Grundlagen der Wirtschaftsinformatik

Hochschule Bochum | Fachbereich Wirtschaft | Marcel David M.Sc.

(Diese Unterlage ist ausschließlich für Vorlesungszwecke bestimmt)

VERBUNDSTUDIUM
Berufsbegleitend zum Hochschulabschluss

Hochschule Bochum
Bochum University
of Applied Sciences

Datenmanipulation

- Eine Datenmanipulation ist eine beliebige Operation auf eine Datenbank.“ (ZEHNDER, S.153)
- Diese Operation kann schreibend oder lesend sein.
- “Eine Abfrage (query) erlaubt, einen bestimmten Ausschnitt einer Datenbank abzugrenzen und aufzufinden sowie dessen Inhalt in geeigneter Form darzustellen.“ (ZEHNDER, S.153)
- “Eine Mutation oder Nachführung (update) erlaubt, einen bestimmten Ausschnitt einer Datenbank abzugrenzen und aufzufinden sowie dessen Inhalt konsistenzerhaltend zu verändern.“ (ZEHNDER, S.153)

SQL - Structured Query Language

- SQL wurde Mitte der 1970er Jahre entwickelt und 1992 als ISO-Standard (SQL2) definiert; erste SQL-DB: Oracle (1979)
  SQL ist die vorherrschende Sprache zur Beschreibung und Manipulation von Relationen
- SQL-Implementierungen existieren für
  - Dialoganwendung
  - Einbettung in Programme, die in anderen Programmiersprachen erstellt sind (= embedded SQL, ESQL)
- SQL ist sowohl eine Datenmanipulationssprache (DML) als auch eine Datendefinitionssprache (DDL)

Grundlegende Operationen

- SQL kennt vier grundlegende Operationen:

  - SELECT
    wählt Daten aus der Menge aus
  - INSERT
    fügt ein oder mehrere Tupel hinzu
  - UPDATE
    verändert Daten
  - DELETE
    löscht ein oder mehrere Tupel

Grundlegende Operationen – Select
Allgemeine Syntax

- Abfrage an eine oder mehrere Tabellen zur Auswahl von Daten

[]: optional

|: Alternativen

{}: zwingend

- Syntax:

```
SELECT [DISTINCT | ALL] {* | columnExpression
[AS newName]] [,...]}
FROM tableName [alias] [,...]
[WHERE condition]
[GROUP BY columnList]
[HAVING condition]
[ORDER BY columnList];
```

UPPER CASE: reservierte Wörter

lower case: user-definierte Wörter

Grundlegende Operationen – Select
Allgemeine Syntax

- Abfrage an eine oder mehrere Tabellen zur Auswahl von Daten

Angabe der Tabelle

Angabe der Spalten

- Syntax:

```
SELECT [DISTINCT | ALL] {* | columnExpression
[AS newName]] [,...]}
FROM tableName [alias] [,...]
[WHERE condition]
[GROUP BY columnList]
[HAVING condition]
[ORDER BY columnList];
```

Auswahlbedingung

Gruppenbildung

Selektionsbedingung für Gruppen

Sortierung des Ergebnisses

Grundlegende Operationen – Select Feldauswahl

- Beispiel:

Anzeige einer gesamten Tabelle / mit Sortierung nach Preis

```
SELECT * FROM Produkt;
SELECT * FROM Produkt ORDER BY Stückpreis;
```

Beispiel:

Anzeige von 2 ausgewählten Feldern

```
SELECT ProduktName, Stückpreis FROM Produkt
```

In manchen SQL-Dialekten ist auch bei der Selektion aus einer Tabelle deren Angabe vor dem Feldnamen üblich:

```
SELECT Produkt .Produkt, Produkt .Stückpreis
FROM Produkt ;
```

Grundlegende Operationen – Select -Where Klausel

- Beispiel:
  Anzeige einzelner Tupel, die eine Bedingung erfüllen

```
SELECT * FROM Ort WHERE PLZ = 44444;
```

- Beispiel:
  Anzeige einzelner Tupel, die mehrere Bedingungen erfüllen:

```
SELECT * FROM Ort WHERE PLZ = 44444 OR PLZ = 12345;
```

- Frage: Was würde in folgenden Fällen passieren:

```
SELECT * FROM Ort WHERE PLZ = 44445 OR PLZ = 12345;
SELECT * FROM Ort WHERE PLZ = 44444 AND PLZ = 12345;
```

Grundlegende Operationen – Select -Where Klausel

- Verknüpfungen in der WHERE-Klausel:
- Der logische Ausdruck in der WHERE-Klausel muss wahr sein, damit ein Tupel selektiert wird.
- Mehrere Operationen können durch Vergleichsoperatoren und logische Operatoren zum komplexen Ausdrücken kombiniert werden.
  - Vergleichsoperatoren: =, <>, <, > ,<=, >=
  - ggf. auch ! für <> (= ungleich)

  Logische Operatoren: AND, OR, NOT
  Suchbedingungen: BETWEEN, IN
- Auswertungsreihenfolge:

  (1) Klammern zuerst, von innen nach außen
  (2) Auswertungsreihenfolge von links nach rechts
  (3) NOT vor AND vor OR

Grundlegende Operationen – Select -Where Klausel – Besondere Abfragen

- Suche nach NULL-Strings:

  SELECT * FROM tabelle WHERE feld IS [NOT] NULL;

- Suche nach Zeichenketten mittels [NOT] LIKE:

  SELECT * FROM tabelle WHERE feld LIKE ‘string’;

  Hier können auch Wildcards eingesetzt werden:

  % steht für eine Zeichenkette beliebiger Länge

  _ steht für genau ein Zeichen

Beispiel:

- **SELECT * FROM Kunde WHERE** KundeName **LIKE** 'M__er'

Grundlegende Operationen – Select -Where Klausel – Besondere Abfragen

- Besondere Abfragen in der WHERE-Klausel:
- Suche mit BETWEEN:

```
SELECT * FROM tabelle WHERE feld BETWEEN
wert1 AND wert2;
```

- Abfrage auf Mengenzugehörigkeit mit IN:

```
SELECT * FROM tabelle WHERE feld IN
        (wert1, wert2);
```

Grundlegende Operationen – Select All / Distinct

- Beispiel:

  Anzeige aller Werte, die eine Bedingung erfüllen

AuftrNr.	Datum	KD-Nr.
1001	12.12.2015	1
1002	14.12.2015	2
1003	14.12.2015	3
1004	14.12.2016	2
1005	16.12.2015	4

SELECT ALL Datum **FROM** Auftrag

Beispiel:

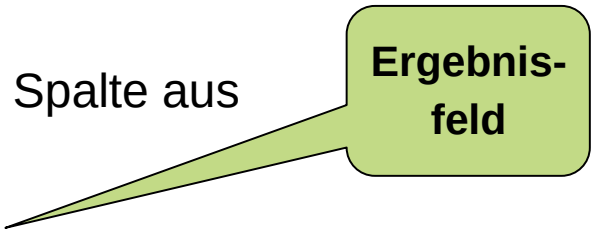
Anzeige nur verschiedener Werte, die eine Bedingung erfüllen:

SELECT DISTINCT Datum **FROM** Auftrag

Grundlegende Operationen – Select Aggregation

- In SQL können einfache mathematische Operationen direkt ausgeführt werden:

- COUNT	gibt die Anzahl
- SUM	gibt die Summe
- AVG	gibt den Mittelwert
- MIN	gibt den kleinsten
- MAX	gibt den größten
-	... der Werte einer Spalte aus



- Beispiel:

```
SELECT MIN(Stückpreis) AS kleinster_preis FROM Produkt
```

Grundlegende Operationen – Select Group by und Having

- Die GROUP BY-Klausel ermöglicht die Gruppierung der Ergebnisse einer SELECT-Abfrage:

```
SELECT Datum, COUNT(AuftrNr) AS AnzAufträge
FROM Auftrag
GROUP BY Datum
ORDER BY Datum
```

- Die Auswahl aus Gruppierungen kann mittels der HAVING-Klausel eingeschränkt werden:

```
SELECT Datum, COUNT(AuftrNr) AS AnzAufträge
FROM Auftrag
GROUP BY datum
HAVING COUNT(AuftrNr) > 2
ORDER BY datum
```

Grundlegende Operationen – Select Group by und Having

- Die GROUP BY-Klausel ermöglicht die Gruppierung der Ergebnisse einer SELECT-Abfrage:

Ausgewählte Spalten müssen entweder mittels einer Aggregationsfunktion zusammengefasst werden …

- ```
  SELECT Datum, COUNT(AuftrNr) AS AnzAufträge
  FROM Auftrag
  GROUP BY Datum
  ORDER BY Datum
  ```

… oder Teil des „group by“ Statements sein
  ```

Unterabfragen

- Ermöglichen das Heranziehen des Ergebnisses einer Abfrage in einer neuen Abfrage zu benutzen.
- Kein neues Schlüsselwort nötig
- Syntax:

  SELECT columnList FROM tableName1 WHERE column = (SELECT …)

- **SELECT** AuftrNr **FROM** Auftrag
  WHERE AuftrNr =
  (**SELECT** MAX(Datum) **FROM** Auftrag)

AuftrNr.	Datum	KD-Nr.
1001	12.12.2015	1
1002	14.12.2015	2
1003	14.12.2015	3
1004	14.12.2016	2
1005	16.12.2015	4

Abfragen über mehrere Tabellen

- Mit SQL können Abfragen über Daten, die in mehreren Tabellen abgespeichert sind, durchgeführt werden.
- Eine solche (über Fremdschlüssel durchgeführte) Verknüpfung wird als JOIN bezeichnet.
- Die Verbindung erfolgt durch übereinstimmende Schlüsselwerte in den verknüpften Feldern (Primärschlüssel / Fremdschlüssel).

Abfragen über mehrere Tabellen

Auftrag

AuftrNr.	Datum	KD-Nr.
1001	12.12.2015	1
1002	14.12.2015	2
1003	14.12.2015	3
1004	14.12.2016	2
1005	16.12.2015	4

Kunde

KD-Nr.	Kunde	Straße	PLZ
1	Müller	Hauptstr. 3	98756
2	Meyer	Dorfplatz 7	12345
3	Meyer	Am Deich 9	44444
4	Schulz	Kirchstr. 37	87655

Für die Beantwortung der Frage „Wie heißt der Kunde, der den Auftrag 1001 erteilt hat?“ müssen die Daten aus 2 Tabellen kombiniert werden

Rückblick Unterabfragen

- Ermöglichen das Heranziehen des Ergebnisses einer Abfrage in einer neuen Abfrage zu benutzen.
- Kein neues Schlüsselwort nötig
- Syntax:

  SELECT columnList FROM tableName1 WHERE column = (SELECT …)

- **SELECT** AuftrNr **FROM** Auftrag
  WHERE AuftrNr =
  (**SELECT** MAX(Datum) **FROM** Auftrag)

AuftrNr.	Datum	KD-Nr.
1001	12.12.2015	1
1002	14.12.2015	2
1003	14.12.2015	3
1004	14.12.2016	2
1005	16.12.2015	4

Abfragen über mehrere Tabellen

- Verknüpfung von zwei Tabellen:
- Syntax:

```
SELECT columnList FROM tableName1, tableName2
        WHERE tableName1.fk = tableName2.pk;
```

- Beispiel:

Die Reihenfolge der Tabellen ist egal.

```
SELECT Auftrag.AuftrNr, Kunde.KundeName FROM Auftrag, Kunde WHERE
Auftrag.KdNr = Kunde.KdNr AND Auftrag.AuftrNr = 1001
```

Abfragen über mehrere Tabellen

- Alternativ zur WHERE-Klausel kann auch die FROM .. INNER JOIN-Klausel gewählt werden:

- Syntax:

  SELECT columnList FROM tableName1 INNER JOIN tableName2 ON tableName1.fk = tableName2.pk;

- Beispiel:

```
SELECT Auftrag.AuftrNr, Kunde.KundeName
FROM Auftrag INNER JOIN Kunde
ON Auftrag.KdNr = Kunde.KdNr
WHERE Auftrag.AuftrNr = 1001
```

Abfragen über mehrere Tabellen

Auftrag

AuftrNr.	Datum	KD-Nr.
1001	12.12.2015	1
1002	14.12.2015	2
1003	14.12.2015	3
1004	14.12.2016	2
1005	16.12.2015	4

Kunde

KD-Nr.	Kunde	Straße	PLZ
1	Müller	Hauptstr. 3	98756
2	Meyer	Dorfplatz 7	12345
3	Meyer	Am Deich 9	44444
4	Schulz	Kirchstr. 37	87655

Ort

PLZ	Ort
98756	Adorf
12345	Bestadt
44444	Deburg
87655	Effkirch

Für die Beantwortung der Frage „Wo wohnt der Kunde, der den Auftrag 1001 erteilt hat?“ müssen die Daten aus 3 Tabellen kombiniert werden

Abfragen über mehrere Tabellen

- Auch bei der Verknüpfung von mehr als 2 Tabellen kann die FROM .. INNER JOIN-Klausel gewählt werden:

- Syntax:

```
SELECT columnList FROM
        (tableName3 INNER JOIN
                tableName2
                ON
                tableName2.fk = tableName3.pk)
INNER JOIN tableName1 ON tableName1.fk =  tableName2.pk;
```

- Beispiel:

```
SELECT Auftrag.AuftrNr, Kunde.KundeName, Ort.OrtName
FROM Ort INNER JOIN Kunde ON Kunde.PLZ = Ort.PLZ
INNER JOIN Auftrag ON Auftrag.KdNr = Kunde.KdNr
WHERE Auftrag.AuftrNr = 1001
```

Grundlegende Operationen – Insert

- **INSERT** = Einfügen eines neuen Tupels
- **Syntax:** INSERT INTO tableName [(columnList)]
  VALUES (dataValueList);
- Beispiele (**SQL-Schlüsselwörter**, Tabellen, Attribute):

INSERT INTO Produkt (ProdNr, Stückpreis, ProduktName) **VALUES** (5011, 7.95, 'Vanille')

INSERT INTO Produkt **VALUES** (5011, 'Vanille', 7.95)

Grundlegende Operationen – Insert

... kann sowohl eine Tabelle als auch eine Sicht aus einer Abfrage sein

- **INSER** euen Tupels
- **Syntax:** INSERT INTO tableName [(columnList)]
  VALUES (dataValueList);

Reihenfolge muss nicht der Tabellendefinition entsprechen!

Primärschlüssel mit Autowert wird nicht! angegeben

- Beispiele (**SQL-Schlüsselwörter**, Tabellen, Attribute):

```
INSERT INTO Produkt (ProdNr, Stückpreis, ProduktName) VALUES
(5011, 7.95, 'Vanille')

INSERT INTO Produkt VALUES (5011, 'Vanille', 7.95)
```

Achtung: Literale! Zeichenketten werden in einfache oder doppelte Anführungsstriche gesetzt

Update-Operationen – Insert

- INSERT
- Nachdem die SELECT-Klausel erklärt wurde, können wir sie nun im Kontext des INSERT-Befehls anwenden, um Spalten aus einer oder mehreren Tabellen in eine andere Tabelle zu kopieren.
- Beispiel:

```
INSERT INTO Adressen (KundeName, Straße, PLZ, OrtName) SELECT
Kunde.KundeName, Kunde.Straße, Kunde.PLZ, Ort.OrtName FROM Kunde, Ort
WHERE Kunde.PLZ = Ort.PLZ
```

Update-Operationen – Update

- UPDATE
- Änderung der Attributwerte in einer Tabelle. Werden mittels einer WHERE-Klausel einzelne Datensätze selektiert, werden nur diese geändert, sonst alle Datensätze einer Tabelle.
- Syntax:

```
UPDATE tableName SET columnName1 = dataValue1
                    [, columnName2 = dataValue2 ... ]
                    [WHERE searchCondition];
```

- Beispiele:

```
UPDATE Produkt SET Stückpreis = Stückpreis * 1.05

UPDATE Produkt SET Stückpreis
```

Update-Operationen – Delete

- DELETE
- Löschen von Datensätzen in einer Tabelle. Werden mittels einer WHERE-Klausel einzelne Datensätze selektiert, werden nur diese gelöscht, sonst alle Datensätze einer Tabelle. (ACHTUNG: Dabei würde nicht die Tabelle selbst gelöscht werden; ihre Strukturinformationen bleiben erhalten!)
- Syntax:

```
DELETE FROM tableName
[WHERE searchCondition];
```

- Beispiel:

```
DELETE FROM Auftrag WHERE datum < '2014-1-1'
```