

Datenbanken

Informatik, ICS und als Wahlfach

6. Structured Query Language (SQL)

Prof. Dr. Markus Goldstein

SoSe 2022

6.1 Einführung in SQL

6.2 Datendefinition (DDL)

6.3 Datenmodifikation (DML)

6.4 Aggregatfunktionen

- Zweck und Bedeutung von SQL
- Anlegen einer Datenbank und Tabellen mit SQL **DDL** (Data Definition Language)
 - Anlegen von Tabellen mit Hilfe von `CREATE TABLE`
 - Definition von Fremdschlüsseln
 - Ändern von Tabellenstrukturen mit `ALTER TABLE`
- Datenpflege mittels SQL **DML** (Data Manipulation Language)
 - Einfügen neuer Datensätze mit `INSERT`
 - Änderung bestehender Datensätze mit `UPDATE`
 - Löschen von Datensätzen mit `DELETE`

- Auffinden von Daten in der Datenbank mit SQL **DML**
 - unter Verwendung von `SELECT`
 - `WHERE`-Klausel zur Eingrenzung der Ergebnismenge
 - Sortierte Ausgabe mittels `ORDER BY`
 - Verwendung der Aggregatfunktionen
 - Gruppierung mittels `GROUP BY` und `HAVING`
 - Verwendung von Unterabfragen
 - Verbundoperationen (`JOINS`) von Tabellen
 - Mengenoperationen (`UNION`, `INTERSECT`, `EXCEPT`)

- Idealerweise erlaubt eine Datenbanksprache ...
 - Erstellung und Modifikation der Datenbank und Relationenstrukturen
 - Einfügen, Modifizieren und Löschen von Daten aus Relationen
 - Einfache und komplexe Abfragen

- Soll nur minimalen Benutzeraufwand erfordern
 - Syntax und Kommandostruktur leicht erlernbar
 - Portabel (standardisiert)
 - Unterstützt die logische und physische Datenunabhängigkeit

- SQL ist eine Sprache mit den Hauptbestandteilen
 - **DDL** zur Definition der Datenbankstruktur
 - **DML** zum Auffinden und Ändern von Daten
- Und zusätzlichen Bestandteilen
 - *DCL* (Data Control Language) zur Rechtevergabe
 - *TCL* (Transaction Control Language) für Transaktionsunterstützung („bündelt“ mehrere Abfragen)
 - Ablaufkontrollkommandos (erst seit SQL3) „Programmiersprache“
 - Zusätzliche, herstellereigenspezifische Funktionalität

- SQL ist eine **deklarative Sprache**
 - Beschreiben, **was** man möchte, nicht **wie** man es bekommt
 - Für das **wie** ist das DBMS zuständig
 - Früher auch deklarative Programmiersprachen, z.B. Prolog
- Gegenteil: imperative Sprachen
 - Beschreibung des Lösungswegs (**wie**)
 - Java, C, C++, Python, ...

- SQL ist intuitiv: Benutzt einfache englische Worte

```
1) CREATE TABLE Laden (  
    LadNo    INTEGER AUTO_INCREMENT,  
    MwSt     INTEGER,  
    LadName  VARCHAR(15),  
    PRIMARY KEY (LadNo));  
  
2) INSERT INTO Laden VALUES  
    (NULL, 234567, 'Migros Zürich');  
  
3) SELECT LadNo, MwSt, LadName  
    FROM    Laden  
    WHERE   MwSt = 234567;
```


Kann von den **verschiedenen Benutzern** verwendet werden

- Datenbank Administrator
- Anwendungsentwickler
- Data Scientist (→ Direkte Auswertung von Daten: „Analytics“)
- andere Endbenutzer (Management, ...)
- **ISO Standard für SQL existiert**
 - Wird ständig erweitert: 14 Teilstandards, 8 Erweiterungen
 - SQL damit sowohl formal als auch de facto Standard für relationale Datenbanken
 - Dennoch Besonderheiten bei verschiedenen DBMS'

1974: Definition der Sprache ‘Structured English Query Language’ (SEQUEL) durch D. Chamberlin (IBM San Jose Laboratory)

1976: Nachfolgeversion SEQUEL/2 Namensänderung in SQL aus rechtlichen Gründen

- Aussprache ist offiziell ‘S-Q-L’ (manchmal noch „sie-quell“)
- Fertigstellung eines prototypischen DBMS namens System R durch IBM basierend auf SEQUEL/2.

1979-1985: Auftauchen von ORACLE mit dem ersten kommerziellen, auf SQL basierenden RDBMS

1987: Erster SQL Standard durch ANSI und ISO

1989: Addendum der ISO 'Integrity Enhancement Feature'

1992: Erste Revision zu SQL2 oder SQL/92.

1999: SQL3 (SQL/99) mit Unterstützung von Objektorientierung und Kontrollstrukturen

Seit 2000: Immer weitere Neuerungen (z.B. Java-Typunterstützung (JRT), XML, Geoinformationen,...)

Ein SQL-Statement besteht aus reservierten Wörtern und benutzerdefinierten Wörtern

- Reservierte Wörter
 - sind fester Bestandteil von SQL
 - müssen wie vorgegeben geschrieben werden
 - dürfen nicht durch Zeilenumbruch getrennt werden
- Benutzerdefinierte Wörter
 - vom Benutzer festgelegt
 - stehen für die Namen verschiedener Datenbankobjekte wie Relationen, Spalten, Views etc.

Bis auf Literale sind alle Teile eines SQL-Statements von Groß- und Kleinschreibung unabhängig

Lesbarkeit wird durch Einrücken und Aufteilung auf Zeilen erhöht

- Neue Zeile für jede Klausel (Teil eines Statements)
- Gleiche Einrücktiefe für alle Klauseln
- Klauseln mit mehreren Teilen:
 - Jeder Teil in eigene Zeile
 - Einrückung unter den Beginn der Klausel

Verwendung der EBNF-Notation:

- RESERVIERTE WORTE in Großbuchstaben
- Benutzerdefinierte Worte in Kleinbuchstaben
- | bezeichnet Alternativen
- { } bezeichnen ein erforderliches Element
- [] bezeichnen ein optionales Element
- ... bezeichnet optionale Wiederholung (0 oder mehr)

Literale sind Konstanten in SQL Statements (wie in Java auch)

- Nicht-numerische Literale stehen in einfachen Anführungszeichen (z.B. 'London')
- Numerische Literale dürfen nicht in Anführungszeichen stehen (z.B. 65000).
- Sind case-sensitive
- **Reservierte Wörter** sind nicht case-sensitive, wir schreiben Sie aber immer GROSS
- **Benutzerdefinierte Wörter** (Bezeichner) können case-sensitive sein (abhängig vom DBMS), wir wählen daher eine einheitliche Schreibweise. Werden auch in ``backticks`` dargestellt.

6.1 Einführung in SQL

6.2 Datendefinition (DDL)

6.3 Datenmodifikation (DML)

6.4 Aggregatfunktionen

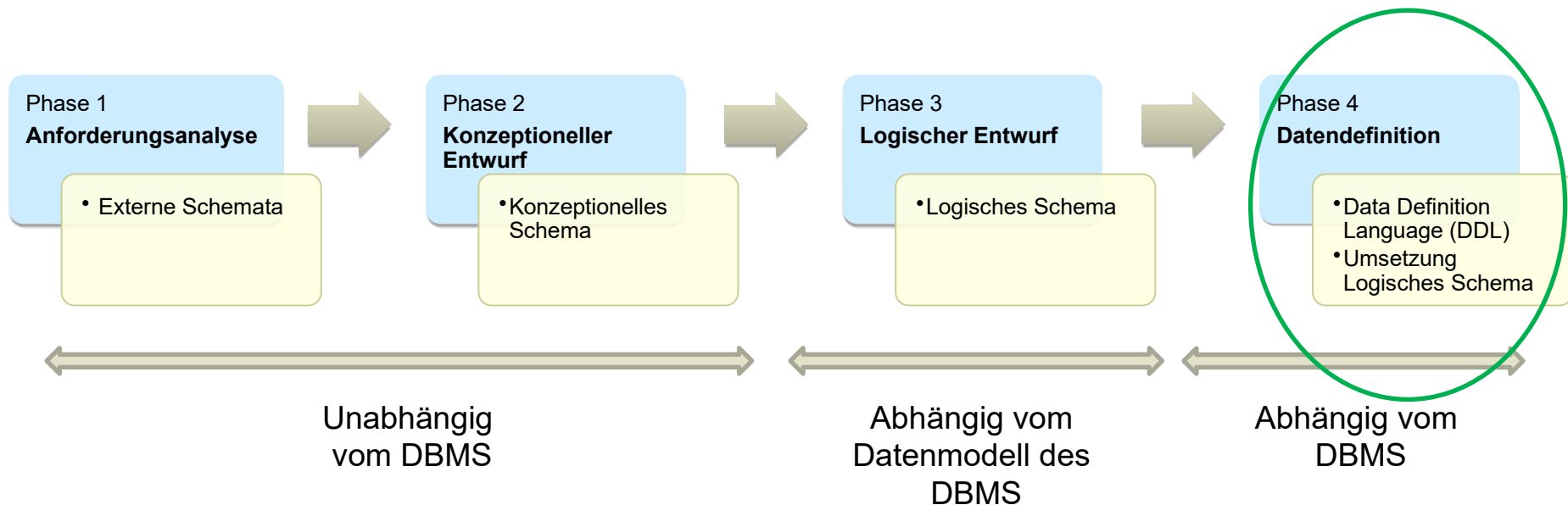
DDL (Data Definition Language) dient zum

- Anlegen von Tabellen
 - Spalten, Datentypen, Eingabepflicht, Standardwerte, Autoinkrement, Primärschlüssel
 - Fremdschlüssel
 - Tabellenstruktur kopieren
- Ändern von Tabellenstrukturen
 - Spaltenmanipulation

Datendefinition

Letzte Phase der Erstellung einer Datenbank.

Implementierung abhängig vom DBMS



(angelehnt an Thomas Kudraß, Taschenbuch Datenbanken, 2. Auflage, Hanser Verlag 2015, S. 45 ff)

Syntax CREATE TABLE

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name (  
  col_name data_type [NOT NULL | NULL]  
    [DEFAULT default_value] [AUTO_INCREMENT] , ... ,  
  [PRIMARY KEY (col_name)]  
);
```

- Pflicht-Elemente
 - Tabellenname, Spaltenname und Datentyp
- Optionale Angabe
 - NOT NULL → Eingabepflicht
 - DEFAULT → Standardwert falls ohne Wert beim Einfügen
 - AUTO_INCREMENT → Autom. Vergabe von PK-Werten
 - PRIMARY KEY → Primärschlüssel Spalte(n)

Anlegen von Tabellen

Beispiel: Tabelle Artikel in 2NF

```
CREATE TABLE Artikel (  
    ArtID INT NOT NULL AUTO_INCREMENT,  
    Bezeichnung VARCHAR(45) NOT NULL,  
    Sparte VARCHAR(15) DEFAULT 'unbekannt',  
PRIMARY KEY( ArtID )  
);
```

Syntax CREATE TABLE mit Fremdschlüssel

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name (  
  col_name data_type [NOT NULL | NULL]  
  [DEFAULT default_value] [AUTO_INCREMENT] , ... ,  
  [PRIMARY KEY (col_name)] ,  
  [CONSTRAINT [fk_name]] FOREIGN KEY (col_name)  
    REFERENCES src_tbl_name (src_col_name)  
    ON DELETE reference option  
    ON UPDATE reference option)  
);
```

- **FOREIGN KEY** → Spalte mit Fremdschlüsselwert
- **REFERENCES** → Tabelle (Spalte) des referenzierten Primärschlüssels
- **ON DELETE** → Maßnahmen bei Löschen des ref. Primärschlüssels
- **ON UPDATE** → Maßnahmen bei Ändern des ref. Primärschlüssels

- `fk_name` ist nur ein frei wählbarer Bezeichner, um den FK später noch einmal zu modifizieren
 - Empfohlene Bezeichnung: `fk_*`

Reference Option

RESTRICT | CASCADE | SET NULL | NO ACTION

- `RESTRICT` → Operation verbieten
- `CASCADE` → Operation für Fremdschlüssel auch durchführen
- `SET NULL` → Fremdschlüsselwert löschen (nur bei erlaubten `NULL`)
- `NO ACTION` → Keine Anpassung
(Vorsicht: referentielle Integrität möglicherweise verletzt!)

Beispiel: Tabelle Artikel in 3NF

```
CREATE TABLE IF NOT EXISTS Artikel (  
  ArtID INT(11) NOT NULL AUTO_INCREMENT,  
  Bezeichnung VARCHAR(45) NOT NULL,  
  SparteId INT NOT NULL,  
  PRIMARY KEY (ArtID),  
  CONSTRAINT fk_Sparte FOREIGN KEY (SparteId)  
    REFERENCES Sparte(SparteId)  
    ON DELETE RESTRICT  
    ON UPDATE CASCADE) ;
```

Referenziert Tabelle Sparte
und dort den PK SparteId

- Wird der PK der Sparte geändert, werden die FK's in Artikel geupdated
- Nicht möglich, eine Sparte zu löschen, falls noch Artikel der Sparte existieren

Definition von unikalen Spalten als Constraint:

```
CONSTRAINT constr_name UNIQUE (col_name)
```

- Verschiedene Schreibweisen möglich
(`UNIQUE KEY` oder `UNIQUE INDEX` auch möglich)
- Der Name kann beliebig gewählt werden (wie `fk_name` auch)
- Beispiel:

```
CREATE TABLE IF NOT EXISTS Artikel(  
  ArtID INT(11) NOT NULL AUTO_INCREMENT,  
  Bezeichnung VARCHAR(45) NOT NULL,  
  SparteId INT NOT NULL,  
  PRIMARY KEY (ArtID),  
  CONSTRAINT uq_Bez UNIQUE (Bezeichnung);
```


Anlegen von Tabellen

CREATE TABLE LIKE Tabellenstruktur kopieren (ohne Daten)

```
CREATE TABLE tbl_name1 LIKE tbl_name2;
```

- Kopiert nur die Tabellenstruktur, nicht die Daten

Änderungen über ALTER TABLE

- CHANGE - Spalte ändern

```
ALTER TABLE <Tabellenname>  
CHANGE <Spaltenname> <Spaltenname_neu> <Datentyp>  
[<Eigenschaft>] [ FIRST | AFTER <Spaltenname >];
```

- ADD - Neue Spalte hinzufügen

```
ALTER TABLE <Tabellenname>  
ADD <Spaltenname> <Datentyp> [<Eigenschaft>]  
[ FIRST | AFTER <Spaltenname >];
```

- DROP - Spalte entfernen

```
ALTER TABLE <Tabellenname> DROP COLUMN <Spaltenname>;
```

Änderungen über ALTER TABLE

- ADD PRIMARY KEY- Primärschlüssel definieren

```
ALTER TABLE <Tabellenname>  
ADD PRIMARY KEY <Keyname> (<Spaltenname>  
[,<Spaltenname>]) ;
```

- ADD FOREIGN KEY - Fremdschlüssel definieren

```
ALTER TABLE <Tabellenname>  
ADD [CONSTRAINT [fk_name]] FOREIGN KEY (<Spaltenname_1>  
,... ,<Spaltenname_n>])  
REFERENCES <STabellenname> (<SSpaltenname_1> ,... ,  
<Sspaltenname_n>])  
[ON DELETE|UPDATE reference option] ;
```

- DROP FOREIGN KEY - Fremdschlüssel (Constraint) entfernen

Umbenennen mit `RENAME TABLE`

```
RENAME TABLE <Tabellenname> TO <Tabellenname_neu>;
```

- **Vorsicht bei `ALTER TABLE`**
 - Bei `CHANGE`-Änderungen an Datentypen können Daten verändert werden! (z.B. Texte werden abgeschnitten, Kommazahlen gerundet, etc.)
 - Das Hinzufügen und Entfernen von Spalten erfordert die komplette Neuorganisation einer Tabelle – bei großen Tabellen kann dies sehr lange dauern
 - Nach Möglichkeit `ALTER TABLE` sparsam verwenden!

Aufgaben

Bitte bearbeiten Sie jetzt die Aufgaben in Moodle zum Kapitel 6.

- **Teil A**

6.1 Einführung in SQL

6.2 Datendefinition (DDL)

6.3 Datenmodifikation (DML)

6.4 Aggregatfunktionen

DML (Data Modification Language) dient zum

- Schreiben, Lesen, Ändern, Löschen von Daten (CRUD-Operationen)
- Daten lesen mit `SELECT` und `FROM`
 - **Projektion** durch Spaltenauswahl
 - Einschränkung der Zeilen mit `DISTINCT` und `WHERE`
 - Berechnungen mit Hilfe von Aggregationsfunktionen (`COUNT`, `SUM`, `MIN`, `MAX` und `AVG`)
 - Gruppieren von Ergebniszeilen mit `GROUP BY`
 - Auswahl von Gruppen anhand von Bedingung mit `HAVING`
 - Sortierung des Ergebnisses mit `ORDER BY`

Einfügen mit INSERT

- Das `INSERT`-Statement
- Syntax:

```
INSERT INTO <Tabellenname> [ ( Spaltenliste ) ]  
VALUES ( Werteliste );
```

Spaltenliste

- Angabe ist optional
- Ohne Angabe: alle Spalten in `CREATE TABLE` Reihenfolge
- Bevorzugt: Spalten benennen (→ Logische Datenunabhängigkeit!)

Einfügen mit INSERT

- Das `INSERT`-Statement
- Bei Spaltenliste und Werteliste beachten
 - jeweils gleiche Anzahl Elemente
 - korrespondierende Elemente haben gleiche Position
 - korrekte Datentypen
- Fehlende Spalten in Spaltenliste
 - beim Anlegen als `NULL` deklariert
 - oder `DEFAULT` war angegeben
 - bei MySQL auch `AUTO_INCREMENT` (beim Primärschlüssel) möglich

Einfügen mit INSERT

- Beispiel
- IdArtikel ist AUTO_INCREMENT



- **INSERT INTO** Artikel2NF (Bezeichnung , Sparte)
VALUES ('Klingenfix Rasierer', 'Kosmetik');
- **INSERT INTO** Artikel2NF **VALUES**
(**NULL** , 'Wattebäuschchen', 'Kosmetik ');
- Mehrere Zeilen auf einmal auch möglich
 - **INSERT INTO** SchuelerIn **VALUES**
(1, 'Oster ', 'Hase ', 'm'),
(2, 'Harry ', 'Hurtig ', 'm'),
(3, 'Horst ', 'Schlaemmer ', 'm'),
(4, 'Evje ', 'van Dampen ', 'w');

Lesen mit SELECT

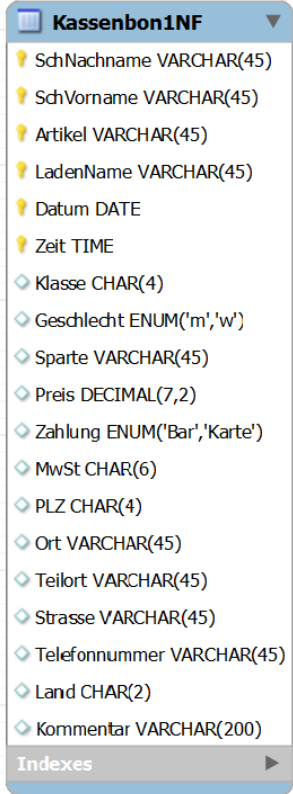
- Das SELECT-Statement
- Syntax:

```
SELECT [DISTINCT | ALL]
{* | [SpaltenAusdruck [AS neuerName]]
[, ...] }
FROM Tabelle [ALIAS] [, ...]
[WHERE Bedingung]
[GROUP BY SpaltenListe]
[HAVING Bedingung]
[ORDER BY SpaltenListe]
```

- DISTINCT eliminiert Duplikate
- WHERE schränkt die Auswahl der Zeilen ein
- Minimale Syntax: SELECT * FROM Tabelle

Lesen mit SELECT - Beispiele

- Für die folgenden Beispiele verwenden wir die „Kassenbon“-Tabelle in 1NF (siehe Datenbankserver des Instituts)
- Diese Tabelle ist mit den Beispieldaten aus dem Einführungsbeispiel gefüllt
- Das Erstellen aus einem ER-Diagramm aus einer Datenbank nennt man auch **„Reverse Engineering“**
- Rechts ist das Ergebnis des Reverse Engineering der Tabelle des DBMS
- MWB: Database → Reverse Engineer ...



Kassenbon1NF	
🔑	SchNachname VARCHAR(45)
🔑	SchVorname VARCHAR(45)
🔑	Artikel VARCHAR(45)
🔑	LadenName VARCHAR(45)
🔑	Datum DATE
🔑	Zeit TIME
◇	Klasse CHAR(4)
◇	Geschlecht ENUM('m','w')
◇	Sparte VARCHAR(45)
◇	Preis DECIMAL(7,2)
◇	Zahlung ENUM('Bar','Karte')
◇	MwSt CHAR(6)
◇	PLZ CHAR(4)
◇	Ort VARCHAR(45)
◇	Teilort VARCHAR(45)
◇	Strasse VARCHAR(45)
◇	Telefonnummer VARCHAR(45)
◇	Land CHAR(2)
◇	Kommentar VARCHAR(200)
Indexes ▶	

Lesen mit SELECT - Beispiele

- Zeige alle Angaben über alle Einkäufe:
- **SELECT** SchVorname, SchNachname, Klasse,
Geschlecht, Sparte, Artikel, Preis,
Zahlung, MwSt, LadName, PLZ, Ort,
Ortsteil, Strasse, Land, Datum, Zeit, Kommentar
FROM Kassenbon1NF;
- „*“ (Stern, engl. asterisk) dient als Abkürzung für ‘alle Spalten’:
- **SELECT** *
FROM Kassenbon1NF;

Lesen mit SELECT - Beispiele

- Liste die Umsätze der Einkäufe, wobei nur Nummer, Sparte und Preis dargestellt werden

- Projektion**

- SELECT** Sparte, Preis
FROM Kassenbon1NF;

- Einschränken der Zeilen mit
LIMIT <Anzahl>:

- SELECT** Sparte, Preis
FROM Kassenbon1NF
LIMIT 10;

- Oder
LIMIT <Start, Anzahl>:

- LIMIT** 30,10;

Sparte	Preis
Unterhaltung	15.00
Kleider	5.25
Kosmetik	8.90
Lebensmittel	10.20
Unterhaltung	32.00
Kosmetik	9.80
Lebensmittel	1.05
Lebensmittel	1.45
Papeterie	6.90
Übriges	22.90
Kleider	250.90
Kleider	98.00
Transport	349.00
Kosmetik	3.60
Lebensmittel	4.80
Papeterie	63.75
Lebensmittel	2.00
Sport	10.00
Unterhaltung	16.00
Lebensmittel	1.75

Lesen mit SELECT und DISTINCT

- DISTINCT zeigt nur unikale Zeilen (Duplikate werden entfernt)
- Beispiel: Zeige die MwSt-IDs der Läden
- **SELECT** MwSt
FROM Kassenbon1NF;

MwSt
380507
231125
231125
231125
231704
216009
231074
227937
319644
231102

- Eliminierung von Duplikaten mit DISTINCT
- **SELECT DISTINCT** MwSt
FROM Kassenbon1NF;

MwSt
380507
231125
231704
216009
231074
227937
319644
231102
50220
0

Lesen mit SELECT - Berechnungen

- Berechnen von Spalten
- Liste die Nettopreise der Einkäufe zusammen mit Einkaufsnummer und Artikeln
- **SELECT** Artikel, Preis/1.2
FROM Kassenbon1NF;

Artikel	Preis/1.2
Kino	12.500000
Kniesocken	4.375000
Gemey-Great-Lash	7.416667
Coca-Cola 6x	8.500000
Kino	26.666667
Schmuckarmbänder	8.166667
Joghurt	0.875000
Honigmilch	1.208333
Papeterie	5.750000
CC Fl 3L	19.083333
Jacke	209.083333
T-shirt	81.666667

Name durch DBMS vergeben

Lesen mit SELECT und AS

- Berechnen von Spalten
- Umbenennen der Spalte durch AS-Klausel
- **SELECT** Artikel, ROUND(Preis/1.2,2) **AS** Nettopreis
FROM Kassenbon1NF;
- ROUND rundet den Wert auf zwei Nachkommastellen

Artikel	Nettopreis
Kino	12.50
Kniesocken	4.38
Gemey-Great-Lash	7.42
Coca-Cola 6x	8.50
Kino	26.67
Schmuckarmbänder	8.17
Joghurt	0.88
Honigmilch	1.21
Papeterie	5.75
CC Fl 3L	19.08
Jacke	209.08
T-shirt	81.67

Eigener Name

Lesen mit SELECT und WHERE

- WHERE-Klausel zum „Filtern“ von Zeilen
- Hier mit Vergleichsbedingung:
Zeige alle Einkäufe, die mehr als 100 SFr kosteten
- **SELECT** *
FROM Kassenbon1NF
WHERE Preis > 100;



SchVorname	SchNachname	Klasse	Geschlecht	Sparte	Artikel	Preis	Zahlung	MwSt	LadName
Laura	Aeberli	2e00	w	Kleider	Jacke	250.90	bar	-1	Ochsner Schuh
Judith	Affolter	2d99	w	Transport	Velozubehör	349.00	bar	121734	Cachet
Judith	Affolter	2d99	w	Kleider	Strickpullover	259.00	Karte	232987	Musik Hug
Selina	Alioth	2c01	w	Lebensmittel	Vitello Donato	229.00	bar	231074	Coop
Selina	Alioth	2c01	w	Unterhaltung	Bon Jovi-Konzert	140.00	bar	231102	Migros
Lydia	Allenspach	2f00	w	Unterhaltung	Polaroid	239.40	bar	231102	Migros
Chantal	Amberg	2e00	w	Übriges	Möbel	270.00	bar	231125	Interio
Chantal	Amberg	2e00	w	Kleider	Boots	129.00	Karte	231102	Migros
Oliver	Angehrn	2b99	m	Kleider	Grosslederwaren	239.00	Karte	231124	Jelmoli

Lesen mit SELECT und WHERE

- WHERE-Klausel zum „Filtern“ von Zeilen
- Hier mit zusammengesetzter Vergleichsbedingung:
Zeige alle Kleider-Einkäufe, die mehr als 100 SFr kosteten
- **SELECT** *
FROM Kassenbon1NF
WHERE Preis > 100 **AND** Sparte = 'Kleider';

SchVorname	SchNachname	Klasse	Geschlecht	Sparte	Artikel	Preis	Zahlung	MwSt	LadName
Laura	Aeberli	2e00	w	Kleider	Jacke	250.90	bar	-1	Ochsner Schuhe
Judith	Affolter	2d99	w	Kleider	Strickpullover	259.00	Karte	232987	Musik Hug
Chantal	Amberg	2e00	w	Kleider	Boots	129.00	Karte	231102	Migros
Oliver	Angehrn	2b99	m	Kleider	Grosslederwaren	239.00	Karte	231124	Jelmoli
Oliver	Angehrn	2b99	m	Kleider	Damenschuhe	340.00	bar	231102	Migros
Nora	Biedermann	2a99	w	Kleider	Schuhe	129.00	bar	-1	Kino Academy
Cliff	Bruckmann	2a99	m	Kleider	Pullover	129.00	bar	314400	Stadion Schluefweg
Jonas	Christen	2d00	m	Kleider	Schuhe	129.90	bar	231102	Migros
Jessica	Clavadetscher	2b01	w	Kleider	Jacke	129.90	bar	-1	Kino Academy
Michael	Eitle	2a01	m	Kleider	Hosen	219.00	bar	116247	Manor
Simon	Fleischhacker	2d99	m	Kleider	Jacke	119.00	bar	24139	Agip

Lesen mit SELECT, WHERE, BETWEEN

- WHERE-Klausel mit Bereichsbedingung **BETWEEN**
- Zeige alle Einkäufe, die zwischen 100 SFr und 250 SFr kosteten
- **SELECT** *
FROM Kassenbon1NF
WHERE Preis >= 100 **AND** Preis <= 250;
- **SELECT** *
FROM Kassenbon1NF
WHERE Preis **BETWEEN** 100 **AND** 250;
- BETWEEN ist inklusive, d.h. die genannten Werte gehören zum Wertebereich dazu
- NOT BETWEEN auch möglich (alles außer ...)

Lesen mit SELECT, WHERE, IN

- WHERE-Klausel mit Mengenzugehörigkeit **IN**
- Zeige alle Schüler(innen) der Klassen 2d00 und 2e00
- **SELECT DISTINCT** SchVorname, SchNachname, Klasse, Geschlecht
FROM Kassenbon1NF
WHERE Klasse **IN** ('2d00', '2e00');




SchVorname	SchNachname	Klasse	Geschlecht
Laura	Aeberli	2e00	w
Chantal	Amberg	2e00	w
Viviane	Aubert	2d00	w
Angela	Aumiller	2e00	w
Laura	Breitenmoser	2e00	w
Frank	Brunner	2d00	m
Jonas	Christen	2d00	m
Andreas	Dietlicher	2e00	m
Nadia	Elisa	2d00	w
Raphael	Ferrari	2e00	m

Lesen mit SELECT, WHERE, IN

- Negation NOT IN ist auch möglich
- Alternative zu IN:
- **SELECT DISTINCT** SchVorname, SchNachname, Klasse,
Geschlecht
FROM kassenbon1NF
WHERE Klasse = '2d00' **OR** Klasse = '2e00';
- IN effizienter bei Vergleich mit vielen Werten
- Auch für Mengenberechnung
(Unterabfragen, nächstes Kapitel)

Lesen mit SELECT und Pattern Matching

- WHERE-Klausel mit LIKE zum Vergleich von Zeichenketten
- Finde alle Migros-Läden (Migros, migros, Migrosmarkt,...)
- **SELECT DISTINCT** LadName, PLZ, Ort, Ortsteil, Strasse, Land
FROM Kassenbon1NF
WHERE LadName **LIKE** '%igros%';



LadName	PLZ	Ort	Ortsteil	Strasse	Land
Migros	8132	Egg		Im Dorf	CH
Migros	8301	Glattzentrum	Wallisel...		CH
Migros	8706	Stäfa			CH
Hobbvcenter /Migrosmarkt.		Zürich		Brunaupark	CH
Migros		Zürich			CH
Migros	8640	Rapperswil			CH
Migros		Schützenmatt			CH
Migros		Greifensee			CH
Migros		Wiedikon			CH
Migros	8053	Zürich			CH


Lesen mit SELECT und Pattern Matching

SQL kennt zwei spezielle Symbole für Pattern Matching:

- %: Folge von 0 oder mehr beliebigen Zeichen;
- _ (Unterstrich): beliebiges einzelnes Zeichen
- **LIKE** ' %igros% ' bedeutet eine beliebig lange Zeichenkette, in der „igros“ vorkommt
- **LIKE** ' %z ' bedeutet alle Strings, die mit „z“ enden
- **LIKE** ' A__ ' bedeutet alle Strings, die 3 Zeichen haben und mit „A“ beginnen

Lesen mit SELECT und NULL-Bedingung

- Abfrage auf NULL-Wert durch Schlüsselwort `IS NULL`
- Negation (`IS NOT NULL`) testet auf das Vorhandensein von Daten
- Zeige Alle Einkäufe, für die ein Kommentar existiert
- **SELECT** *
FROM Kassenbon1NF
WHERE Kommentar **IS NOT NULL** AND
Kommentar **NOT IN** ('?', '-', 'N/A');



SchVorname	SchNachname	Klasse	Geschlecht	Sparte	Artikel	Preis	Zahlung	MwSt	LadName	PLZ	Ort	Ortsteil	Strasse	Land	Datum	Zeit	Kommentar
Laura	Aeberli	2e00	w	Kleider	Kniesocken	5.25	bar	453571	Bijoux One	8031	Zürich		Quellenstrass	CH		08:40:00	Hr.Hiltbrunner
Laura	Aeberli	2e00	w	Unterhaltung	Kino	32.00	bar	501543	Aquarium Pfi						2000-08-30	16:27:00	Frau Vejseli
Laura	Aeberli	2e00	w	Lebensmittel	Honigmilch	1.45	bar	-1	Kino IMAX		Luzern			CH	1999-09-05	16:40:00	Migros-Team
Laura	Aeberli	2e00	w	Übriges	CC FI 3L	22.90	bar	231102	Migros	8032	Zürich		Kreuzplatz	CH	2001-03-22	11:43:00	Heidi
Judith	Affolter	2d99	w	Lebensmittel	Makrelen	4.80	bar	231102	Migros	7130	Ilanz			CH	1999-09-05	13:47:00	E.Hagen

Änderungen mit UPDATE und WHERE

- Zum Ändern UPDATE-Statement verwenden
- Syntax

```
UPDATE <Tabellenname>  
SET <SpaltenName1> = Wert1  
[, < SpaltenName2> = Wert2 ...]  
[ WHERE Bedingung ];
```

- in SET-Klausel Angabe von (einer oder mehreren) Spalte(n), die geändert werden soll(en)
- WHERE-Klausel ist optional
 - ohne bewirkt Änderung der Spalten in **allen(!)** Zeilen
 - mit nur Änderung in Zeilen, die Bedingung erfüllen
 - neue Werte müssen typkompatibel mit der Spalte sein

Änderungen mit UPDATE und WHERE

- Änderungen können auch mit Spaltennamen berechnet werden
- Beispiele:
 - **UPDATE** Zaehler
 SET Anzahl = Anzahl + 1
 WHERE Id=4711;
 - **UPDATE** Laden **SET** Ort = 'Freiburg im Breisgau'
 WHERE Ort **LIKE** 'Freiburg%'
 AND Land = 'D';
 - **UPDATE** Laden **SET** PLZ = '79098',
 Strasse = 'Kaiser-Joseph-Strasse 195'
 WHERE LadName = 'Galeria Kaufhof'
 AND Ort **LIKE** 'Freiburg%';
- Vorsicht bei Updates mit der WHERE-Klausel – am besten vorher mit einem SELECT-Statement überprüfen

Löschen mit DELETE und WHERE

- Zum Löschen DELETE-Statement verwenden
- Syntax

```
DELETE FROM <Tabellenname>  
[WHERE Bedingung];
```

- WHERE-Klausel ist optional
 - ohne bewirkt Löschung **aller(!)** Zeilen
- Beispiel:

```
DELETE FROM SchuelerIn  
WHERE SchVorname = 'Oster'  
AND SchNachname = 'Hase';
```
- Alternative zum Leeren einer Tabelle (→ Löschen aller Zeilen)

```
TRUNCATE <Tabellenname>;
```

Sortieren der Ergebnisse

- In der Regel bekommen Sie die Ergebnisse in der Reihenfolge, wie diese gespeichert wurden
- **Machen Sie aber keine Annahmen über die Reihenfolge** der zurückgelieferten Zeilen
- Andere Reihenfolge möglich, z.B. bei Datenbankcluster
- Sortieren mit ORDER BY
- **SELECT** Sparte, Artikel
FROM Kassenbon1NF
ORDER BY Sparte;



Sparte	Artikel
k.A.	k.A.
Kleider	Schuhe
Kleider	Hose
Kleider	Schuhe
Kleider	L.O.G.G. Sport
Kleider	Smith Brille
Kleider	strumpfhose
Kleider	jersey modisch
Kleider	T-shirt
Kleider	Strumpfhosen
Kleider	Sommersport
Kleider	Sweatshirt
Kleider	Nachtwäsche
Kleider	Jupe Trend

Sortieren der Ergebnisse

- Ohne zweites Sortierkriterium erscheinen Artikel in beliebiger Reihenfolge
- Zur Sortierung nach Artikel: Angabe einer zweiten Reihenfolge

▪ **SELECT** Sparte, Artikel
FROM Kassenbon1NF
ORDER BY Sparte, Artikel **DESC;**

aufsteigend (default)
oder ASC

absteigend

Sparte	Artikel
k.A.	k.A.
Kleider	ZIP-Hose , schwarz
Kleider	Wollkappe
Kleider	Wolle
Kleider	Wolle
Kleider	WE Men
Kleider	Waschhandschuhe
Kleider	versch.Kleider
Kleider	Unterwäsche
Kleider	Unterwäsche
Kleider	Unterwäsche
Kleider	Unterwäsche
Kleider	Unterwäsche

- ASC : Ascending
- DESC : Descending

6.1 Einführung in SQL

6.2 Datendefinition (DDL)

6.3 Datenmodifikation (DML)

6.4 Aggregatfunktionen

- Im `SELECT`-Statement können auch gleichzeitig Werte/ Ergebnisse **über mehrere Zeilen berechnet** werden
- Zusammenfassen → Aggregation
- Im ISO-Standard sind fünf Aggregatfunktionen definiert
 - `COUNT` – Anzahl der Werte der angegebenen Spalte
 - `SUM` – Summe der Werte der angegebenen Spalte
 - `AVG` – Durchschnitt der Werte der Spalte
 - `MIN` – kleinster Wert der angegebenen Spalte
 - `MAX` – größter Wert der angegebenen Spalte

- COUNT, MIN, MAX: numerische und nicht-numerische Spalten
 - Bei alphanumerischen Spalten: kleinster Wert nach Sortierung
- SUM, AVG: nur numerische Spalten
- NULL-Werte werden bei COUNT () nicht berücksichtigt
- NULL wird nur bei COUNT (*) berücksichtigt
 - COUNT (*) zählt alle Zeilen einer Tabelle (inkl. NULL-Zellen)
- **Duplikate:** Elimination durch DISTINCT
 - MIN/MAX: keine Auswirkung
 - SUM/AVG: Auswirkung möglich

Aggregatfunktionen: COUNT

- Zeige die Anzahl der Einkäufe, die über 100 SFr kosten
- **SELECT** COUNT(*) **AS** Expensive
FROM Kassenbon1NF
WHERE Preis > 100;

Expensive
133

Aggregatfunktionen: COUNT DISTINCT

- Aus welchen (verschiedenen) Sparten kauften Mitglieder der Klasse 2e00 Artikel ein?

- **SELECT COUNT**(Sparte) **AS** count
FROM Kassenbon1NF
WHERE Klasse = '2e00';

count
412

- Zählt alle Zeilen mit Duplikaten (wie COUNT (*))
- **SELECT COUNT**(**DISTINCT** Sparte) **AS** countDis
FROM Kassenbon1NF
WHERE Klasse = '2e00';

countDis
8

Aggregatfunktionen: SUM

- Finde die Anzahl der SchülerInnen der Klasse 2e00 sowie die Summe ihrer Umsätze
- SELECT COUNT(DISTINCT** SchNachname, SchVorname)
 AS Anzahl,
 SUM(Preis) **AS** Umsatz
FROM kassenbon1NF
WHERE Klasse = '2e00';

Anzahl	Umsatz
25	7227.88

Aggregatfunktionen: SUM

- Finde den kleinsten, den größten und den durchschnittlichen Umsatz der Einkäufe
- SELECT** MIN(Preis) **AS** min,
MAX(Preis) **AS** max,
AVG(Preis) **AS** avg
FROM Kassenbon1NF;

min	max	avg
-5.50	999.95	18.978552

- Verwendung von Aggregatfunktionen nur in
 - `SELECT` Liste und/oder
 - `HAVING` Klausel (später)
- Mischung von **Aggregatfunktion(en)** und **Attributnamen** in der `SELECT`-Liste erfordert `GROUP BY`-Klausel, um nach allen einzeln referenzierten Attributen zu gruppieren

So ist z.B.

- **`SELECT`** SchNachname, **`COUNT`**(Preis)
`FROM` kassenbon1NF;

unsinnig. Es wird der Preis/Umsatz aller Schüler addiert

Aggregatfunktionen mit Gruppierung

- `GROUP BY` zur Berechnung von Teilergebnissen von Aggregaten
- Ablauf
 - Auswahl der Daten im `SELECT` Statement
 - Filterung mit `WHERE`
 - Gruppierung der Daten
 - Anwendung der Aggregatfunktion auf die Gruppen
- Jedes Element der `SELECT` Liste muss innerhalb einer Gruppe einen eindeutigen Wert besitzen
- die `SELECT` Klausel darf nur beinhalten
 - Spaltennamen, Konstanten
 - Aggregatfunktionen
 - Ausdrücke, die aus den vorgenannten bestehen

Aggregatfunktionen mit Gruppierung

- `GROUP BY` zur Berechnung von Teilergebnissen von Aggregaten
- `SELECT` und `GROUP BY` hängen stark zusammen
- Spaltennamen aus `GROUP BY` Klausel sollten in der `SELECT`-Liste vorkommen
- `WHERE` mit `GROUP BY`
 - erst `WHERE` auswerten
 - bilde Gruppen im Resultat
- `NULL`-Werte: für `GROUP BY` ein eigener Wert (nach ISO)
d.h. alle `NULL`-Werte landen in einer Gruppe

Aggregatfunktionen mit Gruppierung

- GROUP BY zur Berechnung von Teilergebnissen von Aggregaten
- Finde die Anzahl der SchülerInnen der Klassen sowie die Summen deren Umsätze
- SELECT** Klasse, **COUNT**(**DISTINCT** SchNachname, SchVorname) **AS** Anzahl, **SUM**(Preis) **AS** Umsatz
FROM Kassenbon1NF
GROUP BY Klasse;

Klasse	Anzahl	Umsatz
2a01	22	9107.50
2a99	21	7247.75
2b01	17	3607.20
2b99	33	12129.49
2c01	25	7018.42
2c99	28	6890.09
2d00	12	4468.68
2d99	28	9806.40
2e00	25	7227.88
2e99	23	5486.17
2f00	22	7475.18
2f99	2	402.65

Aggregatfunktionen mit Gruppierung

- Sollen die Gruppen **nach** der Aggregatfunktion nochmals gefiltert werden, kann man `HAVING` verwenden (`WHERE` filtert ja davor)
- Spaltennamen der `HAVING` Klausel müssen in der `GROUP BY` Liste oder in einer Aggregatfunktion vorkommen
- **SELECT** Klasse, **COUNT**(**DISTINCT** SchNachname,
SchVorname) **AS** Anzahl,
SUM(Preis) **AS** Umsatz
FROM Kassenbon1NF
GROUP BY Klasse
HAVING Anzahl >= 25;

Klasse	Anzahl	Umsatz
2b99	33	12129.49
2c01	25	7018.42
2c99	28	6890.09
2d99	28	9806.40
2e00	25	7227.88

Kombination von DDL und DML möglich

Verschiedene Kombinationsmöglichkeiten

- Anlegen von (temporären) Tabellen

```
CREATE (TEMPORARY) TABLE <Tabellenname>  
AS ( SELECT ... )
```

- Kopiert Struktur und Daten!
- Kopiert nicht Primär- und Fremdschlüssel(!) oder Indizes
- Einfügen von Daten

```
INSERT INTO <Tabellenname> (SELECT ...)
```

- Ändern von Daten

```
UPDATE <Tabellenname> SET Spalte = (  
SELECT Quellspalte ...)
```

Aufgaben

Bitte bearbeiten Sie jetzt die Aufgaben in Moodle zum Kapitel 6.

- **Teil B**