

Datenbanksysteme

2 - SQL Teil I

Prof. Dr. Katja Zeume

katja.zeume@w-hs.de

WHS Gelsenkirchen

Bis jetzt:

- Willkommen und Motivation

Bis jetzt:

- Willkommen und Motivation

Jetzt:

- Praktische erste Schritte zum Thema Datenbanken
- Was ist eine Datenbank-Anfragesprache?
- **SQL (Structured Query Language)** als de-facto Standard für eine Relationale Datenbanksprache

In dieser Vorlesung

- Einführung in SQL
- SQL in der Vorlesung & Entwicklungsumgebung
- SQL zur Datendefinition - CREATE - DROP - ALTER
- Änderungen von Daten mit SQL - INSERT - UPDATE - DELETE
- SQL als Anfragesprache - SELECT

In dieser Vorlesung

- **Einführung in SQL**
- SQL in der Vorlesung & Entwicklungsumgebung
- SQL zur Datendefinition - CREATE - DROP - ALTER
- Änderungen von Daten mit SQL - INSERT - UPDATE - DELETE
- SQL als Anfragesprache - SELECT

Rund um ein DBMS gibt es verschiedene Aufgaben zu lösen. Dafür werden verschiedene Arten von Operationen benötigt:

Rund um ein DBMS gibt es verschiedene Aufgaben zu lösen. Dafür werden verschiedene Arten von Operationen benötigt:

- Das Datenbankschema muss umgesetzt werden, Tabellen- und Benutzersichtendefinition. → **Datendefinitionssprache (DDL)**

Rund um ein DBMS gibt es verschiedene Aufgaben zu lösen. Dafür werden verschiedene Arten von Operationen benötigt:

- Das Datenbankschema muss umgesetzt werden, Tabellen- und Benutzersichtendefinition. → **Datendefinitionssprache (DDL)**
- Es sollen Informationen aus der DB gelesen werden und vorhandene Daten geändert werden. → **Anfragesprache (DQL, DML)**

Rund um ein DBMS gibt es verschiedene Aufgaben zu lösen. Dafür werden verschiedene Arten von Operationen benötigt:

- Das Datenbankschema muss umgesetzt werden, Tabellen- und Benutzersichtendefinition. → **Datendefinitionssprache (DDL)**
- Es sollen Informationen aus der DB gelesen werden und vorhandene Daten geändert werden. → **Anfragesprache (DQL, DML)**
- Benutzerrechte sollen definiert und verwaltet werden. → **Rechteverwaltung (DCL)**

Rund um ein DBMS gibt es verschiedene Aufgaben zu lösen. Dafür werden verschiedene Arten von Operationen benötigt:

- Das Datenbankschema muss umgesetzt werden, Tabellen- und Benutzersichtendefinition. → **Datendefinitionssprache (DDL)**
- Es sollen Informationen aus der DB gelesen werden und vorhandene Daten geändert werden. → **Anfragesprache (DQL, DML)**
- Benutzerrechte sollen definiert und verwaltet werden. → **Rechteverwaltung (DCL)**
- Transaktionen (eine Menge mehrerer Anfragen) soll kontrolliert an die DB abgesetzt werden. → **Transaktionskontrolle (TCL)**

Rund um ein DBMS gibt es verschiedene Aufgaben zu lösen. Dafür werden verschiedene Arten von Operationen benötigt:

- Das Datenbankschema muss umgesetzt werden, Tabellen- und Benutzersichtendefinition. → **Datendefinitionssprache (DDL)**
- Es sollen Informationen aus der DB gelesen werden und vorhandene Daten geändert werden. → **Anfragesprache (DQL, DML)**
- Benutzerrechte sollen definiert und verwaltet werden. → **Rechteverwaltung (DCL)**
- Transaktionen (eine Menge mehrerer Anfragen) soll kontrolliert an die DB abgesetzt werden. → **Transaktionskontrolle (TCL)**

Die **Datenbanksprache SQL** umfasst mehrere dieser Teile. Im folgenden werden wir uns zunächst auf den Datendefinitions- und Anfragesprachenteil konzentrieren.

Structured Query Language (SQL)

- In den 70er Jahren von Chamberlin, Boyce und Codd entwickelt
- Ehemals SEQUEL für *System R* von IBM entwickelt

Structured Query Language (SQL)

- In den 70er Jahren von Chamberlin, Boyce und Codd entwickelt
- Ehemals SEQUEL für *System R* von IBM entwickelt
- De-facto Datenbanksprache (meist-verbreitet)

Structured Query Language (SQL)

- In den 70er Jahren von Chamberlin, Boyce und Codd entwickelt
- Ehemals SEQUEL für *System R* von IBM entwickelt
- De-facto Datenbanksprache (meist-verbreitet)
- Deklarativ, an englischer Sprache angelehnt (Endanwendernutzung)

Structured Query Language (SQL)

- In den 70er Jahren von Chamberlin, Boyce und Codd entwickelt
- Ehemals SEQUEL für *System R* von IBM entwickelt
- De-facto Datenbanksprache (meist-verbreitet)
- Deklarativ, an englischer Sprache angelehnt (Endanwendernutzung)
- Basierend auf der Relationalen Algebra

Structured Query Language (SQL)

- In den 70er Jahren von Chamberlin, Boyce und Codd entwickelt
- Ehemals SEQUEL für *System R* von IBM entwickelt
- De-facto Datenbanksprache (meist-verbreitet)
- Deklarativ, an englischer Sprache angelehnt (Endanwendernutzung)
- Basierend auf der Relationalen Algebra
- ISO und IEC standardisieren SQL

Structured Query Language (SQL)

- In den 70er Jahren von Chamberlin, Boyce und Codd entwickelt
- Ehemals SEQUEL für *System R* von IBM entwickelt
- De-facto Datenbanksprache (meist-verbreitet)
- Deklarativ, an englischer Sprache angelehnt (Endanwendernutzung)
- Basierend auf der Relationalen Algebra
- ISO und IEC standardisieren SQL
- systemabhängige Abwandlungen existieren und sind üblich

Kurzvorstellung SQL - Standard heute¹

- ISO/IEC 9075-1:2023 Part 1: Framework (SQL/Framework)
- ISO/IEC 9075-2:2023 Part 2: Foundation (SQL/Foundation)
- ISO/IEC 9075-3:2023 Part 3: Call-Level Interface (SQL/CLI)
- ISO/IEC 9075-4:2023 Part 4: Persistent stored modules (SQL/PSM)
- ISO/IEC 9075-9:2023 Part 9: Management of External Data (SQL/MED)
- ISO/IEC 9075-10:2023 Part 10: Object language bindings (SQL/OLB)
- ISO/IEC 9075-11:2023 Part 11: Information and definition schemas (SQL/Schemata)
- ISO/IEC 9075-13:2023 Part 13: SQL Routines and types using the Java TM programming language (SQL/JRT)
- ISO/IEC 9075-14:2023 Part 14: XML-Related Specifications (SQL/XML)
- ISO/IEC 9075-15:2023 Part 15: Multi-dimensional arrays (SQL/MDA)
- ISO/IEC 9075-16:**2023** Part 16: Property Graph Queries (SQL/PGQ)

¹<https://www.iso.org/home.html>

SQL Multimedia und Application Packages:

- ISO/IEC 13249-1:2016 Part 1: Framework
- ISO/IEC 13249-2:2003 Part 2: Full-Text
- ISO/IEC 13249-3:2016 Part 3: Spatial
- ISO/IEC 13249-5:2003 Part 5: Still image
- ISO/IEC 13249-6:2006 Part 6: Data mining

Mehr zur Benutzung von SQL in spezifischen Applikationen.

²<https://www.iso.org/home.html>

Beispiel Datenbank

In dieser Beispieldatenbank sind Daten über **Schauspieler**, **Regisseure** und **Filme** abgebildet.

schauspieler:

<u>sid</u>	vorname	nachname
1	Zendaya	Coleman
2	Cillian	Murphy
3	Margot	Robbie
4	Timothee	Chalamet
5	Keanu	Reeves

regisseur:

<u>rid</u>	vorname	nachname
1	Christopher	Nolan
2	Paul	King
3	Greta	Gerwig
4	Noah	Baumbach

film:

<u>fid</u>	titel	erscheinungsdatum	genre	dauer
1	Wonka	2023-12-07	Family	117
2	Dune:Part Two	2024-03-02	Action	166
3	Barbie	2023-07-20	Comedy	114
4	Oppenheimer	2023-07-20	Thriller	180
5	John Wick 4	2023-03-23	Thriller	169

- Zuvor fehlten jegliche Beziehungen zwischen zwei *Entitäten*.

- Zuvor fehlten jegliche Beziehungen zwischen zwei *Entitäten*.
- Diese können wir durch die folgenden Tabellen ausdrücken und speichern.

fuehrt_regie:

<u>fid</u>	<u>rid</u>
4	1
3	3
3	4

Beispiel Datenbank - Beziehungen

- Zuvor fehlten jegliche Beziehungen zwischen zwei *Entitäten*.
- Diese können wir durch die folgenden Tabellen ausdrücken und speichern.

fuehrt_regie:

<u>fid</u>	<u>rid</u>
4	1
3	3
3	4

spielt_in:

<u>fid</u>	<u>sid</u>	gehalt
2	1	2 Mio.
1	4	8 Mio.
5	5	1,5 Mio.

Kurzvorstellung SQL - 1. Beispielanfrage

Eine SELECT-Anfrage beschreibt welche Daten aus der Datenbank ausgewählt werden sollen.

```
SELECT fid, titel  
FROM film  
WHERE genre = 'Thriller'
```

Kurzvorstellung SQL - 1. Beispielanfrage

Eine SELECT-Anfrage beschreibt welche Daten aus der Datenbank ausgewählt werden sollen.

```
SELECT fid, titel  
FROM film  
WHERE genre = 'Thriller'
```

film:

<u>fid</u>	titel	erscheinungsdatum	genre	dauer
1	Wonka	2023-12-07	Family	117
2	Dune:Part Two	2024-03-02	Action	166
3	Barbie	2023-07-20	Comedy	114
4	Oppenheimer	2023-07-20	Thriller	180
5	John Wick 4	2023-03-23	Thriller	169

Kurzvorstellung SQL - 1. Beispielanfrage

Eine SELECT-Anfrage beschreibt welche Daten aus der Datenbank ausgewählt werden sollen.

```
SELECT fid, titel
FROM film
WHERE genre = 'Thriller'
```

film:

<u>fid</u>	titel	erscheinungsdatum	genre	dauer
1	Wonka	2023-12-07	Family	117
2	Dune:Part Two	2024-03-02	Action	166
3	Barbie	2023-07-20	Comedy	114
4	Oppenheimer	2023-07-20	Thriller	180
5	John Wick 4	2023-03-23	Thriller	169

Ausgabe:

<u>fid</u>	titel
------------	-------

Kurzvorstellung SQL - 1. Beispielanfrage

Eine SELECT-Anfrage beschreibt welche Daten aus der Datenbank ausgewählt werden sollen.

```
SELECT fid, titel
FROM film
WHERE genre = 'Thriller'
```

film:

<u>fid</u>	titel	erscheinungsdatum	genre	dauer
1	Wonka	2023-12-07	Family	117
2	Dune:Part Two	2024-03-02	Action	166
3	Barbie	2023-07-20	Comedy	114
4	Oppenheimer	2023-07-20	Thriller	180
5	John Wick 4	2023-03-23	Thriller	169

Ausgabe:

<u>fid</u>	titel
4	Oppenheimer
5	John Wick 4

Kurzvorstellung SQL - 2. Beispielanfrage

Eine SELECT-Anfrage beschreibt welche Daten aus der Datenbank ausgewählt werden sollen.

```
SELECT genre, COUNT(titel) AS anzahl  
FROM film  
GROUP BY genre  
ORDER BY anzahl
```

Kurzvorstellung SQL - 2. Beispielanfrage

Eine SELECT-Anfrage beschreibt welche Daten aus der Datenbank ausgewählt werden sollen.

```
SELECT genre, COUNT(titel) AS anzahl  
FROM film  
GROUP BY genre  
ORDER BY anzahl
```

film:				
<u>fid</u>	titel	erscheinungsdatum	genre	dauer
1	Wonka	2023-12-07	Family	117
2	Dune:Part Two	2024-03-02	Action	166
3	Barbie	2023-07-20	Comedy	114
4	Oppenheimer	2023-07-20	Thriller	180
5	John Wick 4	2023-03-23	Thriller	169

Kurzvorstellung SQL - 2. Beispielanfrage

Eine SELECT-Anfrage beschreibt welche Daten aus der Datenbank ausgewählt werden sollen.

```
SELECT genre, COUNT(titel) AS anzahl  
FROM film  
GROUP BY genre  
ORDER BY anzahl
```

film:				
<u>fid</u>	titel	erscheinungsdatum	genre	dauer
1	Wonka	2023-12-07	Family	117
2	Dune:Part Two	2024-03-02	Action	166
3	Barbie	2023-07-20	Comedy	114
4	Oppenheimer	2023-07-20	Thriller	180
5	John Wick 4	2023-03-23	Thriller	169

<u>genre</u>	<u>anzahl</u>
--------------	---------------

Kurzvorstellung SQL - 2. Beispielanfrage

Eine SELECT-Anfrage beschreibt welche Daten aus der Datenbank ausgewählt werden sollen.

```
SELECT genre, COUNT(titel) AS anzahl  
FROM film  
GROUP BY genre  
ORDER BY anzahl
```

film:

<u>fid</u>	titel	erscheinungsdatum	genre	dauer
1	Wonka	2023-12-07	Family	117
2	Dune:Part Two	2024-03-02	Action	166
3	Barbie	2023-07-20	Comedy	114
4	Oppenheimer	2023-07-20	Thriller	180
5	John Wick 4	2023-03-23	Thriller	169

genre	anzahl
Action	1
Comedy	1
Family	1
Thriller	2

Kurzvorstellung SQL - 3. Beispielanfrage

Durch eine CREATE-Anfrage wird eine neue Tabelle in der Datenbank angelegt.

```
CREATE TABLE schauspieler (  
  sid INTEGER PRIMARY KEY,  
  vorname VARCHAR(25),  
  nachname VARCHAR(25)  
)
```

Kurzworstellung SQL - 3. Beispielanfrage

Durch eine CREATE-Anfrage wird eine neue Tabelle in der Datenbank angelegt.

```
CREATE TABLE schauspieler (  
  sid INTEGER PRIMARY KEY,  
  vorname VARCHAR(25),  
  nachname VARCHAR(25)  
)
```

schauspieler:

<u>sid</u>	vorname	nachname

Kurzvorstellung SQL - 3. Beispielanfrage

Durch eine CREATE-Anfrage wird eine neue Tabelle in der Datenbank angelegt.

```
CREATE TABLE schauspieler (  
  sid INTEGER PRIMARY KEY,  
  vorname VARCHAR(25),  
  nachname VARCHAR(25)  
)
```

schauspieler:

<u>sid</u>	vorname	nachname

Es wird das **Was** beschrieben und weniger **Wie** etwas berechnet wird.
Deklarative Sichtweise anstatt **imperative** Programmierung.

Kurzvorstellung SQL - 4. Beispielanfrage

Zum Ende der Vorlesung können wir auch Anfragen der folgenden Form verstehen:

```
1  SELECT s.vorname, s.nachname,  
2         COUNT(DISTINCT f.fid) AS anzahl_filme,  
3         AVG(f.dauer) AS durchschnittliche_filmlaenge,  
4         MIN(f.erscheinungsdatum) AS erster_film,  
5         MAX(f.erscheinungsdatum) AS letzter_film,  
6         STRING_AGG(DISTINCT f.genre, ',') AS genres  
7  FROM  schauspieler s  
8        JOIN spielt_in si ON s.sid = si.sid  
9        JOIN film f ON si.fid = f.fid  
10       JOIN fuehrt_regie fr ON f.fid = fr.fid  
11       JOIN regisseur r ON fr.rid = r.rid  
12  WHERE f.genre IN ('Drama', 'Action', 'Thriller')  
13         AND (f.erscheinungsdatum BETWEEN '2000-01-01' AND '2025-12-31')  
14  GROUP BY s.sid, s.vorname, s.nachname  
15  HAVING COUNT(DISTINCT f.fid) >= 3 AND AVG(f.dauer) > 100  
16  ORDER BY anzahl_filme DESC, durchschnittliche_filmlaenge DESC;
```

In dieser Vorlesung

- Einführung in SQL
- **SQL in der Vorlesung & Entwicklungsumgebung**
- SQL zur Datendefinition - CREATE - DROP - ALTER
- Änderungen von Daten mit SQL - INSERT - UPDATE - DELETE
- SQL als Anfragesprache - SELECT

- Als **Datenbank und Relationales DBMS** nutzen wir PostgreSQL.
<https://www.postgresql.org/>

- Als **Datenbank und Relationales DBMS** nutzen wir PostgreSQL.
<https://www.postgresql.org/>
- Als GUI-Tool verwenden wir **PGAdmin**.
- Als **Entwicklungsumgebung** werden wir IntelliJ benutzen.
<https://www.jetbrains.com/idea/>

- Als **Datenbank und Relationales DBMS** nutzen wir PostgreSQL.
<https://www.postgresql.org/>
- Als GUI-Tool verwenden wir **PGAdmin**.
- Als **Entwicklungsumgebung** werden wir IntelliJ benutzen.
<https://www.jetbrains.com/idea/>

Im Moodle-Kurs finden Sie die Unterlagen zum Einrichten der jeweiligen Komponenten.

PostgreSQL - eine Objekt-Relationale DB

PostgreSQL (kurz Postgres)

- Entwickelt von Michael Stonebraker (+ Team) an der University of Berkeley California
- Open Source
- Nachfolger von Ingres - „Interactive Graphics and Retrieval System“ (auch M. Stonebraker), eines der beiden ersten RDBMS (neben *System R* von IBM), daher der Name „Post-gres“ (um 1985).
- Um 1996 Weiterentwicklung durch Open Source Community unter dem Namen PostgreSQL (gesprochen: Post-gres-Q-L)



PostgreSQL

Turing Award für Michael Stonebraker 2014^{abc}

^a <https://amturing.acm.org/byyear.cfm>

^b https://amturing.acm.org/vp/stonebraker_1172121.cfm

^c <https://dblp.org/pid/s/MichaelStonebraker.html>

Verfügbarkeit:

- Sie können die aktuellste stabile Version zur Installation nutzen.
- Folgende Plattformen werden mit fertigen Installationspaketen unterstützt:

<https://www.postgresql.org/download/>

- Linux (Debian, Red Hat/Rocky/CentOS, SUSE, Ubuntu, ...)
- Berkeley Software Distribution (OpenBSD, FreeBSD, NetBSD)
- Windows
- macOS
- Oracle Solaris

- Der Quellcode ist frei verfügbar:

<https://www.postgresql.org/ftp/source/>

Installation und Nutzung:

- Wie die meisten Datenbanken arbeitet Postgres als *Client-Server-System*.
- Die Datenbank läuft in einer Server-Anwendung.
- Wir nutzen eine Shell oder graphische Schnittstelle um mit dem Server zu kommunizieren.
- Die Kommunikation erfolgt über eine Netzwerk-Verbindung (standarmäßig Port 5432).

Anbindung an ein Programm

- Später lernen wir auch eine Möglichkeit kennen unsere Datenbank an ein Java-Programm anzubinden.
- Beispielsweise über eine JDBC-Schnittstelle:
<https://jdbc.postgresql.org/>
- Oder auch ODBC: <https://odbc.postgresql.org/>
- In modernen Anwendungen auch über ein ORM, wie beispielsweise JPA, Hibernate, oder Spring.

Shell und pgAdmin:

- Um direkt mit der Datenbank zu kommunizieren, nutzen wir entweder `psql` (Shell) oder das graphische Tool `pgAdmin` 4.

```
SQL Shell (psql)
Server [localhost]:
Database [postgres]: mynsqldb
Port [5432]:
Username [postgres]:
Passwort für Benutzer postgres:
psql (15.3)
Warnung: Konsolencodeseite (850) unterscheidet sich von der Windows-
Codeseite (1252). 8-Bit-Zeichen funktionieren möglicherweise nicht
richtig. Einzelheiten finden Sie auf der psql-Handbuchseite unter
»Notes for Windows users«.
Geben Sie »help« für Hilfe ein.
mynsqldb=#
```



- `pgAdmin` ist hauptsächlich für die Administration gedacht. Es können aber auch SQL-Anfragen gestellt werden.
- `psql` performanter und entwicklungsfreundlicher (Skripte, Remote-Zugriff, etc.).

- Wenn man `psql` aufruft, und der Server läuft nicht, bekommt man die Fehlermeldung „Verbindung zum Server auf ... fehlgeschlagen ...“.
- In pgAdmin wird ein Timeout erzeugt.
- Man kann auch schauen, ob postgres Prozesse laufen. Unter Windows mit dem Taskmanager/Dienste (Strg+Shift+Esc).
- Es ist wichtig Datenbanken vor dem Herunterfahren des Computers (Servers) ordnungsgemäß zu schließen.
- Das Programm `pg_ctl` („postgres control“) dient u.a. zum Starten und Stoppen des Servers.

PostgreSQL - Installation und Benutzung

- Nach der Installation kann ein neues leeres Datenbank(schema) auf dem Server erstellt werden.
- Erstellen Sie eine neue Datenbank (Schema) mydba.

```
SQL Shell (psql)
Server [localhost]:
Database [postgres]: mynosqlldb
Port [5432]:
Username [postgres]:
Passwort für Benutzer postgres:
psql (15.3)
Warnung: Konsolencodeseite (850) unterscheidet sich von der Windows-
Codeseite (1252). 8-Bit-Zeichen funktionieren möglicherweise nicht
richtig. Einzelheiten finden Sie auf der psql-Handbuchseite unter
»Notes for Windows users«.
Geben Sie »help« für Hilfe ein.
mynosqlldb=#
```

- Der letzte Prompt zeigt, dass Sie mit der Datenbank verbunden sind und es als Administrator (#) oder regulärer Nutzer (\$) ausführen.
- Die recht ausführliche Hilfe wird wie folgt aufgerufen:
 - \? für psql-Befehle
 - \h für Hilfe mit SQL-Befehlen

Vokabular in PostgreSQL:

- Ein **PostgreSQL Cluster** besteht aus mehreren Datenbanken, die von einer Instanz eines Datenbank-Servers verwaltet werden.
- Ein PostgreSQL Cluster entspricht einem Verzeichnis im Dateisystem, in dem alle Datenbanken gespeichert werden.
- **Anders** als in anderen Datenbanksystemen, die ein Cluster eher als Gruppe von Datenbank-Servern verstehen.

Eine **ausführliche Erklärung** für die Einrichtung erfolgt in den Praktikumsunterlagen.

In dieser Vorlesung

- Einführung in SQL
- SQL in der Vorlesung & Entwicklungsumgebung
- **SQL zur Datendefinition - CREATE - DROP - ALTER**
- Änderungen von Daten mit SQL - INSERT - UPDATE - DELETE
- SQL als Anfragesprache - SELECT

- In einer relationalen Datenbank wird zu Beginn ein **Datenbankschema** festgelegt.
- Das Datenbankschema wird durch SQL-Befehle auf der Datenbank initialisiert.

Jetzt:

- Tabellendefinition
`CREATE TABLE`
- Tabellenänderungen
`ALTER TABLE, DROP TABLE`

Zur Tabellendefinition wird die folgende Anweisung benutzt.

```
CREATE TABLE tabelle (  
    spaltenname1 wertebereich1 [ NOT NULL ],  
    ...  
    spaltennamen wertebereichn [ NOT NULL ],  
    [ PRIMARY KEY (spaltenname1, ...),]  
    [ FOREIGN KEY (spaltenname1) REFERENCES tabelle(spaltenname) ,]  
    ... )
```

Zur Tabellendefinition wird die folgende Anweisung benutzt.

```
CREATE TABLE tabelle (  
    spaltenname1 wertebereich1 [ NOT NULL ],  
    ...  
    spaltennamen wertebereichn [ NOT NULL ],  
    [ PRIMARY KEY (spaltenname1, ... ),]  
    [ FOREIGN KEY (spaltenname1) REFERENCES tabelle(spaltenname) ,]  
    ... )
```

- Zwischen eckigen Klammern [...] stehen optionale Teile.

Datendefinition - CREATE TABLE

Zur Tabellendefinition wird die folgende Anweisung benutzt.

```
CREATE TABLE tabelle (  
    spaltenname1 wertebereich1 [ NOT NULL ],  
    ...  
    spaltennamen wertebereichn [ NOT NULL ],  
    [ PRIMARY KEY (spaltenname1, ... ), ]  
    [ FOREIGN KEY (spaltenname1) REFERENCES tabelle(spaltenname) , ]  
    ... )
```

- Zwischen eckigen Klammern [...] stehen optionale Teile.
- Dies ist nicht die vollständige Syntax der CREATE TABLE-Anweisung. Hier ist warum!

Datendefinition - CREATE TABLE - Syntax

Quelle: SQL Server - <https://docs.microsoft.com/>

Folie 27 bis 31 sind nicht **klausurrelevant**.

Sie sollen sich nur davon überzeugen, dass vollständige SQL-Syntax recht umfangreich werden kann. ☺

```
syntaxsql

-- Disk-Based CREATE TABLE Syntax
CREATE TABLE
    { database_name.schema_name.table_name | schema_name.table_name | table_name }
    [ AS FileTable ]
    ( {
        <column_definition>
        | <computed_column_definition>
        | <column_set_definition>
        | [ <table_constraint> ] [ ,... n ]
        | [ <table_index> ] }
        [ ,...n ]
        [ PERIOD FOR SYSTEM_TIME ( system_start_time_column_name
            , system_end_time_column_name ) ]
    )
    [ ON { partition_scheme_name ( partition_column_name )
        | filegroup
        | "default" } ]
    [ TEXTIMAGE_ON { filegroup | "default" } ]
    [ FILESTREAM_ON { partition_scheme_name
        | filegroup
        | "default" } ]
    [ WITH ( <table_option> [ ,...n ] ) ]
[ ; ]

<column_definition> ::=
column_name <data_type>
    [ FILESTREAM ]
    [ COLLATE collation_name ]
    [ SPARSE ]
```

Datendefinition - CREATE TABLE - Syntax

```
<data_type> ::=
[ type_schema_name. ] type_name
  [ ( precision [ , scale ] | max |
    [ { CONTENT | DOCUMENT } ] xml_schema_collection ) ]

<column_constraint> ::=
[ CONSTRAINT constraint_name ]
{
  { PRIMARY KEY | UNIQUE }
    [ CLUSTERED | NONCLUSTERED ]
    [
      WITH FILLFACTOR = fillfactor
      | WITH ( <index_option> [ , ...n ] )
    ]
    [ ON { partition_scheme_name ( partition_column_name )
      | filegroup | "default" } ]

  | [ FOREIGN KEY ]
    REFERENCES [ schema_name. ] referenced_table_name [ ( ref_column ) ]
    [ ON DELETE { NO ACTION | CASCADE | SET NULL | SET DEFAULT } ]
    [ ON UPDATE { NO ACTION | CASCADE | SET NULL | SET DEFAULT } ]
    [ NOT FOR REPLICATION ]

  | CHECK [ NOT FOR REPLICATION ] ( logical_expression )
}

<column_index> ::=
INDEX index_name [ CLUSTERED | NONCLUSTERED ]
  [ WITH ( <index_option> [ , ... n ] ) ]
  [ ON { partition_scheme_name ( column_name )
    | filegroup_name
    | default
  }
  ]
  [ FILESTREAM_ON { filestream_filegroup_name | partition_scheme_name | "NULL" } ]
```

Datendefinition - CREATE TABLE - Syntax

```
<computed_column_definition> ::=  
column_name AS computed_column_expression  
[ PERSISTED [ NOT NULL ] ]  
[  
    [ CONSTRAINT constraint_name ]  
    { PRIMARY KEY | UNIQUE }  
    [ CLUSTERED | NONCLUSTERED ]  
    [  
        WITH FILLFACTOR = fillfactor  
        | WITH ( <index_option> [ , ...n ] )  
    ]  
    [ ON { partition_scheme_name ( partition_column_name )  
        | filegroup | "default" } ]  
  
    [ FOREIGN KEY ]  
    REFERENCES referenced_table_name [ ( ref_column ) ]  
    [ ON DELETE { NO ACTION | CASCADE } ]  
    [ ON UPDATE { NO ACTION } ]  
    [ NOT FOR REPLICATION ]  
  
    | CHECK [ NOT FOR REPLICATION ] ( logical_expression )  
]  
  
<column_set_definition> ::=  
column_set_name XML COLUMN_SET FOR ALL_SPARSE_COLUMNS  
  
< table_constraint > ::=  
[ CONSTRAINT constraint_name ]  
{  
    { PRIMARY KEY | UNIQUE }  
    [ CLUSTERED | NONCLUSTERED ]  
    (column [ ASC | DESC ] [ ,...n ] )  
    [  
        WITH FILLFACTOR = fillfactor  
        | WITH ( <index_option> [ , ...n ] )  
    ]  
    [ ON { partition_scheme_name (partition_column_name)  
        | filegroup | "default" } ]  
    | FOREIGN KEY  
    ( column [ ,...n ] )
```

Datendefinition - CREATE TABLE - Syntax

```
< table_index > ::=
{
    {
        INDEX index_name [ UNIQUE ] [ CLUSTERED | NONCLUSTERED ]
        (column_name [ ASC | DESC ] [ ,... n ] )
    | INDEX index_name CLUSTERED COLUMNSTORE
    | INDEX index_name [ NONCLUSTERED ] COLUMNSTORE ( column_name [ ,... n ] )
    }
    [ WITH ( <index_option> [ ,... n ] ) ]
    [ ON { partition_scheme_name ( column_name )
        | filegroup_name
        | default
        }
    ]
    [ FILESTREAM_ON { filestream_filegroup_name | partition_scheme_name | "NULL" } ]
}

<table_option> ::=
{
    [ DATA_COMPRESSION = { NONE | ROW | PAGE }
    [ ON PARTITIONS ( { <partition_number_expression> | <range> }
    [ , ...n ] ) ] ] ]
    [ FILETABLE_DIRECTORY = <directory_name> ]
    [ FILETABLE_COLLATE_FILENAME = { <collation_name> | database_default } ]
    [ FILETABLE_PRIMARY_KEY_CONSTRAINT_NAME = <constraint_name> ]
    [ FILETABLE_STREAMID_UNIQUE_CONSTRAINT_NAME = <constraint_name> ]
    [ FILETABLE_FULLPATH_UNIQUE_CONSTRAINT_NAME = <constraint_name> ]
    [ SYSTEM_VERSIONING = ON
        [ ( HISTORY_TABLE = schema_name.history_table_name
        [ , DATA_CONSISTENCY_CHECK = { ON | OFF } ]
        ) ]
    ]
    [ REMOTE_DATA_ARCHIVE =
        {
            ON [ ( <table_stretch_options> [,...n] ) ]
            | OFF ( MIGRATION_STATE = PAUSED )
        }
    ]
    [ DATA_DELETION = ON
```

Datendefinition - CREATE TABLE - Syntax

```
<ledger_option> ::=
{
    [ LEDGER_VIEW = schema_name.ledger_view_name [ ( <ledger_view_option> [,...n ] ) ]
    [ APPEND_ONLY = ON | OFF ]
}

<ledger_view_option> ::=
{
    [ TRANSACTION_ID_COLUMN_NAME = transaction_id_column_name ]
    [ SEQUENCE_NUMBER_COLUMN_NAME = sequence_number_column_name ]
    [ OPERATION_TYPE_COLUMN_NAME = operation_type_id column_name ]
    [ OPERATION_TYPE_DESC_COLUMN_NAME = operation_type_desc_column_name ]
}

<table_stretch_options> ::=
{
    [ FILTER_PREDICATE = { NULL | table_predicate_function } , ]
    MIGRATION_STATE = { OUTBOUND | INBOUND | PAUSED }
}

<index_option> ::=
{
    PAD_INDEX = { ON | OFF }
    | FILLFACTOR = fillfactor
    | IGNORE_DUP_KEY = { ON | OFF }
    | STATISTICS_NORECOMPUTE = { ON | OFF }
    | STATISTICS_INCREMENTAL = { ON | OFF }
    | ALLOW_ROW_LOCKS = { ON | OFF }
    | ALLOW_PAGE_LOCKS = { ON | OFF }
    | OPTIMIZE_FOR_SEQUENTIAL_KEY = { ON | OFF }
    | COMPRESSION_DELAY = { 0 | delay [ Minutes ] }
    | DATA_COMPRESSION = { NONE | ROW | PAGE | COLUMNSTORE | COLUMNSTORE_ARCHIVE }
      [ ON PARTITIONS ( { partition_number_expression | <range> }
        [ , ...n ] ) ]
}

<range> ::=
<partition_number_expression> TO <partition_number_expression>
```

Beispiel 1:

```
CREATE TABLE film (  
  fid INTEGER NOT NULL,  
  titel VARCHAR(50),  
  erscheinungsdatum DATE,  
  genre VARCHAR(30),  
  PRIMARY KEY (fid)  
)
```

Beispiel 1:

```
CREATE TABLE film (  
  fid INTEGER NOT NULL,  
  titel VARCHAR(50),  
  erscheinungsdatum DATE,  
  genre VARCHAR(30),  
  PRIMARY KEY (fid)  
)
```

film:

<u>fid</u>	titel	erscheinungsdatum	genre

Datendefinition - CREATE TABLE - Primärschlüssel

film:			
<u>fid</u>	titel	erscheinungsdatum	genre
1	Wonka	2023-12-07	Family
2	Dune:Part Two	2024-03-02	Action
3	Barbie	2023-07-20	Comedy

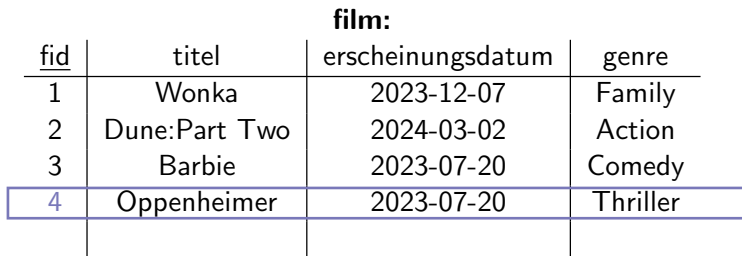
Datendefinition - CREATE TABLE - Primärschlüssel

film:			
<u>fid</u>	titel	erscheinungsdatum	genre
1	Wonka	2023-12-07	Family
2	Dune:Part Two	2024-03-02	Action
3	Barbie	2023-07-20	Comedy
4	Oppenheimer	2023-07-20	Thriller

Datendefinition - CREATE TABLE - Primärschlüssel

film:

<u>fid</u>	titel	erscheinungsdatum	genre
1	Wonka	2023-12-07	Family
2	Dune:Part Two	2024-03-02	Action
3	Barbie	2023-07-20	Comedy
4	Oppenheimer	2023-07-20	Thriller

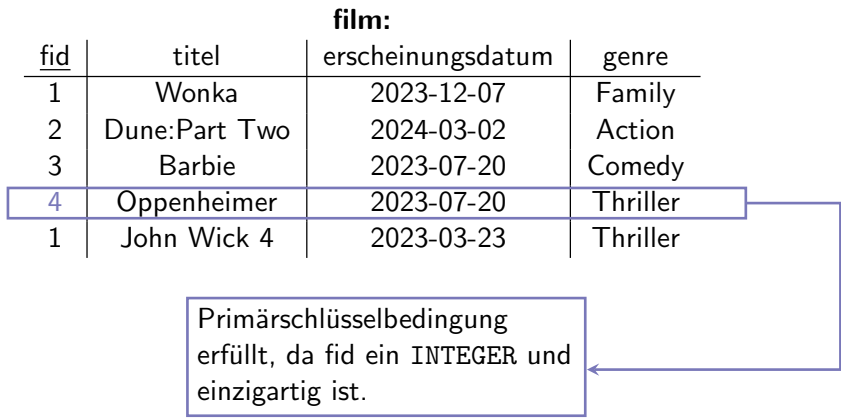


Primärschlüsselbedingung
erfüllt, da fid ein INTEGER und
einzigartig ist.

Datendefinition - CREATE TABLE - Primärschlüssel

film:

<u>fid</u>	titel	erscheinungsdatum	genre
1	Wonka	2023-12-07	Family
2	Dune:Part Two	2024-03-02	Action
3	Barbie	2023-07-20	Comedy
4	Oppenheimer	2023-07-20	Thriller
1	John Wick 4	2023-03-23	Thriller



Primärschlüsselbedingung
erfüllt, da fid ein INTEGER und
einzigartig ist.

Datendefinition - CREATE TABLE - Primärschlüssel

film:			
<u>fid</u>	titel	erscheinungsdatum	genre
1	Wonka	2023-12-07	Family
2	Dune:Part Two	2024-03-02	Action
3	Barbie	2023-07-20	Comedy
4	Oppenheimer	2023-07-20	Thriller
1	John Wick 4	2023-03-23	Thriller

Primärschlüsselbedingung erfüllt, da fid ein INTEGER und einzigartig ist.

Verletzt den Primärschlüssel, da die fid 1 bereits vergeben ist.

Beispiel 2:

```
CREATE TABLE film2 (  
    titel VARCHAR(50),  
    erscheinungsdatum DATE,  
    genre VARCHAR(30),  
    PRIMARY KEY (titel, erscheinungsdatum)  
)
```

Beispiel 2:

```
CREATE TABLE film2 (  
  titel VARCHAR(50),  
  erscheinungsdatum DATE,  
  genre VARCHAR(30),  
  PRIMARY KEY (titel, erscheinungsdatum)  
)
```

film2:

<u>titel</u>	<u>erscheinungsdatum</u>	genre

- (*titel, erscheinungsdatum*) ist ein **zusammengesetzter Primärschlüssel**.
- In dieser Modellierung können keine zwei Filme mit dem gleichen Titel am selben Tag erscheinen.

Primärschlüssel können durch verschiedene Strategien generiert werden:

- **GENERATED AS IDENTITY** - moderner SQL-Standard
- **Manuelle Sequenzen** - maximale Kontrolle
- **UUIDs** - global eindeutige IDs
- **Eigene Generatoren / Trigger** - komplexe Logik möglich

Datendefinition - CREATE TABLE - GENERATED AS IDENTITY

Empfohlene Methode (SQL-Standardkonform)

Varianten:

- GENERATED ALWAYS AS IDENTITY
- GENERATED BY DEFAULT AS IDENTITY

Beispiel

```
CREATE TABLE film (  
  fid INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,  
  titel TEXT,  
  ...  
)
```

Hinweis: Bei BY DEFAULT kann manuell ein Wert gesetzt werden.

Volle Kontrolle über Startwert und Inkrement:

Beispiel

```
CREATE SEQUENCE user_id_seq START 100;  
CREATE TABLE schauspieler (  
    sid INT PRIMARY KEY DEFAULT nextval('user_id_seq'),  
    vorname TEXT,  
    ...  
)
```

Vorteil: Flexibel, z. B. anpassbare Startwerte.

Nachteil: Verwaltung der Sequenz liegt in der Hand des Entwicklers.

Datendefinition - CREATE TABLE - UUIDs (Universally Unique Identifier)

Globale Eindeutigkeit ohne Sequenzen:

(benötigt Erweiterung)

Beispiel

```
CREATE EXTENSION IF NOT EXISTS 'uuid-oss';  
CREATE TABLE schauspieler (  
    sid UUID PRIMARY KEY DEFAULT uuid_generate_v4(),  
    name TEXT,  
    ..., )
```

Vorteile:

- Keine Kollisionen bei verteilten Systemen
- Kein zentraler Zähler nötig

Nachteil: IDs sind länger und schwerer lesbar.

Datendefinition - CREATE TABLE - Eigene Generatoren / Trigger

Komplexe ID-Logik mit Funktionen möglich

Beispiel

```
CREATE SEQUENCE film_seq;  
CREATE FUNCTION generate_film_id() RETURNS TEXT AS  
BEGIN  
    RETURN 'FID-' || to_char(NOW(), 'YYYYMMDD')  
        || '-' || nextval('film_seq');  
END;  
LANGUAGE plpgsql;  
CREATE TABLE film (  
    fid TEXT PRIMARY KEY DEFAULT generate_film_id(),  
    titel TEXT,  
    ...,  
)
```

Datendefinition - CREATE TABLE - Zusammenfassung

Strategie	Typ	Einsatzgebiet
GENERATED AS IDENTITY	Integer	Standardlösung
Manuelle Sequenzen	Integer	Flexible Kontrolle
UUID	128-bit	Verteilte Systeme
Custom Generator	Beliebig	Business-Logik, Formatierung

Grundbegriffe im Relationalen Datenbankmodell:

- Ein Schlüssel einer Relation/Tabelle T identifiziert die Tupel/Zeilen in T **eindeutig**.
- Die Spalten des Primärschlüssels sind automatisch mit NOT NULL gekennzeichnet.
- Ein Schlüssel kann aus einer Spalte bestehen, oder aus mehreren Spalten **zusammengesetzt** sein. (Composite Key)
- Schlüsselbedingungen gehören, wie auch andere Constraints, zu den Integritätsbedingungen.

Grundbegriffe im Relationalen Datenbankmodell:

- Eine Relation kann mehr als einen Schlüssel besitzen.
- Eine Relation besitzt aber immer nur genau einen **Primärschlüssel**.
 - Bestimmt die Ablage im Speicher.
 - Die Wahl ist formal beliebig.
 - Durch die vermehrte Nutzung des dazugehörenden Index machen einfache Primärschlüssel (ein Attribut) aus festen Werten (z.Bsp. IDs) oft am meisten Sinn.
- Alle anderen Schlüssel sind **alternative** oder **sekundäre** Schlüssel (durch SQL UNIQUE gekennzeichnet).

Beispiel 3:

```
CREATE TABLE spielt_in (  
    fid INTEGER NOT NULL,  
    sid INTEGER NOT NULL,  
    FOREIGN KEY (fid) REFERENCES film(fid),  
    FOREIGN KEY (sid) REFERENCES schauspieler(sid)  
)
```

Beispiel 3:

```
CREATE TABLE spielt_in (  
    fid INTEGER NOT NULL,  
    sid INTEGER NOT NULL,  
    FOREIGN KEY (fid) REFERENCES film(fid),  
    FOREIGN KEY (sid) REFERENCES schauspieler(sid)  
)
```

spielt_in:

<u>fid</u>	<u>sid</u>

Beispiel 3:

```
CREATE TABLE spielt_in (  
    fid INTEGER NOT NULL,  
    sid INTEGER NOT NULL,  
    FOREIGN KEY (fid) REFERENCES film(fid),  
    FOREIGN KEY (sid) REFERENCES schauspieler(sid)  
)
```

spielt_in:

<u>fid</u>	<u>sid</u>

- Die Tabelle besitzt einen impliziten Primärschlüssel.
- Was bewirken die Anweisungen beginnend mit FOREIGN KEY?

- Ein **Fremdschlüssel (Foreign Key)** verknüpft Datensätze zwischen Tabellen.
- Er verweist auf den **Primärschlüssel** einer anderen Tabelle.
- Sichert **Referentielle Integrität** ? d. h. keine "verwaisten" Datensätze.

Datendefinition - CREATE TABLE - Fremdschlüssel

film:

<u>fid</u>	titel	erscheinungsdatum	genre
1	Wonka	2023-12-07	Family
2	Dune:Part Two	2024-03-02	Action
3	Barbie	2023-07-20	Comedy
4	Oppenheimer	2023-07-20	Thriller
5	John Wick 4	2023-03-23	Thriller

schauspieler:

<u>sid</u>	vorname	nachname
1	Zendaya	Coleman
2	Cillian	Murphy
3	Margot	Robbie
4	Timothee	Chalamet
5	Keanu	Reeves

spielt_in:

<u>fid</u>	<u>sid</u>
2	1
1	4

Datendefinition - CREATE TABLE - Fremdschlüssel

film:

<u>fid</u>	titel	erscheinungsdatum	genre
1	Wonka	2023-12-07	Family
2	Dune:Part Two	2024-03-02	Action
3	Barbie	2023-07-20	Comedy
4	Oppenheimer	2023-07-20	Thriller
5	John Wick 4	2023-03-23	Thriller

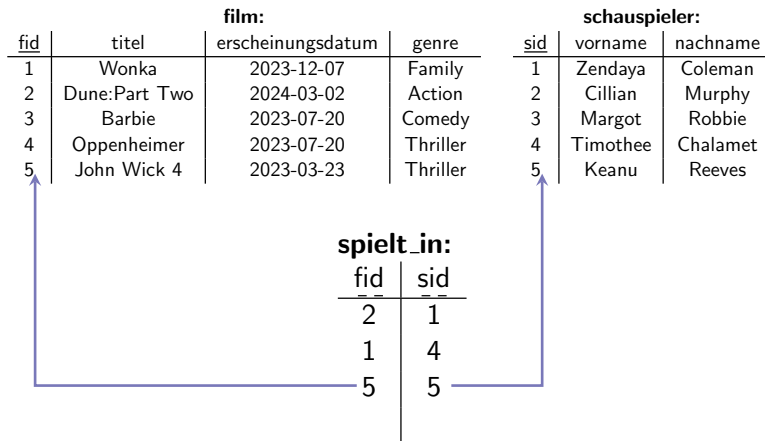
schauspieler:

<u>sid</u>	vorname	nachname
1	Zendaya	Coleman
2	Cillian	Murphy
3	Margot	Robbie
4	Timothee	Chalamet
5	Keanu	Reeves

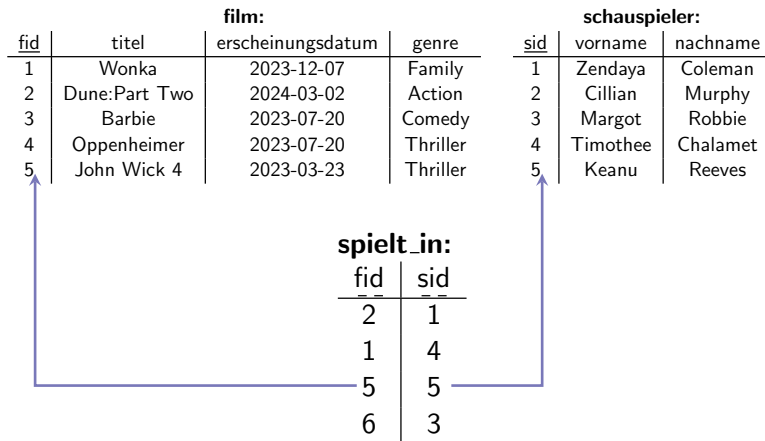
spielt_in:

<u>fid</u>	<u>sid</u>
2	1
1	4
5	5

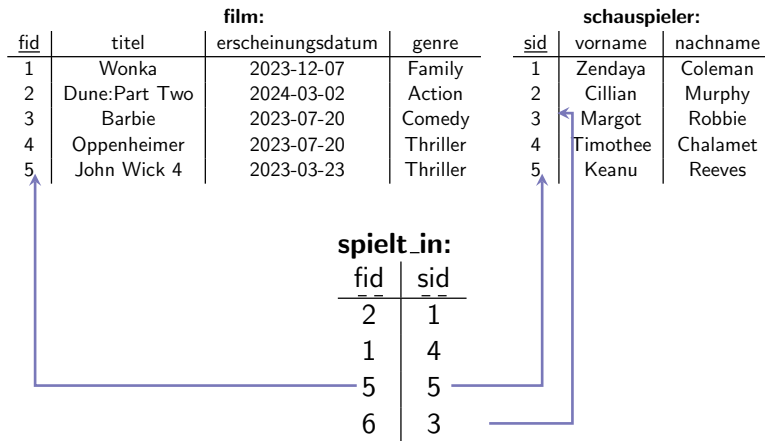
Datendefinition - CREATE TABLE - Fremdschlüssel



Datendefinition - CREATE TABLE - Fremdschlüssel



Datendefinition - CREATE TABLE - Fremdschlüssel



Datendefinition - CREATE TABLE - Fremdschlüssel

film:				schauspieler:		
<u>fid</u>	titel	erscheinungsdatum	genre	<u>sid</u>	vorname	nachname
1	Wonka	2023-12-07	Family	1	Zendaya	Coleman
2	Dune:Part Two	2024-03-02	Action	2	Cillian	Murphy
3	Barbie	2023-07-20	Comedy	3	Margot	Robbie
4	Oppenheimer	2023-07-20	Thriller	4	Timothee	Chalamet
5	John Wick 4	2023-03-23	Thriller	5	Keanu	Reeves

spielt_in:

<u>fid</u>	<u>sid</u>
2	1
1	4
5	5
6	3

fid 6 verletzt die Fremdschlüsselbeziehung, da diese ID in der Tabelle *film* nicht existiert.

Datendefinition - CREATE TABLE - Fremdschlüssel - Referentielle Aktionen

Optionen:

Beispiel

```
FOREIGN KEY (kunde_id) REFERENCES kunden(id)  
ON DELETE CASCADE  
ON UPDATE SET NULL;
```

Wichtige Varianten:

- ON DELETE CASCADE → löscht abhängige Datensätze mit
- ON DELETE SET NULL → setzt Fremdschlüssel auf NULL
- ON DELETE RESTRICT → verhindert Löschung
- ON UPDATE CASCADE → aktualisiert referenzierende Werte

Die Spalten einer Tabelle können mit Hilfe von verschiedenen Datentypen initialisiert werden.

PostgreSQL unterstützt eine große Vielfalt an Datentypen:

- **Numerische Typen** - Ganzzahlen, Fließkommazahlen, Dezimalzahlen
- **Zeichenketten** - CHAR, VARCHAR, TEXT
- **Zeit- und Datumswerte**
- **Boolesche Werte**
- **Strukturierte Typen** - ARRAY, JSON, HSTORE
- **Spezialtypen** - UUID, INET, BYTEA, GEOMETRIC

Datendefinition - CREATE TABLE - Numerische Datentypen

Ganzzahlen:

- SMALLINT - 2 Byte, Wertebereich -32.768 ... 32.767
- INTEGER - 4 Byte, Standardtyp
- BIGINT - 8 Byte, sehr große Zahlen

Fließkommazahlen:

- REAL - einfache Genauigkeit (4 Byte)
- DOUBLE PRECISION - doppelte Genauigkeit (8 Byte)

Exakte Dezimalzahlen:

- NUMERIC(10,2) - 10 Stellen insgesamt, 2 Nachkommastellen

Beispiel: Tabelle preise(betrag NUMERIC(8,2)) speichert exakte Geldbeträge.

Datendefinition - CREATE TABLE - Zeichenketten und Text

Typen:

- CHAR(n) - feste Länge
- VARCHAR(n) - variable Länge, begrenzt
- TEXT - unbegrenzt lang

Beispiel:

- vorname VARCHAR(50)
- nachname TEXT

Hinweis: In PostgreSQL ist TEXT intern genauso performant wie VARCHAR(n).

Häufige Typen:

- DATE - nur Datum 'YYYY-MM-DD'
- TIME - nur Uhrzeit 'HH:MI:SS'
- TIMETZ - Uhrzeit + Zeitzone 'HH:MI:SS+TZ' ('14:30:00-05')
- TIMESTAMP - Datum + Uhrzeit 'YYYY-MM-DD HH:MI:SS' ('2025-10-13 14:30:00')
- TIMESTAMPTZ - mit Zeitzone '2025-10-13 12:30:00 UTC'
- INTERVAL - Zeitspanne '1 day', '3 hours', ...

Datendefinition - CREATE TABLE - Boolesche und Spezialtypen

Boolesche Werte:

- BOOLEAN - TRUE, FALSE, NULL.

Spezialtypen:

- UUID - eindeutige Kennungen
- INET - IP-Adressen (IPv4 oder IPv6)
- BYTEA - Binärdaten (z. B. Dateien)

Datendefinition - CREATE TABLE - Strukturierte Datentypen

Arrays:

- Jeder Datentyp kann als Array gespeichert werden, z. B. `film_ids INT[]`.

JSON / JSONB:

- Speicherung von strukturierten Daten in JSON-Form. JSONB ist binär und effizienter.

Datendefinition - CREATE TABLE - Benutzerdefinierte Datentypen

Eigene Enumerationen:

- `CREATE TYPE status_enum
AS ENUM ('neu', 'aktiv', 'archiviert');`

Domänen (Domains):

- `CREATE DOMAIN positive_int AS INT CHECK (VALUE > 0);`

Zusammengesetzte Typen:

- `CREATE TYPE adresse AS (strasse TEXT, plz INT);`

Vorteil: Eigene Typen fördern Konsistenz und Wiederverwendung.

Tabellenänderungen - DROP TABLE

Tabellen können gelöscht werden:

```
DROP TABLE tabelle
```

Beispiel:

```
DROP TABLE film
```

Tabellenänderungen - DROP TABLE

Tabellen können gelöscht werden:

```
DROP TABLE tabelle
```

Beispiel:

```
DROP TABLE film
```

- DROP wird zurückgewiesen wenn noch Abhängigkeiten (Sichten, Schlüssel) existieren.

Tabellenänderungen - DROP TABLE

Tabellen können gelöscht werden:

```
DROP TABLE tabelle
```

Beispiel:

```
DROP TABLE film
```

- DROP wird zurückgewiesen wenn noch Abhängigkeiten (Sichten, Schlüssel) existieren.
- Wird eine Tabelle gelöscht, gehen auch alle Daten in der Tabelle unwiderruflich verloren.

Tabellenänderungen - DROP TABLE

Tabellen können gelöscht werden:

```
DROP TABLE tabelle
```

Beispiel:

```
DROP TABLE film
```

- DROP wird zurückgewiesen wenn noch Abhängigkeiten (Sichten, Schlüssel) existieren.
- Wird eine Tabelle gelöscht, gehen auch alle Daten in der Tabelle unwiderruflich verloren.
- **Hinweis:** Mit “DELETE FROM tabelle” werden (nur) alle Daten in der Tabelle gelöscht. Die Tabelle selbst bleibt leer in der DB erhalten.

Tabellenänderungen - ALTER TABLE

Tabellen können geändert werden:

```
ALTER TABLE tabelle modifikation
```

Als *modifikation* stehen weitere Anweisungen zur Verfügung:

- ADD COLUMN spaltendefinition
- DROP COLUMN spaltenname
- etc.

Beispiel:

```
ALTER TABLE schauspieler  
ADD COLUMN geburtsdatum DATE
```

schauspieler:

<u>sid</u>	vorname	nachname
1	Zendaya	Coleman
2	Cillian	Murphy
3	Margot	Robbie
4	Timothee	Chalamet
5	Keanu	Reeves

Tabellenänderungen - ALTER TABLE - Beispiele

Beispiel:

```
ALTER TABLE schauspieler  
ADD COLUMN geburtsdatum DATE
```

schauspieler:				schauspieler:			
<u>sid</u>	vorname	nachname		<u>sid</u>	vorname	nachname	geburtsdatum
1	Zendaya	Coleman	→	1	Zendaya	Coleman	
2	Cillian	Murphy		2	Cillian	Murphy	
3	Margot	Robbie		3	Margot	Robbie	
4	Timothee	Chalamet		4	Timothee	Chalamet	
5	Keanu	Reeves		5	Keanu	Reeves	

Tabellenänderungen - ALTER TABLE - Beispiele

Beispiel:

```
ALTER TABLE schauspieler  
ADD COLUMN hat_oskar BOOLEAN NOT NULL DEFAULT false
```

schauspieler:			
sid	vorname	nachname	geburtsdatum
1	Zendaya	Coleman	
2	Cillian	Murphy	
3	Margot	Robbie	
4	Timothee	Chalamet	
5	Keanu	Reeves	

Tabellenänderungen - ALTER TABLE - Beispiele

Beispiel:

```
ALTER TABLE schauspieler  
ADD COLUMN hat_oskar BOOLEAN NOT NULL DEFAULT false
```

schauspieler:			
<u>sid</u>	vorname	nachname	geburtsdatum
1	Zendaya	Coleman	
2	Cillian	Murphy	
3	Margot	Robbie	
4	Timothee	Chalamet	
5	Keanu	Reeves	



schauspieler:				
<u>sid</u>	vorname	nachname	geburtsdatum	hat_oskar
1	Zendaya	Coleman		false
2	Cillian	Murphy		false
3	Margot	Robbie		false
4	Timothee	Chalamet		false
5	Keanu	Reeves		false

Beispiel:

```
ALTER TABLE schauspieler  
DROP COLUMN geburtsdatum
```

schauspieler:				
<u>sid</u>	vorname	nachname	geburtsdatum	hat_oskar
1	Zendaya	Coleman		false
2	Cillian	Murphy		false
3	Margot	Robbie		false
4	Timothee	Chalamet		false
5	Keanu	Reeves		false

Tabellenänderungen - ALTER TABLE - Beispiele

Beispiel:

```
ALTER TABLE schauspieler  
DROP COLUMN geburtsdatum
```

schauspieler:				
<u>sid</u>	vorname	nachname	geburtsdatum	hat_oskar
1	Zendaya	Coleman		false
2	Cillian	Murphy		false
3	Margot	Robbie		false
4	Timothee	Chalamet		false
5	Keanu	Reeves		false



schauspieler:				
<u>sid</u>	vorname	nachname	hat_oskar	
1	Zendaya	Coleman	false	
2	Cillian	Murphy	false	
3	Margot	Robbie	false	
4	Timothee	Chalamet	false	
5	Keanu	Reeves	false	

Tabellenänderungen - ALTER TABLE - Beispiele

Beispiele:

```
ALTER TABLE schauspieler  
  ADD COLUMN geburtsdatum DATE
```

```
ALTER TABLE schauspieler  
  ADD COLUMN hat_oskar BOOLEAN NOT NULL DEFAULT false
```

```
ALTER TABLE schauspieler  
  DROP COLUMN geburtsdatum
```



PostgreSQL

In dieser Vorlesung

- Einführung in SQL
- SQL in der Vorlesung & Entwicklungsumgebung
- SQL zur Datendefinition - CREATE - DROP - ALTER
- **Änderungen von Daten mit SQL - INSERT - UPDATE - DELETE**
- SQL als Anfragesprache - SELECT

- Wir haben jetzt schon gesehen wie Tabellen in der Datenbank erstellt werden, und
- wie Abfragen in SQL geschrieben werden können.
- Im Folgenden schauen wir uns noch an, wie die Daten in der Datenbank geändert werden können (Einfügen, Ändern, Löschen).

Datenänderungen - INSERT

Einfügungen werden durch die INSERT-Anweisung erstellt.

```
INSERT INTO  
  tabelle (attribut1, ..., attributn)  
  VALUES (attributwert1, ..., attributwertn)
```

Datenänderungen - INSERT

Einfügungen werden durch die INSERT-Anweisung erstellt.

```
INSERT INTO  
  tabelle (attribut1, ..., attributn)  
  VALUES (attributwert1, ..., attributwertn)
```

- Attribute müssen in derselben Reihenfolge wie die Spaltenreihenfolge der Tabelle angegeben werden.

Datenänderungen - INSERT

Einfügungen werden durch die INSERT-Anweisung erstellt.

```
INSERT INTO  
  tabelle (attribut1, ..., attributn)  
  VALUES (attributwert1, ..., attributwertn)
```

- Attribute müssen in derselben Reihenfolge wie die Spaltenreihenfolge der Tabelle angegeben werden.
- Einzelnen Spalten können ausgelassen werden. Allerdings nur wenn die ausgelassenen Spalten NULL-Werte erlauben oder mit einem Default-Wert erzeugt werden.

Datenänderungen - INSERT

Einfügungen werden durch die INSERT-Anweisung erstellt.

```
INSERT INTO  
  tabelle (attribut1, ..., attributn)  
  VALUES (attributwert1, ..., attributwertn)
```

- Attribute müssen in derselben Reihenfolge wie die Spaltenreihenfolge der Tabelle angegeben werden.
- Einzelnen Spalten können ausgelassen werden. Allerdings nur wenn die ausgelassenen Spalten NULL-Werte erlauben oder mit einem Default-Wert erzeugt werden.
- Die Spaltennamen (*attribut₁, ..., attribut_n*) können auch weggelassen werden. Dann müssen aber Werte für *alle* Spalten angegeben werden.

Datenänderungen - INSERT

Beispiele:

```
INSERT INTO
  film (fid, titel, erscheinungsdatum, genre)
VALUES (6, 'Kung Fu Panda 4', '2024-03-14', 'Family')
```

Film:

<u>Film-ID</u>	Titel	Erscheinungsdatum	Genre
1	Wonka	2023-12-07	Family
2	Dune:Part Two	2024-03-02	Action
3	Barbie	2023-07-20	Comedy
4	Oppenheimer	2023-07-20	Thriller
5	John Wick 4	2023-03-23	Thriller
6	Kung Fu Panda 4	2024-03-14	Family

Datenänderungen - INSERT

Beispiele:

```
INSERT INTO thriller
(fid, titel, erscheinungsdatum, genre)
SELECT fid, titel, erscheinungsdatum, genre
FROM film
WHERE genre = 'Thriller'
```

→ Die Anweisung arbeitet dabei auf dem DB-Status von vor der Anweisung.

film:			
<u>fid</u>	titel	erscheinungsdatum	genre
1	Wonka	2023-12-07	Family
2	Dune:Part Two	2024-03-02	Action
3	Barbie	2023-07-20	Comedy
4	Oppenheimer	2023-07-20	Thriller
5	John Wick 4	2023-03-23	Thriller

thriller:			
<u>fid</u>	titel	erscheinungsdatum	genre

Datenänderungen - INSERT

Beispiele:

```
INSERT INTO thriller
(fid, titel, erscheinungsdatum, genre)
SELECT fid, titel, erscheinungsdatum, genre
FROM film
WHERE genre = 'Thriller'
```

→ Die Anweisung arbeitet dabei auf dem DB-Status von vor der Anweisung.

film:			
<u>fid</u>	titel	erscheinungsdatum	genre
1	Wonka	2023-12-07	Family
2	Dune:Part Two	2024-03-02	Action
3	Barbie	2023-07-20	Comedy
4	Oppenheimer	2023-07-20	Thriller
5	John Wick 4	2023-03-23	Thriller

thriller:			
<u>fid</u>	titel	erscheinungsdatum	genre
4	Oppenheimer	2023-07-20	Thriller
5	John Wick 4	2023-03-23	Thriller

Datenänderungen - UPDATE

Zeilen können wie folgt in einer Tabelle geändert werden.

```
UPDATE tabelle  
  SET spalte1 = value1, ...  
  WHERE bedingung
```

Beispiel:

```
UPDATE film  
  SET dauer = dauer + 10
```

- Erhöhe die Dauer von allen Filmen um 10 Minuten (z.B. wegen Werbung)

film:			
fid	titel	erscheinungsdatum	dauer
1	Wonka	2023-12-07	117
2	Dune:Part Two	2024-03-02	166
3	Barbie	2023-07-20	114
4	Oppenheimer	2023-07-20	180
5	John Wick 4	2023-03-23	169

Datenänderungen - UPDATE

Zeilen können wie folgt in einer Tabelle geändert werden.

```
UPDATE tabelle  
  SET spalte1 = value1, ...  
  WHERE bedingung
```

Beispiel:

```
UPDATE film  
  SET dauer = dauer + 10
```

- Erhöhe die Dauer von allen Filmen um 10 Minuten (z.B. wegen Werbung)

film:					film:			
<u>fid</u>	titel	erscheinungsdatum	dauer		<u>fid</u>	titel	erscheinungsdatum	dauer
1	Wonka	2023-12-07	117	→	1	Wonka	2023-12-07	127
2	Dune:Part Two	2024-03-02	166		2	Dune:Part Two	2024-03-02	176
3	Barbie	2023-07-20	114		3	Barbie	2023-07-20	124
4	Oppenheimer	2023-07-20	180		4	Oppenheimer	2023-07-20	190
5	John Wick 4	2023-03-23	169		5	John Wick 4	2023-03-23	179

Datenänderungen - DELETE

Zeilen können außerdem aus einer Tabelle gelöscht werden.

```
DELETE FROM tabelle  
WHERE bedingung
```

Beispiel:

```
DELETE FROM film  
WHERE dauer > 120
```

film:			
<u>fid</u>	titel	erscheinungsdatum	dauer
1	Wonka	2023-12-07	117
2	Dune:Part Two	2024-03-02	166
3	Barbie	2023-07-20	114
4	Oppenheimer	2023-07-20	180
5	John Wick 4	2023-03-23	169

Datenänderungen - DELETE

Zeilen können außerdem aus einer Tabelle gelöscht werden.

```
DELETE FROM tabelle  
WHERE bedingung
```

Beispiel:

```
DELETE FROM film  
WHERE dauer > 120
```

film:			
<u>fid</u>	titel	erscheinungsdatum	dauer
1	Wonka	2023-12-07	117
2	Dune:Part Two	2024-03-02	166
3	Barbie	2023-07-20	114
4	Oppenheimer	2023-07-20	180
5	John Wick 4	2023-03-23	169



film:			
<u>fid</u>	titel	erscheinungsdatum	dauer
1	Wonka	2023-12-07	117
3	Barbie	2023-07-20	114

Datenänderungen - DELETE

Wird keine WHERE-Bedingung angegeben, werden alle Zeilen in der Tabelle gelöscht. Die Tabelle wird dabei nicht gelöscht.

```
DELETE FROM film
```

film:			
<u>fid</u>	titel	erscheinungsdatum	dauer
1	Wonka	2023-12-07	117
2	Dune:Part Two	2024-03-02	166
3	Barbie	2023-07-20	114
4	Oppenheimer	2023-07-20	180
5	John Wick 4	2023-03-23	169

Datenänderungen - DELETE

Wird keine WHERE-Bedingung angegeben, werden alle Zeilen in der Tabelle gelöscht. Die Tabelle wird dabei nicht gelöscht.

```
DELETE FROM film
```

film:			
<u>fid</u>	titel	erscheinungsdatum	dauer
1	Wonka	2023-12-07	117
2	Dune:Part Two	2024-03-02	166
3	Barbie	2023-07-20	114
4	Oppenheimer	2023-07-20	180
5	John Wick 4	2023-03-23	169



film:			
<u>fid</u>	titel	erscheinungsdatum	dauer

Tabellen können mit DROP TABLE von der DB und dem Datenbankschema entfernt werden.



PostgreSQL

In dieser Vorlesung

- Einführung in SQL
- SQL in der Vorlesung & Entwicklungsumgebung
- SQL zur Datendefinition - CREATE - DROP - ALTER
- Änderungen von Daten mit SQL - INSERT - UPDATE - DELETE
- **SQL als Anfragesprache - SELECT**

Im folgenden werden SQL-Anweisungen zum Verarbeiten von Daten in der Datenbank betrachtet. Dies umfasst:

- die Selektion von Daten,
SELECT
- das Ändern und Einfügen von Daten,
INSERT und UPDATE
- und das Löschen von Daten.
DELETE

Zuvor haben wir bereits eine einfache SQL Anfrage gesehen:

```
SELECT fid, titel  
FROM film  
WHERE genre = 'Thriller'
```

- SELECT-Anweisungen spielen eine zentrale Rolle in der Datenbankanwendung.
- Der SQL-Standard zur Syntax von SELECT ist umfangreich.
- Wir schauen uns im folgenden einige mögliche Erweiterungen und Verwendungen der Anweisung an.
- **Notiz:** Seien Sie ruhig neugierig und schauen Sie einmal selbst nach mehr Funktionen für SQL Anweisungen.

Zuvor haben wir bereits eine einfache SQL Anfrage gesehen:

```
SELECT fid, titel  
      FROM film  
      WHERE genre = 'Thriller'
```

SELECT Projektionsliste (inklusive arithmetischer und Aggregatfunktionen)

FROM Gibt die verwendeten Relationen (mit Umbenennungen) an

WHERE sowohl Selektions- und Verbundbedingungen, als auch geschachtelte Anweisungen erlaubt

Syntax einfache(!) SELECT-Anweisung:

```
SELECT  $A_1, \dots, A_n$   
FROM  $R_1, \dots, R_m$   
WHERE  $C$ 
```

Semantik:

- Die Relationen R_1, \dots, R_m werden zu dem
kartesischen Produkt $R_1 \times \dots \times R_m$.
- Die Bedingungen in C filtern die Zeilen von $R_1 \times \dots \times R_m$.
- Die Attribute A_1, \dots, A_n kennzeichnen die Spalten, die im Ergebnis zurückgegeben werden.

Beispiel *-Selektion:

```
SELECT * FROM film  
WHERE erscheinungsdatum = '2023-07-20'
```

- Selektiert alle Spalten / Attribute der Tabelle film die die Bedingung im WHERE-Teil erfüllen.

Beispiel *-Selektion:

```
SELECT * FROM film  
WHERE erscheinungsdatum = '2023-07-20'
```

→ Selektiert alle Spalten / Attribute der Tabelle film die die Bedingung im WHERE-Teil erfüllen.

film:			
<u>fid</u>	titel	erscheinungsdatum	genre
1	Wonka	2023-12-07	Family
2	Dune:Part Two	2024-03-02	Action
3	Barbie	2023-07-20	Comedy
4	Oppenheimer	2023-07-20	Thriller
5	John Wick 4	2023-03-23	Thriller

Datenselektion - SELECT - Beispiele

Beispiel *-Selektion:

```
SELECT * FROM film  
WHERE erscheinungsdatum = '2023-07-20'
```

→ Selektiert alle Spalten / Attribute der Tabelle film die die Bedingung im WHERE-Teil erfüllen.

film:			
<u>fid</u>	titel	erscheinungsdatum	genre
1	Wonka	2023-12-07	Family
2	Dune:Part Two	2024-03-02	Action
3	Barbie	2023-07-20	Comedy
4	Oppenheimer	2023-07-20	Thriller
5	John Wick 4	2023-03-23	Thriller



film:			
<u>fid</u>	titel	erscheinungsdatum	genre
3	Barbie	2023-07-20	Comedy
4	Oppenheimer	2023-07-20	Thriller

Beispiel DISTINCT

```
SELECT DISTINCT fid  
FROM spielt_in
```

→ Entfernt alle Duplikate aus der Ergebnistabelle.

Beispiel DISTINCT

```
SELECT DISTINCT fid  
FROM spielt_in
```

→ Entfernt alle Duplikate aus der Ergebnistabelle.

spielt_in:

<u>fid</u>	<u>sid</u>
2	1
1	4
5	4
2	1
5	5

Datenselektion - SELECT - Beispiele

Beispiel DISTINCT

```
SELECT DISTINCT fid  
FROM spielt_in
```

→ Entfernt alle Duplikate aus der Ergebnistabelle.

spielt_in:			spielt_in:	
<u>fid</u>	<u>sid</u>		<u>fid</u>	
2	1	→	2	
1	4		1	
5	4		5	
2	1			
5	5			



Datenselektion - SELECT - WHERE allgemein

Einschränkung der Ergebnismenge durch logische Bedingungen:

Grundsyntax:

```
SELECT * FROM tabelle WHERE bedingung;
```

Typische Vergleichsoperatoren:

- =, <>, !=, <, >, <=, >= - Logische Vergleiche
- BETWEEN a AND b - Bereichsprüfung
- IN (?), NOT IN (?) - Werteliste
- LIKE - Mustervergleich (z. B. 'A%')
- IS NULL / IS NOT NULL

Beispiele:

- WHERE name LIKE 'A%'
- WHERE ort IN ('Berlin', 'Hamburg')
- WHERE datum BETWEEN '2025-01-01' AND '2025-12-31'
- WHERE aktiv IS TRUE

Tipp: Bedingungen können mit AND, OR und NOT kombiniert werden.

Bis jetzt:

- Sehr einfacher Rohbau einer SELECT-Anweisung.

Bis jetzt:

- Sehr einfacher Rohbau einer SELECT-Anweisung.

Heute und in der Vorlesung:

- Im Folgenden schauen wir uns die einzelnen Teile der Anweisung noch einmal genauer an.
- Später in der Vorlesung betrachten wir immer mehr Möglichkeiten komplexere SELECT-Anweisungen zu schreiben.

Beispiel 1:

```
SELECT DISTINCT titel, rid  
  FROM film AS f, fuehrt_Regie AS r  
 WHERE f.fid = r.fid
```

- Zur besseren Lesbarkeit können in einer SQL-Anfrage auch Variablen (bzw. Aliase) eingeführt werden.

Beispiel 1:

```
SELECT DISTINCT titel, rid  
  FROM film AS f, fuehrt_Regie AS r  
 WHERE f.fid = r.fid
```

- Zur besseren Lesbarkeit können in einer SQL-Anfrage auch Variablen (bzw. Aliase) eingeführt werden.
- Wird kein Alias angegeben, existiert automatisch ein Alias mit dem Namen der Tabelle
`FROM table ≡ FROM table AS table`

Beispiel 1:

```
SELECT DISTINCT titel, rid  
  FROM film AS f, fuehrt_Regie AS r  
 WHERE f.fid = r.fid
```

- Zur besseren Lesbarkeit können in einer SQL-Anfrage auch Variablen (bzw. Aliase) eingeführt werden.
- Wird kein Alias angegeben, existiert automatisch ein Alias mit dem Namen der Tabelle
`FROM table ≡ FROM table AS table`
- Wird ein neuer Alias angegeben, wird der implizite Alias (gleich dem Tabellennamen) aber nicht erzeugt.

Beispiel 1:

```
SELECT DISTINCT titel, rid  
  FROM film AS f, fuehrt_Regie AS r  
 WHERE f.fid = r.fid
```

- Zur besseren Lesbarkeit können in einer SQL-Anfrage auch Variablen (bzw. Aliase) eingeführt werden.
- Wird kein Alias angegeben, existiert automatisch ein Alias mit dem Namen der Tabelle
`FROM table ≡ FROM table AS table`
- Wird ein neuer Alias angegeben, wird der implizite Alias (gleich dem Tabellennamen) aber nicht erzeugt.
- Das Schlüsselwort AS ist optional.

Beispiel 1:

```
SELECT DISTINCT titel, rid
  FROM film AS f, fuehrt_Regie AS r
 WHERE f.fid = r.fid
```

- Zur besseren Lesbarkeit können in einer SQL-Anfrage auch Variablen (bzw. Aliase) eingeführt werden.
- Wird kein Alias angegeben, existiert automatisch ein Alias mit dem Namen der Tabelle
`FROM table` \equiv `FROM table AS table`
- Wird ein neuer Alias angegeben, wird der implizite Alias (gleich dem Tabellennamen) aber nicht erzeugt.
- Das Schlüsselwort AS ist optional.

titel	rid
Barbie	3
Barbie	4
Oppenheimer	1

Datenselektion - SELECT - Aliase (AS)

Beispiel 2:

film:			
<u>fid</u>	titel	erscheinungsdatum	genre
1	Wonka	2023-12-07	Family
2	Dune:Part Two	2024-03-02	Action
3	Barbie	2023-07-20	Comedy
4	Oppenheimer	2023-07-20	Thriller
5	John Wick 4	2023-03-23	Thriller

fuehrt_Regie:		
<u>fid</u>	<u>rid</u>	genre
3	3	Comedy
3	4	Comedy
4	1	Thriller

Datenselektion - SELECT - Aliase (AS)

Beispiel 2:

film:				fuehrt_Regie:		
<u>fid</u>	titel	erscheinungsdatum	genre	<u>fid</u>	<u>rid</u>	genre
1	Wonka	2023-12-07	Family	3	3	Comedy
2	Dune:Part Two	2024-03-02	Action	3	4	Comedy
3	Barbie	2023-07-20	Comedy	4	1	Thriller
4	Oppenheimer	2023-07-20	Thriller			
5	John Wick 4	2023-03-23	Thriller			

- Die Aliase für titel und rid können weggelassen werden, jedoch nicht der Alias für genre

Datenselektion - SELECT - Aliase (AS)

Beispiel 2:

film:				fuehrt_Regie:		
<u>fid</u>	titel	erscheinungsdatum	genre	<u>fid</u>	<u>rid</u>	genre
1	Wonka	2023-12-07	Family	3	3	Comedy
2	Dune:Part Two	2024-03-02	Action	3	4	Comedy
3	Barbie	2023-07-20	Comedy	4	1	Thriller
4	Oppenheimer	2023-07-20	Thriller			
5	John Wick 4	2023-03-23	Thriller			

- Die Aliase für titel und rid können weggelassen werden, jedoch nicht der Alias für genre
- Da dieser in beiden Tabellen die gleiche Bezeichnung hat, muss spezifiziert werden welche Tabelle gemeint ist.

Datenselektion - SELECT - Aliase (AS)

Beispiel 2:

film:				fuehrt_Regie:		
<u>fid</u>	titel	erscheinungsdatum	genre	<u>fid</u>	<u>rid</u>	genre
1	Wonka	2023-12-07	Family	3	3	Comedy
2	Dune:Part Two	2024-03-02	Action	3	4	Comedy
3	Barbie	2023-07-20	Comedy	4	1	Thriller
4	Oppenheimer	2023-07-20	Thriller			
5	John Wick 4	2023-03-23	Thriller			

- Die Aliase für titel und rid können weggelassen werden, jedoch nicht der Alias für genre
- Da dieser in beiden Tabellen die gleiche Bezeichnung hat, muss spezifiziert werden welche Tabelle gemeint ist.

```
SELECT DISTINCT fid, rid, f.genre
FROM film AS f, fuehrt_Regie AS r
WHERE f.fid = r.fid
```

Datenselektion - SELECT - Aliase (AS)

Beispiel 2:

film:				fuehrt_Regie:		
<u>fid</u>	titel	erscheinungsdatum	genre	<u>fid</u>	<u>rid</u>	genre
1	Wonka	2023-12-07	Family	3	3	Comedy
2	Dune:Part Two	2024-03-02	Action	3	4	Comedy
3	Barbie	2023-07-20	Comedy	4	1	Thriller
4	Oppenheimer	2023-07-20	Thriller			
5	John Wick 4	2023-03-23	Thriller			

- Die Aliase für titel und rid können weggelassen werden, jedoch nicht der Alias für genre
- Da dieser in beiden Tabellen die gleiche Bezeichnung hat, muss spezifiziert werden welche Tabelle gemeint ist.

```
SELECT DISTINCT fid, rid, f.genre
FROM film AS f, fuehrt_Regie AS r
WHERE f.fid = r.fid
```

titel	rid	genre
Oppenheimer	1	Thriller
Barbie	3	Family

Beispiel 3:

Beispiel 3:

```
SELECT fid, titel,  
       dauer+10 AS kinolaenge,  
       '3D' AS vorstellung  
FROM film
```

- Aliase können in Verbindung mit arithmetischen Ausdrücken, Konstanten oder Umbenennungen verwendet werden.

Beispiel 3:

```
SELECT fid, titel,  
       dauer+10 AS kinolaenge,  
       '3D' AS vorstellung  
FROM film
```

- Aliase können in Verbindung mit arithmetischen Ausdrücken, Konstanten oder Umbenennungen verwendet werden.

film:					film:			
fid	titel	erscheinungsdatum	dauer		fid	titel	kinolaenge	vorstellung
1	Wonka	2023-12-07	117	→	1	Wonka	127	3D
2	Dune:Part Two	2024-03-02	166		2	Dune:Part Two	176	3D
3	Barbie	2023-07-20	114		3	Barbie	124	3D
4	Oppenheimer	2023-07-20	180		4	Oppenheimer	190	3D
5	John Wick 4	2023-03-23	169		5	John Wick 4	179	3D

In der WHERE-Klausel können Bedingungen angegeben werden, die bestimmte Tupel aus den selektierten Tabellen filtern.

Beispiel:

```
SELECT DISTINCT titel, rid  
  FROM film AS f, fuehrt_Regie AS r  
 WHERE f.fid = r.fid
```

In der WHERE-Klausel können Bedingungen angegeben werden, die bestimmte Tupel aus den selektierten Tabellen filtern.

Beispiel:

```
SELECT DISTINCT titel, rid
  FROM film AS f, fuehrt_Regie AS r
 WHERE f.fid = r.fid
```

Was passiert wenn kein Filter angegeben wird?

In der Beispielanfragen wurden zwei Tabellen gleichzeitig selektiert.

```
SELECT DISTINCT titel, rid  
FROM film AS f, fuehrt_regie AS r
```

- Ohne die WHERE-Klausel wird die Anfrage über das *kartesische Produkt* von film und fuehrt_regie ausgewertet.

Exkurs - Kartesisches Produkt

Definition

Sei R_1 eine n -stellige und R_2 eine m -stellige Relation für beliebige $n, m \in \mathbb{N}$. Das **kartesische Produkt** (oder **Kreuzprodukt**) $R_1 \times R_2$ ist definiert als:

$$R_1 \times R_2 = \{(x_1, \dots, x_n, y_1, \dots, y_m) \mid (x_1, \dots, x_n) \in R_1 \wedge (y_1, \dots, y_m) \in R_2\}$$

film:				fuehrt_regie:	
<u>fid</u>	titel	erscheinungsdatum	genre	<u>fid</u>	<u>rid</u>
1	Wonka	2023-12-07	Family	4	1
2	Dune:Part Two	2024-03-02	Action	3	3
3	Barbie	2023-07-20	Comedy	3	4

film:					
<u>fid</u>	titel	erscheinungsdatum	genre	<u>fid</u>	<u>rid</u>
1	Wonka	2023-12-07	Family	4	1
2	Dune:Part Two	2024-03-02	Action	4	1
3	Barbie	2023-07-20	Comedy	4	1
1	Wonka	2023-12-07	Family	3	3
2	Dune:Part Two	2024-03-02	Action	3	3
3	Barbie	2023-07-20	Comedy	3	3
1	Wonka	2023-12-07	Family	3	4
2	Dune:Part Two	2024-03-02	Action	3	4
3	Barbie	2023-07-20	Comedy	3	4

Exkurs - Kartesisches Produkt mit JOIN

```
SELECT * FROM film AS f
  JOIN fuehrt_regie AS r
    ON f.fid = r.fid
```

- Die Verwendung von JOIN ON-Bedingungen statt WHERE f.fid = r.fid ermöglicht eine optimierte Abfrageleistung durch die Verknüpfung der Tabellen direkt auf Datenbankebene.

film:				fuehrt_regie:	
<u>fid</u>	titel	erscheinungsdatum	genre	<u>fid</u>	<u>rid</u>
1	Wonka	2023-12-07	Family	4	1
2	Dune:Part Two	2024-03-02	Action	3	3
3	Barbie	2023-07-20	Comedy	3	4

film:					
<u>fid</u>	titel	erscheinungsdatum	genre	<u>fid</u>	<u>rid</u>
1	Wonka	2023-12-07	Family	1	4
2	Dune:Part Two	2024-03-02	Action	2	1

Bis jetzt verhalten sich die gesehenen SQL Anfragen **monoton**
→ (siehe Kapitel Relationale Algebra).

Bis jetzt verhalten sich die gesehenen SQL Anfragen **monoton**
→ (siehe Kapitel Relationale Algebra).

Beispiel für eine nicht-monotone Anfrage:

Selektiere alle schauspieler, die in noch keinem film mitgespielt haben.

- Diese Anfrage ist *nicht-monoton*. Werden neue Zeilen zu der Tabelle *hoert* hinzugefügt, enthält die Anfrage evtl. weniger Studenten.
- Diese Anfrage können wir zur Zeit noch nicht beantworten.

Wie das geht, sehen wir jetzt!

Datenselektion - IN und NOT IN

Selektiere alle schauspieler, die in keinem film mitgespielt haben.

```
SELECT sid, vorname, nachname  
FROM schauspieler  
WHERE sid NOT IN (  
    SELECT sid FROM spielt_in )
```

Datenselektion - IN und NOT IN

Selektiere alle schauspieler, die in keinem film mitgespielt haben.

```
SELECT sid, vorname, nachname  
FROM schauspieler  
WHERE sid NOT IN (  
        SELECT sid FROM spielt_in )
```

- Durch das Schlüsselwort NOT IN wird getestet ob ein Attribut in der Ergebnisrelation der angegebenen **geschachtelten Anfrage** liegt (oder nicht liegt).
- Die sid in der Unteranfrage braucht keinen Alias, weil sie unabhängig von der Hauptanfrage abgearbeitet wird.

Datenselektion - IN und NOT IN

Selektiere alle schauspieler, die in keinem film mitgespielt haben.

```
SELECT sid, vorname, nachname  
FROM schauspieler  
WHERE sid NOT IN (  
    SELECT sid FROM spielt_in )
```

- Durch das Schlüsselwort NOT IN wird getestet ob ein Attribut in der Ergebnisrelation der angegebenen **geschachtelten Anfrage** liegt (oder nicht liegt).
- Die sid in der Unteranfrage braucht keinen Alias, weil sie unabhängig von der Hauptanfrage abgearbeitet wird.



PostgreSQL

Datenselektion - IN und NOT IN

- Die Unteranfrage wird ausgewertet und es wird eine Liste von sid zurückgegeben

```
SELECT sid
FROM spielt_in
```

- Jetzt werden sid, vorname und nachname aus der Tabelle schauspieler ausgelesen und jede übereinstimmende Zeile verworfen

```
SELECT sid, vorname, nachname
FROM schauspieler
WHERE sid NOT IN (
    SELECT sid FROM spielt_in )
```

schauspieler:					schauspieler:		
<u>sid</u>	vorname	nachname		<u>sid</u>	<u>sid</u>	vorname	nachname
1	Zendaya	Coleman	⋈	1	2	Cillian	Murphy
2	Cillian	Murphy		4	3	Margot	Robbie
3	Margot	Robbie		5			
4	Timothee	Chalamet					
5	Keanu	Reeves					

Eine **Unteranfrage (Subquery)** ist eine SQL-Abfrage, die in einer anderen Abfrage verschachtelt ist. Mögliche Einsatzmöglichkeiten:

- Im WHERE-Teil zur Filterung nach Ergebnissen anderer Abfragen
- Im FROM-Teil als virtuelle Tabelle (*Derived Table*)
- Im SELECT-Teil zur Berechnung einzelner Werte

Formen:

- **Skalare Unteranfrage:** liefert genau einen Wert → z.B. `WHERE preis > (SELECT AVG(preis) FROM produkte)`
- **Mehrzeilige Unteranfrage:** liefert mehrere Werte → Verwendung mit `IN`, `ANY`, `ALL`
- **Korrelierte Unteranfrage:** Mit Bezug auf die äußere Abfrage

Tipp: Unteranfragen können oft durch JOINS ersetzt werden - das ist meist performanter.

Unteranfragen können auch ineinander **verschachtelt** werden.

Abfrage:

```
SELECT k.name,  
FROM kunden k  
WHERE (SELECT SUM(b.preis)  
       FROM besuche b  
       WHERE b.kunde_id = k.id) >  
       (SELECT AVG(summe)  
        FROM (SELECT SUM(b2.preis) AS summe  
               FROM besuche b2  
               GROUP BY b2.kunde_id));
```

Erläuterung (Nummerierung von Außen nach Innen):

- Unteranfrage 3: berechnet die *Gesamtausgaben* pro Kunde
- Unteranfrage 2: berechnet die Durchschnittsausgaben pro Kunde
- Unteranfrage 1: liefert die Gesamtausgaben pro Kunde und zieht den Vergleich zum Durchschnitt

In dieser Vorlesung

- Einführung in SQL
- SQL in der Vorlesung & Entwicklungsumgebung
- SQL zur Datendefinition - CREATE - DROP - ALTER
- Änderungen von Daten mit SQL - INSERT - UPDATE - DELETE
- SQL als Anfragesprache - SELECT

- Heuer, Sattler, Saake. Datenbanken: Konzepte und Sprachen. mitp-Verlag
- Saake, Heuer: Datenbanken - Implementierungstechniken, mitp-Verlag, 2005
- Serge Abiteboul, Rick Hull, Victor Vianu. Foundations of Databases. 1995
- Beispiele und Folien basieren teilweise auf Folien von Prof. Schwentick
- Beispiele und Folien basieren teilweise auf Folien von Prof. Teubner TU Dortmund
<http://dbis.cs.tu-dortmund.de/cms/de/lehre/ss17/infosys/index.html>
- Beispiele und Folien basieren teilweise auf Folien von Prof. Naumann HPI
<https://hpi.de/naumann/teaching/teaching/ss13/datenbanksysteme-i.html>