

VL Datenbanksysteme

Wintersemester 2023/2024

Prof. Dr.-Ing. Kai-Uwe Sattler¹ Prof. Dr. Gunter Saake²

¹ TU Ilmenau
FG Datenbanken & Informationssysteme

² Universität Magdeburg
Institut für Technische & Betriebliche Informationssysteme, Okt. 2023



Zugrundeliegendes Lehrbuch

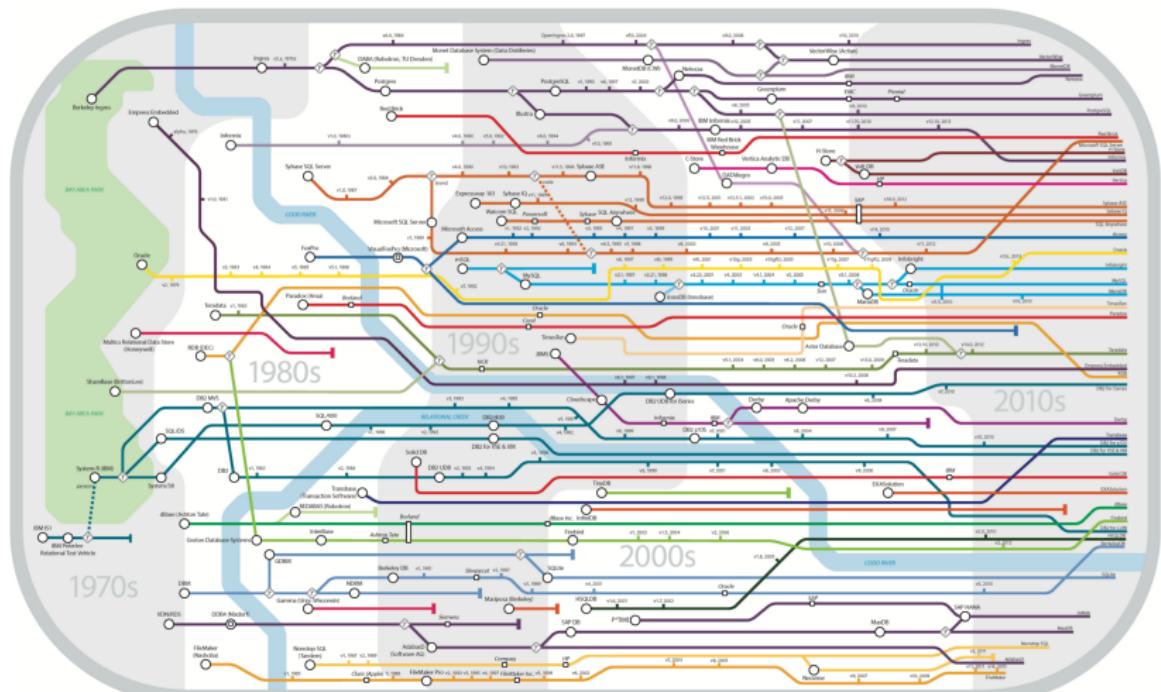


G. Saake; K. Sattler; A. Heuer:
Datenbanken — Konzepte und Sprachen

6. Auflage, mitp-Verlag, 2018



Stammbaum Relationaler Datenbanken



Organisatorisches

Dozent



Gunter Saake

Arbeitsgruppenleiter DBSE

saae@ovgu.de

Es werden folgende Inhalte vorgestellt, eingeübt und diskutiert:

Inhalte dieser Veranstaltung

1. Was sind Datenbanken – Grundlegende Konzepte
2. Relationale Datenbanken – Daten als Tabellen
3. Datenbankentwurf im ER-Modell
4. Relationaler DB-Entwurf
5. Relationale Entwurfstheorie
6. Die Datenbanksprache SQL
7. Grundlagen von Anfragen: Algebra & Kalkül
8. Transaktionen, Integrität und Trigger
9. Sichten und Zugriffskontrolle
10. NoSQL-Datenbanken
11. Anwendungsprogrammierung mit Datenbanken

Übungen, Klausur & Abschluss

Für den erfolgreichen Abschluss der Veranstaltung ist nötig

- Die erfolgreiche Teilnahme an der **Klausur**
 - Termine durch Prüfungsamt bestimmt und bekanntgegeben
 - Sie werden durch uns auf geeignete Weise darauf hingewiesen
 - Erfolgreicher Abschluss der **Übungen** ist **Klausurvoraussetzung**
 - Sie haben **min. 66 %** der Aufgaben in der Übung votiert (= zu Hause bearbeitet)
 - Sie haben **min. 4 Vortragspunkte** erhalten (= Übungsaufgaben vorgestellt)
 - **oder: min. 60 % + 4 VP + Ihr Team hat die praktische Aufgabe** bestanden (dazu später mehr)

Übungen und Übungsbetrieb



Wöchentlich finden Übungsgruppen statt (ab der zweiten Vorlesungswoche).

- Dienen zur Kontrolle und Präsentation von Lösungen des aktuellen Zettels und als Plattform für Ihre Fragen zum neuen Zettel
- Tutorien werden durch Übungsleiter bzw. Tutoren begleitet

Tutoren



Elias Kuiter



Josefine Becker



Lukas Eichel



Nele Hüsemann



Arne Schaumburg



Hubert Schmidt



Victor Obionwu

1. **Montag** (09.10.2023): Ausgabe Zettel 1 zur selbstständigen Bearbeitung
2. Ab heute (09.10.2023) **schreiben** Sie sich für einen Übungstermin ein.
3. die Übungen finden ab der **zweiten** Vorlesungswoche (ab 16.10.2023) statt
4. **nächsten Montag** (16.10.2023): Ausgabe Zettel 2
5. Nach der Vorlesungszeit findet die **Klausur** statt (weitere Details durch das Prüfungsamt)

Bitte besuchen Sie regelmäßig die Vorlesungsseite im eLearning-System <https://elearning.ovgu.de/course/view.php?id=15040>, um aktuelle Informationen in Erfahrung zu bringen, Änderungshinweise zu erhalten, und für Übungszettel.

- Die Übungsaufgaben enthalten eine Reihe von zu erstellenden SQL Anfragen.
- Zur Selbstkontrolle stellen wir das Tool SQLValidator zur Verfügung
<https://propra14.iti.cs.ovgu.de/sqlValidator/vali/publicRoot/sqlvali/index.php>
- Nutzung der Nutzererfahrung im Forschungsprojekt “DiP-IT”
<https://www.wihoforschung.de/de/dip-it-3041.php>
- Passphrase zur Registrierung wird auf der Moodle-Seite bekanntgegeben.

Weitere Literatur

Weitere Literatur

-  R. Elmasri, S.B. Navathe.
Grundlagen von Datenbanksystemen.
3. Auflage, Pearson Studium, 2009
-  A. Kemper, A. Eickler.
Datenbanksysteme. Eine Einführung.
10. Auflage, De Gruyter, 2015
-  A. Heuer, G. Saake, K. Sattler, H. Grunert, H. Meyer.
Datenbanken Kompaktkurs
mitp-Verlag, Bonn, 2020

Teil I

Was sind Datenbanken?

Was sind Datenbanken?



1. Überblick & Motivation
2. Architekturen
3. Einsatzgebiete
4. Historisches

Lernziele für heute . . .

- Motivation für den Einsatz von Datenbanksystemen
- Kenntnis grundlegender Architekturen



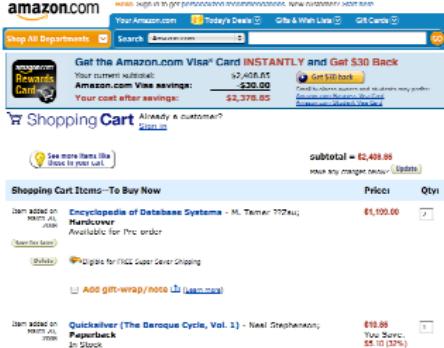
Überblick & Motivation

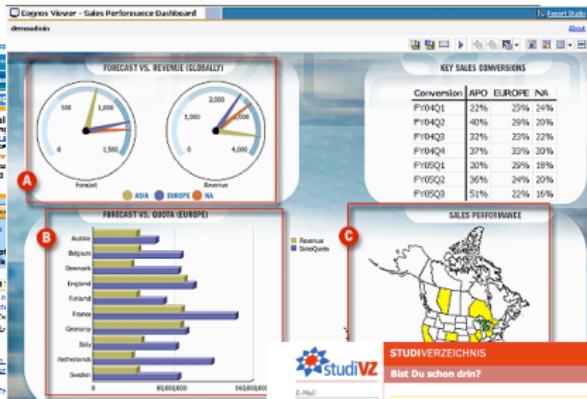
Was sind Datenbanken?

- Daten = logisch gruppierte Informationseinheiten
- Bank = ...
- Die **Sicherheit vor Verlusten** ist eine Hauptmotivation, etwas „auf die Bank zu bringen“.
- Eine Bank bietet **Dienstleistungen für mehrere Kunden** an, um effizient arbeiten zu können.
- Eine Datenbank hat die (langfristige) **Aufbewahrung** von Daten als Aufgabe.

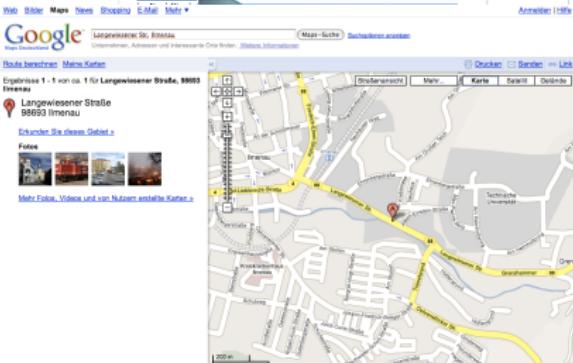


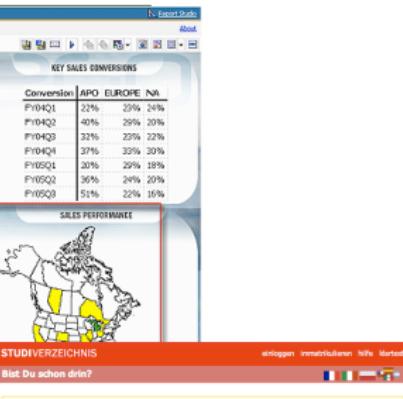
Anwendungsbeispiele











Jetzt kostenlos anmelden!

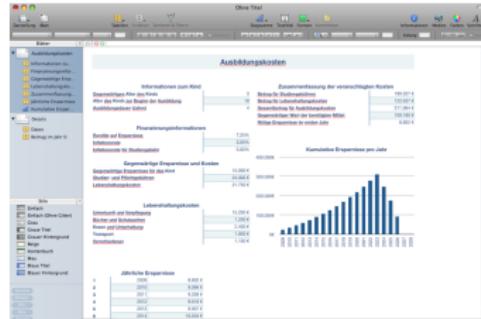
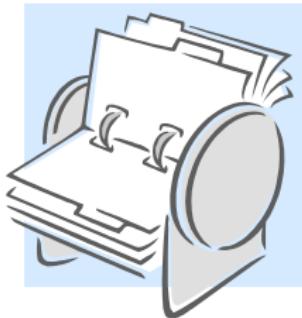
- Finde andere Studenten an Deiner Hochschule
- Finde alte Freunde wieder
- Finde heraus, wer über welche Ecken kennt
- Finde Partner für Sport, Lernen und Freizeit
- Finde heraus, was für Leute in Deinen Lehrveranstaltungen sitzen
- Vereinige Dich jetzt auch mit meinVZ-Nutzen
- Vergiss keine Geburtstage mehr – studiVZ erinnert Dich automatisch

stu
diVZ

Bedeutung der Datenverwaltung

- Daten als das Öl des 21. Jahrhunderts (Toonders @Wired 2014)
- Ash Ashutosh (CEO Actifio):
 - das größte Hotelunternehmen (AirBnB) hat keine Hotels bzw. Zimmer
 - das größte Taxiunternehmen (Uber) hat keine eigenen Taxis!

Wie verwaltet man Datenbanken?



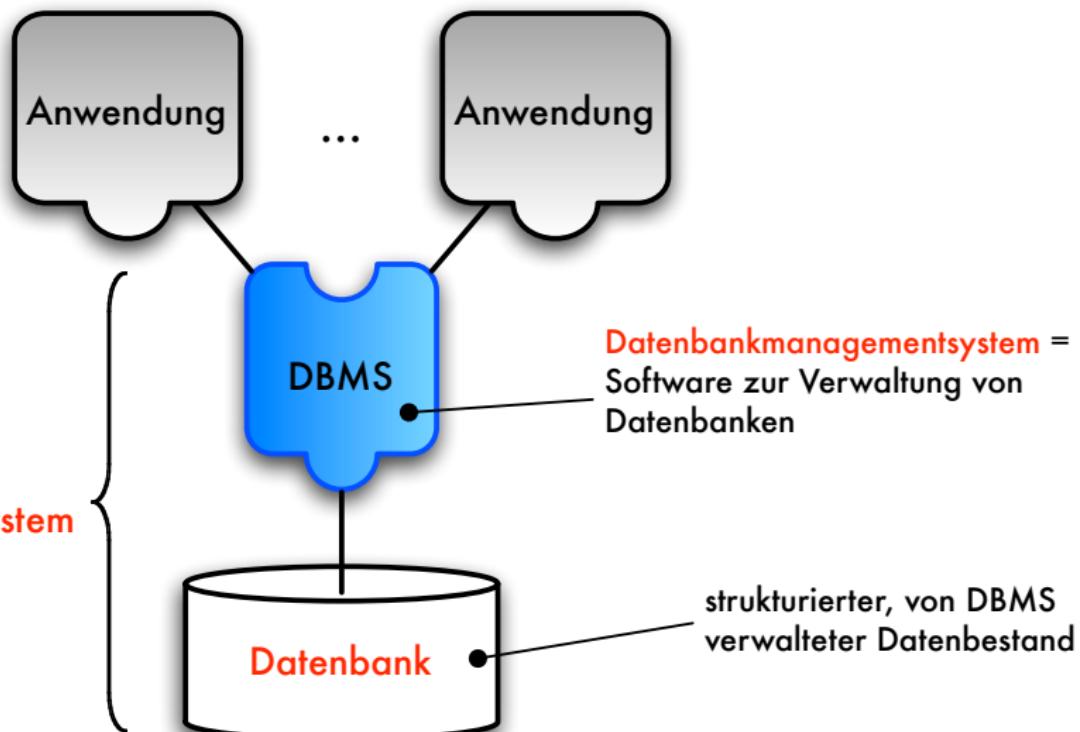
Ohne Datenbanken

- jedes Anwendungssystem verwaltet seine eigenen Daten
- Daten sind mehrfach gespeichert \rightsquigarrow redundant
- Probleme
 - Verschwendungen von Speicherplatz
 - „Vergessen“ von Änderungen
 - keine zentrale, „genormte“ Datenhaltung

Probleme der Datenredundanz

- Andere Softwaresysteme können große Mengen von Daten nicht effizient verarbeiten
- Mehrere Benutzer oder Anwendungen können nicht parallel auf den gleichen Daten arbeiten, ohne sich zu stören
- Anwendungsprogrammierer / Benutzer können Anwendungen nicht programmieren / benutzen, ohne
 - interne Darstellung der Daten
 - Speichermedien oder Rechnerzu kennen (**Datenunabhängigkeit** nicht gewährleistet)
- Datenschutz und Datensicherheit sind nicht gewährleistet

Idee: Datenintegration durch Datenbanksysteme

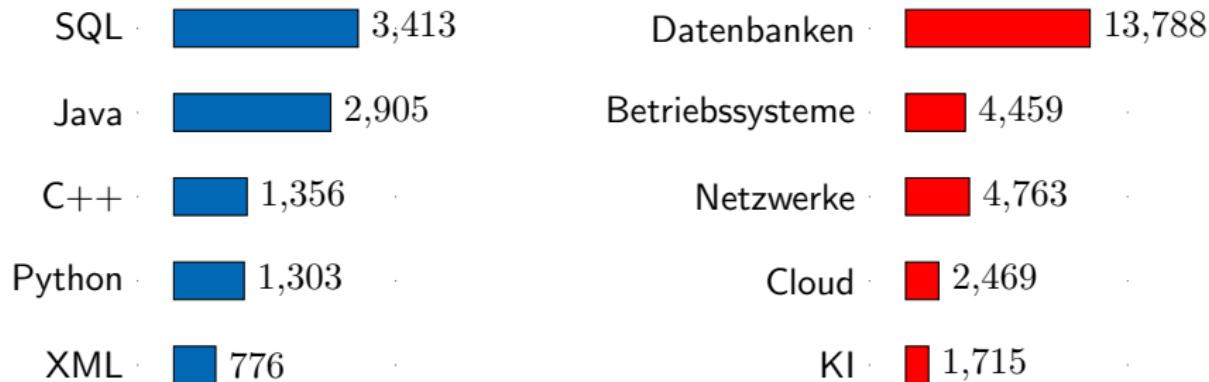


Motivation

- Datenbanksysteme sind Herzstück heutiger IT-Infrastrukturen
- ... allgegenwärtig
- Datenbank-spezialisten sind gefragt



Motivation: Stellenangebote (Quelle: monster.com)



- 1. Wie organisiert (modelliert und nutzt) man Daten?**
2. Wie werden Daten dauerhaft verlässlich gespeichert?
3. Wie kann man riesige Datenmengen (\geq Terabytes) effizient verarbeiten?
4. Wie können viele Nutzer (≥ 10.000) gleichzeitig mit den Daten arbeiten?

Architekturen

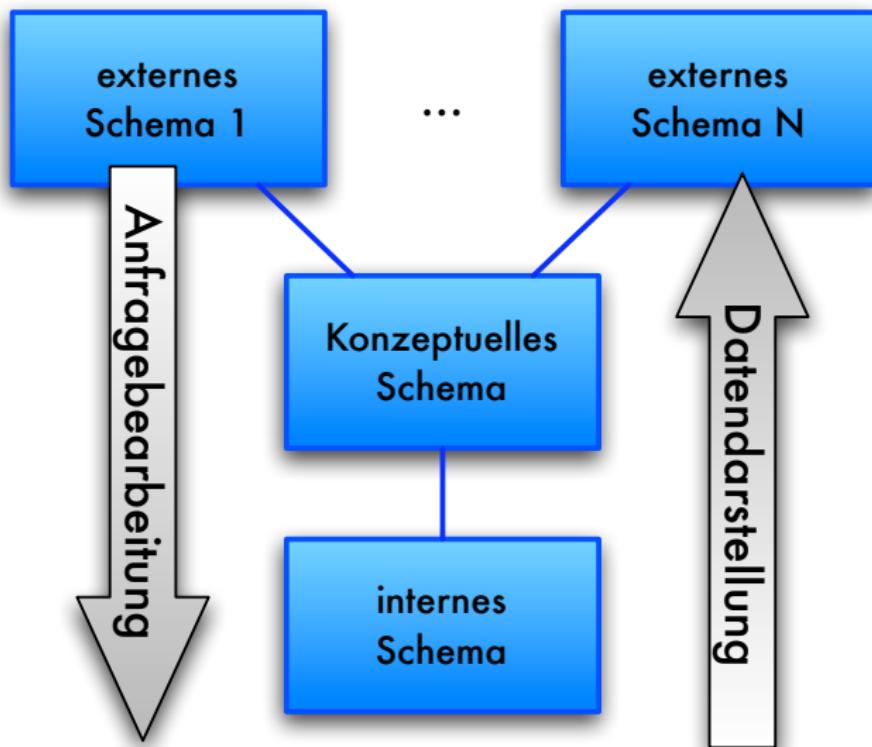
1. **Integration:** einheitliche, nichtredundante Datenverwaltung
2. **Operationen:** Speichern, Suchen, Ändern
3. **Katalog:** Zugriffe auf Datenbankbeschreibungen im Data Dictionary
4. **Benutzersichten**
5. **Integritätssicherung:** Korrektheit des Datenbankinhalts
6. **Datenschutz:** Ausschluss unauthorisierter Zugriffe
7. **Transaktionen:** mehrere DB-Operationen als Funktionseinheit
8. **Synchronisation:** parallele Transaktionen koordinieren
9. **Datensicherung:** Wiederherstellung von Daten nach Systemfehlern

- Basierend auf DBMS-Grobarchitektur
- Entkopplung von Benutzer- und Implementierungssicht
- Ziele u.a.:
 - Trennung von Modellierungssicht und interner Speicherung
 - Portierbarkeit
 - Tuning vereinfachen
 - standardisierte Schnittstellen

- Zusammenhang zwischen
 - Konzeptuellem Schema (Ergebnis der Datendefinition)
 - Internem Schema (Festlegung der Dateiorganisationen und Zugriffspfade)
 - Externen Schemata (Ergebnis der Sichtdefinition)
 - Anwendungsprogrammen (Ergebnis der Anwendungsprogrammierung)

- Trennung Schema — Instanz
 - Schema (Metadaten, Datenbeschreibungen)
 - Instanz (Anwenderdaten, Datenbankzustand oder -ausprägung)
- Datenbankschema besteht aus
 - internem, konzeptuellem, externen Schemata und den Anwendungsprogrammen
- im konzeptuellen Schema etwa:
 - Strukturbeschreibungen
 - Integritätsbedingungen
 - Autorisierungsregeln (pro Benutzer für erlaubte DB-Zugriffe)

Schemaarchitektur /3



Datenunabhängigkeit /2

- Stabilität der Benutzerschnittstelle gegen Änderungen
- **physisch:** Änderungen der Dateiorganisationen und Zugriffspfade haben keinen Einfluss auf das konzeptuelle Schema
- **logisch:** Änderungen am konzeptuellen und gewissen externen Schemata haben keine Auswirkungen auf andere externe Schemata und Anwendungsprogramme

Datenunabhängigkeit /3

- mögliche Auswirkungen von Änderungen am konzeptuellen Schema:
 - eventuell externe Schemata betroffen (Ändern von Attributen)
 - eventuell Anwendungsprogramme betroffen (Rekompilieren der Anwendungsprogramme, eventuell Änderungen nötig)
- nötige Änderungen werden jedoch vom DBMS erkannt und überwacht

Anwendungsbeispiel: Mitfahरgelegenheit

- Angebote von Mitfahरgelegenheiten
 - Wann? Von wo? Wohin? Wer? Plätze?
 - Kontaktdaten
 - Reservierungsmöglichkeiten



CC-BY-2.0: Luo Shaoyang

Ebenen-Architektur am Beispiel

- Konzeptuelle Sicht: Darstellung in Tabellen (Relationen)

Fahrer	<u>FahrerID</u>	Name	Telefon
	103	Lilo Pause	01234
	104	Just Vorfan	01246
	105	Heiko Heizer	01756

Mitfahrangebot	<u>ANr</u>	Abfahrt	Ankunft	Zeit	FahrerID
	1014	Ilmenau	Erfurt	Freitag 10:00	103
	1015	Magdeburg	Halle	Freitag 15:00	104
	1016	Magdeburg	Leipzig	Freitag 15:00	104
	1021	Magdeburg	Ilmenau	Freitag 14:00	105
	1025	Ilmenau	Jena	Freitag 10:00	103

Ebenen-Architektur am Beispiel /2

- Externe Sicht: Daten in einer flachen Relation

ANr	Abfahrt	Ankunft	Zeit	Fahrer
1014	Ilmenau	Erfurt	Freitag 10:00	Lilo Pause
1015	Magdeburg	Halle	Freitag 15:00	Just Vorfan
1016	Magdeburg	Leipzig	Freitag 15:00	Just Vorfan
1021	Magdeburg	Ilmenau	Freitag 14:00	Heiko Heizer
1025	Ilmenau	Jena	Freitag 10:00	Lilo Pause

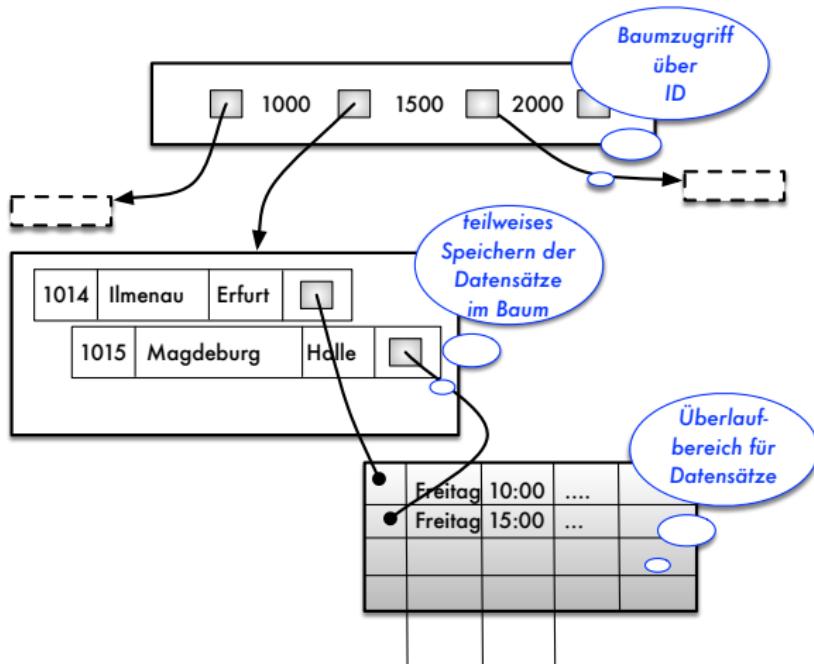
Ebenen-Architektur am Beispiel /3

- Externe Sicht: Daten in einer hierarchisch aufgebauten Relation

Fahrer	Mitfahrangebot		
	Abfahrt	Ankunft	Zeit
Lilo Pause	Ilmenau	Erfurt	Freitag 10:00
		Jena	Freitag 10:00
Just Vorfan	Magdeburg	Halle	Freitag 15:00
		Leipzig	Freitag 15:00
Heiko Heizer	Magdeburg	Ilmenau	Freitag 14:00

Ebenen-Architektur am Beispiel /4

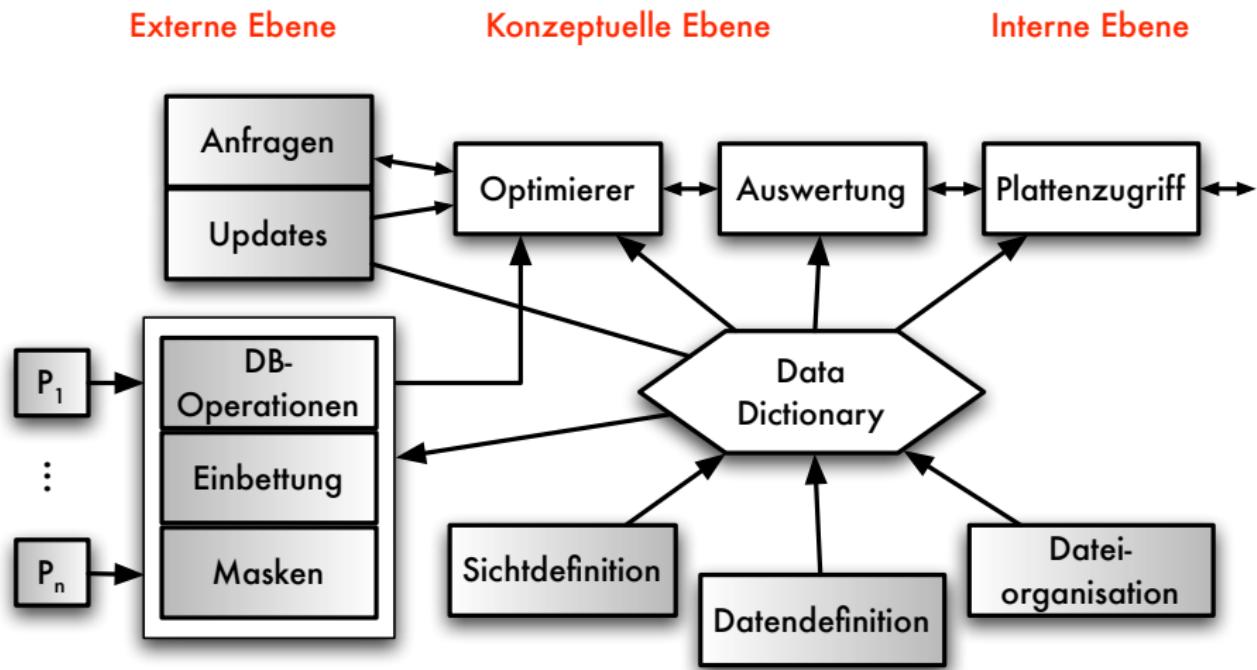
- Interne Darstellung



- Beschreibung der Komponenten eines Datenbanksystems
- Standardisierung der Schnittstellen zwischen Komponenten
- Architekturvorschläge
 - ANSI-SPARC-Architektur
 - ~~> Drei-Ebenen-Architektur
 - Fünf-Schichten-Architektur
 - ~~> beschreibt Transformationskomponenten im Detail
 - Vorlesung „Datenbank-Implementierungstechniken“**

- ANSI: American National Standards Institute
- SPARC: Standards Planning and Requirement Committee
- Vorschlag von 1978
- Im Wesentlichen Grobarchitektur verfeinert
 - Interne Ebene / Betriebssystem verfeinert
 - Mehr Interaktive und Programmier-Komponenten
 - Schnittstellen bezeichnet und normiert

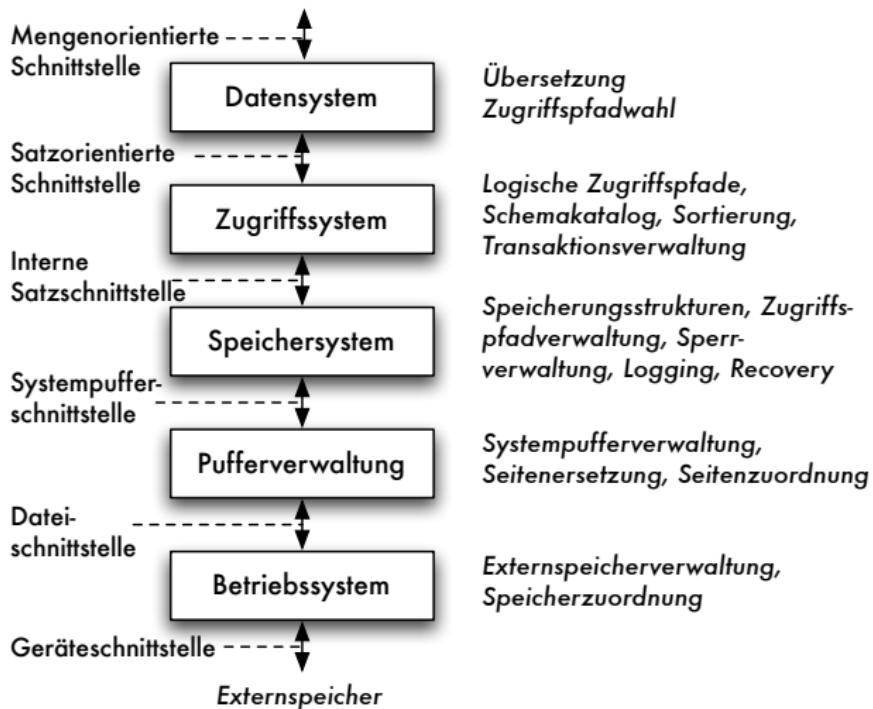
ANSI-SPARC-Architektur /2



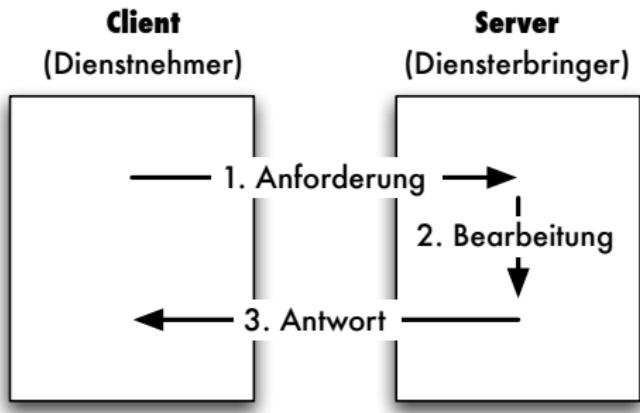
Klassifizierung der Komponenten

- Definitionskomponenten: Datendefinition, Dateiorganisation, Sichtdefinition
- Programmierkomponenten: DB-Programmierung mit eingebetteten DB-Operationen
- Benutzerkomponenten: Anwendungsprogramme, Anfrage und Update interaktiv
- Transformationskomponenten: Optimierer, Auswertung, Plattenzugriffssteuerung
- Data Dictionary (Datenwörterbuch): Aufnahme der Daten aus Definitionskomponenten, Versorgung der anderen Komponenten

Fünf-Schichten-Architektur: Verfeinerung der Transformation

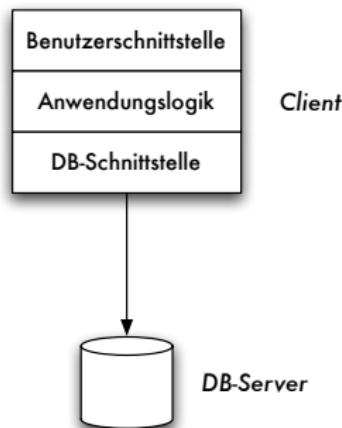


- Architektur von Datenbankanwendungen typischerweise auf Basis des Client-Server-Modells: Server \equiv Datenbanksystem

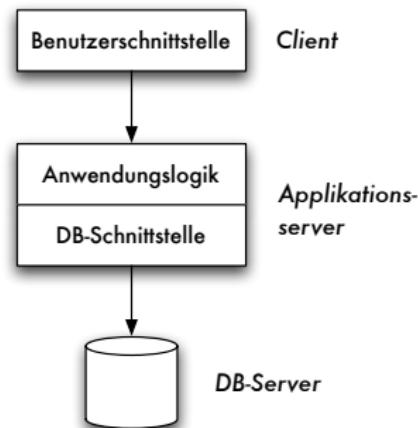


Anwendungsarchitekturen /2

- Aufteilung der Funktionalitäten einer Anwendung
 - Präsentation und Benutzerinteraktion
 - Anwendungslogik („Business“-Logik)
 - Datenmanagementfunktionen (Speichern, Anfragen, ...).



Zwei-Schichten-Architektur



Drei-Schichten-Architektur

Einsatzgebiete

- (Objekt-)Relationale DBMS
 - Oracle21c, IBM DB2 UDB V.11, Microsoft SQL Server 2017, SAP HANA
 - MySQL (www.mysql.org) & Friends (MariaDB), PostgreSQL (www.postgresql.org)
 - Amazon Aurora, Amazon Redshift, Microsoft Azure SQL, ...
- Pseudo-DBMS
 - MS Access
- NoSQL-Systeme
 - Graph-Datenbanksysteme (InfiniteGraph, neo4j), Dokument-Datenbanken (MongoDB), Key-Value-Stores, ...

- Klassische Einsatzgebiete:
 - viele Objekte (100.000 Bücher, 10.000 Benutzer, 1000 Ausleihvorgänge pro Tag, ...)
 - wenige Objekttypen (BUCH, BENUTZER, AUSLEIHUNG)
 - etwa Buchhaltungs- bzw. ERP-Systeme, Bestellsysteme, Bibliothekssysteme, ...
- Aktuelle Anwendungen:
 - E-Commerce, Buchung- und Abrechnungssysteme, entscheidungsunterstützende Systeme (Data Warehouses, OLAP), wiss. Anwendungen (Erdbeobachtung, Klima), ...

Datenbankgrößen

Kommerzielle Systeme, Zahlen aus 2010

eBay Data Warehouse 10 PB ($\approx 10 \cdot 10^{15}$ Bytes)

Teradata DBMS, 72 Knoten, 10.000 Nutzer,
mehrere Millionen Anfragen/Tag

WalMart Data Warehouse 2,5 PB

Teradata DBMS, NCR MPP-Hardware;
Produktinfos (Verkäufe etc.) von 2.900 Märkten;
50.000 Anfragen/Woche

Historisches

- Anfang 60er Jahre: elementare Dateien, anwendungsspezifische Datenorganisation (geräteabhängig, redundant, inkonsistent)
- Ende 60er Jahre: Dateiverwaltungssysteme (SAM, ISAM) mit Dienstprogrammen (Sortieren) (geräteunabhängig, aber redundant und inkonsistent)
- DBS basierend auf **hierarchischem Modell, Netzwerkmodell**
 - Zeigerstrukturen zwischen Daten
 - Schwache Trennung interne / konzeptuelle Ebene
 - Navigierende DML
 - Trennung DML / Programmiersprache

- 70er Jahre: Datenbanksysteme (Geräte- und Datenunabhängigkeit, redundanzfrei, konsistent)
- **Relationale Datenbanksysteme**
 - Daten in Tabellenstrukturen
 - 3-Ebenen-Konzept
 - Deklarative DML
 - Trennung DML / Programmiersprache

Historie von RDBMS

- 1970: Ted Codd (IBM) → Relationenmodell als konzeptionelle Grundlage relationaler DBS
- 1974: System R (IBM) → erster Prototyp eines RDBMS
 - zwei Module: RDS, RSS; ca. 80.000 LOC (PL/1, PL/S, Assembler), ca. 1,2 MB Codegröße
 - Anfragesprache SEQUEL
 - erste Installation 1977
- 1975: University of California at Berkeley (UCB) → Ingres
 - Anfragesprache QUEL
 - Vorgänger von Postgres, Sybase, ...
- 1979: Oracle Version 2

- **Wissensbanksysteme**
 - Daten in Tabellenstrukturen
 - Stark deklarative DML, integrierte Datenbankprogrammiersprache
- **Objektorientierte Datenbanksysteme**
 - Daten in komplexeren Objektstrukturen (Trennung Objekt und seine Daten)
 - Deklarative oder navigierende DML
 - Oft integrierte Datenbankprogrammiersprache
 - Oft keine vollständige Ebenentrennung

- Neue Hardwarearchitekturen
 - Multicore-Prozessoren, Hauptspeicher im TB-Bereich:
In-Memory-Datenbanksysteme (z.B. SAP HANA)
 - Hardwarebeschleunigung durch FPGA oder GPU
- Unterstützung für spezielle Anwendungen
 - **Cloud-Datenbanken:** Hosting von Datenbanken, Skalierbare Datenmanagementlösungen (Amazon RDS, Microsoft Azure)
 - **Datenstromverarbeitung:** Online-Verarbeitung von Live-Daten, z.B. Börseninfos, Sensordaten, RFID-Daten, ... (StreamBase, MS StreamInsight, IBM InfoSphere Streams)
 - **Big Data:** Umgang mit Datenmengen im PB-Bereich durch hochskalierbare, parallele Verarbeitung, Datenanalyse (Hadoop, Hive, Google Spanner & F1, ...)

- **NoSQL-Datenbanken** („Not only SQL“):
 - nicht-relationale Datenbanken, flexibles Schema (dokumentenzentriert)
 - „leichtgewichtig“ durch Weglassen von SQL-Funktionalitäten wie Transaktionen, mächtige deklarative Anfragesprachen mit Verbunden etc.
 - Beispiele: CouchDB, MongoDB, Cassandra, ...

- Nutzergenerierte Inhalte, z.B. Google:
 - Verarbeitung von 20 PB täglich
 - 15h Video-Upload auf YouTube in jeder Minute
 - Lesen von 20 PB würde 12 Jahre benötigen bei 50 MB/s-Festplatte
- Linked Data und Data Web
 - Bereitstellung, Austausch und Verknüpfung von strukturierten Daten im Web
 - ermöglicht Abfrage (mit Anfragesprachen wie SPARQL) und Weiterverarbeitung
 - Beispiele: DBpedia, GeoNames

Zusammenfassung

- Motivation für Einsatz von Datenbanksystemen
- Codd'sche Regeln
- 3-Ebenen-Schemaarchitektur & Datenunabhängigkeit
- Einsatzgebiete

Kontrollfragen

- Welchen Vorteil bieten Datenbanksysteme gegenüber einer anwendungsspezifischen Speicherung von Daten?
- Was versteht man unter Datenunabhängigkeit und wie wird sie erreicht?
- In welchen Bereichen kommen Datenbanksysteme zum Einsatz?



Teil II

Relationale Datenbanken – Daten als Tabellen

1. Relationen für tabellarische Daten
2. SQL-Datendefinition
3. Grundoperationen: Die Relationalalgebra
4. SQL als Anfragesprache
5. Änderungsoperationen in SQL
6. Anwendungsbeispiel

Lernziele für heute . . .

- Grundverständnis zur Struktur relationaler Datenbanken
- Kenntnis der Basisoperationen relationaler Anfragesprachen
- elementare Fähigkeiten in der Anwendung von SQL



Relationen für tabellarische Daten

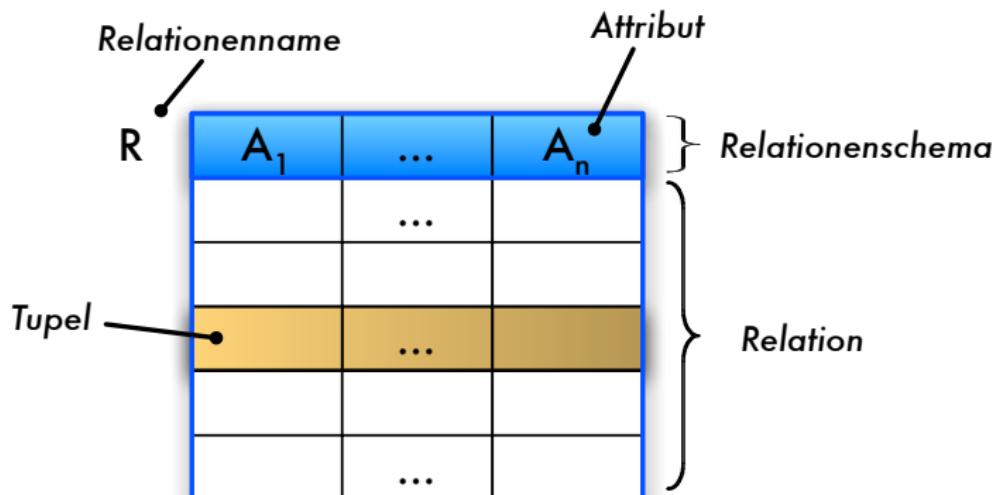
Relationenmodell

Konzeptuell: Datenbank = Menge von Tabellen (= Relationen)

WEINE	WeinID	Name	Farbe	Jahrgang	Weingut
	1042	La Rose Grand Cru	Rot	1998	Château La Rose
	2168	Creek Shiraz	Rot	2003	Creek
	3456	Zinfandel	Rot	2004	Helena
	2171	Pinot Noir	Rot	2001	Creek
	3478	Pinot Noir	Rot	1999	Helena
	4711	Riesling Reserve	Weiβ	1999	Müller
	4961	Chardonnay	Weiβ	2002	Bighorn

ERZEUGER	Weingut	Anbaugebiet	Region
	Creek	Barossa Valley	Südaustralien
	Helena	Napa Valley	Kalifornien
	Château La Rose	Saint-Emilion	Bordeaux
	Château La Pointe	Pomerol	Bordeaux
	Müller	Rheingau	Hessen
	Bighorn	Napa Valley	Kalifornien

- „Tabellenkopf“: **Relationenschema**
- Eine Zeile der Tabelle: **Tupel**; Menge aller Einträge: **Relation**
- Eine Spaltenüberschrift: **Attribut**
- Ein Eintrag: **Attributwert**



Integritätsbedingungen: Schlüssel

- Attribute einer Spalte **identifizieren** eindeutig gespeicherte Tupel:
Schlüsseleigenschaft
- etwa **Weingut** für Tabelle **ERZEUGER**

ERZEUGER	<u>Weingut</u>	Anbaugebiet	Region
	Creek	Barossa Valley	Südaustralien
	Helena	Napa Valley	Kalifornien
	Château La Rose	Saint-Emilion	Bordeaux
	Château La Pointe	Pomerol	Bordeaux
	Müller	Rheingau	Hessen
	Bighorn	Napa Valley	Kalifornien

- auch Attributkombinationen können Schlüssel sein!
- Schlüssel können durch Unterstrichen gekennzeichnet werden

Integritätsbedingungen: Fremdschlüssel



- Schlüssel einer Tabelle können in einer anderen (oder derselben!) Tabelle als eindeutige Verweise genutzt werden: **Fremdschlüssel, referentielle Integrität**
- etwa **Weingut** als Verweise auf **ERZEUGER**
- ein Fremdschlüssel ist ein **Schlüssel in einer „fremden“ Tabelle**

Fremdschlüssel /2

WEINE	WeinID	Name	Farbe	Jahrgang	Weingut → ERZEUGER
	1042	La Rose ...	Rot	1998	Château La Rose
	2168	Creek Shiraz	Rot	2003	Creek
	3456	Zinfandel	Rot	2004	Helena
	2171	Pinot Noir	Rot	2001	Creek
	3478	Pinot Noir	Rot	1999	Helena
	4711	Riesling ...	Weiß	1999	Müller
	4961	Chardonnay	Weiß	2002	Bighorn

ERZEUGER	Weingut	Anbaugebiet	Region
	Creek	Barossa Valley	Südaustralien
	Helena	Napa Valley	Kalifornien
	Château La Rose	Saint-Emilion	Bordeaux
	Château La Pointe	Pomerol	Bordeaux
	Müller	Rheingau	Hessen
	Bighorn	Napa Valley	Kalifornien

SQL-Datendefinition

Die Anweisung create table

```
create table basisrelationenname (
    spaltenname1 wertebereich1 [not null],
    ...
    spaltennamek wertebereichk [not null])
```

- Wirkung dieses Kommandos ist sowohl
 - die Ablage des **Relationenschemas** im Data Dictionary, als auch
 - die Vorbereitung einer „**leeren Basisrelation**“ in der Datenbank

Löschen einer Tabelle: drop table

- komplettes Löschen einer Tabelle (Inhalt und Eintrag im Data Dictionary)

```
drop table basisrelationenname
```

Mögliche Wertebereiche in SQL

- integer (oder auch integer4, int),
- smallint (oder auch integer2),
- float(p) (oder auch kurz float),
- decimal(p,q) und numeric(p,q) mit jeweils q Nachkommastellen,
- character(n) (oder kurz char(n), bei $n = 1$ auch char) für Zeichenketten (Strings) fester Länge n ,
- character varying(n) (oder kurz varchar(n) für Strings variabler Länge bis zur Maximallänge n ,
- bit(n) oder bit varying(n) analog für Bitfolgen, und
- date, time bzw. datetime für Datums-, Zeit- und kombinierte Datums-Zeit-Angaben

Beispiel für create table

```
create table WEINE (
    WeinID int primary key,
    Name varchar(20) not null,
    Farbe varchar(10),
    Jahrgang int,
    Weingut varchar(20))
```

- primary key kennzeichnet Spalte als **Schlüsselattribut**

create table mit Fremdschlüssel



```
create table WEINE (
    WeinID int,
    Name varchar(20) not null,
    Farbe varchar(10),
    Jahrgang int,
    Weingut varchar(20),
    primary key(WeinID),
    foreign key(Weingut)
        references ERZEUGER(Weingut))
```

- `foreign key` kennzeichnet Spalte als **Fremdschlüssel**

- `not null` schließt in bestimmten Spalten **Nullwerte** als Attributwerte aus
- Kennzeichnung von Nullwerte in SQL durch `null`; hier \perp
- `null` repräsentiert die Bedeutung „*Wert unbekannt*“, „*Wert nicht anwendbar*“ oder „*Wert existiert nicht*“, gehört aber zu keinem Wertebereich
- `null` kann in allen Spalten auftauchen, außer in Schlüsselattributen und den mit `not null` gekennzeichneten

Grundoperationen: Die Relationalalgebra

- **Basisoperationen** auf Tabellen, die die Berechnung von neuen Ergebnistabellen aus gespeicherten Datenbanktabellen erlauben
- Operationen werden zur sogenannten **Relationenalgebra** zusammengefasst
- Mathematik: Algebra ist definiert durch Wertebereich sowie darauf definierten Operationen
 - für Datenbankanfragen entsprechen die Inhalte der Datenbank den Werten, Operationen sind dagegen **Funktionen zum Berechnen der Anfrageergebnisse**
- Anfrageoperationen sind **beliebig kombinierbar** und bilden eine Algebra zum „Rechnen mit Tabellen“ – die **Relationenalgebra**

Relationenalgebra: Übersicht

Selektion

Projektion

Verbund

A	B
a1	b2
a2	b2
a2	b3

B	C
b2	c3
b3	c4
b4	c5

A	B	C
a1	b2	c3
a2	b2	c3
a2	b3	c4

Selektion σ

- **Selektion:** Auswahl von Zeilen einer Tabelle anhand eines Selektionsprädikats

$$\sigma_{\text{Jahrgang} > 2000}(\text{WEINE})$$

WeinID	Name	Farbe	Jahrgang	Weingut
2168	Creek Shiraz	Rot	2003	Creek
3456	Zinfandel	Rot	2004	Helena
2171	Pinot Noir	Rot	2001	Creek
4961	Chardonnay	Weiβ	2002	Bighorn

Projektion π

- **Projektion:** Auswahl von Spalten durch Angabe einer Attributliste

$$\pi_{\text{Region}}(\text{ERZEUGER})$$

Region
Südaustralien
Kalifornien
Bordeaux
Hessen

- Die Projektion entfernt doppelte Tupel.

Natürlicher Verbund ✖

- **Verbund** (engl. *join*): verknüpft Tabellen über **gleichbenannte Spalten**, indem er jeweils zwei Tupel verschmilzt, falls sie dort **gleiche Werte** aufweisen

Natürlicher Verbund: Beispiel

WEINE \bowtie ERZEUGER

WeinID	Name	...	Weingut	Anbaugebiet	Region
1042	La Rose Grand Cru	...	Ch. La Rose	Saint-Emilion	Bordeaux
2168	Creek Shiraz	...	Creek	Barossa Valley	Südaustralien
3456	Zinfandel	...	Helena	Napa Valley	Kalifornien
2171	Pinot Noir	...	Creek	Barossa Valley	Südaustralien
3478	Pinot Noir	...	Helena	Napa Valley	Kalifornien
4711	Riesling Reserve	...	Müller	Rheingau	Hessen
4961	Chardonnay	...	Bighorn	Napa Valley	Kalifornien

- Das Weingut „Château La Pointe“ ist im Ergebnis verschwunden ~
Tupel, die keinen Partner finden (dangling tuples), werden eliminiert

Kombination von Operationen

$$\pi_{\text{Name}, \text{Farbe}, \text{Weingut}}(\sigma_{\text{Jahrgang} > 2000}(\text{WEINE}) \bowtie \sigma_{\text{Region} = \text{'Kalifornien'}}(\text{ERZEUGER}))$$

ergibt

Name	Farbe	Weingut
Zinfandel	Rot	Helena
Chardonnay	Wei�	Bighorn

Umbenennung β

- Anpassung von Attributnamen mittels **Umbenennung**:

WEINLISTE	Name
	La Rose Grand Cru
	Creek Shiraz
	Zinfandel
	Pinot Noir
	Riesling Reserve

EMPFEHLUNG	Wein
	La Rose Grand Cru
	Riesling Reserve
	Merlot Selection
	Sauvignon Blanc

- Angleichen durch:

$$\beta_{\text{Name} \leftarrow \text{Wein}} (\text{EMPFEHLUNG})$$

Mengenoperationen

- **Vereinigung** $r_1 \cup r_2$ von zwei Relationen r_1 und r_2 : Gesamtheit der beiden Tupelmengen
- Attributmengen beider Relationen müssen identisch sein

WEINLISTE $\cup \beta_{\text{Name} \leftarrow \text{Wein}}(\text{EMPFEHLUNG})$

Name
La Rose Grand Cru
Creek Shiraz
Zinfandel
Pinot Noir
Riesling Reserve
Merlot Selection
Sauvignon Blanc

Mengenoperationen /2

- **Differenz** $r_1 - r_2$ eliminiert die Tupel aus der ersten Relation, die auch in der zweiten Relation vorkommen

WEINLISTE – $\beta_{\text{Name} \leftarrow \text{Wein}}(\text{EMPFEHLUNG})$

ergibt:

Name
Creek Shiraz
Zinfandel
Pinot Noir

- **Durchschnitt** $r_1 \cap r_2$: ergibt die Tupel, die in beiden Relationen gemeinsam vorkommen

$$\text{WEINLISTE} \cap \beta_{\text{Name} \leftarrow \text{Wein}}(\text{EMPFEHLUNG})$$

liefert:

Name
La Rose Grand Cru Riesling Reserve

SQL als Anfragesprache

SQL-Anfrage als Standardsprache



- Anfrage an eine einzelne Tabelle

```
select Name, Farbe  
from WEINE  
where Jahrgang = 2002
```

- SQL hat **Multimengensemantik** — Duplikate in Tabellen werden in SQL nicht automatisch unterdrückt!
- Mengensemantik durch `distinct`

```
select distinct Name from WEINE
```

Verknüpfung von Tabellen

- Kreuzprodukt als Basisverknüpfung

```
select *  
from WEINE, ERZEUGER
```

- Verbund durch Operator natural join

```
select *  
from WEINE natural join ERZEUGER
```

Verknüpfung von Tabellen /2

- Verbund alternativ durch Angabe einer **Verbundbedingung!**

```
select *  
from WEINE, ERZEUGER  
where WEINE.Weingut = ERZEUGER.Weingut
```

Kombination von Bedingungen

- Ausdruck in Relationenalgebra

$$\pi_{\text{Name}, \text{Farbe}, \text{Weingut}}(\sigma_{\text{Jahrgang} > 2000}(\text{WEINE}) \bowtie \sigma_{\text{Region} = \text{'Kalifornien'}}(\text{ERZEUGER}))$$

- Anfrage in SQL

```
select Name, Farbe, WEINE.Weingut
from WEINE, ERZEUGER
where Jahrgang > 2000 and
      Region = 'Kalifornien' and
      WEINE.Weingut = ERZEUGER.Weingut
```

Mengenoperationen in SQL

- Vereinigung in SQL explizit mit union
- Differenzbildung durch geschachtelte Anfragen

```
select *
from WINZER
where Name not in (
    select Nachname
    from KRITIKER)
```

Änderungsoperationen in SQL

- insert: **Einfügen** eines oder mehrerer Tupel in eine Basisrelation oder Sicht
- update: **Ändern** von einem oder mehreren Tupel in einer Basisrelation oder Sicht
- delete: **Löschen** eines oder mehrerer Tupel aus einer Basisrelation oder Sicht
- **Lokale und globale Integritätsbedingungen müssen bei Änderungsoperationen automatisch vom System überprüft werden**

Die update-Anweisung

- Syntax:

```
update basisrelation
set      attribut1 = ausdruck1
        ...
attributn = ausdruckn
[ where bedingung ]
```

Beispiel für update

WEINE	WeinID	Name	Jahrgang	Weingut	Preis
	2168	Creek Shiraz	2003	Creek	7.99
	3456	Zinfandel	2004	Helena	5.99
	2171	Pinot Noir	2001	Creek	10.99
	3478	Pinot Noir	1999	Helena	19.99
	4711	Riesling Reserve	1999	Müller	14.99
	4961	Chardonnay	2002	Bighorn	9.90

```
update WEINE
set Preis = Preis * 1.10
where Jahrgang < 2000
```

Beispiel für update: neue Werte

WEINE	WeinID	Name	Jahrgang	Weingut	Preis
	2168	Creek Shiraz	2003	Creek	7.99
	3456	Zinfandel	2004	Helena	5.99
	2171	Pinot Noir	2001	Creek	10.99
	3478	Pinot Noir	1999	Helena	21.99
	4711	Riesling Reserve	1999	Müller	16.49
	4961	Chardonnay	2002	Bighorn	9.90

Weiteres zu update

- Realisierung von Eintupel-Operation mittels Primärschlüssel:

```
update WEINE
set Preis = 7.99
where WeinID = 3456
```

- Änderung der gesamten Relation:

```
update WEINE
set Preis = 11
```

Die delete-Anweisung

- Syntax:

```
delete
from basisrelation
[ where bedingung ]
```

- Löschen eines Tupels in der WEINE-Relation:

```
delete from WEINE
where WeinID = 4711
```

Weiteres zu delete

- Standardfall ist das Löschen mehrerer Tupel:

```
delete from WEINE
where Farbe = 'Weiß'
```

- Löschen der gesamten Relation:

```
delete from WEINE
```

Weiteres zu delete /2

- Löschoperationen können zur Verletzung von Integritätsbedingungen führen!
- Beispiel: Verletzung der Fremdschlüsseleigenschaft, falls es noch Weine von diesem Erzeuger gibt:

```
delete from ERZEUGER
where Anbaugebiet = 'Hessen'
```

Die insert-Anweisung

- Syntax:

```
insert
into basisrelation
  [ (attribut1, ..., attributn) ]
values (konstante1, ..., konstanten)
```

- optionale Attributliste ermöglicht das Einfügen von unvollständigen Tupeln

insert-Beispiele

```
insert into ERZEUGER (Weingut, Region)
values ('Wairau Hills', 'Marlborough')
```

- nicht alle Attribute angegeben ~> Wert des fehlenden Attribut Land wird null

```
insert into ERZEUGER
values ('Château Lafitte', 'Medoc', 'Bordeaux')
```

Einfügen von berechneten Daten

- Syntax:

```
insert
into basisrelation [ (attribut1, ..., attributn) ]
    SQL-anfrage
```

- Beispiel:

```
insert into WEINE (
    select ProdID, ProdName, 'Rot', ProdJahr,
        'Château Lafitte'
    from LIEFERANT where LName = 'Wein-Kontor' )
```

Anwendungsbeispiel

Mitfahrzentrale

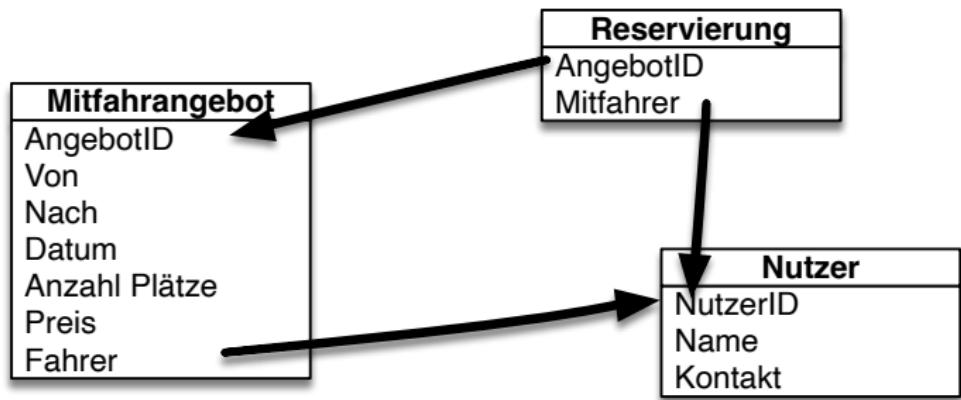
- Welche Daten?

- **Mitfahrangebote:** Wann? Von wo? Wohin? Wer? Plätze?
- **Nutzer:** Anmeldung, Kontaktdaten
- **Reservierung:** Wer? Welches Angebot?



CC-BY-2.0: Luo Shaoyang

Mitfahrzentrale: Datenbank



Mitfahrzentrale: Datenbank in SQL



```
create table Nutzer (
    NutzerID varchar(10) primary key,
    Name varchar(100),
    Kontakt varchar(500));
```

Mitfahrzentrale: Datenbank in SQL /2



```
create table Mitfahrangebot (
    AngebotID int primary key,
    Von varchar(100) not null,
    Nach varchar(100) not null,
    Datum date not null,
    AnzPlaetze int,
    Preis decimal,
    Fahrer varchar(10)
        references Nutzer(NutzerID));
```

Mitfahrzentrale: Datenbank in SQL /3



```
create table Reservierung (
    AngebotID int
        references Mitfahrangebot(AngebotID),
    Mitfahrer varchar(10)
        references Nutzer(NutzerID));
```

- Welche Angebote gibt es heute von Ilmenau nach Erfurt?

```
select * from Mitfahrangebot
where Von = 'Ilmenau' and Nach = 'Erfurt'
      and Datum = date('now');
```

- Reservierung für eine bestimmte Mitfahrgelegenheit

```
insert into Reservierung values (1, 'holgi');
```

Mitfahrzentrale: Anfragen /2



- Wer will bei mir mitfahren?

```
select R.Mitfahrer  
from Reservierung R, Mitfahrangebot M  
where R.AngebotID = M.AngebotID  
    and M.Fahrer = 'heike';
```

Zusammenfassung

- Relationenmodell: Datenbank als Sammlung von Tabellen
- Integritätsbedingungen im Relationenmodell
- Tabellendefinition in SQL
- Relationenalgebra: Anfrageoperatoren
- Grundkonzepte von SQL-Anfragen und -Änderungen

Kontrollfragen

- Was ist eine Relation?
- Was definiert die Relationenalgebra?
- Wie wird eine Realweltobjekt in einer relationalen Datenbank repräsentiert?
- Wie werden Tabellen in SQL definiert und manipuliert?
- Was sind Integritätsbedingungen?



Teil III

Entity-Relationship-Modell

Entity-Relationship-Modell



1. Datenbankmodelle
2. ER-Modell
3. Weitere Konzepte im ER-Modell

Lernziele für heute . . .

- Kenntnis der Konzepte des Entity-Relationship-Modells
- Fähigkeiten zur konzeptuellen Modellierung eines Anwendungsbereichs



Datenbankmodelle

Datenbankmodell

Ein **Datenbankmodell** ist ein System von Konzepten zur Beschreibung von Datenbanken. Es legt Syntax und Semantik von Datenbankbeschreibungen für ein Datenbanksystem fest.

- Datenbankbeschreibungen = Datenbankschemata

1. **statische Eigenschaften**

1.1 Objekte

1.2 Beziehungen

inklusive der Standard-Datentypen, die Daten über die Beziehungen und Objekte darstellen können,

2. **dynamische Eigenschaften** wie

2.1 Operationen

2.2 Beziehungen zwischen Operationen,

3. **Integritätsbedingungen** an

3.1 Objekte

3.2 Operationen

- Klassische Datenbankmodelle sind speziell geeignet für
 - große Informationsmengen mit relativ starrer Struktur und
 - die Darstellung statischer Eigenschaften und Integritätsbedingungen (also die Bereiche 1(a), 1(b) und 3(a))
- Entwurfsmodelle: (E)ER-Modell, UML, ...
- Realisierungsmodelle: Relationenmodell, objektorientierte Modelle,
...

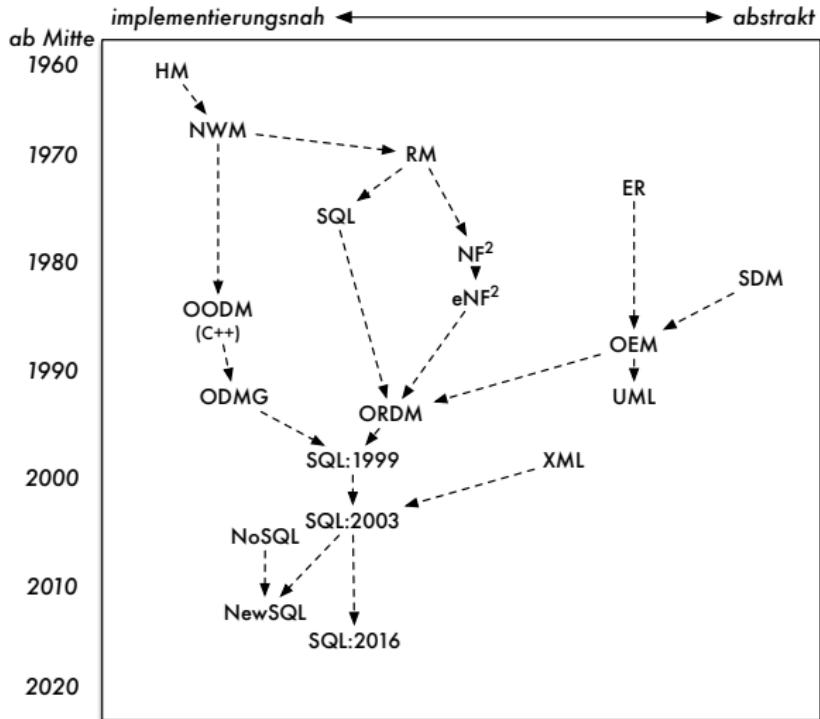
Datenbanken versus Programmiersprachen

Datenbankkonzept	Typsystem einer Programmiersprache
Datenbankmodell <i>Relation, Attribut ...</i>	Typsystem int, struct ...
Datenbankschema <code>relation WEIN = (...)</code>	Variablen-deklaration <code>var x: int,</code> <code>y: struct Wein</code>
Datenbank <code>WEIN(4961, 'Chardonnay',</code> <code>'Weiß', ...)</code>	Werte 42, 'Cabernet Sauvignon' 42, 'Cabernet Sauvignon'

Abstraktionsstufen

Modelle	Daten	Algorithmen
abstrakt	Entity-Relationship-Modell	Struktogramme
konkret	Hierarchisches Modell Netzwerkmodell Relationenmodell	Pascal C, C++ Java, C#

Datenbankmodelle im Überblick



Datenbankmodelle im Überblick /2

- HM: hierarchisches Modell, NWM: Netzwerkmodell, RM: Relationenmodell
- NF²: Modell der geschachtelten (Non-First-Normal-Form = NF²) Relationen, eNF²: erweitertes NF²-Modell
- ER: Entity-Relationship-Modell, SDM: semantische Datenmodelle
- OODM / C++: objektorientierte Datenmodelle auf Basis objektorientierter Programmiersprachen wie C++, OEM: objektorientierte Entwurfsmodelle (etwa UML), ORDM: objektrelationale Datenmodelle

ER-Modell

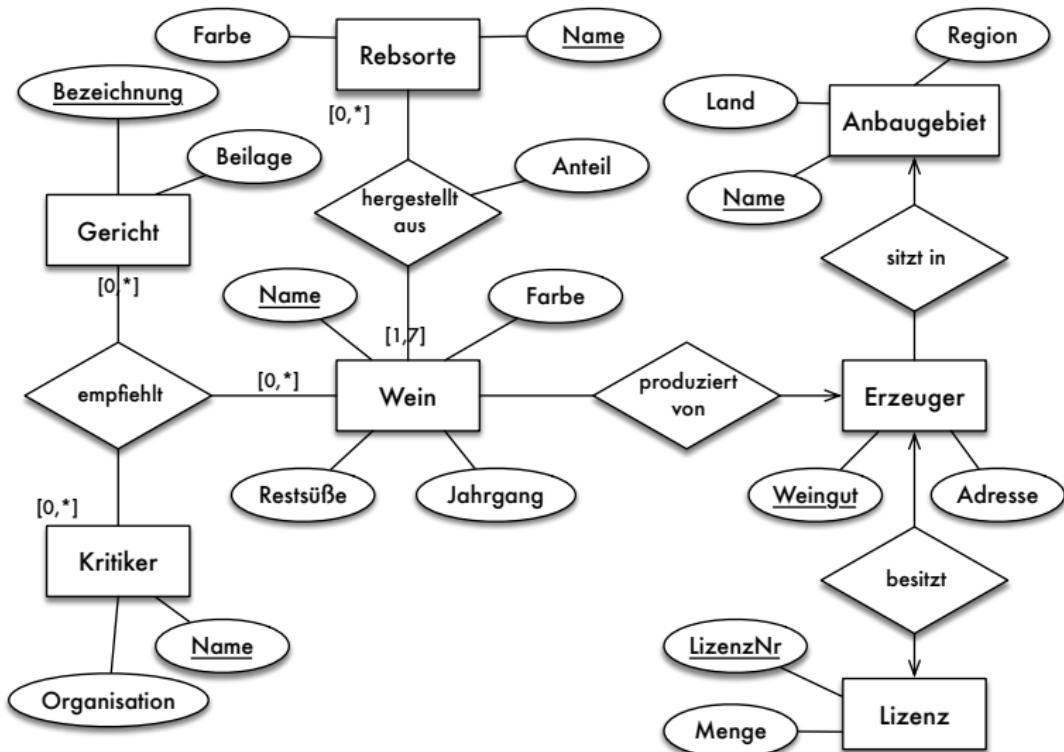
Das ER-Modell

Entity: Objekt der realen oder der Vorstellungswelt, über das Informationen zu speichern sind, z.B. **Produkte** (Wein, Katalog), Winzer oder Kritiker; aber auch Informationen über Ereignisse, wie z.B. **Bestellungen**

Relationship: beschreibt eine Beziehung zwischen Entities, z.B. ein Kunde **bestellt** einen Wein oder ein Wein wird von einem Winzer **angeboten**

Attribut: repräsentiert eine Eigenschaft von Entities oder Beziehungen, z.B. **Name** eines Kunden, **Farbe** eines Weines oder **Datum** einer Bestellung

ER-Beispiel



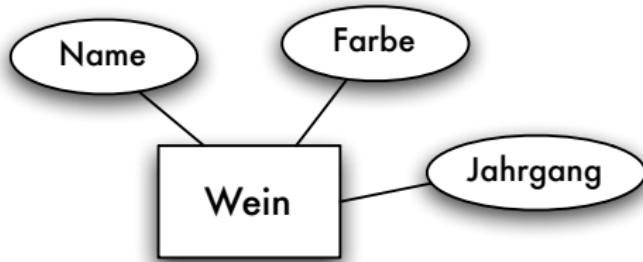
- **Werte**: primitive Datenelemente, die direkt darstellbar sind
- Wertemengen sind beschrieben durch **Datentypen**, die neben einer Wertemenge auch die Grundoperationen auf diesen Werten charakterisieren
- ER-Modell: vorgegebene Standard-Datentypen, etwa die ganzen Zahlen `int`, die Zeichenketten `string`, Datumswerte `date` etc.
- jeder Datentyp stellt Wertebereich mit Operationen und Prädikaten dar

- **Entities** sind die in einer Datenbank zu repräsentierenden Informationseinheiten
- im Gegensatz zu Werten nicht direkt darstellbar, sondern nur über ihre Eigenschaften beobachtbar
- Entities sind eingeteilt in **Entity-Typen**, etwa $E_1, E_2 \dots$

Wein

- Menge der aktuellen Entities: $\mathcal{E} = \{e_1, e_2, \dots, e_n\}$

- **Attribute** modellieren Eigenschaften von Entities oder auch Beziehungen
- alle Entities eines Entity-Typs haben dieselben Arten von Eigenschaften; Attribute werden somit für Entity-Typen deklariert



- textuelle Notation $E(A_1 : D_1, \dots, A_m : D_m)$

Identifizierung durch Schlüssel

- Schlüsselattribute: Teilmenge der gesamten Attribute eines Entity-Typs $E(A_1, \dots, A_m)$
$$\{S_1, \dots, S_k\} \subseteq \{A_1, \dots, A_m\}$$
- in jedem Datenbankzustand identifizieren die aktuellen Werte der Schlüsselattribute eindeutig Instanzen des Entity-Typs E
- bei mehreren möglichen Schlüsselkandidaten: Auswahl eines **Primärschlüssels**
- Notation: markieren durch Unterstreichung:

$$E(\dots, \underline{S_1}, \dots, \underline{S_i}, \dots)$$

Beziehungstypen

- Beziehungen zwischen Entities werden zu **Beziehungstypen** zusammengefasst
- allgemein: beliebige Anzahl $n \geq 2$ von Entity-Typen kann an einem Beziehungstyp teilhaben
- zu jedem n -stelligen Beziehungstyp R gehören n Entity-Typen E_1, \dots, E_n
- Ausprägung \mathcal{R} eines Beziehungstyps

$$\mathcal{R} \subseteq \mathcal{E}_1 \times \mathcal{E}_2 \times \cdots \times \mathcal{E}_n$$

- Notation

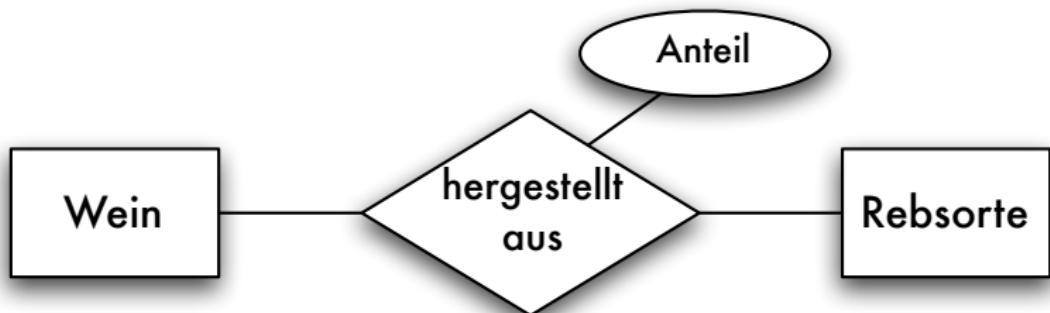


- textuelle Notation: $R(E_1, E_2, \dots, E_n)$
- wenn Entity-Typ mehrfach an einem Beziehungstyp beteiligt:
Vergabe von **Rollennamen** möglich

verheiratet(Frau: Person, Mann: Person)

Beziehungsattribute

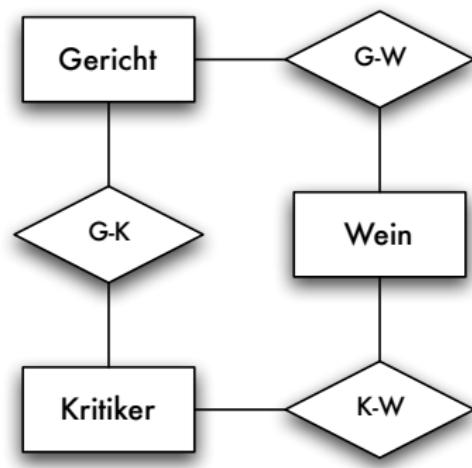
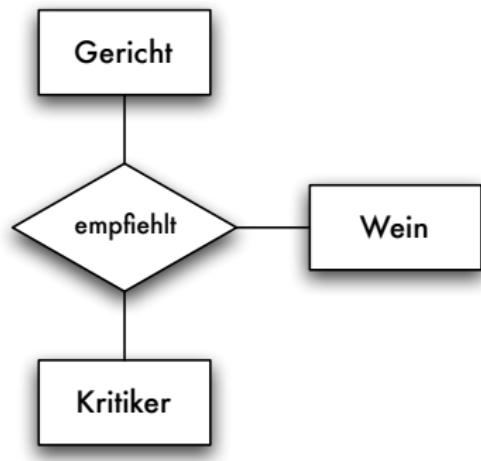
- Beziehungen können ebenfalls Attribute besitzen
- Attributdeklarationen werden beim Beziehungstyp vorgenommen; gilt auch hier für alle Ausprägungen eines Beziehungstyps \leadsto
Beziehungsattribute



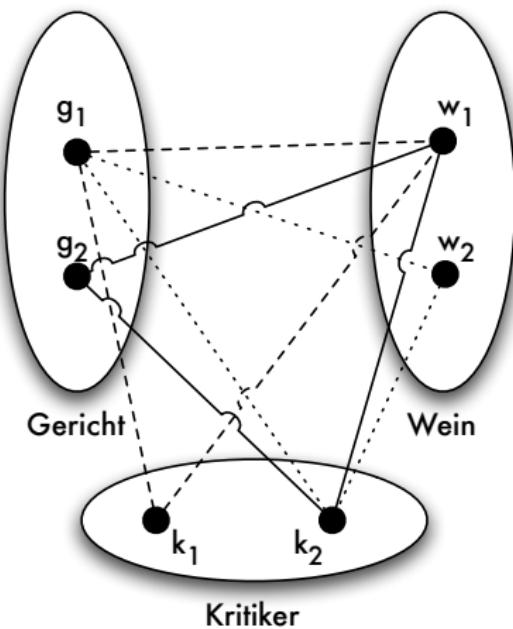
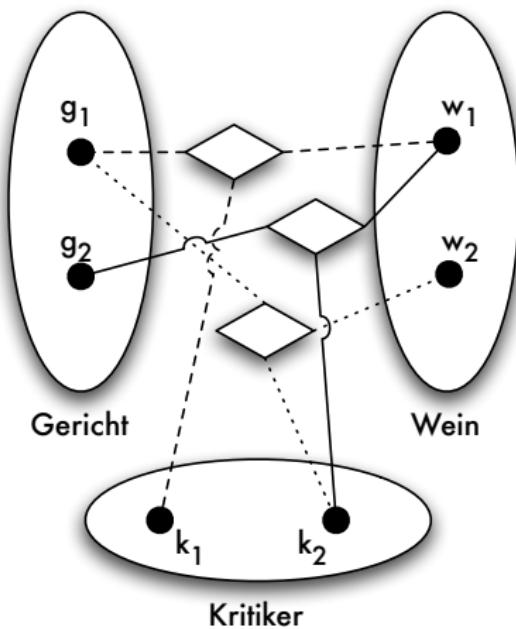
- textuelle Notation: $R(E_1, \dots, E_n; A_1, \dots, A_k)$

- **Stelligkeit** oder Grad:
 - Anzahl der beteiligten Entity-Typen
 - häufig: binär
 - Beispiel: *Lieferant liefert Produkt*
- **Kardinalität** oder Funktionalität:
 - Anzahl der eingehenden Instanzen eines Entity-Typs
 - Formen: 1:1, 1:n, m:n
 - stellt Integritätsbedingung dar
 - Beispiel: **maximal 5 Produkte pro Bestellung**

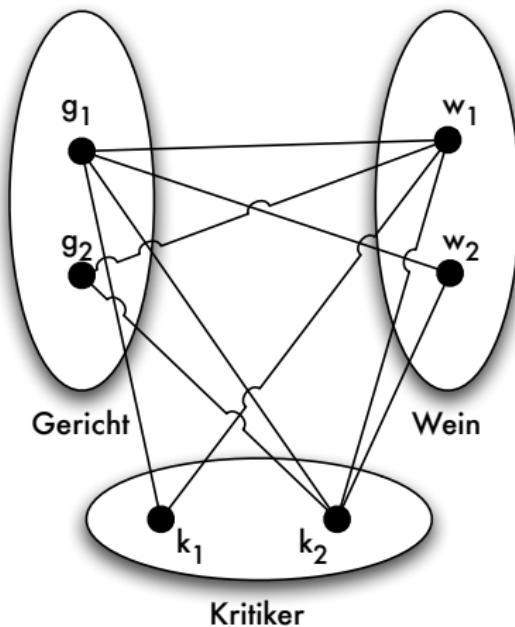
Zwei- vs. mehrstellige Beziehungen



Ausprägungen im Beispiel



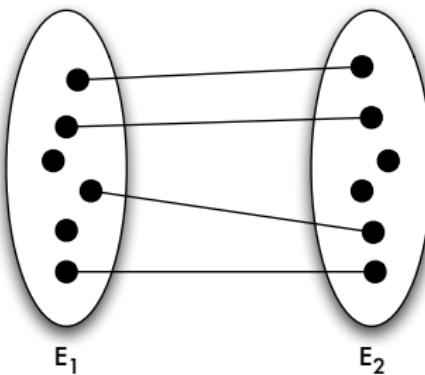
Rekonstruktion der Ausprägungen



- $g_1 - k_1 - w_1$
- $g_1 - k_2 - w_2$
- $g_2 - k_2 - w_1$
- aber auch: $g_1 - k_2 - w_1$

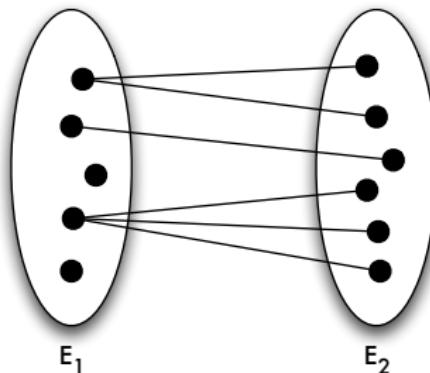
1:1-Beziehungen

- jedem Entity e_1 vom Entity-Typ E_1 ist maximal ein Entity e_2 aus E_2 zugeordnet und umgekehrt
- Beispiele: *Prospekt beschreibt Produkt, Mann ist verheiratet mit Frau*



1:N-Beziehungen

- jedem Entity e_1 vom Entity-Typ E_1 sind beliebig viele Entities E_2 zugeordnet, aber zu jedem Entity e_2 gibt es maximal ein e_1 aus E_1
- Beispiele: *Lieferant liefert Produkt, Mutter hat Kinder*



N:1-Beziehung

- invers zu 1:N, auch **funktionale** Beziehung
- zweistellige Beziehungen, die eine **Funktion** beschreiben:

Jedem Entity eines Entity-Typs E_1 wird maximal ein Entity eines Entity-Typs E_2 zugeordnet.

$$R : E_1 \rightarrow E_2$$

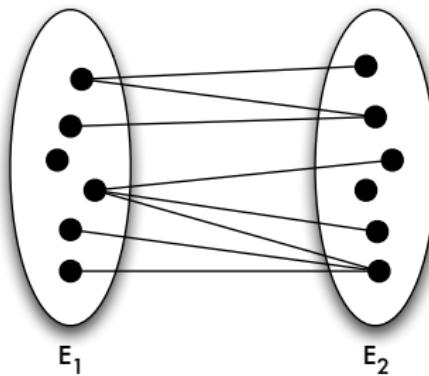


1:1-Beziehung

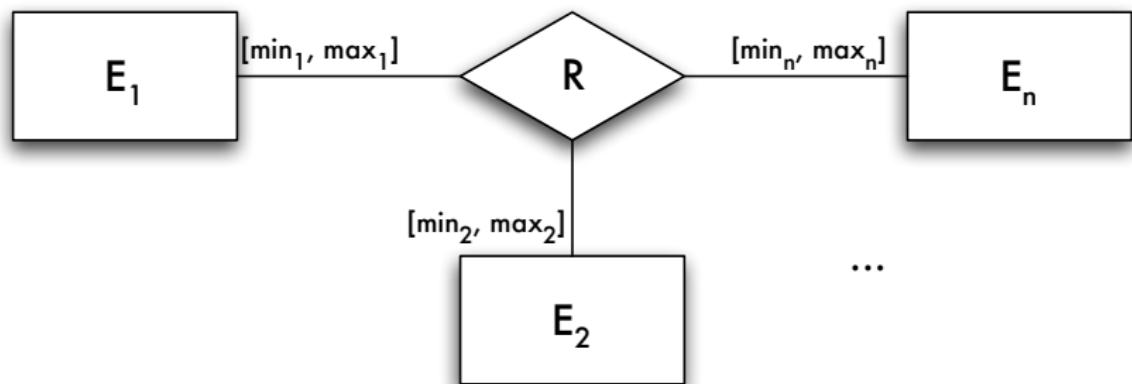


M:N-Beziehungen

- keine Restriktionen
- Beispiel: *Bestellung umfasst Produkte*



[min,max]-Notation



- schränkt die möglichen **Teilnahmen** von Instanzen der beteiligten Entity-Typen an der Beziehung ein, indem ein minimaler und ein maximaler Wert vorgegeben wird

[min,max]-Notation /2

- Notation für Kardinalitätsangaben an einem Beziehungstyp

$$R(E_1, \dots, E_i[min_i, max_i], \dots, E_n)$$

- Kardinalitätsbedingung: $min_i \leq |\{r \mid r \in R \wedge r.E_i = e_i\}| \leq max_i$
- Spezielle Wertangabe für max_i ist *

Kardinalitätsangaben

- $[0, *]$ legt keine Einschränkung fest (default)
- $R(E_1[0, 1], E_2)$ entspricht einer (partiellen) funktionalen Beziehung $R : E_1 \rightarrow E_2$, da jede Instanz aus E_1 maximal einer Instanz aus E_2 zugeordnet ist
- totale funktionale Beziehung wird durch $R(E_1[1, 1], E_2)$ modelliert

Kardinalitätsangaben: Beispiele

- partielle funktionale Beziehung

```
lagert_in(Produkt[0,1],Fach[0,3])
```

„Jedes Produkt ist im Lager in einem Fach abgelegt, allerdings wird ausverkauften bzw. gegenwärtig nicht lieferbaren Produkte kein Fach zugeordnet. Pro Fach können maximal drei Produkte gelagert werden.“

- totale funktionale Beziehung

```
liefert(Lieferant[0,*],Produkt[1,1])
```

„Jedes Produkt wird durch genau einen Lieferant geliefert, aber ein Lieferant kann durchaus mehrere Produkte liefern.“

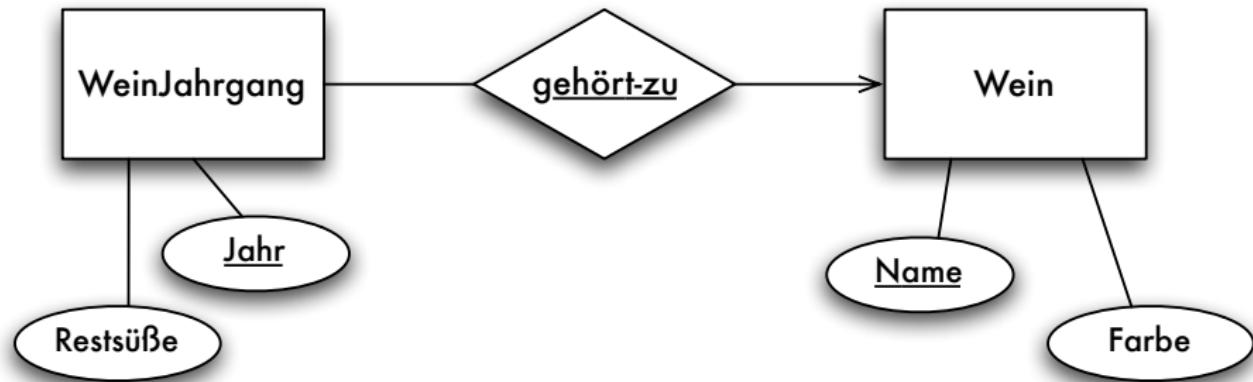
Alternative Kardinalitätsangabe



Weitere Konzepte im ER-Modell

Abhängige Entity-Typen

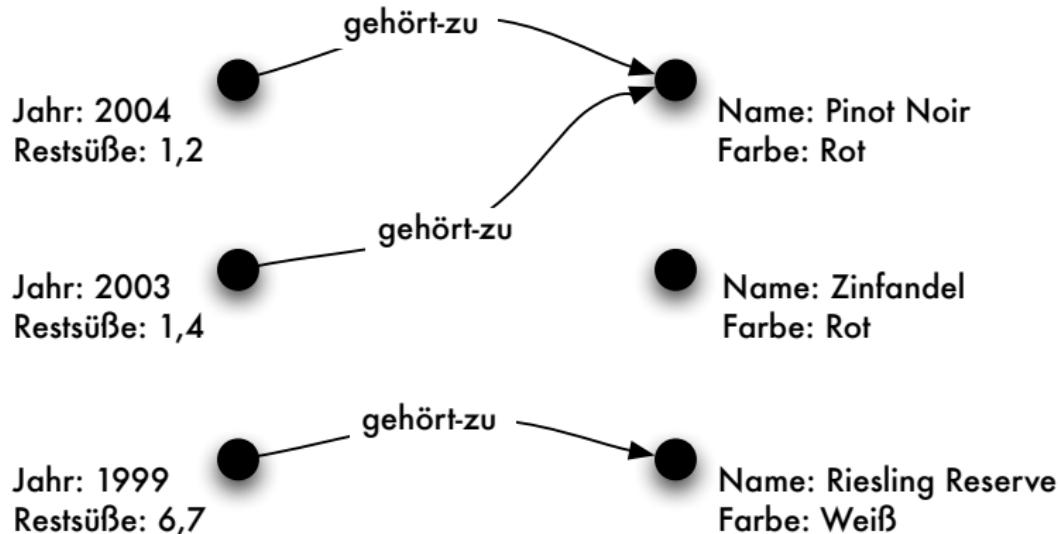
- *abhängiger Entity-Typ*: Identifikation über funktionale Beziehung



- Abhängige Entities im ER-Modell: Funktionale Beziehung als Schlüssel

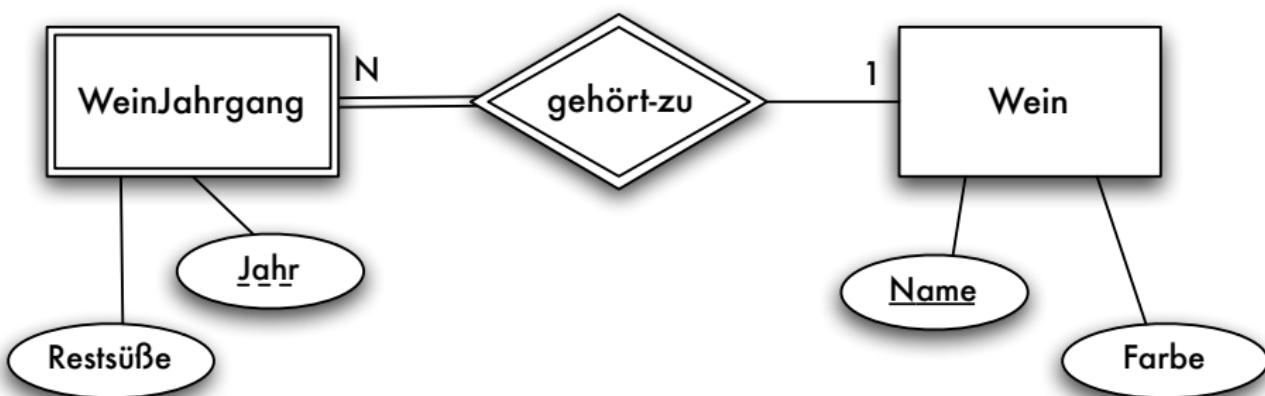
Abhängige Entity-Typen /2

- Mögliche Ausprägung für abhängige Entities



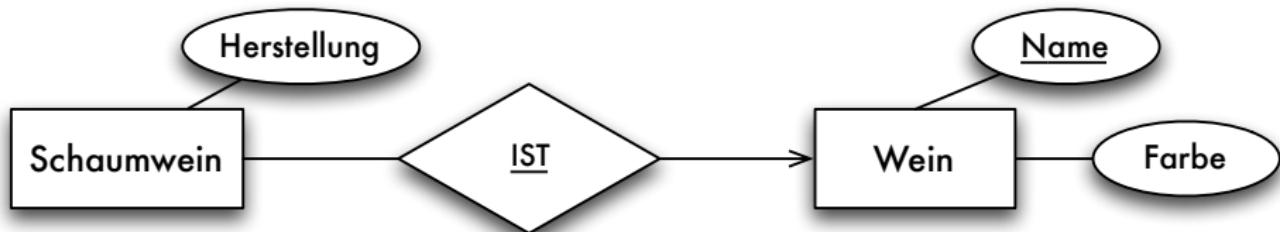
Abhängige Entity-Typen /3

- Alternative Notation



Die IST-Beziehung

- **Spezialisierungs-/Generalisierungsbeziehung** oder auch IST-Beziehung (engl. *is-a relationship*)
- textuelle Notation: E_1 IST E_2
- IST-Beziehung entspricht semantisch einer **injektiven** funktionalen Beziehung

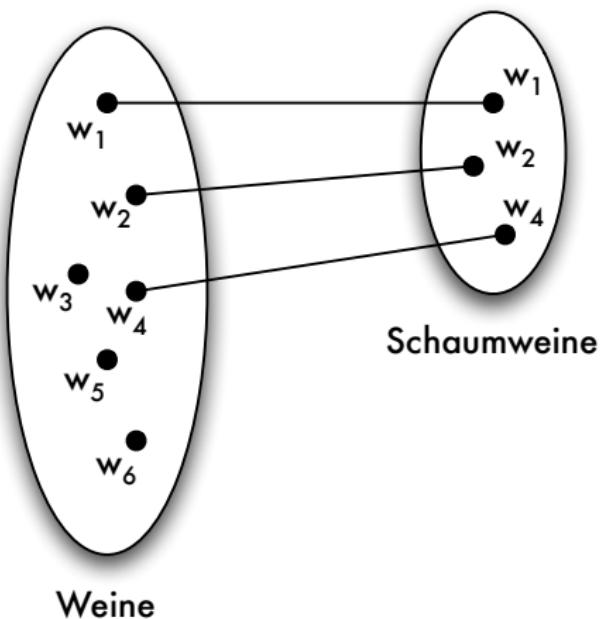


- Jeder Schaumwein-Instanz ist genau eine Wein-Instanz zugeordnet
~~> Schaumwein-Instanzen werden durch die funktionale IST-Beziehung identifiziert
- Nicht jeder Wein ist zugleich ein Schaumwein
- Attribute des Entity-Typs Wein treffen auch auf Schaumweine zu:
„vererbte“ Attribute

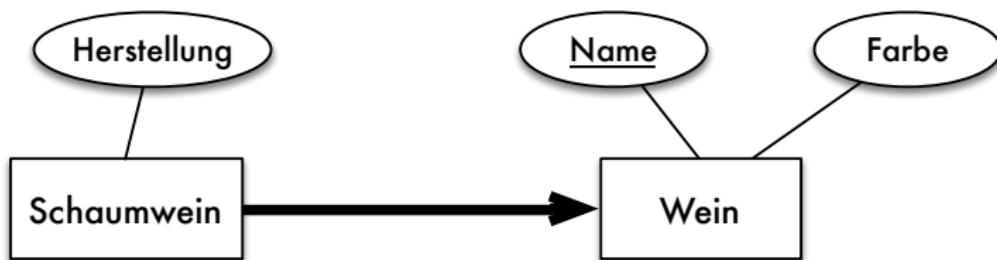
Schaumwein(Name, Farbe, Herstellung)
von Wein

- nicht nur die Attributdeklarationen vererben sich, sondern auch jeweils die aktuellen Werte für eine Instanz

Ausprägung für IST-Beziehung



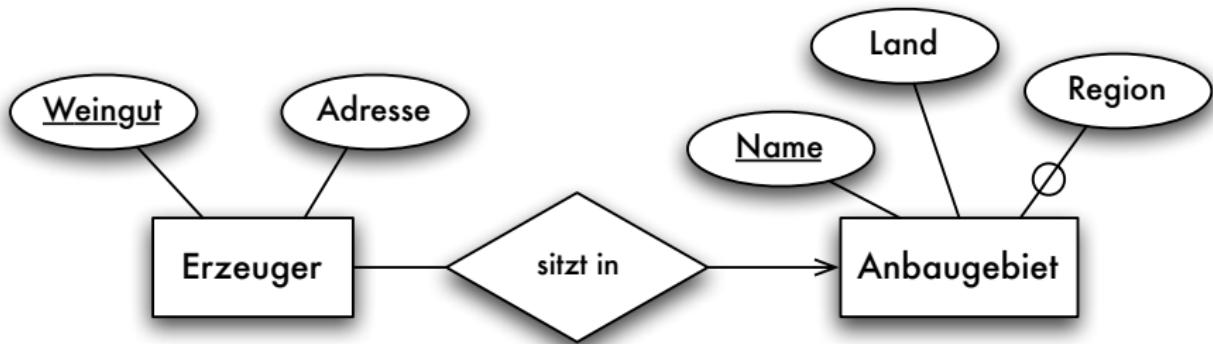
Alternative Notation für IST-Beziehung



Kardinalitätsangaben: IST

- für Beziehung E_1 ist E_2 gilt immer: $\text{IST}(E_1[1,1], E_2[0,1])$
- Jede Instanz von E_1 nimmt genau einmal an der IST -Beziehung teil, während Instanzen des Obertyps E_2 nicht teilnehmen müssen
- Aspekte wie Attributvererbung werden hiervon nicht erfasst

Optionalität von Attributen



Konzepte im Überblick

Begriff	Informale Bedeutung
Entity	zu repräsentierende Informationseinheit
Entity-Typ	Gruppierung von Entitys mit gleichen Eigenschaften
Beziehungstyp	Gruppierung von Beziehungen zwischen Entitys
Attribut	datenwertige Eigenschaft eines Entitys oder einer Beziehung
Schlüssel	identifizierende Eigenschaft von Entitys
Kardinalitäten	Einschränkung von Beziehungstypen bezüglich der mehrfachen Teilnahme von Entitys an der Beziehung
Stelligkeit	Anzahl der an einem Beziehungstyp beteiligten Entity-Typen
funktionale Beziehung	Beziehungstyp mit Funktionseigenschaft

Konzepte im Überblick /2

Begriff	Informale Bedeutung
abhängige Entitys	Entitys, die nur abhängig von anderen Entitys existieren können
IST-Beziehung	Spezialisierung von Entity-Typen
Optionalität	Attribute oder funktionale Beziehungen als partielle Funktionen

Zusammenfassung

- Datenbankmodell, Datenbankschema, Datenbank(instanz)
- Entity-Relationship-Modell
- Weitere Konzepte im ER-Modell
- *Basis: Kapitel 3 von [SSH13]*

Kontrollfragen

- Was definiert ein Datenbankmodell? Was unterscheidet Modell und Schema?
- Welche Konzepte definiert das ER-Modell?
- Durch welche Eigenschaften sind Beziehungstypen charakterisiert?
- Was unterscheidet abhängige Entity-Typen von normalen Entity-Typen?



Teil IV

Datenbankentwurf

1. Phasen des Datenbankentwurfs
2. Weiteres Vorgehen beim Entwurf
3. Kapazitätserhaltende Abbildungen
4. ER-auf-RM-Abbildung

Lernziele für heute . . .

- Kenntnisse über Ziele und Ablauf des Datenbankentwurfsprozesses
- Prinzip der Kapazitätserhaltung
- Kenntnisse der Regeln zur Abbildung von ER-Schemata auf Relationenschemata

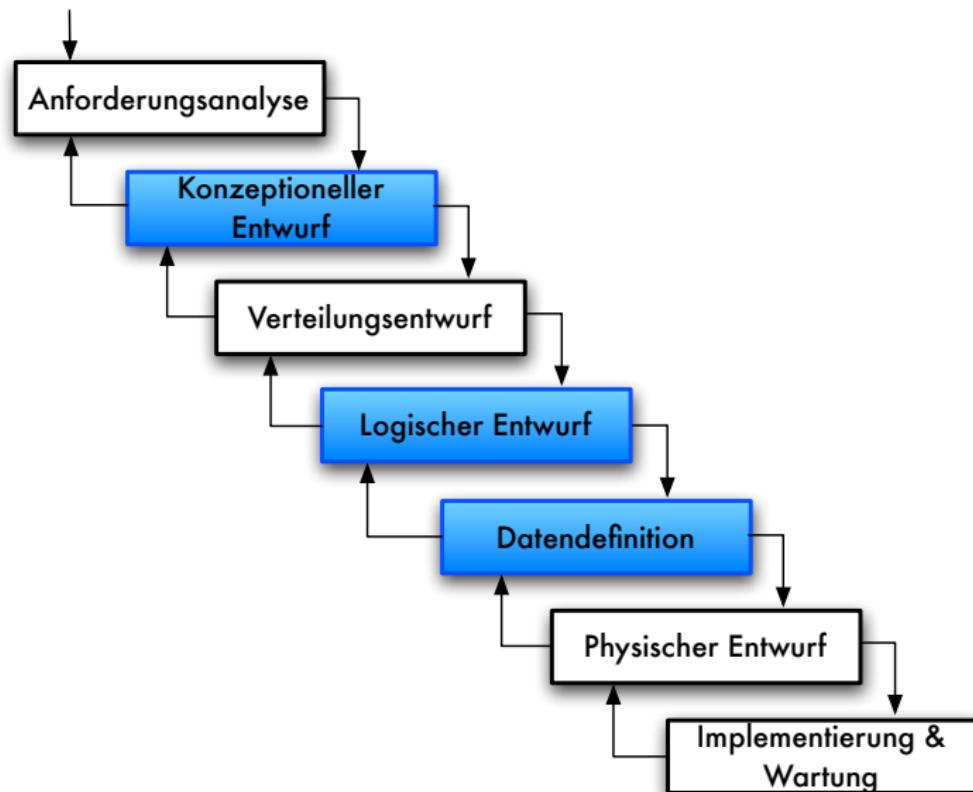


Phasen des Datenbankentwurfs

Entwurfsaufgabe

- Datenhaltung für mehrere Anwendungssysteme und mehrere Jahre
- daher: besondere Bedeutung
- Anforderungen an Entwurf
 - Anwendungsdaten jeder Anwendung sollen aus Daten der Datenbank ableitbar sein (und zwar möglichst effizient)
 - nur „vernünftige“ (wirklich benötigte) Daten sollen gespeichert werden
 - nicht-redundante Speicherung

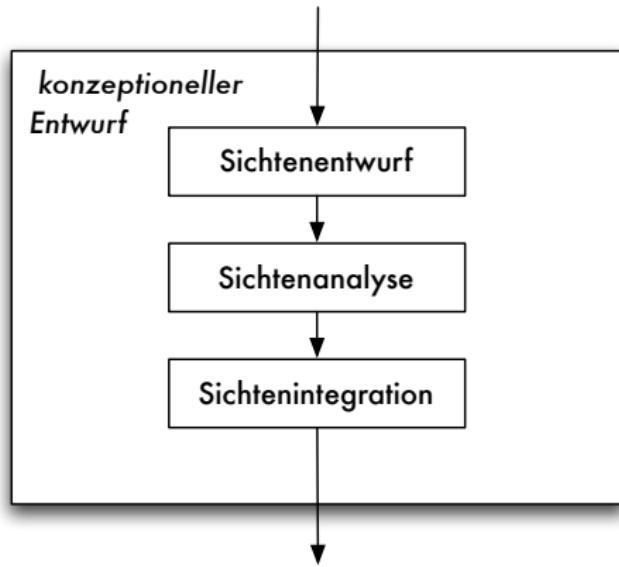
Phasenmodell



- **Vorgehensweise:** Sammlung des Informationsbedarfs in den Fachabteilungen
- **Ergebnis:**
 - informale Beschreibung (Texte, tabellarische Aufstellungen, Formblätter, usw.) des Fachproblems
 - Trennen der Information über Daten (Datenanalyse) von der Information über Funktionen (Funktionsanalyse)
- „Klassischer“ DB-Entwurf:
 - nur Datenanalyse und Folgeschritte
- **Funktionsentwurf:**
 - siehe Methoden des Software Engineering

- erste formale Beschreibung des Fachproblems
- **Sprachmittel:** semantisches Datenmodell
- **Vorgehensweise:**
 - Modellierung von Sichten z.B. für verschiedene Fachabteilungen
 - Analyse der vorliegenden Sichten in Bezug auf Konflikte
 - Integration der Sichten in ein Gesamtschema
- **Ergebnis:** konzeptionelles Gesamtschema, z.B. ER-Diagramm

Phasen des konzeptionellen Entwurf

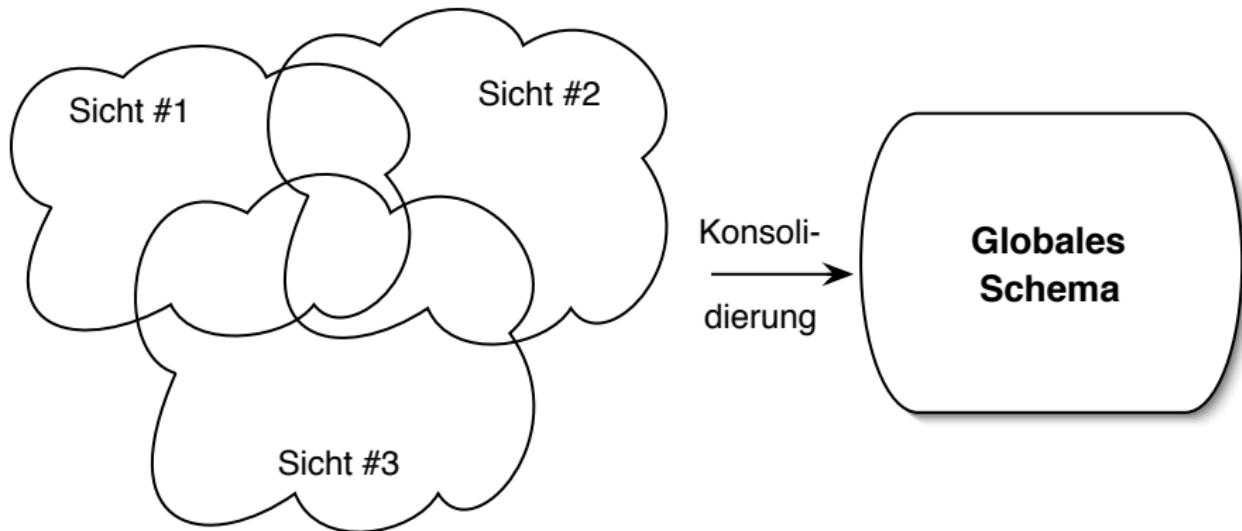


Weiteres Vorgehen beim Entwurf

- ER-Modellierung von verschiedenen **Sichten** auf Gesamtinformation, z.B. für verschiedene Fachabteilungen eines Unternehmens \rightsquigarrow **konzeptueller Entwurf**
 - Analyse und Integration der Sichten
 - Ergebnis: konzeptionelles Gesamtschema
- Verteilungsentwurf bei verteilter Speicherung
- Abbildung auf konkretes Implementierungsmodell (z.B. Relationenmodell) \rightsquigarrow **logischer Entwurf**
- Datendefinition, Implementierung und Wartung \rightsquigarrow **physischer Entwurf**

Sichtenintegration

- Analyse der vorliegenden Sichten in Bezug auf Konflikte
- Integration der Sichten in ein Gesamtschema



- **Namenskonflikte:** Homonyme / Synonyme
 - Homonyme: Schloss; Kunde
 - Synonyme: Auto, KFZ, Fahrzeug
- **Typkonflikte:** verschiedene Strukturen für das gleiche Element
- **Wertebereichskonflikte:** verschiedene Wertebereiche für ein Element
- **Bedingungskonflikte:** z.B. verschiedene Schlüssel für ein Element
- **Strukturkonflikte:** gleicher Sachverhalt durch unterschiedliche Konstrukte ausgedrückt

- sollen Daten auf mehreren Rechnern verteilt vorliegen, muss Art und Weise der **verteilten Speicherung** festgelegt werden
- z.B. bei einer Relation
KUNDE (KNr, Name, Adresse, PLZ, Konto)
 - **horizontale** Verteilung:
KUNDE_1 (KNr, Name, Adresse, PLZ, Konto)
where PLZ < 50.000
KUNDE_2 (KNr, Name, Adresse, PLZ, Konto)
where PLZ >= 50.000
 - **vertikale** Verteilung (Verbindung über KNr Attribut):
KUNDE_Adr (KNr, Name, Adresse, PLZ)
KUNDE_Konto (KNr, Konto)

- **Sprachmittel:** Datenmodell des ausgewählten „Realisierungs“-DBMS z.B. relationales Modell
- **Vorgehensweise:**
 1. (automatische) Transformation des konzeptionellen Schemas z.B. ER → relationales Modell
 2. Verbesserung des relationalen Schemas anhand von Gütekriterien (Normalisierung, siehe Kapitel 5):
Entwurfsziele: Redundanzvermeidung, ...
- **Ergebnis:** logisches Schema, z.B. Sammlung von Relationenschemata

- Umsetzung des logischen Schemas in ein konkretes Schema
- **Sprachmittel:** DDL und DML eines DBMS z.B. Oracle, DB2, SQL Server
 - Datenbankdeklaration in der DDL des DBMS
 - Realisierung der Integritätssicherung
 - **Definition der Benutzersichten**

- Ergänzen des physischen Entwurfs um Zugriffsunterstützung bzgl. Effizienzverbesserung, z.B. Definition von Indexen
- Index
 - Zugriffspfad: Datenstruktur für zusätzlichen, schlüsselbasierten Zugriff auf Tupel ($\langle \text{Schlüsselattributwert}, \text{Tupeladresse} \rangle$)
 - meist als B*-Baum realisiert
- **Sprachmittel:** Speicherstruktursprache SSL

Indexe in SQL

```
create [ unique ] index indexname
    on relname (
        attrname [ asc | desc ],
        attrname [ asc | desc ],
        ...
    )
```

- Beispiel

```
create index WeinIdx on WEINE (Name)
```

Notwendigkeit für Zugriffspfade

- Beispiel: Tabelle mit 100 GB Daten, Festplattentransferrate ca. 50 MB/s
- Operation: Suchen eines Tupels (Selektion)
- Implementierung: sequentielles Durchsuchen
- Aufwand: $102.400 / 50 = 2.048 \text{ sec.} \approx 34 \text{ min.}$

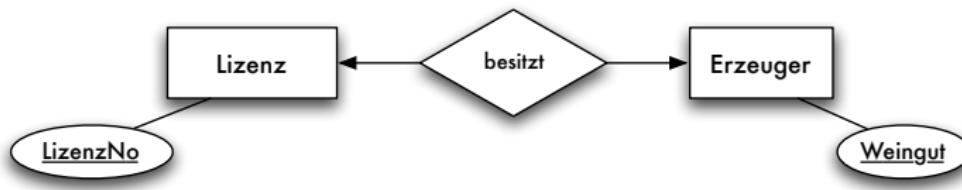
- Phasen
 - der Wartung,
 - der weiteren Optimierung der physischen Ebene,
 - der Anpassung an neue Anforderungen und Systemplattformen,
 - der Portierung auf neue Datenbankmanagementsysteme
 - etc.

Kapazitätserhaltende Abbildungen

Umsetzung des konzeptionellen Schemas

- Umsetzung auf logisches Schema
 - Beispiel: ER → RM
 - korrekt?
 - Qualität der Abbildung?
- Erhaltung der **Informationskapazität**
 - Kann man nach der Abbildung genau die selben Daten abspeichern wie vorher?
 - ... oder etwa mehr?
 - ... oder etwa weniger?

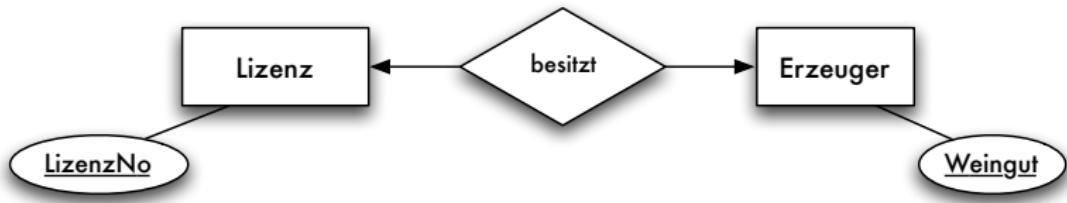
Kapazitätserhöhende Abbildung



- Abbildung auf $R = \{\text{LizenzNo}, \text{Weingut}\}$ mit genau einem Schlüssel $K = \{\{\text{LizenzNo}\}\}$
- mögliche ungültige Relation:

BESITZT	LizenzNo	Weingut
007		Helena
42		Helena

Kapazitätserhaltende Abbildung



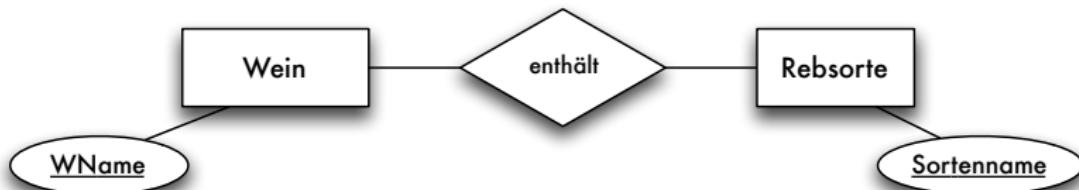
- korrekte Ausprägung

BESITZT	LizenzNo	Weingut
	007	Helena
	42	Müller

- korrekte Schlüsselmenge

$$K = \{\{LizenzNo\}, \{Weingut\}\}$$

Kapazitätsvermindernde Abbildung



- Relationenschema mit einem Schlüssel {WName}
- als Ausprägung nicht mehr möglich:

ENTHÄLT	WName	Sortenname
	Zinfandel Red Blossom	Zinfandel
	Bordeaux Blanc	Cabernet Sauvignon
	Bordeaux Blanc	Muscadelle

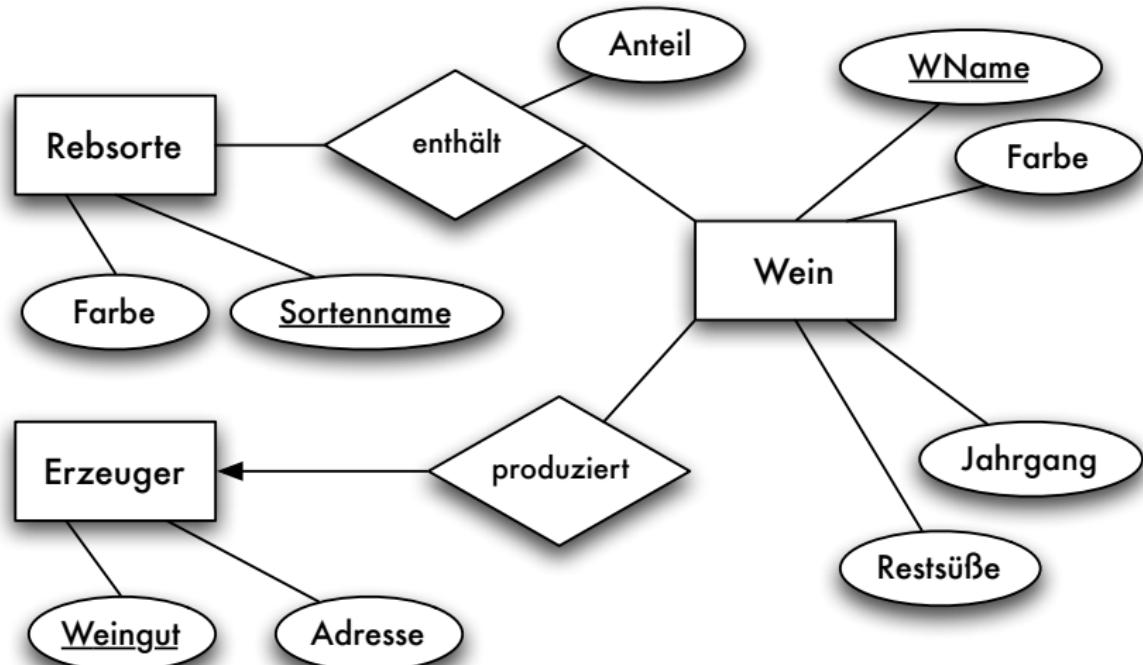
Kapazitätserhaltende Abbildung

- kapazitätserhaltend mit Schlüssel beider Entity-Typen im Relationenschema als neuer Schlüssel

$$K = \{\{\text{WName}, \text{Sortenname}\}\}$$

ER-auf-RM-Abbildung

Beispielabbildung ER-RM: Eingabe

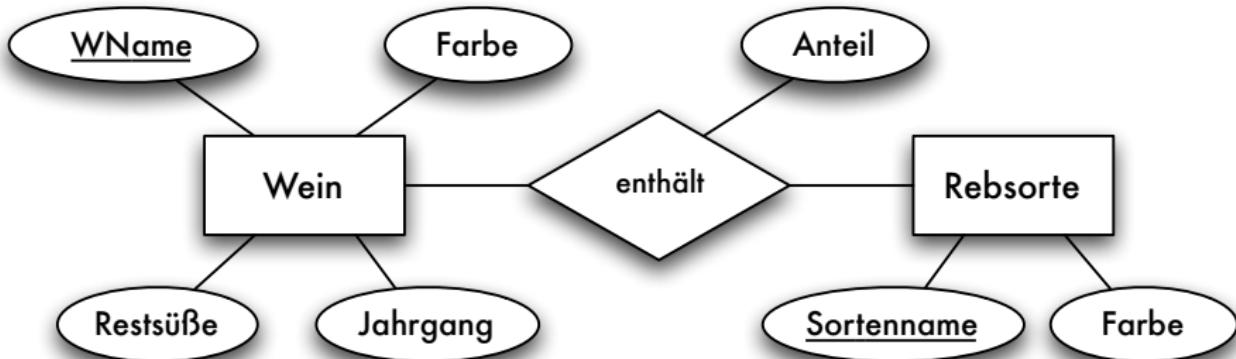


Beispielabbildung ER-RM: Ergebnis

1. REBSORTE(Farbe, Sortenname)
2. ENTHÄLT(Sortenname → REBSORTE, WName → WEIN, Anteil)
3. WEIN(Farbe, WName, Jahrgang, Restsüße)
4. PRODUZIERT(WName → WEIN, Weingut → ERZEUGER)
5. ERZEUGER(Weingut, Adresse)

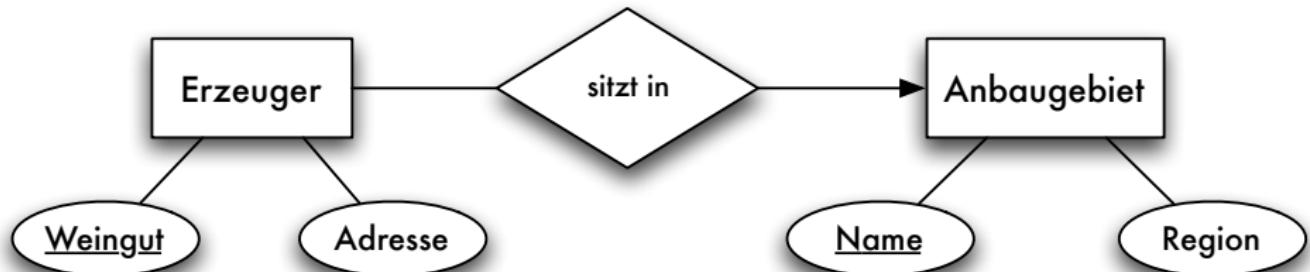
- **Entity-Typen und Beziehungstypen:** jeweils auf Relationenschemata
- **Attribute:** Attribute des Relationenschemas, **Schlüssel** werden übernommen
- **Kardinalitäten** der Beziehungen: durch Wahl der Schlüssel bei den zugehörigen Relationenschemata ausgedrückt
- in einigen Fällen: **Verschmelzen** der Relationenschemata von Entity- und Beziehungstypen
- zwischen den verbleibenden Relationenschemata diverse **Fremdschlüsselbedingungen** einführen

- neues Relationenschema mit allen Attributen des Beziehungstyps, zusätzlich Übernahme aller Primärschlüssel der beteiligten Entity-Typen
- **Festlegung der Schlüssel:**
 - **m:n-Beziehung:** beide Primärschlüssel zusammen werden Schlüssel im neuen Relationenschema
 - **1:n-Beziehung:** Primärschlüssel der n-Seite (bei der funktionalen Notation die Seite ohne Pfeilspitze) wird Schlüssel im neuen Relationenschema
 - **1:1-Beziehung:** beide Primärschlüssel werden je ein Schlüssel im neuen Relationenschema, der Primärschlüssel wird dann aus diesen Schlüsseln gewählt



- Umsetzung
 1. REBSORTE(Farbe, Sortenname)
 2. ENTHÄLT(Sortenname → REBSORTE, WName → WEIN, Anteil)
 3. WEIN(Farbe, WName, Jahrgang, Restsüße)
- Attribute Sortenname und WName sind gemeinsam Schlüssel

1:n-Beziehungen

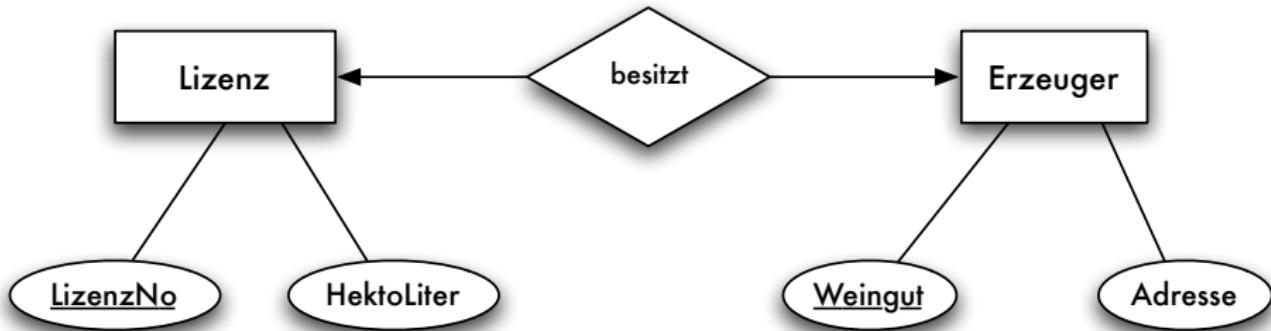


- Umsetzung (zunächst)
 - ERZEUGER mit den Attributen Weingut und Adresse,
 - ANBAUGEBIET mit den Attributen Name und Region und
 - SITZT_IN mit den Attributen Weingut und Name und dem Primärschlüssel der n-Seite Weingut als Primärschlüssel dieses Schemas.

Mögliche Verschmelzungen

- **optionale Beziehungen** ($[0,1]$ oder $[0,n]$) werden nicht verschmolzen
- bei Kardinalitäten $[1,1]$ oder $[1,n]$ (**zwingende Beziehungen**) Verschmelzung möglich:
 - **1:n-Beziehung**: das Entity-Relationenschema der n-Seite kann in das Relationenschema der Beziehung integriert werden
 - **1:1-Beziehung**: beide Entity-Relationenschemata können in das Relationenschema der Beziehung integriert werden

1:1-Beziehungen



- Umsetzung (zunächst)
 - ERZEUGER mit den Attributen Weingut und Adresse
 - LIZENZ mit den beiden Attributen LizenzNo und Hektoliter
 - BESITZT mit den Primärschlüsseln der beiden beteiligten Entity-Typen jeweils als Schlüssel dieses Schemas, also LizenzNo und Weingut

1:1-Beziehungen: Verschmelzung

ERZEUGER	Weingut	Adresse	LizenzNo	Hektoliter
	Rotkäppchen	Freiberg	42-007	10.000
	Weingut Müller	Dagstuhl	42-009	250

1:1-Beziehungen: Verschmelzung /2

Erzeuger ohne Lizenz erfordern Nullwerte:

ERZEUGER	Weingut	Adresse	LizenzNo	Hektoliter
	Rotkäppchen Weingut Müller	Freiberg Dagstuhl	42-007 ⊥	10.000 ⊥

freie Lizenzen führen zu weiteren Nullwerten:

ERZEUGER	Weingut	Adresse	LizenzNo	Hektoliter
	Rotkäppchen Weingut Müller ⊥	Freiberg Dagstuhl ⊥	42-007 ⊥ 42-003	10.000 ⊥ 100.000

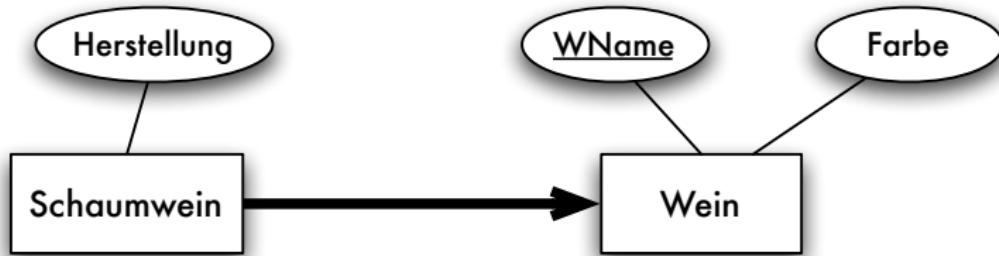
Abhängige Entity-Typen



- Umsetzung

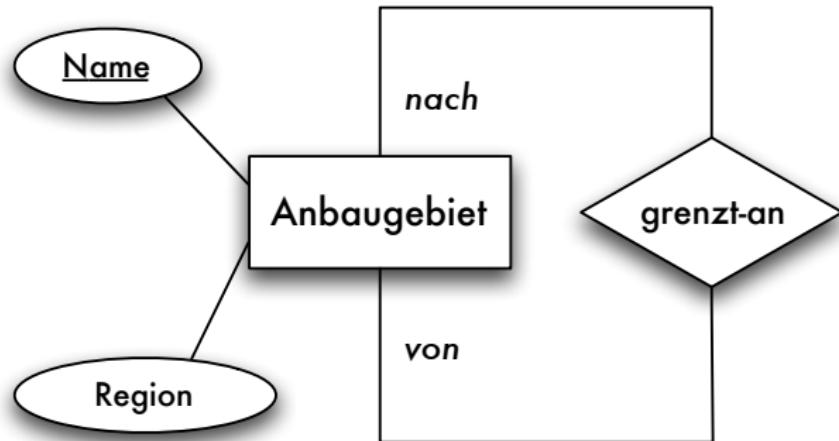
- WEINJAHRGANG(WName → WEIN, Jahr, Restsüße)
- WEIN(Farbe, WName)

- Attribut **WName** in **WEINJAHRGANG** ist Fremdschlüssel zur Relation **WEIN**



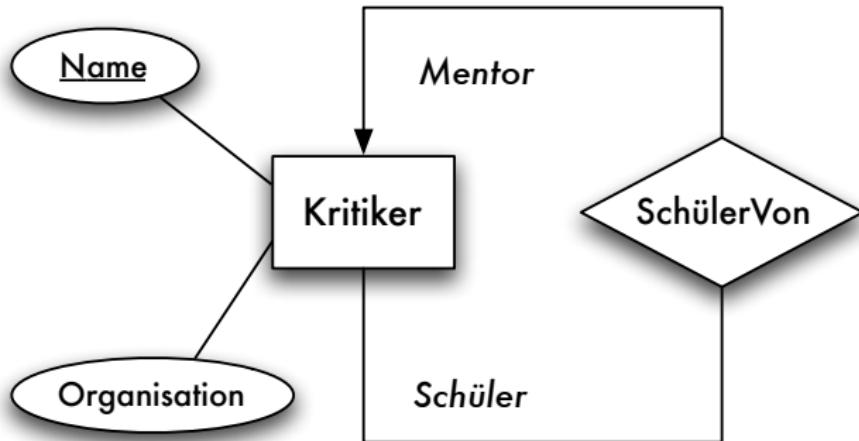
- Umsetzung
 1. WEIN(Farbe, WName)
 2. SCHAUMWEIN(WName → WEIN, Herstellung)
 - WName in SCHAUMWEIN ist Fremdschlüssel bezüglich der Relation WEIN

Rekursive Beziehungen



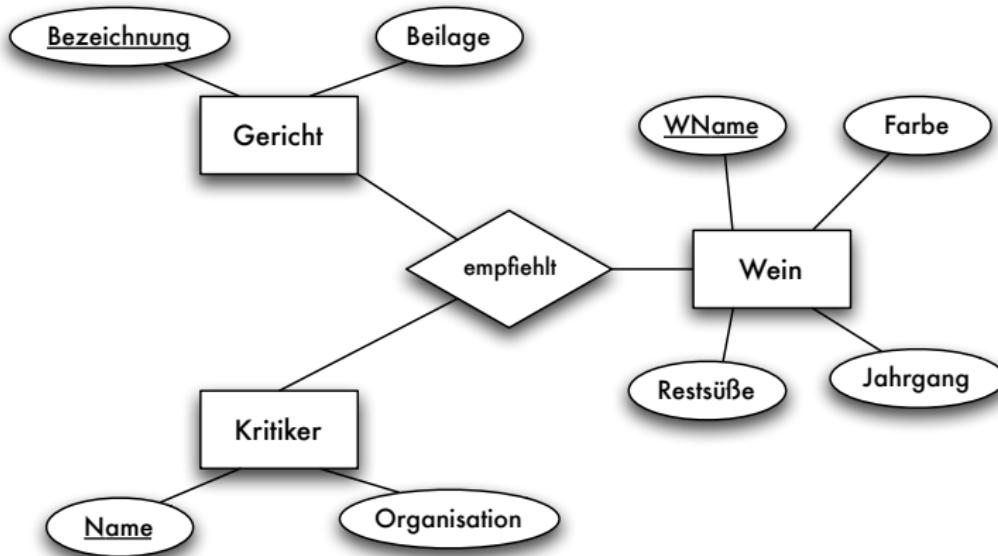
- Umsetzung
 - 1. **ANBAUGEBIET**(Name, Region)
 - 2. **GRENZT_AN**(nach → **ANBAUGEBIET**, von → **ANBAUGEBIET**)

Rekursive funktionale Beziehungen



- Umsetzung
 1. KRITIKER(Name, Organisation, Mentorname → KRITIKER)
 - Mentorname ist Fremdschlüssel auf das Attribut Name der Relation KRITIKER.

Mehrstellige Beziehungen



Mehrstellige Beziehungen: Ergebnis

- jeder beteiligte Entity-Typ wird nach den obigen Regeln behandelt
- für Beziehung **Empfiehlt** werden Primärschlüssel der drei beteiligten Entity-Typen in das resultierende Relationenschema aufgenommen
- Beziehung ist allgemeiner Art (k:m:n-Beziehung): alle Primärschlüssel bilden zusammen den Schlüssel
 1. EMPFIEHLT(WName → WEIN,
Bezeichnung → GERICHT, Name → KRITIKER)
 2. GERICHT(Bezeichnung, Beilage)
 3. WEIN(Farbe, WName, Jahrgang, Restsüße)
 4. KRITIKER(Name, Organisation)
- Die drei Schlüsselattribute von EMPFIEHLT sind wiederum Fremdschlüssel

Übersicht über die Transformationen

ER-Konzept	wird abgebildet auf relationales Konzept
Entity-Typ E_i Attribute von E_i Primärschlüssel P_i	Relationenschema R_i Attribute von R_i Primärschlüssel P_i
Beziehungstyp dessen Attribute $1 : n$ $1 : 1$ $m : n$	Relationenschema Attribute: P_1, P_2 weitere Attribute P_2 wird Primärschlüssel der Beziehung P_1 und P_2 werden Schlüssel der Beziehung $P_1 \cup P_2$ wird Primärschlüssel der Beziehung
IST-Beziehung	R_1 erhält zusätzlichen Schlüssel P_2

E_1, E_2 : an Beziehung beteiligte Entity-Typen,

P_1, P_2 : deren Primärschlüssel,

$1 : n$ -Beziehung: E_2 ist n -Seite,

IST-Beziehung: E_1 ist speziellerer Entity-Typ

Zusammenfassung

- Phasen des Datenbankentwurfs
- Informationskapazität
- Abbildung vom Entity-Relationship-Modell auf das Relationale Modell

Kontrollfragen

- Welche Schritte umfasst der Datenbankentwurfsprozess?
- Welche Forderungen müssen die Abbildungen (Transformationen) zwischen den einzelnen Entwurfsschritten erfüllen? Warum?
- Wie werden die Konzepte des ER-Modells auf die des Relationenmodell abgebildet?
- Wie werden die verschiedenen Kardinalitäten von Beziehungstypen bei der Abbildung berücksichtigt?



Teil V

Relationaler Entwurf

Relationaler Entwurf

1. Zielmodell des logischen Entwurfs
2. Relationaler DB-Entwurf
3. Normalformen
4. Transformationseigenschaften
5. Weitere Abhängigkeiten

Lernziele für heute . . .

- Kenntnisse zur Verfeinerung des relationalen Entwurfs
- Verständnis der Normalformen



Zielmodell des logischen Entwurfs

Relationenmodell

WEINE	WeinID	Name	Farbe	Jahrgang	Weingut
	1042	La Rose ...	Rot	1998	Château ...
	2168	Creek Shiraz	Rot	2003	Creek
	3456	Zinfandel	Rot	2004	Helena
	2171	Pinot Noir	Rot	2001	Creek
	3478	Pinot Noir	Rot	1999	Helena
	4711	Riesling ...	Weiβ	1999	Müller
	4961	Chardonnay	Weiβ	2002	Bighorn

ERZEUGER	Weingut	Anbaugebiet	Region
	Creek	Barossa Valley	Südaustralien
	Helena	Napa Valley	Kalifornien
	Château La Rose	Saint-Emilion	Bordeaux
	Château La Pointe	Pomerol	Bordeaux
	Müller	Rheingau	Hessen
	Bighorn	Napa Valley	Kalifornien

Begriffe des Relationenmodells

Begriff	Informale Bedeutung
Attribut	Spalte einer Tabelle
Wertebereich	mögliche Werte eines Attributs (auch Domäne)
Attributwert	Element eines Wertebereichs
Relationenschema	Menge von Attributen
Relation	Menge von Zeilen einer Tabelle
Tupel	Zeile einer Tabelle
Datenbankschema	Menge von Relationenschemata
Datenbank	Menge von Relationen (Basisrelationen)

Begriffe des Relationenmodells /2

Begriff	Informale Bedeutung
Schlüssel	minimale Menge von Attributen, deren Werte ein Tupel einer Tabelle eindeutig identifizieren
Primärschlüssel	ein beim Datenbankentwurf ausgezeichneter Schlüssel
Fremdschlüssel	Attributmenge, die in einer anderen Relation Schlüssel ist
Fremdschlüsselbedingung	alle Attributwerte des Fremdschlüssels tauchen in der anderen Relation als Werte des Schlüssels auf

- **Attribute und Domänen**

- \mathcal{U} nichtleere, endliche Menge: **Universum**
- $A \in \mathcal{U}$: **Attribut**
- $\mathcal{D} = \{D_1, \dots, D_m\}$ Menge endlicher, nichtleerer Mengen: jedes D_i : **Wertebereich** oder **Domäne**
- total definierte Funktion $\text{dom} : \mathcal{U} \longrightarrow \mathcal{D}$
- $\text{dom}(A)$: **Domäne von A**
 $w \in \text{dom}(A)$: **Attributwert** für A

- **Relationenschemata und Relationen**

- $R \subseteq \mathcal{U}$: **Relationenschema**
- **Relation** r über $R = \{A_1, \dots, A_n\}$ (kurz: $r(R)$) ist endliche Menge von Abbildungen $t : R \longrightarrow \bigcup_{i=1}^m D_i$, **Tupel** genannt
- Es gilt $t(A) \in \text{dom}(A)$ ($t(A)$ Restriktion von t auf $A \in R$)
- für $X \subseteq R$ analog $t(X)$ **X-Wert** von t
- Menge aller Relationen über R : **REL**(R) := $\{r \mid r(R)\}$

- **Datenbankschema und Datenbank**

- Menge von Relationenschemata $S := \{R_1, \dots, R_p\}$:
Datenbankschema
- **Datenbank** über S : Menge von Relationen $d := \{r_1, \dots, r_p\}$, wobei
 $r_i(R_i)$
- Datenbank d über S : $d(S)$
- Relation $r \in d$: **Basisrelation**

- Identifizierende Attributmenge $K := \{B_1, \dots, B_k\} \subseteq R$:
$$\forall t_1, t_2 \in r [t_1 \neq t_2 \implies \exists B \in K : t_1(B) \neq t_2(B)]$$
- **Schlüssel**: ist minimale identifizierende Attributmenge
 - {Name, Jahrgang, Weingut} und
 - {WeinID} für WEINE
- **Primattribut**: Element eines Schlüssels
- **Primärschlüssel**: ausgezeichneter Schlüssel
- **Oberschlüssel** oder **Superkey**: jede Obermenge eines Schlüssels (= identifizierende Attributmenge)
- **Fremdschlüssel**: $X(R_1) \rightarrow Y(R_2)$
$$\{t(X)|t \in r_1\} \subseteq \{t(Y)|t \in r_2\}$$

Relationaler DB-Entwurf

- Verfeinern des logischen Entwurfs
- Ziel: Vermeidung von Redundanzen durch Aufspalten von Relationenschemata, ohne gleichzeitig
 - semantische Informationen zu verlieren (Abhängigkeitstreue)
 - die Möglichkeit zur Rekonstruktion der Relationen zu verlieren (Verbundtreue)
- Redundanzvermeidung durch Normalformen (s.u.)

Relation WEINE mit Redundanzen

WeinID	Name	...	Weingut	Anbaugebiet	Region
1042	La Rose Gr. Cru	...	Ch. La Rose	Saint-Emilion	Bordeaux
2168	Creek Shiraz	...	Creek	Barossa Valley	Südaustralien
3456	Zinfandel	...	Helena	Napa Valley	Kalifornien
2171	Pinot Noir	...	Creek	Barossa Valley	Südaustralien
3478	Pinot Noir	...	Helena	Napa Valley	Kalifornien
4711	Riesling Res.	...	Müller	Rheingau	Hessen
4961	Chardonnay	...	Bighorn	Napa Valley	Kalifornien

- Redundanzen in Basisrelationen aus mehreren Gründen unerwünscht:
 - Redundante Informationen belegen unnötigen **Speicherplatz**
 - **Änderungsoperationen** auf Basisrelationen mit Redundanzen nur schwer korrekt umsetzbar: wenn eine Information redundant vorkommt, muss eine Änderung diese Information in allen ihren Vorkommen verändern
 - mit normalen relationalen Änderungsoperationen und den in relationalen Systemen vorkommenden lokalen Integritätsbedingungen (Schlüsseln) nur schwer realisierbar

- Einfügen in die redundanzbehaftete WEINE-Relation:

```
insert into WEINE (WeinID, Name, Farbe,  
    Jahrgang, Weingut, Anbaugebiet, Region)  
values (4711, 'Chardonnay', 'Weiß', 2004,  
    'Helena', 'Rheingau', 'Kalifornien')
```

- WeinID 4711 bereits anderem Wein zugeordnet: verletzt FD
 $\text{WeinID} \rightarrow \text{Name}$
- Weingut Helena war bisher im Napa Valley angesiedelt: verletzt FD
 $\text{Weingut} \rightarrow \text{Anbaugebiet}$
- Rheingau liegt nicht in Kalifornien: verletzt FD
 $\text{Anbaugebiet} \rightarrow \text{Region}$
- auch update- und delete-Anomalien

Funktionale Abhangigkeit zwischen Attributemengen X und Y

Wenn in jedem Tupel der Relation der Attributwert unter den X -Komponenten den Attributwert unter den Y -Komponenten festlegt.

- Unterscheiden sich zwei Tupel in den X -Attributen nicht, so haben sie auch gleiche Werte fur alle Y -Attribute
- Notation fur funktionale Abhangigkeit (FD, von functional dependency): $X \rightarrow Y$
- Beispiel:

WeinID \rightarrow Name, Weingut
Anbaugebiet \rightarrow Region

- aber nicht: Weingut \rightarrow Name

Schlüssel als Spezialfall

- für Beispiel auf Folie 5-11

WeinID → Name, Farbe, Jahrgang, Weingut,
Anbaugebiet, Region

- Immer: WeinID → WeinID,
dann gesamtes Schema auf rechter Seite
- Wenn linke Seite minimal: Schlüssel
- Formal: Schlüssel X liegt vor, wenn für Relationenschema R FD $X \rightarrow R$ gilt und X minimal

Ziel des Datenbankentwurfs

alle gegebenen funktionalen Abhängigkeiten in **Schlüsselabhängigkeiten** umformen, ohne dabei semantische Information zu verlieren

Ableitung von FDs

r	A	B	C
	a_1	b_1	c_1
	a_2	b_1	c_1
	a_3	b_2	c_1
	a_4	b_1	c_1

- Tabelle genügt $A \rightarrow B$ und $B \rightarrow C$
- dann gilt auch $A \rightarrow C$
- nicht ableitbar $C \rightarrow A$ oder $C \rightarrow B$

Formalisierung im nächsten Abschnitt!

Normalformen

- Relationenschemata, Schlüssel und Fremdschlüssel so wählen, dass
 1. alle Anwendungsdaten aus den Basisrelationen hergeleitet werden können,
 2. nur semantisch sinnvolle und konsistente Anwendungsdaten dargestellt werden können und
 3. die Anwendungsdaten möglichst nicht-redundant dargestellt werden.
- Hier: Forderung 3
 - Redundanzen innerhalb einer Relation: **Normalformen**
 - globale Redundanzen: **Minimalität**

Normalformen

- legen Eigenschaften von Relationenschemata fest
- verbieten bestimmte Kombinationen von funktionalen Abhängigkeiten in Relationen
- sollen Redundanzen und Anomalien vermeiden

Erste Normalform

- erlaubt nur atomare Attribute in den Relationenschemata, d.h. als Attributwerte sind Elemente von Standard-Datentypen wie `integer` oder `string` erlaubt, aber keine Konstruktoren wie `array` oder `set`
- Nicht in 1NF:

Weingut	Anbaugebiet	Region	WName
Ch. La Rose	Saint-Emilion	Bordeaux	La Rose Grand Cru
Creek	Barossa Valley	Südaustralien	Creek Shiraz, Pinot Noir
Helena	Napa Valley	Kalifornien	Zinfandel, Pinot Noir
Müller	Rheingau	Hessen	Riesling Reserve
Bighorn	Napa Valley	Kalifornien	Chardonnay

Erste Normalform /2

- in erster Normalform:

Weingut	Anbaugebiet	Region	WName
Ch. La Rose	Saint-Emilion	Bordeaux	La Rose Grand Cru
Creek Creek	Barossa Valley Barossa Valley	Südaustralien Südaustralien	Creek Shiraz Pinot Noir
Helena Helena	Napa Valley Napa Valley	Kalifornien Kalifornien	Zinfandel Pinot Noir
Müller Bighorn	Rheingau Napa Valley	Hessen Kalifornien	Riesling Reserve Chardonnay

Zweite Normalform

- **partielle Abhängigkeit** liegt vor, wenn ein Attribut funktional schon von einem **Teil** des Schlüssels abhängt

Name	Weingut	Farbe	Anbaugebiet	Region	Preis
La Rose ...	Ch. La Rose	Rot	Saint-Emilion	Bordeaux	39.00
Creek Shiraz	Creek	Rot	Barossa Valley	Südaustralien	7.99
Pinot Noir	Creek	Rot	Barossa Valley	Südaustralien	10.99
Zinfandel	Helena	Rot	Napa Valley	Kalifornien	5.99
Pinot Noir	Helena	Rot	Napa Valley	Kalifornien	19.99
Riesling Reserve	Müller	Weiβ	Rheingau	Hessen	14.99
Chardonnay	Bighorn	Weiβ	Napa Valley	Kalifornien	9.90

$f_1: \text{Name}, \text{Weingut} \rightarrow \text{Preis}$

$f_2: \text{Name} \rightarrow \text{Farbe}$

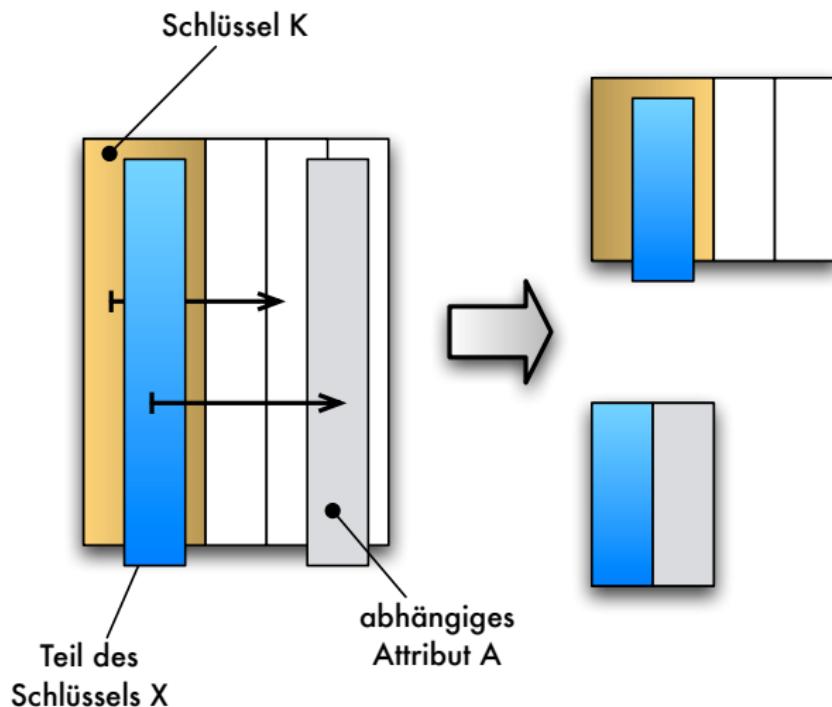
$f_3: \text{Weingut} \rightarrow \text{Anbaugebiet, Region}$

$f_4: \text{Anbaugebiet} \rightarrow \text{Region}$

Zweite Normalform

Zweite Normalform eliminiert derartige partielle Abhängigkeiten bei Nichtschlüsselattributen

Eliminierung partieller Abhängigkeiten



Zweite Normalform /2

- Beispielrelation in 2NF

R1(Name, Weingut, Preis)

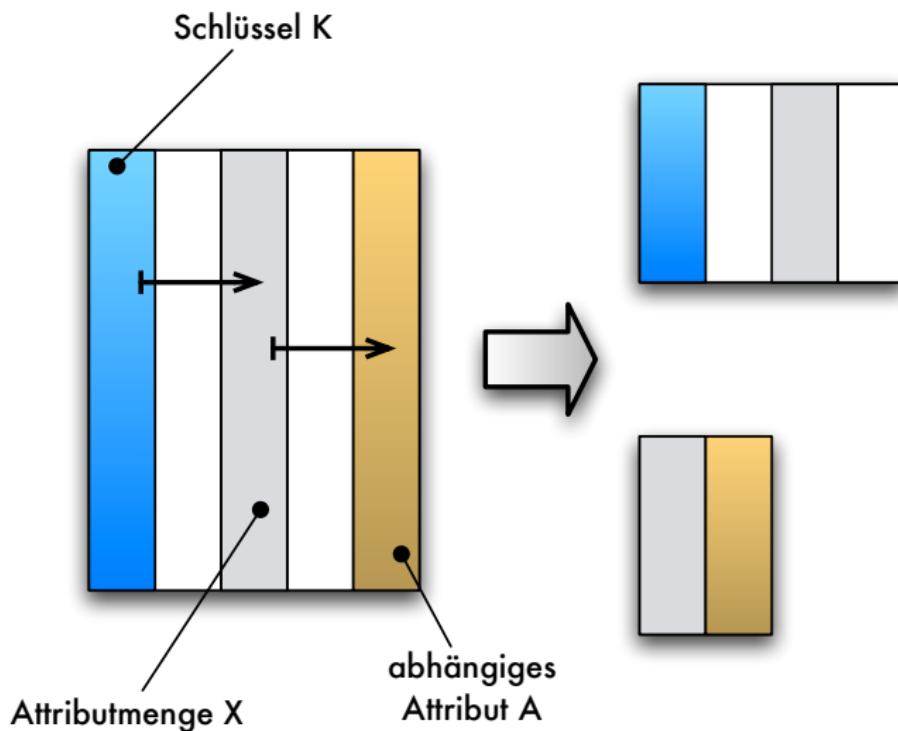
R2(Name, Farbe)

R3(Weingut, Anbaugebiet, Region)

Dritte Normalform

- eliminiert (zusätzlich) transitive Abhängigkeiten
- etwa Weingut → Anbaugebiet und Anbaugebiet → Region in Relation auf Folie 5-21
- man beachte: 3NF betrachtet nur Nicht-Schlüsselattribute als Endpunkt transitiver Abhängigkeiten

Eliminierung transitiver Abhängigkeiten



Dritte Normalform /2

- transitive Abhängigkeit in R3, d.h. R3 verletzt 3NF
- Beispielrelation in 3NF

R3_1(Weingut, Anbaugebiet)

R3_2(Anbaugebiet, Region)

Relationenschema R , $X \subseteq R$ und F ist eine FD-Menge über R

Dritte Normalform

- $A \in R$ heißt **transitiv abhängig** von X bezüglich F genau dann, wenn es ein $Y \subseteq R$ gibt mit $X \rightarrow Y, Y \not\rightarrow X, Y \rightarrow A, A \notin XY$
- erweitertes Relationenschema $\mathcal{R} = (R, \mathcal{K})$ ist in **3NF** bezüglich F genau dann, wenn $\forall A \in R$:
 - A ist Nicht-Primattribut in R
 - $\wedge A$ transitiv abhängig von einem $K \in \mathcal{K}$ bezüglich F_i .
- Nicht-Primattribut: A ist in keinem Schlüssel von R enthalten

Verschärfung der 3NF: Eliminierung transitiver Abhängigkeiten auch zwischen Primattributen

Name	Weingut	Händler	Preis
La Rose Grand Cru	Château La Rose	Weinkontor	39.90
Creek Shiraz	Creek	Wein.de	7.99
Pinot Noir	Creek	Wein.de	10.99
Zinfandel	Helena	GreatWines.com	5.99
Pinot Noir	Helena	GreatWines.com	19.99
Riesling Reserve	Müller	Weinkeller	19.99
Chardonnay	Bighorn	Wein-Dealer	9.90

Boyce-Codd-Normalform

Name, Weingut → Preis

Weingut → Händler

Händler → Weingut

- Schlüsselkandidaten: { Name, Weingut } und { Name, Händler }
- in 3NF, nicht jedoch in BCNF

- erweitertes Relationenschema $\mathcal{R} = (R, \mathcal{K})$, FD-Menge F
- BCNF formal:

Boyce-Codd-Normalform

$\nexists A \in R : A$ transitiv abhängig von einem $K \in \mathcal{K}$ bezüglich F .

- Schema in BCNF:

WEINE(Name, Weingut, Preis)
WEINHANDEL(Weingut, Händler)

- BCNF kann jedoch **Abhängigkeitstreue** verletzen, daher oft nur bis 3NF

Minimalität

- Global Redundanzen vermeiden
- andere Kriterien (wie Normalformen) mit möglichst wenig Schemata erreichen
- Beispiel: Attributmenge ABC , FD-Menge $\{A \rightarrow B, B \rightarrow C\}$
- Datenbankschemata in dritter Normalform:

$$S = \{(AB, \{A\}), (BC, \{B\})\}$$

$$S' = \{(AB, \{A\}), (BC, \{B\}), (AC, \{A\})\}$$

Redundanzen in S'

Schemaeigenschaften

Kennung	Schemaeigenschaft	Kurzcharakteristik
	1NF	nur atomare Attribute
	2NF	keine partielle Abhängigkeit eines Nicht-Primatributes von einem Schlüssel
S1	3NF	keine transitive Abhängigkeit eines Nicht-Primatributes von einem Schlüssel
	BCNF	keine transitive Abhängigkeit eines Attributes von einem Schlüssel
S2	Minimalität	minimale Anzahl von Relationenschemata, die die anderen Eigenschaften erfüllt

Transformationseigenschaften

- Bei einer Zerlegung einer Relation in mehrere Relationen ist darauf zu achten, dass
 1. nur semantisch sinnvolle und konsistente Anwendungsdaten dargestellt (**Abhängigkeitstreue**) und
 2. alle Anwendungsdaten aus den Basisrelationen hergeleitet werden können (**Verbundtreue**)

- **Abhangigkeitstreue:** eine Menge von Abhangigkeiten kann quivalent in eine zweite Menge von Abhangigkeiten transformiert werden
- spezieller: in die Menge der Schlsselabhangigkeiten, da diese vom Datenbanksystem effizient berpruft werden kann
 - die Menge der Abhangigkeiten soll quivalent zu der Menge der Schlsselbedingungen im resultierenden Datenbankschema sein
 - quivalenz sichert zu, dass mit den Schlsselabhangigkeiten semantisch genau die gleichen Integritatsbedingungen ausgedrckt werden wie mit den funktionalen oder anderen Abhangigkeiten vorher

Abhängigkeitstreue: Beispiel

- Zerlegung des Relationenschemas WEINE (Folie 5-21) in 3NF:

R1(Name, Weingut, Preis)

R2(Name, Farbe)

R3_1(Weingut, Anbaugebiet)

R3_2(Anbaugebiet, Region)

mit Schlüsselabhängigkeiten

Name, Weingut → Preis

Name → Farbe

Weingut → Anbaugebiet

Anbaugebiet → Region

- äquivalent zu FDs $f_1 \dots f_4$ (Folie 5-21) \rightsquigarrow abhängigkeitstreu

Abhängigkeitstreue: Beispiel /2

- Postleitzahl-Struktur der Deutschen Post

ADRESSE(PLZ (P), Ort (O), Strasse(S), Hausnummer(H))

und funktionalen Abhängigkeiten F

$$OSH \rightarrow P, P \rightarrow O$$

- Schlüsselkandidaten: OSH und PSH \rightsquigarrow 3NF
- nicht in BCNF (wegen $PSH \rightarrow P \rightarrow O$): daher Zerlegung von ADRESSE
- aber: jede Zerlegung würde $OSH \rightarrow P$ zerstören
- Menge der sich ergebenden FDs ist nicht äquivalent zu F , die Zerlegung damit nicht abhängigkeitstreu

- lokal erweitertes Datenbankschema $S = \{(R_1, \mathcal{K}_1), \dots, (R_p, \mathcal{K}_p)\}$;
ein Menge F lokaler Abhangigkeiten

Abhangigkeitstreue

S charakterisiert vollstandig F (oder: ist abhangigkeitstreu bezuglich F) genau dann, wenn

$$F \equiv \{K \rightarrow R \mid (R, \mathcal{K}) \in S, K \in \mathcal{K}\}$$

- zur Erfüllung des Kriteriums der Normalformen müssen Relationenschemata teilweise in kleinere Relationenschemata zerlegt werden
- für Beschränkung auf „sinnvolle“ Zerlegungen gilt Forderung, dass die Originalrelation wieder aus den zerlegten Relationen mit dem natürlichen Verbund zurückgewonnen werden kann
~~> **Verbundtreue**

Verbundtreue: Beispiele

- Zerlegung des Relationenschemas $R = ABC$ in

$$R_1 = AB \text{ und } R_2 = BC$$

- Dekomposition bei Vorliegen der Abhängigkeiten

$$F = \{A \rightarrow B, C \rightarrow B\}$$

ist nicht verbundtreu

- dagegen bei Vorliegen von

$$F' = \{A \rightarrow B, B \rightarrow C\}$$

verbundtreu

Verbundtreue Dekomposition

- Originalrelation:

A	B	C
1	2	3
4	2	3

- Dekomposition:

A	B
1	2
4	2

B	C
2	3

- Verbund (verbundtreu):

A	B	C
1	2	3
4	2	3

Nicht verbundstreue Dekomposition

- Originalrelation:

A	B	C
1	2	3
4	2	5

- Dekomposition:

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

- Verbund (nicht verbundtreu):

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

Verbundtreue

Die Dekomposition einer Attributmenge X in X_1, \dots, X_p mit $X = \bigcup_{i=1}^p X_i$ heißt **verbundtreu** ($\pi \bowtie$ -treu, lossless) bezüglich einer Menge von Abhängigkeiten F über X genau dann, wenn

$$\forall r \in \mathbf{SAT}_X(F) : \pi_{X_1}(r) \bowtie \cdots \bowtie \pi_{X_p}(r) = r$$

gilt.

- einfaches Kriterium für Verbundtreue bei Dekomposition in zwei Relationenschemata: Dekomposition von X in X_1 und X_2 ist verbundtreu bzgl. F , wenn $X_1 \cap X_2 \rightarrow X_1 \in F^+$ oder $X_1 \cap X_2 \rightarrow X_2 \in F^+$

Transformationseigenschaften

Kennung	Transformationseigenschaft	Kurzcharakteristik
T1	Abhängigkeitstreue	alle gegebenen Abhängigkeiten sind durch Schlüssel repräsentiert
T2	Verbundtreue	Originalrelationen können durch den Verbund der Basisrelationen wiederergewonnen werden

Weitere Abhängigkeiten

- Mehrwertige Abhangigkeit (kurz: MVD)
 - innerhalb einer Relation r wird einem Attributwert von X eine Menge von Y -Werten zugeordnet, unabhangig von den Werten der restlichen Attribute \rightsquigarrow Vierte Normalform
- Verbundabhangigkeit (kurz: JD)
 - R kann ohne Informationsverlust in R_1, \dots, R_p aufgetrennt werden:
 $\bowtie [R_1, \dots, R_p]$
- Inklusionsabhangigkeit (kurz: IND)
 - auf der rechten Seite einer Fremdschlsselabhangigkeit nicht unbedingt der Primarschlssel einer Relation

Mehrwertige Abhängigkeiten

- Folge der 1NF: Mehrwertige Abhängigkeiten erzeugen Redundanz:

WEIN_EMPFEHLUNG	WName	Jahrgang	Gericht
	Chardonnay	2002	Geflügel
	Chardonnay	2002	Fisch
	Chardonnay	2003	Fisch
	Chardonnay	2003	Geflügel
	Shiraz	2003	Wild
	Shiraz	2003	Lamm
	Shiraz	2004	Wild
	Shiraz	2004	Lamm

Mehrwertige Abhangigkeiten /2

- eine (oder mehrere) Gruppe von Attributwerten ist von einem Schlssel bestimmt, unabhangig von anderen Attributen
- hier: Menge von Jahrgangen plus Menge von Gerichten

$\text{WName} \rightarrow\!\!\! \rightarrow \text{Jahrgang}$, $\text{WName} \rightarrow\!\!\! \rightarrow \text{Gericht}$

- Resultat: Redundanz durch Bildung aller Kombinationen

Mehrwertige Abhängigkeiten formal

- Relation $r(R)$ mit $X, Y \subseteq R$, $Z := R - (X \cup Y)$ **genügt** der MVD $X \rightarrow\!\!\! \rightarrow Y$ gdw.

$$\begin{aligned} \forall t_1, t_2 \in r : & [(t_1 \neq t_2 \wedge t_1(X) = t_2(X))] \\ \implies & \exists t_3 \in r : t_3(X) = t_1(X) \wedge t_3(Y) = t_1(Y) \wedge \\ & t_3(Z) = t_2(Z)] \end{aligned}$$

- Relation $r(R)$ mit $R = XYZ$ und $X \rightarrow\!\!\! \rightarrow Y$:
 - wenn $(x_1, y_1, z_1) \in r$ und $(x_1, y_2, z_2) \in r$
 - dann auch: $(x_1, y_1, z_2) \in r$ und $(x_1, y_2, z_1) \in r$
- Bsp.: wegen ('Chardonnay', 2002, 'Geflügel') und ('Chardonnay', 2003, 'Fisch') müssen auch ('Chardonnay', 2002, 'Fisch') und ('Chardonnay', 2003, 'Geflügel') enthalten sein

- wunschenswerte Schemaeigenschaft bei Vorliegen von MVDs: **vierte Normalform**
- fordert die Beseitigung derartiger Redundanzen: keine zwei MVDs zwischen Attributen einer Relation
- Beispiel von Folie 5-46 verletzt diese Forderung
- Prinzip
 - Elimination der rechten Seite einer der beiden mehrwertigen Abhangigkeiten,
 - linke Seite mit dieser rechten Seite in neue Relation kopiert

Vierte Normalform

WEIN_JAHR	WName	Jahrgang
	Chardonnay	2002
	Chardonnay	2003
	Shiraz	2003
	Shiraz	2004

WEIN_GERICHT	WName	Gericht
	Chardonnay	Geflügel
	Chardonnay	Fisch
	Shiraz	Wild
	Shiraz	Lamm

- Relationenschema R mit $X, Y \subseteq R$, MVD-Menge M über R
- MVD $X \rightarrow\!\!\rightarrow Y$ heißt trivial genau dann, wenn $Y \subseteq X$ oder $X \cup Y = R$

Vierte Normalform

erweitertes Relationenschema $\mathcal{R} = (R, \mathcal{K})$ ist in **vierter Normalform** (4NF) bezüglich M genau dann, wenn für alle $X \rightarrow\!\!\rightarrow Y \in M^+$ gilt:

$X \rightarrow\!\!\rightarrow Y$ ist trivial oder $X \supseteq K$ für ein $K \in \mathcal{K}$.

Nichttriviale MVDs

- Erweiterung der Relation WEIN_JAHR von Folie 5-50 um Attribute Farbe und Restsüße
- MVD $\text{WName} \rightarrow\!\!\! \rightarrow \text{Jahrgang}$ ist nicht mehr trivial
- Zerlegung:

$\text{WEIN_JAHR1}(\underline{\text{WName}}, \underline{\text{Jahrgang}})$

$\text{WEIN_JAHR2}(\underline{\text{WName}}, \text{Farbe}, \text{Restsüße})$

Zusammenfassung

- funktionale Abhängigkeiten
- Normalformen (1NF - 3NF, BCNF)
- Abhängigkeitstreue und Verbundtreue
- Entwurfsverfahren
- mehrwertige Abhängigkeiten

Kontrollfragen

- Welches Ziel hat die Normalisierung relationaler Schemata?
- Welche Eigenschaften relationaler Schemata werden bei den Normalformen berücksichtigt?
- Was unterscheidet 3NF und BCNF?
- Was fordern Abhängigkeitstreue und Verbundtreue?



Teil VI

Relationale Theorie

1. Formalisierung
2. Rechnen mit FDs
3. Mehr zu Normalformen
4. Entwurfsverfahren

Lernziele für heute . . .

- Vertiefte Kenntnisse der theoretischen Grundlagen des relationalen Entwurfs
- Korrektheit der Normalisierung
- Details des Syntheseverfahrens



Formalisierung

- **Attribute und Domänen**

- \mathcal{U} nichtleere, endliche Menge: **Universum**
- $A \in \mathcal{U}$: **Attribut**
- $\mathcal{D} = \{D_1, \dots, D_m\}$ Menge endlicher, nichtleerer Mengen: jedes D_i : **Wertebereich** oder **Domäne**
- total definierte Funktion $\text{dom} : \mathcal{U} \longrightarrow \mathcal{D}$
- $\text{dom}(A)$: **Domäne von A**
 $w \in \text{dom}(A)$: **Attributwert** für A

- **Relationenschemata und Relationen**

- $R \subseteq \mathcal{U}$: **Relationenschema**
- **Relation** r über $R = \{A_1, \dots, A_n\}$ (kurz: $r(R)$) ist endliche Menge von Abbildungen $t : R \longrightarrow \bigcup_{i=1}^m D_i$, **Tupel** genannt
- Es gilt $t(A) \in \text{dom}(A)$ ($t(A)$ Restriktion von t auf $A \in R$)
- für $X \subseteq R$ analog $t(X)$ **X-Wert** von t
- Menge aller Relationen über R : **REL**(R) := $\{r \mid r(R)\}$

- **Datenbankschema und Datenbank**

- Menge von Relationenschemata $S := \{R_1, \dots, R_p\}$: **Datenbankschema**
- **Datenbank** über S : Menge von Relationen $d := \{r_1, \dots, r_p\}$, wobei $r_i(R_i)$
- Datenbank d über S : $d(S)$
- Relation $r \in d$: **Basisrelation**

Rechnen mit FDs

Wiederholung: Ableitung von FDs

r	A	B	C
	a_1	b_1	c_1
	a_2	b_1	c_1
	a_3	b_2	c_1
	a_4	b_1	c_1

- genügt $A \rightarrow B$ und $B \rightarrow C$
- dann gilt auch $A \rightarrow C$
- nicht ableitbar $C \rightarrow A$ oder $C \rightarrow B$

- Gilt für f über R $\text{SAT}_R(F) \subseteq \text{SAT}_R(f)$, dann **impliziert** F die FD f (kurz: $F \models f$)
- obiges Beispiel:

$$F = \{A \rightarrow B, B \rightarrow C\} \models A \rightarrow C$$

- Hüllenbildung: Ermittlung **aller** funktionalen Abhängigkeiten, die aus einer gegebenen FD-Menge abgeleitet werden können
- **Hülle** $F_R^+ := \{f \mid (f \text{ FD über } R) \wedge F \models f\}$
- Beispiel:

$$\begin{aligned} \{A \rightarrow B, B \rightarrow C\}^+ &= \{A \rightarrow B, B \rightarrow C, A \rightarrow C, AB \rightarrow C, A \rightarrow BC, \dots, \\ &\quad AB \rightarrow AB, \dots\} \end{aligned}$$

F1	Reflexivität	$X \supseteq Y \implies X \rightarrow Y$
F2	Augmentation	$\{X \rightarrow Y\} \implies XZ \rightarrow YZ \text{ sowie } XZ \rightarrow Y$
F3	Transitivität	$\{X \rightarrow Y, Y \rightarrow Z\} \implies X \rightarrow Z$
F4	Dekomposition	$\{X \rightarrow YZ\} \implies X \rightarrow Y$
F5	Vereinigung	$\{X \rightarrow Y, X \rightarrow Z\} \implies X \rightarrow YZ$
F6	Pseudotransitivität	$\{X \rightarrow Y, WY \rightarrow Z\} \implies WX \rightarrow Z$

F1-F3 bekannt als **Armstrong-Axiome** (sound, complete)

- **gültig** (sound): Regeln leiten keine FDs ab, die logisch nicht impliziert
- **vollständig** (complete): alle implizierten FDs werden abgeleitet
- **unabhängig** (independent) oder auch bzgl. \subseteq minimal: keine Regel kann weggelassen werden

Beweis: F1

- Annahme: $X \supseteq Y$, $X, Y \subset R$, $t_1, t_2 \in r(R)$ mit $t_1(X) = t_2(X)$
- dann folgt: $t_1(Y) = t_2(Y)$ wegen $X \supseteq Y$
- daraus folgt: $X \rightarrow Y$

Beweis: F2

- Annahme: $X \rightarrow Y$ gilt in $r(R)$, jedoch nicht: $XZ \rightarrow YZ$
- dann müssen zwei Tupel $t_1, t_2 \in r(R)$ existieren, so dass gilt
 - (1) $t_1(X) = t_2(X)$
 - (2) $t_1(Y) = t_2(Y)$
 - (3) $t_1(XZ) = t_2(XZ)$
 - (4) $t_1(YZ) \neq t_2(YZ)$
- Widerspruch wegen $t_1(Z) = t_2(Z)$ aus (1) und (3), woraus folgt:
 $t_1(YZ) = t_2(YZ)$ (in Verbindung mit (4))

Beweis: F3

- Annahme: in $r(R)$ gelten:
 - (1) $X \rightarrow Y$
 - (2) $Y \rightarrow Z$
- demzufolge für zwei beliebige Tupel $t_1, t_2 \in r(R)$ mit $t_1(X) = t_2(X)$ muss gelten:
 - (3) $t_1(Y) = t_2(Y)$ (wegen (1))
 - (4) $t_1(Z) = t_2(Z)$ (wegen (3) und (2))
- daher gilt: $X \rightarrow Z$

- B-Axiome oder **RAP-Regeln**

R Reflexivität $\{\} \implies X \rightarrow X$

A Akkumulation $\{X \rightarrow YZ, Z \rightarrow AW\} \implies X \rightarrow YZA$

P Projektivität $\{X \rightarrow YZ\} \implies X \rightarrow Y$

- Regelmenge ist vollständig, da Armstrong-Axiome daraus abgeleitet werden können

Membership-Problem

Kann eine bestimmte FD $X \rightarrow Y$ aus der vorgegebenen Menge F abgeleitet werden, d.h. wird sie von F impliziert?

Membership-Problem: „ $X \rightarrow Y \in F^+ ?$ “

- **Hülle einer Attributmenge** X bzgl. F ist
$$X_F^+ := \{A \mid X \rightarrow A \in F^+\}$$
- Membership-Problem kann durch das modifizierte Problem

Membership-Problem (2): „ $Y \subseteq X_F^+ ?$ “

in linearer Zeit gelöst werden

Algorithmus Closure: Ermittlung der Hülle X^+ bzgl. F

Closure(F, X):

$X^+ := X$

 repeat

$\overline{X}^+ := X^+ \text{ /* R-Regel */}$

 forall FDs $Y \rightarrow Z \in F$

 if $Y \subseteq X^+$ then $X^+ := X^+ \cup Z \text{ /* A-Regel */}$

 until $X^+ = \overline{X}^+$

 return X^+

Member($F, X \rightarrow Y$): /* Test auf $X \rightarrow Y \in F^+$ */

 return $Y \subseteq \text{Closure}(F, X) \text{ /* P-Regel */}$

Algorithmus Closure: Beispiel

$$A \rightarrow C \in \{\underbrace{A \rightarrow B}_{f_1}, \underbrace{B \rightarrow C}_{f_2}\}^+?$$

- **Member**($\{f_1, f_2\}$, $A \rightarrow C$)
- $C \subseteq \text{Closure}(\{f_1, f_2\}, A)$
- X^+ ist initial $\{A\}$, schrittweises Hinzunehmen von B und C

- F heißt **äquivalent** zu G
- oder: F **Überdeckung** von G ; kurz: $F \equiv G$ falls $F^+ = G^+$

- d.h.:

$$\forall g \in G : g \in F^+ \wedge \forall f \in F : f \in G^+$$

- wichtige Entwurfsaufgabe: Finden einer Überdeckung, die
 - einerseits so wenig Attribute wie möglich in ihren funktionalen Abhängigkeiten und
 - andererseits möglichst wenig funktionale Abhängigkeiten insgesamt enthält
- verschiedene Formen von Überdeckung: nicht-redundant, reduziert, minimal, ringförmig

- Ziel: Entfernen überflüssiger Attribute auf linker bzw. rechter Seite von FDs
- **Linksreduktion:** entfernt unwesentliche Attribute auf der linken Seite einer FD
- **Rechtsreduktion:** entsprechend auf der rechten Seite
- erw. Relationenschema $\mathcal{R} = (R, \mathcal{K})$, FD-Menge F über R , A ist ein Attribut aus R und $X \rightarrow Y$ eine FD aus F

Unwesentliche Attribute

A heißt **unwesentlich** in $X \rightarrow Y$ bzgl. F , wenn

- $X = AZ, Z \neq X \implies (F - \{X \rightarrow Y\}) \cup \{Z \rightarrow Y\} \equiv F$ oder
- $Y = AW, W \neq Y \implies (F - \{X \rightarrow Y\}) \cup \{X \rightarrow W\} \equiv F$

Reduktionsoperationen /2

- A kann also aus der FD $X \rightarrow Y$ entfernt werden, ohne dass sich die Hülle von F ändert
- FD $X \rightarrow Y$ heißt **linksreduziert**, wenn kein Attribut in X unwesentlich ist.
- FD $X \rightarrow Y$ heißt **rechtsreduziert**, wenn kein Attribut in Y unwesentlich ist.

Minimale Überdeckung

- Eine **minimale Überdeckung** ist eine Überdeckung, die eine minimale Anzahl von FDs enthält
- Auswahl der **kleinsten** aller nicht-redundanten Überdeckungen
- FD-Menge F heißt **minimal** gdw.

$$\forall F' [F' \equiv F \Rightarrow |F| \leq |F'|]$$

- Bestimmung etwa durch reduzierte Überdeckung mit anschließender Äquivalenzklassenbildung (später)

Reduzierte Überdeckung

ReducedCover(F):

```
forall FD  $X \rightarrow Y \in F$  /* Linksreduktion */
  forall  $A \in X$  /* A unwesentlich ? */
    if  $Y \subseteq \text{Closure}(F, X - \{A\})$ 
      then ersetze  $X \rightarrow Y$  durch  $(X - A) \rightarrow Y$  in  $F$ 
forall verbleibende FD  $X \rightarrow Y \in F$  /* Rechtsreduktion */
  forall  $B \in Y$  /* B unwesentlich ? */
    if  $B \subseteq \text{Closure}(F - \{X \rightarrow Y\} \cup \{X \rightarrow (Y - B)\}, X)$ 
      then ersetze  $X \rightarrow Y$  durch  $X \rightarrow (Y - B)$ 
```

Eliminiere FDs der Form $X \rightarrow \emptyset$

Vereinige FDs der Form $X \rightarrow Y_1, X \rightarrow Y_2, \dots$ zu $X \rightarrow Y_1 Y_2 \dots$

return resultierende FDs

Reduzierte Überdeckung: Beispiel

- Geg.: FD-Menge

$$F = \{f_1 : A \rightarrow B, f_2 : AB \rightarrow C, f_3 : A \rightarrow C, f_4 : B \rightarrow A, f_5 : C \rightarrow E\}$$

1. Linksreduktion: bei FD f_2 Attribut A streichen, da $C \subseteq \text{Closure}(F, \{A\})$ (wegen f_3)
 2. Rechtsreduktion: FD f_3 durch $A \rightarrow \{\}$ ersetzt, da $C \subseteq \text{Closure}(\{A \rightarrow B, B \rightarrow C, A \rightarrow \{\}, B \rightarrow A, C \rightarrow E\}, \{A\})$
 3. Streichen von $A \rightarrow \{\}$
- Ergebnis:

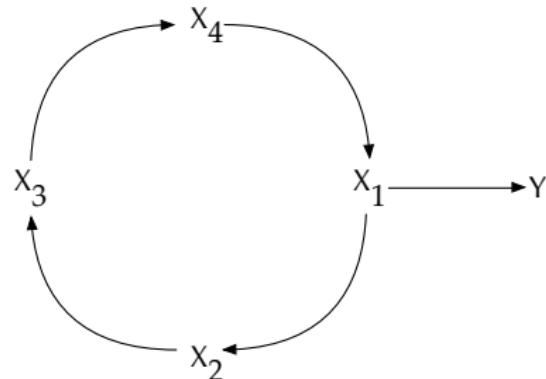
$$\text{ReducedCover}(F) = \{A \rightarrow B, B \rightarrow C, B \rightarrow A, C \rightarrow E\}$$

Äquivalenzklassen

- FDs mit äquivalenten linken Seiten werden zu einer Äquivalenzklasse zusammengefasst
- FDs $X_1 \rightarrow Y_1$ und $X_2 \rightarrow Y_2$ liegen in einer **Äquivalenzklasse**, wenn $X_1 \rightarrow X_2$ und $X_2 \rightarrow X_1$ gelten
- In einigen Fällen können nun zwei solche FDs in einer Äquivalenzklasse zu einer FD $X \rightarrow Y_1Y_2$ zusammengefasst werden
- Da die FDs einer Äquivalenzklasse in die Form $X_1 \rightarrow X_2, X_2 \rightarrow X_3, \dots, X_n \rightarrow X_1, X_1 \rightarrow Y$ überführt werden können, nennt man eine Überdeckung dieser Form eine **ringförmige Überdeckung**

Äquivalenzklassen /2

- linke Seiten sind äquivalent, wenn sie sich gegenseitig funktional bestimmen
- Relationenschema R mit $X_i, Y \subset R$, FD-Menge $X_i \rightarrow X_j$ und $X_i \rightarrow Y$ mit $1 \leq i, j \leq n$ kann dargestellt werden durch $(X_1, X_2, \dots, X_n) \rightarrow Y$



Mehr zu Normalformen

- Identifizierende Attributmenge $K := \{B_1, \dots, B_k\} \subseteq R$:

$$\forall t_1, t_2 \in r [t_1 \neq t_2 \implies \exists B \in K : t_1(B) \neq t_2(B)]$$

- Schlüssel:** ist minimale identifizierende Attributmenge
 - {Name, Jahrgang, Weingut} und
 - {WeinID} für WEINE
- Primattribut:** Element eines Schlüssels
- Primärschlüssel:** ausgezeichneter Schlüssel
- Oberschlüssel oder Superkey:** jede Obermenge eines Schlüssels (= identifizierende Attributmenge)

- Identifizierende Attributmenge $K := \{B_1, \dots, B_k\} \subseteq R$:

$$\forall t_1, t_2 \in r [t_1 \neq t_2 \implies \exists B \in K : t_1(B) \neq t_2(B)]$$

$$\implies K \rightarrow R$$

- Hülle einer Attributmenge X bzgl. F ist

$$X_F^+ := \{A \mid X \rightarrow A \in F^+\} \implies$$

- $K_F^+ = R$

$$\implies K = \text{Identifizierende Attributmenge}$$

- $K_F^+ = R \wedge X$ minimal (linksreduziert)

$$\implies K = \text{Schlüsselkandidat}$$

- Attribute, die nur funktional-abhängig sind, sind **nicht Teil** eines Schlüssels
- Attribute, die nie funktional-abhängig sind (nur bestimend), sind **immer Teil** eines Schlüssels

Wiederholung: Zweite Normalform

- **partielle Abhängigkeit** liegt vor, wenn ein Attribut funktional schon von einem **Teil** des Schlüssels abhängt

Name	Weingut	Farbe	Anbaugebiet	Region	Preis
La Rose ...	Ch. La Rose	Rot	Saint-Emilion	Bordeaux	39.00
Creek Shiraz	Creek	Rot	Barossa Valley	Südaustralien	7.99
Pinot Noir	Creek	Rot	Barossa Valley	Südaustralien	10.99
Zinfandel	Helena	Rot	Napa Valley	Kalifornien	5.99
Pinot Noir	Helena	Rot	Napa Valley	Kalifornien	19.99
Riesling Reserve	Müller	Weiβ	Rheingau	Hessen	14.99
Chardonnay	Bighorn	Weiβ	Napa Valley	Kalifornien	9.90

$f_1: \text{Name, Weingut} \rightarrow \text{Preis}$

$f_2: \text{Name} \rightarrow \text{Farbe}$

$f_3: \text{Weingut} \rightarrow \text{Anbaugebiet, Region}$

$f_4: \text{Anbaugebiet} \rightarrow \text{Region}$

- Hinweis: partiell abhängiges Attribut stören nur, wenn es **kein** Primattribut ist
- 2NF formal: erweitertes Relationenschema $\mathcal{R} = (R, \mathcal{K})$, FD-Menge F über R

Zweite Normalform

- Y **hängt partiell** von X bzgl. F ab, wenn die FD $X \rightarrow Y$ nicht linksreduziert ist
- Y **hängt voll** von X ab, wenn die FD $X \rightarrow Y$ linksreduziert ist
- \mathcal{R} ist in **2NF**, wenn \mathcal{R} in 1NF ist und jedes Nicht-Primattribut von R voll von jedem Schlüssel von \mathcal{R} abhängt

Entwurfsverfahren

Entwurfsverfahren: Ziele

- Universum \mathcal{U} und FD-Menge F gegeben
- lokal erweitertes Datenbankschema $S = \{(R_1, \mathcal{K}_1), \dots, (R_p, \mathcal{K}_p)\}$ berechnen mit
 - **T1:** S charakterisiert vollständig F
 - **S1:** S ist in 3NF bezüglich F
 - **T2:** Dekomposition von \mathcal{U} in R_1, \dots, R_p ist verbundstreu bezüglich F
 - **S2:** Minimalität, d.h.
 $\nexists S' : S'$ erfüllt **T1**, **S1**, **T2** und $|S'| < |S|$

Entwurfsverfahren: Beispiel

- Datenbankschemata schlecht entworfen, wenn nur eins dieser vier Kriterien nicht erfüllt
- Beispiel: $S = \{(AB, \{A\}), (BC, \{B\}), (AC, \{A\})\}$ erfüllt **T1**, **S1** und **T2** bezüglich $F = \{A \rightarrow B, B \rightarrow C, A \rightarrow C\}$
in dritter Relation AC -Tupel redundant oder inkonsistent
- korrekt: $S' = \{(AB, \{A\}), (BC, \{B\})\}$

Dekomposition

- Geg.: initiales Universalrelationenschema $\mathcal{R} = (\mathcal{U}, \mathcal{K}(F))$ mit allen Attributen und einer von erfassten FDs F über R implizierten Schlüsselmenge
 - Attributmenge \mathcal{U} und eine FD-Menge F
 - suche alle $K \rightarrow \mathcal{U}$ mit K minimal, für die $K \rightarrow \mathcal{U} \in F^+$ gilt ($\mathcal{K}(F)$)
- Ges.: Zerlegung in $D = \{\mathcal{R}_1, \mathcal{R}_2, \dots\}$ von 3NF-Relationenschemata

Dekomposition: Algorithmus

Decompose(\mathcal{R}) :

 Setze $D := \{\mathcal{R}\}$

while $\mathcal{R}' \in D$, das 3NF nicht erfüllt

/ Finde Attribut A, das transitiv von K abhängig ist */*

if Schlüssel K mit $K \rightarrow Y, Y \not\rightarrow K, Y \rightarrow A, A \notin KY$ **then**

/ Zerlege Relationenschema R bzgl. A */*

$R_1 := R - A$, $R_2 := YA$

$\mathcal{R}_1 := (R_1, \mathcal{K})$, $\mathcal{R}_2 := (R_2, \mathcal{K}_2 = \{Y\})$

$D := (D - \mathcal{R}') \cup \{\mathcal{R}_1\} \cup \{\mathcal{R}_2\}$

end if

end while

return D

Dekomposition: Beispiel

- initiales Relationenschema $R = ABC$
- funktionale Abhängigkeiten $F = \{A \rightarrow B, B \rightarrow C\}$
- Schlüssel $K = A$

Dekomposition: Beispiel /2

- initiales Relationenschema R mit Name, Weingut, Preis, Farbe, Anbaugebiet, Region
- funktionale Abhängigkeiten

$f_1: \text{Name, Weingut} \rightarrow \text{Preis}$

$f_2: \text{Name, Weingut} \rightarrow \text{Weingut}$

$f_3: \text{Name, Weingut} \rightarrow \text{Name}$

$f_4: \text{Name} \rightarrow \text{Farbe}$

$f_5: \text{Weingut} \rightarrow \text{Anbaugebiet, Region}$

$f_6: \text{Anbaugebiet} \rightarrow \text{Region}$

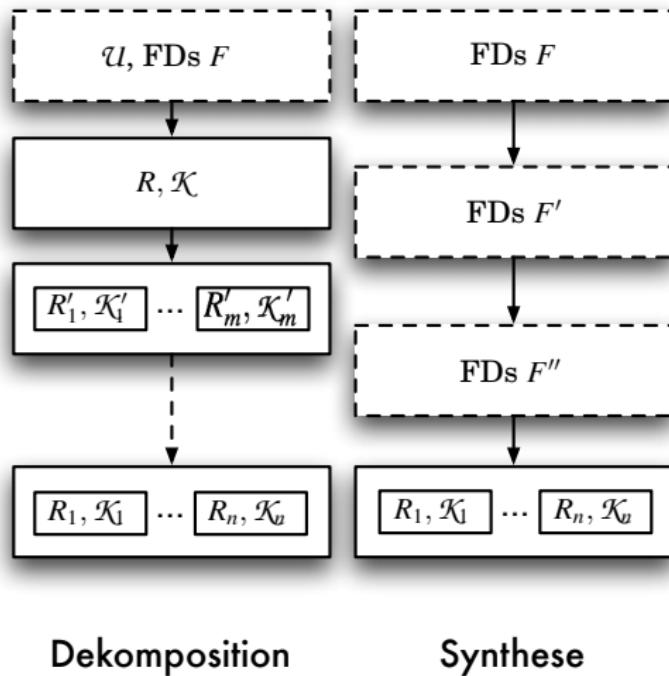
Dekomposition: Bewertung

- Vorteile: 3NF, Verbundtreue
- Nachteile: restliche Kriterien nicht, reihenfolgeabhängig, NP-vollständig (Schlüsselsuche)

Details zum Syntheseverfahren

- Prinzip: Synthese formt Original-FD-Menge F in resultierende Menge von Schlüsselabhängigkeiten G so um, dass $F \equiv G$ gilt
- „Abhängigkeitstreue“ im Verfahren verankert
- 3NF und Minimalität wird auch erreicht, reihenfolgeunabhängig
- Zeitkomplexität: quadratisch

Vergleich Dekomposition — Synthese



Dekomposition

Synthese

Syntheseverfahren für Relationenschema R mit FDs



Ges.: verlustfreie und abhängigkeitstreue Zerlegung in R_1, \dots, R_n , wobei alle R_i in 3NF sind

Synthesize(F):

$\hat{F} := \text{MinimalCover}(F)$ /* Bestimme minimale Überdeckung */

Bilde Äquivalenzklassen C_i von FDs aus \hat{F} mit gleichen oder äquivalenten linken Seiten, d.h. $C_i = \{X_i \rightarrow A_{i1}, X_i \rightarrow A_{i2}, \dots\}$

Bilde zu jeder Äquivalenzklasse C_i ein Schema der Form

$R_{C_i} = \{X_i \cup \{A_{i1}\} \cup \{A_{i2}\} \cup \dots\}$

if keines der Schemata R_{C_i} enthält einen Schlüssel von R

then erzeuge weiteres Relationenschema R_K mit Attributen

aus R , die Schlüssel bilden

return $\{R_K, R_{C_1}, R_{C_2}, \dots\}$

Synthese Beispiel

- FD-Menge

$$F = \{A \rightarrow B, AB \rightarrow C, A \rightarrow C, B \rightarrow A, C \rightarrow E\}$$

- minimale Überdeckung

$$\hat{F} = \{A \rightarrow B, B \rightarrow C, B \rightarrow A, C \rightarrow E\}$$

- Zusammenfassung zu Äquivalenzklassen

$$C_1 = \{A \rightarrow B, B \rightarrow C, B \rightarrow A\}$$

$$C_2 = \{C \rightarrow E\}$$

- Syntheseergebnis

$$(ABC, \{\{A\}, \{B\}\}), (CE, \{C\})$$

Erreichung der Verbundtreue

- Erreichen der Verbundtreue durch einfachen „Trick“:
 - Erweitern der Original-FD-Menge F um $\mathcal{U} \rightarrow \delta$ um Dummy-Attribut δ
 - δ wird nach Synthese entfernt
- Beispiel: $\{A \rightarrow B, C \rightarrow E\}$
 - Syntheseergebnis $(AB, \{A\}), (CE, \{C\})$ ist nicht verbundtreu, da Universalschlüssel in keinem Schema enthalten ist
 - Dummy-FD $ABCE \rightarrow \delta$; reduziert auf $AC \rightarrow \delta$
 - liefert drittes Relationenschema

$$(AC, \{AC\})$$

Synthese: Beispiel

- FD-Menge

$$\begin{aligned}f_1 &= \{\text{Name}, \text{Weingut} \rightarrow \text{Preis}\} \\f_2 &= \{\text{Name}, \text{Weingut} \rightarrow \text{Weingut}\} \\f_3 &= \{\text{Name}, \text{Weingut} \rightarrow \text{Name}\} \\f_4 &= \{\text{Name} \rightarrow \text{Farbe}\} \\f_5 &= \{\text{Weingut} \rightarrow \text{Anbaugebiet}, \text{Region}\} \\f_6 &= \{\text{Anbaugebiet} \rightarrow \text{Region}\}\end{aligned}$$

Synthese: Beispiel /2

- Ablauf

1. minimale Überdeckung: Entfernen von f_2 , f_3 sowie Region in f_5
2. Äquivalenzklassen:

$$C_1 = \{\text{Name}, \text{Weingut} \rightarrow \text{Preis}\}$$

$$C_2 = \{\text{Name} \rightarrow \text{Farbe}\}$$

$$C_3 = \{\text{Weingut} \rightarrow \text{Anbaugebiet}\}$$

$$C_4 = \{\text{Anbaugebiet} \rightarrow \text{Region}\}$$

3. Ableitung der Relationenschemata

Zusammenfassung

- Formalisierung des Relationenmodells und der funktionalen Abhängigkeiten
- Algorithmen zur Normalisierung

Kontrollfragen

- Was muß beim Syntheseverfahren beachtet werden, um Spezialfälle wie zyklische Abhängigkeiten oder fehlende Schlüssel zu berücksichtigen?



Teil VII

Die relationale Anfragesprache SQL

Die relationale Anfragesprache SQL



1. Aufbau von SQL-Anfragen
2. Erweiterungen des SFW-Blocks
3. Aggregatfunktionen und Gruppierungen

Lernziele für heute . . .

- Erweiterte Kenntnisse zum relationalen SQL
- Kenntnisse von Erweiterungen des SFW-Blocks



Aufbau von SQL-Anfragen

Struktur einer SQL-Anfrage

```
-- Anfrage
select projektionsliste
from relationenliste
[ where bedingung ]
```

select

- Projektionsliste
- arithmetische Operationen und Aggregatfunktionen

from

- zu verwendende Relationen, evtl. Umbenennungen

where

- Selektions-, Verbundbedingungen
- Geschachtelte Anfragen (wieder ein SFW-Block)

Auswahl von Tabellen: Die from-Klausel



- einfachste Form; hinter jedem Relationennamen kann optional eine Tupelvariable stehen

```
select *  
from relationenliste
```

- Beispielanfrage:

```
select *  
from WEINE
```

Die select-Klausel

- Festlegung der **Projektionsattribute**

```
select [distinct] projektionsliste
from ...
```

- mit

```
projektionsliste := {attribut |
arithmetischer-ausdruck |
aggregat-funktion } [, ...]
```

Die select-Klausel: Projektionsliste

- Attribute der hinter `from` stehenden Relationen, optional mit Präfix, der Relationennamen oder Namen der Tupelvariablen angibt
- arithmetische Ausdrücke über Attributen dieser Relationen und passenden Konstanten
- Aggregatfunktionen über Attributen dieser Relationen

Die select-Klausel

- Spezialfall der Projektionsliste: *
- liefert alle Attribute der Relation(en) aus dem `from`-Teil

```
select *
from WEINE
```

`distinct` eliminiert Duplikate

```
select Name from WEINE
```

- liefert die Ergebnisrelation als Multimenge:

Name
La Rose Grand Cru
Creek Shiraz
Zinfandel
Pinot Noir
Pinot Noir
Riesling Reserve
Chardonnay

distinct eliminiert Duplikate /2

```
select distinct Name from WEINE
```

- ergibt Projektion aus der Relationenalgebra:

Name
La Rose Grand Cru
Creek Shiraz
Zinfandel
Pinot Noir
Riesling Reserve
Chardonnay

Tupelvariablen und Relationennamen



- Anfrage

```
select Name from WEINE
```

- ist äquivalent zu

```
select WEINE.Name from WEINE
```

- und

```
select W.Name from WEINE W
```

Kartesisches Produkt

- bei mehr als einer Relation wird das kartesische Produkt gebildet:

```
select *
from WEINE, ERZEUGER
```

- **alle Kombinationen werden ausgegeben!**

Tupelvariablen für mehrfachen Zugriff

- Einführung von Tupelvariablen erlaubt mehrfachen Zugriff auf eine Relation:

```
select *
from WEINE w1, WEINE w2
```

- Spalten lauten dann:

```
w1.WeinID, w1.Name, w1.Farbe, w1.Jahrgang,
w1.Weingut,
w2.WeinID, w2.Name, w2.Farbe, w2.Jahrgang,
w2.Weingut
```

- frühe SQL-Versionen
 - üblicherweise realisierter Standard in aktuellen Systemen
 - kennen nur Kreuzprodukt, keinen expliziten Verbundoperator
 - Verbund durch Prädikat hinter `where` realisieren
- Beispiel für natürlichen Verbund:

```
select *
from WEINE, ERZEUGER
where WEINE.Weingut = ERZEUGER.Weingut
```

Verbund explizit: natural join

- neuere SQL-Versionen
 - kennen mehrere explizite Verbundoperatoren (engl. *join*)
 - als Abkürzung für die ausführliche Anfrage mit Kreuzprodukt aufzufassen

```
select *  
from WEINE natural join ERZEUGER
```

Verbunde als explizite Operatoren: join



- Verbund mit beliebigem Prädikat:

```
select *
from WEINE join ERZEUGER
    on WEINE.Weingut = ERZEUGER.Weingut
```

- Gleichverbund mit using:

```
select *
from WEINE join ERZEUGER
    using (Weingut)
```

Verbund explizit: cross join

- Kreuzprodukt

```
select *
from WEINE, ERZEUGER
```

- als cross join

```
select *
from WEINE cross join ERZEUGER
```

Tupelvariable für Zwischenergebnisse

- „Zwischenrelationen“ aus SQL-Operationen oder einem SFW-Block können über Tupelvariablen mit Namen versehen werden

```
select Ergebnis.Weingut
  from (WEINE natural join ERZEUGER) as Ergebnis
```

- für `from` sind Tupelvariablen Pflicht
- `as` ist optional

Präfixe für Eindeutigkeit

```
select Name, Jahrgang, Weingut      -- (falsch!)\nfrom WEINE natural join ERZEUGER
```

- Attribut Weingut existiert sowohl in der Tabelle WEINE als auch in ERZEUGER!
- richtig mit Präfix:

```
select Name, Jahrgang, ERZEUGER.Weingut\nfrom WEINE natural join ERZEUGER
```

Tupelvariablen für Eindeutigkeit

- bei der Verwendung von Tupelvariablen, kann der Name einer Tupelvariablen zur Qualifizierung eines Attributs benutzt werden:

```
select w1.Name, w2.Weingut
from WEINE w1, WEINE w2
```

```
select ...from ...
where bedingung
```

- Formen der Bedingung:
 - Vergleich eines Attributs mit einer Konstanten:

attribut θ *konstante*

mögliche Vergleichssymbole θ abhängig vom Wertebereich; etwa $=$,
 $<>$, $>$, $<$, \geq sowie \leq .

- Vergleich zwischen zwei Attributen mit kompatiblen Wertebereichen:

attribut1 θ *attribut2*

- logische *Konnektoren* or, and und not

Verbundbedingung

- *Verbundbedingung* hat die Form:

```
relation1.attribut = relation2.attribut
```

- Beispiel:

```
select Name, Jahrgang, ERZEUGER.Weingut
from WEINE, ERZEUGER
where WEINE.Weingut = ERZEUGER.Weingut
```

- *Bereichsselektion*

attrib between konstante₁ and konstante₂

ist Abkürzung für

*attrib ≥ konstante₁ and
attrib ≤ konstante₂*

- schränkt damit Attributwerte auf das abgeschlossene Intervall [*konstante₁*, *konstante₂*] ein
- Beispiel:

```
select * from WEINE
where Jahrgang between 2000 and 2005
```

- Notation

attribut like spezialkonstante

- Mustererkennung in Strings (Suche nach mehreren Teilzeichenketten)
- Spezialkonstante kann die Sondersymbole '%' und '_' beinhalten
 - '%' steht für kein oder beliebig viele Zeichen
 - '_' steht für genau ein Zeichen

Ungewissheitsselektion /2

```
select * from WEINE  
where Name like 'La Rose%'
```

ist Abkürzung für

```
select * from WEINE  
where Name = 'La Rose'  
      or Name = 'La RoseA' or Name = 'La RoseAA' ...  
      or Name = 'La RoseB' or Name = 'La RoseBB' ...  
      ...  
      or Name = 'La Rose Grand Cru' ...  
      or Name = 'La Rose Grand Cru Classe' ...  
      ...  
      or Name = 'La RoseZZZZZZZZZZZZZ' ...
```

- Mengenoperationen erfordern kompatible Wertebereiche für Paare korrespondierender Attribute:
 - beide Wertebereiche sind gleich oder
 - beide sind auf character basierende Wertebereiche (unabhängig von der Länge der Strings) oder
 - beide sind numerische Wertebereiche (unabhängig von dem genauen Typ) wie integer oder float
- Ergebnisschema := Schema der „linken“ Relation

```
select A, B, C from R1
union
select A, C, D from R2
```

Mengenoperationen in SQL

- Vereinigung, Durchschnitt und Differenz als union, intersect und except
- orthogonal einsetzbar:

```
select *  
from (select Weingut from ERZEUGER  
      except select Weingut from WEINE)
```

äquivalent zu

```
select *  
from ERZEUGER except corresponding WEINE
```

Mengenoperationen in SQL /2

- über corresponding by-Klausel: Angabe der Attributliste, über die Mengenoperation ausgeführt wird

```
select *  
from ERZEUGER except corresponding by (Weingut)  
WEINE
```

- bei Vereinigung: Defaultfall ist Duplikateliminierung (`union distinct`); **ohne** Duplikateliminierung durch `union all`

Mengenoperationen in SQL /2

R	A	B	C
1	2	3	
2	3	4	

S	A	C	D
2	3	4	
2	4	5	

R union S	A	B	C
1	2	3	
2	3	4	
2	4	5	

R union all S	A	B	C
1	2	3	
2	3	4	
2	3	4	
2	4	5	

Mengenoperationen in SQL /3

R	A	B	C
1	2	3	
2	3	4	

S	A	C	D
2	3	4	
2	4	5	

R union corresponding S

A	C
1	3
2	4
2	3

R union corresponding by (A) S

A
1
2

Schachtelung von Anfragen

- für Vergleiche mit Wertemengen notwendig:
 - Standardvergleiche in Verbindung mit den Quantoren all (\forall) oder any (\exists)
 - spezielle Prädikate für den Zugriff auf Mengen, in und exists

- Notation:

```
attribut in ( SFW-block )
```

- Beispiel:

```
select Name
from WEINE
where Weingut in (
    select Weingut from ERZEUGER
    where Region = 'Bordeaux')
```

Auswertung von geschachtelten Anfragen



1. Auswertung der inneren Anfrage zu den Weingütern aus Bordeaux
2. Einsetzen des Ergebnisses als Menge von Konstanten in die äußere Anfrage hinter in
3. Auswertung der modifizierten Anfrage

```
select Name from WEINE  
where Weingut in (  
    'Château La Rose', 'Château La Pointe')
```

Name
La Rose Grand Cru

Auswertung von geschachtelten Anfragen /2



- interne Auswertung: Umformung in einen Verbund

```
select Name  
from WEINE natural join ERZEUGER  
where Region = 'Bordeaux'
```

Negation des in-Prädikats

- Simulation des Differenzoperators

$$\pi_{\text{Weingut}}(\text{ERZEUGER}) - \pi_{\text{Weingut}}(\text{WEINE})$$

durch SQL-Anfrage

```
select Weingut
from ERZEUGER
where Weingut not in (
    select Weingut from WEINE )
```

Mächtigkeit des SQL-Kerns

Relationenalgebra	SQL
Projektion	<code>select distinct</code>
Selektion	<code>where</code> ohne Schachtelung
Verbund	<code>from, where</code> <code>from mit join oder natural join</code>
Umbenennung	<code>from mit Tupelvariable; as</code>
Differenz	<code>where mit Schachtelung</code> <code>except corresponding</code>
Durchschnitt	<code>where mit Schachtelung</code> <code>intersect corresponding</code>
Vereinigung	<code>union corresponding</code>

Erweiterungen des SFW-Blocks

Weiteres zu SQL

- Erweiterungen des SFW-Blocks
 - innerhalb der `from`-Klausel weitere Verbundoperationen (äußerer Verbund),
 - innerhalb der `where`-Klausel weitere Arten von Bedingungen und Bedingungen mit Quantoren,
 - innerhalb der `select`-Klausel die Anwendung von skalaren Operationen und Aggregatfunktionen,
 - zusätzliche Klauseln `group by` und `having`
- rekursive Anfragen

- Umbenennung von Spalten: `ausdruck` as `neuer-name`
- skalare Operationen auf
 - numerischen Wertebereichen: etwa `+`, `-`, `*` und `/`,
 - Strings: Operationen wie `char_length` (aktuelle Länge eines Strings), die Konkatenation `||` und die Operation `substring` (Suchen einer Teilzeichenkette an bestimmten Positionen des Strings),
 - Datumstypen und Zeitintervallen: Operationen wie `current_date` (aktuelles Datum), `current_time` (aktuelle Zeit), `+`, `-` und `*`
- bedingte Ausdrücke
- Typkonvertierung

Skalare Ausdrücke: Hinweise

- skalare Ausdrücke können mehrere Attribute umfassen
- Anwendung ist tupelweise: pro Eingabetupel entsteht ein Ergebnistupel

Skalare Ausdrücke: Beispiele

- Ausgabe der Namen aller Grand Cru-Weine

```
select substring(Name from 1 for
    (char_length(Name) -
     position('Grand Cru' in Name)))
from WEINE where Name like '%Grand Cru'
```

Skalare Ausdrücke: Beispiele /2

- Annahme: zusätzliches Attribut HerstDatum in WEINE

```
alter table WEINE add column HerstDatum date
```

```
update WEINE set HerstDatum = date '2004-08-13'  
where Name = 'Zinfandel'
```

- Anfrage:

```
select Name,  
       year(current_date - HerstDatum) as Alter  
  from WEINE
```

- case-Anweisung: Ausgabe eines Wertes in Abhängigkeit von der Auswertung eines Prädikats

```
case
    when prädikat1 then ausdruck1
    ...
    when prädikatn-1 then ausdruckn-1
    [ else ausdruckn ]
end
```

Bedingte Ausdrücke: Beispiele

- Einsatz in select- und where-Klausel

```
select case
    when Farbe = 'Rot' then 'Rotwein'
    when Farbe = 'Weiß' then 'Weißwein'
    else 'Sonstiges'
end as Weinart, Name from WEINE
```

Typkonvertierung

- explizite Konvertierung des Typs von Ausdrücken

```
cast(ausdruck as typename)
```

- Beispiel: int-Werte als Zeichenkette für Konkatenationsoperator

```
select cast(Jahrgang as varchar) || 'er ' ||  
       Name as Bezeichnung  
  from WEINE
```

- Quantoren: all, any, some und exists
- Notation

```
attribut θ { all | any | some } (
    select attribut
    from ...where ...)
```

- all: where-Bedingung wird erfüllt, wenn für **alle** Tupel des inneren SFW-Blocks der θ -Vergleich mit *attribut* true wird
- any bzw. some: where-Bedingung wird erfüllt, wenn der θ -Vergleich mit mindestens einem Tupel des inneren SFW-Blocks true wird

Bedingungen mit Quantoren: Beispiele

- Bestimmung des ältesten Weines

```
select *  
from WEINE  
where Jahrgang <= all (  
    select Jahrgang from WEINE)
```

- alle Weingüter, die Rotweine produzieren

```
select *  
from ERZEUGER  
where Weingut = any (  
    select Weingut from WEINE  
    where Farbe = 'Rot')
```

Vergleich von Wertemengen

- Test auf Gleichheit zweier Mengen allein mit Quantoren nicht möglich
- Beispiel: „Gib alle Erzeuger aus, die sowohl Rot- als auch Weißweine produzieren.“
- falsche Anfrage

```
select Weingut
from WEINE
where Farbe = 'Rot' and Farbe = 'Weiß'
```

Vergleich von Wertemengen /2

- richtige Formulierung

```
select w1.Weingut
from WEINE w1, WEINE w2
where w1.Weingut = w2.Weingut
      and w1.Farbe = 'Rot' and w2.Farbe = 'Weiß'
```

Das exists/not exists-Prädikat

- einfache Form der Schachtelung

```
exists ( SFW-block )
```

- liefert true, wenn das Ergebnis der inneren Anfrage **nicht** leer ist
- speziell bei **verzahnt geschachtelten** (korrelierte) Anfragen sinnvoll
 - in der inneren Anfrage wird Relationen- oder Tupelvariablen-Name aus dem from-Teil der äußeren Anfrage verwendet

Verzahnt geschachtelte Anfragen

- Weingüter mit 1999er Rotwein

```
select * from ERZEUGER
where 1999 in (
    select Jahrgang from WEINE
    where Farbe = 'Rot' and
        WEINE.Weingut = ERZEUGER.Weingut)
```

Verzahnt geschachtelte Anfragen: konzeptionelle Auswertung

1. Untersuchung des ersten ERZEUGER-Tupels in der äußeren Anfrage (Creek) und Einsetzen in innere Anfrage
2. Auswertung der inneren Anfrage

```
select Jahrgang from WEINE  
where Farbe='Rot' and WEINE.Weingut = 'Creek'
```

3. Weiter bei 1. mit zweitem Tupel ...

Alternative: Umformulierung in Verbund

Beispiel für exists

- Weingüter aus Bordeaux ohne gespeicherte Weine

```
select * from ERZEUGER e
where Region = 'Bordeaux' and not exists (
    select * from WEINE
    where Weingut = e.Weingut)
```

Aggregatfunktionen und Gruppierungen

Aggregatfunktionen und Gruppierung

- Aggregatfunktionen berechnen neue Werte für eine gesamte Spalte, etwa die Summe oder den Durchschnitt der Werte einer Spalte
- Beispiel: Ermittlung des Durchschnittspreises aller Artikel oder des Gesamtumsatzes über alle verkauften Produkte
- bei zusätzlicher Anwendung von Gruppierung: Berechnung der Funktionen pro Gruppe, z.B. der Durchschnittspreis pro Warengruppe oder der Gesamtumsatz pro Kunde

Aggregatfunktionen

- Aggregatfunktionen in Standard-SQL:
 - `count`: berechnet Anzahl der Werte einer Spalte oder alternativ (im Spezialfall `count(*)`) die Anzahl der Tupel einer Relation
 - `sum`: berechnet die Summe der Werte einer Spalte (nur bei numerischen Wertebereichen)
 - `avg`: berechnet den arithmetischen Mittelwert der Werte einer Spalte (nur bei numerischen Wertebereichen)
 - `max` bzw. `min`: berechnen den größten bzw. kleinsten Wert einer Spalte

Aggregatfunktionen /2

- Argumente einer Aggregatfunktion:
 - ein Attribut der durch die `from`-Klausel spezifizierten Relation,
 - ein gültiger skalarer Ausdruck oder
 - im Falle der `count`-Funktion auch das Symbol *

- vor dem Argument (außer im Fall von `count(*)`) optional auch die Schlüsselwörter `distinct` oder `all`
 - `distinct`: vor Anwendung der Aggregatfunktion werden doppelte Werte aus der Menge von Werten, auf die die Funktion angewendet wird
 - `all`: Duplikate gehen mit in die Berechnung ein (Default-Voreinstellung)
 - Nullwerte werden in jedem Fall vor Anwendung der Funktion aus der Wertemenge eliminiert (außer im Fall von `count(*)`)

Aggregatfunktionen - Beispiele

- Anzahl der Weine:

```
select count(*) as Anzahl  
from WEINE
```

ergibt

Anzahl
7

Aggregatfunktionen - Beispiele /2

- Anzahl der **verschiedenen** Weinregionen:

```
select count(distinct Region)
from ERZEUGER
```

- Weine, die älter als der Durchschnitt sind:

```
select Name, Jahrgang
from WEINE
where Jahrgang < (
    select avg(Jahrgang) from WEINE)
```

Aggregatfunktionen /2

- Schachtelung von Aggregatfunktionen nicht erlaubt

```
select f1(f2(A)) as Ergebnis  
from R ... -- (falsch!)
```

- mögliche Formulierung:

```
select f1(Temp) as Ergebnis  
from ( select f2(A) as Temp from R ... )
```

Aggregatfunktionen in where-Klausel



- Aggregatfunktionen liefern nur einen Wert \rightsquigarrow Einsatz in Konstanten-Selektionen der where-Klausel möglich
- alle Weingüter, die nur einen Wein liefern:

```
select * from ERZEUGER e
where 1 = (
    select count(*) from WEINE w
    where w.Weingut = e.Weingut)
```

group by und having

- Notation

```
select ...
from ...
[where ...]
[group by attributliste ]
[having bedingung ]
```

Gruppierung: Schema

- Relation REL:

A	B	C	D
1	2	3	4
1	2	4	5
2	3	3	4
3	3	4	5
3	3	6	7
...			

- Anfrage:

```
select A, sum(D) from REL where ...
group by A, B
having A<4 and sum(D)<10 and max(C)=4
```

Gruppierung: Schritt 1

- from und where

A	B	C	D
1	2	3	4
1	2	4	5
2	3	3	4
3	3	4	5
3	3	6	7
...			



A	B	C	D
1	2	3	4
1	2	4	5
2	3	3	4
3	3	4	5
3	3	6	7

Gruppierung: Schritt 2

- group by A, B

A	B	C	D
1	2	3	4
1	2	4	5
2	3	3	4
3	3	4	5
3	3	6	7



A	B	N	
		C	D
1	2	3	4
		4	5
2	3	3	4
		4	5
3	3	6	7

Gruppierung: Schritt 3

- select A, sum(D)

A	B	N	
		C	D
1	2	3	4
		4	5
2	3	3	4
3	3	4	5
		6	7



A	sum(D)	N	
		C	D
1	9	3	4
		4	5
2	4	3	4
3	12	4	5
		6	7

Gruppierung: Schritt 4

- having $A < 4$ and $\text{sum}(D) < 10$ and $\max(C) = 4$

A	$\text{sum}(D)$	N	
		C	D
1	9	3	4
		4	5
2	4	3	4
3	12	4	5
		6	7



A	$\text{sum}(D)$
1	9

Gruppierung - Beispiel

- Anzahl der Rot- und Weißweine:

```
select Farbe, count(*) as Anzahl
from WEINE
group by Farbe
```

- Ergebnisrelation:

Farbe	Anzahl
Rot	5
Weiß	2

having - Beispiel

- Regionen mit mehr als einem Wein

```
select Region, count(*) as Anzahl
from ERZEUGER natural join WEINE
group by Region
having count(*) > 1
```

Attribute für Aggregation bzw. having

- zulässige Attribute hinter select bei Gruppierung auf Relation mit Schema R
 - Gruppierungsattribute G
 - Aggregationen auf Nicht-Gruppierungsattributen $R - G$
- zulässige Attribute für having
 - ditto

- zusätzlich zu klassischen Verbund (`inner join`): in SQL-92 auch äußerer Verbund \rightsquigarrow Übernahme von „dangling tuples“ in das Ergebnis und Auffüllen mit Nullwerten
- `outer join` übernimmt alle Tupel beider Operanden (Langfassung: `full outer join`)
- `left outer join` bzw. `right outer join` übernimmt alle Tupel des linken bzw. des rechten Operanden
- äußerer natürlicher Verbund jeweils mit Schlüsselwort `natural`, also z.B. `natural left outer join`

Äußere Verbunde /2

LINKS

A	B
1	2
2	3

RECHTS

B	C
3	4
4	5

NATURAL JOIN

A	B	C
2	3	4

OUTER

A	B	C
1	2	⊥
2	3	4
⊥	4	5

LEFT

A	B	C
1	2	⊥
2	3	4

RIGHT

A	B	C
2	3	4
⊥	4	5

Äußerer Verbund: Beispiel

```
select Anbaugebiet, count(WeinID) as Anzahl
from ERZEUGER natural left outer join WEINE
group by Anbaugebiet
```

Anbaugebiet	Anzahl
Barossa Valley	2
Napa Valley	3
Saint-Emilion	1
Pomerol	0
Rheingau	1

Simulation des (linken) äußeren Verbundes



```
select *
from ERZEUGER natural join WEINE
      union all
select ERZEUGER.* , cast(null as int),
       cast(null as varchar(20)),
       cast(null as varchar(10)),
       cast(null as int),
       cast(null as varchar(20))
from ERZEUGER e
where not exists (
      select * from WEINE
      where WEINE.Weingut = e.Weingut)
```

Sortierung mit order by

- Notation

```
order by attributliste
```

- Beispiel:

```
select *  
from WEINE  
order by Jahrgang
```

- Sortierung aufsteigend (asc) oder absteigend (desc)
- Sortierung als letzte Operation einer Anfrage \rightsquigarrow **Sortierattribut muss in der select-Klausel vorkommen**

Sortierung /2

- Sortierung auch mit berechneten Attributen (Aggregaten) als Sortierkriterium

```
select Weingut, count(*) as Anzahl
from ERZEUGER natural join WEINE
group by Weingut
order by Anzahl desc
```

Sortierung: Top-k-Anfragen

- Anfrage, die die **besten** k Elemente bzgl. einer Rangfunktion liefert

```
select w1.Name, count(*) as Rang
from WEINE w1, WEINE w2
where w1.Jahrgang <= w2.Jahrgang      -- Schritt 1
group by w1.Name, w1.WeinID           -- Schritt 2
having count(*) <= 4                  -- Schritt 3
order by Rang                         -- Schritt 4
```

Sortierung: Top-k-Anfragen

- Ermittlung der $k = 4$ jüngste Weine
- Erläuterung
 - Schritt 1: Zuordnung aller Weine die älter sind
 - Schritt 2: Gruppierung nach Namen, Berechnung des Rangs
 - Schritt 3: Beschränkung auf Ränge ≤ 4
 - Schritt 4: Sortierung nach Rang

Name	Rang
Zinfandel	1
Creek Shiraz	2
Chardonnay	3
Pinot Noir	4

Behandlung von Nullwerten

- skalare Ausdrücke: Ergebnis null, sobald Nullwert in die Berechnung eingeht
- in allen Aggregatfunktionen bis auf count(*) werden Nullwerte vor Anwendung der Funktion entfernt
- fast alle Vergleiche mit Nullwert ergeben Wahrheitswert unknown (statt true oder false)
- Ausnahme: is null ergibt true, is not null ergibt false
- Boolesche Ausdrücke basieren dann auf dreiwertiger Logik

Behandlung von Nullwerten /2

and	true	unknown	false
true	true	unknown	false
unknown	unknown	unknown	false
false	false	false	false

or	true	unknown	false
true	true	true	true
unknown	true	unknown	unknown
false	true	unknown	false

not	
true	false
unknown	unknown
false	true

- **Null-Selektion** wählt Tupel aus, die bei einem bestimmten Attribut Nullwerte enthalten
- Notation

attribut is null

- Beispiel

```
select * from ERZEUGER  
where Anbaugebiet is null
```

Zusammenfassung

- SQL als Standardsprache
- SQL-Kern mit Bezug zur Relationenalgebra
- Erweiterungen: Gruppierung etc.

Kontrollfragen

- Welche Möglichkeiten der Formulierung von Verbunden gibt es?
- Was berechnen Aggregationen und Gruppierungen?
- Welche Operationen stehen für den Umgang mit Nullwerten zur Verfügung?



Teil VIII

Fortgeschrittene Konzepte in SQL

Fortgeschrittene Konzepte in SQL



1. Rekursion in SQL
2. Fortgeschrittenes SQL
3. Prozedurale SQL-Erweiterungen: SQL/PSM

Lernziele für heute . . .

- Verständnis der Formulierung und Auswertung rekursiver Anfragen
- Beispiele komplexer SQL-Anfragen
- Prozedurale SQL-Erweiterungen: SQL/PSM



Rekursion in SQL

Benannte Anfragen

- Anfrageausdruck, der in der Anfrage mehrfach referenziert werden kann

```
with anfrage-name [(spalten-liste) ] as  
( anfrage-ausdruck )
```

- Anfrage ohne with

```
select *  
from WEINE  
where Jahrgang >= (  
    select avg(Jahrgang) from WEINE) - 2  
    and Jahrgang <= (  
        select avg(Jahrgang) from WEINE) + 2
```

Benannte Anfragen /2

- Anfrage mit with

```
with ALTER(Durchschnitt) as (
    select avg(Jahrgang) from WEINE)
select *
from WEINE, ALTER
where Jahrgang >= Durchschnitt - 2
and Jahrgang <= Durchschnitt + 2
```

Rekursive Anfragen

- Anwendung: **Bill of Material-Anfragen**, Berechnung der **transitiven Hülle** (Flugverbindungen etc.)
- Beispiel:

BUSLINIE	Abfahrt	Ankunft	Distanz
	Nuriootpa	Penrice	7
	Nuriootpa	Tanunda	7
	Tanunda	Seppeltsfield	9
	Tanunda	Bethany	4
	Bethany	Lyndoch	14

Rekursive Anfrage: Busfahrt mit max. 2x Umsteigen

```
select Abfahrt, Ankunft from BUSLINIE
where Abfahrt = 'Nuriootpa'
    union
select B1.Abfahrt, B2.Ankunft
from BUSLINIE B1, BUSLINIE B2
where B1.Abfahrt = 'Nuriootpa'
    and B1.Ankunft = B2.Abfahrt
    union
select B1.Abfahrt, B3.Ankunft
from BUSLINIE B1, BUSLINIE B2, BUSLINIE B3
where B1.Abfahrt = 'Nuriootpa'
    and B1.Ankunft = B2.Abfahrt
    and B2.Ankunft = B3.Abfahrt
```

Rekursion in SQL:2003

- Formulierung über erweiterte with recursive-Anfrage
- Notation

```
with recursive rekursive-tabelle as (
    anfrage-ausdruck -- rekursiver Teil
)
[traversierungsklausel] [zyklusklausel]
anfrage-ausdruck -- nicht rekursiver Teil
```

- nicht rekursiver Teil: Anfrage auf Rekursionstabelle

Rekursion in SQL:2003 /2



- rekursiver Teil:

```
-- Initialisierung
select ...
from tabelle where ...
-- Rekursionsschritt
union all
select ...
from tabelle, rekursionstabelle
where rekursionsbedingung
```

Rekursion in SQL:2003: Beispiel

```
with recursive TOUR(Abfahrt, Ankunft) as (
    select Abfahrt, Ankunft
    from BUSLINIE
    where Abfahrt = 'Nuriootpa'
        union all
    select T.Abfahrt, B.Ankunft
    from TOUR T, BUSLINIE B
    where T.Ankunft = B.Abfahrt)
select distinct * from TOUR
```

Schrittweiser Aufbau der Rekursionstabelle TOUR

Initialisierung

Abfahrt	Ankunft
Nuriootpa	Penrice
Nuriootpa	Tanunda

Schritt 1

Abfahrt	Ankunft
Nuriootpa	Penrice
Nuriootpa	Tanunda
Nuriootpa	Seppeltsfield
Nuriootpa	Bethany

Schritt 2

Abfahrt	Ankunft
Nuriootpa	Penrice
Nuriootpa	Tanunda
Nuriootpa	Seppeltsfield
Nuriootpa	Bethany
Nuriootpa	Lyndoch

Rekursion: Beispiel /2

- arithmetische Operationen im Rekursionsschritt

```
with recursive TOUR(Abfahrt, Ankunft, Strecke) as (
    select Abfahrt, Ankunft, Distanz as Strecke
    from BUSLINIE
    where Abfahrt = 'Nuriootpa'
        union all
    select T.Abfahrt, B.Ankunft,
        Strecke + Distanz as Strecke
    from TOUR T, BUSLINIE B
    where T.Ankunft = B.Abfahrt)
select distinct * from TOUR
```

- Sicherheit (= Endlichkeit der Berechnung) ist wichtige Anforderung an Anfragesprache
- Problem: Zyklen bei Rekursion

```
insert into BUSLINIE (Abfahrt, Ankunft, Distanz)
    values ('Lyndoch', 'Tanunda', 12)
```

- Behandlung in SQL
 - Begrenzung der Rekursionstiefe
 - Zyklenerkennung

- Einschränkung der Rekursionstiefe

```
with recursive TOUR(Abfahrt, Ankunft, Umsteigen) as (
    select Abfahrt, Ankunft, 0
    from BUSLINIE
    where Abfahrt = 'Nuriootpa'
        union all
    select T.Abfahrt, B.Ankunft, Umsteigen + 1
    from TOUR T, BUSLINIE B
    where T.Ankunft = B.Abfahrt and Umsteigen < 2)
select distinct * from TOUR
```

Sicherheit durch Zyklenerkennung

- Zyklusklausel
 - beim Erkennen von Duplikaten im Berechnungspfad von *attrib*:
Zyklus = '*' (Pseudospalte vom Typ `char(1)`)
 - Sicherstellen der Endlichkeit des Ergebnisses „von Hand“

```
cycle attrib set marke to '*' default '-'
```

Sicherheit durch Zyklenerkennung



```
with recursive TOUR(Abfahrt, Ankunft, Weg) as (
    select Abfahrt, Ankunft,
        Abfahrt || '-' || Ankunft as Weg
    from BUSLINIE where Abfahrt = 'Nuriootpa'
        union all
    select T.Abfahrt, B.Ankunft,
        Weg || '-' || B.Ankunft as Weg
    from TOUR T, BUSLINIE B
    where T.Ankunft = B.Abfahrt)
cycle Ankunft set Zyklus to '*', default '-'
select Weg, Zyklus from TOUR
```

Sicherheit durch Zyklenerkennung /2

Weg	Zyklus
Nuriootpa-Penrice	-
Nuriootpa-Tanunda	-
Nuriootpa-Tanunda-Seppeltsfield	-
Nuriootpa-Tanunda-Bethany	-
Nuriootpa-Tanunda-Bethany-Lyndoch	-
Nuriootpa-Tanunda-Bethany-Lyndoch-Tanunda	*

- Geschichte
 - SEQUEL (1974, IBM Research Labs San Jose)
 - SEQUEL2 (1976, IBM Research Labs San Jose)
 - SQL (1982, IBM)
 - ANSI-SQL (SQL-86; 1986)
 - ISO-SQL (SQL-89; 1989; drei Sprachen Level 1, Level 2, + IEF)
 - (ANSI / ISO) SQL2 (als SQL-92 verabschiedet)
 - (ANSI / ISO) SQL3 (als SQL:1999 verabschiedet)
 - (ANSI / ISO) SQL:2003 ... aktuell SQL:2011
- trotz Standardisierung: teilweise Inkompatibilitäten zwischen Systemen der einzelnen Hersteller

Fortgeschrittenes SQL

Fortgeschrittenes SQL

- SQL ist weitaus mächtiger als SFW
- Unterstützung von komplexen Transformationen, Manipulationen und Analysen auch extrem großer Datenbestände
- Turing-Vollständigkeit durch prozedurale Erweiterungen (SQL/PSM, PL/SQL, Transact-SQL, ...)
- nachfolgend: erweiterte SQL-Konstrukte mit Beispielen als Muster für Problemlösungen

Pivotierung

Problem: Zeilen in Spalten bzw. umgekehrt umwandeln

LDATA	Quartal	Jahr	Umsatz	↔	RDATA	Jahr	Q1	Q2	Q3	Q4
	1	2020	12			2020	12	10	11	9
	2	2020	10			2021	13	12	⊥	⊥
	3	2020	11							
	4	2020	9							
	1	2021	13							
	2	2021	12							

Lösung (Zeilen in Spalten): Aggregatfilter

```
select Jahr, sum(Umsatz) filter (where Quartal=1) Q1,  
      sum(Umsatz) filter (where Quartal=2) Q2,  
      sum(Umsatz) filter (where Quartal=3) Q3,  
      sum(Umsatz) filter (where Quartal=4) Q4  
from LDATA  
group by Jahr
```

- Warum `sum` und `group by`?
- `filter` nicht in allen Systemen verfügbar \rightsquigarrow Alternative über `case`

Lösung (Spalten in Zeilen): union all

```
select 1 as Quartal, Jahr, Q1 as Umsatz from RDATA
    union all
select 2 as Quartal, Jahr, Q2 as Umsatz from RDATA
    union all
select 3 as Quartal, Jahr, Q3 as Umsatz from RDATA
    union all
select 4 as Quartal, Jahr, Q4 as Umsatz from RDATA
```

- Warum union all statt union?

Telefonverzeichnis

Problem: Pivotierung mit fehlenden Werten → Ausgabe der Telefonnummern zu einer Person in einer Zeile

Personal			Kontakt		
PNr	Vorname	Name	PNr	KTyp	Nummer
101	Bernd	K.	101	Mobil	0175...
102	Simone	S.	101	Arbeit	5556
103	Franz	M.	102	Mobil	0165...
			103	Arbeit	5557

- zweifacher Verbund mit Kontakt?

Lösung: linker äußerer Verbund

```
select Vorname, Name, K.Nummer as Mobil, K2.Nummer as Arbeit
from (Personal P left outer join Kontakt K
      on P.PNr = K.PNr and K.KTyp = 'Mobil')
      left outer join Kontakt K2
      on P.PNr = K2.PNr and K.KTyp = 'Arbeit'
```

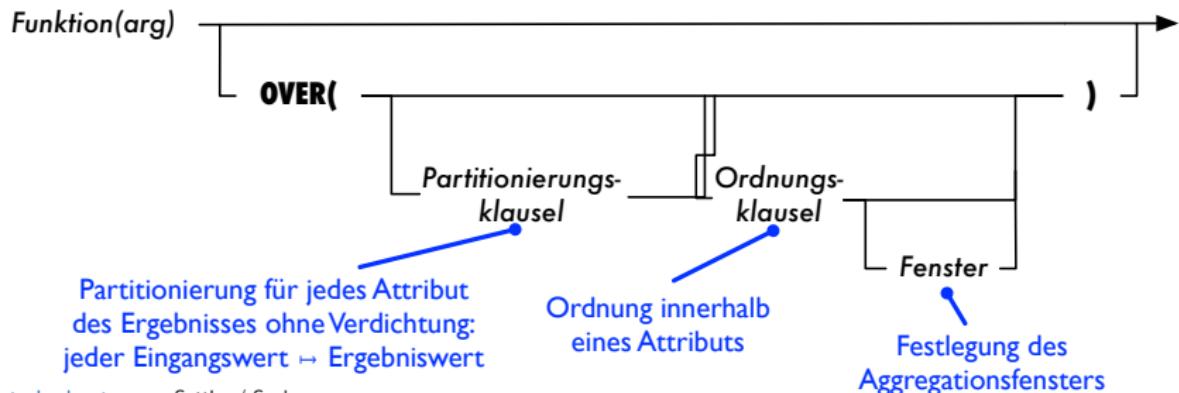
Problem: Beschränkung der Ergebnismenge auf k Elemente, z.B. nach Sortierung

RACE	Name	Distanz	Zeit
	Klaus	HM	1:20
	Bernd	HM	1:40
	Tanja	M	3:58
	Franz	SM	5:42
	Martina	M	3:05
	Heike	HM	1:34
	Corinna	SM	5:53
	Jens	M	2:51
	Herbert	SM	6:07

Lösungen:

- proprietäre Erweiterungen wie z.B. `limit`
- Window-Funktionen in SQL + `rank()`

Window-Funktionen: Syntax



Window-Funktionen: Prinzip

- Anzahl der Tupel, die in ein Ergebnistupel eingehen entspricht Position des Tupels bzgl. gegebener Ordnung
- Eingangstupel t_i , Ergebnistupel s_i

$$\begin{array}{rcl} t_1 & \longrightarrow & \text{sum}(\{t_1\}) \\ t_2 & \longrightarrow & \text{sum}(\{t_1, t_2\}) \\ t_3 & \longrightarrow & \text{sum}(\{t_1, t_2, t_3\}) \\ & \dots & \end{array} \quad \longrightarrow \quad \begin{array}{l} s_1 \\ s_2 \\ s_3 \end{array}$$

- Schrittweise Vergrößerung des Analyse**fensters**

Window-Funktionen: Anwendung

Was liefert ...

```
select count(*) over()  
from RACE
```

count
9
9
9
...

und ...

```
select count(*) over(order by Zeit)  
from RACE
```

count
1
2
3
...
9

Ranking-Funktionen

- `rank()`: liefert Rang eines Tupels bzgl. vorgegebener Ordnung innerhalb der Partition
 - Bei Duplikaten gleicher Rang (mit Lücken)
- `dense_rank()`: wie `rank()`, jedoch ohne Lücken

```
select Name, Zeit, rank() over (order by Zeit)
from RACE
where Distanz = 'M'
```

Name	Zeit	rank
Jens	2:51	1
Martina	3:05	2
Tanja	3:58	3

Top-k mit Ranking-Funktion

```
select * from (
    select Name, Zeit, rank() over (order by Zeit) Rang
    from RACE
    where Distanz = 'M') T
where Rang <= 2
```

- Schachtelung erforderlich, weil Projektion **nach** Selektion ausgeführt wird: Rang in der where-Klausel der inneren Anfrage nicht verfügbar
- Top-2: Rang <= 2

Top-k mit Ranking-Funktion /2

- Top-2 pro Distanz durch Gruppierung **in** der Window-Funktion über `partition by`

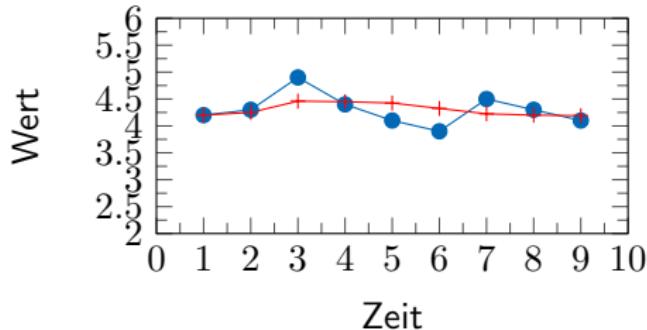
```
select * from (
    select Name, Distanz, Zeit,
           rank() over (partition by Distanz order by Zeit) Rang
      from RACE) T
 where Rang <= 2
```

Name	Distanz	Zeit	Rang
Klaus	HM	1:20	1
Heike	HM	1:34	2
Jens	M	2:51	1
Martina	M	3:05	2
Franz	SM	5:42	1
Corinna	SM	5:53	2

Gleitender Durchschnitt

Problem: Glättung von Daten- bzw. Zeitreihen \rightsquigarrow Berechnung des Mittelwertes über ein Fenster (=Ausschnitt) der Datenwerte

SERIES	Zeit	Wert
	0:01	4.2
	0:02	4.3
	0:03	4.9
	0:04	4.4
	0:05	4.1
	0:06	3.9
	0:07	4.5
	0:08	4.3
	0:09	4.1



Lösung: Window-Funktion mit Aggregationsfenster

explizite Angabe des Fensters

- rows: Anzahl der Tupel; range: Anzahl der wertmäßig verschiedenen Tupel

ausgehend von definierten Startpunkt bis zum aktuellen Tupel

- unbounded preceding: erstes Tupel der jeweiligen Partition; n preceding: n-ter Vorgänger relativ zur aktuellen Position; current row: aktuelles Tupel (nur mit range und Duplikaten sinnvoll)

Angabe der unteren und oberen Schranken

BETWEEN *untereGrenze* AND *obereGrenze*

Spezifikation der Grenzen

- unbounded preceding, unbounded following, n preceding, n following, current row

obereGrenze muss höhere Position als *untereGrenze* spezifizieren

Gleitender Durchschnitt /2

Glättung der Datenreihe über Fenster von 3 Werten

```
select Zeit,  
       avg(Wert) over(order by Zeit asc rows 3 preceding)  
from SERIES
```

Mengengleichheit

Problem: Test auf Gleichheit von zwei Mengen

Beispiel: Welche zwei Personen haben exakt die gleichen Hobbies?

Name	Hobby
Kevin	Schach
Kevin	Musik
Corinna	Parties
Martin	Handball
Martin	Musik
Katja	Handball
Katja	Musik

Lösung: 7 verschiedene Varianten in Celko's SQL Puzzles; hier:
Mengentheorie

- $A = B \leftrightarrow A \subset B \wedge B \subset A$
- in SQL für \subset besser: $\nexists e \in A : e \notin B$ und demzufolge $A \subset B$
- über `not exists` bzw. `except`

Teilschritt: verschiedene Hobbies, z.B. für Kevin und Martina vs. Katja und Martin

```
select h3.Hobby from Hobbies as h3
where 'Kevin' = h3.Name
      except
select h4.Hobby from Hobbies as h4
where 'Martina' = h4.Name)
```

Mengengleichheit: Lösung

```
select distinct h1.Name, h2.Name
from Hobbies as h1, Hobbies as h2
where h1.Name < h2.Name -- nicht die gleiche Person
    and not exists ( -- e in h1 aber nicht in h2
        select h3.Hobby from Hobbies as h3
        where h1.Name = h3.Name
            except
        select h4.Hobby from Hobbies as h4
        where h2.Name = h4.Name)
    and not exists ( -- e in h2 aber nicht in h1
        select h5.Hobby from Hobbies as h5
        where h2.Name = h5.Name
            except
        select h6.Hobby from Hobbies as h6
        where h1.Name = h6.Name)
```

Anschlussbus

Problem: Finde die nächste Busverbindung an einer Haltestelle

Buslinie	Abfahrt	Ankunft
10	8:00	9:20
11	9:10	10:35
12	9:10	11:00
13	9:55	10:45

z.B. für aktuelle Zeit (`current_time`) oder gegebene Zeit (`time '8:00'`)

Lösung: Finde minimale Abfahrtszeit, die nach der gewünschten Zeit liegt

```
select * from Fahrplan f1
where f1.Abfahrt = (
    select min(f2.Abfahrt) from Fahrplan f2
    where f2.Abfahrt >= time '8:05')
```

mit Berechnung der Wartezeit:

```
select Buslinie, f1.Abfahrt - time '8:55'
...
```

Mode – Häufigster Wert

Problem: Bestimmung des häufigsten Wertes in einer Spalte

- Aggregatfunktion avg aber kein mode?

Name	Studiengang
Kevin	Informatik
Heike	BWL
Corinna	Mathematik
Martina	BWL
Ronny	BWL
Katharina	Informatik

Mode – Häufigster Wert

Lösung: Bestimmung der Anzahl des häufigsten Wertes + having count(*) bezogen auf diesen Wert

```
select Studiengang, count(*)
from Students
group by Studiengang
having count(*) >= all (
    select count(*)
    from Students group by Studiengang)
```

Sudoku

Problem: Lösen von Sudoku

- 81 Felder (9x9) mit Ziffern 1 ... 9
- Auffüllen mit Ziffern, so dass **in jeder der neun Zeilen, Spalten und Blöcke jede Ziffer von 1 bis 9 nur einmal auftritt**

3								
		1	9	5				
	8					6		
8			6					
4		8					1	
			2					
6				2	8			
		4	1	9			5	
					7			

Von Wikipit - Eigenes Werk, CC BY-SA 3.0, <https://commons.wikimedia.org/w/index.php?curid=19843405>

Lösung: systematisch alle Felder mit Werten von 1 bis 9 auffüllen und prüfen

Quellen: [http://technology.amis.nl/blog/6404/
oracle-rdbms-11gr2-solving-a-sudoku-using-recursive-subquery-function](http://technology.amis.nl/blog/6404/oracle-rdbms-11gr2-solving-a-sudoku-using-recursive-subquery-function)

https://wiki.postgresql.org/wiki/Sudoku_puzzle

Hilfsfunktionen (PostgreSQL)

- `generate_series(s, e)`: erzeugt eine Folge von Werten von s bis e
- `substr(s, start, len)`: liefert einen Substring der Länge len aus s beginnend bei start
- `repeat(s, n)`: erzeugt einen String durch n-fache Wiederholung von s
- `position(sub in s)`: liefert die Position von sub in s

Sudoku

Einsetzen aller Ziffern von 1 bis 9 an einem bestimmten Feld (hier: 3)

```
select substr(s.sud, 1, 3 - 1) || z || substr(s.sud, 3 + 1),  
      position(' ' in repeat('x', 3)) || substr(s.sud, 3 + 1))  
from (select '53 7 ' || '6 195 ' || '98 6 ' ||  
        '8 6 3' || '4 8 3 1' || '7 2 6' || '6 28 ' ||  
        '419 5' || '8 79'::text as sud) s,  
(select gs::text as z from generate_series(1,9) gs) z
```

Sudoku: Überprüfen der Lösung

- s ist die aktuell betrachtete Lösung
- ind ist die Position, an der Ziffer eingesetzt wurde
- $z.z$ ist die aktuell eingesetzte Ziffer
- lp enthält die zu prüfenden Positionen von 1 bis 9

– für $ind = 3$

```
select * from generate_series(1, 9) lp
```

– Zeile: 1, 2, 3, 4, ...

```
where z.z = substr(s, ((ind - 1) / 9) * 9 + lp, 1)
```

– Spalte: 3, 12, 21, 30, ...

```
or z.z = substr(s, mod(ind - 1, 9) - 8 + lp * 9, 1)
```

– Block: 1, 2, 3, 10, 11, 12, ...

```
+ ((ind - 1) / 27) * 27 + lp + ((lp - 1) / 3) * 6, 1))
```

Sudoku: Vollständige Anfrage

```
with recursive x(s, ind) as (
    select sud, position(' ' in sud)
        from (select 'rätselstring'::text as sud) xx,
union all
    select substr(s, 1, ind - 1) || z || substr(s, ind + 1),
           position(' ' in repeat('x', ind)) || substr(s, ind + 1))
        from x,
(select gs::text as z from generate_series(1,9) gs) z
where ind > 0 and not exists (
    select null from generate_series(1,9) lp
    where z.z = substr(s, ((ind - 1) / 9) * 9 + lp, 1)
        or z.z = substr(s, mod(ind - 1, 9) - 8 + lp * 9, 1)
        or z.z = substr(s, mod(((ind - 1) / 3), 3) * 3
            + ((ind - 1) / 27) * 27 + lp + ((lp - 1) / 3) * 6, 1))
)
select s from x where ind = 0
```

Sudoku: Verbesserte Ausgabe



- `regexp_replace (txt, pat, repl, flag)`: ersetzt in `txt` einen zum Muster `pat` passenden String durch `repl`
- `regexp_split_to_table (txt, pat)`: teilt den String `txt` anhand des Musters `pat` und liefert eine Relation

```
...
select regexp_replace(regexp_split_to_table(
    regexp_replace(s, '.9(?:$)', '\&-', 'g'), '-'),
    '.3(?:$)', '\&|', 'g')
from x
where position(' ' in s) = 0
```

Prozedurale SQL-Erweiterungen: SQL/PSM

- SQL-Standard für prozedurale Erweiterungen
- PSM: Persistent Stored Modules
 - gespeicherte Module aus Prozeduren und Funktionen
 - Einzelroutinen
 - Einbindung externer Routinen (implementiert in C, Java, ...)
 - syntaktische Konstrukte für Schleifen, Bedingungen etc.
 - Basis für Methodenimplementierung für objektrelationale Konzepte

Vorteile gespeicherter Prozeduren

- bewährtes Strukturierungsmittel
- Angabe der Funktionen und Prozeduren erfolgt in der Datenbanksprache selbst; daher nur vom DBMS abhängig
- Optimierung durch DBMS möglich
- Ausführung der Prozeduren erfolgt vollständig unter Kontrolle des DBMS
- zentrale Kontrolle der Prozeduren ermöglicht eine redundanzfreie Darstellung relevanter Aspekte der Anwendungsfunktionalität
- Konzepte und Mechanismen der Rechtevergabe des DBMS können auf Prozeduren erweitert werden
- Prozeduren können in der Integritätssicherung verwendet werden (etwa als Aktionsteil von Triggern)

SQL/PSM: Variablen-deklaration

- Variablen vor Gebrauch deklarieren
- Angabe von Bezeichner und Datentyp
- optional mit Initialwert

```
declare Preis float;  
declare Name varchar(50);  
declare Menge int default 0;
```

- Zuweisung

```
set var = 42;
```

- Bedingte Verzweigungen

```
if Bedingung then Anweisungen
  [ else Anweisungen ] end if;
```

- Schleifen

```
loop Anweisungen end loop;  
while Bedingung do  
    Anweisungen end while;  
repeat Anweisungen  
    until Bedingung end repeat;
```

- Schleifen mit Cursor

```
for SchleifenVariable as CursorName cursor for
  CursorDeklaration
do
  Anweisungen
end for;
```

SQL/PSM: Ablaufkontrolle /4

```
declare wliste varchar(500) default ' ';
declare pos integer default 0;

for w as WeinCurs cursor for
    select Name from WEINE where Weingut = 'Helena'
do
    if pos > 0 then
        set wliste = wliste || ',' || w.Name;
    else
        set wliste = w.Name;
    end if;
    set pos = pos + 1;
end for;
```

- Auslösen einer Ausnahme (Condition)

```
signal ConditionName;
```

- Deklarieren von Ausnahmen

```
declare fehlendes_weingut condition;
declare ungültige_region
    condition for sqlstate value '40123';
```

Ausnahmebehandlung

```
begin
    declare exit handler for ConditionName
        begin
            -- Anweisungen zur Ausnahmebehandlung
        end
        -- Anweisungen, die Ausnahmen auslösen können
    end
```

Funktionsdefinition

```
create function geschmack (rz int)
    returns varchar(20)
begin
    return case
        when rz <= 9 then 'Trocken'
        when rz > 9 and rz <= 18 then 'Halbtrocken'
        when rz > 18 and rz <= 45 then 'Lieblich'
        else 'Süß'
    end
end
```

- Aufruf innerhalb einer Anfrage

```
select Name, Weingut, geschmack(Restzucker)
from WEINE
where Farbe = 'Rot' and
      geschmack(Restzucker) = 'Trocken'
```

- Nutzung außerhalb von Anfragen

```
set wein_geschmack = geschmack (12);
```

- Prozedurdefinition

```
create procedure weinliste (in erz varchar(30),
                           out wliste varchar(500))
begin
    declare pos integer default 0;

    for w as WeinCurs cursor for
        select Name from WEINE where Weingut = erz
    do
        -- siehe Beispiel von Folie 13-56
        end for;
    end; end;
```

SQL/PSM: Prozeduren /2

- Nutzung über call-Anweisung

```
declare wliste varchar(500);
call weinliste ('Helena', wliste);
```

- Eigenschaften von Prozeduren, die Anfrageausführung und -optimierung beeinflussen
 - `deterministic`: Routine liefert für gleiche Parameter gleiche Ergebnisse
 - `no sql`: Routine enthält keine SQL-Anweisungen
 - `contains sql`: Routine enthält SQL-Anweisungen (Standard für SQL-Routinen)
 - `reads sql data`: Routine führt SQL-Anfragen (select-Anweisungen) aus
 - `modifies sql data`: Routine, die DML-Anweisungen (`insert`, `update`, `delete`) enthält

- prozedurale SQL-Erweiterung in PostgreSQL
- ähnlich zu Oracle's PL/SQL und zu SQL/PSM

Funktionsdefinition

```
create function geschmack (rz int) returns varchar(20) as $$  
begin  
    return case when rz <= 9 then 'Trocken'  
        when rz > 9 and rz <= 18 then 'Halbtrocken'  
        when rz > 18 and rz <= 45 then 'Lieblich'  
        else 'Süß'  
    end;  
end;  
$$ language plpgsql;
```

Zusammenfassung

- Rekursion in SQL
- komplexe SQL-Anfragen mit erweiterten Sprachmitteln
- Prozedurale SQL-Erweiterungen

Kontrollfragen

- Welchem Zweck dienen rekursive Anfragen in SQL?
- Wie kann SQL prozedural erweitert werden?



Teil IX

Grundlagen von Anfragen: Algebra & Kalkül

1. Kriterien für Anfragesprachen
2. Anfragealgebren
3. Erweiterungen der Relationenalgebra
4. Anfragekalküle
5. Beispiele für Bereichskalkül
6. Eigenschaften des Bereichskalküls

Lernziele für heute . . .

- Verständnis der formalen Grundlagen relationaler Anfragesprachen
- Kenntnisse zur Formulierung von Anfragen in der relationalen Algebra
- Kenntnisse zur Formulierung von Kalkülanfragen



Kriterien für Anfragesprachen

Einführung

- bisher:
 - Relationenschemata mit Basisrelationen, die in der Datenbank gespeichert sind
- jetzt:
 - „abgeleitete“ Relationenschemata mit virtuellen Relationen, die aus den Basisrelationen berechnet werden (Basisrelationen bleiben unverändert)

- **Anfrage:** Folge von Operationen, die aus den Basisrelationen eine Ergebnisrelation berechnet
 - Ergebnisrelation interaktiv auf dem Bildschirm anzeigen oder
 - per Programm weiterverarbeiten („Einbettung“)
- **Sicht:** Folge von Operationen, die unter einem Sichtnamen langfristig abgespeichert wird und unter diesem Namen wieder aufgerufen werden kann; ergibt eine Sichtrelation
- **Snapshot:** Ergebnisrelation einer Anfrage, die unter einem Snapshot-Namen abgelegt wird, aber nie ein zweites Mal (mit geänderten Basisrelationen) berechnet wird (etwa Jahresbilanzen)

Kriterien für Anfragesprachen

- **Ad-Hoc-Formulierung:** Benutzer soll eine Anfrage formulieren können, ohne ein vollständiges Programm schreiben zu müssen
- **Deskriptivität:** Benutzer soll formulieren „Was will ich haben?“ und nicht „Wie komme ich an das, was ich haben will?“
- **Mengenorientiertheit:** jede Operation soll auf Mengen von Daten gleichzeitig arbeiten, nicht navigierend nur auf einzelnen Elementen („one-tuple-at-a-time“)
- **Abgeschlossenheit:** Ergebnis ist wieder eine Relation und kann wieder als Eingabe für die nächste Anfrage verwendet werden

- **Adäquatheit:** alle Konstrukte des zugrundeliegenden Datenmodells werden unterstützt
- **Orthogonalität:** Sprachkonstrukte sind in ähnlichen Situationen auch ähnlich anwendbar
- **Optimierbarkeit:** Sprache besteht aus wenigen Operationen, für die es Optimierungsregeln gibt
- **Effizienz:** jede Operation ist effizient ausführbar (im Relationenmodell hat jede Operation eine Komplexität $\leq O(n^2)$, n Anzahl der Tupel einer Relation).

- **Sicherheit:** keine Anfrage, die syntaktisch korrekt ist, darf in eine Endlosschleife geraten oder ein unendliches Ergebnis liefern
- **Eingeschränktheit:** (folgt aus Sicherheit, Optimierbarkeit, Effizienz) Anfragesprache darf keine komplette Programmiersprache sein
- **Vollständigkeit:** Sprache muss mindestens die Anfragen einer Standardsprache (wie etwa die in diesem Kapitel einzuführende Relationenalgebra oder den sicheren Relationenkalkül) ausdrücken können

Anfragealgebren

- Mathematik: Algebra definiert durch Wertebereich und auf diesem definierte Operatoren
- für Datenbankanfragen: Inhalte der Datenbank sind Werte, und Operatoren definieren Funktionen zum Berechnen von Anfrageergebnissen
 - Relationenalgebra
 - Algebra-Erweiterungen

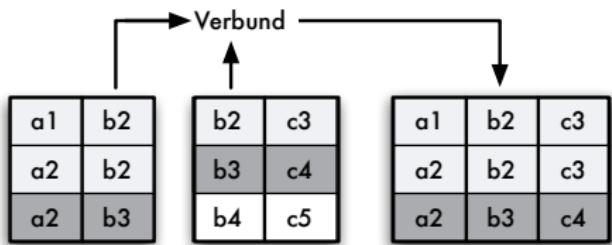
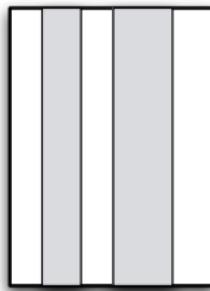
- **Spalten ausblenden:** Projektion π
- **Zeilen heraussuchen:** Selektion σ
- **Tabellen verknüpfen:** Verbund (Join) \bowtie
- **Tabellen vereinigen:** Vereinigung \cup
- **Tabellen voneinander abziehen:** Differenz $-$
- **Spalten umbenennen:** Umbenennung β
(wichtig für \bowtie und $\cup, -$)

Relationenalgebra: Übersicht

Selektion



Projektion



Projektion

- Syntax

$\pi_{Attributmenge} (Relation)$

- Semantik

$$\pi_X(r) := \{t(X) \mid t \in r\}$$

für $r(R)$ und $X \subseteq R$ Attributmenge in R

- Eigenschaft für $Y \subseteq X \subseteq R$

$$\pi_Y(\pi_X(r)) = \pi_Y(r)$$

- **Achtung:** π entfernt Duplikate (Mengensemantik)

Projektion: Beispiel

$\pi_{\text{Region}}(\text{ERZEUGER})$

Region
Südaustralien
Kalifornien
Bordeaux
Hessen

Projektion: Beispiel 2

$\pi_{\text{Anbaugebiet}, \text{Region}}(\text{ERZEUGER})$

Anbaugebiet	Region
Barossa Valley	Südaustralien
Napa Valley	Kalifornien
Saint-Emilion	Bordeaux
Pomerol	Bordeaux
Rheingau	Hessen

- Syntax

$$\sigma_{Bedingung}(Relation)$$

- Semantik (für $A \in R$)

$$\sigma_{A=a}(r) := \{t \in r \mid t(A) = a\}$$

- **Konstantenselektion**

Attribut θ Konstante

boolesches Prädikat θ ist $=$ oder \neq , bei linear geordneten Wertebereichen auch \leq , $<$, \geq oder $>$

- **Attributselektion**

Attribut1 θ Attribut2

- logische Verknüpfung mehrerer Konstanten- oder Attribut Selektionen mit \wedge , \vee oder \neg

- Kommutativität

$$\sigma_{A=a}(\sigma_{B=b}(r)) = \sigma_{B=b}(\sigma_{A=a}(r))$$

- falls $A \in X$, $X \subseteq R$

$$\pi_X(\sigma_{A=a}(r)) = \sigma_{A=a}(\pi_X(r))$$

- Distributivität bzgl. \cup , \cap , $-$

$$\sigma_{A=a}(r \cup s) = \sigma_{A=a}(r) \cup \sigma_{A=a}(s)$$

Selektion: Beispiel

$\sigma_{\text{Jahrgang} > 2000}(\text{WEINE})$

WeinID	Name	Farbe	Jahrgang	Weingut
2168	Creek Shiraz	Rot	2003	Creek
3456	Zinfandel	Rot	2004	Helena
2171	Pinot Noir	Rot	2001	Creek
4961	Chardonnay	Weiβ	2002	Bighorn

- Syntax des (natürlichen) Verbundes (engl.: natural join)

$$Relation1 \bowtie Relation2$$

- Semantik

$$\begin{aligned} r_1 \bowtie r_2 := & \{ t \mid t(R_1 \cup R_2) \wedge \\ & [\forall i \in \{1, 2\} \exists t_i \in r_i : t_i = t(R_i)] \} \end{aligned}$$

- Verbund verknüpft Tabellen über gleichbenannten Spalten bei gleichen Attributwerten

Verbund: Eigenschaften

- Schema für $r(R) \bowtie r(S)$ ist Vereinigung der Attributmengen
 $RS = R \cup S$
- aus $R_1 \cap R_2 = \{\}$ folgt $r_1 \bowtie r_2 = r_1 \times r_2$
- Kommutativität: $r_1 \bowtie r_2 = r_2 \bowtie r_1$
- Assoziativität: $(r_1 \bowtie r_2) \bowtie r_3 = r_1 \bowtie (r_2 \bowtie r_3)$
- daher erlaubt:

$$\bowtie_{i=1}^p r_i$$

Verbund: Beispiel

WEINE × ERZEUGER

WeinID	Name	...	Weingut	Anbaugebiet	Region
1042	La Rose Grand Cru	...	Ch. La Rose	Saint-Emilion	Bordeaux
2168	Creek Shiraz	...	Creek	Barossa Valley	Südaustralien
3456	Zinfandel	...	Helena	Napa Valley	Kalifornien
2171	Pinot Noir	...	Creek	Barossa Valley	Südaustralien
3478	Pinot Noir	...	Helena	Napa Valley	Kalifornien
4711	Riesling Reserve	...	Müller	Rheingau	Hessen
4961	Chardonnay	...	Bighorn	Napa Valley	Kalifornien

Umbenennung

- Syntax

$$\beta_{neu \leftarrow alt}(Relation)$$

- Semantik

$$\beta_{B \leftarrow A}(r) := \{t' \mid \exists t \in r : t'(R - A) = t(R - A) \wedge t'(B) = t(A)\}$$

- ändert Attributnamen von *alt* in *neu*

$$\beta_{Name \leftarrow Nachname} \text{ (KRITIKER)}$$

- durch Umbenennung nun möglich

- Verbunde, wo bisher kartesische Produkte ausgeführt wurden (unterschiedliche Attribute werden gleich benannt),
- kartesische Produkte, wo bisher Verbunde ausgeführt wurden (gleiche Attribute werden unterschiedlich genannt),
- Mengenoperationen

Berechnung des Kreuzproduktes

- natürlicher Verbund **entartet zum Kreuzprodukt**, wenn keine gemeinsamen Attribute existieren
- Erzwingen durch Umbenennung
 - Beispiel: $R1(A, B, C)$ und $R2(C, D)$

$$R1 \times R2 \equiv R1 \bowtie_{E \leftarrow C} (R2)$$

- Kreuzprodukt + Selektion simuliert natürlichen Verbund

$$R1 \bowtie R2 \equiv \sigma_{R1.C=R2.C}(R1 \times R2)$$

Mengenoperationen: Semantik

- formal für $r_1(R)$ und $r_2(R)$
 - Vereinigung $r_1 \cup r_2 := \{t \mid t \in r_1 \vee t \in r_2\}$
 - Durchschnitt $r_1 \cap r_2 := \{t \mid t \in r_1 \wedge t \in r_2\}$
 - Differenz $r_1 - r_2 := \{t \mid t \in r_1 \wedge t \notin r_2\}$
- Durchschnitt \cap wegen $r_1 \cap r_2 = r_1 - (r_1 - r_2)$ überflüssig

- Minimale Relationenalgebra:

$$\Omega = \pi, \sigma, \bowtie, \beta, \cup \text{ und } -$$

- unabhängig: kein Operator kann weggelassen werden ohne Vollständigkeit zu verlieren
- andere unabhängige Menge: \bowtie und β durch \times ersetzen
- **Relationale Vollständigkeit:** jede andere Menge von Operationen genauso mächtig wie Ω
- **strenge relationale Vollständigkeit:** zu jedem Ausdruck mit Operatoren aus Ω gibt es einen Ausdruck auch mit der anderen Menge von Operationen

Erweiterungen der Relationenalgebra

Erweiterungen der Relationenalgebra



- weitere Verbundoperationen
- Division
- Gruppierung und geschachtelte Relationen
- ...

- für $L(AB)$, $R(BC)$, $S(DE)$
- **Gleichverbund** (engl. *equi-join*): Gleichheitsbedingung über explizit angegebene und evtl. verschiedene Attribute

$$r(R) \bowtie_{C=D} r(S)$$

- **Theta-Verbund** (engl. *θ -join*): beliebige Verbundbedingung

$$r(R) \bowtie_{C>D} r(S)$$

- **Semi-Verbund**: nur Attribute eines Operanden erscheinen im Ergebnis

$$r(L) \ltimes r(R) = \pi_L(r(L) \bowtie r(R))$$

- **äußere Verbunde** (engl. *outer join*)

- Übernahme von „dangling tuples“ in das Ergebnis und Auffüllen mit Nullwerten
- **voller äußerer Verbund** übernimmt alle Tupel beider Operanden

$$r \bowtie s$$

- **linker äußerer Verbund** übernimmt alle Tupel des linken Operanden

$$r \bowtie\llcorner s$$

- **rechter äußerer Verbund** übernimmt alle Tupel des rechten Operanden

$$r \bowtie\lrcorner s$$

Äußere Verbunde /2

LINKS

A	B
1	2
2	3

RECHTS

B	C
3	4
4	5

\bowtie

A	B	C
2	3	4

$\bowtie\bowtie$

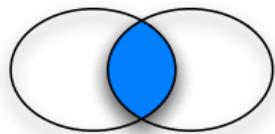
A	B	C
1	2	\perp
2	3	4
\perp	4	5

$\bowtie\bowtie$

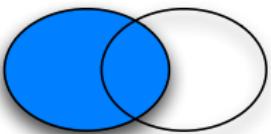
A	B	C
1	2	\perp
2	3	4

$\bowtie\bowtie$

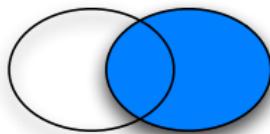
A	B	C
2	3	4
\perp	4	5



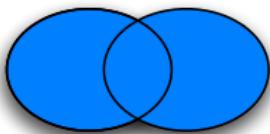
natural join



left outer join



right outer join



full outer join

Problem: Quantoren

- Allquantor in Relationenalgebra ausdrücken, obwohl in Selektionsbedingungen nicht erlaubt
- **Division** (kann aus Ω hergeleitet werden)
- $r_1(R_1)$ und $r_2(R_2)$ gegeben mit $R_2 \subseteq R_1$, $R' = R_1 - R_2$. Dann ist

$$\begin{aligned}r'(R') &= \{t \mid \forall t_2 \in r_2 \exists t_1 \in r_1 : t_1(R') = t \wedge t_1(R_2) = t_2\} \\&= r_1 \div r_2\end{aligned}$$

- Division von r_1 durch r_2

$$r_1 \div r_2 = \pi_{R'}(r_1) - \pi_{R'}((\pi_{R'}(r_1) \bowtie r_2) - r_1)$$

Division: Beispiel

WEIN-EMPFEHLUNG	Wein	Kritiker
	La Rose Grand Cru	Parker
	Pinot Noir	Parker
	Riesling Reserve	Parker
	La Rose Grand Cru	Clarke
	Pinot Noir	Clarke
	Riesling Reserve	Gault-Millau

GUIDES1	Kritiker
	Parker
	Clarke

GUIDES2	Kritiker
	Parker
	Gault-Millau

Divisionsbeispiele

- Division mit erster Tabelle

WEIN_EMPFEHLUNG ÷ GUIDES1

liefert

Wein
La Rose Grand Cru Pinot Noir

- Division mit zweiter Kritikerliste

WEIN_EMPFEHLUNG ÷ GUIDES2

liefert

Wein
Riesling Reserve

- Analogie zur arithmetischen Operation der ganzzahligen Division

Division

Die ganzzahlige Division ist in dem Sinne die Inverse zur Multiplikation, indem sie als Ergebnis die größte Zahl liefert, für die die Multiplikation mit dem Divisor kleiner ist als der Dividend.

Analog gilt: $r = r_1 \div r_2$ ist die größte Relation, für die $r \bowtie r_2 \subseteq r_1$ ist.

Division in SQL: Simulation des Allquantors



```
select distinct Wein
from WEIN_EMPFEHLUNG w1
where not exists (
    select * from GUIDES2 g
    where not exists (
        select * from WEIN_EMPFEHLUNG w2
        where g.Kritiker = w2.Kritiker
        and w1.Wein = w2.Wein))
```

- „Gib alle Weine aus, so dass kein Wein existiert, der nicht von allen Kritikern in der Relation GUIDES2 empfohlen wird“.

Gruppierungsoperator γ

$$\gamma_{f_1(x_1), f_2(x_2), \dots, f_n(x_n); A}(r(R))$$

- erweitert Attributschema von $r(R)$ um neue Attribute, die mit den Funktionsanwendungen $f_1(x_1), f_2(x_2), \dots, f_n(x_n)$ korrespondieren
- Anwendung der Funktionen $f_i(x_i)$ auf die Teilmenge derjenigen Tupel von $r(R)$ die gleiche Attributwerte für die Attribute A haben

```
select f1(x1), f2(x2), ..., fn(xn), A
from R
group by A
```

Semantik des Gruppierungsoperators

- leere Attributmenge $A = \emptyset$:

$$\gamma_{F(X);\emptyset}(r(R)) = r(R) \times r(R)^{F(X)}$$

mit $r(R)^{F(X)}$ ist Relation mit Attribut $F(X)$ und einem Tupel als Wert von $F(X)$ auf $r(R)$

- ohne Funktion:

$$\gamma_{\emptyset;\emptyset}(r(R)) = r(R)$$

- allgemeiner Fall:

$$\gamma_{F(X);A}(r(R)) = \bigcup_{t \in R} \gamma_{F(X);\emptyset}(\sigma_{A=t.A}(r(R)))$$

Anfragekalküle

- Kalkül: eine formale logische Sprache zur Formulierung von Aussagen
- Ziel: Einsatz eines derartigen Kalküls zur Formulierung von Datenbank-Anfragen
- Logikbasierter Ansatz:
 - Datenbankinhalte entsprechen Belegungen von Prädikaten einer Logik
 - Anfragen abgeleiteten Prädikaten

Ein allgemeiner Kalkül

- Motivation: mathematische Notation

$$\{x^2 \mid x \in \mathbb{N} \wedge x^3 > 0 \wedge x^3 < 1000\}$$

- **Anfrage** hat die Form

$$\{f(\bar{x}) \mid p(\bar{x})\}$$

- \bar{x} bezeichnet Menge von freien Variablen

$$\bar{x} = \{x_1 : D_1, \dots, x_n : D_n\}$$

Ein allgemeiner Kalkül /2

- Funktion f bezeichnet Ergebnisfunktion über \bar{x}
 - wichtige Spezialfälle: Angabe einer Variable selber (f ist hier die Identitätsfunktion) und Tupelkonstruktion (Ergebnis vom Typ **tuple of**)
- p Selektionsprädikat über freien Variablen \bar{x}
 - Terme aus Variablen, Konstanten und Funktionsanwendungen
 - Prädikate der Datentypen, etwa $\leq, <, >, \geq, \dots$
→ atomare Formeln über Termen
 - Bezug zur aktuellen Datenbank → Datenbankprädikate, z.B. Relationennamen im RM
 - prädikatenlogischen Operatoren $\wedge, \vee, \neg, \forall, \exists$
→ Formeln

$$\bar{x} = \{x_1 : D_1, \dots, x_n : D_n\}$$

1. Bestimme aller Belegungen der freien Variablen in \bar{x} , für die das Prädikat p wahr wird.
2. Wende Funktion f auf die durch diese Belegungen gegebenen Werte an.

Sicherheit von Anfragen

Unter welchen Umständen liefern Kalkülanfragen endliche Ergebnisse?

→ **Sicherheit von Anfragen**

- **Bereichskalkül:** Variablen nehmen Werte elementarer Datentypen (*Bereiche*) an
- **Tupelkalkül:** Variablen variieren über Tupelwerte (entsprechend den Zeilen einer Relation)

Tupelkalkül

- Grundlage von SFW-Anfragen in SQL
- Variablen sind **tupelwertig**
- Beispiel:

$$\{w \mid w \in \text{WEINE} \wedge w.\text{Farbe} = \text{'Rot'}\}$$

Tupelkalkül: Beispiele

- konstruierte Tupel

$$\{\langle w.\text{Name}, w.\text{Weingut} \rangle \mid w \in \text{WEINE} \wedge w.\text{Farbe} = \text{'Rot'}\}$$

- Verbund

$$\begin{aligned} \{\langle e.\text{Weingut} \rangle \mid e \in \text{ERZEUGER} \wedge w \in \text{WEINE} \\ \wedge e.\text{Weingut} = w.\text{Weingut}\} \end{aligned}$$

- Schachtelung

$$\begin{aligned} \{\langle w.\text{Name}, w.\text{Weingut} \rangle \mid w \in \text{WEINE} \wedge \\ \exists e \in \text{ERZEUGER}(w.\text{Weingut} = e.\text{Weingut} \wedge \\ e.\text{Region} = \text{'Bordeaux'})\} \end{aligned}$$

Motivation: Die Sprache QBE

- „Query by Example“
- Anfragen in QBE: Einträge in Tabellengerüsten
- Intuition: **Beispieleinträge** in Tabellen
- Vorläufer verschiedener tabellenbasierter Anfrageschnittstellen kommerzieller Systeme
- basiert auf logischem Kalkül mit Bereichsvariablen

Anfragen in QBE: Selektion und Projektion

- Anfrage: „Alle Rotweine, die vor 2015 produziert wurden“

WEINE	Name	Weingut	Farbe	Jahrgang
P.			Rot	< 2015

$$\{n \mid \text{WEINE}(n, _, \text{'Rot'}, j) \wedge j < 2015\}$$

Anfragen in QBE: Verbund

- Anfrage: „Alle Rotweine aus der Region Bordeaux“

WEINE	Name	Weingut	Farbe	Jahrgang
P.	_w	Rot		
ERZEUGER	Weingut	Region	Anbaugebiet	
	_w	Bordeaux		

$$\{n \mid \text{WEINE}(n, w, \text{'Rot'}, _) \wedge \text{ERZEUGER}(w, \text{'Bordeaux'}, _)\}$$

Anfragen in QBE: Selbstverbund

- Anfrage: „Regionen mit zwei oder mehr Erzeugern“

ERZEUGER	Weingut	Region	Anbaugebiet
	_eins	P. _region	
	¬ _eins	_region	

$$\{r \mid \text{ERZEUGER}(x, r, _) \wedge \text{ERZEUGER}(y, r, _) \wedge x \neq y\}$$

- MS-Access: Datenbankprogramm für Windows
 - Basisrelationen mit Schlüsseln
 - Fremdschlüssel über graphische Angabe von Beziehungen
 - graphische Definition von Anfragen (SQL-ähnlich)
 - interaktive Definition von Formularen und Berichten
- Unterstützung von QBE

Access: Projektion und Selektion

Abfrage1 : Auswahlabfrage

WEINE
*
Name
Jahrgang
Farbe
Weingut

Feld: Name Jahrgang Farbe
Tabelle: WEINE WEINE WEINE
Sortierung:
Anzeigen:
Kriterien: >2005 "Rot"
oder:

Access: Verbund

Abfrage5 : Auswahlabfrage



Feld:	Name	Jahrgang	Anbaugebiet
Tabelle:	WEINE	WEINE	ERZEUGER
Sortierung:			
Anzeigen:	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Kriterien:			"Napa Valley"
oder:			

- **Terme:**

- Konstanten, etwa 42 oder 'MZ-4'
- Variablen zu Datentypen, etwa x
Datentypangabe erfolgt in der Regel implizit und wird nicht explizit deklariert!
- Funktionsanwendung $f(t_1, \dots, t_n)$: Funktion f , Terme t_i , etwa $\text{plus}(12, x)$ bzw. in Infixnotation $12 + x$

- **Atomare Formeln:**

- Prädikatanwendung $\Theta(t_1, \dots, t_n)$, $\Theta \in \{<, >, \leq, \geq, \neq, =, \dots\}$
Datentypprädikat, Terme t_i
Zweistellige Prädikate wie üblich in Infix-Notation.
Beispiele: $x = y$, $42 > x$ oder $3 + 7 = 11$.

- **Atomare Formeln** (fortg.):
 - Prädikatanwendungen für Datenbankprädikate, notiert als $R(t_1, \dots, t_n)$ für einen Relationennamen R
Voraussetzung: n muss die Stelligkeit der Relation R sein und alle t_i müssen vom passenden Typ sein
Beispiel: $\text{ERZEUGER}(x, \text{'Hessen'}, z)$
 - Formeln wie üblich mit \wedge , \vee , \neg , \forall und \exists

- *Anfragen:* $\{x_1, \dots, x_n \mid \phi(x_1, \dots, x_n)\}$
 - ϕ ist Formel über den in der Ergebnisliste aufgeführten Variablen x_1 bis x_n
 - Ergebnis ist eine Menge von Tupeln
 - Tupelkonstruktion erfolgt implizit aus den Werten der Variablen in der Ergebnisliste
- Beispiel

$$\{x \mid \text{ERZEUGER}(x, y, z) \wedge z = \text{'Hessen'}\}$$

- Einschränkung des Bereichskalküls:
 - **Wertebereich:** ganze Zahlen
 - **Datentypprädictate** werden wie bei der Relationenalgebra auf Gleichheit und elementare Vergleichsoperatoren eingeschränkt
 - **Funktionsanwendungen** sind nicht erlaubt; nur Konstanten dürfen neben Bereichsvariablen als Terme verwendet werden

Semantisch sichere Anfragen

Anfragen, die für jeden Datenbankzustand $\sigma(\mathcal{R})$ ein endliches Ergebnis liefern

- Beispiel für nicht sichere Anfrage:

$$\{x, y \mid \neg R(x, y)\}$$

- Beispiel für sichere Anfrage:

$$\{x, y \mid R(x, y)\}$$

- Weiteres Beispiel für sichere Anfrage:

$$\{x, y \mid y = 10 \wedge x > 0 \wedge x < 10\}$$

Sicherheit folgt direkt aus den Regeln der Arithmetik.

Semantische Sicherheit

Semantische Sicherheit ist im Allgemeinen **nicht entscheidbar!**

- **Syntaktisch sichere Anfragen:** Anfragen, die syntaktischen Einschränkungen unterliegen, um die semantische Sicherheit zu erzwingen.
- Grundidee:

Syntaktische Sicherheit

Jede freie Variable x_i muss überall in $\phi(x_1, \dots)$ durch **positives Auftreten** $x_i = t$ oder $R(\dots, x_i, \dots)$ an endliche Bereiche gebunden werden.

- Bindung an endliche Bereiche muss für die ganze Bedingung gelten, also insbesondere für alle Zweige einer Disjunktion

Anfragen

sichere Anfragen

syntaktisch sichere Anfragen

Beispiele für Bereichskalkül

Beispiele Bereichskalkül

- Anfrage: „Alle Weingüter von Erzeugern in Hessen.“

$$\{x \mid \text{ERZEUGER}(x, y, z) \wedge z = \text{'Hessen'}\}$$

- Vereinfachte Notation: Ansonsten ungebundene Variablen (hier y und z) im Bedingungsteil existentiell mit \exists gebunden
- Vollständige Version:

$$\{x \mid \exists y \exists z \text{ERZEUGER}(x, y, z) \wedge z = \text{'Hessen'}\}$$

- Einsparung von Bereichsvariablen, indem Konstanten als Parameter des Prädikats eingesetzt werden:

$$\{x \mid \text{ERZEUGER}(x, y, \text{'Hessen'})\}$$

Beispiele Bereichskalkül /2

- Abkürzung für beliebige, unterschiedliche existentiell gebundene Variablen ist _ Symbol:

$$\{x \mid \text{ERZEUGER}(x, _, z) \wedge z = \text{'Hessen'}\}$$

- Verschiedene Auftreten des Symbols _ stehen hierbei für paarweise verschiedene Variablen

Beispiele Bereichskalkül /3

- Anfrage: „Regionen mit mehr als zwei Weingütern.“

$$\{z \mid \text{ERZEUGER}(x, y, z) \wedge \text{ERZEUGER}(x', y', z) \wedge x \neq x'\}$$

- Anfrage zeigt eine Verbundbildung über das dritte Attribut der ERZEUGER-Relation
- Verbundbildung kann einfach durch die Verwendung der selben Bereichsvariablen als Parameter in verschiedenen Relationsprädikaten erfolgen

- Anfrage: „Aus welcher Region sind welche Weine mit Jahrgang vor 1970 im Angebot?“

$$\{y, r \mid \text{WEINE}(x, y, z, j, w) \wedge \text{ERZEUGER}(w, a, r) \wedge j < 1970\}$$

- Verbund über zwei Relationen

- Anfrage: „Aus welchen Regionen gibt es Rotweine?“

$$\{z \mid \text{ERZEUGER}(x, y, z) \wedge \exists a \exists b \exists c \exists d (\text{WEIN}(a, b, c, d, x) \wedge c = \text{'Rot'})\}$$

- Einsatz einer existentiell gebundenen Unteranfrage
- derartige Unteranfragen können aufgrund der Regeln der Prädikatenlogik wie folgt aufgelöst werden:

$$\{z \mid \text{ERZEUGER}(x, y, z) \wedge (\text{WEIN}(a, b, c, d, x) \wedge c = \text{'Rot'})\}$$

Beispiele Bereichskalkül /6

- Anfrage: „Welches Weingut hat nur Weine mit Jahrgang nach 1995 im Angebot?“

$$\{x \mid \text{ERZEUGER}(x, y, z) \wedge \forall a \forall b \forall c \forall d \\ (\text{WEIN}(a, b, c, d, x) \implies d > 1995)\}$$

- universell gebundene Teilformeln können nicht aufgelöst werden

Eigenschaften des Bereichskalküls

Ausdrucksfähigkeit des Bereichskalküls

Bereichskalkül ist **streng relational vollständig**, d.h. zu jedem Term τ der Relationenalgebra gibt es einen äquivalenten (sicheren) Ausdruck η des Bereichskalküls.

Umsetzung von Relationenoperationen



Geg.: Relationenschemata $R(A_1, \dots, A_n)$ und $S(B_1, \dots, B_m)$

- Vereinigung (für $n = m$)

$$R \cup S \hat{=} \{x_1 \dots x_n \mid R(x_1, \dots, x_n) \vee S(x_1, \dots, x_n)\}$$

- Differenz (für $n = m$)

$$R - S \hat{=} \{x_1 \dots x_n \mid R(x_1, \dots, x_n) \wedge \neg S(x_1, \dots, x_n)\}$$

- Natürlicher Verbund

$$R \bowtie S \hat{=} \{x_1 \dots x_n x_{n+1} \dots x_{n+m-i} \mid R(x_1, \dots, x_n) \wedge S(x_1, \dots, x_i, x_{n+1}, \dots, x_{n+m-i})\}$$

Annahme: die ersten i Attribute von R und S sind die Verbundattribute, also $A_j = B_j$ für $j = 1 \dots i$

- Projektion

$$\pi_{\bar{A}}(R) \doteq \{y_1 \dots y_k \mid \exists x_1 \dots \exists x_n (R(x_1, \dots, x_n) \wedge y_1 = x_{i_1} \wedge \dots \wedge y_k = x_{i_k})\}$$

Attributliste der Projektion: $\bar{A} = (A_{i_1}, \dots, A_{i_k})$

- Selektion

$$\sigma_\phi(R) \doteq \{x_1 \dots x_n \mid R(x_1, \dots, x_n) \wedge \phi'\}$$

ϕ' wird aus ϕ gewonnen, indem Variable x_i an Stelle der Attributnamen A_i eingesetzt werden

Zusammenfassung

- formale Modelle für Anfragen in Datenbanksystemen
- Relationenalgebra
 - operationaler Ansatz
 - Anfrage als Schachtelung von Operatoren auf Relationen
- Anfragekalküle
 - logikbasierter Ansatz
 - Anfragen als abgeleitete Prädikate
 - *im Buch: Abschnitte 4.2.3, 4.2.4 und 9.3*

Kontrollfragen

- Welche Bedeutung haben Äquivalenz, Unabhängigkeit und Vollständigkeit in der Relationenalgebra?
- Wie lässt sich die Semantik von erweiterten SQL-Operationen in der Relationenalgebra ausdrücken?
- Was unterscheidet Relationenalgebra und relationale Anfragekalküle?
- Welche Rolle spielt die Sicherheit von Anfragen?



Teil X

Transaktionen, Integrität und Trigger

1. Grundbegriffe
2. Transaktionsbegriff
3. Transaktionen in SQL
4. Integritätsbedingungen in SQL
5. Trigger
6. Schemaevolution

Lernziele für heute . . .

- Verständnis des Transaktionskonzeptes in Datenbanken
- Verständnis der Grundlagen der Integritätssicherung in Datenbanken
- Kenntnisse zur Formulierung und Implementierung von Integritätsbedingungen sowie Schemaänderungen



Grundbegriffe

- **Integritätsbedingung** (engl. *integrity constraint* oder *assertion*):
Bedingung für die „Zulässigkeit“ oder „Korrektheit“
- in Bezug auf Datenbanken:
 - (einzelne) Datenbankzustände,
 - Zustandsübergänge vom alten in den neuen Datenbankzustand,
 - langfristige Datenbankentwicklungen

Klassifikation von Integrität

Bedingungsklasse	zeitlicher Kontext	
statisch	Datenbankzustand	
dynamisch	transitional temporal	Zustandsübergang Zustandsfolge

1. *Typintegrität:*

- SQL erlaubt Angabe von Wertebereichen zu Attributen
- Erlauben oder Verbieten von Nullwerten

2. *Schlüsselintegrität:*

- Angabe eines Schlüssels für eine Relation

3. *Referentielle Integrität:*

- die Angabe von Fremdschlüsseln

Transaktionsbegriff

Beispielszenarien

- Platzreservierung für Flüge gleichzeitig aus vielen Reisebüros
 - Platz könnte mehrfach verkauft werden, wenn mehrere Reisebüros den Platz als verfügbar identifizieren
- überschneidende Kontooperationen einer Bank
- statistische Datenbankoperationen
 - Ergebnisse sind verfälscht, wenn während der Berechnung Daten geändert werden

Transaktion

Eine **Transaktion** ist eine Folge von Operationen (Aktionen), die die Datenbank von einem konsistenten Zustand in einen konsistenten, eventuell veränderten, Zustand überführt, wobei das **ACID-Prinzip** eingehalten werden muss.

- Aspekte:
 - Semantische Integrität: Korrekter (konsistenter) DB-Zustand nach Ende der Transaktion
 - Ablaufintegrität: Fehler durch „gleichzeitigen“ Zugriff mehrerer Benutzer auf dieselben Daten vermeiden

- **Atomicity** (Atomarität):
Transaktion wird entweder ganz oder gar nicht ausgeführt
- **Consistency** (Konsistenz oder auch Integritätserhaltung):
Datenbank ist vor Beginn und nach Beendigung einer Transaktion jeweils in einem konsistenten Zustand
- **Isolation** (Isolation):
Nutzer, der mit einer Datenbank arbeitet, sollte den Eindruck haben, dass er mit dieser Datenbank alleine arbeitet
- **Durability** (Dauerhaftigkeit / Persistenz):
nach erfolgreichem Abschluss einer Transaktion muss das Ergebnis dieser Transaktion „dauerhaft“ in der Datenbank gespeichert werden

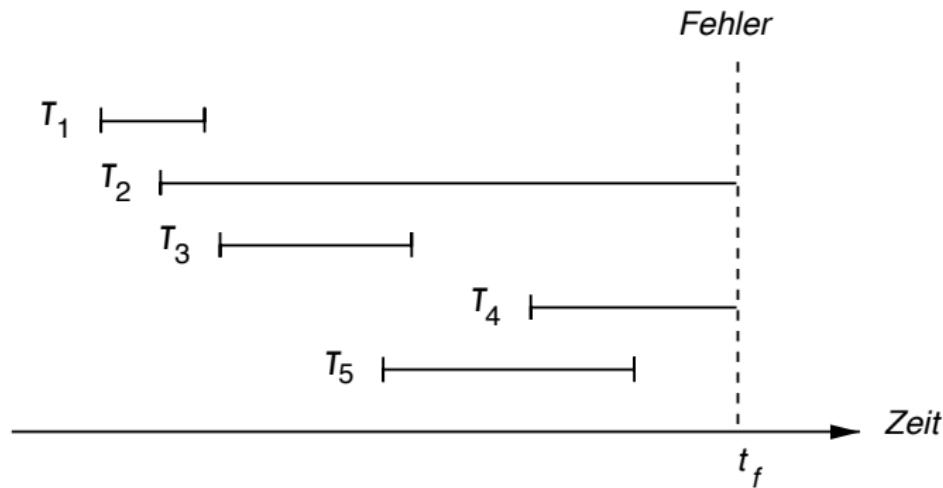
Kommandos einer Transaktionssprache

- Beginn einer Transaktion: Begin-of-Transaction-Kommando BOT (in SQL implizit!)
- commit: die Transaktion soll erfolgreich beendet werden
- abort: die Transaktion soll abgebrochen werden

Transaktion: Integritätsverletzung

- Beispiel:
 - Übertragung eines Betrages B von einem Haushaltsposten $K1$ auf einen anderen Posten $K2$
 - Bedingung: Summe der Kontostände der Haushaltsposten bleibt konstant
- vereinfachte Notation
$$Transfer = < K1 := K1 - B; K2 := K2 + B >;$$
- Realisierung in SQL: als Sequenz zweier elementarer Änderungen \rightsquigarrow
Bedingung ist zwischen den einzelnen Änderungsschritten nicht unbedingt erfüllt!

Transaktion: Verhalten bei Systemabsturz



- **Folgen:**

- Inhalt des flüchtigen Speichers zum Zeitpunkt t_f ist unbrauchbar → Transaktionen in unterschiedlicher Weise davon betroffen

- **Transaktionszustände:**

- zum Fehlerzeitpunkt noch aktive Transaktionen (T_2 und T_4)
- bereits vor dem Fehlerzeitpunkt beendete Transaktionen (T_1 , T_3 und T_5)

- Repräsentation von Datenbankänderungen einer Transaktion
 - `read(A,x)`: weise den Wert des DB-Objektes A der Variablen x zu
 - `write(x, A)`: speichere den Wert der Variablen x im DB-Objekt A
- Beispiel einer Transaktion T :

```
read(A, x); x := x - 200; write(x, A);
read(B, y); y := y + 100; write(y, B);
```
- Ausführungsvarianten für zwei Transaktionen T_1, T_2 :
 - seriell, etwa T_1 vor T_2
 - „gemischt“, etwa abwechselnd Schritte von T_1 und T_2

Probleme im Mehrbenutzerbetrieb



- Inkonsistentes Lesen: Nonrepeatable Read
- Abhängigkeiten von nicht freigegebenen Daten: Dirty Read
- Das Phantom-Problem
- Verlorengegangenes Ändern: Lost Update

Nonrepeatable Read

Beispiel:

- Zusicherung $x = A + B + C$ am Ende der Transaktion T_1
- x, y, z seien lokale Variablen
- T_i ist die Transaktion i
- Integritätsbedingung $A + B + C = 0$

Beispiel für inkonsistentes Lesen

T_1	T_2
<pre>read(A, x); read(B, y); read(C, z); commit;</pre> <pre>read(A, x); x := x + y; read(C, z); x := x + z; commit;</pre>	<pre>read(A, y); y := y/2; write(y, A); read(C, z); z := z + y; write(z, C); commit;</pre>

Dirty Read

T_1	T_2
<pre>read(A, x); x := x + 100; write(x, A); abort;</pre>	<pre>read(A, x); read(B, y); y := y + x; write(y, B); commit;</pre>

Das Phantom-Problem

T_1	T_2
<pre>select count (*) into X from Kunde; update Kunde set Bonus = Bonus +10000/X; commit;</pre>	<pre>insert into Kunde values ('Meier', 0, ...); commit;</pre>

Lost Update

T_1	T_2	A
read (A, x);		10
	read (A, x);	10
$x := x + 1;$		10
	$x := x + 1;$	10
write (x, A);		11
	write (x, A);	11

Serialisierbarkeit

Eine verschränkte Ausführung mehrerer Transaktionen heißt **serialisierbar**, wenn ihr Effekt identisch zum Effekt einer (beliebig gewählten) seriellen Ausführung dieser Transaktionen ist.

- **Schedule:** „Ablaufplan“ für Transaktion, bestehend aus Abfolge von Transaktionsoperationen

Transaktionen in SQL

Aufweichung von ACID in SQL: Isolationsebenen

```
set transaction
  [ { read only | read write }, ]
  [isolation level
    { read uncommitted |
      read committed |
      repeatable read |
      serializable }, ]
  [ diagnostics size ...]
```

Standardeinstellung:

```
set transaction read write,
           isolation level serializable
```

- **read uncommitted**
 - schwächste Stufe: Zugriff auf nicht geschriebene Daten, nur für *read only* Transaktionen
 - statistische und ähnliche Transaktionen (ungefährer Überblick, nicht korrekte Werte)
 - keine Sperren → effizient ausführbar, keine anderen Transaktionen werden behindert
- **read committed**
 - nur Lesen endgültig geschriebener Werte, aber *nonrepeatable read* möglich
- **repeatable read**
 - kein *nonrepeatable read*, aber Phantomproblem kann auftreten
- **serializable**
 - garantierte Serialisierbarkeit

Isolationsebenen: read committed

	T_1	T_2
	set transaction isolation level read committed	
1	select Name from WEINE where WeinID = 1014 → <i>Riesling</i>	
2		update WEINE set Name = 'Riesling Superiore', where WeinID = 1014
3	select Name from WEINE where WeinID = 1014 → <i>Riesling</i>	
4		commit
5	select Name from WEINE where WeinID = 1014 → <i>Riesling Superiore</i>	

read committed /2

	T_1	T_2
	set transaction isolation level read committed	
1	select Name from WEINE where WeinID = 1014	
2		update WEINE set Name = 'Riesling Superore' where WeinID = 1014
3	update WEINE set Name = 'Superiore Riesling' where WeinID = 1014 → blockiert	
4		commit
5	commit	

Isolationsebenen: serializable

	T_1	T_2
	set transaction isolation level serializable	
1	select Name into N from WEINE where WeinID = 1014 → N := Riesling	
2		update WEINE set Name = 'Riesling Superiore' where WeinID = 1014
4		commit
5	update WEINE set Name = 'Superior' N where WeinID = 1014 → Abbruch	

Integritätsbedingungen in SQL

Integritätsbedingungen in SQL-DDL

- `not null`: Nullwerte verboten
- `default`: Angabe von Default-Werten
- `check (search-condition)`: Attributspezifische Bedingung
(in der Regel *Ein-Tupel-Integritätsbedingung*)
- `primary key`: Angabe eines Primärschlüssel
- `foreign key (Attribut(e))
references Tabelle(Attribut(e))`:
Angabe der referentiellen Integrität

- `create domain`: Festlegung eines benutzerdefinierten Wertebereichs
- Beispiel

```
create domain WeinFarbe varchar(4)
    default 'Rot'
    check (value in ('Rot', 'Weiß', 'Rose'))
```

- Anwendung

```
create table WEINE (
    WeinID int primary key,
    Name varchar(20) not null,
    Farbe WeinFarbe, ...)
```

- **check:** Festlegung weitere lokale Integritätsbedingungen innerhalb der zu definierenden Wertebereiche, Attribute und Relationenschemata
- Beispiel: Einschränkung der zulässigen Werte
- Anwendung

```
create table WEINE (
    WeinID int primary key,
    Name varchar(20) not null,
    Jahr int check(Jahr between 1980 and 2010),
    ...
)
```

Erhaltung der referentiellen Integrität

- Überprüfung der Fremdschlüsselbedingungen nach Datenbankänderungen
- für $\pi_A(r_1) \subseteq \pi_K(r_2)$,
z.B. $\pi_{\text{Weingut}}(\text{WEINE}) \subseteq \pi_{\text{Weingut}}(\text{ERZEUGER})$
 - Tupel t wird eingefügt in $r_1 \Rightarrow$ überprüfen, ob $t' \in r_2$ existiert mit:
 $t'(K) = t(A)$, d.h. $t(A) \in \pi_K(r_2)$
falls nicht \Rightarrow abweisen
 - Tupel t' wird aus r_2 gelöscht \Rightarrow überprüfen, ob $\sigma_{A=t'(K)}(r_1) = \{\}$,
d.h. kein Tupel aus r_1 referenziert t'
falls nicht leer \Rightarrow abweisen oder Tupel aus r_1 , die t' referenzieren,
löschen (bei kaskadierendem Löschen)

- `on update | delete`
Angabe eines Auslöseereignisses, das die Überprüfung der Bedingung anstößt
- `cascade — set null — set default — no action`
Kaskadierung: Behandlung einiger Integritätsverletzungen pflanzt sich über mehrere Stufen fort, z.B. Löschen als Reaktion auf Verletzung der referentieller Integrität
- `deferred — immediate` legt Überprüfungszeitpunkt für eine Bedingung fest
 - `deferred`: Zurückstellen an das Ende der Transaktion
 - `immediate`: sofortige Prüfung bei jeder relevanten Datenbankänderung

Überprüfungsmodi: Beispiel

- Kaskadierendes Löschen

```
create table WEINE (
    WeinID int primary key,
    Name varchar(50) not null,
    Preis float not null,
    Jahr int not null,
    Weingut varchar(30),
    foreign key (Weingut)
        references ERZEUGER (Weingut)
        on delete cascade)
```

Die assertion-Klausel

- Assertion: Prädikat, das eine Bedingung ausdrückt, die von der Datenbank immer erfüllt sein muss
- Syntax (SQL:2003)

```
create assertion name check ( prädikat )
```

- Beispiele:

```
create assertion Preise check
  ( ( select sum (Preis)
      from WEINE) < 10000 );
create assertion Preise2 check
  ( not exists (
      select * from WEINE where Preis > 200) )
```

Trigger

- Trigger: Anweisung/Prozedur, die bei Eintreten eines bestimmten Ereignisses automatisch vom DBMS ausgeführt wird
- Anwendung:
 - Erzwingen von Integritätsbedingungen („Implementierung“ von Integritätsregeln)
 - Auditing von DB-Aktionen
 - Propagierung von DB-Änderungen
- Definition:

```
create trigger ...
  after Operation
    Anweisungen
```

Beispiel für Trigger

- Realisierung eines berechneten Attributs durch zwei Trigger:
 - Einfügen von neuen Aufträgen

```
create trigger Auftragszählung+
  on insertion of Auftrag A:
  update Kunde
    set AnzAufträge = AnzAufträge + 1
      where KName = new A.KName
```

- analog für Löschen von Aufträgen:

```
create trigger Auftragszählung-
  on deletion ....:
  update .... - 1 ...
```

- Spezifikation von
 - *Ereignis* und *Bedingung* für Aktivierung des Triggers
 - *Aktion(en)* zur Ausführung
- Syntax in SQL:2003 festgelegt
- verfügbar in den meisten kommerziellen Systemen (aber mit anderer Syntax)

- Syntax:

```
create trigger Name
after | before Ereignis
on Relation
[ when Bedingung ]
begin atomic SQL-Anweisungen end
```

- Ereignis:

- insert
- update [of *Attributliste*]
- delete

Weitere Angaben bei Triggern

- for each row bzw. for each statement: Aktivierung des Triggers für jede Einzeländerungen einer mengenwertigen Änderung oder nur einmal für die gesamte Änderung
- before bzw. after: Aktivierung vor oder nach der Änderung
- referencing new as bzw. referencing old as: Binden einer Tupelvariable an die neu eingefügten bzw. gerade gelöschten („alten“) Tupel einer Relation
 - ~~> Tupel der *Differenzrelationen*

Beispiel für Trigger

- *Kein Kundenkonto darf unter 0 absinken:*

```
create trigger bad_account
after update of Kto on KUNDE
referencing new as INSERTED
when (exists
      (select * from INSERTED where Kto < 0)
)
begin atomic
    rollback;
end
```

~~~ ähnlicher Trigger für insert

# Beispiel für Trigger /2

- *Erzeuger müssen gelöscht werden, wenn sie keine Weine mehr anbieten:*

```
create trigger unnützes_Weingut
after delete on WEINE
referencing old as O
for each row
when (not exists
      (select * from WEINE W
       where W.Weingut = O.Weingut))
begin atomic
    delete from ERZUEGER where Weingut = O.Weingut;
end
```

1. Bestimme Objekt  $o_i$ , für das die Bedingung  $\phi$  überwacht werden soll
  - i.d.R. mehrere  $o_i$  betrachten, wenn Bedingung relationsübergreifend ist
  - Kandidaten für  $o_i$  sind Tupel der Relationsnamen, die in  $\phi$  auftauchen
2. Bestimme die elementaren Datenbankänderungen  $u_{ij}$  auf Objekten  $o_i$ , die  $\phi$  verletzen können
  - Regeln: z.B. Existenzforderungen beim Löschen und Ändern prüfen, jedoch nicht beim Einfügen etc.

3. Bestimme je nach Anwendung die Reaktion  $r_i$  auf Integritätsverletzung
  - Rücksetzen der Transaktion (rollback)
  - korrigierende Datenbankänderungen
4. Formuliere folgende Trigger:

```
create trigger t-phi-ij after uij on oi
when  $\neg\phi$ 
begin ri end
```

5. Wenn möglich, vereinfache entstandenen Trigger

# Trigger in Oracle

- Implementierung in PL/SQL
- Notation

```
create [ or replace ] trigger trigger-name
       before | after
       insert or update [ of spalten ]
                      or delete on tabelle
       [ for each row
       [ when ( prädikat ) ] ]
PL/SQL-Block
```

- Anweisungsebene (*statement level trigger*): Trigger wird ausgelöst vor bzw. nach der DML-Anweisung
- TupelEbene (*row level trigger*): Trigger wird vor bzw. nach jeder einzelnen Modifikation ausgelöst (*one tuple at a time*)

Trigger auf TupelEbene:

- Prädikat zur Einschränkung (*when*)
- Zugriff auf altes (:old.col) bzw. neues (:new.col) Tupel
  - für delete: nur (:old.col)
  - für insert: nur (:new.col)
  - in when-Klausel nur (new.col) bzw. (old.col)

# Trigger in Oracle /2

- Transaktionsabbruch durch `raise_application_error(code, message)`
- Unterscheidung der Art der DML-Anweisung

```
if deleting then ... end if;  
if updating then ... end if;  
if inserting then ... end if;
```

# Trigger in Oracle: Beispiel

- *Kein Kundenkonto darf unter 0 absinken:*

```
create or replace trigger bad_account
after insert or update of Kto on KUNDE
for each row
when (:new.Kto < 0)
begin
    raise_application_error(-20221,
        'Nicht unter 0');
end;
```

# Schemaevolution

- Änderung eines Datenbankschemas durch neue/veränderte Anforderungen
  - Hinzufügen oder Löschen von Tabellen, Spalten, Integritätsbedingungen
  - Umbenennen oder Datentypänderungen
- erfordert oft auch Anpassung/Übertragung der vorhandenen Datenbank ↵ **Datenbankmigration**
- leider nur eingeschränkte Unterstützung durch DB-Werkzeuge (DDL + Export/Import der Daten)

# SQL-DDL zum Löschen von Tabellen

- Löschen von Tabellendefinitionen  
(beachte Unterschied zu delete)

```
drop table relationename [ restrict | cascade ]
```

- **cascade**: erzwingt Löschen aller Sichten und Integritätsbedingungen, die zu dieser Basisrelation gehören
- **restrict** (Defaultfall): das drop-Kommando wird zurückgewiesen, falls noch solche Sichten und Integritätsbedingungen existieren

# SQL-DDL zur Änderung von Tabellen

```
alter table relationenname modifikation
```

- `add column spaltendefinition` fügt eine neue Spalte hinzu; alle bereits in der Tabelle existierenden Tupel erhalten als Wert der neuen Spalte den angegebenen Defaultwert bzw. den null-Wert
- `drop column spaltenname` löscht die angegebene Spalte (inkl. restrict- bzw. cascade)
- `alter column spaltenname set default defaultwert` verändert Defaultwert der Spalte

# Änderung von Tabellen: Beispiele



```
alter table WEINE  
    add column Preis decimal(5,2)
```

```
alter table WEINE  
    alter column Jahrgang set default 2007
```

# Änderung von Integritätsbedingungen

- nachträgliches Hinzufügen/Löschen von Tabellenbedingungen über `alter table`
- Vergabe von Namen für Bedingungen über constraint *bed-name*-Klausel

```
alter table WEINE
    add constraint WeinBed_Eindeutig
        unique (Name, Weingut)
```

- Löschen über Namen

```
alter table WEINE
    drop constraint WeinBed_Eindeutig
```

# Zusammenfassung

- Zusicherung von Korrektheit bzw. Integrität der Daten
- inhärente Integritätsbedingungen des Relationenmodells
- zusätzliche SQL-Integritätsbedingungen: check-Klausel, assertion-Anweisung
- Trigger zur „Implementierung“ von Integritätsbedingungen bzw. -regeln

# Kontrollfragen

- Welchem Zweck dient die Integritätssicherung?  
Welche Formen von Integritätsbedingungen gibt es?
- Wie lassen sich Integritätsbedingungen und -regeln in SQL-Systemen formulieren?
- Welche Forderungen ergeben sich aus dem ACID-Prinzip? Wie werden diese in Datenbanksystemen erreicht?



## Teil XI

# Sichten und Zugriffskontrolle

1. Sichtenkonzept
2. Änderungen auf Sichten
3. Rechtevergabe
4. Privacy-Aspekte

# Lernziele für heute . . .

- Verständnis des Sichtenkonzeptes von Datenbanken
- Kenntnisse zur Formulierung und Nutzung von Sichten in SQL
- Verständnis der Probleme bei Änderungen über Sichten
- Verständnis zu Datenschutzaspekten im Zusammenhang mit aggregierten/statistischen Daten



# Sichtenkonzept

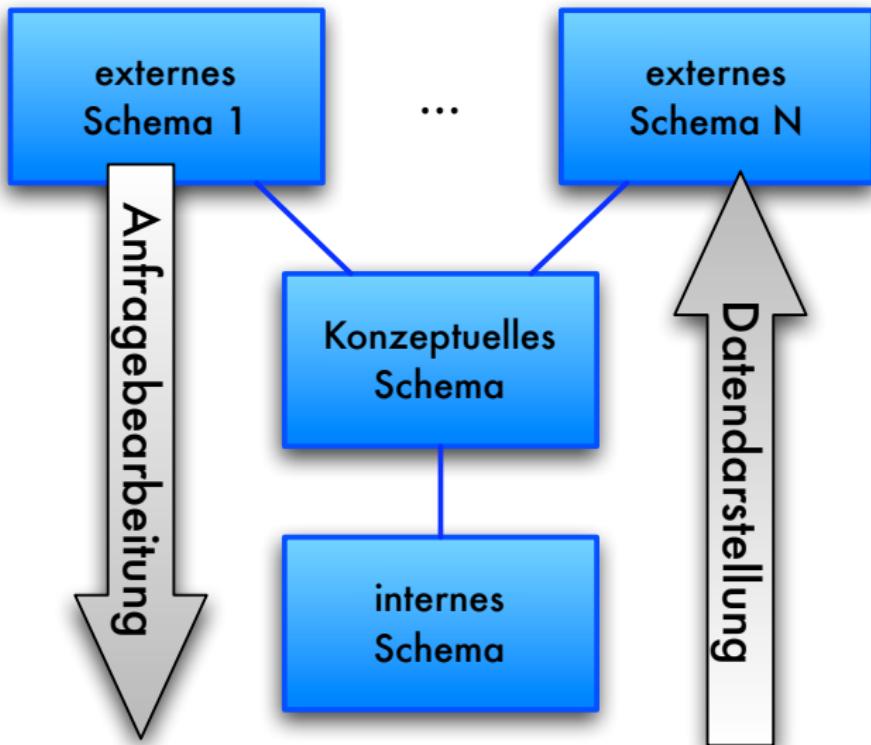
## Sichten

**virtuelle Relationen** (bzw virtuelle Datenbankobjekte in anderen Datenmodellen) (englisch **view**)

- Sichten sind externe DB-Schemata folgend der 3-Ebenen-Schemaarchitektur
- Sichtdefinition
  - Relationenschema (implizit oder explizit)
  - Berechnungsvorschrift für virtuelle Relation, etwa SQL-Anfrage

- Vorteile
  - **Vereinfachung von Anfragen** für den Benutzer der Datenbank, etwa indem oft benötigte Teilanfragen als Sicht realisiert werden
  - Möglichkeit der **Strukturierung der Datenbankbeschreibung**, zugeschnitten auf Benutzerklassen
  - **logische Datenunabhängigkeit** ermöglicht Stabilität der Schnittstelle für Anwendungen gegenüber Änderungen der Datenbankstruktur
  - Beschränkung von Zugriffen auf eine Datenbank im Zusammenhang mit der **Zugriffskontrolle**
- Probleme
  - automatische Anfragetransformation
  - Durchführung von Änderungen auf Sichten

# Drei-Ebenen-Schema-Architektur



# Definition von Sichten in SQL



```
create view SichtName [ SchemaDeklaration ]  
as SQLAnfrage  
[ with check option ]
```

# Sichten - Beispiel

- alle Rotweine aus Bordeaux

```
create view Rotweine as
    select Name, Jahrgang, WEINE.Weingut
    from WEINE natural join ERZEUGER
    where Farbe = 'Rot'
        and Region = 'Bordeaux'
```

## Änderungen auf Sichten

- **Effektkonformität**

Benutzer sieht Effekt als wäre die Änderung auf der Sichtrelation direkt ausgeführt worden

- **Minimalität**

Basisdatenbank sollte nur **minimal geändert werden**, um den erwähnten Effekt zu erhalten

- **Konsistenzerhaltung**

Änderung einer Sicht darf zu **keinen Integritätsverletzungen** der Basisdatenbank führen

- **Respektierung des Datenschutzes**

Wird die Sicht aus Datenschutzgründen eingeführt, **darf der bewusst ausgeblendete Teil der Basisdatenbank von Änderungen der Sicht nicht betroffen werden**

# Projektionssicht

$$\text{WNW} := \pi_{\text{WeinID}, \text{Name}, \text{Weingut}}(\text{WEINE})$$

- In SQL mit create view-Anweisung:

```
create view WNW as
    select WeinID, Name, Weingut from WEINE
```

# Projektionssicht /2

- Änderungsanweisung für die Sicht WNW:

```
insert into WNW
values (3333, 'Dornfelder', 'Müller')
```

- Korrespondierende Anweisung auf der Basisrelation WEINE:

```
insert into WEINE
values (3333, 'Dornfelder',
        null, null, 'Müller')
```

→ Problem der **Konsistenzerhaltung** falls Farbe oder Jahrgang als not null deklariert!

# Selektionssichten

$$WJ := \sigma_{\text{Jahrgang} > 2000}(\pi_{\text{WeinID}, \text{Jahrgang}}(\text{WEINE}))$$

```
create view WJ as
select WeinID, Jahrgang
from WEINE
where Jahrgang > 2000
```

# Selektionssichten /2

- Tupelmigration: Tupel  
WEINE(3456, 'Zinfandel', 'Rot', 2004, 'Helena'), wird aus der Sicht „herausbewegt“:

```
update WJ
set Jahrgang = 1998
where WeinID = 3456
```

# Kontrolle der Tupelmigration

```
create view WJ as
select WeinID, Jahrgang
from WEINE
where Jahrgang > 2000
with check option
```

WE := WEINE ⋈ ERZEUGER

- In SQL:

```
create view WE as
select WeinID, Name, Farbe, Jahrgang,
       WEINE.Weingut, Anbaugebiet, Region
  from WEINE, ERZEUGER
 where WEINE.Weingut = ERZEUGER.Weingut
```

# Verbundsichten /2

- Änderungsoperationen hier in der Regel nicht eindeutig übersetzbare:

```
insert into WE
values (3333, 'Dornfelder', 'Rot', 2002,
        'Helena', 'Barossa Valley', 'Südaustralien')
```

- Änderung wird transformiert zu

```
insert into WEINE
values (3333, 'Dornfelder', 'Rot', 2002,
        'Helena')
```

- plus Änderung auf ERZEUGER !

- zusätzliche Aktionen auf ERZEUGER

1. Einfügeanweisung auf ERZEUGER:

```
insert into ERZEUGER
values ('Helena', 'Barossa Valley', 'Südaustralien')
```

2. oder alternativ:

```
update ERZEUGER
set Anbaugebiet = 'Barossa Valley',
    Region = 'Südaustralien'
where Weingut = 'Helena'
```

besser bzgl. **Minimalitätsforderung**, widerspricht aber  
**Effektkonformität!**

# Aggregierungssichten

```
create view FM (Farbe, MinJahrgang) as
select Farbe, min(Jahrgang)
from WEINE
group by Farbe
```

- Folgende Änderung ist nicht eindeutig umsetzbar:

```
update FM
set MinJahrgang = 1993
where Farbe = 'Rot'
```

# Klassifikation der Problembereiche

1. Verletzung der Schemadefinition (z.B. Einfügen von Nullwerten bei Projektionssichten)
2. Datenschutz: Seiteneffekte auf nicht-sichtbaren Teil der Datenbank vermeiden (Tupelmigration, Selektionssichten)
3. nicht immer eindeutige Transformation: Auswahlproblem
4. Aggregierungssichten (u.a.): keine sinnvolle Transformation möglich
5. elementare Sichtänderung soll genau einer atomaren Änderung auf Basisrelation entsprechen: 1:1-Beziehung zwischen Sichttupeln und Tupeln der Basisrelation (kein Herausprojizieren von Schlüsseln)

- SQL-92-Standard
  - Integritätsverletzende Sichtänderungen nicht erlaubt
  - datenschutzverletzende Sichtänderungen: Benutzerkontrolle (*with check option*)
  - Sichten mit nicht-eindeutiger Transformation: Sicht nicht änderbar (SQL-92 restriktiver als notwendig)

# Einschränkungen für Sichtänderungen

- änderbar nur Selektions- und Projektionssichten (Verbund und Mengenoperationen nicht erlaubt)
- 1:1-Zuordnung von Sichttupeln zu Basistupeln: kein `distinct` in Projektionssichten
- Arithmetik und Aggregatfunktionen im `select`-Teil sind verboten
- genau eine Referenz auf einen Relationsnamen im `from`-Teil erlaubt (auch kein Selbstverbund)
- keine Unteranfragen mit „Selbstbezug“ im `where`-Teil erlaubt (Relationsname im obersten SFW-Block nicht in `from`-Teilen von Unteranfragen verwenden)
- `group by` und `having` verboten

- seit SQL:2003 Aufhebung einiger Einschränkungen, insbesondere
  - Updates auf `union all`-Sichten (ohne Duplikateliminierung)
  - Inserts in Verbundsichten mit Primär-/Fremdschlüsselbeziehungen (mit einigen Einschränkungen)
  - Updates auf Verbundsichten mit Cursor (siehe folgendes Kapitel)

# Alternative: Sichtänderungen mit Instead-of-Triggern

- Definition von Triggern auf Sichten zur anwendungsspezifischen Propagierung der Änderungen auf die Basistabellen

```
create view V_WEINERZEUGER as
    select * from WEINE natural join ERZEUER;

create trigger V_WEINERZEUGER_Insert
    instead of insert on V_WEINERZEUGER
referencing new as N
for each row
begin
    insert into WEINE values (:N.WeinID, :N.Name,
        :N.Farbe, :N.Jahrgang, :N.Weingut);
end;
```

# Auswertung von Anfragen an Sichten

- `select`: Sichtattribute evtl. umbenennen bzw. durch Berechnungsterm ersetzen
- `from`: Namen der Originalrelationen
- konjunktive Verknüpfung der `where`-Klauseln von Sichtdefinition und Anfrage (evtl. Umbenennungen)
- Vorsicht bei Aggregationssichten!
  - `having` versus `where`
  - keine geschachtelten Aggregationen in SQL

# Rechtevergabe

- *Zugriffsrechte*

(AutorisierungsID, DB-Ausschnitt, Operation)

- AutorisierungsID ist interne Kennung eines „Datenbankbenutzers“
- Datenbank-Ausschnitte: Relationen und Sichten
- DB-Operationen: Lesen, Einfügen, Ändern, Löschen

# Rechtevergabe in SQL

```
grant Rechte
on Tabelle
to BenutzerListe
[with grant option]
```

# Rechtevergabe in SQL /2

- Erläuterungen:
  - In *Rechte*-Liste: all bzw. Langform all privileges oder Liste aus select, insert, update, delete
  - Hinter on: Relationen- oder Sichtname
  - Hinter to: Autorisierungidentifikatoren (auch public, group)
  - spezielles Recht: Recht auf die Weitergabe von Rechten (with grant option)

# Autorisierung für public



```
create view MeineAufträge as
select *
from AUFTTRAG
where KName = user;

grant select, insert
on MeineAufträge
to public;
```

*„Jeder Benutzer kann seine Aufträge sehen und neue Aufträge einfügen (aber nicht löschen!).“*

```
revoke Rechte
on Tabelle
from BenutzerListe
[restrict | cascade ]
```

- **restrict:** Falls Recht bereits an Dritte weitergegeben: Abbruch von `revoke`
- **cascade:** Rücknahme des Rechts mittels `revoke` an alle Benutzer propagiert, die es von diesem Benutzer mit `grant` erhalten haben

# Privacy-Aspekte

## Privacy (Privatsphäre)

das Recht jedes Einzelnen auf einen geschützten privaten Raum, der von anderen nur in definierten Ausnahmefällen verletzt werden darf

- elektronische Autobahn-Mautsysteme: Überwachung von Fahrzeugen
- Kreditkartenaktivitäten und diverse Payback- bzw. Rabattkarten: Kaufverhalten von Kunden
- Mobilfunksysteme: Bewegungsprofile der Nutzer
- RFID-Technologie: etwa im Einzelhandel Kaufverhalten, Warenflüsse, etc.

- Datenbanken, in denen die Einzeleinträge dem Datenschutz unterliegen, aber statistische Informationen allen Benutzern zugänglich sind
- statistische Information = aggregierte Daten  
(Durchschnittseinkommen etc.)
- Problem: Gewinnung von Einzelinformationen durch indirekte Anfragen

# Statistische Datenbanken: Beispiel

- Benutzer  $X$  darf Daten über Kontoinhaber sowie statistische Daten abfragen, jedoch keine einzelnen Kontostände
  1. Verfeinerung des Suchkriteriums (nur ein Kunde)

```
select count (*) from KONTO
where Ort = 'Manebach' and Alter = 24 and ...
```

2. Name des Kontoinhabers

```
select Name from KONTO
where Ort = 'Manebach' and Alter = 24 and ...
```

3. statistische Anfrage, die tatsächlich aber Einzeleintrag liefert

```
select sum(Kontostand) from KONTO
where Ort = 'Manebach' and Alter = 24 and ...
```

- Abhilfe: keine Anfragen, die weniger als  $n$  Tupel selektieren

# Statistische Datenbanken: Beispiel /2

- $X$  will Kontostand von  $Y$  herausfinden
- $X$  weiss, dass  $Y$  nicht in Ilmenau lebt
- $X$  hat abgefragt, dass in Ilmenau mehr als  $n$  Kontoinhaber leben

1. Gesamtkontostand der Ilmenauer Kunden

```
select sum(Kontostand) from KONTO
where Ort = 'Ilmenau'
```

2. Gesamtkontostand der Ilmenauer Kunden + Kunde  $Y$

```
select sum(Kontostand) from KONTO
where Name = :Y or Ort = 'Ilmenau'
```

3. Differenz der Ergebnisse liefert Kontostand von  $Y$

- Abhilfe: statistische Anfragen nicht erlauben, die paarweise einen Durchschnitt von mehr als  $m$  vorgegebenen Tupeln betreffen

- kritische Parameter
  - Ergebnisgröße  $n$
  - Größe der Überlappung der Ergebnismengen  $m$

Sind nur Ergebnisse von Aggregatfunktionen erlaubt, dann benötigt eine Person  $1 + (n - 2)/m$  Anfragen, um einen einzelnen Attributwert zu ermitteln

# k-Anonymität

- für viele Zwecke (klinische Studien etc.) werden auch Detaildaten (Mikrodaten) benötigt

| Name  | Alter | PLZ   | Geschlecht | FamStand | Krankheit    |
|-------|-------|-------|------------|----------|--------------|
| ***** | 38    | 98693 | männl.     | verh.    | Schnupfen    |
| ***** | 29    | 39114 | weibl.     | ledig    | Fieber       |
| ***** | 29    | 39114 | weibl.     | ledig    | Anämie       |
| ***** | 34    | 98693 | männl.     | verh.    | Husten       |
| ***** | 34    | 98693 | männl.     | verh.    | Knochenbruch |
| ***** | 27    | 18055 | weibl.     | ledig    | Fieber       |
| ***** | 27    | 18055 | weibl.     | ledig    | Schnupfen    |

# k-Anonymität: Problem

- ist von einer Person aus dieser Relation bekannt, dass sie
  - männlich
  - 38 Jahre alt
  - verheiratet ist
  - in 98693 Ilmenau wohnt
- $\rightsquigarrow$  Schnupfen
- weitere Zuordnungen (Namen etc.) etwa durch Verbund mit anderen Daten möglich?
- Lösung: Data Swapping (??)
  - Vertauschen von Attributwerten einzelner Tupel
  - statistische Analysen noch gültig?

## k-Anonymität

ein bestimmter Sachverhalt kann nicht zwischen einer vorgegebenen Anzahl  $k$  von Tupeln unterschieden werden

- eine Anfrage nach einer beliebigen Kombination von Alter, Geschlecht, Familienstand und Postleitzahl liefert entweder eine leere Relation oder mindestens  $k$  Tupel

- **Generalisierung:** Attributwerte durch allgemeinere Werte ersetzen, die einer Generalisierungshierarchie entnommen sind
  - die Verallgemeinerung des Alters einer Person zu Altersklassen: {35, 39}  $\rightsquigarrow$  30-40
  - Weglassen von Stellen bei Postleitzahlen: { 39106, 39114 }  $\rightsquigarrow$  39\*\*\*
- **Unterdrücken von Tupeln:** Löschen von Tupeln, welche die  $k$ -Anonymität verletzen und damit identifizierbar sind

# Zusammenfassung

- Sichten zur Strukturierung von Datenbanken
- Probleme bei Änderungen über Sichten
- Rechtesystem in SQL-DBS
- Privacy-Aspekte

# Kontrollfragen

- Was versteht man unter einer Datenbank-Sicht?  
Wie werden Sichten definiert?
- Sind Sichten änderbar? Unter welchen Bedingungen?
- Wie kann in Datenbanken der Datenschutz erreicht werden?



# Teil XII

## NoSQL

1. Motivation für NoSQL
2. Datenmodelle für NoSQL
3. KV-Stores und Wide Column
4. Document Stores
5. Graph Stores

# Motivation für NoSQL

# Motivation für NoSQL

NoSQL = Not only SQL

- im Umfeld vieler aktueller Buzzwords
  - NoSQL
  - Big Data
  - BASE  
*für Basically Available, Soft State, Eventually Consistent*
  - ...
- oft einfach als Etikett einer Neuentwicklung eines DBMS pauschal vergeben

# Was ist NoSQL?

- **SQL - No!**
  - SQL-Datenbanken sind zu komplex, nicht skalierbar, ...
  - man braucht was einfacheres!
- **Not only SQL**
  - SQL-Datenbanken haben zu wenig (oder die falsche) Funktionalität
  - Operationen auf Graphen, Data Mining Operatoren, ...
- **New SQL**
  - SQL-Datenbanken sind (software-technisch) in die Jahre gekommen
  - eine neue Generation von DBMS muss her (ohne die etablierten Vorteile von SQL zu ignorieren)

- nicht skalierbar
  - Normalisierung von Relationen, viele Integritätsbedingungen zu prüfen
  - **kann man in RDBMS auch vermeiden!**
- starre Tabellen nicht flexibel genug
  - schwach typisierte Tabellen (Tupel weichen in den tatsächlich genutzten Attributen ab)
    - viele Nullwerte wenn alle potentiellen Attribute definiert
    - alternativ Aufspaltung auf viele Tabellen
    - Schema-Evolution mit `alter table` skaliert bei Big Data nicht
  - **tatsächlich in vielen Anwendungen ein Problem**
- Integration von spezifischen Operationen (Graphtraversierung, Data-Mining-Primitive) mit Stored Procedures zwar möglich führt aber oft zu schwer interpretierbarem Code

# Datenmodelle für NoSQL

# Datenmodelle für NoSQL



- KV-Stores
- Wide Column Stores
- Dokumenten-orientierte Datenhaltung
- Graph-Speicher
- ....

# Anfragesprachen für NoSQL

- unterschiedliche Ansätze:
  - einfache funktionale API
  - Programmiermodell für parallele Funktionen
  - angelehnt an SQL-Syntax
  - ....

## KV-Stores und Wide Column

- *Key-Value-Store*: binäre Relationen, bestehend aus
  - einem Zugriffsschlüssel (dem Key) und
  - den Nutzdaten (dem Value)
- Nutzdaten
  - binäre Daten ohne Einschränkung,
  - Dateien oder Dokumente,  
→ *Document Databases*
  - oder schwachstrukturierte Tupel  
→ *Wide Column Store*

# Anfragen an KV-Stores

- einfache API

```
store.put(key, value)
value = store.get(key)
store.delete(key)
```

- aufgesetzte höherer Sprache angelehnt an SQL
- Map-Reduce
  - Framework zur Programmierung paralleler Datenaggregation auf KV-Stores

# Beispielsysteme für KV-Stores

- Amazon DynamoDB
- Riak

# Datenmodell: Wide Column

- Basisidee: KV-Store mit schwachstrukturiertem Tupel als Value
  - Value = Liste von Attributname-Attributwert-Paaren
    - schwache Typisierung für Attributwerte (auch Wiederholgruppen)
  - nicht alle Einträge haben die selben Attributnamen
    - offene Tupel
    - Hinzufügen eines neuen Attributs unproblematisch
    - Nullwerte aus SQL ersetzt durch fehlende Einträge
- Beispiel in DynamoDB

# Datenmodell: Wide Column /2

| Key        | Value (Attributliste) |                           |                   |
|------------|-----------------------|---------------------------|-------------------|
| WeinID = 1 | Name = Zinfandel      | Farbe = Rot               | Jahrgang = 2004   |
| WeinID = 2 | Name = Pinot Noir     | Weingut = {Creek, Helena} |                   |
| WeinID = 3 | Name = Chardonnay     | Jahrgang = 2002           | Weingut = Bighorn |

# Anfragen bei Wide Column

- *CRUD*: Create, Read, Update und Delete
  - PutItem fügt einen neuen Datensatz mit der gegebenen Attribut-Wert-Liste ein bzw. ersetzt einen existierenden Datensatz mit gleichem Schlüssel.
  - GetItem-Operation liest alle Felder eines über einen Primärschlüssel identifizierten Datensatzes.
  - Scan erlaubt einen Lauf über alle Datensätze mit Angabe von Filterkriterien.
- Aufruf über HTTP oder aus Programmiersprachen heraus

# Beispielanfrage in DynamoDB

```
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.GetItem
content-type: application/x-amz-json-1.0

{
    "TableName": "Weine",
    "Key": {
        "HashKeyElement": {"N": "1" },
        "RangeKeyElement": {"S": "Zinfandel" }
    },
    "AttributesToGet": ["Farbe", "Jahrgang"],
    "ConsistentRead": false
}
```

- Primärschlüssel (HashKeyElement) ist numerisch (N)
- Feld Name ist Bereichsschlüssel vom Typ String (S)

# Beispielanfrage in DynamoDB: Ergebnis



```
HTTP/1.1 200
x-amzn-RequestId: ...
content-type: application/x-amz-json-1.0
content-length: ...

{"Item":
  {"Farbe": {"S": "Rot" },
   "Jahrgang": {"N": "2004" }
  },
  "ConsumedCapacityUnits": 0.5
}
```

# Document Stores

- Basisidee: KV-Store mit (hierarchisch) strukturiertem Dokument als Value
- strukturiertes Dokument:
  - JSON-Format
    - geschachtelte Wide Column-Daten
  - XML (eher unüblich auf KV-Stores)

# Beispiel für Dokument in JSON



```
{  
    "id" : "kritiker08154711",  
    "Name" : "Bond",  
    "Vorname" : "Jamie",  
    "Alter" : 42,  
    "Adresse" :  
    {  
        "Strasse" : "Breiter Weg 1",  
        "PLZ" : 39007,  
        "Stadt" : "Machdeburch"  
    },  
    "Telefon" : [7007, 110]  
}
```

# Anfragen bei dokumentenorientierter Speicherung

- CRUD erweitert um dokumentspezifische Suche
- Beispiele (MongoDB mit BSON statt JSON)

```
db.kritiker.find({Name: "Bond"})
db.kritiker.find({Alter: 40})
db.kritiker.find({Alter{$lt: 50}})
db.kritiker.find({Name: "Bond", Alter: 42})
db.kritiker.find($or[ {Name: "Bond"} ,
{ Alter: 42} ] )
```

# Beispielsysteme für dokumentenorientierte Speicherung

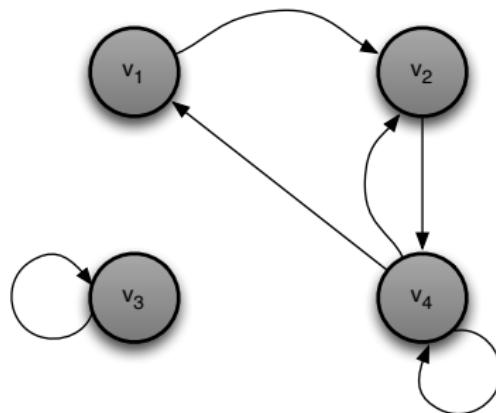
- MongoDB
- CouchDB

# Graph Stores

# Graph-Datenmodelle: Grundlagen

- spezielle Form der Datenrepräsentation = Graphen, insb. Beziehungen zwischen Objekten
- Anwendungsgebiete:
  - Transportnetze
  - Networking: Email-Verkehr, Mobilfunk-Nutzer
  - Soziale Netzwerke: Eigenschaften, Communities
  - Web: Verlinkte Dokumente
  - Chemie: Struktur chemischer Komponenten
  - Bioinformatik: Proteinstrukturen, metabolische Pathways, Genexpressionen

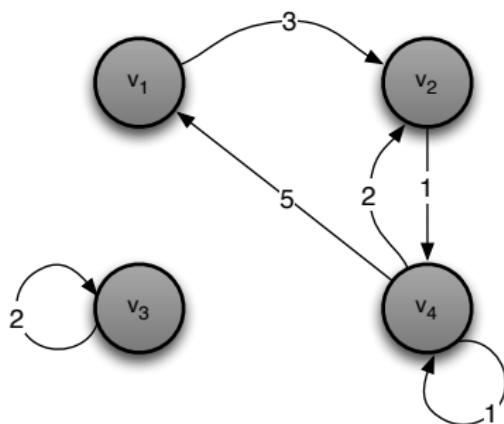
- Graph  $G = (V, E)$ 
  - $V$ : Menge der Knoten (vertices)
  - $E \subseteq V \times V$ : Menge der Kanten (edges)



- Kanten können mit Gewicht versehen werden

# Grundbegriffe: Adjazenzmatrix

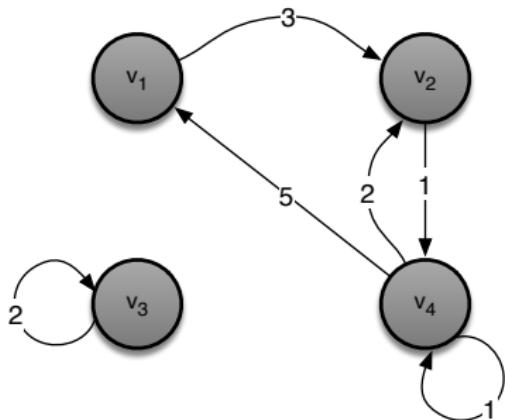
- Repräsentation von Graphen durch Matrix (Knoten als Zeilen und Spalten)
- ungerichteter Graph: symmetrische Matrix
- ungewichteter Graph: Zellen nur 0 oder 1



|       | nach  |       |       |       |
|-------|-------|-------|-------|-------|
| von   | $v_1$ | $v_2$ | $v_3$ | $v_4$ |
| $v_1$ | 0     | 3     | 0     | 0     |
| $v_2$ | 0     | 0     | 0     | 1     |
| $v_3$ | 0     | 0     | 2     | 0     |
| $v_4$ | 5     | 2     | 0     | 1     |

# Grundbegriffe: Knotengrad

- Eigenschaft eines Knotens: Anzahl der verbundenen Knoten
- bei gerichteten Graphen: Unterscheidung in Eingangs- und Ausgangsgrad



|       | nach  |       |       |       | Ausgangsgrad |
|-------|-------|-------|-------|-------|--------------|
| von   | $v_1$ | $v_2$ | $v_3$ | $v_4$ |              |
| $v_1$ | 0     | 1     | 0     | 0     | 1            |
| $v_2$ | 0     | 0     | 0     | 1     | 1            |
| $v_3$ | 0     | 0     | 1     | 0     | 1            |
| $v_4$ | 1     | 1     | 0     | 1     | 3            |

*Eingangsgrad*

|   |   |   |   |
|---|---|---|---|
| 1 | 2 | 1 | 2 |
|---|---|---|---|

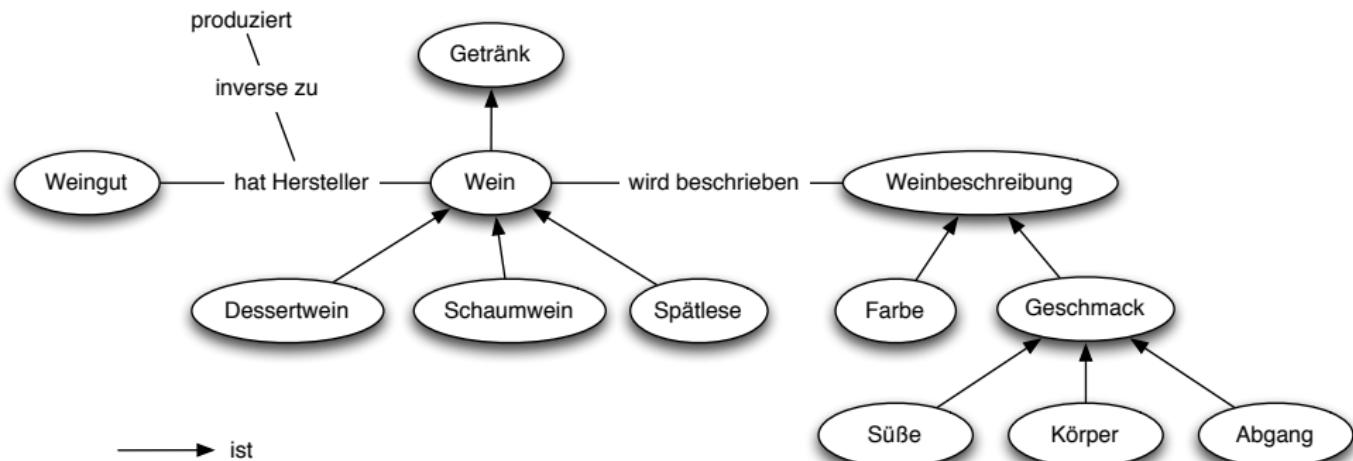
# Grundbegriffe: Traversierung

- Tiefensuche (DFS): zunächst rekursiv alle Kindknoten besuchen bevor alle Geschwisterknoten besucht werden
  - Bestimmung der Zusammenhangskomponente
  - Wegsuche im Labyrinth
- Breitensuche (BFS): zunächst alle Geschwisterknoten besuchen bevor die Kindknoten besucht werden
  - Bestimmung des kürzesten Weges

# Subjekt-Prädikat-Objekt-Modell: RDF

- Sprache zur Repräsentation von Informationen über (Web)-Ressourcen
- Ziel: automatisierte Verarbeitung
- zentraler Bestandteil von Semantic Web, Linked (Open) Data
- Repräsentation von Daten, aber auch Wissensrepräsentation (z.B. Ontologie)

- Ontologie = formale Spezifikation einer Konzeptualisierung, d.h. einer Repräsentation von Begriffen (Konzepten) und deren Beziehungen
- Anwendung: Annotation von Daten, semantische Suche



# RDF: Graphen & Tripel

- Graph = Menge von Tripeln, die Aussagen über Web-Ressourcen repräsentieren
- Identifikation der Web-Ressourcen über Uniform Resource Identifier (URI)
- Tripel:

```
subjekt prädikat objekt .
```

- Beispiel

```
<http://weindb.org/weine/3478> \  
<http://weindb.org/ontologie/name> "Pinot Noir".
```



# RDF: Graphen & Tripel

- **Subjekt:** URI-Referenz, d.h. Ressource, auf die sich die Aussage bezieht
- **Prädikat:** Eigenschaft, ebenfalls in Form einer URI-Referenz
- **Objekt:** Wert der Eigenschaft als Literal (Konstante) oder URI-Referenz

# RDF: Abkürzende Notation

- abkürzende Notation für Namensräume über Präfixe:

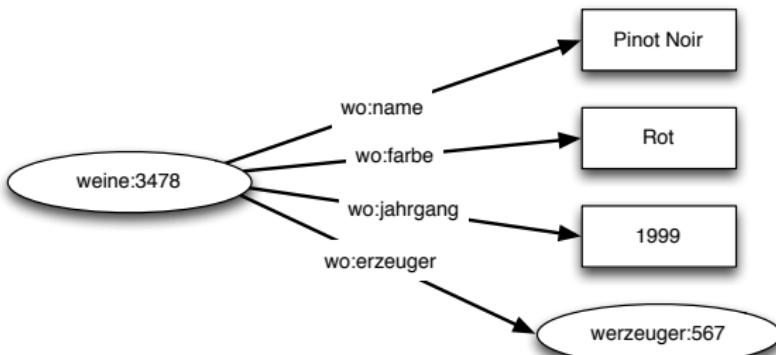
```
prefix wo: <http://weindb.org/ontologie/>
prefix weine: <http://weindb.org/weine/>

weine:2171 wo:name "Pinot Noir".
```

# RDF: Komplexe Graphen

- mehrere Aussagen zum gleichen Subjekt
- Objekte nicht nur Literale sondern selbst Objekte (URI)

```
weine:2171 wo:name "Pinot Noir".  
weine:2171 wo:farbe "Rot".  
weine:2171 wo:jahrgang "1999".  
weine:2171 wo:erzeuger werzeuger:567 .
```



- Repräsentation von RDF-Daten: N-Tripel (siehe oben), RDF/XML
- RDF Schema:
  - objektorientierte Spezifikationssprache
  - erweitert RDF um Typsystem: Definition von Klassen und Klassenhierarchien mit Eigenschaften, Ressourcen als Instanzen von Klassen
  - RDF Schema ist selbst RDF-Spezifikation

- Beispiel RDF Schema

```
Wein rdf:type rdfs:Class .  
Schaumwein rdf:type rdfs:Class .  
Schaumwein rdfs:subClassOf Wein .  
Name rdf:type rdf:Property .  
Jahrgang rdf:type rdf:Property .  
Jahrgang rdfs:domain Wein .  
Jahrgang rdfs:range xsd:integer .
```

- für komplexere Ontologien: OWL (Web Ontology Language)

- Vokabular: vordefinierte Klassen und Eigenschaften
  - Bsp: Dublin Core (Metadaten für Dokumente), FOAF (Soziale Netze), ...
  - wichtig z.B. für Linked Open Data

- SPARQL Protocol And RDF Query Language: Anfragesprache für RDF
- W3C-Recommendation
- unterschiedliche Implementierungen möglich:
  - Aufsatz für SQL-Backends (z.B. DB2, Oracle)
  - Triple Stores (RDF-Datenbank)
  - SPARQL-Endpoints
- syntaktisch an SQL angelehnt, aber Unterstützung für Graph-Anfragen

- Grundelemente: select-where-Block und Tripelmuster

```
?wein wo:name ?name .
```

- Auswertung: finden aller Belegungen (Bindung) für Variable (?name) bei Übereinstimmung mit nicht-variablen Teilen

```
<http://weindb.org/weine/2171> wo:name "Pinot Noir".  
<http://weindb.org/weine/2168> wo:name "Creek Shiraz".  
<http://weindb.org/weine/2169> wo:name "Chardonnay".
```

# SPARQL: Basic Graph Pattern

- Graphmuster (BGP = Basic Graph Pattern): Kombination von Tripelmustern über gemeinsame Variablen

```
?wein wo:name ?name .  
?wein wo:farbe ?farbe .  
?wein wo:erzeuger ?erzeuger .  
?erzeuger wo:weingut ?ename .
```

- Einsatz in SPARQL-Anfragen im where-Teil

```
select ?wein ?name ?farbe ?ename  
where { ?wein wo:name ?name .  
        ?wein wo:farbe ?farbe .  
        ?wein wo:erzeuger ?erzeuger .  
        ?erzeuger wo:weingut ?ename . }
```

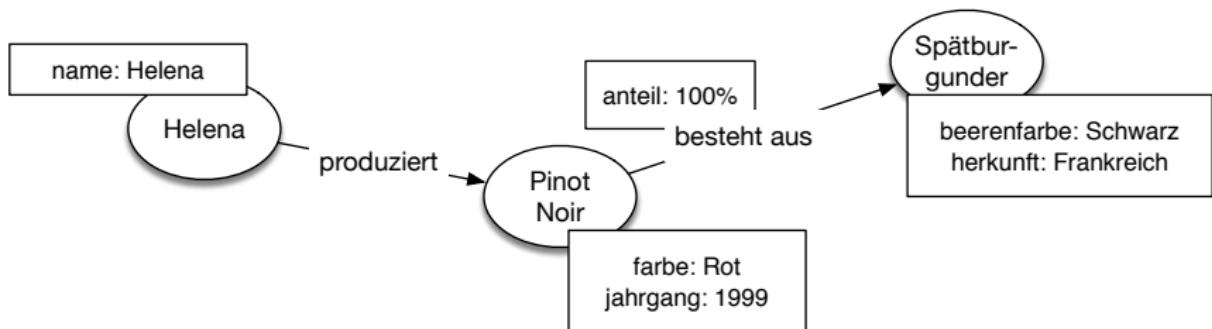
# SPARQL: Weitere Elemente

- filter: Filterbedingungen für Bindungen
- optional: optionale Muster – erfordern nicht zwingend ein Matching

```
prefix wo: <http://weindb.org/ontologie/>
select ?name
where { ?wein wo:name ?name . }
      optional { ?wein wo:jahrgang ?jahrgang } .
      filter ( bound(?jahrgang) && ?jahrgang < 2010 )
```

# Property-Graph-Modell

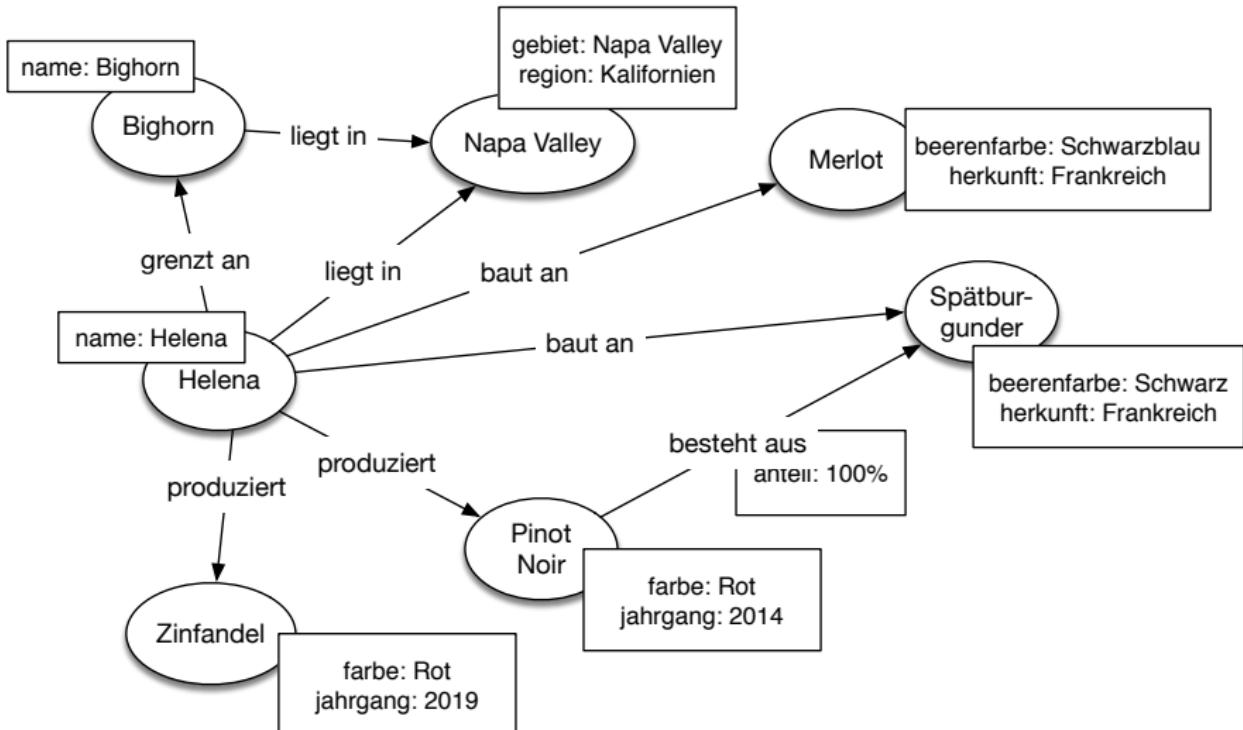
- Knoten und (gerichtete) Kanten mit Eigenschaften (Properties)
- nicht streng typisiert, d.h. Eigenschaften als Name-Wert-Paare
- Unterstützung in diversen Graph-Datenbanksystemen: neo4j, Microsoft Azure Cosmos DB, OrientDB, Amazon Neptune, ...



# Property-Graph-Modell in Neo4j

- Elemente: Nodes, Relationships, Properties, Labels
- Properties = Key-Value-Paare: Key (=String), Value (=Java-Datentypen + Felder)
- Nodes mit Labels ( $\approx$  Klassenname)
- Relationships: sind gerichtet, mit Namen und ggf. Properties

# Property-Graph-Modell: Beispiel



# Anfragen auf Graphen

- keine Standardsprache
- aber wiederkehrende Grundelemente
  - Graph Matching: Knoten, Kanten, Pfade (siehe BGP in SPARQL)
  - Filter für Knoten- und Kanteneigenschaften
  - Konstruktion neuer Graphen
- hier: Cypher (neo4j)

# Anfragen in Cypher

- Basis: Muster der Form „Knoten → Kante → Knoten ...“

```
(von)-[:relationship]->(nach)
```

- Beschränkung über Label und Properties

```
(e:ERZEUGER)-[:LIEGT_IN]->(a:ANBAUGEBIET {  
    gebiet: 'Napa Valley' } )
```

# Cypher: Klauseln

- `match`: Beispilmuster für Matching
- `return`: Festlegung der Rückgabedaten (Projektion)
- `where`: Filterbedingung für „gematchte“ Daten
- `create`: Erzeugen von Knoten oder Beziehungen
- `set`: Ändern von Property-Werten
- ...

# Cypher: Beispiele

- Anlegen von Daten

```
create
    (napavalley:ANBAUGEBIET {
        gebiet: 'Napa Valley', region: 'Kalifornien' }),
    (helena:ERZEUGER { name: 'Helena' }),
    ...
    (helena)-[:LIEGT_IN]->(napavalley),
    ...
```

# Cypher: Beispiele

- Alle Weingüter aus dem Napa Valley

```
match (e:ERZEUGER)-[:LIEGT_IN]->(a:ANBAUGEBIET {  
    gebiet: 'Napa Valley' })  
return e
```

- Alle Regionen, in denen die Merlot-Traube angebaut wird

```
match (r:REBE { name: 'Merlot' })<-[ :BAUT_AN]-(w) \  
      -[:LIEGT_IN]->(g)  
return g
```

# Cypher: Beispiele /2

- Alle Weingüter, die Weine mit einem Spätburgunder-Anteil von mehr als 50% produzieren sowie die Anzahl dieser Weine pro Weingut

```
match (e:ERZEUGER)-[:PRODUZIERT]->(w:WEIN)-  
      [b:BESTEHT_AUS]->(r:REBE { name: 'Spätburgunder' })  
where b.anteil > 50  
return e.name, count(w.name)
```

- Alle Weingüter, direkt an das Weingut Helena grenzen oder an ein Weingut, das direkt an Helena grenzt

```
match (e1:ERZEUGER { name: 'Helena' })-[:GRENZT_AN*..2] \  
      -(e2:ERZEUGER)  
return e2
```

# Cypher: Beispiele /3

- alle Knoten des Typs WEINE

```
match (w)
where w:WEINE
return w
```

- Knotengrade pro Knoten im Graph

```
match (n)-[r]-()
return n, count(r)
```

# Zusammenfassung

- NoSQL als Oberbegriff für diverse Datenbanktechniken
- große Bandbreite: von einfachen KV-Stores bis zu Graphdatenbanken
- höhere Skalierbarkeit / Performance gegenüber SQL-DBMS meist durch Einschränkungen erkauft
  - Abschwächung von ACID-Eigenschaften
  - begrenzte Anfragefunktionalität
  - Nicht-Standard bzw. proprietäre Schnittstellen

- Lena Wiese: Advanced Data Management for SQL, NoSQL, Cloud and Distributed Databases. De Gruyter / Oldenburg, 2015
- Ian Robinson, Jim Webber, Emil Eifrem: Graph Databases. O'Reilly, 2015

# Teil XIII

## Anwendungsprogrammierung

1. Programmiersprachenanbindung

2. JDBC

3. SQLJ

4. LINQ

5. Objekt-relationales Mapping

# Lernziele für heute . . .

- Wissen zu Konzepten und Schnittstellen zum Zugriff auf SQL-Datenbanken aus Programmiersprachen heraus
- Verständnis prozeduraler Schnittstellen am Beispiel von JDBC
- Kenntnisse zu Embedded SQL
- Grundverständnis objektrelationaler Abbildungen

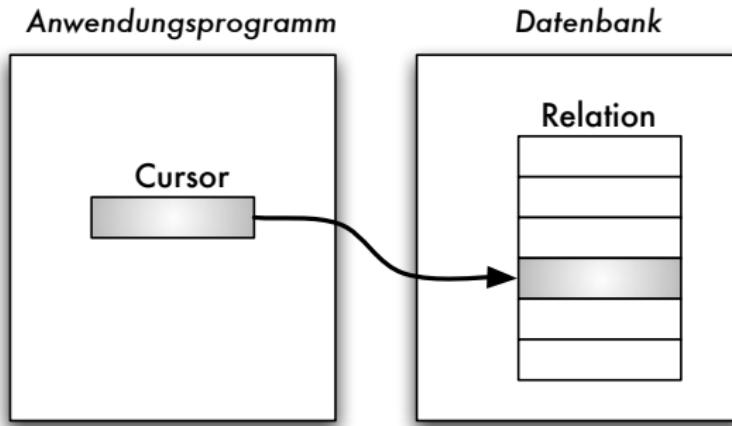


# Programmiersprachenanbindung

- Kopplungsarten:
  - prozedurale oder CALL-Schnittstellen (**call level interface**)
    - Beispiele: SQL/CLI, ODBC, JDBC, ...
  - Einbettung einer DB-Sprache in Programmiersprachen
    - statische Einbettung: **Vorübersetzer-Prinzip**  
~~ SQL-Anweisungen *zur Übersetzungszeit* festgelegt
    - Beispiele: Embedded SQL, SQLJ
    - dynamische Einbettung:  
~~ Konstruktion von SQL-Anweisungen zur Laufzeit
  - **Spracherweiterungen** und neue *Sprachentwicklungen*
    - Beispiele: SQL/PSM, PL/SQL, Transact-SQL, PL/pgSQL

# Cursor-Konzept

- **Cursor:** Iterator über Liste von Tupeln (Anfrageergebnis)

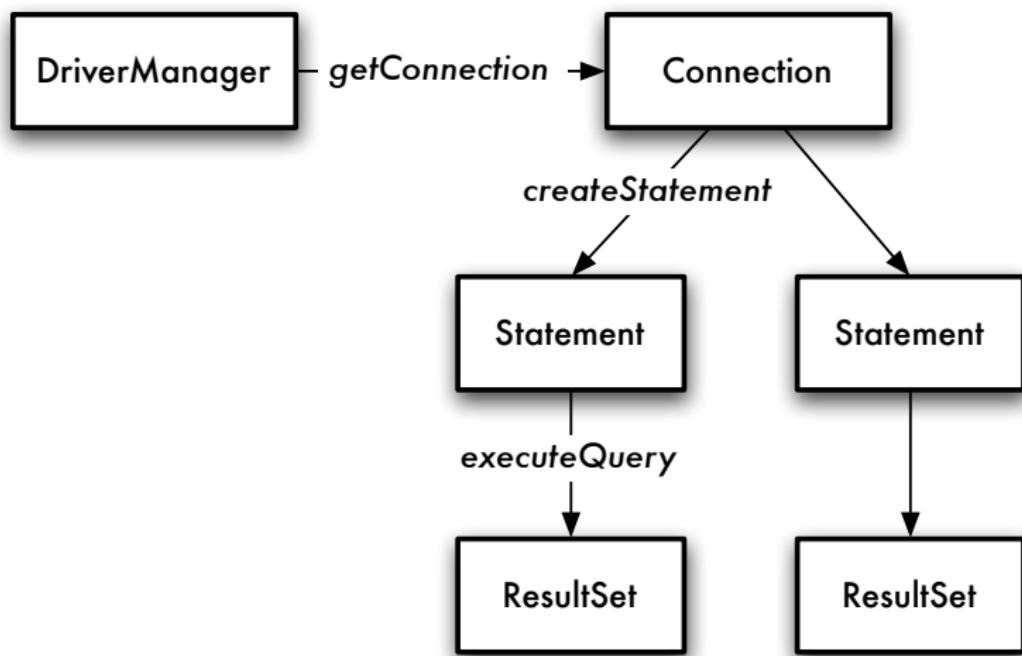


# JDBC

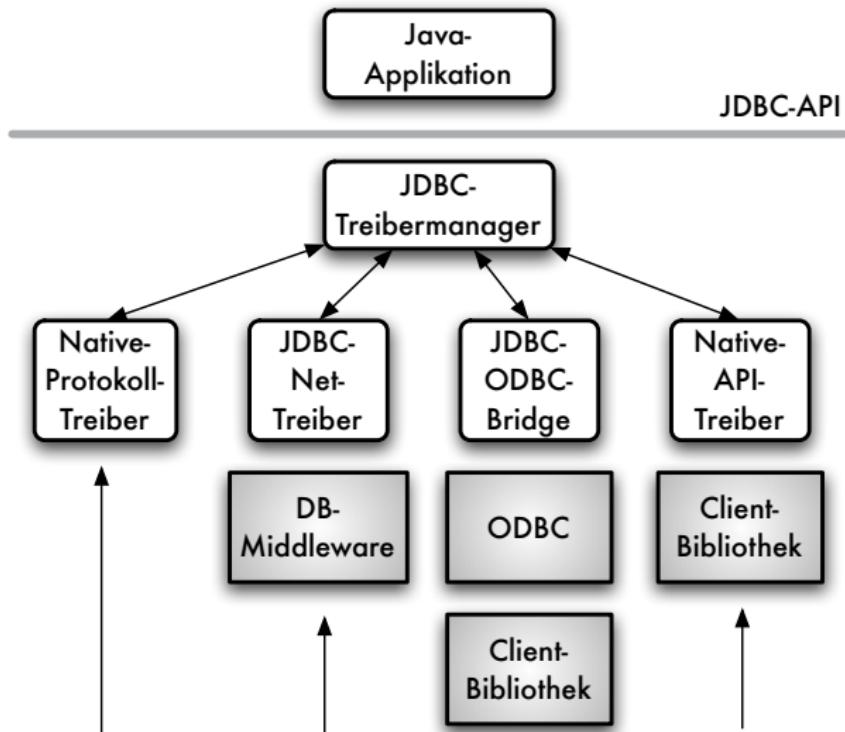
# JDBC: Überblick

- Datenbankzugriffsschnittstelle für Java
- abstrakte, datenbankneutrale Schnittstelle
- vergleichbar mit ODBC
- Low-Level-API: direkte Nutzung von SQL
- Java-Package `java.sql`
  - `DriverManager`: Einstiegspunkt, Laden von Treibern
  - `Connection`: Datenbankverbindung
  - `Statement`: Ausführung von Anweisungen über eine Verbindung
  - `ResultSet`: verwaltet Ergebnisse einer Anfrage, Zugriff auf einzelne Spalten

# JDBC: Struktur



# JDBC: Treiberkonzept



## 1. Aufbau einer Verbindung zur Datenbank

- Angabe der Verbindungsinformationen
- Auswahl und Laden des Treibers

## 2. Senden einer SQL-Anweisung

- Definition der Anweisung
- Belegung von Parametern

## 3. Verarbeiten der Anfrageergebnisse

- Navigation über Ergebnisrelation
- Zugriff auf Spalten

## 1. Treiber laden

```
Class.forName("com.company.DBDriver");
```

## 2. Verbindung herstellen

```
String url = "jdbc:subprotocol:datasource";
Connection con = DriverManager.getConnection
    (url, "scott", "tiger");
```

JDBC-URL spezifiziert

- Datenquelle/Datenbank
- Verbindungsmechanismus (Protokoll, Server und Port)

## 1. Anweisungsobjekt (Statement) erzeugen

```
Statement stmt = con.createStatement();
```

## 2. Anweisung ausführen

```
String query = "select Name, Jahrgang from WEINE";  
ResultSet rSet = stmt.executeQuery(query);
```

### Klasse java.sql.Statement

- Ausführung von Anfragen (SELECT) mit executeQuery
- Ausführung von Änderungsanweisungen (DELETE, INSERT, UPDATE) mit executeUpdate

## 1. Navigation über Ergebnismenge (Cursor-Prinzip)

```
while (rSet.next()) {  
    // Verarbeitung der einzelnen Tupel  
    ...  
}
```

## 2. Zugriff auf Spaltenwerte über getType-Methoden

- über Spaltenindex

```
String wName = rSet.getString(1);
```

- über Spaltenname

```
String wName = rSet.getString("Name");
```

# JDBC: Fehlerbehandlung

- Fehlerbehandlung mittels Exception-Mechanismus
- SQLException für alle SQL- und DBMS-Fehler

```
try {
    // Aufruf von JDBC-Methoden
    ...
} catch (SQLException exc) {
    System.out.println("SQLException: " +
        exc.getMessage());
}
```

# JDBC: Probleme mit Nutzereingaben

```
ResultSet rSet = stmt.executeQuery("select Name, Jahrgang "+  
    "from WEINE WHERE JAHRGANG < "+ userJahr );
```

- userJahr ist Nutzereingabe und erwartet Jahr
- Aber: Nutzer können beliebige Eingaben tätigen
- Strings werden einfach konateniert



~~ SQL-Injection

# JDBC: Präparierte Anfragen

- Anfrage mit Variablen präparieren
- Eingabe kann von potentiellen Schlüsselworten bereinigt und Typen entsprechend konvertiert werden
- Beispiel als präparierte Anfrage:

```
PreparedStatement pStmt =  
    con.prepareStatement("select Name, Jahrgang from WEINE "+  
    "WHERE JAHRGANG < ?");  
pStmt.setInt(1, userJahr); // Parameter Nummerierung startet bei 1  
ResultSet rSet = pStmt.executeQuery();
```

- **Präparierte Anfragen verhindern SQL-Injection**

# JDBC: Änderungsoperationen

- DDL- und DML-Operationen mittels `executeUpdate`
- liefert Anzahl der betroffenen Zeilen (für DML-Operationen)

```
Statement stmt = con.createStatement();
int rows = stmt.executeUpdate(
    "update WEINE set Preis = Preis * 1.1 " +
    "where Jahrgang < 2000");
```

- Methoden von Connection
  - `commit ()`
  - `rollback ()`

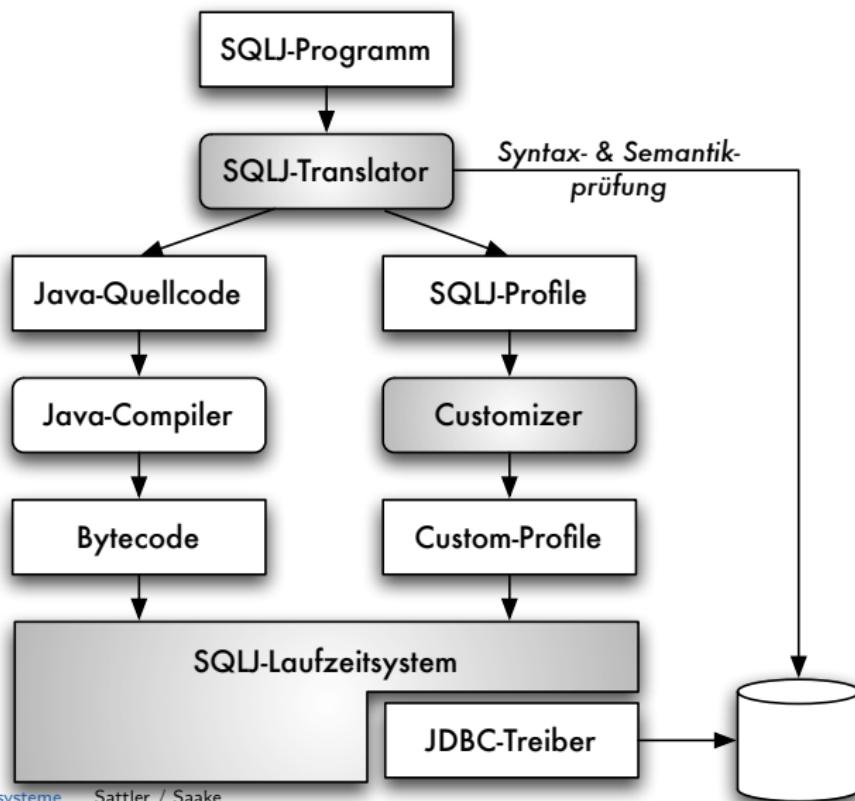
## Auto-Commit-Modus

- implizites Commit nach jeder Anweisung
- Transaktion besteht nur aus einer Anweisung
- Umschalten mittels `setAutoCommit (boolean)`

# SQLJ

- Einbettung von SQL-Anweisungen in Java-Quelltext
- Vorübersetzung des erweiterten Quelltextes in echten Java-Code durch Translator **sqlj**
- Überprüfung der SQL-Anweisungen
  - korrekte Syntax
  - Übereinstimmung der Anweisungen mit DB-Schema
  - Typkompatibilität der für Datenaustausch genutzten Variablen
- Nutzung von JDBC-Treibern

# SQLJ: Prinzip



# SQLJ-Anweisungen

- Kennzeichnung durch #sql Deklarationen
- Klassendefinitionen für Iteratoren
- SQL-Anweisungen: Anfragen, DML- und DDL-Anweisungen

```
#sql { SQL-Operation };
```

- Beispiel:

```
#sql { insert into ERZEUGER (Weingut, Region) values
      ( 'Wairau Hills', 'Marlborough' ) };
```

- Variablen einer Host-Sprache (hier Java), die in SQL-Anweisungen auftreten können
- Verwendung: Austausch von Daten zwischen Host-Sprache und SQL
- Kennzeichnung durch "*:variable*"
- Beispiel:

```
String name;
int weinID = 4711;
#sql { select Name into :name
       from WEINE where WeinID = :weinID };
System.out.println("Wein = " + name);
```

- Nullwerte: Indikatorvariable "*:variable:indvar*"

## 1. Deklaration des Iterators

```
#sql public iterator WeinIter(String Name, String Weingut,  
    int Jahrgang);
```

## 2. Definition des Iteratorobjektes

```
WeinIter iter;
```

## 3. Ausführung der Anweisung

```
#sql iter = { select Name, Weingut, Jahrgang from WEINE };
```

## 4. Navigation

```
while (iter.next()) {  
    System.out.println(iter.Name() + " " iter.Weingut());  
}
```

- SQL-Statements als zur Laufzeit konstruierte Strings

```
exec sql begin declare section;
        AnfrageString char(256) varying;
exec sql end declare section;
exec sql declare AnfrageObjekt statement;
AnfrageString =
        'delete from WEINE where WeinID = 4711';
...
exec sql prepare AnfrageObjekt from :AnfrageString;
exec sql execute AnfrageObjekt;
```

# LINQ

# Language Integrated Query (LINQ)



- Einbettung einer DB-Sprache (SQL) in eine Programmiersprache (C#)
- spezielle Klassenmethoden

```
IEnumerable<string> res = weine
    .Where(w => w.Farbe == "Rot")
    .Select(w => new { w.Name });
```

- eigene Sprachkonstrukte (ab C# 3.0)

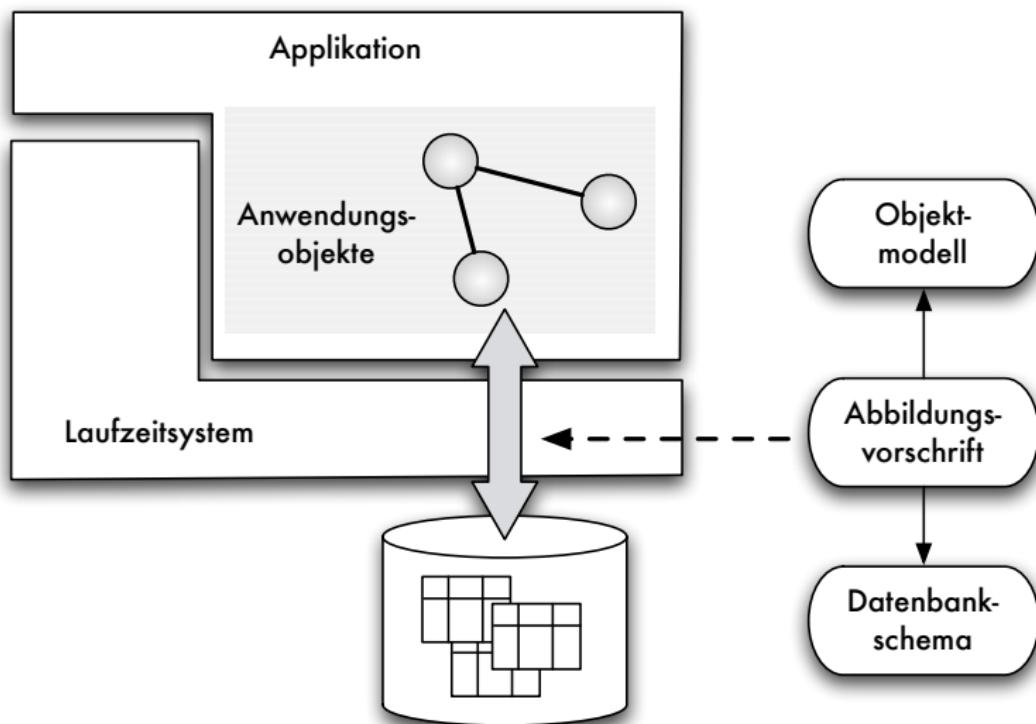
```
IEnumerable<op> res = from w in weine
    where w.Farbe == "Rot"
    select new { w.Name };
```

# Objekt-relationales Mapping

# Objekt-relationales Mapping

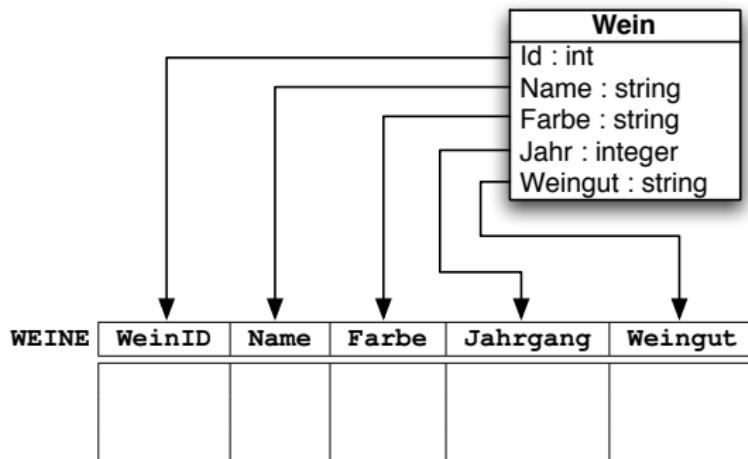
- Einsatz von
  - relationalen Backends (SQL-DBMS)
  - objektrelationalen Anwendungen, Applikationsservern, Middleware,  
...
- Implementierung von „Geschäftslogik“ in Form von Objekten  
(Kunde, Bestellung, Vorgang, ...)
  - z.B. als Java Bean, CORBA-Objekt
- erfordert: Abbildung Klasse ↔ Relation
- Aspekte:
  - konzeptionelle Abbildung
  - Laufzeitunterstützung
- Technologien/Produkte: JDO, Hibernate, ADO.NET Entity Framework...

# Objekt-relationales Mapping: Prinzip



- OO: Klasse definiert Eigenschaften von Objekten (Intension) + umfasst Menge aller Objekte (Extension)
- RM: Relation umfasst alle Tupel, Relationenschema beschreibt Struktur
- naheliegend: Klasse = Tabelle
- aber: Normalisierung zerlegt Relationen!
  - 1 Klasse = 1 Tabelle
  - 1 Klasse =  $n$  Tabellen
  - $n$  Klassen = 1 Tabelle

# Klassen und Tabellen: Beispiel

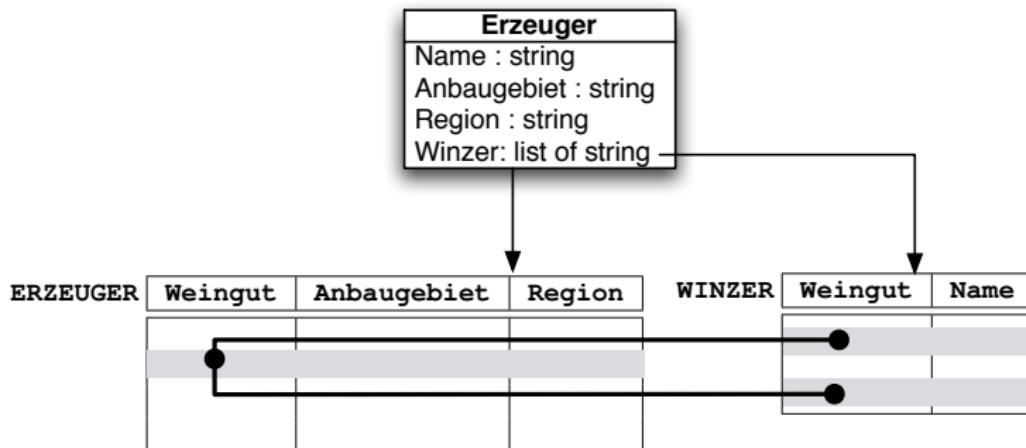


- **eingebetteter Fremdschlüssel** in der Relation der Klasse, d.h. der Identifikator des assoziierten Objektes wird als Fremdschlüssel in zusätzlichen Spalten gespeichert
- **Fremdschlüsseltabellen**: die Beziehungsinstanz wird als Tupel mit den Schlüsseln der beteiligten Objekte repräsentiert
- Abbildung der in Beziehung stehenden Klassen auf **eine einzelne Tabelle**: Verletzung der Normalformen

# Beziehungen: konkrete Abbildung

- 1:1-Beziehungen: eingebettete Fremdschlüssel
- 1:n-Beziehungen: eingebettete Fremdschlüssel oder Fremdschlüsseltabellen
- Beziehungen mit Attributen: Fremdschlüsseltabellen
- m:n-Beziehungen: Fremdschlüsseltabellen
- Drei- und mehrstellige Beziehungen: Fremdschlüsseltabellen

# Beziehungen /2



- Java-Framework für objekt-relationales Mapping
- Idee: Abbildung von Java-Objekten auf Tupel einer relationalen Datenbank
- Prinzip: Java-Klasse + Abbildungsvorschrift  $\rightsquigarrow$  SQL-Tabelle
- keine expliziten SQL-Anweisungen nötig!
- Unterstützung der Navigation über Beziehungen (automatisches Nachladen der referenzierten Objekte)
- Anfragen über eigene Sprache (HQL bzw. QBC/QBE)

# Hibernate: Beispiel

```
public class Wein {  
    private int id;  
    private String name;  
    private String farbe;  
    private int jahr;  
    private String weingut;  
  
    public void setName(String n) { name = n; }  
    public String getName() { return name; }  
    public void setFarbe(String f) { farbe = f; }  
    public String getFarbe() { return farbe; }  
    public void setJahr(int j) { jahr = j; }  
    public int getJahr() { return jahr; }  
    ...  
}
```

# Hibernate: Beispiel /2

- Deklaration der Abbildung in einer XML-Mapping-Datei
- Abbildungsvorschrift wird zur Systemlaufzeit interpretiert

```
<hibernate-mapping>
    <class name="Wein" table="WEINE">
        <id name="id">
            <generator class="native" />
        </id>
        <property name="name" />
        <property name="farbe" />
        <property name="jahr" column="jahrgang"/>
        <property name="weingut" />
    </class>
</hibernate-mapping>
```

# Hibernate: Objekterzeugung

```
Transaction tx = null;

Wein wein = new Wein();
wein.setName("Pinot Noir");
wein.setFarbe("Rot");
wein.setJahr(1999);
wein.setWeingut("Helena");

try {
    tx = session.beginTransaction();
    session.save(wein);
    tx.commit();
} catch (HibernateException exc) {
    if (tx != null) tx.rollback();
}
```

# Hibernate: Anfragen

- Anfragen über Hibernate-eigene Anfragesprache HQL
- Formulierung auf dem *konzeptuellen* Schema (Java-Klassen)
- Select-Klausel nicht benötigt (Ergebnisse sind immer Objekte)
- Beispiel

```
Query query =
    session.createQuery("from Wein where Farbe = 'Rot'");
Iterator iter = query.iterate();
while (iter.hasNext()) {
    Wein wein = (Wein) iter.next();
    ...
}
```

# Zusammenfassung

- Verbindung zwischen SQL und imperativen Sprachen
- Call-Level-Schnittstelle vs. Embedded SQL
- Objektrelationales Mapping

# Kontrollfragen

- Welche Konzepte gibt es, um auf SQL-Datenbanken zuzugreifen?
- Was sind die Vor- und Nachteile von Call-Level-Schnittstellen wie JDBC im Vergleich zur Einbettung von SQL?
- Wie lassen sich Anwendungsobjekte auf SQL-Tabellen abbilden? Welche Aufgaben bestehen dabei?

