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 <relation name>` 
  ```
  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
```
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

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

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 \textit{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

- attributes (further ones available)
- relation name
- commands related to tuples
- specification of the query

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

<table>
<thead>
<tr>
<th>customer P.</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>

- attribute names are inserted by the system

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

<table>
<thead>
<tr>
<th>vendor</th>
<th>vname</th>
<th>vaddr</th>
<th>product</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>P.</td>
<td>P.</td>
<td>milk</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

in the domain relational calculus this corresponds to:

\[ \{ [x_1, x_2] \mid \exists x_4 ( [x_1, x_2, \text{milk}, x_4] \in \text{vendor}) \} \]

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

<table>
<thead>
<tr>
<th>vendor</th>
<th>vname</th>
<th>vaddr</th>
<th>product</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>P.</td>
<td>P.</td>
<td>milk</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

example: Which vendors deliver milk \textit{or} flour?

<table>
<thead>
<tr>
<th>vendor</th>
<th>vname</th>
<th>vaddr</th>
<th>product</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>P.</td>
<td>P.</td>
<td>milk</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P.</td>
<td>P.</td>
<td>flour</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

in the domain relational calculus this corresponds to:
\{[x_1, x_2] | \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>\geq 1.00</td>
</tr>
<tr>
<td>_X</td>
<td></td>
<td>milk</td>
<td></td>
<td>\leq 1.20</td>
</tr>
</tbody>
</table>

in the domain relational calculus this corresponds to:

\{[x_1, x_2] | \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
example: Which vendors deliver milk or flour?

<table>
<thead>
<tr>
<th>vendor</th>
<th>vname</th>
<th>vaddr</th>
<th>product</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>P.</td>
<td>P.</td>
<td></td>
<td>_M</td>
<td></td>
</tr>
</tbody>
</table>

Conditions

_M = (milk or flour)

example: Which vendors deliver Brie and Perrier for a total price not more than $7?

<table>
<thead>
<tr>
<th>vendor</th>
<th>vname</th>
<th>vaddr</th>
<th>product</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>P._N</td>
<td></td>
<td></td>
<td>Brie</td>
<td>_X</td>
</tr>
<tr>
<td>_N</td>
<td></td>
<td></td>
<td>Perrier</td>
<td>_Y</td>
</tr>
</tbody>
</table>

Conditions

_X + _Y <= 7.00

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></td>
<td></td>
<td>milk</td>
<td>_USD</td>
</tr>
</tbody>
</table>

Conditions

_USD >= 1.00 and _USD <= 1.20