Folien zur Vorlesung Datenbanken

Kapitel 4

Datenbanksprache SQL - Einrichten der Datenbank

Fachhochschule Wedel

Prof. Dr. Ulrich Hoffmann

basierend auf den Lehrmaterialien von
Prof. Dr. Hans–Detlef Gerhardt

Kapitel 4: Datenbanksprache SQL – Einrichten der DB

fhwedel UNIVERSITY OF APPLIED SCIENCES

4.1 Tabellendefinitionen

Mit dem **DDL** (Data Definition Language)-Teil von SQL wird das Ergebnis des Datenbankentwurfes in Form von Tabellendefinitionen implementiert.

4.1.1 Erstellen von Tabellen

Die Definition der Struktur einer Tabelle **(Tabellenschema)** besteht aus

- **dem Tabellennamen**
- **einer Menge von Spaltenüberschriften (Attribute)**
- **den Wertebereichen der Spalten**
- **den *Constraints* der Spalten**
  - **NULL** oder **NOT NULL-Angaben**
  - **UNIQUE-Angaben**
- **INDEX-Definitionen**
- **den *Constraints* der Tabelle**
  - **Primary Key**
  - **Foreign Key**
  - **Check**
  - **UNIQUE**
  - **INDEX**

Bereiche einer Tabellendefinition:

- **Spalten-Definitionen** mit

  Spaltennamen,

  Datentypen und

  eventuell mit einer **default** – Definition und

  einer **in-line Constraint** – Definition

- **Constraint-Definitionen** als **out-of-line** Definition

Syntax:

```
CREATE TABLE  [datenbank.] tabellenname
      ( spaltenname  datentyp  [NOT NULL| NULL]
      [,spaltenname  datentyp  [NOT NULL | NULL]...] );
```

- **Tabelle** angelegt, mit der sofort gearbeitet werden kann

Beispiel:

```
CREATE TABLE  PERSONAL
  (PNR                integer(4)  NOT NULL,
   NAME               char(20) NOT NULL,
   VORNAME            char(20),
   GEH_STUFE          char(3),
   ABT_NR             char(3),
   KRANKENKASSE       char(3) );
```

Bemerkungen:

MySQL-Tabellen haben maximal 4096 Spalten, aber zusätzlich eine maximale Zeilengröße von 65535 Bytes.

Namensbildung erfolgt nach folgenden Regeln:

- **1-64 Zeichen**
- **Namen ohne Quotes `:**
  - **Beginnt mit Buchstabe, außer Buchstaben können enthalten sein: _ $ #**
  - **Groß- und Kleinbuchstaben werden nicht unterschieden,**
  - **Leerzeichen ist nicht zugelassen,**
  - **ebenso wenig die reservierten Wörter.**
- **Namen in `Quotes`**
  - **quasi beliebige Zeichen, aber dürfen nicht mit Zwischenraum enden oder / \ . enthalten.**

NOT NULL für eine Spalte heißt:

Bei Eingabe einer Zeile muss für diese Spalte ein Wert aus dem angegebenen Wertebereich eingetragen werden.

NULL für eine Spalte heißt:

Bei Eingabe einer Zeile muss für diese Spalte kein Wert eingetragen werden.
Damit wird die Tatsache modelliert, dass zum Zeitpunkt des Einfügens einer neuen Zeile nicht alle Informationen vorliegen müssen.

Default-Wert ist **NULL**

4.1.2 Semantische Datenintegrität

Deklarative Integritätsmethoden

- **Referentielle Integritätsregeln**
  zur **Definition von Abhängigkeiten** zwischen unterschiedlichen DB–Tabellen

- **Referentielle Integritätsaktionen**
  zur **Beschreibung von Aktionen**, die dann ausgeführt werden, wenn innerhalb einer übergeordneten Tabelle ein Datensatz gelöscht wird.

- **Entity-Integritätsregeln**
  zur **Definition von Primärschlüsseln, Wertebereichen und Null-Werten.**

Deklarative Integritätsdefinitionen werden zum **Erstellungszeitpunkt** der Tabelle innerhalb des **CREATE TABLE** angegeben.

Jede deklarative Integritätsdefinition (jedes **Constraint**) **sollte** einen **Namen** erhalten.

Entity – Integritätsregeln

Regeln, die die DB - Tabellen selbst betreffen.

Dazu gehören
- primary - key - Constraints
- unique - Constraints
- NOT NULL - Constraints
- check - Constraints

Prozedurale Integritätsmethoden

Damit können **zusätzliche Abhängigkeiten** innerhalb der Datenstrukturen implementiert werden.
Beliebig komplexe Regeln und Aktionen werden in Form von **Datenbank-Triggern** implementiert.

Datenbank-Trigger:

Ein Trigger ist ein benanntes Datenbankobjekt, das mit einer Tabelle verbunden ist und aktiviert wird, wenn für diese Tabelle ein bestimmtes **Ereignis** eintritt.

Jeder Datenbank-Trigger gehört **unmittelbar** zu einer Datenbank-Tabelle.

4.1.3 Beispiele für Tabellendefinitionen mit deklarativen Constraints

Definition nur mit Festlegung des Primärschlüssels:

```
CREATE TABLE  PERSONAL

(PNR            integer(4) primary key,
NAME            char(20) NOT NULL,
VORNAME         char(20),
GEH_STUFE       char(3),
ABT_NR          char(3),
KRANKENKASSE    char(3) );
```

Allgemeine Definition:

```
CREATE  TABLE MASCHINE

(MNR              integer(4),
NAME              char(20) default ('Bohrmaschine') NOT NULL,
PNR               integer(4),
ANSCH_DATUM  date default (sysdate),
NEUWERT         integer,
ZEITWERT        integer,

constraint        PK_MNR primary key (MNR),

constraint        C_ MASCHINE_NEUWERT_check
                  check
                  (NEUWERT BETWEEN 28000 AND 32000
                  AND NAME = 'Bohrmaschine'),

constraint        FK_PERSONAL_MASCHINE foreign key(PNR)
                  references PERSONAL(PNR)
) COMMENT 'Gesamter Maschinenpark mit personeller Zuordnung';
```

Das **NOT NULL** Constraint hat keinen expliziten Namen erhalten und ist in-line definiert.

Die **in-line** Definition wird dann benutzt, wenn die **Constraint–Definition** innerhalb der Spaltendefinition angegeben wird.

Beispiele:

```
MNR        integer(4) NOT NULL,
MNR        integer(4) primary key,
```

Die drei anderen Constraints sind **out-of-line** definiert.

Eine **Primary-Key-Definition** kann pro Tabelle **nur einmal** vorhanden sein und bewirkt,

- dass bezüglich der Primary-Key-Spalten keine doppelten Werte vorhanden sein können
- dass zum Zeitpunkt der Ausführung des **CREATE** TABLE Befehls ein **unique-Index** für die Primary-Key-Spalten erzeugt wird
- dass alle Spalten, die als Primary-Key-Spalten definiert sind, keine Nullwerte annehmen dürfen, d.h. diese Spalten haben ein implizites **NOT NULL Constraint.**

Beim **check-Constraint** werden Wertebereiche für die Spalten definiert. Die **out-of-line** Schreibweise ist übersichtlicher.

Bilden **mehrere Attribute** den Primärschlüssel ist nur die **out-of-line** Schreibweise anwendbar.

UNIQUE-Constraint:

- Anwendbar für eine oder mehrere Spalten einer Tabelle, auch mehrfach für eine Tabelle nutzbar.
- Werte dürfen nicht mehrfach auftreten.
- Nullwerte sind zugelassen.

Beispiele:

```
MNR    integer(4)          UNIQUE
MNR    integer(4)          NOT NULL UNIQUE


constraint  UN_MNR  UNIQUE (MNR)
```

DEFAULT:

- definiert default Wert, falls Wert für Tabellenspalte bei `insert`-Operation nicht angegeben wurde
- kein eigenständiges Constraint mit eigenen Namen

Beispiel:

CREATE TABLE tbl_table (created_date TIMESTAMP **DEFAULT** NOW())

Mit **referentiellen Integritätsregeln** werden Abhängigkeiten zwischen unterschiedlichen DB **-** Tabellen definiert.

Eine **Beziehungsdefinition** besteht aus zwei Komponenten:
- **primary key** oder **unique**–Constraint
- **foreign key** (Fremdschlüssel) des abhängigen Objektes.

Wird ein **Fremdschlüssel** definiert, so referenziert er zu einem **Primärschlüssel** oder einem **unique-**Constraint derselben oder einer beliebigen anderen Tabelle.

Eine **Löschoperation** in einer referenzierten Tabelle (**Master-**Tabelle) ist nur dann möglich, wenn

- kein abhängiger Datensatz in einer foreign-key-Tabelle existiert
  oder
- wenn die foreign-key-Beziehung mit der **on delete cascade** Option definiert wurde.

Die **on delete cascade** Option innerhalb der **foreign - key** Definition bewirkt, dass eine Löschoperation eines Master–Datensatzes gleichzeitig alle abhängigen Datensätze in der foreign-key-Tabelle löscht.

Beispiel:

```
constraint   FK_PERSONAL_MASCHINE foreign key(PNR)
             references PERSONAL(PNR) on delete cascade;
```

Zuerst werden die **abhängigen** Datensätze gelöscht und dann der **Master-Datensatz**.

Beispiele: Inline - Definition:

```
CREATE TABLE  PERSONAL

(PNR                    integer(4),
                        constraint PK_PNR primary key(pnr),
NAME                    char(20) NOT NULL,
VORNAME                 char(20),
GEH_STUFE               char(3),
                        constraint FK_GEHALT_PERSONAL foreign key
                        (GEH_STUFE) references GEHALT(GEH_STUFE),
ABT_NR                  char(3),
                        constraint FK_ABTEILUNG_PERSONAL
                        foreign key (ABT_NR)  references ABTEILUNG(ABT_NR),
KRANKENKASSE            char(3)
                        comment 'Kuerzel der gesetzlichen Krankenversicherung des Mitarbeiters'
) comment  'Auflistung aller Betriebsangehoerigen';


ALTER TABLE PERSONAL COMMENT 'Auflistung aller Betriebsangehoerigen';
```

Out-of-line-Definition von Integritätsbedingungen:

```
CREATE TABLE  PERSONAL
(PNR                    integer(4),
NAME                    char(20) NOT NULL,
VORNAME                 char(20),
GEH_STUFE               char(3),
ABT_NR                  char(3),
KRANKENKASSE            char(3),

constraint      PK_PNR primary key (PNR),
constraint      FK_GEHALT_PERSONAL
                foreign key (GEH_STUFE)
                references GEHALT(GEH_STUFE),
constraint      FK_ABTEILUNG_PERSONAL
                foreign key (ABT_NR)
                references ABTEILUNG(ABT_NR) );
```

```
CREATE TABLE   GEHALT
(GEH_STUFE     char(3),
               constraint PK_ GEH_STUFE primary key(GEH_STUFE),
BETRAG         smallint NOT NULL
) COMMENT 'Staffelung der Gehaelter';

CREATE TABLE   ABTEILUNG
(ABT_NR        char(3),
               constraint PK_ABT_NR primary key(ABT_NR),
NAME           char(20)
) COMMENT 'Auflistung der Abteilungsnamen';

CREATE TABLE   KIND
(PNR           integer(4),
K_NAME         char(20) NOT NULL,
K_VORNAME      char(20),
K_GEB          smallint,
constraint     PK_SCHLUESSEL_KIND
               primary key  (K_NAME,K_VORNAME),
constraint     FK_PERSONAL_KIND
               foreign key (PNR)
               references PERSONAL(PNR));
```

```
CREATE TABLE  PRAEMIE
(PNR            integer(4),
P_BETRAG        smallint NOT NULL,
constraint      FK_PERSONAL_PRAEMIE foreign key(PNR)
                references PERSONAL(PNR)
) COMMENT 'Praemien fuer ausgesuchte Mitarbeiter';

CREATE TABLE  MASCHINE
(MNR            integer(4),
NAME            char(20) NOT NULL,
PNR             integer(4),
ANSCH_DATUM     date,
NEUWERT         integer,
ZEITWERT        integer,
constraint      PK_MNR primary key(MNR),
constraint      FK_PERSONAL_MASCHINE foreign key(PNR)
                references PERSONAL(PNR)
) COMMENT 'Gesamter Maschinenpark mit personeller Zuordnung';
```

4.1.4 Systemtabellen und Views

In diesen Tabellen stehen alle wichtigen Informationen über die eingerichtete Datenbank.
Auf diese Informationen kann man lesend zugreifen.

Beispiel für eine **Systemtabelle**, die die Informationen über alle Relationen enthält: **information_schema.tables**

```
+-----------------+---------------------+------+-----+---------+-------+
| Field           | Type                | Null | Key | Default | Extra |
+-----------------+---------------------+------+-----+---------+-------+
| TABLE_CATALOG   | varchar(512)        | YES  |     | NULL    |       |
| TABLE_SCHEMA    | varchar(64)         | NO   |     |         |       |
| TABLE_NAME      | varchar(64)         | NO   |     |         |       |
...
| TABLE_COMMENT   | varchar(80)         | NO   |     |         |       |
+-----------------+---------------------+------+-----+---------+-------+
```

Anfragen an Systemtabellen mit **SELECT:**

```
SELECT   table_name, table_comment
FROM     information_schema.tables
WHERE    table_name='PERSONAL';
```

Ergebnis:

TABLE_NAME	TABLE_COMMENT
PERSONAL	Alle Betriebsangehörigen

wenn als Kommentar angegeben war:
COMMENT 'Alle Betriebsangehörigen';

Tabelle, die alle Informationen über die Spalten enthält: **information_schema.columns**

```
+--------------------------+---------------------+------+-----+---------+-------+
| Field                    | Type                | Null | Key | Default | Extra |
+--------------------------+---------------------+------+-----+---------+-------+
| TABLE_CATALOG            | varchar(512)        | YES  |     | NULL    |       |
| TABLE_SCHEMA             | varchar(64)         | NO   |     |         |       |
| TABLE_NAME               | varchar(64)         | NO   |     |         |       |
| COLUMN_NAME              | varchar(64)         | NO   |     |         |       |
| ORDINAL_POSITION         | bigint(21) unsigned | NO   |     | 0       |       |
| COLUMN_DEFAULT           | longtext            | YES  |     | NULL    |       |
| IS_NULLABLE              | varchar(3)          | NO   |     |         |       |
| DATA_TYPE                | varchar(64)         | NO   |     |         |       |
| CHARACTER_MAXIMUM_LENGTH | bigint(21) unsigned | YES  |     | NULL    |       |
| CHARACTER_OCTET_LENGTH   | bigint(21) unsigned | YES  |     | NULL    |       |
| NUMERIC_PRECISION        | bigint(21) unsigned | YES  |     | NULL    |       |
| NUMERIC_SCALE            | bigint(21) unsigned | YES  |     | NULL    |       |
| CHARACTER_SET_NAME       | varchar(32)         | YES  |     | NULL    |       |
| COLLATION_NAME           | varchar(32)         | YES  |     | NULL    |       |
| COLUMN_TYPE              | longtext            | NO   |     | NULL    |       |
| COLUMN_KEY               | varchar(3)          | NO   |     |         |       |
| EXTRA                    | varchar(27)         | NO   |     |         |       |
| PRIVILEGES               | varchar(80)         | NO   |     |         |       |
| COLUMN_COMMENT           | varchar(255)        | NO   |     |         |       |
+--------------------------+---------------------+------+-----+---------+-------+
```

```
SELECT  COLUMN_NAME , DATA_TYPE, COLUMN_COMMENT
FROM    information_schema.columns
WHERE   TABLE_NAME = 'PERSONAL';
```

4.2 Erweiterung von Tabellen

d.h. Erweiterung von Tabellen um neue Attribute/Constraints:

```
ALTER TABLE [datenbank.] tabellenname
ADD
      (spaltenname  datentyp  [NULL]
      [,spaltenname  datentyp  [NULL]...])
```

- Tabellen werden um die genannten Attribute erweitert
- Spalten werden mit NULL-Werten vorbesetzt
- Spezifikation **NOT NULL** ist nicht erlaubt (nur bei leeren Tabellen).

Beispiele:

Erweiterung der Tabelle PERSONAL um die Versicherungsnummer V_NR:

```
ALTER TABLE   PERSONAL
ADD           (V_NR number(4) );
```

bzw. bei nur einer zuzufügenden Spalte

```
ALTER TABLE   PERSONAL
ADD           V_NR number(4);
```

Einfügen eines Foreign-Key-Constraint:

```
ALTER TABLE   MASCHINE
ADD           constraint FK_MASCHINE foreign key(PNR)
              references PERSONAL (PNR);
```

Löschen des Foreign-Key-Constraint:

```
ALTER TABLE   MASCHINE
DROP          foreign key PNR;
```

Modifizieren der Spalte einer Tabelle:

- Ändern des Datentyps
- Ändern der Größe
- Ändern des Default-Wertes

Einschränkungen:

- Verkleinerungen sind nur möglich, wenn die Spalte nur Null-Werte oder die Tabelle keine Zeilen enthält.
- Ein Wechsel des Datentyps ist nur möglich, wenn die Spalte nur Null-Werte oder die Tabelle keine Zeilen enthält
- Ein Wechsel des Default-Wertes wirkt sich nur in der Zukunft aus.

Informationen über das CREATE TABLE einer gegebenen Tabelle:

```
SHOW TABLE CREATE PERSONAL;
```

Beispiel:

```
ALTER TABLE  PERSONAL
MODIFY COLUMN KRANKENKASSE char(5)
                    comment 'Kuerzel der gesetzlichen Krankenversicherung des Mitarbeiters';
```

4.3 Entfernen einer Tabelle aus einer Datenbank

```
DROP  TABLE [datenbank.] tabellenname
```

Damit werden gelöscht:

- die Tabelle
- die Einträge zur Tabelle in den Systemtabellen
- alle Views, Indexe, Rechte, Synonyme, die mit der Tabelle zusammenhängen

Beispiel:

```
DROP TABLE PERSONAL;
```

Benutzer muss das Recht besitzen, Tabellen der Datenbank zu löschen.

Systemtabellen können nicht gelöscht werden.

4.5 Indizes

- können erzeugt und wieder gelöscht werden
- automatisch angelegt, wenn **Primary Key-Constraints** angelegt werden
- damit kann der Nutzer das Zugriffsverhalten des Optimierers beeinflussen

Durch geschickte Wahl von Indizes können Antwortzeiten unter Umständen drastisch gesenkt werden.

Durch die Definition geeigneter Indizes gibt man dem Optimierer die Möglichkeit, aus mehreren möglichen Zugriffspfaden einen günstigen auszuwählen.

Für Spalten, die die Datentypen long oder long raw haben, können keine Indizes definiert werden.

4.5.1 Erstellen eines Index

Syntax:

```
CREATE [UNIQUE] INDEX indexname
ON  tabellenname
      (spaltenname [ASC|  DESC]
      [,spaltenname [ASC|  DESC]
       ...])
```

- Index für das/die genannten Spalten angelegt
- Indexdatei aufsteigend/absteigend sortiert angelegt
- Default-Wert: ASC
- bei mehreren Spaltennamen erfolgt die Sortierung von links nach rechts
- UNIQUE sichert Eindeutigkeit der Werte in den indizierten Spalten, auftretende Duplikate werden auch beim Betrieb der Datenbank nicht akzeptiert

Werte werden sortiert in Indexdatei abgelegt.
Bei Update der Tabelle wird Indexdatei automatisch nachgeführt (beachte: **Rechenzeit**).

Anwendung von Indizes:

- für Attribute, auf deren Basis häufig Tupel selektiert werden
- für Attribute, auf deren Basis häufig eine sortierte Ausgabe benötigt wird
- für Attribute, auf deren Basis häufig gruppiert werden soll.

Beispiele:

```
CREATE UNIQUE INDEX   vn_index
ON                    PERSONAL (VORNAME);
```

```
CREATE INDEX          KRANKENKASSE_NAME_index
ON                    PERSONAL  (KRANKENKASSE,NAME);
```

Sortiert wird alphabetisch nach dem Namen der Krankenkasse und bei gleicher Krankenkasse nach dem Namen des Mitarbeiters.

Information über Indizes findet man in der Tabelle information_schema.statistics

```
SELECT table_schema, table_name, index_name column_name, comment
FROM        information_schema.statistics
WHERE       table_name = 'PERSONAL';
```

```
SHOW INDEX FROM PERSONAL;
```

Ein Index darf auch definiert werden, wenn die Tabelle bereits Werte enthält.

⇒ Werte werden sortiert in Indexdatei abgelegt.

Bei **UPDATE** der Tabelle wird Indexdatei automatisch nachgeführt.

Empfehlung für die Wahl des index_NAME:

NAME + Zusatz „key“ Index primär zur Kennzeichnung des Schlüssels.

NAME + Zusatz „index“ Index primär für Zugriffsoptimierung.

4.5.2 Löschen eines Index:

```
DROP INDEX tabellenname.indexname
```

Benutzer muss das Recht besitzen, Indizes der Tabelle zu löschen.

Ein nicht benötigter Index sollte auf jeden Fall gelöscht werden
(benötigt Speicherplatz und Rechenzeit).

Beispiele:

```
DROP INDEX PNR_KEY;
```

```
DROP INDEX NAME_INDEX;
```

4.6 Einfügen und Ändern von Daten in der Datenbank

D. h.

- Einfügen neuer Zeilen
- Einfügen von Mengen von Zeilen
- Löschen von Zeilen
- Änderung von Attributwerten

4.6.1 Einfügen neuer Zeilen (Tupel)

nur zeilenweise, d.h. pro Zeile eine INSERT - Anweisung:

Syntax:

```
INSERT INTO {tabellenname|viewname}
[(liste von spaltennamen)]
{VALUES (konst. ausdruck [,konst. ausdruck...]) | SELECT-Anweisung}
```

Beispiele:

```
INSERT INTO PERSONAL
VALUES(167,' Krause','Gustav','it3',' d12',' dak');
```

```
INSERT INTO PERSONAL (PNR, NAME, VORNAME)
VALUES (777, 'Graf','Gerd');
```

Damit

- wird eine neue Zeile in die Tabelle table_name eingefügt und
- es erhalten alle nicht aufgelisteten Attribute den Wert NULL, falls **NOT NULL** vereinbart, folgt eine Fehlermeldung

Die Reihenfolge muss genau eingehalten werden.

Für fehlende Attributwerte wird **NULL** eingetragen.

4.6.2 Einfügen von Zeilen (Tupeln) aus anderen Tabellen

Mittels **SELECT-**Anweisung werden die Tupel aus einer zweiten Tabelle bestimmt, die in eine erste Tabelle eingefügt werden sollen.

Definition:

INSERT INTO tabellenname
[(liste von spaltennamen)]
SELECT-Anweisung;

Dabei gilt:

- **SELECT-**Anweisung darf nicht auf die hinter **INSERT INTO** stehende Tabelle verweisen
- **SELECT-**Anweisung ist voll nutzbar
- Anzahl der Attributnamen der **INSERT INTO-Komponente** muss mit der Anzahl derselben in der **SELECT-**Komponente übereinstimmen
- **Datentypen** müssen ebenfalls übereinstimmen.

Beispiel:

```
CREATE TABLE   PERSONAL_NEU
(PNR           integer(4), constraint PK_PNR primary key(PNR),
NAME           char(20) NOT NULL,
VORNAME        char(20));
```

```
INSERT INTO    PERSONAL_NEU
SELECT         PNR,NAME,VORNAME
FROM           PERSONAL
WHERE          KRANKENKASSE = 'aok';
```

Beispiel:

In der Tabelle PERSONAL sind alle Attribute bis auf PNR, NAME und VORNAME zu löschen.
Es existiert dazu keine direkte Möglichkeit.

Möglich in den folgenden Schritten:

Erstelle neue Tabelle mit den gewünschten Attributen:

```
CREATE TABLE HILFE
(PNR        INTEGER(4) NOT NULL,
NAME        CHAR(20) NOT NULL,
VORNAME     CHAR(20));
```

Die Zeilen der Tabelle PERSONAL werden in die Tabelle HILFE übertragen:

```
INSERT INTO   HILFE (PNR, NAME, VORNAME)
SELECT              PNR, NAME, VORNAME
FROM          PERSONAL;
```

Abfragen der Katalogtabellen, um Views, Indexe, Synonyme usw. zu bestimmen, die von der Tabelle PERSONAL abhängig sind.

Löschen der Tabelle PERSONAL

```
DROP TABLE   PERSONAL;
```

Erstellen einer neuen Tabelle PERSONAL mit der Struktur der Tabelle HILFE:

```
CREATE TABLE PERSONAL
(PNR        INTEGER(4) NOT NULL,
NAME        CHAR(20) NOT NULL,
VORNAME     CHAR(20));
```

Übertragen der Tupel von HILFE nach PERSONAL:

```
INSERT INTO   PERSONAL
SELECT        *
FROM          HILFE;
```

Löschen der Tabelle HILFE

```
DROP TABLE HILFE;
```

4.7 UPDATE - Anweisung

zur Änderung von Werten in Tupeln

Syntax:

```
UPDATE
  {tabellenname|viewname}
SET
  spaltenname1 = {ausdruck1|NULL|(SELECT-anweisung)}
  [,spaltenname2={ausdruck2|NULL|(SELECT-anw.)} ...]
[WHERE  suchbedingung]
```

UPDATE:
Es steht der Name der Tabelle, in der die Werte verändert werden sollen.

SET:
- Es stehen die Spalten, in denen Werte verändert werden sollen.
- Es stehen die Werte, die diese Spalten jetzt an Stelle der alten Werte haben sollen.
- Diese Werte können auch aus anderen Tabellen entnommen werden mittels **SELECT**.

WHERE:
Spezifiziert die Zeile(n), in denen der Spaltenwert verändert werden soll.

Verarbeitung der Anweisung:

- Für jedes Tupel wird **WHERE**-Komponente geprüft.
  Ergebnis TRUE ⇒ Kopie des Tupels wird erzeugt

- Auf Basis der Werte in der Kopie und der **SET**-Komponente werden die neuen Werte für die Attribute berechnet

- Ergebnis kommt in das ursprüngliche Tupel

- Kopie des Tupels wird gelöscht.

Beispiele:

Für den Mitarbeiter Graf werden fehlende Werte (Krankenkasse, Gehaltsstufe) eingetragen .

```
UPDATE    PERSONAL
SET       KRANKENKASSE='TKK',Geh_Stufe='it5'
WHERE     NAME='Graf';
```

In der Tabelle GEHALT sollen die Gehälter aller Stufen um 7% steigen.

```
UPDATE    GEHALT
SET       BETRAG=BETRAG * 1.07
```

Beachte:

Unterabfragen in der **WHERE**-Komponente der **UPDATE**-Anweisung dürfen nicht auf die Relation verweisen, in der Veränderungen durchgeführt werden sollen.

4.8 Löschen von Zeilen einer Tabelle

Syntax:

```
DELETE
[FROM]{tabellenname|viewname}
[WHERE  suchbedingung]
```

Damit werden

- die Zeilen einer Tabelle gelöscht, die die **WHERE** Bedingung erfüllen
- alle Zeilen gelöscht, wenn keine **WHERE -** Bedingung angegeben ist.

Unterabfragen in der **WHERE -** Komponente dürfen sich nicht auf die Tabelle beziehen, in der Zeilen gelöscht werden sollen.

Beispiele:

Lösche den Mitarbeiter mit der PNR = 167:

```
DELETE    PERSONAL
WHERE     PNR = 167
```

Lösche alle Prämien, die der Mitarbeiter Hahn erhalten hat:

```
DELETE
FROM   PRAEMIE
WHERE PNR =    (SELECT   PNR
                FROM     PERSONAL
                WHERE    NAME = 'Hahn');
```