Internship for lecture databases handout

Ioannis Karatassis, M.Sc. Alfred Sliwa, M.Sc.

Winter semester 2019/2020

Date	
Team (Account)	
Password	

Up-to-date information, contact persons, materials and uploads at:
http://www.is.inf.uni-due.de/courses/db_ws18/

v1.1

1Ablauf des Praktikums

An internship session is scheduled for two hours. This time may not be enough to solve all the bloc's tasks. Please note that, as in the case of a lecture, self-study time and an exercise are provided for the home-based processing of practice tasks, additional time is also required for the internship of at least two hours per week for the pre- and follow-up of the substance must be planned.

The internship takes place in a total of ten groups in room LF 230 (only **one** appointment is required). The participants of the internship are asked to avoid delays in order not to miss important explanations.

The submission of the tasks is evaluated. There is also an evaluated acceptance of the software project to be created in the last working block. The points achieved together with the cloistered points form the overall score for the event "Databases". If necessary, an ungraded note about the successful attendance of the internship will also be issued. Details of the grading were already discussed in the lecture and in the exercise.

In the course of the internship, the handling of a commercial database system is practiced on the basis of enquiries and then a case study from modelling to use in two task blocks is processed. The processing is to be done in small groups of two participants. The tasks should be worked **on together** so that each participant gets an insight into the tools and languages used. An internship block ends with the results of the block being uploaded via the web form (if charges are provided in the block). Both participants in a small group should be able to explain or demonstrate the solutions developed by the group on request.

1.1Accounts

The tasks to be processed on the computer take place on the machines in room LF 230 under a Linux environment (we use Ubuntu and thus the graphical interface Gnome). A short introduction will be given in the first internship session. The database system is IBM DB2 V9.7, which can be obtained free of charge from the IBM website even after prior registration. Accounts for each small group are issued in the first session.

Each account name is built according to the dbpxxx scheme, where xxx is the small group number. The password issued must be changed in the first session and should be carefully noted by both members of the small group.

Each account has a database manager instance that can be used by the small group. Within this instance, the group has all the necessary rights to create and delete its own databases.

1.2Lernziele

The handling of a commercial database system is to be practiced on the basis of requests over a predetermined schema. Advanced concepts such as sifting, recursion and triggers are also used. The SQL basics are referred to slides, books, and exercises for the lecture.

The design and implementation of a database should be practiced on the basis of a case study. First of all, a conceptual modeling must be carried out using an entity-relationship diagram. This is converted into a database schema, supplemented by necessary static and referential integrity conditions. The database must then be filled with sample data.

In the final weeks of the internship, the database will be addressed from a programming language. Html and JSPs or servlets are used to create a simple user interface. If necessary, the database must be filled with additional data to test all functionalities of the application in a meaningful way.

2Einführung in die Arbeit mit IBM DB2

2.1Benutzungsmodi

Before working with DB2, consider the difference between the Linux shell and the DB2 shell. A shell is a textual user interface that interprets commands on the command line. When you open a terminal window on the database machines (see task section), you are first in the Linux shell by default (to be precise, the *bash* or *Bourne-Again*shell). This allows e.g. fast navigation through directories or the call of programs. The shell indicates its willingness to take commands by a . . . and a blinking cursor. For multiline commands, the dollar sign is replaced with a >. Shell commands in the internship documents are identified by the preceding dollar sign. The dollar sign is not to be typed in.

DB2 also has a shell that is not to be confused with the Linux shell. DB2 commands are not understood by the Linux shell and vice versa. The DB2 shell is explained in the Interactive Mode section. It can be seen by the fact that the command prompt for input readiness changes to db2 =>. For multiline commands, the prompt becomes db2 (cont.) =>.

2.1.1Aufruf aus der Linux-Shell

The simplest use of DB2 is done directly from the Linux shell by prepend each DB2 command with the program name db2. This is how you can connect to a database using the following shell command from the Linux command line:

$ db2 "CONNECT TO mygeo"

It is recommended that you put all DB2 commands in brackets that are set in this way. Some characters used in DB2 commands, such as , (or) are special characters of the shell and are interpreted by the shell without bracketing. In most cases, this leads to undesirable results.

2.1.2Interaktiver Modus

In addition to the direct call with a command, you can also start the database client in interactive mode. This is achieved by calling db2 without a command. You then get into the DB2 shell described above. The dialog with the shell can be terminated with terminate.

By default, in this mode, each command must be on **a** line, a final semicolon is not necessary, and in most cases even results in an error message. If you prefer to formulate commands across multiple lines, you should call db2 with additional parameters: The following call starts a DB2 shell in which a command does not complete until a line ends with a semicolon:

$ db2 -t

A list of available parameters at the time of call and the presets can be obtained with the shell command

$ db2 "LIST COMMAND OPTIONS"

The interactive mode of DB2 is unfortunately very rudimentary and therefore only recommended for hard-bodied people. In particular, the possibilities for editing the command line are severely limited. However, it is possible to change the command history using history or h

to display it. This shows a numbered list of the most recently executed commands in the shell. If you want to execute a command again, you can do this by using the command roundcmd or short r, followed by the line number of the desired command, e.g. r 2 (to call the second command from the command history)

If you want to edit a previous command you use edit or e followed by the line number. This then starts the system's default text editor. After editing, you quit the editor and can now execute your modified command.

2.1.3Batch-Modus

The most comfortable and recommended working mode is batch mode. In batch mode, you use a text editor of your choice (there are a number available that also offer syntax highlighting for SQL) to save your commands to a file. This text file is then passed to the program db2 when called:

db2 -tvf script.sql (executes the commands from the script.sql file)

As before, the t parameter allows SQL commands to be written over multiple lines (completed by a semicolon). The v makes the program black and f instructs DB2 to read its commands from the specified file.

The batch mode is especially favorable because it gives you a trial protocol with the commands you use, which you can use for your delivery. To insert comments or comment out individual commands in the batch file, use two minus signs at the beginning of a line.

This is an example batch file:

```
-- Batch file for group 99 CONNECT TO
mygeo;

SELECT name FROM dbmaster.country;

-- don't forget to disconnect
DISCONNECT mygeo;
```

If you want to log not only the commands, but also the results, the output of DB2 can be easily redirected to a file:

db2 -vtf script.sql > filename.txt

The > creates a new file filename.txt (or overwrites an existing one) with the output of the preceding command. In order to connect an existing
Appending file instead of overwriting, you can use >> instead.

2.2Grundlegendes

The basic organizational unit of the DB2 database system is the database manager instance. An **instance** is bound to a specific SystemAccount and manages all system resources for the databases that belong to that account. Each database must be created within an instance. Each small group owns its own DB2 instance. Within this you can operate freely, change configurations, create and delete databases without getting in the way of other groups. The database manager of the instance can start with the db2start command, and with db2stop

stopped. If you receive an error message that no database manager is running, then run this command first. Specifically, the database manager must be restarted if your configuration settings change.

In order to work with databases that do not belong to your own instance, you first have to make them known to the system. For this purpose, DB uses a catalog called **Node Directory**. The Node Directory lists all foreign instances you want to work with. Connection parameters (host, port) and instance owners are recorded to them. At the beginning, the Node Directory for your small group is still empty. You can do it with the list node directory command

display. As long as there are no entries, DB2 will be output

SQL1027N The node directory could not be found.

Answers. To make the system aware of other (remote) instances, the catalog tcpip node command is used. This command expects some parameters that are important for the later connection to the remote database. If there are connection difficulties, incorrect cataloging is often to blame. Then the catalog entry must be deleted and recreated. The parameters are: **remote** the host name on which the instance is located **server** the port through which the instance can be accessed **remote_instance** the name of the instance **system** the name of the remote system **ostype** the operating system of the remote machine

For example, an exemplary cataloging command looks like this (the command belongs completely in a line). This catalogues the intanz with which you will work in the first internship block locally under the name sample.

CATALOG TCPIP NODE sample REMOTE bijou.is.inf.uni-due.de SERVER 50005 REMOTE_INSTANCE dbmaster SYSTEM bijou OSTYPE linux

DB2 should respond with a success message. Redisplaying the Node Directory should now show the cataloged instance. You can remove an incorrectly cataloged instance with the command

UNCATALOG NODEName

Now you can access a remote database in the cataloged
instance by assigning it a local alias. With this alias, you can then treat the database as if it were local. The command for this is similar to cataloging an instance:

CATALOG DATABASE *RemoteName* AS *localAlias* AT NODE *RemovedInstance*
Suppose you have cataloged the instance as sample as above locally, then you can access our sample database with the following command:

CATALOG DATABASE mondial AS mygeo AT NODE sample

You can also view the list of cataloged databases:

LIST DB DIRECTORY oder LIST DATABASE DIRECTORY

Databases that are no longer needed or catalogued incorrectly can be used with the UNCATALOG DATABASE *Name*

remove it from the catalog. Attention: locally created databases are automatically entered in the catalog. If you delete the catalog entry, you will no longer have access to the local database and will need to recatalogitate it. The procedure is the same as for remote databases, but no alias and no remote instance must be specified (CATALOG DATABASE *localname*).

2.3Verbindung herstellen

Because you can theoretically have as many databases available on an instance, you must first connect to a specific cataloged database before you can execute commands on it. The command to do so is:

CONNECT TO *Name* (z.B. CONNECT TO mygeo)

All other commands until the connection is scan or until you connect to another database now refer to that database. Connections that are no longer needed should always be closed: DISCONNECT *name* or

TERMINATE (terminates all open connections)

2.4Schema und Tabellen

A database schema is a collection of tables. The definition of a scheme in DB2 is somewhat freer than that of the lecture. Within a database, there can be tables from multiple schemas, and tables from a schema can be found in multiple databases. The system tables of DB2 can be found in the SYSIBM schema and in the SYSCAT schema. For example, here are the tables where your table and view definitions are stored.

By default, new tables are created in a schema whose name matches that of the instance owner, such as dbp001 or dbmaster.

To display a list of tables in the default schema, use the command

LIST TABLES

To display a list of tables in a different schema, use

LIST TABLES FOR SCHEMA *SchemaName*

As long as you want to use tables from a foreign schema, you have to precede the schema name with the table name with the help of point notation. For example, you can use the following query to find out how many rows are in the tables system table:

SELECT count(*) FROM sysibm.tables

When working with a foreign database for a long time, it can be quite annoying to constantly precede the schema. Then you can also change the default scheme for the current session:

SET SCHEMA SchemaName

In order to find out more about a table in the currently connected database, e.g. which columns it has, which data types are expected or whether null values are allowed, you can have these described by the system:

DESCRIBE TABLE SchemaName.TabellenName

Optionally, you can add SHOW DETAIL to the command to get a little more information, but this information is rather unimportant for most purposes. Most other things, such as existing constraints on tables, can be read from the system tables. We will come back to this in a later working block.

2.5Verbindung mit JDBC

To connect to the DB from outside the university:

```
public static Connection createConnection() throws ClassNotFoundException, SQLException {
  Class.forName("com.ibm.db2.jcc.DB2Driver"); Properties properties = new

  Properties();

  properties.setProperty("user","DBPXXX"); properties.setProperty("password","password");

  Connection connection = DriverManager.getConnection("jdbc:db2 ://<rechnername>.is.inf.uni-
       due.de:50005/<DBNAME>: currentSchema=<DBPXXX>;", properties);

  return connection;
}
```

3 Anfragen mit SQL

3.1 Subselect

In DB2-SQL, there are three forms of requests: subselects, full selects, and the full SELECT statement. For each request (query) you must have at least the SELECT right for each table or view used.

A subselect consists of seven clauses, with only the SELECT and FROM clauses being mandatory. The order is set:

SELECT ...
FROM ...
WHERE ...
GROUP BY ...
HAVING ...
ORDER BY ...
FETCH FIRST ...

For example, to complete the specification of the syntax, refer to the DB2 documentation.

3.1.1SELECT

The SELECT part specifies what the result should look like, i.e. which columns should have the result relation, and, if necessary, how they should be named. The individual columns of the result are separated by commas, the re-enlistment of an unqualified column name names the column of the result relation (um). * or table.* ink inwhich all columns (the named table) are transferred to the result.

Select by renaming a result column:

SELECT name, code AS 'kfz'
FROM country

SELECT DISTINCT performs duplicate limination. It is possible to have the values of the result columns in the SELECT part calculated by an *expression.* The data types of the result columns are essentially the same as the
Data types of the origin columns, or the result data type of an expression.

3.1.2FROM

The FROM part specifies the ratios (tables) used for the request, whereby one can temporarily give a new name to tables by re-enchanging a unique identifier (with the optional keyword AS) tables (keyword: tuple variables). If several tables are used in a case, they shall be specified separately in the FROM part by commas. The request then works on the Cartesian product of these tables.

Example of tupel variables for unique labeling tables:

SELECT c1.*
FROM city AS c1, city
AS c2

It is possible to specify a subquery (a fullselect) as a relation in the FROM part. In this case, it is imperative to give this relation a name.

Instead of a single table, you can specify a "joined table" clause as an intermediate relation, which is the result of a JOIN:

tab1 JOIN tab2 ON condition tab1 INNER JOIN tab2 ON
condition tab1 {LEFT,RIGHT,FULL} (OUTER) JOIN tab2 ON condition

The definition of the different TYPES of JOIN is as discussed in the lecture **databases** or as described in the accompanying literature. Without specifying a JOIN operator, the INNER JOIN is accepted. For multiple JOINs, the order is important for OUTER JOINs, so it should be noted that the JOINs are usually calculated from left to right, with the position of the JOIN condition playing a role.

SELECT DISTINCT ci.name,co.name
FROM city AS ci, country AS co

forms all possible combinations (i.e. the Cartesian product) of the tuples from city and country and returns the name of the cities and countries from the combined relation. Such a combination without a JOIN condition is desired in the selder.

SELECT ci.name, co.name
FROM city AS ci JOIN country AS co on ci.country = co.code forms only the combinations of the tuples of city and countrywhere the countrycode from city matches the country code in country, i.e. exactly the combinations of city and countrywhere the city is located in the country inquestion.

The following statement does the same as the last example:

SELECT ci.name, co.name
FROM city AS ci, country AS co
WHERE ci.country = co.code

3.1.3 WHERE

The WHERE clause selects all tuples for which the specified search condition evaluates to *true*. The search condition must include:

- the column names used must be uniquely a column of the
  part of the specified relations or the result of a JOIN from the FROM part.
- an aggregate function may only be used if the WHERE clause is used in a subquery of a HAVING clause

SELECT *
FROM user
WHERE substr(nick,1,2) = 'Op'

provides all the tuples of the relation userswhere the attribute Nick starts with 'Op'.
The same condition could also be written in the following form:

SELECT *
FROM user
WHERE nick like 'Op%'

3.1.4GROUP BY

The GROUP BY clause can be used to group the tuples of the previous result, specifying at least one grouping expression. This must not contain subqueries, and all tuples for which the grouping expression has the same value belong to a group. A result tupel is calculated for each group. Most of the time, YOU use GROUP BY together with aggregator functions, and the groups determine to which range the function is applied.

If a Subselect GROUP USES BY or HAVING and column names appear in the expressions of the SELECT clause, they must occur in an aggregate function, are derived from an expression of the GROUP BY clause, or originate from a surrounding case.

3.1.5HAVING

The HAVING clause allows conditions to be set up for the groups formed. The groups for which the search condition evaluates to *true* are selected. If no GROUP BY clause is specified in the case, the entire previous result is treated as a group.

Only column names that originate from an external request, are used within the aggregation function, or that occur in a grouping expression, may be used in this search condition.

3.1.6ORDER BY

The ORDER BY clause specifies how the result relation should be sorted, i.e. in which order the result tuples should be output. Multiple sort keys can be specified, then sorted first according to the first key, then for tuples matching in that key after the second key, and so on.

The information provided by ASC or DESC determines whether the sorting is ascending or descending. The default is ASC, where the null value is assumed to be greater than all other values.

The sort key is either a column name of the result relation (alternatively, an integer value can be used to select the column based on its position), or one of the relations used in the FROM clause or allows an expression. If DISTINCT was used in the SELECT part, the sort key must correspond exactly to one expression of the SELECT list. If the subselect is grouped, only selectlist expressions, grouping expressions, and expressions that contain an aggregate function, constant, or host variable can be used.

The following request lists all movie data by years and titles, with the latest movies first:

```
SELECT *
FROM film
ORDER BY drehjahr DESC,titel ASC
```

3.1.7FETCH FIRST

The FETCH FIRST clause allows the result relation to be limited to a specific number of tuples. This is recommended for testing requests in any case, because you may not need to calculate all tuples, so performance increases.

The first 5 series episodes where the sequence title is not empty (zero) or the empty string:

```
SELECT series title,turning year,staffelnr,follow-up,followtitle
FROM serienfolge
WHERE folgentitel IS NOT NULL AND
   folgentitel<>''
ORDER BY drehjahr,serientitel,staffelnr,folgennr DESC
```

FETCH FIRST 5 ROWS ONLY

3.2Aggregatfunktionen

Aggregate functions are applied to a set of values derived from an expression. The expressions used can refer to attributes, but not to scalar cases or other aggregate functions. The range to which an aggregate function is applied is a group or an entire relation. If the result of the previous edit is an empty quantity and GROUP BY was used, the aggregate functions are not calculated and an empty quantity is returned. If GROUP BY is used, the aggregate functions are applied to the empty set.

If no GROUP BY and HAVING were used in a subselect, aggregate functions can only occur in the SELECT clause if they include all attribute names.

The following aggregator functions take exactly one argument at a time. NULL values may be eliminated before the calculation. Set to the empty quantity, they return NULL. The additional keyword DISTINCT can be used to achieve duplicate limination before the calculation.

AVG: Average of values

MAX: Maximum of values

MIN: Minimum of values

STDDEV: Standard deviation of the values

SUM: Sum of variANCE **values:** Variance

Example:

What is the last year for which films are available in the database?

```
SELECT max(drehjahr)
FROM film
```

The COUNT function evaluates the number of values to which the argument expression evaluates. COUNT(*) returns the number of tuples in the respective range, groupings are taken into account. The expression eliminates null values. The result is then the number of values (when using DISTINCT in pairs) in each range.

The result of COUNT can never be null.

3.2.1Beispiele

Number of episodes per "Buffy" relay:

```
SELECT series title,turning year,staffelnr,COUNT(following) AS number
   FROM serienfolge
   WHERE series title like 'Buffy%'
```

GROUP BY series title,turning year,staffelnr *Longest film*

title:

```
SELECT title
FROM film
WHERE length(titel) >= ALL (SELECT DISTINCT length(titel) FROM film)
```

How many episodes do the seasons of "Buffy" average? (With an accuracy of 2 digits behind the comma.)

```
SELECT series title,turning year,DEC(AVG(FLOAT(number)),4,2) AS avg sequences FROM
   (SELECT series title,turn-year,staffelnr,COUNT(following) AS number FROM
      series sequence
      WHERE series title like 'Buffy%'
      GROUP BY series title,turning year,staffelnr
   ) as tmp
   GROUP BY series title,turning year
```

Further examples of subselects, joins and grouping can be found in the documentation for DB2, or can be found in the accompanying material for the lecture.

3.3Fullselect

A fullselect consists of two or more subselects that are linked through set operators. The result tuples of the subselects must each have the same digit, and the data types at the corresponding locations must be compatible. The available quantity operators in SQL are UNION, UNION ALL, EXCEPT, EXCEPT ALL, INTERSECT, INTERSECT ALL. Two tuples are considered equal if the corresponding components are equal (where two null values are considered equal).

If more than two join operators are used, the clipped full selects are calculated first, then all INTERSECT operations are executed and finally the operations are evaluated from left to right.

3.3.1UNION

UNION is the quantity association, where duplicate limimination takes place. When using UNION ALL, no duplicates are eliminated at the union.

All actors from the series "Buffy the Vampire Slayer" and the movie "Buffy the Vampire Slayer" from 1992:

```
   SELECT schauspielerid
   FROM spieltInFolge
   WHERE series title='Buffy the Vampire Slayer'
UNION
   SELECT schauspielerid
   FROM spieltInFilm
   WHERE titel='Buffy the Vampire Slayer' and drehjahr=1992
```

3.3.2INSERSECT

INTERSECT is the quantity average, i.e. the result is formed by all tuples that lie in both relations. When using INTERSECT ALL, the result also contains duplicates, in the case of INTERSECT it does not.

All actors who have stared in both the 1997 series and the 1992 film "Buffy the Vampire Slayer":

```
   SELECT schauspielerid
   FROM spieltInFolge
   WHERE series title='Buffy the Vampire Slayer'
INTERSECT
   SELECT schauspielerid
   FROM spieltInFilm
   WHERE titel='Buffy the Vampire Slayer' and drehjahr=1992
```

3.3.3EXCEPT

With EXCEPT ALL, the result is determined by removing from the left-hand relation all tuples that also occur in the right relation, taking into account duplicates. When using EXCEPT, the result is formed by all tuples of the left relation that do **not** occur in the right relation. Then the result does not contain any duplicates.

All actors from the first "Buffy" season who did not (!) also play in the seventh season:

```
   SELECT schauspielerid
   FROM spieltInFolge
   WHERE series title='Buffy the Vampire Slayer'
      AND staffelnr=1
EXCEPT
   SELECT schauspielerid
   FROM dbprak.spieltInFolge
   WHERE serientitel='Buffy the Vampire Slayer' AND
      staffelnr=7
```

4Views

An important database concept are views or views. These can be generated in SQL with the CREATE VIEW statement.

A view that only indicates the nicks, residences and email addresses of the users, but keeps passwords and real names hidden:

```
CREATE VIEW Nicks AS
   SELECT nick,place of residence,email
   FROM user
```

A view provides a specific view of the database. Views can be used for different purposes. In data protection, for example, you can exclude sensitive data through a view for certain groups of users. You can simplify requests by looking at views as a kind of macro that can be incorporated into a SELECT statement in an appropriate place. Finally, views can also be used to recreate models of generalization:

```
CREATE TABLE film ( titel
    VARCHAR(200) NOT NULL,
    drehjahr INTEGER NOT NULL
)

CREATE TABLE serienfolge ( serientitel
    VARCHAR(200) NOT NULL, drehjahr
    INTEGER NOT NULL folgentitel
    VARCHAR(200), staffelnr INTEGER NOT
    NULL, folgennr INTEGER NOT NULL
)

CREATE VIEW productions AS
      SELECT titel,drehjahr FROM film
   UNION
      SELECT series title AS title,turning year FROM series
```

Just as in this example the upper type was realized as a view, subtypes can also be realized as views of an upper type.

Views can be used like tables. However, the problem is that they are often not updateable. A view can be changed (i.e. allows UPDATE) if it does not contain aggregation functions, or statements such as DISTINCT, GROUP BY, or HAVING, the SELECT clause contains only unique column names and contains a key of the base relation, and it contains only exactly one table that is also modifiable.

5Unterstützte Funktionen

A complete description of the features that DB2 supports can be found in the online documentation. Here is a list of the most important functions.

Examples:

- FLOAT (number) converts the values of the number of attributes to a different data type before further processing
- DEC (averageasfloat,4,2) represents the value of averageAsfloat as a decimal number with total

  4 digits, of which 2 digits are behind the comma, represent
- SUBSTR(nick, 1, 2) returns the first 2 characters of the values of nick

- CAST(null AS int) führt eine Typkonvertierung des NULL-Wert auf integer durch **Typkonvertierung:** BIGINT, BLOB, CAST, CHAR, CLOB, DATE, DBCLOB, DECIMAL, DREF, DOUBLE, FLOAT, GRAPHIC, INTEGER,

  LONG, LONG_VARCHAR, LONG_VARGRAPHIC, REAL, SMALLINT, TIME, TIMESTAMP, VARCHAR, VARGRAPHIC

Mathematik: ABS, ACOS, ASIN, ATAN, CEIL, COS, COT, DEGREES, EXP, FLOOR, LN, LOG, LOG10, MOD, POWER, RADIANS, RAND, ROUND, SIGN, SIN, SQRT, TAN, TRUNC

Stringmanipulation: ASCII, CHR, CONCAT, DIFFERENCE, DIGITS, HEX, INSERT, LCASE, LEFT, LENGTH, LOCATE, LTRIM, POSSTR, REPEAT, REPLACE, RIGHT, RTRIM, SOUNDEX, SPACE, SUBSTR, UCASE, TRANSLATE

Datumsmanipulation: DAY, DAYNAME, DAYOFWEEK, DAYOFYEAR, DAYS, HOUR, JULIAN_DAY, MICROSECOND, MIDNIGHT_SECONDS, MINUTE, MONTH, MONTHNAME, QUARTER, SECOND, TIMESTAMP_ISO, TIMESTAMPDIFF, WEEK, YEAR

System: EVENT_MON_STATE, NODENUMBER, PARTITION, RAISE_ERROR, TABLE_NAME, TABLE_SCHEMA, TYPE_ID, TYPE_NAME, TYPE_SCHEMA

Other: COALESCE, GENERATE_UNIQUE, NULLIF, VALUE

6SQL Programming Language

The SqlPl is defined in the SQL99 standard and is a very simple programming language that can be used within a DBMS. In particular, it is used in stored procedures, triggers, and sql UDFs. In the case of triggers and functions, the SqlPl statements are stored in the DBMS and interpreted as needed.

The SqlPl includes, among other things, the following briefly described instructions. Procedures allow additional control statements, which should not be dealt with here.

IMPORTANT: The individual statements in SqlPl are separated by semicolons. Therefore, it is important to note that lines in the CLP do not end with a semicolon (except for the actual end of the statement), or use an alternative termination character. For example, when the CLP is called with db2 -c -td@, the command line expects the character
Line.

To summarize several SqlPl statements, you can use a begin ... END bracketed instruction block. This is described in more detail in the documentation, and looks like this:

```
name: BEGIN
ATOMIC
        DECLARE counter INTEGER;
        SET counter = (SELECT count(*) FROM table);

        Instructions;
END name
```

The ATOMIC keyword specifies that if errors within the statement block are made, **all** statements of the block are rolled back. Variables that you want to use within the instruction block are defined with DECLARE. The variable is given a local name, specifies the data type and defines a default value for the variable (null by default). The value of a variable is set by SET.

The *label name* makes it possible to leave the named block with a LEAVEStatement. In addition, the label can be used to qualify variables.

6.0.1SQL-Statements

Many of the usual SQL statements are also allowed within function definitions: the Fullselect, UPDATE, or DELETE with search condition, INSERT, and SET for variables.

6.0.2FOR

The FOR statement executes one or more SQL statements for each tuple of a relation. The FOR loop can be named by pre-setting *label:*. This label can then be used in LEAVE or ITERATEStatements.

```
BEGIN
      DECLARE vollertitel CHAR(40);
      FOR t AS
          SELECT title, year, budget FROM production
          OF the
             SET vollertitel = t.titel || ' (' || t.jahr || ')';
             INSERT INTO einetabelle VALUES (vollertitel, t.budget); END FOR;
END
```

The above example uses the '||' -String concatenation operator (merge two strings into a new one).

6.0.3 IF

The IF statement allows conditional execution of statements. The statement is executed when the search condition evaluates to *true.* If it returns *false* or *unknown,*the next search condition after anELSEIF is executed instead. If no search condition is true, the ELSE branch is run instead.

```
IF anzahl_postings < 10 THEN SET typ
    = 'Anfnger';
ELSEIF anzahl_postings < 100 THEN SET
    typ = 'Member';
ELSEIF anzahl_postings < 1000 THEN SET
    typ = 'Seniormitglied';
ELSE
    SET type = 'Professional';
END IF

IF folgen_gesamt > (SELECT count(*) FROM
            seriefolge
            WHERE series title = title1
               AND drehjahr = titel2)
THEN RETURN 'unvollstndig';
ELSE RETURN 'vollstndig';
END IF
```

6.0.4 WHILE

A WHILE loop is traversed until the specified search condition is no more than *true*. Each pass is performed by the loop body. The WHILE loop can be named by pre-setting *label:*. This label can be used in LEAVE or ITERATE statements.

```
WHILE counter < 10
OF the
    Schleifenkörper END
WHILE
```

6.0.5 ITERATE

With the ITERATE statement, you leave the processing of the current loop pass and jump back to the beginning of the named loop (FOR, LOOP, REPEAT or WHILE).

```
ITERATE schleifenname
```

6.0.6 LEAVE

The LEAVE statement is used to leave a named loop (FOR, LOOP, REPEAT, WHILE) or an instruction block prematurely. If necessary, all cursors are closed, except those that define the result relation.

```
LEAVE schleifenname
```

6.0.7 RETURN

With the RETURN statement, m canjump out of a function.

A return value must be defined that is compatible with the return type of the function. Table functions require a fullselect.

RETURN substring(name,locate(name,','))

RETURN SELECT name, death date
FROM person
WHERE death date IS NOT NULL

6.0.8 Zuweisungen

Using SET, assignments to variables can be made within an instruction block or in the fuselage of a control statement.

SIGNAL

The SIGNAL statement allows you to throw a SQL error or a SQL warning. The explanatory error text can be a maximum of 70 characters long and can also be a variable. To name the SQL state:

- an SQL state is a five-characterstring of uppercase letters and Digits
- it must not start with '00' (success),
- in triggers or functions, it must not start with '01' (warning) or '02',
- an SQL state that does not start with '00', '01' or '02' indicates an error,
- if the string starts with 0-6 or A-H, the last three characters must start with I-Z

SIGNAL SQLSTATE '03ILJ'
SET MESSAGE_TEXT='Error: Illegal Year'

7 Rekursive Anfragen

In a full SELECT statement, it is also possible to make recursiverequests. Recursive requests are those that relate to themselves in their definition. For example, you can calculate the *transitive shell* of a tuple or bills of materials.

A short trip to explain the concept of recursion:

> **Recursion**
> *from Wikipedia, the free encyclopedia* (http://de.wikipedia.org/wiki/Rekursion)
>
> Recursion means self-referentiality. It always occurswhen something refers to itself or several things to each other, so that strange loops arise. For example, the phrase "This
> Sentence is untrue" recursive, as he speaks of himself. [...]
> Recursion is a general principle for solving problems, which in many cases leads to "elegant" mathematical solutions. Recursion is the call or

definition of a function by itself. Without an appropriate cancellation condition, such back-referential calls aso-called infinite recourse (endless recursion)

[...] The definition of recursively defined functions is a basic approach in functional programming. Starting from some given functions (such as the Sum function below), new functions are defined, withthe help of which further functions can be defined.

Here is an example of a function *sum:* N_0 - N_0, whichcalculates the sum of the first n numbers:

$$\{0, - \text{falls } n \not= 0 \text{ (Rekursionsbasis) } sum(n) = sum(n\ 1) +$$

n, falls $n = 0$ (Rekursionsschritt)

In the case of a recursive request (RA), the following rules must be observed:

- Any fullselect that is part of an RA must start with SELECT (but not with SELECT DISTINCT). It must not be nested. UNION ALL **must** be used as an association.
- In the WITH part, column names mustbe explicitly specified.
- The first full select in the first association must not refer to the table to be defined. It forms the basis of recursion.
- The data types and lengths of the columns of the table to be defined areset by the SELECT clause of the recursion base.
- No fullselect in the recursive definition can contain an aggregate function, a GROUP BY clause, or a HAVING clause. The FROM clauses in the recursive definition may have a maximum of one reference tothe table to be defined.
- Subqueries must not be used.

If the data can contain cycles, then it is possible that an infinite recursivesion is generated by an RA. Then it is important to define a cancellation conditionn:

- The table to be defined must contain an attribute that is initialized by an INTEGER constant and is incremented regularly.
- Each WHERE clause must have a check condition.

The evaluation of a recursive request runs as follows asfollows: First, the tuples of the recursion base are calculated, then further tuples are calculated based on these tuples according to the second part of the fullselect (according to the UNION ALL). However, only those tuples from the recursively defined process thathave been added at the last calculation step areused (starting with the tuples of the recursion base). If no new tuples are added in a calculation step, the calculation ends.

Example:

There is a table with constructionparts in which the components for workpieces are recorded. These components themselves can be composed of individual parts.

```
CREATE TABLE components (
        stueck
            VARCHAR(8),
        component VARCHAR(8),
        menge          INTEGER);
```

In order to determine which basic components are needed in total to produce an example part, you can use a recursive request. In the following RA, thequery rek defined in the first part of the WITH clause is the recursion base, and in the second part of the WITH clause the definition of recursion follows. The result relation thus recursively defined is then used in the SELECT DISTINCT part.

```
WITH rek (stueck, komponente, menge) AS ( SELECT
            r.stueck, r.kompenente, r.menge
            FROM components r
            WHERE r.stueck = 'Example Part'
    UNION ALL
            SELECT kind.stueck, kind.komponente, kind.menge
            FROM rek father, parts child
            WHERE father.component = kind.stueck
      )
SELECT DISTINCT stueck, component, quantity FROM
rek
ORDER BY stueck, component, quantity
```

8 Trigger

Triggers in SQL are a sequence of statements (a sequence of actions) that is executed when a modifying statement is to be executed on a particular table. We remember from the lecture that changinginstructions are for tables DELETE, INSERT and UPDATE.

For example, you can use triggers to

- to check the validity of newly inserted or modified tuples with regard to predetermined conditions,
- to generate values,
- read in other tables (e.g. for the resolution of cross-references) or write (e.g. for logging)

In particular, you can use triggers to model rules in the database. A trigger is created with the CREATE TRIGGER statement, with DROP TRIGGER it istriggered.

You can write triggers either to run once for each tuple to be changed (FOR EACH ROW) or once for each changing statement (FOR EACH STATEMENT). The latter, however, is only necessary for the so-called AFTERtriggers, which act after a changing operation. In this case, the trigger is processed even if no tuple is affected by the DELETE or UPDATE statement.

The REFERENCING clause specifies names for the transitionstates: OLD AS name or NEW AS name defines name as the name for the values of the affected tuple before or after the triggering operation is performed. Accordingly, OLD_TABLE asidentifier or NEW_TABLE AS identifier this as a name for a hypothetical table containing the affected tuples before or after the triggering operation. The following applies:

triggering event and time	**ROW trigger must not be Use**	**STATEMENT-Trigger may use**
BEFORE INSERT	NEW	-
BEFORE UPDATE	OLD, NEW	-
BEFORE DELETE	OLD	-
AFTER INSERT	NEW, NEW_TABLE	NEW_TABLE
AFTER UPDATE	OLD, NEW, OLD_TABLE, NEW_TABLE	OLD_TABLE, NEW_TABLE
AFTER DELETE	OLD, OLD_TABLE	OLD_TABLE

You can use existing triggers on a database to use the system table SYSIBM. Consult SYSTRIGGERS:

SELECT name,text FROM
SYSIBM. SYSTRIGGERS

Example:

Suppose we had a table afps (number of episodes per series), then a trigger for inserting new tuples into the series sequence table might look like this:

```
CREATE TRIGGER neue_folge
    AFTER INSERT ON serienfolge
    REFERENCING NEW AS neu
    FOR EACH ROW MODE DB2SQL
    UPDATE afps
        SET anzahl_folgen = anzahl_folgen + 1
        WHERE neu.serientitel = afps.serientitel
           AND neu.drehjahr = afps.drehjahr
```

And inserting a tuple in series sequence would automatically result in an UPDATE of the table afps:

INSERT INTO afps (serientitel, drehjahr, anzahl_folgen) VALUES ('Babylon 5 - The Telepath Tribes', 0);

INSERT INTO series sequence (serial title, turning year, stag, follow-up, follow-up title)
VALUES ('Babylon 5 - The Telepath Tribes', 2007, 1, 1,

```
        'Forget Byron');

INSERT INTO series sequence (serial title, drehjahr, staffelnr, follownr, follow-up
        title)
VALUES ('Babylon 5 - The Telepath Tribes', 2007, 1, 2,
        'Bester''s Best');

SELECT anzahl_folgen AS Number
FROM afps
WHERE land='Babylon 5 - The Telepath Tribes'

Number
------
     2
```

Another example:

```
CREATE TRIGGER check_ausleihe
    NO CASCADE BEFORE INSERT ON loan
    REFERENCING NEW AS neu
    FOR EACH ROW MODE DB2SQL
    IF neu.ausleihdatum > current date
    THEN
        SIGNAL SQLSTATE '75000'
        SET MESSAGE_TEXT =
          'A loan can't be pre-entered.'; END IF
```

This trigger prohibitsthe insertion of new tuples inborrowed with the loan date in the future. (So only loans that have already taken place should be entered .) The attempt will result in an error and the statement will not be executed.

9 Vorgehen bei der Modellierung

The design of a database describes the process of implementing a mini-world into a database schema capable of representing the desired dates ofthis world with their properties and relationships.

The design that the implementation then follows essentially consists of these steps:

- Modeling
- Conversion into a relational scheme
- Normalization

10 Entwurfsschritt 1: Modellierung

The information in this section should be known from the lecture.

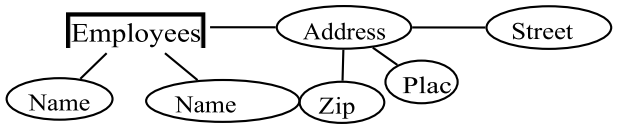
10.1 Modellierung mit E/R-Diagrammen

The modelling serves the systematic representation of an abstraction of the mini-world to be depicted. It is also used to communicate with non-experts, who are usually not used to thinking in relations or attributes. As a way to model a mini-world, the EntityRelationship model is widely used. ERDiagrams are used to represent these models.

10.1.1 Entitäten und Attribute

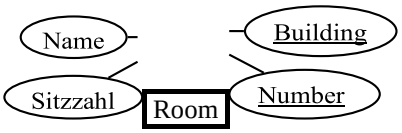
An object of the real world is modeled as an **entity.** An entity is usually a person, process, or real-world object, such as an employee, delivery, inventory item, or document. Similar entities form an **entity type,**e.g. all scientific employees, all auto parts orall employmentcontracts. In an ER diagram, an entity type is represented by a rectangle that encloses the name of the entity type.

An entity (or entity type) is described by a set of **attributes.** In an ER slide,an ellipse encloses the name of the attribute and a line connects it to its entity type. For example, an attribute of employee is the name. One speaks of a **composite attribute,**if in turn different attributes are assignedto it. An ER diagram shows composite attributes exactly as atomic attributes.



Attribute combinations used to uniquely identify an entity are called **keys.** For example, for the Employee entity type, the key (name, first name, address), or (employee number). A key uniquely identifies an entity through the attribute values, specifically, and no two entities can match in all of its key attributes. A key is awarded among all keys, which is called the primary **key**.

An ER diagram underlines the names of aller attributes that belong to the primary key.

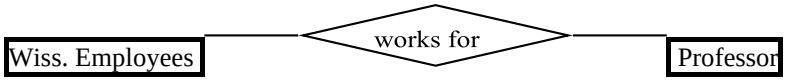


10.1.2 Beziehungen (Relationships)

The objects of the real world are related to each other. Relationships in the ER model model model links, relationships, and interactions between entities, or entity types. Examples are for examplepart-of, works-for, etc.

Relationships of the same kind are grouped intoso-called **relationship types.** In an ER diagram, relationship types are represented by a diamond that contains the name of the relationship type and is associated with all entity types thatparticipate in thatrelationship. An entity type can participate in a relationship multiple times. It is also possible for an entity type to be related to itself (this is called a *recursive* relationship).

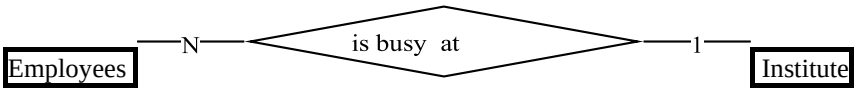
If a relationship type connects only two entity typeseach a time, one speaks of a binary relationship type, as well as ternary relationships, etc.



Relationships can have attributes like entity types. These are displayed in the ERDiagram in the same way as forentity types. Because relationship types model only relationships and relationships, they cannot have identifying keys. The ER diagram does not award keys for relationship types.

For relationship types, it makes sense to understand how often the different entities can participate in a relationship at the modeling stage. These considerations lead to the concept of **functionalities:**

If each entity of a type participates exactly n times in a relationship, the connecting line is marked with that number in the ERslide. If each entity participates in a relationship at least p times and at most k times, theconnecting line is marked with (p,k) in the (*min*,*max*) notation in the ER diagram. The *N* character as the maximum usually indicates that an entity of this type can participate in this relationship as often as you want.



In the ER model, it is not possible for relationships themselves to re-participate in relationships. But this can sometimes be necessary and useful. To do this, you can use a relationship type with the entity types involved in it to become an aggregated

entitytype. The latter, in turn, can participate in relationships. In the ER diagram, an aggregated entity type is represented by including the entity type and relationship type symbols that define it in a rectangle.

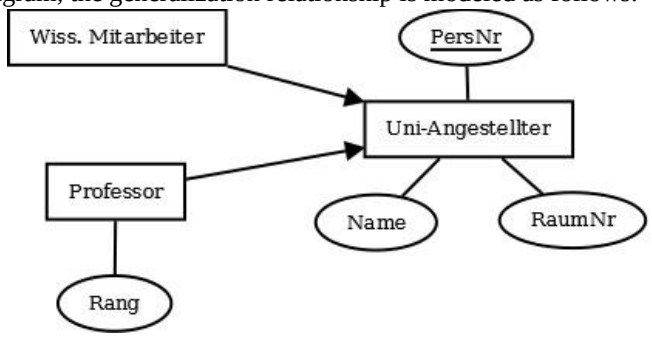
10.1.3 Generalisierung

To better structure the entity types, you can introduce **generalizations** in the design. This object-oriented concept abstracts entity types and forms supertypes. The entity types that have been generalized to an upper type are categories or subtypes of the supertype.

A supertype summarizes common properties for all categories. These are inherited from the subtype; in addition, a subtype can have properties that are characteristic only for it. The entity set of the subtypes is a subset of the entity set of the supertype, although two cases are usually particularly interesting:

- *disjunctute* specialization: the intersection of two subtypes is empty
- *full* specialisation: upper type results as a combination ofsubtypes

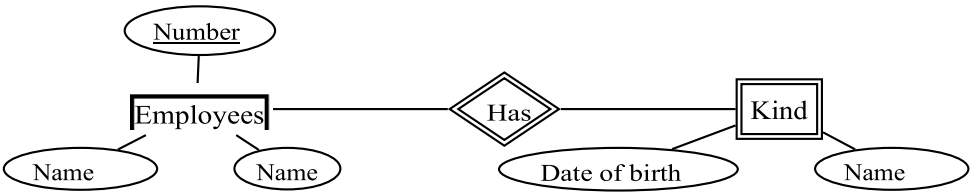
In an ER diagram, the generalization relationship is modeled as follows:



10.1.4 Schwache Entitätstypen

If an entity type does not have its own key candidate, it is called **weak entity types**. You are participating in a relationship with another entity type, the parent entity type. Each
Object of the weak entityst yp takes at least one relationship to the
Parent entity type. The object is identified by this relationship. The primary key of the parent entity type is also used for the weak entity type (possibly extended by a subkey).

Typically, the objects of the sc hwacht entity type dependon the objects of the parent entity type, i.e. without them they are not of interest or have no existence of their own. In the ER diagram, weak entity types are distinguished from normal entity types by the fact that their name is enclosed by a double rectangle. The partial keys are awarded with a dashed underline.



11 Entwurfsschritt 2: Relationenschema

As a second step in the design of a database, the conversion of the model into a relation altogether is pending after modeling. The goal should be a schema that can be implemented as directly as possible in a concrete relational datab.

The information in this section should be known from the lecture.

In general, there are several suitable schematic variants in this implementation. A first approximation for implementation can be asfollows:

- an entity type is converted to a relation with the same attributes as the entity type
- a relationship is transferred to a relationship whose attributes are the keys of the relationships it associates

Normally, however, there are a few other things to keep in mind. Here are a few simple rules of thumb. It should be borne in mind that, with regard to a specific application, it may sometimes be useful to choose a theoretically not optimal implementation.

Relation schemes are specified as described in **Kemper/Eickler:**

SchemaName: [$Attribute_1$: $Type_1$, $Attribute_2$: $Type_2$,...]

The primary key of a relation is characterized by underlining. Attributes within a relation must be uniquely named.

11.1 Entitätentypen

An entity type is usually mapped as a relation whose attributes are the attribute of that type. First of all, this relationship has no connection to the relationships in which the entity type participated.

Rooms : "[Number: integer, Building: string, Number of seats: integer]"

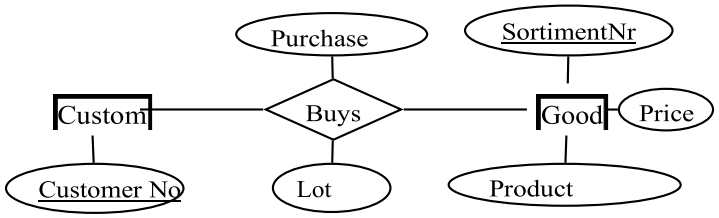
Composite attributes are either resolved, i.e. the attributes of the composite attribute become attributes of the implementation of the entity type, or they are understood as their own relation. This new relation contains all the attributes of the attribute to beset together and all primary key attributes of the origin relation. These foreign keys form the primary key of the new relation.

11.2 Beziehungstypen

A relationship type is usually mapped as a separate relation. As Attribute for this relation are used:

- the attributes of the relationship type
- the primary key attributes of the participating entity types as foreign keys

The primary key of the relation contains the foreign keys and possibly additional attributes.



buys : "[CustomerNo: integer, SortimentNr: integer,
Purchase date: date, Quantity: integer]"

If an entity type participates in a relationship multiple times, its key attributes must be transferred to the relationship that corresponds to that relationship. You have to rename the attributes to avoid name conflicts.

11.3 Schwache Entitätstypen

A weak entity type is mapped as a separate relation that, in addition to its own attributes, also contains the primary key of the parent entity type as a foreign key. This, together with the excellent subkeys of the weak entity type, forms the primary key of the relation.

In general, the relationship between a weak entity type and the parent entity type does not need to be replicated at all.

11.4 Zusammenführen von Relationen

The directe transfer as described above often does not provide the best possible relations. Often you can now merge relations. As a general rule, relations with the same key can be summarized (but only these).

A binary relationshipyp R between entity types E and F, in which at least one of the two entity types involved (say E) with functionality 1 participates, can be merged with the E corresponding relation. For this purpose, a relation is entered withthe following attributene:

- the attributes of E
- the primary key attributes of F
- the attributes of the relationship R itself considers why this is more advantageous than the transfer of the relationship R into its own relation. also consider what counter-arguments there are.

12 Entwurfsschritt 3: Normalformen

Normal forms provide a theoretische basis for evaluating relational database schemas. In particular, normalization is intended to prevent redundancy and implicit presentation of information. By maintaining the normal forms in the modeling,change, insertion or deletion anomalies can be prevented during later use.

The information in this section should be known from the lecture.

12.1 Erste Normalform

A relation is in *first normal form (1NF)*if all attributesare allowed to take onlyat omare values. Compound or quantity attributes are not allowed under 1NF. In classic relational DBMS, it is
Do not save relations that are not in 1NF. The relation

Examination: 'Matrikel, Name, Fachbereich, FBNr, 'Examination No, Prfnote'

is not in 1NF, as there is a collective and quantity-valued attribute 'Examination No, Prfnote'.

Disadvantages of a relation that is not in 1NF include a lack of symmetry, storage of redundant data, andac tualization anomalies. New redundancies resulting from the transfer to 1NF will be eliminated in the course of further normalization.

Approach:

- List or quantity attributes are replaced by their elements. At the tuple level, this is achieved bymultiplying the list with the remaining tuple.
- Composite attributes are replaced by the sub-attributes.
- Alternatively, new relations can be formed.

This provides the relation

Examination: 'Matrikel, Name, Department, FBNr, Examination No, Prfnote'

in which a separate tuple exists for each examination with an examination grade, i.e. a separate row in the table.

12.2 Zweite Normalform

A relation is in *second normal form (2NF)*if each non-key attribute is fully functionally dependent on each key. Relationships that violate this normal form contain information about two entity types, resulting in change anomalies. When transferred to 2NF, the different types of entitäts are separated.

The following relation with the functional dependencies*matriculations* - *departmentPrfnote* , is not in 2NF, since name, department and FBNr only*matriculations* , *FBNr*, *department* - *FBNrMatrikel*and"*matriculation,name*,

exam no. . . . . . .

. . . .

Examination: "Matrikel, Name, Department, FBNr, Exam No, Prfnote"

Procedure:

- Determination of functional dependencies between key and non-key attributes as shown in the lecture
- Elimination of partative functional dependencies, by transferring the dependent attributes together with the key attributes into a separate relation and removal of the dependent attributes from the original relation.

This method provides the relations:

Student: "Matrikel, Name, Department, FBNr"

Exam: "Matrikel, Exam No, Prfnote"

12.3 Dritte Normalform

A relation is in *third normal form (3NF)*if each non-key attribute is not transitively dependent on any key. In the relation Student, the transitive dependence *matriculation* is present, it is therefore not in 3NF.

Approach:

- Determination of functional and transitive dependencies
- For each transitive functional dependency *A* . . . . . .

  . . . . . .

This method provides the relations:

Student: "Matrikel, Name, Department"

I am re-editing:"Department of Department, FBNr"

13 Arbeiten mit IBM DB2

13.1 Struktur der DB2-Installation

Within a DB2 installation, the followingimportant database system objects exist:

- Instances
- Databases
- Schemas
- Tables
- Views

For instances and schemas, reference is made to the week 1 introduction.

13.1.1 Datenbank

A relational database is a collection of tables. A table consists of a fixed number of columns corresponding to the attributes of the relation schema, as well as any number ofrows or tuples. Each database includes system tables that describe the logical and physical structure of the data, a configuration file with the parameters of the database, and a transaction log. A new database is created with the DB2 create database command .

Created. This creates a database with the default configuration. For the purposes of the internship, it should not be necessary to make changes to the default configuration. The database can be deleted again with the DB2 command drop database 'name'

(Attention: the command is usually irreversible and deletes the entire contents of the database).

As discussed in the introduction, the DB2 connect to 'name' command connects to a name database that terminates with terminate.

13.1.2 Tabellen und Sichten

A relational database stores data as a set of two-dimensional tables. Each column of the table represents an attribute of the relation schema, and each row corresponds to a specific instance ofthat schema. Data in these tables is manipulated using the Structured Query Language (SQL), a standardized language for defining and manipulating data in relational databases. Views are persistent requests thatcan be accessed in other requests such as atable.

In order to display the existing tables and views in an existing database, you can use the DB2 command list tables for schema .name.

Using the DB2 command describe table . . . . . you can find out details about the structure of an existing table. Creating tables and views will be covered in detail in later chapters.

13.2 Benutzerverwaltung

Db2's authentication consignmentdoes not provide for separate user names, but builds on the operating system's user concept. Each small group logs on to the database under the name of their account, in addition, each small group belongs to the user group students.

In a DB2 instance, there are three levels of access permission. As an instance owner, each group in its instance has the SYSADM right.

SYSADM: Access all data and resources of the instance

SYSCTRL: Manage the instance, but no direct trainriff ondata

SYSMAINT: lowest permission level; can, for example, create backups

To set or change permissions in the database to which you are currently connected, use the GRANT statement. Conversely, the REVOKE stateismisare t is used to revoke rights on a database. GET AUTHORIZATIONS displays the current permissions. Examples:

GRANT dbadm ON DATABASE TO USER dbpw0301
GRANT connect, createtab ON DATABASE TO GROUP students
GRANT connect ON DATABASE TO public
REVOKE connect ON DATABASE TO public

A user or group with the DBADM right automatically also has all other rights on this database, in particularthe right IMPLICIT_SCHEMA. The dBADM right cannot be granted to PUBLIC. Only a user with SYSADM rights can grant or revoke the DBADM right to other users or groups. If a useris deprived of a right, it necessarily means that the userno longer has the right, since rights can also exist by belonging to groups.

Anyone who has the dBADM or SYSADM right can change permissions on a schema. A user may also be given the right to pass on the rights obtained to third parties. The command to assign or
Removing rights to schemas is GRANT/REVOKE ... ON SCHEMA.

13.3 DB2: Der Datenbank-Manager

A database manager instance manages the system resources for theservers that belong to a specific system account. This is started with the db2start shell command or the DB2 start database manager command. To conserve the computer's resources, the instance should be paused if your long timeis not working withit: to do this, use the db2stop shell command or the DB2 stop database managercommand.

In DB2, you can set configuration values at different access levels that regulate the default behavior of an instance or a dataobject.

While a Database Manager instance is running, you can view its configuration as follows:

get dbm cfg bzw. (db2) get database manager configuration

A configuration value is changed for a (running) instance as follows:

(db2) update dbm cfg using *config-keyword value*

Here, config-keyword stands for a property of the configuration (e.g. AUTHENTICATION), and *value* for the new value.

If you do not explicitly specify otherwise, in most cases the Database Manager instance must be restarted to make the configuration change active.

14 DB2: SQL als DDL

A relational database stores data as a set of two-dimensional tables. Each column of the table represents an attribute of the relation schema, each row corresponds to a specific expression of this relation scheme.

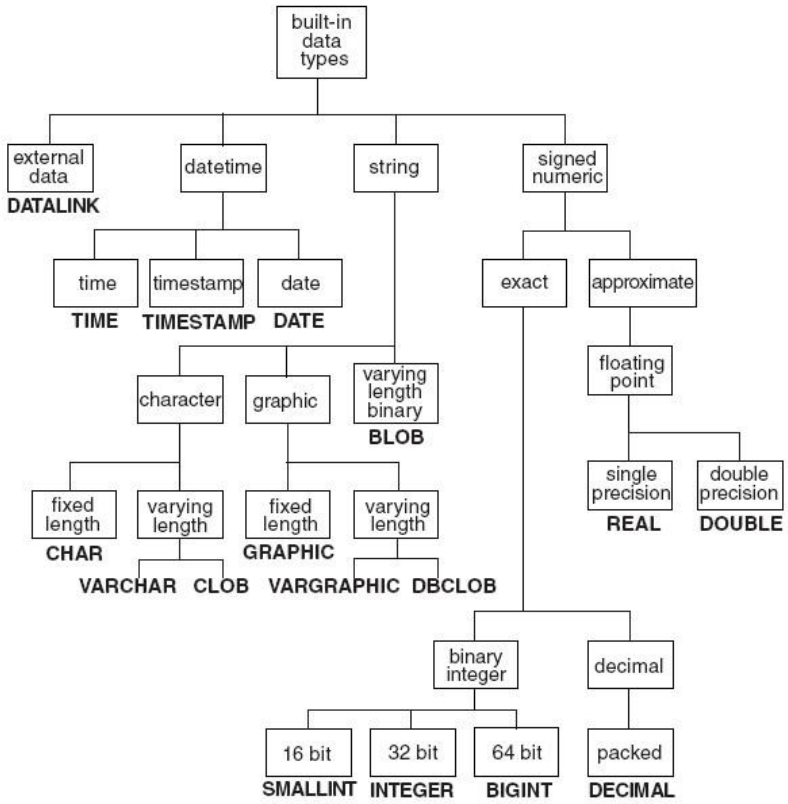
In relational database systems such as DB2, Structured Query Language (SQL) has several roles:

- Datendefinition – *Data Definition Language, DDL*
  Definition of table structures: Create, customize, and delete tables with their attributes, data types, and consistency conditions
- Datenmanipulation – *Data Manipulation Language, DML*
  Manipulating table contents: Inserting, deleting, and modifying records
- Anfragesprache – *Query Language*
  Select records according to specific selection criteria
- Zugriffskontrolle – *Data Control Language, DCL*
  We already got to know this rolein the last week in the form of the granting of rights by means of GRANT.

In the first block, SQL was used as the request language. In this block, we'll look at the other three roles of SQL.

14.1 Datentypen

DB2 knows the data types shown in the graphicbelow. More
Data types can be defined by the user — keyword: expandable databases (but this is no longer covered as part of the internship). Note that there is no special data type BOOLEAN for truth values.



SMALLINT	kleine Ganzzahl
INTEGER	Ganzzahl
BIGINT	große Ganzzahl
REAL	Fließkommazahl mit einfacher Genauigkeit
DOUBLE	Fließkommazahl mit doppelter Genauigkeit
DECIMAL(*p*,*s*)	Decimal fraction of length *p* with *s* decimal places CHAR(*n*) String oflength *n* (max. 254)
VARCHAR(*n*)	String variabler Länge mit maximal *n* Zeichen (max. 32672)
BLOB(*sF*)	Binary Large Object (e.g. BLOB(2 M) for a 2MB BLOB
DATE	Datum
TIME	Time
TIMESTAMP	Zeitstempel
DATALINK	Verweis auf eine Datei außerhalb der Datenbank

Each data type has a null value. This is distinguishable from all other values of the data type because it does not represent a value per se, but indicates the absence of a value. For example, a value of 0 inan employee's field could mean that the employee is a volunteer, while the null value could mean that the salary is not known. IBM DB2 offers predicates that allow, for example, in requests to check whether thereis a NULLWho t or another form of the datatype.

Default values are values entered by the DBMS if no value is specified for that attribute. They provide the ability to automatically add data to attributes, such as a specific constant, user name (USER), current time (CURRENT TIME), current DATE, or auto-generated values (GENERATE). If no specific default value is specified, the null value is used as the default value.

14.2 Schemata (DB2-spezifisch)

A schema is a collection of named objects (tables, views, triggers, functions,...). Each object in the database is in a schema. Object names must be unique within a schema, so aschema is similar to a namespace. A new schema can be created using the DB2 create schema command, and can be deleted using drop schema .name.

A user who has database permission IMPLICIT_SCHEMA can implicitly create a schema if they use a schema that does not exist in a CREATE statement. This new schema, created by DB2, belongs to SYSTEM by default, and every user has the right to create objects in that schema.

14.3 Erstellen von Tabellen

Each team can create new tables based on the predetermined rights in the databases they create, possibly with implicit creation of a new schema (see above). The CREATE T ABLE statement is used to create atable. The user who creates the table automatically receives the CONTROL right on that table.

When you create a table, a primary key is also specified. A primary key consists of the specified attributes that cannot allow null values (NOT NULL must be explicitly specified for each attribute of the primary key) and the limitations on possible data types apply. DB2 automatically creates an index on the primary key. Each table can have only one primary key, but it can consist of multiple attributes.

```
CREATE TABLE employee (
    id          SMALLINT NOT NULL,
    name        VARCHAR(50), department
     SMALLINT, job  CHAR(10), hiredate
     DATE, salary       DECIMAL(7,2),
    PRIMARY KEY (ID)
)
```

PRIMARY KEY sets a primary key for the table. This consists of the specified attributes. The attributes in the primary key must not allow null values. If the key consists of only one attribute, it is sufficient to specify the keyword after the respective attribute:

```
CREATE TABLE employee (
    id                  SMALLINT NOT NULL PRIMARY KEY,
...
```

As a shortcut, you can also create tables with the following statement:

CREATE TABLE name LIKE other_table

This inherits the attributes of the table with the exact same names and definitions from another table or view.

Tables with the DROP statement are deleted. This requires the CONTROL right for the table, the DROPIN right inthe table, the DBADM or the SYSADM right. If a table is deleted, all tables that reference that table delete the associated foreign key relationships. All indexes that exist on the table are deleted.

14.4 Tabellenerstellung: Schlüssel und Constraints

The consistency conditions described below can be specified directly when creating a table as part of the CREATE TABLEcommand.

14.4.1 Primärschlüssel

PRIMARY KEY sets a primary key for the table. This consists of the specified attributes. The attributes in the primary key must not allow null values. If the key consists of only one attribute, it is sufficient to specify the keyword after the respective attribute: id SMALLINT NOT NULL PRIMARY KEY,

It can be helpful here if the values of the primary key as IDENTITY are automatically generated by the system.

id SMALLINT NOT NULL GENERATED ALWAYS AS IDENTITY,

The GENERATED ALWAYS AS IDENTITY keywords will always generate a uniquevalue forthis attribute. The database system expects that no values are specified for this attribute when inserting data - otherwise it throws an error. Alternatively, you can set GENERATED BY DEFAULT AS IDENTITY one, so that the system generatesa unique key value only in the absence ofpredetermined values.

DB2 allows such auto-generated keys for INTEGER, SMALLINT, BIGINT or DECIMAL (without decimal places). It is also possible tospecify thevalue with which the automatically generated keys start and in which steps they should be incremented:

```
id    SMALLINT NOT NULL GENERATED ALWAYS AS IDENTITY
                        (START WITH 1, INCREMENT BY 1, NO CACHE),
```

14.4.2 Sekundärschlüssel

The UNIQUE keyword sets a secondary key. An index is automatically created for this one, too, and the same restrictions apply as for primary keys. Attributes of a secondary key cannot already be part of another key. Example: jobnr INTEGER NOT NULL UNIQUE,

14.4.3 Fremdschlüssel

Foreign keys are used to force one or more attributes of a dependent table to take only values that also occur as values of one key from another table. This concept is also called *Referential Integrity.* The user must have the REFERENCES or a parent right on the referenced table. The referenced table must exist and the reclaimedattributes must be defined as keys there.

Foreign keys are specified either as *constraint* (see below) or with the REFERENCES keyword. In this example, the jobnr attribute references the column internal_number the jobs table: jobnr INTEGER REFERENCES jobs (internal_number)

14.4.4 Checks

When defining atabular enattribute, a constraint of the values can be specified in addition to the data type. This is initiated with the keyword CHECK:

department SMALLINT CHECK (department BETWEEN 1 AND 5), job

CHAR(10) CHECK (job in ('Professor','WiMi','NichtWiMi')),

14.4.5 Constraints

Checks and *foreign keys* can also be specified as named *constraints* with the CONSTRAINT keyword:

CONSTRAINT jobref FOREIGN KEY jobnr REFERENCES jobs (int_no)

CONSTRAINT yearsal CHECK (YEAR(hiredate) > 1996 OR SALARY > 1500)

14.4.6 Beispiel

An example of creating a table with multiple consistency conditions would be:

```
CREATE TABLE employee (
     id                SMALLINT NOT NULL,
     name              VARCHAR(50),
     department        SMALLINT CHECK (department BETWEEN 1 AND 5), job
      CHAR(10) CHECK (job in ('Prof','WiMi','NiWiMi')),
     hiredate DATE, salary
     DECIMAL(7,2),

     PRIMARY KEY (ID),
     CONSTRAINT yearsal CHECK (YEAR(hiredate) > 1996
                                        OR salary > 1500)
)
```

14.5 Index

An index isa database object designed to speed up access through certain attributes of a table, and in the vast majority of cases it does. You can also use (UNIQUE) indexes to ensure compliance with key conditions.

If thetable definition specifies primary key (PRIMARY KEY) or secondary key conditions (UNIQUE), DB2 independently creates indexes to ensure compliance with them. Once created index is then automatically maintained in case of changes to the contents of the corresponding table. This, of course, entails an increased administrative burden.

More information about indexes and the data structures used for the realization can be found in the lecture **databases** or in the accompanying literatur.

The explicit creation of an index goes with the CREATE INDEX statement, the deletion accordingly via DROP INDEX. You need at least the INDEX or CONTROL right for the table, as well as the CREATEIN right in the specified schema. The userwho creates an index receives the CONTROL right on it.

the UNIQUE keyword allows you to specify when creating an index that no tuples may match in all values of the specified attributes. If tupules in the table violate this at the time ofindegeneration, an error message is issued and the index is not created.

It is not possible to create two equal indexes. An index 16 may be
attributes and the attributes together cannot be longer than 1024 bytes. The asC or DESC order indicates whether the index entries are created in ascending or descendingsequences. Only if UNIQUE is specified can additional attributes be included in the index using the INCLUDE clause, but no key condition should apply.

Example:

```
CREATE INDEX addresses
    ON user (address)
```

Indexes are not called explicitly, but are implicitly used by the database management system when processing requests.

14.6 Verändern von Tabellendefinitionen

To change a table definition, the ALTER or CONTROL right to changethe table, the ALTERIN right for the schema of the corresponding table, or a parent right are required. In order to introduce or remove a foreign key relationship, the REFERENCEd table also requires the REFERENCES, CONTROL, DBADM or SYSADM right. Similarly, when deleting a key condition, the initially named rights must exist for all tables referencing the associated attributes.

A table definition is changed by an ALTER TABLEtable:

```
ALTER TABLE example.employee
    ALTER name SET DATA TYPE VARCHAR (100)
    ADD hasemail CHAR(1)
```

```
    DROP CONSTRAINT yearsal
```

The example demonstrates the three possible changes in an ALTER TABLE statement for the (hypothetical) employee table in the EXAMPLE schema:

- an attribute definition is modified using the ALTER clause, VARCHAR attributes can be enlarged (but not reduced), and this clause can change the expression to generate auto-generated attribute values.
- a new attribute is added

  When using the ADD clause, the same applies to creating a new table. If a new key condition is added and there is no index for it, a new index is created. The ADD COLUMN clause is always executed first.
- a named CONSTRAINT is deleted from the table

  You cannot delete attributes or constrants or checks that were specified directly and without identifiers in an attribute definition.

15 Pflege von Tabelleninhalten

15.1 Einfügen von Tupeln

Inserting new tuples into an existing table is done with the
INSERT statement. The values of the tuples to be inserted must fit the respective attributes in terms of data type andlength. In addition, the tuples to be inserted must meet all constraints (key conditions, foreign keys, and checks). If it is inserted into a view, it must allow it to do so.

If no value is entered for an attribute in the INSERT statement, the
(falls present) the default value or the null value. A value must be specified for each attribute that does not have a default value and does not allow null values.

```
INSERT INTO category (cat_id, cat_name)
VALUES
     (4, 'Fiction'), (9, 'Poetry')

INSERT INTO werke
     SELECT titel, year
     FROM work
```

15.2 Löschen von Tupeln

Deleting tuples is done with the DELETE statement. This deletes all tuples from the current view that meet the search condition. This requires the DELETE right orthe CONTROL right on the table/view, the DBADM or the SYSADM right. If an error occurs during the deletion of a tuple, **all** deletions of this statement shall remain ineffective.

If no search condition is specified, **all** tuples are deleted.

```
DELETE FROM anwender
        WHERE nick='Hase'

DELETE FROM work
        WHERE titel NOT LIKE '%wizard%of%'
          OR year>2000
```

15.3 Ändern von Tupeln

Changing tuples is done with the UPDATE statement. Similar to the DELETE statement, a search condition specifies which tuples of the specified view/table should be changed. Again, the necessary rights must be given: the UPDATE-Recht on all attributes to be changed, the UPDATE right on the table/view, the CONTROL right on the table/view, the DBADM right or the SYSADM right.

```
UPDATE enthaelt
   SET language = UCASE (language)

UPDATE user
   SET address =
      (SELECT address FROM user
          WHERE name='Mouse' AND vorname='Minnie')
   WHERE name='Mouse' AND vorname='Mickey'
```

16 Füllen von Tabellen

16.1 Export und Import

From an existing table you can export data to one file and later insert that data back into another table. The EXPORT statement is used.

When exported, the generated file can either be a text file with limiters from which a table can be created in another application, or a file in the IXF (*integrated exchange format*). The latter format can contain additional information about the schema of the table.

When a table is exported, you enter in addition to the export file
SELECT statement. The simplest SELECT statement SELECT * FROM

schema.table

exports a complete table with all rows and columns. Two sample export commands with selection of specific columns and rows of the table look something like this:

```
EXPORT TO werke.ixf OF ixf
   SELECT title,year
   FROM work
   WHERE titel like 'B%'
```

```
EXPORT TO myfile.del OF del modified by
   chardel" coldel;
   SELECT cat_ID, cat_name
   FROM category
```

Similarly, an exported file can also be imported again. This is the easiest way to dothis for tables previously exported as IXF files. It makes sense to collect all the data you need in a table as early as export. You can use the IMPORT statement to import.

```
IMPORT FROM werke.ixf OF ixf
   INSERT INTO works
```

Anotheroption is the more powerful and convenient LOADStatement, which will be covered in the next section.

16.2 Load

The pooling of data into a table from any text file can be done with the IMPORT or LOAD commands. The file name must be specified, the file type (ASC = text file without limiter, DEL = text file with limiter, IXF), the log file andd additional options. The exact syntax and possible options are described in the manual.

When pooling text files (not IXF files), you can use the following import modes:

INSERT**:** Adding the new tuples

INSERT_UPDATE**:** Add the nyourtuple, or change old tuples with the same primary key

REPLACE**:** Delete all old tuples, insert the new tuples

IXF files contain additional schema information. Here there are also the modes REPLACE_CREATE and CREATE, with which a table is created ifnecessary. To pool data into a table, the necessary rights must be set.

Suppose there is a table category with the columns cat_id INTEGER NOT NULL, cat_name VARCHAR(40) NOT NULL and top_cat INTEGER, as well as eine text file with category name, category number, and top category of categories, each separated by a .|'. Then the following LOADStatement would select the 2nd, 1st and 3rd elements from each line of the text file and use this data toadd thetable:

```
LOAD FROM "textdatei" OF DEL MODIFIED BY COLDEL|
      METHOD P (2,1,3)
      MESSAGES "logfile"
      INSERT INTO category
        (cat_id, cat_name, top_cat)
```

Thus, this import uses all values of each row in reversed order (to match the series sequenceof the table attributes). INSERT adds the data to the table instead of overwriting the existing data.

16.2.1 Tabellenintegrität

The LOADoperation can place the table in the CHECK PENDING state when filling a target table with generatedcolumns. To restore the integrity of generated columns, SET INTEGRITY must be executed. SET INTEGRITY FOR tablename IMMEDIATE CHECKED FORCE GENERATED

If erroneous rows are found after importing the tables, they must be removed before you can continue working with the table. Such erroneous rows can occur when importedif the imported data violates the health conditions. You can check the integrity by using the following DB2 command that ejects the first violation:

SET INTEGRITY FOR tablename IMMEDIATE CHECKED

If there are integrity errors in the table due to the import, you can either empty them (e.g. delete and recreate or by removing all tuples) or repair them. If you want to repair the table, you need an error table to which the rowsn are moved, which violate the integrity conditions. This table must have the same structure as the actual table, which can be seen, for example, by

CREATE TABLE exception_table LIKE table

error table **k**may have**constraints** or generated values! Possible co-copied constraints (referential constraints, checks, or keys) may need to be deleted before the error table can be used. In addition, the error table cancontain two additional columns, of which the first of thetimestamps is recorded (data type TIMESTAMP) and in the second the name of the integrity condition violated by the tuple (data type CLOB(32K)).

Now you can call SET INTEGRITY with the table:

```
SET INTEGRITY FOR tablename IMMEDIATE CHECKED
      FOR EXCEPTION IN tablename USE exception_table
```

The SET INTEGRITY command also allows you to completely turn off integrity checks. This is useful, for example, if youwant to add new constraints, as in the following example:

```
SET INTEGRITY FOR table OFF;
ALTER TABLE table ADD CHECK (something < other);
ALTER TABLE table ADD FOREIGN KEY (attribute) REFERENCES other_table;
SET INTEGRITY FOR table IMMEDIATE CHECKED;
```

16.3 Optionen für IDENTITY-Spalten

If IDENTITYcolumns are defined in the targettable, you can IMPORTand LOADcommands explicitly specify how to handlethese columns.

Possible options are:

- IDENTITYMISSING —This option states thatthe source data does not contain itsown IDENTITYcolumn. This means that the system automatically generates matching new values when inserted.

  IMPORT FROM werke.ixf OF ixf

```
MODIFIED BY IDENTITYMISSING
INSERT INTO works
```

- IDENTITYIGNORE —This option contains IDs in the source data, but they are ignored and replaced with new system-generated ones.
- IDENTITYOVERRIDE —Here, the IDs are inherited from the source data, even if they already exist in the destination table. Because this may allow existing data to be replaced or modified, this option is only available with the LOADcommand.

```
LOAD FROM "textdatei" OF DEL
  MODIFIED BY IDENTITYOVERRIDE
    INSERT INTO works
      (title,year)
```

For more information about IDENTITYcolumns and their connection to IMPORT, EXPORT , and LOAD, please visit:

http://www.ibm.com/developerworks/db2/library/techarticle/0205pilaka/ 0205pilaka2.html

17 Vordefinierte Funktionen

When working with IBM DB2-SQL, there are a number of predefined functions that can be used to manipulate values. These are divided into two types:

Scalar functions: evaluation of a list of scalar parameters with return of a scalar value; used in expressions

Aggregate functions: Application on columns of a group or a relation with return of a scalar value; are used e.g. in cases

You can also create custom functions of the two mentionedareas, as well as table functions that return whole relations. These can then be used just like the system functions. For all functions, care mustbe taken to ensure that the data types they return depend on the parametertypes. Many of the functions are overloaded, i.e. defined for different data types.

17.1 Skalare Funktionen

The most important scalar functions are mentioned here, for the definition and detailed description of the functions, please refer to the manual oof the online documentation. http://publib.boulder.ibm.com/infocenter/db2luw/v9/

Typkonvertierung: BIGINT, BLOB, CHAR, CLOB, DATE, DBCLOB, DECIMAL, DREF, DOUBLE, FLOAT, GRAPHIC, INTEGER, LONG, LONG_VARCHAR, LONG_VARGRAPHIC, REAL, SMALLINT, TIME, TIMESTAMP, VARCHAR, VARGRAPHIC

Mathematik: ABS, ACOS, ASIN, ATAN, CEIL, COS, COT, DEGREES, EXP, FLOOR, LN, LOG, LOG10, MOD, POWER, RADIANS, RAND, ROUND, SIGN, SIN, SQRT, TAN, TRUNC

Stringmanipulation: ASCII, CHR, CONCAT, DIFFERENCE, DIGITS, HEX, INSERT, LCASE, LEFT, LENGTH, LOCATE, LTRIM, POSSTR, REPEAT, REPLACE, RIGHT, RTRIM, SOUNDEX, SPACE, SUBSTR, UCASE, TRANSLATE

Datumsmanipulation: DAY, DAYNAME, DAYOFWEEK, DAYOFYEAR, DAYS, HOUR, JULIAN_DAY, MICROSECOND, MIDNIGHT_SECONDS, MINUTE, MONTH, MONTHNAME, QUARTER, SECOND, TIMESTAMP_ISO, TIMESTAMPDIFF, WEEK, YEAR

System: EVENT_MON_STATE, NODENUMBER, PARTITION, RAISE_ERROR, TABLE_NAME, TABLE_SCHEMA, TYPE_ID, TYPE_NAME, TYPE_SCHEMA

Other: COALESCE, GENERATE_UNIQUE, NULLIF, VALUE

18 Ausdrücke

Expressions are used in SELECT clauses and search conditions to represent values. They are composed of one or more *operands*,parentheses and non-binary or binary *operators:* + (sum, positive value), - (difference, negative value), * (product), / (quotient) and || (string concatenation).

As operands are allowed:

Attribute names: possibly qualified by table names or tupel variables

Constants: integers (12, -10), decimals (2.7, -3.24), floating point numbers (-12E3, 3.2E-12), strings ('hello'), dates ('12/25/1998', '25.12.1998','1998-12-25'), time values ('13.50.00', '13:50', 1:50 PM'), time stamp ('2001-01-05-12.00.00.000000')

Functions

Durations: for arithmetitic operations with date, time or time stamp values, time durations can be used, e.g. 5 DAYS, 1 HOUR (MONTH/S, DAY/S, HOUR/S, SECOND/S, MICROSECOND/S)

Register values: This includes

CURRENT DATE: current date

CURRENT TIME: current time

CURRENT TIMESTAMP: current timestamp

CURRENT TIMEZONE: current time zone

CURRENT NODE: current node group number

CURRENT SCHEMA: current schema

CURRENT SERVER: Servername

USER: Username

```
SELECT title
FROM book
WHERE YEAR(pubyear) > 1980
```

Type conversions: Explicit conversion to a specific type, e.g. CAST (NULL AS VARCHAR(20)) or CAST (salary*1.2345 AS DECIMAL(9,2))

Subqueries: a request always returns a scalar value, then you can use it as an operand

Case distinctions: e.g.

```
CASE format
      WHEN 'Hardcover' THEN 'Bound Book'
      WHEN 'Mass Market Paperback' THEN 'Small-
            format paperback'
      WHEN 'Paperback' THEN 'Pocket Book'
END

CASE
      WHEN YEAR(pubyear) > 1945 THEN 'Nach WK II'
      WHEN YEAR(pubyear) > 2008 THEN 'Huch!?'
ELSE 'Before the end of the 2nd WK'
END
```

The conditions are evaluated sequentially. Result is the first case to be evaluated as true, otherwise the value in the ELSE clause. If there is no ELSE clause, the default value is NULL. The result expression must allow the data type of the total expression to be determined, in particular, the result types of each expression must be compatible and not all must be null.

19 Prädikate

Predicates are logical tests that can be used in search conditions. You can be *considered true*, *false* , and *unknown* . The known predicates are:

- Comparison predicates: =, <>, <, >, <=, >= (comparisons with null values result *in unknown*)
- NOT as a denial of a predicate
- BETWEEN, z.B. runtime BETWEEN 1 AND 3
- NULL, z.B. dateofdeath IS NOT NULL

- LIKE, string comparison with placeholders _ for a character or % for any number: name LIKE 'Schr_der%' would be e.g. on 'Schröder', 'Schrader' or 'Schrederbein' fit
- EXISTS, test for empty quantity
- IN, Test for occurrences in a collection ofproducts

It is possible to quantify comparison predicates by SOME, ANY or ALL and thus compare a single tuple with a set of tuples.
Examples:

```
category IN ('Fiction','Mythology','Non-Fiction','Poetry')

('2002,'Reaper Man') = ANY (SELECT year(pubyear), title
                                        FROM book)

EXISTS (SELECT *
          FROM book
          WHERE pubyear < 2000)

1000000000 > ALL (SELECT population FROM land)
```

Transactions

A new concept is now to be considered as additional transactions. A transaction is a summary of database operations, in the extreme case of exactly one operation. Transactions in databases are characterized by the so-called **ACID**properties. This means that a transaction should satisfy the four following properties:

A tomicity (Atomarität),

C onsistency (consistency),

I solation, and

D urability

Thus, a transaction should take effect either as a whole or not at all; the correct execution of a transaction transfers the database from one consistent state to another; each transaction should be unaffected by other transactions; and the changes of an effective transaction must no longer be lost.

In SQL, there are several linguistic constructs to support transactions. In general, we have used DB2 so far that all statements are automatically handled implicitly as transactions. This is achieved by the -c (*auto commit*) option when callingthe command *line processor* (CLPs). Any changes to each SQL statement are **immediately** written through and effective.

If you call the CLP with +c, the *auto commit* will be deactivated. Now the following applies:

- Each read or write access to the database implicitly starts a new transaction.

- All subsequent reads or writes belong to this transaction.
- Any changes within this transaction will only apply provisionally.
- The commit command writes the changes to the database. At this point, integrity conditions must be adhered to. In the meantime, an integrity condition can also be violated.
- Commit makes thev on the following commands changes take effect: ALTER, COMMENT ON, CREATE, DELETE, DROP, GRANT, INSERT, LOCK TABLE, REVOKE, SET INTEGRITY, SET transition variable and UPDATE.
- As long as this transaction has not been written through, the rollback command will reverse all changes since the last **commit** as a whole.

When working on a database, such as when multiple users access the date bank and modify or insert data from a Web page at the same time, the competition betweentwo transactions can cause problems or anomalies. A typical example is concurrent changes to a table by two independent users:

Phantom tuple: Within one transaction, you get additional tuples on the same request for the second execution.

Non-repeatability: Reading a tuple produces different results within a transaction.

"Dirty" reading: Reading a tuple not yetwritten.

Lost Changes: Two simultaneous changes overwrite one by the other and are lost.

Therefore, in DB2-SQL, you can set the degree of isolation you are working with by using the CHANGE ISOLATION command **before** connecting to a database. There are four degrees of insulation that protect against the above-mentioned abnormalities in differentways:

	RR	RS	CS	Born
Phantomtupel	No	And	And	And
Nichtwiederholb.	No	No	And	And
Dirty reading	No	No	No	And
Lost Update	No	No	No	No

In addition, tables or an entire database can be explicitly locked by using the LOCK TABLE command or by connecting to the database by specifying the lock mode. SHARE locks are the default and mean that no onecan request an EXCLUSIVE lock. Examples:

LOCK TABLE land IN EXCLUSIVE MODE; CONNECT

TO almanach IN SHARE MODE USER username;

20 Webanwendungen in Java mit JSPs

20.1 Idee und Hintergrund

A web application (Webapp) is a program that runs on the serverside and communicates with the user over HTTP. Often, the communication is done on the user side through a web browser that sends requests to the web app by calling specific URLs and displays the responses of the web app as HTML pages.

If you want to implement in Java, you usually write code that runs in a servlet container, such as Apache Tomcat, at the end. To do this, you extend a class HttpServlet and calculate the response that is transmitted to the user asan addition to the stringsequence:

```
StringBuffer out = new StringBuffer (); out
.append("<html>" );
out .append("<head>" ).append("<title>Titel</title>" ).append("</head>" ); out .append("<body>" );
out .append("<h1>überschrift </h1>" ); String result = calculate (); out .append("<p>Ergebnis :□"
).append( result ).append("</p>" ); out .append("</body>" ); out .append("</html>" );
```

Listing 1: example.java

1
2
3
4
5
6
7
8
9

This method has several drawbacks. One is that the HTML is scattered in small fragments between a lot of Java code and thus the connection of the individual HTML parts is lost. Another drawback is that the division of labor between Javadevelopers and designers or front-end developers (who is often solely responsible for HTML) is poorly supported: if the front-end developer has submitted his HTML, the Java developer has to painstakingly cut the page tointegrate it into the Javacode. If a change to the HTML has to be made afterwards (e.g. the customer wants a different color), it is sometimes very difficult to understand the change in the JavaHTML mix and the code may have to be completely rewritten.

In order to avoid these disadvantages (and the relatively complex definition of servlets via
XML files), Java Server Pages (JSP) were developed. The basic idea is to embed Java code in HTML. The above example in JSP is as follows:

Listing 2: example.jsp

```
<html>
<head><title>Titel</title></head> <body>


<h1>heading</h1>
<p>Ergebnis :<%=calculate()%></p>
</body>
</html>
```

Such a JSP file is stored directly in the Servlet container and automatically converted into Java code when called by a user, translated and used as a servlet without the user or the developer noticing.

20.2 Webapps schreiben

20.2.1 JSP

The first time a JSP page is called up, a servlet is generated (automatically) from the contents of the .jspfile in a first step. This is a Java class that adds logic to an upper class. The logik is inserted into a doGet() method. This includes assemly the output.

The above test JSP simplifies the following servlet class:

```
package jsp_servlet ;
import java . util .*; /*...*/

/*1*/
 class _myservlet
implements javax . servlet . Servlet , javax . servlet . jsp . HttpJspPage {
/*2*/
public void _jspService( javax . servlet . http . HttpServletRequest request ,
    javax . servlet . http . HttpServletResponse response )
    throws javax . servlet . ServletException , java . io . IOException
    { javax . servlet . jsp . JspWriter out = pageContext .getOut ();  HttpSession session
        = request . getSession ( true );  try {/*3*/
            Out. print ( "Hello ,'World.' );
        } catch ( Exception _exception ) {
            /*  ...  */
        }
    }
}
```

Listing 3: example.java

Different JSP tags allow you to place code in different places in this servlet class. The JSP standard knows a few tags that perform special functions.

<%-...%> Tags of this variety generate code that is inserted at the place /* 1 */ in the samplecode. However, the syntax is set. An example of importing packages is <%'page import="java.sql.*,com.*" %>.

<%!...%> The contents of these tags are inserted into the class body, on an equal footing with e.g. object variables inplace /* 2 */. Here you can define methods, for example.

<%...%> The contents of these tags are inserted into the method that handles an HTTP query—in place /* 3 */, but without out.print(). The content is Java statements made by semicolon ("; ") must be terminated. For example, <% int i = 1;%>.

<%=...%> The contents of these tags are written to the output stream at the appropriate point—for objects using the implicit call of the toString()method. In the example, this is at out.print("Hello, World."); . Since only one expression can be printed here, the content of these tags does *not* end with semicolon. For example, <%=i%>.

<%--...--%> This tag encloses a comment that is not written to the (HTML) output.

20.2.2 Servlets

By default, browsers and servers on the web communicate through the HTTP protocol. The browser sends commands to the server, e.g. to request or save documents. For the internship, we focus on the HTTP operations **GET** and **POST**. GET is used by the browser to request a document. POST to send data to the server.

A servlet is called from a predefined URL. The corresponding class in the server provides methods to respond to HTTP operations from the browser. The minimum requirement for a servlet class is that it provides implementations for both GET and POST.

The following code demonstrates the implementation of a servlet that outputs the text "Welcome!". It can be called via the path **/HelloServlet** (e.g. http://localhost:8080/ServletTest/**HelloServlet**).

```
import java . io . IOException ; import javax . servlet .
ServletException ; import javax . servlet . annotation . WebServlet ;
import javax . servlet . http . HttpServlet ; import javax . servlet .
http . HttpServletRequest ;  import javax . servlet . http .
HttpServletResponse ;

@WebServlet(”/HelloServlet”)
public class HelloServlet extends HttpServlet { private static final long

  serialVersionUID = 1L;

  public HelloServlet () { super();
  }

  protected void doGet(HttpServletRequest request , HttpServletResponse response) throws
                                ServletException , IOException {
        Response. getWriter ().append("Welcome!" );
```

Listing 4: HelloServlet.java

```
  }


  protected void doPost(HttpServletRequest request , HttpServletResponse response) throws
                                ServletException , IOException {
        doGet(request , response ); }
}
```

20.3 Datenbankzugriff mit Java: JDBC

20.3.1 Einführung

To access the DB2 database, we use JDBC (*Java DataBase Connectivity*). JDBC is a unified database interface for Java that makes it transparent to the application developer whether to access a DB2, Oracle, MySQL, or otherwise database behind thatinterface. The connection is made by a driver, which is only integrated into the program at run time.

On JDBC, you access the database via a URL, the structure of which looks like the following: jdbc:subprotocol:[//host:port/]database. Host and port are therefore optional. The exact format for connecting to the DB2 instances used in the internship is presented in section 20.4.

The actual connection is built via the drivermanager class, specified by the Interface Connection. A program can also keep connections to multiple databases open, whereby each connection is realized by an object that implements the interface connection. This provides, among other things, the following methods:

- createStatement(): creates objects that implement the interface statement
- createPreparedStatement (String): creates objects that implement the PreparedStatement interface
- commit(): writes transactions through
- rollback(): withdraws transactions
- setAutoCommit(boolean): sets the autoCommit property for transactions to true or false
- close(): closes an open connection

SQL statements are used through the Statement and PreparedStatement interfaces. The authoritiesdefine a number of methods that allow statements to be executed on the database:

- executeQuery(string): executes an SQL statement (specified as a paramater) and returns a result
- executeQuery(): executes an SQL statement and returns a result
- executeUpdate (String): executes INSERT, UPDATE, DELETE, where the return is the number of affected tuples or nothing

The ResultSet interface defines the functionalities for further processing of request results. On objects that implement this interface, the following methods are available, among others:

- next(): goes to the ersten or next tuple of the result
- getMetaData(): returns an object that implements the ResultSetMetaData interface, which can be used to read information about the structure of the result
- findColumn (String): finds the position in the result tuple to an attribute name
- get<Type>(int) or get<Type>(String): read the (named) column of the result tuple and return an object of the specified type

The different types of data known in SQLare mapped to Java classes:

SQL-Typ	**Java-Typ**
CHAR, VARCHAR, LONG VARCHAR	String
DECIMAL, NUMERIC	java.math.BigDecimal
SMALLINT	short
INTEGER	int
BIGINT	long
REAL	float
DOUBLE, FLOAT	double
DATE	java.sql.Date
TIME	java.sql.Time
TIMESTAMP	java.sql.TimeStamp

20.3.2 Parameter in SQL-Anweisungen

If you want to use parameters in an SQL statement, it is important to use a PreparedStatement instead of a "traditional" statement. For example, the SQL request SELECT name FROM user WHERE login = 'meier' can be used to determine the name of the user meier. For example, the login value could come from a user who has entered their login into an HTML form. This value should notbe used for a SQL request, because the user's input itself could contain SQL injection. One mistake that is often made is that the user's inputis used to generate an SQL statement without additional checks(for example, by simply concatenation strings). Instead, PrepareStatements is a goodway to automatically validate and escape parameters. The following is an exampleof the correct handling ofparameters:

```
import java . sql . Connection ;
```

Listing 5: Incorrect handling of parameters in SQL statements

1

```
import java . sql . DriverManager ; import
java . sql . ResultSet ; import java . sql .
SQLException; import java . sql . Statement
; public class Sample {

        public static void main(String []        args) {
        . . .
        try {
            . . .
            Statement st = db2Conn. createStatement ();
            String myQuery = ”SELECT□name□FROM□user□” +
                ”WHERE□login□=□ ”” + loginFromForm + ” ””; ResultSet
            resultSet = st . executeQuery(myQuery);
            . . .
        }
        catch (SQLException e) {
            And. printStackTrace ();
        } finally {
            Close resources
            . . .
        }
    }
}
```

Listing 6: Correct handling of parameters in SQL statements

```
import java . sql . Connection ; import java
. sql . DriverManager ; import java . sql .
ResultSet ; import java . sql .
SQLException; import java . sql . Statement
; public class Sample {

        public static void main(String []  args) {
        . . .
        try {
            . . .
            String myQuery = ”SELECT□name□FROM□user□WHERE□login□=□?”;
            PreparedStatement st = db2Conn. prepareStatement(myQuery); st . setString (1 ,
            loginFromForm);
            ResultSet resultSet = st . executeQuery ();
            . . .
        }
        catch (SQLException e) {
            And. printStackTrace ();
        } finally {
            Close resources
            . . .
        }
    }
}
```

20.3.3 Schließen von DB-Resourcen

Connections to the database must be explicitly closed with JDBC when they are no longer in use. This is typically done in a finallyblock that joins a tryblockthat encloses the database code. For example, becauseDB connections must be closed frequently, it is recommended that you offload the corresponding code to a helper method. The following example shows how to close DB resources correctly. If resources are not closed properly orr, resources are closed in the wrong place, this can lead to errors (for example, disconnection due to too many open connections to the database).

```
import java . sql . Connection ; import java
. sql . DriverManager ; import java . sql .
ResultSet ; import java . sql .
SQLException; import java . sql . Statement
; public class Sample {

    public static void main(String [] args) { Connection db2Conn = null ;
        try { b2Conn = DriverManager . getConnection ( . . . ) ;
            . . .
            String myQuery = ”SELECT□name□FROM□user□WHERE□login□=□?”;
            PreparedStatement st = db2Conn. prepareStatement(myQuery); st . setString (1 ,
            loginFromForm);
            ResultSet resultSet = st . executeQuery ();
            . . .
        }
        catch (SQLException e) {
            And. printStackTrace ();
        } finally {
            Close Resources if (db2Conn !=
            null)
                db2Conn. close ();
                } catch (IOException e) {
                    And. printStackTrace ();
                }
            }
        }
    }
}
```

Listing 7: Correct closing of DB resources

20.3.4 Transaktionen

The correct use of transactions must also be ensured. By default, for connections to the database, the autoCommit property is set to true. This means that after each SQL statement, the transaction commits. Now, however, it may be necessary to take control of the transactions manually, because you want multiple SQL statements to run in the same transaction. An example of this is inserting data into two tables. If the datain the tablefails, inconsistent data would be in the database because the data for the first table is already in the database (the transaction performs an automatic commit after the first SQL statement). To address this issue,it is important toensure that both SQL statements run in the same transaction that does not commit until both statements are successfully executed. In the event of an error, the transaction should rollback. The followingtwo examples show the incorrect or correct use of transactions with JDBC.

Listing 8: Incorrect use of transactions

```
import java . sql . Connection ; import java
. sql . DriverManager ; import java . sql .
ResultSet ; import java . sql .
SQLException; import java . sql . Statement
; public class Sample {

        public static void main(String []  args) {
        User user = . . . ;
        Connection db2Conn = null ;
        try { db2Conn = DriverManager . getConnection ( . . .  ) ;
            . . .
            final String sql1 = ”INSERT□INTO□user□VALUES□(? ,? ,? ,?)”;
            PreparedStatement st = db2Conn. prepareStatement( sql1 ); st . setString (1 , user
            .getName()); st . setInt (2 , . . . ) ;
            . . .
            st . execute ();
            . . .
            final String sql2 = ”INSERT□INTO□user_roles□VALUES□(? ,?)”;
            PreparedStatement st2 = db2Conn. prepareStatement( sql2 ); st2 . setString (1 , . . . ) ;
            . . .
            st2 . execute ();
            . . .
        }
        catch (SQLException e) {
            And. printStackTrace ();
        } finally {
            Close Resources if (db2Conn !=
            null)
                db2Conn. close ();
                } catch (IOException e) {
                    And. printStackTrace ();
                }
            }
        }
    }
}
```

```
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
```

```
import    java . sql . Connection ;

import    java . sql . DriverManager ;

import    java . sql . ResultSet ;

import    java . sql . SQLException;

import    java . sql . Statement ;

public class Sample { public static void
      main(String []
                                              args) {
```

Listing 9: Correct use of transactions

```
1
2
3
4
5
6
7
8
9
```

```
            User user = . . . ;
            Connection db2Conn = null ;
            try { db2Conn = DriverManager . getConnection ( . . . ) ;
                  db2Conn.setAutoCommit(false );
                  . . .
                  final String sql2 = ”INSERT□INTO□user□VALUES□(? ,? ,? ,?)”;
                  PreparedStatement st = db2Conn. prepareStatement( sql1 ); st . setString (1 , user
                  .getName());
                  . . .
                  st . execute ();
                  . . .
                  final String sql2 = ”INSERT□INTO□user_roles□VALUES□(? ,?)”;
                  PreparedStatement st2 = db2Conn. prepareStatement( sql2 ); st2 . setString (1 , . . . ) ;
                  . . .
                  st2 . execute ();
                  . . .
                  Transaction commits
                  db2.commit();
            }
            catch (SQLException e) {
                  Rollback of the transaction in case of error db2Conn. rollback ();
                  And. printStackTrace ();
            } finally {
                  Close Resources if (db2Conn !=
                  null)
                        db2Conn. close ();
                        } catch (IOException e) {
                              And. printStackTrace ();
                        }
                  }
            }


      }
}
```

20.4 Verbindung mit DB2

Example code for establishing a connection to the local DB2 with query of data is given in the following listing. For Java to find the DB2 driver, db2jcc.jar must exist in the web app directory under WEB-INF/lib. You can find the file in thedirectory "/sqllib/java/ " for which it stands for the user's home directory.

In the untigen code we establish a database connection to the database mondial and read from the database the capital of Germany.

```
<%@ page import=”java . sql .∗” %> <%! private String
getCapital(String country) { String out=””; try {
        Class .forName(”com.ibm.db2. jcc . DB2Driver” );
```

Listing 10: connection.java

```
        Connection connection =
            DriverManager . getConnection(”jdbc :db2: mondial” );

        String stStr = ”SELECT capital ” +
            ”FROM dbmaster. country WHERE name=?”;
        PreparedStatement stmt = connection . prepareStatement( stStr ); stmt. setString (1 ,
        country );
        ResultSet rs = stmt . executeQuery ();

        StringBuffer outb = new StringBuffer ();  while ( rs . next ()) {
            String name = rs . getString (”capital” ); outb
            .append(name).append(”<br/>” );
        }
        out = outb . toString ();

    } catch (Exception e) {
        And. printStackTrace ();
        return ”#fail ”;
    }
    return out ;
}
%>
<h1>Capital</h1>
<%= getCapital(”Germany”) %>
```

Please make sure that there are no quotation marks around the placeholders in the PreparedStatements. So not WHERE name='?' , but WHERE name=? . Assemble SQL statements using string operations is not a good idea.

20.5 HTML-Formulare

HTML form elements can be used to record user input and pass it to a servlet or JSP. When the form is submitted, the test servlet is called and the contents of the form elements are passed via POST or GET. More about HTML and HTMLForms can be found e.g. at

- http://www.w3.org/TR/html4/ or
- http://de.selfhtml.org/.

The following example of an HTML form that calls a servlet shows various form elements that can be used:

```
<form enctype=”multipart/form−data” action=”test” method=”post”>
  <table>
    <tr><td colspan=”2”>Beispiel</td></tr>
    <tr>
      <td><b>Matrikelnummer:</b></td>
      <td><input type=”text” name=”matrikel” size=”60”/></td> </tr>
    <tr>
      <td><b>Aufgabe 1:</b></td>
            <td><textarea name=”aufgabe_1” rows=”15” cols=”60”></textarea>
```

Listing 11: form.html

```
1
2
3
4
5
6
7
8
9
10
```

```
        <input type=”reset”/>
      </td>
    </tr>
  </table>
  <input type=”hidden” name=”woche” value=”1”/>
  <input type="submit"  value="send"/> </form>
```

```
11
12
13
14
15
16
17
```

- input of type text allows the input of a short text
- hidden input passes additional values without displaying them to the user
- input of type resetreset a form

- input of type submit sends the form
- textarea allows the input of longer, multi-line texts

Any input to be read by a Javaweb application requires a name. The HTML element form then specifies the script to call and the method (POST or GET).

20.6 Reading POST/GET parameters in Java and JSP

In Java servlets (and thus also inJSP-Sei ten), some objects are implicitly predefined that provide access to relevant data. The detailed description of the following objects can be found in the JavaEE API documentation.

20.6.1 request

In the requestobject of the class HttpServletRequest there is a paar interesting methods for accessing data that is directly related to the current request –that's basically all that the web browser transferred the last time you clicked on a link or similar. In particular, the parameters canbe read out with the Methode getParameter (String name):

```
<%
    String country = request . getParameter ("country" ); if  (country ==
    null) { out .println ("" );
    }
    else { out . println (”Country:□”+country+”<br/>” ); }
%>
```

Listing 12: parameter.jsp

1
2
3
4
5
6
7
8
9

There are also methods to access components of the query, read cookies, etc.

20.6.2 session

HTTP is a stateless protocol: a client sends a request, the server sends a response, and the entire context ends. Thenext request for the HTTP-level server is not connected to the previous one. If you want to do this, e.g. because a user should log in only once and should still be recognized correctly with every subsequent page view (incl. shopping cart, etc.), you have to work with sessions. SessionHandling works automatically with Java servlets. To support it, there is the sessionobjectof type HttpSession. The most

prominent method of this class is getId(), which wouldrun the ID of the session as a string. The session ID uniquely identifies a running session.

A Tipps zum Debugging

Sometimes programs don't work right at the first start and you have to debug. When debugging web applications, there are a few unpleasant effects, some of which are caused by caching. Let's say your code isn't working and you've inserted debug outputs. But for some reason, the debug output is not output in the browser. This can be due to caching, either on the server side or on the browser side. To be on the safe line, you should clear the browser cache and exit and restart your server (Apache or Tomcat).

If you want to make sure that you are working with a certain version of your web app (e.g. with the one in which you made a change for testing) and not with pages cached, you can temporarily insert a counter in asuitable place (e.g. in an h1 tag) which you count up on changes. If the modified code is taken from the server and displayed in the browser, you will see from the counter that the changehas takeneffect. For example, in the first attempt, your code looks like this:

```
<h1>Capital</h1>
<%= getCapital("Germany") %>
```

After changing the country name to "USA", for example, "Berlin" still comes back as an issue. To determine that the code hasbeen translated and adopted, change the heading:

```
<h1>Capital 1</h1>
<%= getCapital("Germany") %>
```

Now the headline should be "Capital 1" in the browser. If not, you have a problem with a cache.

B Overview of the commands used

?	Help
? *command*	Help for a command
start dbm	Start database manager
stop dbm	Stop database manager
get dbm configuration	View database manager configuration
catalog tcpip node *name*	Catalog remote database servers as local node *name*
list node directory	View the directory of all cataloged nodes
uncatalog node *name*	Remove cataloged node *name* from the directory
catalog database *name* as *alias*	Database *name* under local alias *alias* catalog
list database directory	View the directory ofall cataloged databases
uncatalog database *alias*	Remove as *alias* cataloged database from the directory
create databse *name*	Create a new database *name*
drop database *name*	Delete database *name*
connect to *name*	Connect to database *name*
terminate	Stop connecting to database
set schema *name*	Set the current schema to *name*
get authorizations	View the privileges on the current database
grant *privilege* on database to public/group *name*/user *name*	toall, a group, or a user on the current database. Giving privilege
revoke *privilege* on database to public/group *name*/user *name*	totake back a privilege on the current database
create schema *name*	create a new schema (a namespace for database objects)
drop schema *name* restrict	delete an existing schema
create table *name*	create a new table
drop table *name*	delete an existing table
alter table *name*	change the definition of an existing table
create sequence *name* as integer	create a sequence with auto-Inkrement
alter sequence *name* restart	resets a sequence back to the start value
create index *name* on *table* …	create a new index on a table
drop index *name* …	delete the trigger *name*
describe indexes for table *name*	show the indexes on the *name* table
list tables for schema *name*	View all tables for the *schema name*

describe table *name*	Show the structure and attributes of the Show Name table65
export to *file* of *type*	export data to a file
import from *file* of *type*	import data from a file
load from *file* of *type*	import data with definition of the import method
delete from *name* where *condition*	delete tuples from a table to the *condition* applies
update *name* set *assignment* where *condition*	change the contents of tuples to the *Condition* applies
insert into *name* values	directly insert new values into an existing table
insert into *name statement*	insert the result of a *statement* as new values into an existing table