Universität Münster

Datenbanken Zusammenfassung

Peter Minor
Sommersemester 2025

18. Juli 2025

Inhaltsverzeichnis

1 Kapitel 1: Einführung

Sehr viel Geyappe über Datenbanken und Entwurfsmodelle, später.

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.

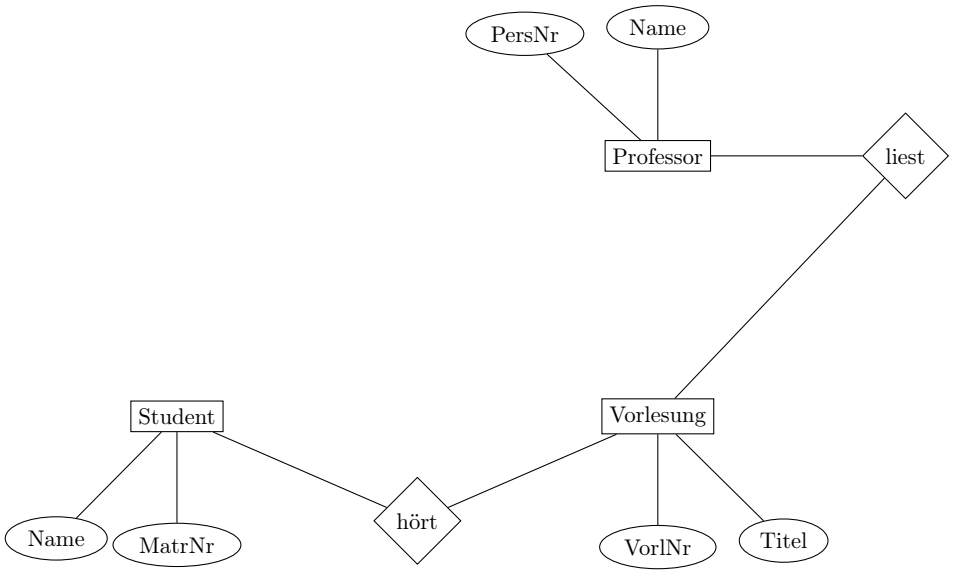
2.6 Partizipation

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:

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

3.5 Projektion (π)

Reduziert die Anzahl der Attribute. Beispiel:

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

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

Klassische Mengenoperationen für Relationen mit gleichem Schema.

3.7 Kartesisches Produkt ($\times$)

Kombiniert zwei Relationen durch paarweise Tupelkombination.

3.8 Join ($\bowtie$)

Verknüpft zwei Relationen über gemeinsame Attribute. Spezialformen:

- natürlicher Join
- theta-Join
- equi-Join

3.9 Umbenennung (ρ)

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

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:

$$\alpha \to \beta$$

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 $\alpha \to R$
- **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 Relationenschema 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 $\alpha \to \beta$ 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 Beispiel: Normalisierung auf 3NF

Gegeben sei folgende Relation:

$$R(\underline{MatrNr}, Name, Studiengang, Fakultaet)$$

mit den funktionalen Abhängigkeiten:

F1: MatrNr → Name, Studiengang, Fakultaet
F2: Studiengang → Fakultaet

Analyse:

- F1: MatrNr ist ein Schlüsselkandidat.
- F2: transitive Abhängigkeit: MatrNr → Studiengang → Fakultaet
- ⇒ Verstoß gegen 3NF.

Zerlegung in 3NF:

- $R_1(\underline{MatrNr}, Name, Studiengang)$
- $R_2(\underline{Studiengang}, Fakultaet)$

Ergebnis: Beide Relationen sind in 3NF, keine Redundanz, keine Anomalien.

4.7 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);`
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
`cascade`	?
`set null`	Setzt die Referenz auf null
`set default`	Setzt die Referenz auf den Default-Wert
`No Action/Restrict`	?
Beispiel	Constraints beispiel?
`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

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)