From Conceptual Model to Relational Model

Professor Matti Rossi
23rd March 2017
Figure 10-1 Systems development life cycle with design phase highlighted

- Planning
- Analysis
- Implementation
- Design
- Maintenance

- Files and Databases
  - Forms and Reports
  - Interfaces and Dialogues
  - System and Program Structure
  - Distributed Systems
Phases of Database Design

- Conceptual Design
  - End users, Application designers
- Logical Design
  - Application designers, Database programmers
- Physical Design
  - Database programmers
Logical Database Design

• Based upon the conceptual data model
• Four key steps
  1. Develop a logical data model for each known user interface for the application using normalization principles.
  2. Combine normalized data requirements from all user interfaces into one consolidated logical database model (view integration).
  3. Translate the conceptual E-R data model for the application into normalized data requirements.
  4. Compare the consolidated logical database design with the translated E-R model and produce one final logical database model for the application.
Physical Database Design

• Based upon results of logical database design
• Key decisions
  1. Choosing storage format for each attribute from the logical database model
  2. Grouping attributes from the logical database model into physical records
  3. Arranging related records in secondary memory (hard disks and magnetic tapes) so that records can be stored, retrieved and updated rapidly
  4. Selecting media and structures for storing data to make access more efficient
Deliverables and Outcomes

• Logical database design
  – must account for every data element on a system input or output
  – normalized relations are the primary deliverable

• Physical database design
  – converting relations into database tables
Relational Database vs. Relational Model

- **Relational Database**: an implementation of a relational model, data represented as a set of related tables (one table per relation)
- **Relational Model**: a set of relations
- **Relation**: a named, two-dimensional table of data. Each relation consists of a set of named columns and an arbitrary number of unnamed rows
- **Well-Structured Relation**: a relation that contains a minimum amount of redundancy and allows users to insert, modify, and delete the rows without errors or inconsistencies
Relational model

- The relational model is a logical data model, which represents data as a set of relations (term table is often substituted for relation in informal presentations).
- Relational model defines data from the end user point of view. It does not define physical organization of data.
- Relational model can be directly implemented in a relational database.
Key concepts

- **Relation (relaatio)** is a set of structurally identical tuples.
- **Tuple (monikko)** is an ordered list of values in which one value exists for every attribute named in the relational schema (relaatiokaavio).
- **Attribute (attribuutti)** is a header, which implies the meaning of an entity in the relation.
- **Domain (alue)** is a set of atomistic values defining the legal contents of an attribute.
Domain

• Domains consist of indivisible (atomistic) values.
• A domain is usually defined as a combination of the domain name and data format.
• There is no standard format for defining domains.
Key concepts pictured

Relation

Attributes

<table>
<thead>
<tr>
<th>EMPLOYEE</th>
<th>SSN</th>
<th>LNAME</th>
<th>FNAME</th>
<th>DEPT NO</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>220333-042F</td>
<td>Kirk</td>
<td>James</td>
<td>666</td>
</tr>
</tbody>
</table>
## Relational database model

**Employee1**

<table>
<thead>
<tr>
<th>EMP_ID</th>
<th>NAME</th>
<th>DEPT</th>
<th>SALARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>Margaret Simpson</td>
<td>Marketing</td>
<td>42,000</td>
</tr>
<tr>
<td>140</td>
<td>Allen Beeton</td>
<td>Accounting</td>
<td>39,000</td>
</tr>
<tr>
<td>110</td>
<td>Chris Lucero</td>
<td>Info Systems</td>
<td>41,500</td>
</tr>
<tr>
<td>190</td>
<td>Lorenzo Davis</td>
<td>Finance</td>
<td>38,000</td>
</tr>
<tr>
<td>150</td>
<td>Susan Martin</td>
<td>Marketing</td>
<td>38,500</td>
</tr>
</tbody>
</table>

**EMPLOYEE1 (Emp_ID, Name, Dept, Salary)**
Key concepts of a relational database

• **Table (taulu)** is a set of tuples based on the same row definition. Table is the implementation of a relation in a relational database.

• **Row (rivi)** is an ordered list of values, in which there exists one value for every domain of the row definition.

• **Column (sarake)** is a list of values implementing the same domain in different rows of the table.
Comparison of concepts

The following analogies can be drawn between the concepts of relational model, relational database table, and a data file:

<table>
<thead>
<tr>
<th>RELATION</th>
<th>TABLE</th>
<th>FILE</th>
</tr>
</thead>
<tbody>
<tr>
<td>tuple</td>
<td>row</td>
<td>record</td>
</tr>
<tr>
<td>attribute</td>
<td>column</td>
<td>field</td>
</tr>
</tbody>
</table>
Relational database model – example 2: relation with redundancy

<table>
<thead>
<tr>
<th>EMP ID</th>
<th>NAME</th>
<th>DEPT</th>
<th>SALARY</th>
<th>COURSE</th>
<th>Date_Completed</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>Margaret Simpson</td>
<td>Marketing</td>
<td>42,000</td>
<td>SPSS</td>
<td>6/19/2005</td>
</tr>
<tr>
<td>100</td>
<td>Margaret Simpson</td>
<td>Marketing</td>
<td>42,000</td>
<td>Surveys</td>
<td>12/8/2010</td>
</tr>
<tr>
<td>140</td>
<td>Allen Beeton</td>
<td>Accounting</td>
<td>39,000</td>
<td>SPSS</td>
<td>06/05/2009</td>
</tr>
<tr>
<td>140</td>
<td>Allen Beeton</td>
<td>Accounting</td>
<td>39,000</td>
<td>Investments</td>
<td>30/05/2006</td>
</tr>
<tr>
<td>110</td>
<td>Chris Lucero</td>
<td>Info Systems</td>
<td>41,500</td>
<td>SPSS</td>
<td>15/06/2011</td>
</tr>
<tr>
<td>190</td>
<td>Lorenzo Davis</td>
<td>Finance</td>
<td>38,000</td>
<td>Surveys</td>
<td>10/06/2012</td>
</tr>
<tr>
<td>150</td>
<td>Susan Martin</td>
<td>Marketing</td>
<td>38,500</td>
<td>SPSS</td>
<td>14/12/2009</td>
</tr>
</tbody>
</table>
Characteristics of relation

• A relation is by definition a set of tuples, with no prescribed order.
• Relational data is addressed with the relation name, attribute name and key value identifying the tuple.
• Example:
  SQL statements to retrieve data from Employee:

  ```sql
  SELECT FNAME, LNAME
  FROM EMPLOYEE
  WHERE SSN='010170-666A';

  SELECT *
  FROM EMPLOYEE;
  ```
Characteristics of a tuple

• A tuple is an ordered list of attributes. The order of the attributes is (at least in theory) irrelevant.
• Tuples are unique.
• Tuples are identified by the key value (one attribute or a combination of multiple attributes).
Values in tuples

• The values of tuples are atomistic, i.e. one value for one attribute for any given tuple.
• Multivalued attributes should be represented in separate relations.
• If no value exists for an attribute in a tuple, the missing value will be represented by a special *null* value.
Constraints

• Domain constraints specify that within each tuple, the value of each attribute $A$ must be an atomic value from the domain $\text{dom}(A)$.

• Key Constraints. A relation is defined as a set of tuples. By definition, all elements of a set are distinct; hence, all tuples in a relation must also be distinct.
Superkey

• Any subset of attributes of a relational schema with the property that no two tuples in any relation state should have the same combination of values is called a superkey.

• A superkey specifies a uniqueness constraint.

• A superkey can have redundant attributes, however, so a more useful concept is that of a key, which has no redundancy.
A key $K$ of a relation schema $R$ is a superkey with the additional property that removing any attribute $A$ from $K$ leaves a set of attributes $K'$ that is not a superkey of $R$ any more.

In general, a relation schema may have more than one key. In this case, each of the keys is called a candidate key.

It is common to designate one of the candidate keys as the primary key in the relation.
Integrity

• Integrity constraints:
  – The entity integrity constraint (perusavaimen eheys) states that no primary key value can be null.
  – The referential integrity constraint (viite-eheys) states that a tuple in one relation that refers to another relation must refer to an existing tuple in that relation.
  – Semantic integrity constraints (erityiseheys) are application specific, yet, general rules specified on a relational database.

• Integrity constraints safeguard against inconsistent data due to update operations.
Relation update operations

- **INSERT (LISÄYS)**
  - Insert a tuple into the relation.

- **DELETE (POISTO)**
  - Deletion of a tuple in the relation.

- **UPDATE (MUUTOS)**
  - Changing value(s) of an attribute (attributes) in the tuple.
Insert Operation

• Inserting a tuple may violate the integrity constraints of a relation if
  – some of the offered attribute values are not consistent with the domain,
  – the new tuple has identical primary key value with an existing tuple, or
  – referential key attribute refers to a non-existing tuple in another relation.
Delete Operation

• Deletion may offend only referential integrity. If deletion is attempted for a tuple, which is referenced from another relation.

• Example:
  – An attempt to Delete the basic employee information when dependants exist in another relation for the employee.
Update Operation

• Updating an attribute that is neither a primary key nor a foreign key usually causes no problems; the DBMS need only check to confirm that the new value is of the correct data type and domain.

• Changing the primary key value is tantamount to deleting the existing tuple and inserting a new one.
Transforming E-R Diagrams into Relations

• It is useful to transform the conceptual data model into a set of normalized relations

• Steps
  – Represent entity types
  – Represent relationship types
  – Normalize the relations
  – Merge the relations
Representing Entity types

- Each regular entity type is transformed into a relation.
- The identifier of the entity type becomes the primary key of the corresponding relation.
- The primary key must satisfy the following two conditions.
  1) The value of the key must uniquely identify every row in the relation.
  2) The key should be non-redundant.
Figure 10-10a  Transforming an entity type to a relation - E-R Diagram

Figure 10-10b  Transforming an entity type to a relation - Relation

<table>
<thead>
<tr>
<th>Customer_ID</th>
<th>Name</th>
<th>Address</th>
<th>City_State_ZIP</th>
<th>Discount</th>
</tr>
</thead>
<tbody>
<tr>
<td>1273</td>
<td>Contemporary Designs</td>
<td>123 Oak St.</td>
<td>Austin, TX 28384</td>
<td>5%</td>
</tr>
<tr>
<td>6390</td>
<td>Casual Corner</td>
<td>18 Hoosier Dr.</td>
<td>Bloomington, IN 45821</td>
<td>3%</td>
</tr>
</tbody>
</table>
Represent Relationships

• Binary 1:N Relationships
  – Add the primary key attribute (or attributes) of the entity on the one side of the relationship as a foreign key in the relation on the right side.
  – The one side *migrates* to the many side.

• Binary or Unary 1:1
  – Three possible options
    a. Add the primary key of A as a foreign key of B.
    b. Add the primary key of B as a foreign key of A.
    c. Both of the above.
Represent Relationships (cont.)

Figure 10-11a Representing a 1:N relationship - E-R Diagram
Figure 10-11b  Representing a 1:N relationship - Relations

<table>
<thead>
<tr>
<th>Customer_ID</th>
<th>Name</th>
<th>Address</th>
<th>City_State_ZIP</th>
<th>Discount</th>
</tr>
</thead>
<tbody>
<tr>
<td>1273</td>
<td>Contemporary Designs</td>
<td>123 Oak St.</td>
<td>Austin, TX 28384</td>
<td>5%</td>
</tr>
<tr>
<td>6390</td>
<td>Casual Corner</td>
<td>18 Hoosier Dr.</td>
<td>Bloomington, IN 45821</td>
<td>3%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Order_Number</th>
<th>Order_Date</th>
<th>Promised_Date</th>
<th>Customer_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>57194</td>
<td>3/15/0X</td>
<td>3/28/0X</td>
<td>6390</td>
</tr>
<tr>
<td>63725</td>
<td>3/17/0X</td>
<td>4/01/0X</td>
<td>1273</td>
</tr>
<tr>
<td>80149</td>
<td>3/14/0X</td>
<td>3/24/0X</td>
<td>6390</td>
</tr>
</tbody>
</table>
Represent Relationships (cont.)

• Binary and Higher M:N relationships
  – Create another relation and include primary keys of all relations as primary key of new relation.
Represent Relationships (cont.)

Figure 10-12a  Representing an M:N relationship - E-R Diagram
Represent Relationships

**Figure 10-12b** Representing an M:N relationship - Relations

**ORDER**

<table>
<thead>
<tr>
<th>Order_Number</th>
<th>Order_Date</th>
<th>Promised_Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>61384</td>
<td>2/17/2005</td>
<td>3/01/2005</td>
</tr>
<tr>
<td>62009</td>
<td>2/13/2005</td>
<td>2/27/2005</td>
</tr>
<tr>
<td>62807</td>
<td>2/15/2005</td>
<td>3/01/2005</td>
</tr>
</tbody>
</table>

**ORDER LINE**

<table>
<thead>
<tr>
<th>Order_Number</th>
<th>Product_ID</th>
<th>Quantity_Ordered</th>
</tr>
</thead>
<tbody>
<tr>
<td>61384</td>
<td>M128</td>
<td>2</td>
</tr>
<tr>
<td>61384</td>
<td>A261</td>
<td>1</td>
</tr>
</tbody>
</table>

**PRODUCT**

<table>
<thead>
<tr>
<th>Product_ID</th>
<th>Description</th>
<th>Room</th>
<th>(Other Attributes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>M128</td>
<td>Bookcase</td>
<td>Study</td>
<td>–</td>
</tr>
<tr>
<td>A261</td>
<td>Wall unit</td>
<td>Family</td>
<td>–</td>
</tr>
<tr>
<td>R149</td>
<td>Cabinet</td>
<td>Study</td>
<td>–</td>
</tr>
</tbody>
</table>
Represent Relationships (cont.)

- **Unary 1:N Relationships**
  - Relationship between instances of a single entity type
  - Utilize a recursive foreign key
    - A foreign key in a relation that references the primary key values of that same relation.

- **Unary M:N Relationships**
  - Create a separate relation.
  - Primary key of new relation is a composite of two attributes that both take their values from the same primary key.
**Figure 10-13a** Two unary relations - EMPLOYEE with Manages relationship (1:N)

EMPLOYEE(Emp_ID, Name, Birthdate, Manager_ID)
ITEM(Item_Number, Name, Cost)
ITEMCOMPONENT(Item_Number, Component_Number, Quantity)
<table>
<thead>
<tr>
<th>E-R Structure</th>
<th>Relational representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regular entity</td>
<td>Create a relation with primary key and nonkey attributes.</td>
</tr>
<tr>
<td>Weak entity</td>
<td>Create a relation with a composite primary key (which includes the primary key of the entity on which this weak entity depends) and nonkey attributes.</td>
</tr>
<tr>
<td>Binary or unary 1:1 relationship</td>
<td>Place the primary key of either entity in the relation for the other entity or do this for both entities.</td>
</tr>
<tr>
<td>Binary 1:N relationship</td>
<td>Place the primary key of the entity on the one side of the relationship as a foreign key in the relation for the entity on the many side.</td>
</tr>
<tr>
<td>Binary or unary M:N relationship or associative entity</td>
<td>Create a relation with a composite primary key using the primary keys of the related entities, plus any nonkey attributes of the relationship or associate entity.</td>
</tr>
<tr>
<td>Binary or unary M:N relationship or associative entity with additional key(s)</td>
<td>Create a relation with a composite primary key using the primary keys of the related entities and additional primary key attributes associated with the relationship or associative entity, plus any nonkey attributes of the relationship or associative entity.</td>
</tr>
<tr>
<td>Binary or unary M:N relationship or associative entity with its own key</td>
<td>Create a relation with the primary key associated with the relationship or associative entity, plus any nonkey attributes of the relationship or associative entity and the primary keys of the related entities (as foreign key attributes).</td>
</tr>
<tr>
<td>Supertype/subtype relationship</td>
<td>Create a relation for the superclass, which contains the primary key and all nonkey attributes in common with all subclasses, plus create a separate relation for each subclass with the same primary key (with the same or local name) but with only the nonkey attributes related to that subclass.</td>
</tr>
</tbody>
</table>
Representing relationship types with foreign keys

- **SSN** is the primary key for the employee table, and **DNUMBER** for the Department table.
- **DNO** attribute in the Employee table is a foreign key referencing tuples in the department table.
- **MGRSSN** attribute in the Department table is a foreign key referencing tuples in the employee table.