Geographic Information Technology Training Alliance (GITTA) presents:

Structured Query Language SQL

Responsible persons: Anca Dobre, Dominique Schneuwly, Samuel Wiesmann, Susanne Bleisch

Table Of Content

1. Structured Query Language SQL

SQL (Structured Query Language) is a query language for relational databases. The roots of SQL go back to SQUARE, a more mathematical oriented language and SEQUEL, a predecessor of SQL from the seventies. There exist different standards (ISO and ANSI) of SQL, but the most common one is SQL-92. The latest standard SQL-99 (or SQL3) even includes XML. In the following Units we will give a simple overview of the most important SQL functions.

This lesson is geared to the SQL-99 standard (GULUTZAN et al. 1999). However, this standard is not implemented by all database systems constantly. For example, some commands are named differently or their syntax is constructed differently. For this reason, SQL commands from this lesson might not work in certain systems. If you encounter problems, please check in the respective user manuals whether or not the command is supported at all, whether it is named, or constructed differently. Most database systems also go beyond the standard. However, caution is advised with these SQL extensions since they can vary considerably from system to system.

Learning Objectives

- You understand the basic SQL concepts and are able to explain the use of SQL in the areas of data definition, data manipulation, and data control.
- You are able to use SQL to create, modify, and delete tables.
- You can express simple and complex queries with the help of SQL and know how SQL queries are used in nesting or in combinations. You are also able to correctly apply arithmetic operators and set operators within SQL queries.
- You master SQL to add, modify, and delete data tuples.

1.1. SQL overview

SQL (Structured Query Language) is one of the main reasons for the commercial success of relational databases. The ANSI (American National Standards Institute) and the ISO (International Standards Organization) developed in 1986 the first SQL-version with the name SQL-86 or SQL1. In 1992 a second and more extended standard with the name of SQL-92 or SQL2 was established. The latest standard includes XML, dates from 1999 and is therefore called SQL-99 or SQL3. With the use of SQL in most commercial database systems the migration from one system to another has become easier for the user. In the ideal case, the user need not consider which system is used because query formation in SQL remains the same.

1.1.1. SQL Concepts

SQL is a descriptive, entity-oriented query language for data manipulation with its roots in relational algebra. Today SQL is used either as a stand-alone programming language or within other languages like C, C++, Java, ADA, COBOL, FORTRAN, PL/1, PASCAL etc.
SQL actually consists of three sub languages:

- DDL - Data Definition Language: Used for creating databases and tables and for maintaining the structure.
- DML - Data Manipulation Language: Used for accessing and extracting data from a database (add, update, delete etc.).
- DCL - Data Control Language: Used to control access to the database and therefore essential for the security system.

In most implementations of SQL functions from other programming languages (if-clauses, iterations etc.) have been added. Some SQL-versions, such as Oracle's PL/SQL, can therefore be seen as independent programming languages.

1.1.2. Data Definition (DDL)

In SQL the terms table, row and column are synonyms for relation, tuple and attribute. To create a new relation scheme in the database we start with the `create table`-command. Together with the creation we must provide the relations attributes and their domains (e.g. number, char or date). Additional we can define other constraints, checks and keys etc. The keyword `NOT NULL` tells the system that this attribute cannot be empty. Primary keys are declared using a special `"table constraint"-clause`.

```
create table Subscription (
   Name char (30) not null,
   CustID number (5) not null,
   SubType char (15) default 'monthly' check (SubType in ('monthly', 'weekly')),
   SubStart date,
       constraint pk_sub primary key  (Name, CustID),
       constraint fk_name foreign key (Name) references Magazine (Name),
       constraint fk_custid foreign key  (CustID) references Customer (CustID));
```

Code-Beispiel: Create Table

This example shows how a new table is created in SQL. In the intermediate lesson we take a closer look at all the SQL statements. For the moment you don't have to understand these statements in detail.

1.1.3. Data Manipulation (DML)

There are two sorts of data manipulation commands in SQL. The first type are only used for database queries and do not alter the tables. The second type are used for adding, updating or deleting values in a database. We will take a closer look at the different data manipulation commands in the following units.

- **Basic database queries**
- **SQL Insert, Delete and Update**

1.1.4. Data control (DCL)

Data control commands in SQL control access privileges and security issues of a database system or parts of it. These commands are closely related to the DBMS (Database Management System) and can therefore vary in different SQL implementations.
Some typical commands are:

- `GRANT` - give user access privileges to a database
- `DENY` - deny user access
- `REVOKE` withdraws access privileges given with the GRANT or taken with the DENY command

Since these commands depend on the actual database management system (DBMS), we will not cover DCL in this module.

1.2. Creation and modification of tables

In this unit, we will show you how to create, modify, and delete tables using SQL commands.

1.2.1. Create tables

With `CREATE TABLE`a new table can be created in a database. The command has the following basic structure:
`CREATE TABLE <table name> (<Attribute definitions and constraints>);`
The table name must be unique within the current database or the current scheme.

Attribute definition
Attributes are defined by a name and a datatype, whereas the name must be unique within the table. These specifications are compulsory for all attributes.
The order of the attributes at definition corresponds to the order of the columns in the table created. If a certain order is aspired, you need to define it at the creation of the table. Unless there are no changes made to the table (see **Change table structure**, the order stays this way.

Constraints
There are two types of constraints: table constraints and attribute constraints. The difference is that attribute restrictions apply to only one attribute whereas table constraints may apply to more than one attribute but this need not be. With these restrictions, the range of values of the attributes can be restricted or it is prevented that the entered values are not allowed. A record cannot be recorded if it violates a restriction.
There are four kinds of constraints:

- `UNIQUE` - the attribute or the attribute combination need to be unique within the table
- `PRIMARY KEY` - the attribute or the attribute combination is the primary key of the table
- `FOREIGN KEY` - the attribute is a foreign key
- `CHECK` - Condition that must be fulfilled for an attribute or an attribute combination

The constraints can be named. However, this is not necessary.

Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed. [link]

In this example, a table is added to a database. That table is linked to an already existing table. The difference between an attribute and a table constraint can be seen in the SQL command. projekt_ID and leiter_ID have an attribute constraint (the constraint is written directly behind the attribute definition). projekt_ID has the constraint PRIMARY KEY - it is therefore the primary key of this table, i.e. the attribute must be unique and must not be NULL. leiter_ID has the constraint NOT NULL (special case of a CHEK constraint), meaning it needs to hold a value at all times.
The link to the existing table is defined as a table constraint (FOREIGN KEY) and is named (projektleiter). This constraint could also be defined as an attribute constraint since it only includes one attribute.
The example shows that there are basically no difference between attribute and table constraints as long as only one attribute is affected. It is about two different ways of collecting constraints.

1.2.2. Changing the table structure

With `ALTER TABLE` the structure of a table can be modified. The attributes and constraints that were created with CREATE TABLE can be modified, new ones can be added, and existing ones can be deleted. The command has the following syntax:

```
ALTER TABLE <table name> <Change> ;
```

whereas `<Change>` can include various commands:

- `ADD [COLUMN] <Attribute defintion>`
  Add an attribute (Attribute definiton as in CREATE)
- `ALTER [COLUMN] <Attribute name> SET DEFAULT <Standard value>`
  define a new standard value
- `ALTER [COLUMN] <Attribute name> DROP DEFAULT`
  delete current standard value
- `DROP [COLUMN] <Attribute name> {RESTRICT | CASCADE}`
  delete an attribute
- `ADD <Table constraint>`
  add new table constraint (table constraint as in CREATE)
- `DROP CONSTRAINT <Table constraint>`
  delet a table constraint

With the above commands, attributes and constraints can be added or deleted respectively. In addition, standard values for the attributes can be set or deleted. There are other SQL commands that are not listed here.
Default SQL does not include any commands for modification or renaming of attributes. This would lead to problems when data already exists. However, in some databases these commands are included (e.g. MODIFY or RENAME). The syntax is different in every system though. If there are no data, the attribute to be changed can be delete and reattached.

Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed. [link]

In this example, an attribute is added to a table. The dataset shown contains NULL for this attribute because there was no value assigned yet. Afterwards, this attribute is delted from the table again. The keyword `RESTRICT` provokes that only attributes that are not linked to other tables can be deleted (foreign key). Alternatively, the keyword `CASCADE` can be used. Using this, not only the designated column but also the linked column in the other table is delted.

1.2.3. Deleting tables

With DROP TABLE an existing table can be deleted. The data and structure of the table are deleted.
The command has the following syntax:

```
DROP TABLE <Table name>;
```

Attention: With `DROP TABLE` the table structure is deleted with all the data. In most cases, this cannot be undone!

1.3. Basic database queries

In this Unit we will take a closer look at how to do database queries using SQL.

1.3.1. SELECT-FROM-WHERE clause

In languages like SQL data from a certain domain (FROM) that match some conditions (WHERE) are selected and presented (SELECT). The result can be seen as a new relation.
The syntax of a basic SQL query is:

```
SELECT <select-list>
FROM <from-list>
WHERE <condition>
```

In this syntax the...

- <select-list> contains the names of the attributes (columns) values to be returned.
- <from-list> is the list of the relations (tables) used for the query.
- <condition> is an expression that identifies the tuples that we are looking for.

Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed. [link]

We saw the basic statements that are needed for making queries with SQL. Of course there are extensions which allow more specific or more flexible queries.
These extensions include:

- multiple conditions (in boolean AND/OR combination)
- complex conditions (subqueries, joins etc.)
- pattern matching and arithmetical operators
- non-relational functions (sort, group, aggregate)
- set operators (union, intersect, minus)

1.3.2. Multiple conditions

In SQL it is possible to have multiple conditions and combine them with boolean operators (AND, OR). For negating a condition the NOT operator is used. The data is selected if the whole condition returns as TRUE.

Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed. [link]

Other examples of multiple conditions:

- `PLZ = 8000 AND NOT Last name = 'Schmidt'`
- `PLZ = 8000 OR PLZ = 8006`

- `(Last Name = 'Müller' OR Last name = 'Meier') AND Place = 'Zürich'`

> In SQL queries (e.g. conditions) text must be set in single quotation marks. Numerical values are written without single quotation marks. This can be seen in the examples.

Instead of a long chaining with AND there is also the possibility to compare several attributes simultaneously with each other:
`First name = 'Ursula' AND Last name = 'Müller'` newLine space="long"/> can be written as:
`(First name, Last name) = ('Ursula', 'Müller')`

1.3.3. Comparison operators

SQL supports different comparison operators. They can be used to compare attributes with constants or with other attributes. DIf an operator is used to compare an attribute with a constant, we talk about restriction. If the operator is used to compare two attributes, then we talk about a join. With joins, data from different relations can be compared and combined. Of course only attributes with the same domain (value range) can be compared. The following comparison operators for numerical attributes are supported:

- `=` equal
- `>` greater
- `<` less
- `>=` greater than or equal
- `<=` less than or equal
- `<>` not equal
- `BETWEEN` between

Conditions usually have the syntax `<atribute> <operator> <value>`. The operator `BETWEEN` has a somehow different syntax (see the example).
Other comparison operators are discussed in the next paragraphs.

Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed. [link]

> If in the SELECT-clause a star (*) instead of an attribute list is used, then all attributes of the relations specified in the FROM part of the query are displayed.

1.3.4. Pattern matching and arithmetical operators

Pattern matching using LIKE
The LIKE condition allows you to use wildcards in the WHERE clause of an SQL statement. This allows pattern matching.
The patterns that you can choose from are:

- "%" allows you to match any string of any length (including zero length)
- "_" allows you to match on a single character

Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed. [link]

`NOT LIKE` examines whether the given section is not present in the string.

Arithmetical operators

The arithmetical standard operators for addition (+), subtraction (-), multiplication (*) and division (/) can all be used for numerical constant or for attributes with a numerical domain. This allows attributes in a request to be calculated together.

Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed. [link]

In the above example the result is a relation with three attributes. The third attribute is named "Usury" since "Price/Size" is not a good name and stands for the price per unit. We define that if it lies over 15, it is overpriced ("Usury").

Arithmetical operators can be used in the SELECT part of a request or in the conditions.

In the SELECT or FROM part of a request, attributes and relations can be given a different name by the command `AS`: `<Attribute/Relation> AS <new name>` (The key word `AS` can be omitted). This can be used to give a calculated value a meaningful name (see example), or to make SQL requests easier to read.

1.3.5. Nested queries

Nested queries

Conditions are usually made of an attribute, a value and an operator that forms the condition (eg. Name="John"). But the values themselves don't have to be constants, they can be the result of another sub-query. We then talk about nested queries. Using the IN-operator nested queries can be as deep as necessary.

There are three types of sub-queries that differ in their result:

- Sub-queries that return a value (one column and one row)
- Sub-queries that return a row
- Sub-queries that return several rows

If only one value or row is returned the normal comparison operators can be used.

If more than one row is returned special operators are used:

- `IN` examines whether the value exists in the sub-query
- `<Comparison operator> ALL` the condition must return TRUE for all rows in the sub-query
- `<Comparison operator> ANY (SOME)` the condition must return TRUE for at least one row in the sub-query

Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed. [link]

Sub-queries can also be used in the FROM part of a query. Thus, a relation can be compiled specifically for a request. The sub-query must be assigned a name with AS: `(<Sub-query>) AS <Name>`

1.3.6. Join

It happens often that data from multiple realtions are required in a query. For this, relations need to be linked. The identical attributes (foreign keys) in both relations are linked.
There are two ways to link relations in queries:

in the WHERE part
In this option, the identical attributes are linked with the comparison operator and inserted as "condition" (not a real condition) into the WHERE part of the query.

Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed. [link]

Basically, e.g. a < can also be used for a join. However, this is usually used in combination with a join condition with = since such a join alone does not make sense.

in the FROM part
Another possibility is to do the join in the FROM part of a query. This makes more sense because the actual search criteria (conditions in the WHERE part) are separated from the table joins.
The following commands are available for this purpose:

- `<Relation> JOIN <Relation> USING (<Attribute>)`
  the relations are joined by an attribute with the same name in both tables
- `<Relation> NATURAL JOIN <Relation>`
  automatically joins all attributes that are of the same name in both relations.
- `<Relation> JOIN <Relation> ON <Attribute> <comparison operator> <Attribute>`

with this command, you can decide by which attributes the relations should be joined and which operator is to be used (there can be more than one join as well).

Using these commands, the example from above would look like the following:

```
SELECT name, surname, newspaper_name
FROM customer JOIN subscription USING (CustNo);
```

or

```
SELECT name, surname, newspaper_name
FROM customer JOIN subscription ON customer.CustNo =
 subscription.CustNo;
```

or

```
SELECT name, surname, newspaper_name
FROM customer NATURAL JOIN subscription;
```

The commands above only return datasets present in both relations. Should **all** datasets of a relation be returned with the corresponding datasets of the second relation, the following commands are applied:

- `<Table> RIGHT OUTER JOIN <Table> USING (<Attribute>)`
  all datasets of the right relation and the corresponding datasets of the left relation
- `<Table> LEFT OUTER JOIN <Table> USING (<Attribute>)`
  all datasets of the left relation and the corresponding datasets of the right relation

If there are no joins to be made in the left relation using `RIGHT OUTER JOIN`, NULL is returned for the attributes of this relation. The same is true for the opposite (using `LEFT OUTER JOIN`).

1.3.7. Non-relational constructs

ORDER BY clause

SQL contains some operators that have nothing to do with relational algebra. For example an entity per definition does not have an order. Nevertheless in SQL you can order your tables using the ORDER BY clause. Using the keywords ASC and DESC the sorting can either be ascending or descending. If no keyword is used, the sorting is in ascending order by default.

Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed. [link]

In this example all customers are sorted in ascending order according to their names and as a second sort parameter in descending order according to their surname. The ASC keyword is default and could therefore be omitted.

GROUP BY clause

Grouping methods are used to form a subset of tuples of a relation according to certain criteria. These subsets can then be used to calculate statistical parameters such as average, sum etc. The value of a certain attribute serves as grouping criteria. All tuples with the same value for this attribute are grouped. These groups can be used in further processes (a special case would be another grouping to be able to use groups of groups). For this, so-called group functions are used. They can only be applied to numerical attributes.

The group functions that SQL usually offers are the following:

- `min` returns the smallest value ignoring null values
- `max` returns the largest value ignoring null values
- `sum` returns the sum of all values ignoring null values
- `count` returns the number of rows
- `avg` returns average value ignoring null values
- `stdev` returns the standard deviation ignoring null values
- `varriance` returns the variance ignoring null values

Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed. [link]

In this query we want to find the customers that spent more than 250 SFR for all their small advertisings (a small add is an add that costs less than 300 SFR). In a first step restriction is applied: A004 is sorted out because it is not a small add (price is over 300 SFR). The remaining tuples are grouped by customer and those with a sum of over 250 SFR are selected (customer groups 002 adn 005 are sorted out because they have not reached the limit yet).
Requests including a GROUP BY clause are processed as follows: First, the condition in the WHERE part is processed (if applicable). Then, the specified columns are grouped. With the condition in the HAVING part of the request, another condition can, if necessary, be specified for the grouped attribute values. Due to this order, you can see that there cannot be a group function in the condition of the WHERE part of the request since there has been no grouping yet. However, all attributes appearing in the request must be either in the GROUP BY clause or in a group function.

1.3.8. Set operators

Set operators are used to connect different queries and produce a resulting relation. Of course these set operations are only allowed if the attributes match (eg. the domain or the number of attributes etc.). The following set operators are used to join together sets of tuples. They might already be known from the algebra of sets.

- `union` produces a union of different sets
- `intersect` produces an intersection of different sets.
- `minus` subtracts one set from another one.

By default, duplicates are not returned using a request with these set operators. If duplicates need to be returned, the keyword `ALL` must be put behind `UNION`, `INTERSECT` or `EXCEPT`.

Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed. [link]

This query is used to find all names of customers who have a magazine subscription and who have placed an advertising in this magazine.

1.3.9. Summary

In this Unit it was shown step by step how SQL statements are phrased. To show all the parts in context a complete SQL request is listed below.
Complete syntax of a SQL request:

```
SELECT [DISTINCT | ALL]
<Attribute> [AS <Name>] [, ...] | *
FROM <Relation> [, <Relation>]
[WHERE <Condition>]
[GROUP BY <Column> [HAVING <Condition>]
[ORDER BY <Column> [ASC | DESC], [, ...]];
```

A large part of this syntax is optional and only needs to be stated only in certain cases. Teh simplest syntactically correct request has only a SELECT- and a FROM-part. Only with more complex problems, all parts are used in the same request.

It is possible that certain datasets appear more than once in the result of a request. The keyword `DISTINCT` is used to delete such duplicates from the result. The keyword `ALL` is used to prevent the duplicates from deletion. This is the default and does not have to be stated explicitly.

1.3.10. Database queries

This self assessment should give you the possibility to test different SQL queries and get to know them more in depth. We use the following **sample database** (please leave this overview open during the exercise).
Find the correct solution tables for the following SQL queries. Put your solutions (either Word or PDF format) on the discussion board and check the other solutions.
You are encouraged to comment on other postings or to ask questions using the discussion board. Please do not email the tutor directly so that others can also benefit from your questions and the tutors answer.

SQL Requests

-
  ```
  SELECT vname, nname
  FROM angestellter
  WHERE salaer = 25000
  AND ahvnr > 5000
  ```

-
  ```
  SELECT pname
  FROM projekt
  WHERE abt = 5
  OR pnummer = 30
  ```

-
  ```
  SELECT nname
  FROM angestellter
  WHERE salaer BETWEEN 28000 AND 41000
  ```

-
  ```
  SELECT ahvnr
  FROM angestellter
  WHERE adresse LIKE 'Zu%'
  ```

-
  ```
  SELECT Angestellter.nname
  FROM Angestellter, Arbeitet_an
  WHERE Arbeitet_an.projekt = '20'
  AND Angestellter.ahvnr = Arbeitet_an.ang
  ```

-
  ```
  SELECT name
  FROM angehoeriger
  WHERE ang IN
  (SELECT ahvnr
  FROM angestellter
  WHERE adresse = 'Kuesnacht')
  ```

Solutions to the exercises

download solutions

This self assessment uses the same **sample database**as before but now using an interactive SQL tool called **WebSQL.** This tool was developed by the GITTA-partner IFI (Institute of informatics, University of Zurich). Please contact your tutor if you didn't get your account information or if the login doesn't work.

Some remarks about the WebSQL interface

- The semicolon at the end of a SQL query or SQL command is optional.
- On the tab 'Dataset' you find information about the content of the original database.
- You changes to the database are saved and are available at your next login.
- On the tab 'Settings' you can change your user information and reset the database to its original state. All your changes will be deleted. Note that this command is irreversible.
- On the tab 'Protocol' you find a list of all your SQL queries.
- Please contact your tutor and not the IFI for technical questions.

The goal of this self assessment is that you are able to form SQL queries that answer the questions below. Write a report where you explain your queries and comment on the solutions and put this document on the discussion board. If you have any questions please post them also on the discussion board. Of course you are welcome to check the other students solutions, comment on them or answer questions from other students.

Aufgaben

- Select the names of all employees and sort them in descending order (Z to A).
- Find the names of all employees who live in Dübendorf and earn more than 30'000 Fr.
- Select all children whose parents are employees and live in Zurich. Sort the result by birth date.
- Find the total amount of time that is spent on all projects in the research (Forschung) department. Use one query only!
- How many children has the manager of the administration (Verwaltung) department?
- Find projects of Zurich where the total amount of work is over 50 hours?
- Create your own SQL query and post the query (in word form) together with the solution on the discussion board (under the topic 'eLSQL').
- Solve some queries that other students have posted.

Possible solutions

download solutions

1.4. SQL Insert, Delete and Update

To keep a database accurate we have to be able to not only create and delete tables but to also modify the content. In SQL this is done with the commands INSERT, DELETE and UPDATE.

1.4.1. Inserting tuples

In its most basic form the INSERT INTO command adds a tuple to an existing table.
The syntax is:
`INSERT INTO` <Tablename> (<Attribute list>)
`VALUES` (<Valuelist>);
Please note that the attribute list can be omitted if a complete tuple is inserted.

Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed. [link]

With the INSERT command shown above, new tuples are inserted into the database. Often, the values to be inserted are the result of a request. For this situation the command can be changed as follows:
`INSERT INTO <Tablename> (<Attribute list>)`
`SELECT ...` (normal request)
VALUES is replaced by an SQL request. The result of this request is inserted into specified table. If the request returns a complete tuple in the correct order the attribute list does not need to be stated here either.

1.4.2. Deleting tuples

The DELETE FROM command removes one or more rows (tuples) of a table (relation). The WHERE-clause specifies which tuples have to be deleted. If it is missing, all tuples are deleted!
The syntax is:
`DELETE FROM` <Tablename>
`WHERE` <Condition>;

Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed. [link]

NOTE: If `DELETE FROM` is used without `WHERE`, all datasets of this table are deleted. This is irreversible in most systems.

1.4.3. Updating tuples

The UPDATE command is used to change one or more attribute values of existing tuples. The WHERE-clause specifies which tuples should be updated. The SET-clause specifies which attributes should be changed and their new values.

The syntax is:

```
UPDATE <Tablename>
SET Attributevalues
WHERE <Condition>;
```

Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed. [link]

The assignment of the new value happens as follows:

```
<Attributname> = <Value>
```

Multiple assingments are separated by commas. The assigned value can either be a constant (absolute term) or the result of a request, i.e. there can be a request to the right of the equals sign.
If a whole row is updated, the assignment looks as follows:

```
ROW = ROW(<list of values>)
```

1.4.4. eLSQL Exercise 'Insert, Delete und Update'

This exercise predominantly addresses the SQL commands INSERT, DELETE, and UPDATE. To complete the exercise however, the SQL requests from unit "Basic database queries" need to be known as well. During the following exercise you will work with this **sample database**. (please leave this overview open during the exercise). You can also find the overview in the eLSQL Tool on the tab 'Dataset'. The exercise is executed through a web interface on a MySQL database. Your tutor will provide you the information needed (webaddress and registration).

Some remarks about eLSQL interface

- The semicolon at the end of a SQL query or SQL command is optional.
- On the tab 'Dataset' you find information about the content of the original database.
- You changes to the database are saved and are available at your next login.
- On the tab 'Settings' you can change your user information and reset the database to its original state. All your changes will be deleted. Note that this command is irreversible.
- On the tab 'Protocol' you find a list of all your SQL queries.
- Please contact your tutor and not the IFI for technical questions.

Exercises

- You have been hired by a company (represented in the **sample database**) and have to add yourself in the table employee. Add at least someone of your family in the according table. Also you should add some hours of work to one of the projects.
  Remark: You don't have to add your birth date since the date format may be a bit complicated!
- Select the columns you added in the different tables.
- Project 20 has been finished. Please delete all entries related to project 20.
  How did you proceed and why?
- Beate Tell leaves the company. Her position is taken over by Sonja Maradona. Please make sure that these changes are made correctly in your company database. Delete all entries that are no longer needed.

How did you proceed and why?

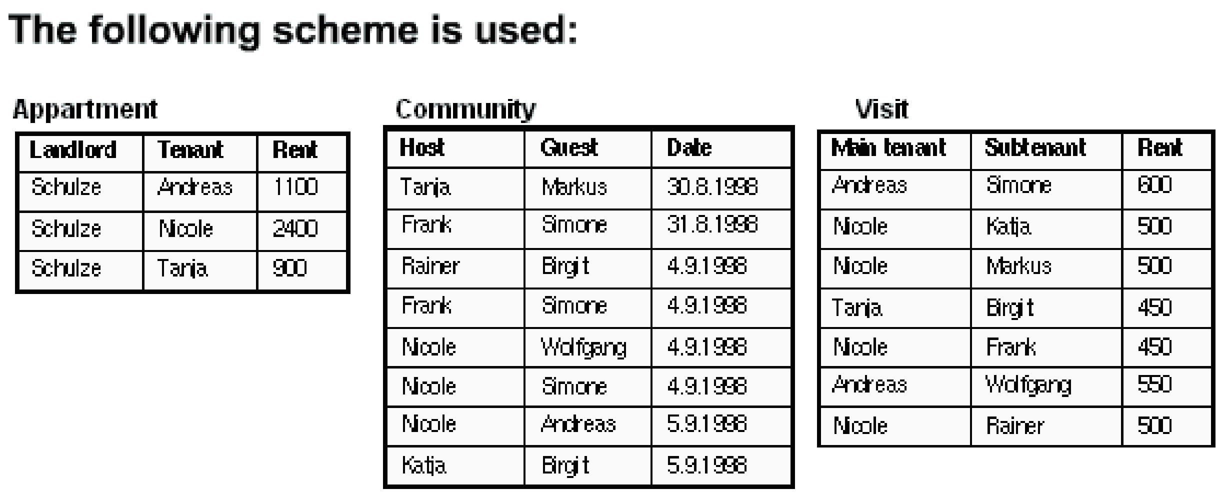
- All employees who's boss is Boris Frisch get a wage raise of 10'000 Fr.
- Present some other typical database changes (and their solutions) that might have to be done in this company.

Possible solutions

download solutions

1.5. Usage of SQL

This flash-animation should allow you to exercise all kinds of database queries.
To start the animation, klick on the graphic



The following scheme is used:

Appartment

Landlord	Tenant	Rent
Schulze	Andreas	1100
Schulze	Nicole	2400
Schulze	Tanja	900

Community

Host	Guest	Date
Tanja	Markus	30.8.1998
Frank	Simone	31.8.1998
Rainer	Birgit	4.9.1998
Frank	Simone	4.9.1998
Nicole	Wolfgang	4.9.1998
Nicole	Simone	4.9.1998
Nicole	Andreas	5.9.1998
Katja	Birgit	5.9.1998

Visit

Main tenant	Subtenant	Rent
Andreas	Simone	600
Nicole	Katja	500
Nicole	Markus	500
Tanja	Birgit	450
Nicole	Frank	450
Andreas	Wolfgang	550
Nicole	Rainer	500

Flash-Animation

The data of this exercise can be downloaded as **Access-Datenbase**. All requests from the exercise can be tested to find the difference.

1.6. Summary

SQL stands for "Structured Query Language" and is a language to communicate with relational and object oriented databases. With SQL new tables (relations, schemes) can be created, altered, and deleted using the commands `CREATE TABLE`, `ALTERTABLE`and `DROP TABLE`. This part of SQL is also known as the Data Definition Language (DDL). More important in the daily use of SQL are the data query and manipulation (DML) commands. These commands allow you to `INSERT, DELETE, and UPDATE` values in the database. SQL is also used to control access restrictions to the database or to parts of it.

1.7. Recommended Reading

- **ELMASRI, R.; NAVATHE, S.B.**, 1994. *Fundamentals of Database Systems*. 2nd. Redwood City, California: Addison-Wesley.
  Introduction to databases and SQL.

1.8. Bibliography

- **ELMASRI, R.; NAVATHE, S.B.**, 1994. *Fundamentals of Database Systems*. 2nd. Redwood City, California: Addison-Wesley.
- **GULUTZAN, P.; PELZER, T.**, 1999. *SQL-99 Complete, Really*. 1nd. Lawrence, USA: R&D Books.