedingungen auf einer Relation:

- **PRIMARY KEY**
- **NOT NULL**
- **UNIQUE**
- **CHECK** (*<Prädikat>*)

Constraint CHECK

CHECK (<Prädikat>) stellt sicher, dass jedes eingefügte Tupel <Prädikat> erfüllt.

Beispiel: Nur ausreichend hohe Gehälter einfügen

```
CREATE TABLE Dozent (  
    Name VARCHAR(30),  
    Abteilung VARCHAR(30),  
    Gehalt NUMBER,  
    PRIMARY KEY (Name),  
    FOREIGN KEY (Abteilung) REFERENCES Sekretariat,  
    CHECK (Gehalt >= 65000) );
```

```
INSERT INTO Dozent  
VALUES ( 'Einstein', 'Physik', 60000 ); -- funktioniert nicht
```

```
INSERT INTO Dozent  
VALUES ( 'Einstein', 'Physik', 65000 ); -- funktioniert
```


Referentielle Integrität

Fremdschlüsselbedingungen (Foreign Key Constraints) stellen referentielle Integrität (Gültigkeit von Verweisen zwischen Relationen) sicher

- Fremdschlüssel kann auf Attribute einer Relation verweisen, die als **PRIMARY KEY** oder **UNIQUE** definiert sind

Syntax:

```
... FOREIGN KEY (<Attr1>, ..., <AttrN>)  
    REFERENCES <Relation>(<FAttr1>, ..., <FAttrN>), ...
```

Alternativ wenn nur Fremdschlüssel nur aus einem Attribut:

```
... <Attr> REFERENCES <Relation>(<FAttr1>), ...
```

Beispiel: Referentielle Integrität

```
CREATE TABLE Dozent (  
    Name VARCHAR(30),  
    Abteilung VARCHAR(30),  
    Gehalt NUMBER,  
    PRIMARY KEY (Name),  
    FOREIGN KEY (Abteilung) REFERENCES Sekretariat );
```

oder

```
CREATE TABLE Dozent (  
    Name VARCHAR(30),  
    Abteilung VARCHAR(30) REFERENCES Sekretariat(Abteilung),  
    Gehalt NUMBER,  
    PRIMARY KEY (Name) );
```

Referentielle Integrität und Aktualisierungen

Was passiert, wenn wir bei Fremdschlüsselbeziehungen

- ... referenzierte Tupel löschen?
- ... das referenzierte Attribut verändern?

Beispiel: Löschen und Ändern von referenzierten Tupeln

```
UPDATE Sekretariat  
  SET Abteilung='InformatikNeu'  
  WHERE Abteilung='Informatik'; -- funktioniert nicht
```

```
DELETE FROM Sekretariat  
  WHERE Abteilung='Informatik'; -- funktioniert nicht
```

Referentielle Integrität und Aktualisierungen

Zusätze

```
... ON DELETE CASCADE  
... ON UPDATE CASCADE*)
```

bei Definition der Fremdschlüsselbeziehung sorgen dafür, dass referenzierende Tabelle aktualisiert wird

**) SQL Standard, aber in Oracle nicht implementiert*

Beispiel: ON DELETE CASCADE

```
CREATE TABLE Dozent (  
  Name VARCHAR(30),  
  Abteilung VARCHAR(30),  
  Gehalt NUMBER,  
  PRIMARY KEY (Name),  
  FOREIGN KEY (Abteilung) REFERENCES Sekretariat  
    ON DELETE CASCADE );
```

```
DELETE FROM Sekretariat WHERE Abteilung='Physik'; -- löscht auch Physik-Dozenten
```

Referentielle Integrität und Aktualisierungen

Zusatz **CASCADE** kann durch **SET NULL** oder **SET DEFAULT** ersetzt werden

Beispiel: ON DELETE SET NULL

```
CREATE TABLE Dozent (  
    Name VARCHAR(30),  
    Abteilung VARCHAR(30),  
    Gehalt NUMBER,  
    PRIMARY KEY (Name),  
    FOREIGN KEY (Abteilung) REFERENCES Sekretariat  
        ON DELETE SET NULL);
```

```
DELETE FROM Sekretariat WHERE Abteilung='Physik'; -- Setzt Abt. Physik =  
NULL
```

Referentielle Integrität und Transaktionen

Manchmal lassen sich Integritätsbedingungen nur schwer einhalten

Beispiel: Ehepaare in einer Tabelle

```
CREATE TABLE Ehepaare (  
    Name VARCHAR(30),  
    Partner VARCHAR(30),  
    PRIMARY KEY (Name),  
    FOREIGN KEY (Partner) REFERENCES Ehepaare(Name) );
```

```
INSERT INTO Ehepaare VALUES ('John', 'Mary'); -- funktioniert nicht
```

Fremdschlüsselattribute können NULL sein, daher gibt es eine (umständliche) Lösung

Beispiel: Einfügen mit Hilfe von NULL-Werten

```
INSERT INTO Ehepaare VALUES ('John', NULL);  
INSERT INTO Ehepaare VALUES ('Mary', 'John');  
UPDATE Ehepaare SET Partner='Mary' WHERE Name='John'; -- funktioniert
```

Referentielle Integrität und Transaktionen

Einfacher unter Nutzung des Transaktionskonzepts

- Fremdschlüsselbedingung kann während Transaktion verletzt sein, muss aber am Ende der Transaktion eingehalten werden
- Schlüsselwort: ... **INITIALLY DEFERRED**

Beispiel: Korrektes Einfügen bis zum Transaktionsende

```
CREATE TABLE Ehepaare (  
    Name VARCHAR(30),  
    Partner VARCHAR(30),  
    PRIMARY KEY (Name),  
    FOREIGN KEY (Partner) REFERENCES Ehepaare(Name) INITIALLY DEFERRED );
```

```
INSERT INTO Ehepaare VALUES ('John', 'Mary');  
INSERT INTO Ehepaare VALUES ('Mary', 'John');  
COMMIT;
```

Referentielle Integrität und Transaktionen

Beispiel: Fremdschlüsselbeziehung am Transaktionsende verletzt

-- Bei leerer Relation Ehepaare

INSERT INTO Ehepaare **VALUES** ('John', 'Mary');

SELECT * **FROM** Ehepaare;

COMMIT; *-- Tupel ('John', , 'Mary') wird gelöscht*

SELECT * **FROM** Ehepaare;

Default-Werte

Pro Attribut können Default-Werte festgelegt werden

- Syntax: *<Attribut> <Typ> DEFAULT <Wert>*

Beispiel: Standarddozentengehalt 65.000 Euro

```
CREATE TABLE Dozent (  
    Name VARCHAR(30),  
    Abteilung VARCHAR(30),  
    Gehalt NUMBER DEFAULT 65000,  
    PRIMARY KEY (Name),  
    FOREIGN KEY (Abteilung) REFERENCES Sekretariat);
```

```
INSERT INTO Dozent(Name, Abteilung)  
VALUES ( 'Einstein', 'Physik' );  
SELECT * FROM Dozent;
```

Index-Erzeugung

Indizes ermöglichen die schnelle Suche auf einem/mehreren Attributen

- Vermeidet Durchlaufen aller Attribute bei einer Suche
- Index wird von DBMS in der Regel durch Suchbäume realisiert
- Wenn Index vorhanden, wird er vom DBMS automatisch verwendet
- Syntax:

```
CREATE INDEX <Name> ON <Relation>(<Attr1>, ..., <AttrN>);
```

Beispiel: Index für Abteilungen

```
CREATE INDEX AbteilungsIndex ON Dozent (Abteilung);  
SELECT Name, Abteilung, Gehalt  
  FROM Dozent  
  WHERE Abteilung='Physik';
```

Automatisch generierte Primärschlüssel

Wiederholung: Es gibt *natürliche* und *künstliche* Primärschlüssel

- Natürlich: Attribut aus dem „wahren Leben“, z.B. Name, Matrikelnummer, ...
- Künstlich: Nur für die Verwendung in der Datenbank definiert

Herausforderung: Eindeutige künstliche Primärschlüssel vergeben

- DBMS kann dabei mit **CREATE SEQUENCE** Statement unterstützen
- Es lassen sich Minimum, Maximum, Schrittweite, Verhalten bei Überlauf angeben (s. Oracle-Dokumentation)

Automatisch generierte Primärschlüssel

Beispiel: Dozenten mit künstlichem Primärschlüssel DozentenId, generiert über Sequenz

```
CREATE TABLE DozentMitId (  
    DozentenId NUMBER,  
    Name VARCHAR(30),  
    Abteilung VARCHAR(30),  
    Gehalt NUMBER DEFAULT 65000,  
    PRIMARY KEY (DozentenId),  
    FOREIGN KEY (Abteilung) REFERENCES Gebaeude );
```

```
CREATE SEQUENCE SeqDozentId;
```

```
INSERT INTO DozentMitId VALUES ( SeqDozentId.NEXTVAL, 'Einstein', 'Physik', 60000 );  
INSERT INTO DozentMitId VALUES ( SeqDozentId.NEXTVAL, 'Hawking', 'Physik', 75000 );  
INSERT INTO DozentMitId VALUES ( SeqDozentId.NEXTVAL, 'Turing', 'Informatik', 87000 );  
INSERT INTO DozentMitId VALUES ( SeqDozentId.NEXTVAL, 'Berners-Lee', 'Informatik', 92000 );
```

Benutzerdefinierte Datentypen

Häufig haben mehrere Attribute in einer Relation aus technischer Sicht die gleichen Datentypen

Beispiel: Name und Abteilung in Relation Dozent sind beide vom Typ VARCHAR (30) - die Zuweisung einer Abteilung als Name ist in den meisten Fällen aber wahrscheinlich unbeabsichtigt (z.B. Programmierfehler)

DBMS kann durch die Definition von Datentypen unterstützen

- **SET** <Type> / **DROP** <Type> / **ALTER** <Type>
- **Achtung: SQL-Standard, aber von Oracle so nicht unterstützt!**

Benutzerdefinierte Datentypen

Beispiel: Datentypen für Name und Gebäude

```
CREATE TYPE Namen AS VARCHAR(30);
```

```
CREATE TYPE Abteilungen AS VARCHAR(30);
```

```
CREATE TABLE Dozent (  
    Name Namen,  
    Abteilung Abteilungen,  
    Gehalt NUMBER,  
    PRIMARY KEY (Name),  
    FOREIGN KEY (Abteilung) REFERENCES Sekretariat );
```

```
UPDATE Dozent SET Name = Abteilung; -- schlägt fehl
```

Benutzerdefinierte Domains

Benutzer können auch Domains definieren

– ähnlich Datentypen aber

- Erlaubt Verwendung von Bedingungen / Constraints
- Dafür keine strenge Typ-Prüfung
- **Achtung: Von Oracle so nicht unterstützt!**

Beispiel: Domain für Gehalt mit Mindestlohn

```
CREATE DOMAIN Mindestlohn CHECK (VALUE >= 65000);
```

Data Control Language (DCL)

Mit Hilfe der DCL können Zugriffe auf Teile einer Datenbank autorisiert werden mit folgenden **Privilegien**:

- Lesen von Daten – **SELECT**
- Einfügen von Daten – **INSERT**
- Aktualisieren von Daten – **UPDATE**
- Löschen von Daten – **DELETE**
- Alle Typen von Operationen – **ALL PRIVILEGES**

Syntax für Rechtevergabe:

```
GRANT <Privilegien>  
      ON <Relation oder View>  
      TO <Benutzer / Rollen-Liste>
```


Vergabe von Privilegien

Beispiel: Vergabe von Abfrage-Privilegien

```
GRANT SELECT ON Dozent TO hinz;
```

Beispiel: Vergabe von Einfüge-Privilegien

```
GRANT INSERT ON Dozent TO hinz;
```

```
GRANT INSERT (Abteilung) ON Sekretariat TO hinz, kunz;
```

Beispiel: Vergabe von Aktualisierungs-Privilegien

```
GRANT UPDATE ON Dozent TO hinz;
```

```
GRANT UPDATE (Abteilung) ON Sekretariat TO hinz, kunz;
```

Beispiel: Vergabe von Lösch-Privilegien

```
GRANT DELETE ON Dozent TO hinz, kunz;
```

Rücknahme von Privilegien

Rücknahme funktioniert analog Vergabe

- Syntax:

```
REVOKE <Privilegien>  
      ON <Relation oder View>  
      FROM <Benutzer / Rollen-Liste>
```

Beispiel: Rücknahme von Privilegien

```
REVOKE SELECT ON Dozent FROM hinz;
```

```
REVOKE INSERT ON Dozent FROM hinz;
```

```
REVOKE UPDATE (Abteilung) ON Sekretariat FROM hinz,  
kunz;
```

Rollen

Es ist müßig, für viele Nutzer mit ähnlichen Aufgaben und Berechtigungen alle Privilegien einzeln pro Nutzer zu vergeben

- Konzept der Rolle hilft hier
- Syntax: **CREATE ROLE** <Rollenname>
- Rollen können Benutzern oder anderen Rollen zugewiesen werden

Beispiel: Rolle für Hauspost und Personalverwaltung Informatik

```
CREATE ROLE Hauspost;
```

```
CREATE ROLE HrInfo;
```

```
GRANT SELECT ON Postadressen TO Hauspost;
```

```
GRANT SELECT, INSERT, UPDATE ON InformatikDozenten TO HrInfo;
```

```
GRANT Hauspost TO hinz;
```

```
GRANT Hauspost TO HrInfo;
```

```
GRANT HrInfo TO kunz;
```

Transfer von Privilegien

Benutzer darf Privilegien weiter übertragen, die ihm mit Zusatz ... **WITH GRANT OPTION** übertragen worden sind

Beispiel: Benutzer *hinz* darf Auswertungsprivileg weiter vergeben

```
GRANT SELECT ON Dozent TO hinz WITH GRANT OPTION;
```

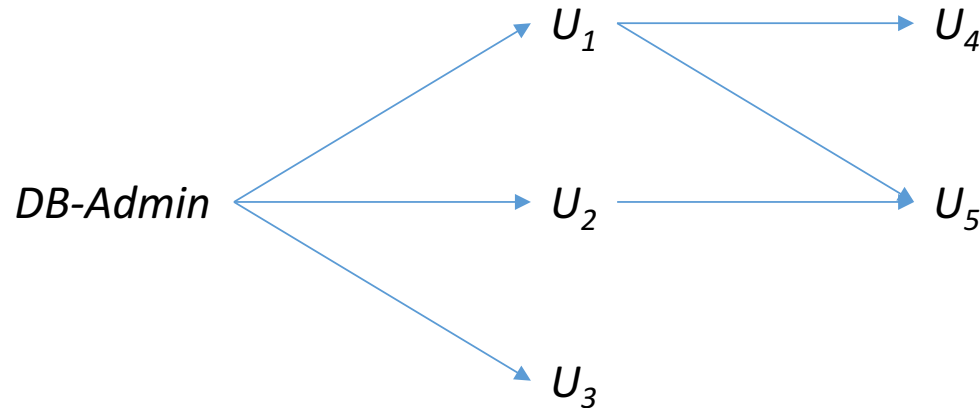
Bei Rücknahme werden auch weiter übertragene Privilegien zurückgenommen!

- Standardverhalten, kann explizit abgeschaltet werden

Transfer von Privilegien

Transfer eines Privilegs kann als **Autorisierungsgraph** dargestellt werden.

Beispiel für ausgewähltes Privileg:



- Rücknahme des Privileg von *U₁* bewirkt Rücknahme bei *U₄*
- Rücknahme des Privileg von *U₁* und *U₂* bewirkt Rücknahme bei *U₅*

Zusammenfassung Kapitel 8

- Mit Hilfe von Sichten (Views) können virtuelle Tabellen für verschiedene Zielgruppen erstellt werden
- Transaktionen definieren atomare Gruppen von SQL-Anweisungen
- Fremdschlüsselbeziehungen dürfen u.U. innerhalb von Transaktionen verletzt werden
- Sequenzen ermöglichen die komfortable Generierung von künstlichen Primärschlüsseln
- Data Control Language ermöglicht feingranulare Rechtevergabe



Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

Kapitel 9

Datenbankmodellierung

Vorlesung Datenbanken und Informationssysteme I
Sommersemester 2017

Prof. Dr. rer. nat. Oliver P. Waldhorst

Datenbankmodellierung - Motivation

Designfehler im Datenmodell haben Auswirkungen auf den gesamten Entwicklungsprozess

- Daher sollten Datenbank-Designer, Entwickler, Anwender gemeinsam am Design arbeiten
- Gemeinsame „Sprache“ vermeidet dabei Missverständnisse
- Daher wurde **Entity-Relationship-Modell** entwickelt
 - „Bauplan“ der Datenbankanwendung

Das Entity-Relationship-Model (ER-Modell)

Beschreibt

- **Entitäten** (engl. Entity) – genauer Entitätstypen (s.u.)
- **Attribute**
- **Beziehungen** (engl. Relationship)

Gebräuchliche Darstellungen für ER-Modelle

- **Chen-Diagramme** – eher konzeptionelle Sicht, früh im Design-Prozess
- **Crow's-Foot-Diagramme** (Krähenfuß-Diagramme) – eher Implementierungssicht, später im Design-Prozess

Werkzeuge für professionellen Datenbankentwurf

Entitätstypen und Beziehungen

Achtung: Entitäten im Sinne des ER-Modells sind Entitätstypen im Sinne von Kapitel 3!

- Entität im ER-Diagramm führt zur Implementierung einer Tabelle, nicht einer Zeile in einer Tabelle!
→ **Wir sprechen von Entitätstypen**
- Konvention für Namen von Entitätstypen
 - Nomen im Singular
 - Werden in GROSSBUCHSTABEN geschrieben (Bsp. KUNDE, ARTIKEL, BESTELLUNG)

Entitätstypen stehen zueinander in **Beziehung**

- **Kardinalitäten**: 1-1, 1-N, M-N
- Konvention für Namen von Beziehungen
 - Verben im Infinitiv in kleinbuchstaben geschrieben (Bsp. kauft)

Darstellung von Entitätstypen

Crow's Foot Diagramm



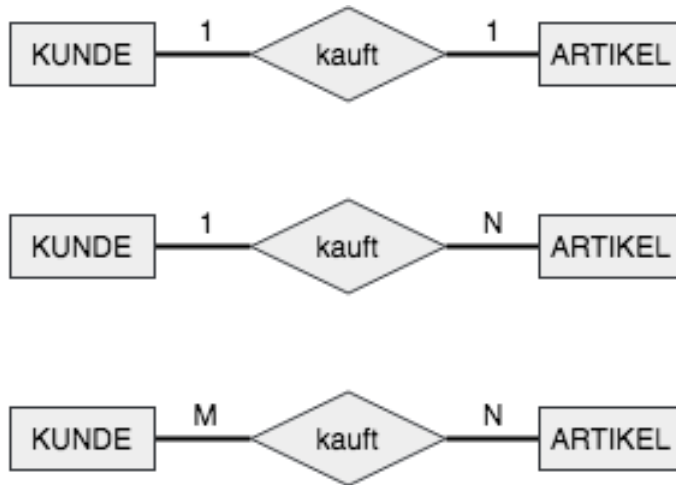
Chen-Diagramm



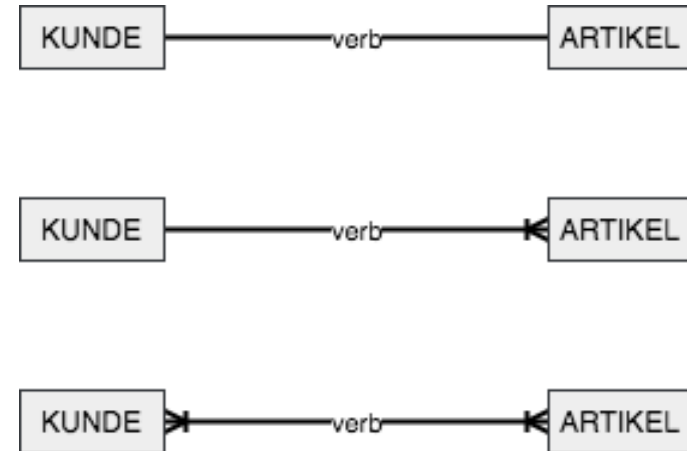
Diagramme erstellt mit draw.io (<https://www.draw.io/>)

Darstellung von Beziehungen

Chen-Diagramm



Crow's Foot-Diagramm





Einschub: Nutzung von Daten(bank)modellen

Modelle abstrahieren von der wirklichen Welt

- Nicht Darstellung aller verfügbarer Informationen, sondern nur der relevanten Informationen

Je nach Abstraktionsgrad unterscheidet man vier verschiedene Datenmodelle

- Konzeptionelles Modell
- Internes Modell
- Externes Modell
- Physisches Modell

Konzeptionelles Modell

- Höchster Abstraktionsgrad, Übersichtscharakter
- Identifikation und Beschreibung der wichtigsten Entitätstypen der Datenbank und ihrer Beziehungen (1:1, 1:N, M:N)
- Keine (technischen) Details, sehr einfach zu verstehen

Beispiel:



Übung: Konzeptionelles Modell

Erstellen Sie ein konzeptionelles Modell für eine Studienverwaltung mit den Entitätstypen DOZENT, STUDENT und KURS und den folgenden Beziehungen:

- Ein Student belegt mehrere Kurse
- Jeder Kurs kann von mehreren Studenten belegt werden
- Ein Dozent unterrichtet mehrere Kurse
- Jeder Kurs wird von genau einem Dozenten unterrichtet

Internes Modell

Berücksichtigt Charakteristiken des Datenbankmanagementsystem

- Grundlegendes Datenmodell des DBMS, Eigenschaften, Beschränkungen, ...
- Z.B. Tabellen mit Primär- und Fremdschlüsseln für relationales DBMS; Pfade für Hierarchisches DBMS

Ein internes Modell lässt sich ...

- ... zwischen DBMS mit gleichem Datenmodell transferieren (z.B. MySQL nach Oracle)
- ... nicht zwischen DBMS mit unterschiedlichem Datenmodell transferieren (z.B. von hierarchischem auf relationales DBMS)

Internes Modell für relationale DBMS

Wichtigste Anpassung für Relationales DBMS: Umwandlung von M:N-Beziehungen in 1:N Beziehungen

- M:N-Beziehungen lassen sich in RDBMS nicht darstellen
- Einführung von **Brückenentitäten** ermöglicht Abbildung

Beispiel:



Übung: Internes Modell

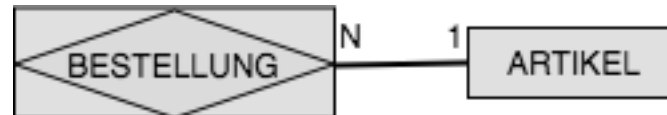
Wandeln Sie das konzeptionelle Modell für die Studienverwaltung in ein internes Modell für ein RDBMS um!

Externes Modell

Teilt das interne Modell in kleinere Einheiten auf

- Ziel: Anwender mit Teilaufgabe muss nicht alle Tabellen und Beziehungen in der Datenbank kennen

Beispiel: Prozessschritt Verpackung einer Bestellung benötigt nur



Externes Modell

Externes Modell ermöglicht Prüfung des konzeptionellen Modells

- Kann ein Prozess nicht dargestellt werden, muss konzeptionelles Modell überarbeitet werden
- Bestimmung der Entitätstypen, auf die ein Prozess Zugriff benötigt

! Übung externes Modell: Welche Prozesse gibt es in der Studienverwaltung?

Physisches Modell

- Geringste Abstraktion
- Legt fest wie Daten auf Datenträgern gespeichert werden inkl. physische Geräte, Zugriffswege, ...
 - Entwicklung benötigt Kenntnisse von Hard- und Software
- Wichtig für Netzwerk- oder hierarchische Datenbanken, da Performanz maßgeblich von Datenablage abhängt
- Für RDBMS ist physisches Modell nur bedingt wichtig, da Details vom DBMS versteckt werden

Attribute im ER-Modell

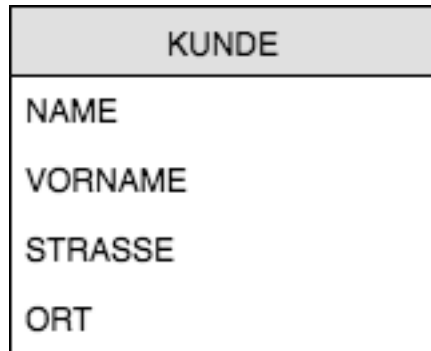
Attribute beschreiben Eigenschaften von Entitätstypen

- Ebenfalls in GROSSBUCHSTABEN geschrieben
- Entsprechen den Attributen / Spalten im Sinne von Kapitel 3

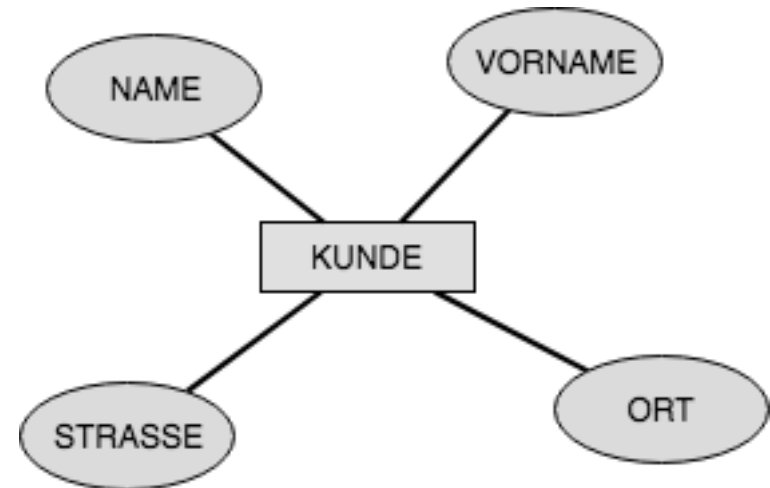
! Übung: Was sind Attribute für die Entitätstypen der Studienverwaltung?

Darstellung von Attributen

Crow's Foot Diagramm



Chen-Diagramm



Einfache und zusammengesetzte Attribute

- Attribute können u.U. aus mehreren Informationen **zusammengesetzt** sein
 - Beispiel: STRASSE könnte Straßennamen und Hausnummer enthalten, die sich auch mit zwei Attributen STRASSENAMEN und HAUSNUMMER abbilden lassen würden
- Nicht weiter unterteilbare Attribute heißen **einfach** oder **atomar** (vgl. Kapitel 4)
- Abhängig vom Anwendungsfall ist es sinnvoll, zusammengesetzte Attribute in atomare aufzuteilen

Mehrwertige Attribute

Manche Attribute können für eine Entität zu gegebenen Zeitpunkt mehrere Wert haben, z.B.

- Attribut TELEFONNUMMER für Entitätstyp KUNDE

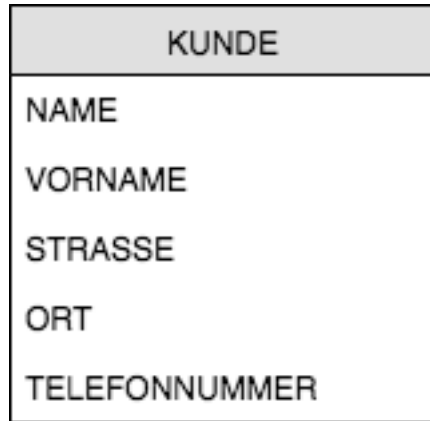
Solche Attribute werden als **mehrwertige Attribute** (engl. multi-value attribute) bezeichnet

Attribute, die nicht mehrwertig sind, müssen nicht zwingend atomar sein, z.B. Kennzeichen

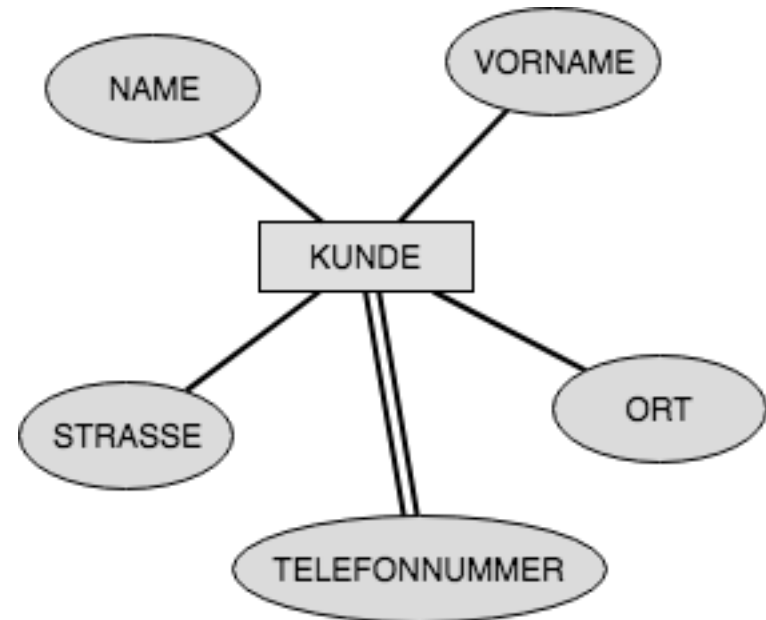
 Übung: Nennen Sie mehrwertige Attribute für die Studiendatenbank!

Darstellung von mehrwertigen Attributen

Crow's Foot Diagramm



Chen-Diagramm



Auflösung von mehrwertigen Attributen

Mehrwertige Attribute können in relationalen Datenbanken nicht direkt abgebildet werden

- Müssen bei Erstellung des internen Modells aufgelöst werden!



Übung: Sehen Sie Lösungsmöglichkeiten?

- *Auflösen in feste Zahl von Einzelattributen*
- *Mehrere Zeilen mit unterschiedlicher Telefonnummer, sonst identisch*
- *Separate Tabelle*

Welche Lösung gewählt wird, hängt vom Anwendungsfall ab!

Abgeleitete Attribute

Manche Attribute können aus anderen abgeleitet werden

- Beispiel: Für Entitätstyp PERSON kann Attribut ALTER kann aus Attribut GEBURTSDATUM berechnet werden

Abgeleitetes Attribut und Attribut, von dem abgeleitet wird, können nicht immer vertauscht werden

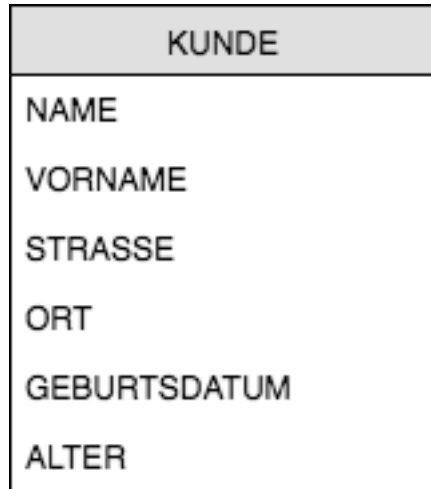
- ALTER (in Jahren) ermöglicht keine Berechnung des genauen GEBURTSDATUM

Abgeleitete Attribute werden in der Regel nicht gespeichert

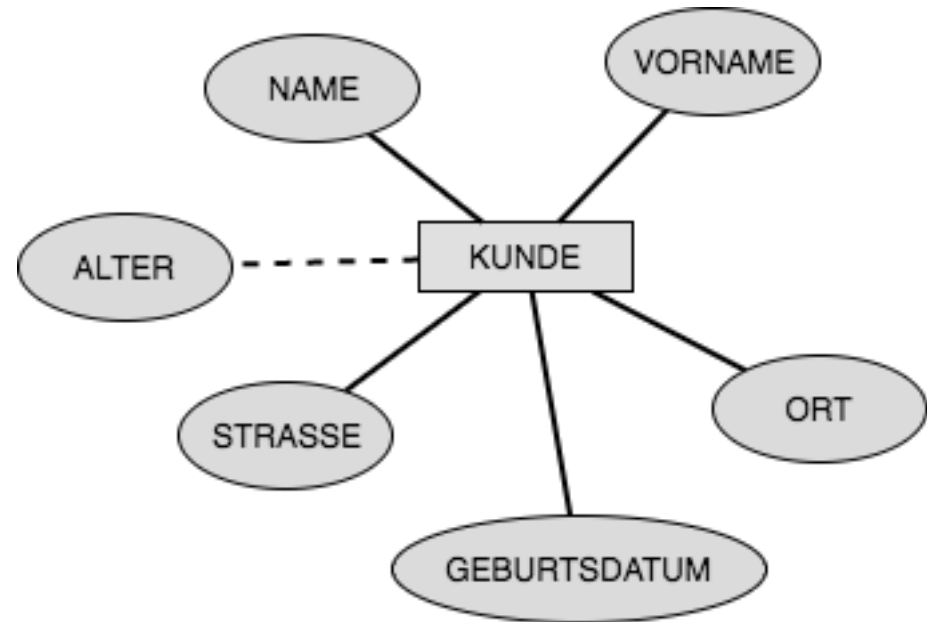
- Ggf. Berechnung über View

Darstellung von abgeleiteten Attributen

Crow's Foot Diagramm



Chen-Diagramm



Domänen

Domäne = Menge der Werte, die ein Attribut annehmen kann

- Nicht zu verwechseln mit dem Datentyp!

Im konzeptionellen Modell haben Attribute mit gleicher Domäne gleiche Namen

- ... auf dieser Ebene besitzen Attribute noch keinen Datentyp

 Übung: Domänen vs. Datentyp am Beispiel Schulnoten

- Welche Domäne? Welcher Datentyp?

Primärschlüssel

Primärschlüssel identifizieren Tupel in Relationen eindeutig

- Für jedes Tupel definiert
- Keine doppelten Werte

Beispiel:

PERSONALNR	NACHNAME	VORNAME	FUNKTION	ABTEILUNG
1	Geldberg	Günter	Geschäftsführer	1
2	Buchhalter	Benno	Abteilungsleiter	4
3	Fehler	Fritz	Programmierer	3
...

Struktur kann auch textuell dargestellt werden:

ANGESTELLTER(PERSONALNR, NACHNAME, VORNAME, FUNKTION, ABTEILUNG)

Natürliche vs. künstliche Primärschlüssel

Natürliche Primärschlüssel

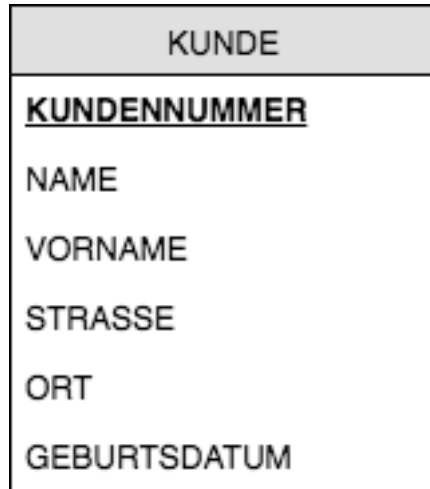
- Werden über konzeptionelles Modell eingeführt
- Geeignete Attribute erfüllen bereits alle Anforderungen an Primärschlüssel (eindeutig, für jedes Tupel definiert)
 - Beispiele: Matrikelnummer, Personalausweisnummer, KFZ-Kennzeichen

Künstliche Primärschlüssel

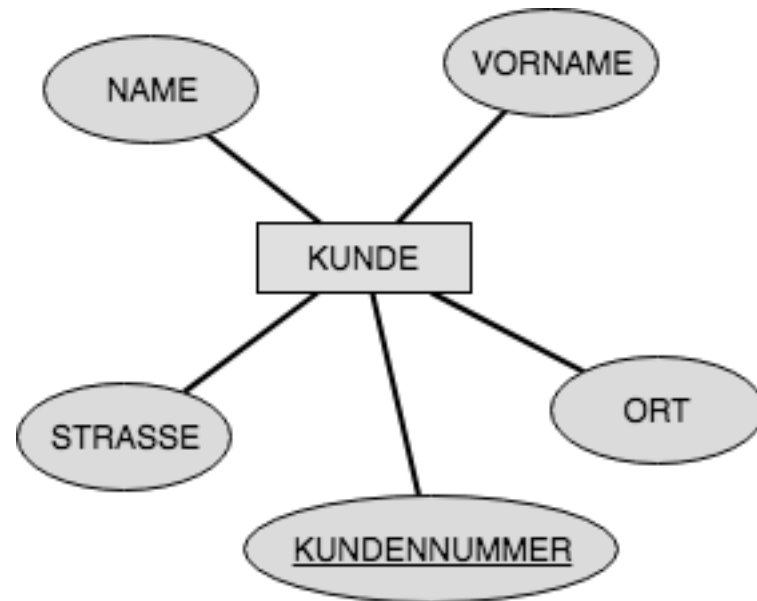
- Werden während des Designs in die Datenbankstruktur eingebracht
- Vorteil: Komplette Kontrolle, vor allem bzgl. der Eigenschaften von Primärschlüsseln
 - Beispiele: Kundennummer, Artikelnummer

Darstellung von Primärschlüsseln

Crow's Foot Diagramm



Chen-Diagramm



Zusammengesetzte Primärschlüssel

Teilweise ist es notwendig, Primärschlüssel aus mehr als einem Attribut zu verwenden

- Damit ist jede Kombination der Schlüsselattribute eindeutig
- Primärschlüssel aus mehreren Attributen heißt zusammengesetzt

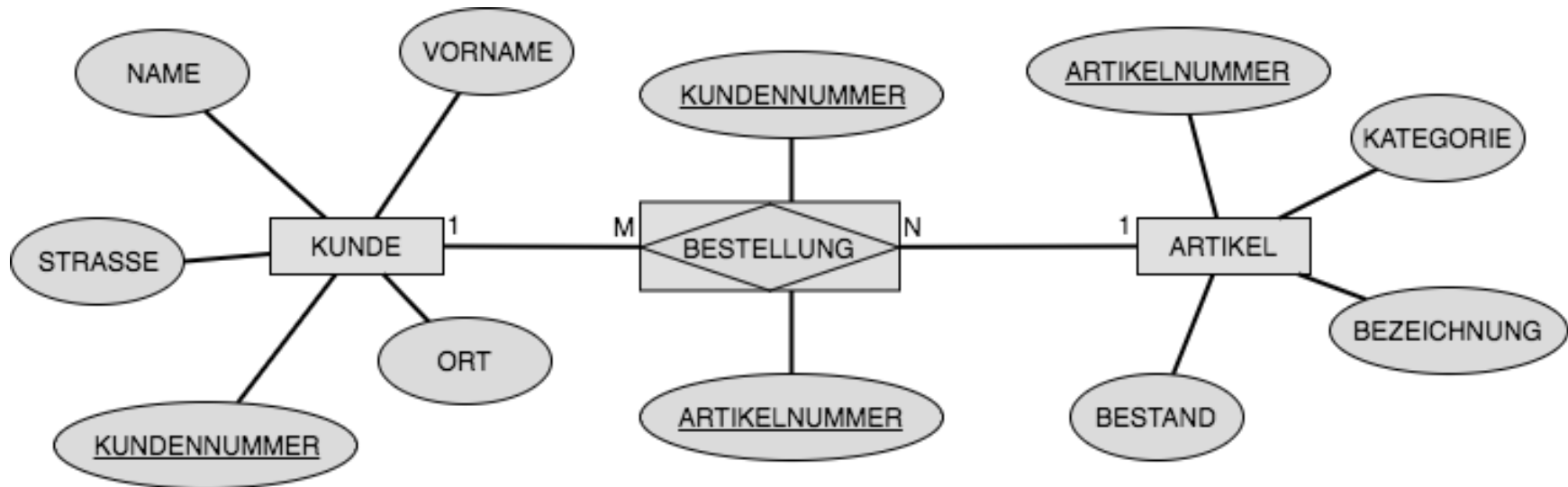
Beispiel:

KUNDE(KUNDENNR, NAME, ...)

ARTIKEL(ARTIKELNR, BEZEICHNUNG, ...)

BESTELLUNG(KUNDENNR, ARTIKELNR)

Zusammengesetzte Primärschlüssel – Chen-Diagramm



Zusammengesetzte Primärschlüssel – Crow's Foot Diagramm



Übung: Primärschlüssel

Benennen und kennzeichnen Sie die Primärschlüssel in der Studienverwaltung!

Mehr zu Beziehungen

Eine Beziehung ist Verbindung zwischen Entitätstypen

- Durch aktives oder passives Verb benannt
- Funktioniert in beide Richtungen

Wichtig: Beide Richtungen betrachten, um Beziehung vollständig zu erfassen!

Beispiel:

- Ein KUNDE bestellt mehrere ARTIKEL
- Ein ARTIKEL wird von mehreren KUNDEN bestellt



Minimale und maximale Kardinalität

- Art der Beziehung (1:1, 1:N, M:N) wird als **Kardinalität** der Beziehung bezeichnet
- **Minimale** und **maximale Kardinalität** bezeichnet die Anzahl der Entitäten in den bezogenen Entitätstypen, die mit einer gegebenen Entität in Beziehung stehen

Beispiel minimale und maximale Kardinalität

M:N-Beziehung KUNDE – ARTIKEL aufgelöst in 1:N Beziehungen mit Brückenentität BESTELLUNG stellt sich aus Datenbanksicht z.B. so dar:

KUNDENNR	NAME	VORNAME	STRASSE	ORT
1	Meier	Hans	Kittelgasse 3	Karlsruhe
2	Müller	Gerda	Moltkestraße 2	Mannheim
3	Kunz	Heinz	Fasanengarten 1	Karlsruhe

KUNDENNR	ARTIKELNR
1	2
2	3
1	3

ARTIKELNR	BEZEICHNUNG	KATEGORIE	BESTAND
1	Fernseher	Elektronik	3
2	Schrank	Möbel	6
3	Tisch	Möbel	2

Minimale und maximale Kardinalität

Es kann sinnvoll sein, Schranken für die Konnektivität anzugeben:

- Minimale Kardinalität: Zu wie vielen Entitäten im bezogenen Entitätstyp besteht mindestens eine Beziehung
- Maximale Kardinalität: Zu wie vielen Entitäten im bezogenen Entitätstyp besteht höchstens eine Beziehung

Beispiel:

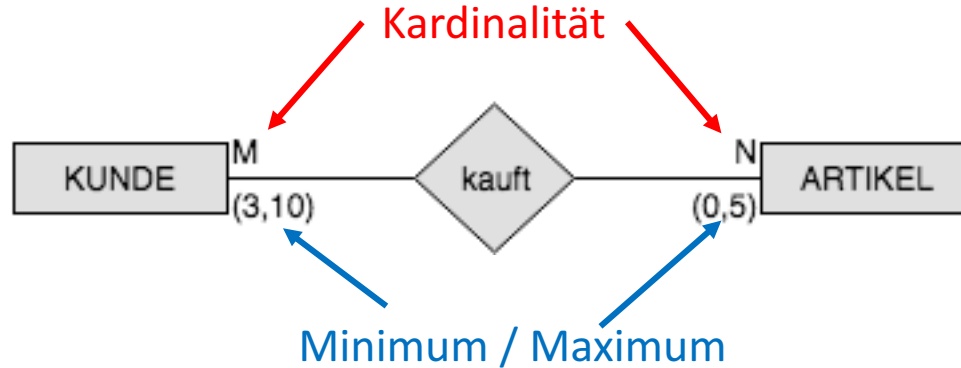
- Von einem Kunden müssen mindestens drei und maximal zehn Artikel bestellt werden
- Jeder Artikel kann maximal fünf mal bestellt werden

Minimale und maximale Kardinalitäten werden in konzeptionellem Modell entwickelt

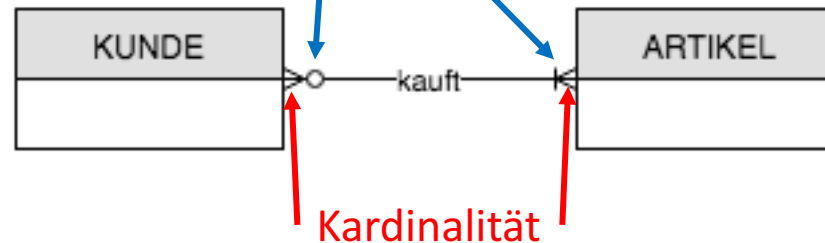
- Teil der Business-Logik der Datenbankanwendung; wichtig bei der Anwendungsentwicklung, da auf Tabellen-Ebene keine Mechanismen zur Definition existieren

Darstellung minimaler / maximaler Kardinalitäten

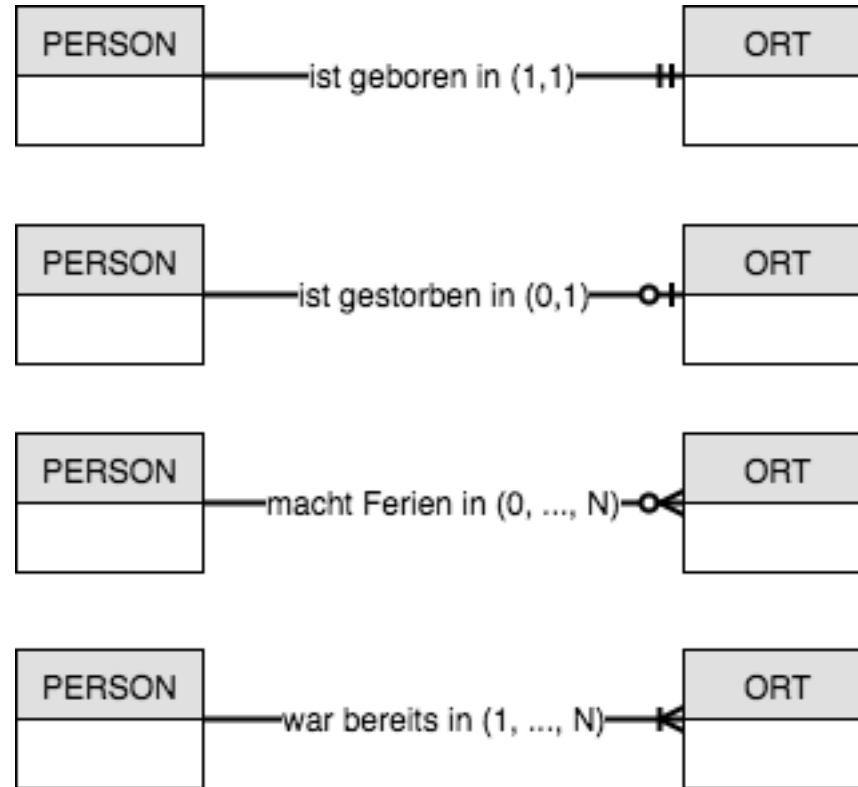
Chen-Diagramm(Min-Max-Notation):



Crow's Foot-Diagramm:



Darstellung von minimalen Kardinalitäten in Crow's Foot-Diagrammen



Beispiel aus Wikipedia

Optionale Beziehungen

Wenn Auftreten eines bestimmten Entitätstyps A nicht heißt, dass eine anderer Entitätstyp B auftreten muss, dann ist die Beziehung zwischen A und B **optional**

Beispiel: KUNDE kann ohne BESTELLUNG existieren

Datenbanksicht: Fremdschlüssel des beziehenden Entitätstyps nimmt nicht alle Werte des Primärschlüssel desbezogenen Entitätstyps an (im Beispiel kommen nicht alle KUNDENNUMMERn in BESTELLUNG vor)

Bei optionaler Beziehung ist minimale Kardinalität 0!

Nicht-optionale Beziehungen

Wenn Auftreten eines bestimmten Entitätstyps A bedeutet, dass eine anderer Entitätstyp B auftreten muss, dann ist die Beziehung zwischen A und B **nicht optional**

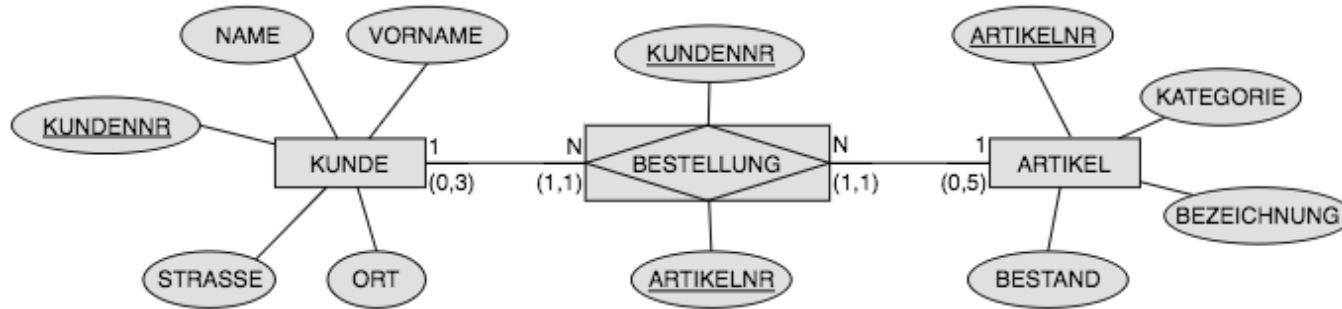
Beispiel: Jedem Kunden muss eine Telefonnummer zugeordnet sein
KUNDE(KUNDENNR, ...), TELEFON(KUNDENNR, RUFNR, TYP)

Datenbanksicht: Jeder Wert des Primärschlüssels des bezogenen Entitätstyps wird von einer beziehenden Entität einmal referenziert (im Beispiel wird jeder Wert von KUNDENNUMMER in TELEFON einmal referenziert)

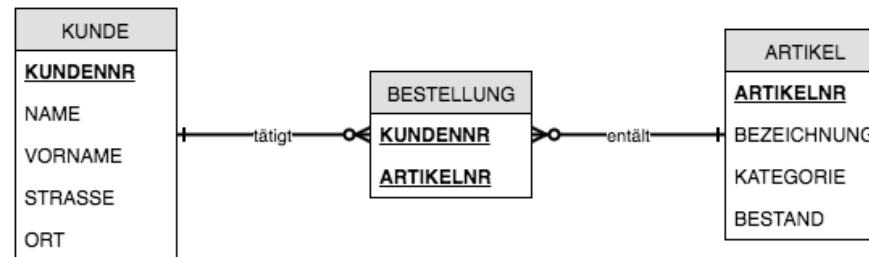
Bei nicht-optionalen Beziehungen ist minimale Kardinalität 1!

Darstellung Optionaler Beziehungen

Chen-Diagramm:



Crow's Foot-Diagramm:



Stärke von Beziehungen

Existentielle Abhängigkeit

- Ein Entitätstyp A ist von einem anderen, in Beziehung stehenden Entitätstyp B **existentiell abhängig**, wenn eine Entität von A nicht ohne eine Entität von B existieren kann

Beispiel: BESTELLUNG existiert nicht ohne KUNDE, daher ist BESTELLUNG von KUNDE existentiell abhängig

Existentielle Unabhängigkeit

- Ein Entitätstyp A ist von einem anderen, in Beziehung stehenden Entitätstyp B **existentiell unabhängig**, wenn eine Entität von A auch ohne eine Entität von B existieren kann

Beispiel: KUNDE existiert auch ohne BESTELLUNG, daher ist KUNDE von BESTELLUNG existentiell unabhängig

Schwache Beziehungen

Eine Beziehung zwischen voneinander existentiell unabhängigen Entitätstypen (die also gegenseitig ohne den jeweils anderen existieren können) heißt **schwach** oder **nicht-identifizierend**

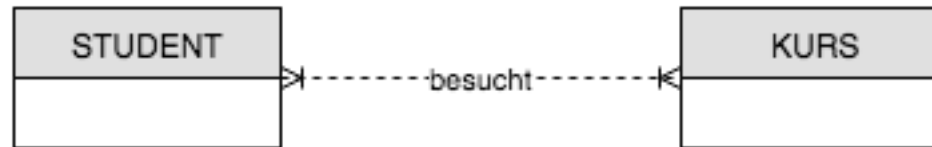
Datenbank-Sicht: Für Entitätstypen A und B mit schwacher Beziehung kann Primärschlüssel von A als Fremdschlüssel von B verwendet werden, ist aber nicht Teil des Primärschlüssels von B (und umgekehrt)

 Übung: Finden Sie eine schwache Beziehung in der Studienverwaltung

Darstellung schwache Beziehung

In Chen-Diagrammen gibt es keine besondere Darstellung für schwache Beziehungen!

Crow's Foot-Diagramm:





Starke Beziehungen / schwache Entitätstypen

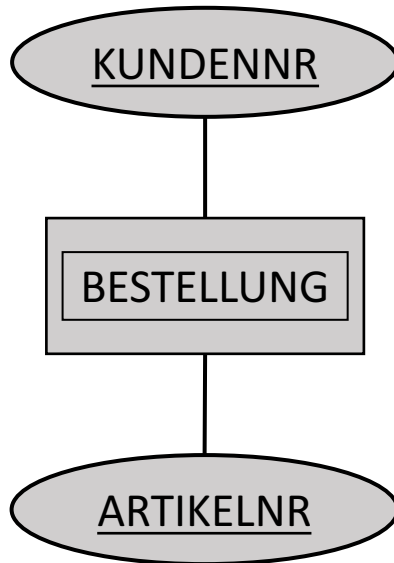
Eine Beziehung zwischen zwei Entitätstypen A und B heißt **stark** oder **identifizierend**, wenn Entitätstyp B von Entitätstyp A existentiell abhängig ist; B heißt dann **schwacher Entitätstyp**

Datenbank-Sicht: Für Entitätstypen A und B mit starker Beziehung ist der Primärschlüssel von A Teil des Primärschlüssels von B

Beispiel: Beziehung zwischen KUNDE und BESTELLUNG ist stark, BESTELLUNG ist schwacher Entitätstyp

Darstellung von schwachen Entitätstypen

Chen-Diagramm:



Crow's Foot-Diagramm:





Implementierung von schwachen Entitätstypen

Bei Implementierung auf Datenbankebene von schwachen Entitätstypen ist die Reihenfolge des Anlegens wichtig

Beispiel: KUNDE muss angelegt sein, bevor BESTELLUNG definiert werden kann

Datenbankdesigner legt fest, ob eine Entitätstyp stark oder schwach ist (durch entsprechende Definition ihres Primärschlüssels)

- Ändert nichts an den eigentlich gespeicherten Daten

Übung: Schwache Entitätstypen

Finden Sie schwache Entitätstypen in der Studienverwaltung

Wichtig: Unterscheidung (nicht) optional vs. starke und schwache der Beziehung

Man kann nicht folgern, dass optionale Beziehungen automatisch schwache Beziehungen sind

- Bei starker Beziehung von A zu B enthält Primärschlüssel von A den Primärschlüssel von B → Struktur der Schlüssel
- Bei optionaler Beziehung von A zu B müssen nicht alle Werte des Primärschlüssels von A als Fremdschlüssel von B verwendet werden → Werte der Schlüssel

Ermessensspielraum des Datenbankdesigners, ob statt schwacher starke Beziehung definiert wird

Grad von Beziehungen

Grad gibt an, wie viele Entitätstypen an Beziehung beteiligt sind

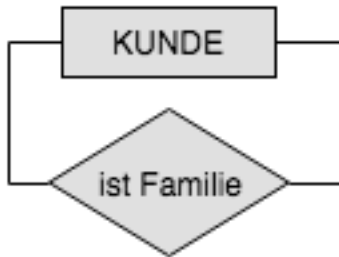
- **Unär:** ein Entitätstyp
- **Binär:** zwei Entitätstypen
- **Ternär:** drei Entitätstypen
- (es gibt höhere, aber selten verwendet)

Unäre Beziehungen

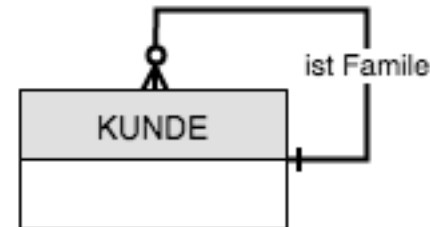
Beziehung zwischen zwei Entitäten des selben Entitätstyps

Beispiel: Familien von Kunden

Chen-Diagramm:



Crow's Foot-Diagramm:



Binäre Beziehungen

Beziehungen zwischen Entitäten unterschiedlicher Entitätstypen

- Häufigster Beziehungstyp
- Bei Implementierung in relationalem Datenbanksystem werden Beziehungen höheren Grades in binäre Beziehungen zerlegt

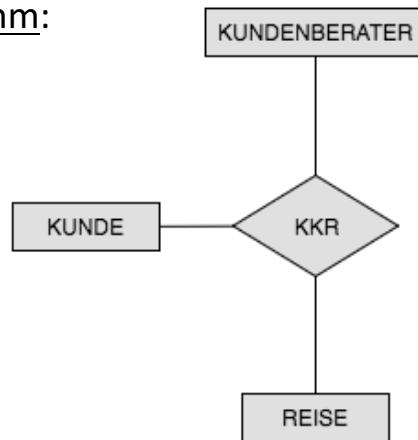
Beispiele für binäre Beziehungen: Hatten wir bereits viele...

Ternäre Beziehung

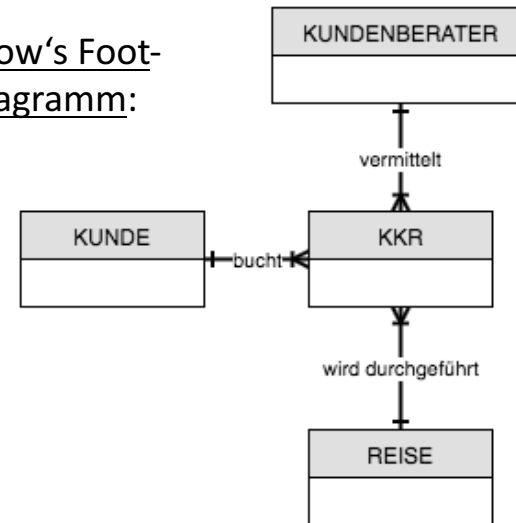
Können beim Datenbankdesign hilfreich sein, auch wenn sie letztendlich in binäre Beziehungen umgewandelt werden

Beispiel: Kundenberater vermittelt Reise an Kunden

Chen-Diagramm:



Crow's Foot-Diagramm:



Im Crow's Foot-Diagramm werden zugrundeliegende binäre Beziehungen direkt sichtbar!

Rekursive Beziehung

Anwendung der unären Beziehung

- 1:1-, 1:N-, M:N-Beziehungen möglich
- Ermöglicht Abbildung von hierarchischen Strukturen auf linearen Strukturen eines Entitätstyps
- Achtung: Fremdschlüssel darf nie den Wert des Primärschlüssels des eigenen Tupels enthalten → Endlosschleife
- 1:1 und 1:N kann auf einziger Tabellen implementiert werden, M:N braucht mehrere Tabellen

Beispiel: Person...

- Ist verheiratet mit
- Ist vorgesetzt
- Arbeitet zusammen

Zusammengesetzte Entitätstypen

Zusammengesetzte Entitätstypen (auch Brückenentitäten oder Zwischenentitäten)

- Verbindet zwei Entitätstypen
- Primärschlüssel setzt sich aus den Primärschlüsseln der verbundenen Entitätstypen zusammen
- Existentiell abhängig von verbundenen Entitätstypen
- Kann neben den Primärschlüsseln weitere Attribute enthalten (z.B. kann BESTELLUNG DATUM enthalten)

Darstellung (s.o.):

- Chen-Diagramm: Rechteck und Raute
- Crow's Foot-Diagramm: Keine Kennzeichnung

Supertypen und Subtypen

Es kann hierarchische „Vererbungsstruktur“ für Entitätstypen geben

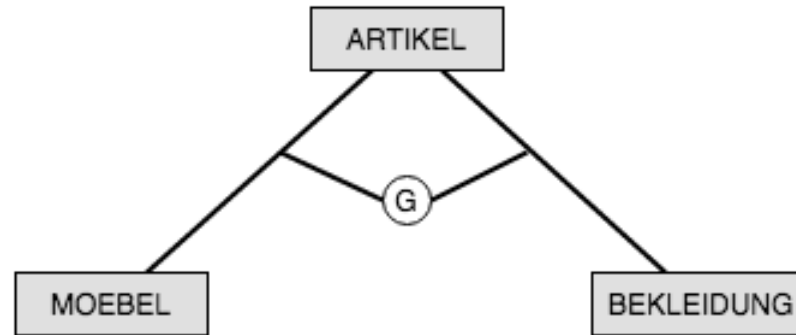
Beispiel: ARTIKEL kann BEKLEIDUNG oder MOEBEL sein
ARTIKEL(ARTIKELNUMMER, PREIS, BESTAND)
BEKLEIDUNG(ARTIKELNUMMER, PREIS, BESTAND, GROESSE)
MOEBEL(ARTIKELNUMMER, PREIS, BESTAND, HOEHE, BREITE, TIEFE)

Im Beispiel ist ARTIKEL ein **Supertyp**, BEKLEIDUNG und MOEBEL sind **Subtypen**

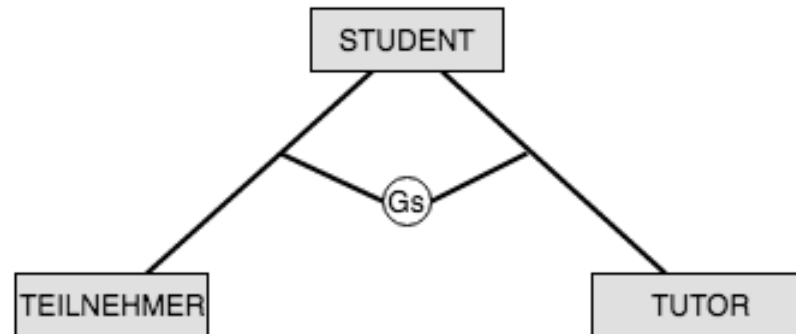
! Übung: Wie würden Sie den Zusammenhang modellieren?

Darstellung von Super-/Subtypen

Sich ausschließende Subtypen:



Sich nicht ausschließende Subtypen:



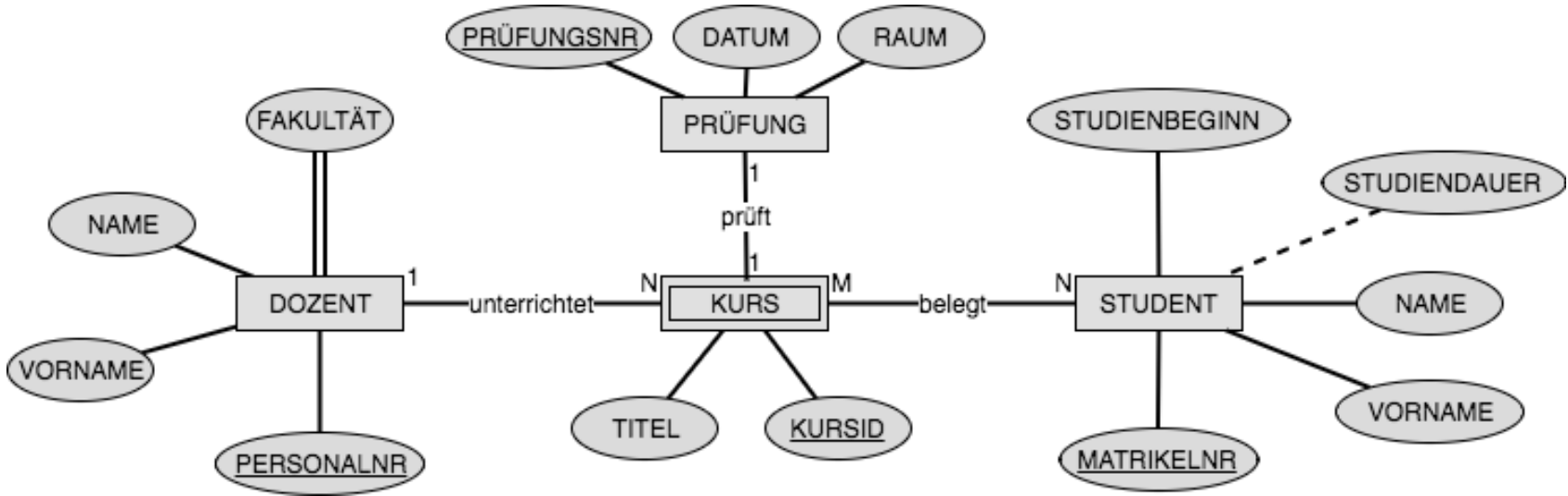
Transformation ER-Modell in Relationales Modell (aka Relationenmodell)

Ausgangspunkt: ER-Modell

- Konzeptionelles Modell mit Schlüsseln für jeden Entitätstyp (liegt bereits ein internes Modell vor, entfallen einige Schritte)
- Nur unäre / binäre Beziehungen

Ziel: Relationenmodell

- Relationsschemata (Tabellen) mit Attributen, Primär- und Fremdschlüsseln, ggf. Beschränkungen / Constraints
- Umwandlung in sieben Einzelschritten
(angelehnt an J. C. Freytag, Vorlesung Grundlagen von Datenbanksystemen, HU Berlin)





Schritt 1: Bestimmung der Relationsschemata

Jeder Entitätstyp wird in ein Relationsschema (Tabelle) umgewandelt

- Attribute des Entitätstyps werden Attribute des Relationsschemas
 - Ausnahme: Abgeleitete Attribute entfallen (ggf. später über Views dargestellt)
- Primärschlüssel des Entitätstyps werden Primärschlüssel des Relationsschemas

Beispiel:

Dozent(PersonalNr, Name, Vorname, Fakultät)

Student(MatrikelNr, Name, Vorname, Studienbeginn)

Kurs(KursId, Titel)

Prüfung(PrüfungsNr, Datum, Raum)

Schritt 2: Ergänzung für schwache Entitäten

Primärschlüssel des starken Entitätstyps wird als Fremdschlüsselattribut im schwachen Entitätstyp ergänzt

- Ergänzte Attribute werden Teil des Primärschlüssels

Beispiel:

Dozent(PersonalNr, Name, Vorname, Fakultät)

Kurs(KursId, PersonalNr, Titel)

FOREIGN KEY (PersonalNr)



Schritt 3: Umwandlung von 1-1-Beziehungen

Ein an der Beziehung beteiligtes Relationsschema wird um den Primärschlüssel des anderen als Fremdschlüssel mit Constraint UNIQUE erweitert

- Falls möglich, auf der Seite, an der Beziehung nicht optional ist; dann zusätzlich mit Constraint NOT NULL
- Keine zusätzlichen Relationsschemata notwendig!

Beispiel:

Kurs(KursId, PersonalNr, Titel)

*Prüfung(PrüfungsNr, Datum, Raum, **KursId**)*

FOREIGN KEY (KursId) REFERENCES Kurs

NOT NULL KursId

UNIQUE KursId

Schritt 4: Umwandeln von 1-N-Beziehungen

Primärschlüssel des Relationsschemas auf Seite der 1-Kardinalität wird als Fremdschlüsselattribut des Relationsschemas auf Seite der N-Kardinalität ergänzt

- Wenn nicht optional mit Constraint NOT NULL

Beispiel:

Hier bereits durch Schritt 2 erledigt (NOT NULL implizit):

Dozent(PersonalNr, Name, Vorname, Fakultät)

Kurs(KursId, PersonalNr, Titel)

FOREIGN KEY (PersonalNr)

Schritt 5: Umwandeln der M-N-Beziehungen

Anlegen eines neuen Relationsschemas für M-N-Beziehung

- Enthält Primärschlüssel der beiden bezogenen Entitätstypen als Fremdschlüssel; diese bilden gleichzeitig den Primärschlüssel

Beispiel:

Kurs(KursId, PersonalNr, Titel)

Student(MatrikelNr, Name, Vorname, Studienbeginn)

Belegt(KursId, PersonalNr, MatrikelNr)

FOREIGN KEY (KursId, PersonalNr) REFERENCES Kurs

FOREIGN KEY (MatrikelNr) REFERENCES Student

Schritt 6: Mehrwertige Attribute

Jedes mehrwertige Attribute führt zu einem zusätzlichen Relationsschema

- Primärschlüssel der ursprünglichen Relation bildet zusammen mit mehrwertigem Attribut den Primärschlüssel des neuen Relationsschemas

Beispiel:

Dozent(PersonalNr, Name, Vorname, **Fakultät**)

Fakultät(PersonalNr, Fakultät)

FOREIGN KEY (PersonalNr) REFERENCES Dozent

Zwischenstand: Komplettes Relationenmodell

Dozent(PersonalNr, Name, Vorname)

Kurs(KursId, PersonalNr, Titel)

FOREIGN KEY (PersonalNr)

Student(MatrikelNr, Name, Vorname, Studienbeginn)

Prüfung(PrüfungsNr, Datum, Raum, KursId)

FOREIGN KEY (KursId) REFERENCES Kurs

NOT NULL KursId

UNIQUE KursId

Belegt(KursId, PersonalNr, MatrikelNr)

FOREIGN KEY (KursId, PersonalNr) REFERENCES Kurs

FOREIGN KEY (MatrikelNr) REFERENCES Student

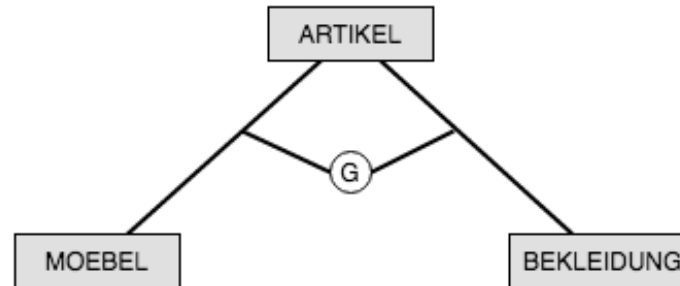
Fakultät(PersonalNr, Fakultät)

FOREIGN KEY (PersonalNr) REFERENCES Dozent

Schritt 7: Umwandeln von Super-/Subtypen

Übernehmen des Primärschlüssels des Supertypen für alle Subtypen

Beispiel:



Artikel(Artikelnummer, Preis, Bestand)

Moebel(Artikelnummer, Höhe, Breite, Tiefe)

Bekleidung(Artikelnummer, Größe)

Zusammenfassung Kapitel 9

ER-Modelle sind Sprache zur Modellierung von Datenbanken

- Können für verschiedenen Entwurfsphasen eingesetzt werden, insbesondere für konzeptionelle und interne Modelle
- ER-Diagramme visualisieren ER-Modelle; hier betrachtet: Crow's Foot- und Chen-Diagramme

Interne Modelle für relationale DBMS haben Einschränkungen

- Keine M-N-Beziehungen
- Keine mehrwertigen Attribute

Starke und schwache Entitäten drücken sich über Primär- und Fremdschlüssel aus

Umwandlung von Konzeptionellen Modell in Relationenmodell in sieben Schritten möglich