Spaß mit Datenbanken

1

Kurzer Rückblick

Surrogate Keys - Künstlicher Schlüssel

- Zusätzliches Schlüsselattribut, ohne Anwendung in der realen Welt
- In der Regel Datentyp: NUMBER
- Dient zur eindeutigen Identifizierung der Entität
- Ersetzen aus mehreren Attributen zusammengesetzten Primärschlüssel
- einfacherer Index-Aufbau
- schnellere Suche..

Ziel: Anomalien vermeiden

- **Änderungsanomalie:** Beim Ändern eines Wertes müssen viele andere Tupel ebenfalls geändert werden
- **Einfügeanomalie:** Beim Einfügen eines Tupels können bestimmte Werte nicht angegeben werden, da sie noch nicht bekannt sind. Wenn bspw. Schlüsselwerte fehlen, kann Tupel nicht einmal eingefügt werden
- **Löschanomalie:** Beim Löschen geht mehr Information verloren, als beabsichtigt.

Ziel: Anomalien vermeiden

Professoren

PersNr	Name	Rang	Raum	VorlNr	Titel	SWS
2125	Sokrates	C4	226	5041	Ethik	4
2125	Sokrates	C4	226	5049	Mäeutik	2
2125	Sokrates	C4	226	4052	Logik	4
...	...	...	...	...	...	...
2133	Popper	C3	52	5295	Der Wiener Kreis	2
2137	Kant	C4	7	4630	Die 3 Kritiken	4

- **Änderungsanomalie:** Bsp. Sokrates zieht um
- **Einfügeanomalie:** Bsp. Curie ist neu und liest noch keine Vorlesung
- **Löschanomalie:** Bsp. “Die 3 Kritiken” fällt weg.

Normalformen

- Legen **Eigenschaften** von Relationsschemata fest
- **Verbieten** bestimmte **Kombinationen** in Relationen
- Sollen Redundanzen und Anomalien vermeiden

Normalformen

Erste Normalform

- Erlaubt nur **atomare Attribute** in den Relationsschemata. D.h. Attributwerte sind Elemente von **Standard-Datentypen** wie *integer* oder *string*, aber keine Mengenwerte wie *array* oder *set*

Nicht in 1NF:

Eltern

Vater	Mutter	Kinder
Johann	Martha	{Else, Lucie}
Heinz	Martha	{Cleo}
...	...	...

in 1NF (= flache Relation)

Eltern

Vater	Mutter	Kinder
Johann	Martha	Else
Johann	Martha	Lucie
Heinz	Martha	Cleo

Normalformen

Zweite Normalform

- **Partielle Abhängigkeit** liegt vor, wenn ein Attribut funktional nur von einem Teil des Schlüssel abhängt.
- Verstoß gegen 2NF deutet darauf hin, dass in der Relation Informationen über mehr als ein Konzept modelliert werden.
- Zweite Normalform eliminiert **partielle Abhängigkeiten** bei Nichtschlüsselattributen.

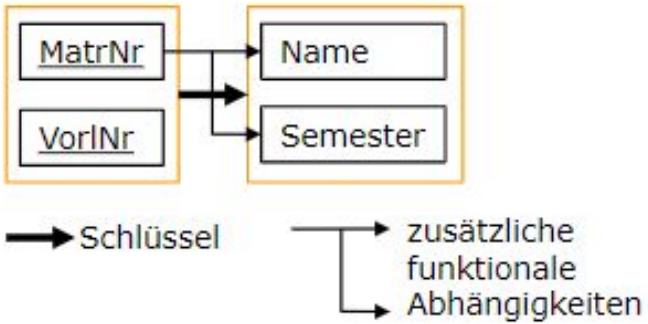
Normalformen

Zweite Normalform (Negativbeispiel)

StudentenBelegung

MatrNr	VorlNr	Name	Semester
26120	5001	Fichte	10
27550	5001	Schopenhauer	6
27550	4052	Schopenhauer	6
28106	5041	Carnap	3
28106	5052	Carnap	3
28106	5216	Carnap	3
28106	5259	Carnap	3
...	...		...

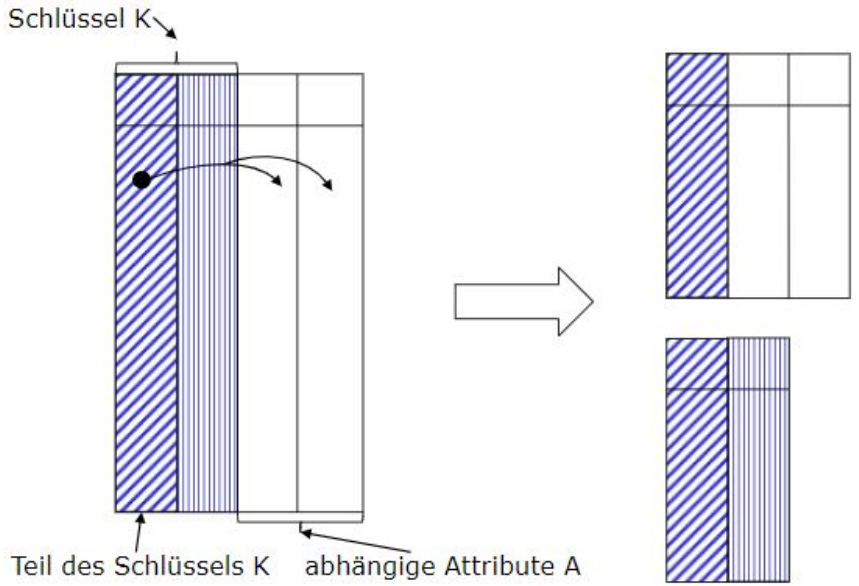
- {MatrNr} → {Name} und
- {MatrNr} → {Semster}



Normalformen

Zweite Normalform

- Eliminierung partieller Abhängigkeiten



Normalformen

Zweite Normalform

- Eliminierung partieller Abhängigkeiten

Relation in 2NF:

StudentenBelegung: {MatrNr, VorlNr, Name, Semester}

hören: {MatrNr, VorlNr}

Studenten: {MatrNr, Name, Semester}

Normalformen

Dritte Normalform

- Eliminiert (zusätzlich) transitive Abhängigkeiten

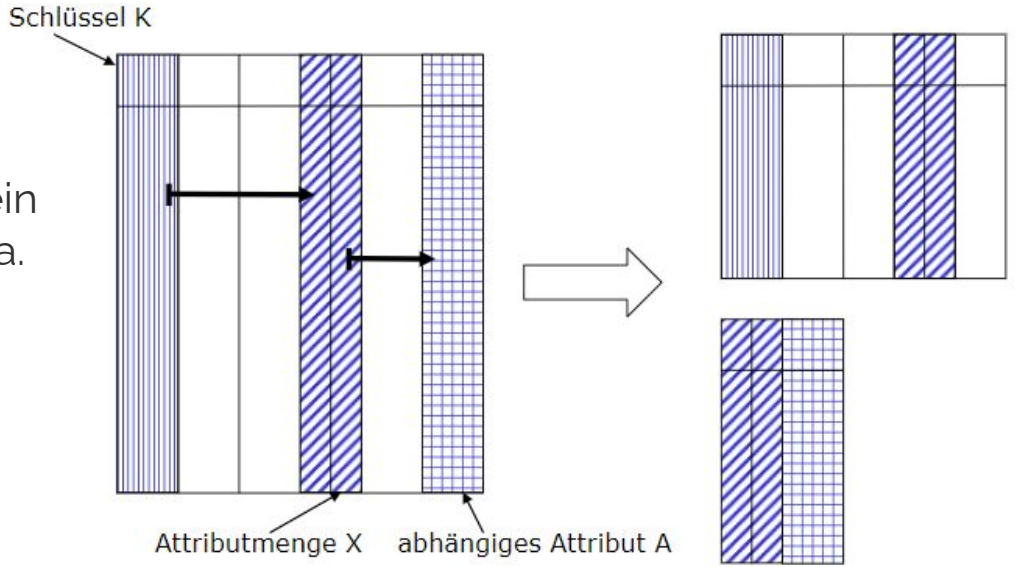
- Beispiel:
  - R = {PersNr, Name, Raum, Rang, PLZ, Ort, Straße}
  - {PersNr} → {PLZ} und {PLZ} → {Ort}

- Man beachte: 3.NF betrachtet **nur** Nichtschlüsselattribute als Endpunkt transitiver Abhängigkeiten.

Normalformen

Dritte Normalform

- Eliminierung transitiver Abhängigkeiten durch Verschiebung transitiv abhängiger Attribute in ein neues Relationenschema.



Normalformen

Dritte Normalform

- Eliminierung transitiver Abhängigkeiten

Relation in 3NF:

Professoren: {ProfNr, Name, Raum, Rang, PLZ, Ort, Straße}

Professoren: {ProfNr, Name, Raum, Rang, PLZ, Straße}

Orte: {PLZ, Ort}

Normalformen

- **1NF:** Ein Relationenschema ist in 1. Normalform, wenn dessen Wertebereiche atomar sind.
- **2NF:** Ein Relationenschema ist in 2. Normalform, wenn es in 1. Normalform ist und jedes Nichtschlüsselattribut voll funktional vom Primärschlüssel abhängig ist.
- **3NF:** Ein Relationenschema ist in 3. Normalform, wenn es sich in 2. Normalform befindet, und kein Nichtschlüsselattribut vom Primärschlüssel transitiv abhängig ist.

Schlüssel von 1:N-Beziehungen

Initialentwurf:

Professoren: {[PersNr:integer, Name:string, Rang:string, Raum:integer]}
Vorlesungen: {[VorlNr:integer, Titel:string, SWS:integer]}
lesen: {[PersNr:integer, VorlNr:integer]} (1:N)

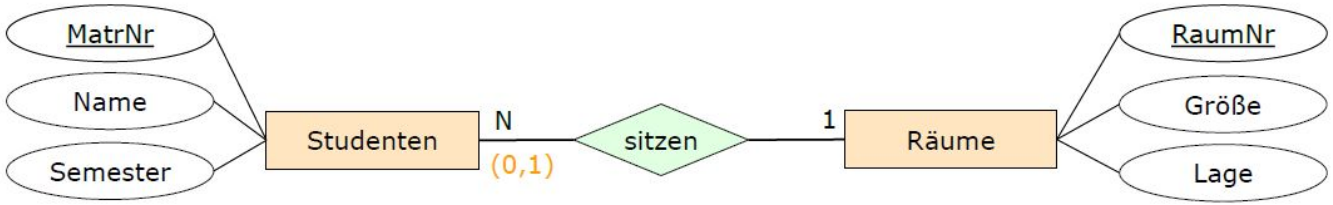
Verfeinerung durch Zusammenfassung von Relationen:

Professoren: {[PersNr:integer, Name:string, Rang:string, Raum:integer]}
Vorlesungen: {[VorlNr:integer, Titel:string, SWS:integer, gelesenVon:integer]}

- Relationen **mit gleichem Schlüssel** kann man zusammenfassen. Aber nur diese. Und keine anderen!

NULL-Werte vermeiden (1:N)

- **Beispiel:** Studierende, die als Assistenten arbeiten, bekommen einen Arbeitsraum. Es gibt 25.000 Studierende und 200 davon sind Assistenten



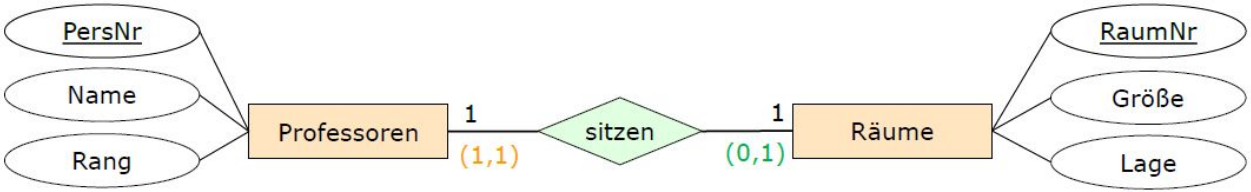
Logischer Entwurf:

Räume: {[RaumNr:integer, Größe:decimal, Lage:string]}

~~Studenten: {[MatrNr:integer, Name:string, Semester:integer, RaumNr:integer]}~~

- Hier **nicht** zusammenfassen! NULL-Werte vermeiden!

NULL-Werte vermeiden (1:1)



Logischer Entwurf:

Professoren: {[PersNr:integer, Name:string, Rang:string]}
Räume: {[RaumNr:integer, Größe:decimal, Lage:string]}
sitzen: {[PersNr:integer, RaumNr:integer]} oder
sitzen: {[PersNr:integer, RaumNr:integer]}

Professoren: {[PersNr:integer, Name:string, Rang:string, RaumNr:integer]}
Räume: {[RaumNr:integer, Größe:decimal, Lage:string]}

Professoren: {[PersNr:integer, Name:string, Rang:string]}
Räume: {[RaumNr:integer, Größe:decimal, Lage:string, PersNr:integer]}

2

Datendefinition
mit SQL

Structured Query Language

- **SQL** basiert auf relationaler Algebra
- Ist eine **deklarative** Sprache
- Ist eine **mengenorientierte** Sprache

Basis Datentypen

`BOOLEAN`	Wahrheitswert	true, false

`SMALLINT`
`INTEGER`
`BIGINT`
	ganze Zahl	1337

`DECIMAL(p,q)` `NUMERIC(p,q)`	Festkommazahl mit q Nachkommastellen	109.99

`FLOAT(p)`
`REAL`
`DOUBLE PRECISION`
	Fließkommazahl	1.5E-4

`CHAR(q)`
`VARCHAR(q)`
`LONG / CLOB`
	Zeichenkette mit fester Länge q
	variable Länge bis max. q
		“Proffessionalist”

Basis Datentypen

`BIT(q)` `BLOB`	binäre Zeichenkette (Bitfolge)	B ‘11011011’

`DATE`
`TIME`
`TIMESTAMP`
`INTERVAL`
	Datum
	Zeit
	Zeitstempel
	Zeitinterval
		DATE ‘2019-02-07’
		TIME ‘11:23:22’
		TIMESTAMP ‘2019-02-14 14:15:00’
		INTERVAL ‘48’ HOUR

`XML`	XML-Wert	<Title>CodersBay - SQL</Title>

`RAW`
`LONG RAW`
`BLOB`
`BFILE`
	Binärdaten (max 2.000 B)
	(max 2 GB)
	(max 4 GB)
	In externer Datei (max. 4GB)	

Definition von Attributen

- Attributname (ohne Umlaute/Sonderzeichen)
- Datentyp
- Default-Wert
- Constraints (Bedingungen)

Constraints

- Stellen Korrektheit der Daten sicher
- Werden vom Datenbanksystem überwacht und sichergestellt
- Nicht passende Operationen werden zurückgewiesen

`CONSTRAINT` mit `CHECK`-Bedingung → Semantische Integrität

`CONSTRAINT` mit `PRIMARY KEY` → Entity-Integrität

`CONSTRAINT` mit `FOREIGN KEY` und `REFERENCES` → Referentielle Integrität

Beispiele
Professoren haben Rang C2, C3, C4
Matrikelnummer der Studierenden ist eindeutig
Anmeldung zur Übung nur bei aufrechter Inskription

Constraints

- **NOT NULL** - NULL-Werte verboten
- **CHECK(cond-exp)** - Attributspezifische Bedingung
- **UNIQUE** - Jeder Wert darf nur einmal auftreten
- **PRIMARY KEY** - Angabe eines Primärschlüssel
- **FOREIGN KEY (attribute) REFERENCES TABLE (attribute)** - Angabe der referentiellen Integrität

Constraints - NOT NULL

- Schließt in Spalten den Wert “NULL” aus
- **NULL** steht für “Wert unbekannt” “Wert existiert nicht” etc.
- **NULL** kann in allen Spalten auftreten, außer bei **NOT NULL** und bei **Schlüsselattributen.**

BSP:

```
Name VARCHAR(30) NOT NULL
```

Constraint-namen

- Constraints können mit einem Namen versehen werden.
- Erleichtert Diagnose bei Fehlern.

BSP:

```
Semester INTEGER
    CONSTRAINT Semesterzahl CHECK(Semester BETWEEN 1 AND 6)
```

- Bei einer Verletzung wird “CHECK_Semesterzahl” ausgegeben.

Referentielle Integrität

- **Schlüssel** identifiziert ein Tupel einer Relation
- **Fremdschlüssel** verweist auf ein Tupel einer in Beziehung stehenden Relation
- Referentielle Integritätsbedingungen entstehen zwischen **Primär-** und **Fremdschlüssel**.

Beispiel

```
CREATE TABLE Professoren(
  PersNr        INTEGER PRIMARY KEY,
  Name          VARCHAR(30) NOT NULL,
  ...);

CREATE TABLE Vorlesungen(
  VorlNr        INTEGER PRIMARY KEY,
  Titel         VARCHAR(30),
  SWS           INTEGER,
  gelesenVon    INTEGER,
  CONSTRAINT    FK_gelesenVon
    FOREIGN KEY (gelesenVon)
    REFERENCES Professoren(PersNr)
);
```

Vatertabelle

Kindtabelle

Referentielle Integrität

- Operationen auf der **Kindtabelle** sind immer unkritisch.
- Operationen auf der **Vatertabelle** (löschen, ändern des PK) muss entsprechend geregelt werden:
  - **CASCADE** - Übernimmt die Änderung rekursiv für alle Tupel mit zugehörigem Fremdschlüssel.
  - **SET NULL** - Setzt den Fremdschlüssel ggf. auf **NULL**
  - **RESTRICT** - Verbiete operation, wenn Fremdschlüsselverweise vorhanden sind.
  - **SET DEFAULT** - Setzt den Fremdschlüssel auf den Default-Wert

ON DELETE oder **ON UPDATE**

Referentielle Integrität

Beispiel

```
CREATE TABLE Professoren(
  PersNr       INTEGER PRIMARY KEY,
  Name         VARCHAR(30) NOT NULL,
  ...);

CREATE TABLE Vorlesungen(
  VorlNr       INTEGER PRIMARY KEY,
  Titel        VARCHAR(30),
  SWS          INTEGER,
  gelesenVon   INTEGER,
  CONSTRAINT   FK_gelesenVon
    FOREIGN KEY (gelesenVon)
    REFERENCES Professoren(PersNr)
    ON DELETE SET NULL
);
```

Referentielle Aktion - wird auf `Vorlesungen`-Tabelle (=Kindtabelle) ausgeführt

Hinzufügen von Tabellen

- Einfügen mit `CREATE TABLE`

```
CREATE TABLE Professoren (Name VARCHAR(30), PRIMARY KEY (Name));

CREATE TABLE IF NOT EXISTS Professoren
    (Name VARCHAR(30) NOT NULL,
    Persnr INT(8), PRIMARY KEY(Persnr));

CREATE TABLE IF NOT EXISTS Vorlesungen
    (Name Varchar(30), SWS INT(4), GelesenVon INT(8),
    PRIMARY KEY (Name),
    FOREIGN KEY (GelesenVon) REFERENCES Professoren(Persnr));
```

Ändern der Tabellenstruktur

- Nachträgliche Änderungen mit `ALTER TABLE`

BSP:

```
ALTER TABLE Professoren ADD (Titel VARCHAR(30));
ALTER TABLE Professoren MODIFY (Titel VARCHAR(40));
ALTER TABLE Professoren DROP COLUMN Titel;
```

Ändern der Tabellenstruktur

- Nachträgliche Löschung mit `DROP TABLE`

BSP:

```
DROP TABLE Professoren;
DROP TABLE Professoren CASCADE;
```

- Mit CASCADE werden ‘abhängige’ Objekte ebenfalls gelöscht

Referentielle Integrität

Jede Bedingung **muss** bei Veränderungen am Datenbestand überprüft werden (Rechenzeit)

3

Datenanfrage mit SQL

Grundlagen von Anfragen

- **Anfrage:** Folge von Operationen
  - Berechnet **Ergebnisrelation** aus **Basisrelation**

- Benutzer formuliert “Was will ich haben?”, und nicht “Wie komme ich an das ran?”

- Ergebnis einer Anfrage ist **wieder eine Relation** und kann wieder als Eingabe für die nächste Anfrage verwendet werden

- **Syntaktisch korrekte Anfragen** können **nicht** zu Endlosschleifen oder unendlichen Ergebnisse führen

Grundlagen von Anfragen

Folgende Anfragen sind möglich

- **Selektion:** Auswahl von Zeilen/Tupel einer Relation
- **Projektion:** Auswahl einer Menge von Spalten einer Relation
- **Kartesisches Produkt:** Verknüpfung jeder Zeile zweier Relationen
- **Umbenennung** von **Attributen** oder **Relationen**
- **Vereinigung:** Liefert die Vereinigung zweier Relationen gleichen Schemas
- **Mengendifferenz:** Liefert Differenz zweier Relationen gleichen Schemas
- **Natürlicher Verbund:** Verknüpfung zweier Relationen über Spalte mit gleichen Attributwerten im gleichen Spaltennamen (doppelt vorkommende Spalten werden weggelassen)
- **Allg. Verbund:** Verknüpfung zweier Relationen, auch wenn sie keine gleichnamige Spalte haben. Verbund aufgrund logischer Bedingung)

Grundlagen von Anfragen

Keywords

- **SELECT:** Projektionsliste, Abfrage von Daten
- **FROM:** zu verarbeitende Relation
- **WHERE:** Selektions-, oder Verbundbedingungen
- **GROUP BY:** Gruppierung für Aggregatfunktionen
- **HAVING:** Selektionsbedingungen für Gruppen
- **ORDER BY:** Sortierung der Ergebnisrelation

```
SELECT attribute
FROM tabelle
WHERE bedingungen
```

Grundlagen von Anfragen

Beispiel

Geben Sie Personalnummer und Name aller C4-Professoren an:

Professoren

PersNr	Name	Rang	Raum
2125	Sokrates	C4	226
2126	Russel	C4	232
2127	Kopernikus	C3	310
2133	Popper	C3	52
2134	Augustinus	C3	309
2136	Curie	C4	36
2137	Kant	C4	7

```
SELECT PersNr, Name
FROM Professoren
WHERE Rang = 'C4';
```

$$\pi_{\text{PersNr, Name}}(\sigma_{\text{Rang='C4'}}(\text{Professoren}))$$

Ergebnis

PersNr	Name
2125	Sokrates
2126	Russel
2136	Curie
2137	Kant

Grundlagen von Anfragen

Beispiel

```
SELECT *
FROM Professoren;
```

Professoren

PersNr	Name	Rang	Raum
2125	Sokrates	C4	226
2126	Russel	C4	232
2127	Kopernikus	C3	310
2133	Popper	C3	52
2134	Augustinus	C3	309
2136	Curie	C4	36
2137	Kant	C4	7

Grundlagen von Anfragen

DISTINCT: Ergebnismenge ist frei von Duplikaten

Duplikatelimination

Geben Sie alle Rangbezeichnungen für Professoren ohne Duplikate aus.

```
SELECT DISTINCT Rang
FROM Professoren;
```

Ergebnis

Rang
C4
C3

Beispiel ohne DISTINCT:

Keine Duplikatelimination

```
SELECT ALL Rang
FROM Professoren;
```

Ergebnis

Rang
C4
C4
C3
C3
C3
C4
C4

Grundlagen von Anfragen

ALIASNAME:

- Benennt Spalte in Ergebnisrelation.
- Wird direkt nach dem Spaltennamen angegeben.
- Keyword: **AS**

Spaltenüberschrift

```
SELECT PersNr AS Personalnummer, Name Familienname
FROM Professoren;
```

Ergebnis

PersNr → | Personalnummer | Familienname | ← Name

Personalnummer	Familienname
2125	Sokrates
2126	Russel
2127	Kopernikus
2133	Popper
2134	Augustinus
2136	Curie
2137	Kant

Grundlagen von Anfragen

Sortierung:

- Klausel steht am Ende der Anfrage.
- Keyword: **ORDER-BY**

Beispiel

```
SELECT PersNr, Name, Rang
FROM Professoren
ORDER BY Rang DESC, Name ASC;
```

Ergebnis

PersNr	Name	Rang
2136	Curie	C4
2137	Kant	C4
2126	Russel	C4
2125	Sokrates	C4
2134	Augustinus	C3
2127	Kopernikus	C3
2133	Popper	C3