

# **DBI Zusammenfassung**

Martin Linhard

May 23, 2022



# Contents

<b>1</b>	<b>Themenkorb 1 - Konzeptionelles Datenbankdesign</b>	<b>5</b>
1.1	ER-Modell . . . . .	5
1.2	ER-Diagramm (ERD) . . . . .	5
1.2.1	Entity Typen . . . . .	5
1.2.2	Beziehungen . . . . .	5
<b>2</b>	<b>Themenkorb - Information Retrieval</b>	<b>7</b>
2.1	SQL . . . . .	7
2.1.1	Reihenfolge der Ausführung . . . . .	7
2.1.2	Befehle . . . . .	7
2.1.3	Wichtige Funktionen . . . . .	7
2.1.4	Joins . . . . .	9
2.1.5	Subselects . . . . .	12
2.1.6	Andere, wichtige Keywords . . . . .	13
2.1.7	Indizes . . . . .	14
2.1.8	Hierarchisches SQL . . . . .	15
<b>3</b>	<b>Themenkorb - Relationales Datenbankmodell</b>	<b>17</b>
3.1	DDL . . . . .	17
3.1.1	Datentypen . . . . .	17
3.1.2	Constraints . . . . .	17
3.1.3	Tabellen im Nacheinander bearbeiten . . . . .	18
3.2	DML . . . . .	19
3.2.1	Views . . . . .	19
3.2.2	Sequences . . . . .	19
3.2.3	MERGE . . . . .	19
3.3	Normalisierung . . . . .	19
3.3.1	Normalformen . . . . .	19
3.3.2	Anwendung - Zirkelbezug . . . . .	21
3.3.3	Abhängigkeitsdiagramm - Beispiele . . . . .	21



# 1 Themenkorb 1 - Konzeptionelles Datenbankdesign

## 1.1 ER-Modell

- ER  $\Rightarrow$  Entity Relationship

## 1.2 ER-Diagramm (ERD)

### 1.2.1 Entity Typen

- Fundamental  $\Rightarrow$  Unabhängig von anderen
- Attributiv  $\Rightarrow$  Abhängig von genau einer anderen Entity
- Assoziativ  $\Rightarrow$  Abhängig von mindestens 2 anderen Entities

### 1.2.2 Beziehungen

- 1:1
- 1:n
- n:m

**Übung macht den Meister!**



## 2 Themenkorb - Information Retrieval

### 2.1 SQL

#### 2.1.1 Reihenfolge der Ausführung

1. FROM
2. WHERE
3. GROUP BY
4. HAVING
5. SELECT / ORDER BY
  - Es ist hier nicht ganz klar, was zuerst ausgeführt wird!

#### 2.1.2 Befehle

##### GROUP BY

- Wenn eine "normale" Spalte neben einer Gruppenfunktion im SELECT steht, muss diese "normale" Spalte im Group By enthalten sein!
  - Das Gruppen-Statement (z.B. MAX) wird dann für jeden unterschiedlichen Wert der "normalen" Spalte ausgeführt!
    - \* z.B. für jede Abteilungsnummer, wenn danach gruppiert wird!

```
SELECT deptno AS "Department", AVG(sal) "Average"
FROM emp
GROUP BY deptno;
```

##### HAVING

- Wird verwendet, wenn man das Ergebnis einer Gruppenfunktion als Bedingung haben möchte
  - z.B. Durchschnittsgehalt aller Jobs, die ein durchschnittliches Gehalt > 1500 haben:

```
SELECT job, ROUND( AVG(sal),2 ) "Average Salary"
FROM emp
GROUP BY job
HAVING AVG(sal) > 1500;
```

#### 2.1.3 Wichtige Funktionen

##### Case / Character

- LOWER / UPPER
- INITCAP  $\implies$  Erster Buchstabe wird groß geschrieben!

- SUBSTR(string, start, length)
  - Substring ab *start* mit Länge von *length*
- LENGTH  $\implies$  Länge des Strings
- LPAD / RPAD(column, length, 'ValueUsedForPadding')
- TRIM(string)  $\implies$  Löscht Whitespaces an beiden Enden
  - TRIM(string1, string2)  $\implies$  Trimmt string2 von string1 (am Anfang und am Ende)
- REPLACE(input, toBeReplaced, replaceWith)  $\implies$  Ersetzt in Input den 2. String mit dem 3.

### Number

- ROUND(number, decimalPlaces)  $\implies$  Rundet *number* auf *decimalPlaces* Nachkommastellen
- TRUNC(number, decimalPlaces)  $\implies$  Schneidet *number* nach *decimalPlaces* Stellen ab
- MOD(number1, number2)  $\implies$  number1 % number2

### Date

- MONTHS\_BETWEEN(date1, date2)  $\implies$  Anzahl der Monate dazwischen
- ADD\_MONTHS(date, numberOfMonths)  $\implies$  Fügt *numberOfMonths* Monate zu *date* hinzu
- NEXT\_DAY(date, 'Day')  $\implies$  Gibt den nächsten Wochentag *nach* diesem Datum mit dem gewählten Namen zurück
- ROUND(date, ['MONTH' — 'YEAR'])
  - Rundet Auf das nächste / vorherige Jahr / Monat auf / ab
- TRUNC(date, ['MONTH' — 'YEAR'])
  - Setzt das Datum auf den 1. des Monats / Jahres

### Conversion

- TO\_CHAR(columnWithDate — columnWithNumber, 'Format')
- TO\_NUMBER(input, 'Format')
  - String zu Zahl parsen
- TO\_DATE()
  - String zu Datum parsen



YYYY	Full year in numbers
YEAR	Year spelled out
MM	Two-digit value for month
MONTH	Full name of the month
MON	Three-letter abbreviation of the month
DY	Three-letter abbreviation of the day of the week
DAY	Full name of the day of the week
DD	Numeric day of the month

HH24:MI:SS AM	15:45:32 PM
DD "of" MONTH	12 of October

DDspth	FOURTEENTH
Ddspth	Fourteenth
ddspth	fourteenth
DDD or DD or D	Day of year, month or week

### Multi row

- MAX, MIN
- COUNT
- AVG
- SUM
- (STDDEV, VARIANCE)

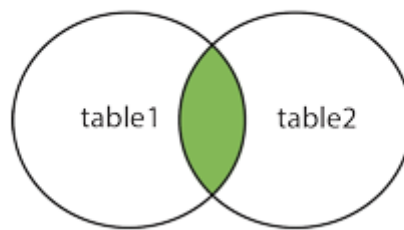
### 2.1.4 Joins

- Entweder mit ON oder mit USING
  - INNER JOIN DEPT D ON EMP.DEPTNO = D.DEPTNO;  $\implies$  Beide Spalten werden ausgegeben!
  - INNER JOIN DEPT D USING(DEPTNO);  $\implies$  Spalte muss in beiden Tables gleich heißen, wird nur 1x ausgegeben!

### INNER JOIN

- Inkludiert nur Zeilen, die beiden Tables gleich sind!

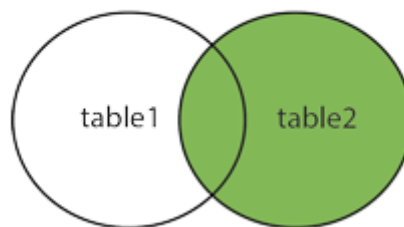
### INNER JOIN



### RIGHT OUTER JOIN

- Inkludiert alle Zeilen der rechten Tabelle (= die Tabelle, auf die gejoint wird) und Werte, die in beiden Tabellen gleich sind

### RIGHT JOIN



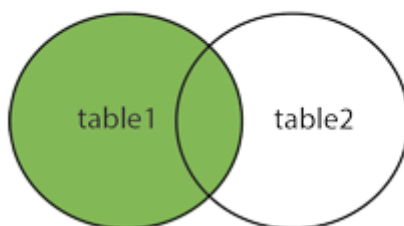
– Beispiel: Gib jene Abteilungen aus, die keine Mitarbeiter haben:

```
SELECT DISTINCT d.*  
FROM emp e  
RIGHT OUTER JOIN dept d ON e.DEPTNO = d.DEPTNO  
WHERE e.DEPTNO IS NULL;
```

### LEFT OUTER JOIN

- Inkludiert alle Zeilen der linken Tabelle (= die Tabelle, von der weg gejoint wird) und Werte, die in beiden Tabellen gleich sind

### LEFT JOIN



– Beispiel: Gib jene Abteilungen aus, die keine Mitarbeiter haben:

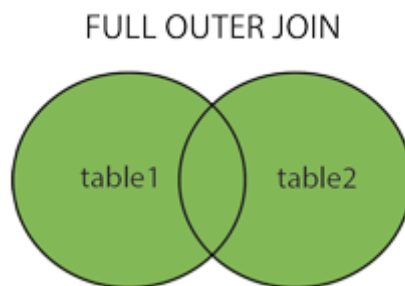
```

SELECT DISTINCT d.*
FROM dept d
LEFT OUTER JOIN emp e ON e.deptno = d.deptno
WHERE e.deptno IS NULL;

```

## FULL OUTER JOIN

- Inkludiert alle Zeilen der linken Tabelle (= die Tabelle, von der weg gejoint wird) und alle Werte aus der rechten Tabelle



– Beispiel: Gib alle Mitarbeiter und Abteilungen aus

```

SELECT e.ename, d.deptno
FROM emp e
FULL OUTER JOIN dept d ON e.deptno = d.deptno;

```

## CROSS JOIN

- Gibt jede Zeile in einer Tabelle mit jeder Zeile aus einer anderen aus
- Problem: Auch jede Zeile mit sich selbst!

```

SELECT a.teamname, b.teamname, c.teamname
FROM teamA a
CROSS JOIN teamB b
CROSS JOIN teamC c;


```

## SELF JOIN

- Es wird nochmal auf den gleichen Table gejoint (z.B. um den Vorgesetzten zu bestimmen)

## NATURAL JOIN

- Spalten, die beide Tabellen beinhalten werden nur 1x zurückgegeben!
- "Automatischer Inner Join"  $\implies$  Es werden nur Spalten zurückgegeben, die den gleichen Wert haben (kein NULL!)
- Es wird AUF ALLE GLEICH BENANNTEN SPALTEN IN BEIDEN TABELLEN gejoint!
  - Wenn eine neue Spalte hinzugefügt wird, welche zufällig so wie eine existierende heißt, werden nur Werte zurückgegeben, bei denen diese Spalten übereinstimmen!



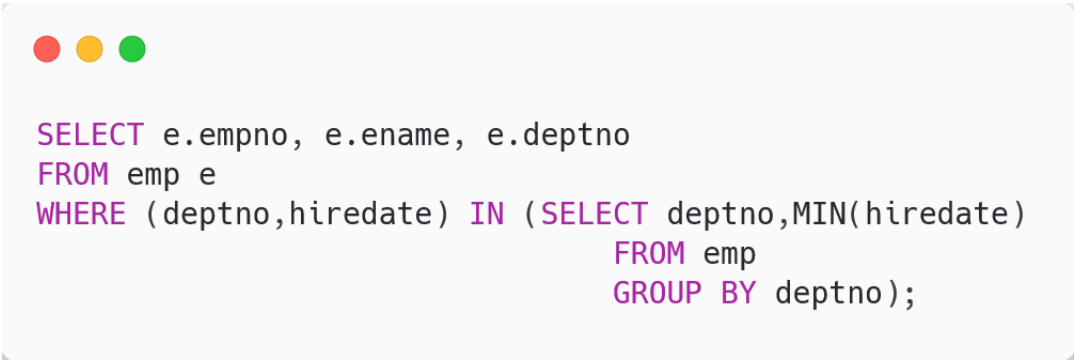
```
SELECT e.ename, d.loc
FROM emp e
NATURAL JOIN dept d;
```

## EQUI / NON-EQUI Joins

- EQUI  $\implies$  =
- NON-EQUI  $\implies$  Alles andere (Größer / Kleiner, Between and...)

### 2.1.5 Subselects

- Können in der WHERE, HAVING und FROM Klausel vorkommen
- Kann kein ORDER BY beinhalten
- Können eine (Single Row) oder mehrere (Multi-Row) Zeilen zurückliefern
  - Single Row  $\implies$  =, <, >, ...
- Wenn mehrere Werte aus dem Subselect zurückgegeben werden  $\implies$  IN muss verwendet werden:



```
SELECT e.empno, e.ename, e.deptno
FROM emp e
WHERE (deptno,hiredate) IN (SELECT deptno,MIN(hiredate)
                           FROM emp
                           GROUP BY deptno);
```

### Multiple-Row Subselects


- Es müssen spezielle Operatoren verwendet werden:
  - IN  $\implies$  Es werden nur Zeilen zurückgegeben, dessen Wert in der Ergebnisliste des Subselects enthalten ist.

- ANY/SOME  $\implies$  Ein Wert muss =, <, > als irgendein Wert in der Ergebnisliste sein
- ALL  $\implies$  Ein Wert muss =, <, > als alle Werte in der Ergebnisliste sein
- Correlation  $\implies$  Es werden Werte von "Außen" in einer Subquery verwendet

### 2.1.6 Andere, wichtige Keywords

#### UNION


- Der Output von 2 SQL-Statements kann verbunden werden
- **UNION ALL**  $\implies$  Macht das gleiche, doppelte Werte werden allerdings angezeigt!
- Wichtig: Anzahl der Spalte + Datentypen müssen gleich sein, doppelte Werte werden ignoriert!



```
SELECT date
FROM store_info
UNION
SELECT date
FROM internet_sales;
```

#### INTERSECT


- Gibt nur Werte aus, die in beiden Statements vorhanden sind!



```
SELECT date
FROM store_info
INTERSECT
SELECT date
FROM internet_sales;
```

#### MINUS

- Gibt nur Werte aus, die in dem ersten Statement, nicht aber in dem 2. vorkommen!



```
SELECT date
FROM store_info
MINUS
SELECT date
FROM internet_sales;
```

### 2.1.7 Indizes

- Kann auf eine / mehrere (Composite Index) Spalten gleichzeitig angelegt werden
- Enthält den Wert + die zugehörige Spalte
- Muss bei jedem Insert / Delete / Update neu erstellt werden

#### Wann?


- Es werden aus einem großen Table nur wenige Ergebnisse erwartet
- Die Spalte enthält häufig NULL Werte

#### Wann nicht?

- Wenn die Tabelle oft bearbeitet / selten verwendet wird
- Wenn häufig mehr als 2-4% der Tabelle ausgegeben werden

#### Function based

- Die Werte im Index werden durch Funktionen berechnet:




```
CREATE INDEX upper_last_name_idx  
ON employees (UPPER(last_name));
```

- Es können auch selbst geschriebene Funktionen verwendet werden, diese müssen allerdings als "deterministic" markiert werden


#### Erstellen & Löschen

- Erstellen



```
CREATE INDEX index_name  
ON table_name(column...,column)
```

- Löschen



```
DROP INDEX upper_last_name_idx;
```

### 2.1.8 Hierarchisches SQL

- Parent  $\implies$  Wert über einer Node
- Child  $\implies$  Wert unter einer Node
- Sibling  $\implies$  Wert auf der gleichen Höhe
- Leaf  $\implies$  Node ohne Child

#### Abfragen

- Pseudospalten
  - LEVEL  $\implies$  Level ab Root (hat Level 1)
  - CONNECT\_BY\_ISCYCLE  $\implies$  Gibt 1 zurück, wenn das Element Grund für einen Loop ist (letzter in der Hierarchie, bevor es von vorne los geht!)
  - CONNECT\_BY\_ISLEAF  $\implies$  Gibt 1 zurück, wenn das Element ein Leaf ist
- Funktionen
  - SYS\_CONNECT\_PATH(column, char)  $\implies$  Pfad des Elements von der Root Node weg, getrennt durch *char*
- Operatoren
  - SYS\_CONNECT\_BY\_ROOT  $\Leftarrow$  Gibt den Wert der Spalte der Root Node zurück
  - PRIOR  $\Leftarrow$  Um Parent Nodes zu verbinden
- Clauses
  - START WITH *condition*  $\implies$  Auswahl der Root-Zeile
  - CONNECT BY ...PRIOR  $\implies$  Gibt Verbindung zwischen Parent und Child an (mit PRIOR kann auf den Parent zugegriffen werden)
  - ORDER SIBLINGS BY  $\implies$  Sortiert die Siblings des Parents nach einer Spalte



```
SELECT e.ename, PRIOR e.ENAME, SYS_CONNECT_BY_PATH(e.ENAME, '/'), LEVEL
FROM EMP e
WHERE LEVEL >= 2
START WITH e.MGR IS NULL
CONNECT BY PRIOR e.EMPNO = e.MGR
ORDER SIBLINGS BY e.ENAME;
```





# 3 Themenkorb - Relationales Datenbankmodell

## 3.1 DDL

### 3.1.1 Datentypen

- CHAR(n)  $\implies$  Fixed-length, Rest wird mit Leerzeichen aufgefüllt bzw. abgeschnitten!
- VARCHAR(n)  $\implies$  Variable Länge (max. n)
- Date
- Timestamp
- NUMBER(s, p)  $\implies$  Einzige Zahlen-Datentyp in Oracle: s gibt die Gesamtstellen an, p die nach dem Komma
  - NUMERIC, DECIMAL sind nur die ANSI Name für diese Datentypen!
  - Float / Real / Double Precision steht in den Docs zwar als Subtyp von Number, wird aber (im Unterschied zu NUMERIC...) als FLOAT in Describe angezeigt!

### 3.1.2 Constraints

- NOT NULL  $\implies$  Null-Werte sind nicht erlaubt
- UNIQUE  $\implies$  Der Wert muss innerhalb der Spalte einzigartig sein
- PRIMARY KEY
  - Sofort nach dem Attribut, wenn er nur aus einem Attribut besteht
  - Am Ende des Tables, wenn er aus mehreren Attributen besteht!

```
CREATE TABLE bookLending
(
  isbn INTEGER,
  lendingDate DATE,
  CONSTRAINT pk_bookLending PRIMARY KEY (isbn, lendingDate)
);
```

- FOREIGN KEY
  - Am Ende des Tables

```
CREATE TABLE Orders
(
  O_Id INTEGER PRIMARY KEY,
  P_Id INTEGER,
  CONSTRAINT fk_PerOrder FOREIGN KEY (P_Id)
  REFERENCES Person(P_Id)
);
```

- CHECK  $\Rightarrow$  Um sicherzustellen, dass ein Wert ein gewisses Kriterium erfüllt

```
CREATE TABLE Persons
(
  P_Id INT NOT NULL,
  sal NUMBER,
  CONSTRAINT chk_Person CHECK (P_Id>0 AND sal > 0)
);
```

- DEFAULT  $\Rightarrow$  Default-Wert, falls dieser beim Insert weggelassen wird

#### 3.1.3 Tabellen im Nachhinein bearbeiten

- Vor allem bei FKs relevant, da dann nicht mehr auf die Reihenfolge von Tabellen geachtet werden muss!
- Es können Constraints & Spalten bearbeitet werden!
  - Constraints

```
ALTER TABLE Orders
ADD CONSTRAINT fk_PerOrder FOREIGN KEY(P_Id)
REFERENCES Person(P_Id);
```

- Spalten

```
ALTER TABLE TABLE_NAME
ADD column_name datatype;
```

```
ALTER TABLE TABLE_NAME
RENAME COLUMN column_name TO new_column_name;
```

```
ALTER TABLE TABLE_NAME
MODIFY column_name NEW DATA TYPE;
```

- Es können sowohl einzelne Constraints, als auch Columns und Tables gedroppt werden!
  - Beim Droppen von Tables empfiehlt es sich, vorher die Foreign-Key-Constraints zu entfernen, damit im Falle von Cascade Constraints keine Daten aus Versehen gelöscht werden!

## 3.2 DML

### 3.2.1 Views

- Sind das Ergebnis eines Select-Statements

### 3.2.2 Sequences

- Erstellen
- Beim Inserten → sequence.NEXTVAL

```
CREATE SEQUENCE seqOne
  START WITH 100
  INCREMENT BY 1
  [MAXVALUE 1000]
  [CYCLE]
```

### 3.2.3 MERGE

- Inserted ein Item, falls das gesuchte nicht gefunden wurde
- Updated ein existierendes Item, falls es gefunden wurde

## 3.3 Normalisierung

### 3.3.1 Normalformen

#### Nullte Normalform

- Mehrere Werte stehen in einer Zeile:

PersNr	Name	Vorname	AbtNr	Abteilung	ProjektNr	Beschreibung	Zeit
1	Lorenz	Sophia	1	Personal	2	Verkaufspromotion	83
2	Hohl	Tatjana	2	Einkauf	3	Konkurrenzanalyse	29
3	Willschrein	Theodor	1	Personal	1,2,3	Kundenumfrage, Verkaufspromotion, Konkurrenzanalyse	140, 92, 110
4	Richter	Hans-Otto	3	Verkauf	2	Verkaufspromotion	67
5	Wiesenland	Brunhilde	2	Einkauf	1	Kundenumfrage	160

#### Erste Normalform

- Jede Zeile enthält nur einen Wert
- Es muss ein Primary Key gefunden werden (unterstreichen!), welcher jede **Zeile** eindeutig kennzeichnet!

<u>PersNr</u>	Name	Vorname	<u>AbtNr</u>	Abteilung	<u>ProjektNr</u>	Beschreibung	Zeit
1	Lorenz	Sophia	1	Personal	2	Verkaufspromotion	83
2	Hohl	Tatjana	2	Einkauf	3	Konkurrenzanalyse	29
3	Willschrein	Theodor	1	Personal	1	Kundenumfrage	140
3	Willschrein	Theodor	1	Personal	2	Verkaufspromotion	92
3	Willschrein	Theodor	1	Personal	3	Konkurrenzanalyse	110
4	Richter	Hans-Otto	3	Verkauf	2	Verkaufspromotion	67
5	Wiesenland	Brunhilde	2	Einkauf	1	Kundenumfrage	160

## Zweite Normalform

- Die Relation befindet sich in der 1. Normalform + jedes Attribut ist vom Gesamtschlüssel der Relation abhängig, und nicht nur von einem Teil!
- Praxis: Ursprüngliche Tabelle in mehrere unterteilen, sodass oben genannte Anforderungen erfüllt sind!
  - Diese Tables dürfen nur Attribute enthalten, die vom gesamten PK abhängig sind  $\Rightarrow$  Es kann sein, dass eine Relation 2 Primary Key Attribute benötigt!

### Relation Projekt

<u>ProjektNr</u>	Beschreibung
2	Verkaufspromotion
3	Konkurrenzanalyse
1	Kundenumfrage

### Relation Personal

<u>PersNr</u>	Name	Vorname	<u>AbtNr.</u>	Abteilung
1	Lorenz	Sophia	1	Personal
2	Hohl	Tatjana	2	Einkauf
3	Willschrein	Theodor	1	Personal
4	Richter	Hans-Otto	3	Verkauf
5	Wiesenland	Brunhilde	2	Einkauf

### Relation Firma

<u>PersNr</u>	<u>ProjektNr</u>	Zeit
1	2	83
2	3	29
3	1	140
3	2	92
3	3	110
4	2	67
5	1	160

## Dritte Normalform

- Die Relation befindet sich in der 2. Normalform + Kein Attribut ist von einem anderen Nicht-Schlüssel-Attribut abhängig!

### Relation Projekt

<u>ProjektNr</u>	Beschreibung
2	Verkaufspromotion
3	Konkurrenzanalyse
1	Kundenumfrage

### Relation Personal

<u>PersNr</u>	Name	Vorname	<u>AbtNr.</u>
1	Lorenz	Sophia	1
2	Hohl	Tatjana	2
3	Willschrein	Theodor	1
4	Richter	Hans-Otto	3
5	Wiesenland	Brunhilde	2

### Relation Firma

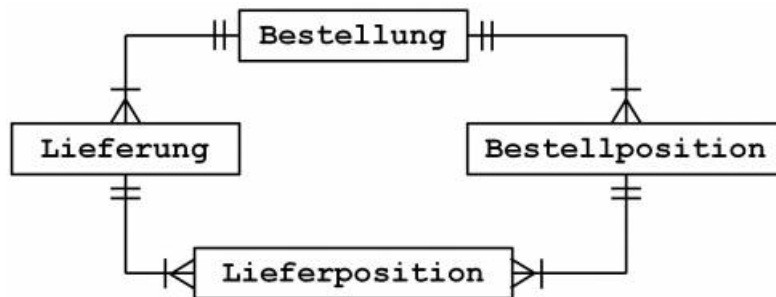
<u>PersNr</u>	<u>ProjektNr</u>	Zeit
1	2	83
2	3	29
3	1	140
3	2	92
3	3	110
4	2	67
5	1	160

### Relation Abteilung

<u>AbtNr.</u>	Abteilung
1	Personal
2	Einkauf
3	Verkauf

### 3.3.2 Anwendung - Zirkelbezug

- Problem des Zirkelbezugs
  - Eine Entity kann von einer Ausgangsentity auf 2 verschiedene Wege erreicht werden:



```

Bestellung(BestellNr)
Lieferung(LieferNr, BestellNrFK)
Bestellposition(PositionsNr, BestellNrFK)
Lieferposition(LieferNr, BestellNr, PositionsNr, BestellNrFK)
  
```

- Je nachdem, ob über die Lieferung oder die Bestellposition auf die Bestellung zugegriffen wird, kann es zu unterschiedlichen Ergebnissen kommen!
- Lösung: Die doppelten Attribute werden zu einem zusammengezogen und in einen Foreign Key verpackt:

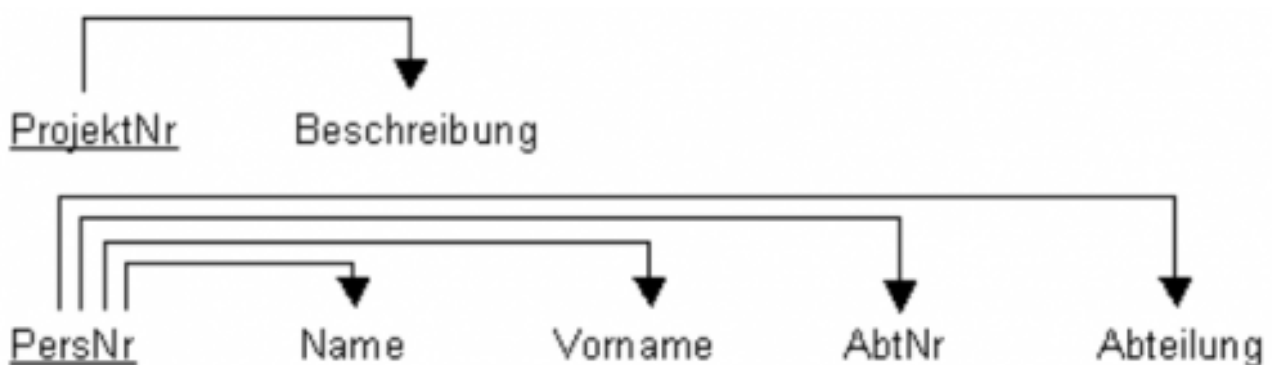
**Solution:**

```

Lieferposition((LieferNr, PositionsNr, BestellNr)FK)
  
```

### 3.3.3 Abhängigkeitsdiagramm - Beispiele

#### 2. Normalform



#### 3. Normalform

