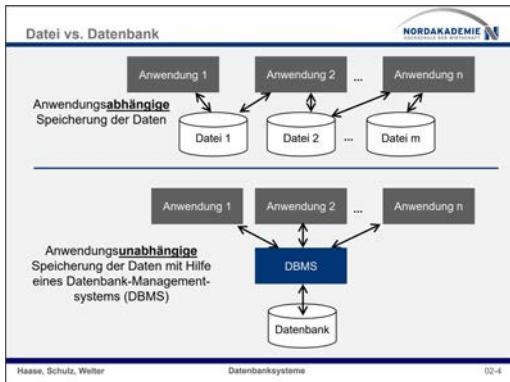


Anforderungen einer DB:

Persistenz	Dauerhafte Speicherung + Wiederverwendbarkeit von Daten
Datenschema	Strukturierung von Daten + Zusammenhänge

**Datenbank-Managementsystem:**

- Umfasst alle Programme, die zum Aufbau, der Nutzung und Verwaltung einer Datenbank benötigten werden
- Ermöglicht verschiedenen Benutzern einen Zugriff auf die Daten

Datenbank:

- Eine Sammlung von strukturierten, inhaltlich zusammengehörigen Daten

Datenbanksystem:

- Besteht aus einer oder mehreren Datenbanken und einem Datenbankmanagementsystem

Datenelement:

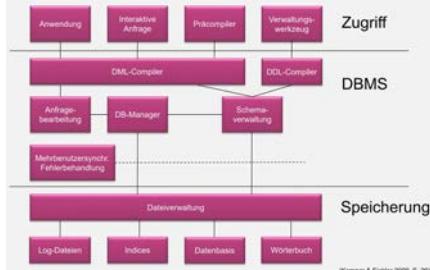
- Gruppierung von mehreren Zeichen, z.B. ein Wort

Datensatz:

- Gruppe von inhaltlich zusammenhängenden Datenelementen (Zeile einer Tabelle)

Datei:

- Zusammenfassung logisch zusammenhängender, gleichartiger Datensätze (Tabelle)

**Informationsbedarf:**

Art, Menge und Qualität der Informationen, die eine Person zur Erfüllung ihrer Aufgaben in einer bestimmten Zeit benötigt

Objektiv:

- Infos, die zur Aufgabenerfüllung verwendet werden
- sachlich ableitbar aus der zu erfüllenden Aufgabe

Subjektiv:

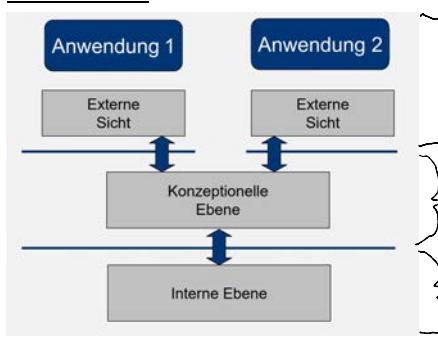
- Infos, relevant für Entscheidungsträger
- entspricht persönlicher Sichtweise des Verwenders (der Infos)

Informations ...

- ... nachfrage:
- subjektiver Infobedarf wird artikuliert
- Nachfrage kann Informationsbedarf übersteigen

... angebot:

- Gesamtheit der Infos, die einem Nachfrager zu einem bestimmten Zeitpunkt + Ort zur Verfügung stehen

3-Ebenen-Modell:

Externe Ebene:
- Benutzersicht

Konzeptionelle Ebene:
- Welche Daten

Interne Ebene:
- Wie und wo

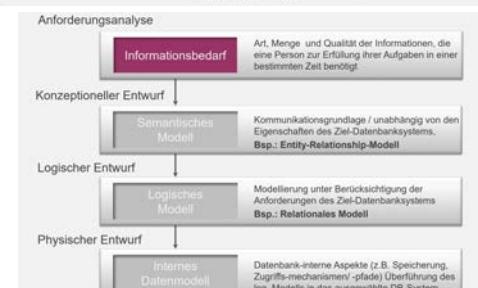
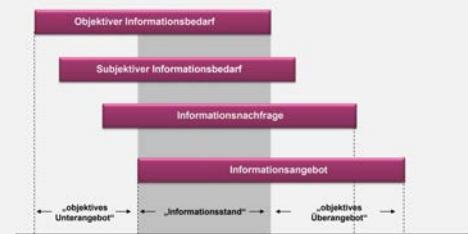
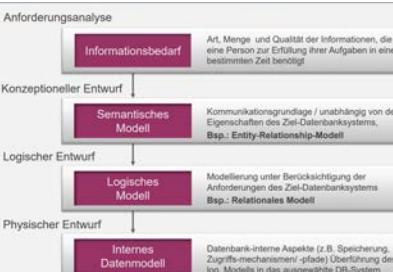
Wozu?:

- Physische Datenunabhängigkeit
 - o Änderungen der internen Darstellung sind unabhängig der anderen Ebenen
- Logische Datenunabhängigkeit
 - o Teile der Datenbankstruktur ändern, ohne Benutzersichten zu ändern

Aufgaben DBMS (Datenbank-Managementsystem):

- Operationen
- Datenunabhängigkeit
 - o zw. Anwendungsprogramme und Datenorga
- Redundanzfreiheit
 - o gleiche Daten nicht mehrfach gespeichert
- Strukturierung
 - o Suche, Sortierung
- Integrität / Konsistenzüberwachung
 - o keine logischen Widersprüche der Daten
- Koordination / Sync. paralleler Zugriffe
- Benutzersichten
 - o Spezielle Sicht (Ausschnitt aus dem Datenbestand) für Benutzer
- Rechteverwaltung + Zugriffskontrolle
- Datensicherung
- Meta-Informationen

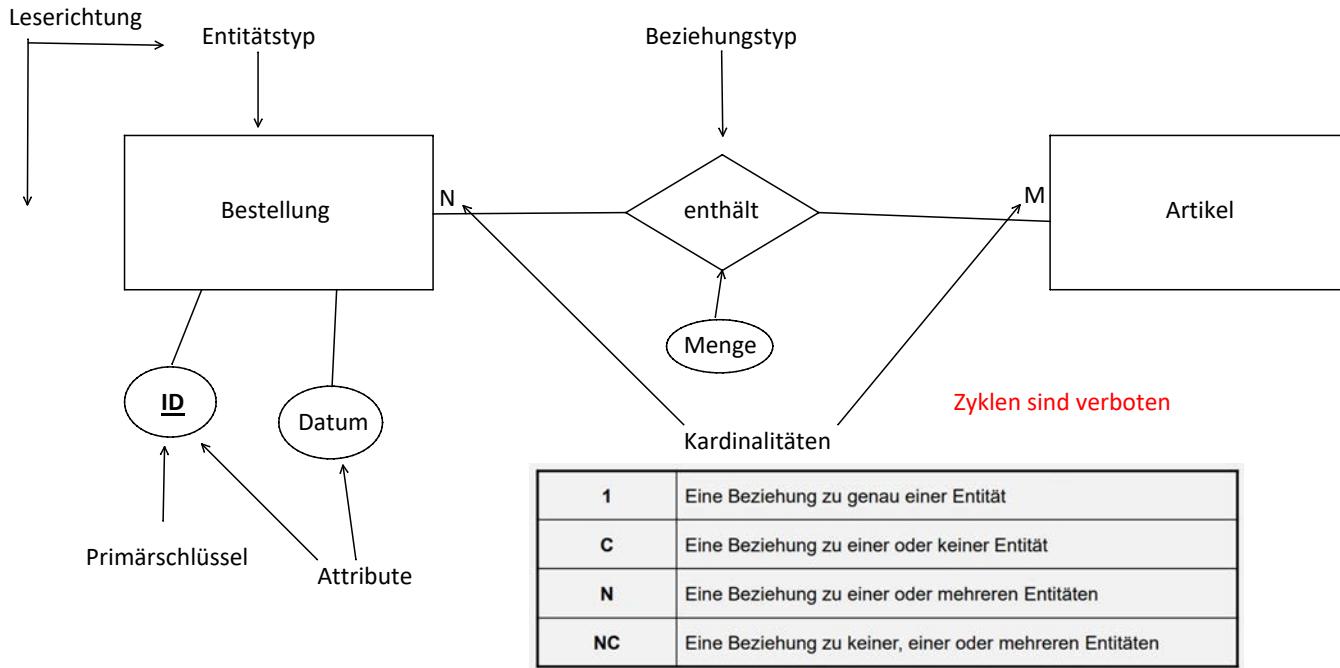
- DQL (Data Query Language)
 - o SELECT
- DML (Data Manipulation Language)
 - o INSERT, UPDATE, DELETE
- DDL (Data Definition Language)
 - o CREATE, ALTER, DROP
- DAL (Data Administration Language)
 - o TCL
 - Transaction Control Language
 - o DCL
 - Data Control Language



Modell:

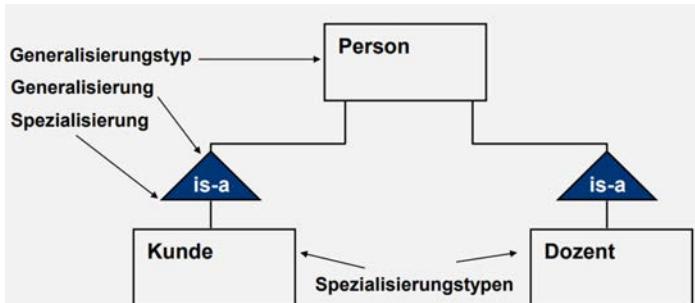
- Eigenschaften
 - o Abbildung
 - o Verkürzung
 - o Zweckbezug
- Ziele
 - o Transparenz
 - o Wissensbewahrung
 - o Durchführung von Experimenten

Begriff	Erläuterung	Beispiel
Entität <i>entity</i> (oft auch „Objekt“)	<ul style="list-style-type: none"> ▪ individuelles, identifizierbares Exemplar von Dingen, Personen oder Begriffen der realen Welt ▪ beschrieben durch Eigenschaften 	Kunde Meier
Entitätsmenge <i>entity set</i> (oft auch „Objekttyp“, „Entitätstyp“)	<ul style="list-style-type: none"> ▪ Zusammenfassung von Entitäten mit gleichartigen Eigenschaften ▪ Name (Substantiv) als Oberbegriff für alle Entitäten der Menge 	Kunde
Attribut <i>attribute, property</i>	<ul style="list-style-type: none"> ▪ Eigenschaft von allen Entitäten einer Entitätsmenge ▪ vorgegebener Wertebereich (auch „Domäne“ <i>domain</i>) ▪ Identifizierende Attribute bestimmen ein Objekt eindeutig („Schlüssel“ <i>key</i>) 	Kundennummer, Name, Adresse
Assoziation <i>relationship</i>	<ul style="list-style-type: none"> ▪ Zusammenfassung von gleichartigen Beziehungen zwischen Entitäten ▪ kann ebenfalls Attribute haben 	Kunde bucht Seminarveranstaltung



Generalisierung/Spezialisierung

- Hierarchie
- Attribute werden vererbt
- weitere Attribute kommen hinzu

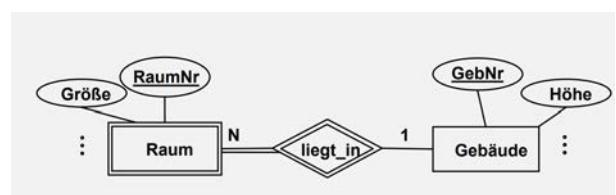


Abhängige Entitätsmenge:

- Eine Entität kann ohne die Existenz einer anderen nicht existieren
 - o Bsp.: Jedes Gebäude hat mindestens einen Raum, jeder Raum gehört zu einem Gebäude und kann ohne diesen nicht existieren

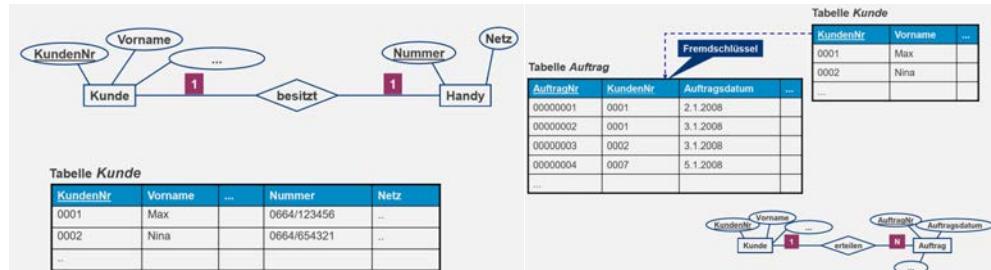
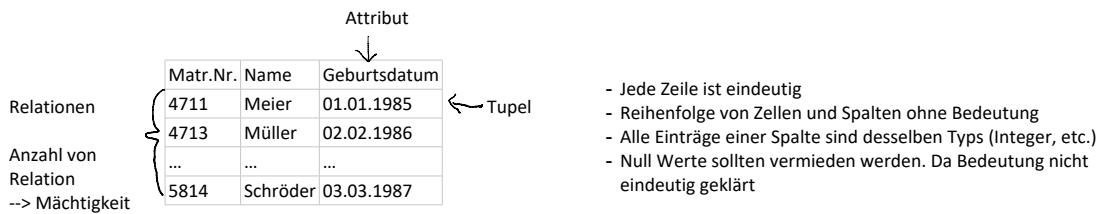
Rekursive Assoziation:

- Entitätsmenge zu sich selbst in Beziehung
- Diagramm wird um Rollennamen ergänzt



Relationales Modell

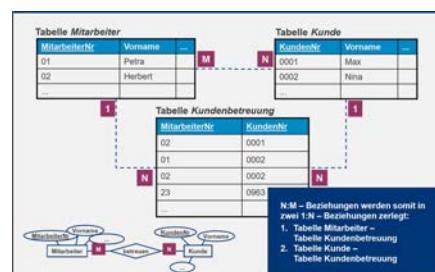
Sonntag, 28. Februar 2021 17:29



- 1:C Beziehungen (Eins zu einem oder keinem) sollten als 1:N behandelt werden, da mit 1:1 sonst Null Werte entstehen
- C:C sollte als M:N behandelt werden

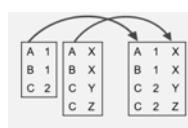
Tabellenoperationen:

- Vereinigung: Alle Einträge, die in Tabelle A oder B vorkommen
- Durchschnitt: Alle Einträge aus Tabelle A und B
- Differenz: Alle Einträge, die in A und nicht in B vorkommen
- Selektion: where Name = "" | Operationen: =,<,<=,>,>= und <> (not)
- Kartesisches Produkt

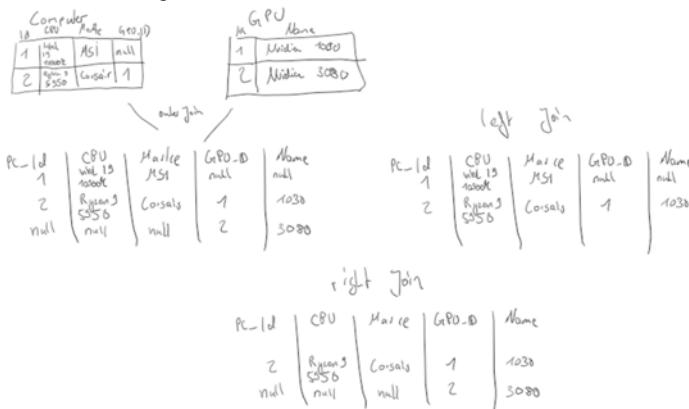


Joins:

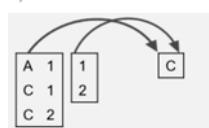
- Join Operatoren verbinden zwei Relationen, die in einer Beziehung Zueinander stehen
 - Theta-Join: Ist eine Operation bei der zuerst das kartesische Produkt und auf das Ergebnis dann eine Selection ausgeführt wird
 - Equi-Join: Ist ein Theta-Join, der aber nur eine Vergleichsoperation zulässt (=)



Outer, Left und right Join:



- Division: Alle aus A, die mit allen aus B verknüpft sind.



Anomalien in schlechten Datenmodellen:

Erklärt auf folgender Tabelle:

PersonalNr	Name	AbteilungNr	ProjNr	Projekt
01	Müller	4711	P10	SAP-Schmutz
01	Müller	4711	P11	Datenbanken
02	Nello	4712	P20	SAP-Schmutz

INSERT-Anomalie:

Projekte können nicht ohne Mitarbeiter angelegt werden. (zumindest wenn NULL-Werte vermieden werden)

DELETION-Anomalie:

Das Löschen eines Projektes führt eventuell zum Löschen des Mitarbeiters

UPDATE-Anomalie:

Mitarbeiter müssen in mehreren Zeilen angepasst werden.

Funktionale Abhängigkeiten:- $\{X,Y\} \rightarrow \{N,E,L,O\}$ - $\{X\} \rightarrow \{N,E,L,O\}$ N, E, L, O sind **funktional** Abhängig von X,Y , d.h. X,Y ist ein SchlüsselN, E, L, O sind **voll funktional** Abhängig von X, d.h. X ist ein Schlüsselkandidat**Begriffe:**

- Ein **Schlüssel** eine Menge von Attributen, die eine Zeile/Tupel einer Tabelle eindeutig identifiziert
- Ein **Schlüsselkandidat** ist ein Schlüssel mit minimaler Anzahl an Attributen
- Eine **Relation** kann mehrere Schlüsselkandidaten haben
- Ein **Primärschlüssel** ist ein beliebig ausgewählter Schlüsselkandidat, der zur Identifizierung jeder Zeile genutzt wird.
- Ein **zusammengesetzter Primärschlüssel** ist ein Primärschlüssel mit mehreren Attributen
- Ein **Schlüsselattribut** ist ein Attribut, das mindestens Teil eines Schlüsselkandidaten ist.

Normalformen:**1. Normalform:**

- o Alle Attribute sind atomar

Name
Max Mustermann

- o Ist nicht atomar

2. Normalform:

- o Die erste Normalform gilt und:
- o Jedes Nicht-Schlüssel-Attribut ist von jedem Schlüsselkandidaten **voll funktional abhängig**

 $\{X\} \rightarrow \{N,E,L,O\}$ $\{X\} \rightarrow \{Y\}$ $\{Y\} \rightarrow \{Z\}$

X	Y	N	E	L	O	Z
---	---	---	---	---	---	---

- o Z ist nicht von X abhängig

X	Y
---	---

X	N	E	L	O
---	---	---	---	---

Y	Z
---	---

Vor- und Nachteile:

- Qualität des Datenmodells
 - o Verständlichkeit
 - o Vermeidung von Anomalien
 - o Eliminierung von Redundanzen
 - o Robusteres Datenmodell
 - o Korrekte Realitätsabbildung
- Nachteil:
 - o Höhere Komplexität
 - o Joins benötigt (Performance)
 - o Ggf. Verzicht auf Normalisierung

3. Normalform:

- o Die zweite Normalform gilt und:
- o Es gibt keine funktionalen Abhängigkeiten zwischen Nicht-Schlüsselattributen

 $\{X\} \rightarrow \{N,E,L,O\}$ $\{X\} \rightarrow \{Y\}$ $\{N\} \rightarrow \{Y\}$

X	N	Y	E	L	O
---	---	---	---	---	---

- o Y ist von N funktional abhängig

X	N	E	L	O
---	---	---	---	---

N	Y
---	---

DQL (Data Query Language): Abfrage und Zusammenstellung von Daten

Legende:

[]	optional
{}	Auswahl
Spalten	Spalte1, Spalte2, ...

SELECT:

```
SELECT [DISTINCT] {*} | Spalten
[INTO var_name]
FROM Tabelle [Alias]
[WHERE {Bedingung | Unterabfrage}]
[GROUP BY Spalten [HAVING {Bedingung | Unterabfrage}]]
[ORDER BY Spalten [ASC | DESC]...];
```

SELECT [DISTINCT]	Wähle Werte aus der Spalte [mehrfache Datensätze nur einmal]
INTO	Speichere das Ergebnis in Variable
FROM [Alias]	Aus der Tabelle [Tabelle als Zwischenvariable]
WHERE	Wobei Bedingungen erfüllt sein müssen
GROUP BY	Gruppiere die Ausgabe von Zeilen mit gleichem Attributwert zu einer
HAVING	Wobei Bedingungen erfüllt sein müssen
ORDER BY [ASC/DESC]	Sortiere nach den Spalten [auf-/absteigend]

Klauseln:

WHERE Operatoren	=, <>, <, <=, >, >=, IN, BETWEEN, LIKE
Verknüpfungen	AND, OR, NOT, Klammersetzung

ASC:	A, B, C
DESC:	C, B, A

Beispiele:

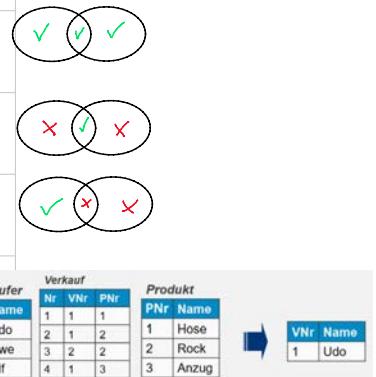
```
SELECT Kunde FROM Rechnung Alternativ:
WHERE Nummer = 10; Nummer IN (1, 3, 5); Nummer BETWEEN 1 AND 3; Name LIKE '__!%' NOT Name = 'Bill'
```

Nummer IS NULL; Nummer IS NOT NULL **Nummer = NULL**

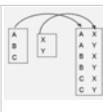
String Formatierungen -> siehe PL/SQL

Unterabfragen (SELECT, FROM, WHERE, WITH, HAVING):

EXISTS	Gibt Ergebnis zurück, wenn Ausdruck wahr ist	SELECT Country.Name FROM Country WHERE EXISTS (SELECT * FROM City WHERE City.Population > 1000 AND City.Country = Country.Code);
UNION	Vereinigung von Ausdrücken/Tabellen	(SELECT Name FROM Country WHERE Area > 1000) UNION (SELECT Name FROM Country WHERE Population > 10000);
INTERSECT	Schnittmenge von Ausdrücken/Tabellen	... INTERSECT ...
MINUS oder EXCEPT	Differenz von Ausdrücken/Tabellen	... MINUS ...
DIVISION	Nicht direkt umsetzbar. Trick mit doppelter Verneinung.	SELECT * FROM Verkaeufer WHERE NOT EXISTS (SELECT PNr FROM Produkt WHERE NOT EXISTS (SELECT PNr FROM Verkauf WHERE Verkauf.PNr = Produkt.PNR AND Verkauf.VNr));



JOINS:

CROSS JOIN	SELECT * FROM Tabelle1 CROSS JOIN Tabelle2	
	SELECT * FROM Tabelle1, Tabelle2	
INNER JOIN	SELECT * FROM Tabelle1 INNER JOIN Tabelle2 ON Tabelle1.Nr = Tabelle2.Nr	
	... JOIN ...	
LEFT JOIN	... LEFT JOIN ...	



RIGHT JOIN	... RIGHT JOIN ...	
FULL JOIN	... FULL JOIN ...	
NATURAL JOIN	... NATURAL [LEFT RIGHT FULL] JOIN ...	

Aggregatfunktionen:

COUNT	SELECT COUNT(Spaltenname) FROM Tabelle;	Anzahl Datensätze
SUM	SELECT SUM(Spaltenname) FROM Tabelle;	Summiert numerische Werte
MIN	SELECT MIN(Spaltenname) FROM Tabelle;	Minimum numerischer Werte
MAX	SELECT MAX(Spaltenname) FROM Tabelle;	Maximum numerischer Werte
AVG	SELECT AVG(Spaltenname) FROM Tabelle	Durchschnitt numerischer Werte
... AS	SELECT COUNT(Spaltenname) AS CountName FROM Tabelle;	Ausgabespalte unbennen
...	SELECT COUNT(DISTINCT Spaltenname) FROM Tabelle;	Beachte nur verschiedene Datensätze
DISTINCT		

GROUP BY	Beispiel: GROUP BY	Gruppieren von selben Werten																																																																																																	
	<table border="1" data-bbox="198 887 436 1224"> <thead> <tr> <th>Modell</th> <th>Jahr</th> <th>Farbe</th> <th>Anzahl</th> </tr> </thead> <tbody> <tr><td>Opel</td><td>1990</td><td>rot</td><td>5</td></tr> <tr><td>Opel</td><td>1990</td><td>weiß</td><td>87</td></tr> <tr><td>Opel</td><td>1990</td><td>blau</td><td>62</td></tr> <tr><td>Opel</td><td>1991</td><td>rot</td><td>54</td></tr> <tr><td>Opel</td><td>1991</td><td>weiß</td><td>95</td></tr> <tr><td>Opel</td><td>1991</td><td>blau</td><td>49</td></tr> <tr><td>Opel</td><td>1992</td><td>rot</td><td>31</td></tr> <tr><td>Opel</td><td>1992</td><td>weiß</td><td>54</td></tr> <tr><td>Opel</td><td>1992</td><td>blau</td><td>71</td></tr> <tr><td>Ford</td><td>1990</td><td>rot</td><td>64</td></tr> <tr><td>Ford</td><td>1990</td><td>weiß</td><td>62</td></tr> <tr><td>Ford</td><td>1990</td><td>blau</td><td>63</td></tr> <tr><td>Ford</td><td>1991</td><td>rot</td><td>52</td></tr> <tr><td>Ford</td><td>1991</td><td>weiß</td><td>9</td></tr> <tr><td>Ford</td><td>1991</td><td>blau</td><td>55</td></tr> <tr><td>Ford</td><td>1992</td><td>rot</td><td>27</td></tr> <tr><td>Ford</td><td>1992</td><td>weiß</td><td>62</td></tr> <tr><td>Ford</td><td>1992</td><td>blau</td><td>39</td></tr> </tbody> </table> <p>SELECT Modell, Jahr, SUM(Anzahl) FROM Autoverkäufe GROUP BY Modell, Jahr</p> <table border="1" data-bbox="531 977 754 1156"> <thead> <tr> <th>Modell</th> <th>Jahr</th> <th>Anzahl</th> </tr> </thead> <tbody> <tr><td>Opel</td><td>1990</td><td>154</td></tr> <tr><td>Opel</td><td>1991</td><td>198</td></tr> <tr><td>Opel</td><td>1992</td><td>156</td></tr> <tr><td>Ford</td><td>1990</td><td>189</td></tr> <tr><td>Ford</td><td>1991</td><td>116</td></tr> <tr><td>Ford</td><td>1992</td><td>128</td></tr> </tbody> </table>	Modell	Jahr	Farbe	Anzahl	Opel	1990	rot	5	Opel	1990	weiß	87	Opel	1990	blau	62	Opel	1991	rot	54	Opel	1991	weiß	95	Opel	1991	blau	49	Opel	1992	rot	31	Opel	1992	weiß	54	Opel	1992	blau	71	Ford	1990	rot	64	Ford	1990	weiß	62	Ford	1990	blau	63	Ford	1991	rot	52	Ford	1991	weiß	9	Ford	1991	blau	55	Ford	1992	rot	27	Ford	1992	weiß	62	Ford	1992	blau	39	Modell	Jahr	Anzahl	Opel	1990	154	Opel	1991	198	Opel	1992	156	Ford	1990	189	Ford	1991	116	Ford	1992	128	
Modell	Jahr	Farbe	Anzahl																																																																																																
Opel	1990	rot	5																																																																																																
Opel	1990	weiß	87																																																																																																
Opel	1990	blau	62																																																																																																
Opel	1991	rot	54																																																																																																
Opel	1991	weiß	95																																																																																																
Opel	1991	blau	49																																																																																																
Opel	1992	rot	31																																																																																																
Opel	1992	weiß	54																																																																																																
Opel	1992	blau	71																																																																																																
Ford	1990	rot	64																																																																																																
Ford	1990	weiß	62																																																																																																
Ford	1990	blau	63																																																																																																
Ford	1991	rot	52																																																																																																
Ford	1991	weiß	9																																																																																																
Ford	1991	blau	55																																																																																																
Ford	1992	rot	27																																																																																																
Ford	1992	weiß	62																																																																																																
Ford	1992	blau	39																																																																																																
Modell	Jahr	Anzahl																																																																																																	
Opel	1990	154																																																																																																	
Opel	1991	198																																																																																																	
Opel	1992	156																																																																																																	
Ford	1990	189																																																																																																	
Ford	1991	116																																																																																																	
Ford	1992	128																																																																																																	

Auswertungsreihenfolge:

Schlüsselwort	Auswertungsreihenfolge	Inhalt
SELECT	6	Attribute, Aggregatfunktionen
FROM	1	Liste von Tabellen, deren Kreuzprodukt betrachtet wird
WHERE	2	Boolsche Bedingung, zur Auswertung von Zeilen der Tabellen (optional)
GROUP BY	3	Liste von Attributen, nach denen gruppiert wird (optional)
HAVING	4	Boolsche Bedingung mit Attributen aus der GROUP-BY-Zeile oder Aggregatfunktion zur Auswahl von Gruppen (optional)
ORDER BY	5	Attribute (oder Aggregatfunktionen bei GROUP-BY) für Sortierreihenfolge bei der Ausgabe (optional)

DML (Data Manipulation Language): Umgang mit Tabelleninhalten

Name	Syntax	Beispiel	Definition
INSERT	INSERT INTO Tabelle [(Spaltenname [, Spaltenname] ...)] {VALUES (Wert [, Wert] ...) <Anfrageausdruck>};	INSERT INTO Kunde VALUES ('Thorge', 'Mustermann', '+4912345', 'Itzehoe', 'Musterstraße 42') Wenn nicht alle Werte ausgefüllt sind, müssen die anderen definiert werden: INSERT INTO Kunde (Vorname, Nachname, Ort) VALUES ('Thorge', 'Mustermann', 'Itzehoe')	Daten einfügen
UPDATE	UPDATE Tabelle SET <Spaltenname> = <Wert>, ... [WHERE <Bedingung>];	UPDATE Kunde SET Vorname = 'Nello' WHERE KundenNr = 1337; UPDATE Kunde SET Ort = 'Itzehoe' WHERE Ort = 'Elmshorn';	Datensätze ändern
DELETE	DELETE FROM Tabelle [WHERE <Bedingung>]	DELETE FROM Kunden WHERE Name = 'Thorge'	Datensatz löschen

DAL (Data Administration Language)

Transaktion

Definition	Feste Folge von Operationen Operation nur Veränderung von Tabelleninhalt (kein Erstellen/Löschen von Tabellen)
ACID - Merkmale	Atomicity: Aktualisierung komplett oder gar nicht ausgeführt Consistency: Von konsistenten Zustand in konsistenten Zustand Isolation: Jede Aktualisierung isoliert von der anderen Durability: Operation ist dauerhaft

Anomalien

DIRTY READ	Wenn eine Transaktion Daten ändert, während eine andere Transaktion noch nicht fertig ist. Deswegen bricht T1 ab. Somit geht die Transaktion von T2 verloren.
LOST UPDATE	T1 überschreibt das write von T2. Hierbei hat T1 die Änderung von T2 nicht gelesen. Somit ist diese verloren gegangen.
PHANTOMPROBLEM	T2 berechnet die Summe der Konten 2 Mal (in kurzer Zeit). Dazwischen werden die Daten verändert.

Transaktionsmanager

BEGIN TRANSACTION	Start der Transaktion. Bei ORACLE implizit (wäre also ein Fehler)!
COMMIT	Vorherige Befehle werden endgültig in DB übernommen
ROLLBACK	Transaktion wird explizit (per Programm) oder implizit (Fehler bei Commit) abgebrochen. Änderung seit Ende letzter Transaktion werden verworfen.
SAVEPOINT	Ermittelt Sicherungspunkt. Mit nächstem ROLLBACK werden alle Befehle nach dem Sicherungspunkt verworfen.

Beispiel DAL

Erfolgreiche Transaktion	UPDATE Konten SET Kontostand = Kontostand - 100 WHERE Name = 'Thorge'; UPDATE Konten SET Kontostand = Kontostand + 100 WHERE Name = 'Nello'; COMMIT;
Abbruch	UPDATE Konten SET Kontostand = Kontostand - 100 WHERE Name = 'Thorge'; UPDATE Konten SET Kontostand = Kontostand + 100 WHERE Name = 'Nello'; ROLLBACK;
Abbruch Savepoint	UPDATE Konten SET Kontostand = Kontostand - 100 WHERE Name = 'Thorge'; SAVEPOINT S1; UPDATE Konten SET Kontostand = Kontostand + 100 WHERE Name = 'Nello'; ROLLBACK TO SAVEPOINT S1;

DDL = Data Definition Language:

CREATE TABLE	ALTER TABLE	DROP TABLE
CREATE TABLE <tabellenname> (<attributname> <datentyp>, ... <attributname> <datentyp>,);	ALTER TABLE <tabellenname> (ADD <attributname> <datentyp> RENAME <attributname> <datentyp> DROP PRIMARY KEY UNIQUE (<spaltenliste>) CONSTRAINT <constraintname> COLUMN <spaltenname> DISABLE PRIMARY KEY UNIQUE (<spaltenliste>) CONSTRAINT <constraintname> COLUMN <spaltenname> ENABLE PRIMARY KEY UNIQUE (<spaltenliste>) CONSTRAINT <constraintname> COLUMN <spaltenname> MODIFY <attributname> <datentyp>);	DROP TABLE <tabellenname>;
CREATE TABLE Verkäufer (Vnr INTEGER, Name VARCHAR2 (6), Status VARCHAR2 (7), Gehalt INTEGER, PRIMARY KEY (Vnr));		DROP TABLE Verkäufer;

Datentypen:

- Ganze Zahlen:
 - SMALLINT
 - INTEGER
 - BIGINT
- Festkommazahlen:
 - DECIMAL bzw. NUMERIC (Bei ORACLE NUMBER)
- Gleitkommazahlen:
 - FLOAT
 - REAL
 - DOUBLE PRECISION
 - NUMERIC (x,p) (X Zahlen gesamt, p davon Nachkommastellen)
(Bei ORACLE NUMBER(x,p))
- Texte
 - CHAR (q)
 - VARCHAR (q) (Bei ORACLE Varchar2(q))
- Zeiten
 - DATE (Bei ORACLE incl. Time)
 - sysdate ist die aktuelle Zeit des Systems
 - TIME
- Große Datenmengen
 - CLOB (Character Large Object)
 - BLOB (Binary Large Object)

Fremdschlüssel:

Ohne Löschweitergabe	Mit Löschweitergabe
CREATE TABKE Auftrag (AuftragsNr NUMBER (6) NOT NULL, KundenNr NUMBER (6) NOT NULL, ... PRIMARY KEY (AuftragsNr, KundenNr), FOREIGN KEY (KundenNr) REFERENCES Kunde(KundenNr));	CREATE TABKE Auftrag (AuftragsNr NUMBER (6) NOT NULL, KundenNr NUMBER (6) NOT NULL, ... PRIMARY KEY (AuftragsNr, KundenNr), FOREIGN KEY (KundenNr) REFERENCES Kunde(KundenNr) ON DELETE CASCADE);

Sequenzen:

Syntax	Beispiel
CREATE SEQUENCE <seq_name> [INCREMENT BY <integer>] [START WITH <integer>] [MAXVALUE <integer> NOMAXVALUE] [MINVALUE <integer> NOMINVALUE] [CYCLE NOCYCLE] ;	CREATE SEQUENCE seq_KundenNr INCREMENT BY 1 START WITH 1000 NOCYCLE ;

- Bei CYCLE fängt er wieder bei MINVALUE an, wenn ein MAXVALUE definiert ist.

CONSTRAINTS:

- Mit Constraints können Bedingungen für Attributwerte formuliert werden, die bei jeder Manipulation überprüft werden.
 - [CONSTRAINT <name>] <bedingung>
 - <bedingung> := CHECK (<boolesche_bedingung>)
- Es gibt Spalten- und Tabellen-Constraints
 - **Spalten Constraints:**
 - Name VARCHAR(10) NOAST NULL PRIMARY KEY
 - VNr NUMBER CHECK(VNr >= 1000)
 - **Tabellen Constraint:**

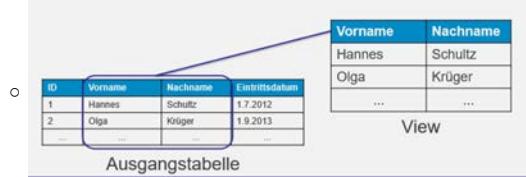
```
CREATE TABLE Kunde (
    ...
    Ort Varchar(50) NOT NULL,
    CONSTRAINT nurElmshorn CHECK (Ort='Elmshorn')
);
```

Löschweitergabe Arten:

- ON DELETE CASCADE
 - Löschen der Primärtabelle führt zum Löschen der Referenzierung
- ON DELETE RESTRICT
 - Standard: Delete wird nur ausgeführt, wenn keine Referenz existiert
- ON DELETE SET NULL
 - Spalten in der Referenzierung wird auf NULL gesetzt
- ON DELETE SET DEFAULT
 - Spalten in der Referenzierung wird auf DEFAULT gesetzt, wenn vorhanden, sonst auf NULL
- Dieselben gibt es auch für ON UPDATE

Views:

- Views sind virtuelle Tabellen, die aktuelle Informationen einer Tabelle abbilden.



- Mit Views kann man genauso arbeiten, wie mit normalen Tabellen.

- Vorteile:
 - Kompaktere Darstellung
 - Kann Abfragen beschleunigen
 - Stabilere Schnittstelle
 - Änderungen am Datenbankmodell bleiben verborgen
 - Ermöglicht Berechtigung auf gewisse Informationen

Syntax	Beispiel
CREATE VIEW <viewname> [(spaltennamen)] AS <abfrage>	CREATE VIEW Leichtteile AS SELECT TeilNr, Bezeichnung, Material, Gewicht FROM Teile WHERE Gewicht <10;
DROP VIEW <viewname>	DROP VIEW Leichtteile

- Inserts in View werden auf die Basistabelle ausgeführt.
 - Nur Spalten, die in der View stehen können eingetragen werden, alle anderen sind NULL
 - Kann zu Problemen führen wenn NOT NULL
 - Primärschlüssel kann nicht richtig gesetzt werden, wenn nicht in VIEW enthalten
 - Constraints können nicht immer beachtet werden.
- Es lassen sich auch Daten in einer View einfügen, die der Abfrage nicht entsprechen (Gewicht > 10)
 - Das lässt sich mit Check Option vermeiden.
 - Bsp.:


```
CREATE VIEW Leichtteile AS
    SELECT TeilNr, Bezeichnung, Material, Gewicht
    FROM Teile
    WHERE Gewicht <10
    WITH CHECK OPTION;
```
- Views können auch mehrere Tabellen abbilden.


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

Syntax:

WICHTIG:
Bei out muss der SERVEROUTPUT aktiviert werden mit:
SET SERVEROUTPUT ON;

Anonymer Block	Prozedur	Funktion
DECLARE (optional)	CREATE [OR REPLACE] PROCEDURE name (parameter 1, ...)	CREATE [OR REPLACE] FUNCTION name (parameter 1, ...)
BEGIN	IS	RETURN datatype
EXCEPTION (optional)	BEGIN -statements	IS
END;	BEGIN	
	EXCEPTION (optional)	RETURN value;
	END;	EXCEPTION (optional)
		END;
	Aufruf: EXECUTE name(vn_1,...)	Aufruf: EXECUTE name(vn_1,...)

IF-Block	LOOP	FOR LOOP
IF a = b THEN ELSIF a>b THEN ELSE END IF;	LOOP EXIT WHEN x = 42 X := x+1 END LOOP;	FOR X IN <start> .. <End> LOOP Dbms_Output.Put_Line('.');

WHILE LOOP	CURSOR
WHILE x != 42 LOOP X := x+1 END LOOP;	CURSOR cur_example IS SELECT name FROM continent; FOR rec IN cur_example LOOP ... END LOOP; Alternativ zu FOR LOOP: my_name continent.name%Type FETCH cur_example INTO my_name

Stringformatierung:

Generell mit '	'String'	String
Anführungszeichen in Strings mit 2 '	'String's'	String's
Begrenzungszeichen - Definition mit q und ' und Begrenzungszeichen	q'String's!'	String's
Bei Klammern werden beide genommen	q'[String's]'	String's
Konkatenation mit (z.B. bei Ausgaben)	Dbms_Output.Put_Line ('Nello ist ' '42');	Nello ist 42
Verwendung von _ und % bei Suchen/Vergleichen (hauptsächlich beim LIKE Operator)	_o%	Thorge, Tho, cool (Beispiele)
_ = Repräsentiert eine Zeichen % = Repräsentiert 0 oder mehr Zeichen		

Records:

- In Records können ganze Tupel gespeichert werden

Bsp.:
TYPE typename IS Record (feld1 type, feld2 type, ...)
TYPE Person IS RECORD (ID NUMBER, Name VARCHAR2(20), ...)

ACHTUNG: Ein kompletter Record kann nicht verglichen werden. Nur die einzelnen Felder
R_test.ID = 42
R_test.Name = "Nello"

- Mit %ROWTYPE wird die Struktur der ganzen Tabelle übernommen
R_test person%ROWTYPE

Sonstige Befehle:

Befehl	Bsp.:	Beschreibung
Dbms_Output.Put_Line('');	Dbms_Output.Put_Line('Test');	Gibt Test in der Console aus

Exceptions:

Vordefinierte Exception:	Eigene ExTception:	Eigener Sql-Fehler
-- SET SERVEROUTPUT ON --	CREATE OR REPLACE FUNCTION noNello (name VARCHAR2) RETURN VARCHAR 2 IS NelloException EXCEPTION; BEGIN IF name='Nello Musmerci' THEN RAISE NelloException; ELSE Return name; END IF;	Create or REPLACE FUNCTION noNello (name VARCHAR2) RETURN VARCHAR 2 IS BEGIN IF name='Nello Musmerci' THEN RAISE_APPLICATION_ERROR(-20001,'Das heißt Musmeci nicht Musmerci!'); ELSE Return name; END IF; END;
CREATE OR REPLACE PROCEDURE exTest IS I INTEGER DEFAULT 0; BEGIN I := I/ DBMS_OUTPUT.PUT_LINE('Nicht Erreicht'); EXCEPTION WHEN ZERO_DIVIDE THEN DBMS_OUTPUT.PUT_LINE(' SQLCODE ' SQLERRM); END;	EXCEPTION WHEN NelloException THEN RETURN 'Ein Nello '; WHEN OTHERS THEN RETURN 'Ein Fehler: ';	Mit RAISE_APPLICATION_ERROR(<Nummer>,<Text>); können eigene Exceptions definiert werden. Die Nummer muss zwischen -20999 und -20000 liegen.

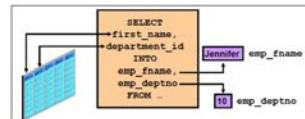
Operatoren:

+,-	Vorzeichen
* , / , +,-	Mathematische Operatoren
**	Exponentieller Operator
-	Konkatenation
- = , < , > , <= , >=	Vergleiche
- IS NULL, LIKE BETWEEN, IN	Logische Operatoren
- NOT, AND, OR	

Variablen:

VariablenTypen:
- NUMBER
- CHAR
- BINARY_INTEGER
- BINARY_FLOAT
- BINARY_DOUBLE
- BFILE
- VARCHAR2 Es muss immer eine Länge angegeben sein
- BOOLEAN
- DATE
- BLOB
- TIMESTAMP
- TIMESTAMP WITH TIME ZONE
- INTERVAL YEAR TO MONTH
- INTERVAL DAY TO SECOND
- CLOB
- %TYPE Bsp.: Table_name.example%TYPE Der Typ der Spalte " in der Tabelle "table_name"

Variablen können Temporär Daten gespeichert werden:



Convention Für Variablen:

- Müssen mit einem Buchstaben beginnen
- Können Buchstaben und Zahlen enthalten
- Können Sonderzeichen enthalten
- Dürfen maximal 30 Zeichen haben
- Dürfen keine reservierten Wörter enthalten

Variablen Declaration in DECLARE oder IS:
Vn_example NUMBER;
Vv_example VARCHAR2(16) := 'Nello der sweety';

Variablen können auch Konstanten sein:
Vn_example CONSTANT NUMBER := 42;

Konstanten müssen initialisiert werden.

Variablen können auch NOT NULL sein:
Vn_example NUMBER NOT NULL := 42;

Auch NOT NULL Variablen müssen initialisiert werden

SEQUENCE:

Abfragen des jetzigen Standes, bzw. des nächsten Standes erfolgt mit

SELECT seqName.CURRVAL [seqName.NEXTVAL] FROM DUAL;

Trigger:

FOR EACH ROW	INSTEAD OF
CREATE [OR REPLACE] TRIGGER <Triggername> (BEFORE AFTER) {INSERT DELETE UPDATE} [OF {Spaltenliste}] [OR {INSERT DELETE UPDATE} [OF {Spaltenliste}]] [OR {INSERT DELETE UPDATE} [OF {Spaltenliste}]] [...] ON <Tabellenname> [FOR EACH ROW] [WHEN <Bedingung>] <Anonymer-Block>;	CREATE [OR REPLACE] TRIGGER <Triggername> INSTEAD OF {INSERT DELETE UPDATE} [OF {Spaltenliste}] [OR {INSERT DELETE UPDATE} [OF {Spaltenliste}]] [OR {INSERT DELETE UPDATE} [OF {Spaltenliste}]] [...] ON <Viewname> [FOR EACH ROW] <Anonymer-Block>;
Bei Benutzung von BEFORE und FOR EACH ROW können die Werte neuen und alten Werte einer Spalte abgefragt werden. Mit :NEW.FieldName oder mit :OLD.FieldName	
Die neuen Werte können immer mit :NEW.fieldname abgefragt werden!	Die neuen Werte können immer mit :NEW.fieldname abgefragt werden!

CURSOR_ALREADY_OPEN ORA-06511 Cursor bereits geöffnet	Mit NelloException EXCEPTION; können eigene Exceptions definiert werden.	
DUP_VAL_ON_INDEX ORA-00001 Schlüssel doppelt eingetragen		
INVALID_CURSOR ORA-01001 Cursor nicht geöffnet		
INVALID_NUMBER ORA-01722 bei impliziter Typkonvertierung		
LOGIN_DENIED ORA-01017 Benutzername oder Passwort falsch		
NO_DATA_FOUND ORA-01403 SELECT liefert kein Ergebnis		
NOT_LOGGED_ON ORA-01012 Keine Verbindung zur Datenbank		
PROGRAM_ERROR ORA-06501 Interner PL/SQL - Fehler		
ROWTYPE_MISMATCH ORA-06504 Strukturvariablen inkompatibel		
STORAGE_ERROR ORA-06500 Speicherprobleme		
TIMEOUT_ON_RESOURCE ORA-00051 Datenbank-Sperre		
TOO_MANY_ROWS ORA-01422 SELECT liefert mehr als eine Zeile		
VALUE_ERROR ORA-06502 meistens: Zeichenkette zu kurz definiert		
ZERO_DIVIDE ORA-01476 durch 0 geteilt		

FindMactory

Kunde		Bestellung		Artikel			
KNr	Name	KNr	ANr	ANr	Name	Kategorie	Bestand
1	Musmerci	1	2	1	Nievida RTX 3080	GPU	1
2	DieMars	1	3	2	AMK 5600X	CPU	20
3	Nöhrens	2	2	3	Intel NEW 14nmKF CPU	CPU	50
4	FürchteNicht	3	2				

View KundeHatBestelltKNrKNA

KNr Name ANr Name Bestand

Prozedur

```
SET SERVEROUTPUT ON;
-- Bei gegebener Artikelnummer und Anzahl (für eine Bestellung) wird der Bestand verringert,
-- sofern der Bestand damit nicht negativ wird.
-- Wäre dies der Fall, wird eine (eigene) Exception geworfen.
```

```
CREATE OR REPLACE PROCEDURE VerringereBestandBeiBestellung(A_Nr Number, Anzahl Number)
```

```
IS
  vn_bestand Artikel.Bestand%Type;
  vn_bestandNachBestellung Artikel.Bestand%Type;
  ve_bestandNegativ EXCEPTION;
```

BEGIN

```
  Dbms_Output.Put_Line(A_Nr);
  SELECT Artikel.Bestand INTO vn_bestand FROM Artikel WHERE Artikel.ANr = A_Nr;
  vn_bestandNachBestellung := vn_bestand - Anzahl;
```

```
  IF vn_bestandNachBestellung < 0
    THEN RAISE ve_bestandNegativ;
```

```
  ELSIF vn_bestandNachBestellung < 5
    THEN Dbms_Output.Put_Line('Achtung! Der Bestand von ' || A_Nr || ' beträgt ' || vn_bestandNachBestellung);
    UPDATE Artikel
```

```
      SET Bestand = vn_bestandNachBestellung
      WHERE Artikel.ANr = A_Nr;
```

ELSE

```
  UPDATE Artikel
    SET Bestand = vn_bestandNachBestellung
    WHERE Artikel.ANr = A_Nr;
```

END IF;

EXCEPTION

```
  WHEN ve_bestandNegativ THEN
    RAISE_APPLICATION_ERROR(-20001, 'Achtung! Der Bestand von ' || A_Nr || ' reicht nicht für die Bestellung');
```

END;

Trigger

```
/*Insert eine neue Bestellung auf KundeHatBestellt, soll ein Insert Bestellung und
den Bestand um 1 verringern. Der Kunde existiert bereits*/
```

```
CREATE OR REPLACE Trigger Bestellung_hinzufügen
INSTEAD OF
```

```
INSERT ON KundeHatBestellt
```

BEGIN

```
  VerringereBestandBeiBestellung(:NEW.Anr, :NEW.Bestand);
  INSERT INTO Bestellung VALUES(:NEW.KNr, :NEW.ANr);
```

END;

Prozedur + Cursor

```
/*Diese Procedure gibt alle in der Konsole aus, die keine RTX 3080 bekommen
haben*/
```

```
CREATE OR REPLACE PROCEDURE keinRaytracing
IS
```

```
  Cursor cur_cur IS SELECT Name FROM Kunde
    WHERE NOT EXISTS (
      SELECT B.KNr FROM Bestellung B
        INNER JOIN Artikel A
          ON A.ANr = B.ANr
        WHERE A.Name = 'Nievida RTX 3080'
    );
```

BEGIN

```
  DBMS_Output.Put_Line('Diese Kunden haben eine Nievida RTX 3080');
```

```
  gekauft:');
```

```
  For rec IN cur_cur
  LOOP
    DBMS_Output.Put_Line(rec.Name);
  END LOOP;
END;
```