5 Aggregatfunktionen

- Auflistung der Gesamtsumme einer Reservierung für jeden Kunden

```
SELECT g.gnr, g.name, r.rnr,
       SUM(z.preis*(rp.bis-rp.von)) AS gesamtsumme
FROM gast g JOIN reservierung r USING (gnr)
            JOIN rposition rp USING (rnr)
            JOIN zimmer z ON z.honr = rp.honr AND z.znr = rp.znr
GROUP BY g.gnr, g.name, r.rnr;
```

Bei der Verwendung von Aggregatfunktionen müssen alle Attribute, auf die im SELECT-Teil projiziert wird und die **nicht** in der Aggregatfunktion verwendet werden, in der GROUP BY-Klausel aufgeführt werden.

5 Aggregatfunktionen

- Anzahl aller Hotels

```
SELECT COUNT(*)
FROM hotel;
```

```
SELECT COUNT(ort)
FROM hotel;
```

- Anzahl der verschiedenen Orte

```
SELECT COUNT(DISTINCT ort)
FROM hotel;
```

- Maximaler Zimmerpreis über alle Hotels

```
SELECT MAX(preis)
FROM zimmer;
```

- Maximaler Zimmerpreis für jedes Hotel

```
SELECT MAX(preis), honr
FROM zimmer
GROUP BY honr;
```

5 Aggregatfunktionen

- Auflistung der Hotelnamen mit teuersten Zimmer

```
SELECT h.honr, h.hname
FROM zimmer z JOIN hotel h USING (honr)
WHERE z.preis IN (
          SELECT MAX(preis)
          FROM zimmer);
```

- Auflistung der teuersten Zimmer eines jeden Hotels

```
SELECT h.honr, h.hname, z.znr, z.zname, z.preis
FROM zimmer z JOIN hotel h USING (honr)
WHERE (z.honr, z.preis) IN (
          SELECT z.honr, MAX(preis)
          FROM zimmer z
          GROUP BY z.honr);
```

5 Eigenschaftsüberprüfung von Aggregatfunktionen

- Auflistung aller Hotels, deren maximaler Zimmerpreis kleiner als 150 Euro ist

```
SELECT h.honr, h.hname
FROM hotel h JOIN zimmer z USING (honr)
GROUP BY h.honr, h.hname
HAVING MAX(z.preis) < 150;
```

5 UNION

- Auflistung des Gesamtumsatzes eines Kunden (nur Zimmerreservierungen)

```
SELECT g.gnr, g.name, SUM(z.preis*(rp.bis-rp.von)) AS umsatz
FROM gast g JOIN reservierung r USING (gnr)
            JOIN rposition rp USING (rnr)
            JOIN zimmer z ON z.honr = rp.honr AND z.znr = rp.znr
GROUP BY g.gnr, g.name;
```

- Problem: Kunden mit Umsatz 0!

```
SELECT g.gnr, g.name, SUM(z.preis*(rp.bis-rp.von)) AS umsatz
FROM gast g LEFT JOIN reservierung r USING (gnr)
            LEFT JOIN rposition rp USING (rnr)
            LEFT JOIN zimmer z ON
                      z.honr = rp.honr AND z.znr = rp.znr
GROUP BY g.gnr, g.name;
```

5 UNION

- Problem: Möglicherweise Anzeige NULL statt 0

```
SELECT g.gnr, g.name, SUM(z.preis*(rp.bis-rp.von)) AS umsatz
FROM gast g JOIN reservierung r USING (gnr)
            JOIN rposition rp USING (rnr)
            JOIN zimmer z ON z.honr = rp.honr AND z.znr = rp.znr
GROUP BY g.gnr, g.name
UNION
SELECT g.gnr, g.name, 0
FROM gast g
WHERE g.gnr NOT IN (SELECT r.gnr FROM reservierung r);
```

- Aber: UNION ist langsam!

5 Bedingte Ausgaben

- Ausgabewerte im SELECT können an Bedingungen geknüpft werden

```
CASE  {<ausdruck> | <spaltenname>}
WHEN  <wert>
THEN  {<wert> | < ausdruck > | <spaltenname>}
…
ELSE  {<wert> | < ausdruck > | <spaltenname>}
END
```

5 Bedingte Ausgaben

- Auflistung des Gesamtumsatzes eines Kunden mit LEFT JOIN und Ersetzung des NULL-Wertes durch 0

```
SELECT g.gnr, g.name, CASE (SUM(z.preis*(rp.bis-rp.von)) IS NULL)
                      WHEN true
                      THEN 0
                      ELSE SUM(z.preis*(rp.bis-rp.von)) END
                      AS umsatz
FROM gast g LEFT JOIN reservierung r USING (gnr)
            LEFT JOIN rposition rp USING (rnr)
            LEFT JOIN zimmer z ON
                      z.honr = rp.honr AND z.znr = rp.znr
GROUP BY g.gnr, g.name;
```

5 View Definition Language

- View ist eine virtuelle Tabelle ohne eigene Datensätze
  - Datenretrieval wie normale Tabelle
  - Datenspeicherung nicht oder nur begrenzt möglich
- Erstellung individueller Sichten auf die Datenbasis
  - Gezielte Denormalisierung zur Datenaufbereitung
  - Zugriffssteuerung
  - Vereinfachung komplexer Abfragen / „Speicherung“ von Zwischenergebnisse
  - Stabile Schnittstelle in andere Systeme

5 View Definition Language

```
CREATE VIEW <viewname> [(<spaltenname> [,  ...])] AS <select befehl>;
```

```
REPLACE VIEW <viewname> [(<spaltenname> [,  ...])] AS <select befehl>;
```

```
CREATE OR REPLACE VIEW <viewname> [(<spaltenname> [,  ...])]
                                                  AS <select befehl>;
```

```
DROP VIEW <viewname>;
```

5 Erstellung einer individuellen Sicht / Datenaufbereitung

- Erstellung der Kundenauswertung als VIEW

```
CREATE VIEW kundenumsatz AS
SELECT g.gnr, g.name, CASE (SUM(z.preis*(rp.bis-rp.von)) IS NULL)
                      WHEN true
                      THEN 0
                      ELSE SUM(z.preis*(rp.bis-rp.von)) END
                      AS umsatz
FROM gast g LEFT JOIN reservierung r USING (gnr)
            LEFT JOIN rposition rp USING (rnr)
            LEFT JOIN zimmer z ON
                      z.honr = rp.honr AND z.znr = rp.znr
GROUP BY g.gnr, g.name;
```

5 View als Zwischenspeicher

- Für welches Hotel wurde am häufigsten reserviert (nur Anzahl Reservierungen)?
  - 1. Schritt: Bestimmung der Anzahl der Reservierungen pro Hotel

```
CREATE VIEW anzahl_res (honr, anzahl) AS
SELECT rp.honr, COUNT(*)
FROM rposition rp
GROUP BY rp.honr;
```

  - 2. Schritt: Selektion des Hotels mit der maximalen Anzahl

```
SELECT a.honr, a.anzahl, h.hname
FROM anzahl_res a JOIN hotel h USING (honr)
WHERE a.anzahl IN (SELECT MAX(anzahl) FROM anzahl_res);
```

5 Indizierung

- Erstellung eines Suchindexes zum schnelleren Auffinden von Datensätzen. Nötig für Fremdschlüssel.

```
CREATE INDEX <indexname> ON <tabellenname> (spaltenname [,...]) ;
```

```
DROP INDEX <indexname>;
```

- Beispiel: Indizierung des Hotelnamens

```
CREATE INDEX ihotelname ON hotel (hname);
```

Bemerkungen zum Index:

- Kann auf mehrere Spalten angewendet werden.
- Erhöht den Speicherplatz, verlangsamt Updates
- Suchzeit **sinkt** von linear auf logarithmisch oder konstant!

5 Vergabe von Zugriffsrechten

```
GRANT {ALL PRIVILEGES | CREATE TABLE | ALTER | DROP | SELECT |
       INSERT | UPDATE | DELETE | EXECUTE} [, …]
ON <objektname>
TO {<username> | PUBLIC | <rollenname>}
[WITH GRANT OPTION];
```

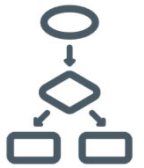
```
REVOKE {CREATE TABLE | ALTER | DROP | SELECT | INSERT | UPDATE |
        DELETE | EXECUTE} [, …]
ON <objektname>
FROM {<username> | PUBLIC | <rollenname>};
```

```
CREATE USER <username> IDENTIFIED BY Password('<password>');
```

```
CREATE ROLE <rollenname>;                    DROP ROLE <rollenname>;
```

- Objekt: Tabelle, View, Stored Procedure
- Grantoption: Weitergabe der Rechte
  Achtung: Revoke nicht kaskadierende Rechte-Zurücknahme!

5 Stored Procedures / Stored Functions



Stored Procedure
In Datenbank gespeicherte, vorkompilierte Routine, die vordefinierte Datenbankoperationen gemäß einer Ablaufsteuerung ausführt

- Nicht von allen DBMS unterstützt
- Stark plattformabhängig

5 Stored Procedures / Stored Functions

- Auslagerung oft verwendeter Befehlsfolgen in das DBMS
- Reduktion des Netzwerkverkehrs
- Erhöhung der Sicherheit durch Vermeidung direkter Zugriffe auf Datenbanktabellen
- Plattformunabhängig
- Wiederverwendbar in mehreren Programmen, da in DB implementiert und nicht im Programmcode
- Laufzeitverringerung durch Vorkompilierung
- Erweiterungen in Postgres: plpgsql

https://www.postgresql.org/docs/15/plpgsql.html

5 Stored Procedures – Syntax PostgreSQL

```
CREATE PROCEDURE <procedurename> (
   [[{IN | OUT | INOUT}] <parametername> DATENTYP][, …]
)
LANGUAGE SQL | PLPGSQL
BEGIN ATOMIC | AS $$
BEGIN
   <anweisung>;
      …
END; | END; $$
```

- IN: Ausschließlich Eingabeparameter (Default)
- OUT: Ausschließlich Ausgabeparameter
- INOUT: Sowohl Eingabe- wie auch Ausgabeparamete

5 Stored Procedures – Syntax MariaDB

```
CREATE PROCEDURE <procedurename> (
   [[{IN | OUT | INOUT}] <parametername> DATENTYP][, …]
)
BEGIN
   <anweisung>;
     …
END
```

- IN: Ausschließlich Eingabeparameter (Default)
- OUT: Ausschließlich Ausgabeparameter
- INOUT: Sowohl Eingabe- wie auch Ausgabeparameter

- Löschen:

```
DROP PROCEDURE <procedurename>;
```

- Aufruf:

```
CALL | EXEC | EXECUTE <procedurename> ();
```

5 Stored Procedure – Beispiel

```
CREATE PROCEDURE hotel_suche_name (IN name CHAR(50))
BEGIN
  SELECT * FROM hotel WHERE hname = name;
END
```

5 Stored Procedures – Trennzeichen

Problem

- Standardmäßiges Trennzeichen (Delimiter) von SQL-Befehlen ist das Semikolon
- Trennzeichen bei Stored Procedures ist ebenfalls das Semikolon

⇒ Temporärer Wechsel des Trennzeichens

```
DELIMITER //  oder SET TERM //
CREATE PROCEDURE hotel_suche_name (IN name CHAR(50))
BEGIN
  SELECT * FROM hotel WHERE hname = name;
END //
DELIMITER ;   oder SET TERM ;
```

5 Stored Procedures – Variablen & Anweisungen

- Variablendeklaration

```
DECLARE <variablenname> DATENTYP [DEFAULT WERT];
```

- Zuweisung PostGreSQL

```
SET <variablenname> := {<variablenname> | <ausdruck>};
```

- Zuweisung MariaDB

```
SET <variablenname> = {<variablenname> | <ausdruck>};
```

- Ausgabe

```
SELECT <ausgabe>;
```

Stored Procedures – Kontrollstrukturen

- Verzweigung

```
IF <bedingung>
  THEN
  <anweisung>;
   …
[[ELSE IF <bedingung>
  THEN
   <anweisung>;
     …]
 [ELSE
   <anweisung>;
     …]]
END IF;
```

Stored Procedures – Kontrollstrukturen

- Simple Case

```
CASE <variable>
  WHEN WERT THEN <anweisung>; …
  …
 [ELSE <anweisung>; …]
END CASE;
```

- Searched Case

```
CASE
  WHEN <bedingung> THEN <anweisung>; …
  …
 [ELSE <anweisung>; …]
END CASE;
```

Stored Procedures – Kontrollstrukturen

- LOOP

```
<loopname>: LOOP
    IF <bedingung> THEN LEAVE <loopname>; END IF;
    <anweisung>; …
END LOOP <loopname>;
```

- WHILE

```
WHILE <bedingung> DO
    <anweisung>; …
END WHILE;
```

- REPEAT

```
REPEAT
    <anweisung>; …
UNTIL <bedingung wahr>
END REPEAT;
```

5 Stored Procedures – Datensätze auslesen

- Variante 1

```
SELECT <attributname> [, …] INTO <variablenname> [, …]
FROM … ;
```

  - Anzahl Attribute und Anzahl Variablen müssen gleich sein
  - Select-Befehl darf nur einen Datensatz zurückgeben

- Variante 2

```
SELECT @<variablenname> := <attributname> [, …]
FROM … ;
```

  - Variable erhält den letzten Datensatz des Select-Befehls

5 Stored Procedures – Datensätze auslesen

- Problem: Select liefert i.d.R. mehrere Datensätze

⇒ CURSOR

```
DECLARE <cursorname> CURSOR FOR SELECT …;
```

```
OPEN <cursorname>;
```

```
CLOSE <cursorname>;
```

- Zugriff auf nächsten Datensatz

```
FETCH <cursorname> INTO <variablenname> [, …];
```

- Herausfinden, ob noch Datensatz vorhanden

```
IF FOUND THEN
```

5 Stored Procedures – Datensätze auslesen

- Problem:
  Nach letztem Datensatz wird bei FETCH ein Fehler geworfen

⇒ Error HANDLER

```
DECLARE CONTINUE HANDLER FOR {NOT FOUND | SQLSTATE '02000'}
<anweisung>;
```

5 Stored Procedures – Beispiel PostgreSQL

```
CREATE PROCEDURE gaestesichern ()
LANGUAGE plpgsql
AS $$
DECLARE
  gastname CHAR(50);
  gastnr INTEGER;
  finished BOOLEAN DEFAULT FALSE;
  gastcursor CURSOR FOR SELECT name, gnr FROM gast;
BEGIN
  OPEN gastcursor;
  WHILE NOT finished LOOP
    FETCH gastcursor INTO gastname, gastnr;
    IF FOUND THEN
       INSERT INTO altgaeste (gnr, gname)
                    VALUES (gastnr, gastname);
    ELSE finished := TRUE;
    END IF;
  END LOOP;
  CLOSE gastcursor;
END $$;
```

5 Stored Procedures – Beispiel MySQL

```
DELIMITER //
CREATE PROCEDURE gaestesichern ()
BEGIN
  DECLARE gastname CHAR(50);
  DECLARE gastnr INTEGER;
  DECLARE finished BOOLEAN DEFAULT FALSE;
  DECLARE gastcursor CURSOR FOR SELECT name, gnr FROM gast;
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET finished = TRUE;

  OPEN gastcursor;
  REPEAT
    FETCH gastcursor INTO gastname, gastnr;
    IF NOT finished THEN
       INSERT INTO altgaeste (gnr, gname)
                   VALUES (gastnr, gastname);
    END IF;
  UNTIL finished END REPEAT;
  CLOSE gastcursor;
END //
DELIMITER ;
```

5 Stored Functions

```
CREATE FUNCTION <functionname> (
  [[{IN | OUT | INOUT}] <parametername> DATENTYP][, …]
) RETURNS (<parametername> DATENTYP][, …])
LANGUAGE plpgsql AS $$
DECLARE <variablenname> DATENTYP [DEFAULT WERT];
BEGIN
  <anweisung>;
    …
 RETURN <rückgabewerte>;
END; $$
```

- Für Rückgabe einer Tabelle: RETURNS TABLE (…)

```
SELECT <functionname> (<parameterliste>);
```

5 Trigger

Trigger
Spezielle Stored Procedure, die durch bestimmte Datenbankoperationen/Ereignisse ausgelöst wird.

Ereignisse

- DML: DELETE, INSERT, UPDATE
- DDL: CREATE, ALTER, DROP
- DB-Operationen: SERVERERROR, LOGON, LOGOFF, STARTUP, SHUTDOWN

5 Trigger

- Umsetzung von Business Rules und komplexeren Konsistenzregeln
- Intelligenz in der Datenbank, nicht im Programm
  - Auslösung nur an einer Stelle
  - Reduktion der Netzlast
- Komplexere Fehlersuche

5 Trigger – Syntax

```
CREATE TRIGGER <triggername>
{BEFORE | AFTER}
{INSERT | UPDATE | DELETE}
ON <tabellenname>
FOR EACH ROW
BEGIN
<anweisung>;
…
END
```

- BEFORE: Ausführung vor der SQL-Anweisung
- AFTER: Ausführung nach der SQL-Anweisung

```
DROP TRIGGER <triggername>;
```

5 Trigger – Zugriff auf Datensatzwerte

- OLD: Alter Wert
- NEW: Neuer Wert

- Zuweisung zu einer Variablen

```
SET @<variablenname> = {NEW | OLD}.<spaltenname>;
```

- Änderung neuer Werte in BEFORE-Trigger möglich

```
SET NEW.<spaltenname> = <ausdruck>;
```

5 Trigger – Verfügbarkeit von Datensatzwerten

UPDATE *Spalte name: von Müller nach Huber*		INSERT *Spalte name: Huber*		DELETE *Spalte name: Müller*	
BEFORE	**AFTER**	**BEFORE**	**AFTER**	**BEFORE**	**AFTER**
Verfügbarkeit des alten Wertes					
OLD.name ***Müller***	–	–	–	**OLD.name** ***Müller***	**OLD.name** ***Müller***
Verfügbarkeit des neuen Wertes					
NEW.name ***Huber***	**NEW.name** ***Huber***	**NEW.name** ***Huber***	**NEW.name** ***Huber***	–	–

5 Trigger – PostgreSQL

```
CREATE OR REPLACE TRIGGER <TriggerName> AFTER
INSERT OR
UPDATE OR
DELETE ON <Tabelle>
FOR EACH ROW EXECUTE FUNCTION <FunktionsName>();
```

```
CREATE OR REPLACE FUNCTION <FunktionsName> ()
RETURNS TRIGGER AS $$
DECLARE
   <variable> integer = 5;
BEGIN
      <Anweisungen>
      RETURN NULL;
END;
$$ LANGUAGE PLPGSQL;
```

```
-- Test auf Triggeroperation
IF (TG_OP = DELETE) THEN
   OLD…
ELSIF (TG_OP = 'UPDATE) THEN
   NEW…
ELSIF (TG_OP = 'INSERT) THEN
   NEW…
END IF;
```

https://www.postgresql.org/docs/current/plpgsql-trigger.html

5 Trigger – Beispiel PostgreSQL

```
CREATE FUNCTION gastsichern() RETURNS TRIGGER
LANGUAGE plpgsql AS $$
BEGIN
  INSERT INTO altgaeste (gnr, gname) VALUES (old.gnr, old.name);
  RETURN new;
END; $$
```

```
CREATE TRIGGER gaestesichern
BEFORE DELETE ON gast
FOR EACH ROW
EXECUTE FUNCTION gastsichern();
```

5 Trigger – Beispiel MySQL

```
CREATE TRIGGER gaestesichern
BEFORE DELETE ON gast FOR EACH ROW
EXECUTE FUNCTION
  INSERT INTO altgaeste (gnr, gname)
                    VALUES (old.gnr, old.name);
```

5 SQL – Übungen

Gegeben sind folgende Tabellen:

Person

Personalnr	Name	Gehalt	Ort	Abteilung
100	Maier	3800.00 €	Künzelsau	1
101	Müller	4100.00 €	Künzelsau	2
102	Schmidt	3500.00 €	Öhringen	1
103	Schulze	4600.00 €	Heilbronn	3
104	Hinz	4200.00 €	Heilbronn	3
105	Kunz	4000.00 €	Schwäbisch Hall	5

Abteilung

Abteilungsnr	Abteilungsname
1	EDV
2	Produktion
3	Vertrieb
4	Verwaltung
5	Entwicklung

5 SQL – Übungen

1) Geben Sie für folgende SQL-Abfragen an, welche Tabelle als Ergebnis erzeugt wird (Tabelle mit Überschriften):

a)
```
SELECT DISTINCT Ort
    FROM Person
```

b)
```
SELECT Name
    FROM Person
    WHERE Abteilung = 1
    ORDER BY Name DESC
```

c)
```
SELECT Name, Abteilungsname
    FROM Person, Abteilung
    WHERE Abteilung=Abteilungsnr
```

5 SQL – Übungen

2) Geben Sie zu den folgenden, verbal formulierten Abfragen die passende SQL-Abfrage an. Die Abfragen müssen unabhängig vom konkreten Inhalt der Datenbank gültig sein!

   a) Liste aller Personen mit Name und Ort.

   b) Liste aller Personen mit einem Gehalt von mehr als 4.000,-- € mit Name und Gehalt, sortiert nach fallendem Gehalt (höchstes Gehalt zuerst).

5 SQL – Übungen

2) Geben Sie zu den folgenden, verbal formulierten Abfragen die passende SQL-Abfrage an. Die Abfragen müssen unabhängig vom konkreten Inhalt der Datenbank gültig sein!

c) Liste aller Orte mit der zusätzlichen Angabe, wie viele Personen von dort kommen.

d) Liste aller Abteilungen mit Abteilungsnamen und der Gehaltssumme (Summe über alle Gehälter der Personen dieser Abteilung). Der String "Gehaltssumme" soll als Überschrift der Spalte vorkommen.

5 SQL – Übungen

Die Tabellen unten sind für folgenden Use-Case:
Ein Fußballverband hat eine Datenbank mit den bei ihm gemeldeten Vereinen und Spielern. Alle Spieler haben eine vereinsübergreifend einmalige Pass-Nr. und können pro Saison (Attribut ist vom Typ String) nur bei einem Verein gemeldet sein.

Spieler

Pass_Nr	Name	Geburtsjahr
1	Peter Maier	1960
2	Klaus Müller	1980
3	Karl Hinz	1987
4	Thomas Kunze	1975

Verein

V_Nr	Name	Ort
1	Schlappen-Kicker 05	Heilbronn
2	Knochen-Knacker 09	Künzelsau
3	Ballermann 1955	Heilbronn
4	Alte Herren 1880	Schwäbisch Hall

Meldung

Spieler	Saison	Verein	Einsätze	Tore
1	00/01	1	6	3
2	00/01	3	12	1
3	00/01	2	1	0
1	99/00	2	6	4
4	00/01	1	2	1

5 SQL – Übungen

3) Geben Sie für folgende SQL-Abfragen an, welche Tabelle als Ergebnis erzeugt wird (Tabelle mit Überschriften):

i)
```
SELECT Verein.Name
    FROM Meldung, Verein, Spieler
    WHERE Verein=V_Nr AND
        Spieler=Pass_Nr AND
        Saison="00/01" AND
        Geburtsjahr>1962
    ORDER BY Verein.Name DESC
```

ii)
```
SELECT COUNT(*)
    FROM Meldung
    WHERE Saison="00/01"
```

5 SQL – Übungen

4) Geben Sie zu den folgenden, verbal formulierten Abfragen die passende SQL-Abfrage an. Die Abfragen müssen unabhängig vom konkreten Inhalt der Datenbank gültig sein.

   i) Liste aller Heilbronner Vereine mit Nr. und Namen, sortiert nach Nr.

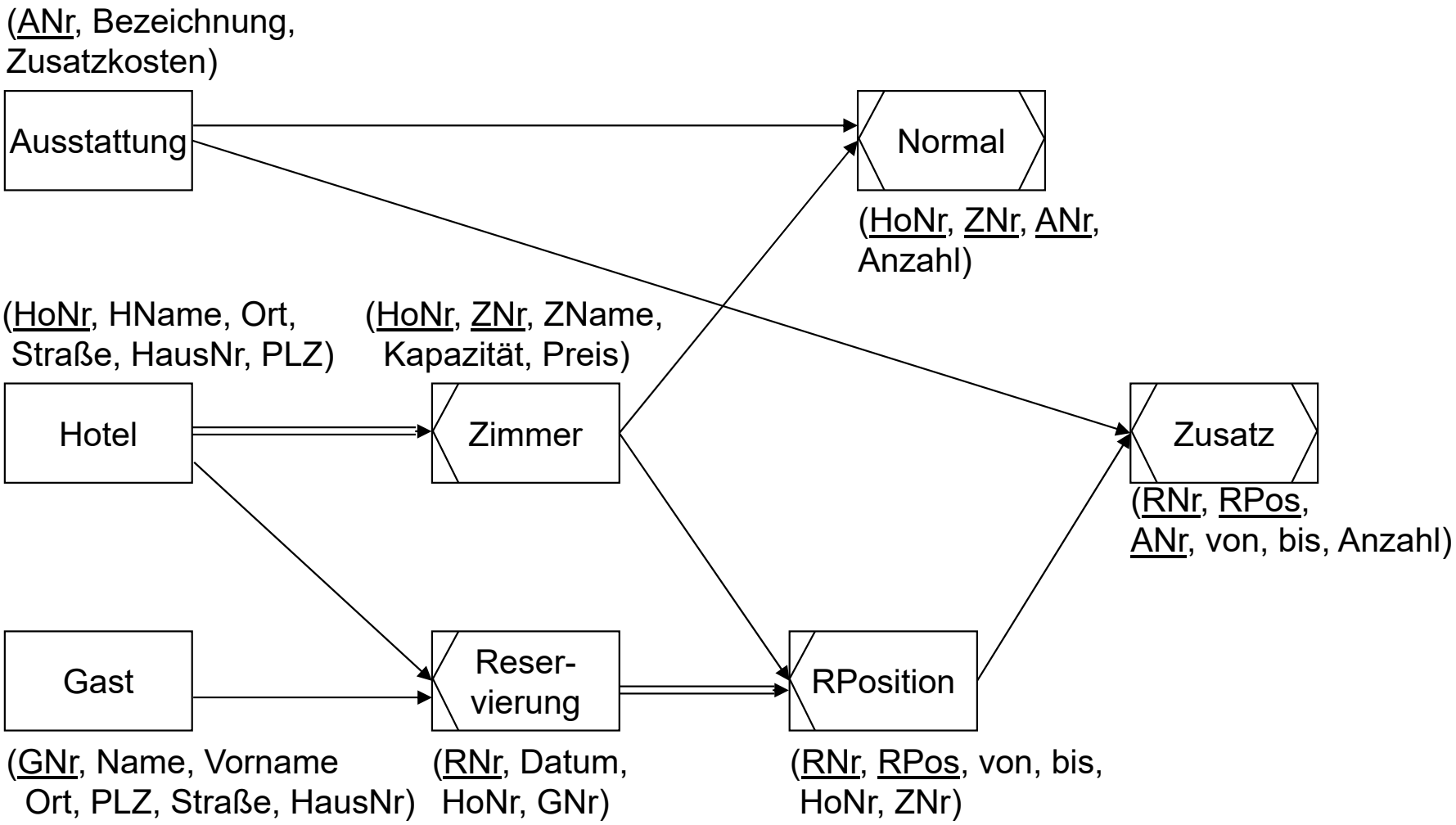
   ii) Liste aller Spieler mit Pass-Nr. und Namen, die in der Saison 00/01 beim Verein "Knochen-Knacker 09" gespielt haben

   iii) Liste mit dem oder den (ev. auch mehrere) Torschützen-Königen der Saison 00/01 mit Namen und Anzahl der Toren

   iv) Liste der Vereine mit Nr. und Namen sowie der Anzahl dort in der Saison 00/01 gemeldeten Spieler

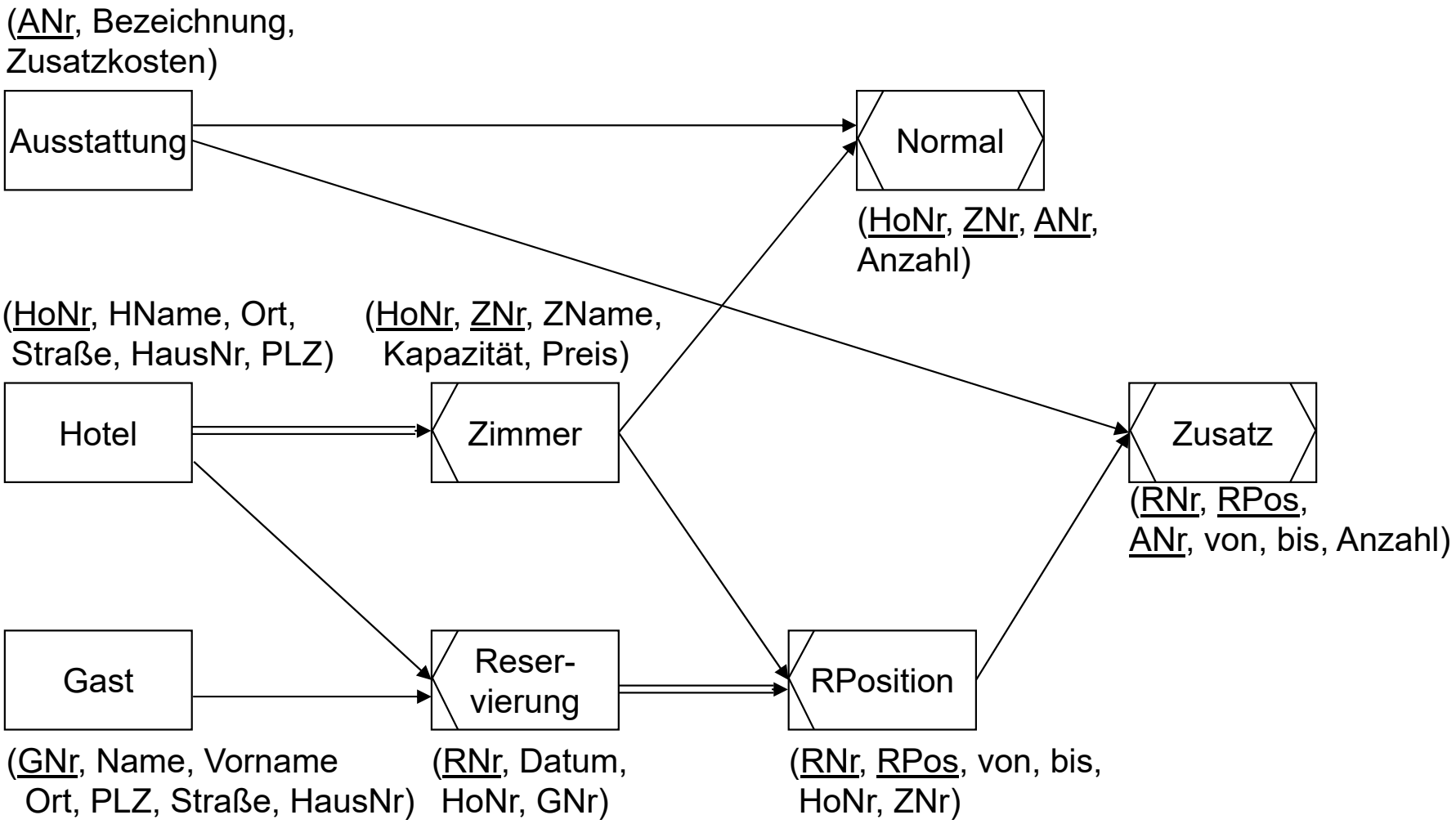
5 Aufgaben – Stored Procedures

- Prozedur zum Anlegen eines Zimmers mit allen Attributen über den Hotelnamen



5 Aufgaben – Stored Procedures

- Prozedur zum Anpassen der Mehrwertsteuer bei Zimmer- und Ausstattungspreisen. Eingabe ist alte und neue MwSt.



5 Aufgaben – Stored Procedures

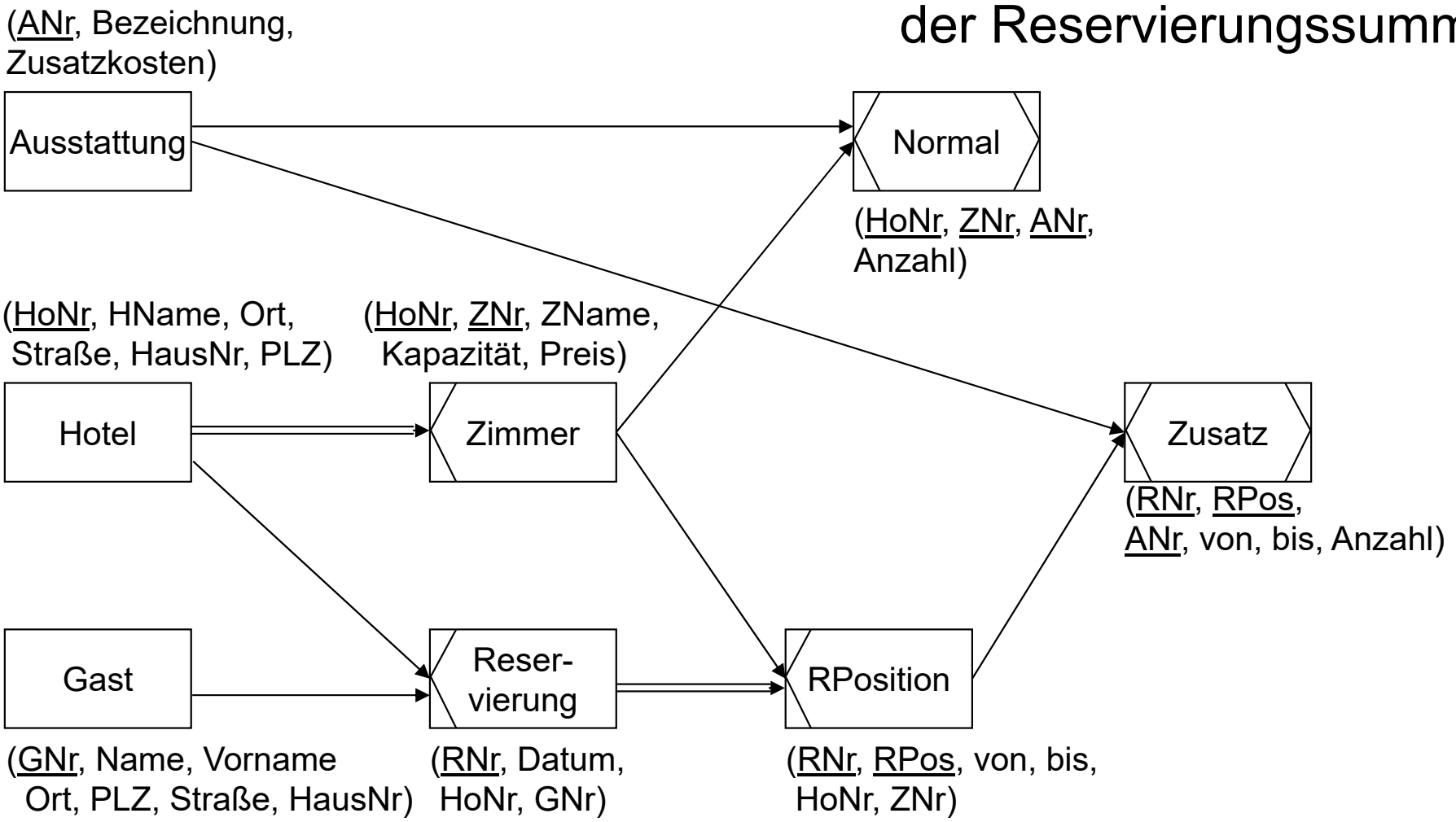
- Prozedur zur Erstellung einer Tabelle, in der eine Maschine linear abgeschrieben wird
  Beispiel:
  - Wert 21.000 Euro
  - Dauer: 7 Jahre

  ⇒Rate: 21.000 Euro / 7 Jahre = 3.000 Euro/Jahr

Jahr	Rate	Restwert
0	0	21.000
1	3.000	18.000
2	3.000	15.000
3	3.000	12.000
4	3.000	9.000
5	3.000	6.000
6	3.000	3.000
7	3.000	0

5 Aufgaben – Trigger

- Verwaltung einer Reservierungssumme in Reservierung nach jeder neuen Rechnungsposition. Vorab Erstellung/Initialisierung der Reservierungssumme.



5 Metadaten

Metadaten (Data Dictionary)
Informationen über die Struktur einer Datenbank sowie der darin enthaltenen Daten

5 Metadaten

- Tabellen
  - Tabellenname
  - Attribute & Typen
  - Constraints
  - Trigger
  - Anzahl Datensätze
  - Indizes
- Views
- Benutzer & Accountinformation

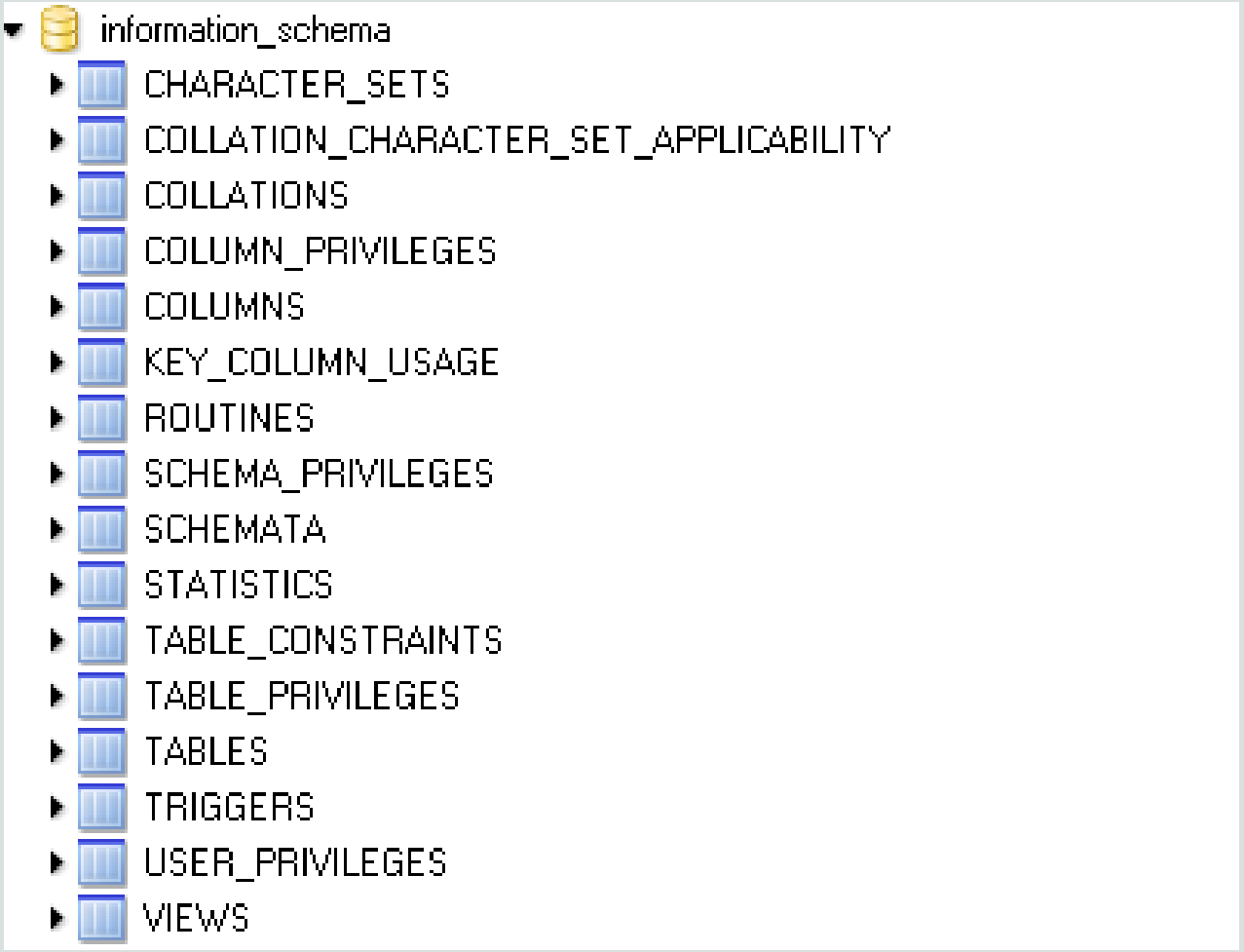
5 Metadaten – Auswahl (Oracle)

- SYS.USER_OBJECTS
  all objects owned by you, where objects include: tables, views, indexes…
- SYS.TAB
  views & tables owned by you
- SYS.USER_TABLES
  tables owned by you
- SYS.USER_VIEWS
  views owned by you
- SYS.ALL_TABLES
  tables that you have permission to access whether your own, or someone else's
- SYS.USER_TAB_COLUMNS
  columns that each table has
- SYS.USER_CONSTRAINTS
  constraints on tables created (owned) by you
- SYS.USER_TRIGGERS
  triggers owned by you
- SYS.USER_CATALOG
  similar to SYS.TAB, with a simpler structure

5 Metadaten – Systematik (Oracle)

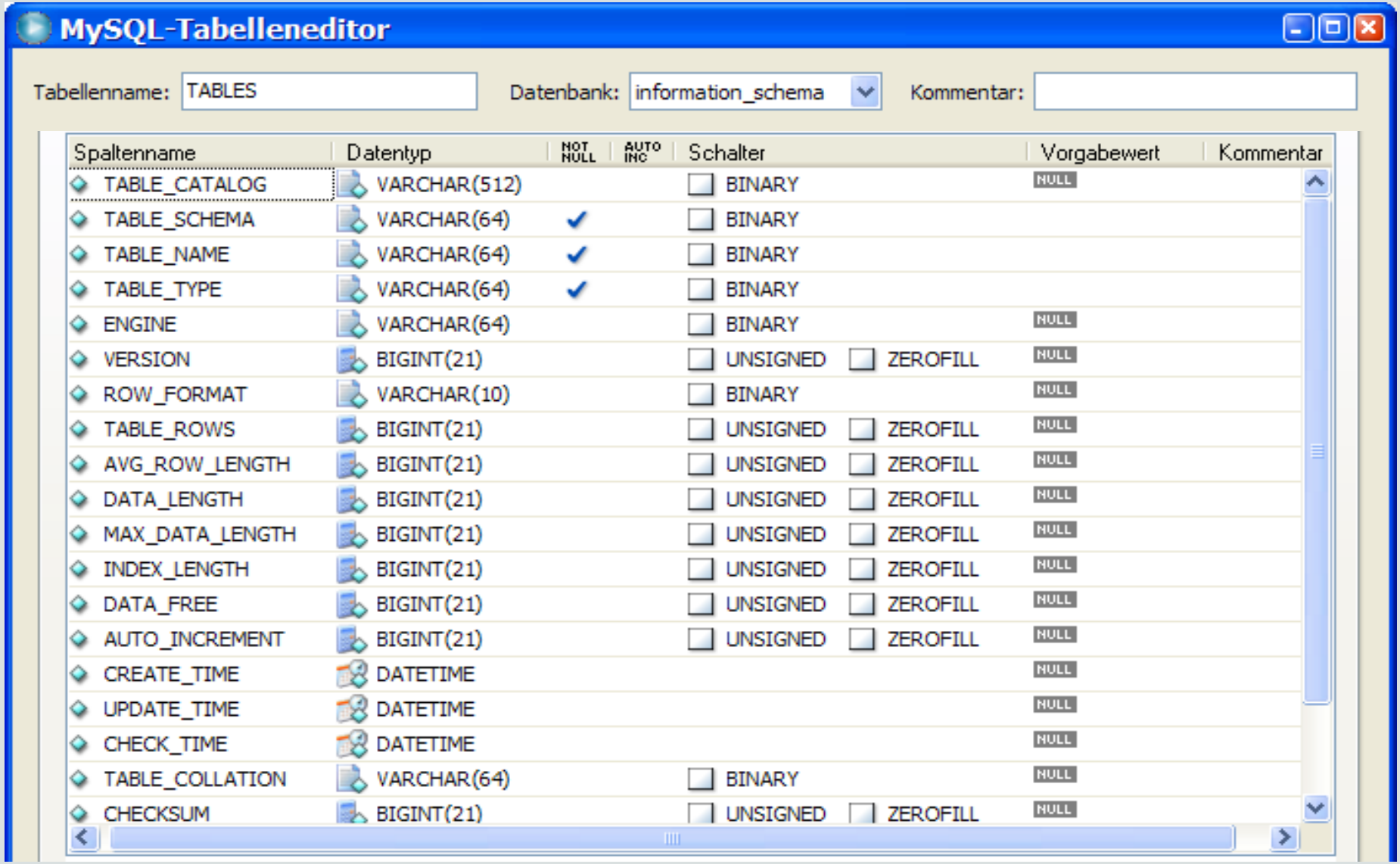
- **USER_**
  Informationen über DB-Objekte, deren Owner des User ist
- **ALL_**
  Informationen über DB-Objekte, auf die ein User Zugriff hat, auch wenn er nicht der Owner ist
- **DBSA_**
  Nur für Administratoren zugängliche Informationen über alle Objekte, unabhängig von Owner-Status und Zugriffsrechten

5 Metadaten – MySQL-Datenbank-Katalog

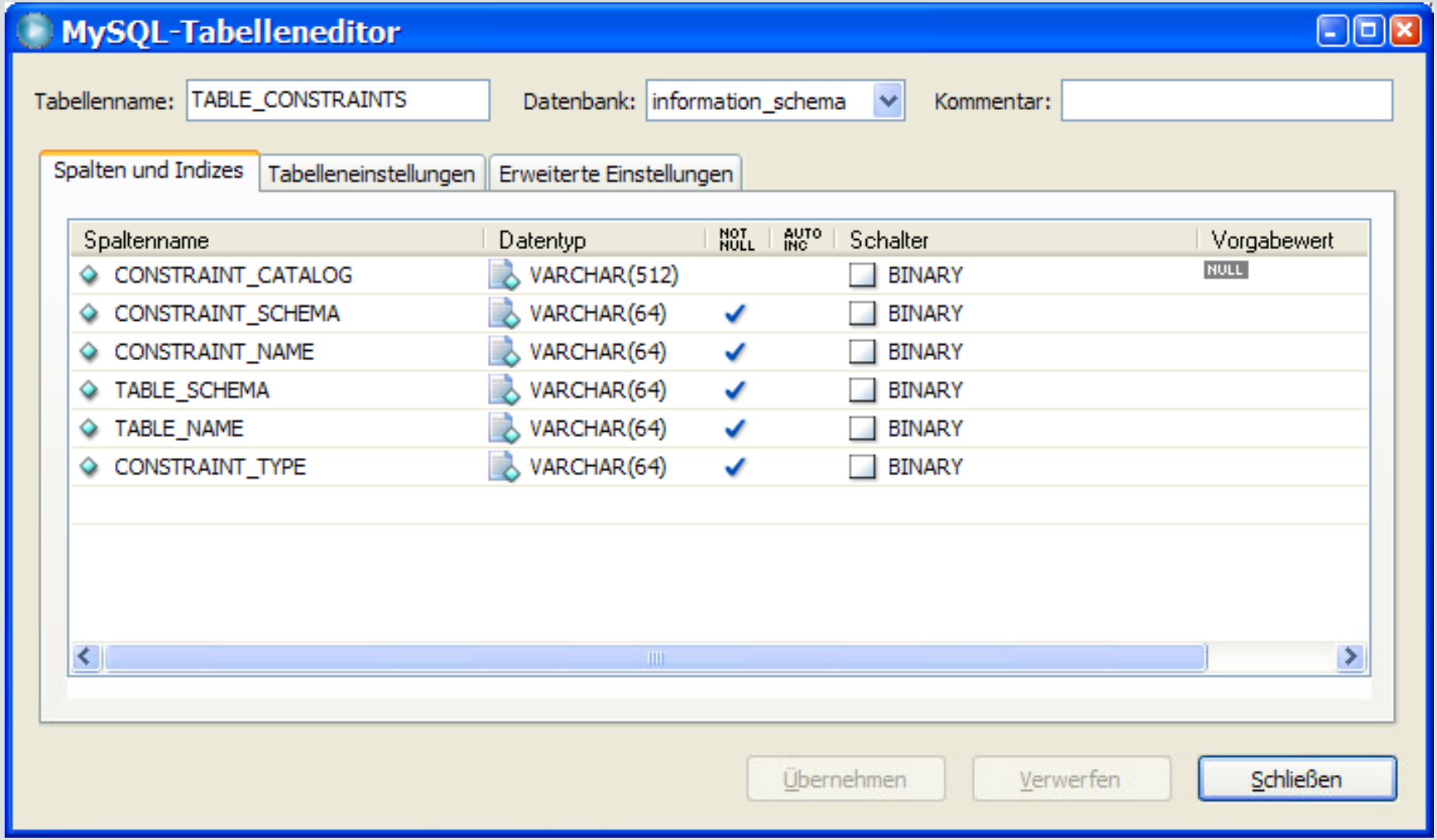


Die Katalogtabelle befinden sich im Schema *information_schema*

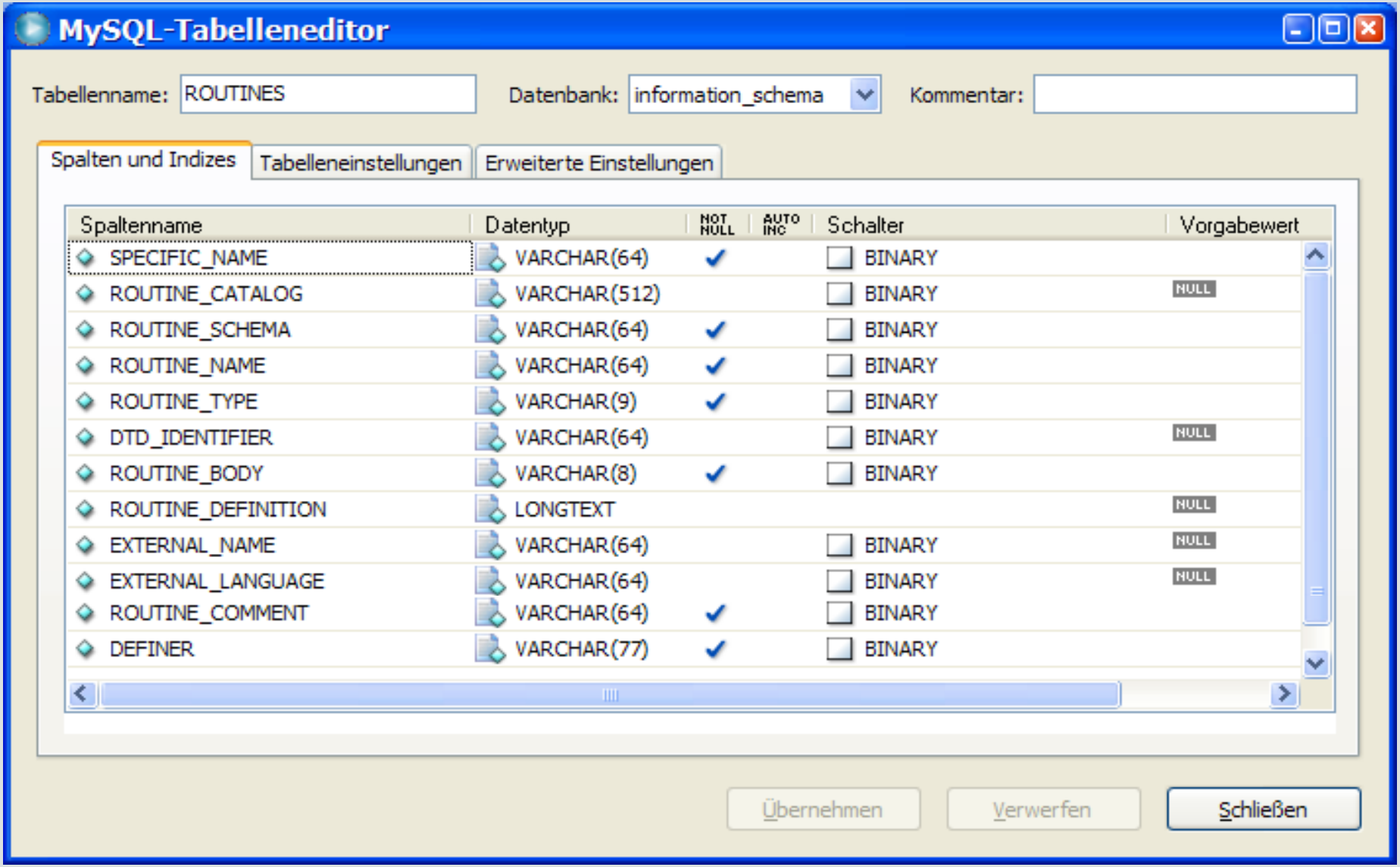
5 Metadaten – MySQL Tables



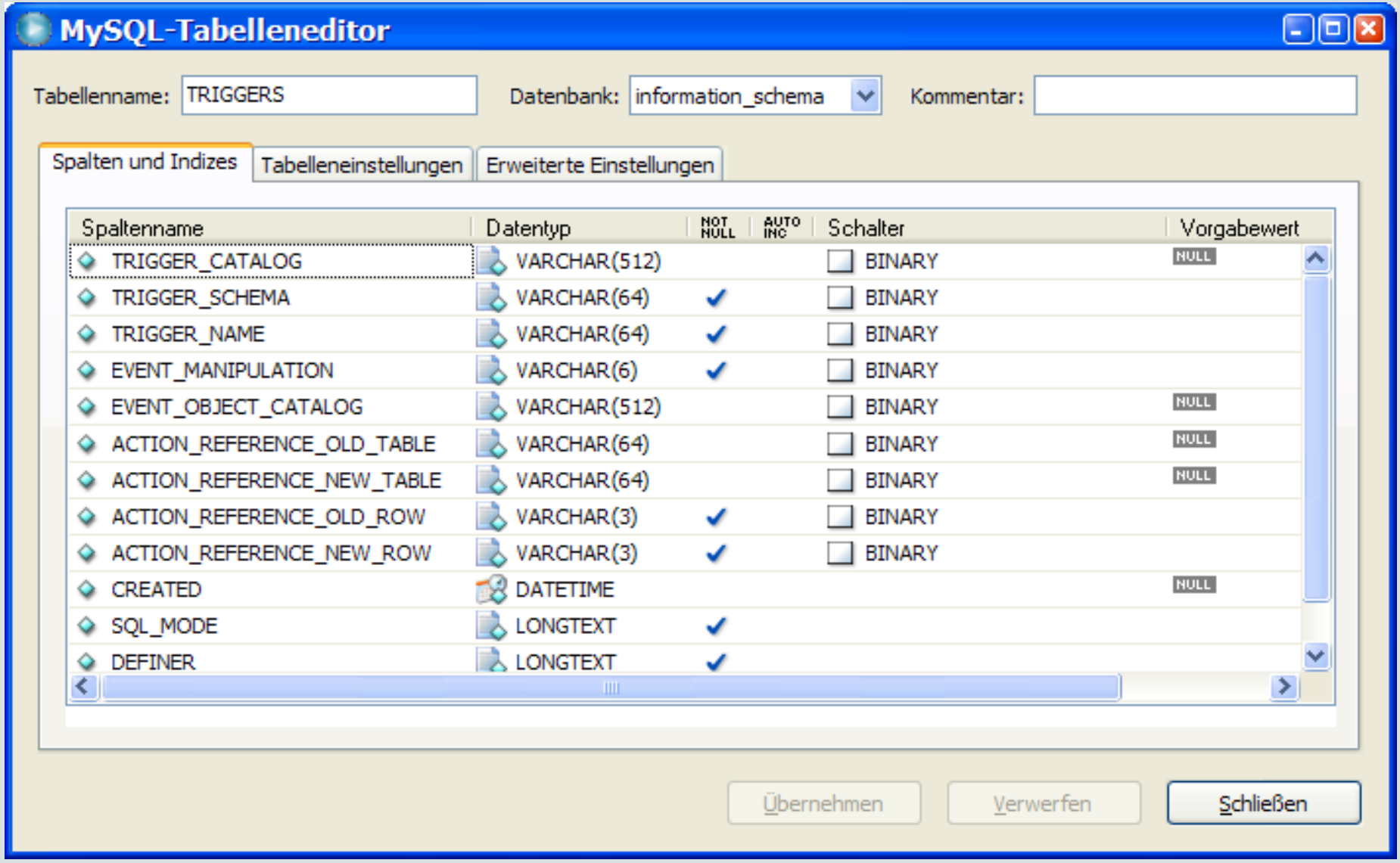
5 Metadaten – MySQL Table_Constraints



5 Metadaten – MySQL Routines



5 Metadaten – MySQL Triggers



5 SQL-Schnittstellentypen

- Kommando-orientierte interaktive SQL-Ausführung
  z.B. ORACLE – SQL*Plus, SQL-Worksheet bzw. mysql-Kommandozeilen-Tool
  - Dialoggesteuerter Zugriff auf die Datenbank
  - Voraussetzung ist ein erfahrener Nutzer
  - Anreichungsmöglichkeit von SQL-Befehlen durch Datenbankherstellabhängige Befehle (SQL*Plus-Befehle)
- Grafisch unterstützte SQL-Ausführung
  z.B. ORACLE 8i: Schema-Builder, Query-Builder, Mysql Workbench, phpmyadmin, MS ACCESS

5 SQL-Schnittstellentypen

- Embedded SQL
  - Einbettung von SQL-Befehlen in eine 3GL-Programmiersprache.
  - Bereitstellung spezieller Erweiterungen wie z.B. Definition, Öffnen, Schließen von CURSOR oder satzweises Übergeben von Datensätzen.
  - z.B. C, C++, ORACLE PL/SQL, MySQL Improved Extension (MySQLi)
- ODBC (Open Database Connectivity)
  - Middleware zur Entwicklung von Datenbankanwendungen im Client/Server-Umfeld.
  - Bereitstellung von Funktionen über API-Schnittstelle („Application Program Interface“)
    - Initialisierung
    - Verbindungsaufbau- und -abbau zum DBMS
    - Übertragung von SQL-Kommandos
    - Auswertung der Antworten

5 Kontrollfragen

- Geben Sie die Definition einer Relation an.
- Welche Operatoren der Relationenalgebra existieren? Erläutern Sie diese jeweils an einem Beispiel.
- Welchen Unterschied gibt es zwischen einem prozeduralen und einem deskriptiven Programm? Ordnen Sie SQL passend ein.
- Wie lautet der SQL-Befehl zum Erstellen einer Datenbank?
- Wie lauten die SQL-Befehle zum Erstellen, Ändern und Löschen von Tabellen?
- Geben Sie den Unterschied zwischen einer Restriktion und einer Projektion an.
- Wie lautet die prinzipielle Syntax einer Abfrage (Query) in SQL?
- In welchem Teil einer SQL-Anweisung wird die Restriktion bzw. die Projektion realisiert?

5 Kontrollfragen

- Was versteht man unter dem kartesischen Produkt zweier Tabellen? Geben Sie ein Beispiel anhand einer SQL-Anweisung und exemplarischer Tabelleneinträge an.
- Wie kann erreicht werden, dass nur tatsächlich bestehende Verbindungen angezeigt werden, d.h. eine Selektion auf das Kreuzprodukt stattfindet?
- Erläutern Sie die Begriffe „Natural Join“, „Equi-Join“, „Inner Join“, „Outer Join“, „Left Join“ und „Right Join“.
- Wann sollte ein „Group By“ – Teil in der SQL-Anweisung integriert werden?
- Sie möchten nicht eine gesamte Tabelle, sondern einzelne Tupel in einer Tabelle löschen. Wie lautet die allgemeine Syntax der SQL-Anweisung?
- Wie ändern und fügen Sie neue Datensätze in eine Tabelle ein? Geben Sie die allgemeine Syntax an.