

# WIRTSCHAFTSINFORMATIK 2

DATENBANKEN: SQL

BERND BLÜMEL, CHRISTIAN BOCKERMANN,  
CHRISTIAN METZGER

HOCHSCHULE BOCHUM

WINTERSEMESTER 2022/2023

## Inhalt

- 1 Wiederholung
- 2 SQL - Structured Query Language
- 3 INSERT – Daten einfügen
- 4 SELECT – Abfragen von Daten
- 5 Gruppieren mit GROUP BY

## Vom Geschäftsprozess zur Datenbank

### Geschäftsprozess



Warenkorb		
Kunden Nr.	20010	
Bestellzeit	04.11.2016 10:07	
Artikel	Bestandmenge	Preis
MP	1	10,20 €
2000	2	24,00 €
300	1	10,00 €
In 20% Gewinnen		10,00 €
Verkaufskosten		20,00 €
Gesamtbetrag		64,20 €

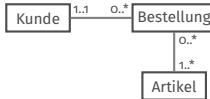
“Kunde bestellt Artikel.”

“Bestellungen haben mindestens 1 Artikel.”

Interviews mit  
Fachabteilungen

Textuelle Beschreibungen,  
Use-Case Diagramme

### Entity-Relationship Modell



### Entitätstypen

### Relationen

ER Diagramm  
in der Sprache UML

### Datenbank

Bestellung		
BestellNr	Datum	KundeNr
1	2016-05-01	1
2	2016-05-04	2
3	2016-05-09	3
4	2016-05-17	1

Artikel		
ArtikelNr	Bezeichnung	Preis
1	Lasermaus	17,99
2	Tastatur	27,99
3	Flexplatte	59,99
4	USB-Stick	9,99
5	WebCam	19,99

Kunde				
KundeNr	Name	Vorname	PLZ	Ort
1	Maier	Mia	44801	Bochum
2	Schmidt	Jonas	44801	Bochum
3	Müller	Emma	44800	Bochum
4	Weber	Lukas	45127	Essen

### SQL Schema Definition

Tabellen, Relationstabellen,  
Constraints/Regeln

Letzte Vorlesung

## Vom Konzept zur Implementierung

- SQL ist **standardisierte** Sprache für Datenbanken
- kleine Unterschiede / Dialekte zwischen Datenbanken
- SQL erlaubt Abfragen durch Benutzer + Programme



Abfrage (SQL)



Datenbank

## Vom Konzept zur Implementierung

- SQL ist **standardisierte** Sprache für Datenbanken
- kleine Unterschiede / Dialekte zwischen Datenbanken
- SQL erlaubt Abfragen durch Benutzer + Programme



A	B	C	D

Abfrage (SQL)



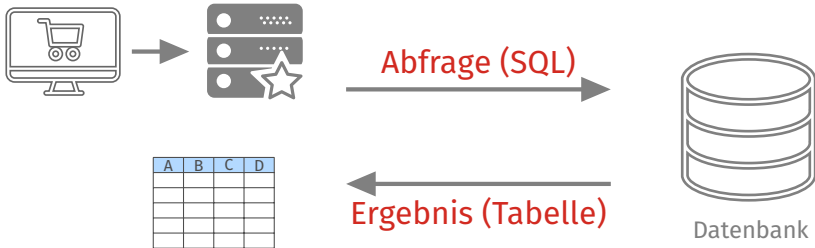
Ergebnis (Tabelle)



Datenbank

## Vom Konzept zur Implementierung

- SQL ist **standardisierte** Sprache für Datenbanken
- kleine Unterschiede / Dialekte zwischen Datenbanken
- SQL erlaubt Abfragen durch Benutzer + Programme

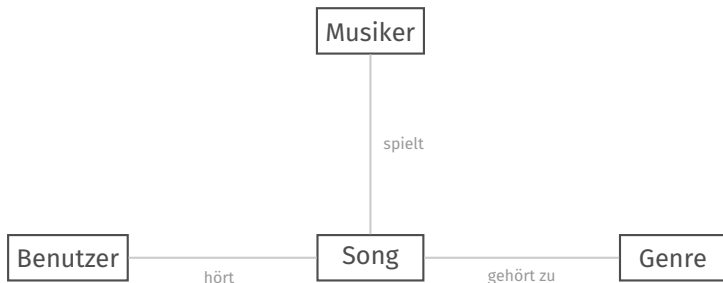


## Die Sprache SQL

Befehle für unterschiedliche Aktionen:

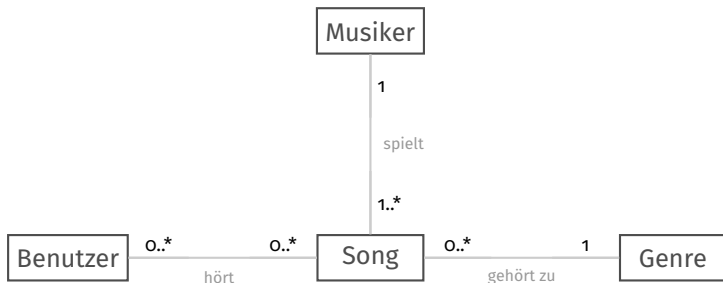
- CREATE – Datenbanken oder Tabellen anlegen
- INSERT – Datensätze in Tabellen eintragen
- UPDATE – Datensätze ändern
- DELETE – Datensätze löschen
- SELECT – Datensätze aus Tabellen selektieren

## Datenbank eines Streaming-Dienstes





## Datenbank eines Streaming-Dienstes



## Beispiel für Tabellen

Tabellenname	Primärschlüssel	weitere Attribute
Benutzer	Id	Name, Alter, Geschlecht, PLZ, Ort
Musiker	Id	Name
Genre	Id	Name
Song	Id	Titel, Laenge, <u>Genre</u> , <u>Musiker</u>
BenutzerSong	<u>Benutzer</u> , <u>Song</u>	Datum

## Anlegen/Definition von Tabellen

Beispiel für das Anlegen der Tabelle **Benutzer** als SQL Befehl:

```
CREATE TABLE Benutzer (  
    Id Integer NOT NULL,  
    Name Varchar(255),  
    Alter Int,  
    Geschlecht Varchar(1),  
    PLZ Varchar(5),  
    Ort Varchar(40),  
    CONSTRAINT pk PRIMARY KEY(Id)  
);
```

## Anlegen/Definition von Tabellen

Beispiel für das Anlegen der Tabellen **Musiker** und **Genre** als SQL Befehle:

```
CREATE TABLE Musiker (  
    Id Integer NOT NULL,  
    Name Varchar(255),  
    CONSTRAINT pk PRIMARY KEY(Id)  
);
```

```
CREATE TABLE Genre (  
    Id Integer NOT NULL,  
    Name Varchar(255),  
    CONSTRAINT pk PRIMARY KEY(Id)  
);
```

## Anlegen/Definition von Tabellen

Beispiel für das Anlegen der Tabelle **Song** als SQL Befehl mit **Foreign Keys** zu den Tabellen **Musiker** und **Genre**:

```
CREATE TABLE Song (  
  id Integer NOT NULL,  
  titel Varchar(255),  
  laenge Decimal,  
  genre Int,  
  musiker Int,  
  CONSTRAINT pk PRIMARY KEY(id),  
  CONSTRAINT genre_fk  
    FOREIGN KEY(genre) REFERENCES Genre(id),  
  CONSTRAINT musiker_fk  
    FOREIGN KEY(musiker) REFERENCES Musiker(id)  
);
```

## Anlegen/Definition von Tabellen

Beispiel für das Anlegen der Verbindungs-Tabelle  
**BenutzerSong** als SQL Befehl:

```
CREATE TABLE BenutzerSong (  
  benutzer Int,  
  song Int,  
  datum Timestamp,  
  CONSTRAINT pk PRIMARY KEY (benutzer, song),  
  CONSTRAINT benutzer_fk  
    FOREIGN KEY(benutzer) REFERENCES Benutzer(id),  
  CONSTRAINT song_fk  
    FOREIGN KEY(song) REFERENCES Song(id)  
);
```

# INSERT – Daten einfügen

## Daten einfügen mit **INSERT**

```
INSERT INTO tabelle [(spalte1, spalte2,..)]  
VALUES (wert1, wert2,..)
```



## Daten einfügen mit **INSERT**

```
INSERT INTO tabelle [(spalte1, spalte2,..)]  
VALUES (wert1, wert2,..)
```

### Beispiel:

```
INSERT INTO Genre (Id, Name) VALUES (1, 'Pop');
```

```
INSERT INTO Genre (Id, Name)  
VALUES (2, 'Rock'), (3, 'Blues');
```

# **SELECT – Abfragen von Daten**

## Datenbank zum Testen

Sie finden unter

`https://datascience.hs-bochum.de/sql/musik_db`

einen Web-Client mit dem Sie auf die Streaming-Datenbank zugreifen können.

## SELECT zum Abfragen von Datensätzen

```
SELECT [DISTINCT | ALL] spalten  
FROM tabelle(n)  
[WHERE bedingung]  
[GROUP BY spalten]  
[HAVING bedingung]  
[ORDER BY spalten]
```

## SELECT zum Abfragen von Datensätzen

```
SELECT spalten FROM tabellen WHERE bedingungen
```

- `spalten` steht für Tabellenspalten und/oder Ausdrücke (Formeln/Funktionen)
- `*` selektiert alle Spalten
- `tabellen` bezeichnet die Tabelle(n), aus denen abgefragt wird
- `bedingungen` beschränkt das Ergebnis

## Beispiel: **Abfrage aller Einträge der Tabelle Benutzer**

```
SELECT * FROM Benutzer
```

## Beispiel: Abfrage aller Einträge der Tabelle Benutzer

```
SELECT * FROM Benutzer
```

id	name	alter	geschlecht	plz	ort
1	Maier	27	m	44801	Bochum
2	Weber	29	w	44801	Bochum
3	Schmidt	19	m	44225	Dortmund
4	Blumenfeld	47	w	45127	Essen
5	Mustermann	58	w	45128	Essen
6	Blumenfeld	63	m	44789	Bochum
7	Dampf	59	w	44135	Dortmund
8	Rolland	51	w	44269	Dortmund
9	Farnsworth	35	m	44225	Dortmund

## Beispiel: Abfrage bestimmter Spalten einer Tabelle

```
SELECT name,plz,ort FROM Benutzer
```



## Beispiel: Abfrage bestimmter Spalten einer Tabelle

```
SELECT name,plz,ort FROM Benutzer
```

name	plz	ort
Maier	44801	Bochum
Weber	44801	Bochum
Schmidt	44225	Dortmund
Blumenfeld	45127	Essen
Mustermann	45128	Essen
Blumenfeld	44789	Bochum
Dampf	44135	Dortmund
Rolland	44269	Dortmund
Farnsworth	44225	Dortmund

## Beispiel: Benennung der Ergebnisspalten

```
SELECT name AS Nachname, plz, ort AS Stadt  
FROM Benutzer
```

Nachname	plz	Stadt
Maier	44801	Bochum
Weber	44801	Bochum
Schmidt	44225	Dortmund
Blumenfeld	45127	Essen
Mustermann	45128	Essen
Blumenfeld	44789	Bochum
Dampf	44135	Dortmund
Rolland	44269	Dortmund
Farnsworth	44225	Dortmund

## Beispiel: Nur die Spalte Ort

```
SELECT ort FROM Benutzer
```

ort
Bochum
Bochum
Dortmund
Essen
Essen
Bochum
Dortmund
Dortmund
Dortmund

## Beispiel: Nur disjunkte Werte der Spalte Ort

```
SELECT DISTINCT ort FROM Benutzer
```

ort
Bochum
Dortmund
Essen

## Beispiel: Abfrage mit Bedingung

```
SELECT * FROM Benutzer WHERE Ort = "Bochum"
```

## Beispiel: Abfrage mit Bedingung

```
SELECT * FROM Benutzer WHERE Ort = "Bochum"
```

SELECT liefert eine Tabelle als Ergebnis:

id	name	alter	geschlecht	plz	ort
1	Maier	27	m	44801	Bochum
2	Weber	29	w	44801	Bochum
6	Blumenfeld	63	m	44789	Bochum

## Weitere Möglichkeiten von SELECT

- Komplexere Bedingungen über AND und OR
- Selektieren ohne Duplikate mit `distinct`
- Zählen von Zeilen/Werten mit `count(*)` bzw. `count(spalte)`
- Arithmetik und Aggregationen (`min`, `max`, `sum`, `avg`)
- Sortierung mit ORDER BY

## Logische AND/OR Verknüpfungen von Bedingungen

Alle Benutzer mit Namen *Blumenfeld* aus Bochum oder Essen:

```
SELECT * FROM Benutzer
  WHERE name = 'Blumenfeld' AND ort = 'Bochum'
                                OR ort = 'Essen'
```



## Logische AND/OR Verknüpfungen von Bedingungen

Alle Benutzer mit Namen *Blumenfeld* aus Bochum oder Essen:

```
SELECT * FROM Benutzer
  WHERE name = 'Blumenfeld' AND ort = 'Bochum'
                                OR ort = 'Essen'
```

**Achtung! Klammerung wichtig!!**

```
SELECT * FROM Benutzer
  WHERE name = 'Blumenfeld' AND (ort = 'Bochum'
                                OR ort = 'Essen')
```

## Mit **LIKE** Texte abgleichen:

Alle Benutzer, deren Name mit 'Blum' beginnt:

```
SELECT * FROM Benutzer  
WHERE name LIKE 'Blum%'
```

## Mit **LIKE** Texte abgleichen:

Alle Benutzer, deren Name mit 'Blum' beginnt:

```
SELECT * FROM Benutzer
WHERE name LIKE 'Blum%'
```

Alle Benutzer, deren Postleitzahl mit '44' beginnt:

```
SELECT * FROM Benutzer
WHERE name LIKE '44%'
```

Platzhalter % steht für beliebige Zeichen, Platzhalter \_ steht für *ein* beliebiges Zeichen.

## Zählen von Zeilen/Werten

Wieviele Benutzer gibt es in der Tabelle?

```
SELECT COUNT(*) FROM Benutzer
```

## Zählen von Zeilen/Werten

Wieviele Benutzer gibt es in der Tabelle?

```
SELECT COUNT(*) FROM Benutzer
```

Wieviele Werte hat die Spalte **Ort**?

```
SELECT COUNT(ort) FROM Benutzer
```

## Zählen von Zeilen/Werten

Wieviele Benutzer gibt es in der Tabelle?

```
SELECT COUNT(*) FROM Benutzer
```

Wieviele Werte hat die Spalte **Ort**?

```
SELECT COUNT(ort) FROM Benutzer
```

Wieviele **verschiedene** Werte hat die Spalte **Ort**?

```
SELECT COUNT(DISTINCT ort) FROM Benutzer
```

## Zählen mit Bedingungen

Wieviele Benutzer aus Bochum gibt es in der Tabelle?

```
SELECT COUNT(*) FROM Benutzer  
WHERE ort = 'Bochum'
```

## Zählen mit Bedingungen

Wieviele Benutzer aus Bochum gibt es in der Tabelle?

```
SELECT COUNT(*) FROM Benutzer  
WHERE ort = 'Bochum'
```

Wieviele Benutzer aus Bochum oder Dortmund gibt es in der Tabelle?

```
SELECT COUNT(*) FROM Benutzer  
WHERE ort = 'Bochum' OR ort = 'Dortmund'
```



## Aggregationen mit MIN/MAX

```
SELECT MIN(alter), MAX(alter) FROM Benutzer
```

## Aggregationen mit MIN/MAX

```
SELECT MIN(alter), MAX(alter) FROM Benutzer
```

Weitere Aggregationsfunktionen z.B.

- SUM
- AVG

```
SELECT AVG(alter) AS Durchschnittsalter  
FROM Benutzer  
WHERE ort = 'Bochum'
```

# Gruppieren mit GROUP BY

**Frage:** Wie viele Benutzer kommen aus den verschiedenen Städten?

```
SELECT count(*)  
FROM Benutzer  
WHERE ort = 'Bochum'
```

count(*)
3

**Frage:** Wie viele Benutzer kommen aus den verschiedenen Städten?

```
SELECT count(*)  
FROM Benutzer  
WHERE ort = 'Bochum'
```

count(*)
3

```
SELECT count(*)  
FROM Benutzer  
WHERE ort = 'Essen'
```

count(*)
2

## Frage: Wie viele Benutzer kommen aus den verschiedenen Städten?

```
SELECT count(*)  
FROM Benutzer  
WHERE ort = 'Bochum'
```

count(*)
3

```
SELECT count(*)  
FROM Benutzer  
WHERE ort = 'Essen'
```

count(*)
2

```
SELECT count(*)  
FROM Benutzer  
WHERE ort = 'Dortmund'
```

count(*)
4

## Gruppierung: GROUP BY ort

id	name	alter	geschlecht	plz	ort
1	Maier	27	m	44801	Bochum
2	Weber	29	w	44801	Bochum
3	Schmidt	19	m	44225	Dortmund
4	Blumenfeld	47	w	45127	Essen
5	Mustermann	58	w	45128	Essen
6	Blumenfeld	63	m	44789	Bochum
7	Dampf	59	w	44135	Dortmund
8	Rolland	51	w	44269	Dortmund
9	Farnsworth	35	m	44225	Dortmund

id	name	alter	geschlecht	plz	ort
1	Maier	27	m	44801	Bochum
2	Weber	29	w	44801	Bochum
6	Blumenfeld	63	m	44789	Bochum

id	name	alter	geschlecht	plz	ort
4	Blumenfeld	47	w	45127	Essen
5	Mustermann	58	w	45128	Essen

id	name	alter	geschlecht	plz	ort
3	Schmidt	19	m	44225	Dortmund
7	Dampf	59	w	44135	Dortmund
8	Rolland	51	w	44269	Dortmund
9	Farnsworth	35	m	44225	Dortmund

**Frage:** Wie viele Benutzer kommen aus den verschiedenen Städten?

```
SELECT ort, count(*) FROM Benutzer  
GROUP BY ort
```

ort	count(*)
Bochum	3
Essen	2
Dortmund	4



**Frage:** Wie viele Benutzer kommen aus den verschiedenen Städten?

```
SELECT ort, count(*) FROM Benutzer  
GROUP BY ort  
HAVING count(*) > 2
```

ort	count(*)
Bochum	3
Dortmund	4