Logical design: The relational model

Databases

2018-2019

Jesús Correas – jcorreas@ucm.es

Departamento de Sistemas Informáticos y Computación
Universidad Complutense de Madrid

Bibliography

- Basic Bibliography:
  - R. Elmasri, S.B. Navathe. **Fundamentals of Database Systems** (6th ed). Addison-Wesley, 2010. (in Spanish: **Fundamentos de Sistemas de Bases de Datos** (5a ed). Addison-Wesley, 2007).
    Chapters 5 and 7 (5th ed) or 3 and 9 (6th ed).

Contents

- Introduction. The relational model.
- Basic elements of the relational model:
  - relations, tuples, attributes, integrity constraints.
- Relation schema, relation instance (or state).
- Relational DB schema, relational DB instance (or state).
- Superkeys, keys, foreign keys.
- Relational model constraints.
- Referential integrity constraints.
- Conversion of the ER model to the MR model.

Introduction. The relational model

- The **relational model (RM)** was proposed by E. F. Codd (at IBM) in 1970, **before the ER model** (proposed in 1976 by P. Chen).
- First implementations appeared in the early 80s (IBM, Oracle).
- The relational model contains the formal principles behind all relational DB systems.
- It has been used in most commercial DBMS: IBM DB2, Informix, Oracle, Sybase, SQLServer, MySQL, PostgreSQL, etc.
- **SQL** is the query language used in these DBMS and it is the industrial standard for relational DBs.

Basic elements of the relational model

- A DB in the relational model is a **colection of relations.**
- A **relation** can be seen informally as a **table of values** (or as a **plain text file with a record in each line**, although it is a different thing).
- **It should not be confused with ER model relationships:** Both entity types and relationship types are converted into relation schemas in the relational model.
  - We will see later the conversion process.
- Each **row** (or **tuple**) of the table of values corresponds to an element of the relation: a set of related data.
  - Each tuple represents one element that corresponds to **an entity or relationship element** in the context of the DB.
- Each **column** has a name and corresponds to an specific **attribute** of the tuples in the table, defined on a specific **domain**.

Relation schema, relation state

- A **relation schema** represents the **structure** of an RM relation:

> A **Relation Schema R** is of the form $R(A_1, \ldots, A_n)$, where A_i are the relation **attributes**.

- **Example:**

```
EMPLOYEE(SSN, Name, Position, BirthDate, PhoneNr)
```

- The **degree** or **arity** of a relation is the number of its attributes.
- Each **attribute** A_i is **the name of a role** played by some **domain**. We use $dom(A_i)$ to refer to the domain of attribute A_i.
- Composite or multi-valued attributes **cannot be used in RM relations.**
- **Example:** In the previous relation schema:

$dom(\texttt{SSN})$	is the set of all possible SS numbers,
$dom(\texttt{Position})$	is the set of all available positions in the company, etc.

Relation schema, relation state

- A **relation**, **relation state** or relation **instance** is a particular set of values for a relation schema:

> A **relation** (also named **relation state** or **relation instance**) of a relation schema $R(A_1, \ldots, A_n)$ is a **set of tuples** $\{t_1, \ldots, t_m\}$, where each tuple is an **ordered sequence** of values: $t_j = \langle v_1, \ldots, v_n \rangle$, where v_i is **NULL** or $v_i \in dom(A_i)$.

- It is a relation **in a mathematical sense:** it is a subset of the cartesian product $dom(A_1) \times \cdots \times dom(A_n)$.
- It may have several attributes defined in the same domain, but playing different roles (e.g., private and office phone numbers).
- **Example:** Given the relation schema:

  `EMPLOYEE(SSN, Name, Position, Age, PhoneNr)`

  An state of relation `EMPLOYEE` is for example:

  {⟨`11234, Arthur Smith, Cook, 37, 911234567`⟩,
  ⟨`43210, John Doe, Kitchen helper, 22, 911234000`⟩, `...`}

Relational DB, Relational DB Schema

More definitions:

- A **Relational DB Schema** is composed of a set of relation schemas R and a set of integrity constraints IR.
- A **Relational DB** (also called **Relational DB state** or **instance**) is a set of relations (sets of tuples) of the corresponding relational DB schema.
  - A relational DB state is **valid** (or **correct**) if the integrity constraints are satisfied.
  - It is **invalid** (**incorrect**) otherwise.
- We will see later what are integrity constraints.

Characteristics of relations

- Relations are **sets of tuples** in mathematical sense: **there is no ordering of tuples** and **repeated tuples** are not allowed.
- The sequence of attribute values inside each tuple **is a list of ordered values.**
- Each attribute value is **atomic:** composite values are not allowed in the relational model.
- Multiple values are not allowed for an attribute: multi-valued attributes of the ER model must be converted to extra RM relations.
- There exists a special value **NULL** when a value for an attribute is not available for some reason:
  - It is not applicable to that tuple, or
  - It is unknown or not available for this DB state
- **Meaning of a relation:** tuples in a relation **represent facts** about entities or relationships (of the ER model).
- By default, attribute values cannot be NULL; attributes that accept NULL**must be marked in the RM with an asterisk.**

Superkeys and keys in relations

- Notions of **superkey** and **key** are similar to those of the ER model:

A **superkey** is a subset of attributes in a relation that can **identify** each tuple in the relation.
Represents a **uniqueness constraint:** In other words: no two tuples in any relation state should have the same combination of values for these attributes.

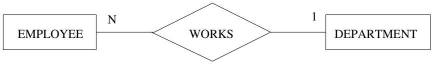
- A **candidate key** is a superkey with a minimal number of attributes.
- One of the candidate keys is selected by the designer to be the **primary key** of the relation.
- Primary key attributes are underlined in the relation schemas of the RM. Example:

  EMPLOYEE(**NIF**, Name, LastName, NSS, BirthDate)

- In addition, the RM includes the notion of **foreign key (FK)**:

Foreign key

- A relational DB can contain several attributes in its relations that **represent the same concept.**
- In particular, some attributes of a relation can be used to **refer** to tuples in another relation.
- Let us see an **example:**
  - We want to represent the department that employees in a company work for:



  - We can model this by means of an attribute in the **`employee`** relation that **refers to** the department for which the employee works:
    `EMPLOYEE(`**`NIF`**`, Name, LastName, SSN, BirthDate, `**`Dept`**`)`
    `DEPARTMENT(`**`DeptId`**`, Description)`
  - In this case, **`Dept`** **forms a foreign key that references** **`DEPARTMENT`**.
  - **`Dept`** and **`DeptId`** represent the same concept and are defined in the same domain.

Foreign key

A subset of attributes FK in a relation R_1 form a **foreign key** that **references** another relation R_2 if the following holds:

1. Attributes in FK have the same domains that the attributes in the **primary key of R_2.**
2. Values of the attributes in FK in any tuple of R_1, either appear in the primary key of a tuple in R_2, or they all are NULL.

- Primary keys and Foreign keys are used for expressing some of the **constraints** that can be used in the RM to enforce that the data in a relational DB state is **correct and consistent.**

Relational model constraints

There are three categories of constraints:

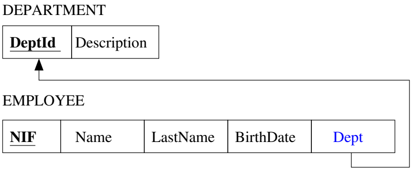
- **Implicit constraints: Inherent data-model constraints.**
  - Relations are sets: they cannot have **duplicate tuples.**
  - Neither composite nor multi-valued attributes are allowed.
- **Explicit constraints: constraints expressible in the relational model.**
  - **Domain constraints:** The value taken by an attribute A_i must be in the set $dom(A_i)$.
  - **Key constraints:** No two distinct tuples **can have the same value** for the attributes of a superkey (and in particular the primary key).
  - **Entity integrity constraints:** values for primary keys attributes **cannot be NULL.**
  - **Referential integrity constraints:** Values of the attributes of a foreign key, either appear in a tuple in the referenced relation, or they all are NULL.
- **Other constraints not expressible in the RM: semantic** constraints (**business rules**): Are checked by application programs and triggers or assertions.

Referential integrity constraints

- All constraints, except referential integrity and semantic constraints, are specified for a single relation.
- **Referential integrity constraints** are specified **between two relations** to preserve the consistency of the tuples in both relations:

> The tuples of a relation R_1 that references another relation R_2 by means of a **foreign key** must refer to **tuples that exist in R_2.**

- These constraints usually correspond to **relationships between entities in the ER model.**
- Referential integrity constraints can be displayed as a diagram with an arrow **from the foreign key to the primary key of the referred relation.**



ER model to relational model mapping

- The procedure for passing from the ER model to the relational model is composed of a series of steps, grouped by the type of ER element to convert:

  Conversion of entity types:

  1. Regular entity types.
  2. Weak entity types.

  Conversion of binary relationship types:

  3. 1:N relationship types.
  4. N:M relationship types.
  5. 1:1 relationship types.

  Conversion of multi-valued attributes:

  6. Multi-valued attributes.

  Conversion of n-ary relationship types:

  7. n-ary relationship types.

  Conversion of Enhanced ER model elements:

  8. Generalizations / specializations.
  9. Aggregations.

ER to RM mapping: Steps 1 and 2. Entity types

Step 1. Regular entity types.

- For each non-weak entity type E, a relation schema R is created with the same name and attributes.
- Composite attributes are included with their atomic components.
- The **primary key** of R is the same as the primary key of E. If a key attribute is composite, the key will be formed by its atomic components.

Step 2. Weak entity types.

- For each weak entity type E (with identifying entity type D), a relation schema R is created with the attributes of E and the attributes in the primary key of D.
- The **primary key** of R is the **combination** of the partial key of E with the primary key of D.

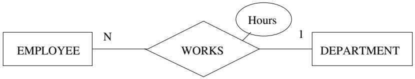
Step 3. 1:N binary relationship types

- We assume that S is a 1:N relationship type between two entity types E y D, where E has cardinality N, and in previous steps relation schemas R_E and R_D have been created for that entity types.
- The mapping is as follows:
  - The attributes of the primary key of R_D (side 1) **are added** to R_E (side N), as well as the attributes of the relationship S.
  - A **foreign key** is added to R_E, with the attributes of the primary key of R_D just added.
- If the participation of E is **partial,** the attributes added to R_E **must admit null values.**[1]
- If the participation of D is **total, that information is lost in the RM** (we should add semantic constraints).

[1] Remember that nullable attributes must be marked with an asterisk.

Step 3. 1:N binary relationship types

- **Example:** (some attributes are not shown in figure)



- The relation schemas are generated as follows:

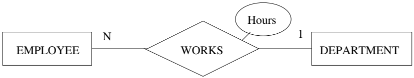
  `DEPARTMENT(`<u>`DeptId`</u>`, Description)`

  `EMPLOYEE(`<u>`NIF`</u>`, Name, LastName, BirthDate,` **`DeptId*, Hours*`**`)`

  - `DeptId` is a **foreign key** referencing `DEPARTMENT`
- **`DeptId` is added to `EMPLOYEE`** to represent the relationship: each employee references the department Id where she works, and in a department can work several employees.
- The **participation** of `EMPLOYEE` in `WORKS` is **partial:** attribute `DeptId` should allow null values (there is an asterisk in `DeptId`).

Step 3. 1:N binary relationship types

- **If the number of null values is expected to be very large,** it may be more convenient to **create a fresh relation schema R_S with:**
  - attributes of the primary keys of R_E and R_D, and the attributes in S;
  - **Primary key** composed of **the attributes in the primary key of R_E;**
  - Appropriate **foreign keys** referring to R_E **and** R_D**.**
- In this case attributes in R_S **should not allow null values** (partial participation is expressed by the absence of tuples in relation R_S.)
- **Example:**



DEPARTMENT(<u>DeptId</u>, Description)

EMPLOYEE(<u>NIF</u>, Name, LastName, BirthDate)

WORKS(**<u>NIF</u>,DeptId,Hours**)

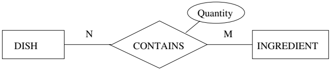
- NIF is a **foreign key** referencing EMPLOYEE
- DeptId is a **foreign key** referencing DEPARTMENT

Step 4. $N{:}M$ binary relationship types

- We assume an $N{:}M$ relationship S between two entity types E y D with relation schemas R_E y R_D in the RM, respectively.
- The mapping is as follows:
  - **A fresh relation schema R_S is created** with the attributes of the primary keys of R_E y R_D, as well as the attributes in S.
  - The **primary key** of R_S is composed of the attributes coming from the primary keys **of both entities** (R_E y R_D).
  - Two **foreign keys** are added to R_S, from each of the sets of attributes copied from the primary keys of R_E and R_D, to the relation schemas R_E and R_D, respectively.
- **The information about total participation of both entity types is lost in the RM** (we should add semantic constraints).

Step 4. *N*:*M* binary relationship types

Example: (some attributes are not shown in figure)



- The relation schemas are generated as follows:

  `DISH(`<u>`DishId`</u>`, Description, price)`

  `INGREDIENT(`<u>`IngrId`</u>`, Description)`

  `CONTAINS(`<u>**`DishId, IngrId`**</u>`, `**`Quantity`**`)`

  - `DishId` is a **foreign key** referencing `DISH`
  - `IngrId` is a **foreign key** referencing `INGREDIENT`
- **`CONTAINS` is added** to represent the relationship: a pair (`DishId`, `IngrId`) in schema `CONTAINS` represents that ingredient `IngrId` is used for preparing the dish `DishId`.
- A dish may contain several ingredients, and vice versa.
- However, **total participation cannot be represented on any side** by means of foreign keys: a semantic constraint is required.

Step 5. 1:1 binary relationship types

- We assume a 1:1 relationship S between two entity types E y D with corresponding relation schemas R_E y R_D in the RM.
- There may be three cases depending on the participation constraints:
  1. **Both entity types have total participation in S:**
     - ⋆ We can **merge both relation schemas in one,** as they refer to different information of the same thing.
     - ⋆ We keep **only one of the primary keys,** the one that we consider most appropriate for the rest of the DB.
     - ⋆ We could apply case 2 below, but we **lose information regarding participation constraints.**
  2. **One entity type (e.g., R_E) has total participation in S:**
     - ⋆ The attributes of the primary key of R_D and relationship attributes are added to R_E
     - ⋆ A **foreign key** is added to R_E referencing R_D.
     - ⋆ Observe that **the information on the cardinality constraint of R_E is lost** in the mapping (as it is just like a 1:N relationship).

Step 5. 1:1 binary relationship types

3. **Both entity types have partial participation in S:**
   - Choose one of the entities (e.g., R_E) and apply case 2.
   - The attributes added to R_E **must admit null values.**
   - Observe that cardinality information is also lost in this case.

- **Example:**

  - The relation schemas are generated as follows:

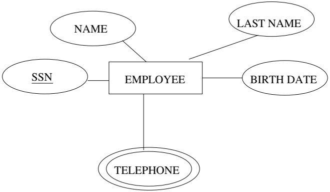
    `DEPARTMENT(`<u>`DeptId`</u>`, Description, `**`NIFManager*`**`)`

    `EMPLOYEE(`<u>`NIF`</u>`, Name, LastName, BirthDate)`

    - `NIFManager` is a **foreign key** referencing `EMPLOYEE`
  - **`NIFManager` is added to `DEPARTMENT`** to represent the relationship: each department references the employee NIF of its manager.
  - The **participation** of `DEPARTMENT` in `MANAGES` can be represented as **partial** or **total,** allowing or not null values to `NIFManager`.
  - **Cardinality 1 of `DEPARTMENT` cannot be represented** (a semantic constraint would be required).

Step 6. Multi-valued attributes

- For each multi-valued attribute M in an entity type E, a fresh relation schema R is added with the attributes in the primary key of E, and an attribute M (single-valued).
- The **primary key** of R is composed of all its attributes.
- Finally, a **foreign key** is added, referencing the primary key of E.
- **Example:**



`EMPLOYEE(SSN,Name,LastName,BirthDate)`

`TELEPHONES(`**`SSN,telephone`**`)`

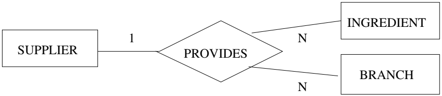
\- `SSN` is a **foreign key** referencing `EMPLOYEE`.

Step 7. Ternary relationship types

- We assume an relationship S between three entity types with at most one entity type with cardinality 1. The mapping is as follows:
  - A fresh relation schema R_S is created with the attributes of the primary keys of participating entity types, as well as the attributes in S.
  - The **primary key** of R_S is composed of the attributes coming from the primary keys **of the three entities.**
  - Three **foreign keys** are added to R_S, from each of the sets of attributes copied from the primary keys of each entity type, to the relation schemas of the participating entity types.
- If the cardinality of one entity type is 1, the primary key of R_S should not include the attributes of primary key of that entity type.

Step 7. Ternary relationship types

Example:



- The relation schemas are generated as follows:

  SUPPLIER(<u>SupId</u>, Name)

  INGREDIENT(<u>IngrId</u>, Description)

  BRANCH(<u>BranchId</u>, Address)

  PROVIDES(<u>**BranchId, IngrId**</u>, **SupId**)

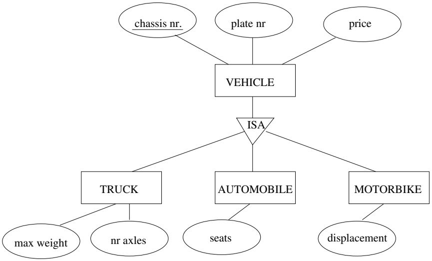
  - BranchId is a **foreign key** referencing BRANCH
  - IngrId is a **foreign key** referencing INGREDIENT
  - SupId is a **foreign key** referencing SUPPLIER
- **PROVIDES is added** to represent the relationship: a tuple (BranchId, IngrId, SupId) in schema PROVIDES represents that Ingredient IngrId is provided by Supplier SupId to Branch BranchId.
- **Cardinality 1** of SUPPLIER is handled **excluding it** from the **primary key.** Can total participations be represented?

Step 8. Generalization/specialization

- We assume an ISA relationship between a superclass entity type P and a subclass entity type S.
- The mapping is as follows:
  - The relation schema of the superclass entity type, R_P, is created as usual, according to Step 1.
  - For the subclass entity type S a relation schema R_S is created, including the attributes in S **and the attributes of the primary key of the superclass R_P.**
  - The **primary key** of R_S is composed of the attributes coming from the primary key of R_P.
  - In addition, a **foreign key** is added to R_S with the attributes of its primary key, referencing the primary key of R_P.

Step 8. Generalization/specialization

Example:



- The relation schemas are generated as follows:

  VEHICLE(<u>Chassis</u>, Plate, Price)

  TRUCK(**<u>Chassis</u>**, MaxWgt, NrAxles)

  AUTOMOBILE(**<u>Chassis</u>**, Seats)

  MOTORBIKE(**<u>Chassis</u>**, Displacement)

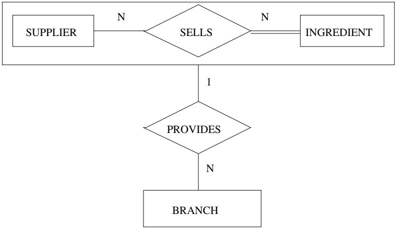
  - Three **foreign keys** of Chassis in TRUCK, AUTOMOBILE, and MOTORBIKE referencing VEHICLE.

Step 9. Aggregations

- Entity and relationship types inside and outside an aggregation are mapped in the RM using previous steps.
- We assume a relationship type S that relates an aggregation A with an entity type E.
- To map S, we apply one of the steps 3 to 7 above, but using the relationship attributes and:
  - The attributes in the primary key of entity type E.
  - The attributes in the primary key of the relation schema that represents the main relationship type in aggregation A.
- Finally, we add **foreign keys** for the sets of attributes that form the primary keys referencing the relation schemas from which the attributes come from.

Step 9. Aggregations

Example:



- The relation schemas are generated as follows:

  `SUPPLIER(`<u>`SupId`</u>`, Name)`

  `INGREDIENT(`<u>`IngId`</u>`, Description, Price)`

  `SELLS(`<u>`SupId, IngId`</u>`)`

  - `SupId` is a **foreign key** referencing `SUPPLIER`
  - `IngId` is a **foreign key** referencing `INGREDIENT`

  `BRANCH(`<u>`BranchId`</u>`, Address,` **`SupId`**`,` **`IngId`**`)`

  - {`SupId, IngId`} is a single **foreign key** referencing `SELLS`.