Datenbanken

1 Gründe für Datenbanksysteme

Datenbanken stecken hinter vielen Computeranwendungen.

- Internet
  - Google
  - Foren
  - Gästebücher
- Anwendungen des täglichen Lebens
  - Banküberweisung
  - Krankenhausaufenthalt
  - Reise

Die Datenbank im Hintergrund kümmert sich um die

- Korrektheit der Daten
- Anwenderfreundlichkeit

Sie bietet Mechanismen zum

- Speichern
- Verwalten
- Anfragen

von Daten, ohne dass sich der Anwender oder Anwendungsprogrammierer um Speicherdetails kümmern muss. Informationsmenge verdoppelt sich ca. alle 5 Jahre. Datenbanksysteme bringen

- Ordnung und Struktur

Komplexere Ansprüche der Datenspeicherung

- Daten sollen mehreren Programmen zur Verfügung stehen

↷ Vermeidung von Redundanz und Einhaltung der Konsistenz

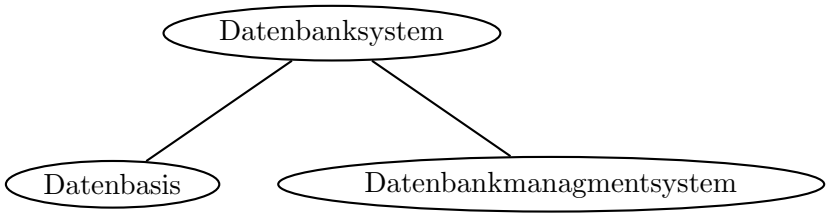
Redundanz: Mehrfachspeicherung (in verschiedenen Dateien)
Konsistenz: widerspruchsfreie und semantisch korrekte Speicherung der Daten

- persistente Daten, die unabhängig, flexibel und geschützt abgelegt werden können

Persistenz: Daten stehen nach Beendigung eines Programms weiterhin zur Verfügung

- mehrere Benutzer sollen gleichzeitig Zugriff auf die Daten haben können
- Daten unterliegen Zugriffsrechten

2 Datenbankmanagementsystem (DBMS)



Datenbasis: Daten eines Problembereiches

Datenbankmanagementsystem: Software zur

- Eingabe, Ausgabe
- Verwaltung,
- Auswertung und Ausgabe

Je nach Managementart unterscheidet man relationale, hierarchische, Netzwerk- oder objektorientierte Datenbanken.

3 Relationale Datenbankmanagementsysteme

Bekannte relationale DBMS sind die von Oracle, MySQL, MS Access oder auch die DBMS-Komponente von OpenOffice.

Relationale Datenbanken organisieren den Datenbestand in Tabellen.

Jede Tabelle wird charakterisiert durch

- den Namen der Tabelle
- die Liste der Attribute (Spaltenüberschriften)
- die Typen der Attributwerte (Welchen Datentyp dürfen die Eintragungen haben?)

Eine Zeile gehört zu einem Datensatz.$\Longrightarrow$ Entität (engl. *entity*)
Eine Tabellenzelle ist ein Datenfeld.

4 Von der Datenanalyse zur Datenbank

4.1 Drei Schritte zur Datenmodellierung

1. **Analyse der Daten und Beziehungen**
   - verbale Beschreibung der notwendigen Daten und deren Beziehungen in der Sprache des Anwenders
   - durch Fragebögen, Bedarfsanalysen, Interviews ...
2. **Freilegen von Entitäts- und Beziehungsmengen**
   - Entwurf des *Entitäten-Beziehungsmodells* (engl. *entity relationship model*)
   - Entitätsmengen $\Longrightarrow$ Rechtecke
     Beziehungsmengen $\Longrightarrow$ Rhomben
   - d.h. strukturierte Darstellung der Fakten der Datenanalyse

3. **Entwurf des relationalen Datenbankschemas**
   - Entitätsmangen und Beziehungsmengen in Tabellenform

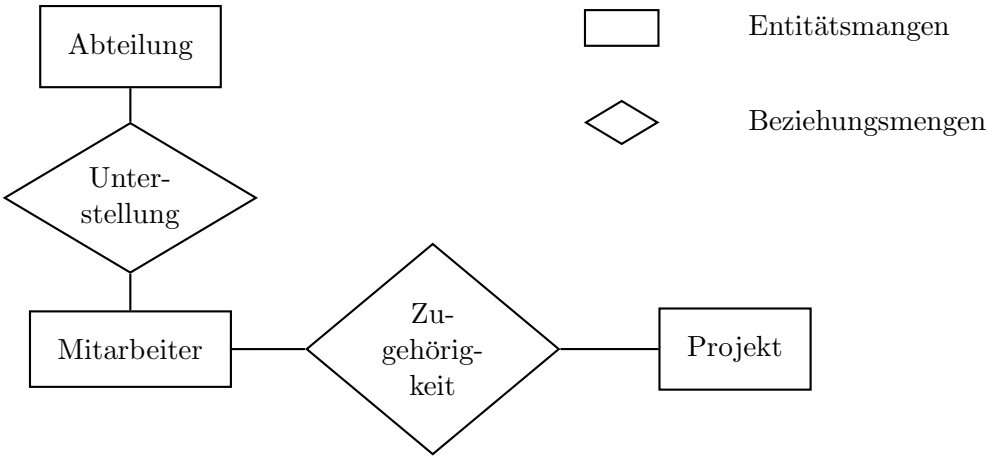
Beispiel

Datenanalyse

Ziel: Zur Projektkontrolle sind pro Abteilung Mitarbeiter, Aufwand und Projektzeiten periodisch zu erstellen

1. Mitarbeiter sind Abteilungen unterstellt, wobei keine mehrfachen Zuordnungen vorkommen
2. Jedem Projekt wird zentral eine eindeutige Projektnummer zugeordnet
3. Mitarbeiter können gleichzeitig an mehreren Projekten arbeiten, wobei die jeweiligen Prozentanteile erfasst werden.
4. ...

Entitäten-Beziehungsmodell



Relationales Datenbankschema

ABTEILUNG

A#	Bezeichnung

MITARBEITER

M#	Name	Straße	Ort

PROJEKT

P#	Inhalt

UNTERSTELLUNG

A#	M#

ZUGEHÖRIGKEIT

M#	P#	%-Anteil

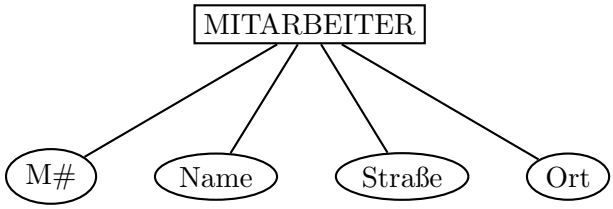
5 Das Entitäten-Beziehungsmodell

5.1 Beziehungen

Entitäten sind wohlunterscheidbare Objekte der der ealen Welt oder unserer Vorstellung. (Individuum, Gegenstand, abstrakter Begriff, Ereignis)

Entitäten des gleichen Typs werden zu *Entitätsmengen* zusammengefasst und durch Merkmale weiter charakterisiert.

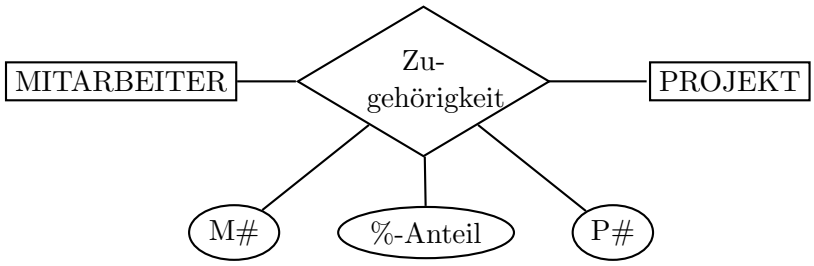
Jede Entitätsmenge erhält einen Identifikationschlüssel, bestehend aus einem Merkmal oder einer minimalen Merkmalskombination, der die Entitäten innerhalb der Entitätsmenge eindeutig festlegt.



Entitätsmenge MITARBEITER

Identifikationsschlüssel:	Mitarbeiternummer als künstlicher Schlüssel
Entität:	Mitarbeiter Meier, wohnhaft in Berlin in der Schulzestraße
Entitätsmenge:	Menge aller Mitarbeiter mit den Merkmalen Name, Ort und Straße

Beziehungen zwischen den Entitäten bilden ebenfalls eine Menge und können durch eigene Merkmale näher charakterisiert werden.



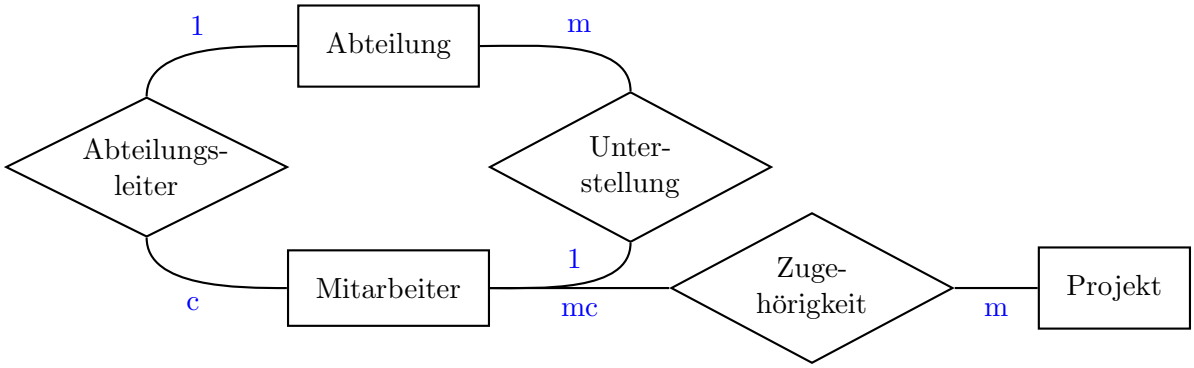
Identifikationsschlüssel:	zusammengesetzter Schlüssel aus Mitarbeiter- und Projektschlüssel
Beziehung:	Mitarbeiter Meier arbeitet zu 70% am Projekt P17
Beziehungsmenge:	Menge aller Mitarbeiter-Projekt-Zugehörigkeiten mit Merkmalen Mitarbeiternummer, Projektnummer und %-Anteil
Assoziationen:	Sicht aus Mitarbeiter bzw. Prospektperspektive • ein Mitarbeiter kann an mehreren Projekten arbeiten • an einem Projekt können mehrere Mitarbeiter arbeiten

5.2 Assoziationen

Unter *Assoziationen* einer Entitätsmenge EM_1 nach einer zweiten Entitätsmenge EM_2 versteht man die Bedeutung der Beziehung in dieser Richtung.

Assoziationstypen:	Typ 1:	genau ein	eindeutige Assoziation	1..1
	Typ c:	kein oder ein	bedingte Assoziation	0..1
	Typ m:	ein oder mehrere	komplexe Assoziation	1..m n..m
	Typ mc:	kein, ein oder mehrere	bedingt-komplexe A.	0..m

Welche Assoziationstypen kann man an die Verbindungen der folgenden Grafik antragen?



Die *Mächtigkeit* einer Beziehung ergibt sich aus dem Paar beider Assoziationstypen.

Beispielsweise ist die Beziehung ZUGEHÖRIGKEIT zwischen MITARBEITER und PROJEKT von der Mächtigkeit (mc,m), also (bedingt-komplex, komplex).

6 Regeln für das relationale Datenbankschema

Regel 1: Jede Entitätsmenge muss als eigenständige Tabelle mit einem eindeutigen Primärschlüssel definiert werden.

ABTEILUNG

A#	Bezeichnung

MITARBEITER

M#	Name	Straße	Ort

PROJEKT

P#	Inhalt

Regel 2: Jede Beziehungsmenge kann als eigenständige Tabelle definiert werden. Dabei müssen die Identifikationsschlüssel der zugehörigen Entitätsmengen als Fremdschlüssel in dieser Tabelle auftreten.

ABTEILUNGSLEITER

A#	M#

ZUGEHÖRIGKEIT

M#	P#	%-Anteil

UNTERSTELLUNG

A#	M#

Die Anwendung dieser beiden Regeln führt nicht immer zu einem optimalen relationalen Datenbankschema. So entstehen eventuell zu viele Tabellen.

Hier kommen nun die Mächtigkeiten ins Spiel.

Regel 3: Jede *komplex-komplexe Beziehungsmenge* muss als eigenständige Tabelle definiert werden.
Der Primärschlüssel der Beziehungsmengentabelle ist entweder aus Fremdschlüsseln zusammengesetzt oder ein anderer Schlüsselkandidat.

Das trifft z.B. für die ZUGEHÖRIGKEIT zu.

Regel 4 Eine *einfach-komplexe* Beziehungsmenge kann ohne eine eigenständige Beziehungsmengentabelle durch die die beiden zugehörigen Entitätsmengentabellen ausgedrückt werden. Dazu werden in der Tabelle mit der einfachen Assoziation (Typ 1) ein Fremdschlüssel mit Rollennamen und ggf. weitere Merkmale geführt.

Nach dieser Regel entfällt die UNTERSTELLUNG.

MITARBEITER

M#	Name	Straße	Ort	A#_Unterst

Regel 5 Eine *einfach-einfache* Beziehungsmenge kann ohne eine eigenständige Beziehungsmengentabelle durch die die beiden zugehörigen Entitätsmengentabellen ausgedrückt werden. Dazu werden in einer der Tabelle ein Fremdschlüssel mit Rollennamen und ggf. weitere Merkmale geführt.

Dies trifft auf die Beziehungsmenge ABTEILUNGSLEITER zu. Zur Vermeidung von *Nullwerten* wird die Tabelle ABTEILUNG um die Mitarbeiternummer des Abteilungsleiters ergänzt.

ABTEILUNG

A#	Bezeichnung	M#_AbtLeiter

Es ist günstig, nach dem Entwurf eines Datenbankschemas dieses auf diese Regeln hin zu überprüfen.

1 Structured Query Language (SQL)

- wichtigste strukturierte Abfrage- und Manipulationssprache
- in den 70-er Jahren in den IBM-Laboren als SEQUEL (Structured English Query Language) entwickelt
- durch das ANSI und die ISO genormt

1 Abfragen

Grundmuster:

Selektiere (`SELECT`) das Merkmal/die Merkmale ...
aus (`FROM`) der Tabelle/den Tabellen ...
wobei (`WHERE`) die Bedingungen ... erfüllt sind

Eine `SELECT-FROM-WHERE`-Anweisung generiert eine Resultattabelle.

Syntax:

```
SELECT   mermalsname[,merkmalsname ...]
FROM     tabellenname[,tabellenname ...]
[WHERE   selektionsbedingung];
```

`SELECT *` würde alle Merkmale der Tabelle auswählen.

Beispiel:

```
SELECT  name,ort
FROM    mitarbeiter;
```

erzeugt eine Tabelle, ...

Aufgabe: Welche (problematische) Ausgabe erzeugt die Anweisung

```
SELECT  ort
FROM    mitarbeiter;
```

Abhilfe schafft das Wort `DISTINCT`.

```
SELECT  DISTINCT ort
FROM    mitarbeiter;
```

Es ...

Selektionbedingungen könne durch die logischen Operationen `AND`, `OR` und `NOT` verknüpft werden, ggf. mit Klammern.

Ebenso sind Vergleichsoperationen `=`, `<` `<=`, `<>` usw. zugelassen oder auch attribut NOT IN tabelle.

Mit `AS` kann man in der Resultattabelle einen neuen Merkmalnamen zuweisen.

Aufgabe: Was bewirkt

```
SELECT  M', name, lohn/12 AS monatsgehalt
FROM    personal
WHERE   monatgehalt<2000;
```

Gleichlautende Attribute aus verschiedenen Tabellen kann man dur Voranstellen der Tabellennamen unterscheiden, z.B. `...WHERE Mitarbeiter.MNummer=arbeitetIn.MNummer`.

Mit `LIKE` kann nach Wörtern oder Teilwörtern gesucht werden. Dabei steht `%` für eine beliebige Zeichenkette und `_` für einen einzelnen Buchstaben.

Aufgabe: Was bewirken dann die Anweisungen

```
SELECT  *
FROM    mitarbeiter
WHERE   ort LIKE 'B%';  oder auch  Ort LIKE '_e___'
```

Weitere Funktionen

- `COUNT(`*merkmalsname*`)` zählt die Anzahl der Werte in der Spalte
- `SUM(`*merkmalsname*`)` gibt die Summe der Spaltenwerte zurück
- `AVG(`*merkmalsname*`)` berechnet den Durchschnitt der Spaltenwerte
- `MIN(`*merkmalsname*`)` evaluiert den kleinsten Wert der Spalte
- `MAX(`*merkmalsname*`)` gib den größten Wert der Spalte zurück

Beispiel:

```
SELECT  COUNT(M#)
FROM    mitarbeiter;
```

degeneriert zu einer einspaltigen Tabelle mit einem einzigen Wert.

Eine Überschrift kann man wieder mit `AS` erzeugen:

Beispiel:

```
SELECT  SUM(lohn) AS Lohnsumme
FROM    mitarbeiter;
```

Sortieren und Gruppieren

Beispiel:

```
SELECT    *
FROM      mitarbeiter
ORDER BY  name ASC;
```

sortiert ...

Beispiel:

```
SELECT    *
FROM      mitarbeiter
GROUP BY  ort
ORDER BY  ort;
```

`GROUP BY` erzeugt eigentlich mehrere gleicharige Tabellen hintereinander. Darauf läßt sich z.B. `COUNT` gut anwenden, weil dieser Befehl jtzt auf jede Teiltabelle einzeln wirkt.

Beispiel:

```
SELECT    Ort, COUNT (Name) AS Anzahl
FROM      mitarbeiter
GROUP BY
GROUP BY  ort;
```

erzeugt eine Tabelle

Ort	Anzahl
Berlin	20
Potsdam	3
Straußberg	4

Tabellen verknüpfen mit JOIN

Beispiel:

```
SELECT  *
FROM    mitarbeiter JOIN abteilung;
```

bildet eine Kreuztabelle aus den beiden Tabellen.

Probiert den noch besseren NATURAL JOIN sowie den LEFT JOIN. (RIGHT JOIN funktioniert aktuell bei SQLite nicht.)
LEFT JOIN erwartet noch den Befehl ON. (tabelle1 LEFT JOIN tabelle2 ON attribut1=attribut2)

Vereinigung

Ebenso interessant ist die Vereinigung UNION bzw. UNION ALL.
Mit diesen Befehlen kann man zwei Tabellen vereinigen.

Dann müssen die Tabellen jedoch vereinigungskompatibel sein, d.h. die Spalten müssen vom gleichen Datentyp sein.

Beispiel:

```
SELECT       Name, Ort
FROM         mitarbeiter
UNION [ALL]
SELECT       Name, Ort
FROM         kunden
```

Die beiden SELECT-Klauseln müssen die gleiche Anzahl an Spalten besitzen.