VIRN DB 19 BUTSCHER

01. Daten

Im Allgemeinen handelt es sich bei Daten um einen beliebigen Zeichensatz, der Informationen darstellt und dem Zweck der Verarbeitung dient. Man unterscheidet:

- **Unstrukturierte Daten**
  Daten ohne erkennbare Struktur / mit einer unterschiedlichen Menge an Attributen.
  - Beliebige Videos, Bilder, Texte
- **Semi-Strukturierte Daten**
  Datenbestand, der nicht streng typisiert ist, sondern einen Teil der Strukturinformation in sich trägt.
  - Daten im HTML5- oder XML-Format
- **Strukturierte Daten**
  Jede Instanz hat die gleiche interne Struktur / Menge an Attributen.
  - In einer Datenbank abgelegte Kundendaten

02. Metadaten

Metadaten sind strukturierte Daten, die andere Daten beschreiben, um diese leichter auffinden und unterscheiden zu können.

Datenbestand:	Bücher
Metadaten:	ISBN, Titel, Verlag, Genre

03. Relation

Eine Relation entspricht einer Tabelle mit n Spalten, m Zeilen und n × m Zellen.

- Spalte = Attribut
  Beschreibt die Art der Daten
- Zeile = Datensatz / Tupel
  Beinhaltet die erfassten Ausprägungen einer Instanz
- Zelle = Attributwerte
  Konkrete Ausprägung des Attributs im Datensatz

04. Datenbanksystem (DBS)

Ein Datenbanksystem besteht aus einer oder mehreren Datenbanken (DB), welche die Datenbasis darstellen und aus einem Datenbank-Management-System (DBMS), das die Verarbeitung der Daten ermöglicht.

Funktionen / Vorteile eines Datenbanksystems

- Datenmanipulation
  Möglichkeit Datensätze zu erzeugen, zu bearbeiten und abzufragen (via SQL)
- Effizienz
  Ermöglicht eine effektive Datenverarbeitung.
- Paralleler Zugriff
  Mehrere Benutzer können parallel an der Datenbank arbeiten
- Datenkonsistenz
  Daten können nur vollständig und den Regeln entsprechend gespeichert werden.
- Redundanzarmut
  Die mehrfache Speicherung identischer Daten soll verhindert werden.
- Datenschutz
  Der Zugriff darf nur durch autorisierte Benutzer erfolgen (verschiedene Zugriffsberechtigungen)
- Datensicherheit
  Datenverlust wird durch interne Backup- und Prüfmechanismen verhindert.
- Unabhängigkeit
  Datenhaltung und Anwendungsprogramm sind voneinander unabhängig.

05. Datenbank (DB)

Eine Datenbank ist ein elektronisches Archiv, in dem inhaltlich zusammenhängende Daten für einen zentralen Zweck strukturiert gespeichert werden. Man unterscheidet:

- Relationale Datenbanken
- NoSQL Datenbanken
- In-Memory-Datenbanken

Datenbank-Management-System (DBMS)

Ein Datenbank-Management-System ist eine Software, welche die Abfrage und sichere Bearbeitung einer Datenbank ermöglicht.

- **Funktionen eines Datenbank-Management-Systems**
  - Aufbau der Datenbank
  - Änderung der Datenbank
  - Abfrage der Datenbank
  - Verwaltung von Metadaten
  - Kontrolle der Datenbank
  - Sicherstellung von Datenintegrität, Datensicherheit und Datenschutz

Create
Read
Update
Delete

- **Populäre Datenbank-Management-Systeme**
  - Oracle
  - MySQL
  - Microsoft SQL Server

Neben serverseitigen Lösungen gibt es auch desktop-orientierte Datenbanksysteme wie beispielsweise MS Access. Jedoch sind diese in ihren Funktionen sowie ihrer Skalierbarkeit stark eingeschränkt.

06. Vorteile und Nutzen von Datenbank-Systemen

- Zeitgleiches Arbeiten
- Flexible Abfragen
- Automatisierbare Backups
- Hohe Datensicherheit
- Effiziente Suche
- Hohe Datenkonsistenz

- Daten sind organisiert
- Zentrale, effiziente, konsistente und permanente Speicherung der Daten
- Gleichzeitiger Zugriff auf Daten durch mehrere Nutzer möglich
- Anwendungsprogramme sind von der physischen Speicherung der Daten unabhängig
- Anwendungen haben nur über das DBMS Zugriff auf die Daten

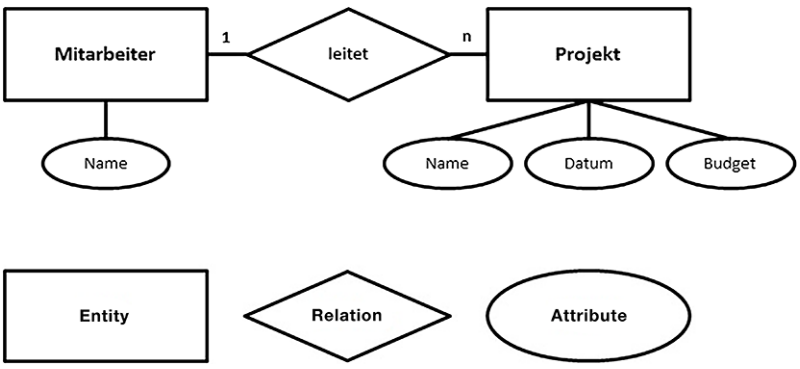
07. Entity-Relationship-Model (semantische Ebene)

Das Entity-Relationship-Model ist ein grafisches Hilfsmittel für den Entwurf von Datenbanken. Es zeigt die realen Beziehungen zwischen Entitäten und deren Kardinalität auf der semantischen Ebene.

- **Vorteile des ERM**
  - Leicht verständlich
  - Intuitive Anwendung
  - Einfache Übertragung in das Relationenmodell nach Codd

08. Bestandteile des Entity-Relationship-Model

- **Entity**
  Objekt aus der realen Welt.
  - Mitarbeiter
  - Abteilungen
  - Kunden
  - Aufträge
- **Relation**
  Beziehung zwischen zwei Entitäten.
  - Mitarbeiter arbeiten in Abteilungen
  - Kunden erteilen Aufträge
- **Attribute**
  Beschreibt die Entity. Attribute geben an, welche Eigenschaften alle Entitys der Entity-Menge gemeinsam haben.
  - Arbeit wird in Stunden gemessen
  - Aufträge haben Auftragsnummern



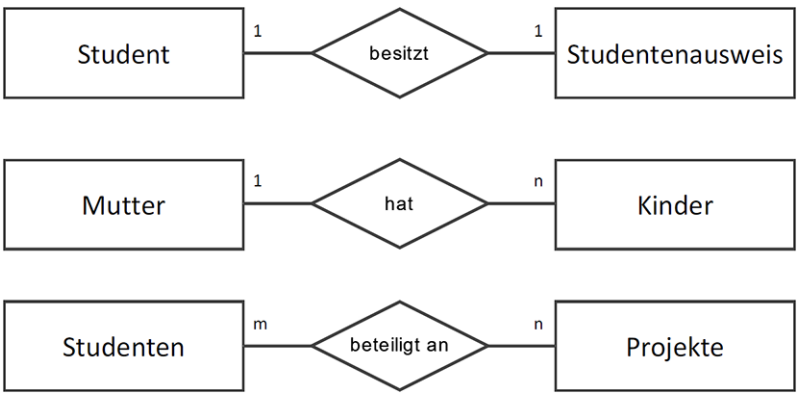
09. Entity-Relationship-Model – Merkmale

- Eine Entität ist ein Objekt aus der realen Welt.
- Gleichartige Entitäten werden zu Entity-Sets zusammengefasst.
- Ein Attribut ist eine Eigenschaft, die allen Entitäten des Entity-Sets gemeinsam haben.
- Eine Entität wird durch ihre Attribute beschrieben; es gibt obligatorische und optionale Attribute
- Es gibt Primär- und Fremdschlüssel.
- Ein Primärschlüssel kann aus einem oder mehreren Attributen zusammengesetzt sein.
- Künstlich erzeugte Primärschlüssel werden als synthetischer Schlüssel bezeichnet.
- Zwischen Entitäten bestehen Beziehungen.
- Beziehungen beschreiben einen Zusammenhang zwischen Entitäten.

10. Entity-Relationship-Model – Kardinalität

Die Kardinalität gibt die Beziehung zwischen zwei Entitäten an. Man unterscheidet drei Arten:

- **1:1 Beziehung**
  Genau eine Beziehung in beide Richtungen
- **1:n Beziehung**
  Keine, eine oder mehrere Beziehungen in eine Richtung
- **n:m Beziehung**
  Keine, eine oder mehrere Beziehungen in beide Richtungen



11. Entity-Relationship-Model – Beispielaufgabe

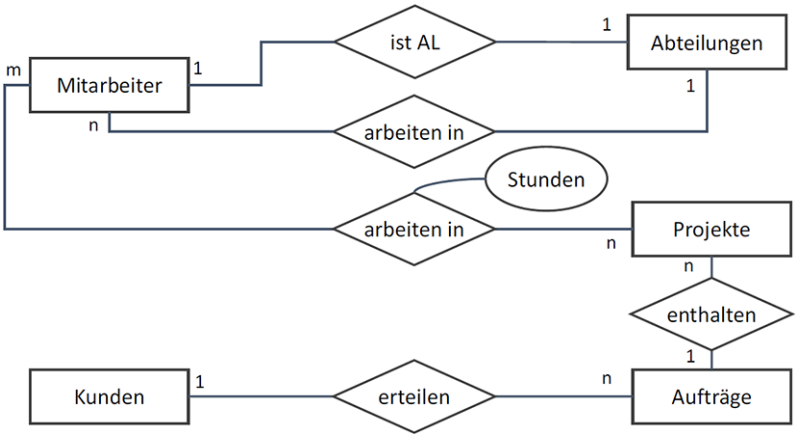
Das »Ingenieurbüro Müller« beschäftigt mehrere Mitarbeiter, von denen jeder genau einer Abteilung zugeordnet ist.

Ein Mitarbeiter in jeder Abteilung übernimmt die Aufgaben eines Abteilungsleiters.

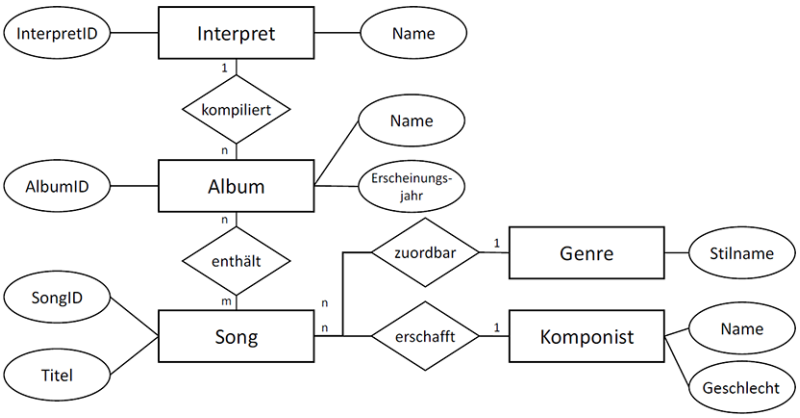
Die Mitarbeiter arbeiten an verschiedenen Projekten die von Kunden in Auftrag gegeben wurden.

Für jeden Mitarbeiter soll erfasst werden, wie viel Zeit er für eines der Projekte verwendet hat.

Kunden können mehrere Aufträge erteilt haben, jeder Auftrag kann mehrere Projekte umfassen.



12. Entity-Relationship-Model – Weiteres Beispiel



13. Datenmodell (logische Ebene)

Beschreibung aller in der Datenbank enthaltenen Entitäten, ihrer Attribute und Beziehungen sowie Zugriffsregeln für Nutzer.

Das ERM wird mit Hilfe des logischen Datenbankdesigns in ein Relationenmodell überführt. Hierbei werden die Daten in Relationen (Tabellen) dargestellt.

14. Relationales Datenbankmodell nach Codd

- Daten werden in Relationen (Tabellen) mit einer festen Anzahl von Attributen (Spalten) und beliebig vielen Datensätzen (Zeilen) mit Merkmalsausprägungen (Zellen) abgelegt.
- Durch Primärschlüssel können Datensätze eindeutig identifiziert werden
- Zwischen den Relationen können Beziehungen bestehen

15. Eigenschaften von Transaktionen in relationalen Datenbanken (ACID)

Das Akronym ACID beschreibt Regeln und Eigenschaften zur Durchführung von Transaktionen in DBMS.

- **Atomicity**
  Transaktionen werden immer vollständig oder gar nicht ausgeführt.
- **Consistency**
  Transaktionen müssen eine Datenbank immer in einem konsistenten Zustand (integer) hinterlassen.
- **Isolation**
  Parallel laufende Transaktionen können sich nicht gegenseitig beeinflussen.
- **Durability**
  Erfolgreiche Transaktionen bleiben dauerhaft gespeichert.

Halten die Transaktionen das ACID-Prinzip ein, gelten die Informationen im Datenbanksystem als verlässlich und konsistent.

16. Normalisierung von Datenbanken

Unter Normalisierung versteht man die Aufteilung von Attributen in mehrere Relationen, sodass eine Form entsteht, die Redundanzen und Inkonsistenzen weitestgehend verhindert.

- Redundanzen
  Mehrfach gespeicherte Informationen
- Inkonsistenzen
  Widersprüchliche / unbeständige Daten

Die Normalisierung erfolgt schrittweise bis zum erforderlichen Grad.

- **1. Normalform**
  Alle Attribute sind atomar, also nicht weiter teilbar.
  - Fehlerbehebung: Attribute weiter unterteilen.
- **2. Normalform**
  1. NF + jedes Nicht-Schlüssel-Attribut ist voll funktional vom Primärschlüssel abhängig.
  - Fehlerbehebung: Attribute in neue Tabelle auslagern.
- **3. Normalform**
  2. NF + es existieren keine funktionalen Abhängigkeiten zwischen Nicht-Schlüsselattributen.
  - Fehlerbehebung: Attribute in neue Tabelle auslagern.

17. Denormalisierung von Datenbanken

Normalisierte Datenbanken eignen sich nicht für eine intuitive Analyse.

Werden Verknüpfungen aufgelöst und Attribute in einzelnen Relationen zusammengeführt spricht man von Denormalisierung.

Hierdurch wird Übersichtlichkeit und Abfrageeffizienz auf Kosten von Redundanzfreiheit und Konsistenz erkauft.

18. Integritätsarten

- **Entity-Integrität**
  Jeder Datensatz muss über einen Primärschlüssel jederzeit eindeutig identifizierbar sein.

- **Referentielle Integrität**
  Für jeden Fremdschlüsselwert muss es jederzeit den korrespondierenden Primärschlüsselwert geben.

- **Semantische Integrität**
  Die formal korrekte Dateneingabe durch den Benutzer ist gewährleistet.

- **Ablaufintegrität**
  Die Konsistenz der Datenbank ist auch bei konkurrierendem Zugriff mehrerer Benutzer sichergestellt.

19. Primärschlüssel (PK)

In jeder normalisierten Relation muss ein Attribut (oder mehrere Attribute) als Primärschlüssel definiert sein. Über den Primärschlüssel können einzelne Datensätze eindeutig identifiziert werden.

Die Werte des Primärschlüssels sind immer eindeutig, obligatorisch und nicht NULL.

20. Fremdschlüssel (FK)

Fremdschlüssel sind Attribute, die auf Primärschlüssel anderer Relationen verweisen. Der Fremdschlüssel kann nur Werte annehmen, welche in der Referenztabelle vorhanden sind (referentiellen Integrität).

Fremdschlüssel sind optional und dürfen NULL sein.

Das relationale Datenmodell nutzt Fremdschlüssel zur Realisierung von Beziehungen zwischen Entity-Mengen.

- **1:1 oder 1:n Beziehung**
  - Verknüpfung über Fremdschlüssel
- **n:m Beziehung**
  - Verknüpfung über zusätzliche Beziehungstabelle mit zwei Fremdschlüsseln.

21. SQL (Structured Query Language)

SQL ist eine Datenbanksprache zur Definition von Datenstrukturen sowie zum Bearbeiten und Abfragen von Datenbeständen.

- Einfacher Aufbau
- An englische Sprache angelehnt
- Case-Insensitive

DDL – Data Definition Language

Beinhaltet Sprachelemente zur Definiton von Datenbankobjekten – anlegen / ändern / löschen.

- Relationen inkl. Schlüssel und Attributen anlegen
- Datentypen der Attribute festlegen
- Änderungen an Relationen vornehmen
- Relationen löschen

DML – Data Manipulation Language

Bietet Befehle zur Manipulation von Datensätzen – einfügen / ändern / löschen.

- Einfügen von Datensätzen
- Ändern von Datensätzen
- Zusammenführen von Datensätzen
- Löschen von Datensätzen

DRL – Data Retrieval Language (DQL)

Ermöglicht die Abfrage von Informationen aus der Datenbank – abfragen / filtern / sortieren.

- Abfrage von Datensätzen

DCL – Data Control Language

Berechtigungsvergabe von Lese- und Schreibrechten – Zugriff gewähren / Zugriff verweigern

- Interaktion einschränken
  read / write, read only, write only, none

22. SQL Datentypen

`integer`	Ganzzahlen inkl. 0
`float`	Fließkommazahl
`char(m)`	Zeichenkette mit fester Länge m
`varchar(m)`	Zeichenkette mit variabler Länge m
`date`	Datum yyyy-mm-dd

23. SQL – Data Definition Language (DDL)

Anlegen einer neuen Relation

```
CREATE TABLE professoren (
pers_id integer PRIMARY KEY,
name varchar(50),
anschrift varchar(255)
);
```

Legt eine Tabelle (Relation) mit dem Namen »professoren« und den Spalten (Attribute) »pers_id«, »name« und »adresse« an.

Ändern einer bestehenden Relation

```
ALTER TABLE professoren
ADD postleitzahl varchar(5);
```

Fügt der Tabelle die Spalte »postleitzahl« hinzu.

```
ALTER TABLE professoren
MODIFY postleitzahl integer;
```

Ändert den Datentyp der Spalte »postleitzahl« von »varchar« auf »integer«.

```
ALTER TABLE professoren
DROP postleitzahl;
```

Löscht die Spalte »postleitzahl«.

```
ALTER TABLE professoren
CHANGE adresse anschrift varchar(255);
```

Ändert den Namen der Spalte »anschrift« in »adresse«.

Löschen einer Relation / Datenbank

```
DROP TABLE professoren;
```

Löscht die Tabelle »professoren«.

```
DROP DATABASE datenbank;
```

Löscht die Datenbank »datenbank«.

Primär- und Fremdschlüssel definieren

```
CREATE TABLE wohnort (
ort_id integer PRIMARY KEY,
postleitzahl varchar(5),
ort varchar(50)
);

CREATE TABLE professoren (
pers_id integer,
name varchar(50),
wohnort integer,
PRIMARY KEY (pers_id),
FOREIGN KEY (wohnort)
REFERENCES wohnort(ort_id)
);
```

Legt zwei Relationen an. Hierbei ist das Attribut »wohnort« der Tabelle »professoren« mit dem Attribut »ort_id« der Tabelle »wohnort« verknüpft.

Der Foreign Key in »professoren« verweist auf den Primary Key von »adressen«.

Primärschlüssel ändern

```
ALTER TABLE professoren
DROP PRIMARY KEY;

ALTER TABLE professoren
ADD PRIMARY KEY (pers_id);
```

Löscht den vorhandenen Primärschlüssel und definiert einen neuen Primärschlüssel.

Fremdschlüssel nachträglich hinzufügen

```
ALTER TABLE professoren
ADD FOREIGN KEY (wohnort)
REFERENCES wohnort(ort_id);
```

Verknüpft das Attribut »wohnort« mit dem Attribut »ort_id« der Tablelle »wohnort«.

24. SQL – Data Manipulation Language (DML)

Einfügen von Datensätzen

```
INSERT INTO professoren
VALUES
(0815, "Heinze", "Domstraße"),
(0816, "Meunier", "NULL");
```

oder

```
INSERT INTO professoren
(pers_id, name, adresse)
VALUES
(0815, "Heinze", "Domstraße"),
(0816, "Meunier", "NULL");
```

Fügt der Tabelle zwei Datensätze mit den jeweiligen Merkmalsausprägungen für »pers_id«, »name« und »adresse« hinzu.

Löschen von Datensätzen

```
DELETE FROM professoren
WHERE name = "Heinze";
```

Löscht jeden Datensatz mit dem Wert »Heinze« in der Spalte »name«.

```
DELETE FROM professoren
WHERE pers_id <= 0815;
```

Löscht jeden Datensatz mit einem Wert kleiner-gleich »0815« in der Spalte »pers_id«.

```
TRUNCATE professoren;
```

Löscht alle Datensätze der Tabelle »professoren«.

Ändern von Datensätzen

```
UPDATE professoren
SET name = "Schmitt"
WHERE name = "Meunier";
```

Ändert den Wert »Meunier« in der Spalte »name« in jedem Datensatz in den Wert »Schmitt«.

25. SQL – Data Retrieval Language (DRL)

Einfache Selektion

```
SELECT *
FROM professoren;
```

Fragt alle Spalten der Relation »professoren« ab.

```
SELECT name
FROM professoren;
```

Fragt die Spalte »name« der Relation »professoren« ab.

Selektion mit Filter

```
SELECT pers_id, name
FROM professoren
WHERE adresse = "Domstraße";
```

Fragt »pers_id« und »name« von jedem Datensatz mit dem Wert »domstraße« in der Spalte »adresse« ab.

```
SELECT pers_id, name
FROM professor
WHERE name LIKE "H%";
```

Fragt »pers_id« und »name« von jedem Datensatz mit einem Wert der mit »H« beginnt in der Spalte »name« ab.

Platzhalter werden mit LIKE oder NOT LIKE verknüpft

%	kein oder beliebig viele Zeichen
_	genau ein Zeichen

Selektion mit Sortierung

```
SELECT pers_id
FROM professoren
ORDER BY name DESC;
```

Fragt die Spalte »pers_id« der Relation »professoren« ab und sortiert diese absteigend.

ASC	Ascendend (aufsteigend)
DSC	Descendend (absteigend)

26. SQL DRL – Kreuzprodukt (Kartesisches Produkt / Cross Join)

Über das Kreuzprodukt werden informationen aus zwei Relationen einer Datenbank zusammengeführt.

```
SELECT *
FROM professoren, vorlesungen;
```

oder

```
SELECT *
FROM professoren CROSS JOIN vorlesungen;
```

Jeder Datensatz der ersten Tabelle wird mit jedem Datensatz der zweiten Tabelle verknüpft.

27. SQL DRL – Joins

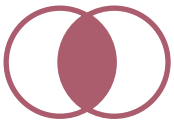
Werden Informationen aus mehreren Relationen einer Datenbank benötigt, können diese über Joins selektiert zusammengeführt werden.

Ein Join bezeichnet die beiden hintereinander ausgeführten Operationen »Kreuzprodukt« und »Selektion«. Die Selektionsbedingung ist dabei üblicherweise ein Vergleich von Attributen.

Inner Join

Der Inner Join verbindet Datensätze aus zwei Tabellen. Hierbei werden nur die Zeilen ausgegeben, in denen in die Werte der angegebenen Spalten übereinstimmen.

```
SELECT *
FROM paare INNER JOIN kinder
ON paare.kinder = kinder.eltern;
```



Zeigt nur Paare, die Kinder haben.

Left Join

Beim Left Join werden die Zeilen ausgegeben, in denen die Werte der angegebenen Spalten übereinstimmen. Sowie alle Datensätze aus der ersten Tabelle, die keinen Join-Partner haben (fehlende Werte = NULL).

```
SELECT *
FROM paare LEFT JOIN kinder
ON paare.kinder = kinder.eltern;
```

Zeigt alle Paare und falls es Kinder gibt, auch diese.

Right Join

Beim Right Join werden die Zeilen ausgegeben, in denen die Werte der angegebenen Spalten übereinstimmen. Sowie alle Datensätze aus der zweiten Tabelle, die keinen Join-Partner haben (fehlende Werte = NULL).

```
SELECT *
FROM paare RIGHT JOIN kinder
ON paare.kinder = kinder.eltern;
```

Zeigt alle Kinder und falls es Eltern gibt, auch diese.

Natural Join

Der Natural Join verbindet Datensätze mit identischen Werten, in Spalten mit gleichem Namen. Spalten mit gleichem Namen werden zusammengefasst.

Achtung: Haben die Tabellen keine Spalten mit gleichem Namen, wird ein Cross Join durchgeführt.

```
SELECT *
FROM eltern NATURAL JOIN kinder;
```