4.2 Definition of the Relational Model

Basic structure

- Given $n$ domains $D_1, D_2, ..., D_n$
  - examples for domains: data types integer, string, real, bool, date, ...
  - domains need not be disjoint, i.e., $D_i = D_j$ is admissible for $i \neq j$
  - domains may contain only atomic values, they must not be structured
- a relation (instance) $r_R$ is defined as a subset of the Cartesian product of $n$ domains:
  \[ r_R \subseteq D_1 \times D_2 \times ... \times D_n \quad (r_R \text{ finite}) \]
- $r_R$ is an occurrence (instance) of a pertaining relation schema $R$ (analogously to the programming language notions of variable and type).
- an element of the set $R$ is called tuple, tuple has arity $n$
- example:
  - domains: $D_1 = \{a, b, c\}$, $D_2 = \{0, 1\}$
  - Cartesian product: $D_1 \times D_2 = \{(a, 0), (a, 1), (b, 0), (b, 1), (c, 0), (c, 1)\}$
  - examples for instances: $r_1 = \{(a, 0), (b, 0), (c, 0), (c, 1)\}$, $r_2 = \{(a, 0)\}$, $r_3 = \emptyset$
Some basic mathematical concepts

- How is the *subset* relationship ("⊆") formally defined?
  Given two sets $A$ and $B$. Then $A \subseteq B \iff$ ?

- How is the *cross product* ("×") formally defined?
  Given two sets $A$ and $B$. Then $A \times B =$ ?

- How many elements does $A \times B$ have?

- What is a *relation* then?

- What is the difference between a *relation* and a *function*?
Schema definition

- distinction between the **schema** of a relation $R$, which is given by the $n$ domains (data types), and the current **instance** of this relation schema, which is given by a subset of the Cartesian product

- schema analogously to the programming language notion of type definition

- a **relation schema** $R$, denoted by $R(A_1, A_2, ..., A_n)$, consists of the **relation name** $R$ and a list of attributes $A_1, A_2, ..., A_n$

- each **attribute** $A_i$ is the name of a role played by domain $D_i$ in the relation schema $R$
  - $D_i$ is also the domain (type) of $A_i$
  - notation: $D_i = \text{dom}(A_i)$

- for the schema $R(A_1, A_2, ..., A_n)$ holds: $r_R \subseteq \text{dom}(A_1) \times \text{dom}(A_2) \times ... \times \text{dom}(A_n)$.

- we describe the schema of $R$ also in the form $R(A_1 : D_1, A_2 : D_2, ..., A_n : D_n)$.

- because we often do *not* make a clear distinction between the meta level (schema) and the instance level (occurrence), we also denote relation instances with the letter $R$
representation of a relation as tables with **rows** (tupels) and **columns**

<table>
<thead>
<tr>
<th>R</th>
<th>A</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

A and N are attributes and have the function of column names

example: relation Students(RegNo : *string*, Name : *string*, Age : *integer*, ...)

<table>
<thead>
<tr>
<th>Students</th>
<th>RegNo</th>
<th>Name</th>
<th>Age</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>123456</td>
<td>Meyer John</td>
<td>22</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>456123</td>
<td>Smith Ben</td>
<td>23</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>321654</td>
<td>Benson Jeff</td>
<td>27</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>654321</td>
<td>Bates Allen</td>
<td>21</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Some basic mathematical concepts

- What is the difference between a set and a list?
- A relation is defined as a set of tuples.
  
  Why could it be interesting to define a relation as a list of tuples?
- A tuple is defined as a list of attribute values.
  
  Why is it interesting to define a tuple as a set of attribute values?
Features of relations

- no order on the tuples in a relation
  - a relation is defined as a set of tuples, i.e., the tuples in a relation are *not* ordered
  - but: in a file all data records are physically ordered
  - also: the rows in a table are ordered

- order on the values in a tuple
  - according to the definition of a relation a tuple is an ordered list of \( n \) values
  - From a logical perspective an order of the attributes and their values is not important. It is only necessary to maintain the correspondence between attributes and their values.

- values in tuples
  - each value in a tuple is atomic (indivisible)
  - no composite or multivalued attributes allowed
  - first normal form
  - values of attributes in a tuple can be unknown or not apply to a specific tuple
  - use of a special **null** value for this case
Keys

- analogously to the notion of key in the E-R model
- due to the set property of relations there are no two tuples that have the same combination of values for all their attributes
- Let us assume $R(A_1, A_2, \ldots, A_n)$, and let $X \subseteq \{A_1, A_2, \ldots, A_n\}$. $X$ is called key, if the following conditions are fulfilled:
  - uniqueness: for all relation instances $r_R$ of $R$ holds:
    $$\forall t_1, t_2 \in r_R : t_1[X] = t_2[X] \Rightarrow t_1 = t_2$$
  - minimality: there is no $Y \subset X$, so that uniqueness is fulfilled
- candidate keys: several possible keys, one of them is selected as the primary key
More notions

- **database schema**: set of relation schemas
- **database**: set of current relation instances

The definitions so far allow instances that cannot exist in reality. Hence, it makes sense to restrict the instances by suitable semantical conditions.

→ **integrity constraints**
4.3 Transformation of an E-R Schema into a Relational Schema

Data structures

- of the E-R model
  - entity sets
  - relationship sets
- of the relational model
  - relation (schemas)

Problem: How can an E-R data model be transferred into a relational model?

Transformation of a strong entity set

- For each strong entity set $E$ an independent relation schema $R$ is created which comprises all simple attributes of $E$. From a composite attribute only the simple component attributes are taken.
- The names of attributes are generally selected according to the names of properties of the entity set
- The key of the entity set becomes the primary key of the relation schema
- Example: conceptual university schema (repeated)
students(reg-id : integer, name : string, sem : integer)
lectures(id : integer, credits : integer, title : string)
professors(pers-id : integer, name : string, rank : string, room : integer)
assistants(pers-id : integer, name : string, room : integer)

students_works_for(assistant : assistants, professor : professors)
students_attends(lecture : lectures, student : students)
lectures_gives(lecture : lectures, professor : professors)
lectures_is_precondition_of(lecture : lectures, test : tests)
students_tests(lecture : lectures, test : tests)

students(reg-id : integer, name : string, sem : integer)
lectures(id : integer, credits : integer, title : string)
professors(pers-id : integer, name : string, rank : string, room : integer)
assistants(pers-id : integer, name : string, room : integer)
Transformation of a weak entity set

- For each weak entity set $W$ with the respective strong entity set $E$, an independent relation schema $R$ is created which comprises all simple attributes and all simple components of composite attributes of $W$ as attributes of $R$.

- In addition, all primary key attributes of $E$ are added to $R$ as foreign key attributes. The primary key of $R$ then arises from the combination of the primary key of $E$ and the partial key of $W$, if the latter one exists.
Transformation of a 1:1-relationship set

For each binary 1:1-relationship set $R$ let $S$ and $T$ be the relation schemas that correspond to the entity sets participating in $R$. One of the relation schemas, let us say $S$, is selected, and the primary key of $T$ is added to $S$ as foreign key. It is advantageous to select an entity set with total participation in $R$ for $S$. In addition, all simple attributes and all simple components of composite attributes of $R$ are taken as attributes of $S$.

example:

department($\text{dept-no, name, location, emp-id, start}$)
employee($\text{emp-id, name, salary}$)
Transformation of a 1:m- and a m:1-relationship set

- For each binary 1:m-relationship set \( R \) let \( S \) be the relation schema which corresponds to the entity set participating in \( R \) on the \( m \)-side. Add to \( S \) as foreign key the primary key of relation schema \( T \), which corresponds to the other entity set participating in \( R \). The reason for this is that each entity on the \( m \)-side is associated with at most one entity on the 1-side of \( R \). Furthermore, all simple attributes and all simple components of composite attributes of \( R \) are taken as attributes of \( S \).

- example university database:
  - lectures(id, credits, title, held_by)
  - professors(pers-id, name, room, rank)
  - assistants(pers-id, name, room, boss)

- The names of attributes of a foreign key have partially to be changed in order to ensure the uniqueness of names in a schema.
Transformation of an $m:n$-relationship set

- For each binary $m:n$-relationship set $R$ a new relation schema $S$ is created. Add to $S$ as foreign keys the primary keys of the relation schemas that correspond to the two entity sets participating in $R$. Their combination forms the primary key of $S$. Furthermore, all simple attributes and all simple components of composite attributes of $R$ are taken as attributes of $S$.

- example university database:
  
  attends(reg-id, id)  
  is_precondition_of(predecessor, successor)

Transformation of multivalued attributes

- For each multivalued attribute $A$ a new relation schema $R$ is created. $R$ comprises an attribute corresponding to $A$ and as foreign key the primary key $K$ of the relation schema which corresponds to the entity set or relationship set containing $A$ as attribute. The primary key of $R$ is the combination of $A$ and $K$. If the multivalued attribute is composite, its simple components are added to $R$.

- example:

  ![Diagram showing the relationship between department, dept-no, location, and name]

  department(dept-no, name)  
  dept-loc(location, dept-no)
Transformation of an $n$-ary relationship set

- For each $n$-ary relationship set $R$ with $n > 2$ a new relation schema $S$ is created. Add to $S$ as foreign keys the primary keys of the relation schemas corresponding to entity sets participating in $R$. Furthermore, all simple attributes and all simple components of composite attributes of $R$ are taken as attributes of $S$. The primary key of $S$ is the combination of all foreign keys.

- example university database:

  tests(reg-id, id, pers-id, grade)

Complete schema of the university database

- students(reg-id : integer, name : string, sem : integer)
- lectures(id : integer, credits : integer, title : string, held_by : integer)
- professors(pers-id : integer, name : string, room : integer, rank : string)
- assistants(pers-id : integer, name : string, room : integer, boss : integer)
- attends(reg-id : integer, id : integer)
- is_precondition_of(predecessor : integer, successor : integer)
- tests(reg-id : integer, id : integer, pers-id : integer, grade : integer)
Transformation of generalizations

- Generalizations are not represented by an own relation. The relationship is already expressed by the fact that the key of the common superclass is also used as key of the specialized subclasses.

- example:
  - employees(pers-id, name, room)
  - professors(pers-id, rank)
  - assistants(pers-id)

- information about a professor distributed to two tuples of two relations, namely to a tuple of the relation employees and to a tuple of the relation professors

- To obtain the complete information requires a connection of both relations and tuples, respectively (join). There is no inheritance in the relational data model.