1 Was sind Datenbanken?

Sammlungen von Tabellen.
Probleme ohne Datenbanken:

- Verschwendung von Speicherplatz
- „Vergessen“ von Änderungen
- keine zentrale, „genormte“ Datenhaltung
- Datenredundanz

1.1 Regeln von Codd

Integration	einheitliche, nichtredundante Datenverwaltung
Operationen	Speichern, Suchen, Ändern
Katalog	Zugriffe auf Datenbankbeschreibungen im Data Dictionary
Benutzersichten	
Integritätssicherung	Korrektheit des Datenbankinhalts
Datenschutz	Ausschluss unauthorisierter Zugriffe
Transaktionen	mehrere DB-Operationen als Funktionseinheit
Synchronisation	parallele Transaktionen koordinieren
Datensicherung	Wiederherstellung von Daten nach Systemfehlern

1.2 3-Ebenen-Schemaarchitektur

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

Beispiel Indexstruktur:

1.3 Datenunabhängigkeit

- 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

2 Relationen

2.1 Begriffe

Datenbank	Menge von Tabellen
Datenbankschema	Menge von Relationenschemata
DataBaseManagmentSystem	Dinge, um Datenbanken zu benutzen, wie MySQL oder PostgreSQL,... (DBMS)
Relationenschema	Spaltennamen
Relation	Weitere Einträge in der Tabelle
Tupel	Eine Zeile der Tabelle
Attribut	Eine Spaltenüberschrift
Attributwert	Ein Eintrag
Wertebereich	mögliche Werte eines Attributs (auch **Domäne**)
Schlüssel	minimale Menge von Attributen, deren Werte ein Tupel einer Tabelle eindeutig identifizieren
Primärschlüssel	Menge von Attributen identifizieren ein Tupel der Relation eindeutig. (Integritätsbedingung)
Fremdschlüssel	Primärschlüssel einer fremden Tabelle, der als eindeutiger Verweis benutzt wird. (Integritätsbedingung)
Fremdschlüsselbedingung	alle Attributwerte des Fremdschlüssels tauchen in der anderen Relation als Werte des Schlüssels auf
Primattribut	Element eines Schlüssels
zusammengesetzter Schlüssel	Der Schlüssel besteht aus mehr als einem Attribut

2.2 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. **timestamp** für Datums-, Zeit- und kombinierte Datums-Zeit-Angaben
- **Null** repräsentiert die Bedeutung „Wert unbekannt“, nimmt auch keinen der vorigen Wertebereiche an (Vergleich mit Null immer false)

2.3 Relationenalgebra

Basisoperation	Beschreibung
Selektion	Auswahl von Zeilen einer Tabelle anhand eines Selektionsprädikats
Projektion	Auswahl von Spalten durch Angabe einer Attributliste (entfernt doppelte Tupel)
Verbund/Join	verknüpft Tabellen über gleichbenannte Spalten, indem er jeweils zwei Tupel verschmilzt, falls sie dort gleiche Werte aufweisen
Umbenennung	Anpassung von Attributnamen (z.B. bei Join mit gleicher Tabelle)
Vereinigung	listet die Tupelmengen zweier Relationen in einer neuen Relation auf, wobei die Attributmengen identisch sein müssen
Differenz	eliminiert Tupel in der ersten Relation, die auch in der zweiten Relation vorhanden sind
Durchschnitt	listet die Tupel auf, die in beiden Relationen vorkommen
Kreuzprodukt	verknüpft alle Tupel der einen Tabelle mit allen Tupel der anderen Tabelle

Basisoperation	Relationenalgebra	SQL-Befehl
Selektion	$\sigma_{Bedingung}(Tabelle)$	select *
		from Tabelle
		where **Selektionsprädikat**
Projektion	$\pi_{Spalte}(Tabelle)$	select **distinct Spalte**
		from Tabelle
Join	$Tabelle1 \bowtie Tabelle2$	select *
		from Tabelle1 **natural join** Tabelle2
Umbenennung	$\beta_{NeuerName \leftarrow Attributname}(Tabelle)$	select *
		from Tabelle **as** NeuerName
Vereinigung	$Tabelle1 \cup Tabelle2$	select * from Tabelle1
		union
		select * from Tabelle2
Differenz	$Tabelle1 - Tabelle2$	select * from Tabelle1
		except
		select * from Tabelle2
Durchschnitt	$Tabelle1 \cap Tabelle2$	select * from Tabelle1
		intersect
		select * from Tabelle2
Kreuzprodukt	$Tabelle1 \times Tabelle2$	select *
		from Tabelle1**,** Tabelle2

Beispiele:

Minimum: $\pi_a(R) - \beta_{a \leftarrow a_2}(\pi_{a_2}(\sigma_{a_1 < a_2}(\beta_{a_1 \leftarrow a}(R) \times \beta_{a_2 \leftarrow a}(R)))$

Maximum: $\pi_a(R) - \beta_{a \leftarrow a_1}(\pi_{a_1}(\sigma_{a_1 < a_2}(\beta_{a_1 \leftarrow a}(R) \times \beta_{a_2 \leftarrow a}(R)))$

Musiker die in mindestens einer Band spielen: $\pi_{a1.name}(\sigma_{a1.type='p' \wedge a2.type='g' \wedge aa.r\_type='member\ of\ band'}(\beta_{a1}(artist) \times_{a1.id=aa.entity0} \beta_{aa}(artist\_artist)) \times_{a2.id=aa.entity1} \beta_{a2}(artist)$

2.4 Erstellungsoperationen - DataDefinitionLanguage (DDL)

>generiert Strukturen

≫ändert Definition der Daten

create: Die Ablage des Relationenschemas im Data Dictionary, als auch die Vorbereitung einer „leeren Basisrelation“ in der Datenbank	**create table** basisrelationenname (spaltenname1 wertebereich1 [not null], ... spaltennamek wertebereichk [not null]).	**create table** EMP_TEST (empID number ename varchar(100) **not null**, departmentID number, salary number, jobID varchar(3), hiredate date **not null,** comm number, **foreign key** (jobID, comm) **references** JOB_TEST **primary key** (empID, departmentID));
drop · Tabelle löschen **drop table** [table name]	**rename** · Tabelle umbenennen **rename table** [table name] **to** [new table name]	

2.5 Änderungsoperationen - DataManipulationLanguage (DML)

≫operiert auf Strukturen
≫SFW-Block gehört auch zu DML
! Löschoperationen können zur Verletzung von Integritätsbedingungen führen! Beispielsweise Verletzung der Fremdschlüsseleigenschaft in einer anderen Relation.

update: Verändern von Tupeln in einer Relation.	**update** basisrelation **set** attribut1 = ausdruck1, ... attributn = ausdruckn [**where** bedingung]	**update** EMP_TEST **set** ename = 'Arne Anonym' **where** empID = 123
insert: Einfügen von Tupeln in eine Relation.	**insert into** basisrelation [(attribut1, ..., attributn)] **values** (konstante1, ..., konstanten)	**insert into** EMP_TEST (empID, ename, departmentID, salray, jobID, hiredate, comm) **values** (1234, 'Max Mustermann', 12, 150000, 'abc', 21.01.2013, 123456);
delete: Löschen eines Tupels aus einer Relation.	**delete** **from** basisrelation [**where** bedingung]	**delete from** EMP_TEST **where** empID = 123;

3 ER-Modell

3.1 Datenbankmodelle

- System von Konzepten zur Beschreibung von Datenbanken. Es legt Syntax und Semantik von Datenbankbeschreibungen für ein Datenbanksystem fest.
- statische Eigenschaften: Objekte, Beziehungen inklusive Datentypen
- dynamische Eigenschaften: Opertaionen und Beziehungen zwischen Operationen
- Integritätsbedingungen an Objekte und Operationen

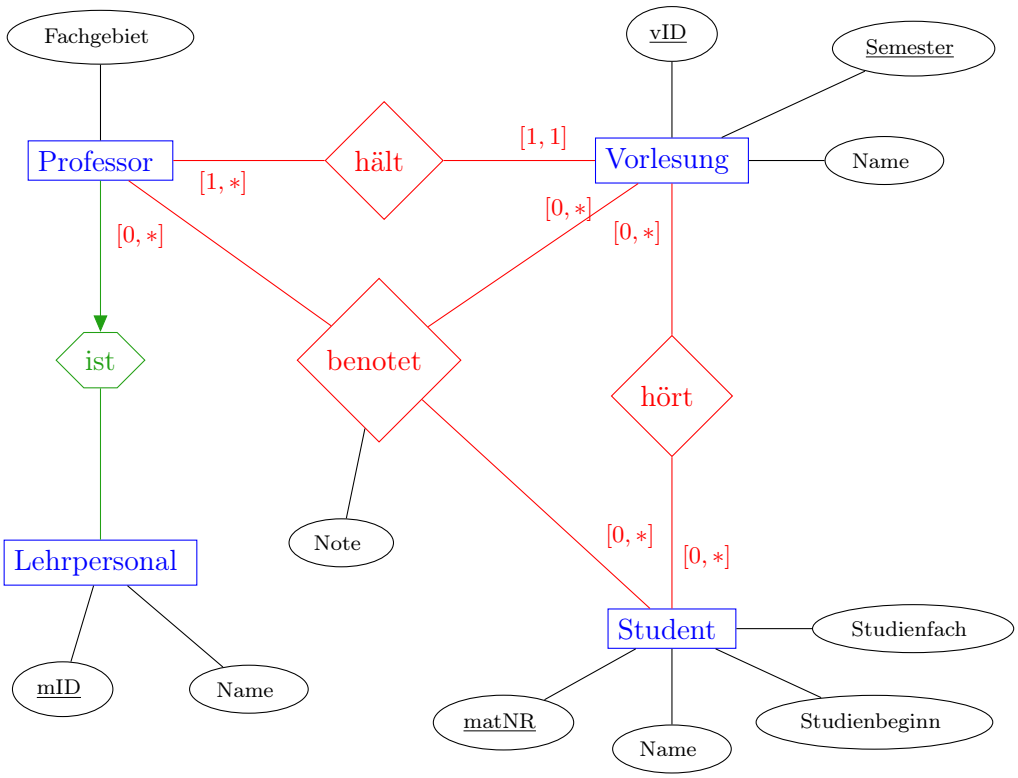
3.2 Bezeichnungen

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 ($[min, max]$-Notation)
Stelligkeit	Anzahl der an einem Beziehungstyp beteiligten EntityTypen
IST-Beziehung	Spezialisierung von Entity-Typen (injektiv, Attributvererbung)
Optionalität	Attribute oder funktionale Beziehungen als partielle Funktionen

3.3 Nice to know

- ≫ für die Art der Beziehung immer die hintere Zahl anschauen
- Wertemengen sind beschrieben durch Datentypen
- vorgegebene Standard-Datentypen (int, string, date)
- Entities sind nicht direkt darstellbar (wie Werte), sondern nur über ihre Eigenschaften beobachtbar
- Attribute werden für Entity-Typen deklariert
- Primärschlüssel markieren durch Unterstreichung (IST-Entities haben keinen eigenen, hat den der Original-Entity)
- Beziehungen können zwischen mindestens 2 Entity-Typen bestehen
- Beziehungen werden auch vererbt
- wenn Entity-Typ mehrfach an einem Beziehungstyp beteiligt: Vergabe von Rollennamen möglich
- Beziehungen können ebenfalls Attribute besitzen
- für Beziehung $Entity_1\ IST\ Entity_2$ gilt immer: $IST(Entity_1[1,1], Entity_2[0,1])$
- ≫ Attribute immer auf die gleiche Seite vom Strich schreiben, bei einer Entity
- ≫ $[min; max]$-Notation begründen, wenn nicht klar, der Default-Wert ist $[0, *]$
- ≫ Schlüssel möglichst klein halten
- ! keine Fremdschlüssel im ER-Modell!

3.4 Beispiel



4 Datenbankentwurf

4.1 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

4.2 Phasenmodell

4.3 Anforderungsanalyse

- Sammlung des Informationsbedarfs in den Fachabteilungen
- informale Beschreibung (Texte, tabellarische Aufstellungen, Formblätter, usw.) des Fachproblems
- Trennen der Information über Daten (Datenanalyse) von den Information über Funktionen (Funktionsanalyse)

4.4 Konzeptioneller Entwurf

- erste formale Beschreibung des Fachproblems
- semantisches Datenmodell (z.B: ER-Modell)
- Modellierung von **Sichten** (virutelle Relation zur vereinfachten Nutzung) z.B. für verschiedene Fachabteilungen
- Analyse der vorliegenden Sichten in Bezug auf Konflikte
- Integration der Sichten in ein Gesamtschema
- Phasen: Sichtenentwurf → Sichtenanalyse → Sichtenintegration

- ER-Modellierung von verschiedenen Sichten auf Gesamtinformation, z.B. für verschiedene Fachabteilungen eines Unternehmens
- Sichtenintegration:
  - Analyse der vorliegenden Sichten in Bezug auf Konflikte
  - Integration der Sichten in ein Gesamtschema

4.5 logischer Entwurf

- Vorgehensweise:
  - Transformation des konzeptionellen Schemas z.B. ER (relationales Modell)
  - Verbesserung des relationalen Schemas anhand von Gütekriterie (Normalisierung)
  - Ziel: Vermeidung von Redundanzen
- Ergebnis: : logisches Schema, z.B. Sammlung von Relationenschemata

4.6 Kapazitätserhaltende Abbildungen

5 Relationale Entwurfstheorie

5.1 Integritätsbedingungen

- Identifizierende Attributmenge $K := B_1, ..., B_k \subseteq R : \forall t_1, t_2 \in r[t_1 \neq t_2 \Rightarrow \exists B \in K : t_1(B) \neq t_2(B)]$
- Schlüssel: ist minimale identifizierende Attributmenge
- Primattribut: Element eines Schlüssels
- Primärschlüssel: ausgezeichneter (ein bestimmter) Schlüssel
- Oberschlüssel oder Superkey: jede Obermenge eines Schlüssels (= identifizierende Attributmenge)
- Fremdschlüssel: $X(R_1) \rightarrow Y(R_2)t(X)|t \in r_1 \subseteq t(Y)|t \in r_2$

6 SQL

6.1 Struktur

>Wo eine Relation steht, kann auch wieder eine Anfrage stehen.

select

- Projektionsliste
- arithmetische Operationen und Aggregatfunktionen

from

- zu verwendende Relationen, evtl. Umbenennungen

where

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

6.2 Verbunde

Natürlicher Verbund in SQL92	**expliziter Verbund: natural join**
• durch Kreuzprodukt **select** * **from** WEINE, ERZEUGER **where** WEINE.Weingut = ERZEUGER.Weingut	• Abkürzung für Anfrage mit Kreuzprodukt **select** * **from** WEINE **natural join** ERZEUGER
Verbunde als explizite Operatoren	**expliziter Verbund: cross 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)	• Kreuzprodukt **select** * **from** WEINE , ERZEUGER • als cross join **select** * **from** WEINE **cross join** ERZEUGER

6.3 Know-How

• Umbenennung von Zwischenrelationen	**select** Ergebnis.Weingut **from** (WEINE natural join ERZEUGER) **as** Ergebnis
• **as** ist optional. Äquivalent:	**select** Ergebnis.Weingut **from** (WEINE natural join ERZEUGER) Ergebnis
• Duplikate werden nur mit **distinct** entfernt	**select distinct** Name **from** WEINE
• **Präfixe** für Eindeutigkeit	**select** Name, **ERZEUGER.**Weingut **from** WEINE natural join ERZEUGER
• Sortierung der Ergebnisrelation aufsteigend: **asc**; absteigend: **desc**	**select** * **from** WEINE **order by** Jahrgang
• Anfrageausdruck, der in der Anfrage mehrfach referenziert werden kann	**with** anfrage-name [(spalten-liste)] **as** (anfrage-ausdruck)

6.4 where

select ... **from** ... **where** bedingung

Form der Bedingung:

- **Vergleiche** Attribut mit Konstante (<,>,=,<> (ungleich),...)
- **Vergleiche** zwei Attribute mit vergleichbaren Wertebereichen
- verwende logische Konnektoren (**or**, **and**, **not**)
- **Verbundbedingung** (s. Verbunde): $relation1.attribut = relation2.attribut$
- **Bereichsselektion**:
  - Notation: *attribut* **between** *konstante*1 **and** *konstante*2
  - als Abkürzung für: $attribut \leq konstante1$ **and** $attribut \geq konstante2$

  - beschränke Attributwerte auf ein abgeschlossenes Intervall
- **Ungewissheitsselektion**:
  - Notation: *attribut* **like** *spezialkonstante*
  - Mustererkennung in Strings
  - %: kein oder beliebig viele Zeichen; _: genau ein Zeichen

6.5 Mengenoperationen

- Mengenoperationen erfordern kompatible Wertebereiche
- Vereinigung, Durchschnitt und Differenz als **union**, **intersect** und **except**
- **corresponding by** gibt die Attributliste an, über der die Mengenoperation ausgeführt werden soll
- Teilmenge: *attribut* **in** $(SFW - Block)$

⩾ siehe Tabelle in 1.3!

⩾ union → Duplikateliminierung

⩾ union **all** → mit Duplikaten

6.6 Skalare Ausdrücke

- Umbenennung von Spalten: **as**
- Aktuelle Länge des Strings: **charlength**
- Suchen einer Teilzeichenkette an bestimmten Positionen des Strings: **substring**
- Aktuelles Datum: **current_date** (+,-,*)
- Aktuelle Zeit: **current_time** (+,-,*)
- Anwendung ist tupelweise

6.7 Quantoren und Mengenvergleiche

⩾in Schachtelung mit in/exists immer **select** * in der Unterabfrage verwenden, ist ja egal, was sie berechnet
⩾exists und in kann man auch durch join ersetzen

Quantoren:

- **all**, **any**, **some**
- Notation: attribut θ { **all** | **any** | **some** } (

  select attribut

  from ...**where** ...)
  - θ vergleicht das Attribut mit den Attributen aus dem SFW-Block
  - **all**: Bedingung wird erfüllt, wenn der Vergleich für alle Tupel aus dem SFW-Block mit attribut true wird
  - **any** bzw. **some**: Bedingung wird erfüllt, wenn der Vergleich mit mindestens einem Tupel des inneren SFW-Blocks true wird
- **in**: Bedingung wird erfüllt, wenn das Attribut einem Tupel der Ergebnisrelation entspricht
- Notation: attribut **in** (SFW-Block)

Beispiel:

Bestimmung des ältesten Weines	alle Weingüter, die Rotweine produzieren	**in**
select *	**select** *	**select** *
from WEINE	**from** ERZEUGER	**from** Weine w1
where Jahrgang <= **all** (	**where** Weingut = **any** (	**where** w1.name **in**(
select Jahrgang **from** WEINE)	**select** Weingut **from** WEINE	**select** w2.name
	where Farbe = ‚Rot'	**from** Weinbestand wb
		where wb.Status = 'verfügbar')

exists:

- einfache Form der Schachtelung
- **where exists** (SFW-Block)
- liefert true, wenn der SFW-Block **nicht** leer ist

Beispiel:

Weingüter, die einen Wein älter als 1990 anbieten	Weingüter aus Bordeaux ohne gespeicherte Weine
select * **from** ERZEUGER e	**select** ***from** ERZEUGER e
where exists (	**where** Region = ‚Bordeaux' **and not exists** (
select * **from** WEINE	**select** * **from** WEINE
where Weingut = e.Weingut **and** Jahrgang < 1990)	**where** Weingut = e.Weingut

6.8 Aggregatfunktionen und Gruppierung

- **Aggregatfunktionen** berechnen neue Werte für eine gesamte Spalte, etwa die Summe oder den Durchschnitt der Werte einer Spalte
  - **count**: berechnet Anzahl der Werte einer Ergebnis-Spalte
  - **sum**: berechnet die Summe der Werte einer Spalte (nur für **numerische** Wertebereiche)
  - **avg**: berechnet den arithmetischen Mittelwert der Werte einer Spalte (nur für **numerische** Wertebereiche)
  - **max** bzw. **min**: berechnen den größten bzw. kleinsten Wert einer Spalte
  - optional auch mit distinct/all außer für count(*)
    - **distinct**: vor Anwendung der Aggregatfunktion werden doppelte Werte aus der Menge von Werten, auf die die Funktion angewendet wird, entfernt
    - **all**: Duplikate gehen mit in die Berechnung ein (Default)
    - Nullwerte werden vor Anwendung eliminiert
  - liefern nur einen Wert, also in **where** -Klausel verwendbar
  - ! Schachtelung von Aggregatfunktionen nicht erlaubt

Beispiel:

Anzahl der verschiedenen Weinregionen	Weine, die älter als der Durchschnitt sind	alle Weingüter, die nur einen Wein liefern
select count (**distinct** Region) **from** ERZEUGER	**select** Name, Jahrgang **from** WEINE **where** Jahrgang < (**select avg**(Jahrgang) **from** WEINE)	**select** * **from** ERZEUGER e **where** 1 = (**select count(*)** **from** WEINE w) **where** w.Weingut = e.Weingut

- **Gruppierung**: Berechnung der Funktionen pro Gruppe, z.B. der Durchschnittspreis pro Warengruppe oder der Gesamtumsatz pro Kunde
- Notation: **select** ... **from** ... [**where** ...] [**group by** attributliste]
- zulässige Attribute hinter select bei Gruppierung
  - Gruppierungsattribute G (Ausgabeattribute müssen in Gruppierung stehen)
  - Aggregationen auf Nicht-Gruppierungsattributen R - G

Beispiel:

Anzahl der Rot- und Weißweine:	Regionen mit mehr als einem Wein:
select Farbe, **count(*)** as Anzahl **from** WEINE **group by** Farbe	**select** Region, **count(*)** as Anzahl **from** ERZEUGER **natural join** WEINE **group by** Region **having count(*)** > 1

7 Algebra

7.1 Begriffe

Anfrage Folge von Operationen, die aus den Basisrelationen eine Ergebnisrelation berechnet

Sicht Folge von Operationen, die unter einem Sichtnamen langfristig abgespeichert wird und unter diesem Namen wieder aufgerufen werden kann, ergibt eine **Sichtrelation**

Kategorien: Relationenalgebra, Kalküle, SQL,...

7.2 Kriterien für Anfragesprachen

Ad-Hoc-Formulierung	Anfrage ohne vollständiges Programm formulieren
Deskriptivität	Benutzer soll formulieren „Was will ich haben?“, nicht „Wie komme ich an das, was ich haben will?“
Mengenorientiertheit	jede Operation soll auf Mengen von Daten gleichzeitig arbeiten, nicht „one-tuple-at-a-time“
Abgeschlossenheit	Ergebnis ist wieder eine Relation
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 (jede Operation hat eine Komplexität $\leq O(n^2)$, n Anzahl der Tupel einer Relation)
Sicherheit	keine Endlosschleife oder unendlich-Ergebnisse bei syntaktisch korrekter Anfrage
Eingeschränktheit	Anfragesprache darf keine komplette Programmiersprache sein
Vollständigkeit	Sprache muss man mindestens die Anfragen einer Standardsprache (Bsp: Relationenalgebra) ausdrücken können

7.3 Verbundvarianten

Gleichverbund (equi-join)	Gleichheitsbedingung über explizit angegebene und evtl. verschiedene Attribute $r(R) \bowtie_{C=D} r(S)$
Theta-Verbund (θ-join)	beliebige Verbundbedingung $r(R) \bowtie_{C>D} r(S)$
Semi-Verbund	nur Attribute eines Operanden erscheinen im Ergebnis $r(L) \times r(R) = \pi_L((r(L) \bowtie r(R))$

8 Kalküle

- eingeführt um zu schauen, ob die Anfrage überhaupt terminiert
- **Kalkül**: eine formale logische Sprache zur Formulierung von Aussagen
- Kalküle zur Formulierung von Datenbank-Anfragen
- **Anfrage** hat die Form $\{f(x)|p(x)\}$
  - x bezeichnet Menge von freien Variablen $x = \{x_1 : D_1, ..., x_n : D_n\}$
  - f bezeichnet Ergebnisfunktion über x
  - p Selektionsprädikat über freien Variablen x

8.1 Tupelkalkül

- Variablen variieren über Tupelwerte (entsprechend den Zeilen einer Relation)
- Beispiel: $\{w|w \in WEINE \wedge w.Farbe = \text{,}Rot\text{‘}\}$

Beispiele:

Verbund
$\{\langle e.Weingut\rangle | e \in ERZEUGER \wedge w \in WEINE \wedge e.Weingut = w.Weingut\}$
Schachtelung
$\{\langle w.Name, w.Weingut\rangle | w \in WEINE \wedge \exists e \in ERZEUGER(w.Weingut = e.Weingut \wedge e.Region = \text{`}Bordeaux\text{`})\}$

9 Physische Datenorganisation

9.1 Hierarchie der Speicher

SCHNELL		OKAY		LANGSAM
Prozessor	Cache-Speicher	Hauptspeicher	Sekundärspeicher	Tertiärspeicher

Für uns sind nur **Hauptspeicher** (RAM, flüchtig, nicht so groß) und **Sekundärspeicher** (z.B. Festplatte, nicht so schnell, groß) wichtig, da man auf dem Prozessor mit Registern oder dem Cache-Speicher des Prozessors nicht manuell arbeitet. Tertiärspeicher sind zum Entfernen gedacht (z.B. USB-Sticks, nicht gut, aber unlimitiert, unterscheide zwischen nearline- und offline(manuell)).

- Eigenschaften der Hierarchie:
- – Ebene x (etwa Ebene 3, der Hauptspeicher) hat wesentlich schnellere Zugriffszeit als Ebene x + 1 (etwa Ebene 4, der Sekundärspeicher)
  - aber gleichzeitig einen weitaus höheren Preis pro Speicherplatz
  - und deshalb eine weitaus geringere Kapazität
  - Lebensdauer der Daten erhöht sich mit der Höhe der Ebenen
- **Zugriffslücke**: Unterschiede zwischen den Zugriffsgeschwindigkeiten auf die Daten

→ um diese zu verringern, verwendet man **Caches**

- **Cache** (Hauptspeicher-Cache) schnellere Halbleiterspeicher-Technologie für die Bereitstellung von Daten an Prozessor
- Plattenspeicher-Cache im Hauptspeicher: **Puffer**

→ funktioniert nicht gut, wenn immer neue Daten benötigt werden

→ deshalb Pufferverwaltung des Datenbanksystems wichtig

9.2 Pufferverwaltung

- **Puffer**: ausgezeichneter Bereich des Hauptspeichers
- in **Pufferrahmen** gegliedert, jeder Pufferrahmen kann eine Seite der Platte aufnehmen
- Aufgaben:
  - muss angeforderte Seiten im Puffer suchen: effiziente Suchverfahren
  - parallele Datenbanktransaktionen: geschickte Speicherzuteilung
  - Puffer gefüllt: adäquate Seitenersetzungsstrategien

Seitenersetzungsstrategien:

- Speichersystem fordert Seite E2 an, die nicht im Puffer vorhanden ist
- Sämtliche Pufferrahmen sind belegt
- vor dem Laden von E2 Pufferrahmen freimachen
- nach den unten beschriebenen Strategien Seite aussuchen

- Ist Seite in der Zwischenzeit im Puffer verändert worden, so wird sie nun auf Platte zurückgeschrieben
- Ist Seite seit Einlagerung in den Puffer nur gelesen worden, so kann sie überschrieben werden (verdrängt)
- Verfahren:
  - Demand-paging-Verfahren: genau eine Seite im Puffer durch angeforderte Seite ersetzen
  - Prepaging-Verfahren: neben der angeforderten Seite auch weitere Seiten in den Puffer einlesen, die eventuell in der Zukunft benötigt werden
  - optimale Strategie: Welche Seite hat maximale Distanz zu ihremnächsten Gebrauch? (nicht realisierbar, zukünftiges Referenzverhalten nicht vorhersehbar)

9.3 Seiten blabla

- **Block**: kleinste adressierbare Einheit auf Externspeicher, Zuordnung zu Seiten im Hauptspeicher
- Aufbau von **Seiten**:
  - Header: Informationen über Vorgänger- und Nachfolger-Seite, eventuell auch Nummer der Seite selbst; Informationen über Typ der Sätze; freier Platz
  - Datensätze
  - unbelegte Bytes
- Organisation der Seiten: doppelt verkettete Liste
- adressierbare Einheiten: Zylinder, Spuren, Sektoren, Blöcke oder Seiten, Datensätze in Blöcken oder Seiten, Datenfelder in Datensätzen
- Maß für die Geschwindigkeit von Datenbankoperationen: Anzahl der Seitenzugriffe auf dem Sekundärspeicher (wegen Zugriffslücke)
- **Sätze** fester Länge: SQL: Datentypen fester und variabler Länge (Verwaltungsblock mit Typ eines Satzes und Löschbit; Freiraum zur Justierung des Offset; Nutzdaten des Datensatzes)

9.4 TID-Konzept

- **Tupel-Identifier** (TID) ist Datensatz-Adresse bestehend ausSeitennummer und Offset
- Offset verweist innerhalb der Seite bei einem Offset-Wert von i auf den i-ten Eintrag in einer Liste von Tupelzeigern (Satzverzeichnis), die am Anfang der Seite stehen
- Jeder Tupel-Zeiger enthält Offsetwert
- Verschiebung auf der Seite: sämtliche Verweise von außen bleiben unverändert
- Verschiebungen auf eine andere Seite: statt altem Datensatz neuer TID-Zeiger
- diese zweistufige Referenz aus Effiziengründen nicht wünschenswert: Reorganisation in regelmäßigen Abständen

9.5 Klassifikation der Speichertechniken

Dateiorganisation:

- Dateiorganisationsform: Form der Speicherung der internen Relation
  - unsortierte Speicherung von internen Tupeln: Heap-Organisation
  - sortierte Speicherung von internen Tupeln: sequenzielle Organisation
  - gestreute Speicherung von internen Tupeln: Hash-Organisation

  - Speicherung von internen Tupeln in mehrdimensionalen Räumen: mehrdimensionale Dateiorganisationsformen
- üblich: Sortierung oder Hashfunktion über Primärschlüssel sortierte Speicherung plus zusätzlicher Primärindex über
- Sortierattributen: **index-sequenzielle** Organisationsform

Zugriffspfade

- **Zugriffspfad**: über grundlegende Dateiorganisationsform hinausgehende Zugriffsstruktur, etwa Indexdatei
  - Einträge $(K, K\uparrow)$: K der Wert eines Primär- oder Sekundärschlüssels, $K\uparrow$ Datensatz oder Verweis auf Datensatz
  - K: Suchschlüssel, genauer: Zugriffsattribute und Zugriffsattributwerte
  - $K\uparrow$: Datensatz selbst: Zugriffspfad wird Dateiorganisationsform; Adresse eines internen Tupels: Primärschlüssel; Liste von Tupeladressen: Sekundärschlüssel; nachteilig ist variable Länge der Indexeinträge

Indexe weiter unten!

9.6 Abbildungen der Datenstrukturen

- Abbildung der konzeptuellen Ebene auf interne Datenstrukturen

Konzeptuelle Ebene		**Interne Ebene**		**Dateisystem/Platte**
Relationen	→	Logische Dateien	→	Physische Dateien
Tupel	→	Datensätze	→	Seiten/Blöcke
Attributwerte	→	Felder	→	Bytes

- Varianten der Abbildung:
  - jede Relation in je einer logischen Datei, diese insgesamt in einer einzigen physischen Datei
  - Cluster-Speicherung, also mehrere Relationen in einer logischen Datei

9.7 Index

Primärindex	Index auf Primärschlüssel
Sekundärindex	Index auf irgendwas anderes
dünnbesetzt	nicht für jeden Zugriffsattributwert K ein Eintrag in Indexdatei bzw. du kommst nicht zu jedem Tupel! → geclustert, sonst kommt man gar nicht hin
indexsequenzielle Datei	sortierte Datei mit dünnbesetztem Index als Primärindex
dichtbesetzter Index	für jeden Datensatz der internen Relation ein Eintrag in Indexdatei bzw. du kommst zu jedem Tupel
geclusterter Index	in der gleichen Form sortiert wie zugehörige interne Relation
nicht-geclusterter Index	Index ist anders organisiert als interne Relation
statische Zugriffsstruktur	optimal nur bei bestimmter (fester) Anzahl von verwaltenden Datensätzen
dynamische Zugriffsstruktur	unabhängig von der Anzahl der Datensätze optimal

- Primärindex kann dünnbesetzt und geclustert sein
- jeder dünnbesetzte Index ist auch ein geclusterter Index, aber nicht umgekehrt
- Sekundärindex kann nur dichtbesetzter, nicht-geclusterter Index sein

9.8 Statische Verfahren

- direkte Organisationsformen: keine Hilfsstruktur, keine Adressberechnung (Heap, sequenziell)
- statische Indexverfahren für Primärindex und Sekundärindex

Heap:

- völlig unsortiert speichern
- physische Reihenfolge der Datensätze ist zeitliche Reihenfolge der Aufnahme von Datensätzen

Sequenzielle Speicherung:

- sortiertes Speichern der Datensätze

Indexsequenzielle Dateiorganisation

- Kombination von sequenzieller Hauptdatei und Indexdatei: indexsequenzielle Dateiorganisationsform
- Indexdatei kann geclusterter, dünnbesetzter Index sein
- mindestens zweistufiger Baum (Blattebene ist Hauptdatei (Datensätze), jede andere Stufe ist Indexdatei)
- Datensätze in Indexdatei: (Primärschlüsselwert, Seitennummer)
- Problem: automatische Anpassung der Stufenanzahl nicht vorgesehen, benötigt unnötig hohen Speicherplatz (unausgeglichen)

9.9 B+-Baum

- Hauptdatei als letzte (Blatt-)Stufe des Baumes integrieren
- in inneren Knoten nur noch Zugriffsattributwert und Zeiger auf nachfolgenden Seite der nächsten Stufe
- jede Blattseite enthält zwischen y und 2y Einträgen
- die Wurzelseite enthält maximal 2x Einträge
- alle anderen enthalten zwischen x und 2x Einträgen
- delete gegenüber B-Baum effizienter
- B+-Baum ist dynamische, mehrstufige, indexsequenziellen Datei
- häufig als Primärindex eingesetzt (Index auf Primärschlüssel)
- ein Tupel hat immer genau einen Tupelidentifier
- Höhe des Baums: $1 + \lceil log_{2x+1}(\frac{n}{2y}) \rceil \leq h \leq 1 + \lfloor log_{x+1}(\frac{n}{y}) \rfloor$ für n Datensätze

10 Transaktionen und so

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.

10.1 ACID-Eigenschaften

Atomicity (**Atomarität**)	Transaktion wird entweder ganz oder gar nicht ausgeführt
Consistency (Konsistenz/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

10.2 Kommandos

commit: die Transaktion soll erfolgreich beendet werden
abort: die Transaktion soll abgebrochen werden
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
rl(x): Lesesperre (engl. read lock bzw. shared lock) auf einem Objekt x
wl(x): Schreibsperre (engl. write lock bzw. exclusive lock) auf einem Objekt x
Entsperren ru(x) und wu(x), oft zusammengefasst u(x) für engl. unlock

10.3 Serialisierbarkeit

Eine verschränkte Ausführung mehrerer Transaktionen heißt **serialisierbar**, wenn ihr Effekt identisch zum Effekt einer (beliebig gewählten) seriellen Ausführung dieser Transaktionen ist.
seriell $\Leftrightarrow$ Konfliktgraph ist **azyklisch** (Kreisfrei)

Voraussetzungen für Konflikt:

- Operationen in verschiedenen Transaktionen
- Operationen auf der gleichen Relation
- mindestens eine Operation ist ein write

T1	T2	T3
r(x)		
	r(y)	
		r(x)
	w(y)	
r(y)	w(x)	
w(y)		
commit	commit	commit

10.4 Redo-Log Buffer

Im wesentlichen ist es einfach nur eine Art auf bestimmte Längen beschränkter (daher Buffer) Redo-Log, in dem du all deine Operationen, die du ausführst, speicherst.
In dem Moment, wo eine Operation abbricht (Szenario 1), oder externe Schäden/Unterbrechungen zum Absturz des Systems führen (Szenario 2), kann es sein, dass man den ursprünglichen Zustand wiederherstellen muss.
Das passiert dann mittels des Redo-Log-Buffers, weil du da praktisch 'rückwärts' alle Operationen aufheben kannst, also die jeweiligen inversen Transaktionen etc. ausführst.
IdR ist es als sog. zirkularer Buffer abgespeichert, d.h. eine cycled linked list, in der der letzte Eintrag wieder auf den ersten zeigt.
In dem Moment, wo du die Maximallänge vollgeschrieben hast, fängst du einfach wieder von vorne an,

und überschreibst den 'älteste' Eintrag, usw.
!Wenn die Transaktion vor einem commit abbricht, sind die Aktionen noch nicht ausgeführt worden!

10.5 Check-Klausel

check: Festlegung weitere lokale Integritätsbedingungen innerhalb der zu definierenden Wertebereiche, Attribute und Relationenschemata

```
create table WEINE (
WeinID int primary key,
Name varchar(20) not null,
Jahr int check(Jahr between 1980 and 2010),
...
)

create domain WeinFarbe varchar(4)
default 'Rot'
check (value in ('Rot', 'Weiss', 'Rose'))
```