Faculdade de Engenharia da Universidade do Porto

Database Project 25/26 - LEIC

Museum Database - Group 1704

Museu Nacional Soares dos Reis, Porto

Coordination:
Professor Michel Ferreira

Students & Authors:
Alejandro Zuheros up202502194@fe.up.pt
Rodrigo Pinto up202403624@fe.up.pt
Tiago Rodrigues up202403626@fe.up.pt

Índice

1. Final changes to the UML Model **3**

2. Relational Schema/Model **4**
  2.1 Relational Schema done by the authors **4**
  2.2 AI Feedback -> Strength and Weak/Discussion Points **6**
  2.3 Final Relational Schema after incorporating AI feedback **9**

3. Initial Analysis of Functional Dependencies and Normal Forms **11**
  3.1 Initial analysis and proposal done by the authors **11**
  3.2 AI Feedback on the authors analysis **16**
  3.3 Final analysis, after incorporating AI feedback **17**

4. SQLite Database Creation - AI **18**

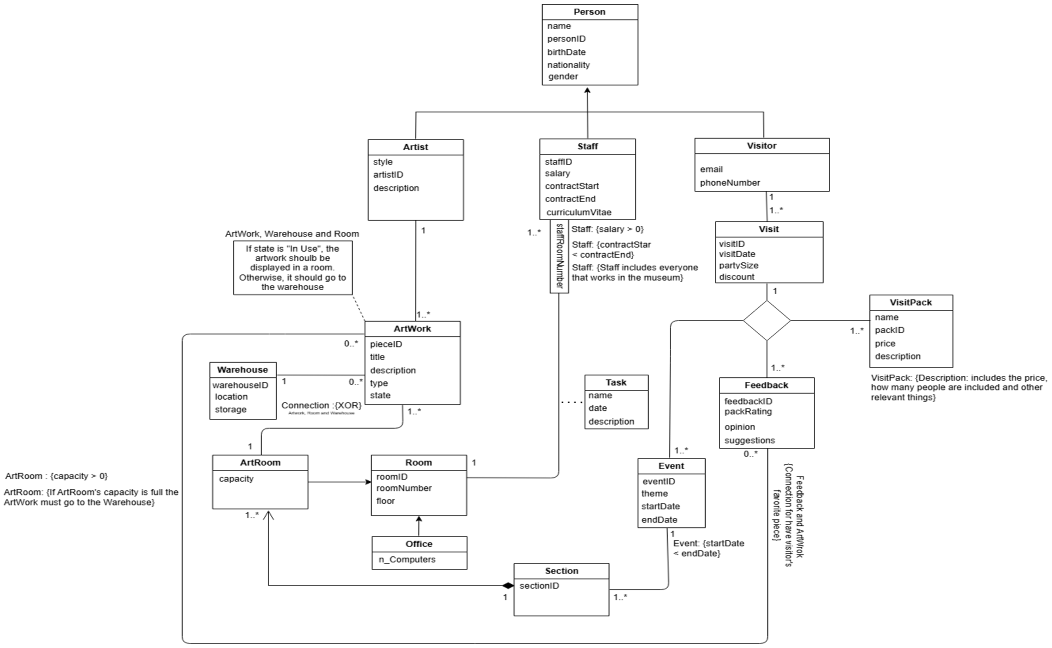
5. Data Loading - AI **20**

6. Opinion on generative AI integration **21**

1. Final changes to the UML Model

We decided to **make some changes to the UML Model** created in the first submission because we thought there were minor inconsistencies that were not referred in the evaluation and that would negatively affect the relational model and, consequently, the final database outcome. The changes were:

1. The moving of the attribute “visitID” from the “Visitor” Class to the “Visit” Class;
2. We added a price attribute to the “VisitPack” Class. We first had decided to indicate the price on the description of the pack, but we think that would be clearer if this attribute were isolated on database;
3. We didn’t have a primary key for the ”Feedback” so we added a “feedbackID” attribute; The same occurs for the “Event” and “Warehouse” classes. We added an “eventID” and “warehouseID” respectively.
4. Eliminated “purchaseDate” from “visitPack”, since there were two attributes in two different classes with the same meaning and value. “purchaseDate” in “VisitPack” and “visitDate” in “Visit”.



URL:https://drive.google.com/file/d/1CLzcpBGRpzEqA6pte_IXEiYduIdaai40/view?usp=sharing

2. Relational Schema/Model

2.1 Relational Schema created by the authors

1. Person (personID, name, birthDate, nationality, gender);

2. Artist (artistID, personID -> Person, style, description);

3. Staff (staffID, personID -> Person, salary CHECK (salary > 0), contractEnd, contractStart CHECK (contractEnd IS NULL OR contractStart < contractEnd), curriculumVitae);

4. Visitor (visitorID, personID -> Person, email NOT NULL UNIQUE, phoneNumber NOT NULL UNIQUE);

5. Visit (visitID, visitDate, partySize, discount, visitorID -> Visitor);

6. VisitPack (packID, name, price, description);

7. Feedback (feedbackID, packRating, opinion, suggestions);

8. Event (eventID, theme NOT NULL UNIQUE, startDate, endDate CHECK (startDate < endDate);

9. Section (sectionID, eventID -> Event);

10. ArtWork (pieceID, title NOT NULL, description, type, state CHECK (state IN ('InUse','Maintenance', 'Stored')), artistID -> Artist, warehouseID -> Warehouse, roomID -> Room CHECK ((state = 'InUse' AND roomID IS NOT NULL AND warehouseID IS NULL) OR (state <> 'InUse' AND roomID IS NULL AND warehouseID IS NOT NULL)));

11. Warehouse (warehouseID, location, storage);

12. Room (roomID, roomNumber NOT NULL, floor NOT NULL);

13. ArtRoom (<u>roomID</u> -> Room, capacity CHECK (capacity > 0), sectionID -> Section);

14. Office (<u>roomID</u> -> Room, n_computers);

15. Task (<u>staffID</u> -> Staff, <u>roomID</u> -> Room, name NOT NULL, date NOT NULL, description);

16. Experience (visitID -> Visit, <u>packID</u> -> VisitPack, <u>feedbackID</u>-> Feedback, <u>eventID</u> -> Event);

17. StaffRoomNumber (<u>staffID</u> -> Staff, <u>roomID</u> -> Room);

18. Favourite (<u>FeedbackID</u> -> Feedback, <u>pieceID</u> -> ArtWork)

2.2 AI Feedback -> Strength and Weak/Discussion Points

AI tool used: ChatGPT

✔ Strengths

- **Clear separation of entity roles through Person as a supertype and Artist, Staff, and Visitor as specializations, maintaining flexibility and semantic clarity.**
- **Use of technical identifiers (e.g., feedbackID, eventID, warehouseID, packID, pieceID, visitID) improves referential integrity and follows database design best practices.**
- **Correct implementation of complex relationships via the Experience table, enabling the representation of N:M relationships between Visit and VisitPack, while linking feedback and events logically.**
- **Strong enforcement of business rules using CHECK constraints (e.g., dates consistency, allowed states for artworks, XOR rule for object location).**
- **Well-structured specialization of Room into ArtRoom and Office using PK=FK, a standard and effective approach to model inheritance in relational databases.**
- **Good normalization level (3NF / BCNF) with no detected redundancy or update anomalies; functional dependencies are maintained correctly.**
- **Introduction of the price attribute in VisitPack, ensuring structured and query-friendly business-critical information.**

- **Logical association between Feedback and ArtWork via Favourite, preserving traceability and allowing statistics or recommendation-style features.**

⚠ Weak / Discussion Points

- **Artist, Staff, and Visitor introduce new IDs (artistID, staffID, visitorID) even though each one already inherits personID from Person. Using separate surrogate keys adds redundancy and complicates joins. Making personID the primary key in each specialization would simplify the structure.**

We also concluded this, but we can demonstrate that this is a good structure. Imagine that a staff member decides, on his day off, to go visit the museum. If we used personId as a primary key in those two classes, we would not be able to distinguish if he was going to work or to visit. So, basically, without those three attributes, we cannot distinguish the roles between Staff, Visitor and Artist, since the class Person only specifies the personal attributes of a person, and not what they do and their advantages/gains from it.

- **In the Person table, both email and phoneNumber are defined as NOT NULL UNIQUE. However, in many museums, visitors may not have an email address, and phone numbers can be shared between family members.**

In our point of view, the email, phone number and address given by the visitor, only belongs to the person in charge of buying all the tickets. If it is not only one person, we can have either the attributes of one person only, or the attributes from each person who bought tickets. When we implemented this, we already checked this rule, since, in our homes, for example, we don't share the same email and phone number. That would be a big mess.

- **Date interval constraint in Event needs correction to match the intended rule (startDate < endDate).**

It is correct, so nothing needs to change. Not a good weak point, since startDate and endDate already do respect the time constraint.

- **Redundant definition of StaffRoomNumber and Task relations (both specify the same, where a staff member is supposed to be).**

Not a good point. We are keeping these to relations because, using Task we can get the name for the function to be completed, date when this happened and what was done using description. And now, using StaffRoomNumber, we can complement the relation, by having a number related to a single staff member inside a single specified room.

- **Currently, in the Task table, to identify some staff member tasks, it is only used staffId and roomID as primary keys, leaving aside the possibility to differentiate the dates they occurred in. To be correct, date should also be a primary key in Task relation.**

We agree — using only staffId and roomId as the primary key is not sufficient, since it does not allow distinguishing tasks that occur on different dates. Including date as part of the primary key in the **Task** relation is the correct approach to ensure each task is uniquely and accurately identified.

2.3 Final Relational Schema after incorporating AI feedback

Final Comment: After carefully reviewing our relational model and comparing it with the strong/weak points pointed out by AI, we can conclude that most of our work was already done correctly, since we did not have to change a lot of things (only the relation “Task”). This process also helped us identify some inconsistencies, explain some of them and refine our model.

Final Relational Schema:

1. **Person (personID, name, birthDate, nationality, gender);**

2. **Artist (artistID, personID -> Person, style, description);**

3. **Staff (staffID, personID -> Person, salary CHECK (salary > 0), contractEnd, contractStart CHECK (contractEnd IS NULL OR contractStart < contractEnd), curriculumVitae);**

4. **Visitor (visitorID, personID -> Person, email NOT NULL UNIQUE, phoneNumber NOT NULL UNIQUE);**

5. **Visit (visitID, visitDate, partySize, discount, visitorID -> Visitor);**

6. **VisitPack (packID, name, price, description);**

7. **Feedback (feedbackID, packRating, opinion, suggestions);**

8. **Event (eventID, theme NOT NULL UNIQUE, startDate, endDate CHECK (startDate < endDate);**

9. **Section (sectionID, eventID -> Event);**

10. ArtWork (pieceID, title NOT NULL, description, type, state CHECK (state IN ('InUse','Maintenance', 'Stored')), artistID -> Artist, warehouseID -> Warehouse, roomID -> Room CHECK ((state = 'InUse' AND roomID IS NOT NULL AND warehouseID IS NULL) OR (state <> 'InUse' AND roomID IS NULL AND warehouseID IS NOT NULL)));

11. Warehouse (warehouseID, location, storage);

12. Room (roomID, roomNumber NOT NULL, floor NOT NULL);

13. ArtRoom (roomID -> Room, capacity CHECK (capacity > 0), sectionID -> Section);

14. Office (roomID -> Room, n_computers);

15. Task (staffID -> Staff, roomID -> Room, date NOT NULL, name NOT NULL, description);

16. Experience (visitID -> Visit, packID -> VisitPack, feedbackID-> Feedback, eventID -> Event);

17. StaffRoomNumber (staffID -> Staff, roomID -> Room);

18. Favourite (FeedbackID -> Feedback, pieceID -> ArtWork)

3. Initial Analysis of Functional Dependencies and Normal Forms

3.1 Initial analysis and proposal done by the authors

A **relation** is in **3rd Normal Form** if, for each nontrivial Function Dependency, either the left side is a superkey, or the right side consists of prime attributes only. In other words, for each no trivial FD X->Y:

1. **X** is a superkey to the relation, **OU**
2. Every attribute in **Y** is a prime attribute

This ensures that non-prime attributes do not depend on other non-prime attributes, preventing transitive dependencies.

For a **relation** to be in **Boyce-Codd Normal Form,** the left side of every nontrivial Functional Dependency must be a superkey. In other words, for every non trivial FD X->Y:

1. **X** is a superkey (X can be one or more);

Finally, <u>if a relation respects these two guidelines, they are ready to be implemented</u>. Now that we understand correctly how to check if a relation respects the 3rd and Boyce-Codd Normal Form, we can create the relations in a correct way:

1. **Person (<u>personID,</u> name, birthDate, nationality, gender);**

personID -> name, birthDate, nationality, gender;

Candidate key: personID;

Non-prime attributes: name, birthDate, nationality, gender;

2. Artist (artistID, personID -> Person, style, description);

artistID -> personID, style, description;

Candidate key: artistID;

Non-prime attributes: personID, style, description;

3. Staff (staffID, personID -> Person, salary CHECK (salary > 0), contractEnd, contractStart CHECK (contractStart < contractEnd), curriculumVitae);

staffID -> personID, salary, contractStart, contractEnd, curriculumVitae;

Candidate key: staffID;

Non-prime attributes: personID, salary, contractStart, contractEnd, curriculumVitae;

4. Visitor (visitorID, personID -> Person, email NOT NULL UNIQUE, phoneNumber NOT NULL UNIQUE);

visitorID -> personID, email, phoneNumber;

Candidate key: visitorID;

Non-prime attributes: personID, email, phoneNumber;

5. Visit (visitID, visitDate, partySize, discount, visitorId -> Visitor);

visitID -> visitDate, partySize, discount, visitorID;

Candidate key: visitID;

Non-prime attributes: visitDate, partySize, discount, visitorID

6. VisitPack (packID, name, price, description);

packID -> name, price, description;

Candidate key: packId;

Non-prime attributes: name, price, description;

7. Feedback (feedbackID, packRating, opinion, suggestions);

feedbackID -> packRating, opinion, suggestions;

Candidate key: feedbackID;

Non-prime attributes: packRating, opinion, suggestions;

8. Event (eventID, theme NOT NULL UNIQUE, startDate, endDate CHECK (startDate < endDate);

eventID -> theme, startDate, endDate;

Candidate key: eventID;

Non-prime attributes: theme, startDate, endDate;

9. Section (sectionID, eventID -> Event);

sectionID -> eventID;

Candidate key: sectionID;

Non-prime attributes: eventID;

10. ArtWork (<u>pieceID</u>, title NOT NULL, description, type, state CHECK (state IN ('InUse','Maintenance', 'Stored')), artistID -> Artist, warehouseID -> Warehouse, roomID -> Room CHECK ((state = 'InUse' AND roomID IS NOT NULL AND warehouseID IS NULL) OR (state <> 'InUse' AND roomID IS NULL AND warehouseID IS NOT NULL)));

pieceID -> artistID, title, description, type, state, warehouseID, roomID;
Candidate key: pieceID;
Non-prime attributes: artistID, title, description, type, state, warehouseID, roomID;

11. Warehouse (<u>warehouseID</u>, location, storage);

warehouseID -> location, storage;
Candidate key: warehouseID;
Non-prime attributes: location, storage;

12. Room (<u>roomID</u>, roomNumber NOT NULL, floor NOT NULL);

roomId -> roomNumber, floor;
Candidate key: roomID;
Non-prime attributes: roomNumber, floor;

13. ArtRoom (<u>roomID</u> -> Room, capacity CHECK (capacity > 0), sectionID -> Section);

roomId -> capacity, sectionID;
Candidate key: roomID;
Non-prime attributes: capacity, sectionID;

14. Office (roomID -> Room, n_computers);

roomID -> n_computers;
Candidate key: roomID;
Non-prime attributes: n_computers;

15. Task (staffID -> Staff, roomID -> Room, date NOT NULL, name NOT NULL, description);

(staffID, roomID, date) -> name, description;
Composite candidate key: (staffID, roomID, date);
Non-prime attributes: name, description;

16. Experience (visitID -> Visit, packID -> VisitPack, feedbackID-> Feedback, eventID -> Event);

(visitID, packId) -> feedbackID, eventID;
Composite candidate keys: (visitID, packID);
Non-prime attributes: feedbackID, eventID;

17. StaffRoomNumber (staffID -> Staff, roomID -> Room);

(staffID, roomId) -> {};
Candidate keys: (staffID, roomID);
Non-prime attributes: {};

18. Favourite (<u>FeedbackID</u> -> Feedback, <u>pieceID</u> -> ArtWork);

(feedbackID, pieceID) -> {}
Candidate keys: (feedbackID, pieceID);
Non-prime attributes: {};

General comment for the relations above (authors) :

After analysing all of the relations above, we can conclude that all of them respect both Boyce-Codd and 3rd Normal Form. All the attributes are properly tied to their respective candidate keys, maintaining full dependency on their keys and not in other prime attributes, handling transitive dependency. Similarly, all the elements on the left side are superkeys. In our case, the superkeys are simple candidate keys (only one element) and composite keys (2 or more elements).

3.2 AI Feedback on the authors analysis

AI tool used: ChatGPT

Comment: “In my view, the relational schema is well-organized and reflects the domain clearly and accurately. Each relation contains only the attributes that belong to it, and the links between entities are consistent and meaningful. The specialization of Person into Artist, Staff, and Visitor is clean and avoids redundancy, while the associative tables are minimal and properly defined with composite keys. The handling of artwork placement and the distinction between room types show good attention to practical constraints. Overall, the schema avoids unnecessary dependencies and keeps each relation focused on its role. **The relations satisfy both 3rd Normal Form and BCNF, and the schema is ready for implementation.”**

3.3 Final analysis, incorporating AI feedback

Final Comment: Since we conclude the same thing as AI, then we can confidently proceed with the implementation of the database, knowing that the schema is well-structured, free of redundancy, and fully compliant with **3rd Normal Form** and **BCNF**. The relations do not change and the final version is:

- **personID ->** name, birthDate, nationality, gender;
- **artistID ->** personID, style, description;
- **staffID ->** personID, salary, contractStart, contractEnd, curriculumVitae;
- **visitorID ->** personID, email, phoneNumber;
- **visitID ->** visitDate, partySize, discount, visitorID;
- **packID ->** name, price, description;
- **feedbackID ->** packRating, opinion, suggestions;
- **eventID->** theme, startDate, endDate;
- **sectionID ->** eventID;
- **pieceID ->** artistID, title, description, type, state, warehouseID, roomID;
- **warehouseID ->** location, storage;
- **roomId ->** roomNumber, floor;
- **roomId ->** capacity, sectionID;
- **roomID ->** n_computers;
- **(staffID, roomID, date) ->** name, description;
- **(visitID, packId) ->** feedbackID, eventID;
- **(staffID, roomId) ->** {};
- **(feedbackID, pieceID) ->** {};

4. SQLite Database Creation - AI

AI tool used: ChatGPT

Prompt given to ChatGPT: I'm sending you the relational model ("relational_model.pdf") and my attempt to create the corresponding database ("create1.sql"). Please list the strengths, some weaknesses and suggestions for improvement. Was there any major mistake made?

"create1.sql" and "relational_model.pdf" were attached to the prompt.

Response:

"Strengths

- **The relational schema is structurally consistent with the conceptual model and follows normalization principles.**
- **Correct use of primary keys, foreign keys, and referential integrity constraints.**
- **The CHECK constraints in some tables demonstrate good attention to data consistency rules.**
- **Identification of entities and relationships aligns well with the museum domain context.**
- **Good separation between Person, Visitor, Staff, and Artist, reflecting proper specialization / inheritance.**

Weaknesses and Issues Identified

Some tables reference others that do not yet exist at the time they are created, which can cause foreign key errors when running the script. Example:

- **Artist references Person, but Person was created after it.**
- **ArtRoom references Room and Section that appear later.**

This must be resolved by reordering the CREATE TABLE statements.”

Final Comment: This is, in fact, a very important subject in our attempt to create the first version of the database. In the improved version, “create2.sql”, we corrected this by reordering the CREATE TABLE statements. First we added all the base tables like Person, Room, Warehouse… followed by dependent tables, for example, Visitor, Staff, Artist and so on. We can now apply the same logic to “populate1.sql” and make the process much easier and more consistent.

5. Data Loading - AI

AI tool used: ChatGPT

Prompt given to ChatGPT: Based on the file attached called “populate1.sql”, can you spot any errors and tell us what should be… And also, do you think the database given is well populated?

“populate1.sql” was attached to the prompt.

Response: **“The populate1.sql script is generally well-structured and demonstrates good understanding of data integrity and relational dependencies. The insertion order correctly respects foreign key constraints, and the sample data is coherent and realistic within the context of the domain. The use of different types of records (multiple staff members, visitors, events, rooms, and artworks with different states) provides a solid basis for testing queries and constraints such as CHECK rules and ON DELETE/UPDATE behaviors.**

Overall, the population script effectively supports the logical model and is suitable for functional validation of the database. But there is one simple mistake.

❗ The problem in your SQL INSERT

Your INSERT statement contains two incorrect semicolons (;) in the middle of the VALUES list, which prematurely end the SQL command.

This problem occurs in line 100 and 156. Should be a comma (,).”

Final Comment: Great error spotted by AI and it is already changed. “populate2.sql” can be created, changing this small mistake. The database can now accept all inserts properly and run smoothly.

6. Opinion on generative AI integration

In general, we liked the idea of using AI to complement our work, especially in the first submission. We felt that AI had a bigger impact on our work during the UML correction phase, particularly in cleaning up the connections between classes and helping us make the database structure more consistent and coherent. It allowed us to identify issues more quickly and focus our time on refining the design rather than getting stuck on smaller details.

On the other hand, at some points, we felt like AI was not really important and made things more complex rather than being helpful. For example, when populating the database — a process that is quite repetitive and extensive — we used AI to maximize efficiency. However, instead of filling it with real data, it often started creating fictional artworks, artists, and other information that did not correspond to what actually exists in the Museu Nacional Soares dos Reis. These hallucinations forced us to manually validate and correct the generated data. So, although AI is a powerful tool, it still requires the user's critical thinking when being used.

Overall, we agree that AI can be used to improve the work done, but we also consider it a bit unnecessary in some parts of our work. It would be better to do a full report on what AI did to our work, rather than, at every task of the work, implement and justify what AI thinks of our work.