Datenbanken Zusammenfassung

Peter Minor
Sommersemester 2025

23. Juli 2025

Inhaltsverzeichnis

1 Kapitel 1: Einführung

Kapitel 1, 8 und 11 sind nicht Klausurrelevant.

2 Kapitel 2: Datenbank-Modellierung

2.1 Modell

Ein Modell ist ein abstrahiertes Abbild der Realität. Es hilft beim Verständnis, bei der Kommunikation und Simulation komplexer Sachverhalte. In der Datenbankmodellierung wird zwischen konzeptuellen, logischen und physischen Modellen unterschieden.

Entity-Relationship-Modell (ER)

2.2 Entitätstyp

Ein Entitätstyp (auch Objekttyp) ist eine Klasse gleichartiger Objekte. Darstellung im ER-Diagramm: Rechteck.

2.3 Attribut

Ein Attribut beschreibt eine Eigenschaft eines Entitätstyps. Darstellung: Ellipse. Attribute können einfach, zusammengesetzt, mehrwertig oder berechnet sein.

2.4 Beziehungstyp (Relationship)

Ein Beziehungstyp stellt eine Relation zwischen Entitäten dar. Darstellung: Raute. Die Kardinalität (1:1, 1:n, m:n) beschreibt die Anzahl möglicher Zuordnungen.

2.5 Schlüssel

Ein Schlüssel ist ein Attribut (oder eine Attributkombination), das jede Entität eindeutig identifiziert. Starke Entitäten haben eigene Schlüssel; schwache Entitäten benötigen eine identifizierende Beziehung zu einer starken Entität.

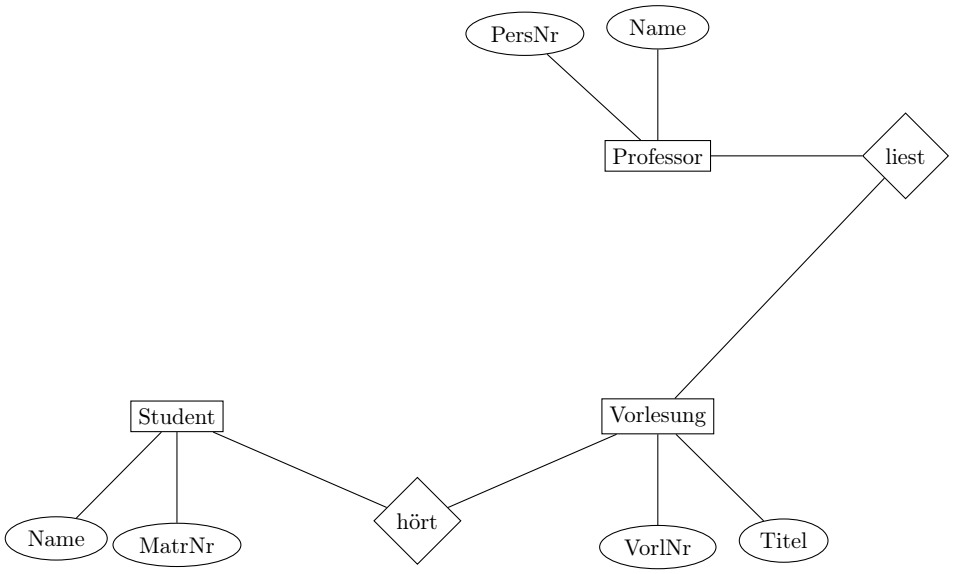
2.6 Partizipation

Beschreibt, ob eine Entität zwingend an einer Beziehung teilnehmen muss:

- **totale Partizipation**: jede Entität muss beteiligt sein
- **partielle Partizipation**: Beteiligung ist optional

2.7 Spezialisierung & Generalisierung (EER)

- **Spezialisierung**: Zerlegung eines Supertyps in Subtypen
- **Generalisierung**: Vereinigung ähnlicher Entitätstypen zu einem Supertyp



3 Kapitel 3: Das relationale Datenmodell

3.1 Relation

Eine Relation ist eine Tabelle mit Attributen (Spalten) und Tupeln (Zeilen). Sie basiert auf dem mathematischen Konzept einer Menge von Tupeln.

3.2 Primärschlüssel

Ein Attribut oder Attributkombination, die ein Tupel eindeutig identifiziert.

3.3 Fremdschlüssel

Ein Attribut, das auf den Primärschlüssel einer anderen Relation verweist und referentielle Integrität sicherstellt.

Relationale Algebra

3.4 Selektion (σ)

Filtert Tupel, die eine bestimmte Bedingung erfüllen. Beispiel:

$$\sigma_{Note \geqslant 4}(\text{Pruefungen})$$

3.5 Projektion (π)

Reduziert die Anzahl der Attribute. Beispiel:

$$\pi_{Name, MatrNr}(\text{Studierende})$$

3.6 Umbenennung (ρ)

Benennung einer Relation oder ihrer Attribute neu, z.B. zur besseren Lesbarkeit von Ausdrücken.

3.7 Vereinigung $\cup$ Schnitt $\cap$ Differenz $-$

- Klassische Mengenoperationen für Relationen mit gleichem Schema.
- Müssen Unionisable sein, d.h. die gleiche Anzahl an Attributen und kompatible Datentypen haben.
  - Die 'outer Union', die die gemeinsamen Attribute wie Union behandelt, die 'übrigen' Attribute auf NULL oder default setzt
  - Dann brauchen die Relationen nicht mehr Unionisable zu sein.

3.8 Division

- R $\div$ S geht nur, wenn R alle Attribute von S enthält(und mehr).
- Dann werden die Attributkombinationen von R ausgewählt, die alle in S vorkommenden Attributkombinationen enthalten

3.9 Kartesisches Produkt ($\times$)

Kombiniert zwei Relationen durch paarweise Tupelkombination. Alle Tupel der ersten Tabelle werden mit der zweiten verbunden

3.10 Join ($\bowtie$)

- Kombiniert Tupel, die die Join.Bedingung erfüllen.
- Entspricht Kartesischem Produkt, gefolgt von Selektion.
- Spezialfälle:
  - Theta-Join: Bedingungen beschränken sich auf $\{=, <, \leqslant, \geqslant, >,$

$\neq\}$.

- Equi-Join: Nur eine Bedingung, die auf Gleichheit prüft.
- Natural Join: Automatischer Join über alle gleichnamigen Attribute die gleichen Attribute werden nur einmal übernommen.
- Outer Join: Beinhaltet auch Tupel, die keine Entsprechung in der anderen Relation haben.
  - Left: Beinhaltet alle Tupel der linken Relation.
  - Right: Beinhaltet alle Tupel der rechten Relation.
  - Full: Beinhaltet alle Tupel beider Relationen.
- Die klassische $\bowtie$ vom Join bekommt hier kleine ärmchen in die jeweilige Richtung.

Beispielhafte Relationen:

- **Student**(MatrNr, Name)
- **Professor**(PersNr, Name)
- **Vorlesung**(VorlNr, Titel)
- **hört**(MatrNr, VorlNr)
  Fremdschlüssel: MatrNr → Student, VorlNr → Vorlesung
- **liest**(PersNr, VorlNr)
  Fremdschlüssel: PersNr → Professor, VorlNr → Vorlesung

4 Kapitel 4: Relationale Entwurfstheorie

4.1 Funktionale Abhängigkeit

Eine Attributmenge α bestimmt eine andere Attributmenge β, geschrieben als:

$$\alpha \rightarrow \beta$$

gilt genau dann, wenn für alle Tupel t_1, t_2 gilt: $t_1[\alpha] = t_2[\alpha] \Rightarrow t_1[\beta] = t_2[\beta]$

4.2 Schlüssel und Superschlüssel

- **Superschlüssel:** α ist Superschlüssel, wenn $\alpha \rightarrow R$
- **Kandidatenschlüssel:** Minimaler Superschlüssel

4.3 Ziel der Normalisierung

Die Normalisierung dient dazu, Redundanzen zu vermeiden und Anomalien (Einfüge-, Update-, Löschanomalien) zu verhindern. Dazu wird ein Relatio-

nenschema anhand funktionaler Abhängigkeiten in wohldefinierte Formen überführt.

4.4 Überblick über die Normalformen

- **1NF (erste Normalform):** Alle Attributwerte sind atomar (nicht weiter teilbar).
- **2NF (zweite Normalform):** 1NF erfüllt + jedes Nicht-Schlüsselattribut ist voll funktional abhängig vom gesamten Primärschlüssel.
- **3NF (dritte Normalform):** 2NF erfüllt + keine transitiven Abhängigkeiten von Nicht-Schlüsselattributen.
- **BCNF (Boyce-Codd Normalform):** Für jede nicht-triviale funktionale Abhängigkeit $\alpha \rightarrow \beta$ gilt: α ist ein Superschlüssel.

4.5 Vorgehen zur Normalisierung

1. Ermittele alle funktionalen Abhängigkeiten (FDs).
2. Bestimme alle Schlüsselkandidaten.
3. Prüfe die aktuelle Normalform.
4. Zerlege die Relation bei Verstoß in mehrere Relationen:
   - Zerlege so, dass jede FD in einer Relation vollständig erfüllt wird.
   - Erhalte dabei die Verlustfreiheit und Abhängigkeitserhaltung.

4.6 Zerlegungsalgorithmus(zu 3NF)

1. Kanonische Überdeckung F_c von F berechnen
2. Relation für jede FD in F_c:
   - X $\rightarrow$ Y wird zu: R_1(X,Y).
   - X $\rightarrow$ Y,Z wird zu: R_2(X,Y,Z).
   - X,Y $\rightarrow$ Z wird zu: R_3(X,Y,Z).
3. Sicherstellen, dass es eine Relation mit einem Schlüssel für R gibt(Ansonsten Schlüssel als Relation R_{n+1} hinzufügen)
4. Doppelte Schemata entfernen

4.7 Zerlegungsalgorithmus(zu BCNF)

1. L_1 bestimmen: Alle Linken und Rechten Seiten der funktionalen Abhängigkeiten
2. Tabelle malen:

α	α^+	R_i	β	Zerlegung

3. Durch FDs iterieren:
   (a) Linke Seite der FD ist α.
   (b) α^+ sind alle von A erreichbaren Attribute
   (c) R_i sind (alle) Relationen in der Zerlegung, die α enthalten
   (d) β wird berechnet mit $\beta = (\alpha^+/\alpha) \cap R_i$
   (e) Falls β leer ist, zum nächsten FD gehen
   (f) Die neue Zerlegung ist die alte Zerlegung, β wird aus R_i entfernt, $\alpha \cup \beta$ wird als neue Relation hinzugefügt.
4. Alle Relationen in der Zerlegung prüfen
   (a) L: Alle Linken Seiten der funktionalen Abhängigkeiten
   (b) Alle R_i mit $|R_i| \leqslant 2$ sind in BCNF
   (c) Iteriere durch die restlichen R_i der Zerlegung:
      i. Überprüfe alle Teilmengen $\alpha \subseteq L \cap R_i$ mit $|\alpha| < |R_i|$-1
      ii. Bilde $\gamma = \alpha^+ \cap R_i$.
      iii. 3 Fälle:
         A. $\gamma = \alpha$ ✓
         B. $\gamma = R_i$ ✓
         C. Sonst: α als FD bei Schritt 3 einsetzen und weiter iterieren. Nur die veränderten Relationen müssen erneut verifiziert werden.

4.8 Anomalien

Anomalien treten auf, wenn Relationen schlecht strukturiert sind – meist durch Redundanz und fehlende Trennung von unabhängigen Daten. Es gibt drei Hauptarten:

- **Einfügeanomalie:** Daten können nicht eingefügt werden, ohne andere zu erzeugen
- **Updateanomalie:** Inkonsistenz bei mehrfacher Speicherung derselben Information
- **Löschanomalie:** Verlust nützlicher Informationen durch Löschung eines Tupels

5 Kapitel 5: SQL - Structured Query Language

Wir befinden uns in der Datenbank-Installation, also im Physischen Schemaentwurf.

5.1 Historie

- 1974: SEQUEL von IBM, Implementierung für System R
- 1983: SQL ist der Standard geworden
- 1986: SQL-86, bzw. SQL 1 ⇒ erster ANSI und ISO-Standard
- 1992: SQL 2, deutliche Erweiterungen im Standard
- Weitere Revisionen:2000(SQL 3), 2003, 2006, 2008, 2011, 2016, 2023

SQL dient als verschiedene Sprachen:

- VDL, DDL, SDL zur Definition von Datenbanken
- DML(Datenmanipulationssprache), DCL(Datenkontrollsprache) zum Zugriff auf Datenbanken

SQL-Befehle:

Befehl	**Beschreibung**
SQL als DDL(Datendefinition)	
`CREATE SCHEMA`	Erstellt ein neues Schema in der Datenbank.
Beispiel	`create schema Unternehmen authorization JSmith create table Projekt;`
Einfacher:	`PID int not null primary key,` geht aber leider nicht mit zusammengesetzten Schlüsseln.
`CREATE Table`	Erstellt eine neue Tabelle im Schema.
Beispiel	`create table Projekt ( PID int not null, Name varchar(50) not null, primary key(PID));`
`ALTER Table`	ändert die angegebene Tabelle.
Es gibt noch andere Verwendungen für `alter:`	
`alter database`	ändert Eigenschaften der Datenbank.
`alter view`	ändert die Definition einer Sicht
`alter index`	modifiziert einen Index
`alter user/role`	ändert die Rollen eines Benutzers
`Add`	Fügt eine Spalte zu einer Tabelle hinzu

Beispiel	`alter table Angestellte add foreign key (Abt) references Abteilung(Nummer);`
`drop`	Löscht das angegebene Objekt. Kann auf Schemen, Tabellen, Sichten, Constraints und Spalten angewendet werden.
Beispiel	`drop table Arbeitszeiten;`
`rename`	Ändert den Namen einer Tabelle
SQL als DML(Datenmanipulation und -abfrage)	
`select [..] from`	Wählt die gegebenen Spalten aus der Tabelle aus und gibt sie zurück
Durch z.B. `select 1.1*Gehalt` kann man Spaltenwerte in der Ausgabe anpassen. Gleiches funktioniert mit +,- und / auf Zahlen. Für Konkatinieren von Zeichenketten verwendet man `||`.	
`insert into`	fügt ein neues Tupel in eine Tabelle ein überprüft automatisch die Vorgaben der Datenbank und weist ggf. zurück
Beispiel	`insert into Student (MNr, VName, NName, Fach) values (123456, 'Max', 'Mustermann', 'Informatik');` Alle nicht angegebenen Infos werden zu NULL bzw. default. Bei SERIAL wird automatisch eingefügt.
`delete from [..]`	Löscht Tupel aus der angegebenen Tabelle. Where bestimmt, was gelöscht werden soll. überprüft automatisch die Vorgaben der Datenbank und weist ggf. zurück
`update [..] set [..]`	setzt bei den Tupeln der Tabelle Attributwerte. kann mit where spezifiziert werden.
`merge into[..] using [..]`	Fügt zwei Tabellen zusammen, die gleiche Attribute erwarten. Durch `when matched` bzw. `when not matched` kann das Verhalten beim mergen bestimmt werden.
Beispiel	`merge into AllStudent c using Student a on AllStudent.MNr = Student.MNr when matched then update set c.VName = a.VName, c.NName = a.NName... when not matched then insert values (a.MNr, a.VName, a.NName, a.Fach);`
`create index [..] on [..]`	Erstellt einen Index auf die angegebene Tabelle.

	Primärschlüssel erhalten automatisch einen Index
`begin_transaction`	Startet eine Transaktion.
`end_transaction`	Beendet eine Transaktion logisch
`commit`	Fügt die Ergebnisse einer Transaktion in den Datenbestand ein.
`abort`	Abbruch einer Transaktion. Änderungen werden verworfen.
Innerhalb einer Transaktion:	
`read_item(X)`	Liest den Wert eines Datenobjekts X.
`write_item(X)`	Schreibt den Wert einer Programmvariablen X in das entsprechende Datenobjekt.
SQL als VCL(Sichtendefinition)	
`create view [..] as select [..]`	Erstellt eine Sicht, die aus der Select-Abfrage resultiert.
SQL als DCL(Rechteverwaltung)	
`grant [..] on [..] to`	Gibt das spezifitierte Recht an der spezifizierten Tabelle an die spezifizierten Nutzer.
`revoke [..] on [..] from`	Entzieht das spezifitierte Recht an der spezifizierten Tabelle von den spezifizierten Nutzern.

SQL-Keywords:

Keyword	**Beschreibung**
SQL als DDL(Datendefinition)	
`not null`	Attribut darf nicht leer sein.
`primary key`	Attribut ist Primärschlüssel der Tabelle.
`unique`	Attributwerte müssen eindeutig sein.
`check`	Ermöglicht komplexere Einschränkungen

Beispiel	`create domain Matrikelnummertyp as numeric(6,0) not null check(value >= 100000)`
	`create table Student (MNr Matrikelnummertyp, VName varchar(50) not null)`
	erstellt einen 'Matrikelnummertyp', sie ist eine 6-Stellige Zahl.

`cascade`	Wenn das Tupel, aus dem die Referenz stammt gelöscht oder geändert wird, werden auch alle Tupel, die darauf referenzieren geupdated/gelöscht.
`set null`	Setzt die Referenz auf null

`set default`	Setzt die Referenz auf den Default-Wert
`No Action/Restrict`	Verhindert das löschen/bearbeiten des Referenzierten Attributs.
`foreign key`	Attribut verweist auf Primärschlüssel einer anderen Tabelle.
`references`	Definiert die referenzierte Tabelle und Spalte für den Fremdschlüssel.
Beispiel:	`foreign key (PID) references Projekt(PID)`
`to_number` oder `to_char`	Konvertiert Datentypen, z.B. von String zu Zahl oder umgekehrt.
Date, Time, Datetimeoffset, interval, year, day, second?	
`where`	filtert nach Bedingungen
Beispiel	`select * from Klausur where Note <= 4;`
`having`	filtert nach Bedingungen, nur auf Gruppen. Tritt nur zusammen mit `Group by` auf
Beispiel	`select * from Projekt, ArbeitetAn where Nummer = projNr group by Nummer, Name having count(*) > 2`
`and`	Verknüpft Bedingungen, alle müssen erfüllt sein
`or`	Verknüpft Bedingungen, mindestens eine muss erfüllt sein
`in`	Überprüft, ob ein Wert in einer Liste von Werten enthalten ist
Beispiel	`select * from Student where Durchschnittsnote in (0.7, 1.0, 1.7, 2.0);`
`order by`	Sortiert die Ergebnisse nach den angegebenen Spalten
`Asc` bzw. `desc`	Sortiert aufsteigend bzw. absteigend, Asc ist der Standardwert
Beispiel	`select * from Klausur order by Note desc;`
`group by`	Gruppiert die Ergebnisse nach den angegebenen Spalten
Beispiel	`select * from Belegung group by KursID;`
`distinct`	Entfernt doppelte Einträge aus dem Ergebnis Aber ist teuer und braucht man nicht unbedingt.

Beispiel	`select distinct Alter from Student;`
`as` Beispiel	Benennt die Spalte um `select Name as StudentName from Student;` Auf Aliasse der äußeren Anfrage kann man innen zugreifen, anders herum aber nicht.
`count`	Zählt die Anzahl der Tupel
`sum`	Summe der Werte der Tupelattribute
`min`	kleinstes Tupelattribut
`max`	größtes Tupelattribut
`avg`	durschschnittlicher Wert der Tupelattribute
Beispiel	`select max(Gehalt) from Angestellte;`
In Kombination mit `group by` werden die Operationen `count, sum, min, max` und `avg` jeweils auf die einzelnen Gruppen angewendet.	
`like`	Vergleicht Zeichenketten

Beispiel	`select * from Student where Name like 'T _ _';`
	sucht alle Studierenden raus, die einen Namen mit drei Buchstaben haben, der mit T anfängt
	`select * from Student where Name like 'T%';`
	sucht alle Studierenden raus, die einen Namen haben, der mit T anfängt

`between`	Überprüft, ob ein Wert in einem Intervall liegt
`exists`	Überprüft, ob das Ergebnis einer Unterabfrage nicht leer ist
`not`	Negiert eine Bedingung
`unique`	überprüft, ob eine Multimenge Duplikate enthält
`is null` bzw. `is not null`	Überprüft, ob ein Attributwert NULL ist. = NULL ist nicht möglich!

5.2 SQL als DDL

- Schema, Tabellen, Datentypen, Constraints definieren
- Strukturelle Änderungen mittels `drop`, `alter`
- SCHEMA:
  - Namensraum in DB
  - Hat eindeutigen Namen
  - Hat Autorisierungsbezeichner

- Beschreibt jedes im Schema enthaltene Objekt
  - Relationen
  - Wertebereiche
  - Restriktionen
  - Sichten
  - Zugriffsrechte

- `information_schema` enthält Metadaten über die Datenbank

5.3 Übergang von relationelem Schema zu SQL Schema

- Name der Relation wird zum Tabellennamen
- Attribute werten untereinander geschrieben(Datentypen angeben)
- Bei einem Schlüssel `primary key` hinterschreiben
- Bei zusammengesetzten Schlüsseln `primary key (A, B)` angeben
- Für IDs ist `serial` als Datentyp sinnvoll
- Fremdschlüssel werden mit `foreign key` gekennzeichnet

Beispiel:

- **Student**(Matrikelnummer, Name, Studiengang)
- **Kurs**(KursID, Titel, Dozent)
- **Belegung**(Matrikelnummer, KursID, Note)

Wird folgendes SQL-Schema:

```
-- Tabelle: Student
CREATE TABLE Student (
    Matrikelnummer INT PRIMARY KEY,
    Name VARCHAR(100),
    Studiengang VARCHAR(100)
);

-- Tabelle: Kurs
CREATE TABLE Kurs (
    KursID SERIAL PRIMARY KEY,
    Titel VARCHAR(100),
    Dozent VARCHAR(100)
);

-- Tabelle: Belegung
CREATE TABLE Belegung (
```

```
    Matrikelnummer INT,
    KursID INT,
    Note DECIMAL(3,1),
    PRIMARY KEY (Matrikelnummer, KursID),
    FOREIGN KEY (Matrikelnummer) REFERENCES Student(Matrikelnummer),
    FOREIGN KEY (KursID) REFERENCES Kurs(KursID)
);
```

5.4 SQL als DML

- Daten manipulieren und abfragen
- Es können Duplikate auftreten, falls nicht gewünscht `distinct` nutzen
- Es wird zuerst Join dann Gruppierung und dann Aggregation durchgeführt
- Abfragen können auch Unterabfragen enthalten, also verschachtelt sein.

5.5 Umsetzung der Operationen der relationalen Algebra in SQL

Operation	**SQL-Äquivalent**
Kartesisches Produkt	`select * from A, B;`
Join	`select * from A inner join b on <Bedingung>;`
Natürlicher Join	`select * from A natural join B;`
Outer Join	`select * from A left outer join B on <Bedingung>;` man kann auch `right` oder `full` nutzen.
Join mit sich selber mit Alias	`select * from Angestellte A, Angestellte B where A.ID = B.Vorgesetzte;`
Hinweis	wenn zweimal ein gleichnamiges Attribut existiert, kann man mit z.B. A.ID und B.ID darauf zugreifen Auf Aliasse der äußeren Anfrage kann man innen zugreifen, anders herum aber nicht.
Vereinigung	`select * from A union select * from B;`
Schnitt	`select * from A intersect select * from B;`
Differenz	`select * from A minus select * from B;`
Bei Vereinigung, Schnitt und Differenz werden Duplikate entfernt	

5.6 SQL als VDL(Verwaltung der Sichten)

Eine Sicht ist eine virtuelle Tabelle, die aus einer Abfrage resultiert.

- Können, müssen aber nicht in der Datenbank gespeichert werden
- Werden immer aktuell gehalten
- Können wie Tabellen abgefragt werden
- Manipulation oft nicht möglich(non-updatable views)

5.7 SQL als DCL(Verwaltung der Zugriffsrechte)

- `grant` und `revoke` für Rechteverwaltung
- Rechte können auf Objekte wie Tabellen, Sichten, Prozeduren angewendet werden
- Rechte: SELECT, INSERT, UPDATE, DELETE, EXECUTE
- Beispiel: `grant select on Tabelle to Benutzer;`

5.8 Datentypen in SQL

Datentyp	Beschreibung
`Integer/int, smallint`	Ganze Zahlen, smallint kleinere Zahlen (⇒ kleinerer Speicherbedarf)
`Float, Real, Double precision`	Gleitkommazahlen, Approximativ. Double precision für mehr Genauigkeit
`Decimal(i, j), Numeric(i, j)`	Feste Dezimalzahlen, i: Stellen insgesamt, j: Stellen nach dem Komma
`Serial`	Automatisch inkrementierende Ganzzahl, oft für Primärschlüssel
`Char(n), Varchar(n)`	Text, bei Char wird bei kürzerer Eingabe mit ' ' aufgefüllt, bei Varchar nicht
`create domain`	Definiert einen benutzerdefinierten Datentyp

Programmiermethoden in SQL

5.9 Zugriff auf die DB

Der Zugriff auf die Datenbank kann von verschiedenen Gruppen erfolgen:

- Administratoren: Die Befehle von den Administratoren werden in der Regel direkt auf der Datenbank ausgeführt.
- Anwendungen: Anwendungen nutzen in der Regel eine Schnittstelle (API) der Datenbank, um auf sie zuzugreifen.

- Gelegentliche Nutzer: Die Befehle von gelegentlichen Nutzern werden in der Regel über eine interaktive Anfrage passieren, die erst einen Übersetzer durchlaufen, um dann auf der Datenbank ausgeführt zu werden.

5.10 SQL-Programmiermethoden

Es gibt mehrere Möglichkeiten, wie ein Anwendungsprogramm auf eine Datenbank zugreifen kann:

- Direkter Aufruf
  - Aufruf von SQL-Befehlen direkt im Programm
- Embedded/Dynamic SQL
  - SQL Wird in die Programmiersprache eingebettet
  - SQL-Befehle werden dynamisch zur Laufzeit generiert
- Module Language
  - SQL wird in Module ausgelagert, die in der Programmiersprache aufgerufen werden
- Call-Level APIs
  - Standardisierte Schnittstellen (z.B. ODBC, JDBC) für den Datenbankzugriff
  - Der Programmierer sieht kein SQL mehr (Mappings)

5.11 impedance mismatch

- Relationales Modell wird von objektorientieren Programmiersprachen nicht unterstützt
- SQL basiert auf Mengen, OO-Programmiersprachen auf Objekten
- Keine Pointer o.Ä.
- Lösung: Embedded SQL

5.12 Embedded SQL

- Problem wird (teilweise) umgangen, indem Variablen zwischen SQL und der Programmiersprache 'geteilt' werden
- `exec sql begin declare section;` bzw. `[..]end[..];`
- Darin können Variablen deklariert werden:

- `char var1[20]; int var2;`
- Dann kann in die Variablen geschrieben werden:
- `exec sql insert into [..] values (:var1, :var2);`
- Außerdem kann gelesen werden:
- `exec sql select [..] into :var1 from [..];`
- Variable `SQLSTATE` enthält den Status der letzten SQL-Anweisung und ggf. Fehlercodes

5.13 Cursor in Embedded SQL

Trotzdem bleibt bestehen: Das Ergebnis von SQL-Abfragen sind meistens Mengen. Lösung: Cursors

- Cursors sind Zeiger auf eine Ergebnismenge
- Sie ermöglichen es, durch die Ergebnismenge zu iterieren
- Beispiel:

```
exec sql begin declare section;
char var1[20];
char SQLSTATE[6];
exec sql end declare section;
exec sql declare c1 cursor for select VName
                            from Student;
exec sql open c1;
while(true) {
    exec sql fetch c1 into :var1;
    if (SQLSTATE == '02000') break;
    // Verarbeite var1
}
exec sql close c1;
```

Ursprünglich wurde für Java SQLJ benutzt, mittlerweile ist das aber veraltet.

5.14 Dynamic SQL

- Standard für dynamische SQL-Abfragen in Programmen
- Also keine Deklaration vorab
- Zwei Möglichkeiten:
- Execute Immediate: Direkte Ausführung eines SQL-Befehls

- Prepare and Execute: SQL-Befehl wird vorbereitet und dann (mehrfach) ausgeführt
- Großer Nachteil: SQL Injection möglich, wenn nicht richtig abgefangen wird

5.15 Module Language

- Trennung von SQL und Anwendungsprogramm
- Modul enthält SQL-Befehle und Deklarationen
- Anwendungsprogramm ruft Modul auf

5.16 Call-Level APIs

- Standardisierte Schnittstellen für den Datenbankzugriff
- Beispiel: ODBC, SQL/CLI, JDBC
- Programmierer sieht kein SQL mehr, sondern nur die API-Funktionen
- Mappings zwischen Objekten und Relationen

5.17 ORM: Mappings zwischen Objekten und Relationen

- Alles nicht perfekt, da Objekte und Relationen unterschiedliche Konzepte sind
- ORM (Object-Relational Mapping) versucht, diese Lücke zu schließen
- Mappings zwischen Objekten und Relationen
- Framework verbirgt SQL komplett und bietet objektorientierte API

6 Kapitel 6: Anfrageverarbeitung

6.1 Motivation

- DBMS muss viele Anfragen möglichst schnell und minimalen Ressourcen verarbeiten
- $\Rightarrow$ effiziente Anfrageverarbeitung notwendig

6.2 DBMS Aufbau

- Zuerst gehen die Anfragen an die Anfrageverarbeitung, die aus einem Operator-Evaluierer und einem Optimierer besteht
- Danach durchlaufen sie die Speicherung, zuerst die Dateiverwaltungs-

und Zugriffsmethoden

- Danach den Puffer-Verwalter und den Verwalter für externen Speicherbedarf
- Diese Interagieren mit dem Transaktionsmanagement, das aus einem Transaktionsverwalter, einem Sperrverwalter und einem Wiederherstellungsverwalter besteht
- Danach geht die Anfrage an die Datenbank, die in der Realität aus Dateien und Daten und Indices besteht

Schaubild Kapitel 6, Seite 4

- Die Verwaltung eines DBMS ähnelt sehr der eines Betriebssystems, daher
- schaltet es häufig Betriebsdienstsysteme aus, um gegenseitige Störungen zu verhindern
- Das DBMS weiß mehr über die Zugriffsmuster, was Prefetching(s.u. In der Praxis) ermöglicht

Speicherung

6.3 Speicherung

- In großen Datenbanken sind die Daten zu groß, um in den Hauptspeicher zu passen
- Speicherung soll eine große Menge an Speicherplatz liefern
- Dabei soll der Zugriff für möglichst geringe Kosten möglichst schnell sein
- Die Daten sollen gesichtert sein, Verlust der Daten ist nicht akzeptabel
- In einem normalen Computer wird das folgendermaßen umgesetzt:

Speichertyp	**Kapazität**	**Latenz**
CPU(Register)	Bytes	< 1 ns
Cache-Speicher	Kilo-/Megabytes	< 10 ns
Hauptspeicher(RAM)	Gigabytes	20-100 ns
Flash-Speicher/SSD	Terabytes	30-250 μs
Festplatte/HDD	Tera-Petabytes	3-10ms
Bandautomat	Petabytes	variiert

6.4 Magnetische (Fest-)Platten

- Arme werden auf bestimmte Spur bewegt, Platte dreht konstant
- Spur: Kreis auf de Oberfläche der Platte
- Sektor: Eine Spur wird in Sektoren unterteilt, die kleinste adressierbare Einheit
- Block: Mehrere Sektoren zusammen in eine logische Einheit gefasst
- Zugriffszeit besteht aus:
  - Suchzeit t_s: Zeit, um den Arm auf die richtige Spur zu bewegen
  - Wartezeit t_r: Zeit, bis der Block unter dem Lesekopf ist
  - Lese- bzw. Schreibzeit t_{tr}
  - Gesamte Zugriffszeit $t_a = t_s + t_r + t_{tr}$
- Sequentieller Zugriff, bei dem die Blöcke direkt hintereinander liegen ist schneller als Wahlfreier Zugriff
- Sehr viel schneller.

6.5 Speichernetzwerk(SAN)

- Speichernetzwerk, das aus mehreren Festplatten bzw. Servern mit jeweils mehreren Festplatten besteht
- Zeigt sich aber als 'logische Platten' an das DBMS
- Vorteile: Hardwarebeschleunidung, Redundanz(Fehlertoleranz), einfachere Verwaltung, Flexibilität
- Alternativ: Cloud-Speicher, der über das Internet zugänglich ist
- System kostet mehr und ist Langsamer(Redundanz von bis zu 1s), aber ist zuverlässiger

Verwaltung

6.6 Abstrahierung der Technischen Details vom Speicher(Seiten)

- Eine 'Seite' ist eine logische Speichereinheit(4-64 KB) für die Restlichen Komponenten
- Die Seitennummer weist auf die physische Adresse der Seite hin
  - Betriebssystemdatei inkl. Versatz
  - Kopf-Sektor-Spur von Festplatte

- Angabe für ein Bandgerät und -nummer inkl. Versatz

6.7 leere Seiten

- `insert` sucht leere Seite zum Einfügen
- `delete` gibt die Seite wieder frei
- Leere Seiten müssen Persistenz gespeichert werden
  - Als Liste von leeren Seiten(Hinzufügen, wenn Seite leer wird, entfernen, wenn Seite alloziert wird)
  - Als Bitmap, Bit p wird umgeklappt, wenn die Seite (de-)alloziert wird

Puffer

6.8 Puffer-Verwalter

- Vermittelt zwischen externem Speicher(Festplatten etc) und internem Speicher(Hauptspeicher)
- Verwaltet Teil des Hauptspeichers(buffer pool)
- Externe Seiten werden in den Puffer geladen
- `pin` und `unpin`, um Seiten anzufordern und freizugeben
- Hier wird die Seitennummer angegeben und beim freigeben auch, ob die Seite bearbeitet wurde
- Wenn die Seite bereits im Puffer ist, wird direkt die Referenz zurückgegeben
- Wenn die Seite nicht im Puffer ist, wird sie geladen und die Referenz zurückgegeben
- Es wird gespeichert, wie oft eine Seite angefordert wurde, damit benutze Seiten immer im Puffer bleiben
- Wenn der Puffer voll ist, muss eine Ersetzungsstrategie angewendet werden

6.9 Ersetzungsstrategien

Es wird ausgewählt, welche Seite aus dem Puffer entfernt wird, wenn eine neue Seite geladen werden soll.

- LRU: Seite mit dem am längsten zurückliegenden unpin
- LRU-k: k-letztes unpin, sonst wie LRU

- MRU: Seite mit dem jüngsten unpin
- Random: Zufällige Seite

..

Problem: Seite muss Pincount = 0 haben, aber was ist, wenn es eine solche Seite nicht gibt?

6.10 In der Praxis

- Anfragen werden 'vorhergesagt', Seiten werden im Puffer gehalten, wenn sie wahrscheinlich benötigt werden(Prefetching)
- Fixierungs- und Verdrängungsempfehlungen von höherem Code, wenn dieser Weiß, dass eine Seite länger benötigt wird oder wahrscheinlich nicht mehr

Zugriff

6.11 Datenbank-Dateien

- Der Inhalt von Seiten ist für die Seitenverwaltung nicht relevant
- DBMS verwaltet die Tabellen von Tupeln, Indexstrukturen, ...
- Eine Datei besteht aus einer oder mehreren Seiten
- jede Seite speichert einen oder mehrere Datensätze
- Ein Datensatz entspricht einem Tupel

6.12 Heap-Dateien

- Datensätze werden in willkürlicher* Ordnung gespeichert
- Umsetzung: Verkettete Liste von Seiten, Problem: Man muss viele Seiten durchsuchen(und auch Laden), bis die richtige Seite gefunden wurde
- Alternative: Verzeichnis von Seiten, ist mit Zusatzaufwand für das Verzeichnis verbunden

*Bei der Speicherung von Datensätzen gibt es mehrere Möglichkeiten, die 'richtige' Seite zu finden:

Methode	Beschreibung
Append only	Datensatz wird immer an die aktuelle Seite angefügt, wenn diese voll ist, wird eine neue Seite angelegt.
Best-fit	Datensatz wird auf die Seite geschrieben, wo die 'kleinste Lücke' ist, in die der Satz noch passt man muss erst alle Seiten durchsuchen, um die beste zu finden
First-fit	Datensatz wird auf die erste Seite geschrieben, wo genug Platz ist
Next-fit	Wie first-fit, aber die Suche beginnt bei der letzten einfügeoperation

6.13 Inhalt einer Seite

- Jeder Datensatz hat eine Datensatz-Kennung, die genau beschreibt, wo sich dieser befindet
- Typisch: Seitennummer und Slot, Slots beginnen bei 0, mit fester Slotgröße
- Suche dann innerhalb der Seite mit `slotnr * slotsize`
- Die Seite hat einen Header, der mit einer Bitmap markiert, welche Slots belegt bzw. gültig sind
- Löschung eines Datensatzes: Slot wird als ungültig markiert, die Datensatz-Kennung ändert sich nicht
- Ggf. haben Datensätze unterschiedliche Längen, diese werden dann ans Ende der Seite gepackt und ein Slot-Verzeichnis wird verwendet
- Das Verzeichnis enthält die Startadresse und Länge jedes Datensatzes
- Dadurch kann die Datensatz-Kennung auf die Adresse im Slot-Verzeichnis zeigen und das Feld auf der Seite verschoben werden, ohne dass sich die Kennung ändert
- Das Slot-Verzeichnis wird am Anfang der Seite gespeichert

6.14 Alternative Seiteneinteilung

- Row-Store:
  - Tupel werden immer zusammen gespeichert
  - Vorteil: Schneller Zugriff auf komplette Tupel, vor allem bei `select * from Student where Name = 'Max';` sinnvoll

- Column-Store:
  - Es wird immer ein Attribut zusammen gespeichert, dafür dann für alle Tupel
  - Vorteil: Schneller Zugriff auf einzelne Attribute, vor allem bei `select avg(Note) from Klausur;` sinnvoll

Diese Konzepte lassen sich auch kompinieren.

Indexierung

6.15 Effiziente Evaluierung einer Anfrage

- `[..] where plz between 48159 and 48163;`
- Wird Ausgewertet, indem nach plz sortiert wird und danach mit binärer Suche der startwert gefunden und bis zum Endwert iteriert wird
- Problem: Bei der Suche sind die Datensätze wahrscheinlich nicht auf einer Seite(das ist das Prinzip von binärer Suche)
- Lösungsidee: (Binär-)Bäume

6.16 ISAM-Indexierung

- Aufteilung in Datenseiten und Indexseiten
- Indexseiten enthalten Schlüssel und Zeiger auf die Datenseiten
- Datenseiten trotzdem sortiert
- Hunderte Einträge pro Indexseite $\Rightarrow$ kleine Baumtiefe
- Felder fester Länge, Navigation mittels Versatz
- Binärsuche auf Indexseiten möglich
- Indexeinträge:
  - Bestehen aus Schlüssel k, der für die Suche verwendet wird
  - Seperator p, der Referenz auf die nächste Index- oder die Datenseite enthält
  - Seperator p ist der kleinste Schlüssel, der größer als k ist
  - p_0 ist kleiner als der kleinste Schlüssel in der Indexseite
- Braucht totale Ordnung der Schlüssel
- Indexseiten sind in einem Baum organisiert, der die Suche effizient macht

- Sucht zuerst mittels Binärsuche in der 'Wurzelseite' nach dem Seperator p_i, sodass $k_i \leqslant k < k_{i+1}$
- p_i ist dann der Zeiger auf die nächste Seite, die durchsucht werden muss
- Die Suche wird rekursiv fortgesetzt, bis eine Datenseite erreicht ist
- Dann wird auf der Datenseite nach k gesucht und ab da Sequentiell gelesen
- Vorteile: Zugriff auf Daten schnell, da nur wenige Seiten gelesen werden müssen, viel Sequientieller Zugriff
- Nachteile: Zusätzlicher Speicherbedarf für Indexseiten
- Für (einigermaßen) statische Daten ist ISAM sehr gut geeignet.

6.17 Aktualisierungen bei ISAM

- Löschen: Datensatz über Index suchen, anschließend aus der Datenseite löschen
- Einfügen: auf passende Datenseite navigieren und Datensatz einfügen
- Problem: Was ist, wenn die Seite voll ist?
  - Im Vorhinein Platz lassen
  - Überlauf-Seite einfügen
  - Vorteil: Index bleibt statisch
  - Nachteil: Sequentielle Ordnung der Datensätze ist nicht mehr gegeben
- Typisch sind 20% Freiraum
- Da Indexseiten statisch sind, ist keine Zugriffskoordination(bei Mehrbenutzerbetrieb) nötig

6.18 Zugriffskoordination

Sperrt den (Zeitgleichen) Zugriff, besonders nahe an der Wurzel.

6.19 B^+-Bäume

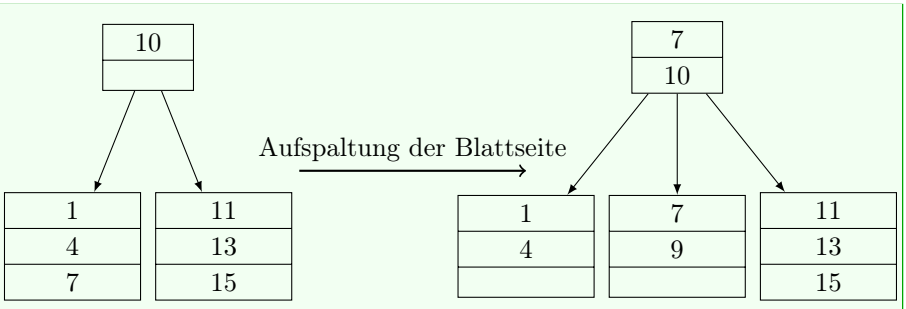
- Ähnlich zu ISAM, aber dynamische Indexseiten
- Dominieren in der Praxis
- Keine Überlauf-Ketten
- Balancierung wird Aufrecht erhalten(50% minimale Besetzung der

Knoten, bei 67% auch B*-Baum genannt)

- Also hat jeder Nicht-Wurzelknoten zwischen d und 2d Einträge, wobei d die Ordnung des Baumes ist
- Hinzufügen und Löschen wird angemessen behandelt
- Referenzierte Datenblätter nicht in sequentieller Ordnung
- Blätter des Indexbaums sind zu doppelt verketteter Liste verbunden
- Es gibt drei Varianten von B^+-Bäumen, sie unterscheiden sich darin, was in den Blättern gespeichert wird:
  1. Vollständiger Datensatz: Die Blätter sind die Datenseiten
  2. Paare <k, rid>, sortiert nach k mit rid als Referenz auf Datensatz
  3. Paare <k, {rid_1, rid_2, ...}>, wobei alle rids den Suchschlüssel k haben
- Variante 2 wird am häufigsten verwendet, nehmen wir ab hier an.
- $\Rightarrow$ Die Datenblätter sind nicht sequentiell geordnet
- Es gibt geclusterte B^+-Bäume, bei denen die Datensätze Sequentiell geordnet sind
- Suche findet wie bei ISAM statt, die Suche bricht aber schon bei den Blättern ab

6.20 Einfügen in B^+-Bäume

- B^+-Baum soll balanciert bleiben
- Suche die Blattseite, auf der der Eintrag eingefügt werden soll
- Falls die Seite noch genug Platz hat(maximal 2d-1 Einträge), wird der Eintrag eingefügt
- Sonst: Teile die Blattseite in Zwei Hälften, wobei die linke Hälfte d Einträge und die rechte Hälfte d-1 Einträge enthält
- Füge die Referenz auf die neue Seite in den Elternknoten ein
- Wird rekursiv bis zur Wurzel durchgeführt
- Falls die Wurzel geteilt wird, wird eine neue Wurzel erstellt, der Baum wächst in der Höhe



Schritt: Einfügen von **9**

6.21 Löschen aus B$^+$-Bäumen

- B$^+$-Baum soll balanciert bleiben
- Suche die Blattseite, auf der der Eintrag gelöscht werden soll
- Falls die Seite noch genug Einträge hat(min. d+1), wird der Eintrag gelöscht
- Sonst: Falls eine der Nachbarseiten mehr als d Einträge hat, wird ein Eintrag von der Nachbarseite übernommen
- Danach wird der Index angepasst
- Falls beide Nachbarseiten nur d Einträge haben, verschmelzt die Seite mit einer der Nachbarseiten
- Der Eintrag in der Elternseite, der auf die Seite zeigt, wird gelöscht
- Das Verschmelzen(oder Umverteilen) wird dann ggf. rekursiv bis zur Wurzel weitergeführt

In der Praxis wird das Verschmelzen meist nicht durchgeführt und die Minimumregel aufgeweicht

6.22 Hash-basierte Indexierung

- Daten werden in mittels einer Hash-Funktion auf Seiten, sogenannte 'Buckets' verteilt
- Hash-Funktion berechnet einen Hashwert für den Schlüssel
- Bei statischem Hashing: Feste Anzahl an Buckets, die Hash-Funktion ist konstant
- Im Bucket sind die Datensätze oder rids gespeichert
- Suche sehr schnell, da nur der Bucket durchsucht werden muss

- Hashing ist besonders geeignet für Gleichheitsabfragen(z.B. `where VName='Max';`)

6.23 Dynamisches Hashing

- statisches Hashing hat das Problem der Wahl der Anzahl der Buckets.
  - Zu wenige Buckets: Viele Kollisionen, langsame Suche
  - Zu viele Buckets: Speicherplatzverschwendung
- Lösung: Anzahl der Buckets wird dynamisch angepasst
- Dann wird die Speicherung langsamer, weil sich eventuell die Hash-Funktion ändert

Indexierung verschnellert den Zugriff auf Daten, aber braucht mehr Speicher und die Indexierung kostet auch Zeit.

Anfragebeantwortung

6.24 Ausführungspläne

- Eine Anfrage wird in einen Ausführungsplan übersetzt
- Der Ausführungsplan beschreibt, wie die Anfrage ausgeführt wird
- Er besteht aus Operatoren, die auf den Daten arbeiten
- Ausführungspläne sind nicht immer eindeutig

6.25 Sortieren

- Sortierung kommt in Datenbanken häufig vor (`asc, desc`, für B^+-Bäume etc.)
- Wie kann aber etwas sortiert werden, was nicht in den Hauptspeicher passt?
- $\Rightarrow$ Zwei-Wege-Mischsortierung

6.26 Zwei-Wege-Mischsortierung

- Sortierung in mehreren Durchgängen mit minimal 3 Pufferseiten(Schneller mit mehr)
- Zeitaufwand ist $O(n \log n)$.
- Startet mit dem Sortieren von jeder einzelnen Seite für sich
- Dann werden rekursiv zwei Sortierte 'Pakete' zusammengeführt, es ist

immer nur eine Seite der 'Pakete' im Puffer

- Um n Seiten zu sortieren, werden $2 \cdot n \cdot (1 + \lceil log_2 n \rceil)$ I/O-Operationen verwendet(bei 3 Pufferseiten)
- Wie wird das bei mehr Pufferseiten weniger?
  - Zwei Möglichkeiten(kombinierbar):
  - Reduktion des initialen 'Pakets', indem mehrere Seiten gleichzeitig sortiert werden
    - Mit k Seiten im Puffer können k Seiten gleichzeitig sortiert werden
    - Dadurch wird die Anzahl der I/O-Operationen reduziert:

$$2 \cdot n \cdot (1 + \lceil log_2 \frac{n}{k} \rceil)$$

  - Reduktion der Anzahl der Durchgänge durch das Sortieren von mehr als 2 'Paketen'
    - Mit k Pufferseiten können $k - 1$ Pakete gleichzeitig sortiert werden
    - Dadurch wird die Anzahl der I/O-Operationen weiter reduziert:

$$2 \cdot n \cdot (1 + \lceil log_{k-1} \frac{n}{k} \rceil)$$

- In der Praxis meist genug Speicherplatz, um die Sortierung in wenigen Durchgängen zu machen

[Hier fehlt was zum externen Mischsortieren]?

6.27 Join-Implementierung

- Intuitive Herangehensweise: Erst Kreuzprodukt, dann Filterung nach Bedingung
- Problem: Zwischenergebnis sehr groß
- Lösung: Join als geschachtelte Schleife
  - Für jedes Tupel der ersten Relation wird die zweite Relation durchsucht
  - Dann wird geprüft, ob für die jeweilige Kombination die Join-Bedingung erfüllt ist
  - Wenn ja, wird das Tupel in das Ergebnis aufgenommen
  - Problem davon: Sehr viele wahlfreie Zugriffe $\Rightarrow$ ineffizient
- Besser: Blockweiser Join mit Schleifen

  - Iteriert auf Blöcken, die dann sequentiell gelesen werden können.
  - Diese werden dann auf die intuitive Weise gejoined und das Ergebnis angehängt.
  - Wichtig: der Join muss im Hauptspeicher ausführbar sein.
  - $\Rightarrow$ mehr Speicherplatzverbrauch, aber weniger langsam
- Wenn die Bedingung ein Gleihheitsprädikat ist, kann eine Hash-Tabelle für den äußeren Block deutlich beschleunigen.
- Weil man nur das Bucket mit der Gleichheit durchsuchen muss
- Wenn bereits ein Index für eine Relation vorhanden ist, die mit der Bedingung verträglich ist(SARGable), kann diese Relation auch nach innen geschoben und dann damit gearbeitet werden.
- Alternative: Hash-Join
  - R und S werden anhand einer Hash-Funktion partitioniert
  - Dann wird join(R_i, S_i) für alle i berechnet werden (einfacher)
  - Anzahl der Partitionen sollte so gewählt werden, dass der Join im Hauptspeicher berechnet werden kann

6.28 Gruppierung + textttunique-Behandlung

- Herausforderung: identische Datensätze in einer Datei finden
- Entweder bzgl. Gruppierungsattributen oder bzgl aller Attribute
- Umsetzung mit Sortierung oder Hash-Join

6.29 Selektion und Projektion

Projektion π

- Entfernen nicht benötigter Spalten
- Eliminierung von Duplikaten

Selektion σ

- Ablaufen aller Datensätze
- Eventuell Sortierung oder Index ausnutzen

6.30 Pipelining

- Bisher: Operatoren verarbeiten ganze Dateien
- Führt zu langen Antwortzeiten etc.

- Alternativ könnte der Operator direkt seine Ergebnisse an den nächsten senden
- Wichtig: größe der Brocken, die weitergegeben werden
- Ausführung wird dadurch deutlich schneller

6.31 Blockierende Operatoren

- Pipelining funktioniert nicht für alle Operatoren, z.B.
  - Misch-Sortierung
  - Gruppierung, Eliminierung von Duplikaten, Max/Min
- Solche Operatoren bekommen immer gesamtes Zwischenergebnis

Anfrageoptimierung

6.32 Anfrageoptimierung

- Ausführungspläne wichtigste Entscheidung
- Welche Implementierung von Join-Operatoren?
- Welche Blockgößen, Pufferallokation?
- Soll ein Index angelegt werden?
- Einige Optimierungen können unabhängig von den Daten erfolgen
  - Selektionsprädikate früh anwenden, einfacher machen
  - Geschachtelte Anfragen entschachteln, Joins explizit machen
  - Duplikat-Eliminierung vermeiden
- Datenabhängige Optimierung macht der Optimiser
  - Kostenbasiert auf Basis der Daten, größen der DB

7 Kapitel 7: Transaktionen

7.1 Transaktion

- logische Verarbeitungseinheit auf einer DB
- typischerweise mehrere Operationen
- werden mit `begin_transaction;` gestartet
- und mit `end_transaction;` beendet

- müssen mit `commit;` physisch durchgeführt werden oder mit `abort` abgebrochen werden
- ACID-Eigenschaften:
  - Atomicity: Alles oder nichts
  - Consistency: Vorher alles ok, hinterher alles ok
  - Isolation: Jede/r denkt, er/sie sei alleine auf der DB
  - Durability: Transaktion bestätigt? Dann sind die Daten sicher
- Fehler in Transaktionen
  - Lost Update: Zwei Transaktionen greifen auf dasselbe Datenobjekt zu
  - Dirty Read: Eine Transaktion, die später abgebrochen wird, schreibt ein Datenobjekt, dass dann von einer anderen Transaktion gelesen wird
  - Ghost-Update: Aggregation wird von einer Transaktion berechnet, eine andere Transaktion aktualisiert davon betroffene Datenobjekte
  - Unrepeatable Read: Eine Transaktion liest ein Datenobjekt zweimal, dazwischen wird es von einer anderen Transaktion aktualisiert

Scheduling

7.2 Scheduling

- Bei großen Mengen von Transaktionen sollten die Transaktionen nicht nacheinander, sondern nebenläufig ausgeführt werden
- Aber es soll keine Anomalien auftreten. Das ist die Aufabe eines Schedulers
- Konflikte zwischen zwei Operationen liegen vor, wenn
  - Sie zu unterschiedlichen Transaktionen gehören
  - Sie auf dasselbe Datenobjekt zugreifen
  - und mindestens eine Operation eine Schreiboperation ist
- Nicht konfliktäre Operationen können gleichzeitig ausgeführt werden

7.3 Korrektheitsprüfung mit Serialisierungsgraphen

- Gerichteter Graph mit allen Transaktionen, Kanten nur von Transaktionen zu anderen Transaktionen, bei denen eine Operation, die im Konflikt steht früher ausgeführt wird.
- Wenn der Graph keine Zyklen enthält, ist S serialisierbar, sonst nicht

Sperrverwaltung

7.4 Sperrverwaltung in Transaktionen

- Transaktion kann mit `lock_item(X)` das Datenobjekt X für andere Transaktionen sperren.
- Entsperrung durch `unlock_item(X)`.
- Wenn eine Transaktion ein Datenobjekt sperren will, das bereits gesperrt ist, muss es warten, bis es von der ursprünglichen Transaktion entsperrt wird.
- Kann zu Deadlocks oder Starvation führen.

8 Kapitel 9-10

Drehen sich um GraphDB und verteilte DB, Nur wissensfragen, kurz das entsprechende Skript durchlesen. Kapitel 1, 8 und 11 sind nicht Klausurrelevant.