MySQL

- SQL – Structured Query Language – Strukturierte Datenbankabfragesprache
- entwickelt 1973 von IBM (heute vor allem durch Oracle)
- heute Standardabfragesprache für DB
- MySQL eingeschränkte Version von SQL
  - Wird durch Oracle weiterentwickelt

1

Datentypen

- Datumstypen
  - DATE: 3 Byte; JJJJ-MM-TT
  - DATETIME: 8 Byte; JJJJ-MM-TT hh:mm:ss
  - TIME: 3 Byte; hh:mm:ss
  - TIMESTAMP: 4 Byte; JJJJMMTThhmmss
  - YEAR: 1 Byte; JJJJ
  - Die Zeitzone muss separat gesetzt werden:
    - SET time_zone='+3:00';
    - das bewirkt beim TIMESTAMP eine generelle Änderung (rückwirkend auf alle vorhandenen Datensätze!).
- Komplexe Datentypen
  - ENUM: bis zu 2 Byte, max. 65.535 Elemente; Zur Einschränkungen von Insert und Update
  - SET: bis zu 8 Byte, max. 64 String-Objekte; ähnlich ENUM

2

- Numerische Datentypen
  - Ganzzahlen
    - BIGINT: 8 Byte; -/+ 9.223.372.036.854.775.808
    - INT/INTEGER: 4 Byte: -/+ 2.147.483.648 (ohne Vorzeichen 4.294.967.295)
    - MEDIUMINT: 3 Byte; -8.388.608 bis +8.388.607 (ohne Vorzeichen 16.777.215)
    - SMALLINT: 2 Byte; -32768 bis +32767 (ohne Vorzeichen 65.535)
    - TINYINT: 1 Byte; -128 bis +127 (ohne Vorzeichen 255)
  - Fließkommazahlen
    - DEC/DECIMAL/NUMERIC: z.B. DECIMAL(9, 2) ohne Angabe evt. keine Kommastelllen.
    - decimal: wird seit MySQL 5.1 binär und nicht mehr als string gespeichert
    - DOUBLE/REAL: 8 Byte; bis zu ca. 300 Nachkommastellen
    - FLOAT: 4 Byte; bis zu ca. 38 Nachkommastellen

3

- String-Datentypen
  - CHAR | CHARACTER | NCHAR | NATIONAL CHAR: 0 bis 255 Zeichen
  - VARCHAR | CHARACTER VARYING | NATIONAL VARCHAR: 0 bis 255 Zeichen + 1 Byte
  - LONGTEXT: 0 bis 4.294.967.295 Zeichen
  - MEDIUMTEXT
  - TEXT: 0 bis 65.535 Zeichen
  - TINYTEXT: 0 bis 255 Zeichen + 1 Byte

4

Operatoren

- Vergleichs Operatoren
  - =, <, >, !=, <>
  - LIKE und NOT LIKE
    - wird für Zeichenketten verwendet, um einen Teilstring abzufragen (=sowie), das Prozentzeichen fungiert dabei als Platzhalter für beliebige Zeichen beliebiger Anzahl. Der Unterstrich ist Paltzhalter für ein Zeichen.
    - like '%rist%'
    - Like prüft seit ca. MySQL 5.x nicht mehr case-sensitve
  - BETWEEN
  - IS NULL
  - usw.
- Logische Operatoren
  - NOT !
  - OR ||
  - XOR (gibt 1 zurück, wenn ein Argument <> 0 und <> Null ist)
- Arithmetische Operatoren
  - +, -, *, /, %
  - Bitweise: |, ^, &

5

Funktionen - Auszug

- FLOOR: schneidet Nachkommastellen ab
- ROUND: rundet auf
- COUNT: Anzahl der DS zählen
- SUM: Summe über Spalten bilden
- AVG: Durchschnittswert berechnen
- CONCAT_WS(Zeichen, string, string)
- CONCAT(string, Zeichen, string, Zeichen …)
- CHAR: ASCII-Wert für einen Integer - select char(65); = A
- WEEKOFYEAR: Kalenderwoche zu bestimmten Datum

6

Datumsfunktionen – bezogen auf MariaDB

- Funktionen für Datum (date, time, datetime)
  - aktuelle Systemzeit
    - NOW(), CURRENT_TIMESTAMP, CURRENT_TIMESTAMP(),
    - LOCALTIME, LOCALTIME(), LOCALTIMESTAMP, LOCALTIMESTAMP()
    - DT TIMESTAMP arbeitet per Standard mit dem DEFAULT-Wert CURRENT_TIMESTAMP
      - Beispiel:
      - CREATE TABLE zeit (zeitpunkt TIMESTAMP DEFAULT CURRENT_TIMESTAMP) ENGINE = InnoDB;
    - einfacher geht es mit Datentyp datetime
      - create table zeit(datum datetime);
  - Zeitdifferenz
    - SELECT DATEDIFF('2007-12-01','2007-12-30'); Differenz -29
    - Addition und Subtraktion: date_add, date_sub

7

- Tag, Monat oder Jahr eines Datums ermitteln
  - SELECT EXTRACT(YEAR FROM '2017-07-02');
  - select dayofmonth('2017-10-25'); alternativ: day()
- Datumsformat ermitteln
  - select get_format(date, 'eur');
    - Ergebnis: %d.%m.%Y
    - Datumsformat: eur, usa, jis, iso, internal
- Letzter Tag eines Monats
  - select last_day('2017-12-12'); Ergebnis 2017-12-31
- Sonstige Datumsfunktionen
  - select quarter('2017-05-01'); Ergebnis 2
- Lokales Datumsformat setzen:
  - set lc_time_names ='de_DE‘; für Ausgabe bei date_format in Deutsch

8

- insert into testen values(null, '1904-10-3');
- select date_format(datum, '%a %d %b %Y') from testen;
- Sun 03 Oct 2004
- select date_format(datum, '%a %j %b %Y') from testen;
- Sun 277 Oct 2004
- **select date_format(datum, '%d-%m-%Y') from testen;**
- **03-10-2004**

9

DATE_FORMAT

Format	Ausgabe
%a	Wochentag: Sun, Mon ...
%b	Monat: Jan, Feb ...
%D	Tag: 1st, 2nd, 3rd ...
%d	Tag des Monats
%H	24-Stunden-Format (z.B. 01)
%h/%I	12-Stunden-Format (z.B. 09)
%i	Minuten
%j	Tag des Jahres
%k	24-Stunden-Format (z.B. 1)
%l	12-Stunden-Format (z.B. 9)
%M	Name des Monats
%m	Nummer des Monats (Januar = 1)

Zurück

10

Format	Ausgabe
%p	A.M oder P.M
%r	12-Stunden-Format (inkl. A.M./P.M.)
%S	Sekunden (z.B. 04)
%s	Sekunden (z.B. 4)
%T	vollständige Zeit (24-Stunden-Format)
%U	Woche des Jahres (Wochenbeginn Sonntag)
%W	Name des Wochentags
%w	Nummer des Wochentags (Sonntag = 0)
%Y	Jahr, vierstellig
%y	Jahr, zweistellig
%%	Prozentzeichen

Zurück

11

MySQL Workbench

- Dieses Tool wird in einer gratis Version zur Verfügung gestellt – u.a.ERM Designer
- Bei der Codegenerierung, werden am Anfang des Scripts folgende Zeilen automatisch eingefügt:
- `SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;`
  - verhindert doppelte Schlüsselüberprüfung – betrifft UNIQUE (wenn neben einem PK in der Tabelle ein alternativer Schlüssel mit UNIQUE gekennzeichnet ist)
  - sollte aus Performancegründen entfernt werden
- `SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;`
  - überprüft, ob die referentielle Integrität gewahrt ist (z.B. existiert ein PK der als FK verwendet wird)
  - wird InnoDB bei referentieller Integrität verwendet, kann diese Anweisung ebenfalls wegfallen
- `SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='TRADITIONAL';`
  - es wird (fast) immer ein Error anstelle einer Warnung gesetzt (und Programm abgebrochen)
  - kann entfernt werden

12

Datenbank anlegen, löschen, verwenden

- Datenbank anlegen:
  - `CREATE DATABASE [IF NOT EXISTS] db_bezeichnung;`

    Beispiel: `CREATE DATABASE Kundenverwaltung;`
  - Anmerkung: CREATE DATABASE ist ein Synonym für Schema (ORACLE Standard)
  - Zeichensatz einstellen
    - auf der Konsole: SET NAMES 'utf8' und
    - `create database mySchools default character set utf8 collate utf8_general_ci;`

13

- Datenbank löschen:
  - `DROP DATABASE [IF EXISTS] db_bezeichnung;`

    Beispiel: `DROP DATABASE Kundenverwaltung;`
- Datenbank verwenden:
  - `USE db_bezeichnung;`

    Beispiel: `USE Kundenverwaltung;`
- Alle vorhandenen DBs anzeigen
  - `SHOW DATABASES;`

14

Arbeiten mit Tabellen

- Wurde die DB angelegt und der Befehl `use db_bezeichnung` abgesetzt, können in der DB Tabellen angelegt, gelöscht, bearbeitet, geändert, abgefragt … werden.

15

Tabelle anlegen

- `CREATE TABLE [IF NOT EXISTS] table_name (attribut_name datentyp [DEFAULT] [NOT NULL] [UNIQUE] [PRIMARY KEY] [AUTO_INCREMENT] REFERENCES table_name, attribut_name datentyp …);`
- Beispiel:
- `CREATE TABLE Kunden(PID int PRIMARY KEY AUTO_INCREMENT, Vorname varchar(10), Nachname varchar(20));`

16

Tabelle anlegen

- **`primary key:`** legt fest, dass das Attribut PID der Primärschlüssel der Tabelle Kunden ist (es reicht auf Spaltenebene auch KEY, auf Tabellenebene bedeutet KEY aber INDEX!)
- **`auto_increment:`** ein als solches gekennzeichnetes Attribut bewirkt die automatische Inkrementierung (Erhöhung) des Werts eines Attributs um 1
- **`unique:`** Wert ist eindeutig (einmal vorhanden) – PK ist automatisch unique

17

RM1

- **not null:** bedeutet, dass Null-Werte (kein Wert) nicht erlaubt sind, d.h. ein Wert muss explizit gesetzt werden - PK ist automatisch not null.
- Anmerkung: Seit MariaDB wird nur mehr eine Warnung angezeigt, wenn auf ein NOT NULL Attribut ein Wert eingefügt wird. Fehler wird nur geworfen, wenn explizit beim Einfügen NULL angegeben wird!
- Dieses Problem kann nur mittels Trigger gelöst werden – dann wird die Eingabe ignoriert und ein Fehler angezeigt, z.B. die Tabelle PERSON enthält ein Attribut per_nname mit der Eigenschaft NOT NULL

```
create trigger beforeInsertPerson before insert on person
for each row
  begin
    if new.per_nname = '' then
      set new.per_nname=null;
    end if;
end //
```

Stand 7.2.18

18

Spalten- und Tabellenebene

- Manche Eigenschaften können nur auf Spaltenebene (beim Anlegen des Attributs) oder auf Tabellenebene gesetzt werden.
- Es gibt Befehle die können sowohl als auch gesetzt werden.
- Beispiel PRIMARY KEY:

Spaltenebene:

```
CREATE TABLE meine (m_id INT PRIMARY KEY, wert
CHAR(2)) Engine=InnoDB;
```

Tabellenebene:

```
CREATE TABLE meine (m_id INT, wert CHAR(2), PRIMARY
KEY(m_id)) Engine=InnoDB;
```

- Tabellenebene bedeutet demnach nichts anderes, das Attributeigenschaften erst dann gesetzt werden, wenn alle Attribute der Tabelle angegeben wurden.

19

Spaltenebene

- MUSS
  - Datentyp
  - Auto_increment
  - NOT NULL, DEFAULT
- KANN
  - PRIMARY KEY für ein einzelnes Attribut
  - UNIQUE
  - CHECK

20

RM1 Regina Martl; 01.10.2020

Tabellenebene

- MUSS
  - PRIMARY KEY wenn mehrere Attribute den PK bilden (Identifying relationship) – composite key
  - FOREIGN KEY
  - Constraint und References (Referentielle Integrität)
  - Index

Anmerkung: laut Definition sind manche Befehle auch auf Spaltenebene möglich, allerdings funktionieren sie dann nicht mehr eindeutig, z.B. kann die referentielle Integrität geschrieben werden, der Bezug wird aber dann nicht durchgeführt.

21

MySQL > Version 5

- Referentielle Integrität
  - PK – FK Beziehung
  - wird ein PK gelöscht oder geändert bzw. ein FK eingefügt, werden alle Tabellen auf Abhängigkeiten zu diesem Schlüssel geprüft.
  - es besteht die Möglichkeit die Änderung oder Löschung eines PKs
    - abzulehnen -> RESTRICT oder NO ACTION (oder keine Angabe)
      - Erläuterung: RESTRICT ist ohne Verzögerung – manche DB (MySQL gehört nicht dazu) haben eine Verzögerungszeit, ansonsten besteht kein Unterschied zu NO ACTION
    - den entsprechenden FK auf NULL zu setzen -> SET NULL
    - den entsprechenden FK-DS ebenfalls zu löschen oder zu ändern -> CASCADE
  - Anmerkung: wird versucht einen FK einzutragen für den kein entsprechender PK besteht, wird dies immer abgelehnt.

22

- Änderungen und Löschung von DS wird überprüft und bei
  - RESTRICT und NO ACTION abgewiesen.
  - SET NULL wird der PK gelöscht/eingefügt/geändert und der FK auf NULL gesetzt..
  - CASCADE wird der PK und der FK gelöscht/geändert.

23

- Die FK-Beziehung funktioniert unter MySQL nur für InnoDB Tabellen!
- MySQL: Daher muss jeder Tabelle die Teil einer Integritätsbeziehung ist und auch die "FK-"Tabelle, mit Engine=InnoDB gekennzeichnet werden (ab 2020 ist Engine automatisch InnoDB – sollte aber immer überprüft werden).
  - CREATE TABLE person (pid INT PRIMARY KEY) ENGINE = InnoDB;
- MariaDB: innoDB ist der Default Wert und muss daher nicht mehr explizit gesetzt werden.

24

FOREIGN KEY

[CONSTRAINT constraint_name] FOREIGN KEY [id] (index_col_name, ...)

REFERENCES tbl_name (index_col_name, ...)

[ON DELETE {RESTRICT | CASCADE | SET NULL | NO ACTION}]

[ON UPDATE {RESTRICT | CASCADE | SET NULL | NO ACTION}]

25

- CONSTRAINT
  - Das Schlüsselwort CONSTRAINT ist optional, wird aber – falls nicht explizit angeführt – automatisch erstellt.
  - Der Bezeichner einer "Beschränkung" kann herangezogen werden, um die Beschränkung zu löschen.
    - kann abgefragt werden mittels:
      - `SHOW CREATE TABLE tablename;`

26

- Zusätzlich zu FOREIGN KEY und REFERENCES kann für eine Tabelle, die sich auf einen PK bezieht noch manuell ein Index erstellt werden.
- Das setzen eines Indexes KANN den Zugriff auf Werte einer Tabelle eventuell beschleunigen.
- Indexes werden unabhängig gespeichert (eigene Dateien) und bei Zugriffen wird somit nicht die Datenbank belastet.

27

Index

- Auf Spalten, die
  - oft in einer WHERE-Klausel verwendet werden
  - große Datenmengen enthalten
  - usw.
- kann ein Index gesetzt werden, der das Suchen beschleunigt
- Da aber jedem DML Statement der Index ebenfalls neugesetzt wird, wird bei kleinen DB (Unterricht) empfohlen darauf zu verzichten.
- Auf CREATE INDEX wird nicht näher eingegangen.

28

Tabelle mit Werten füllen

- INSERT INTO *table_name* VALUES(…)
  - Bei diesem Statement müssen alle Attribute gesetzt werden, es ist z.B nicht erlaubt, keinen Wert für ein Attribut zu setzen.
- INSERT INTO *table_name* (*Vorname*) VALUES("*name*");
  - hier wird explizit angegeben, dass nur ein Wert gesetzt wird.
  - Attribute vom Typ PK werden trotzdem gesetzt (falls auto_increment gesetzt wurde, ansonsten kann maximal ein Datensatz eingefügt werden mit Wert 0).
  - Attribute die mit not null gekennzeichnet sind, erhalten einen Default-Wert (0).

29

REPLACE

- wird wie INSERT verwendet.
- Liegt aber ein Konflikt beim Einfügen vor, kann dieser durch das Replace-Statement umgangen werden.
- Wird z.B. der Primary Key nicht mit auto_increment gefüllt und es soll ein bestehender Eintrag geändert werden, ist dies u. a. mit Replace möglich.
- Wird auch verwendet, um "ältere" Datensätze aus einer Tabelle in eine andere zu übertragen (kann zusammen mit Select-Statement verwendet werden).
  - `REPLACE INTO test_alt SELECT * FROM test WHERE testid < 10;`

30

Tabellenstruktur abfragen

- `DESCRIBE table_name;`
  - Kurzform `DESC table_name;`
  - liefert die Attribute und deren Eigenschaften
  - ist ein Synonym für `SHOW COLUMNS FROM ...`
- Weitere Möglichkeiten die Tabellenstruktur anzuzeigen:
  - EXPLAIN
    - `EXPLAIN table_name;`
    - Wird zusammen mit SELECT-Statements verwendet um die Effizienz eines Selects abzufragen – dazu wird vor das Select-Statement das Schlüsselwort EXPLAIN geschrieben. explain `select * from tbl_name;`
  - `SHOW COLUMNS FROM table_name;`

31

- SHOW CREATE TABLE zeigt den vollständigen Befehl mit allen gesetzten Tabelleneigenschaften an.
  - `SHOW CREATE TABLE kunden;`
- `SHOW TABLE STATUS FROM database LIKE 'tablename';`

32

Select-Statement

- Nach select können Attribute (Spaltennamen) durch Beistrich getrennt angegeben werden, oder ein Platzhalter * (= alle Spalten).
- Die Attribute können explizit einer Tabelle zugeordnet werden, indem der Tabellenname durch Punkt getrennt dem Attribut vorangestellt wird:
  - kunden.vorname

  Dies wird immer dann benötigt, wenn ein Select-Statement über mehrere Tabellen abgesetzt wird.

33

- **select *:** alle Spalten einer/mehrerer Tabelle(n) wird/werden angezeigt
- **select kunden.vorname:** nur das Attribut Vorname aus der Tabelle Kunden wird angezeigt
- **select * from**
  - nach dem from können alle Tabellen (oder auch nur eine) angegeben werden, die für das Select-Statement benötigt werden.

34

Klausel

- **WHERE-Klausel**: wähle nur jene Datensätze, die der Bedingung nach dem WHERE genügen:
  - ```
    SELECT *
    FROM kunden
    WHERE vorname="Christa";
    ```
- **HAVING-Klausel**: entspricht der WHERE-Klausel wird im Zusammenhang mit Funktionen (und Group By) verwendet:
  - ```
    SELECT count(*) AS Anzahl
    FROM kunden
    HAVING Anzahl >= 10;
    ```

Anmerkung: werden mehrere Attribute ausgewählt und wird Group by "vergessen", wird keine Fehlermeldung mehr (mysql > 5) ausgegeben, sondern ein „falsches“ Ergebnis geliefert

35

Aliase

- Für Spalten- und Tabellenbezeichnungen können Aliase vergeben werden.
- Meist in Verbindung mit Funktionen (Summen etc.),
- aber auch um lange Tabellenbezeichnung (für Schreibfaule) abzukürzen.
  - ```
    select m.markename,
       a.artikelbezeichnung, a.preis
    from marken m, artikel a, artikel_marken am
    where a.aid = am.aid
    and m.mid = am.mid;
    ```

36
    ```

Funktionen und Gruppierung

- Group By
  - gruppiert nach Spalten, wird in Verbindung mit Funktionen (wenn mehrere Spalten ausgegeben werden) verlangt .

```
select m.markename,
       a.artikelbezeichnung,
       sum(a.preis) Summe
from   marken m, artikel a,
       artikel_marken am
where  a.aid = am.aid
and    m.mid = am.mid
group by    Summe;
```

37

Sortierung

- ORDER BY
  - Sortiert eine Tabelle nach einer bestimmten Spalte aufsteigend (ASC ascendens - Standard) oder absteigend (DESC descendens).
  - Wird es zusammen mit Group By verwendet, kann es erst nach der Gruppierung verwendet werden!
  - Sowohl ORDER BY als auch GROUP BY akzeptieren keine Aliase die zB ein Leerzeichen verwenden, da die Angabe als Zeichenkette ("") nicht möglich ist.

38

Anzahl der Ergebnisdatensätze beschränken

- LIMIT
  - Gibt nur eine festgelegt Anzahl von Datensätzen (Zeilen) zurück.
  - Die Datensätze sind dabei bei 0 beginnend durchnummeriert.
    - SELECT *
      FROM artikel
      LIMIT 0, 2;
    - liefert beginnend mit Datensatz Nummer 1, zwei Datensätze zurück.

39

Select-Statement – Funktionen

- MySQL stellt einige vordefinierte Funktionen (allgemeine und Aggregatfunktionen) zur Verfügung. Es ist aber auch möglich eigene Funktionen zu deklarieren – PL/SQL.
- Häufig verwendete Funktionen:
- **max**
  - gibt den Datensatz mit dem höchsten Wert (in bestimmter Spalte) zurück

```
select max(preis) from artikel;
```

- **min**
  - Gegenteil von max

40

- **avg**
  - liefert den Durchschnittswert (einer Spalte) zurück

```
select avg(preis) from artikel;
```

- **sum**
  - liefert die Summe (einer Spalte) zurück

```
select sum(preis) from artikel;
```

- **count**
  - liefert die Anzahl von Datensätzen (einer Spalte) zurück

```
select count(aid) from artikel;
```

- **distinct** (Klausel)
  - doppelte Datensätze nicht ausgeben

```
select distinct nachname from kunden;
```

41

- **CURRENT_DATE() oder CURDATE()**
  - Gibt das aktuelle Datum – JJJJ-MM-TT – zurück.
- **CURRENT_TIME() oder CURTIME()**
  - Gibt die aktuelle Zeit – HHMMSS – zurück.
- **DATE_FORMAT(datum, format)**
  - Gibt das Datum in angegebenem Format zurück
  - siehe Folie
- **DATEDIFF:** Anzahl der Tage zwischen zwei Daten, zB. datediff(now(), birth);

42

JOINS

- Die Verknüpfung mehrerer Tabellen erfolgt durch PK und FK.
- Man unterscheidet zwischen Inner und Outer Joins.
- **Inner Joins**
  - liefern (ohne Subquery) nur vorhandene Einträge, z.B. nur Kunden für die auch schon eine Adresse eingetragen wurde.
- **Outer Joins**
  - liefern im Gegensatz zum Inner Join auch DS für die noch keine Einträge vorhanden sind, z.B. alle Kunden, auch jene, für die noch keine Adresse eingetragen wurde.

43

Beispiele Inner Joins

- INNER JOIN (gleiches gilt auch für JOIN, CROSS JOIN oder STRAIGHT_JOIN)

```
SELECT [*, db.attribut_namen]
FROM table_1 INNER JOIN table_2 ON
table_1.pk = table_2.fk;
```

- Oder mit der WHERE-Klausel (Equijoin)

```
SELECT [*, db.attribut_namen]
FROM table_name
WHERE [db.attribut_name] [operator]
AND [db.attribut_name] [operator]
GROUP BY [db.attribut_name]
ORDER BY [db.attribut_name];
```

44

JOINS

- INNER JOIN - Liefern nur verknüpfte DS
- So genannte **Inner-Joins** können mit
  - `SELECT *`
    `FROM artikel a INNER JOIN artikel_marken am`
    `ON a.aid = am.aid;`

  war vor MySQL 5.x schneller als ohne Schlüsselwort, da zuerst der Join und dann erst alle WHERE Bedingungen geprüft werden (WHERE über eingeschränkte Teilmenge)
- STRAIGHT_JOIN ist ähnlich wie INNER JOIN nur wieder in Verbindung mit WHERE-Klausel.

45

- Alternative Schreibweise bei mehreren Tabellen:
- SELECT *attribute*
  FROM fk_tabelle fkt
  INNER JOIN (table1 t1, table2 t2)
  ON (t1.pk = fkt.fk1 AND t2.pk = fkt.fk2);

Die Reihenfolge in der die Tabellen angeführt werden ist unbedeutend.

46

Equi Join

- Inner Join **ohne** Schlüsselwort. Wird mit der Where-Klausel umgesetzt.
- In der Where-Klausel werden PK und FK verglichen und so die Ausgabedatenmenge eingeschränkt.
- Oracle führte ca. 2015 eine Verbesserung durch, was einen allgemeinen Cross-Join wie davor ausschließt. Daher sind INNER JOIN und Equi Join in etwa gleich schnell und speicherintensiv.
  - select *
    from artikel a, artikel_marken am
    where a.aid = am.aid;

47

NATURAL JOIN

- Ist ein sogenannter Inner Join.
- Bei NATURAL JOIN werden alle Attribute der zu vergleichenden Tabellen gleichen Namens miteinander verglichen:
- `SELECT attributliste FROM tabelleFK NATURAL JOIN (tabelle1, tabelle2);`
- `bzw.`
- `SELECT attributliste FROM tabelleA NATURAL JOIN tabelleB NATURAL JOIN tabelleC;`

48

OUTER JOINS

- *table_name1* LEFT [OUTER] JOIN *table_name2* ON
- oder
- *table_name1* LEFT [OUTER] JOIN *table_name2* USING (*attribut*)
  - Wird z.B. verwendet, um abzufragen, welche Datensätze der Tabelle A keinen Verknüpfungseintrag in Tabelle B haben.
- **Anmerkung**: Anstelle von LEFT kann auch RIGHT verwendet werden – siehe nächste Folien

49

Beispiel LEFT JOIN

- Folgende Tabellen sind gegeben:
  - Artikel: AID, Bezeichnung, Preis
  - Marken: MID, Markenbezeichnung
  - Artikel_Marken: AID, MID
  - Um zu erfahren, welche Artikel noch keiner Marke zugeordnet wurden, kann die Abfrage wie folgt aussehen:
- ON
  - ```
    SELECT *
    FROM artikel LEFT OUTER JOIN artikel_marken
    ON artikel.aid = artikel_marken.aid;
    ```
- USING
  - ```
    SELECT *
    FROM artikel LEFT OUTER JOIN artikel_marken
    USING (aid);
    ```

50

- LEFT OUTER JOIN über mehrere Entities.
- Syntax-Beispiel für drei Entities:
  - ```
    SELECT attribut1, attribut2
    FROM (
        (table1 LEFT OUTER JOIN table2
         ON table1.attribut = table2.attribut)
        LEFT JOIN table3
        ON table2.attribut = table3.attribut)
    WHERE attribut IS NULL;
    ```
  Anmerkung: die Klammerung ist optional

51

RIGHT OUTER JOIN

- MySQL unterstützt auch RIGHT-Joins
- Für Outer Joins gilt auch wieder die alternative Schreibeweise:
- `SELECT attribute FROM fk_tabelle fkt LEFT OUTER JOIN (table1 t1, table2 t2) ON (t1.pk = fkt.fk1 AND t2.pk = fkt.fk2);`

Die Auswahl der Tabellenreihenfolge ist aber signifikant!

52
    ```

Full Outer Join

- Möchte man alle DS aus mehreren Verknüpften Tabellen erhalten, auch wenn noch kein Verknüpfungs-DS eingetragen wurde, benötigt man einen sogenannten Full Outer Join.
- z.B. Es gibt Personen ohne Adresse und es gibt Adressen ohne Personen, aber es gibt auch Personen mit Adresse (und umgekehrt). Will man nun alle Personen aber auch alle Adressen in einer Abfrage ausgeben, erfolgt dies in MySQL/MariaDB mittels
- **`UNION oder UNION ALL`**
  - UNION liefert nur die DS mit NULL und die "Überschneidungen"
  - UNION ALL liefert die Ergebnisse beider Abfragen

53

- Dabei muss die Anzahl der Attribute verbundener Abfragen übereinstimmen.
- Beispiel:

```
select per_id, adr_id from person left outer join
person_adresse using(per_id)
union
select per_id, adr_id from adresse left outer join
person_adresse using(adr_id);
```

54

Tabelleninhalt löschen

- DELETE löscht Zeilen aus einer Tabelle. Die Tabelle selbst wird nicht gelöscht.
- DELETE FROM *table_name*;
  - Löscht den gesamten Tabelleninhalt.
- Der Delete-Befehl kann auch zusammen mit einer WHERE-Klausel verwendet werden.
  - DELETE FROM artikel WHERE aid = 7;
- Weiters ist es möglich gleichzeitig Attribute aus mehreren Tabellen zu löschen.
  - `DELETE artikel.*, artikel_marken.*`
    `FROM artikel, artikel_marken (auch möglich ist artikel.*)`
    `WHERE artikel.aid = artikel_marken.aid;`
  - Alias kann auch verwendet werden (`delete a, am from artikel a, artikel_marken am where a.aid = am.aid;`)

55

Tabellen löschen

- `DROP TABLE table_name;`
  - Löscht die gesamte Tabelle inklusive Inhalt und Struktur;
- `TRUNCATE table_name;`
  - Führt ein DROP-Statement auf die Tabelle aus und erstellt diese anschließend neu.
  - Erlaubt kein rollback! Es wird Zeile für Zeile durchgeführt, wenn ein Fehler auftritt, kann nichts mehr rückgängig gemacht werden.

56

Bestehenden Tabelleninhalt ändern

- UPDATE *table_name*
  SET *attribut1 = wert*
  WHERE *attribut2 operator wert;*

mehrere Attribute ändern – Beispiel

```
update      person
set         fname="Lara", lname="Croft"
where       per_id =  1;
```

57

Bestehende Tabellenstruktur ändern

- ALTER TABLE wdednbqdp h
  ADD | MODIFY | CHANGE |
  ALTER | RENAME | DROP
  dwwulexwqhxhubz huw>

58

ADD

- Wird verwendet, um ein Attribut (Spalte) hinzuzufügen.
  - ALTER TABLE *kunden* ADD *nachname varchar(10);*
- Das neue Attribut kann am Anfang (FIRST) oder nach einem bestimmten Attribut (AFTER) eingefügt werden (Standard am Ende).
  - ALTER TABLE *kunden* ADD *vorname varchar(10)* AFTER *nachname;*
  - ALTER TABLE *kunden* ADD *pid int primary key auto_increment* FIRST;
- Foreign Key:
  - ALTER TABLE table_name ADD FOREIGN KEY (schluessel) REFERENCES table_name_pk(pk) ON ...

59

MODIFY

- Wird verwendet, um die Eigenschaft eines Attributs (Spalte) zu verändern, z.B. den Datentyp
  - ALTER TABLE *kunden* MODIFY *nachname varchar(255);*
- Durch die Angabe FIRST können bestehende Attribute in der Tabellenstruktur an die erste Stelle verschoben werden.
- PK kann nicht verschoben werden!
  - ALTER TABLE *kunden* MODIFY *pid int first;*
  - ACHTUNG: auto_increment MUSS auch angegeben werden, sonst geht es durch das Verschieben verloren!
- Durch die Angabe von AFTER kann bestimmt werden, nach welchem Attribut das zu ändernde Attribute zukünftig stehen sollen.
  - ALTER TABLE *kunden* MODIFY *vorname varchar(255)* AFTER *zahl;*

60

CHANGE und RENAME

- Change wird verwendet, um Attributbezeichnung zu ändern.
  - `ALTER TABLE kunden CHANGE nachname lastName varchar(255);`
    - Datentyp muss angegeben werden, kann aber auch gleichzeitig verändert werden!
    - Achtung: Falls die Tabelle bereits DS enthält – z.B. vorher DOUBLE und später INT
- Rename wird verwendet, um Tabellenbezeichnung zu ändern.
  - `ALTER TABLE kunden RENAME [AS] customer;`

61

ALTER

- Standardwerte können mittels SET DEFAULT gesetzt werden.
- `ALTER TABLE kunden ALTER zahl SET DEFAULT 100;`

62

Auto_Increment zurücksetzen

- `ALTER TABLE table_name AUTO_INCREMENT=wert;`
- Der Wert muss dabei Höher als der letzte Eintrag sein (bei keinem bestehenden Eintrag kann der Wert gleich 1 sein).
- Letzten Wert für auto_increment herausfinden:
  - `show create table tbl_name;`
  - `select max(id_attribute) from tbl_name;`
  - `select last_insert_id() from tblname; -- liefert keinen Wert?`

63

DROP

- drop: Bestehende Attribute löschen. Der Befehl ... drop primary key; löscht den PK (sofern dieser nicht vom Typ auto_increment ist) oder den ersten Eintrag vom Typ unique.
- PK Attribute können aber vollständig gelöscht werden
- Um die Eigenschaft PRIMARY KEY AUTO_INCREMENT eines Attributs zu entfernen, muss man wie folgt vorgehen:
  - `CREATE TABLE test(testid INT PRIMARY KEY AUTO_INCREMENT);`
  - `ALTER TABLE test MODIFY testid INT;`
    - entfernt die Eigenschaft AUTO_INCREMENT
  - `ALTER TABLE test DROP PRIMARY KEY;`
    - entfernt die Eigenschaft PRIMARY KEY

64

Beschränkungen löschen

- Beispiel:

```
CREATE TABLE person (
  idperson int(11) PRIMARY KEY AUTO_INCREMENT,
  vname varchar(45) DEFAULT NULL,
  nname varchar(45) NOT NULL,
  lehrberuf_idlehrberuf int(11) NOT NULL,
CONSTRAINT fk_person_lehrberuf FOREIGN KEY
  (lehrberuf_idlehrberuf) REFERENCES lehrberuf
  (idlehrberuf)) ENGINE=InnoDB;
```

- `ALTER TABLE person DROP FOREIGN KEY fk_person_lehrberuf;`
- Wurde der CONSTRAINT Bezeichner automatisch erstellt, kann er mittels SHOW CREATE TABLE ... abgefragt/angezeigt werden.

65

- **Der Datentyp muss bei jeder Änderung – ausgenommen DROP, RENAME und ALTER -bekannt gegeben werden.**

66

Subquery

- IN, ANY, ALL (SOME), NOT EXISTS für Multiple-Row-Queries (Queries die mehr als einen DS zurückliefern)
  - ANY und ALL werden immer mit einem Operator (= < > >= <= !=) verwendet:
    - =ANY ist äquivalent zu IN
    - SOME ist an Synonym für ANY
    - IN kann durch NOT IN negiert werden
- oder = >= <= > < für Single-Row-Queries
- weiters versteht man unter Subquery auch sogenannte temporäre (derived) Tables, das sind Tabellenergebnisse in der FROM-Klausel, die als Attribut eingebettet werden.

67