DHBW Stuttgart

Datenbanken I

Kapitel 6.1 – SQL (DDL und DML)

Modul: T2INF2004

Hinweis

DHBW
Duale Hochschule
Baden-Württemberg

Nutzungshinweis:

**Diese Unterlagen dürfen ausschließlich von Mitgliedern
(das sind Studierende, Bedienstete)
der Dualen Hochschule Baden-Württemberg Stuttgart eingesetzt werden.
Eine Weitergabe an andere Personen oder Institutionen ist untersagt.**

Inhalte Vorlesung

1. Grundlagen und Begriffsdefinitionen
2. Der konzeptionelle Datenbankentwurf (ER-Modell)
3. Der relationale Entwurf (logischer Entwurf)
4. Die relationale Entwurfstheorie (Normalformen)
5. Einführung zum Datenbankentwurf

6.1 Die Sprache SQL (Teile DDL und DML)

7. Relationale Algebra (eine formale Sprache)

6.2 Die Sprache SQL (Teile DQL und DCL)

8. Transaktion und Mehrbenutzersysteme
9. Interne Speicherorganisation (Indexstrukturen)

Datenbank anlegen (DDL)

- SQL kennt in der DDL verschiedene Befehle wie *Create, Alter , Drop…*
- Zuerst müssen wir eine Datenbank anlegen und dazu verwenden wir den Befehl `CREATE DATABASE`
- Für das Anlegen des Softwarehauses schreiben wir:
  `CREATE DATABASE Softwarehaus`
- Was macht nun PostgreSQL aus diesem CREATE?

```
-- Database: Softwarehaus

-- DROP DATABASE "Softwarehaus";

CREATE DATABASE "Softwarehaus"
    WITH
    OWNER = postgres
    ENCODING = 'UTF8'
    LC_COLLATE = 'German_Germany.1252'
    LC_CTYPE = 'German_Germany.1252'
    TABLESPACE = pg_default
    CONNECTION LIMIT = -1;
```

Anforderung an DDL

- Anforderungen beim Anlegen von Tabellen.
  Was muss eine relationale Data Definition Language unterstützen?

  - ..... definieren
  - ..... festlegen
  - .....anlegen
  - .....festlegen

Und schließlich:

I..... festlegen

© D.Reichardt

SQL auf drei Ebenen

Externe Ebene:

CREATE VIEW <Sichtname> [<Schemadeklaration>]
AS <SQL-Anfrage>
DROP VIEW <Sichtname>

Konzeptuelle Ebene:

CREATE TABLE <TabellenName> (<attrib-name> <Wertebereich>, ...)

ALTER TABLE <TabellenName> [...]

DROP TABLE <TabellenName>

Interne Ebene:

CREATE INDEX <IndexName> ON <TabellenName> (<indexcol> ...)

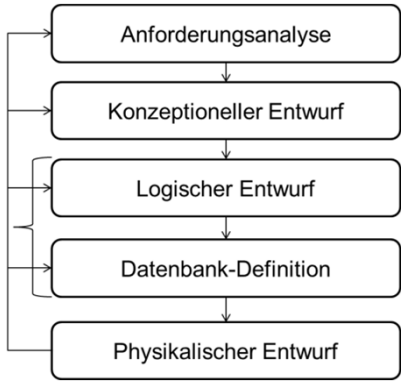
ALTER INDEX <IndexName> [...]

DROP INDEX <IndexName>

Tabellen definieren (DDL)

- Nicht „wild“ Tabellen anlegen, sondern den kompletten DB-Entwurfsprozess durchlaufen
- Der wichtigste und umfangreichste SQL-Befehl in DDL ist

```
CREATE TABLE TabellenName
(
Attribut_1 Domäne_1,
...
Attribut_n Domäne_n )
```



- Attribute sind die Namen der Spalten der Tabelle
- Domänen die …….

Bezeichnungen innerhalb einer Tabelle

Mitarbeiter Softwarehaus Name der Relation

Pers-Nr	Vorname	Nachname	Geschlecht	Geb.-Name	Eintritts-Datum	Skill	Gehalt-Stufe
1	Hans	Müller	m	NULL	1.07.2001	PR	It2
2	Rita	Schulze	w	NULL	1.11.2007	DBA	It3
3	Werner	Maier	m	NULL	1.01.2010	Test	It2
4	Karin	Schwarz	w	Klein	1.03.2005	PR	It2

Relationen-schema

Relation

Tupel

Attribut-wert

Attribut

Wahl des Datentyps

- Die Wahl des „richtigen“ Datentyps für jedes Attribut einer Tabelle hat verschiedene Vorteile:
  - Operationen auf Spalten mit demselben Datentyp liefern auch konsistente Ergebnisse
  - Wird z.B. ein einfacher Datentyp gewählt, kann eine verdichtete Abspeicherung erfolgen
  - Bei der Wahl des geeignetsten Datentyps erfolgt eine effiziente Speicherung, was beim Abfangen einen Performancevorteil bedeuten kann
- PostgreSQL erlaubt auch die Speicherung großer Objekte (LOBs)
- Datentypen für bspw. geometrische Objekte oder Netzwerkadressen sind in PostgreSQL vordefiniert
- Zusätzlich ist es in PostgreSQL auch möglich eigene Datentypen (*CREATE TYPE)* zu definieren

Datentypen (Auszug)

- Drei fundamentale Datentypen werden als Attribut-Domänen verwendet: Zahlen, Zeichenketten und der Datumstyp

Datentyp	Wertebereich / Beschreibung
integer, int	Ganzzahlwert zwischen -2147483648 to +2147483647 (es gibt auch *smallint, bigint)*
decimal(p,s) numeric(p,s)	Zahl mit vorgegebener Vor- und Nachkommastellen (p Ziffern und davon s nach dem Komma)
character(n) char(n)	Zeichenkette mit fixer Länge *n*, wird mit Leerzeichen aufgefüllt
character varying(n) varchar(n)	Zeichenkette mit variabler maximaler Länge n
date	Hier wird eine Zeichenkette speziell für das Datum aufbereitet. Das Format ist 2019-03-28 (interne Speicherung)

Datentypen verschiedener DBMS

	MySQL 3.23 JDBC 2.0a (mm)	PostgreSQL 7.2 JDBC 2, 7.2	Oracle 8.0 JDBC 8.1.6 (Oracle)	Oracle 8.0 ODBC 1.2 (msorcl32.dll)	MS Access 2000 ODBC 2.0001
INT, INTEGER	INT (32 bit signed, BIGINT 64 bit signed)	INT4	NUMBER (38 Stellen)		INTEGER
FLOAT	FLOAT (16 St., E+/-38 DOUBLE 24 St., E+/-308)	FLOAT8 (mit variabler Nachkommastellenzahl)	FLOAT (38 Stellen signed)	FLOAT (38 Stellen signed) Nachkommastellenzahl 0,2,4,...	DOUBLE
DECIMAL	DECIMAL ohne Nachkommastellen	NUMERIC mit variabler Nachkommastellenzahl	NUMBER ohne Nachkommastellen		--
NUMERIC	DECIMAL ohne Nachkommastellen	NUMERIC mit variabler Nachkommastellenzahl	NUMBER ohne Nachkommastellen		DOUBLE mit variabler Nachkommastellenzahl
DECIMAL(p,s) **NUMERIC(p,s)**	DECIMAL mit vorgegebener Nachkommastellenzahl	NUMERIC mit vorgegebener Nachkommastellenzahl	NUMBER mit variabler Nachkommastellenzahl	NUMBER mit vorgegebener Nachkommastellenzahl	--
NUMBER	--	--	NUMBER mit variabler Nachkommastellenzahl	NUMBER Nachkommastellenzahl 0,2,4,...	DOUBLE mit variabler Nachkommastellenzahl
NUMBER(p,s)	--	--	NUMBER mit variabler Nachkommastellenzahl	NUMBER mit vorgegebener Nachkommastellenzahl	--
DATE	DATE (2000-11-28)	DATE (2002-03-14)	DATE (2000-11-28 16:59:57.0)	DATE (2000-11-28 16:59:57)	DATETIME (2000-11-28 16:59:57)
DATETIME	DATETIME (2000-11-28 16:59:57)	TIMESTAMP (2002-03-14 11:12:13)	--		DATETIME (2000-11-28 16:59:57)
CHAR(n)	VARCHAR (bis 255 Zeichen)	BPCHAR	CHAR (bis 255 Zeichen)		CHAR (bis 255 Zeichen)
VARCHAR(n)	VARCHAR (bis 255 Zeichen)	VARCHAR	VARCHAR2 (bis 2000 Zeichen)		VARCHAR (bis 255 Zeichen)
BLOB, ...	BLOB (bis 64 KByte) LONGBLOB (bis 4 GByte)	als BYTEA oder per OID (Object Identifier)	BLOB (bis 4 GByte) LONG RAW (bis 2 GByte)		kein BLOB, aber: LONGCHAR (bis 64 KByte) MEMO (bis 64 KByte)
UPPER / UCASE	UPPER und UCASE	UPPER	UPPER		UCASE
SYSDATE / NOW	SYSDATE und NOW	NOW()	SYSDATE		NOW()

https://www.torsten-horn.de/techdocs/sql.htm#SQL-Datentypen

Tabellen definieren (DDL)

***Mitarbeiter**: {[PersNr: integer, Vorname: string, Nachname: string, Geb_Name: string, Geb_Datum: date, Geschlecht: string, Eintrittsdatum: date]}*

Die Tabelle *Mitarbeiter* dann folgendermaßen angelegt:

```
CREATE TABLE Mitarbeiter
 (
    PERS_Nr integer NOT NULL,
    Vorname character varying,
    Nachname character varying,
    Geb_Name varchar,
    Geb_Datum date,
    Geschlecht char(1),
    Eintrittsdatum date
);
```

Bedingungen festlegen (Constraints)

Aufbau (allgemein) einer Tabelle mit Constraints

```
CREATE TABLE TabellenName
(
Attribut_1 Domäne_1,
...
Attribut_n Domäne_n,
Constraint_1,
...
Constraint_m
)
```

Bedingungen festlegen (Constraints)

Constraint	Verwendung
NOT NULL	Stellt sicher, dass dieses Attribut keinen NULL-Wert enthält.
UNIQUE	Jeder Wert in dieser Spalte der Tabelle ist eindeutig.
PRIMARY Key	Dieses Attribut stellt den Primärschlüssel dar und identifiziert dadurch einen Datensatz eindeutig.
FOREIGN Key	Fremdschlüssel zeigt auf ein Attribut (Spalte) in einer anderen Tabelle (stellt zu dieser eine Verbindung her).
CHECK *Bedingung*	Mit dieser Angabe kann sichergestellt werden, dass ein Wert eines Attributs eine bestimmte Bedingung erfüllt.
DEFAULT *wert*	Mit dieser Bedingung kann ein Attribut (Spalte) automatisch mit einem Wert vorbelegt werden, wenn von außen kein Wert zugewiesen wurde.

Bedingungen festlegen (Constraints)

***Mitarbeiter**: {[PersNr: integer, Vorname: string, Nachname: string, Geb_Name: string, Geb_Datum: date, Geschlecht: string, Eintrittsdatum: date]}*

```
CREATE TABLE Mitarbeiter
 (
    Pers_Nr          integer,
    Vorname          varchar,
    Nachname         ……..,
    Geb_Name         varchar,
    Geb_Datum        date,
    Geschlecht       ……,
    Eintrittsdatum   ……..,

       PRIMARY KEY (……..),
       CHECK (……..)
);
```

Bedingungen festlegen (Constraints)

Frage: Was ist der Unterschied zwischen beiden nachfolgenden Definitionen?

```
CREATE TABLE Mitarbeiter
 (Pers_Nr            integer UNIQUE NOT Null,
    …….
);

CREATE TABLE Mitarbeiter
 (Pers_Nr            integer PRIMARY KEY,
    …….
);
```

Der Primärschlüssel:

- ……
- ……
- Man kann dem Primärschlüssel auch einen Namen geben
  CONSTRAINT persPK PRIMARY KEY (pers_nr)

Bedingungen festlegen (Constraints)

Aufgabe:
Tabelle Mitarbeiter löschen und neu mit den zusätzlichen Attributen *Gehalt* und *Skill* wieder anlegen.

Folgende Bedingungen einführen:
- Die Personalnummer wird als Primärschlüssel festgelegt
- Das Gehalt darf sich nur in bestimmten Grenzen bewegen
- Geschlecht nur „m“ und „w“
- Beim Skill muss eine Vorbelegung stattfinden
- Anschauen, was Postgres daraus macht

Verbindung zweier Tabellen

Mitarbeiter: *{[PersNr: integer, Vorname: string, Nachname: string,…]}*

Abteilung:*{[Abt_Bez_kurz: string, Abt_Bez_lang: string, Standort: string]}*

Arbeitet_in*:{[Pers_Nr, Abt_Bez_kurz]}*

Mitarbeiter: *{[PersNr: integer, Vorname: string, Nachname: string,…*
arbeitet_in]}

- Diese Verbindung erfolgt über den Constraint *FOREIGN KEY*

Verbindung zweier Tabellen

```
CREATE TABLE Mitarbeiter
 (
    Pers_Nr              integer,
    Vorname              varchar,
    …
    arbeitet_in          varchar (8),

   PRIMARY KEY (Pers_Nr),

    FOREIGN KEY(arbeitet_in) REFERENCES Abteilung,
    …
);
```

Beachte:
Beim Anlegen von Tabellen mit Fremdschlüsselbeziehungen muss die referenzierte Tabelle bereits vorhanden sein.
Daher ist es einfacher die Verbindung nachträglich hinzuzufügen.

Verändern von Tabellen (ALTER)

- Um Struktur einer bestehenden Tabelle zu verändern wird der SQL-Befehl ALTER verwendet.

```
ALTER TABLE Table_Name
   ADD Attribut_1 Domäne_1
```

- Konkret wollen wir die Tabelle Mitarbeiter um ein Attribut und den Fremdschlüssel erweitern.

```
ALTER TABLE Mitarbeiter
ADD arbeitet_in varchar (8),
ADD FOREIGN KEY(arbeitet_in) REFERENCES Abteilung;
```

- Es gibt noch weitere Formen für den Befehl ALTER, anbei ein paar Beispiele.

```
ALTER TABLE Table_Name RENAME TO Table_Name_neu
ALTER TABLE Table_Name ADD UNIQUE(Attribut)
ALTER TABLE Table_Name ADD CHECK(Abfrage)
...
```

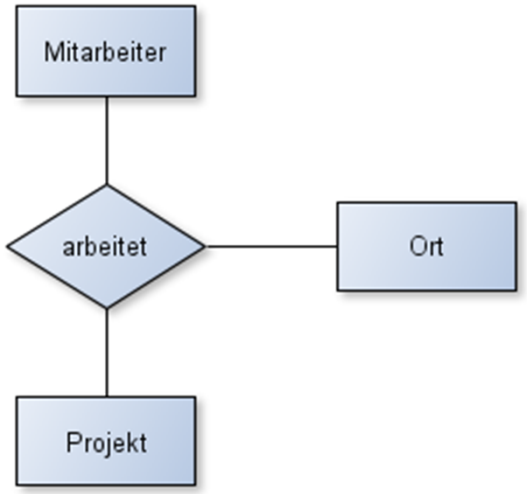
Der Löschbefehl in der DDL

- Datenbanken, Tabellen und Attribute können auch gelöscht werden
- Der zugehörige SQL-Befehl wirkt auf verschiedenen Ebenen der Datenbank

```
DROP TABLE Mitarbeiter
ALTER TABLE Mitarbeiter DROP Skill
```

- Mit dem Befehl DROP DATABASE können auch eine komplette Datenbank oder andere Objekte gelöscht werden

Umsetzung einer Ternäre Beziehung



a) An einem Projekt sind an einem Ort mehrere Mitarbeiter beteiligt.
b) Ein Mitarbeiter arbeitet in einem Projekt genau an einem Ort.
c) An einem Ort arbeitet ein Mitarbeiter genau an einem Projekt.
d) Ein Mitarbeiter kann an mehreren Projekten arbeiten, dann aber an verschiedenen Orten.

Die Datenmanipulationssprache (DML)

Nun müssen wir aber auch Daten in die Tabelle einfügen und diese verwalten und dazu dient uns die Data Manipulation Language (DML) als Teil von SQL.

- Um Daten zu verändern haben wir drei Möglichkeiten
  - Einfügen einer neuen Zeile(n)
  - Modifizieren einer bereits bestehenden Zeile(n)
  - Löschen einer Zeile(n)

Zeilen einfügen INSERT

```
INSERT INTO TabellenName
VALUES (Wert_1, Wert_2, Wert_3);
```

- Daten können nur gemäß der Definition des Datentyps eines Attributs eingefügt werden.
- Die Reihenfolge der Werte, muss der Reihenfolge der Spaltenwerte der Tabelle entsprechen und kein Wert darf ausgelassen werden.

```
INSERT INTO Mitarbeiter
VALUES (1, 'Hans', 'Müller', Null, '1975-03-12,'m',
'2014-07-01',default, 48000);
```

- Default bedeutet, dass der im Constraint vorbelegte Wert verwendet wird
- Unbekannte Werte werden mit NULL angegeben

Zeilen einfügen INSERT

```
INSERT INTO TabellenName (SpaltenName1, SpaltenName n,..)
VALUES (Wert_1, Wert_2,…);
```

- Werte, welche gezielt eingefügt werden sollen, werden explizit angegeben
- Constraints dürfen nicht verletzt werden
  (z.B. fehlender Wert bei NOT NULL)
- Die Reihenfolge der Werte ist frei wählbar

```
INSERT INTO Mitarbeiter (pers_Nr, nachname,geschlecht,
eintrittsdatum)
values (3, 'Klein','m', '20.03.2012')
```

Neu eingefügte Zeilen in einer Tabelle haben keine vorhersagbare Position. Dies ist absolut zufällig und die Zeilen einer Tabelle besitzen keine natürliche Reihenfolge.

Zeilen aus Datei einfügen COPY

- Sollen Massendaten in einer Tabelle eingefügt werden, so verwendet Postgres den Befehl COPY.
  Die Daten können sich z.B. in einer Datei im CSV-Format befinden.

```
copy public.mitarbeiter from
'C:\Users\Postgres\Mitarbeiter_in.csv' delimiter ',' csv;
```

- Es können natürlich auch Werte aus einer Tabelle in eine Datei geschrieben werden.

```
copy (SELECT * FROM Mitarbeiter) TO
'C:\Users\Postgres\Mitarbeiter_out.csv' delimiter ',' csv;
```

Zeilen verändern UPDATE

- Bestehende Zeilen müssen bei falschen Daten, oder wenn Daten sich ändern, modifiziert werden.

```
UPDATE TabellenName
SET Attribut = Wert, Attribut1 = Wert1, …
WHERE <Prädikat>

UPDATE TabellenName
SET Nachname = 'Maier'
WHERE Pers_Nr = 1;
```

- Ohne Where-Klausel wird der Update für alle Zeilen durchgeführt.
- Es können nicht nur Konstanten, sondern auch arithmetische Ausdrücke verwendet werden (z.B Gehalt =Gehalt *1,08).
- Sucht man alle Attribute bei denen ein bestimmter Wert bisher noch nicht belegt wurde (NULL) so schreibt man
  WHERE Gehalt **is** NULL

Zeilen löschen DELETE

```
DELETE FROM TabellenName
WHERE <Prädikat>
```

- Mitarbeiter Maier hat unser Softwarehaus verlassen, dann schreiben wir:

```
DELETE FROM Mitarbeiter
WHERE Pers_Nr = 5;
```

Hier dürfen wir die Where-Klausel nicht weglassen, ansonsten werden alle Zeilen der Tabelle gelöscht.

Daten in verbundenen Tabellen löschen

- Was passiert wenn Daten gelöscht werden sollen, zu denen es Abhängigkeiten gibt (Fremdschlüsselverbindung zu weiterer Tabelle vorhanden)?
- Die sogenannten Löschregeln werden vom DBMS angewandt, wenn der Primärschlüssel der abhängigen Tabelle als Fremdschlüsselwert enthalten ist.
- Ziel dieser Überwachung ist es, die referenzielle Integrität sicher zu stellen.
- Mit den Löschregeln sagen wir dem DBMS wie es auf die Verletzung der referenziellen Integrität reagieren soll.
- Drei Möglichkeiten werden nachfolgend an Beispiele aus unserem Softwarehaus aufgezeigt.

Abhängige Daten auf NULL setzen

- Klassische 1:N-Beziehung soll betrachtet werden (keine hierarchischen Abhängigkeit)
- Was passiert, wenn eine Abteilung gelöscht wird?

```
Delete from Abteilung
where abt_bez_kurz ='EDM1';
```

- Das Löschen .....!
- Da wir die Abteilung aber löschen wollen, müssen wir die zugehörigen Fremdschlüssel-Werte in der Tabelle *Mitarbeiter* in Spalte „*arbeitet_in*“ auf NULL setzen.

Abhängige Daten auf NULL setzen

- Es muss eine Info an das DBMS erfolgen, dass die Löschregel „NULL setzen“ angewendet werden soll.
- Dies erfolgt bereits beim CREATE-Befehl in der Angabe des Constraint.

```
CREATE TABLE Mitarbeiter
 (
    ...
    FOREIGN KEY(arbeitet_in) REFERENCES Abteilung
    on Delete Set NULL ;
```

Abhängige Daten werden gelöscht

- Es besteht eine hierarchische Abhängigkeit zwischen Kunde und Auftrag.
- Ein Auftrag existiert im Normalfall nicht ohne einen Kunden.
- Wenn ein Kunde gelöscht wird, sollten auch alle zugehörigen Aufträge gelöscht werden.
- Dies erreichen wir durch die Löschregel CASCADE (kaskadierendes Löschen).

```
CREATE TABLE Auftrag
    (
        ...
        FOREIGN KEY(erteilt) REFERENCES Kunde
        on Delete CASCADE;
```

Löschen abweisen

- Per Default wird die Löschregel vom DBMS auf RESTRICT gesetzt.
- Wenn eine Fremdschlüsselbeziehung existiert, wird das Löschen eines Datensatzes aus der referenzierten Tabelle abgewiesen.
- Könnte eventuell im Modell Softwarehaus bei *Projekt* und *Leistung* anwendet werden.
- Leistungen, welche einem Projekt aus einem Auftrag zugeordnet sind, dürfen im Regelfall nicht verlorengehen falls da Projekt gelöscht wird.
- Löschen müsste abgewiesen werden.

```
CREATE TABLE Leistung
 (
     ...
    ADD FOREIGN KEY(zu_Projekt) REFERENCES Projekt
    on Delete RESTRICT;
```

Wichtige Hinweis zu Löschen

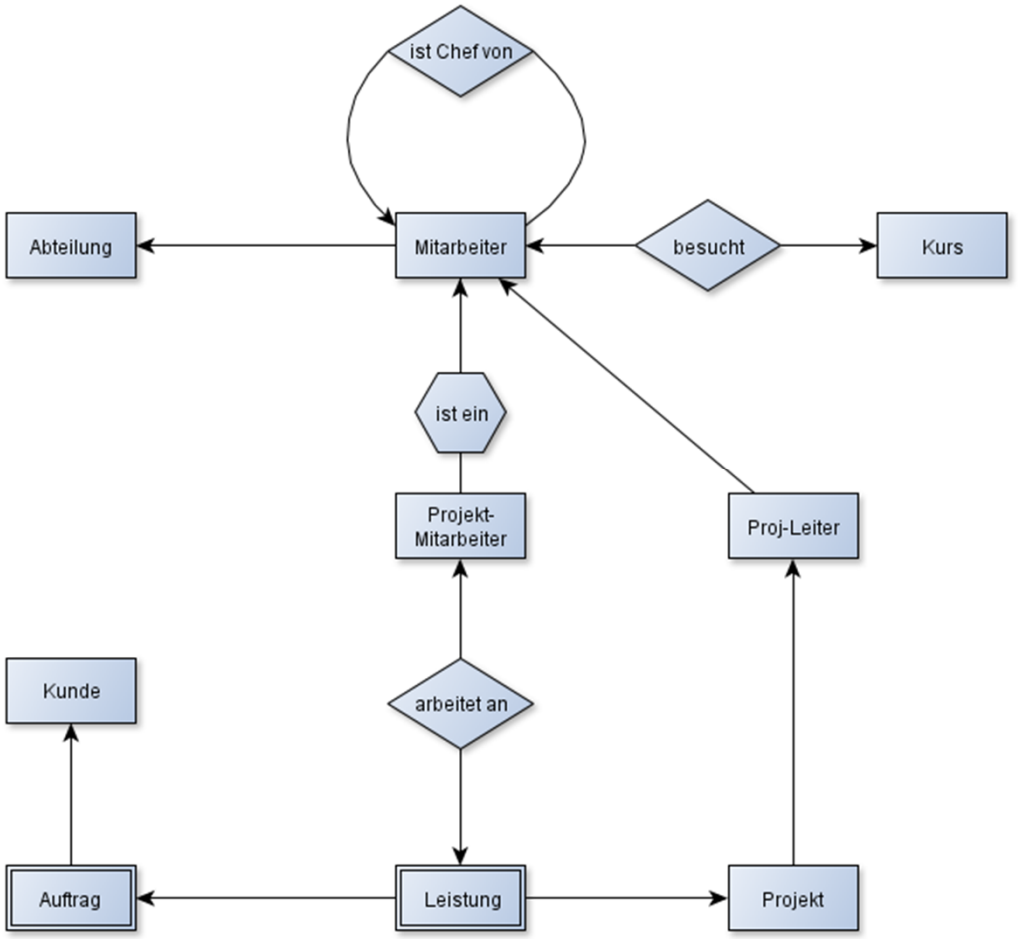
- Das Löschen von abhängigen Datensätzen in komplexen Datenmodellen ist eine sehr schwierige Angelegenheit und mit äußerster Vorsicht anzuwenden.
- In der Regel sind nicht nur zwei Tabellen betroffen, sondern es existieren Beziehungsstrukturen
  (siehe Kunde, Auftrag, Leistung, arbeitet_an, Projekt).
- Zuerst eine Analyse der Miniwelt durchführen und dann abwägen, welche Löschregeln anzuwenden sind.
- Eventuell kaskadierendes Löschen nicht erlauben
- Zusätzliche Spalte in die referenzierende Tabelle einfügen, an der erkannt werden kann, ob der abhängige Datensatz noch aktiv ist.

  Problem: Die Datenintegrität muss per Programm überwacht werden.

Regeln für den UPDATE

- Auch beim Verändern (UPDATE) von Daten in Tabellen mit Fremdschlüsselbeziehungen können Regeln angegeben werden.
- UPDATE ON… beim Create-Befehl sagt aus, wie das DBMS reagieren soll, wenn der Primärschlüssel geändert wird.
- Sollen auch alle Fremdschlüsseleinträge der referenzierten Tabelle angepasst werden, so schreiben wir beim CREATE UPDATE ON CASCADE.
- Wird nichts angegeben, so zieht wieder die RESTICT-Bedingung.
- Da ein Primärschlüssel nicht NULL sein darf, gibt SET NULL als Regel wenig Sinn.

Übung zum Constraint für Delete



Legende:
K = Kaskadierendes Löschen
N = NULL setzen
A = Löschen abweisen

Aufgabe:
Versuchen Sie die Löschregeln für diese Beziehungsstruktur sinnvoll zu setzen und fügen Sie diese bei einigen Tabellen ein.

Ende Kapitel 6.1