

Datenbanksysteme

SQL DDL

Jan Haase

2024

Abschnitt 9

Wiederholung: Kategorien von SQL-Befehlen

- **DQL (Data Query Language)**
Abfrage und Zusammenstellung von Daten

- SELECT

- **DML (Data Manipulation Language)**
Umgang mit Tabelleninhalten

- INSERT
- UPDATE
- DELETE

Externe
Sicht



- **DDL (Data Definition Language)**
Erstellen und Ändern von Datenbanken und Tabellen

- CREATE
- ALTER
- DROP

Konzeptionelle
Ebene




- **DAL (Data Administration Language)**
 - TCL (Transaction Control Language)
 - DCL (Data Control Language)

Interne Ebene

Themenübersicht

- Grundbegriffe und Datenbankentwurf
- Entity-Relationship-Modelle
- Relationales Datenbankmodell
- Normalisierung
- Arbeiten mit relationalen Datenbanken (SQL)
 - DQL
 - DML
 - ➔ **DDL (Data Definition Language)**
 - DAL

Themenübersicht

- Grundbegriffe und Datenbankentwurf
- Entity-Relationship-Modelle
- Relationales Datenbankmodell
- Normalisierung
- Arbeiten mit relationalen Datenbanken (SQL)
 - DQL
 - DML
 - **DDL (Data Definition Language)**
 -  CREATE TABLE
 - Datentypen in SQL
 - ALTER und DROP TABLE
 - CONSTRAINTS
 - SEQUENCES
 - VIEWS
 - DAL

Tabellendefinition in SQL

- Syntax zum **ERSTELLEN** einer Tabelle

```
CREATE TABLE <tabellenname> (  
    <attributsname> <datentyp>,  
    ...  
    <attributsname> <datentyp>,  
)
```

- Regeln für Tabellennamen
 - Max 30 Zeichen
 - Kann Buchstaben und Zahlen enthalten
 - Sollte mit Buchstaben beginnen

Tabellen anlegen – CREATE TABLE

```
CREATE TABLE Verkaeuer (
    Vnr INTEGER,
    Name VARCHAR(6) ,
    Status VARCHAR(7) ,
    Gehalt INTEGER,
    PRIMARY KEY (Vnr)
);
```

Verkaeuer

Vnr	Name	Status	Gehalt
1001	Udo	Junior	1500
1002	Ute	Senior	1900
1003	Uwe	Senior	2000

Themenübersicht

- Grundbegriffe und Datenbankentwurf
- Entity-Relationship-Modelle
- Relationales Datenbankmodell
- Normalisierung
- Arbeiten mit relationalen Datenbanken (SQL)
 - DQL
 - DML
 - **DDL (Data Definition Language)**
 - CREATE TABLE
 - ➔ **Datentypen in SQL**
 - ALTER und DROP TABLE
 - CONSTRAINTS
 - SEQUENCES
 - VIEWS
 - DAL

Basis-Datentypen in SQL: Zahlen

SQL ist eine streng typgebundene Sprache.

- **Ganze Zahlen:**

- SMALLINT
- INTEGER
- BIGINT

- **Festkommazahlen:**

- DECIMAL bzw. NUMERIC

Die Genauigkeit kann festgelegt werden:
DECIMAL (p, q) bedeutet
Genauigkeit p mit q Nachkommastellen.

- **Gleitkommazahlen:**

- FLOAT
- REAL
- DOUBLE PRECISION

Die Zahlenbereichsgrenzen werden durch das jeweilige DBMS festgelegt.

- Dabei gilt:

- $\text{SMALLINT} \subseteq \text{INTEGER} \subseteq \text{BIGINT}$
- $\text{FLOAT} \subseteq \text{REAL} \subseteq \text{DOUBLE PRECISION}$

Basis-Datentypen in SQL: Texte, Zeiten u. a.

● Texte

- CHAR (q)
- VARCHAR (q)
q gibt bei CHAR die genaue, bei VARCHAR die maximale Anzahl der gespeicherten Zeichen an.
CHAR ist daher in der Verarbeitung etwas schneller, benötigt jedoch deutlich mehr Speicherplatz.

● Zeiten

- DATE
- TIME
Genaue Repräsentation hängt vom DBMS und der jeweiligen Konfiguration ab.

● Große Datenmengen

- CLOB (Character Large Object)
- BLOB (Binary Large Object)
CLOB darf im Gegensatz zu BLOB nur druckbare Zeichen enthalten.

Nutzerdefinierte Datentypen in SQL

- Zusätzlich eigene Datentypen definierbar.
- Vorsicht bei zusammengesetzten Datentypen
→ Verstöße gegen die 1NF vermeiden!
- Alle möglichen Datentypen können mit dem Typ VARCHAR als Text simuliert werden.

Unterschiede bei ORACLE-Datentypen

- Für die Attribute (=Spalten der Tabelle) zulässige Datentypen:

Zulässige Datentypen

Standard SQL	Oracle	Bedeutung
NUMERIC	NUMBER	bel. Zahl
NUMERIC (x) INTEGER	NUMBER (x)	x-stellige Ganzzahl
NUMERIC (x,p)	NUMBER (x,p)	x-stellige Zahl, davon p als Nachkommastellen
VARCHAR (x)	VARCHAR (x) VARCHAR2 (x)	Text mit maximal x Zeichen
DATE	DATE	Speichern eines Datums (bei Oracle inkl. Zeit!)
...		

- Hinweise:
 - Viele Oracle-Datentypen unterscheiden sich im Namen von "Standard"-SQL-Datentypen
 - Datentyp DATE haben fast alle DB, aber meist unterschiedlich implementiert
 - VARCHAR2 kann bis zu 4000 Zeichen aufnehmen

Besonderheiten Datentyp DATE (1/2)

- Beispiele:

```
-- Tabelle test anlegen
```

```
CREATE TABLE test (Datum date );
```

- Unter Oracle speichert der Datentyp date automatisch einen time stamp, also Datum und Uhrzeit.

```
-- aktuelles Datum und Zeit einfügen
```

```
INSERT INTO test VALUES (sysdate);
```

- Mit SYSDATE bekommen wir das aktuelle Systemdatum und –Uhrzeit.

```
-- nur Datum auslesen
```

```
SELECT to_Char(Datum, 'dd.mm.yyyy') FROM test;
```

```
-- nur Zeit auslesen
```

```
SELECT to_char(Datum, 'hh:mi:ss') FROM test;
```

```
SELECT to_char(Datum, 'hh24:mi:ss') FROM test;
```

Besonderheiten Datentyp DATE (2/2)

-- spezifisches Datum einfügen

INSERT INTO test

VALUES (to_date('03.05.2014','dd.mm.yyyy')) ;

- Wird nur das Datum angegeben, wird die Uhrzeit auf 00:00:00 Uhr gesetzt.

-- spezifische Uhrzeit einfügen

INSERT INTO test

VALUES (to_date('08:53:17','hh24:mi:ss')) ;

- hh24 bedeutet, dass das 24er-Format für Stunden verwendet wird.
- Wird nur die Uhrzeit angegeben, wird automatisch der 1. des aktuellen Jahres als Datum herangezogen.

Themenübersicht

- Grundbegriffe und Datenbankentwurf
- Entity-Relationship-Modelle
- Relationales Datenbankmodell
- Normalisierung
- Arbeiten mit relationalen Datenbanken (SQL)
 - DQL
 - DML
 - **DDL (Data Definition Language)**
 - CREATE TABLE
 - Datentypen in SQL
 - ➔ **ALTER und DROP TABLE**
 - CONSTRAINTS
 - SEQUENCES
 - VIEWS
 - DAL

Tabellen Ändern – ALTER TABLE

- Zum Ändern von Tabellen-Eigenschaften **ALTER TABLE**

```
ALTER TABLE <tabellenname>
{ ADD <spaltenname > <datentyp>
| RENAME COLUMN <spaltenname> TO <neuerSpaltenname>
| DROP      PRIMARY KEY
      | UNIQUE (<spaltenliste>)
      | CONSTRAINT <constraintname>
      | COLUMN <spaltenname>
| DISABLE PRIMARY KEY
      | UNIQUE (<spaltenliste>)
      | CONSTRAINT <constraintname>
      | ALL TRIGGERS
| ENABLE  PRIMARY KEY
      | UNIQUE (<spaltenliste>)
      | CONSTRAINT <constraintname>
      | ALL TRIGGERS
| MODIFY <spaltenname> <datentyp> }
```

Tabelle löschen – DROP TABLE

- Zum Löschen der gesamten Tabelle: **DROP TABLE**

DROP TABLE *tabelle*

- Tabellen müssen nicht leer sein, um gelöscht zu werden

- Beispiele:

DROP TABLE *City*;

Löscht die gesamte Tabelle
(nicht nur die Daten, sondern auch die Struktur!)

Themenübersicht

- Grundbegriffe und Datenbankentwurf
- Entity-Relationship-Modelle
- Relationales Datenbankmodell
- Normalisierung
- Arbeiten mit relationalen Datenbanken (SQL)
 - DQL
 - DML
 - **DDL (Data Definition Language)**
 - CREATE TABLE
 - Datentypen in SQL
 - ALTER und DROP TABLE
 - ➡ **CONSTRAINTS**
 - SEQUENCES
 - VIEWS

CONSTRAINTS

- Mit der Tabellendefinition können **Bedingungen für konkrete Attributwerte** formuliert werden, die bei jeder Manipulation überprüft werden.
 - Wert muss immer angegeben werden (NULL nicht zulässig)
 - Wertebereichseinschränkung (z.B. nur Kunden aus Elmshorn)
 - Angabe von Default-Werten
 - Festlegung von Schlüsseln und Fremdschlüsseln
 - Forderung an einzelne Einträge (Datensätze) in Form von Prädikaten (z.B. keine Kunden aus Pinneberg mit dem Namen 'Müller'.)
- Die Bedingungen (sog. CONSTRAINTS) können als
 - **Spalten-Constraint** - bezieht sich auf den Wert in einer Spalte
 - **Tabellen-Constraint** - bezieht sich auf den gesamten Datensatzdefiniert werden

CONSTRAINTS – SYNTAX

- Bedingungen mit CONSTRAINTS festlegen

[CONSTRAINT <name>] <bedingung>

<bedingung> := CHECK (<boolsche_bedingung>)

- Besondere <bedingung> mit anderen Formen:
 - Primärschlüssel, Fremdschlüssel, eindeutige Attributwerte
 - Spalten-Constraints zur Angabe ob NULL-Werte erlaubt sind
Name VARCHAR(10) NOT NULL
 - Äquivalent als Tabellen-Constraint
CHECK (Name IS NOT NULL)

CONSTRAINTS – Beispiel

- Beispiele

```
CREATE TABLE Kunde (  
    KundenNr NUMBER(6) NOT NULL PRIMARY KEY,  
    Name VARCHAR(50) NOT NULL,  
    Ort VARCHAR(50) NOT NULL,  
    CONSTRAINT nurElmshorn CHECK (Ort='Elmshorn')  
);  
  
CREATE TABLE Verkaeufer (  
    VNr NUMBER CHECK(VNr >= 1000),  
    Vname VARCHAR(12) NOT NULL,  
    Status VARCHAR(10) NOT NULL,  
    Gehalt NUMBER,  
    CONSTRAINT MaxJunior CHECK  
        (Not(Status = 'Junior') OR Gehalt <= 2500)  
);
```

Beispiel

- Es soll sichergestellt sein, dass ein Mitarbeiter mit dem Status Junior maximal 2000 € verdienen kann.
- Wie sollte eine entsprechende Bedingung aussehen?
 - Erster Ansatz:

```
CONSTRAINT ersterVersuch
```

```
CHECK (Status = 'Junior' AND Gehalt <= 2000)
```

- Ist dies eine gute Lösung? Was würde passieren, wenn folgender Datensatz eingefügt werden soll?

```
INSERT INTO Mitarbeiter
```

```
VALUES ('Susanne', 'Fischer', 'Senior', 2500);
```

Mitarbeiter

Vorname	Nachname	Status	Gehalt
Stefan	Müller	Junior	1900
Herbert	Schmidt	Senior	2500
...			

Beispiel

- Es soll sichergestellt sein, dass ein Mitarbeiter mit dem Status Junior maximal 2000 € verdienen kann.
- Wie sollte eine entsprechende Bedingung aussehen?
 - Zweiter Ansatz:

```
CONSTRAINT zweiterVersuch
    CHECK (NOT (Status = 'Junior')
           OR Gehalt <= 2000)
```

Mitarbeiter

Vorname	Nachname	Status	Gehalt
Stefan	Müller	Junior	1900
Herbert	Schmidt	Senior	2500
...			

**Eine Bedingung der Form
„Wenn A gilt, dann soll auch B gelten“ (Implikation)
kann immer übersetzt werden als NOT (A) OR B.**

Fremdschlüssel

- Fremdschlüssel-Beziehungen werden beim CREATE TABLE mit Hilfe von CONSTRAINTS festgelegt:

```
FOREIGN KEY (<attributsname> [,...])  
    REFERENCES <tabellennamen> (<attributsname>[,...])  
    [ON DELETE CASCADE]
```

- In den Attributs-Listen steht, wie die Attribute in der zu erstellenden und in der referenzierenden Tabelle heißen. Die Attribute müssen in der referenzierten Tabelle der PRIMARY KEY sein und die Tabelle muss vorher bereits definiert sein.

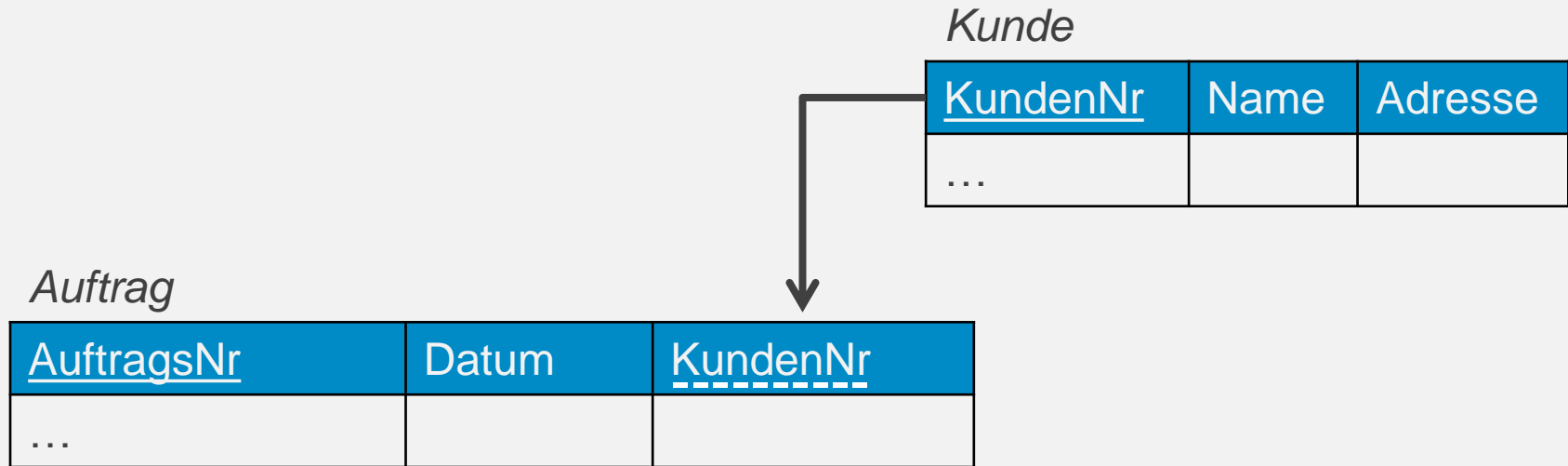
Fremdschlüssel Beispiel

- Zu einer Tabelle "Kunde" soll eine zugehörige Tabelle "Auftrag" erstellt werden. Es gilt eine Fremdschlüssel-Beziehung zwischen Auftrag und Kunde. Ein Auftrag darf nur eingetragen werden, wenn die entsprechende KundenNr in "Kunde" existiert (nicht andersherum!)
Die FOREIGN-KEY Beziehung ist daher in "Auftrag" und nicht in "Kunde" einzutragen)

```
CREATE TABLE Auftrag (  
    AuftragsNr NUMBER (6) NOT NULL,  
    Datum DATE NOT NULL,  
    KundenNr NUMBER (6) NOT NULL,  
    PRIMARY KEY (AuftragsNr),  
    FOREIGN KEY (KundenNr) REFERENCES Kunde (KundenNr)  
);
```


Fremdschlüssel Beispiel

- Was passiert nun, wenn ein Kunde gelöscht wird?



```
CREATE TABLE Auftrag (
    AuftragsNr NUMBER (6) NOT NULL,
    Datum DATE NOT NULL,
    KundenNr NUMBER (6) NOT NULL,
    PRIMARY KEY (AuftragsNr),
    FOREIGN KEY (KundenNr) REFERENCES Kunde (KundenNr)
);
```

Fremdschlüssel - CONSTRAINTS

- Es kann sinnvoll sein, eine Löschweitergabe einzurichten. Löschweitergabe bedeutet beispielsweise, dass ein Auftrag automatisch gelöscht wird, wenn der zugehörige Kunde gelöscht wird. Dies geschieht mit ON DELETE CASCADE.

```
CREATE TABLE Auftrag (  
    AuftragsNr NUMBER (6) NOT NULL,  
    Datum DATE NOT NULL,  
    KundenNr NUMBER (6) NOT NULL,  
    PRIMARY KEY (AuftragsNr),  
    FOREIGN KEY (KundenNr) REFERENCES Kunde (KundenNr)  
        ON DELETE CASCADE  
);
```

Fremdschlüssel - CONSTRAINTS

- Mit Fremdschlüssel-CONSTRAINTS lassen sich unterschiedliche Reaktionen auf das Löschen bzw. Ändern von Datensätzen definieren.
- Folgende Angaben sind möglich:
 - **ON DELETE CASCADE**
Ein DELETE in der Primärtabelle führt zum Löschen der Datensätze in der referenzierenden Fremdschlüsseltabelle.
 - **ON DELETE RESTRICT**
Ein DELETE in der Primärtabelle kann nur ausgeführt werden, wenn in keiner Fremdschlüsseltabelle eine Referenz enthalten ist. Dies ist das Default-Verhalten, wenn kein CONSTRAINT angegeben wird.
Dies ist der Standard bei Oracle und kann nicht explizit angegeben werden.
 - **ON DELETE SET NULL**
Ein DELETE in der Primärtabelle setzt die Spalten des Fremdschlüssels in der Fremdschlüsseltabelle auf NULL.
 - **ON DELETE SET DEFAULT**
Ein DELETE in der Primärtabelle setzt die Spalten des Fremdschlüssels in der Fremdschlüsseltabelle auf den Default-Wert. Ist in der Tabellendefinition kein Default-Wert angegeben, wird die Spalte auf NULL gesetzt.

Fremdschlüssel - CONSTRAINTS

- **ON UPDATE CASCADE**

Ein UPDATE in der Primärtabelle führt auch zu einer Änderung der Fremdschlüsselwerte in der Fremdschlüsseltabelle.

- **ON UPDATE RESTRICT**

Ein UPDATE in der Primärtabelle kann nur ausgeführt werden, wenn in keiner Fremdschlüsseltabelle eine Referenz enthalten ist. Dies ist das Default-Verhalten, wenn kein CONSTRAINT angegeben wird.

Dies ist der Standard bei Oracle und kann nicht explizit angegeben werden.

- **ON UPDATE SET NULL**

Ein UPDATE in der Primärtabelle setzt die Spalten des Fremdschlüssels in der Fremdschlüsseltabelle auf NULL.

- **ON UPDATE SET DEFAULT**

Ein UPDATE in der Primärtabelle setzt die Spalten des Fremdschlüssels in der Fremdschlüsseltabelle auf den Default-Wert. Ist in der Tabellendefinition kein Default-Wert angegeben, wird die Spalte auf NULL gesetzt.

Übungsaufgabe Constraints 1 (Kunde und Auftrag)

- Legen Sie die Tabelle Kunde an.

```
CREATE TABLE Kunde (
    KundenNr NUMBER(6) NOT NULL PRIMARY KEY,
    Name VARCHAR(50) NOT NULL,
    Ort VARCHAR(50) NOT NULL,
    CONSTRAINT nurElmshorn CHECK (Ort='Elmshorn')
);
```

- Versuchen Sie, einen Ort ungleich „Elmshorn“ einzufügen.
- Fügen Sie dann zwei beliebige gültige Datensätze ein.

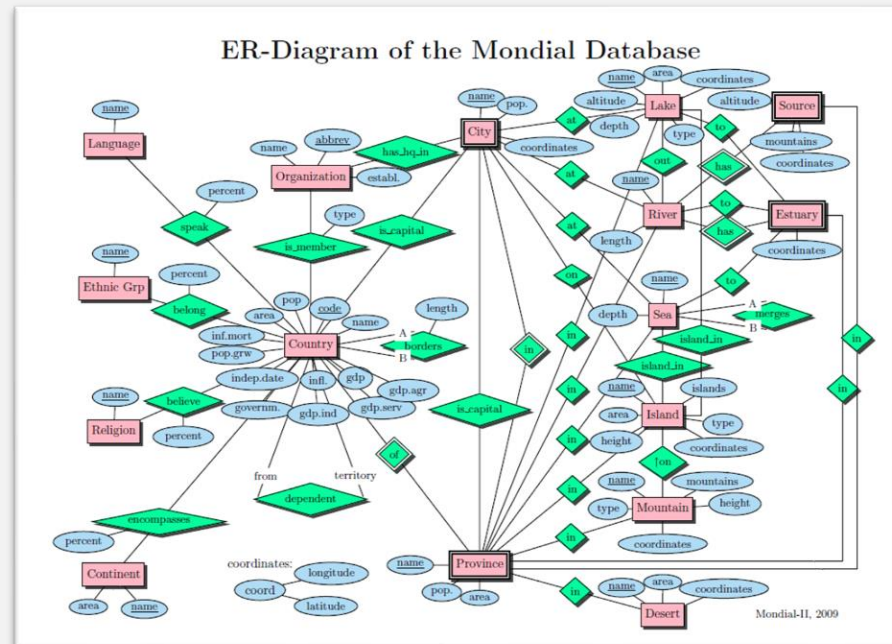
- Legen Sie die Tabelle Auftrag an.

```
CREATE TABLE Auftrag (
    AuftragsNr NUMBER (6) NOT NULL,
    Datum DATE NOT NULL,
    KundenNr NUMBER (6) NOT NULL,
    PRIMARY KEY (AuftragsNr),
    FOREIGN KEY (KundenNr) REFERENCES
        Kunde(KundenNr) ON DELETE CASCADE
);
```

- Fügen Sie drei Aufträge ein.
- Löschen Sie den dazugehörigen Kunden. Was passiert?
- Ändern Sie die DELETE-Option (RESTRICT oder SET NULL), legen Sie Kunden und Aufträge erneut an, löschen Sie den Kunden und beobachten Sie die Abweichungen.

Übungsaufgabe Constraints 2 (Mondial-DB)

- Untersuchen Sie die Constraints aller Tabellen der Mondial-Beispieldatenbank.
- Versuchen Sie, Zeilen hinzuzufügen, die gegen Constraints verstoßen, um damit Fehlermeldungen auszulösen.
- Analysieren Sie das Script mondial-schema.sql, mit dem die Tabellen angelegt wurden. Können Sie jetzt alle Anweisungen nachvollziehen?




Übungsaufgabe Constraints 3 (Projektverwaltung) – optional

- Legen Sie die normalisierten Tabellen zur Projektverwaltung mit SQL-Anweisungen an (vgl. Normalisierungsaufgabe 3):
 - Mitarbeiter: PNr | TelNr
 - Projekte: Projekt | PBudget
 - Projektrolle: PNr | Projekt | Rolle (PNr und Projekt ist auch Fremdschlüssel)
 - Abteilungszugehörigkeit: Projekt | VAbt (Projekt ist auch Fremdschlüssel)
 - Wählen Sie geeignete Datentypen und vergessen Sie Primär- und Fremdschlüssel nicht!
- Fügen Sie folgende Daten mit SQL-Anweisungen in Ihre Tabellen ein:

PNr	TelNr	Projekt	PBudget	Rolle	VAbt
1	13	EAI	42	Entwickler	EBUIS
1	13	EAI	42	Entwickler	INFRA
1	13	ODB	15	Spezifizierer	INFRA
1	13	ODB	15	Spezifizierer	SERVICE
2	14	EAI	42	Spezifizierer	EBUIS
2	14	EAI	42	Spezifizierer	INFRA
2	14	WEB	15	Designer	INFRA

- Machen Sie sich damit vertraut, wie man im Oracle Developer interaktiv Tabellen anlegen, mit Werten füllen und als SQL-Anweisungen exportieren kann.

Themenübersicht

- Grundbegriffe und Datenbankentwurf
- Entity-Relationship-Modelle
- Relationales Datenbankmodell
- Normalisierung
- Arbeiten mit relationalen Datenbanken (SQL)
 - DQL
 - DML
 - **DDL (Data Definition Language)**
 - CREATE TABLE
 - Datentypen in SQL
 - ALTER und DROP TABLE
 - CONSTRAINTS
 -  **SEQUENCES**
 - VIEWS

SEQUENCES – Definition

- Aufsteigende Nummerierung von Datensätzen oft in DBS erforderlich.
- Beispiel: Automatische Erzeugung von generischen Primärschlüsseln beim Einfügen neuer Elemente.
- Sequences sind automatisch generierte Abfolgen von Zahlen in ORACLE.
- Andere DBMS kennen stattdessen einen Datentyp „Autowert“ oder Auto-ID.

ID	Vorname	Nachname	Eintrittsdatum
1	Hannes	Schultz	1.7.2012
2	Olga	Krüger	1.9.2013
...

SEQUENCES – Syntax und Beispiel

- Syntax:

```
CREATE SEQUENCE <seq_name>
    [INCREMENT BY <integer>]
    [START WITH < integer >]
    [MAXVALUE < integer > | NOMAXVALUE]
    [MINVALUE < integer > | NOMINVALUE]
    [CYCLE | NOCYCLE]
;
```

- Beispiel:

```
CREATE SEQUENCE seq_KundenNr
INCREMENT BY 1
START WITH 1000
NOCYCLE;
```

SEQUENCES – Semantik

- Folgende Angaben sind möglich:
 - **INCREMENT BY int**
Definiert die ganze Zahl, um die die Sequenz bei jedem Durchlauf erhöht wird. Ein negativer Wert erzeugt eine absteigende Sequenz.
 - **START WITH**
Definiert den ersten Sequenz-Wert.
 - **MAXVALUE int | NOMAXVALUE**
Definiert den größten Wert bzw. legt fest, dass es keine Beschränkung gibt.
 - **MINVALUE int | NOMINVALUE**
Definiert den kleinsten Wert bzw. legt fest, dass es keine Beschränkung gibt.
 - **CYCLE | NONCYCLE**
Definiert, ob nach Erreichen der oberen/unteren Grenze zyklisch neu begonnen wird. Dies kann zu doppelten Einträgen führen..

SEQUENCES – Aufruf der Sequenz

- Definition der Sequenz:

```
CREATE SEQUENCE seq_KundenNr
INCREMENT BY 1
START WITH 1000
NOCYCLE;
```

- `<seq_name>.nextval` liefert den nächsten Wert.
- `<seq_name>.currval` liefert den aktuellen Wert erneut.

- Beispiel:

```
INSERT INTO Kunde
VALUES (seq_KundenNr.nextval, 'Till',
        'Molitor', '15.9.2013');
```

- Infos über alle Sequenzen:

```
SELECT * FROM user_sequences;
```

Übungsaufgabe Sequenzen

- Legen Sie eine Tabelle von Studierenden an mit
 - Matrikelnummer,
 - Name,
 - Wohnort.
- Legen Sie eine Sequenz an, die vierstellige Matrikelnummern ab 6000 erzeugt. Nach 9999 soll die Nummer 0001 kommen.
- Tragen Sie in die Tabelle einige Einträge ein und verwenden Sie für die Matrikelnummer Werte aus der zuvor erstellten Sequenz.

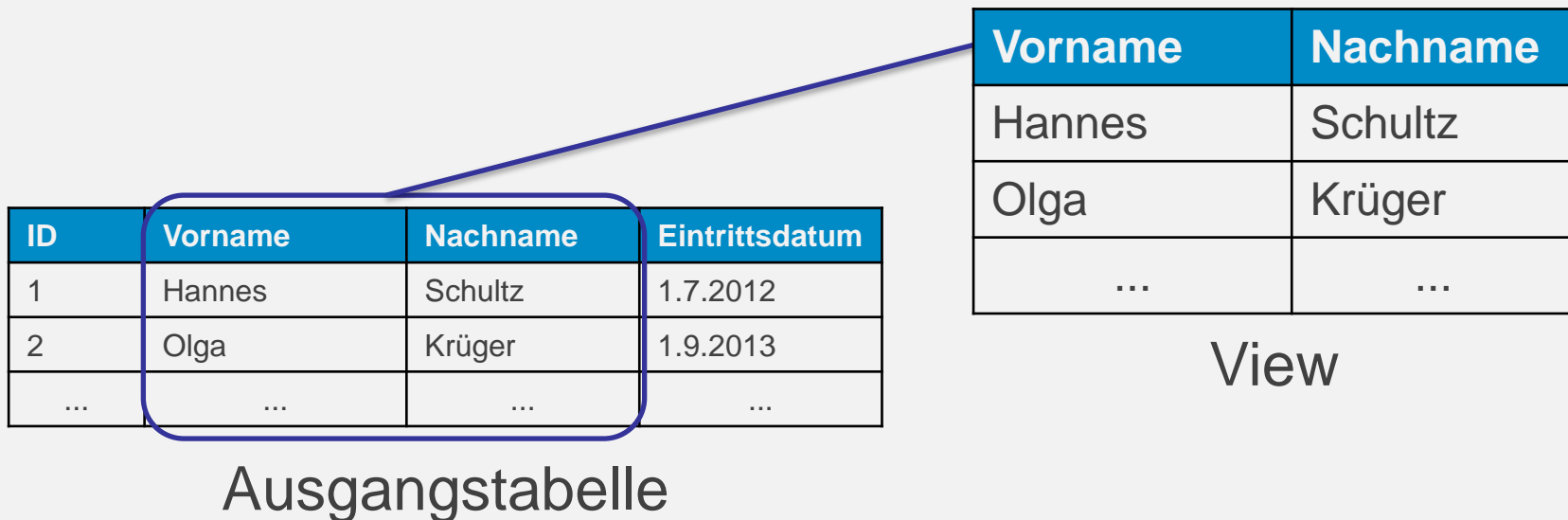
Themenübersicht

- Grundbegriffe und Datenbankentwurf
- Entity-Relationship-Modelle
- Relationales Datenbankmodell
- Normalisierung
- Arbeiten mit relationalen Datenbanken (SQL)
 - DQL
 - DML
 - **DDL (Data Definition Language)**
 - CREATE TABLE
 - Datentypen in SQL
 - ALTER und DROP TABLE
 - CONSTRAINTS
 - SEQUENCES

 **VIEWS**

Views (Sichten) – Definition

- Ein View ist eine Art „virtuelle Tabelle“.
- Eine View enthält immer aktuelle Informationen aus einer oder mehrerer Tabellen.
- Ein View liegt dabei nicht als physikalische Tabelle vor, sondern existiert nur als Sicht für den Benutzer.
- Mit Views lässt sich wie mit normalen Tabellen arbeiten.



Views – Vorteile

- Kompakte Darstellung und Vermeidung komplexer Abfragen bei hochnormalisierten Relationenmodellen. (Eine View statt mehrerer verknüpfter Tabellen)
- Im DBMS vordefiniert und optimiert: beschleunigt Abfragen.
- Stabilere Schnittstelle zu Anwendungsprogrammen, Änderungen am Modell bleiben verborgen (Geheimnisprinzip)
- Ermöglicht dedizierte Berechtigungen

Aber:

- Ein falscher Einsatz von Views kann zu **Performanzproblemen** führen.

Views – Umgang mit Sichten in SQL

- **Erzeugen** einer VIEW:

```
CREATE VIEW <viewname> [(spaltennamen)] AS
    <abfrage>
```

- Dabei ist **abfrage** ein beliebiges Select-Statement.
- Eine **ORDER BY**-Klausel ist darin jedoch nicht zulässig.
- Wenn die Angabe von Spaltennamen fehlt, werden sie aus der zugrunde liegenden Abfrage übernommen.

- **Löschen** einer VIEW:

```
DROP VIEW <viewname>
```

- Views werden vom DBMS persistent gehalten und müssen bei Bedarf explizit gelöscht werden.

Views – Beispiel: Definition

- View Leichtteile soll Teilenummer, Bezeichnung, Material und Gewicht aller Teile mit einem Gewicht kleiner als 10 enthalten.
- Definition:

```
CREATE VIEW Leichtteile AS
    SELECT TeilNr, Bezeichnung, Material, Gewicht
    FROM Teile
    WHERE Gewicht < 10
;
```

TeilNr	Bezeichnung	Material	Gewicht	Preis
23	Schraube	Niro	3	5
56	Mutter	Niro	1	5
17	Bolzen	Verzinkt	17	15
88	Achse	Alu	5	20
12	Lager	Alu	8	50

Ausgangstabelle
Teile

Views – Beispiel: Einfügen von Daten in die View

- Wichtig:
 - Leichtteile ist nur eine Sicht. **Änderungen** an Leichtteile betreffen immer die Ausgangstabelle Teil.
- Beispiel: Einfügen eines Teils:
 - Der Befehl

```
INSERT INTO Leichtteile
VALUES (71, 'Gehäuse', 'Alu', 9);
```

fügt das Gehäuse in die Tabelle Teil ein. Der Preis wird null.

TeilNr	Bezeichnung	Material	Gewicht
23	Schraube	Niro	3
56	Mutter	Niro	1
88	Achse	Alu	5
12	Lager	Alu	8

Tabelle
Leichtteile

Views – Mögliche Probleme bei Änderungen

- **INSERT, UPDATE und DELETE** funktionieren nur bei Views mit einer Basistabelle zuverlässig, sonst nur in Ausnahmefällen.
- Wenn Daten über Views in die Ausgangstabelle eingefügt werden, können Probleme auftauchen, falls:
 - Primärschlüssel nicht richtig gesetzt werden, z. B. weil nicht in der View enthalten.
 - einzutragende Daten nicht alle Spalten füllen und NULL-Werte nicht zugelassen sind.
 - beim Erzeugen der View Werte berechnet wurden (z. B. Summe aus zwei einzelnen Werten).
 - Verstöße gegen weitere Constraints der Ausgangstabelle auftreten.

Views – Einfügen verhindern mit Check Option

- In die View lassen sich prinzipiell auch Datensätze einfügen, die der Sichtendefinition nicht genügen.
- Beispiel:

```
INSERT INTO Leichtteile
VALUES (72, 'Gehäuse', 'Stahl', 11);
```

führt zum Einfügen des Gehäuses in die Ausgangstabelle. Es wird allerdings in Leichtteile aufgrund seines zu großen Gewichts nicht angezeigt.

- Dieses Verhalten lässt sich mit **WITH CHECK OPTION** vermeiden.
- Beispiel:

```
CREATE VIEW Leichtteile AS
SELECT TeilNr, Bezeichnung, Material, Gewicht
FROM Teile WHERE Gewicht < 10
WITH CHECK OPTION
```

führt beim Einfügen des Gehäuses zu einem Fehler.

View – Erweitertes Beispiel

- Das folgende Beispiel zeigt einen View über mehrere Tabellen und die Umbenennung von Spalten.

Material	Lieferant
Niro	Aro KG
Verzinkt	Aro KG
Alu	Aluwerke Schultz AG

Ausgangstabelle
Lieferanten

- View für die Beschaffung aller Teile:

```
CREATE VIEW Beschaffung (Nr, Materialart, Lieferant) AS
SELECT Teilnr, Teile.Material, Lieferant
FROM Teile, Lieferanten
WHERE Teile.Material = Lieferanten.Material;
```

Nr	Materialart	Lieferant
23	Niro	Aro KG
...

View
Beschaffung

Views – Übungsaufgabe 09.1

- Der aktuelle Datenbank-Zustand sei:

TeilNr	Bezeichnung	Material	Gewicht	Preis
23	Schraube	Niro	3	5
56	Mutter	Niro	1	5
17	Bolzen	Verzinkt	17	15
88	Achse	Alu	5	20
12	Lager	Alu	8	50

- Dazu gehört die folgende Sicht Leichtteile:

```
CREATE VIEW Leichtteile AS
```

```
    SELECT TeilNr, Bezeichnung, Material, Gewicht
```

```
    FROM Teil WHERE Gewicht < 10
```

```
WITH CHECK OPTION;
```

- Was ist das Ergebnis folgender Operationen? Machen Sie sich erst Gedanken über die Ergebnisse, bevor Sie die Abfragen praktisch ausprobieren!
 - DELETE FROM Leichtteile WHERE Gewicht > 5;
 - UPDATE Leichtteile SET Bezeichnung = ' ' WHERE TeilNr < 25;
 - UPDATE Leichtteile SET Gewicht = 21 WHERE TeilNr < 25;
 - UPDATE Leichtteile SET Bezeichnung = ' ' WHERE TeilNr = 17;

Views – Übungsaufgabe 09.2

- Legen Sie folgende Tabellen an:
 - Abteilung (Abteilung_ID, Ab_Name)
 - Mitarbeiter (Mitarbeiter_ID, Name, Vorname, Abteilung_ID)
 - Dienstwagen (CarID, Kennzeichen, Fahrzeugtyp, Mitarbeiter_ID)
- Füllen Sie die Tabellen mit beliebigen Datensätzen.
- Bei der Suche nach Dienstwagen sollen mit der View Dienstwagen_Anzeige immer auch angezeigt werden:
 - Name und Vorname des Mitarbeiters,
 - ID und Bezeichnung seiner Abteilung und
 - der Fahrzeugtyp und Kennzeichen.
- Fügen Sie dann auch eine Zeile ein (in die VIEW)!

Zusammenfassung

- DDL (Data Definition Language)
 - CREATE TABLE
 - Datentypen in SQL (als Einschub)
 - Ganze Zahlen: SMALLINT, INTEGER, BIGINT
 - Festkommazahlen: DECIMAL bzw. NUMERIC
 - Gleitkommazahlen: FLOAT, REAL, DOUBLE PRECISION
 - Texte: CHAR (q), VARCHAR (q)
 - Zeiten: DATE, TIME
 - Große Datenmengen: CLOB, BLOB
 - Benutzerdefinierte Datentypen
 - ALTER und DROP TABLE
 - CONSTRAINTS
 - SEQUENCES
 - VIEWS