Datenbanken

- Under Construction
- Grundlagen
- ER-Modell
- EER-Modell
- Das relationale Modell
- Relationale Algebra
- Vom EER-Modell zum relativen Modell
- SQL - DML
- SQL - DDL
- Normalisierung
- DB-Transaktionen
- Protokolle zur Transaktionsverarbeitung
- Recovery
- Data Warehouse
- ETL - Extraktion, Transformation, Laden
- DWS - Multidimensionales Datenmodell
- Begriffsammlung

Under Construction

Grundlagen

Eine **Datenbank** ist eine Sammlung von Daten, die eine Bedeutung zugeordnet bekommen. Sie stellt einen Weltausschnitt "Miniwelt" dar. Die Daten in der DB hängen logisch zusammen.

Ein **DBMS** dient dem Entwurf, der Implementierung und dem Betrieb einer DB.

Datenmodelle definieren DB-Strukturen mit Datentypen, Beziehungen und Einschränkungen.

Modellierungsebenen

konzeptuell (Auftraggeberrecht)
logisch (Implementierungsansätze)
physisch (Datenspeicherung)

Ein **Schema** (Intension) bezeichnet die Beschreibung der kompletten Struktur einer DB.

Die **Instanz** (elem. Extension) beschreibt einen einzelnen, aus Datenelementen bestehenden Datensatz (eine Zeile).

Der **DB-Zustand** zeigt die Gesamtheit der gespeicherten Daten.

Datenabhängigkeit

logisch
→ konzeptuelles Schema ändern, ohne externe Schemata ändern zu müssen

physisch
→ internes Schema ändern, ohne logisches oder externes Schema zu ändern

ER-Modell

Mit der Modellierung soll ein Konzept dargestellt werden, das etwas besser verständlich macht und die Kommunikation erleichtert. Das **ER-Modell** beschreibt die Struktur, Eigenschaften, Beziehungen und das Verhalten einer DB.

Symbole

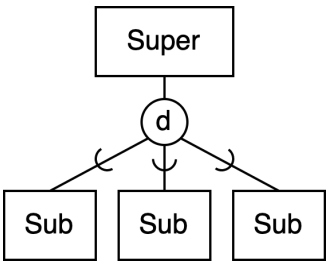
EER-Modell

Das **EER-Modell** erweitert das ER-Modell um Spezialisierung und Generalisierung.

Spezialisierung

Mit Spezialisierung können Subentitäten zu einer Superentität definiert werden. Diese können zusätzliche Attribute oder Beziehungen haben.

disjunkt

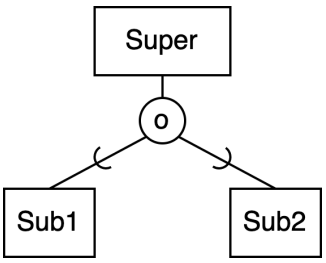


→ exklusive Beziehung

→ Sub1 kann auch gleichzeitig Sub2 sein

---

overlapped



→ Sub1 kann auch Sub2 sein

→ muss aber nicht

Generalisierung

Die Generalisierung ist eine Umkehrung der Spezialisierung, es wird eine Superentität für mögliche Subentitäten geschaffen.

Vorgehensmodelle

Top-Down → immer weiter verfeinern
Bottom-Up → Modellierung kleiner Teilschemata
Inside-Out → zentral nach außen
Mixed

Business Rules

Da ein EER-Schemata oft nicht ausreichend für die gesamte Darstellung einer DB ist, gibt es die Business Rules, die weitere Anforderungen umsetzen. Sie können beschreibend, einschränkend (Constraint) oder abgeleitet sein.

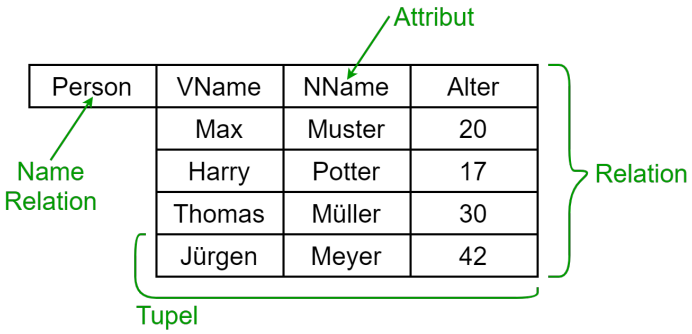
Qualitätskriterien

- Korrektheit
- Vollständigkeit
- Minimalität
- Lesbarkeit
- Verständlichkeit

Das relationale Modell

Das relationale Modell repräsentiert eine Datenbank als Sammlung von Relationen (Tabellen). Zwischen den Relationen existieren wertebasierte Beziehungen.

Aufbau



Wertebereich - Datentypen

- numerische (Integer, float, ...)
- Zeichenketten
- spezielle Datentypen
- benutzerdefinierte Datentypen

Formale Betrachtung

Relationsschema mit Attributen R (A_1, A_2 ... A_n)
Grad der Relation n

R	**A_1**	**A_2**	**...**	**A_n**
t_1	v_1	v_2	...	v_n
t_2	v_1	v_2	...	v_n
...	...	...	...	...
t_n	v_1	v_2	...	v_n

Die Anzahl der möglichen Werte bzw. Kardinalität eines Wertebereichs ergibt multipliziert die Menge möglicher Tupel.

| dom(A_1) | * | dom(A_2) ...

Tupel an sich können umsortiert werden. Einzelne Werte können auch mit Null belegt sein (v_i).

Schlüssel

Ein eindeutiges Attribut nennt sich **Superschlüssel**. Wenn r(R) nicht mehr eindeutig ist, falls ein Superschlüsselelement v aus K entfernt wird, wird K als **Schlüssel** bezeichnet.
Bei mehreren Schlüsseln in einer Relation muss ein **Primärschlüssel** ausgewählt werden.
Beziehungen zwischen Relationen werden mit einem **Fremdschlüssel** beschrieben. Dieser referenziert auf Attribute der anderen Relation. Dies bezeichnet man als **referenzielle Integrität**.
Durch diese Referenzen entstehen Abhängigkeiten zwischen den beiden Relationen. Wenn die Bedingungen der referenziellen Integrität verletzt werden können Konflikte entstehen.

Relationale Algebra

$\sigma_{Bedingung}(R)$

$\pi_{A1,\,A2}\,(R)$

→ Selektion einer Teilmenge mit Bedingung

→ Auswahl bestimmter Attribute, Projektion, Duplikat-Eliminierung

Bedingungen können über >, ≥, =, ≤, < realisiert werden und mit **AND** und **OR** verknüpft werden.

$\rho_{S(B1\ldots Bn)}(R)$ → Rename R→S und (A_1 ... A_n)→(B_1 ... B_n)

Mengentheoretische Operatoren

∪ → UNION → Vereinigung beinhaltet alle Tupel

∩ → INTERSECT → Schnitt beinhaltet nur Tupel die in beiden Relationen sind

− → DIFFERENCE → Alle Tupel die in R_1 aber nicht in R_2 sind

Mit dem kartesischen Produkt **×** werden alle Tupel in allen möglichen Kombinationen kombiniert.

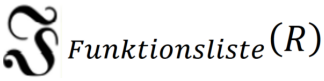
JOIN-Operationen

Beim Join werden zwei Relationen über eine Bedingung miteinander verknüpft.

Theta-Join	→ Bedingung über >, ≥, =, ≤, <, ≠	**θ**
Equi-Join	→ Bedingung über =	**=**
Natural-Join	→ Verbindung mit mehreren gleichen Attributen	*****
Left-Outer-Join	→ Jedes Tupel aus R_1 wird ausgegeben, rechts wird mit NULL aufgefüllt	⟕
Right-Outer-Join	→ Links wird mit NULL aufgefüllt	⟖

Full-Outer-Join	→ Es wird rechts und links mit NULL aufgefüllt	

Aggregatsfunktionen



→ so können mit `count()`, `avg()`, `max()` oder `min()` Werte geliefert werden.

Durch die Angabe eines Attributes können die Tupel auch gruppiert werden.

Vom EER-Modell zum relativen Modell

Das Ziel der Umwandlung ist ein äquivalentes, relationales Schema.

0. Restrukturierung

Die EER-Konstrukte müssen in ER-Konstrukte umgewandelt werden.

1. Erstellen von 1:1 Beziehungen, Entitäten erhalten gleichen Primärschlüssel
2. Superentität löschen, Subentitäten mit Attributen der Superentität füllen
3. Superentität mit Attributen der Subentitäten und einem Typ-Attribut bestücken
4. Superentität mit Attributen der Subentitäten und mehreren boolschen Typ-Attributen

1. Starke Entitäten

Pro Entität ein Relationenschema mit allen einfachen Attributen.

2. Schwache Entitäten

Pro Entität ein Relationenschema mit allen einfachen Attributen. Primärschlüssel ergibt sich aus partiellem Schlüssel und Fremdschlüssel.

3. 1:1-Beziehungen

Ein (total teilnehmendes) Relationenschema um den Fremdschlüssel und die Beziehungsattribute erweitern.

4. 1:N-Beziehungen

Das N-seitige Schema wird um den Fremdschlüssel und die Attribute der Beziehung erweitert.

5. N:M-Beziehungen

Ein neues Schema wird erzeugt mit den Attributen der Beziehung, der Primärschlüssel setzt sich aus beiden Fremdschlüsseln zusammen.

6. Mehrwertige Attribute

Für jedes mehrwertige Attribut wird ein Relationenschema mit dem Fremdschlüssel der Entität als Primärschlüssel erzeugt.

7. N-äre-Beziehungen

Es wird ein neues Relationenschema erzeugt mit Attributen der Beziehung. Der Primärschlüssel setzt sich aus Fremdschlüsseln der teilnehmenden Relationen zusammen.

SQL - DML

SELECT

→ Liefert Attribute einer oder mehrerer Relationen, bei Bedarf kann eine Bedingung angefügt werden.

```
SELECT Attribute FROM Relationen WHERE Bedingung
```

Um alle Attribute zu erhalten kann * verwendet werden.
Die Bedingung kann über >, ≥, =, ≤, <, ≠ realisiert werden.

DISTINCT

→ wird an Select angehängt für Duplikateliminierung.

JOIN

→ Der Join kann über zwei Möglichkeiten realisiert werden.

```
SELECT Attribut FROM Relation1, Relation2
```
→ kartesisches Produkt

oder

```
SELECT Attribut FROM R1 JOIN R2 ON Bedingung
```
→ Join

> → Auch Mehrfach-Joins sind möglich.

Mengenoperationen

`UNION` → Vereinigung
`INTERSECT` → Schnitt
`EXCEPT` → Differenz

> → Duplikate können durch Anhängen eines `ALL` erhalten bleiben.

Zeichenvergleiche

`%` → beliebige Zeichenkette im Suchtext
`_`

→ ein Zeichen im Suchtext

Operatoren

Zahlen: ***, +, -, /**
Zeichenketten: **||** (Verbinden)
Vergleich: `BETWEEN Wert AND wert`

Subselect

Es kann auch ein Subselect ausgeführt werden.

```
SELECT Attribut
FROM Relation
WHERE Attribut
   IN
   Subselect
   OR
   Subselect
```

`ANY` → beliebiger Wert
`ALL` → alle Werte
`EXISTS` → NULL-Prüfung
`UNIQUE` → liefert nur eindeutige Attributwerte, keine Duplikate
`AND+OR` → Verknüpfungen für Mengenvergleich

Aggregatsfunktionen

`count()` → Anzahl der Tupel
`sum()` → Summe
`min()` → Minimum
`max()` → Maximum
`avg()` → Durchschnitt

Gruppierung

`Group by()` → Gleiche Attribute werden zu einem zusammengefasst

Ordnen

`Order by Attribute DESC` → absteigend
`Order by Attribute ASC` → aufsteigend

INSERT

→ Einfügen von Werten in Tabellen

```
INSERT INTO Relation VALUES Attributsliste
```

> → Dabei kann auch die Attributsliste angegeben werden, wenn noch nicht klar ist, wie alle Attribute gefüllt werden sollen.

DELETE

Alle auf die Anweisung passenden Tupel werden gelöscht.

```
DELETE FROM Relation WHERE Bedingung
```

UPDATE

Dient zur Aktualisierung von Tupeln.

```
UPDATE Relation SET Attribut = 'Wert'  WHERE Bedingung
```

SQL - DDL

`CREATE SCHEMA` → erstellt eine Datenbank
`CREATE TABLE` → erstellt eine Datenbank

Datentypen

INTEGER
FLOAT, REAL, DOUBLE
Char(n) **→ fest**
VARCHAR(n) **→ variabel**

Date('YY-MM-DD')

Benutzerdefiniert:
CREATE DOMAIN 'Name' AS 'Wertebereich'

`Constraints`

→ Constraints sollen mögliche Eingaben für Attribute einschränken oder spezielle Anforderungen umsetzen.

`NOT NULL`
`UNIQUE`
`PRIMARY KEY`
`REFERENCES`
`FOREIGN KEY`

> `CASCADE`
> `SET NULL`
> `SET DEFAULT`
> `NO ACTION`
>
> → Was passiert, wenn Constraints verletzt werden?

DROP

→ ist das Schlüsselwort zum Löschen von Datenbank-Konstrukten

`SCHEMA, TABLE, COLUMN, CONSTRAINT`

ALTER

→ Steht für mögliche Änderungen von Relationen, z.B. hinzufügen und entfernen von Attributen

und Constraints

Normalisierung

Normalformen dienen als Entscheidungsgrundlage für die Verwendung von Relationenschemata. Sie ermöglichen einen objektiven Vergleich relationaler DB-Schemata.

Funktionale Abhängigkeit

Bestimmen einige Attribute eindeutig die Werte anderer Attribute, so sind die anderen Attribute funktional abhängig. Daher sind Attribute immer abhängig von Schlüsselattributen. Einige Abhängigkeiten können sich auch transitiv aus anderen ergeben.

Anomalien

Eine Datenbank sollte so entworfen sein, dass keine Einfüge-, Lösch oder Update-Anomalien auftreten.

Einfügen sollte möglich sein, ohne viele NULL-Werte einfügen zu müssen, da Daten fehlen.
Löschen sollte möglich sein, ohne zu viele Informationen zu löschen.
Update sollte ohne großen Aufwand (z.B. Redundanzen) möglich sein.

> Durch Normalisierung soll möglichst Redundanzfreiheit und Eliminierung der Anomalien geschaffen werden.

Normalformen

1. Normalform

Alle Attribute enthalten nur atomare Werte.
Wenn nicht: Ein neues Relationenschema für mehrwertige Attribute erstellen.

Probleme
- unnötige Redundanzen
- Anomalien

2. Normalform

Zusätzlich zu 1. NF sind keine Attribute nur von einem Teil des Primärschlüssels abhängig.
Wenn doch: Neues Relationenschema für Teilabhängigkeiten

Probleme
- Redundanzen

- Anomalien

3. Normalform

Zusätzlich zur 2. NF hängt kein Attribut transitiv vom Primärschlüssel ab.
Wenn doch: Neues Relationenschema.

Probleme
- Änderungsaufwand
- Unvollständiges Einfügen
→ es können bei der Zerlegung Informationen verloren gehen.

BC-Normalform

Für jede funktionale Abhängigkeit gilt: $\boldsymbol{X \rightarrow A}$; X ist ein Superschlüssel von $\boldsymbol{R}$. So sind keine Teile zweier zusammengesetzter Schlüsselkandidaten voneinander abhängig.

Probleme
- Die Überführung in BCNF ist nicht immer abhängigkeitswahrend
- Die BCNF kann nicht immer erreicht werden!
→ Die Eigenschaften der Abhängigkeitswahrung und der informationserhaltenden Zerlegung sind ebenfalls von Bedeutung für ein insgesamt gutes Relationsschema.

Abhängigkeitswahrung

Attributserhaltende Zerlegung

> $\boldsymbol{R}$ ist nun ein Relationsschema, das alle Attribute einer DB enthält. $\boldsymbol{R}$ soll nun so zerlegt werden, dass jedes Attribut in einem $\boldsymbol{R_i}$ erscheint.
>
> Nun soll jede funktionale Abhängigkeit aus $\boldsymbol{R}$ auch in $\boldsymbol{R_i}$ erscheinen oder abgeleitet werden können. Dies ist die Abhängigkeitswahrung.

Algorithmus zur abhängigkeitswahrenden Zerlegung

1. Es muss die minimale Hülle von $\boldsymbol{G}$ für $\boldsymbol{F}$ gefunden werden
    1. $\boldsymbol{n}$ funktionale Abhängigkeiten schaffen.
    2. Abhängigkeiten mit mehreren Attributen als Quelle auf Notwendige reduzieren.
    3. überflüssige Abhängigkeiten streichen.
2. Erzeuge Relationsschemata für funktionale Abhängigkeiten.
3. Bei fehlendem Schlüssel von $\boldsymbol{R}$ muss ein solches gebildet werden.

NICHT-additiver JOIN

Wenn der JOIN zweier Relationen zu unechten Tupeln führt, also dessen Informationen nicht

korrekt sind, ist die Zerlegung nicht korrekt.

	A1	A2	A3	A4	...
R1	b11	b12	b13	b14	...
R2	b21	b22	b23	b24	...
R3	b31	b32	b33	b34	...
...	...	...	...	...	...

$R_1(A_1, A_2)$ $R_2(A_2, A_3)$ $R_3(A_1, A_2, A_3, A_4)$

1. Setze $R_i - A_i = b_{ij}$
2. Wenn das Attribut Teil von R_i ist, setze $b_{ij} = a_j$

	A1	A2	A3	A4
R1	a1	a2	b13	b14
R2	b21	a2	a3	b24
R3	a1	a2	a3	a4

3. Nun werden funktionale Abhängigkeiten behandelt $X \rightarrow Y$
   1. Wenn eine Zeile a enthält aus einer Attributsspalte x, wird in dieser Zeile von Spalte Y $Y = a$ gesetzt.
      z.B. $A_1 \rightarrow A_3$, dann ist $b_{13} = a_3$, aber nur weil $R3 \times A3 = a_3$
   2. Existiert kein a in Y wo a in x ist, werden diese Werte auf b gesetzt.

4. Ist nun eine Zeile vollständig mit a gefüllt ($R3$), dann hat die Zerlegung die Eigenschaft des nicht-additiven JOIN.

Datenqualität

Probleme

- Datenspeicherung
- Datenproduktion
- Datennutzung

- Vollständigkeit
- Genauigkeit
- Zeitnähe

- Relevanz
- Kosten
- Verständlichkeit
- Konsistenz
- Verfügbarkeit
- Glaubwürdigkeit

DB-Transaktionen

Protokolle zur Transaktionsverarbeitung

Recovery

Data Warehouse

ETL - Extraktion, Transformation, Laden

DWS - Multidimensionales Datenmodell

Begriffsammlung