- Set-valued subqueries
  - The keyword [not] in tests if an attribute [does not take] takes a value of a set.
  - If the task is to test whether an attribute is in a certain relationship to all elements of a set, the keyword all can be used.

- queries with forall quantifiers
  - mathematical law: $(\forall x : \varphi(x)) \iff (\exists x : \neg \varphi(x))$. Hence, all queries containing a forall quantifier can be transformed to equivalent queries only containing existential quantifiers.
  - example: Which students attend all lectures offered by professor Curie?

    ```sql
    select s.name from students as s
    where not exists
      (select id from lectures, professors
       where pers-id = held_by and name = "Curie")
    except
      (select l.id from attends as a, lectures as l
       where l.id = a.id and a.reg-id = s.reg-id))
    ```
- subqueries in the `from` clause
  - Since an SQL query creates a relation, a query can also be used in the `from` clause.
  - example: Output the ids of those lectures that are attended by more than 20 students.
    ```sql
    select id
    from (select id, count(*) as number from attends group by id)
    where number > 20
    ```
  - possible to explicitly use a join operator in SQL92 in the `from` clause by means of the keywords
    + `cross join` for the Cartesian (cross) product,
    + `natural join` for the natural join,
    + `join` or `inner join` for theta join,
    + `left outer join`, `right outer join` or `full outer join` for outer join
      analogously to the operators of the relational algebra: also tuples of the left, the right or both relations, which do not fulfil the join, are inserted into the result relation
    + `union join`: some kind of full outer join where no comparison is performed. Both schemas are concatenated. Tuples are united and supplemented by null values.
example

\[ \text{select} \ * \ \text{from} \ R_1, R_2 \ \text{where} \ R_1.A \ \theta \ R_2.B \]

can be explicitly formulated as theta join as follows:

\[ \text{select} \ * \ \text{from} \ R_1 \ \text{join} \ R_2 \ \text{on} \ R_1.A \ \theta \ R_2.B \]

join condition is explicitly specified behind the \text{on} clause

Null values

- A special value \textbf{null} for an attribute in a relation indicates that the value is unknown.
- SQL uses a three-valued logic with the values \textit{true}, \textit{false} and \textit{unknown}.
- Logical expressions yield the following results:

<table>
<thead>
<tr>
<th>not</th>
<th>true</th>
<th>unknown</th>
<th>false</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>false</td>
<td></td>
<td></td>
</tr>
<tr>
<td>unknown</td>
<td>unknown</td>
<td></td>
<td></td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>and</th>
<th>true</th>
<th>unknown</th>
<th>false</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>true</td>
<td>unknown</td>
<td>false</td>
</tr>
<tr>
<td>unknown</td>
<td>unknown</td>
<td>unknown</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
</tbody>
</table>
In the where clause only those tuples are selected where the filter condition yields true. Additionally the condition “where A is null” allows to select all tuples with a null value in attribute A.

For grouping null is considered as a self-contained value.

For sorting null is always interpreted as value of highest priority.

Recursive queries

e.g. example: Which lectures must be attended to understand the lecture “The Vienna Circle”?

```sql
select predecessor
from is_precondition_of, lectures
where successor = id and title = “The Vienna Circle”
```

Query returns only immediate predecessors.
example: Which lectures are required for the immediate predecessors?

```sql
select predecessor
from is_precondition_of
where successor in ( select predecessor
from is_precondition_of, lectures
where successor = id and title = "The Vienna Circle")
```

SQL, relational algebra and relational calculus do not offer possibilities for an efficient computation of recursive queries.
Insertion of tuples

- immediate input of constant values to fill relations with data
  - `insert into <relation name>[(<attribute name> [, <attribute name>]*),] values (<constant> [, <constant>]*)`
  - `insert into professors values(2136, “Curie”, 536, “C4”)`
  - input of attribute values according to the order in the schema definition
  - It is possible to insert only a part of the attribute values of a tuple, if, e.g., some values are unknown. The undefined fields are automatically filled by the system with null values.
  - `insert into students (reg-id, name) values(28121, “Archimedes”)`
- Generation of tuples by means of a query
  - `insert into <relation name>[(<attribute name> [, <attribute name>]*)]
    select ... from ... where ...
  - `insert into` attends `select` reg-id, id
    `from` students, lectures
    `where` title = “logic”

- Deletion of tuples

  - with a given condition those tuples are selected that are to be deleted
  - `delete from <relation name> [where <condition>]`
  - example: students who study longer than 8 semester are to be deleted from the relation
    `delete from students where sem > 8`
Change of tuples

- with a given condition those tuples are selected that are to be changed
- **update** <relation name>
  
  set <attribute name> = <expression> [, <attribute name> = <expression>]*

  [where <condition>]

- increase of the semester number of each student at the beginning of a new semester
  
  **update** students
  set sem = sem + 1
5.4 Views in SQL

Creation of a view

- The purpose of views is to adapt a DBS to the requirements and access rights of different user groups. They correspond to the external DB schemas.

- A view is a virtual relation (virtual table) that is derived from other relations (tables). “Virtual” means that no new tables are created. They are recalculated for each new application.

- A view determines which data a user may access and which data a user must not access.

- **create view** `<view name>` `[(<attribute name> [, <attribute name>]*)]` as `<subquery>` `[with check option]`

- Example: A view on `tests` shall express the restriction that not each user is allowed to see the results of an exam.
  ```sql
  create view tests_view as
  select reg-id, id, pers-id
  from tests
  ```