Mapping an ERD to a Relational Database

We use E/R Modeling to understand the informational needs of a system. Once the model is satisfactory, we then implement our design in a relational database. In order to implement a database we must decide on the relations, their attributes and primary keys. This section contains some simple rules that we use to map an ERD to a relational database.

In general, relations are used to hold entity sets and to hold relationship sets. The considerations to be made are listed below. We consider that points 1a), 1b), 2b), and 3a) are likely used more often than the others. After we present the mapping rules, we discuss their application in a few examples.

1. **Entity Types**

Each entity type is implemented with a separate relation. It is important to distinguish entity types as being either strong or weak.

   a. **Strong Entity Types**

      Strong, or regular, entity types are mapped to their own relation. The key attributes are included and the PK is chosen to be one of these keys.

   b. **Weak Entity Types**

      Weak entity types are mapped to their own relation. The PK attributes of any relations created from the related identifying entity types are included as FK attributes. The PK of the relation for the weak entity type is the combination of the PKs of relations (from the related identifying entity types) and the discriminator of the weak entity.

2. **Relationships**

The implementation of relationships involve the use of foreign keys. Recall, as discussed in point 1b) above, that if the relationship is identifying, then the primary key of the strong entity must be propagated to the relation representing the weak entity. It is important to consider both the degree and the cardinality of a relationship. In the following, the first three cases deal with binary relationships and the fourth concerns $n$-ary relationships.

   a. **Binary One-To-One**

      In general, with a one-to-one relationship, you have a choice regarding where to implement the relationship. You may choose to place a foreign key in one of the
two relations, or in both. Also, include any attributes defined for the relationship in the same table as the foreign key.

b. Binary One-To-Many

With a one-to-many relationship you must place a foreign key (and any attributes defined for the relationship) in the relation corresponding to the many side of the relationship.

c. Binary Many-To-Many

A many-to-many relationship must be implemented with a separate relation. This new relation will have a composite primary key comprising the primary keys of the relations implementing the participating entity types plus any discriminator attribute. Also include as attributes, in this relation, any attributes defined for the relationship.

d. n-ary, n>2

A new relation is generated for an n-ary relationship type. The attributes in this relation include FKs for each relation (of the related entity types) plus any other attributes defined for the n-ary relationship. This new relation has a composite primary key comprising the n primary keys of the relations of the participating entity types plus any discriminator attribute. There is one exception to the formation of the PK: if the cardinality for any entity type is 1, then the primary key of the relations for that entity type is only included as a foreign key and not as part of the primary key of the new relation.

3. Attributes

All attributes, with the exception of derived and composite attributes, must appear in relations. You choose to include derived attributes if their presence will improve performance. If you include a derived attribute in a relation, then you must determine how its values are maintained. In the following we consider attributes according to whether they are atomic(simple), multi-valued, or composite.

a. Simple, atomic

These are included in the relation created for the pertinent entity set, many-to-many relationship, or n-ary relationship.

b. Multi-valued

Each multi-valued attribute is implemented using a new relation. This relation will include the primary key of the original entity type and the multi-valued attribute. The primary key of this relation is the primary key of the original entity
set and the multi-valued attribute. Note that in this new relation, the attribute is no longer considered as multi-valued.

c. Composite

Composite attributes are not included. However the simple attributes comprising the composite attribute do appear in the pertinent relation.

4. Participation constraints

If a relationship is mandatory for an entity type, and if the entity type is on the “many” side of the relationship, then a specification for the relation is needed to ensure a foreign key has a value, and that it cannot be null. This specification would be a "NOT NULL" attribute constraint in the DDL of an SQL system, or a "yes" for an attribute's "required" property in an MS Access relation.

If a relationship is mandatory for an entity type, and if the entity type is on the “one” side of a relationship, then a check constraint, database trigger, or application code would be used to enforce the constraint.
Example 1: Mapping an ERD to a Relational Database

Consider the ERD:

![ERD Diagram]

Each entity is strong and all attributes are simple, so we have an Employee relation and a Department relation. Because of the 1-to-many relationship we must have a foreign key on the many side of the works in relationship. So, the Employee relation has a foreign key DeptNum.

Our relations are:

<table>
<thead>
<tr>
<th>Employee</th>
<th>EmpNum</th>
<th>EmpFname</th>
<th>EmpLname</th>
<th>DeptNum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Department</td>
<td>DeptNum</td>
<td>DeptName</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The FK DeptNum needs a constraint to ensure it is never null.
Example 2: Mapping an ERD to a Relational Database

Consider the following (star schema) design where there is one "fact" entity type and three "dimension" entity types. The terms fact and dimension are used in "dimensional" modeling.

This diagram has three identifying 1-to-many relationships. By applying our mapping rules, we have four relations: three for the dimension entity types (Product, Store, Date) and one for the fact entity type (Sale). The fact relation will have three foreign keys referencing the dimension relations. The primary key of the fact relation comprises those same foreign keys.
Our relations are:

**Store**

<table>
<thead>
<tr>
<th>Ssk</th>
<th>StLocn</th>
</tr>
</thead>
</table>

**Product**

<table>
<thead>
<tr>
<th>Psk</th>
<th>ProdTitle</th>
</tr>
</thead>
</table>

**Date**

<table>
<thead>
<tr>
<th>Dsk</th>
<th>Day</th>
<th>Week</th>
<th>Month</th>
<th>Quarter</th>
<th>Year</th>
</tr>
</thead>
</table>

**Sale**

<table>
<thead>
<tr>
<th>Ssk</th>
<th>Psk</th>
<th>Dsk</th>
<th>Price</th>
<th>Qty</th>
</tr>
</thead>
</table>
Example 3: Mapping an ERD to a Relational Database

This example is similar to the previous one, but we instead of a fact entity set we have a 3-ary (ternary) relationship (a many-to-many-to-many relationship).

Now, applying our mapping rules we end up with the same physical database as the previous example.