(restricted) declaration of a domain

- advantage: simple change of a data type for a domain which is used from several attributes in a schema
- form: `create domain < my type> as < type specification >`
  
  example: `create domain string as varchar(256)`

Specification of integrity constraints and default values

- Since SQL allows null values (`null`), an integrity constraint `not null` can be defined, if for a specific attribute a null value is not allowed.
- It is recommended to specify this condition for each primary key.
- definition of a default value for an attribute by attaching the clause `default < value >` to the attribute definition
- The default value is inserted into each new tuple, if an explicit value for this attribute is not specified. If a default clause is not defined, the default value is `null`.
- The clause `primary key` specifies one or more attributes that form the primary key of the relation.
- definition of a foreign key by the `foreign key` clause (referential integrity)
unique expresses that this attribute is a candidate key. If a candidate key is formed by several attributes $A_1, ..., A_n$, this is specified by the integrity constraint unique($A_1, ..., A_n$).

Creation of a relation schema

- in SQL no relations but tables (duplicates allowed)
- creation of a schema with the aid of the clause

```
create table R($A_1$ $D_1$, $A_2$ $D_2$, ..., $A_n$ $D_n$,
                [< integrity constraint_1 >, ..., < integrity constraint_k >])
```

$R$ relation name, $A_i$ name of an attribute in the schema of relation $R$, $D_i$ domain of $A_i$

- in BNF notation:

```
create table <relation name>(<relation comp> [, <relation comp>]*)
<relation comp> ::= <column definition> | <integrity constraint>
<column definition> ::= <attribute name> <type> [<default value> | not null | unique]
<default value> ::= [default <literal> | null]
```

The exact treatment of integrity constraints is discussed later.
integrity constraints

**primary key** \((A_{j_1}, ..., A_{j_m})\)

The attributes \(A_{j_1}, ..., A_{j_m}\) form the primary key of \(R\).

example: university schema (with incomplete integrity constraints)

```
create table students
  (reg-id     int not null,
   name       varchar(30) not null,
   sem        int,
   primary key (reg-id))

create table professors
  (pers-id    int not null,
   name       varchar(30) not null,
   room       int unique,
   rank       char(2),
   primary key (pers-id))
```
create table assistants
  (pers-id int not null,
   name varchar(30) not null,
   room int unique,
   boss int,
   primary key (pers-id),
   foreign key (boss) references professors(pers-id))

create table lectures
  (id int not null,
   title varchar(30),
   credits int,
   held_by int,
   primary key (id),
   foreign key (held_by) references professors(pers-id))

create table attends
  (reg-id int not null,
   id int not null,
   primary key (reg-id, id),
   foreign key (reg-id) references students(reg-id),
   foreign key (id) references lectures(id))
create table is_precondition_of
    (predecessor  int not null,
     successor    int not null,
     primary key (predecessor, successor),
     foreign key (predecessor) references lectures(id),
     foreign key (successor) references lectures(id))

create table tests
    (reg-id  int not null,
     id     int not null,
     pers-id int not null,
     grade  numeric(2,1),
     primary key (reg-id, id, pers-id),
     foreign key (reg-id) references students(reg-id),
     foreign key (id) references lectures(id),
     foreign key (pers-id) references professors(pers-id))
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>`
5.3 Data Manipulation Language (DML)

select-from-where clause

- simple 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 operation of the relational algebra.

- The predicate $F$ after the where clause contains
  - comparison operators $=, <, <=, >, >=$
  - boolean operators and, or, not
  - set operations in, not in, any, some, all
If the **where** clause is omitted, $F = true$ holds.

The result of an SQL query can contain the same tuple 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 $S(C, D)$

```sql
select * from R, S where F
```

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

```sql
select * from R union select * from S
```

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

```sql
select * from R 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` operation defined on relations automatically eliminates duplicates. If duplicates are not 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.

```sql
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”.

```sql
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? Output student names and lecture titles.

```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, l.title
from students as s, attends as a, lectures as l
where s.reg-id = a.reg-id and
     a.id = l.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)
```