- example
  + \texttt{select * from } R_1, R_2 \texttt{ where } R_1.A \theta R_2.B
    
can be explicitly formulated as theta join as follows:
    \texttt{select * from } R_1 \texttt{ join } R_2 \texttt{ on } R_1.A \theta R_2.B
    
join condition is explicitly specified behind the \texttt{on} clause

\textbf{Null values}

- A special value \texttt{null} for an attribute in a relation indicates that the value is unknown.
- SQL uses a three-valued logic with the values \texttt{true}, \texttt{false} and \texttt{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>
<tr>
<td>or</td>
<td>true</td>
<td>unknown</td>
<td>false</td>
</tr>
<tr>
<td>------</td>
<td>------</td>
<td>---------</td>
<td>-------</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>unknown</td>
<td>true</td>
<td>unknown</td>
<td>unknown</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>unknown</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**

- 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>
  ```sql
  set <attribute name> = <expression> [ , <attribute name> = <expression>]*
  [where <condition>]
  ```
- increase of the semester number of each student at the beginning of a new semester
  ```sql
  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
  ```
Change of a view

- Views have the inherent problem that they frequently cannot be updated.

- example:
  - `create view grading(pers-id, avg-grade) as select pers-id, avg(grade) from tests group by pers-id`
  - This view is not changeable since it contains the computed attribute `avg-grade`. An update operation cannot be transferred to the original base relation `tests`. The following operation is rejected by the DBMS:
    ```
    update grading set avg-grade = 3.0 where pers-id = (select pers-id from professors where name = “Sokrates”)
    ```

- example:
  - `create view lecture-view as select title, credits, name from lectures, professors where held_by = pers-id`
– insertion of a new lecture is impossible

\textbf{insert into} lecture-view
\textbf{values} ("nihilism", 2, "Nobody")

In order to insert tuples, the DBMS would have to be able to assign the entered values to the original relations. This is not always possible. Here the view removes the keys of the original relations.

- In general, views can be updated if
  - they contain neither aggregate functions nor constructs like \texttt{distinct}, \texttt{group by}, \texttt{having}, \texttt{union} and \texttt{minus},
  - only unique column names are in the \texttt{select} list and a key of the base relation is contained and
  - they use exactly one base relation or \texttt{changeable view} in the \texttt{from} clause

- There are also views which can be updated although they do not fulfil the three aforementioned conditions.
6. Other Relational Database Query Languages

6.1 Query-by-Example (QBE)

Features of QBE

- developed at the beginning of the seventies by IBM, later part of DB2
- uses a skeleton table for the specification of a query, i.e., QBE has a two-dimensional syntax
- QBE queries are expressed by means of examples. The system generalizes the examples in order to compute answers for queries.
- declarative approach
- QBE is based on the domain relational calculus: variables are bound to attribute domains
- example database schema
  - customer(cname, caddr, account)
  - order(cname, product, amount)
  - vendor(vname, vaddr, product, price)
Onscreen dialog

- query: Find names and addresses of customers with a negative balance.
  - request for a skeleton table

```
relation name
commands related to tuples
```

- the name of the requested relation is inserted, followed by “P.” (= print)

```
customer P.          
                   
```

- attribute names are inserted by the system

```
customer P.  cname  caddr  account  
              
```

– query is specified by an entry in the table

<table>
<thead>
<tr>
<th>customer</th>
<th>cname</th>
<th>caddr</th>
<th>account</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>P.</td>
<td>P.</td>
<td>&lt; 0</td>
</tr>
</tbody>
</table>

– table is filled with values

<table>
<thead>
<tr>
<th>customer</th>
<th>cname</th>
<th>caddr</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
<td>Gainesville, FL 32611</td>
<td></td>
</tr>
<tr>
<td>Jones</td>
<td>Ocala, FL 35768</td>
<td></td>
</tr>
<tr>
<td>Meyer</td>
<td>Orlando, FL 40567</td>
<td></td>
</tr>
</tbody>
</table>

- language elements
  - commands: e.g. P, I, D.
  - domain variables: _X, _Meyer
  - constants: Smith, 123
  - boolean, arithmetic and relational operators
example: Which vendors deliver milk?

\[
\begin{array}{ccccc}
\text{vendor} & \text{vname} & \text{vaddr} & \text{product} & \text{price} \\
\text{P.}_x & \text{P.}_y & \text{milk} \\
\end{array}
\]

in the domain relational calculus this corresponds to:
\[
\{(x_1, x_2) \mid \exists x_4([x_1, x_2, \text{milk}, x_4] \in \text{vendor})\}
\]

Free domain variables, which are used only once, need not be mentioned. This leads to a simpler notation of the query:

\[
\begin{array}{ccccc}
\text{vendor} & \text{vname} & \text{vaddr} & \text{product} & \text{price} \\
\text{P.} & \text{P.} & \text{milk} \\
\end{array}
\]

example: Which vendors deliver milk or flour?

\[
\begin{array}{ccccc}
\text{vendor} & \text{vname} & \text{vaddr} & \text{product} & \text{price} \\
\text{P.} & \text{P.} & \text{milk} \\
\text{P.} & \text{P.} & \text{flour} \\
\end{array}
\]

in the domain relational calculus this corresponds to:
\([x_1, x_2] \mid \exists x_4([x_1, x_2, \text{milk}, x_4] \in \text{vendor} \lor [x_1, x_2, \text{flour}, x_4] \in \text{vendor})\)

If several pattern rows are inserted, the use of domain variables decides whether the rows are connected by a logical “or” or a logical “and”.

- example: Which vendors deliver milk for a prize between $1 and $1.20?

<table>
<thead>
<tr>
<th>Vendor</th>
<th>VName</th>
<th>VAddr</th>
<th>Product</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>P._X</td>
<td>P.</td>
<td>milk</td>
<td></td>
<td>&gt;= 1.00</td>
</tr>
<tr>
<td>_X</td>
<td></td>
<td>milk</td>
<td></td>
<td>&lt;= 1.20</td>
</tr>
</tbody>
</table>

in the domain relational calculus this corresponds to:
\([x_1, x_2] \mid \exists x_4([x_1, x_2, \text{milk}, x_4] \in \text{vendor} \land x_4 \geq 1.00 \land x_4 \leq 1.20)\)

**Condition Box**

- formulation of conditions, that do not fit into the tables, in special condition boxes
- comparison of values of two different table columns possible
- limitation of conditions inserted into a column since they can only refer to the contents of the column