# Grundlegende Begrifflichkeiten

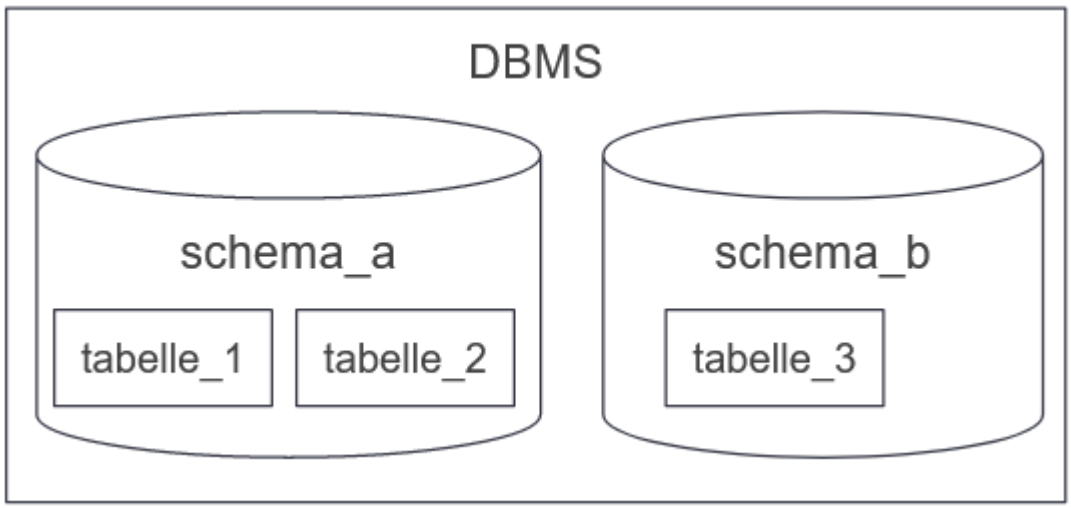
- **Datenbank** (DB)
- **Datenbankmanagementsystem** (DBMS) z.B. Oracle, SQL Server, DB2, MySQL, PostgreSQL und MariaDB
- **SQL** (Structured Query Language) als Datenbanksprache
  - SQL definiert wichtigste Befehle
  - Wird von allen großen DBMS unterstützt
  - Von Organisationen ANSI und ISO standardisiert
  - Da SQL historisch gewachsen haben viele Hersteller SQL Befehle auf ihre Produkte angepasst (bspw. Oracle oder PostgreSQL)
- **Befehlskategorien**
  - DDL (Data Definition Language)
  - DML (Data Manipulation Language)
  - DQL (Data Query Language)
  - TCL (Transaction Control Language)
  - DCL (Data Control Language)

# Data Definition Language (DDL)

Referenz: https://www.postgresql.org/docs/13/ddl.html

## Schema erstellen

- DBMS kann viele Datenmodelle enthalten
- Diese Bereiche enthalten Tabellen und werden Schemas genannt




Syntax

```
CREATE SCHEMA <Schemaname>;
```

Beispiel

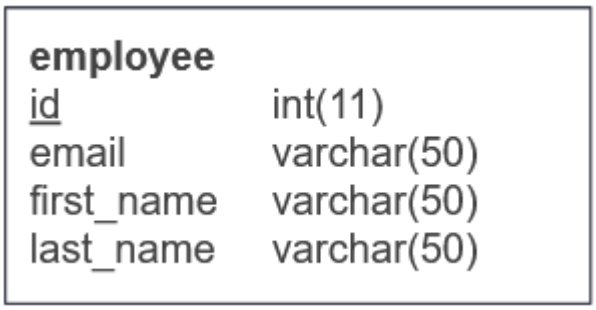
```
CREATE SCHEMA employees;
```

> *Verwenden Sie ein Schema mit* `SET SCHEMA 'employees'`.

Referenz: https://www.postgresql.org/docs/13/sql-createschema.html

## Tabelle erstellen

- Schema enthält Tabellen
- Tabellen enthalten Definitionen der Spalten
- Spaltendefinition beinhaltet Name und Datentyp
- Datentypen unterscheiden sich bei DBMS




Syntax

```
CREATE TABLE <Tabellenname> (
    <Name>    <Datentyp>,
```

```
    ...
    [PRIMARY KEY <Spaltenname>]
);
```

Beispiel

```
CREATE TABLE emp_employee (
    emp_id           INTEGER,
    emp_email        VARCHAR(50),
    emp_first_name   VARCHAR(50),
    emp_last_name    VARCHAR(50),
    PRIMARY KEY (emp_id)
);
```

Referenz: https://www.postgresql.org/docs/13/sql-createtable.html

## Typische Datentypen

- Datentypen variieren zwischen DBMS

| Datentyp | Beschreibung |
|---|---|
| CHAR(n) | Zeichenkette mit fester Länge (belegt Speicher immer) |
| VARCHAR(n) | Zeichenkette mit variabler Länge von maximal n |
| INTEGER | Ganzzahl |
| DECIMAL(n,m) | Kommazahl mit n Stellen (m Nachkommastellen) |
| DATE | Datum |
| DATETIME | Datum und Uhrzeit |
| BOOL | Wahrheitswert (wahr, falsch) |
| TEXT, MEDIUMTEXT, LONGTEXT | Große Textdaten |
| BLOB, MEDIUMBLOB, LONGBLOB | Binärdaten (Binary Large Object) |

Referenz: https://www.postgresql.org/docs/13/datatype.html

## Auto Increment

- Automatische Generierung von Sequenzen
- MySQL unterscheidet sich zu SQL Standard
- Nur ein AUTO_INCREMENT pro Tabelle
- Muss auf einen Key (z.B. Primary Key) angewandt werden
- Startet standardmäßig bei 1
- Kann niemals < 0 sein

Syntax SQL Standard

```
CREATE TABLE <Tabellenname> (
    id    INTEGER GENERATED BY DEFAULT AS IDENTITY
   (START WITH 1000 INCREMENT BY 1),
    ...
```

```
    PRIMARY KEY id
);
```

Syntax PostgreSQL

```
CREATE TABLE <Tabellenname> (
    id    SERIAL,
    ...
    PRIMARY KEY id
);
```

Syntax MySQL

```
CREATE TABLE emp_employees (
    emp_id    INTEGER AUTO_INCREMENT,
    ...
    PRIMARY KEY (emp_id)
) AUTO_INCREMENT = 1000;
```

Referenz: https://mariadb.com/kb/en/auto_increment/, https://www.postgresql.org/docs/13/datatype-numeric.html#DATATYPE-SERIAL

## Löschen und Ändern von Schemas und Tabellen

`DROP` um Schema oder Tabellen zu löschen

Syntax

```
DROP SCHEMA <Schemaname>;
DROP TABLE <Tabellenname>;
```

`ALTER`, um Schema oder Tabelle zu ändern

Syntax

```
ALTER SCHEMA <Schemaname> <Schema>;
```

Syntax Tabelle

```
ALTER TABLE <Tabellenname> <Optionen>;

<Optionen>:
    ADD <Spaltendefinition>
    MODIFY <Spaltendefinition>
    DROP <Spaltendefinition>
```

Beispiel: löschen einer Spalte

```
ALTER TABLE employee DROP COLUMN first_name;
```

Referenz: https://www.postgresql.org/docs/13/ddl-alter.html

## Generelle Verwendung

- `CREATE` um Schemas und Tabellen anzulegen (Definition des Datenlayouts)
- NICHT für das Einfügen konkreter Daten - hier werden andere Befehle genutzt
- `CREATE` wird auch für sämtliche andere Objekte genutzt (zum Beispiel Anlegen von Benutzerrechten mit `CREATE USER`)
- `ALTER`, um die DB Definition zu Ändern
- `DROP`, um DB Definitionen zu löschen

## Integritätsbedingungen

- Qualitätssicherung der Daten
- Mit Integritätsbedingungen stellt DB sicher, dass diese beim Einfügen, Ändern oder Löschen eingehalten werden
- Bei Verstoß werden Befehle nicht ausgeführt
- SQL Constraints (Bedingungen):
  - NOT NULL-Constraint
  - UNIQUE Constraint
  - CHECK Constraint

Referenz: https://www.postgresql.org/docs/13/ddl-constraints.html

### NOT NULL Constraint

- Bei Nutzung von NOT NULL darf Spalte nicht leer sein
- Mit DEFAULT kann beim Leerlassen ein Standardwert gesetzt werden
- NOT NULL und DEFAULT können kombiniert werden

Beispiel NOT NULL und DEFAULT

```
CREATE TABLE emp_employees (
    emp_id    INTEGER,
    emp_email    VARCHAR(50) NOT NULL DEFAULT 'noemail@sth.de',
    PRIMARY KEY (emp_id)
);
```

`emp_email` muss immer gesetzt werden und bekommt ansonsten den Standardwert `noemail@sth.de`.

### UNIQUE Constraint

- Werte einer Spalte muss eindeutig sein

Beispiel

```
CREATE TABLE emp_employees (
    emp_id      INTEGER,
    emp_email   VARCHAR(50) UNIQUE,
    PRIMARY KEY (emp_id)
);
```

- Alle Mitarbeiter benötigen eine eindeutige Email
- Keine Mitarbeiter können die selbe Email haben

Beispiel mit Syntaxvariante

```
CREATE TABLE emp_employees (
    emp_id      INTEGER,
```

```
    emp_email   VARCHAR(50),
    …
    CONSTRAINT eindeutig_email UNIQUE (emp_email),
    ...
    PRIMARY KEY (emp_id)
);
```

> *Constraint kann über einen Namen (hier `endeutig_email` ) referenziert werden*

Mehrere Felder lassen sich mit `UNIQUE` verbinden:

```
CONSTRAINT name_unique UNIQUE (first_name, last_name)
```

### CHECK-Constraint

Zusätzliche Regeln, welche ein Spalteneintrag erfüllen muss

Beispiel

```
CREATE TABLE emp_employees (
    emp_id      INTEGER,
    emp_email   VARCHAR(50) UNIQUE,
    emp_age     INTEGER CHECK (emp_age >= 12)
    PRIMARY KEY (emp_id)
);
```

> *eingetragene Mitarbeiter haben ein Alter von mindestens 12 Jahren*

Beispiel mit Syntaxvariante

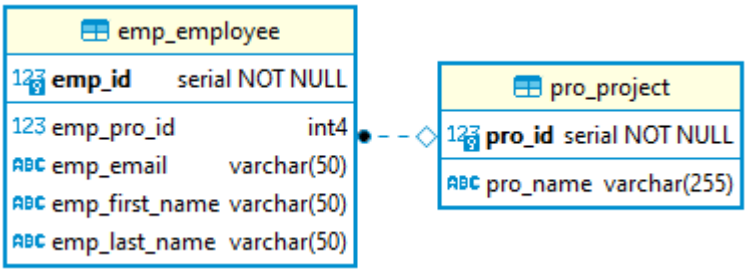
```
CREATE TABLE emp_employees (
    emp_id    INTEGER,
    emp_email    VARCHAR(50),
    emp_age    INTEGER,
    …
    CONSTRAINT altercheck CHECK (emp_age >= 16)
    ...
    PRIMARY KEY (emp_id)
);
```

> *Constraint kann über den Namen `altercheck` referenziert werden*

### Referentielle Integrität

- Abbildung von Beziehungen
- Überprüfung ob Referenzen zu anderen Tabellen eingehalten werden

> *"Der Foreign-Key steht bei der N-Entity"*




Syntax

```
CREATE TABLE pro_project (
    pro_id        SERIAL,
    pro_name    VARCHAR(255),

    PRIMARY KEY (pro_id)
);

CREATE TABLE emp_employee (
    emp_id           SERIAL,
    emp_pro_id       INTEGER,
    emp_email        VARCHAR(50),
    emp_first_name   VARCHAR(50),
    emp_last_name    VARCHAR(50),

    PRIMARY KEY (emp_id),
    FOREIGN KEY (emp_pro_id) REFERENCES pro_project(pro_id)
);
```

- Mitarbeiter sind Projekten zugewiesen
- Ein Mitarbeiter **kann ein** Projekt haben
- Durch `NOT NULL` kann eine Referenz erzwungen werden (bspw. Foreign Key muss definiert werden)

**Alternativsyntax**

`REFERENCES` kann direkt hinter dem Datentyp verwendet werden.

```
CREATE TABLE emp_employee(
    ...
    emp_pro_id INTEGER REFERENCES pro_project (pro_id),
    ...
);
```

**Verhalten von REFERENCES**

Standardverhalten: Ein Projekt, welches durch Mitarbeiter referenziert wird kann nicht gelöscht werden!

Das Verhalten kann durch die Optionen `ON DELETE` oder `ON UPDATE` gesetzt werden.

Beispiel mit `ON DELETE`

```
CREATE TABLE emp_employee(
    ...
```

```
    emp_pro_id INTEGER REFERENCES pro_project (pro_id) ON DELETE <Verhalten>,
    ...
);
```

Verhalten:

- `RESTRICT` / `NO ACTION` : Löschen eines Projekts, welches einen Mitarbeiter hat ist nicht möglich
- `CASCADE` : Wird ein Projekt gelöscht, werden alle referenzierten Mitarbeiter gelöscht
- `SET NULL` : Wird ein Projekt gelöscht, wird `emp_pro_id` des Mitarbeiters auf `NULL` gesetzt
- `SET DEFAULT` : Wird ein Projekt gelöscht, wird die `emp_pro_id` auf einen Standardwert gesetzt

**Zusammengesetzte Schlüssel**

Engl: Composite Keys
Mehrere Spalten werden zu einem Schlüssel kombiniert

Beispiel

```
CREATE TABLE emp_employee (
    emp_first_name  VARCHAR(50) NOT NULL,
    emp_last_name   VARCHAR(50) NOT NULL,

    PRIMARY KEY (emp_first_name, emp_last_name)
);
```

> *Mitarbeiter werden über einen eindeutigen Schlüssel aus Vor- und Nachname referenziert.*

# Data Modification Language (DML)

DML modifiziert Datensätze.

## Daten einfügen

Datensätze werden mit INSERT eingefügt Syntax

```
INSERT INTO <Tabellenname> ( <Spaltennamen> )
VALUES ( <Werte> )
```

Die Werte sind in der gleichen Reihenfolge wie die Spaltennamen Auto-Increment Werte werden erhöht Fehler falls Primärschlüssel bereits existiert

Beispiel

```
INSERT INTO employee (id, email, first_name, last_name)
VALUES (1, 'worker@comp.de', 'Max', 'Muster');
```

Daten ändern

Datensätze werden mit UPDATE geändert Syntax

```
UPDATE <Tabellenname>
SET <Spaltenname>=<Neuer Wert>, ...
[WHERE <Auswahlbedingung>]
```

Bezieht sich immer nur auf eine Tabelle Mit WHERE können zu ändernde Daten gefiltert werden
Auswahlbedingungen können mit AND oder OR verknüpft werden

Beispiel

```
UPDATE employee
SET last_name = 'Mustermann', first_name = 'Maxi'
WHERE email = 'worker@comp.de'
AND last_name='Muster';
```

### Daten löschen

Datensätze werden mit DELETE geändert Syntax

```
DELETE FROM <Tabellenname>
[WHERE <Auswahlbedingung>]
```

Wird keine Auswahlbedingung definiert, werden alle Daten der Tabelle gelöscht

Beispiel

```
DELETE FROM employee
WHERE email = 'worker@comp.de';
```

### Alle Daten abfragen

Datensätze werden mit SELECT abgefragt. Komplexe Abfragen behandeln wir in der nächsten Vorlesung.

SELECT ist ein Teil der Data Query Language (DQL).

Syntax

```
SELECT * FROM <Tabellenname>;
```

> *Frägt alle Daten aus der Tabelle employee ab*

Beispiel

```
SELECT * FROM employee;
```

## Sonstiges

### Kommentare

Kommentare werden mit `--` beschrieben:

```
-- erstellt Tabelle
CREATE TABLE bla (...)
```

## Skripte

## Erstellung der Mitarbeiter-Projekt Beziehung

```
-- IF EXISTS fuehrt das Kommando nur aus, wenn die TABELLE existiert
DROP TABLE IF EXISTS emp_employee;
DROP TABLE IF EXISTS pro_project;
CREATE TABLE pro_project (
    pro_id        SERIAL,
    pro_name    VARCHAR(255),

    PRIMARY KEY (pro_id)
);

CREATE TABLE emp_employee (
    emp_id           SERIAL,
    emp_pro_id       INTEGER,
    -- email muss definiert werden und ist eindeutig
    emp_email        VARCHAR(50) NOT NULL UNIQUE,
    emp_first_name   VARCHAR(50),
    emp_last_name    VARCHAR(50),

    PRIMARY KEY (emp_id),
    FOREIGN KEY (emp_pro_id) REFERENCES pro_project(pro_id)
);
```

## Datenmodifikation

```
-- Erstellen
INSERT INTO emp_employee (emp_email, emp_first_name, emp_last_name) VALUES
('nina@email.de', 'Nina', 'Haus');

-- Loeschen
DELETE FROM emp_employee WHERE emp_email = 'nina@email.de';

-- Keine Email nicht moeglich
INSERT INTO emp_employee (emp_first_name, emp_last_name) VALUES ('Nina', 'Haus');

-- Aktualisieren
UPDATE emp_employee
SET emp_last_name = 'Wohnung'
WHERE emp_email = 'nina@email.de';
```