Diseño lógico: Relacional

CI-0127 Bases de Datos, Universidad de Costa Rica

Sivana Hamer

Importante: Este documento recopila contenidos de diversos de sitios web especializados, académicos y documentos compartidos por universidades. Toda la información es utilizada con fines estrictamente académicos. Para más información, puede ver las referencias.

1 Relational model

The relational model is a logical model based on relations. As it is a logical model, data is not stored physically as it is represented in the model. As an example, we shall use the SCHOOL relation shown in Fig. 1 that represents the SCHOOLs within our university with some fictional data and suppositions.

Name	Acronym	Phone_number	Number_students
Ciencias de la Computación e Informática	ECCI	2511-8000	888
Ciencias de la Comunicación Colectiva	ECCC	2511-3600	999
Lenguas Modernas	ELM	2511 8391	NULL
Administración de Negocios	EAN	2511-9180	3000
Antropología	EAT	2511-6458	500
Matemática	EMat	2511-6551	1500

Figure 1: SCHOOL relation

Relations

A *relation* or *"relationship"* (not really the correct term) is an unordered set of attributes. We can think of a relation as an unordered *table* with columns (i.e., attributes) and rows (i.e., tuples). The order of both the attributes and rows does not matter (with some exceptions).

A *relational scheme, schema or intention* defines a relation. It is represented as $R(A_1, A_2, \cdots, A_n)$

- R is the relational name. We write relational names in upper case. Common relational names include Q, R and S.
- A_i represents an attribute.
- The *degree or arity* of a relation is the number of attributes A_i.
- The *cardinality* is the total number of combinations of tuples (cartesian product $\times$) with values $|D|$ possible for all attributes in the relation.

Example: We can represent the relation schema of $SCHOOL = S$ as either of the following:

$$SCHOOL(Name, Acronym, Phone\_number, Number\_students) =$$

$$S(Name : string, Acronym : string, Phone\_number : string, Number\_students : integer)$$

The relation has a degree of four as there are four attributes.

A *relation state or extension* describes the tuples currently stored in the relation. It is represented as $r = r(R) = \{t_1, t_2, \cdots, t_m\}$.

- r is the current state of the relation. We write relational states in lower case. Common relational states include q, r and s.
- R is the relational name of the relational schema.
- t_j represents a tuple. Tuples have not specified orders within the relation.

Example: We have several tuples within the SCHOOL relation (non gray rows).

Attribute

An *attribute* A_i describes an aspect or role of an unordered set of values within a relation R. Each attribute has a *domain* of possible values within the column. The relation and column name provides meaning to the set of values within the relation.

Example: There are several attributes for each school within our school. These include:

- **Name:** The name of the school.
- **Acronym:** The acronym of the school.
- **Phone_number:** A phone number to contact the school. A school may have multiple phone numbers.
- **Number_students:** The number of students that are currently enrolled in the school.

Domain

A domain $D = dom(A_i)$ details a set of atomic (indivisible) values for attribute A_i. As values are atomic, composite and multivalued attributes are not allowed in the model, but can be integrated with additional representations.

For a domain we can describe the logical definitions, data type and format. We can also define a range of possible values. The number of values within the domain $dom(A_i)$ can be represented as $|dom(A_i)| = |D|$.

Example:

- **Name:** A string with no specific format.
- **Acronym:** A string that begins with a capital E. No school can have an acronym larger than four characters.
- **Phone_number:** In Costa Rica, phone numbers are strings of the form $nnnn - nnnn$, with n as an integer value. The first n cannot be 0.
- **Number_students:** A positive integer with a value above 0.

Tuple

A *tuple* represents a collection of related data values represented as $t = \{v_1, v_2, \cdots, v_m\}$.

- t represents the tuple. We write tuples in lower case. Common tuple names include t, u and v.
- v_i is a value within $dom(A_i)$ or a NULL value.
- NULL values can represent unknown values, values that do not apply or values that were not retrieved. However, it has been difficult to represent different types of NULL values in the

relational model. Two NULL values do not mean that both tuples are the same. It is best to avoid NULL values if possible.

- Values do not need to be ordered as long as the relationship between attributes and values is maintained.
- $t[A_i] = t.A_i$ refers to the value v_i of attribute A_i for tuple t. We can represent multiple attributes as a list separated by commas.

Example: For the tuple of our computer science school:

$$t_{ECCI} =< \text{Ciencias de la Computación e Informática}, \text{ECCI}, 2511 - 8000, 888 >$$

2 Keys

Due to integrity constraints, relations have different types of keys for attributes.

Superkey

A *superkey SK* specifies the set of attributes that no two tuples in R can have the same values. Therefore, it specifies *uniqueness*. Every relation must have one superkey and they may have redundant attributes.

Example: We can define as a SK the attributes Name and Acronym. Another SK can be Phone_number as no school can use the same number.

Key

A *key* of R is a superkey that has the additional property that removing any additional attribute from the set of attributes of the superkey will make it no longer a superkey. Thus it satisfies properties for *uniqueness* and *minimalism*. All keys are superkeys, but not all superkeys are keys.

Example: The set of Name and Acronym is not a key. It is not minimal as we could remove either Name or Acronym. However, Phone_number is a key as it is unique and we cannot remove any attribute while maintaining its uniqueness property.

Primary key

A relational schema can have more than one key, with each key being called a *candidate key*. One of the candidate keys is designated as the *primary key* that is used to identify the relation. We represent this by underlining the attributes of the key in the relational schema. The non-primary candidate keys are considered *unique keys* that are not represented in the model.

Though in theory we can select any candidate key as the primary key, it can be recommended to select an integer as they are faster to find while stored. However, the relational model does not really recommend this as it is supposed to represent business rules.

Example: Both Acronym and Phone_number are candidate keys. We can select the Acronym as the primary key, leaving Phone_number as a unique key.

Foreign key

A foreign key in R_1 references a relation R_2 by storing the primary key of R_2 in an attribute in R_1. Thus, there is a relation between the tuples of R_1 and R_2 that have the same key.

Example: SCHOOL could have another attribute that references the DEPARTMENT of each school. Thus, SCHOOL could have a foreign key attribute that references DEPARTMENT.

3 Constraints

For our database, we shall define several *restrictions or constraints* on the values in the database states due to business rules or model restrictions.

Inherent model-based or implicit constraints

Constraints implicitly inherited by the data model. These constraints are defined by the constraints based on the model described in Section 1.

Schema-based or explicit constraints

Constraints explicitly defined in the schema of the data model via DDL (Data Definition Languages).

Domain constraints. For each tuple t, the value v_i of attribute A_i must be atomic and from the domain of A_i.

Key constraints. All tuples within a relationship must be unique, as defined by a Super Key SK. Thus for every pair of tuples t_1 and t_2 within a relation they have to be distinct, described as:

$$t_1[SK] \neq t_2[SK]$$

Not NULL constraints. Attributes define if NULL values are permitted and must be followed.

Entity integrity constraints. No primary key can be NULL, as it is used to identify individual tuples in the relation.

Referential integrity constraints. A foreign key in relation R_1 that references relation R_2 satisfies this constraint if the primary key attributes of both relations have the same domain and R_1 references an existing tuple of R_2.

A database must be in *valid* in every relational state for every relational schema based on the integrity constraints. A database that does not obey all integrity constraints is *not valid.*

Application-based or semantic constraints

Constraints that **cannot** be explicitly expressed in the data model schema. These must be informed either in the application or with SQL (triggers or assertion). As business rules may be complex to implement in the database, it is common that they are specified in the applications. However, constraints should always be defined at the database level if possible. One could also implement these constraints at several levels to ensure the validity of the database.

4 Mapping ER

For most of the following examples for mapping between the ER to the relational model, we shall use Fig. 2.

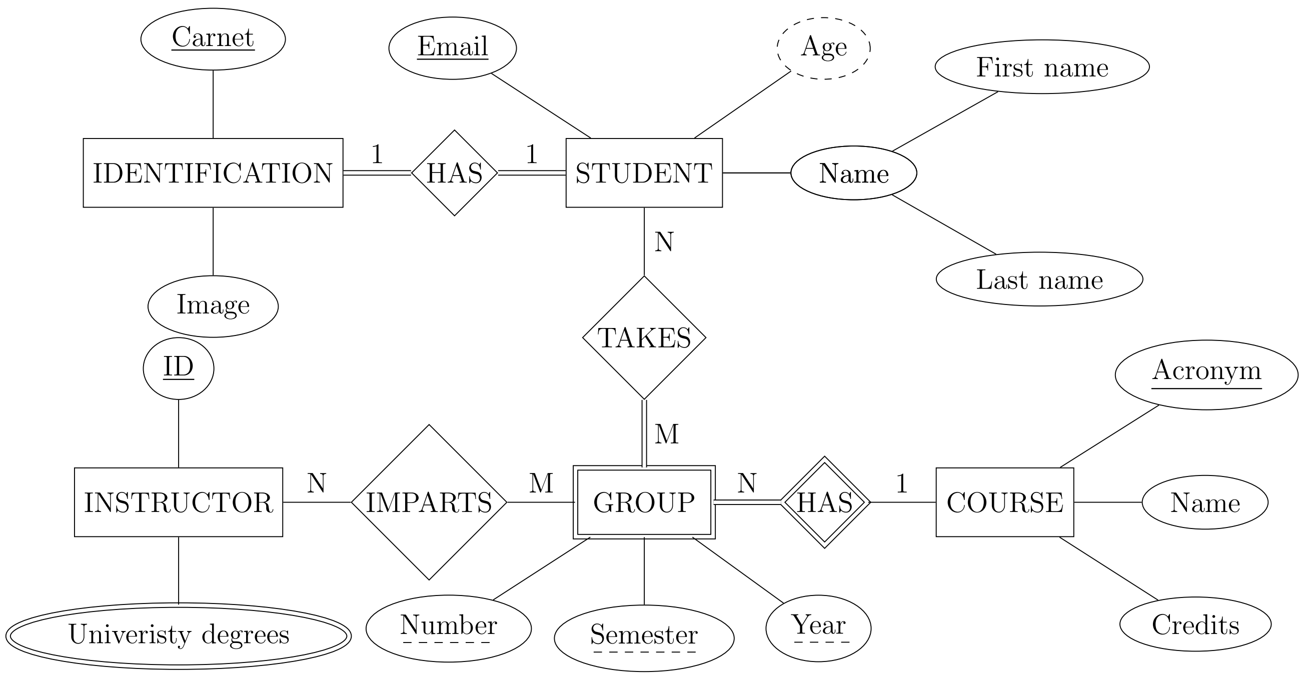

Figure 2: UCR student enrollment ER

The following steps are used to map between both models:

Step 1. Create for every strong entity type E a relation R with all the simple attributes of E as attributes of R. For composite attributes, select the simple components. Choose one of the key attributes in E as the key of R, with a composite key in E using all of the simple attributes as the key of R.

STUDENT

Email	First_name	Last_name

IDENTIFICATION

Carnet	Image

INSTRUCTOR

ID

COURSE

Acronym	Name	Credits

Figure 3: Relations created in step 1

Step 2. Create for every weak entity type W a relation R with all the simple attributes or simple components of composite attributes of W as attributes of R. The key of W is the combination of the primary key of the owner entities and the partial key of the weak entity W.

GROUP

Number	Semester	Year	Acronym

Figure 4: Relations created in step 2

Step 3. For every binary 1:1 relationship R, identify the entity types S and T that participate in the relationship. There are three possibilities:

Foreign key approach. For the relation related to the entity type S, that preferible has total participation to reduce NULLs in the database, add a foreign key of T in S. Furthermore, add the simple attributes or components as attributes in S. This is the most common approach used.

IDENTIFICATION

Carnet	Image	Student_email

Figure 5: Relations updated in step 3 with the foreign key approach

Merged relation. If both participations between entities are total, we can combine the entity types S and T into a single relation.

STUDENT

Email	First_name	Last_name	Carnet	Image

Figure 6: Relations updated in step 3 with the merged relation approach

Cross-reference or relationship relation approach. A third relation U can be created to cross reference the primary keys of S and T as foreign keys. The primary key of U will be a one of the foreign keys, while the other foreign key will be a unique key (as it is a 1:1 relationship). We save the simple attributes or components inside the new relation U.

HAS

Identification_Carnet	Student_Email

Figure 7: Relations updated in step 3 with the relationship relation approach

Step 4. For every binary 1:N relationship R, we apply either the foreign key or the cross-reference approach. The entity S selected to save the foreign key or the primary key, respectively, is the entity at the N-side of the relationship. For example, if GROUP was not weak this would be the entity that would either save the foreign key or have a new table as a primary key.

Step 5. For every binary N:M relationship R, we apply the cross-reference approach. We use both keys of the relations S and T as the primary key.

IMPARTS

ID	Number	Semester	Year	Acronym

TAKES

Email	Number	Semester	Year	Acronym

Figure 8: Relations created in step 5

Step 6. For every multivariate attribute, create a new relation R with attributes of the multivariate attribute and a foreign key to the table S with the multivariate attribute.

INSTRUCTOR_UNIVERSITY_DEGREES

ID	University_degree

Figure 9: Relations created in step 6

Step 7. For every n-ary relationship above $n > 2$, we follow repeat step 5 but with the primary key being all the keys of all participating entities. The participating entities that have a cardinality of 1 are foreign keys, but not part of the primary key.

The final result of the mapping is shown in Fig. 10.

STUDENT

Email	First_name	Last_name	Carnet	Image

INSTRUCTOR

ID

COURSE

Acronym	Name	Credits

GROUP

Number	Semester	Year	Acronym

IMPARTS

ID	Number	Semester	Year	Acronym

TAKES

Email	Number	Semester	Year	Acronym

INSTRUCTOR_UNIVERSITY_DEGREES

ID	University_degree

Figure 10: Relational model after mapping ER

5 Mapping EER

To map the EER, we can extend the previous steps with an additional step.

Step 8 Choose one of the following mapping schemes, based on the type on the constraints of the subclasses:

- **8a** A relation R is created for the superclass as normal. A relation S_i is created for every subclass of R, with foreign and primary keys, the primary key of superclass R. Every relation S_i also has their respective attributes.
- **8b** A relation S_i is created for every subclass of R, with their primary key as the primary key of superclass R. Every relation S_i also has the attributes of superclass R and their respective attributes.

8c A relation R is created for the superclass. The attributes of the subclass S_i of R are saved in R. Furthermore, R saves a type attribute to represent the different subclasses.

8d A relation R is created for the superclass. The attributes of the subclass S_i of R are saved in R. Furthermore, R saves a bool type attribute to represent if the relation belongs to that subclass.

The relationship between the constraints and the type of mapping scheme is shown in Table 11.

	Disjoint	Overlapping
8A	✓	✓
8B	✓	✗
8C	✓	✗
8A	✓	✓

Figure 11: Relationship between the mapping scheme of the subclasses with the constraints

References

[1] R. Elmasri and S. Navathe, *Fundamentals of database systems*, 7th ed. Pearson, 2016, chapters 5 and 9.