Leibniz Universität IT Services

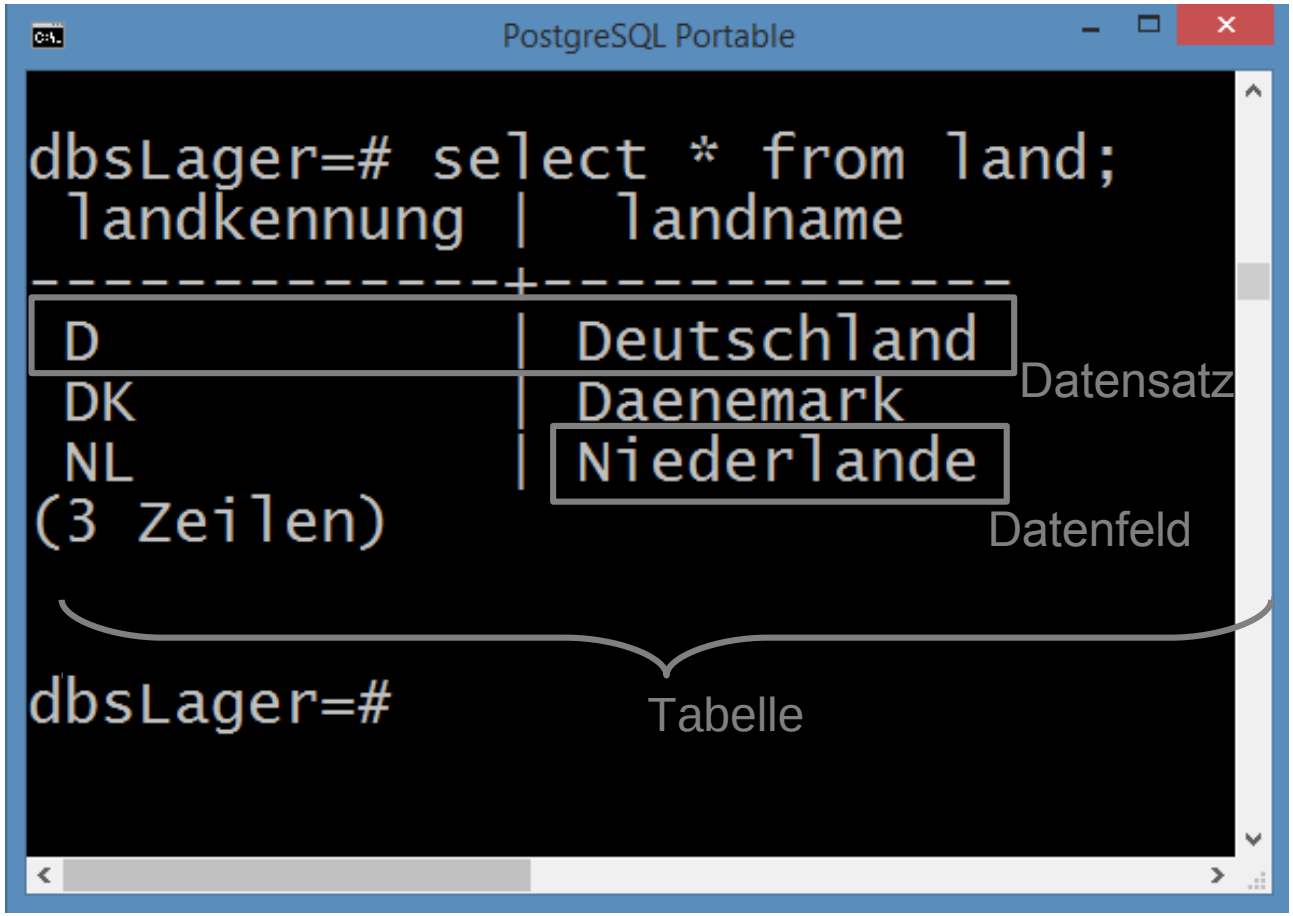
Leibniz Universität Hannover

# S(tructured)Q(uery)L(anguage) Tabellen mit PostgreSQL erstellen

# Tabelle

- Container, um Elemente einer bestimmten Gruppe zu sammeln.
- In den Spalten werden die Attribute der zu speichernden Objekte abgebildet. Objekte in einer Tabelle haben alle die gleichen Attribute.
- Pro Zeile wird ein Datensatz abgelegt. Ein Datensatz beschreibt ein konkretes Objekt aus der realen Welt. Mindestens ein Attribut-Wert in einem Datensatz unterscheidet sich von allen anderen.

# Aufbau einer Tabelle



# Erzeugung einer neuen Tabelle

```
postgres =# CREATE TABLE Land(
postgres =# );
```

- Der Befehl `CREATE TABLE` erzeugt eine neue Tabelle.
- Dem Befehl folgt der Name der neuen Tabelle. Der Name identifiziert eindeutig eine Tabelle in einer Datenbank
- Die runden Klammern sind leer. Datenfelder in der Tabelle werden später definiert.

# ..., wenn diese nicht existiert ...

```
postgres =# CREATE TABLE IF NOT EXISTS Land(
postgres =# );
```

- Erzeuge die Tabelle wenn (`IF`) sie nicht (`NOT`) existiert (`EXISTS`) …
- Andernfalls wird die Anweisung abgebrochen. Ein Fehler wird nicht angezeigt.

# … plus die Datenstruktur

```
postgres =# CREATE TABLE IF NOT EXISTS Land(
postgres =# landkuerzel VARCHAR(3) PRIMARY KEY,
postgres =# landname VARCHAR(150)
postgres =# );
```

- Die Tabelle und deren Struktur wird festglegt.
- In den runden Klammern werden die Datenfelder in der Tabelle definiert.
- Die Datenfelder werden als Spalten in einer Tabelle dargestellt.

# Liste von Datenfeldern

```
postgres =# CREATE TABLE Land(
postgres =# landkuerzel VARCHAR(3) PRIMARY KEY,
postgres =# landname VARCHAR(150)
postgres =# );
```

- Die runden Klammern fassen eine Liste von Felddefinitionen zusammen.
- Die Felddefinitionen werden in der Liste durch ein Komma getrennt.

# Felddefinition

| | | |
|---|---|---|
| `landkuerzel` | `VARCHAR(3)` | `PRIMARY KEY` |
| `landname` | `VARCHAR(150)` | |
| `feldname` | `feldtyp` | `attribute` |

- Der Name des Feldes ist in einer Tabelle eindeutig.
- Jedes Feld hat einen Feldtyp. Der Datentyp beschreibt die Nutzung des Feldes.
- Ein Feld kann weitere Attribute haben. In diesem Beispiel hat das Feld `landkuerzel` das Attribut „Schlüssel“.

# Hinweise zu Bezeichnern

- Der Feldname oder der Tabellenname beginnt mit einem Buchstaben.
- Ein Bezeichner besteht aus den lateinischen Groß- und Kleinbuchstaben und den Ziffern.
- Ein Bezeichner enthält als Sonderzeichen nur den Unterstrich.
- Ein Bezeichner sollte nicht länger als 18 Zeichen sein. Die maximale Länge ist abhängig vom verwendeten System.
- SQL-Befehle können nicht als Bezeichner genutzt werden.

# Datentypen eines Feldes

```
postgres =# CREATE TABLE Land(
postgres =# landkuerzel VARCHAR(3) PRIMARY KEY,
postgres =# landname VARCHAR(150)
postgres =# );
```

- Dem Namen des Datenfeldes folgt der Datentyp.
- Der Datentyp und der Name des Feldes werden durch ein Leerzeichen getrennt.
- Die Implementierung der verschiedenen Datentypen ist abhängig von dem genutzten Datenbanksystem.

# In Abhängigkeit des Datentyps wird ...

- der Speicherbedarf berechnet. Die Größe des Containers, in der der Attribut-Wert abgelegt wird, wird festgelegt.
- der Wertebereich für ein Attribut-Wert festgelegt.
- die Nutzung des Datenfeldes definiert. Der Datentyp `varchar()` kann nicht in Berechnungen genutzt werden.

# Datentypen in PostgresSQL

- Numerische Datentypen. Ablage von Ganz- und Fließkommazahlen zur Berechnung von neuen Werten oder Darstellung von Währungswerten.
- Text für die Ablage von Daten, die aus alphanumerischen und numerischen Daten bestehen. Zum Beispiel Postleitzahlen werden als Text abgelegt.
- Datums- und Zeitangaben.
- Ablage von Ja - / Nein- Werten.
- Ablage von Binärdaten.

# Informationen im Web

- https://www.postgresql.org/docs/9.6/static/datatype.html
- http://cornelia-boenigk.de/pg/pg-datentypen.pdf

# Ganzzahlen

- Zahlen ohne Nachkommastellen.
- Führende Nullen werden entfernt.
- Positive Ganzzahlen als Literale: 3, +13456 und so weiter.
  Negative Ganzzahlen als Literale: -5, -3456 und so weiter.
- Der Datentyp legt den Wertebereich der Ganzzahl fest. Der Wertebereich beschreibt den Zahlenraum, in dem der Inhalt des Datenfeldes liegen darf.

# Datentypen

| Datentyp | Wertebereich | Speicherbedarf |
|---|---|---|
| `Smallint` | -32768 bis +32767 | 2 Byte |
| `Integer` | -2147483648 bis +2147483647 | 4 Byte |
| `Bigint` | -9223372036854775808 bis +9223372036854775807 | 8 Byte |

# … für Zähler

| Datentyp | Wertebereich | Speicherbedarf |
|---|---|---|
| `Smallserial` | 1 bis +32767 | 2 Byte |
| `Serial` | 1 bis +2147483647 | 4 Byte |
| `Bigserial` | 1 bis +9223372036854775807 | 8 Byte |

- Serielle Datentypen werden für Schlüsselwerte genutzt.
- Bei Neuanlage eines Datensatzes wird der Zähler automatisch inkrementiert.

# Nutzung von Zählern

```
CREATE TABLE Kontinent(
      id SERIAL PRIMARY KEY,
      kontientname VARCHAR(150)
);
```

- Zähler werden häufig für Schlüsselfelder genutzt.
- Bei der Neuanlage eines Datensatzes wird der aktuelle Zählerstand um eins erhöht. Der neue Zählerstand wird automatisch in das Datenfeld geschrieben.

# Fließkommazahlen / Gleitkommazahlen

- Zahlen mit Nachkommastellen.
- Als Dezimaltrennzeichen wird ein Punkt genutzt.
- Der Datentyp gibt die Genauigkeit an.
- Zahlen wie zum Beispiel 3.5, 0.345667, 2.0e+24 und so weiter sind Gleitkommazahlen.

# … die sich einem Wert nähren

| Datentyp | Beschreibung | Speicherbedarf |
|---|---|---|
| `Real` | 6 Dezimalstellen genau | 4 Bytes |
| `Double precision` | 15 Dezimalstellen genau | 8 Bytes |

```
CREATE TABLE dezimalzahl(
   feld01 REAL,
   feld02 DOUBLE PRECISION,
)
```

# Nährungswerte

```
CREATE TABLE dezimalzahl(
   feld01 REAL,
   feld02 DOUBLE PRECISION,
)
```

- Der Datentyp legt die Genauigkeit der Dezimalzahl fest.
- Keine exakte Berechnung möglich.

# … mit einer benutzerdefinierten Genauigkeit

```
feld NUMERIC(gesamt,nachkommastellen)
feld DECIMAL(gesamt,nachkommastellen)
```

- Den Feldtypen `NUMERIC` und `DECIMAL` wird in den runden Klammern eine Parameterliste übergeben. Die Parameter in der Liste werden durch Kommata getrennt.
- Der erste Parameter definiert die Gesamtstellen der Gleitkommazahl. Der zweite Parameter definiert die Anzahl der Nachkommastellen.
- Der Speicherbedarf der Zahl ist je nach Größe variabel.

# Beispiel

```
CREATE TABLE dezimalzahl(
   feld01 NUMERIC,
   feld02 DECIMAL(8,2),
   feld03 NUMERIC(3)
)
```

```
INSERT INTO dezimalzahl VALUES(
   1234567.1234,
   123456.12,
   123
)
```

# Erläuterung

```
CREATE TABLE dezimalzahl(
   feld01 NUMERIC,
   feld02 DECIMAL(8,2),
   feld03 NUMERIC(3)
)
```

- Das Datenfeld `feld01` hat beliebig viele Stellen vor und nach dem Dezimaltrennzeichen.
- Das Datenfeld `feld02` hat insgesamt acht Stellen und zwei Nachkommastellen. Vor dem Komma können sechs Stellen genutzt werden.
- Das Datenfeld `feld03` hat insgesamt 3 Stellen, aber keine Nachkommastellen.

# Währungswerte

```
CREATE TABLE produkt(
   feld01 MONEY,
   Feld02 NUMERIC(8,2)
)
```

- Der Datentyp `MONEY` ist kein Standard-Datentyp in SQL.
- Währungswerte werden häufig kaufmännisch auf 2 Nachkommastellen gerundet.

# Boolean

```
CREATE TABLE istLieferbar(
   feld01 BOOLEAN
   feld02 INTEGER
)
```

- Die Angabe `BOOLEAN` spiegelt die Antwort auf eine Ja- / Nein-Frage wieder.
- Der Wert „Ja“ kann zum Beispiel durch folgende Literale dargestellt werden: TRUE, 'true', '1'.
- Der Wert „Nein“ kann zum Beispiel durch folgende Literale dargestellt werden: FALSE, 'false', '0'.

# Datumsangaben

```
CREATE TABLE datumAngabe(
   aktuelldatum DATE DEFAULT now()
)
```

- Bei Nichteingabe eines Datums in das Feld wird das aktuelle Datum des Servers in das Feld eingetragen.

# … eingeben

```
INSERT INTO datumAngabe VALUES('2017-07-06');
INSERT INTO datumAngabe VALUES('06.07.2017');
INSERT INTO datumAngabe VALUES('07/06/2017');
```

- Datumsangaben werden immer als String eingegeben.
- In einer PostgreSQL-Datenbank wird ein Datum immer in der Form yyyy-mm-dd gespeichert.

# Formate

```
INSERT INTO datumAngabe VALUES('2017-07-06');
INSERT INTO datumAngabe VALUES('06.07.2017');
INSERT INTO datumAngabe VALUES('07/06/2017');
```

- Datumsangaben werden in dem Standard-Format yyyy-mm-dd angegeben.
- Datumsangaben können in der Form mm/dd/yyyy eingegeben werden.
- Alle anderen Formate werden entsprechend des Standardformats interpretiert. Die Angabe `'06.07.2017'` wird als `2017-06-07` gespeichert.

# Zeitangaben

```
CREATE TABLE zeitAngabe(
   aktuellzeit TIME DEFAULT CURRENT_TIME
)
```

- Bei Nichteingabe einer Zeit in das Feld wird die aktuelle Zeit des Servers in das Feld eingetragen.

# … eingeben

```
INSERT INTO zeitAngabe VALUES('05:15:06');
INSERT INTO zeitAngabe VALUES('05:15 AM');
INSERT INTO zeitAngabe VALUES('05:15 PM');
```

- Zeitangaben werden in der Form hh:mm:ss eingegeben.

# Zeitstempel

```
CREATE TABLE stempel(
   id SERIAL primary key,
   VerschifftAm TIMESTAMP DEFAULT NULL,
   AngekommenAm TIMESTAMP WITH TIME ZONE DEFAULT NULL
)
```

- Die Angaben werden immer als UTC (koordinierte Weltzeit) gespeichert.
- Die Zeitzone wird entsprechend der Servereinstellungen gewählt.
- Standardmäßig ist das Feld leer.

# Aktuelle Zeitzone

```
postgres =# SHOW TIMEZONE;
```

- Die aktuell eingestellte Zeitzone auf dem Server wird angezeigt.

# Anzeige aller möglichen Zeitzonen

```
postgres =# SELECT * FROM pg_timezone_names
postgres =# ORDER BY name limit 30;
```

- In diesem Beispiel werden die ersten 30 Einträge der Tabelle `pg_timezone_names` angezeigt.
- Das Feld `name` enthält die Bezeichnung einer Zeitzone.

# Setzen der Zeitzone

```
postgres =# SET TIME ZONE 'Asia/Bangkok';
```

- Die gewünschte Zeitzone wird mit Hilfe der Bezeichnung gesetzt.
- Die Einstellung gilt nur für die aktuelle Sitzung.

# Zeichenketten

```
CREATE TABLE zeichenkette(
   postleitzahl varchar(5),
   artikelnummer character(10),
   beschreibung text
)
```

- Alphanumerische und numerische Zeichen.
- Zeichenketten werden am Anfang und Ende durch ein Apostroph begrenzt.

# … mit fester Länge

```
CREATE TABLE zeichenkette(
   artikelnummer character(10)
)
```

- Die Länge der Zeichenkette wird dem Datentyp in runden Klammern übergeben.
- Das Datenfeld `artikelnummer` hat immer eine Länge von 10 Zeichen. Falls ein Text weniger Zeichen hat, wird der Text mit Leerzeichen aufgefüllt.
- Falls ein Text mehr Zeichen hat, wird eine Fehlermeldung ausgegeben.

# … mit einer maximalen Länge

```
CREATE TABLE zeichenkette(
   postleitzahl varchar(5)
)
```

- Die maximale Länge der Zeichenkette wird dem Datentyp in runden Klammern übergeben.
- Das Datenfeld `postleitzahl` kann maximal fünf Zeichen enthalten.
- Falls ein Text mehr Zeichen hat, wird eine Fehlermeldung ausgegeben.

# … mit einer beliebigen Länge

```
CREATE TABLE zeichenkette(
   beschreibung text
)
```

- Der Text kann beliebig lang sein.
- Dieser Datentyp ist nicht im Standard von SQL enthalten.

# Liste aller Tabellen in  einer Datenbank

- Mit Hilfe von `postgres =# \d` können alle Tabellen der Datenbank `postgres` aufgelistet werden.
- Mit Hilfe von `postgres =# \d+` können alle Tabellen und deren Speichergröße in der Datenbank `postgres` aufgelistet werden.
- Die Befehle `\d`, `\d+` beziehen sich immer auf die geöffnete Datenbank. Die geöffnete Datenbank wird im Prompt angezeigt.

# Anzeige der Struktur einer Tabelle

- Mit Hilfe von `\d land` werden die Datenfelder der Tabelle `land` angezeigt.
- Mit Hilfe von `\d+ [tabelle]` werden weitere Informationen angezeigt.