

Campus Santa Fe

***Explicación del UML2***

***Construcción de software y tomas de decisiones***

***Imanol Santisteban Piñeirua A01783637 Ricardo Alfredo Calvo Perez A01028889 Francisco José Urquizo Schnaas A01028786***

***Salvador Gilberto Vaquero Becerra A01027920***

**First Version of UML (Creation date: 29/02/24):**

For the Cards entity it contains all the necessary information for it and it is connected one to many with Hand to know all of the cards that are available in a turn, Played to know when a specific card has been used and Inventory.

Which brings us to Player, detecting many things such as its username and password allowing players to access the game, it also includes the npc player. Player is connected to different things to keep track of its progress, such as the connection with Level to know what levels has the player beaten. It is also connected to Inventory, Reward that lets us know the cards won by the player**,** Hand to know what cards were available to a different player at any given time and similarly to Turn that saves many of the statistics in a certain turn of the match.

Level is connected to turn to know the turns made in a Level. This works similar to match to match where the turns of the match are saved through the connection with Turns.

Deck is connected with hands since many decks can be connected with one Hand and depends on which deck the player has to decide its hand. It is also connected with Match to know what decks were played on how many matches.

The Inventory is an intermediate entity between player, cars, deck and reward. This way the UML fills the necessary requirements.

There is also Turns that it also serves as an intermediary between a lot of things regarding what happens during a turn in a match and connects to Played which are all of the cards played within a turn.

**Second Version of UML (Creation date: 20/03/24):**

**Changes made to the Entity Relationship Diagram:**

To enhance the clarity of the video game database, it was resolved that specifying data types for each table column would be advantageous. Accordingly, these specifications, alongside integrity constraints, have been documented in the UML diagrams to standardize data entry and enforce consistency.

Moreover, in the actual MySQL table implementation, foreign keys have been explicitly defined using the `CONSTRAINT` keyword. This allows each foreign key to be given a unique name, thereby facilitating more comprehensible error messages should any issues occur. It's worth noting that these foreign keys come with stringent rules: `ON DELETE RESTRICT` and `ON UPDATE CASCADE`. Such constraints are crucial as they prevent the deletion of records from a parent table if there are dependent records in the child tables—this is to avoid a cascade of errors. Conversely, updates to the parent table will propagate changes to the child tables, maintaining referential integrity throughout the database.

**Current Normal Form:**

After having analyzed the videogame database, the conclusion that it is in the First Normal Form was reached, since it abides by the following key traits of the aforementioned normal form:

* **Atomicity**: Ensure that each attribute (column) contains atomic values, meaning that the values are indivisible. For example, instead of having a single attribute (column) to store the full name of a person, it's better to have two separate attributes for first name and last name. This rule helps in eliminating groups of values from a single column.
* **No Repeating Groups**: Each table must be structured such that it contains no repeating groups of attributes. In practical terms, this means avoiding the practice of having multiple columns for the same kind of data that could be indefinitely extended, such as phone\_number\_1, phone\_number\_2, etc. Instead, related data that could occur multiple times should be moved to a separate table with a relationship established between the two tables.
* **Unique Key**: Each table should have a primary key. The primary key is a column (or a set of columns) that uniquely identifies each row in the table. This is important for ensuring that each record can be uniquely identified, which is a prerequisite for further normalization.

Each consideration listed was pivotal in constructing the database architecture. Admittedly, further subdivision of data into additional tables might enhance atomicity. However, this approach could inadvertently heighten the risk of errors. Our strategy was to achieve atomicity at a fundamental level, establishing tables only where they served this purpose. It's possible to delve into the details and segment the columns into more tables, yet such granularity is not advisable. The goal was to avoid superfluous tables and maintain atomicity within each column, which we believe our current design achieves.