Change of a relation schema

- adding a new attribute (a new column) by the clause
  - `alter table <relation name> add <column definition>`
  - value `not null` is only allowed if a default value is specified

- deleting an attribute (a column) from a relation by the clause
  `alter table <relation name> drop <column definition>`

Deletion of a relation schema

- `drop table <relation name>`
- deletion of schema and relation instance

Deletion of a relation

- `delete from <relation name>`
- only the relation instance but not the schema is deleted
Creation of an index

- The goal of indexes is to improve query response time.
- An index relates to one or several attributes.
- A measure for the efficiency is in general the number of page accesses to the hard disc.

- `create [unique] index <index name> on <relation name>
  (<attribute name> [<order>] [, <attribute name> [<order>]]*) [cluster]
  
  - `<order>` ::= Asc|Desc
  - `unique`: for all indexed attribute names two tuples with the same values forbidden
    ⇒ attribute fulfil key condition

- `cluster`: The tuples of the relation are actually inserted into the index structure and not only links to the tuples.
  ⇒ only one cluster index per relation

- example: `create unique index room_index on professors (room)`

Deletion of an index

- `drop index <index name>`
Creation of views

- views correspond to external DB schemas
- In relational DBS views are regarded as derived relations which are defined by queries.
- `create view <view name> [(<attribute name> [, <attribute name>]*))] as <subquery>`
- example:
  ```sql
  create view major_students as
  select * from students where sem > 4
  ```
  The keyword "*" is a shortcut for the complete attribute list of those relations placed after `from`.

Deletion of views

- `drop view <view name>`
4.3 Data Manipulation Language (DML)

select-from-where clause

- general form:
  - select distinct $A_1, A_2, ..., A_n$
  - from $R_1, R_2, ..., R_m$
  - where $F$
  - $A_1, A_2, ..., A_n$ attribute names, $R_1, R_2, ..., R_m$ relation names, predicate $F$

- equivalent to the following relational algebra expression:
  $$\pi_{A_1, A_2, ..., A_n}(\sigma_F(R_1 \times R_2 \times ... \times R_m))$$

- The select clause corresponds to the projection operation of the relational algebra and not to the selection operation! The from clause corresponds to the Cartesian product and the where-clause to the selection predicate of the relational algebra.

- The predicate $F$ after the where clause contains
  - comparison operators $=, \neq, <, \leq, >, \geq$
  - boolean operators and, or, not
  - set operations in, not in, any, some, all
If the **where** clause is omitted, \( P = \text{true} \) holds.

The result of an SQL query can contain the same tuple resp. attribute multiple times (multiset!).

If different relations have attributes with equal names, these are distinguished by the relation name.

**Transfer of the operations of the relational algebra into SQL**

- relation \( R \)
  - `select * from R`
  - The declaration “*” in the `select`-clause indicates that all attributes of the relation \( R \) after the `from`-clause belong to the output.

- projection \( \pi_{A, B}(R) \)
  - `select distinct A, B from R`
  - Without the keyword `distinct` the result is a multiset (multi-relation).

- selection \( \sigma_F(R) \)
  - `select distinct * from R where F`

- Cartesian product \( R \times S \)
  - `select * from R, S`
theta join \( R \bowtie_F S \) on relations \( R(A, B) \) and \( S(C, D) \)

\[
\text{select * from } R, S \text{ where } F
\]

union \( R \cup S \) of the relations \( R(A, B) \) and \( S(A, B) \)

\[
\text{select * from } R \text{ union select * from } S
\]

difference \( R - S \) of the relations \( R(A, B) \) and \( S(A, B) \)

\[
\text{select * from } R \text{ minus select * from } S
\]

Duplicates and duplicate elimination

The usual select clause does not eliminate duplicates in the result relation, which therefore is a multiset (multi-relation). But this can be done by using the keyword distinct so that a relation is created as output.

The minus operation on two multisets corresponds to the semantics of the extended relational algebra. Keyword minus is only used by Oracle. In SQL92 the keyword except is used instead.

The union defined on relations automatically eliminates duplicates. This does not hold for M-relations. If duplicates are to be eliminated, the keyword all has to follow the keyword union.
Examples for SQL queries

- Find all personell ids and names of C4 professors.
  
  ```sql
  select pers-id, name from professors where rank = "C4"
  ```

<table>
<thead>
<tr>
<th>pers-id</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>2125</td>
<td>Sokrates</td>
</tr>
<tr>
<td>2126</td>
<td>Russel</td>
</tr>
<tr>
<td>2136</td>
<td>Curie</td>
</tr>
<tr>
<td>2137</td>
<td>Kant</td>
</tr>
</tbody>
</table>

- A strength of SQL is based on the fact that it is near to a natural language formulation of a command.
- Determine the different ranks of professors.

\[
\text{select distinct rank from professors}
\]

<table>
<thead>
<tr>
<th>rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>C3</td>
</tr>
<tr>
<td>C4</td>
</tr>
</tbody>
</table>

- elimination of duplicates in a table is not automatically executed for efficiency reasons (sorting necessary)
- keyword `distinct` for explicit duplicate elimination

- Determine the names of professors who hold the lecture titled “maieutics”.

\[
\text{select name, title from professors, lectures where pers-id = held_by and title = “maieutics”}
\]

<table>
<thead>
<tr>
<th>name</th>
<th>title</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sokrates</td>
<td>maieutics</td>
</tr>
</tbody>
</table>
Which students attend which lecture?

```sql
select name, title
from students, attends, lectures
where students.reg-id = attends.reg-id and
    attends.id = lectures.id
```

alternative formulation using **tuple variables** that are associated to relations:

```sql
select s.name, v.title
from students as s, attends as h, lectures as v
where s.reg-id = h.reg-id and
    h.id = v.id
```

- relationship to the tuple relational calculus observable: a variable is bound to tuples of a relation

Determine the names of all university employees, i.e., the names of all professors and all assistants.

```sql
(select name
from assistants)
union
(select name
from professors)
```
- Find all professors that are not involved in teaching.
  
  ```sql
  select name
  from professors
  where pers-id not in (select held_by from lectures)
  - operator in tests for set membership
  ```

- Find the students with the largest number of semesters.
  
  ```sql
  select name
  from students
  where sem >= all (select sem from students)
  - With the keyword all a comparison is performed with all elements of the set that is parameter of all.
  - An arbitrary comparison operator can be used with all and some.
  - all does not have the functionality of an forall quantifier, since only a comparison of a value with a set can be expressed.
  - rather similarity to an aggregate function
  - A condition with some is satisfied if it is satisfied for at least one element of the set.
Existential quantifier **exists**

- This operator checks whether a set of tuples specified by a subquery is empty. For a non-empty set the **exists** operator yields *true*, otherwise *false*. For the operator **not exists** it is just vice versa.

- Operator **exists** corresponds to the existential quantifier of the relational calculus.

- example: Which professors do not hold lectures?

```sql
select name
from professors
where not exists (select * from lectures where held_by = pers-id)
```

The renaming operator

- application of the **as**-clause

- use:
  - Two relations in the **from** clause have attributes with the same name, which would appear in the result relation without renaming.
  - If an arithmetic expression is used in the **select** clause, the result attribute does not have a name.
  - explicit change of an attribute name
example: Which credit line do the customers still have?

```
select name, credit-line - debit as rest
from customers
```

**Tuple variables**

- A tuple variable in SQL is bound to a relation.
- Tuple variables are defined in the `from` clause by means of the `as` clause.
- In particular useful in order to compare two tuples of the same relation
- example: Which students attend which lectures?

```
select s.name, v.title
from students as s, attend as a, lectures as v
where s.reg-id = a.reg-id and a.id = v.id
```

**String operations**

- Search patterns are described by
  - A percent sign (%): this represents any substring
  - An underscore (_): this represents any character
distinction between upper and lower case

String patterns in SQL are expressed with the aid of the **like** operator.

example: Find all students with names Meier, Maier, Meyer, usw.

```
select reg-id
from students
where name like „M__er“
```
Aggregate functions

- Functions **count** (number of tuples), **sum** (sum), **avg** (average), **min** (minimum) and **max** (maximum) can be applied to a set of numbers given as a column of a relation.

- no other aggregate functions in SQL2

- If the keyword **distinct** is used in front of the aggregation attribute, first duplicates are eliminated before the aggregate is computed.

- examples:
  - How many students are there with different names?
    
    ```sql
    select count (distinct name) from students
    ```
  - How many professors have a name starting with “Sch”?
    
    ```sql
    select count (*) from professors where name like "Sch%"
    ```
  - What is the average number of semesters of all students?
    
    ```sql
    select avg (sem) from students
    ```
Grouping

- general form of the select-from-where clause

```sql
select
from
[where <condition>]
[group by <group-by-expression> [, <group-by-expression>]*]
[having <condition>]
[order by <order-expression>]
```

- **group by** clause
  - A “group-by-expression” is an expression that refers only to those attributes that are *not* used for computing the aggregate. Tuples with equal values for the specified value are summarized in groups (partitions).
  - For each group the query produces a new tuple in the result relation. Hence, only attributes with one value per group are permitted after the `select` clause.

- **having** clause
  - choice of groups with respect to a condition which may contain only arguments with one value per group
examples (relation lectures extended by the attribute hpw (hours per week))

- Determine the number of hours per week in which professors have given lectures.
  ```sql
  select held_by, sum(hpw) as number
  from lectures
  group by held_by
  ```

- Determine the number of hours per week of those lectures held by professors who predominantly give long lectures (> 2 hours per week on average).
  ```sql
  select held_by, sum(hpw) as number
  from lectures
  group by held_by
  having avg(hpw) > 2
  ```

- Determine the number of hours per week of those lectures held by C4 professors who predominantly give long lectures (> 2 hours per week on average).
  ```sql
  select held_by, name, sum(hpw) as number
  from lectures, professors
  where held_by = pers-id and rank = "C4"
  group by held_by, name
  having avg(hpw) > 2
  ```