Faculdade de Engenharia da Universidade do Porto

Conceptual Modeling

Building a database for a Fashion Event

(Second Submission)

Database Project 2024/25 - L.EIC

Group 707

Students & Authors

Leonor Matias up202303772@up.pt

Matilde Nogueira de Sousa up202305502@up.pt

Gabriela de Mattos Barboza da Silva up202304064@up.pt

20 de Outubro de 2024

Summary

This project focuses on designing and refining a database system for a high-fashion runway event through an iterative process, incorporating both manual efforts and generative AI tools to ensure correctness and efficiency. It covers the evolution from conceptual modeling to the final database schema and implementation, emphasizing efficiency, adaptability, and adherence to SQLite requirements.

Overall, this project proposes a normalized data schema, fully implemented in SQLite, for a high-fashion event, while demonstrating the complementary role of generative AI in its creation.

Keywords: UML Diagram, Conceptual Modeling, Relational Schema, Functional Dependencies, SQLite, Constraints, Referential Integrity

Index

I. Introduction

This report is the second part of a Database creation project for a fashion event mimicking Fashion Week's, and it serves as the continuation of the work previously presented in the first submission. This time, we present a refined and final UML model, taking in consideration the feedback received, and do some further analysis and improvements on the relational schema and functional dependencies, that ultimately lead to the implementation in SQLite of the database.

II. Refined Conceptual Model

After submitting our initial database design , we received some suggestions on how we could improve our conceptual model to enhance its efficiency, readability, and adaptability for implementation in SQLite.

A. Person Class

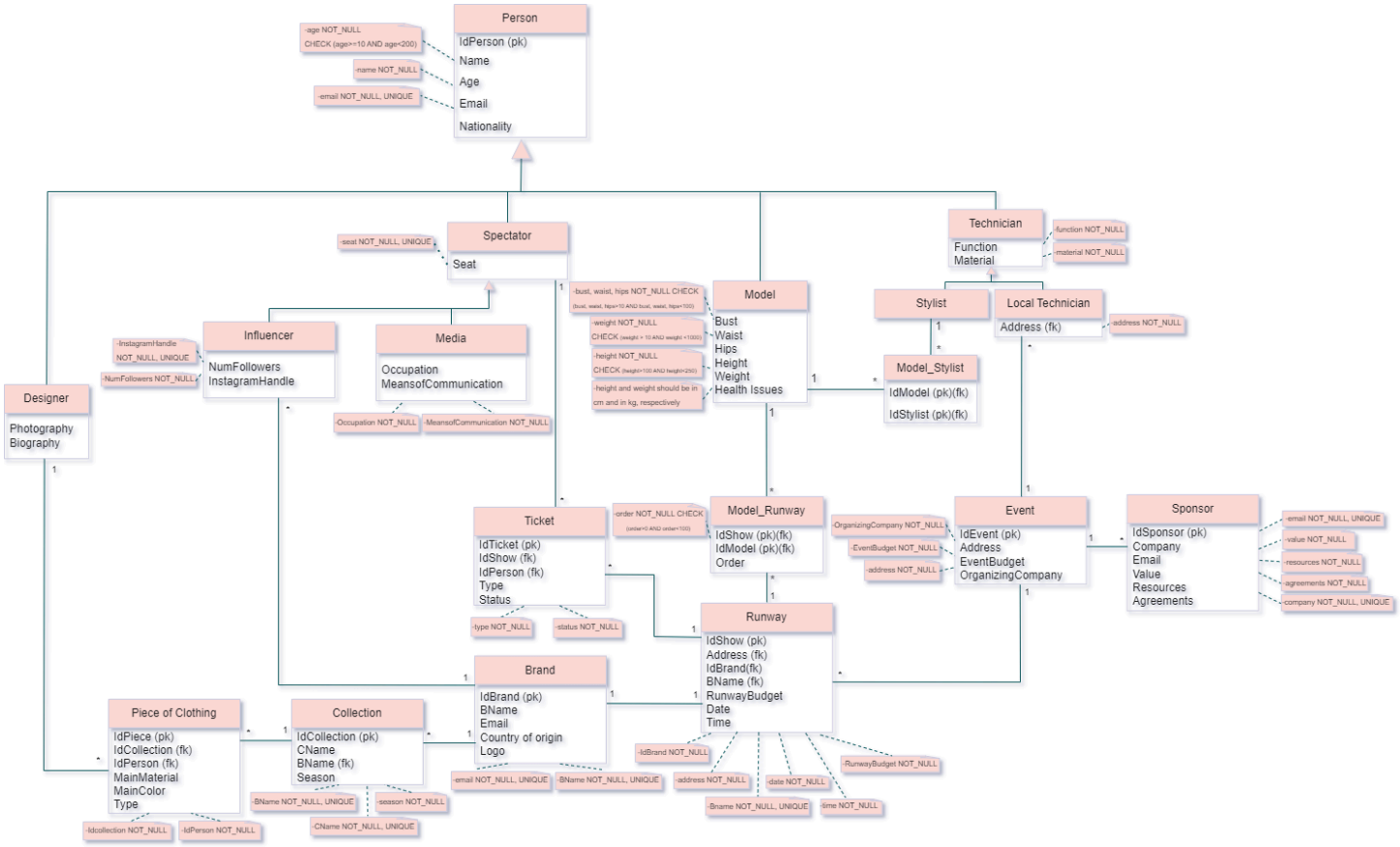
A significant change was the introduction of a new class *Person,* which now serves as the parent class for all existing person-related classes, such as *Model*, *Designer, Spectator,* etc. The *Person*'s attributes include Name, Age, E-mail and Nationality. Additionally, after taking in consideration AI suggestions for the improvement of the Relational Schema (Section IV.) idPerson was added as its primary key, as relying solely on the Name might not be sufficient to uniquely identify a person, especially in a range that includes from models and designers to spectators.

B. Other improvements

The addition of a new class that unifies all persons had a big impact on how individual interactions with the event, and with each other, are portrayed in the conceptual model. So much so that we decided to create two more classes (other than *Person*), *Technician*, and *Model_Stylist*, to better accommodate these changes.

Technician is a subclass of Person, with two further (already existing) subclasses: *Stylist,* which includes makeup artists, hair stylists, and similar roles, and *Local Technician,* which encompasses venue workers such as light technicians and cleaners. Since both groups contribute to the orderly and smooth operation of the event by applying the technical aspects of their craft, it made sense to group them under a *Technician* class.

Model_Stylist class arises from the necessity of making the association between *Model* and *Stylist* clearer. Each model works with at least one styling technician. However, since both the *Model* and the *Stylist* are generalized from the *Person* entity and share the same idPerson attribute, the relationship between them might cause a "self-referencing" issue due to the shared primary key (idPerson). This happens because it's not visually or logically clear which entity the idPerson is referring to in such a relationship. *Model_Stylist* solves this by explicitly relating models to stylists. Attributes include idModel (Foreign Key referencing Model(idPerson)) and idStylist (Foreign Key referencing Stylist (idPerson)).



UML Diagram 1 - Note that the diagram is up-to-date with enhancements made in further sections.

III. Constraints

In the initial submission of the report, a constraints section was overlooked, despite their critical role in transitioning to the implementation of the database in SQLite. To address this oversight, we decided to detail the constraints present in the refined version of the UML in this section.

Person

1. IdPerson primary key - there cannot be more than one person with the same ID, each ID is unique.
2. name NOT NULL, age NOT NULL CHECK (age >= 10 and age <= 200) - each person has a name and their age must also be known and real.
3. email NOT NULL, UNIQUE - email is our chosen way of communication, therefore it must be known and unique for each person.

Spectator

1. seat NOT NULL, UNIQUE - the seat must be known for organization purposes and unique for each spectator.

Model

1. bust, waist, hips NOT NULL CHECK (bust,waist, hips > 10 AND bust,waits, hips<100) - these measurements are crucial to ensure the models fit the clothes they are meant to wear, therefore cannot be null. Additionally, they have to be real measurements, hence the range values.
2. weight NOT NULL CHECK (weight > 10 AND height <1000), also a characteristic of interest for the hiring entity thus must not be null, and a real value. Should be given in kilograms.
3. height NOT NULL CHECK (height>100 AND height<250), same as weight. Should be given in centimeters.

Technician

1. function NOT NULL - the function specifies the type of technician (e.g. if function is hairdresser, then we are referring to a styling technician), consequently cannot be null.
2. material NOT NULL - the material cannot be null, given that it is crucial information for the event´s organization (e.g. a local technician might need a ladder, which would be provided by the event´s responsible entity).

Local Technician

1. Address NOT_NULL - this attribute specifies the location associated with the local technician, therefore cannot be null.

Model_Stylist

1. IdModel primary key, foreign key - part of the composite key and acts as a foreign key, ensuring the uniqueness of the relation model-stylist and establishing it.
2. IdStylist primary key, foreign key - also part of the composite key and acts as a foreign key, contributing to the uniqueness of the relationship and linker of the entities stylists and models.

Piece of Clothing

1. IdPiece primary key - each piece of clothing is unique and identified by its ID.
2. collection NOT NULL - collection has to be known to know where the piece belongs.
3. IdPerson NOT NULL, foreign key - IdPerson identifies the designer who created the piece. It is a reference to the Designers´ primary key, IdPerson.

Collection

1. IdCollection primary key - each collection is unique and identified by its ID.
2. CName NOT_NULL, UNIQUE - the name of the collection cannot be null (e.g. it will be shown at the time of its runway show), and has to be distinctive.
3. Brand foreign key, NOT_NULL - the brand foreign key links a collection to a specific brand. Since the brand's name is defined in the Brand class

(BName), it serves as a foreign key in the collection class. It always holds value.
4. Season NOT NULL - season must be given since it defines the collection (e.g. *BRAND* Fall/Winter Collection)

Brand

1. IdBrand primary key - there can´t be two brands with the same ID.
2. BName NOT_NULL, UNIQUE - every brand has a name which is required and must be unique.
3. email NOT_NULL, UNIQUE - the email field cannot be null. Each brand must provide an email as the chosen means of communication, and this email must be unique to ensure that no two brands share the same email address.

Ticket

1. IdTicket primary key - each ticket is unique and identified by its ID.
2. IdShow foreign key, NOT_NULL - links the ticket to its designated show by referencing the latter´s foreign key.
3. IdPerson foreign key, NOT_NULL - links the ticket to the person that purchased it by referencing the latter´s foreign key.

Influencer

1. NumFollowers NOT_NULL; InstagramHandle NOT_NULL, UNIQUE - all influencers should present their number of followers and their Instagram handle, the latter being distinct for each influencer.

Media

1. Occupation NOT_NULL; MeansofCommunication NOT_NULL - these fields cannot be null to avoid ambiguity in processes that rely on this information.

Runway

1. IdShow primary key - each ID is unique and identifies each runway show.
2. Address foreign key, NOT_NULL - reference to the location details, and it must always have a value.
3. IdBrand foreign key, NOT_NULL - links the runway show to a specific brand. It can never be null, thus the association is always valid.
4. BName foreign key, NOT_NULL - the name of the brand must always have a value.
5. Date NOT_NULL; Time NOT_NULL - the date and time are mandatory.
6. RunwayBudget NOT_NULL - the budget captures the financial plan for each show and must have a value.

Model_Runway

1. IdShow primary key, foreign key - the IdShow being a primary key ensures uniqueness when combined with IdModel, forming a composite key. As a foreign key, it establishes a relationship between runway shows and models.
2. IdModel primary key, foreign key - as part of the composite key, it ensures uniqueness for each model-runway show pairing. As a foreign key, it reinforces the relationship described above.
3. Order not null - the order attribute specifies the sequence in which models appear in a show, hence being mandatory.

Event

1. IdEvent primary key - each event has a unique identifier, IdEvent.
2. Address NOT_NULL - this attribute specifies the location of the event, therefore cannot be null.
3. EventBudget NOT_NULL - the budget captures the financial plan for the event and must have a value.
4. OrganizingCompany NOT_NULL - this field corresponds to the responsible organization accountable for the event, thus cannot be null.

Sponsor

1. IdSponsor primary key - this attribute makes sure each sponsor is uniquely identified.
2. Company NOT_NULL, UNIQUE - the name of the company must have a value and be different for each company.
3. Email NOT_NULL, UNIQUE - this attribute stores the contact of the sponsor, it must always be provided and unique.
4. Value NOT_NULL - captures the monetary value provided by the sponsor, and it must be given.
5. Resources NOT_NULL - specifies any additional resources, cannot be null.
6. Agreements NOT_NULL - the agreements outline terms and conditions agreed upon with the sponsor, hence never being null.

➔ In SQL, restrictions like ON UPDATE and ON DELETE were used in order to maintain the integrity of the data.

IV. Relational Schema

A. Initial Relational Schema

Person (Name, Email, Age)

Model (Name->Person, Nationality, Bust, Waist, Hips, Height, Weight, Health Issues)

Designer (Name->Person, Nationality, Photography, Biography)

Spectator (Name->Person, Seat)

Influencer (Name->Spectator, NumFollowers, InstagramHandle)

Media (Name->Spectator, Occupation, MeansofCommunication)

Technician (Name->Person, Function, Material)

LocalTechnician (Name->Technician, Address->Event)

Stylist (Name->Technician, Model->Model)

Runway (idShow, Address->Event, BName->Brand, Date, Time, Budget)

Model_Runway (idShow->Runway, Model->Model, Order)

Event (Address, Budget, OrganizingCompany)

Sponsor (Company, Email, Value, Resources, Agreements)

Ticket (idShow->Runway, Name->Spectator, Type, Status)

Brand (BName, Email, Country of origin, Logo)

Collection (CName, BName->Brand, Season)

PieceofClothing (idPiece, Collection->Collection, Designer->Designer, MainMaterial, Color, Type)

B. AI Input Analysis and Suggestions

The initial relational schema in section A. was submitted to ChatGPT-4, developed by OpenAI. We asked for “input and changes we could potentially implement”. It told us “The schema is well-structured (...). However, there are areas for improvement to enhance clarity, normalization, and flexibility.”

We received 12 suggestions, of which we fully implemented only 1 (~8%). While some raised relevant issues and brought up questions we hadn’t considered, others felt out of context.

Starting with the suggestion we adopted: we improved the linking of entities by using IDs instead of names, like in *Person*, *Brand*, *Collection*, *Event* and *Sponsor*, resulting in more consistent and reliable associations.

Other suggestions we didn’t implement include, for example:

1. Creating a relationship table for Designers working with specific Brands
    - Designers are associated with a collection, that is then associated with a brand. Making this addition would clutter the UML with a useless table.

2. Add a table to associate collections with specific runway shows.
    - Collections are associated with a brand, that is then associated with a runway. Making this addition would clutter the UML with a useless table.
3. Ensure clear separation of concepts. Remove the budget from Runway if it is already in Event.
    - Although both Runway and Event share a Budget attribute, it is relevant for each to retain this value independently. The Runway budget is critical for brands, while the Event budget serves broader purposes such as planning future editions or managing sponsorships. However, to address potential confusion, we renamed the attributes to EventBudget and RunwayBudget for clarity.
4. Extract Material into a separate table to model many-to-many relationships between FashionPiece and Material.
    - This suggestion highlighted an issue we hadn't considered. Our intent with the Material attribute was to identify the primary material of a FashionPiece to facilitate collection statistics, not to list all materials used. Instead of implementing the suggestion, we renamed Material to MainMaterial to clarify its purpose. Similarly, we renamed Color to MainColor to better reflect our intent.
5. Add a PersonType attribute or create separate tables for categories of people (e.g., Spectator, Model, Technician).
    - This suggestion was redundant, as our schema already included types of people such as Spectator, Model, and Technicians as subclasses of Person.

Overall, we can conclude that while AI tools like ChatGPT-4 are powerful and can provide valuable insights, their suggestions must be carefully analyzed and evaluated in the context of the specific problem. Blindly implementing all recommendations without considering the schema's goals and constraints can lead to unnecessary complexity or misalignment with the intended design.

C. Final Relational Schema

Person (idPerson, Name, Email, Age, Nationality)

Model (idPerson->Person, Bust, Waist, Hips, Height, Weight, HealthIssues)

Designer (idPerson->Person, PhotographyURL, BiographyURL)

Spectator (idPerson->Person, Seat)

Influencer (idPerson->Spectator, NumFollowers, InstagramHandle)

Media (idPerson->Spectator, Occupation, MeansofCommunication)

Technician (idPerson->Person, Function, Material)

LocalTechnician (idPerson->Technician, idEvent->Event)

Stylist (idPerson->Technician)

Model_Stylist (idModel->Model, idStylist->Stylist)

Runway (idShow, idEvent->Event, idBrand->Brand, Date, Time, RunwayBudget)

Model_Runway (idShow->Runway, idModel->Model, Order)

Event (idEvent, Address, EventBudget, OrganizingCompany)

Sponsor (idSponsor, Company, Email, Value, Resources, Agreements)

Ticket (idTicket, idShow->Runway, idPerson->Spectator, Type, Status)

Brand (idBrand, BName, Email, Country of origin, Logo)

Collection (idCollection, CName, idBrand->Brand, Season)

PieceofClothing (idPiece, idCollection->Collection, idPerson->Designer, MainMaterial, MainColor, Type)

V. Functional Dependencies and Normal Forms Analysis

Relation	Person (idPerson, Name, Email, Age, Nationality)
Keys	{idPerson} {Email}
Functional Dependencies	idPerson -> Name, Email, Age, Nationality Email -> idPerson, Name, Age, Nationality
BCNF	Yes, since idPerson and Email are superkeys
3NF	Yes, attributes are dependent on the superkeys

Relation	Model (idPerson->Person, Bust, Waist, Hips, Height, Weight, HealthIssues)
Keys	{idPerson}
Functional Dependencies	idPerson -> Bust, Waist, Hips, Height, Weight, HealthIssues
BCNF	Yes, since idPerson is a superkey
3NF	Yes, attributes are dependent on the superkey

Relation	Designer (idPerson->Person, PhotographyURL, BiographyURL)
Keys	{idPerson}
Functional Dependencies	idPerson -> PhotographyURL, BiographyURL
BCNF	Yes, since idPerson is a superkey
3NF	Yes, attributes are dependent on the superkey

Relation	Spectator (idPerson->Person, Seat)
Keys	{idPerson}
Functional Dependencies	idPerson -> Seat
BCNF	Yes, since idPerson is a superkey
3NF	Yes, attributes are dependent on the superkey

Relation	Influencer (idPerson->Spectator, NumFollowers, InstagramHandle)
Keys	{idPerson} {InstagramHandle}
Functional Dependencies	idPerson -> NumFollowers, InstagramHandle InstagramHandle -> idPerson, NumFollowers
BCNF	Yes, since idPerson and InstagramHandle are superkeys
3NF	Yes, attributes are dependent on the superkeys

Relation	Media (idPerson->Spectator, Occupation, MeansofCommunication)
Keys	{idPerson}
Functional Dependencies	idPerson -> Occupation, MeansofCommunication
BCNF	Yes, since idPerson is a superkey
3NF	Yes, attributes are dependent on the superkey

Relation	Technician (idPerson->Person, Function, Material)
Keys	{idPerson}
Functional Dependencies	idPerson -> Function, Material

BCNF	Yes, since idPerson is a superkey
3NF	Yes, attributes are dependent on the superkey

Relation	LocalTechnician (idPerson->Technician, idEvent->Event)
Keys	{idPerson}
Functional Dependencies	idPerson -> idEvent
BCNF	Yes, since idPerson is a superkey
3NF	Yes, attributes are dependent on the superkey

Relation	Stylist (idPerson->Technician)
BCNF	Yes
3NF	Yes

Relation	Model_Stylist (idModel->Model, idStylist->Stylist)
BCNF	Yes
3NF	Yes

Relation	Runway (idShow, idEvent->Event, idBrand->Brand, Date, Time, RunwayBudget)
Keys	{idShow}
Functional Dependencies	idShow -> idEvent, idBrand, Date, Time, RunwayBudget
BCNF	Yes, since idShow is a superkey
3NF	Yes, attributes are dependent on the superkey

Relation	Model_Runway (idShow->Runway, idModel->Model, Order)
Keys	{idShow, idModel}
Functional Dependencies	idShow, idModel -> Order
BCNF	Yes, since idShow, idModel is a superkey

3NF	Yes, attributes are dependent on the superkeys

Relation	Event (idEvent, Address, EventBudget, OrganizingCompany):
Keys	{idEvent}
Functional Dependencies	idEvent -> Address, EventBudget, OrganizingCompany
BCNF	Yes, since idEvent is a superkey
3NF	Yes, attributes are dependent on the superkey

Relation	Sponsor (idSponsor, Company, Email, Value, Resources, Agreements)
Keys	{idSponsor}
Functional Dependencies	idSponsor -> Company, Email, Value, Resources, Agreements
BCNF	Yes, since idSponsor is a superkey
3NF	Yes, attributes are dependent on the superkey

Relation	Ticket (idTicket, idShow->Runway, idPerson->Spectator, Type, Status)
Keys	{idTicket} {idShow, idPerson}
Functional Dependencies	idTicket -> idShow, idPerson, Type, Status idShow, idPerson -> idTicket, Type, Status
BCNF	Yes, since idSponsor and idShow, idPerson are superkeys
3NF	Yes, attributes are dependent on the superkey

Relation	Brand (idBrand, BName, Email, Country of origin, Logo)
Keys	{idBrand}
Functional Dependencies	idBrand ->BName, Email, Country of origin, Logo
BCNF	Yes, since idBrand is a superkey
3NF	Yes, attributes are dependent on the superkey

Relation	Collection (idCollection, CName, idBrand->Brand, Season):

Keys	{idCollection}
Functional Dependencies	idCollection -> CName, idBrand, Season
BCNF	Yes, since idCollection is a superkey
3NF	Yes, attributes are dependent on the superkey

Relation	PieceOfClothing (idPiece, idCollection->Collection, idPerson->Designer, MainMaterial, MainColor, Type)
Keys	{idPiece}
Functional Dependencies	idPiece -> Collection, idPerson, MainMaterial, MainColor, Type
BCNF	Yes, since idPiece is a superkey
3NF	Yes, attributes are dependent on the superkey

After submission of this version of our Functional Dependencies and Normal Forms Analysis, ChatGPT-4 said “no further changes were necessary for improvement”.

VI. SQLite Database Creation

When adapting our database for implementation in SQLite, we initially created a baseline version to serve as a functional starting point, create1.sql. This initial design underwent significant refinement based on insights and suggestions generated by ChatGPT-4, once again.

A key aspect of our initial version was its loosely defined constraints. That resulted in many of the AI-generated suggestions focusing on refining these constraints and enhancing the structure to improve functionality and maintainability. Let us analyse the suggestions given:

1. Add `DEFAULT` values for attributes where appropriate.
    - We added `DEFAULT` value ‘Valid’ for Status in *Ticket.*
2. Use appropriate data types (`BOOLEAN` for flags like Photography in Designer, `NUMERIC` for monetary values like Budget).
    - In the *Designer* class, we kept Photography as text, but changed it to PhotographyURL for clarity. The same was applied to Biography, which was changed to BiographyURL.
    - We adopted the change on the Budget attributes from `INTEGER` to `NUMERIC`, as it made sense considering the nature of monetary values.
3. Use `CHECK` constraints or enums for certain fields - Fields like Type and Status in Ticket, Function in Technician, and Type in FashionPiece are likely to have predefined values.

- For the *Ticket* class, we added CHECK's for predefined values in Status ('Valid', 'Used', 'Expired') and in Type ('VIP', 'Regular', 'Media').
- We decided not to adopt these CHECK's in other classes, as the variety of types of technicians required may vary depending on multiple aspects, like the type of venue the event is set in, type of models and their requirements, etc. Same goes to the type of clothing a designer makes and classifies as. Overall, we figured it would be too extensive to implement a CHECK on these situations.

4. Explicitly name constraints. Use prefixes like 'chk_' for check constraints (e.g., chk_age_person) , 'nn_' for not-null constraints (e.g., nn_age_person) , 'uq_' for unique constraints (e.g., uq_email_person), 'fk_' for foreign keys (e.g., fk_idperson_model) etc., for better maintainability and debugging.
    - We implemented this in all classes, as it made identifying the constraints easier, and deleted 'DEFAULT NULL' instances present in the first version.
5. Review cascade settings for foreign keys to ensure appropriate handling of deletions. For example: Consider using ON DELETE CASCADE for relationships like *Runway -> Event* and *Collection -> Brand*, where deleting a parent entity would naturally remove its dependent rows.
    - This wasn't implemented, as the database is configured to RESTRICT, so the parent cannot be deleted if child entities exist.

This section explores the AI-driven recommendations, and the rationale behind the decisions to adopt or reject certain suggestions. The final version, with the implemented suggestions discussed above, is create2.sql.

While many of these suggestions were incorporated, others were set aside after careful evaluation, as they did not align with our requirements or practical constraints. However, we believe the overall input on this topic was more valuable than that on the Relational Schema (discussed in Section III.B.).

VII. Data Loading

Data loading in a scenario like ours, where real data is not involved, is essentially about getting fake identities, budgets, materials, collection names, sponsors—anything you can think of. What would at first seem like a tedious task, is easily turned into a trivial one, with the use of AI to generate data.

The prompts given to ChatGPT-4 were simple and straightforward, requesting for:

- A list with 20 names (first and last) from different countries;
- A possible budget for a fashion week event;
- Fashion week address in Paris and in Italy;
- Collection name for Louis Vuitton for spring/summer;
- Examples of material, type and color of fashion pieces;
- Common sponsors of Fashion Week Events;
- (...)

We consider the prompts made to provide a nice blend between fully fake AI-generated data (like people's names, event budget, etc.) and real-life information (like actual sponsors, real brands and addresses that usually hold the Fashion Week, the event we drew inspiration from for this project, etc.).

Finally, with the data in hand, we then manually proceeded with the data insertion in populate1, keeping the relations between tables in mind. We must highlight the only difference between the two populate files is the order in which the data is read, as in the first version some tables needed data that had not been read yet, leading to errors.

VIII. Generative AI integration

The integration of AI, specifically ChatGPT-4, into the design and refinement of the database schema for this project proved to be a beneficial tool in guiding the development process. However, while many suggestions were implemented, many others were discarded for not meeting the project's needs and constraints.

We found that, throughout the project, AI proved most useful for validation and optimization tasks, while manual efforts were more effective in the creative process, and in areas where domain knowledge is required.

This goes to show the importance of **critical evaluation** when using AI, as not every recommendation will be applicable, correct or beneficial in every context. By reading the sections where the suggestions given by AI are analysed, it becomes clear that the success of AI's application depends on the team's ability to **discern which suggestions are suitable**.