2.6 References for Chapter 2

The original paper on the Entity-Relationship model is [2]. Two modern books on the subject of E/R design are [1] and [3].

1. Batini, Carlo., S. Ceri, S. B. Navathe, and Carol Batini, *Conceptual Database Design: an Entity/Relationship Approach*, Addison-Wesley, Reading MA, 1991.

2. Chen, P. P., "The entity-relationship model: toward a unified view of data," *ACM Trans. on Database Systems* 1:1, pp. 9–36, 1976.

3. Thalheim, B., "Fundamentals of Entity-Relationship Modeling," Springer-Verlag, Berlin, 2000.

Chapter 3

The Relational Data Model

While the entity-relationship approach to data modeling that we discussed in Chapter 2 is a simple and appropriate way to describe the structure of data, today's database implementations are almost always based on another approach, called the *relational model*. The relational model is extremely useful because it has but a single data-modeling concept: the "relation," a two-dimensional table in which data is arranged. We shall see in Chapter 6 how the relational model supports a very high-level programming language called SQL (structured query language). SQL lets us write simple programs that manipulate in powerful ways the data stored in relations. In contrast, the E/R model generally is not considered suitable as the basis of a data manipulation language.

On the other hand, it is often easier to design databases using the E/R notation. Thus, our first goal is to see how to translate designs from E/R notation into relations. We shall then find that the relational model has a design theory of its own. This theory, often called "normalization" of relations, is based primarily on "functional dependencies," which embody and expand the concept of "key" discussed informally in Section 2.3.2. Using normalization theory, we often improve our choice of relations with which to represent a particular database design.

3.1 Basics of the Relational Model

The relational model gives us a single way to represent data: as a two-dimensional table called a *relation*. Figure 3.1 is an example of a relation. The name of the relation is `Movies`, and it is intended to hold information about the entities in the entity set *Movies* of our running design example. Each row corresponds to one movie entity, and each column corresponds to one of the attributes of the entity set. However, relations can do much more than represent entity sets, as we shall see.

title	*year*	*length*	*filmType*
Star Wars	1977	124	color
Mighty Ducks	1991	104	color
Wayne's World	1992	95	color

Figure 3.1: The relation Movies

3.1.1 Attributes

Across the top of a relation we see *attributes*; in Fig. 3.1 the attributes are title, year, length, and filmType. Attributes of a relation serve as names for the columns of the relation. Usually, an attribute describes the meaning of entries in the column below. For instance, the column with attribute length holds the length in minutes of each movie.

Notice that the attributes of the relation Movies in Fig. 3.1 are the same as the attributes of the entity set *Movies*. We shall see that turning one entity set into a relation with the same set of attributes is a common step. However, in general there is no requirement that attributes of a relation correspond to any particular components of an E/R description of data.

3.1.2 Schemas

The name of a relation and the set of attributes for a relation is called the *schema* for that relation. We show the schema for the relation with the relation name followed by a parenthesized list of its attributes. Thus, the schema for relation Movies of Fig. 3.1 is

```
Movies(title, year, length, filmType)
```

The attributes in a relation schema are a set, not a list. However, in order to talk about relations we often must specify a "standard" order for the attributes. Thus, whenever we introduce a relation schema with a list of attributes, as above, we shall take this ordering to be the standard order whenever we display the relation or any of its rows.

In the relational model, a design consists of one or more relation schemas. The set of schemas for the relations in a design is called a *relational database schema*, or just a *database schema*.

3.1.3 Tuples

The rows of a relation, other than the header row containing the attribute names, are called *tuples*. A tuple has one *component* for each attribute of the relation. For instance, the first of the three tuples in Fig. 3.1 has the four components Star Wars, 1977, 124, and color for attributes title, year, length, and filmType, respectively. When we wish to write a tuple in isolation, not as part of a relation, we normally use commas to separate components, and we use parentheses to surround the tuple. For example,

```
(Star Wars, 1977, 124, color)
```

is the first tuple of Fig. 3.1. Notice that when a tuple appears in isolation, the attributes do not appear, so some indication of the relation to which the tuple belongs must be given. We shall always use the order in which the attributes were listed in the relation schema.

3.1.4 Domains

The relational model requires that each component of each tuple be atomic; that is, it must be of some elementary type such as integer or string. It is not permitted for a value to be a record structure, set, list, array, or any other type that can reasonably have its values broken into smaller components.

It is further assumed that associated with each attribute of a relation is a *domain*, that is, a particular elementary type. The components of any tuple of the relation must have, in each component, a value that belongs to the domain of the corresponding column. For example, tuples of the Movies relation of Fig. 3.1 must have a first component that is a string, second and third components that are integers, and a fourth component whose value is one of the constants color and blackAndWhite. Domains are part of a relation's schema, although we shall not develop a notation for specifying domains until we reach Section 6.6.2.

3.1.5 Equivalent Representations of a Relation

Relations are sets of tuples, not lists of tuples. Thus the order in which the tuples of a relation are presented is immaterial. For example, we can list the three tuples of Fig. 3.1 in any of their six possible orders, and the relation is "the same" as Fig. 3.1.

Moreover, we can reorder the attributes of the relation as we choose, without changing the relation. However, when we reorder the relation schema, we must be careful to remember that the attributes are column headers. Thus, when we change the order of the attributes, we also change the order of their columns. When the columns move, the components of tuples change their order as well. The result is that each tuple has its components permuted in the same way as the attributes are permuted.

For example, Fig. 3.2 shows one of the many relations that could be obtained from Fig. 3.1 by permuting rows and columns. These two relations are considered "the same." More precisely, these two tables are different presentations of the same relation.

year	*title*	*filmType*	*length*
1991	Mighty Ducks	color	104
1992	Wayne's World	color	95
1977	Star Wars	color	124

Figure 3.2: Another presentation of the relation `Movies`

3.1.6 Relation Instances

A relation about movies is not static; rather, relations change over time. We expect that these changes involve the tuples of the relation, such as insertion of new tuples as movies are added to the database, changes to existing tuples if we get revised or corrected information about a movie, and perhaps deletion of tuples for movies that are expelled from the database for some reason.

It is less common for the schema of a relation to change. However, there are situations where we might want to add or delete attributes. Schema changes, while possible in commercial database systems, are very expensive, because each of perhaps millions of tuples needs to be rewritten to add or delete components. If we add an attribute, it may be difficult or even impossible to find the correct values for the new component in the existing tuples.

We shall call a set of tuples for a given relation an *instance* of that relation. For example, the three tuples shown in Fig. 3.1 form an instance of relation `Movies`. Presumably, the relation `Movies` has changed over time and will continue to change over time. For instance, in 1980, `Movies` did not contain the tuples for `Mighty Ducks` or `Wayne's World`. However, a conventional database system maintains only one version of any relation: the set of tuples that are in the relation "now." This instance of the relation is called the *current instance.*

3.1.7 Exercises for Section 3.1

Exercise 3.1.1: In Fig. 3.3 are instances of two relations that might constitute part of a banking database. Indicate the following:

a) The attributes of each relation.

b) The tuples of each relation.

c) The components of one tuple from each relation.

d) The relation schema for each relation.

e) The database schema.

f) A suitable domain for each attribute.

g) Another equivalent way to present each relation.

acctNo	*type*	*balance*
12345	savings	12000
23456	checking	1000
34567	savings	25

The relation `Accounts`

firstName	*lastName*	*idNo*	*account*
Robbie	Banks	901-222	12345
Lena	Hand	805-333	12345
Lena	Hand	805-333	23456

The relation `Customers`

Figure 3.3: Two relations of a banking database

!! **Exercise 3.1.2:** How many different ways (considering orders of tuples and attributes) are there to represent a relation instance if that instance has:

* a) Three attributes and three tuples, like the relation `Accounts` of Fig. 3.3?

b) Four attributes and five tuples?

c) n attributes and m tuples?

3.2 From E/R Diagrams to Relational Designs

Let us consider the process whereby a new database, such as our movie database, is created. We begin with a design phase, in which we address and answer questions about what information will be stored, how information elements will be related to one another, what constraints such as keys or referential integrity may be assumed, and so on. This phase may last for a long time, while options are evaluated and opinions are reconciled.

The design phase is followed by an implementation phase using a real database system. Since the great majority of commercial database systems use the relational model, we might suppose that the design phase should use this model too, rather than the E/R model or another model oriented toward design.

However, in practice it is often easier to start with a model like E/R, make our design, and then convert it to the relational model. The primary reason for doing so is that the relational model, having only one concept — the relation —

Schemas and Instances

Let us not forget the important distinction between the schema of a relation and an instance of that relation. The schema is the name and attributes for the relation and is relatively immutable. An instance is a set of tuples for that relation, and the instance may change frequently.

The schema/instance distinction is common in data modeling. For instance, entity set and relationship descriptions are the E/R model's way of describing a schema, while sets of entities and relationship sets form an instance of an E/R schema. Remember, however, that when designing a database, a database instance is not part of the design. We only imagine what typical instances would look like, as we develop our design.

rather than several complementary concepts (e.g., entity sets and relationships in the E/R model) has certain inflexibilities that are best handled after a design has been selected.

To a first approximation, converting an E/R design to a relational database schema is straightforward:

- Turn each entity set into a relation with the same set of attributes, and
- Replace a relationship by a relation whose attributes are the keys for the connected entity sets.

While these two rules cover much of the ground, there are also several special situations that we need to deal with, including:

1. Weak entity sets cannot be translated straightforwardly to relations.
2. "Isa" relationships and subclasses require careful treatment.
3. Sometimes, we do well to combine two relations, especially the relation for an entity set E and the relation that comes from a many-one relationship from E to some other entity set.

3.2.1 From Entity Sets to Relations

Let us first consider entity sets that are not weak. We shall take up the modifications needed to accommodate weak entity sets in Section 3.2.4. For each non-weak entity set, we shall create a relation of the same name and with the same set of attributes. This relation will not have any indication of the relationships in which the entity set participates; we'll handle relationships with separate relations, as discussed in Section 3.2.2.

Example 3.1: Consider the three entity sets *Movies*, *Stars* and *Studios* from Fig. 2.17, which we reproduce here as Fig. 3.4. The attributes for the *Movies* entity set are *title*, *year*, *length*, and *filmType*. As a result, the relation `Movies` looks just like the relation `Movies` of Fig. 3.1 with which we began Section 3.1.

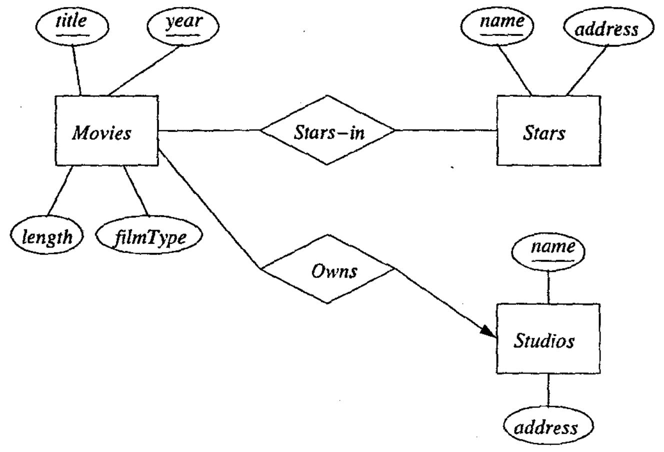

Figure 3.4: E/R diagram for the movie database

Next, consider the entity set *Stars* from Fig. 3.4. There are two attributes, *name* and *address*. Thus, we would expect the corresponding `Stars` relation to have schema `Stars(name, address)` and for a typical instance of the relation to look like:

name	*address*
`Carrie Fisher`	`123 Maple St., Hollywood`
`Mark Hamill`	`456 Oak Rd., Brentwood`
`Harrison Ford`	`789 Palm Dr., Beverly Hills`

□

3.2.2 From E/R Relationships to Relations

Relationships in the E/R model are also represented by relations. The relation for a given relationship R has the following attributes:

1. For each entity set involved in relationship R, we take its key attribute or attributes as part of the schema of the relation for R.
2. If the relationship has attributes, then these are also attributes of relation R.