# Challenge 1: Database Design and Implementation

Irene Escofet (U213899), Isabel Expósito (U214967) and Tània Pazos (U185115)

Pompeu Fabra University

24303: Databases

November 1,  2024

**Conceptual Model**

The first step in the database implementation process was designing the Conceptual Model, particularly the Entity-Relation diagram (Figure 1), to address all the requirements specified in the NBA scenario. This section explores the assumptions made regarding the database requirements, with a special emphasis on cardinality assumptions.

Firstly, two weak entities were identified in the NBA scenario: SEAT and ZONE. A zone cannot be uniquely identified by its code alone; the stadium name is also required. Similarly, a seat cannot be uniquely identified by its color and number; we need both the zone code and the stadium name to determine the exact seat.

Regarding generalization, we considered grouping COACH, PLAYER and STAFF into a superclass PERSON. However, while PLAYER and COACH share several attributes (national id, name, surname, nationality, gender and age), STAFF has only one attribute in common (national id). For this reason, we decided against creating the superclass, despite being aware of the attribute duplication between PLAYER and COACH.

Although not explicitly stated in the requirements, we considered it beneficial to add a primary key attribute, game_id, to uniquely identify GAME, as no attribute or combination of attributes specified in the requirements could ensure uniqueness.

Thirdly, the cardinality of PLAYER and FRANCHISE was set to N:M to allow a player to change franchises over time. Similarly, the relationships HEAD_COACH-NATIONAL_TEAM and HEAD_COACH-FRANCHISE were given many-to-many cardinality, allowing a head coach to work for different franchises and national teams over the years. Furthermore, the design assumed that the figure of a head coach was mandatory for any type of team (franchise or national team). Nevertheless, the requirements specified that a certain assistant coach will always work for the same franchise. Thus, the cardinality between ASSISTANT_COACH and FRANCHISE was set to N:1, i.e., a franchise is trained by multiple assistant coaches but one assistant coach will always train only one franchise. It is also worth noting the optional recursive relationship of ASSISTANT_COACH to represent that “some assistant coaches will be the boss of other assistant coaches”.
We considered the relationship between PLAYER and NATIONAL_TEAM as optional, since not all players are selected for their national teams. Although a player can only belong to one national team (the one representing their country), they may participate across multiple years, giving this relationship a P:M cardinality. Regarding the ternary relationship

DRAFT_LIST, PLAYER and FRANCHISE, it is true that each player only enters the draft list once (hence the cardinality between DRAFT_LIST and PLAYER would be 1:M), but it is a ternary relationship and FRANCHISE does appear in the draft list more than once. As a result, the relationship is N:M:P. In our implementation, we assumed that each franchise would consistently have the same person representing its mascot, resulting in a 1:1 cardinality between MASCOT and FRANCHISE. Similarly, we assumed that each staff member would always work for the same franchise and that each security guard would be assigned to a single stadium, establishing a one-to-many cardinality for both relationships. Since we consider the local and visiting teams separately for each match, the cardinality between GAME and STADIUM is M:1, as each game is always played in the local team's stadium.

Lastly, we would like to highlight that additional relationships could be established among entities in the diagram, such as between STADIUM and FRANCHISE. However, we have only represented the relationships explicitly stated in the requirements. Additionally, since we already have the STADIUM-GAME and GAME-FRANCHISE relationships, we believe that adding a STADIUM-FRANCHISE relationship would introduce redundancy to the diagram.

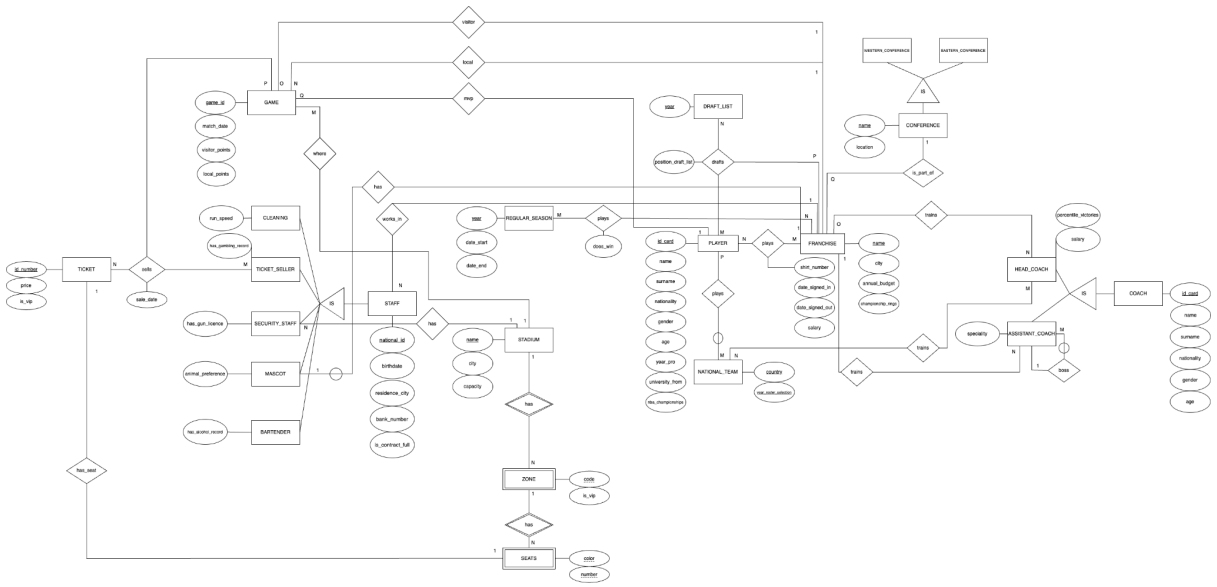


Figure 1: Conceptual Model (Entity-Relation) design diagram.

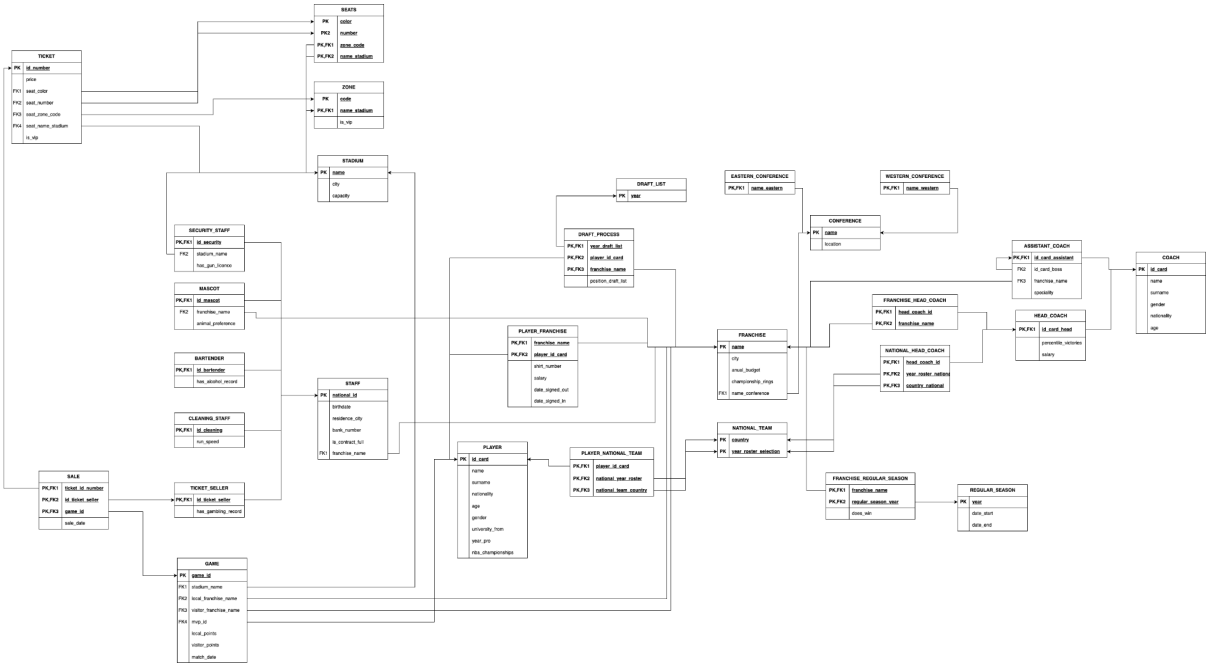


Figure 2: Relational Model diagram.

**Relational Model**

Once we had completed the Conceptual Model, in order to do the Relational Model we had to take into account the following: an entity, an M:N relationship and a n-ary relationship are transformed into new relational tables. In the case of 1:1 or 1:N relationships we used a foreign key in one of the entities referencing the other entity. Figure 2 shows the Relational Model diagram for our NBA database.

For example, in the case of the relationship between FRANCHISE and MASCOT, which we have considered to have cardinality 1:1, we added a FK in the relational table of MASCOT called ‘franchise_name’, which refers to the primary key of the entity franchise (which is the name). Similarly, in the case of the relationship between SECURITY_STAFF and STADIUM, which we have considered it to be N:1, we added a FK in the table on the N side (SECURITY_STAFF) called ‘stadium_name’ that referenced the primary key of STADIUM (which is also the name). This is a similar case as in the recursive relationship BOSS, where in the relational model we have added a FK in the relation ASSISTANT_COACH called ‘id_card_boss’ that references the PK of itself (since it is a recursive relationship).

Another important aspect to comment on is how we transformed the weak entities SEATS and ZONE into the relational model. In this case, the foreign key of the identifier relationship is part of the primary key. Thus, the relation SEATS contains as primary keys (apart from ‘color’ and ‘number’) the primary keys of ZONE, that is, ‘zone_code’ and ‘name_stadium’ and these are also foreign keys. Similarly, ZONE contains as primary and foreign key, the primary key coming from the strong entity STADIUM (stadium_name).

We would also like to point out that the foreign key ‘franchise_name’ in the relation STAFF (as a result of the N:1 relationship between STAFF and FRANCHISE) will already indicate the name of the franchise for which the staff is working NOWADAYS (as the statement asks to specify).

**Creating the Database**

Now that we have done both the conceptual and relational model we can focus on creating the database itself with SQL coding. The base for this code is mainly the relational model as it helps us to know which are the tables that we have to create. To do so, we took into consideration the order of the tables implementation, as it is important that the already created tables do not need any instance of a key from a non created table. We had this into account not only when creating the tuples themselves, but also when dropping them first.

Moreover, we haven't specified any update policies as we do agree with the restriction policy which is the default one (it does not allow any modification that violates the integrity principle).

We have used commands such as:

- DEFAULT (when using a boolean type) which we setted as 0 or 1 depending on what was more appropriate for each of the situations. For example, in BARTENDER we added ‘has_alcohol_record BIT(1) DEFAULT 0’.
- AUTO_INCREMENT (when we want every element created to increment +1). For example, in TICKET, we added ‘ id_number INT UNSIGNED NOT NULL PRIMARY KEY AUTO_INCREMENT ‘ assuming each ticket has a consecutive id in order to number all the tickets sold.
- UNSIGNED for example, in TICKET, we added ‘seat_number INT UNSIGNED’ as they can not be negative integers describing the seat number. Highlight that in the anual_budget ( FRANCHISE ) we did not include this because we thought that, if there are existing debts, it might be a negative number and we wanted to allow that.
- NOT NULL was established in all PK definitions to ensure the values are different from null in the created tables.
- SMALLINT was used for integers that we know would not reach a huge number, for example, in GAME, the points of each team.

We also should highlight that the most used data types were :

- VARCHAR(100) as most things could easily reach this amount of characters.
- INT or UNSIGNED INT as mentioned before
- DECIMAL( , ) where the first input is the total number of ciphers including decimals and the second is the allowed number of decimals.