Relationale Algebra

Begriffe

Zeichen: entstammen einem Zeichenvorrat, der beliebig angeordnet sein kann.
Daten: Es handelt sich hier um bereits strukturierte Zeichen. Das können Zeichen aus Alphabete, Zahlen, Satzzeichen sowie Piktogramme sein. Sie befinden sich immer auf einem Datenträger, (z.B. Blatt Papier).
Information: Erhält man, wenn diese Daten in einen Kontext gestellt werden.
Wissen: Wenn es zur Verknüpfung von Informationen und zur intellektuellen Einbettung kommt.
Beispiel: **Zeichen**: «0», «1», «.»; **Daten**: «50», «Hans»; **Information**: «50 Km/h», «Vorname: Hans»
Wissen: «Dieses Auto fährt mit 50 Stundenkilometer»

Datenarten

Strukturierte: Klare, tabellarische Struktur (Relationale Datenbanken).
Semi-str: Teilweise Struktur (XML,JSON).
Un-str: Keine festgelegte Struktur(Texte, Bilder).

Relationales Modell

Domäne: Wertebereich (Zahlen, Zeichenketten).
Attribut: Spalteneigenschaft mit Domäne (Name, Geb-Dat).
Tupel: Zeile, Sammlung von Attributwerten.
Relation: Tabelle mit Schema und Tupeln. Neue Duplikate werden entfernt, alte nicht
Schlüssel:
PK: Eindeutige Identifikation (Matrikelnummer).
FK: Verknüpft Tabellen über PK einer anderen Relation.

R

Name	Adresse	Geschlecht	Geburt
Carrie Fisher	123 Maple St., Hollywood	F	9/9/99
Mark Hamill	456 Oak Rd., Brentwood	M	8/8/88

S

Name	Adresse	Geschlecht	Geburt
Carrie Fisher	123 Maple St., Hollywood	F	9/9/99
Harrison Ford	789 Palm Dr., Beverly Hills	M	7/7/77

Vereinigung (UNION), U

R ∪ S

Name	Adresse	Geschlecht	Geburt
Carrie Fisher	123 Maple St., Hollywood	F	9/9/99
Mark Hamill	456 Oak Rd., Brentwood	M	8/8/88
Harrison Ford	789 Palm Dr., Beverly Hills	M	7/7/77

Durchschnitt (INTERSECT), ∩

R ∩ S

Name	Adresse	Geschlecht	Geburt
Carrie Fisher	123 Maple St., Hollywood	F	9/9/99

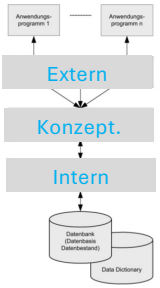
Differenz (EXCEPT), \ oder -

R - S

Name	Adresse	Geschlecht	Geburt
Mark Hamill	456 Oak Rd., Brentwood	M	8/8/88

3-Ebenen-Architektur eines RDBS

Extern: Sicht einer einzelnen Anwendung / Benutzergruppe
Konzeptionell: Logische Gesamtsicht
Intern: Speicherung, Datenorganisation, Zugriffsstrukturen
Logische Datenunabhängigkeit: Externes Schema ändert → modifizieren der Transformationsregeln zum konzeptionellen Schema.
Physische Datenunabhängigkeit: Internes Schema ändert → modifizieren der Transformationsregeln zum konzeptionellen Schema.



Selektion (σ) = select * where

$\sigma_{\text{Bedingung}}(\text{Relation})$
Wählt alle Spalten aus, die eine Bedingung erfüllen

Projektion (π) = select Spalte 1, Spalte 2

$\pi_{\text{Spalte 1, Spalte 2}}(\text{Relation})$
Wählt bestimmte Attribute aus & **entfernt Duplikate**

Umbenennung (ρ)

Ausgang: **Studierende(MatNr, Name, GebDat)**
Operation: $\rho_{\text{Studenten(ID, Vorname, Geburtsdatum)}}(\text{Studierende})$

Kreuz-Produkt (x)

R

A	B
1	2
3	4

S

B	C	D
2	5	6
4	7	8
9	10	11

R × S

A	R.B	S.B	C	D
1	2	2	5	6
1	2	4	7	8
1	2	9	10	11
3	4	2	5	6
3	4	4	7	8
3	4	9	10	11

Natural Join (⋈)

R

A	B	C
1	2	3
6	7	8
9	7	8

S

B	C	D
2	5	6
2	3	5
7	8	10

R ⋈ S

A	B	C	D
1	2	3	5
6	7	8	10
9	7	8	10

Theta Join ($⋈_P$) P = Bedingung (Kreuzprodukt)

R

A	B	C
1	2	3
6	7	8
9	7	8

S

B	C	D
2	5	6
2	3	5
7	8	10

$R ⋈_{A<D} S$

A	R.B	R.C	S.B	S.C	D
1	2	3	2	5	6
1	2	3	2	3	5
1	2	3	7	8	10
6	7	8	7	8	10
9	7	8	7	8	10

Bag-Algebra

Selektion, σ: Gleich wie relationale Algebra, Duplikate behandeln wie 'normale' Tupel.
Projektion, π: Wie relationale Algebra, **aber Duplikate nicht entfernen.**
Kreuzprodukt, x: Gleich wie relationale Algebra, Duplikate behandeln wie 'normale' Tupel.
Joins, ⋈: Gleich wie relationale Algebra, Duplikate behandeln wie 'normale' Tupel.
Vereinigung, U: Duplikate: Übernimmt die höhere Multiplitzität eines Tupels aus beiden Bags
Durschnitt, ∩: Multiplizität im Ergebnis entspricht der kleineren Multiplizität der Tupel
Bag concatenation, ⊔: Duplikate: Multiplitzitäten der Tupel werden aufsummiert
Differenz, \ - : Subtrahiert die Multiplitzitätenen der Tuepl, negative Werte werden auf 0 gesetzt
Duplikatelimination, δ: Entfernt Duplikate

Joins

Natürlicher join:

L

A	B	C
a_1	b_1	c_1
a_2	b_2	c_2

⋈

R

C	D	E
c_1	d_1	e_1
c_3	d_2	e_2

=

Resultat

A	B	C	D	E
a_1	b_1	c_1	d_1	e_1

Left outer join:

L

A	B	C
a_1	b_1	c_1
a_2	b_2	c_2

⋈

R

C	D	E
c_1	d_1	e_1
c_3	d_2	e_2

=

Resultat

A	B	C	D	E
a_1	b_1	c_1	d_1	e_1
a_2	b_2	c_2	NULL	NULL

Right outer join:

L

A	B	C
a_1	b_1	c_1
a_2	b_2	c_2

⋈

R

C	D	E
c_1	d_1	e_1
c_3	d_2	e_2

=

Resultat

A	B	C	D	E
a_1	b_1	c_1	d_1	e_1
NULL	NULL	c_3	d_2	e_2

Full outer join:

L

A	B	C
a_1	b_1	c_1
a_2	b_2	c_2

⋈

R

C	D	E
c_1	d_1	e_1
c_3	d_2	e_2

=

Resultat

A	B	C	D	E
a_1	b_1	c_1	d_1	e_1
a_2	b_2	c_2	NULL	NULL
NULL	NULL	c_3	d_2	e_2

Outer Joins

Left ⟕: Alle Tupel aus der **linken** Relation, fehlende Werde rechts mit ***null***
Right ⟖: Alle Tupel aus der **rechten** Relation, fehlende Werde links mit ***null***
Full ⟗: Alle Tupel aus beiden Relationen, fehlende Werde mit ***null***

Muster

1. Alle X ohne Y: $\pi_A(X) \setminus \pi_A(X \bowtie Y)$
2. Alle X, die mit Y übereinstimmen: $X \bowtie_{\text{Bedingung}} Y$
3. Alle X, die nicht mit Y übereinstimmen: $X \setminus (X \bowtie Y)$
4. Alle X, die sowohl Y als auch Z erfüllen: $(X \bowtie Y) \cap (X \bowtie Z)$
5. Alle X, die Y haben, aber kein Z: $(X \bowtie Y) \setminus (X \bowtie Z)$
6. Alle X, die nur mit bestimmten Y übereinstimmen: $X \bowtie_{\text{Bedingung}} Y$
7. Kombination aus Selektion und Projektion: $\pi_{\text{Attribute}}(\sigma_{\text{Bedingung}}(X))$
8. Kombinationen aus Differenzen und Vereinigungen: $(X \setminus Y) \cup Z$

Gesetze

$\sigma_\Phi(\sigma_\psi(r)) = \sigma_\psi(\sigma_\Phi(r))$ — **Kommutativität**
$\pi_A(\sigma_\Phi(r)) = \sigma_\Phi(\pi_A(r))$ falls Φ nur Attribute aus der Menge A referenziert
$r ⋈ s = s ⋈ r$ (Achtung: Relationenformat ist verschieden!)

$r ⋈ (s ⋈ t) = (r ⋈ s) ⋈ t$ — **Assoziativität**

$\pi_A(\pi_C(r)) = \pi_A(r)$ falls $A \subseteq C$
$\sigma_\Phi(\sigma_\psi(r)) = \sigma_{\Phi \wedge \psi}(r)$ — **Idempotenz**

$\pi_A(r \cup s) = \pi_A(r) \cup \pi_A(s)$
$\sigma_\Phi(r \cup s) = \sigma_\Phi(r) \cup \sigma_\Phi(s)$ — **Distributivität**
$\sigma_\Phi(r ⋈ s) = \sigma_\Phi(r) ⋈ s$ falls Φ nur Attribute von r referenziert
$\pi_{A,B}(r ⋈ s) = \pi_A(r) ⋈ \pi_B(s)$ falls für die Joinattribute J gilt: $J \subseteq A \cap B$
$r ⋈ (s \cup t) = (r ⋈ s) \cup (r ⋈ t)$

Beispiele

Gast(Besucher, Restaurant)
Sortiment(Restaurant, Biersorte)
Vorzug(Besucher, Biersorte)
sowie je eine Relation, g zum Format Gast, s zum Format Sortiment, und v zum Format Vorzug.

Alle Biersorten, die Besucher Müller vorzieht
$\pi_{Biersorte}(\sigma_{Besucher='Müller'}(v))$

Alle Restaurants, welche die Biersorte Cardinal nicht im Sortiment haben
$(\pi_{Restaurant}(s) \cup \pi_{Restaurant}(g)) \setminus \pi_{Restaurant}(\sigma_{Biersorte='Cardinal'}(s))$
Die erste Hälfte ist wichtig, da ALLE Restaurants gefragt sind, auch solche ohne Bier im Sortiment

$(g ⋈ s) \setminus (g ⋈ s ⋈ v)$
Alle Tupel <x, y, z>, so dass Besucher x im Restaurant y Gast ist und y die Biersorte z im Sortiment hat und z nicht zu den Vorzugsbiersorten von x gehört.

Die Restaurants, die zwar Gäste haben, aber keine Bier trinkenden
$\pi_{Restaurant}(g) \setminus \pi_{Restaurant}(g ⋈ v)$

Die Restaurants mit lauter Bier trinkenden Gästen, welche keines ihrer Vorzugsbiere erhalten können
$\pi_{Restaurant}(g ⋈ v) \setminus \pi_{Restaurant}(g ⋈ v ⋈ s)$

r1

A	B	C
0	1	2
0	1	2

s1

B	C	D
1	2	0
1	2	0

r2

A	B	C
0	1	2
1	1	2

s2

B	C	D
2	1	2
2	1	0
2	0	2

In Bag-Algebra

a) $(r1 \cup r2) ⋈ \delta(s1 \sqcup s2)$
b) $\pi_{B,C}(s1 \setminus \sigma_{D=0}(s2)) ⋈ r1$
c) $\pi_A(\delta(r1)) ⋈ \delta(\pi_B(s1)) ⋈ \pi_C(s2)$
d) $\delta(r1 \cap r2) ⋈ (s1 \sqcup s2)$
e) $\pi_{B,C}(r1 \setminus \sigma_{B=1}(r2)) ⋈ s1$
f) $\pi_A(r2) ⋈ \pi_B(\delta(s2)) ⋈_{A<C} \delta(\pi_C(s2))$

a)

A	B	C	D
0	1	2	0
0	1	2	0
1	1	2	0

b)

B	C	A
1	2	0
1	2	0
1	2	0
1	2	0

c)

A	B	C
0	1	1
0	1	1
0	1	0

d)

A	B	C	D
0	1	2	0
0	1	2	0

e)

B	C	D
1	2	0
1	2	0

f)

A	B	C
0	2	1
0	2	1
0	2	1

Noah Kriesi, Januar 2026

SQL

Delete

```
delete from Person where PersonID = 10; /* Löscht spezifisch */
delete from Person; /* Löscht alle Datensätze der Tabelle */
```

Update

```
update Person set Name = 'Müller' where PersonID = 5; /* Ändert einen Wert */
update Person set Salaer = Salaer * 1.1, Status = 'Aktiv' where Abteilung = 'IT'; /* Mehrere */
```

Datentypen

```
create table Datentyp_Demo (
  Ganzzahl int not null, /* Standard Ganzzahl */
  Gross_Zahl bigint not null, /* Für sehr große Zahlen */
  Dezimalzahl decimal(10,2) not null, /* Festkommazahl (z.B. Währung) */
  Fliesskomma float not null, /* Gleitkommazahl */
  Kurztext varchar(255) not null, /* Variabler Text */
  Langtext text not null, /* Große Textmengen */
  Wahrheitswert boolean not null, /* true / false */
  Datum date not null, /* 'YYYY-MM-DD' */
  Uhrzeit time not null, /* 'HH:MM:SS' */
  Zeitstempel timestamp not null, /* Datum + Uhrzeit */
  Binärdaten blob not null /* Bilder, Dateien etc. */
);
```

Insert

```
insert into Person (PersonID, Name, Abteilung) values (1, 'Meier', 'IT'); /* Einzelner Datensatz */
insert into Person (PersonID, Name) select Kundennummer, Name from Neukunden; /* Insert aus anderer Tabelle */
```

Create Table with Select / As

```
create table Projekt_Backup as select * from Projekt; /* Erstellt Tabelle mit Daten einer anderen */
create table IT_Personal as select Name, Salaer from Person where Abteilung = 'IT'; /* Mit Filter */
```

Select Into

```
select * into Archiv_Tabelle from Person; /* Erstellt neue Tabelle und kopiert Daten */
```

Alter Table

```
alter table Person add column Telefon varchar(20) not null; /* Spalte hinzufügen */
alter table Person drop column Telefon; /* Spalte löschen */
alter table Person modify column Name varchar(100) not null; /* Datentyp ändern */
alter table Person rename to Mitarbeiter; /* Tabelle umbenennen */
alter table Mitarbeiter rename column Name to Nachname; /* Spalte umbenennen */
```

Order By

```
select * from Person order by Name asc; /* Aufsteigend sortieren */
select * from Person order by Salaer desc; /* Absteigend sortieren */
select * from Person order by Abteilung asc, Salaer desc; /* Mehrfache Sortierung */
```

Projektion

```
select Name from Person; /* π Name(Person) */
select DISTINCT Name from Person; /* δ(π Name(Person)) */
```

Selektion

```
select * from Person where Name = 'Müller'; /* σ(Name='Müller'(Person)) */
```

JOINs

```
select Salaer from Person p join Verkauft v ON p.PersonID = v.PersonID; /* ⋈ */
```

Outer Joins

```
select * from Person p left join Verkauf v ON p.PID = v.PID; /* Links alle */
select * from Person p right join Verkauf v ON p.PID = v.PID; /* Rechts alle */
select * from Person p full outer join Verkauf v ON p.PID = v.PID; /* Beide Seiten */
```

Intersect / Except

```
select Name from Person intersect select Name from Kunde; /* Schnittmenge */
select Name from Person except select Name from Kunde; /* Differenzmenge */
```

NULL

```
... WHERE Preis is null;
```

Case

```
select Name, case when Salaer > 50000 THEN 'Gut' else 'Normal' end as Status from Person;
```

Count

```
select count(*) from Person;
select count(distinct(Name)) from Person;
```

SUM / AVG

```
select SUM(Preis) from Produkt;
select AVG(Preis) from Produkt;
```

Group By / Having

```
select Abteilung, COUNT(*) AS Gesamtgehalt from Person GROUP BY Abteilung;
select Abteilung, SUM(Salaer) as Gesamtgehalt from Person GROUP BY Abteilung HAVING SUM(Salaer) > 100000;
```

Referenzielle Integrität

```
on delete cascade; /* Löscht Kind, wenn Elternteil gelöscht wird */
on update cascade; /* Aktualisiert FK, wenn PK geändert wird */
on delete restrict; /* Verhindert Löschen des Elternteils bei Existenz von Kindern */
on delete set null; /* Setzt FK auf NULL */
on delete no action; /* Standardverhalten */
```

Table Creation (Full)

```
create table Projekt_Zuweisung (
  PersonID int not null,
  ProjektID int not null,
  Email varchar(100) unique not null,
  Alter int check (Alter >= 18) not null,
  Rolle varchar(50) not null,
  Status varchar(20) default 'Aktiv' not null,
  Ref_ID1 int not null,
  Ref_ID2 int not null,
  primary key (PersonID, ProjektID),
  constraint fk_person foreign key (PersonID) references Person(PersonID) on delete cascade on update cascade,
  constraint fk_projekt foreign key (ProjektID) references Projekt(ProjektID) on delete restrict on update cascade,
  constraint fk_composite foreign key (Ref_ID1, Ref_ID2) references AndereTabelle(Spalte1, Spalte2),
  constraint chk_Status check (Status in ('Aktiv', 'Inaktiv', 'Urlaub'))
);
```

IMMER NOT NULL !

Exists

```
select Name from Person p where exists (select 1 from Verkauf v where v.PID = p.PID);
select Name from Person p where not exists (select 1 from Verkauf v where v.PID = p.PID);
```

IN / NOT IN

```
select * from Person where Name in ('Müller', 'Meier', 'Schmid');
select * from Person where Name not in ('Müller', 'Meier');
```

Views

```
create view Personal_Liste as select Name, Abteilung from Person;
```

CTEs

```
with Ergebnis as (select * from Person) select * from Ergebnis;
```

Like / Between

```
select * from Person where Name like 'Mü%'; /* % Platzhalter */
select * from Person where Salaer between 40000 and 60000;
```

Any / All

```
select * from Person where Salaer > any (select Salaer from Person where Abt = 'IT');
select * from Person where Salaer > all (select Salaer from Person where Abt = 'IT');
```

Stored Procedures

```
create procedure Gehalt_Anpassen (in p_ID int, in p_Prozent dec(5,2))
begin
  update Person set Salaer = Salaer * (1 + p_Prozent / 100) where PersonID = p_ID;
end;
/* Aufruf: call Gehalt_Anpassen(10, 5.0); */
```

Trigger

```
create trigger trigger_name
after update on Person
for each row
execute procedure log_funktion(); /* Führt Logik bei Änderung aus */
```

Index

```
create index idx_name on Person(Name);
create unique index idx_email on Person(Email);
drop index idx_name;
```

Union / Union All

```
select Name from Person union select Name from Kunde; /* Vereinigt Mengen (ohne Duplikate) */
select Name from Person union all select Name from Kunde; /* Vereinigt Mengen (mit Duplikaten) */
```

Wann Index?

1. Attribute, die oft abgefragt werden, sollten indiziert werden.
2. Auch Fremdschlüssel sollten indexiert werden, insbesondere dann, wenn über «Primär-Fremdschlüssel» gejoint wird (was häufig der Fall ist).

   Allenfalls auch nach FK clustern.
3. Attribute über die oft gejoint wird; wenn über mehrere Attribute gejoint wird dann muss ein zusammengesetzter Index verwendet werden.
4. Attribute mit niedriger Kardinalität (Extrembeispiele: Geschlecht, Ja/Nein-Flags u.ä.) sollten nicht indexiert werden (es gibt dafür spezielle Indexstrukturen, hier aber nicht behandelt).

Achtung:

Überindexierung kostet Ausführungszeit und Speicherplatz und kann den Optimizer in die Irre führen (d.h. Abfragen können sogar langsamer werden, siehe vorheriges Beispiel).

Noah Kriesi, Januar 2026

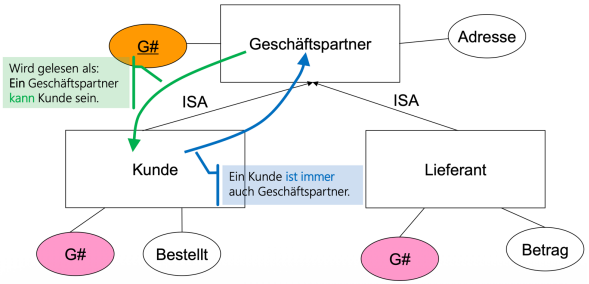
ERD

Begriffe

Entitätstyp: Eine Klasse von Objekten, z. B. "Kunde" oder "Lieferant".
Attribut: Eine Eigenschaft eines Entitätstyps, z. B. "Name" oder "Adresse".
Beziehungstyp: Verknüpfung zwischen Entitätstypen, z. B. "Angestellt".
Primärschlüssel: Attribut oder Kombination von Attributen, die eine Entität eindeutig identifiziert.
Fremdschlüssel: Attribut in einer Tabelle, das auf den Primärschlüssel einer anderen Tabelle verweist.
Normalisierung: Prozess zur Strukturierung von Datenbanken, um Redundanzen und Anomalien zu minimieren.

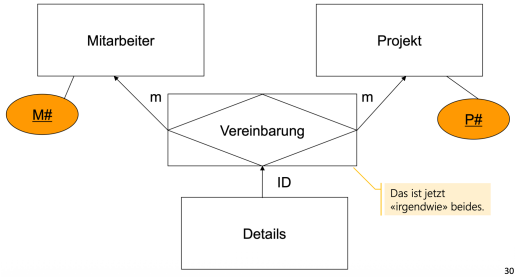
ISA-abhängiger Entitätstyp (ISA-dependent entity type)

- Jeder Kunde resp. jeder Lieferant ist auch Geschäftspartner



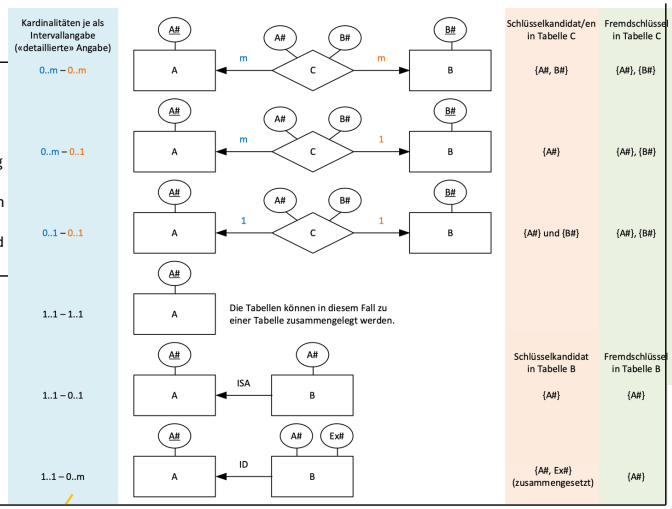
- Die Pfeile zeigen eine existentielle Abhängigkeit ("is a" = "ist ein")
- Die Pfeile führen zu Schlüsselbedingungen
  - Und gleichzeitig Fremdschlüssel in Kunde und Lieferant.
- {G#} ist Schlüssel in Kunde und in Lieferant. Es wird erzwungen, dass jedem Kunden ein Geschäftspartner entspricht – der Kunde selbst!
- Ein Geschäftspartner kann Kunde UND Lieferant sein
- "ISA" ist ein Generalisierungs-/Spezialisierungsmuster.
- Kunde und Lieferant werden zu Geschäftspartnern verallgemeinert
- Kunde und Lieferant sind Spezialisierungen von Geschäftspartner

Zusammengesetzter Entitätstyp (composite entity type)



30

- Wir hängen einen ID-abhängigen Entitätstyp "Details" an, der von Vereinbarung zu Vereinbarung unterschiedlich viele Details aufnehmen soll.
- Der Beziehungstyp wird "umgewandelt" in einen zusammengesetzten Entitätstyp (zeichnerisch: wir zeichnen ein Rechteck um den Beziehungstyp)



- Es sei {Verein, Spieler} Schlüssel. Dann ist aber {Stadt} nur von {Verein} abhängig.

Verein	Spieler	Salär
Juventus	Meier	70000
Juventus	Müller	72000
Juventus	Gerber	66000
Concordia	Müller	67000
Concordia	Gerber	87000
Concordia	Baumann	55000
Concordia	Berger	61000

Verein	Stadt
Juventus	Basel
Concordia	Zürich

Hat es noch mehr Schlüssel (z.B. {Stadt, Spieler})? → «Designentscheid»!

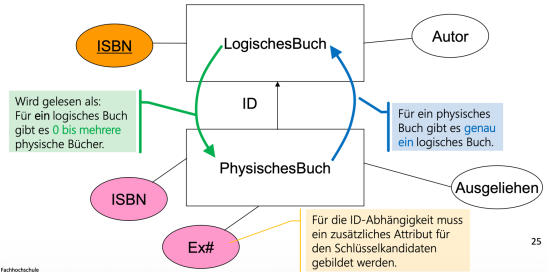
Ausgangstabelle:

- Die rechte Tabelle enthält alle Attribute, welche nur von {Verein} abhängig sind
- Die linke Tabelle enthält alle Attribute, welche nur von {Verein, Spieler} abhängig sind

38

ID-abhängiger Entitätstyp (ID-dependent entity type)

- Eine Bibliothek hat für beliebte Titel mehrere Exemplare



25

- Der Entitätstyp "PhysischesBuch" ist ID-abhängig vom Entitätstyp "LogischesBuch".
- Einem logischen Begriff eines Buchs (Primärschlüssel {ISBN}) entspricht eine Menge von physikalischen Büchern
- "Innerhalb" eines logischen Buchs brauchen wir ein weiteres Attribut, z.B. "Ex#", um die einzelnen physikalischen Exemplare zu unterscheiden
  - Und ISBN ist gleichzeitig Fremdschlüssel in PhysischesBuch.
- Es bilden also {ISBN, Ex#} einen Schlüssel
- Eine Entität des ID-abhängigen Typs kann "auf natürliche Weise" nur innerhalb der Hierarchie identifiziert werden (Bsp. Kind durch Vornamen innerhalb der Familie)

NF: Häufige Abkürzung in der Praxis.

1NF (Erste Normalform)

Person		Hobbies
Name	Vorname	
Meier	Hans	Fussball, Lesen, Jogging
Müller	Peter	Briefmarken, Fussball
Gerber	Susanne	Volleyball, Ikebana

Name	Vorname	Hobby
Meier	Hans	Fussball
Meier	Hans	Lesen
Meier	Hans	Jogging
Müller	Peter	Briefmarken
Müller	Peter	Fussball
Gerber	Susanne	Volleyball
Gerber	Susanne	Ikebana

- Jedes Attribut muss atomare Wertebereiche haben
- Zusammengesetzte oder mengenwertige Attribute sind nicht erlaubt
- Anfragen werden erleichtert, da Attributwertebereiche atomar

1. NF basiert auf Praxiserfahrung.

35

2NF (Zweite Normalform)

Verein	Stadt	Spieler	Salär
Juventus	Basel	Meier	70000
Juventus	Basel	Müller	72000
Juventus	Basel	Gerber	66000
Concordia	Zürich	Müller	67000
Concordia	Zürich	Gerber	87000
Concordia	Zürich	Baumann	55000
Concordia	Zürich	Berger	61000

d.h. von allen Attributen im Schlüsselkandidaten

- Jedes Nichtschlüsselattribut muss von jedem Schlüsselkandidaten voll funktional abhängig sein, d.h.
- Same same: Alle Attribute, die nicht Teil des Schlüssels sind, sind vom ganzen Schlüssel abhängig, nicht nur von Teilen

Vorsicht, für jeden Schlüsselkandidaten

37

3NF (Dritte Normalform)

- {Spieler} und {Verein} lassen sich aus {Spieler_ID} bestimmen
- {Gründungsjahr} hängt dagegen eigentlich von {Verein} ab, und damit nur indirekt von {Spieler_ID}

Spieler_ID	Spieler	Verein
3207	Meier	Juventus
3209	Müller	Juventus
3216	Gerber	Juventus
7002	Müller	Concordia
7003	Gerber	Concordia
7007	Baumann	Concordia
7011	Berger	Concordia

Verein	Gründungsjahr
Juventus	1992
Concordia	1994

Transitive Abhängigkeit

Ausgangstabelle:

Verein ist nicht Schlüsselkandidat.

Divers

ACID

Atomarität (Atomicity): Alles oder nichts. Eine Transaktion wird entweder komplett ausgeführt oder bei einem Fehler vollständig rückgängig gemacht (Rollback).
Konsistenz (Consistency): Die Datenbank ist vorher und nachher in einem logisch korrekten Zustand; Regeln werden niemals verletzt.
Isolation (Isolation): Gleichzeitige Zugriffe stören sich nicht. Jeder Nutzer arbeitet so, als wäre er allein im System.
Dauerhaftigkeit (Durability): Einmal gespeicherte Daten gehen nicht mehr verloren, selbst bei einem Systemabsturz (Recovery).

Nebenläufigkeit

Lost-Update: Überschreiben bereits getätigter Updates

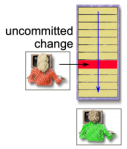
	Transaktion Benutzer 1	Transaktion Benutzer 2
1	SELECT Wert INTO W FROM Tbl	
2		SELECT Wert INTO W FROM Tbl
3	UPDATE Tbl SET Wert = 100	
4		UPDATE Tbl SET Wert = 200
5	SELECT Wert INTO W FROM Tbl	
6		SELECT Wert INTO W FROM Tbl

Die nachfolgenden Probleme sind nicht nur theoretisch Interessant, diese sind für die Praxis (mittels Isolationsebenen definiert) wichtig.

- Keine Isolation der Transaktionen beider Benutzer → Update-Operation von Benutzer 1 geht verloren!
- Lösung: Durch andere Transaktion gelesene Daten dürfen bis zur deren Beendigung nicht verändert werden.

Dirty-Read: Lesen von Veränderungen noch nicht abgeschlossener Transaktionen

	Transaktion Benutzer 1	Transaktion Benutzer 2
1	SELECT Wert INTO W FROM Tbl	
2	UPDATE Tbl SET Wert = NeuerWert	
3		SELECT Wert INTO W FROM Tbl
4	ROLLBACK	
5		UPDATE Tbl SET Wert = W + 1
6		SELECT Wert INTO W FROM Tbl
7	SELECT Wert INTO W FROM Tbl	



- Keine Isolation der Transaktionen beider Benutzer → Benutzer 2 arbeitet mit einem nicht bestätigten Wert, der später sogar zurückgenommen wird!
- Lösung: Nur Updates bestätigter Transaktionen lesen.

Non-Repeatable-Read: Wiederholtes Lesen von zwischenzeitlich von anderen Transaktionen durchgeführten Veränderungen

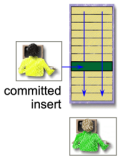
	Transaktion Benutzer 1	Transaktion Benutzer 2
1	SELECT Wert INTO W FROM Tbl	
2		UPDATE Tbl SET Wert = Wert + 5
3		COMMIT
4	SELECT Wert INTO W FROM Tbl	

2 mal gelesen

- Keine Isolation der Transaktionen beider Benutzer → Benutzer 1 erhält nicht immer denselben Wert!
- Lösung: Nur den Datenbankzustand sehen, der bei Beginn einer Transaktion vorliegt.

Phantom-Read: Lesen von zwischenzeitlich von anderen Transaktionen eingefügten Daten

	Transaktion Benutzer 1	Transaktion Benutzer 2
1	SELECT count(*) INTO cnt FROM Tbl	
2	N = cnt	
3		INSERT INTO Tbl VALUES (…)
4		COMMIT
5	SELECT count(*) INTO cnt FROM Tbl	
6		



2 mal gelesen

- Keine Isolation der Transaktionen beider Benutzer → Benutzer 1 sieht Zwischenresultate, die von Benutzer 2 eingefügt wurden!
- Lösung: Nur den Datenbankzustand sehen, der bei Beginn einer Transaktion vorliegt

Lost Update: kann in einem Transaktionssystem fast nie toleriert werden.
Dirty-Read, Non-Repeatable-Read, Phantom-Read:
bei konkurrentem Lesen oft zu einem gewissen Ausmass tolerierbar

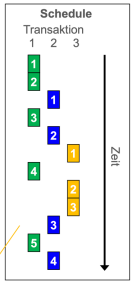
Schedules

Erzeugt einen serialisierbaren Ablaufplan s für parallel auszuführende Transaktionen T → welche Transaktion macht den nächsten Schritt?
Verwaltet notwendige Synchronisationsinformationen, z.B. die Lese- und Schreibsperren

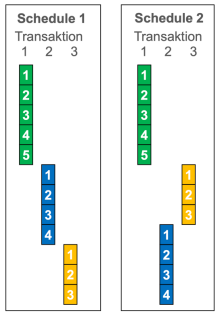
Typ	Strategie	Vorteil	Nachteil	Beispiel
Aggressiv	Konflikte erst am Ende lösen	Maximale Parallelität	Risiko von Rollbacks	Postgres, Oracle
Konservativ	Konflikte durch Warten vermeiden	Wenig Abbruch-Aufwand	Langsamer (Stau)	SQL Server

Schedule (Ablaufplan):
- Folge von Lese- bzw. Schreiboperationen für die (parallele) Ausführung einer oder mehrerer Transaktionen.
- Schedules können ACID-Eigenschaften verletzen.

Vollständiger Schedule = History
- Sämtliche Schritte aller anstehenden Transaktionen inklusive Terminierung (COMMIT, ABORT)
- Für jede Transaktion ist festgehalten, ob sie erfolgreich endet oder abbricht (dies kann in einem Schedule noch offen sein).



Serieller Schedule
- Alle Transaktionen nacheinander.
- Für n Transaktionen existieren n! verschiedene serielle Schedules.
- Serielle Schedules werden als konsistenzerhaltend betrachtet.
- Sicher, aber schlechte Performance → verschränkte Ausführung ist erwünscht.



Isolationsebene

```
SET TRANSACTION ISOLATION LEVEL {READ UNCOMMITTED | …}
```

Isolationsebenen vs. Phänomene:

Isolationsebene	Dirty Read	Non-Repeatable Read	Phantom Read	Lost Update
READ UNCOMMITTED	~~möglich~~ (nicht in Postgres)	möglich	möglich	~~möglich~~ (nicht in Postgres)
READ COMMITTED	verhindert	möglich	möglich	verhindert
REPEATABLE READ	verhindert	verhindert	~~möglich~~ (nicht in Postgres)	verhindert
SERIALIZABLE	verhindert	verhindert	verhindert	verhindert

Häufig in der Praxis

Default bei Postgres, MS SQL Server und Oracle.

In Postgres: READ UNCOMMITTED = READ COMMITTED

Transaktionen

COMMIT; : Transaktionsresultat wird permanent.
ROLLBACK; : Annulation alles Veränderungen.
Transaktionsanweisungen in SQL:
BEGIN TRANSACTION (implizit und/oder explizit)
COMMIT TRANSACTION
ROLLBACK TRANSACTION

Lese-Sperre (Share Lock): read lock (rl) – read unlock (ru)
andere Transaktionen weiterhin lesend auf das Datenbankobjekt zugreifen und Lesesperren auf dieses Datenbankobjekt setzen. Sie können jedoch keine Schreibsperre setzen)

Schreib-Sperre (Exclusive Lock): write lock (wl) – write unlock (wu)
das betreffende Datenbankobjekt nur dieser Transaktion zur Verfügung und kann nur durch diese Transaktion geändert werden

Unlock: read unlock (ru) und write unlock (wu) werden üblicherweise zu unlock (u) zusammengefasst.

Noah Kriesi, Januar 2026