Name:

Klasse:

Datum:

2 Metadaten

Bei der Einführung wurde schon gezeigt, wie Metadaten über die Verbindung (Klasse `Connection`) angefragt werden können.

Es lassen sich aber noch viel mehr Informationen abfragen (Auszug):

- Informationen über die Datenbank
- Informationen über Tabellen, Views, ...
- Informationen über Schlüssel (`primary key`, `foreign key`, ...)
- ...

2.1 Datenbank

Mit der Methode `getMetaData()` lassen sich zusätzliche Verbindungsinformationen abfragen. Dabei erhält man eine Instanz von `DatabaseMetaData`.

```java
String catalog = con.getCatalog();
String schema = con.getSchema();

DatabaseMetaData meta = con.getMetaData();

System.out.println("DatabaseMajorVersion   : "
    + meta.getDatabaseMajorVersion());
System.out.println("DatabaseMinorVersion   : "
    + meta.getDatabaseMinorVersion());
System.out.println("DatabaseProductName    : "
    + meta.getDatabaseProductName());
System.out.println("DatabaseProductVersion : "
    + meta.getDatabaseProductVersion());
System.out.println("DriverMajorVersion     : "
    + meta.getDriverMajorVersion());
System.out.println("DriverMinorVersion     : "
    + meta.getDriverMinorVersion());
System.out.println("DriverName             : "
    + meta.getDriverName());
System.out.println("DriverVersion          : "
    + meta.getDriverVersion());
System.out.println("JDBCMajorVersion       : "
    + meta.getJDBCMajorVersion());
System.out.println("JDBCMinorVersion       : "
    + meta.getJDBCMinorVersion());
System.out.println("catalog                : " + catalog);
System.out.println("schema                 : " + schema);
```

Ausgabe: (Beispiel)

```
... connected
DatabaseMajorVersion    : 11
DatabaseMinorVersion    : 2
DatabaseProductName     : MariaDB
DatabaseProductVersion  : 11.2.3-MariaDB-1:11.2.3+maria~ubu2204
DriverMajorVersion      : 3
DriverMinorVersion      : 0
DriverName              : MariaDB Connector/J
DriverVersion           : 3.0.8
JDBCMajorVersion        : 4
JDBCMinorVersion        : 2
catalog                 : gm3
schema                  : null
```

Aufgabe   JDBC-02-1

1. Erstellen Sie eine Klasse (`Bsp_MetaDatabase`), die obige Werte ausgibt.

2. Was liefert die Information `catalog` bzw. `schema`?

Notiz:

2.2 Tabellen

Tabellen werden über die Methode `getTables()` ermittelt. Dabei muss der Type angegeben werden, hier `TABLE`.

```
DatabaseMetaData meta = con.getMetaData();
ResultSet rs = meta.getTables(catalog, schema, null,
                          new String[] { "TABLE" });

System.out.println("Tabellen aus " + catalog + " ...");
while (rs.next()) {
   String name = rs.getString("TABLE_NAME");
   System.out.print("'" + name + "' ");
}
```

Ausgabe:

```
... connected
Tabellen aus gm3 ...
'abteilung' 'bestellung' 'bestpos' 'bundesland' 'einheit' 'funktion' 'gehalt' ⤸
'kunde' 'landkreis' 'lieferant' 'lieferpos' 'lieferung' 'mitarbeiter' '⤸
mitarbeitererw' 'ort' 'produkt' 'typ' 'version' 'xliefer'
```

2.2.1 Tabellen-Typen

Welche Typen das DBMS-System unterstützt, kann mit der Methode `getTableTypes()` ermittelt werden.

Dabei sind folgende Typen definiert, werden aber nicht von allen Systemen unterstützt:

- TABLE
- VIEW
- SYSTEM TABLE
- GLOBAL TEMPORARY
- LOCAL TEMPORARY
- ALIAS
- SYNONYM

```
DatabaseMetaData meta = con.getMetaData();
ResultSet rs = meta.getTableTypes();

System.out.println("Tabellentypen");
while (rs.next()) {
   String name = rs.getString("TABLE_TYPE");
   System.out.print("'" + name + "' ");
}
```

Ausgabe:

```
... connected
Tabellentypen
'TABLE' 'SYSTEM VIEW' 'VIEW'
```

2.2.2 Tabellen-Spalten

Welche Tabellenspalten verwendet werden, kann mit der Methode `getColumns()` ermittelt werden.

```java
    DatabaseMetaData meta = con.getMetaData();
    ResultSet rs = meta.getColumns(catalog,schema,null,null);

    System.out.println("Spaltennamen");
    int colcount = rs.getMetaData().getColumnCount();
    for (int i = 1; i <= colcount; i++) {
       System.out.print("'" + rs.getMetaData().getColumnLabel(i) + "' ");
    }
```

Ausgabe:

```
... connected
Spaltennamen
'TABLE_CAT' 'TABLE_SCHEM' 'TABLE_NAME' 'COLUMN_NAME' 'DATA_TYPE' 'TYPE_NAME' '⤸
COLUMN_SIZE' 'BUFFER_LENGTH' 'DECIMAL_DIGITS' 'NUM_PREC_RADIX' 'NULLABLE' '⤸
REMARKS' 'COLUMN_DEF' 'SQL_DATA_TYPE' 'SQL_DATETIME_SUB' 'CHAR_OCTET_LENGTH' '⤸
ORDINAL_POSITION' 'IS_NULLABLE' 'SCOPE_CATALOG' 'SCOPE_SCHEMA' 'SCOPE_TABLE' '⤸
SOURCE_DATA_TYPE' 'IS_AUTOINCREMENT' 'IS_GENERATEDCOLUMN'
```

aus der API:

```
Each column description has the following columns:

TABLE_CAT String => table catalog (may be null)
TABLE_SCHEM String => table schema (may be null)
TABLE_NAME String => table name
COLUMN_NAME String => column name
DATA_TYPE int => SQL type from java.sql.Types
TYPE_NAME String => Data source dependent type name, for a UDT the type name ⤸
is fully qualified
COLUMN_SIZE int => column size.
BUFFER_LENGTH is not used.
DECIMAL_DIGITS int => the number of fractional digits. Null is returned for ⤸
data types where DECIMAL_DIGITS is not applicable.
NUM_PREC_RADIX int => Radix (typically either 10 or 2)
NULLABLE int => is NULL allowed.
   columnNoNulls - might not allow NULL values
   columnNullable - definitely allows NULL values
   columnNullableUnknown - nullability unknown
REMARKS String => comment describing column (may be null)
COLUMN_DEF String => default value for the column, which should be interpreted⤸
 as a string when the value is enclosed in single quotes (may be null)
SQL_DATA_TYPE int => unused
SQL_DATETIME_SUB int => unused
CHAR_OCTET_LENGTH int => for char types the maximum number of bytes in the ⤸
column
ORDINAL_POSITION int => index of column in table (starting at 1)
IS_NULLABLE String => ISO rules are used to determine the nullability for a ⤸
column.
   YES --- if the column can include NULLs
   NO --- if the column cannot include NULLs
   empty string --- if the nullability for the column is unknown
SCOPE_CATALOG String => catalog of table that is the scope of a reference ⤸
attribute (null if DATA_TYPE isn't REF)
SCOPE_SCHEMA String => schema of table that is the scope of a reference ⤸
attribute (null if the DATA_TYPE isn't REF)
SCOPE_TABLE String => table name that this the scope of a reference attribute ⤸
(null if the DATA_TYPE isn't REF)
SOURCE_DATA_TYPE short => source type of a distinct type or user-generated Ref⤸
 type, SQL type from java.sql.Types (null if DATA_TYPE isn't DISTINCT or user-⤸
generated REF)
```

```
IS_AUTOINCREMENT String => Indicates whether this column is auto incremented
    YES --- if the column is auto incremented
    NO --- if the column is not auto incremented
    empty string --- if it cannot be determined whether the column is auto ⟩
    incremented
IS_GENERATEDCOLUMN String => Indicates whether this is a generated column
    YES --- if this a generated column
    NO --- if this not a generated column
    empty string --- if it cannot be determined whether this is a generated ⟩
    column


The COLUMN_SIZE column specifies the column size for the given column. For ⟩
numeric data, this is the maximum precision. For character data, this is the ⟩
length in characters. For datetime datatypes, this is the length in characters⟩
 of the String representation (assuming the maximum allowed precision of the ⟩
fractional seconds component). For binary data, this is the length in bytes. ⟩
For the ROWID datatype, this is the length in bytes. Null is returned for data⟩
 types where the column size is not applicable.
```

Nun kann man die Spaltendefinition einer Tabelle ausgeben.

```
DatabaseMetaData meta = con.getMetaData();
ResultSet rs = meta.getColumns(catalog, schema, "produkt", null);

Table t = new Table(6);
t.addCell("Spalte");
t.addCell("Type");
t.addCell("Größe");
t.addCell("Dezimal");
t.addCell("NULL");
t.addCell("Position");

while (rs.next()) {
   t.addCell(rs.getString("COLUMN_NAME"));
   t.addCell(rs.getString("TYPE_NAME"));
   t.addCell(rs.getString("COLUMN_SIZE"));
   t.addCell(rs.getString("DECIMAL_DIGITS"));
   t.addCell(rs.getString("NULLABLE"));
   t.addCell(rs.getString("ORDINAL_POSITION"));
}
System.out.println(t.render());
```

Ausgabe:

```
... connected
+------------+-------+-----+-------+----+--------+
|Spalte      |Type   |Größe|Dezimal|NULL|Position|
+------------+-------+-----+-------+----+--------+
|id          |INT    |10   |0      |0   |1       |
|bez         |VARCHAR|50   |       |1   |2       |
|vpreis      |DECIMAL|9    |2      |1   |3       |
|mwst        |DECIMAL|5    |2      |1   |4       |
|lagerbestand|INT    |10   |0      |1   |5       |
|eid         |INT    |10   |0      |1   |6       |
|tid         |INT    |10   |0      |1   |7       |
+------------+-------+-----+-------+----+--------+
```

2.3 Schlüssel

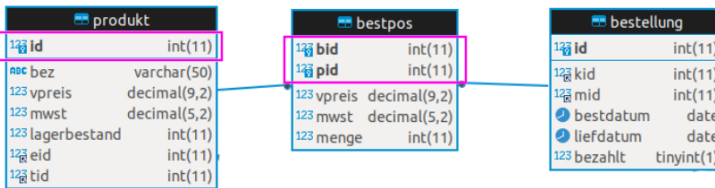Im nächsten Schritt werden die Schlüssel (Primärschlüssel und Fremdschlüssel) ermittelt.

2.3.1 Primärschlüssel

Die Primärschlüssel werden über die Methode `getPrimaryKeys` ermittelt. Zu beachten ist, dass bei einem relationalen DB-System pro Tabelle nur ein Primärschlüssel definiert werden kann, aber dieser aus mehreren Spalten (Verbundschlüssel) bestehen kann.

```java
DatabaseMetaData meta = con.getMetaData();
ResultSet rs = meta.getPrimaryKeys(catalog, schema, "produkt");

Table t = new Table(3);
t.addCell("COLUMN_NAME");
t.addCell("PK_NAME");
t.addCell("KEY_SEQ");

while (rs.next()) {
   t.addCell(rs.getString("COLUMN_NAME"));
   t.addCell(rs.getString("PK_NAME"));
   t.addCell(rs.getString("KEY_SEQ"));
}
System.out.println(t.render());
```


Ausgabe:

```
... connected
+-----------+-------+-------+
|COLUMN_NAME|PK_NAME|KEY_SEQ|
+-----------+-------+-------+
|id         |PRIMARY|1      |
+-----------+-------+-------+
```

aus der API:

```
Each primary key column description has the following columns:

TABLE_CAT String => table catalog (may be null)
TABLE_SCHEM String => table schema (may be null)
TABLE_NAME String => table name
COLUMN_NAME String => column name
KEY_SEQ short => sequence number within primary key( a value of 1 represents ⤸
the first column of the primary key, a value of 2 would represent the second ⤸
column within the primary key).
PK_NAME String => primary key name (may be null)
```

Ein zusammengesetzter PK bei der Tabelle `bestpos` sieht dabei wie folgt aus:

```
... connected
+-----------+-------+-------+
|COLUMN_NAME|PK_NAME|KEY_SEQ|
+-----------+-------+-------+
|bid        |PRIMARY|1      |
|pid        |PRIMARY|2      |
+-----------+-------+-------+
```
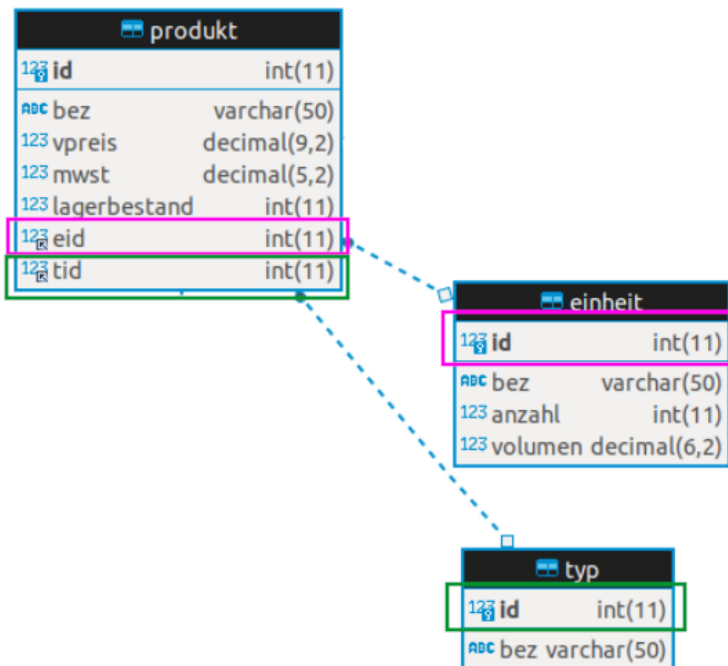
2.3.2 Fremdschlüssel

Die Fremdschlüssel werden über zwei Wege ermittelt. Einmal direkt über die Tabelle, wenn dort ein Fremdschlüssel definiert worden ist mit der Methode `getExportedKeys()` und über die Beziehung mit der Tabelle, bei der der Primärschlüssel definiert worden ist, mit der Methode `getImportedKeys()`.

```
DatabaseMetaData meta = con.getMetaData();
ResultSet rs = meta.getImportedKeys(catalog, schema, "produkt");

Table t = new Table(5);
t.addCell("FK_NAME");
t.addCell("FKTABLE_NAME");
t.addCell("FKCOLUMN_NAME");
t.addCell("PKTABLE_NAME");
t.addCell("PKCOLUMN_NAME");

while (rs.next()) {
    t.addCell(rs.getString("FK_NAME"));
    t.addCell(rs.getString("FKTABLE_NAME"));
    t.addCell(rs.getString("FKCOLUMN_NAME"));
    t.addCell(rs.getString("PKTABLE_NAME"));
    t.addCell(rs.getString("PKCOLUMN_NAME"));
}
System.out.println(t.render());
```


Ausgabe:

```
... connected
+------------------+------------+-------------+------------+-------------+
|FK_NAME           |FKTABLE_NAME|FKCOLUMN_NAME|PKTABLE_NAME|PKCOLUMN_NAME|
+------------------+------------+-------------+------------+-------------+
|fk_produkt_einheit|produkt     |eid          |einheit     |id           |
|fk_produkt_typ    |produkt     |tid          |typ         |id           |
+------------------+------------+-------------+------------+-------------+
```

Zum Vergleich, hier die verkürzte Ausgabe des Befehls `SHOW CREATE TABLE PRODUKT`:

```
CREATE TABLE `produkt` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `bez` varchar(50) DEFAULT NULL,
  `vpreis` decimal(9,2) DEFAULT NULL,
  `mwst` decimal(5,2) DEFAULT NULL,
  `lagerbestand` int(11) DEFAULT NULL,
  `eid` int(11) DEFAULT NULL,
  `tid` int(11) DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `fk_produkt_einheit` (`eid`),
  KEY `fk_produkt_typ` (`tid`),
  CONSTRAINT `fk_produkt_einheit`
          FOREIGN KEY (`eid`) REFERENCES `einheit` (`id`),
  CONSTRAINT `fk_produkt_typ`
          FOREIGN KEY (`tid`) REFERENCES `typ` (`id`)
)
```

aus der API:

```
Each primary key column description has the following columns:

PKTABLE_CAT String => primary key table catalog being imported (may be null)
PKTABLE_SCHEM String => primary key table schema being imported (may be null)
PKTABLE_NAME String => primary key table name being imported
PKCOLUMN_NAME String => primary key column name being imported
FKTABLE_CAT String => foreign key table catalog (may be null)
FKTABLE_SCHEM String => foreign key table schema (may be null)
FKTABLE_NAME String => foreign key table name
FKCOLUMN_NAME String => foreign key column name
KEY_SEQ short => sequence number within a foreign key( a value of 1 represents↲
 the first column of the foreign key, a value of 2 would represent the second ↲
column within the foreign key).
UPDATE_RULE short => What happens to a foreign key when the primary key is ↲
updated:
    importedNoAction - do not allow update of primary key if it has been ↲
    imported
    importedKeyCascade - change imported key to agree with primary key update
    importedKeySetNull - change imported key to NULL if its primary key has ↲
    been updated
    importedKeySetDefault - change imported key to default values if its ↲
    primary key has been updated
    importedKeyRestrict - same as importedKeyNoAction (for ODBC 2.x ↲
    compatibility)
DELETE_RULE short => What happens to the foreign key when primary is deleted.
    importedKeyNoAction - do not allow delete of primary key if it has been ↲
    imported
    importedKeyCascade - delete rows that import a deleted key
    importedKeySetNull - change imported key to NULL if its primary key has ↲
    been deleted
    importedKeyRestrict - same as importedKeyNoAction (for ODBC 2.x ↲
    compatibility)
    importedKeySetDefault - change imported key to default if its primary key ↲
    has been deleted
FK_NAME String => foreign key name (may be null)
PK_NAME String => primary key name (may be null)
DEFERRABILITY short => can the evaluation of foreign key constraints be ↲
deferred until commit
    importedKeyInitiallyDeferred - see SQL92 for definition
    importedKeyInitiallyImmediate - see SQL92 for definition
    importedKeyNotDeferrable - see SQL92 for definition
```

Hier nun die Sichtweise von der „anderen" Tabelle aus.

```
DatabaseMetaData meta = con.getMetaData();
ResultSet rs = meta.getExportedKeys(catalog, schema, "produkt");

Table t = new Table(5);
t.addCell("FK_NAME");
t.addCell("FKTABLE_NAME");
t.addCell("FKCOLUMN_NAME");
t.addCell("PKTABLE_NAME");
t.addCell("PKCOLUMN_NAME");

while (rs.next()) {
    t.addCell(rs.getString("FK_NAME"));
    t.addCell(rs.getString("FKTABLE_NAME"));
    t.addCell(rs.getString("FKCOLUMN_NAME"));
    t.addCell(rs.getString("PKTABLE_NAME"));
    t.addCell(rs.getString("PKCOLUMN_NAME"));
}
System.out.println(t.render());
```

Ausgabe:

```
... connected
+-------------------+------------+-------------+------------+-------------+
|FK_NAME            |FKTABLE_NAME|FKCOLUMN_NAME|PKTABLE_NAME|PKCOLUMN_NAME|
+-------------------+------------+-------------+------------+-------------+
|fk_bestpos_produkt |bestpos     |pid          |produkt     |id           |
|fk_lieferpos_produkt|lieferpos  |pid          |produkt     |id           |
+-------------------+------------+-------------+------------+-------------+
```

Zum Vergleich, hier die verkürzte Ausgabe der Befehle `SHOW CREATE TABLE ...`:

```
CREATE TABLE `bestpos` (
  `bid` int(11) NOT NULL,
  `pid` int(11) NOT NULL,
  `vpreis` decimal(9,2) DEFAULT NULL,
  `mwst` decimal(5,2) DEFAULT NULL,
  `menge` int(11) DEFAULT NULL,
  PRIMARY KEY (`bid`,`pid`),
  KEY `fk_bestpos_produkt` (`pid`),
  CONSTRAINT `fk_bestpos_bestellung`
            FOREIGN KEY (`bid`) REFERENCES `bestellung` (`id`),
  CONSTRAINT `fk_bestpos_produkt`
            FOREIGN KEY (`pid`) REFERENCES `produkt` (`id`)
)
```

```
CREATE TABLE `lieferpos` (
  `lid` int(11) NOT NULL,
  `pid` int(11) NOT NULL,
  `vpreis` decimal(9,2) DEFAULT NULL,
  `mwst` decimal(5,2) DEFAULT NULL,
  `menge` int(11) DEFAULT NULL,
  PRIMARY KEY (`lid`,`pid`),
  KEY `fk_lieferpos_produkt` (`pid`),
  CONSTRAINT `fk_lieferpos_lieferung`
            FOREIGN KEY (`lid`) REFERENCES `lieferung` (`id`),
  CONSTRAINT `fk_lieferpos_produkt`
            FOREIGN KEY (`pid`) REFERENCES `produkt` (`id`)
)
```

Notiz:

Aufgabe   JDBC-02-2

1. Erstellen Sie ein Java-Programm, welches die Metadaten aller Tabellen einer Datenbank in einer Textdatei ausgibt. Als Vorlage dient hier der Befehl DESC, jedoch soll dieser nicht verwendet werden, da dieser nur bei MySQL bzw. MariaDB funktioniert, da dieser nicht Standard-SQL ist. Verwenden Sie dazu die Bibliothek zum Texttabellen erstellen.

Beispiel:

```
abteilung
+-----+----------+----+---+-------+-------------+
|Field|Type      |Null|Key|Default|Extra        |
+-----+----------+----+---+-------+-------------+
|id   |INT(10)   |NO  |PRI|NULL   |auto_increment|
|name |VARCHAR(50)|YES |   |NULL   |             |
|lid  |INT(10)   |YES |MUL|NULL   |             |
+-----+----------+----+---+-------+-------------+
bestellung
+---------+--------+----+---+-------+-------------+
|Field    |Type    |Null|Key|Default|Extra        |
+---------+--------+----+---+-------+-------------+
|id       |INT(10) |NO  |PRI|NULL   |auto_increment|
|kid      |INT(10) |YES |MUL|NULL   |             |
|mid      |INT(10) |YES |MUL|NULL   |             |
|bestdatum|DATE(10)|YES |   |NULL   |             |
|liefdatum|DATE(10)|YES |   |NULL   |             |
|bezahlt  |BIT(3)  |YES |   |NULL   |             |
+---------+--------+----+---+-------+-------------+
...
```

2. Erstellen Sie ein Java-Programm, welches die Daten jeder Tabelle in eine entsprechende CSV-Datei export. Verwenden Sie hierzu Apache Commons CSV.

```xml
<dependency>
    <groupId>org.apache.commons</groupId>
    <artifactId>commons-csv</artifactId>
    <version>1.10.0</version>
</dependency>
```

Beispiel Tabelle `abteilung`:

```
"id";"name";"lid"
"1";"Einkauf";"7"
"2";"Verkauf";"19"
"3";"Fahrdienst";"9"
...
```

3. Erstellen Sie ein Java-Programm, welches die Metadaten aller Tabellen einer Datenbank in einer JSON-Datei ausgibt (anspruchsvoll). Verwenden Sie hierzu jackson und folgendes Tutorial.

```xml
    <dependency>
        <groupId>com.fasterxml.jackson.core</groupId>
        <artifactId>jackson-core</artifactId>
        <version>2.16.2</version>
    </dependency>

    <dependency>
        <groupId>com.fasterxml.jackson.core</groupId>
        <artifactId>jackson-annotations</artifactId>
        <version>2.16.2</version>
    </dependency>

    <dependency>
        <groupId>com.fasterxml.jackson.core</groupId>
        <artifactId>jackson-databind</artifactId>
        <version>2.16.2</version>
    </dependency>
```

Beispiel:

```json
{
  "funktion" : [ {
    "cols" : [ "id", "INT(10)", "NO", "PRI", "NULL", "auto_increment" ]
  }, {
    "cols" : [ "taetigkeit", "VARCHAR(50)", "YES", "", "NULL", "" ]
  } ],
...
```

4. Erstellen Sie ein Java-Programm, welches die Metadaten aller Tabellen einer Datenbank in einer Textdatei ausgibt (erstmal ohne Tabellendaten). Als Vorlage dient hier der Dump-Befehl mit SHOW CREATE TABLE ... (anspruchsvoll).

Beispiel Tabelle `abteilung`:

```
...
--
-- Table structure for table abteilung
--
DROP TABLE IF EXISTS abteilung;
CREATE TABLE `abteilung` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `name` varchar(50) DEFAULT NULL,
  `lid` int(11) DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `fk_abteilung_mitarbeiter` (`lid`),
  CONSTRAINT `fk_abteilung_mitarbeiter` FOREIGN KEY (`lid`) REFERENCES `↩
  mitarbeiter` (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=8 DEFAULT CHARSET=utf8mb3
...
```