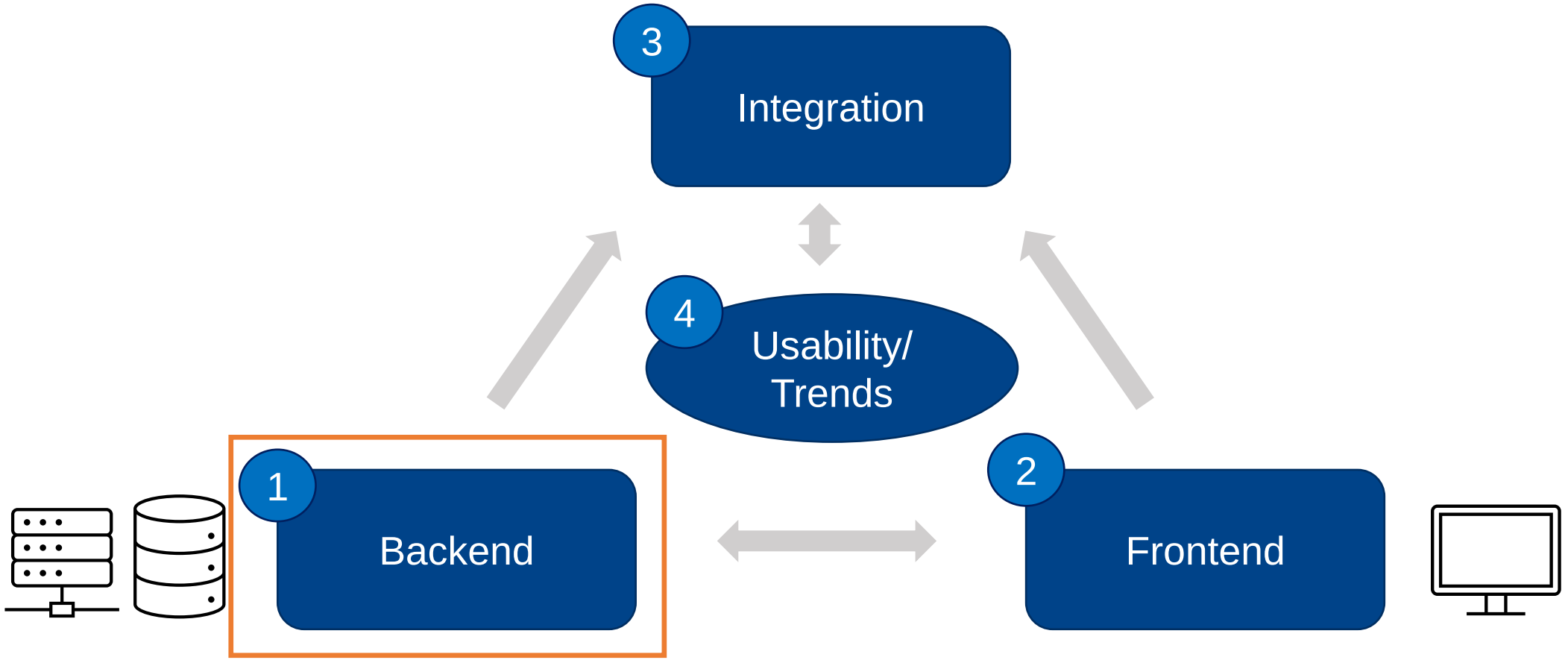
Web Programming

Modul 1.1: Backend - Datenbanken

Überblick über den Kurs



Ziele

In diesem Kapitel werden Ihnen die Grundlagen und wesentlichen Techniken beigebracht, um **Datenstrukturen zu verstehen** und für die Webentwicklung **Daten bereitzustellen**. Es werden hierzu im Folgenden die Grundlagen der **ER-Modellierung** aufgezeigt. Anschließend werden **(relationale) Datenbanken** erklärt und anhand der **Datenbanksprache SQL** durch Abfragen praktisch umgesetzt.

Die Implementierung eines **Backends** in der Webentwicklung erfolgt häufig über die **Programmiersprache Python**. Es werden hierfür die spezifischen Grundlagen der Programmiersprache im Zusammenspiel mit konkreten Code-Beispielen angereichert. Abschließend wird eine **Web-Applikation** für das Backend mit dem **Web-Framework Flask** implementiert.

[ABB001]

Backend – Teil 1

1. **Einführung**
2. **Entity-Relationship-Modellierung**
3. **SQL**
4. **NoSQL**
5. **Python**
6. **Flask**

Datenbanken

Datenbanken und Datenbanksysteme spielen eine wesentliche Rolle in vielen Bereichen der modernen Gesellschaft:

Beispiele:

- in einem Reisebüro eine Reise buchen
- in der Bibliothek ein Buch in einem computergestützten Bibliothekskatalog suchen
- bei einem Medienverlag eine Zeitschrift abonnieren
- bei der Verwaltung von Artikelbeständen im Supermarkt

[SCHI17], [ELMA09]

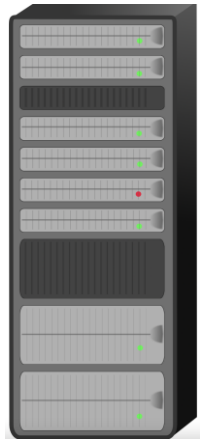
[ABB002]

Datenbankmanagementsysteme

Ein Datenbankmanagementsystem (DBMS) ist ein Softwaresystem (i.e. Sammlung von Programmen), das dem Benutzer das Erstellen und die Pflege einer Datenbank ermöglicht.

Das DBMS vereinfacht damit die Prozesse der Definition, Konstruktion und Manipulation von Datenbanken für verschiedene Anwendungen:

- Die **Definition** einer Datenbank bedeutet die Spezifikation der Datentypen, Strukturen und Einschränkungen für die in der Datenbank zu speichernden Daten.
- Die **Konstruktion** der Datenbank ist der Prozess des Speicherns der Daten auf einem Speichermedium, das vom DBMS kontrolliert wird.
- Die **Manipulation** einer Datenbank beinhaltet Funktionen wie Anfragen der Datenbank, um spezifische Daten abzurufen, Fortschreiben der Datenbank, Änderungen in der Miniwelt und Erzeugen von Berichten aus den Daten.



[ABB003]

[SCHI17], [ELMA09]

Datenbankmanagementsysteme

Beispiele für solche Datenbankmanagementsysteme sind:

- Die My-SQL-Workbench:

[ABB004]

- phpMyAdmin:

[ABB005]

Datenbank und DBMS-Software bilden zusammen ein Datenbanksystem.

Datenmodelle (1/2)

Doch wie genau sind diese Daten in den Datenbanken hinterlegt?

Ein grundlegendes Merkmal des Datenbankansatzes ist, dass er eine gewisse **Datenabstraktion** bietet, indem Details der Datenspeicherung, von denen die meisten Datenbanknutzer nichts zu wissen brauchen, verborgen werden.

Ein **Datenmodell**, d.h. eine Sammlung von Konzepten, die benutzt werden können, um die Struktur einer Datenbank zu beschreiben, bietet die notwendige Grundlage, um diese Abstraktion zu erreichen.

[SCHI17], [ELMA09]

Datenmodelle (2/2)

In diesem Kurs wird auf das **relationale Datenmodell** nach Ted Codd eingegangen. Es existieren darüber hinaus jedoch noch andere Arten.

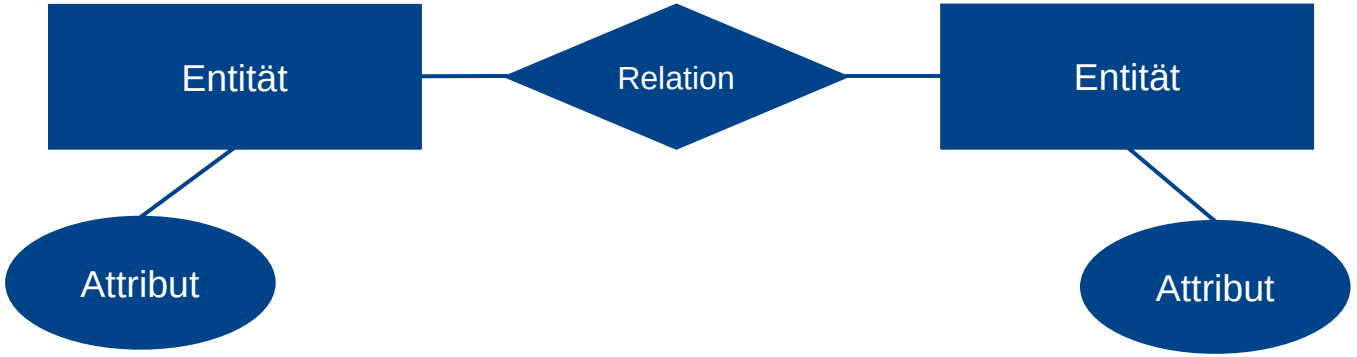
Als Basis für das relationale Datenmodell kann das sogenannte **Entity-Relationship-Modell**, welches die Architektur der Datenbank (also das Verhältnis der einzelnen Daten untereinander) aufzeigt, verwendet werden. Dies ist besonders in der Entwurfsphase einer Datenbank sehr hilfreich, wenn nicht intuitiv klar ist, wie später die Datensätze untereinander zusammenhängen.

[SCHI17], [ELMA09]

Backend – Teil 1

Entity-Relationship-Modell (1/9)

- Das Entity-Relationship-Modell (kurz ER-Modell) beschreibt Daten als Entitäten, Beziehungen und Attribute.
- Eine **Entität** (Entity) stellt ein Objekt oder Konzept aus der realen Welt dar, z.B. ein Produkt oder einen Angestellten, die in der Datenbank beschrieben werden → *Notation als Rechteck*
- Ein **Attribut** stellt eine Eigenschaft dar, die die Beschreibung einer Entität weiter ausführt, z.B. den Namen oder die Gehaltsstufe des Angestellten → *Notation als Kreis*
- Eine **Beziehung** (Relationship) zwischen zwei oder mehr Entitäten stellt einen Zusammenhang zwischen den Entitäten dar, z.B. eine Arbeitsbeziehung zwischen einem Mitarbeiter und einem Projekt → *Notation als Raute*

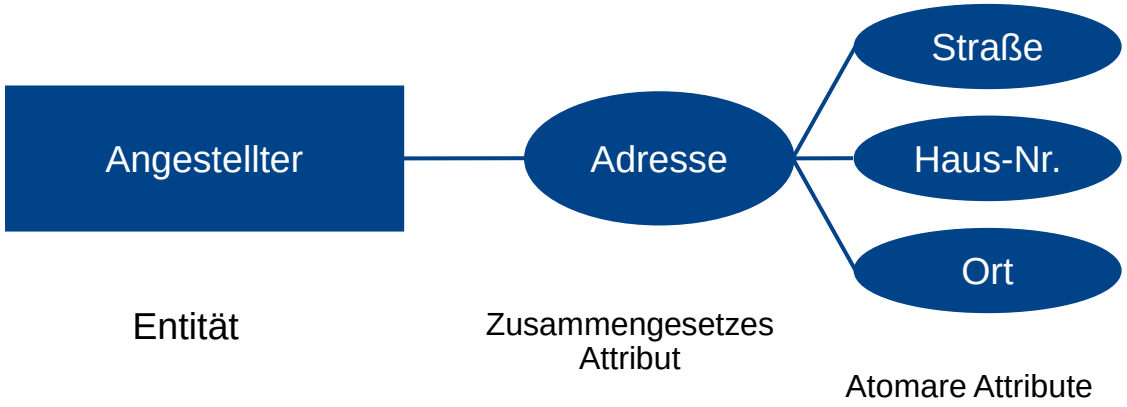


[SCHI14] [ELMA09]

Entity-Relationship-Modell (2/9)

Man unterscheidet zwischen **zusammengesetzten** oder **einfachen** (*atomare*) Attributen. Zusammengesetzte Attribute lassen sich in kleinere Teile zerlegen, die grundlegendere Attribute mit unabhängigen Bedeutungen darstellen. Ein Attribut „Adresse“ der Entität Angestellter lässt sich beispielsweise in die Attribute „Straße“, „Hausnummer“ und „Ort“ unterteilen.

Nicht teilbare Attribute nennt man atomare Attribute. Im oben aufgeführten Beispiel wären „Straße“, „Hausnummer“ und „Ort“ atomare Attribute.

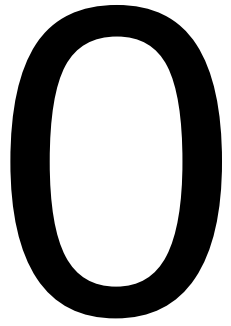


[SCHI17], [ELMA09]

Entity-Relationship-Modell (3/9)

In manchen Fällen trifft auf ein Attribut einer bestimmten Entität vielleicht kein Wert zu. Für solche Fälle wird ein spezieller Wert erzeugt, der als **Nullwert** (NULL) bezeichnet wird.

- Ein Nullwert kann auch benutzt werden, wenn der Wert eines Attributs für eine bestimmte Entität nicht bekannt ist
- Nullwert bedeutet also nichts anderes als *nicht bekannt.*



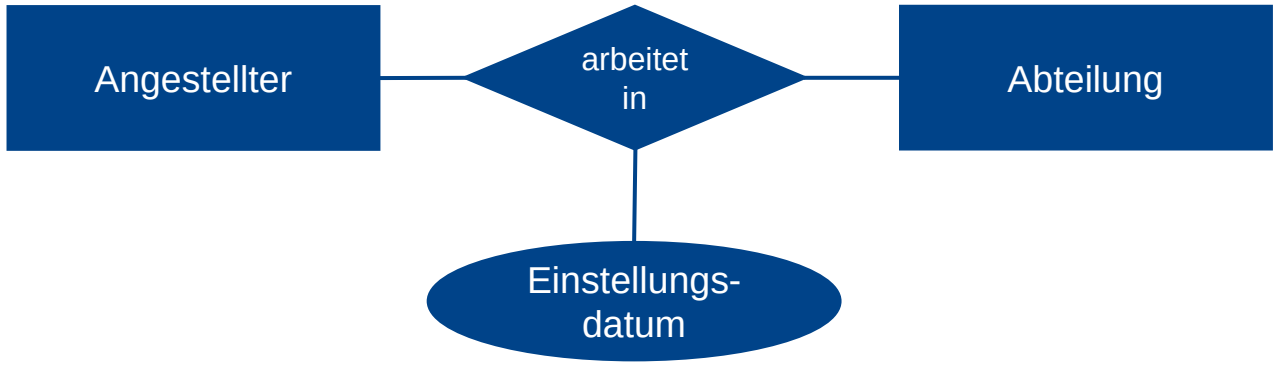
Nullwert

[SCHI17], [ELMA09]

Entity-Relationship-Modell (4/9)

Eine **Beziehung** beschreibt das Verhältnis zweier Entitätstypen miteinander. Beispielsweise arbeitet ein Angestellter in einer Abteilung.

Auch ein Beziehungstyp kann über Attribute verfügen. In unserem Beispiel könnte dies der Beginn des Beschäftigungsverhältnisses als Datum (Einstellungsdatum) sein.



[SCHI17], [ELMA09]

Entity-Relationship-Modell (5/9)

Für Beziehungstypen gelten normalerweise bestimmte Einschränkungen, mit denen die möglichen Kombinationen von Entitäten, die an der entsprechenden Beziehungsmenge teilnehmen dürfen, begrenzt werden.

Diese Einschränkungen werden durch die Situation vorgegeben, die von den Beziehungen dargestellt wird. So kann beispielsweise eine Regel gelten, die besagt, dass jeder Angestellte für **genau eine** Abteilung arbeiten muss; dann müsste man diese Einschränkung im Schema beschreiben.

→ Diese Einschränkung wird auch als **Kardinalitätsverhältnis** bezeichnet

[SCHI17], [ELMA09]

Entity-Relationship-Modell (6/9)

Es gibt verschiedene Notationsformen für das Kardinalitätsverhältnis zweier Entitätstypen. In diesem Kurs werden wir uns auf die sogenannte **Min-Max-Notation** beschränken.

Die Min-Max-Notation besteht aus zwei Zahlen, welche für jeden Entitätstypen angeben, wie oft dieser **mindestens** eine Beziehung mit einem anderen Entitätstyp **eingehen muss** (erste Zahl) und **maximal** eingehen **darf** (zweite Zahl).

Dabei gibt es folgende Ausprägungen:

- keins-zu-eins (0:1)
- eins-zu-eins (1:1)
- keins-zu-viele (0:n)
- eins-zu-viele (1:n)
- viele-zu-viele (n:m)

[SCHI17], [ELMA09]

Entity-Relationship-Modell (7/9)

In unserem Beispiel sieht dies dann wie folgt aus:

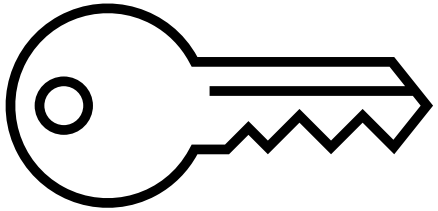
Ein Angestellter kann **mindestens einer** Abteilung zugeordnet werden und darf auch **nur in einer einzigen** Abteilung tätig sein (1:1).

Eine Abteilung besteht aus **mindestens einem** Angestellten, kann aber der **Arbeitsplatz für n** Angestellte sein (1:n).

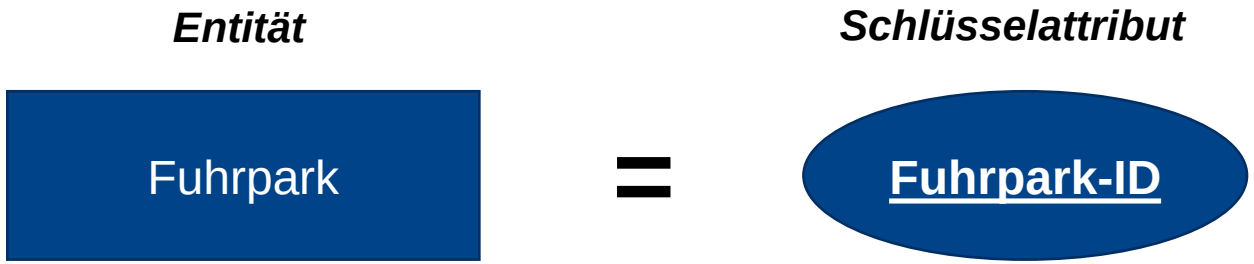
[SCHI17], [ELMA09]

Entity-Relationship-Modell (8/9)

- Eine wichtige Einschränkung für die Entitäten eines Entitätstyps ist der **Schlüssel** bzw. eine Eindeutigkeitseinschränkung für Attribute.
- Ein Entitätstyp hat normalerweise **ein Attribut, dessen Werte sich für jede einzelne Entität der Sammlung unterscheiden.**
- Typischerweise ist das **Schlüsselattribut** eines Entitätstyps eine Kombination aus dem **Entitätsnamen** und einer **eindeutigen Identifikationsnummer (ID)**
- Ein solches Attribut nennt man Schlüsselattribut und anhand seiner Werte lässt sich jede Entität eindeutig identifizieren.



[ABB006]



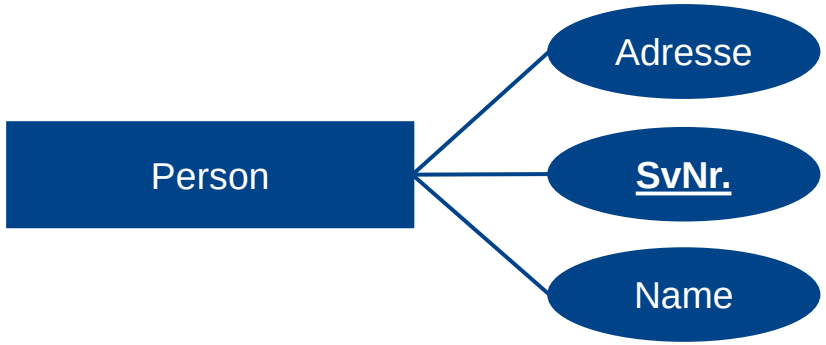
[SCHI17], [ELMA09]

Entity-Relationship-Modell (9/9)

Zum Beispiel ist das Attribut *Name* ein Schlüssel für den Entitätstyp Firma, weil keine zwei Firmen den gleichen Namen haben dürfen. Für den Entitätstyp Person ist die *Sozialversicherungsnummer* (SvNr.) ein typisches Schlüsselattribut.

Manchmal bilden mehrere Attribute zusammengenommen einen Schlüssel, was bedeutet, dass sich die Kombination der Attributwerte jeder Entität unterscheiden muss. Kann ein solcher Schlüssel nicht aus den vorgegebenen Attributen erzeugt werden oder erfordern es die Umstände, so kann ein Schlüssel auch künstlich erzeugt werden. Dieser ist dann häufig eine in der Relation eindeutige Identifikationsnummer (ID).

Im ER-Modell wird jedes Schlüsselattribut als Oval mit darin befindlicher **unterstrichener Attributsbezeichnung** dargestellt.



[SCHI17], [ELMA09]

ERM Q1

Wie könnte die Kardinalitätsbeziehung zwischen den Entitäten „Firma“ und „Abteilung“ aussehen?

[ABB007]

ERM A1

Eine Firma besteht aus **mindestens einer Abteilung**, kann aber **n viele Abteilungen** beinhalten.

- Dies ist nur eine von mehreren möglichen Lösungen.
- Beispielsweise könnte eine Abteilung auch zu mehreren Firmen gehören, also eine 1:n Beziehung eingehen. Wichtig ist dabei immer die Umstände zu erfassen, unter denen das Modell erstellt werden soll.
- So ist z.B. die aufgezeigte Lösung korrekt, wenn es darum geht, nur eine Firma abzubilden. Die Alternativlösung wäre korrekt, wenn mehrere Firmen abgebildet werden sollen.

ERM Q2

- Um die Vorgänge in einem Atomkraftwerk noch besser überwachen zu können und damit Sicherheit gewährleisten zu können, wurden Sie beauftragt ein Datenbankmodell des Atomkraftwerks anzufertigen.

[ABB008]

- Hierzu sollen Sie nun im ersten Schritt ein ER-Modell erstellen, welches die wichtigsten Personalinformationen aufzeigt.

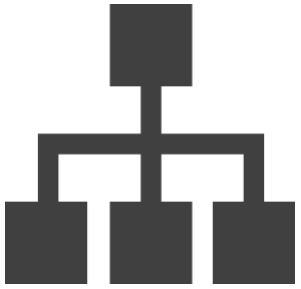
ERM Q2

- Das Atomkraftwerk besteht (stark vereinfacht) aus Angestellten, die in Abteilungen arbeiten und Gehalt aus einer bestimmten Gehaltsstufe beziehen. Zu jedem Angestellten sollen weiterhin Vorname, Nachname und Adresse hinterlegt werden. Zudem verfügt jeder Angestellte noch über eine eindeutige Identifikationsnummer (ID).

- Zu jeder Abteilung ist ebenfalls eine eindeutige ID und ein Name hinterlegt.

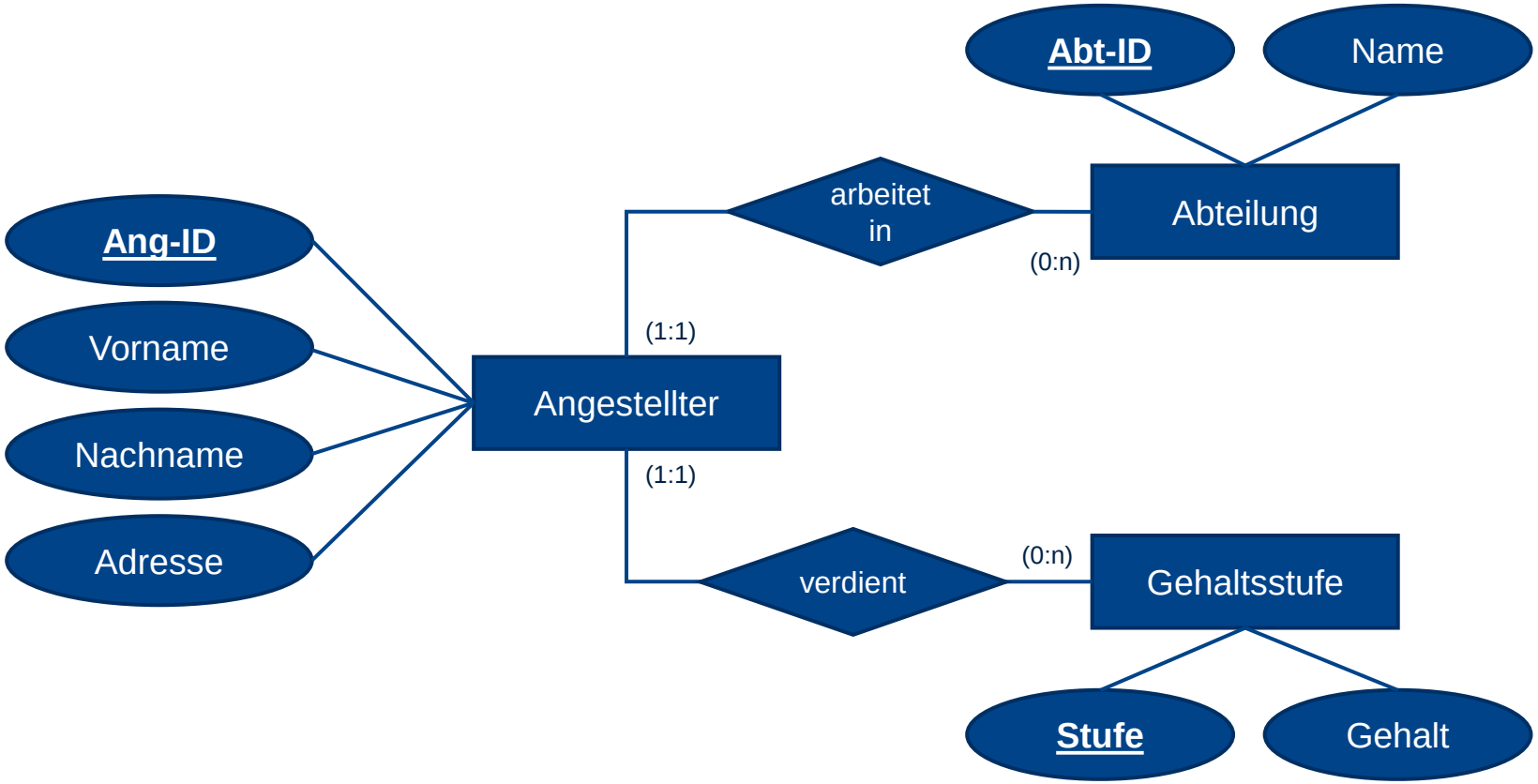
[ABB009]

- Ein Angestellter ist genau einer Abteilung zugeteilt. Eine Abteilung besteht aus mehreren Mitarbeitern. Ihre Untersuchungen haben ergeben, dass im AKW allerdings auch Abteilungen ohne Mitarbeiter existieren.

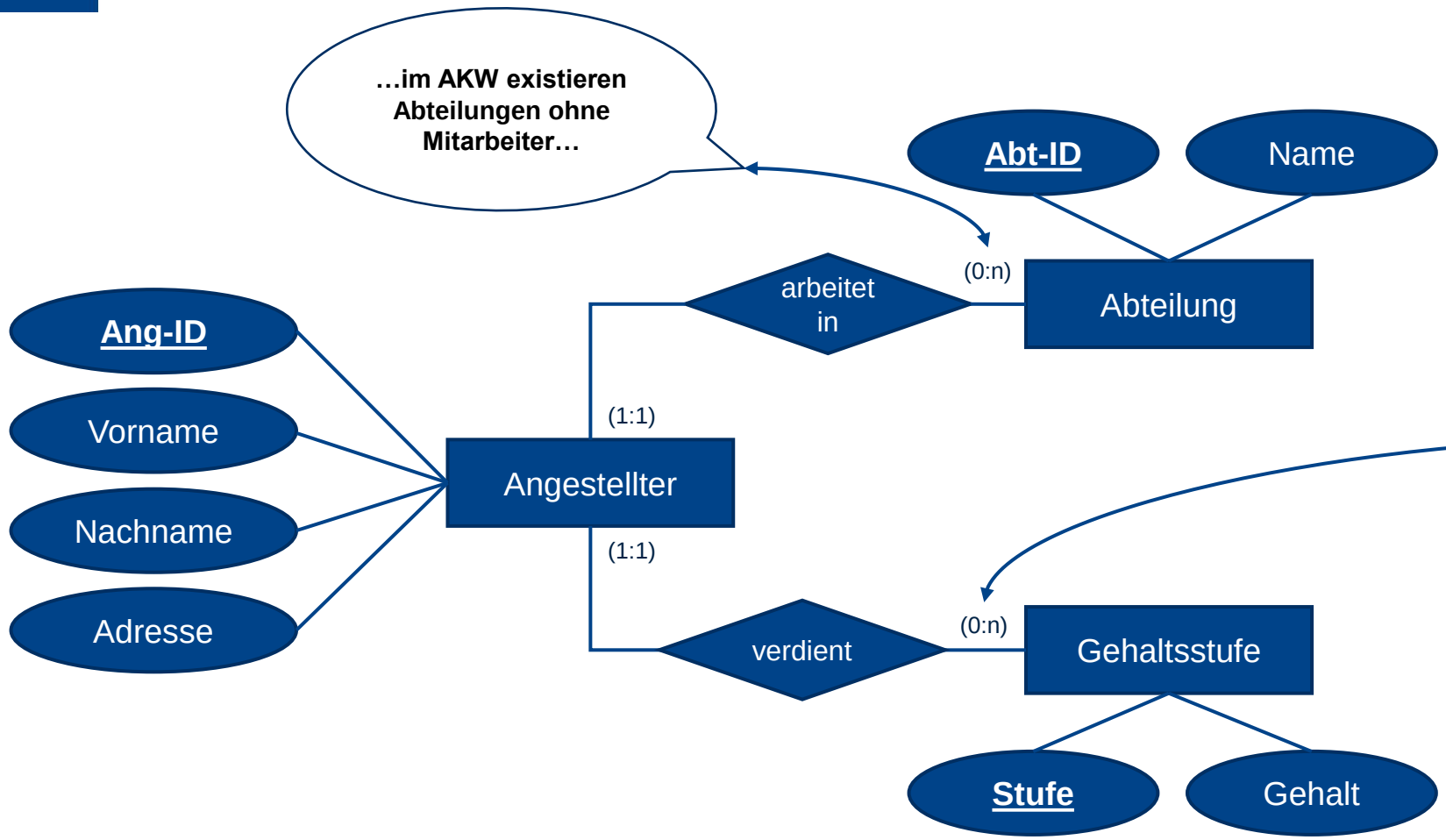


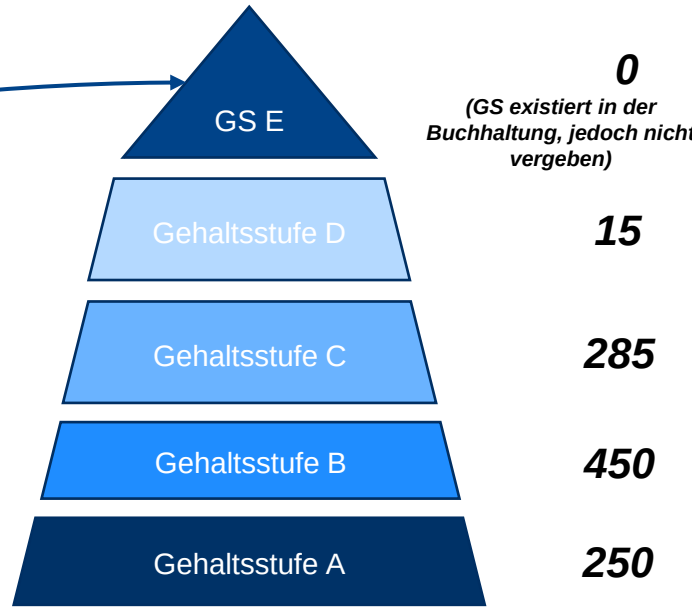
[ABB010]

ERM des Atomkraftwerks



ERM des Atomkraftwerks





Aufteilung der Mitarbeiter nach Gehaltsstufen (1000 Mitarbeiter)

ERM Abschluss

Ein vollständiges ER-Modell zeigt alle wichtigen Aspekte des Datenbanksystems auf und dient als **Grundlage für die eigentliche Implementierung.**

Nun gilt es diese Vorlage in das eigentliche System (z.B. das relationale Datenmodell) zu übernehmen.

Hierzu wird entsteht **aus jeder Entität eine eigene Tabelle** (Relation). Die Attribute der Entität werden als Spalten in diese Tabelle übernommen und über die Beziehungstypen werden die Tabellen über die Eigen- und Fremdschlüssel „verknüpft“.

[SCHI17], [ELMA09]

Relationales Datenmodell (1/3)

Das relationale Modell repräsentiert die Datenbank als Sammlung von Relationen. Informell ähnelt jede Relation einer Tabelle oder in gewissem Sinn einer „flachen“ Datei mit Datensätzen.

Stellt man sich eine Relation als Tabelle mit Werten vor, so repräsentiert jede Zeile der Tabelle eine Liste zusammenhängender Datenwerte: Ein sogenanntes ***Tupel*.**

[SCHI17]

Relationales Datenmodell (2/3)

- Wir haben Entitätstypen und Beziehungstypen als Konzept für die Modellierung von Daten der realen Welt vorgestellt. Beim relationalen Modell stellt **jede Zeile** einer Tabelle eine Tatsache dar, die normalerweise einer Entität oder Beziehung der realen Welt entspricht (Tupel). **Jede Spalte** entspricht einem Attribut dieser Entität.

Angestellte:

Attribute

ID	Vorname	Nachname	Adresse	Abteilung	Gehalts-stufe
1	Martina	Kaiser	Sanderweg 12, 97070 Würzburg	5	5
2	Christian	Brandt	Gotenstr. 7, 97070 Würzburg	5	4
3	Silke	Jung	Schlossstr. 4, 97070 Würzburg	5	4

Tupel

[SCHI17]

Relationales Datenmodell (3/3)

Verschiedene Tabellen können **über ihre Schlüsselattribute miteinander verknüpft** werden. Man spricht dann von **Primär- (unterstrichen)** bzw. **Fremdschlüsseln (#).** So stellt der Fremdschlüssel einer Relation den Primärschlüssel einer anderen Relation dar.

Beispielsweise werden Angestellte und Abteilungen in eigenen Relationen verwaltet. Da jeder Angestellte jedoch in einer bestimmten Abteilung arbeitet, muss dies auch in der Angestelltenrelation hinterlegt werden.

Angestellte:

ID	Vorname	Nachname	Adresse	#Abteilung	#Gehaltsstufe
1	Martina	Kaiser	Sanderweg 12, 97070 Würzburg	5	5
2	Christian	Brandt	Gotenstr. 7, 97070 Würzburg	5	4
3	Silke	Jung	Schlossstr. 4, 97070 Würzburg	5	4

Abteilungen:

ID	AbteilungsName
...	...
5	Sicherheit

[SCHI17]

Backend – Teil 1

Structured Query Language (SQL)

Nachdem nun eine Datenbank entworfen und implementiert wurde, gilt es jetzt mit dieser zu arbeiten, also Datensätze aus ihr abzufragen, neue hinzuzufügen oder zu entfernen. Für diese Operationen gibt es die sogenannte **Structured Query Language** (SQL).

Die SQL-Sprache gilt als einer der wichtigsten Gründe für den Erfolg relationaler Datenbanken in der kommerziellen Welt. Da sie sich als Standard für relationale Datenbanken durchgesetzt hat, müssen sich die Benutzer **um die Migration ihrer Datenbankanwendungen** von Datenbanksystemtypen, z.B. Netzwerk- oder hierarchische Systeme, auf relationale Systeme **weniger sorgen**.

Ein weiterer Vorteil eines solchen Standards: Die Benutzer können Anweisungen in einem Datenbankanwendungsprogramm schreiben, das auf Daten zugreift, die in zwei oder mehr relationalen DBMS gespeichert sind, **ohne die Datenbanksprache (SQL) ändern zu müssen**, falls beide relationale DBMS den SQL-Standard unterstützen.

[SCHI17]

SQL Relation erstellen (1/2)

Bevor jedoch Datensätze abgefragt werden können, ist es zuerst notwendig eine (oder mehrere) Tabellen zu erstellen. Dies geschieht für die Relation Angestellte wie folgt:

CREATE TABLE Angestellte

SQL Relation erstellen (2/2)

Die Tabelle enthält noch keinerlei Platz für Daten. Deshalb ist der nächste Schritt das Einfügen von Spalten, um die Daten abzulegen

```
(ID            INT            NOT NULL,
 Vorname       VARCHAR(15)    NOT NULL,
 Nachname      VARCHAR(20)    NOT NULL,
 Adresse       VARCHAR(30),
 Abteilung     INT            NOT NULL,
 Gehaltsstufe  INT                     );
```

Dabei wird zuerst ein Spaltenname gewählt (ID, Vorname, Nachname, Adresse, Abteilung, Gehaltsstufe) und anschließend der **Datentyp** für diese Tabelle festgelegt. Die Zahl in Klammern hinter dem Datentyp gibt die Anzahl der zu speichernden Zeichen an. Mit NOT NULL wird festgelegt, dass jedes Mal ein Wert übergeben werden muss, dieses Feld also nie leer sein darf.

[SCHI17]

SQL Datentypen

In SQL stehen verschiedene Datentypen für die Speicherung von Datensätzen zur Verfügung. Die Wichtigsten können der untenstehenden Tabelle entnommen werden.

Datentyp	Bezeichnung
INTEGER oder INT	Ganzzahl
DECIMAL(x,y)	X-stellig Zahl, mit y Nachkommastellen
CHAR(n)	Zeichenkette der festen Länge n
VARCHAR(n)	variable Zeichenkette mit bis zu n Zeichen
DATE	Datum (Jahr, Monat, Tag)
TIME	Uhrzeit (Stunde, Minute, Sekunde)
BLOB	Binary Large Object. große, binäre Objekte wie z. B. Bild- oder Audiodateien

[SCHI17]

SQL Relation erstellen

Weiterhin lassen sich noch die Primär- und Fremdschlüssel einer Relation festlegen. Hierzu werden diese im Create Befehl nach der Deklaration der Spalten wie folgt festgelegt:

PRIMARY KEY (ID),

FOREIGN KEY (Abteilung) REFERENCES Abteilung(ID)

Im ersten Befehl wird die ID des Angestellten als Primärschlüssel festgelegt. Dieser muss selbstverständlich eindeutig sein. Der zweite Befehl gibt die Spalte Abteilung unserer Tabelle als Fremdschlüssel an und verweist (REFERENCES) auf die Relation, welche die Werte enthält (Tabelle Abteilung, Spalte ID).

Natürlich muss die Relation Abteilung auch vorhanden sein, wenn wir auf sie verweisen. Deshalb werden die Tabellen Abteilungen und Gehaltsstufen analog erstellt.

[SCHI17]

Komplette Create-Befehle

Die Create-Befehle unserer Tabelle sehen fertig also wie folgt aus:

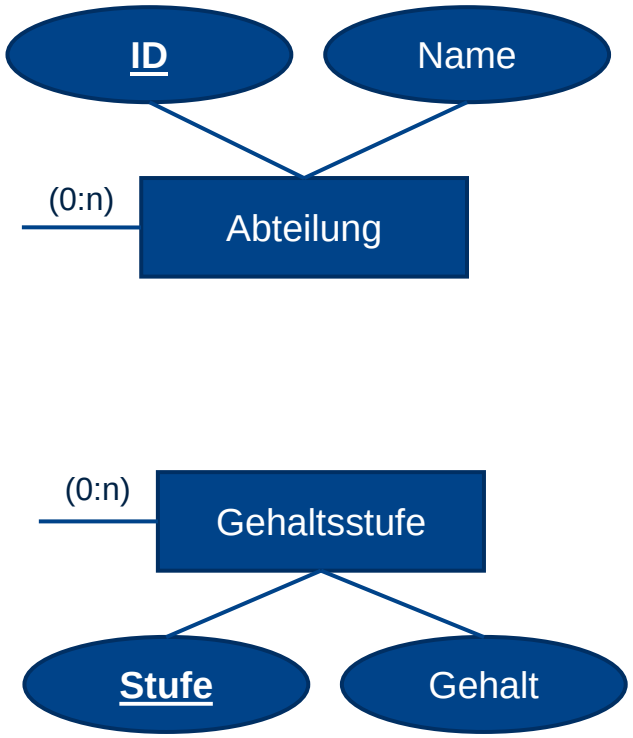
```
CREATE TABLE Angestellte (
ID              INT             NOT NULL,
Vorname         VARCHAR(15)     NOT NULL,
Nachname        VARCHAR(20)     NOT NULL,
Adresse         VARCHAR(30),
Abteilung       INT             NOT NULL,
Gehaltsstufe    INT,

PRIMARY KEY (ID),
FOREIGN KEY (Abteilung)      REFERENCES Abteilung(ID),
FOREIGN KEY (Gehaltsstufe)   REFERENCES Gehaltsstufen(Stufe) );
```

[SCHI17]

SQL Q1

- Wie lauten die SQL-Befehle zum Erstellen der beiden Tabellen „Abteilung“ und „Gehaltsstufe“?

- Als Hilfestellung ist rechts der Abschnitt aus dem ERM dargestellt:



SQL A1

```
CREATE TABLE Abteilungen (
ID              INT             NOT NULL,
Abteilung       VARCHAR(15)     NOT NULL,
PRIMARY KEY (ID) );
```

und

```
CREATE TABLE Gehaltsstufen (
Stufe           INT             NOT NULL,
Gehalt          VARCHAR(15)     NOT NULL,
PRIMARY KEY (Stufe) )
```

SQL Daten einfügen

Unsere Tabellen sind somit erstellt. Jetzt müssen sie noch mit Inhalten gefüllt werden. Dies geht über den Insert Befehl. In seiner einfachsten Form wird der Befehl benutzt, um ein einziges Tupel in eine Relation einzufügen. Die Werte sollten in der **gleichen Reihenfolge** aufgelistet werden, in der die entsprechenden Attribute im Create Befehl angegeben wurden. Wollen wir nun einen neuen Angestellten in unsere Relation aufnehmen, geht das wie folgt:

INSERT INTO Angestellte

VALUES (1, „Martina“, „Kaiser“, „Sanderweg 12, Würzburg“, 3, 5);

Führt zu:

ID	Vorname	Nachname	Adresse	#Abteilung	#Gehaltsstufe
1	Martina	Kaiser	Sanderweg 12, Würzburg	3	5

[SCHI17]

SQL Abfragen (1/5)

Da die Tabellen mit Datensätzen gefüllt wurden, lassen sich nun auch Abfrage (Select) Befehle auf ihnen ausführen. Die Grundform der Select-Anweisung, die auch als Abbildung oder SELECT-FROM-WHERE-Block bezeichnet wird, setzt sich aus den drei Klauseln SELECT, FROM und WHERE zusammen und hat folgendes Format:

SELECT **<Attributliste>**

FROM **<Tabellenliste>**

WHERE **<Bedingung>;**

[SCHI17]

SQL Abfragen (2/5)

Dabei ist die **Attributliste** eine Liste mit Attributnamen (Spalten), deren Werte mit der Ausführung der Anfrage erzeugt werden. Die **Tabellenliste** ist eine Liste mit den Namen der Relationen, auf die für die Ausführung der Anfrage zugegriffen wird und die **Bedingung** beschreibt einen bedingten Ausdruck, der die mit der Anfrage zu bearbeitenden Tupel qualifiziert.

Wollen wir nun beispielsweise die vollen Namen aller Angestellten wissen, die in Abteilung 5 arbeiten, sieht die Abfrage wie folgt aus:

```
SELECT    Vorname, Nachname
FROM      Angestellte
WHERE     Abteilung = 5;
```

[SCHI17]

SQL Abfragen (3/5)

Als Ergebnisrelation ergibt sich folgende Tabelle:

Vorname	Nachname
Martina	Kaiser
Christian	Brandt
Silke	Jung
Kathrin	Weber
Paul	Neumann

SQL Abfragen (4/5)

In der WHERE-Klausel der Abfrage lassen sich auch mehrere Bedingungen abfragen. Diese werden durch das Schlüsselwort **AND** miteinander verbunden.

SELECT Vorname, Nachname FROM Angestellte WHERE Abteilung = 5 AND Vorname = „Martina“;

Diese Abfrage gibt nur diejenigen Tupel zurück, die den Vornamen Martina haben **UND** in der Abteilung 5 arbeiten.

Vorname	Nachname
Martina	Kaiser

Analog hierzu werden mit dem Schlüsselwort **OR** diejenigen Tupel ausgegeben, welche entweder die eine Bedingung oder die andere Bedingung oder beide gleichzeitig erfüllen.

[SCHI17]

SQL Abfragen (5/5)

Weiterhin steht der Operator **NOT** zur Verfügung, wenn Ergebnisrelationen ein bestimmtes Kriterium nicht aufweisen sollen.

So gibt **SELECT Vorname, Nachname FROM Angestellte WHERE Gehaltsstufe NOT 3;** die Namen aller Angestellten aus, die nicht in Gehaltsstufe 3 gespeichert sind.

Vorname	Nachname
Martina	Kaiser
Christian	Brandt
Silke	Jung
Kathrin	Weber
Paul	Neumann
Michael	Schwarz
Lea	Winkler

[SCHI17]

SQL weitere Schlüsselwörter (1/5)

- Möchte man seine Ergebnisrelation nach einem bestimmten Attribut sortiert ausgegeben bekommen, so kann man das durch den Befehl **ORDER BY** <Attribut> **ASC/DESC** am Ende der Anfrage erreichen. DESC steht dabei für absteigend sortiert und ASC für aufsteigend sortiert.
- **SELECT Nachname FROM Angestellte WHERE Abteilung = 5 ORDER BY Nachname [ASC/DESC]**

ASC:

Nachname
Brandt
Jung
Kaiser
Neumann
Weber

DESC:

Nachname
Weber
Neumann
Kaiser
Jung
Brandt

[SCHI17]

SQL weitere Schlüsselwörter (2/5)

Will man Attributen eine andere Bezeichnung zuteilen als diejenige, die in der Relation verwendet wird, so kann man ihnen ein Alias mittels des Schlüsselwortes **AS** zuweisen.

SELECT Nachname, Adresse AS Anschrift FROM Angestellte WHERE Abteilung = 3;

Nachname	Anschrift
Kluge	Sanderweg 23, Würzburg
Hartmann	Schillerweg 1, Höchberg

[SCHI17]

SQL weitere Schlüsselwörter (3/5)

Will man eine Ergebnisrelation **ohne Duplikat-Tupel** erhalten, muss das Schlüsselwort **DISTINCT** in der SELECT-Klausel verwendet werden, was bedeutet, dass nur sich unterscheidende Tupel im Resultat verbleiben.

SELECT DISTINCT Gehaltsstufe FROM Angestellte;

Gehaltsstufe
5
4
3
1
2
NULL
6

…gibt alle unterschiedlichen Gehaltsstufen der Angestellten aus. Sollte eine Gehaltsstufe mehrmals vorkommen, wird diese nur einmal aufgeführt.

[SCHI17]

SQL weitere Schlüsselwörter (4/5)

Die Group-By-Klausel ermöglicht das Zusammenfassen von Tupeln nach bestimmten Eigenschaften.

SELECT Abteilung, COUNT(*)[1] AS Anzahl FROM Angestellte GROUP BY Abteilung;

Abteilung	Anzahl
1	1
2	1
3	2
4	2
5	5
6	2

…gibt eine Ergebnisrelation aus, welche die Abteilung und die jeweilige Anzahl ihrer Mitarbeiter enthält.

- [1] COUNT(*) Zählt alle Einträge einer Spalte

[SCHI17]

SQL weitere Schlüsselwörter (5/5)

Möchte man nun eine Ergebnisrelation mit den Abteilungen, in denen mindestens zwei Mitarbeiter arbeiten, erhalten, so ist eine weitere Restriktion nach der Gruppierung nötig. Dies kann mithilfe der Having-Klausel erreicht werden, die nur zusammen mit der Group-By-Klausel möglich ist:

SELECT Abteilung, COUNT(*) AS Anzahl FROM Angestellte GROUP BY Abteilung HAVING COUNT(*) > 1

Abteilung	Anzahl
3	2
4	2
5	5
6	2

…gibt eine Ergebnisrelation aus, welche die Abteilungen mit mindestens zwei Mitarbeitern und deren Anzahl enthält.

[SCHI17]

SQL vergleichs- und arithmetische Operatoren (1/8)

Als Erstes werden Vergleichsbedingungen auf bestimmte Teile einer Zeichenkette mit dem Vergleichsoperator **LIKE** beschrieben. Teilketten (Substrings) werden durch Verwendung zweier reservierter Zeichen spezifiziert: Das Prozentzeichen (%) ersetzt eine beliebige Anzahl von Zeichen und der Unterstrich (_) ein einzelnes Zeichen.

SELECT Vorname, Nachname FROM Angestellte WHERE Adresse LIKE „%Würzburg“;

Vorname	Nachname
Martina	Kaiser
Christian	Brandt
Silke	Jung
Kathrin	Weber
Paul	Neumann
Michael	Schwarz
Lea	Winkler
Felix	Kluge

…gibt die Vor- und Nachnamen aller Angestellten aus, die in Würzburg wohnen. Dabei ist es egal welche Zeichen vor dem gesuchten Wort „Würzburg“ stehen.

[SCHI17]

SQL vergleichs- und arithmetische Operatoren (2/8)

SELECT Vorname, Nachname FROM Angestellte WHERE Vorname LIKE „_ _ l_ _“;

…gibt die Vor- und Nachnamen aller Angestellten aus, deren Vorname genau 5 Zeichen lang ist und als dritten Buchstaben ein „l“ haben.

Vorname	Nachname
Silke	Jung
Felix	Kluge

[SCHI17]

SQL vergleichs- und arithmetische Operatoren (3/8)

Weiterhin erlaubt SQL die Verwendung von arithmetischen Operationen in Anfragen. Die arithmetischen Standardoperationen Addition (+), Subtraktion (-), Multiplikation (*) und Division (/) können auf numerische Werte oder Attribute mit numerischen Wertebereich angewandt werden.

SELECT Gehalt*1.1 FROM Gehaltsstufen WHERE Stufe = 3;

Gehalt * 1.1
66.000

…gibt das um 10% erhöhte Gehalt der Gehaltsstufe 3 aus.

[SCHI17]

SQL vergleichs- und arithmetische Operatoren (4/8)

Ein weiterer Vergleichsoperator ist **BETWEEN**, der die Tupel ausgibt, welche sich zwischen zwei Werten befinden.

SELECT Vorname, Gehaltsstufe FROM Angestellte WHERE (Gehaltsstufe BETWEEN 1 AND 3);

Vorname	Gehaltsstufe
Martina	3
Christian	3
Silke	3
Max	1
Paul	2
Jonas	3
Sarah	3

…gibt die Angestellten und deren Gehaltsstufe zwischen 1 und 3 liegt, wobei Randwerte mit eingeschlossen werden.

[SCHI17]

SQL vergleichs- und arithmetische Operatoren (5/8)

SQL unterstützt Anfragen, die prüfen können, ob ein Wert NULL ist, d.h. fehlt, undefiniert oder nicht verfügbar ist.

SELECT Vorname, Nachname FROM Angestellte WHERE Gehaltsstufe IS NULL;

Vorname	Nachname
Felix	Kluge

…gibt die Angestellten aus, für die keine Gehaltsstufe hinterlegt ist.

[SCHI17]

SQL vergleichs- und arithmetische Operatoren (6/8)

Da Gruppierung und Aggregation in vielen Datenbankanwendungen erforderlich sind, bietet SQL diese Konzepte als integrierte Funktionen. Zu diesen Funktionen zählen **COUNT**, **SUM**, **MAX**, **MIN** und **AVG**. Die Funktionen SUM, MAX, MIN und AVG werden auf eine Menge oder Multimenge numerischer Werte angewandt und geben die Summe, den maximalen Wert, den minimalen Wert bzw. den Durchschnitt (Mittel) dieser Werte zurück. Der COUNT-Befehl zählt die für einen Wert hinterlegten Einträge.

SELECT COUNT(*) AS Mitarbeiterzahl“ FROM Angestellte;

…zählt alle hinterlegten Einträge der Tabelle Angestellte (*) und gibt diese als numerischen Wert zurück.

Mitarbeiterzahl
13

[SCHI17]

SQL vergleichs- und arithmetische Operatoren (7/8)

Die SUM-Funktion summiert die Werte einer Tabellenspalte und gibt das Ergebnis aus.

SELECT SUM(Gehalt) FROM Angestellte;

Gehalt
60.000
40.000
50.000

Gehalt
150.000

…gibt eine Ergebnisrelation aus, welche die Summe der Gehälter aller Mitarbeiter enthält.

[SCHI17]

SQL vergleichs- und arithmetische Operatoren (8/8)

Die MIN/MAX-Funktion gibt das Minimum/Maximum der Werte einer Tabellenspalte aus.

SELECT MAX(Gehalt) FROM Angestellte;

Gehalt
60.000
40.000
50.000

Gehalt
60.000

…gibt eine Ergebnisrelation aus, welche das Maximum der Gehälter aller Mitarbeiter enthält.

[SCHI17]

- Analog auch für MIN(Gehalt) möglich

SQL Abfragen mehrerer Tabellen (1/5)

Das Konzept einer zusammengesetzten (joined) Tabelle wurde in SQL2 integriert, um in der FROM-Klausel eine Tabelle spezifizieren zu können, die durch einen **JOIN** erzeugt wird. Dieses Konstrukt ist übersichtlicher als die Definition aller SELECT- und JOIN-Bedingungen in der WHERE-Klausel. Man betrachte beispielsweise das unten abgebildete Beispiel.

```
SELECT    Angestellte.Nachname, Angestellte. Vorname, Abteilungen.AbteilungsName
FROM      Angestellte INNER JOIN Abteilungen
ON        Angestellte.Abteilung = Abteilungen.ID
WHERE     Abteilungen.AbteilungsName = „Sicherheit“;
```

Hinweis: Zur eindeutigen Verwendung der richtigen Spalte wird in der SQL-Syntax bei einem **Join** der entsprechende Tabellenname von der Ursprungstabelle vor den Spaltennamen platziert:

Tabellenname.Spaltenname

Nachname	Vorname	AbteilungsName
Kaiser	Martina	Sicherheit
Brandt	Christian	Sicherheit
Jung	Silke	Sicherheit
Weber	Kathrin	Sicherheit
Schwarz	Michael	Sicherheit

[SCHI17]

SQL Abfragen mehrerer Tabellen (2/5)

Bei dieser Anfrage werden in der FROM-Klausel die beiden Tabellen *Angestellte* und *Abteilungen* verknüpft und zwar an der Stelle (**ON**), bei welcher die *Abteilung* der Angestellten-Tabelle der *ID* der Abteilungen-Tabelle entspricht.

Als Kurzform für ON lässt sich das Schlüsselwort **USING** gefolgt von der Attributs-Bezeichnung verwenden, sofern beide Spalten dieselbe Bezeichnung haben.

Somit lässt sich nun in der SELECT-Klausel auch das Attribut *Name* abfragen, welches nur in der Angestellte-Tabelle hinterlegt ist.

[SCHI17]

SQL Abfragen mehrerer Tabellen (3/5)

Der Operator **JOIN** ist sehr umfangreich. Neben dem **INNER JOIN** gibt es noch unter anderem den **OUTER JOIN**, welcher in **FULL OUTER JOIN**, **LEFT JOIN** und **RIGHT JOIN** unterteilt werden kann. Als Beispiel hierfür kann folgende Situation genannt werden.

Die Aufsichtsbehörde des Atomkraftwerks möchte wissen, wie viele Mitarbeiter in allen Abteilungen arbeiten.
Mit einem **INNER JOIN** Befehl würden jedoch nur diejenigen Abteilungen aufgelistet, welche auch (gegenwärtig) über Mitarbeiter verfügen. Abteilungen ohne Mitarbeiter würden gar nicht erst angezeigt werden.

[SCHI17]

SQL Abfragen mehrerer Tabellen (4/5)

Hier kommt der **OUTER JOIN** ins Spiel.

```
SELECT Abteilungen.AbteilungsName, Count(Angestellte.Nachname) As Mitarbeiter
FROM Abteilungen LEFT JOIN Angestellte
ON Abteilungen.ID = Angestellte.Abteilung
GROUP BY Abteilungen.AbteilungsName;
```

AbteilungsName	Mitarbeiter
Buchhaltung	0
Chefetage	1
Public Relations	2
Forschung	2
Reaktor	2
Rechtsabteilung	0
Sicherheit	5
oberes Management	1

[SCHI17]

SQL Abfragen mehrerer Tabellen (5/5)

Im Gegensatz zum INNER JOIN, bei dem nur diejenigen Tupel angezeigt werden, welche einen JOIN Partner haben, werden beim (LEFT) JOIN **alle Einträge angezeigt**. Diejenigen, für die kein JOIN Partner existiert werden dementsprechend **mit 0 Mitarbeitern angezeigt**.

LEFT bezieht sich hierbei auf die erstgenannte („linke“) Abteilungs-Tabelle. **RIGHT JOIN** würde sich dementsprechend auf die zweitgenannte („rechte“) Angestellten-Tabelle beziehen.

[SCHI17]

Exkurs: SQL verschachtelte Abfragen

Einige Anfragen setzen voraus, dass existierende Werte zuerst aus der Datenbank gelesen und dann in einer Vergleichsbedingung benutzt werden. Solche Anfragen können als verschachtelte Anfragen formuliert werden; dies sind SELECT...FROM...WHERE-Anfragen innerhalb der WHERE-Klausel einer anderen Anfrage, die als äußere Anfrage bezeichnet wird.

```
SELECT    Vorname
FROM      Angestellte
WHERE     Abteilung    IN    (SELECT ID
                        FROM  Abteilungen
                        WHERE AbteilungsName = „Sicherheit“);
```

Die „hintere“ verschachtelte Abfrage wählt die ID aller Abteilungen aus, welche mit Sicherheit zu tun haben. Die „vordere“ Abfrage gibt die Vornamen aller Angestellten aus, deren Abteilung sich in (IN) der Menge der abgefragten Abteilungen befindet.

[SCHI17]

Backend – Teil 1

Von relationalen zu verteilten Datenbanken

Die beschriebenen **SQL-Datenbanken** sind grundsätzlich **relational**, d.h. alle Daten werden in Tabellen (Relationen) abgelegt.

Obwohl dieses Datenbankmodell aufgrund verschiedener Vorteile in der Praxis sehr weit verbreitet ist, hat sich in den letzten Jahren ein **Trend hin zu verteilten Datenbanken** und weg vom ausschließlich relationalen Modell entwickelt, um den Anforderungen massiv verteilter Anwendungen im Web oder Big Data Anwendungen gerecht zu werden.

Diese verteilten Datenbankmodelle werden unter dem Begriff **NoSQL** zusammengefasst, um die es im folgenden Abschnitt gehen soll.

Definition NoSQL

NoSQL steht für **„Not Only SQL“** und soll die Abkehr von reinen SQL-Datenbanken darstellen.

Bedingungen eines NoSQL-Datenbanksystems:

- Nicht relationales Datenmodell
- Architektur unterstützt verteilte Webanwendungen + horizontale Skalierung
- Unterliegt keinem fixen Datenbankschema
- Unterstützt große Datenbestände, flexible Strukturen und Echtzeitverarbeitung
- Unterstützt Datenreplikation
- Mehrbenutzerbetrieb mit differenzierten Konsistenzeinstellungen möglich
- Konsistenz nur verzögert gewährleistet, falls hohe Verfügbarkeit und Ausfalltoleranz angestrebt

→ Warum ist dies der Fall?

[MEI18]

CAP-Theorem (1/3)

Diese letzte Bedingung folgt aus dem **CAP-Theorem** nach Eric Brewer (2000).

Das Theorem besagt, dass in einem **massiv verteilten Rechnersystem nicht zur gleichen Zeit** alle der folgenden **drei Eigenschaften** erfüllt sein können:

C

A

P

1. **Consistency:** Bei jeder Veränderung von Daten in einer Datenbank erhalten alle lesenden Transaktionen den aktuellen Zustand, egal über welchen Knoten zugegriffen wird
2. **Availability:** Ununterbrochener Betrieb und akzeptable Antwortzeiten einer laufenden Anwendung
3. **Partition Tolerance:** Ein Ausfall eines Knotens oder einer Verbindung zwischen einzelnen Knoten hat keinen Einfluss auf das Gesamtsystem
   → einzelne Knoten lassen sich ohne Anwendungsunterbrechung hinzufügen/wegnehmen

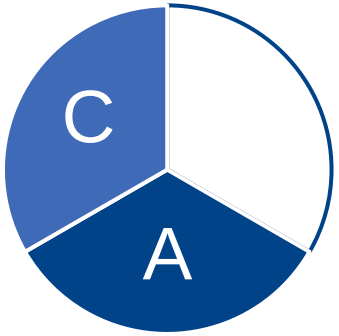
[MEI18]

CAP-Theorem (2/3)

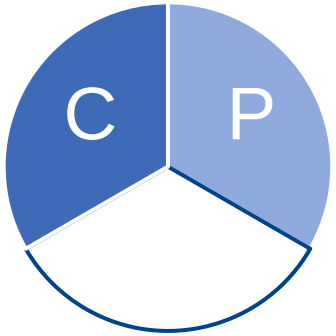
Einfacher formuliert:

In einem **massiv verteilten Datensystem** können immer **nur maximal zwei** der drei Eigenschaften (Konsistenz, Verfügbarkeit, Ausfalltoleranz) erfüllt sein.

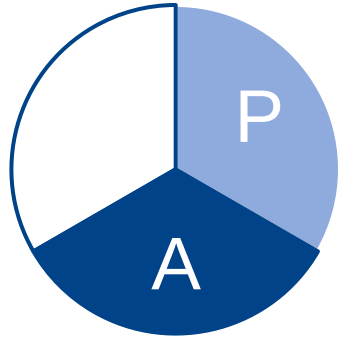
Mögliche Kombinationen sind daher:



Konsistenz + Verfügbarkeit



Konsistenz + Ausfalltoleranz



Verfügbarkeit + Ausfalltoleranz

[MEI18]

CAP-Theorem (3/3)

- **Beispiele:**

[ABB011]

An einem **Börsenplatz** sind Konsistenz und Verfügbarkeit am wichtigsten

C + A

C + P

Ein **Netz von Geldautomaten** einer Bank verlangt vor allem Konsistenz und Ausfalltoleranz, längere Antwortzeiten sind hinnehmbar

[ABB012]

[ABB013]

Der Internetdienst **Domain Name System (DNS)** braucht Ausfallsicherheit und Verfügbarkeit, um Website-Namen zu numerischen IP-Adressen aufzulösen

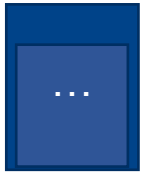
A + P

[MEI18]

Arten von NoSQL-Datenbanksystemen

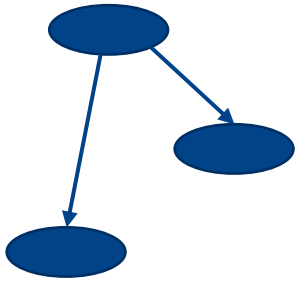
NoSQL-Datenbanken lassen sich in drei verschiedene Modellarten einordnen:

1. Key/Value und dokumentenbasierte Modelle



2. Spaltenorientierte Modelle

3. Graphenorientierte Datenbankmodelle



[MEI18]

1. Key/Value und dokumentenbasierte Modelle (1/3)

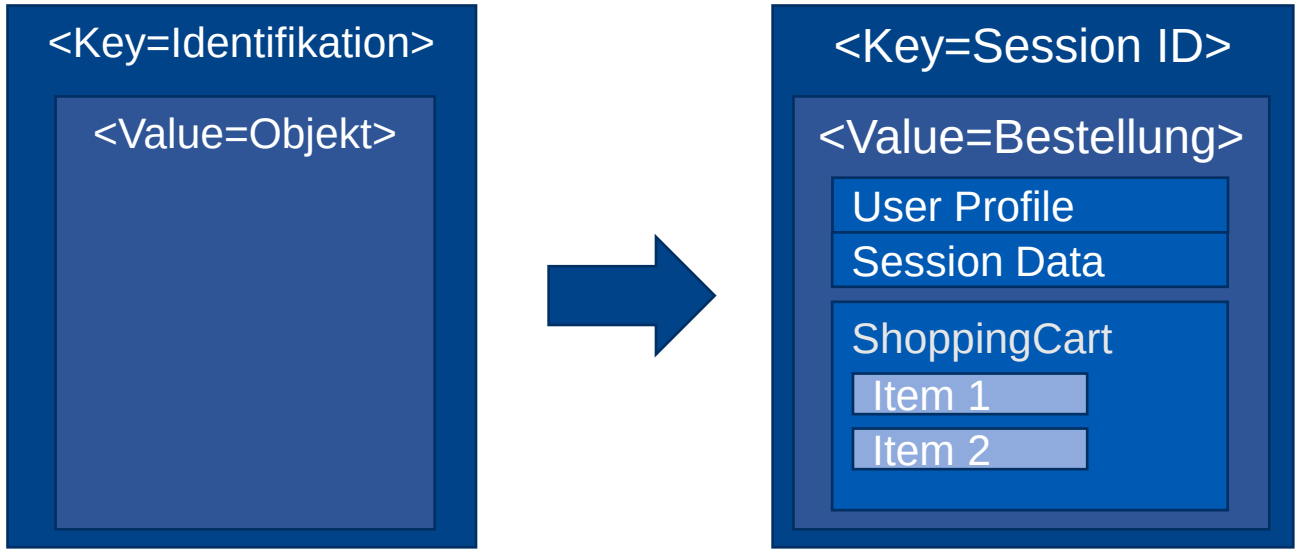
Bei Key/Value und dokumentenbasierten Modellen wird der **Aufbau der Datensätze** nicht schon wie bei relationalen Modellen beim Anlegen der Datenbank vorgegeben, sondern kann **noch bis kurz vor Ablegen der Daten modifiziert werden**.

Dies führt zu einer sehr hohen Flexibilität, da Dokumente von beliebiger Form, z.B. auch binär kodierte Objekte, in der Datenbank abgelegt werden können. Bekannte Vertreter dieser Modelle sind bspw. **„Lotus Notes“** von IBM, **„Berkeley-DB“** als Key/Value-Datenbank oder **„MongoDB“** als dokumentenbasierte Datenbank.

[MEI18]

1. Key/Value und dokumentenbasierte Modelle (2/3)

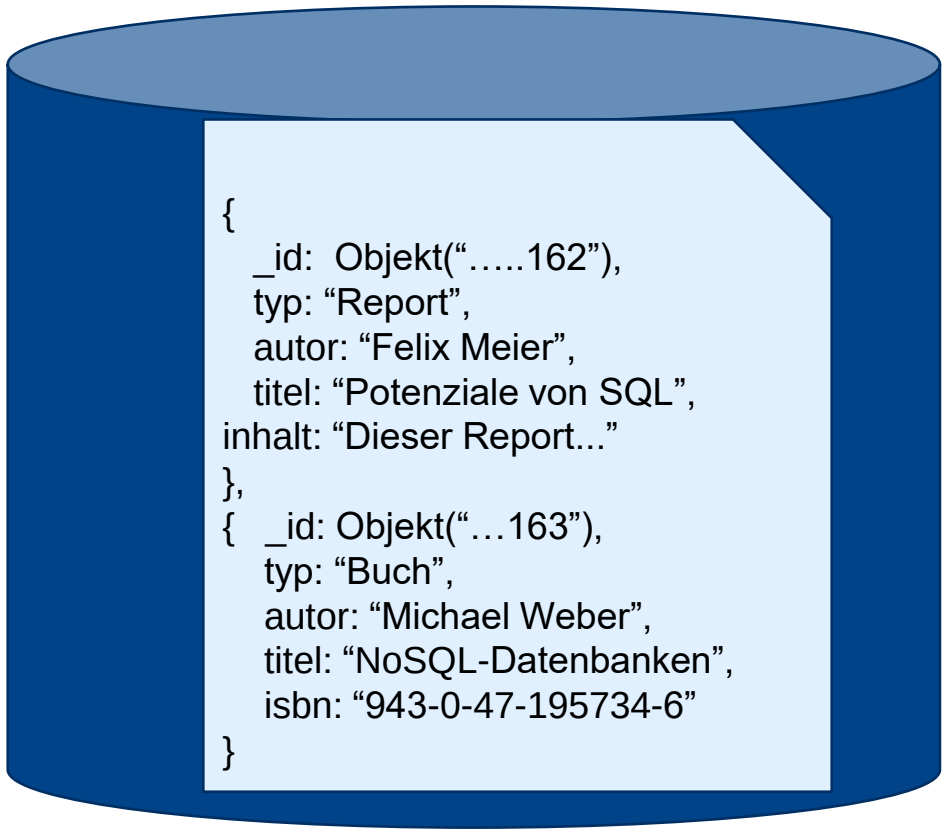
Nachfolgend ist das Grundmuster und ein Beispiel eines **Key/Value Onlineshops** abgebildet. Damit können die Daten für Bestellungen im Onlineshop flexibel und einfach abgelegt und verarbeitet werden.



[MEI18]

1. Key/Value und dokumentenbasierte Modelle (3/4)

Als **dokumentenbasierte Datenbank** soll der folgende „Document Store“ als Beispiel dienen.



→ Das erste Dokument ‚Report 162‘ könnte in einer relationalen Datenbank abgelegt werden, wenn eine Tabelle REPORT mit den entsprechenden Attributen definiert wird

→ Das Dokument ‚Buch 163‘ kann allerdings nicht aufgenommen werden, da es teilweise abweichende Merkmale (wie z. B. ISBN) besitzt

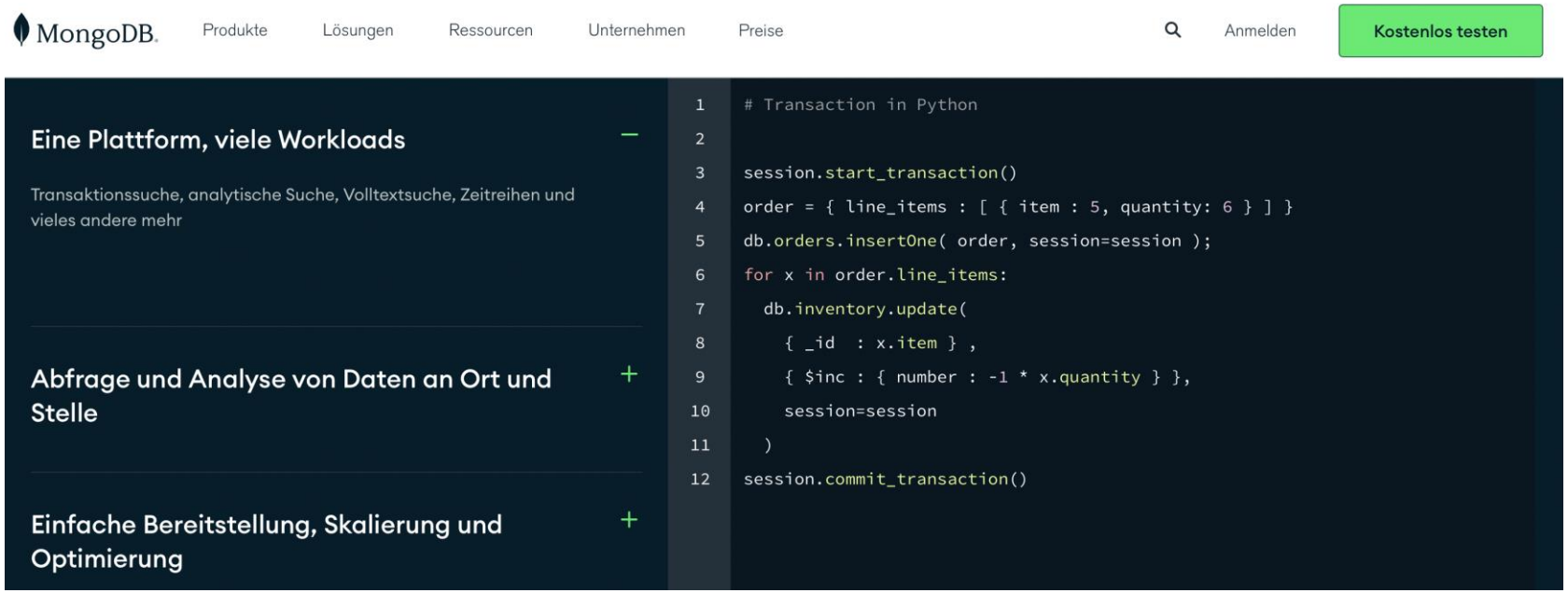
→ Im Document Store werden alle relevanten Daten unter einem eindeutigen Schlüssel (_id) zusammengefasst, sodass Dokumente als Einheit erfasst werden können

→ Ein Document Store verlangt **kein Schema** und ist **offen für Änderungen**

[MEI18]

1. Key/Value und dokumentenbasierte Modelle (4/4)

Das bekannteste dokumentenbasierte Datenbankmanagementsystem ist wie bereits erwähnt MongoDB, mit dem bspw. Volltextsuche, Abfragen und Datenanalyse, Skalierungen und Optimierungen möglich sind:



[ABB014]

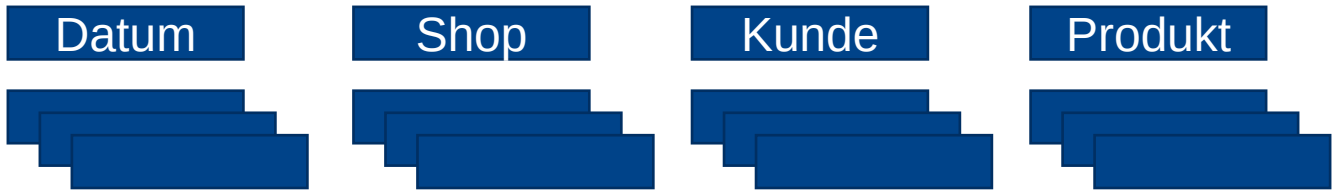
[MEI18]

2. Spaltenorientierte Modelle

Bei **spaltenorientierten Datenbanken**, den sogenannten **„Column Stores“** werden die Attribute und Merkmale einer Tabelle nicht zeilenweise, sondern **spaltenweise** abgespeichert.

Vorteile dieses Ansatzes sind:

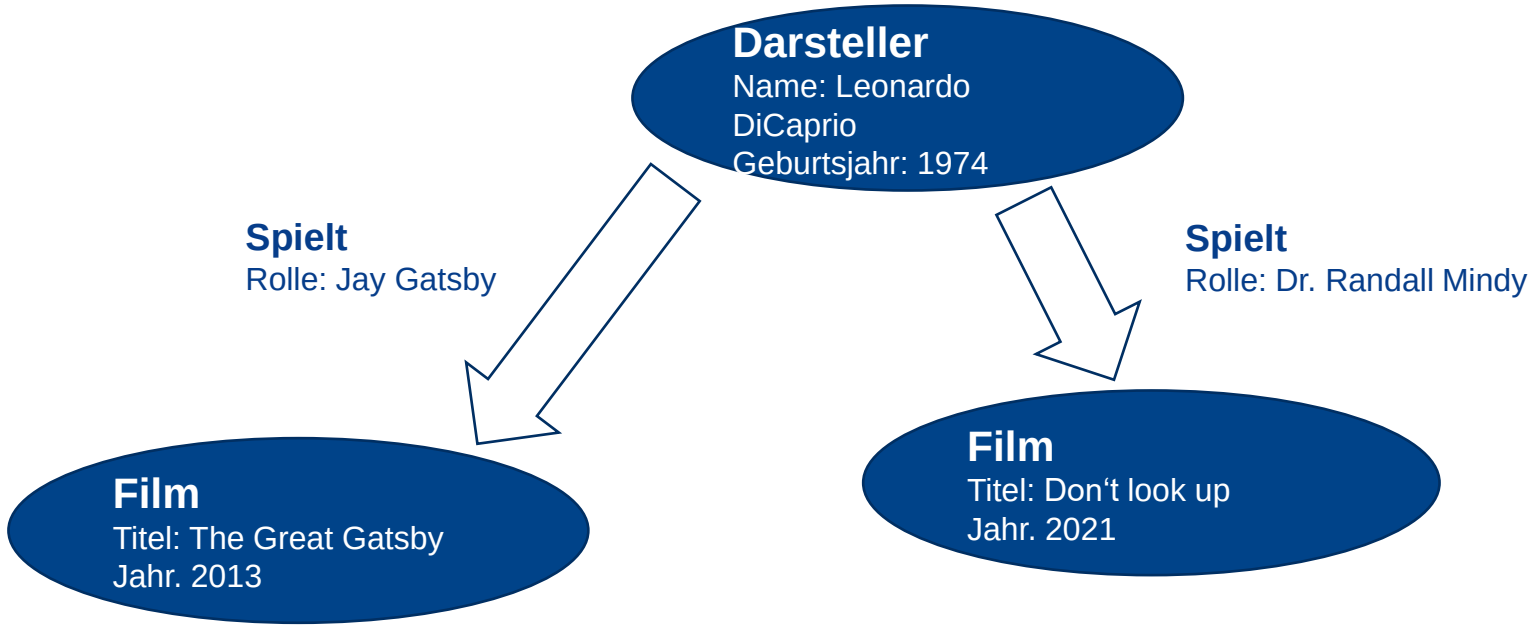
- die kostengünstige Speicherung großer Datenmengen
- die einfache und effiziente Skalierbarkeit
- die Möglichkeit von SQL-ähnlichen Abfragen
- die Kombinierbarkeit mit zeilenorientierten Ansätzen, z.B. bei elektronischen Einkaufssystemen



[MEI18]

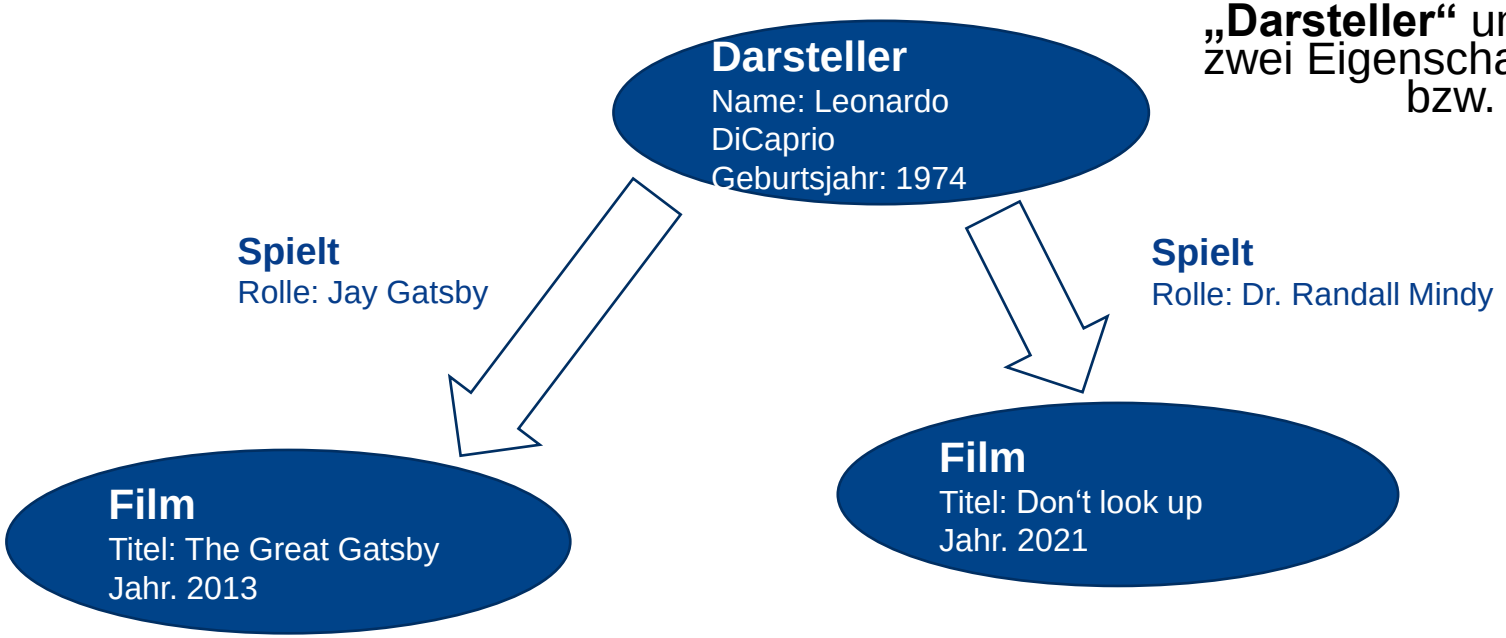
3. Graphenorientierte Modelle (1/3)

Graphenorientierte Datenbanken zeichnen sich dadurch aus, dass sie aus einer Menge von **„Knoten“** (Objekten) und einer Menge von **„Kanten“** (Beziehungen zwischen Objekten) bestehen. Die Knoten und Kanten können wie in folgendem Beispiel verschiedene Eigenschaften besitzen:



[MEI18]

3. Graphenorientierte Modelle (2/3)



Die beiden Knoten in diesem Beispiel sind **„Darsteller“** und **„Film“,** denen jeweils noch zwei Eigenschaften (**Name** und **Geburtsjahr** bzw. **Titel** und **Jahr**) zugeordnet sind. Als Kante, also als Beziehung zwischen den Knoten, wurde **„Spielt“** ausgewählt, um auszudrücken in welchen Filmen der Darsteller mitwirkt. Diese Kante verfügt wiederum über die Eigenschaft **„Rolle“.**

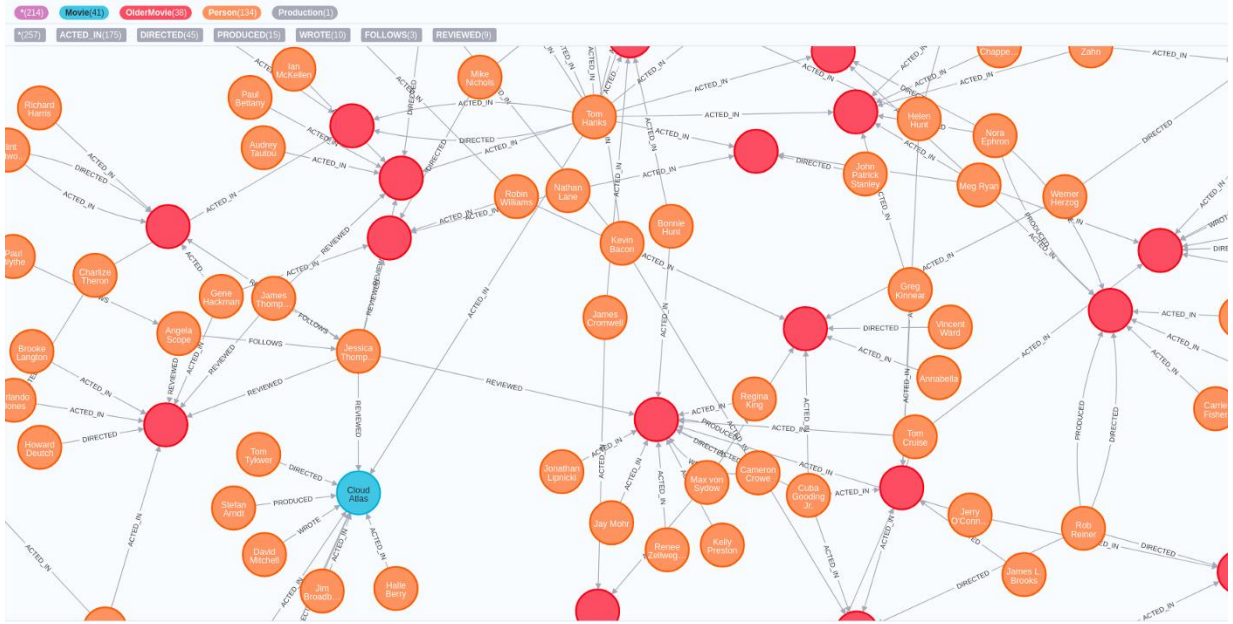
- In der Praxis werden Graphen-Datenbanken vor allem in Navigationsgeräten, aber auch in Geodatenbanken und sozialen Netzwerken eingesetzt. Ihr **Vorteil** liegt darin, dass die gespeicherten Daten auch **unstrukturiert** sein können und **viele semantische Zusammenhänge** ausgedrückt werden können.

[MEI18]

3. Graphenorientierte Modelle (3/3)

Mit vielen Ausprägungen können die Graphen allerdings schnell **unübersichtlich** werden, weshalb verschiedene Visualisierungstechniken zum Einsatz kommen (z.B. verschiedene Farben). Bekannt dafür sind vor allem die Open Source Datenbanken **OrientDB, GraphDB** und **Neo4J**.

Mit Neo4J kann ein Graph wie folgt aussehen:



[MEI18]

[ABB015]

NoSQL vs. SQL: Vorteile

Wie bereits teilweise an den gegebenen Beispielen zu erkennen, hat NoSQL mit seiner verteilten Datenhaltung gegenüber der zentralen Datenhaltung mit SQL einige **Vorteile**:

1. **Schnellere Verfügbarkeit** bei hoher Zahl an Zugriffen
2. **Lokaler Zugriff möglich**, ohne Zentralrechner zu belasten
3. **Ausfallsicherheit**, da kein zentraler Server als Schwachstelle
4. **Niedrige Antwortzeiten** durch optimale Verteilung paralleler Abfragen

[MEI18]

NoSQL vs. SQL: Nachteile

Demgegenüber stehen jedoch auch einige **Nachteile**, die eine NoSQL-Datenbank mit sich bringen kann:

1. **Hoher Kommunikationsaufwand,** um verteilte Daten über alle Systeme konstant zu halten
2. **Weniger verständliche und zugängliche Abfragesprache** als SQL
3. **Wenig Konsistenz** in Syntax und Grammatik der Abfragesprachen
4. **Schlechtere vertikale Skalierbarkeit** als SQL (aber besser horizontal skalierbar)

[MEI18]

Literatur

- **[ELMA09]:** Grundlagen von Datenbanksystemen; Elmasri, R./Navathe, S.; 3. Auflage, 2009; Pearson Verlag
- **[MEI18]:** Werkzeuge der digitalen Wirtschaft: Big Data, NoSQL & Co. Eine Einführung in relationale und nicht-relationale Datenbanken; Meier, A.; 2018; Springer Verlag
- **[SCHI17]:** Datenbanken und SQL - Eine praxisorientierte Einführung mit Anwendungen in Oracle, SQL Server und MySQL; Schicker, E.; 5. Auflage 2017; Springer Verlag

Abbildungsverzeichnis

- **[ABB001] :** https://openclipart.org/detail/150223/a-target-with-a-dart
- **[ABB002] :** https://openclipart.org/download/183938/icon-db.svg
- **[ABB003] :** https://openclipart.org/download/139525/server3d.svg
- **[ABB004] :** https://upload.wikimedia.org/wikipedia/en/thumb/6/62/MySQL.svg/978px-MySQL.svg.png
- **[ABB005] :** https://www.phpmyadmin.net/static/images/logo-og.png
- **[ABB006] :** https://openclipart.org/detail/181973/key
- **[ABB007] :** https://openclipart.org/detail/148219/question
- **[ABB008] :** https://openclipart.org/search/?query=power+plant#google_vignette
- **[ABB009] :** https://openclipart.org/image/800px/297371
- **[ABB010] :** https://openclipart.org/detail/281669/workflow
- **[ABB011] :** https://www.pexels.com/photo/stock-exchange-board-210607/
- **[ABB012] :** https://www.pexels.com/photo/man-standing-in-front-of-a-machine-8555256/
- **[ABB013] :** https://www.pexels.com/photo/black-laptop-computer-turned-on-showing-computer-codes-177598/
- **[ABB014] :** https://www.mongodb.com/
- **[ABB015] :** https://medium.com/analytics-vidhya/how-to-import-restore-neo4j-database-from-a-backup-compressed-file-tar-gz-f6e355a03b80

KONTAKT

Universität Würzburg –
Lehrstuhl für BWL und Wirtschaftsinformatik
Sanderring 2
97070 Würzburg