Tabellen, Constraints und relationaler Entwurf

Grundlagen zur Definition von Tabellen

Erzeugung von Tabellen

```
CREATE TABLE products (
    product_no integer,
    name       text,
    price      numeric
);

CREATE TABLE orders (
    order_id   integer,
    product_no integer,
    quantity   integer
);
```

Löschen von Tabellen

```
DROP TABLE products;
DROP TABLE orders;
```

Default-Werte

```
CREATE TABLE products (
    product_no integer,
    name       text    DEFAULT '',
    price      numeric DEFAULT 9.99
);
```

Arten von Constraints

- Check Constraint
- NOT NULL Constraint
- Unique Constraint
- Primary Key (Primärschlüssel)
- Foreign Key (Fremdschlüssel)

Check Constraint

Check-Klause, Column-Constraint, Constraint-Name

```
CREATE TABLE products (
    product_no integer,
    name       text,
    price      numeric CHECK (price > 0)
);

CREATE TABLE products (
    product_no integer,
    name       text,
    price      numeric CONSTRAINT positive_price CHECK (price > 0)
);
```

Constraint-Name wird in Fehlermeldungen bei Constraint-Verletzungen angegeben

Table Constraint

```
CREATE TABLE products (
    product_no       integer,
    name             text,
    price            numeric CHECK (price > 0),
    discounted_price numeric CHECK (discounted_price > 0),
    CHECK (price > discounted_price)
);

CREATE TABLE products (
    product_no       integer,
    name             text,
    price            numeric,
    CHECK (price > 0),
    discounted_price numeric,
    CHECK (discounted_price > 0),
    CHECK (price > discounted_price)
);

CREATE TABLE products (
    product_no       integer,
    name             text,
    price numeric CHECK (price > 0),
    discounted_price numeric,
    CHECK (discounted_price > 0 AND price > discounted_price)
);

CREATE TABLE products (
    product_no       integer,
    name             text,
    price            numeric,
    CHECK (price > 0),
    discounted_price numeric,
    CHECK (discounted_price > 0),
    CONSTRAINT valid_discount CHECK (price > discounted_price)
);
```

Not-Null Constraint

```
CREATE TABLE products (
    product_no integer NOT NULL,
    name       text    NOT NULL,
    price      numeric
);

CREATE TABLE products (
    product_no integer NOT NULL,
    name       text    NOT NULL,
    price      numeric NOT NULL CHECK (price > 0)
);

CREATE TABLE products (
    product_no integer NULL,
    name       text    NULL,
    price      numeric NULL
);
```

NULL Constraint *nicht* im SQL-Standard

Unique Constraints

Die Angabe UNIQUE bedeutet, dass die Spalte oder die Gruppe von Spalten in der Tabelle für jedes Tupel eindeutig

```
CREATE TABLE products (
    product_no integer UNIQUE,
    name       text,
    price      numeric
);

CREATE TABLE products (
    product_no integer,
    name       text,
    price      numeric,
    UNIQUE (product_no)
);

CREATE TABLE person (
    fname   text,
    lname   text,
    address text,
    UNIQUE (fname, lname)
);

CREATE TABLE products (
    product_no integer CONSTRAINT must_be_unique UNIQUE,
    name       text,
    price      numeric
);
```

Primary Keys

Primary Key Constraint: Kombination von Unique Constraint und Not-NULL Constraint

```
CREATE TABLE products (
    product_no integer UNIQUE NOT NULL,
    name       text,
    price      numeric
);

CREATE TABLE products (
    product_no integer PRIMARY KEY,
    name       text,
    price      numeric
);

CREATE TABLE person (
    fname   text,
    lname   text,
    address text,
    PRIMARY KEY (fname, lname)
);
```

Die Definition eines Primary Key bedeutet, dass die Spalte oder die Gruppe von Spalten in der Tabelle für jedes Tupel eindeutig und nicht NULL sind

Foreign Keys

```
CREATE TABLE products (
    product_no integer PRIMARY KEY,
    name       text,
    price      numeric
);

CREATE TABLE orders (
    order_id   integer PRIMARY KEY,
    product_no integer REFERENCES products (product_no),
    quantity   integer
);

CREATE TABLE orders (
    order_id   integer PRIMARY KEY,
    product_no integer REFERENCES products,
    quantity   integer
);

CREATE TABLE company (
  cname         text PRIMARY KEY,
  manager_fname text,
  manager_lname text,
  FOREIGN KEY (manager_fname, manager_lname)
    REFERENCES person (fname, lname)
);
```

```
CREATE TABLE products (
    product_no integer PRIMARY KEY,
    name       text,
    price      numeric
);

CREATE TABLE orders (
    order_id         integer PRIMARY KEY,
    shipping_address text,
    shipping_date    text
);

CREATE TABLE order_items (
    product_no integer REFERENCES products,
    order_id   integer REFERENCES orders,
    quantity   integer,
    PRIMARY KEY (product_no, order_id)
);
```

Ein Fremdschlüssel muss Spalten referenzieren die (in der referenzierten Tabelle) entweder als Primärschlüssel oder mit einem UNIQUE Constraint gekennzeichnet sind.

Optionen für den Fall, dass versucht wird ein referenziertes Tupel zu löschen

- Verbot des Löschens des referenzierten Tupels

- Löschen des referenzierten Tupels und Löschen der referenzierenden Tupels

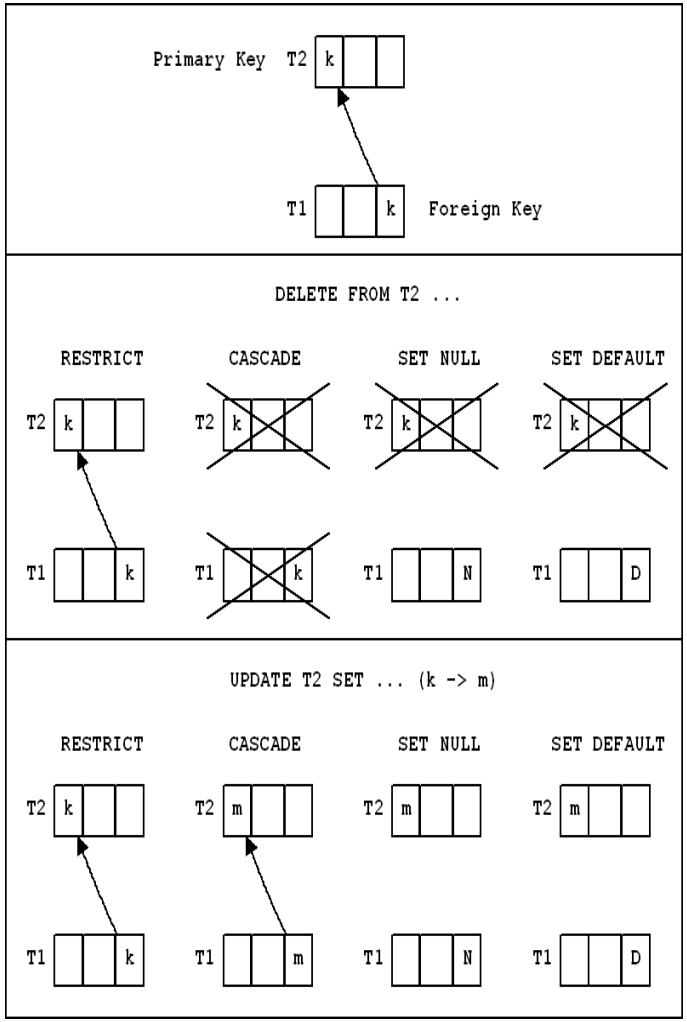
```
CREATE TABLE products (
    product_no integer PRIMARY KEY,
    name       text,
    price      numeric
);

CREATE TABLE orders (
    order_id         integer PRIMARY KEY,
    shipping_address text,
    shipping_date    text
);

CREATE TABLE order_items (
    product_no integer REFERENCES products ON DELETE RESTRICT,
    order_id   integer REFERENCES orders ON DELETE CASCADE,
    quantity   integer,
    PRIMARY KEY (product_no, order_id)
);
```

RESTRICT verhindert das Löschen des referenzierten Tupels

CASCADE zeigt an, dass das referenzierende Tupel auch gelöscht werden soll

Weitere Optionen: SET NULL und SET DEFAULT; das referenzierende Tupel erhält als Referenzwert NULL bzw. den Default-Wert

Analog zu ON DELETE ist auch die Angabe ON UPDATE erlaubt



Beispiel

```
CREATE TABLE author ( fname VARCHAR,
                      lname VARCHAR,
                      title VARCHAR,
                      yearp INTEGER ); -- publication

INSERT INTO author VALUES ('Ada','Black','UML',2002);
INSERT INTO author VALUES ('Bob','Green','UML',2002);
```

```
INSERT INTO author VALUES ('Cyd','White','DBS',1990);

CREATE TABLE person ( fname VARCHAR,
                      lname VARCHAR,
                      yearb INTEGER ); -- birth

INSERT INTO person VALUES ('Ada','Black',1965);
INSERT INTO person VALUES ('Ada','White',1975);
INSERT INTO person VALUES ('Bob','Green',1970);
INSERT INTO person VALUES ('Dan','Green',1992);
```

Ohne Contraints ist eine Tabelle *keine* Relation

```
INSERT INTO person VALUES ('Flo','Brown',1992);
INSERT INTO person VALUES ('Flo','Brown',1992);

SELECT * FROM person ORDER BY 1, 2;

 fname | lname | yearb
-------+-------+-------
 Ada   | Black |  1965
 Ada   | White |  1975
 Bob   | Green |  1970
 Dan   | Green |  1992
 Flo   | Brown |  1992
 Flo   | Brown |  1992
(6 rows)

-- TABLE person war *ohne* Key und *ohne* Constraints definiert worden

DELETE FROM person WHERE fname='Flo';

>DELETE 2

SELECT * FROM person;

 fname | lname | yearb
-------+-------+-------
 Ada   | Black |  1965
 Ada   | White |  1975
 Bob   | Green |  1970
 Dan   | Green |  1992
(4 rows)

ALTER TABLE person ADD CONSTRAINT personIsRelation
  UNIQUE (fname,lname,yearb);

> NOTICE:  ALTER TABLE / ADD UNIQUE will create implicit
> index "personisrelation" for table "person"
> ALTER TABLE

INSERT INTO person VALUES ('Flo','Brown',1992);

> INSERT 0 1
```



```
SELECT * FROM person;

 fname | lname | yearb
-------+-------+-------
 Ada   | Black |  1965
 Ada   | White |  1975
 Bob   | Green |  1970
 Dan   | Green |  1992
 Flo   | Brown |  1992
(5 rows)

INSERT INTO person VALUES ('Flo','Brown',1992);

> ERROR: duplicate key violates unique constraint "personisrelation"

DELETE FROM person WHERE fname='Flo';

>DELETE 1
```

Beispiel für Zyklische Fremdschlüssel

```
ALTER TABLE author ADD PRIMARY KEY (fname,lname);
> NOTICE:  ALTER TABLE / ADD PRIMARY KEY will create implicit
> index "author_pkey" for table "author"
> ALTER TABLE

ALTER TABLE person ADD PRIMARY KEY (fname,lname);
> NOTICE:  ALTER TABLE / ADD PRIMARY KEY will create implicit
> index "person_pkey" for table "person"
> ALTER TABLE

ALTER TABLE author ADD FOREIGN KEY (fname,lname)
  REFERENCES person(fname,lname);
> ERROR:  insert or update on table "author" violates foreign key
> constraint "author_fname_fkey"
> DETAIL: Key (fname,lname)=(Cyd,White) not present in "person".

DELETE FROM author WHERE fname='Cyd' AND lname='White';
> DELETE 1

ALTER TABLE author ADD FOREIGN KEY (fname,lname)
  REFERENCES person(fname,lname);
> ALTER TABLE

ALTER TABLE person ADD FOREIGN KEY (fname,lname)
  REFERENCES author(fname,lname);
> ERROR:  insert or update on table "person" violates foreign key
> constraint "person_fname_fkey"
> DETAIL: Key (fname,lname)=(Ada,White) not present in "author".

DELETE FROM person WHERE (fname='Ada' AND lname='White') OR
                         (fname='Dan' AND lname='Green');
> DELETE 2
```

```
ALTER TABLE person ADD FOREIGN KEY (fname,lname)
  REFERENCES author(fname,lname);
> ALTER TABLE

SELECT * FROM author;
 fname | lname | title | yearp
-------+-------+-------+-------
 Ada   | Black | UML   |  2002
 Bob   | Green | UML   |  2002

SELECT * FROM person;
 fname | lname | yearb
-------+-------+-------
 Ada   | Black |  1965
 Bob   | Green |  1970

PRIMARY KEY (fname,lname) in author eigentlich unglücklich,
da z.B. ('Ada','Black','DBS',2004) nicht eingefügt werden kann
```

Syntax für CREATE TABLE

```
CREATE [ [ GLOBAL | LOCAL ] { TEMPORARY | TEMP } ]
  TABLE table_name ( [
  { column_name data_type [ DEFAULT default_expr ]
    [ column_constraint [ ... ] ]
    | table_constraint
    | LIKE parent_table [ { INCLUDING | EXCLUDING }
      { DEFAULTS | CONSTRAINTS } ] ... }
    [, ... ]
] )
[ INHERITS ( parent_table [, ... ] ) ]
[ WITH ( storage_parameter [= value] [, ... ] ) |
  WITH OIDS | WITHOUT OIDS ]
[ ON COMMIT { PRESERVE ROWS | DELETE ROWS | DROP } ]
[ TABLESPACE tablespace ]

column_constraint:

[ CONSTRAINT constraint_name ]
{ NOT NULL |
  NULL |
  UNIQUE index_parameters |
  PRIMARY KEY index_parameters |
  CHECK ( expression ) |
  REFERENCES reftable [ ( refcolumn ) ]
    [ MATCH FULL | MATCH PARTIAL | MATCH SIMPLE ]
    [ ON DELETE action ] [ ON UPDATE action ] }
[ DEFERRABLE | NOT DEFERRABLE ]
[ INITIALLY DEFERRED | INITIALLY IMMEDIATE ]

table_constraint:

[ CONSTRAINT constraint_name ]
{ UNIQUE ( column_name [, ... ] ) index_parameters |
  PRIMARY KEY ( column_name [, ... ] ) index_parameters |
```



```
  CHECK ( expression ) |
  FOREIGN KEY ( column_name [, ... ] )
    REFERENCES reftable [ ( refcolumn [, ... ] ) ]
    [ MATCH FULL | MATCH PARTIAL | MATCH SIMPLE ]
    [ ON DELETE action ] [ ON UPDATE action ] }
[ DEFERRABLE | NOT DEFERRABLE ]
[ INITIALLY DEFERRED | INITIALLY IMMEDIATE ]

index_parameters in UNIQUE und PRIMARY KEY constraints:

[ WITH ( storage_parameter [= value] [, ... ] ) ]
[ USING INDEX TABLESPACE tablespace ]
```

Zusammenfassung Tabellen

- Erzeugung, Löschen, Ändern von Tabellen
  CREATE TABLE, DROP TABLE, ALTER TABLE

- Spalten (Attribute) mit Datentypen und möglicherweise
  mit Default-Werten

- Constraints:
  Check, Not Null, Unique, Primary key, Foreign Key

- CREATE FUNCTION zur Kapselung von komplexen SQL-Ausdrücken

Conceptual UML and OCL Example Model

Classes and Associations

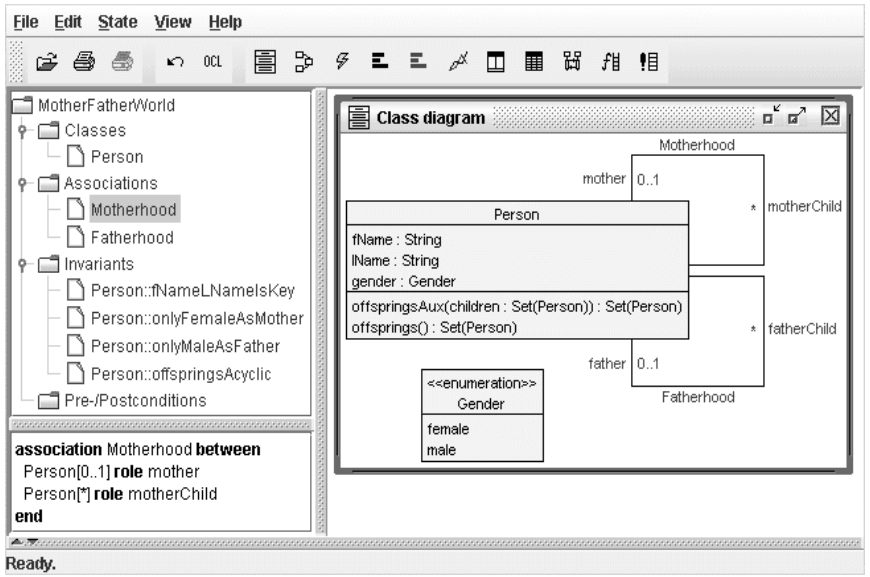
```
model MotherFatherWorld

enum Gender {female, male}

class Person
attributes
  fName:String
  lName:String
  gender:Gender
operations
  offspringsAux(children:Set(Person)):Set(Person)=
    let oneStep:Set(Person)=
      children.motherChild->union(children.fatherChild)->asSet in
    if oneStep->exists(p|not(children->includes(p)))
      then offspringsAux(children->union(oneStep))
      else children endif
  offsprings():Set(Person)=
    offspringsAux(self.motherChild->union(self.fatherChild))
end
```

```
association Motherhood between
  Person[0..1] role mother
  Person[0..*] role motherChild
end

association Fatherhood between
  Person[0..1] role father
  Person[0..*] role fatherChild
end
```

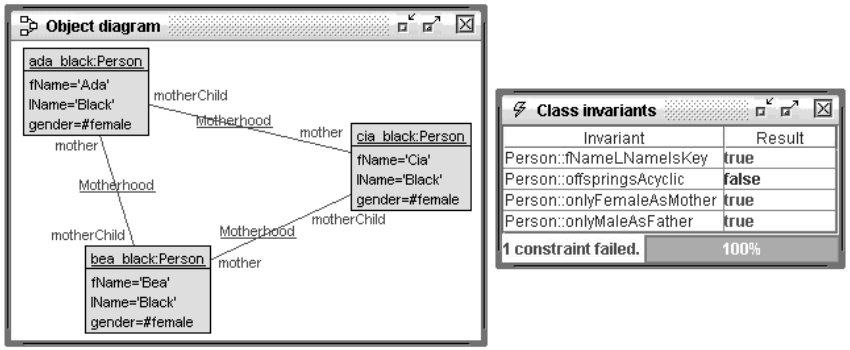
Constraints

```
constraints

context self:Person inv fNameLNameIsKey:
  Person.allInstances->forAll(self2| self<>self2 implies
    (self.fName<>self2.fName or self.lName<>self2.lName))

context self:Person inv onlyFemaleAsMother:
  self.motherChild->notEmpty implies
  (self.gender=#female and self.fatherChild->isEmpty)

context self:Person inv onlyMaleAsFather:
  self.fatherChild->notEmpty implies
  (self.gender=#male and self.motherChild->isEmpty)

context self:Person inv offspringsAcyclic:
  not(self.offsprings()->includes(self))
```

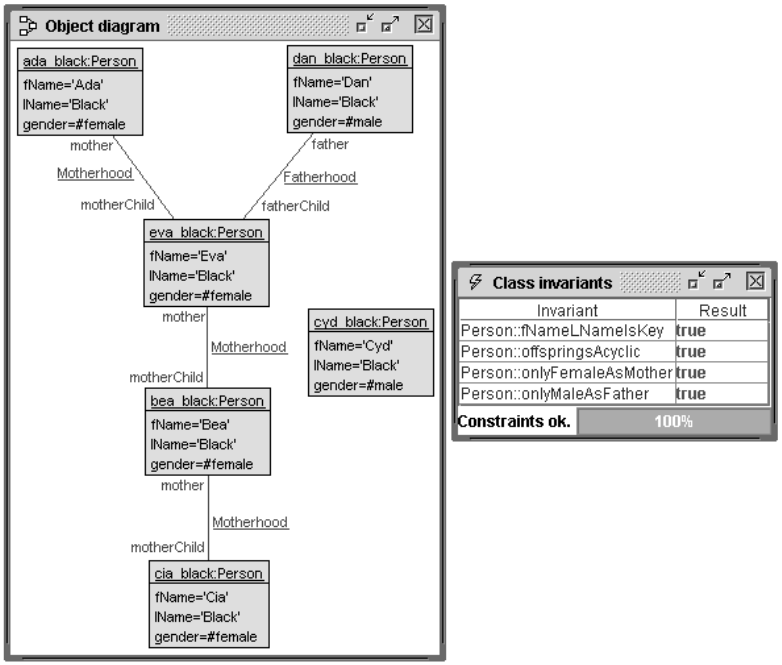
Class Diagram



Object Diagram Violating Constraints



Object Diagram Satisfying Constraints

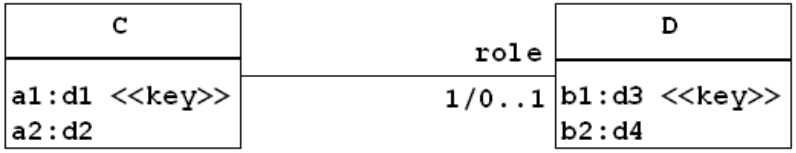


Transformation zwischen UML und relationalen DB-Schemata

Übersetzung von Klassen


```
CREATE TABLE C (
  a1 d1,
  a2 d2,
  PRIMARY KEY ( a1 )
)
```

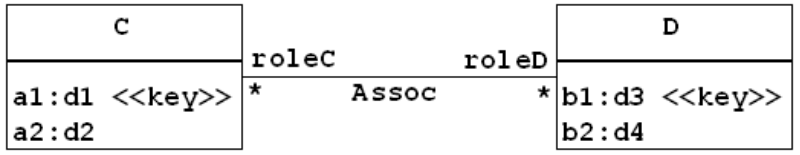

Übersetzung von funktionalen Assoziationen (Multiplizität 1 oder 0..1)




```
CREATE TABLE C (
  a1 d1,
  a2 d2,
  role d3 NOT NULL / NULL,
  PRIMARY KEY ( a1 ),
  FOREIGN KEY ( role ) REFERENCES D
)
```

Liegt eine (1/0..1,1/0..1) vor, kann diese Assoziation (in C) oder (in D) oder (in C und D mit zusätzlichem Constraint) realisiert werden

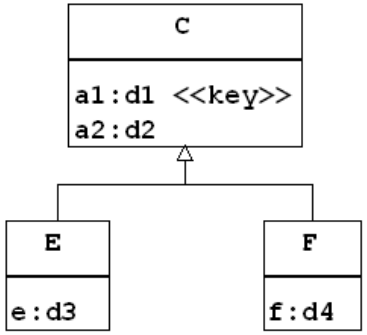
Übersetzung von allgemeinen Assoziationen




```
CREATE TABLE C ( ... )

CREATE TABLE D ( ... )

CREATE TABLE Assoc (
  roleC d1,
  roleD d3,
  PRIMARY KEY ( roleC, roleD ),
  FOREIGN KEY ( roleC ) REFERENCES C,
  FOREIGN KEY ( roleD ) REFERENCES D
)
```

Übersetzung von Generalisierungen (3 Varianten, 2 Varianten nur bedingt einsetzbar)

Schlüssel werden in der Generalisierungsklasse, nicht in den Spezialisierungsklassen angegeben



Erinnerung: In UML gibt es als Constraints für Generalisierungen {incomplete, complete} und {overlapping, disjoint}. Default: incomplete, overlapping.

(1) Generalisierungsklasse und Spezialisierungsklassen in getrennten Tabellen mit Fremdschlüsseln

```
CREATE TABLE C (
  a1 d1,
  a2 d2,
  PRIMARY KEY ( a1 )
)

CREATE TABLE E (
  a1 d1,
  e  d3,
  PRIMARY KEY ( a1 ),
  FOREIGN KEY ( a1 ) REFERENCES C
)

CREATE TABLE F (
  a1 d1,
  f  d4,
  PRIMARY KEY ( a1 ),
  FOREIGN KEY ( a1 ) REFERENCES C
)
```

(2) Generalisierungsklasse und Spezialisierungsklassen in einer Tabelle

Übersetzung nur bedingt einsetzbar!

Problem bei Transformation (2): Umgang mit Assoziationen auf Spezialisierungsklassen

```
CREATE TABLE C (
  a1 d1,
  a2 d2,
```



```
  e  d3 NULL,
  f  d4 NULL,
  PRIMARY KEY ( a1 )
)
```

```
                                          ( a1 , a2 , e   , f    )
Für Objekt o in C mit o nicht in F gilt: ( ..., ..., ... , NULL )
Für Objekt o in C mit o nicht in E gilt: ( ..., ..., NULL, ...  )
```

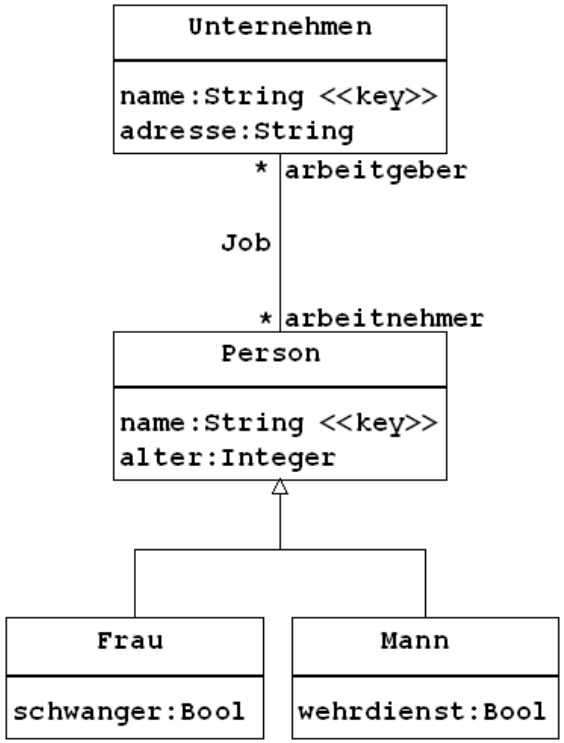
(3) Generalisierungsklasse und Spezialisierungsklassen in getrennten Tabellen mit Disjunktheitsforderung

Übersetzung nur bedingt einsetzbar!

Problem bei Transformation (3): Umgang mit Assoziationen auf Generalisierungsklasse

```
CREATE TABLE C (
  a1 d1,
  a2 d2,
  PRIMARY KEY ( a1 )
)

CREATE TABLE E (
  a1 d1,
  a2 d2,
  e  d3,
  PRIMARY KEY ( a1 )
)

CREATE TABLE F (
  a1 d1,
  a2 d2,
  f  d4,
  PRIMARY KEY ( a1 )
)

Für Objekt o in E gilt: o nicht in C
Für Objekt o in F gilt: o nicht in C
Für Objekt o in C gilt: o nicht in E oder F

Äquivalent:

SELECT * FROM E
WHERE EXISTS (SELECT * FROM C WHERE E.a1=C.a1)
= EMPTY

SELECT * FROM F
WHERE EXISTS (SELECT * FROM C WHERE F.a1=C.a1)
= EMPTY

SELECT * FROM C
WHERE EXISTS (SELECT * FROM E WHERE C.a1=E.a1) OR
      EXISTS (SELECT * FROM F WHERE C.a1=F.a1)
= EMPTY
```

Beispiel




```
CREATE TABLE unternehmen (
  name VARCHAR,
  adresse VARCHAR,
  PRIMARY KEY ( name )
)

CREATE TABLE job (
  arbeitgeber VARCHAR,
  arbeitnehmer VARCHAR,
  PRIMARY KEY ( arbeitgeber, arbeitnehmer ),
  FOREIGN KEY ( arbeitgeber ) REFERENCES unternehmen,
  FOREIGN KEY ( arbeitnehmer ) REFERENCES person,
)

(1)

CREATE TABLE person (
  name VARCHAR,
  alter VARCHAR,
  PRIMARY KEY ( name )
)
```



```
)

CREATE TABLE frau (
  name VARCHAR,
  schwanger BOOLEAN,
  PRIMARY KEY ( name ),
  FOREIGN KEY ( name ) REFERENCES person
)

CREATE TABLE mann (
  name VARCHAR,
  wehrdienst BOOLEAN,
  PRIMARY KEY ( name ),
  FOREIGN KEY ( name ) REFERENCES person
)

(2)

CREATE TABLE person (
  name VARCHAR,
  alter VARCHAR,
  schwanger BOOLEAN NULL,
  wehrdienst BOOLEAN NULL,
  PRIMARY KEY ( name )
)

Nachteilig an dieser Lösung ist, dass Attribute teilweise unnötig
geführt werden. Ferner bleibt die Behandlung von Assoziationen auf die
Klassen Frau oder Mann offen.

(3)

CREATE TABLE person (
  name VARCHAR,
  alter VARCHAR,
  PRIMARY KEY ( name )
)

CREATE TABLE frau (
  name VARCHAR,
  alter VARCHAR,
  schwanger BOOLEAN,
  PRIMARY KEY ( name ),
)

CREATE TABLE mann (
  name VARCHAR,
  alter VARCHAR,
  wehrdienst BOOLEAN,
  PRIMARY KEY ( name ),
)

Die Tabellen person, mann und frau sollten disjunkte Sachverhalte
wiedergeben, denn das Attribut alter wird in allen drei Tabellen
geführt. Bei Nichtdisjunktheit besteht die Möglichkeit der
Inkonsistenz. Nachteilig an dieser Lösung ist, dass die obige Tabelle
job aber nur auf die Tabelle person, nicht aber auf frau oder mann
```

verweist.

Relationaler Datenbankentwurf

Grundidee: Zerlegung von Tabellen

```
SKRIPT(VeranstNr, PersonalNr, Name, SkriptNr, Preis)

VeranstNr, PersonalNr -> Name, SkriptNr, Preis
PersonalNr -> Name
SkriptNr -> Preis

SKRIPT aufteilen wegen PersonalNr -> Name:
DOZENT(PersonalNr, Name)
SKRIPT2(VeranstNr, PersonalNr, SkriptNr, Preis)

SKRIPT2 aufteilen wegen SkriptNr -> Preis:
SKRIPT3(VeranstNr, PersonalNr, SkriptNr)
SKRIPTPREIS(SkriptNr, Preis)

Resultat:
DOZENT(PersonalNr, Name)
SKRIPT3(VeranstNr, PersonalNr, SkriptNr)
SKRIPTPREIS(SkriptNr, Preis)

Resultat erweitert um Fremdschlüssel:
DOZENT(PersonalNr, Name)
SKRIPT3(VeranstNr,
  PersonalNr->DOZENT.PersonalNr, SkriptNr->SKRIPTPREIS.SkriptNr)
SKRIPTPREIS(SkriptNr, Preis)

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

Skript VeranstNr PersonalNr Name   SkriptNr Preis
       112       198        Schulz 2        25
       112       237        Lange  9        44
       112       011        Meyer  2        25
       202       198        Schulz 4        22

Redundanz: VeranstNr PersonalNr Name   ... [PersonalNr -> Name]
           112       198        Schulz
           202       198        Schulz

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

DOZENT PersonalNr  Name
       198         Schulz
       237         Lange
       011         Meyer
```



```
Skript2 VeranstNr PersonalNr SkriptNr Preis
        112       198        2        25
        112       237        9        44
        112       011        2        25
        202       198        4        22

Redundanz: ... PersonalNr SkriptNr Preis [SkriptNr -> Preis]
               198        2        25
               011        2        25

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

DOZENT PersonalNr  Name
       198         Schulz
       237         Lange
       011         Meyer

Skript3 VeranstNr PersonalNr SkriptNr
        112       198        2
         112       237        9
        112       011        2
        202       198        4

SkriptPREIS SkriptNr Preis
            2        25
            9        44
            4        22

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

Funktionale Abhängigkeiten und Normalformen


Zentrale Begriffe im relationalen Datenbankentwurf: Funktionale
Abhängigkeit, Schlüssel, partielle Abhängigkeit, transitive
Abhängigkeit, Zerlegung von Relationenschemata in Normalform (NF),
1NF, 2NF, 3NF

Gegeben sei ein (oder mehrere) Relationenschema R(A1, ..., An) über
einer Attributmenge A={A1, ..., An}. Eine funktionale Abhängigkeit
(FA) ist ein Paar von Attributmengen notiert als X -> Y (X und Y
Teilmengen von A).

Zentraler Schritt im Entwurf: Alle funktionalen Abhängigkeiten über
der Attributmenge finden. Also fest vorgeben: Menge F von funktionalen
Abhängigkeiten. Gegenenfalls: Aufdecken aller funktionalen
Abhängigkeiten durch systematisches Untersuchen aller Teilmengen X und
Y von A.

Beispiel: SKRIPT(VeranstNr, PersonalNr, Name, SkriptNr, Preis)
F1: VeranstNr, PersonalNr -> Name, SkriptNr, Preis
    PersonalNr -> Name
    SkriptNr -> Preis

Jede funktionale Abhängigkeit X -> Y schränkt die erlaubten
Datenbankzustände ein: Wenn in R zwei Tupel in X übereinstimmen,
```

müssen sie auch in Y übereinstimmen. Anders ausgedrückt: X bestimmt Y, oder X legt Y fest. Eine funktionale Abhängigkeit ist also eine spezielle Integritätsbedingung. Die Integritätsbedingung verbietet Tupel t1 und t2 mit t1.X=t2.X und t1.Y<>t2.Y.

Beispiel: Durch SkriptNr -> Preis wird z.B. verboten:

Skript	VeranstNr	PersonalNr	Name	SkriptNr	Preis
	112	198	Schulz	2	25
	112	237	Lange	2	44

Ein Schlüssel S bezüglich R und F ist eine Attributmenge S (Teilmenge von A), so daß der Schlüssel S die Gesamtattributmenge bestimmt (S -> A) und S minimal ist, d.h. aus S darf nichts weggelassen werden. Ein Nicht-Schlüsselattribut ist ein Attribut das in *keinem* Schlüssel vorkommt.

Beispiel: VeranstNr, PersonalNr ist Schlüssel hinsichtlich F1:
Mit VeranstNr, PersonalNr -> Name, SkriptNr, Preis und
VeranstNr, PersonalNr -> VeranstNr, PersonalNr ergibt sich
VeranstNr, PersonalNr -> Name, SkriptNr, Preis, VeranstNr, PersonalNr
Aus der Menge VeranstNr, PersonalNr darf nichts weggelassen werden, andernfalls würde VeranstNr, PersonalNr -> SkriptNr nicht gelten.

Beispiel: VeranstNr, PersonalNr, Name ist kein Schlüssel hinsichtlich F1, da zwar die Gesamtattributmenge bestimmt wird, die Menge aber nicht minimal ist, denn Name kann weggelassen werden.

Beispiel: Die Menge bestehend nur aus VeranstNr ist kein Schlüssel, da z.B. nicht gilt VeranstNr -> SkriptNr. Ebenso wäre die Menge bestehend nur aus PersonalNr kein Schlüssel.

Beispiel: SKRIPT(VeranstNr, PersonalNr, Name, SkriptNr, Preis)
F2: VeranstNr, PersonalNr -> SkriptNr
    PersonalNr -> Name
    SkriptNr -> Preis
Die Menge VeranstNr, PersonalNr ist ein Schlüssel hinsichtlich F2. Aus der FA-Menge F1 und der FA-Menge F2 ergeben sich die gleichen funktionalen Abhängigkeiten als Konsequenzen.

Beispiel: STADT(SName,XKoor,YKoor,EinwAnz)
F3: SName -> XKoor, YKoor, EinwAnz
    XKoor, YKoor -> SName
Es gibt 2 Schlüssel
- SName
- XKoor, YKoor
und ein Nicht-Schlüsselattribut
- EinwAnz

Ein Nicht-Schlüsselattribut N ist partiell abhängig von einem Schlüssel S, wenn es eine echte Teilmenge Y von S gibt mit Y -> N und N nicht Element von Y.

Ein Nicht-Schlüsselattribut N ist transitiv abhängig von einem Schlüssel S, wenn es eine Teilmenge Y von A gibt mit S -> Y, Y -> N, N nicht Element von Y und nicht (Y -> S).

Ein Relationenschema ist bezüglich einer gegebenen Menge F von

funktionalen Abhängigkeiten in
- erster Normalform (1NF), wenn alle Attributwerte atomar sind, d.h. ein Attributwert nicht eine Menge von Werten darstellt,
- zweiter Normalform (2NF), wenn es keine partiellen Abhängigkeiten eines Nicht-Schlüsselattributs von einem Schlüssel gibt, und
- dritter Normalform (3NF), wenn es keine transitiven Abhängigkeiten eines Nicht-Schlüsselattributs von einem Schlüssel gibt.

Beispiel:
SKRIPT(VeranstNr, PersonalNr, Name, SkriptNr, Preis)
  mit VeranstNr, PersonalNr -> Name, SkriptNr, Preis
      PersonalNr -> Name
      SkriptNr -> Preis
Schlüssel: VeranstNr, PersonalNr
Nicht-Schlüsselattribute: Name, SkriptNr, Preis
Nicht in 2NF wegen PersonalNr -> Name
Nicht in 3NF wegen VeranstNr, PersonalNr -> SkriptNr -> Preis

Beispiel:
SKRIPT(VPId, VeranstNr, PersonalNr, Name, SkriptNr, Preis)
  mit VPId -> VeranstNr, PersonalNr, Name, SkriptNr, Preis
Schlüssel: VPId
Nicht-Schlüsselattribute: VeranstNr, PersonalNr, Name, SkriptNr, Preis
Relationenschema inklusive der FAen ist zwar in 2NF und 3NF, aber nur weil die FAen *nicht* vollständig angegeben worden sind.

Beispiel:
DOZENT(PersonalNr, Name)
  mit PersonalNr -> Name
SKRIPT3(VeranstNr, PersonalNr, SkriptNr)
  mit VeranstNr PersonNr -> SkriptNr
SKRIPTPREIS(SkriptNr, Preis)
  mit SkriptNr -> Preis
Jedes Relationenschema ist bzgl. der angegebenen FAen in 3NF.

Beispiel: STADT(SName,XKoor,YKoor,EinwAnz)
F3: SName -> XKoor, YKoor, EinwAnz
    XKoor, YKoor -> SName
Ist in 3NF obwohl XKoor, YKoor -> SName -> EinwAnz gilt
(Es gilt SName -> XKoor, YKoor)
F4: SName -> XKoor, Ykoor
    XKoor, YKoor -> SName, EinwAnz
F3 und F4 haben die gleichen funktionalen Abhängigkeiten als Konsequenzen.

More complex example: Persons and their children

A person has a first and a last name and a gender indicated by 'F' for female or 'M' for male. A person can have other another person as a mother or as a father. Only female persons can play the mother role, only male persons the father role. A female person cannot be mother of herself. Analogously for male persons: A male person cannot be father

of himself. The following cycle involving two persons is not allowed: There are two distinct persons p1 and p2 where p1 is the mother of p2 and p2 is the mother of p1; analogously for fathers. An analogous requirement for cycles with three persons has to be guaranteed as well.

Desirable: Exclude cycles with arbitrary lenght; not possible in SQL-86, SQL-89, or SQL-92; possible in SQL:1999 with recursive query feature

Table definition

```
CREATE TABLE person (
  fname VARCHAR NOT NULL,
  lname VARCHAR NOT NULL,
  gender CHAR(1) NOT NULL,
  motherfname VARCHAR NULL,
  motherlname VARCHAR NULL,
  fatherfname VARCHAR NULL,
  fatherlname VARCHAR NULL,
  CONSTRAINT fname_lname_key
    PRIMARY KEY ( fname, lname ),
  CONSTRAINT mother_fname_lname_foreign_key
    FOREIGN KEY ( motherfname, motherlname ) REFERENCES person,
  CONSTRAINT father_fname_lname_foreign_key
    FOREIGN KEY ( fatherfname, fatherlname ) REFERENCES person
);

> NOTICE: CREATE TABLE / PRIMARY KEY will create implicit index
> "fname_lname_key" for table "person"
> CREATE TABLE
```

Remark: Responses from PostgreSQL are indicated by '> '.

Constraints on name format and gender

```
CREATE FUNCTION name_format(TEXT) RETURNS BOOLEAN AS $$
  (SELECT NOT ( $1='' OR
          $1 LIKE '%0%' OR $1 LIKE '%1%' OR $1 LIKE '%2%' OR
          $1 LIKE '%3%' OR $1 LIKE '%4%' OR $1 LIKE '%5%' OR
          $1 LIKE '%6%' OR $1 LIKE '%7%' OR $1 LIKE '%8%' OR
          $1 LIKE '%9%' OR $1 LIKE '% %' OR $1 LIKE '%/_%' ) )
$$ LANGUAGE SQL;

> CREATE FUNCTION

ALTER TABLE person ADD CONSTRAINT fname_format
  CHECK ( name_format(fname) );

> ALTER TABLE

ALTER TABLE person ADD CONSTRAINT lname_format
  CHECK ( name_format(lname) );

> ALTER TABLE

ALTER TABLE person ADD CONSTRAINT gender_female_or_male
  CHECK ( gender='F' OR gender='M' );

> ALTER TABLE
```

Constraints no_mother_cycle

```
CREATE FUNCTION cycle1(TEXT,TEXT,TEXT,TEXT) RETURNS BOOLEAN AS $$
  (SELECT $1=$3 AND $2=$4)
$$ LANGUAGE SQL;

> CREATE FUNCTION


-- Forbid:
-- new [ fname lname fname     lname     ...       ...       ]
--       FNAME LNAME MOTHFNAME MOTHLNAME FATHFNAME FATHLNAME

ALTER TABLE person ADD CONSTRAINT no_mother_cycle1
--CHECK ( NOT (fname,lname)=(motherfname,motherlname) );
  CHECK ( NOT cycle1(fname,lname,motherfname,motherlname ) );

> ALTER TABLE

CREATE FUNCTION mother_cycle2(TEXT,TEXT,TEXT,TEXT) RETURNS BOOLEAN AS
$$
  (SELECT EXISTS (SELECT * FROM person p
                  WHERE p.motherfname=$1 AND p.motherlname=$2 AND
                        p.fname=$3 AND p.lname=$4) )
$$ LANGUAGE SQL;

> CREATE FUNCTION

-- Forbid:
-- old [ mothfname mothlname fname     lname     ... ... ]
-- new [ fname     lname     mothfname mothlname ... ... ]

ALTER TABLE person ADD CONSTRAINT no_mother_cycle2
  CHECK ( NOT mother_cycle2(fname,lname,motherfname,motherlname) );

> ALTER TABLE

CREATE FUNCTION mother_cycle3(TEXT,TEXT,TEXT,TEXT) RETURNS BOOLEAN AS
$$
  (SELECT EXISTS
    (SELECT * FROM person p1, person p2
     WHERE p1.motherfname=$1 AND p1.motherlname=$2 AND
           p2.motherfname=p1.fname AND p2.motherlname=p1.lname AND
           p2.fname=$3 AND p2.lname=$4) )
$$ LANGUAGE SQL;
```

```
> CREATE FUNCTION

-- Forbid:
-- old1 [ xxx       yyy       fname     lname     ... ... ]
-- old2 [ mothfname mothlname xxx       yyy       ... ... ]
-- new  [ fname     lname     mothfname mothlname ... ... ]

ALTER TABLE person ADD CONSTRAINT no_mother_cycle3
  CHECK ( NOT mother_cycle3(fname,lname,motherfname,motherlname) );

> ALTER TABLE
```

Constraints no_father_cycle

```
ALTER TABLE person ADD CONSTRAINT no_father_cycle1
--CHECK ( NOT (fname,lname)=(fatherfname,fatherlname) );
  CHECK ( NOT cycle1(fname,lname,fatherfname,fatherlname) );

> ALTER TABLE

CREATE FUNCTION father_cycle2(TEXT,TEXT,TEXT,TEXT) RETURNS BOOLEAN AS
$$
  (SELECT EXISTS (SELECT * FROM person p
                  WHERE p.fatherfname=$1 AND p.fatherlname=$2 AND
                        p.fname=$3 AND p.lname=$4) )
$$ LANGUAGE SQL;

> CREATE FUNCTION

ALTER TABLE person ADD CONSTRAINT no_father_cycle2
  CHECK ( NOT father_cycle2(fname,lname,fatherfname,fatherlname) );

> ALTER TABLE

CREATE FUNCTION father_cycle3(TEXT,TEXT,TEXT,TEXT) RETURNS BOOLEAN AS
$$
  (SELECT EXISTS
    (SELECT * FROM person p1, person p2
     WHERE p1.fatherfname=$1 AND p1.fatherlname=$2 AND
           p2.fatherfname=p1.fname AND p2.fatherlname=p1.lname AND
           p2.fname=$3 AND p2.lname=$4) )
$$ LANGUAGE SQL;

> CREATE FUNCTION

ALTER TABLE person ADD CONSTRAINT no_father_cycle3
  CHECK ( NOT father_cycle3(fname,lname,fatherfname,fatherlname) );

> ALTER TABLE
```

Constraint mother_is_female


```
CREATE FUNCTION mother_is_female(TEXT,TEXT) RETURNS BOOLEAN AS $$
  SELECT NOT EXISTS (SELECT * FROM person
                     WHERE fname=$1 and lname=$2 AND gender='M') AND
         NOT EXISTS (SELECT * FROM person
                     WHERE fatherfname=$1 AND fatherlname=$2)
$$ LANGUAGE SQL;

> CREATE FUNCTION

ALTER TABLE person ADD CONSTRAINT mother_is_female
  CHECK ( (motherfname IS NULL AND motherlname IS NULL) OR
          mother_is_female(motherfname,motherlname) );

> ALTER TABLE

CREATE FUNCTION not_father(TEXT,TEXT) RETURNS BOOLEAN AS $$
  SELECT NOT EXISTS (SELECT * FROM person
                     WHERE fatherfname=$1 AND fatherlname=$2)
$$ LANGUAGE SQL;

> CREATE FUNCTION

ALTER TABLE person ADD CONSTRAINT female_not_father
  CHECK (NOT(gender='F') OR not_father(fname,lname));
--CHECK (gender='F' IMPLIES not_father(fname,lname));

> ALTER TABLE
```

Constraint father_is_male

```
CREATE FUNCTION father_is_male(TEXT,TEXT) RETURNS BOOLEAN AS $$
  SELECT NOT EXISTS (SELECT * FROM person
                     WHERE fname=$1 and lname=$2 AND gender='F') AND
         NOT EXISTS (SELECT * FROM person
                     WHERE motherfname=$1 AND motherlname=$2)
$$ LANGUAGE SQL;

> CREATE FUNCTION

ALTER TABLE person ADD CONSTRAINT father_is_male
  CHECK ( (fatherfname IS NULL AND fatherlname IS NULL) OR
          father_is_male(fatherfname,fatherlname) );

> ALTER TABLE

CREATE FUNCTION not_mother(TEXT,TEXT) RETURNS BOOLEAN AS $$
  SELECT NOT EXISTS (SELECT * FROM person
                     WHERE motherfname=$1 AND motherlname=$2)
$$ LANGUAGE SQL;

> CREATE FUNCTION

ALTER TABLE person ADD CONSTRAINT male_not_mother
  CHECK (NOT(gender='M') OR not_mother(fname,lname));
--CHECK (gender='M' IMPLIES not_mother(fname,lname));
```

```
> ALTER TABLE
```

Overview on specified constraints

```
CONSTRAINT fname_lname_key
CONSTRAINT mother_fname_lname_foreign_key
CONSTRAINT father_fname_lname_foreign_key

CONSTRAINT fname_format
CONSTRAINT lname_format
CONSTRAINT gender_female_or_male

CONSTRAINT no_mother_cycle1
CONSTRAINT no_mother_cycle2
CONSTRAINT no_mother_cycle3

CONSTRAINT no_father_cycle1
CONSTRAINT no_father_cycle2
CONSTRAINT no_father_cycle3

CONSTRAINT mother_is_female
CONSTRAINT female_not_father

CONSTRAINT father_is_male
CONSTRAINT male_not_mother
```

Test scenarios I

```
INSERT INTO person VALUES ('Ada','Black','F','Ada','Black',NULL,
NULL);

> ERROR: new row for relation "person" violates check constraint
> "no_mother_cycle1"

INSERT INTO person VALUES ('Bea','Black','F',NULL, NULL,  NULL,
NULL);

> INSERT 0 1

INSERT INTO person VALUES ('Cia','Black','F','Bea','Black',NULL,
NULL);

> INSERT 0 1

UPDATE person SET motherfname='Cia', motherlname='Black'
WHERE fname='Bea' AND lname='Black';

> ERROR: new row for relation "person" violates check constraint
> "no_mother_cycle2"

INSERT INTO person VALUES ('Ada','Black','F',NULL, NULL, NULL, NULL);

> INSERT 0 1
```



```
UPDATE person SET motherfname='Ada', motherlname='Black'
WHERE fname='Bea' AND lname='Black';

> UPDATE 1

UPDATE person SET motherfname='Cia', motherlname='Black'
WHERE fname='Ada' AND lname='Black';

> ERROR: new row for relation "person" violates check constraint
> "no_mother_cycle3"

INSERT INTO person VALUES ('Bea','Black','F',NULL, NULL, NULL, NULL );

> ERROR: duplicate key violates unique constraint "fname_lname_key"

INSERT INTO person VALUES ('Cyd','Black','M',NULL, NULL, NULL, NULL );

> INSERT 0 1

INSERT INTO person
VALUES ('Ada','Black','F','Bea','Black','Cyd','Black');

> ERROR: new row for relation "person" violates check constraint
> "no_mother_cycle2"

INSERT INTO person VALUES ('Dan','Green','M',NULL, NULL, NULL, NULL );

> INSERT 0 1

INSERT INTO person
VALUES ('Eva','Black','F','Ada','Black','Dan','Green');

> INSERT 0 1

UPDATE person SET motherfname='Eva', motherlname='Black'
WHERE fname='Bea' AND lname='Black';

> UPDATE 1

UPDATE person SET gender='F' WHERE fname='Dan' AND lname='Green';

> ERROR: new row for relation "person" violates check constraint
> "female_not_father"

SELECT * FROM person ORDER BY 1;

 fname | lname | gender | mothfname | mothlname | fathfname |
fathlname
-------+-------+--------+-----------+-----------+-----------+-----------
 Ada   | Black | F      |           |           |           |
 Bea   | Black | F      | Eva       | Black     |           |
 Cia   | Black | F      | Bea       | Black     |           |
 Cyd   | Black | M      |           |           |           |
 Dan   | Green | M      |           |           |           |
 Eva   | Black | F      | Ada       | Black     | Dan       | Green
(6 rows)
```

```
Remark: Column names shortened for layout reasons

-- Ada Black   Dan Green   Cyd Black
--     \         /
--      Eva Black
--          |
--      Bea Black
--          |
--      Cia Black
```

Test scenarios II

```
INSERT INTO person
VALUES ('Flo','Black','W','Bea','Black','Cyd','Black');

> ERROR: new row for relation "person" violates check constraint
> "gender_female_or_male"

INSERT INTO person
VALUES (Null, 'Black','F','Bea','Black','Cyd','Black');

> ERROR: null value in column "fname" violates not-null constraint

INSERT INTO person
VALUES ('Ada','Black','M','Bea','Black','Cyd','Black');

> ERROR: new row for relation "person" violates check constraint
> "male_not_mother"

INSERT INTO person
VALUES ('','Black','M','Bea','Black','Cyd','Black');

> ERROR: new row for relation "person" violates check constraint
> "fname_format"

INSERT INTO person
VALUES ('Eva','Black','F','Eva','Black','Cyd','Black');

> ERROR: new row for relation "person" violates check constraint
> "no_mother_cycle1"

INSERT INTO person
VALUES ('Eva','Black','F','Bea','Black','Eva','Black');

> ERROR: new row for relation "person" violates check constraint
> "father_is_male"
```

Overview on occured errors (in scenarios)

```
ERROR: new row for relation "person" violates check constraint
"no_mother_cycle1"

ERROR: new row for relation "person" violates check constraint
```



```
"no_mother_cycle2"

ERROR: new row for relation "person" violates check constraint
"no_mother_cycle3"

ERROR: duplicate key violates unique constraint
"fname_lname_key"

ERROR: new row for relation "person" violates check constraint
"no_mother_cycle2"

ERROR: new row for relation "person" violates check constraint
"female_not_father"

ERROR: new row for relation "person" violates check constraint
"gender_female_or_male"

ERROR: null value in column "fname" violates
not-null constraint

ERROR: new row for relation "person" violates check constraint
"male_not_mother"

ERROR: new row for relation "person" violates check constraint
"fname_format"

ERROR: new row for relation "person" violates check constraint
"no_mother_cycle1"

ERROR: new row for relation "person" violates check constraint
"father_is_male"
```