Hochschule RheinMain

Hochschule RheinMain

SQL-WIEDERHOLUNG UND NEUES

Datenauslese und –modifikation
Datenbankbeschreibung
Funktionen, Prozeduren, Rekursion

Prof. Dr. Eva-Maria Iwer
SS2022

Hochschule RheinMain

AUSLESEN VON DATEN

Übersicht der wichtigsten SELECT Befehle

DIE SYNTAX DES SELECT BEFEHLS

Hochschule RheinMain

```
Select-Hauptteil
   [  { UNION [ ALL ] | EXCEPT | INTERSECT }
Select-Hauptteil                                  ]
                                                    [ ... ]
[ ORDER BY Ordnungsliste ]
```

DIE SYNTAX DES SELECT BEFEHLS

Hauptteil

Hochschule RheinMain

```
SELECT    [ ALL | DISTINCT ] Spaltenauswahlliste
 FROM     Tabellenliste
 [ WHERE          Bedingung ]
 [ GROUP BY       Spaltenliste
    [ HAVING      Bedingung ] ]
```

DIE SYNTAX DES SELECT BEFEHLS

Die FROM Klausel

Hochschule RheinMain

Tabellenliste:

Tabellenreferenz [, ...]

Tabellenreferenz:

Tabellenname [[AS] Aliasname]

| (Select-Hauptteil) [[AS] Aliasname]

| (Tabellenreferenz) [[AS] Aliasname]

| Joinausdruck [[AS] Aliasname]

DIE SYNTAX DES SELECT BEFEHLS

Aggregatfunktionen

Hochschule RheinMain

AVG	Average	Mittelwert, ermittelt über alle Zeilen
COUNT	Count	Anzahl aller Zeilen
MAX	Maximum	Maximalwert aller Zeilen
MIN	Minimum	Minimalwert aller Zeilen
SUM	Sum	Summenwert, summiert über alle Zeilen

DIE SYNTAX DES SELECT BEFEHLS

Die Where Klausel

Hochschule **RheinMain**

```
SELECT    MIN( Gehalt )
 FROM     Personal
 WHERE    Gehalt > 3000 ;
```

Kleinstes Gehalt größer 3000

nicht: !=

Boolesche Operatoren	NOT , AND , OR
Vergleichsoperatoren	< , <= , > , >= , = , <>
Intervalloperator	[NOT] BETWEEN ... AND
Enthaltenoperator	[NOT] IN
Ähnlichkeitsoperator	[NOT] LIKE
Nulloperator	IS [NOT] NULL
Auswahloperatoren	ALL , ANY , SOME
Existenzoperator	EXISTS

DIE SYNTAX DES SELECT BEFEHLS

Like

Hochschule RheinMain

Wildcardsymbole	in SQL
Beliebig viele Zeichen (0..n)	%
Genau ein Zeichen (1..1)	_

DIE SYNTAX DES SELECT BEFEHLS

Die Group by Klausel

Hochschule RheinMain

- Beachten!!
  - Es gibt Aggregationsoperatoren und Gruppierungsattribute.
  - In der Select-Klausel dürfen nur Aggregationsoperatoren und Gruppierungsattribute stehen.

- Having ist eine Restriktion nach der Gruppierung

```
SELECT      Ort, COUNT (*) As Anzahl,
            SUM(12*Gehalt) AS Jahresgehalt,
            12 * MAX(Gehalt) AS MaxJahresgehalt
FROM        Personal
GROUP BY    Ort
HAVING      COUNT(*) > 1 ;
```

DIE SYNTAX DES SELECT BEFEHLS

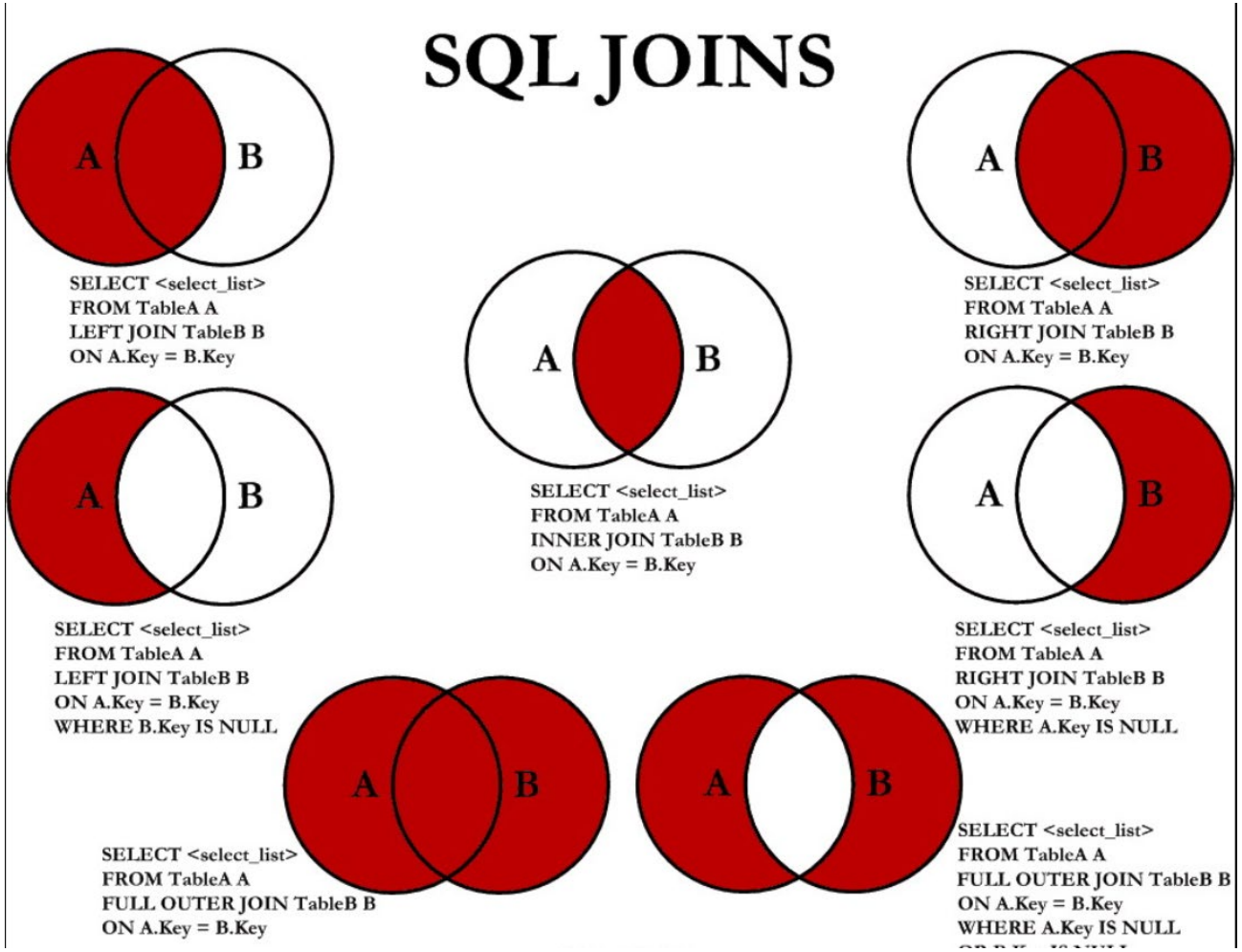
Join

Hochschule RheinMain

```
Tabellenreferenz
{  [ NATURAL ]  [ INNER ]
|  [ NATURAL ] {  LEFT  |  RIGHT  |  FULL } [ OUTER ]
}   JOIN Tabellenreferenz
          [ ON  Bedingung  |  USING ( Spaltenliste ) ]
```

SQL JOINS

Hochschule RheinMain



DIE SYNTAX DES SELECT BEFEHLS

Order by Klausel

Hochschule RheinMain

```
SELECT    Ort, COUNT (*) AS Anzahl
FROM      Personal
GROUP BY  Ort
ORDER BY  Anzahl DESC, Ort ;
```

Absteigendes Sortieren

Nur Namen sind erlaubt …

- oder:

```
SELECT    Ort, COUNT (*) AS Anzahl
FROM      Personal
GROUP BY  Ort
ORDER BY  2 DESC, 1 ;
```

… oder Zahlen

Hochschule RheinMain

MODIFIKATIONEN VON DATENBANKEN

Übersicht der wichtigsten Befehle

ÜBERSICHT DER BEFEHLE

Hochschule RheinMain

UPDATE	Ändert bestehende Einträge
INSERT	Fügt neue Tupel (Zeilen) ein
DELETE	Löscht bestehende Tupel (Zeilen)

ÜBERSICHT DER BEFEHLE
DELETE

Hochschule RheinMain

DELETE FROM Tabellenname [[AS] Aliasname]
[WHERE Bedingung]

ÜBERSICHT DER BEFEHLE
UPDATE

Hochschule RheinMain

```
UPDATE   Tabellenname  [ [AS] Aliasname ]
 SET     Spalte = Spaltenausdruck  [ , ... ]
 [WHERE  Bedingung ]
```

ÜBERSICHT DER BEFEHLE
INSERT

Hochschule RheinMain

```
INSERT INTO Tabellenname [ ( Spaltenliste ) ]
  { VALUES ( Auswahlliste ) [, … ]
  | Select-Befehl }
```

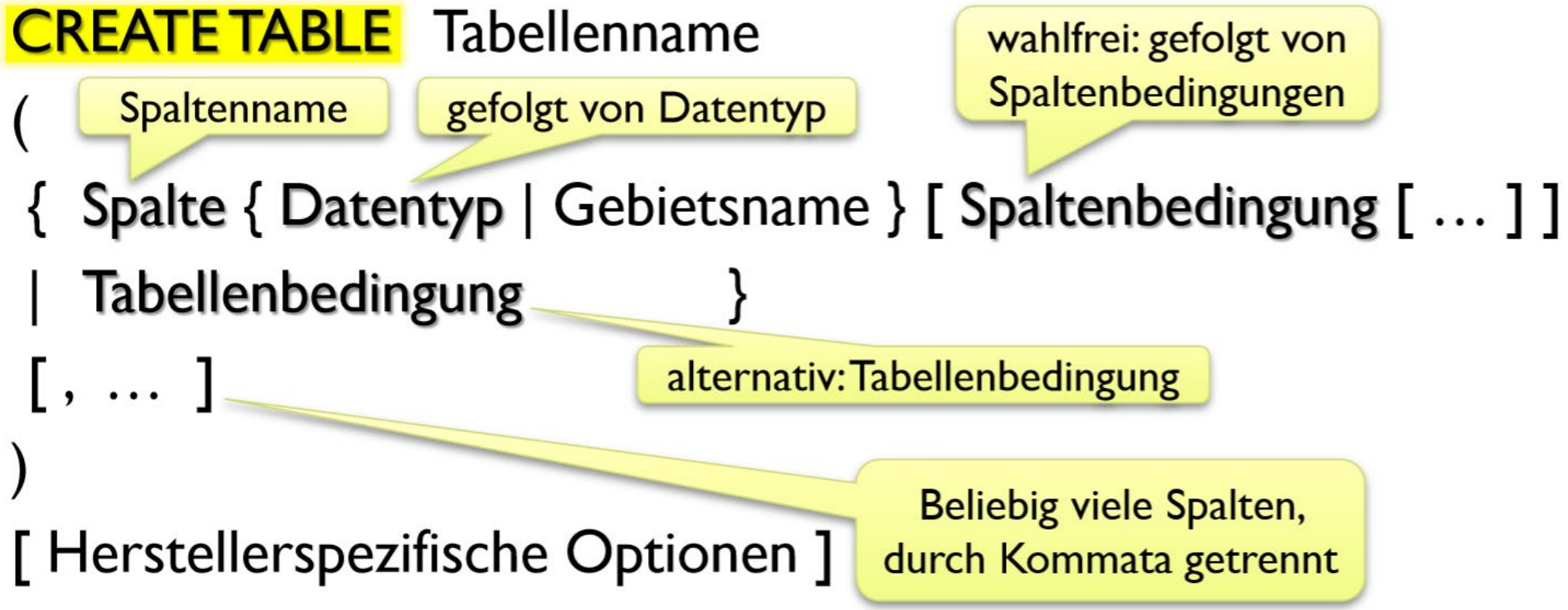
Hochschule RheinMain

ERSTELLEN VON RELATIONEN

Relationen erstellen
Rechte vergeben
Trigger erstellen

ÜBERSICHT DER BEFEHLE
CREATE TABLE

Hochschule RheinMain



ÜBERSICHT DER BEFEHLE

Datentypen

Hochschule **RheinMain**

INTEGER **INT**	Ganzzahl
SMALLINT	Ganzzahl
NUMERIC(x,y)	x stellige Dezimalzahl mit y Nachkommastellen
DECIMAL(x,y)	x stellige Dezimalzahl mit y Nachkommastellen
FLOAT(x)	Gleitpunktzahl mit x Nachkommastellen
CHARACTER(n) **CHAR(n)**	Zeichenkette der festen Länge n
CHARACTER VARYING(n) **VARCHAR(n)**	Variable Zeichenkette mit bis zu n Zeichen
BIT(n)	Bitleiste der festen Länge n
DATE	Datum (Jahr, Monat, Tag)
TIME	Uhrzeit (Stunde, Minute, Sekunde)
DATETIME	Kombination aus DATE und TIME

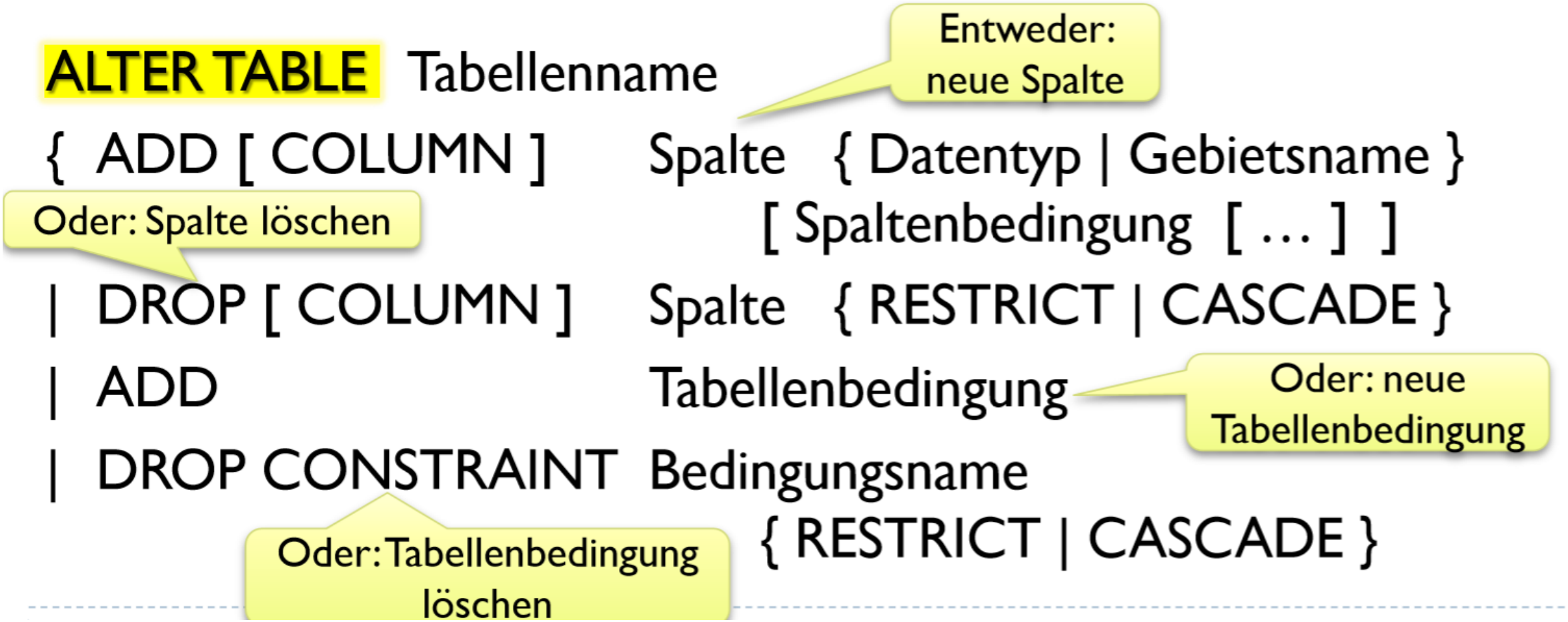
ÜBERSICHT DER BEFEHLE
Spaltenbedingungen

Hochschule RheinMain

- NOT NULL
- { PRIMARY KEY | UNIQUE }
- REFERENCES Tabellenname [(Spalte [, ...])]
  [ON DELETE { NO ACTION | CASCADE | SET NULL }]
  [ON UPDATE { NO ACTION | CASCADE | SET NULL }]

ÜBERSICHT DER BEFEHLE
ALTER TABLE

Hochschule RheinMain



ÜBERSICHT DER BEFEHLE
DROP TABLE

Hochschule RheinMain

DROP TABLE Tabellenname { RESTRICT | CASCADE }

ÜBERSICHT DER BEFEHLE
TEMPORÄRE RELATIONEN

Hochschule RheinMain

```
CREATE { LOCAL | GLOBAL } TEMPORARY TABLE Tabellenname
( { Spalte { Datentyp | Gebietsname } [ Spaltenbedingung [ … ] ]
  | Tabellenbedingung                }
    [ , … ]
) [ ON COMMIT { PRESERVE | DELETE } ROWS ]
```

Analog zu Create Table

Inhalt löschen bei Transaktionsende

- Automatisches Löschen am Sessionende
- In mysql und mariaDB kein von Commit und kein Global

ÜBERSICHT DER BEFEHLE
VIEWS

Hochschule RheinMain

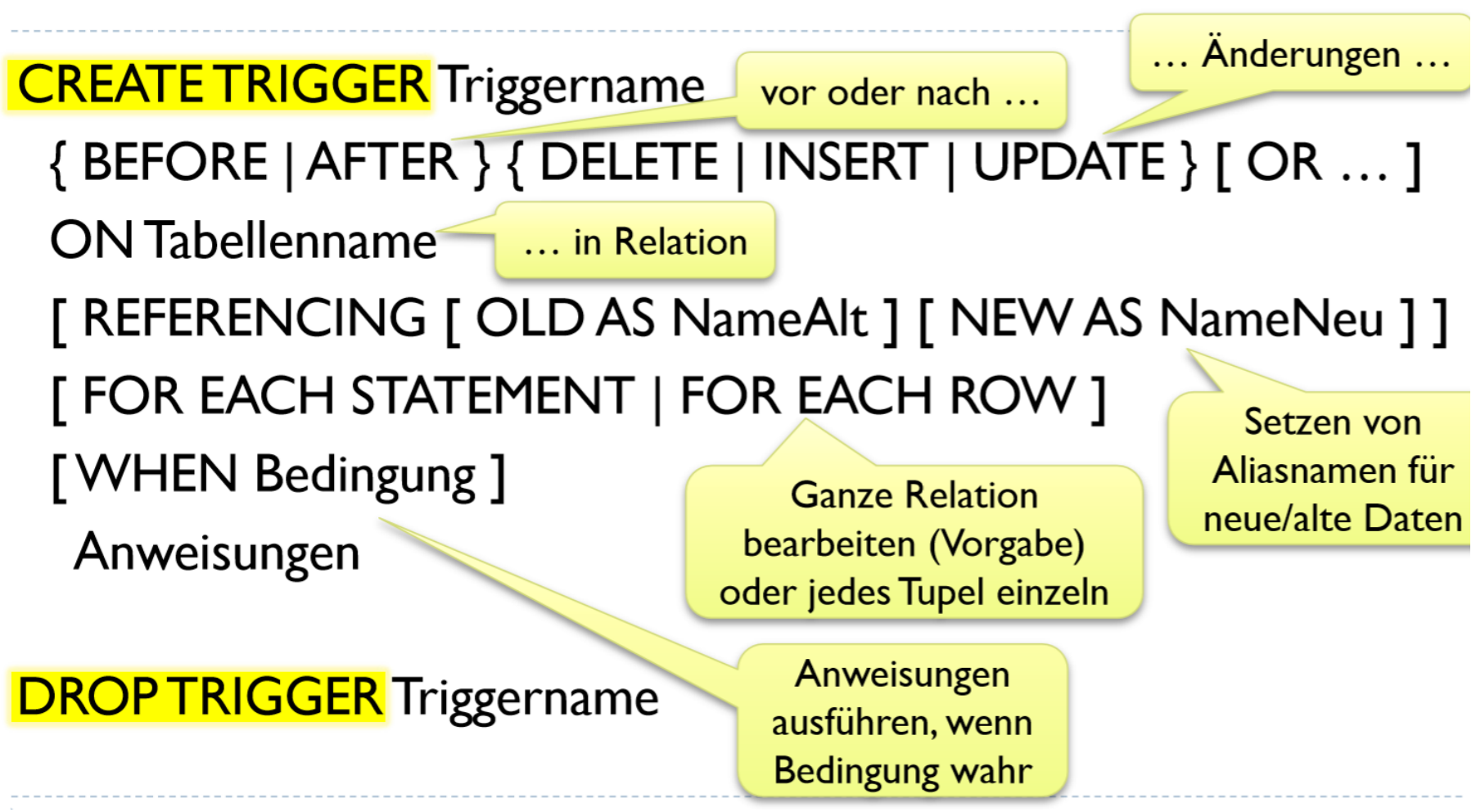
```
CREATE VIEW Sichtname [ ( Spalte [ , ... ] ) ]
  AS  Select-Befehl
  [WITH CHECK OPTION ]
```

```
DROP VIEW  Sichtname { RESTRICT | CASCADE }
```

ÜBERSICHT DER BEFEHLE

Trigger

Hochschule **RheinMain**



ÜBERSICHT DER BEFEHLE
Sequenzen

Hochschule RheinMain

CREATE SEQUENCE Sequenzname [AS Datentyp]
[START WITH Konstante] [INCREMENT BY Konstante]

DROP SEQUENCE Sequenzname

```
create table R (a int, b int);
create sequence test start with 100;
insert into R Values (NEXT VALUE FOR test, 7);
select * from R;
```

ÜBERSICHT DER BEFEHLE
GRANT

Hochschule RheinMain

```
GRANT          Zugriffsrecht [ , ... ]
ON [ TABLE ]   { Tabellenname | Sichtname }
TO             Benutzer [ , ... ]
[ WITH GRANT OPTION ]
```

ÜBERSICHT DER BEFEHLE
ZUGRIFFSRECHTE

Hochschule **RheinMain**

Zugriffsrecht	erlaubt ...
Select	den lesenden Zugriff auf eine Relation
Update	das Ändern von Inhalten einer Relation
Update (x1, ...)	das Ändern der Attributwerte x1,… einer Relation
Delete	das Löschen von Tupeln einer Relation
Insert	das Einfügen neuer Tupel in eine Relation
Insert (x1, ...)	das Einfügen der Attributwerte x1,… in eine Relation
References (x1, ...)	das Referenzieren der Attribute x1,… einer Relation
Usage	das Verwenden eines Gebietes

ÜBERSICHT DER BEFEHLE
GRANT

Hochschule RheinMain

- Alle Rechte vergeben:
  - ALL [PRIVILEGES]
- Rechte an alle Benutzer vergeben:
  - Benutzernamen PUBLIC verwenden
- Benutzer erhält Recht die Zugriffsrechte weiter zu geben
  - WITH GRANT OPTION

ÜBERSICHT DER BEFEHLE
REVOKE

Hochschule RheinMain

```
REVOKE [ GRANT OPTION FOR ]
    { Zugriffsrecht [ , … ] | ALL PRIVILEGES }
    ON [ TABLE ] { Tabellenname | Sichtname }
    FROM Benutzer [ , … ]
    { RESTRICT | CASCADE }
```

Hochschule RheinMain

STORED PROCEDURES UND FUNKTIONEN

PROBLEME

Hochschule RheinMain

1. Nutzer der Datenbank müssen immer einige Rechte auf den Tabellen haben
2. Neben der reinen Datenhaltung kann es sinnvoll sein, den Nutzern weitere Funktionalität anzubieten
3. Kontrollmenchanismen, die mehrere Tabellen berücksichtigen sind erstrebenswert

- Punkt 3 wird durch Trigger abgedeckt
- Punkt 1 und 2 durch sogenannte Stored Procedures und Funktionen

EINFÜHRUNG IN PL/SQL

Erstellen einer Stored Procedure

Hochschule **RheinMain**

- Klassische prozedurale Programmiersprache (ähnlich wie C)
- Eine gespeicherte Prozedur ist ein vorbereiteter SQL-Code und immer und immer wieder wiederverwendet können
- Parameter können an eine gespeicherte Prozedur übergeben werden, so dass die gespeicherte Prozedur auf dem Parameterwert handeln kann.

```
CREATE
    [OR REPLACE]
    [DEFINER = { user | CURRENT_USER | role | CURRENT_ROLE }]
    PROCEDURE sp_name ([proc_parameter[,...]])
    [characteristic ...] routine_body

proc_parameter:
    [ IN | OUT | INOUT ] param_name type

type:
    Any valid MariaDB data type

characteristic:
    LANGUAGE SQL
  | [NOT] DETERMINISTIC
  | { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
  | SQL SECURITY { DEFINER | INVOKER }
  | COMMENT 'string'

routine_body:
    Valid SQL procedure statement
```

EINFÜHRUNG IN PL/SQL

Ausführen einer Stored Procedure

Hochschule **RheinMain**

```
CALL sp_name([parameter[,...]])
CALL sp_name[()]
```

BEISPIEL

Hochschule RheinMain

```
DELIMITER //
create PROCEDURE FindeKunde (City char(20))
BEGIN
    SELECT * FROM kunde WHERE Ort = City;
END;
//
```

```
CALL FindeKunde('Regensburg');
```

BEISPIEL 2

Hochschule RheinMain

```
DELIMITER //
CREATE OR REPLACE PROCEDURE FindeKundeNameOrt (vname char(20), city char(20))
BEGIN
    SELECT * FROM kunde WHERE Name = vname and Ort = city;
END;
//

CALL FindeKundeNameOrt ('Fahrrad Shop', 'Regensburg');
```

BEISPIEL 3

Hochschule RheinMain

```
4  CREATE OR REPLACE PROCEDURE Hallo (name varchar(20), alt integer,
5  beruf varchar(20))
6  BEGIN
7      DECLARE zaehler INTEGER DEFAULT 0;
8      DECLARE name2 VARCHAR(10) DEFAULT name;
9      IF name IS NULL THEN
10         BEGIN
11             SET zaehler=zaehler+1;
12             SET name2='Unbekannt';
13         END;
14     END IF;
15     IF beruf IS NULL AND alt IS NULL THEN
16         SET zaehler=zaehler+2;
17     ELSEIF alt IS NULL THEN
18         SET zaehler=zaehler+1;
19     ELSEIF beruf IS NULL    THEN
20         SET zaehler=zaehler+1;
21     END IF;
22     SELECT name2 as 'Name', beruf as 'Beruf',
23     alt as 'Alter', zaehler as 'Fehlende Angaben';
24 END;
25 //
26
```

BEISPIEL 4

Hochschule RheinMain

```
DELIMITER //

CREATE OR REPLACE PROCEDURE sayItAgain (input varchar(20), anzahl integer)
BEGIN
    DECLARE zaehler integer DEFAULT 1;
    DECLARE ausgabe varchar(500) default '';
    WHILE zaehler <= anzahl DO
        set ausgabe = concat(ausgabe, input);
        set zaehler = zaehler + 1;
    END WHILE;
    SELECT ausgabe;
END;
//

CALL sayItAgain('Alles doof! ', 5);
```

EINFÜHRUNG IN STORED FUNKTIONEN

Hochschule RheinMain

- Ähnlich wie Prozeduren, Unterschied ist der Return-Wert mit einem bestimmten Typ

```
CREATE [OR REPLACE]
    [DEFINER = {user | CURRENT_USER | role | CURRENT_ROLE }]
    [AGGREGATE] FUNCTION [IF NOT EXISTS] func_name ([func_parameter[,...]])
    RETURNS type
    [characteristic ...]
    RETURN func_body

func_parameter:
    param_name type

type:
    Any valid MariaDB data type

characteristic:
    LANGUAGE SQL
  | [NOT] DETERMINISTIC
  | { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
  | SQL SECURITY { DEFINER | INVOKER }
  | COMMENT 'string'

func_body:
    Valid SQL procedure statement
```

BEISPIEL 1

Hochschule RheinMain

```
DELIMITER //
CREATE OR REPLACE FUNCTION AnzahlTeiler
    (zahl INTEGER) RETURNS INT DETERMINISTIC
    BEGIN
        DECLARE ergebnis INTEGER;
        SET ergebnis=0;
        FOR i IN 1..zahl
        DO
            IF (zahl % i)=0
            THEN SET ergebnis=ergebnis+1;
            END IF;
        END FOR;
        RETURN ergebnis;
    END
//
```

```
SELECT AnzahlTeiler(200);
```

BEISPIEL 2

Hochschule RheinMain

```
DELIMITER //
CREATE OR REPLACE PROCEDURE Primzahl (zahl INTEGER)
    BEGIN
        IF anzahlTeiler(zahl)=2
            THEN SELECT('ist eine Primzahl');
            ELSE SELECT('ist keine Primezahl');
            END IF;
    END;
//
```

```
CALL Primzahl(3);
```

WICHTIGE STATEMENTS

Hochschule RheinMain

```
IF search_condition THEN statement_list
    [ELSEIF search_condition THEN statement_list] ...
    [ELSE statement_list]
END IF;
```

```
[begin_label:]
FOR var_name IN [ REVERSE ] lower_bound .. upper_bound
DO statement_list
END FOR [ end_label ]
```

```
[begin_label:] WHILE search_condition DO
    statement_list
END WHILE [end_label]
```

```
[begin_label:] LOOP
    statement_list
END LOOP [end_label]
```

```
[begin_label:] REPEAT
    statement_list
UNTIL search_condition
END REPEAT [end_label]
```

RECURSION

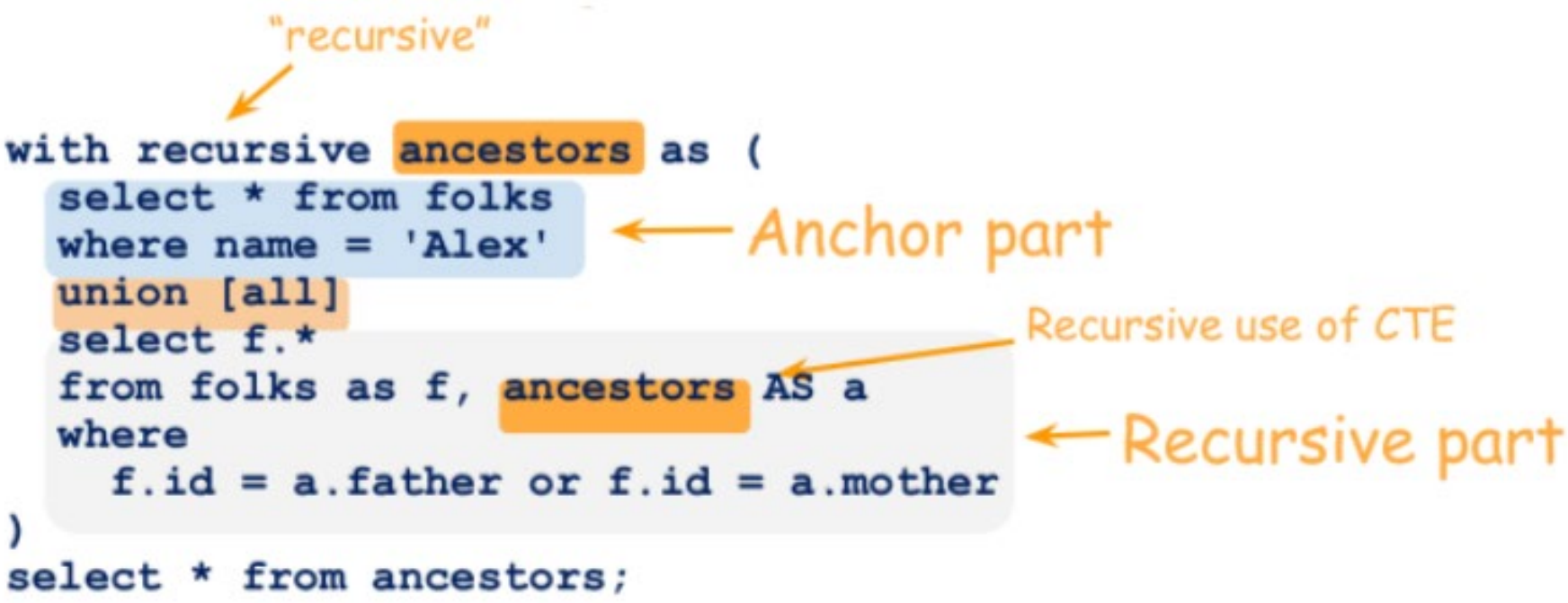
Hochschule RheinMain

```
WITH [RECURSIVE] table_reference as (SELECT ...)
  SELECT ...
```

- Das WITH Keywort zeigt auf eine Common Table Expression (CTE)
- Hier kann man eine Subquerz mehrere mal ablaufen lassen
- Diese temporäre Tabellen besteht nur während der Laufzeit der Query

FUNKTIONSWEISE REKURSIVE FUNKTIONEN

Hochschule RheinMain

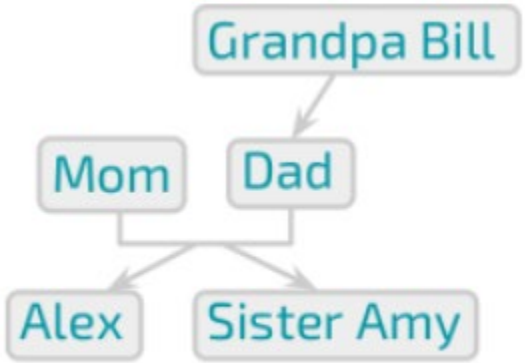


FUNKTIONSWEISE REKURSIVE FUNKTIONEN

Hochschule RheinMain

Recursive CTE computation

id	name	father	mother
100	Alex	20	30
20	Dad	10	NULL
30	Mom	NULL	NULL
10	Grandpa Bill	NULL	NULL
98	Sister Amy	20	30



FUNKTIONSWEISE REKURSIVE FUNKTIONEN

Hochschule RheinMain

Computation

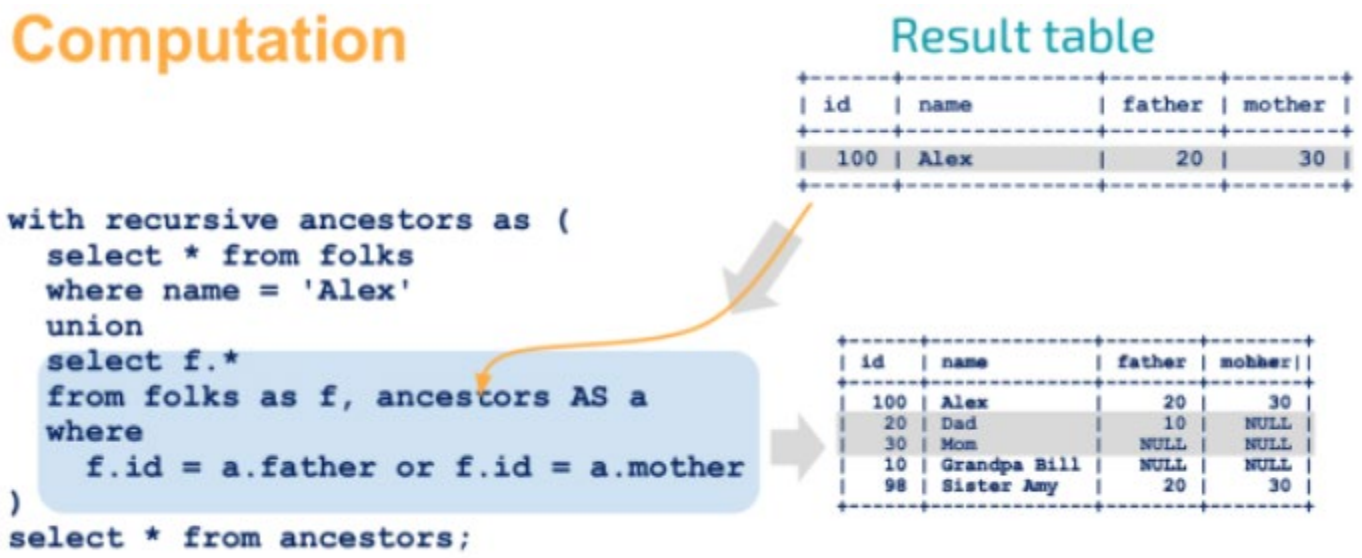
```
with recursive ancestors as (
  select * from folks
  where name = 'Alex'
  union
  select f.*
  from folks as f, ancestors AS a
  where
    f.id = a.father or f.id = a.mother
)
select * from ancestors;
```

Result table

id	name	father	mother
100	Alex	20	30

FUNKTIONSWEISE REKURSIVE FUNKTIONEN

Hochschule RheinMain



FUNKTIONSWEISE REKURSIVE FUNKTIONEN

Hochschule RheinMain

Computation

```
with recursive ancestors as (
  select * from folks
  where name = 'Alex'
  union
  select f.*
  from folks as f, ancestors AS a
  where
    f.id = a.father or f.id = a.mother
)
select * from ancestors;
```

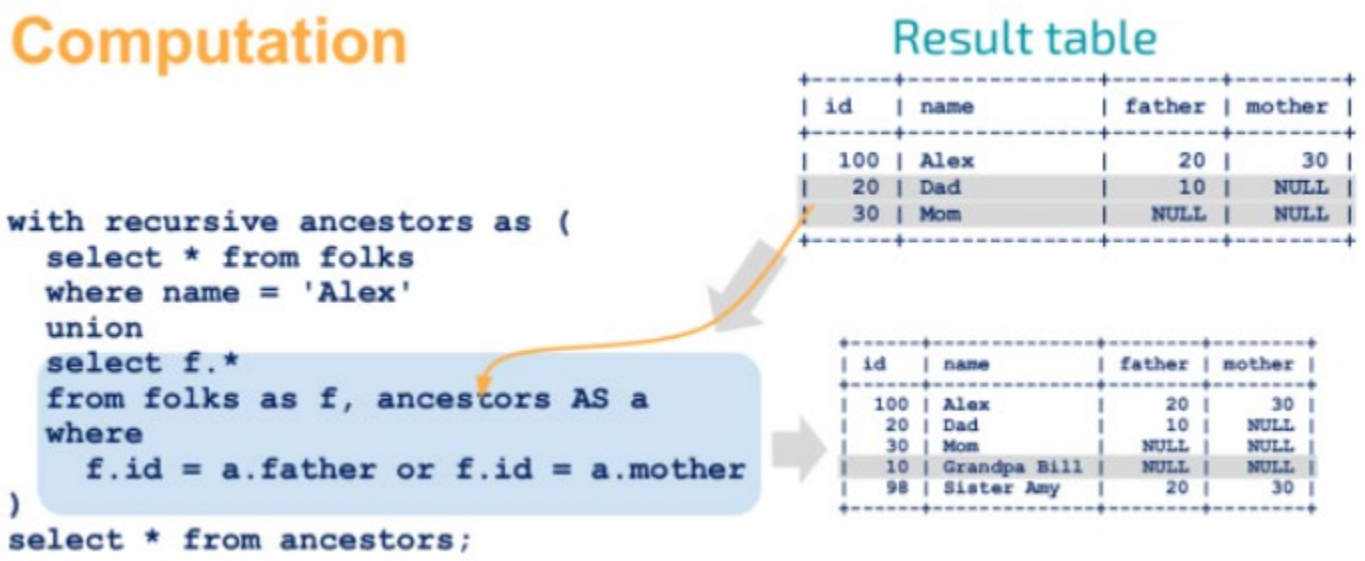
Result table

id	name	father	mother
100	Alex	20	30
20	Dad	10	NULL
30	Mom	NULL	NULL

id	name	father	mother
100	Alex	20	30
20	Dad	10	NULL
30	Mom	NULL	NULL
10	Grandpa Bill	NULL	NULL
98	Sister Amy	20	30

FUNKTIONSWEISE REKURSIVE FUNKTIONEN

Hochschule RheinMain



FUNKTIONSWEISE REKURSIVE FUNKTIONEN

Hochschule RheinMain

Computation

```
with recursive ancestors as (
  select * from folks
  where name = 'Alex'
  union
  select f.*
  from folks as f, ancestors AS a
  where
    f.id = a.father or f.id = a.mother
)
select * from ancestors;
```

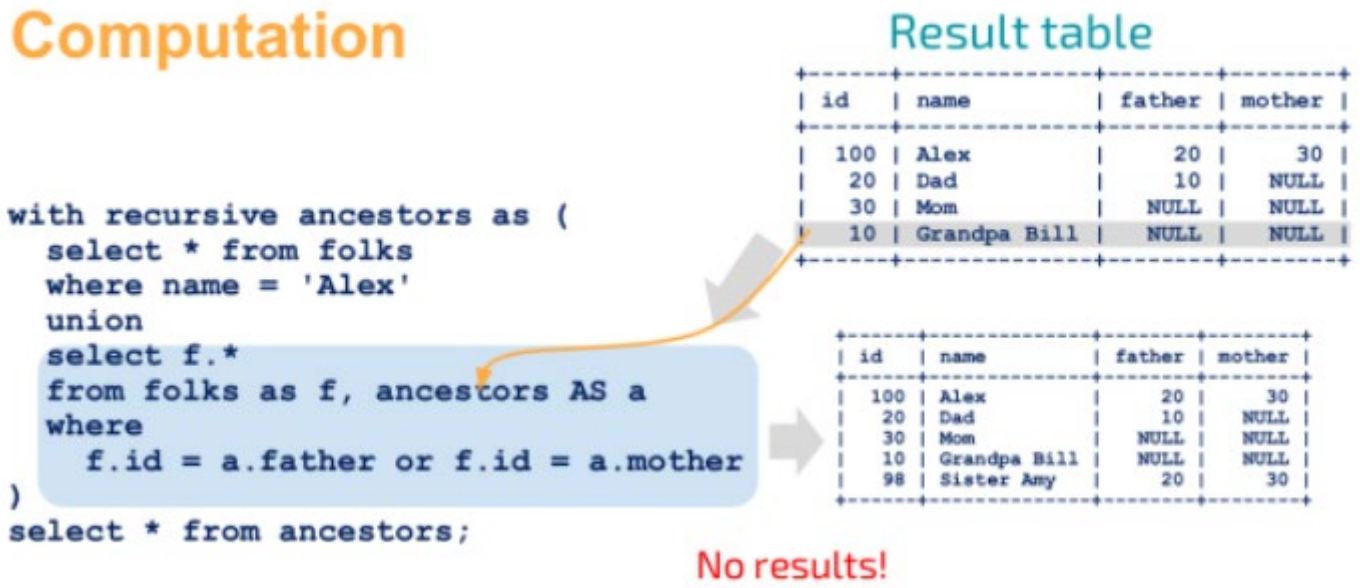
Result table

id	name	father	mother
100	Alex	20	30
20	Dad	10	NULL
30	Mom	NULL	NULL
10	Grandpa Bill	NULL	NULL

id	name	father	mother
100	Alex	20	30
20	Dad	10	NULL
30	Mom	NULL	NULL
10	Grandpa Bill	NULL	NULL
98	Sister Amy	20	30

FUNKTIONSWEISE REKURSIVE FUNKTIONEN

Hochschule RheinMain



FUNKTIONSWEISE REKURSIVE FUNKTIONEN

Hochschule RheinMain

```
with recursive R as (
  select anchor_data
  union [all]
  select recursive_part
  from R, ...
)
select ...
```

1. Compute anchor_data
2. Compute recursive_part to get the new data
3. if (new data is non-empty) goto 2;

FUNKTIONSWEISE REKURSIVE FUNKTIONEN

Hochschule RheinMain

bus_routes

origin	dst
New York	Boston
Boston	New York
New York	Washington
Washington	Boston
Washington	Raleigh

FUNKTIONSWEISE REKURSIVE FUNKTIONEN

Hochschule RheinMain

```
WITH RECURSIVE bus_dst as (
    SELECT origin as dst FROM bus_routes WHERE origin='New York'
  UNION
    SELECT bus_routes.dst FROM bus_routes, bus_dst WHERE bus_dst.dst= bus_routes.origin
)
SELECT * FROM bus_dst;
```

FUNKTIONSWEISE REKURSIVE FUNKTIONEN

Hochschule RheinMain

```
WITH RECURSIVE paths (cur_path, cur_dest) AS (
    SELECT origin, origin FROM bus_routes WHERE origin='New York'
  UNION
    SELECT CONCAT(paths.cur_path, ',', bus_routes.dst), bus_routes.dst
      FROM paths, bus_routes
      WHERE paths.cur_dest = bus_routes.origin AND
      NOT FIND_IN_SET(bus_routes.dst, paths.cur_path)
)
SELECT * FROM paths;
```

QUELLENVERZEICHNIS

Hochschule RheinMain

1. Database Systems – The complete book, Garcia-Molina, Ullman, Widom
2. Grundkurs Datenbankentwicklung – Von der Anfoderungsanalyse zur komplexen Datenbankanfrage, Kleuker
3. Datenbanken und SQL, Schicker
4. https://mariadb.com/kb/search/