Milestone 1

1.1. Conceptual Modeling – Team (12 points)

Verify that your model meets the following requirements:

- A minimum of six entities and a maximum of nine entities.
- Each entity must have at least two attributes in addition to key attributes.
- At least one weak entity (i.e., an entity that is existence-dependent).
- At least one unary (recursive) relationship.
- At least one binary relationship with a 1:n cardinality.
- At least one binary relationship with a m:n cardinality.
- At least one IS-A (inheritance) relationship with at least two child entities.

1.1.1. Describe the Application Domain

In 3 to 5 paragraphs, describe the essential components of your application domain. Clearly identify the key entities and their relationships. Be precise in explaining how these elements interact and function.

To make your description clearer, follow this color-coding convention:

- Entities: BLUE
- Attributes: YELLOW
- Relationships: PINK
- Cardinalities: GREEN

For example, a simplified library could be described like this (not complete):

> *Our library has multiple locations [address, ZIP] to which book copies [purchase, price, is rented] are assigned.*
>
> *Customers [first name, last name] can rent books [ISBN, title, publication year] multiple times. […]*

Ensure that your submission differs substantially from the example to avoid point deductions.

1.1.2. Logical Design

Create an Entity-Relationship (ER) diagram using CHEN notation. Using any notation other than CHEN (see https://vertabelo.com/blog/chen-erd-notation/) will result in a point deduction. Ensure that all elements described in the application domain are represented and that the necessary functionality is supported in your model.

Note: The logical design has to be database-agnostic.

1.1.3. Relational Modeling

Derive a relational model from your logical design / ER diagram.

- In the report/template: Display your relational model as the headers of the SQL tables. Don't forget to highlight the primary and foreign keys as stated in the QA1 slides.
- In a separate .sql file: Use SQL CREATE statements to define all entities and relationships, ensuring consistency with your model.

Briefly describe the differences if your relational model differs from your logical model, for example, if the logical model is not in the third normal form.

1.2. Use Case Definition and Design – Individual (4 points)

Each team member must define one use case involving at least **three entities.** Each use case must involve data manipulation (insertion or updates) and fulfill one of the following:

- **Version 1:** Including a weak entity, or
- **Version 2:** Including an IS-A relationship. In this case, at least one entity needs to be outside of the IS-A relationship.

Every use case requires a textual description and a graphical representation (see below).

1.2.1 Textual Description

The textual description should adhere to UML standards and include:

- **Use Case Name:** Clear and concise.
- **Trigger:** What initiates the use case.
- **Preconditions:** Conditions required before starting the use case, e.g., user is logged in.
- **Main Flow:** Brief description of normal operations.
- **Postconditions:** The outcome after successful execution.

1.2.2. Graphical Representation

To visualize your use cases, use an Activity Diagram.

- **Swim Lanes:** Use the three swim lanes **frontend**, **backend**, and **database** to identify different system components and especially their interactions.
- **Message/Control Flow:** Arrows or lines should indicate the flow of control or messages between activities or components.
- **Start/Endpoints:** Clearly mark the beginning and end points of each use case.
- **Decisions vs. Activities:** Differentiate decision points from activities. The number of outgoing connections from a decision point should match the defined conditions.
- **Parallel Flows and Joins:** Depict parallel activities and the points where control flows converge.

Keep in mind, not all concepts may be relevant for every use case, and additional ones may be necessary based on your specific application. Unclear representations (resolution of screenshot too low, flow difficult to follow) will lead to point deduction.

1.3. Analytics Report – Individual (8 points)

1.3.1 Concept

Each team member must provide statistics on the data created/altered by their use case. To acquire these statistics, each team member must provide a query statement fulfilling the following requirements:

- Use data generated from your use case.
- Define a filter field.
- Involve three or more entities (excluding bridging tables).
- Data of each entity involved is included in the report

Describe your analytics report and provide (a screenshot of) the SQL query in your PDF report.

Important: The results of the query statement should change after executing the use case.

1.3.2 Proof of Concept

Use Maria DB and test the query on dummy data. The result of your query statement must be reproducible via SQL CREATE statements, SQL INSERT statements, and SQL QUERY statements in MariaDB.:

1) Use the SQL CREATE statements for the creation of your tables from 1.1.3.

   For the remaining parts of the proof of concept, you can ignore tables that are not involved in the analytics report (1.3.1)

2) Create and run SQL INSERT statements that insert a few rows (3-6 rows) of examples into each table necessary for the report. Ensure relational consistency.
3) Run your report (execute the SQL QUERY statement from 1.3.1).
4) Create and run SQL INSERT statements that mimic the execution of your use case.
5) Re-run your report.

Again: Ensure that the results from 3) and 5) should differ.

- In the report/template: Add screenshots of tables and query results.
- In three separate .sql files: To ensure the reproducibility of the analytics report and the proof of concept, hand in the create, insert, and query statements.

1.4. NoSQL Design – Individual (6 points)

1.4.1 Design Overview

Based on the use case you selected in Task 1.2, propose a possible data structure design in a document-based NoSQL database using MongoDB. Ensure that every entity involved in your use case is represented in the NoSQL design. Provide a formatted JSON example document for your NoSQL Design.

1.4.2 Expected Execution and Possible Changes

Make assumptions about the expected execution of your use case and the number of resulting database operations, and explain why your chosen NoSQL design is beneficial for your use case[1].

Analyze how changes in the number of database operations of your USE CASE (e.g., more frequent reads or writes) would affect your NoSQL data structure. Would you need to adjust document embedding, referencing, or indexing strategies? Explain your reasoning.

1.4.3 Five Rules of Thumb

Discuss your NoSQL design based on these five rules of thumb for document-based NoSQL databases. Provide brief, individual explanations on how you applied each rule to your design.

1. Favor embedding unless there is a compelling reason not to.
2. The need to access an object on its own is a compelling reason not to embed it
3. High-cardinality arrays are a compelling reason not to embed.
4. Consider the write/read ratio of a collection/document when denormalizing.
5. Structure your data to match the ways that your application queries and updates it.

[1] Applies to 1.4.2 and 1.4.3: Do not discuss the general advantages or disadvantages of document-based NoSQL databases; focus strictly on your design and use cases. No AI-generated answers.

Milestone 2

2.1. Infrastructure – Team (3 points)

Ensure that all compilation and dependencies are handled within a container during the build process. Test your project on a clean virtual machine (Debian 12) with minimal setup (Docker and unzip). Only expose necessary ports, and self-signed certificates are acceptable for HTTPS. Remove local bind mounts before submitting your code. (→ *Docker Tutorial 26. November*)

In the report, provide a ReadMe section with all necessary information to start up your application.

2.2. RDBMS Implementation (9 points)

2.2.1. DB Setup / Data Import / Base Function of Web System – Team (5p)

Implement a DB-filling script to generate randomized data for your use cases and reports. Add a button in your GUI to trigger the data import, replacing existing data.

Create the base functionality of your web system and a connection to your relational DB.

2.2.2. Implementation of the Use Case and the Analytics Report – Individual (4p)

Each team member must implement their use case and the corresponding analytics report. The web system must connect to a relational database.

Note: *Do not use SQLite. We recommend MariaDB for the best support. A GUI with excessive reliance on ID/PK fields may incur point deductions. While a full-featured login system is not required, users should be selectable in the GUI, assuming your use case is defined to be executed by different users.*

2.3. NoSQL Implementation (16 points) [2]

2.3.1 NoSQL Re-Design – Team (3p)

Define a NoSQL structure to support your **complete** application including the use cases **and** reports of **both** team members. Consider using techniques like denormalization, N-side referencing, and schemaless design where appropriate. Evaluate the relevance of foreign and primary keys from the RDBMS in the NoSQL context, explaining why they are retained or discarded.

Mention the necessary changes compared to your individual design in MS1. Justify your design decisions and compare them to at least one alternative, explaining the impact on the number of required joins and I/O operations.

[2] **Info for 2.3 (and all sub-points):** The past has shown that students who opted in on using ORM libraries to seamlessly enable support for MongoDB in the backend, without having to redesign entities and relationships, did not manage to properly discuss and argue for their NoSQL design decisions. **Therefore, it is not permissible to use ORM libraries.**

2.3.2 NoSQL DB Setup and Data Migration – Team (5p)

Ensure NoSQL collections uphold the same functionality as the RDBMS schema, preserving semantic information. The data migration process should not involve recreating data or writing simultaneously to both databases.

Introduce a button in the GUI to start the migration, clearing the NoSQL data beforehand. No requirement exists for migrating from NoSQL back to SQL.

2.3.3 NoSQL Use Case and Analytics Report – Individual (4p)

Implement the logic of your use case and analytics report (SQL query) the same way you did for 2.2 using your NoSQL data model / NoSQL Database instead.

2.3.4 NoSQL Analytics Report Statement – Individual (2p)

Discuss the query statement for your analytics report (in MongoShell syntax) and explain how the design impacts the execution of your report. Additionally, discuss any compromises made (if any) and describe the design decisions you made to mitigate/avoid these compromises.

2.3.5 NoSQL Indexing – Individual (2p)

Review your execution steps and implement MongoDB indexing that can benefit your application, especially for analytics reports. Review execution stats (`totalDocsExamined` and `indexesUsed`) to assess the impact of your indices and discuss the results using screenshots in your report.

2.4. Final Presentation after submission of M2 – Team (2 points)

- Upload a PDF of your slides (via the dedicated service on the course page).
- The presentation is mandatory – no presentation → 0 points on Part B.
- Prepare your project and presentation on your computer.
- Present both within 20 minutes total.

More information about the presentation will follow in the QA2 session.