

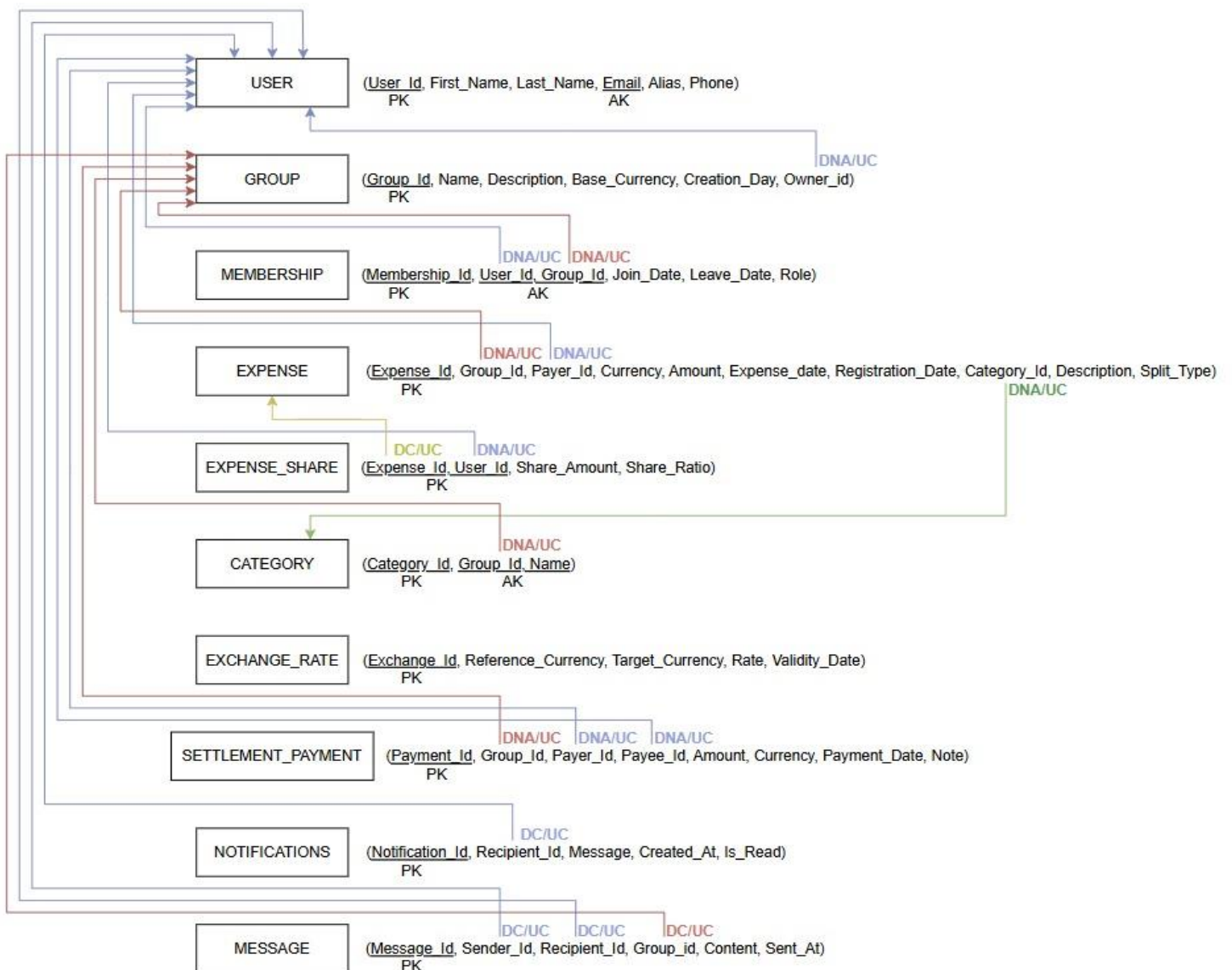
PROJECT 1. PART 1.

RELATIONAL DESIGN

Víctor Ortega (100550963) and Héctor Rodríguez (100550972)

Group 97

1. RELATIONAL DIAGRAM



Reasoning for Our Selection of Delete Cascade (DC) or Delete Restrict (DNA)

For the Group entity, the foreign key Group.owner_id uses delete restrict (DNA). The reason is that a group cannot be left without at least one active owner, as the project requires. If the owner user was deleted, the group would lose its main reference, so we block this deletion. So, if the owner of a group wants to delete their use, he must transfer the ownership of the group first.

For the Membership entity and the Membership.user_id key, we decided to use delete restrict (DNA). If a user was deleted, the records about when they joined or left a group would lose their meaning. To solve this, we added a new attribute to the User entity called is_deleted, a Boolean that lets us “soft delete” a user. This way, the user can no longer log in or appear to others, but their information is kept so we don’t lose the history. We also used delete restrict (DNA) for the Membership.group_id key, because a group shouldn’t be deleted if it still has members connected to it. This makes sure we can always track who was part of the group. The project also says that a member can’t leave if their balance is not zero and that every group must always have at least one active owner. So, with this decision, a group with debts can’t just be removed. Only when all balances are settled can the application first remove the memberships and then delete the group.

For the Expense entity, we used delete restrict (DNA) to all its foreign keys, that are Expense.group_id, Expense.payer_id and Expense.category_id, all of them following the same reasoning, if there exists an expense linked to a group, an user or a category, they cannot be deleted. Otherwise, the expense record would lose its context and the history of who paid, in which group, and under which category would be broken.

For the Expense_Share entity, delete cascade (DC) is applied for the Expense_Share.expense_id key, because how an expense is shared has no meaning by itself, so if an expense is deleted, its shares must also be removed. However, for the Expense_Share.user_id key, we decided to use delete restrict (DNA), because if a user has been involved in the payment of an expense, that user cannot be deleted without breaking the history of who participated.

In the Category entity, we apply delete restrict (DNA) to the Category.group_id key because if a group has expenses for a particular category, it can’t be removed and neither the category that is being used.

In the Settlement_Payment entity, we used delete restrict (DNA) for all its foreign keys: Settlement_Payment.group_id, Settlement_Payment.payer_id and

Settlement_Payment.payee_id. The idea is the same in all cases: if there is a payment registered between a group and two users, none of them can be deleted. Otherwise, we would lose the history of who paid, to whom, and in which group, and that information is essential because balances are calculated from these payments.

The Notification entity employs delete cascade (DC) for the Notification.recipient_id key. This is because the notifications are auxiliary. If the user disappears, their notifications are useless, so they are safely deleted with cascade.

For the Message entity, we applied delete cascade (DC) to all its foreign keys. For Message.sender_id, if the user is deleted, their messages are deleted too. For Message.recipient_id, the logic is the same, because a private message has no sense without the recipient. Finally, for Message.group_id, if a group is deleted, all the messages inside that group must also be removed.

In summary, we applied delete cascade (DC) to all auxiliary data, like notifications or messages while delete restrict (DNA) was used for historical and financial records, like memberships, expenses or payments. In summary, we ensured that the use of data remained practical, while always preserving its value and meaning within the project.

2. Semantic assumptions

There are many doubts or uncertainties that have come to our minds while reading the project instructions. This has led us to create assumptions, which we have classified as semantic assumptions because they come from the statement and do not affect the creation of the schema/diagram.

Firstly, we have assumed that every time an attribute stores a date, the different dates always belong to the same time zone. If this didn't happen, the date attributes and keys (such as Join_date, Leave_date, Validity_date, etc.) would confuse instead of giving information. This is not our only assumption related to dates, as we have noticed that the leaving date of a group should always be later than the joining one, otherwise it would be impossible. This would only be possible when rejoining a group, but if there is just one Join_date, it must be before the Leave_date.

Also, we have assumed that when talking about money and currency, there exist unique identifiers for each type of currency, such as 'EUR' for euros, 'USD' for US dollars, etc. This is crucial, for example, when referring to the attribute Expense.Currency. Furthermore, we have had to assume that when modifying the base currency of a group, the value of every key will automatically update to the new currency, computing the needed operations to translate between currencies. For example, right now 1 USD corresponds to 0.85 EUR, so that if the base currency is updated from USD to EUR the values of each key will be multiplied by 0.85.

We have had to make assumptions about the notifications as well, as it is not specified in the instructions whether this entity is destined exclusively to payments or there should be notifications for other information. Using our experience with similar systems or applications, we have assumed that the notifications exist in general and not only for payments. We have also assumed that when a user opens a notification, the user reads the information. This does not always happen in real life situations, where the user may not read the message or may not pay attention while reading.

Another assumption we have made is that two users who do not belong to a same group cannot share an expense. This may seem obvious, considering that expenses belong to groups. However, it is not specified anywhere in the project instructions. Also, for the settlement payment entity, both the payer and the payee must belong to the group in the Group_Id attribute. Otherwise, if both or one of them didn't belong to the group, the payment would be impossible.

There is an assumption that could be classified into the schema assumptions, as it is related to the domain and the format of some attributes. However, this has not affected the creation and design of the schema, and that is why we have agreed to classify it as a semantic statement assumption. We have assumed that the values of specific attributes follow a specific format (email – “__@__.__”, id – just letters and numbers, first name and last name – only letters, etc.).

The last semantic assumption we have made is related to messages. The instructions only mention communication between group members, but it is not specified if this is just for members of a same group or there can also exist messages between users that do not share a group. We have assumed that messages can only exist between member of a same group.

3. Schema assumptions

We have also encountered some doubts or uncertainties while translating the information from the instructions to the schema. We have had to create more assumptions to design the diagram. These assumptions may have changed the result of the diagram, as they affect entities, attributes, relationships, etc.

Firstly, the most obvious one: we have assumed that net balance is not an entity, as the instructions claim that “this balance is not stored explicitly in the database; it is derived dynamically from the recorded expenses and settlement payments.”

Then, for the User entity, we have assumed that the first name and the last name are not unique, two people can share first name, last name or both. That is why these two attributes are not alternate. Also, we have assumed that there is no restriction implied to the alias, which means that the alias can be repeated, and it is not unique (when some value is unique, it is specified in the instructions, so if it is not specified, we assume it is not unique). Furthermore, at first, we assumed that the phone number should be an alternate key, as it is unique for each person. However, the fact that the number is an optional key means that its value is NULL by default, and when two phone numbers are NULL, they are not unique, so the attribute User.Phone is not AK.

For the Group entity we have assumed that two different groups can share the same name, and that is why the primary key is the Group_Id attribute, because it contains unique values. We have also assumed that each group can only have one owner. There cannot exist a group with more than one owner.

When creating the Membership entity, we assumed that a user cannot have two different roles. Each user has just one role: owner, admin or member.

About the Expense entity, we had to assume that there are no restrictions between the registration date and the expense date, so that a user can register a date before

paying, the same day of the payment or after paying. This way the user has more freedom to add an expense whenever he/she wants to.

Additionally, for the Message entity, we have made two assumptions. Firstly, we assume that each message is sent by one user, each message has exclusively one sender. Secondly, we assume that when the message is sent to one user (individual message), the group_id attribute is NULL. This also happens in the opposite way. When the message is sent to more than one user (group message), the recipient_id attribute is NULL.

Finally, the identifier for the Expense, Settlement Payment and Notification entities is unique and therefore a candidate key. It is a great option for a primary key. The fact that an identifier is unique may seem obvious. However, the instructions do not specify this.