ConstructionMasters: A Database Systems Case Study

Katya Trufanova

1 Introduction

This report details the design and implementation of a database system for ConstructionMasters Ltd., a firm that oversees different construction projects. The document offers an exhaustive breakdown of the project, detailing each step of the process, from the initial specifications to the final physical design and web application implementation.

2 Specifications

The project discussed in this report revolves around designing a database for ConstructionMasters Ltd., a company that manages a of construction projects, according to the specifications described below.

"ConstructionMasters Ltd"
ConstructionMasters Ltd. aims to create a versatile database encompassing essential information about construction projects and their various attributes. The database will serve as a comprehensive resource for project managers, architects, engineers, and stakeholders involved. The database will store data related to different types of construction projects, including residential, commercial, and industrial structures. Each project will be described by its name, purpose, location, and project manager. Specifics such as project start date, expected completion date, and budget will also be included. Construction materials will play a pivotal role in the database, with comprehensive details about various construction elements. Each material type, such as concrete, steel, wood, and glass, will be profiled with characteristics like strength, durability, cost, and recommended applications. The database will cover architectural and engineering aspects as well. Architectural details will include building designs, blueprints, floor plans, and 3D models. Engineering specifications will outline structural designs, load-bearing capacities, and safety measures. Environmental considerations will also find a place in the database. Information about sustainable construction practices, energy-efficient designs, and green building materials will be available to promote eco-friendly construction methods. For each project, the database will maintain a record of progress updates, including milestones achieved and challenges faced. This will facilitate real-time tracking of project development and help identify potential bottlenecks. Safety protocols and regulations will be highlighted to ensure compliance with industry standards and legal requirements.

The database must be designed to support a variety of operations, including but not limited to:

1. Insertion of a new project
2. Description of a project's information along with its milestones
3. Determination of the total cost of construction materials for a specific project
4. Presentation of the projects and their pertaining challenges, sorted chronologically
5. Displaying the projects in descending order based on the cost of construction materials used
6. Information retrieval of architectural and engineering features and progress updates for the project with the fewest milestones

3 Conceptual Design

3.1 Requirements collection and analysis

3.1.1 Structuring of requirements

The initial stage of the conceptual design process involves the collection and analysis of requirements. In order to structure these requirements effectively, they have been divided into general phrases and specific phrases related to various aspects of the projects.

General Phrases
ConstructionMasters Ltd. aims to create a versatile database for construction projects.

Phrases related to Projects
For each project, the database will store the name, purpose, location, project manager's name, start date, expected completion date, and budget.

Phrases related to Materials
For each type of material, the database will store the characteristics such as strength, durability, cost, and recommended applications.

Phrases related to Architectural Details
The database will include architectural details such as building designs, blueprints, floor plans, and 3D models.

Phrases related to Engineering Specifications
Engineering aspects such as structural designs, load-bearing capacities, and safety measures will be stored in the database.

Phrases related to Progress Updates
For each project, progress updates including milestones achieved and challenges faced will be maintained.

Phrases related to Safety Protocols
Safety protocols and regulations will be highlighted to ensure compliance with industry standards and legal requirements.

Phrases related to Environmental Considerations
Environmental considerations will also find a place in the database. Information about sustainable construction practices, energy-efficient designs, and green building materials will be available to promote eco-friendly construction methods.

3.1.2 Glossary of terms

To ensure clarity and consistency across the design process, a glossary of terms used in the requirements is provided.

Term	Description	Synonyms	Links
Project	Construction endeavor of a specific type (residential, commercial, industrial).	Construction Project	Project Manager, Material, Architectural Detail, Engineering Specification, Progress Update, Safety Protocol
Project Manager	Responsible for overseeing the project.	Manager	Project
Material	Elements used in construction (concrete, steel, wood, glass).	Construction elements, Construction materials	Project
Architectural Detail	Architectural designs including building designs, blueprints, floor plans, 3D models.	-	Project
Engineering Specification	Structural designs, load-bearing capacities, safety measures.	-	Project
Progress Update	Milestones achieved and challenges faced in a project.	-	Project
Safety Protocol	Safety measures and regulations for the project.	-	Project
Environmental Consideration	Promotes eco-friendly construction methods.	-	Project

3.2 Conceptual schema

3.2.1 Skeleton schema

The skeleton schema for the database is provided below. It offers an abstract, high-level view of the database structure, outlines the entities involved and the relationships between them, serving as a roadmap for the more detailed complete schema.

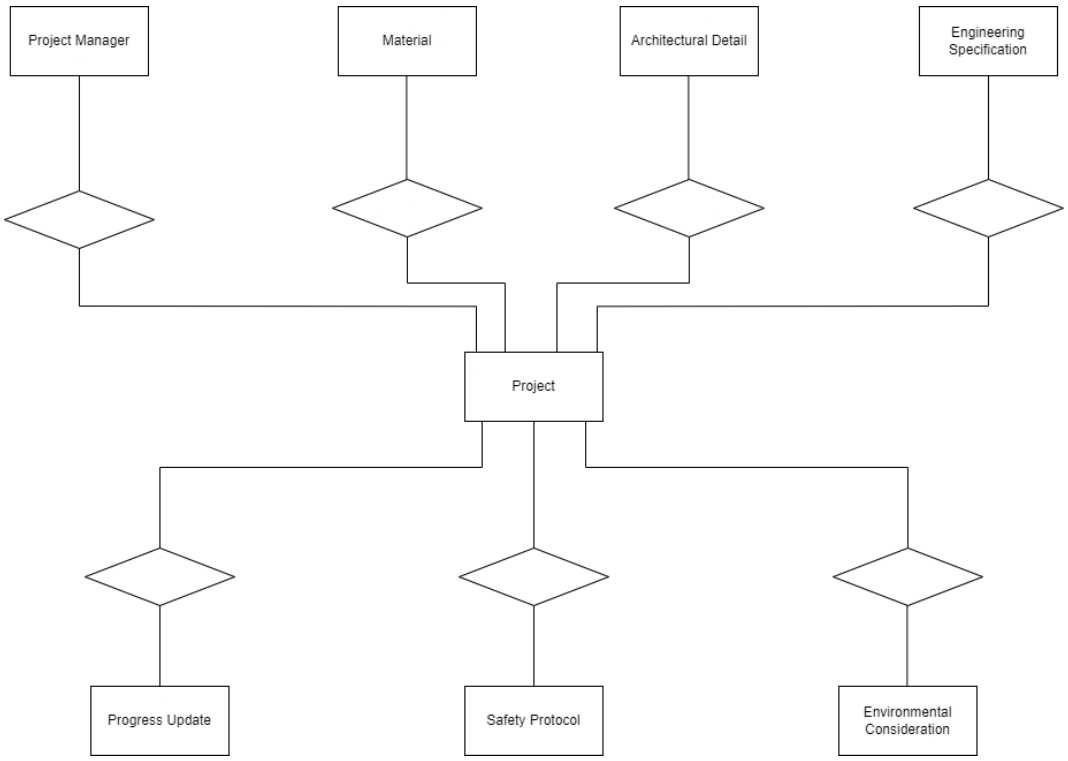

Figure 1: Skeleton schema.

3.2.2 Complete schema

In the figure below, the complete E-R schema for the database is presented, enriching the skeleton schema with attributes, relationship names, and cardinalities.

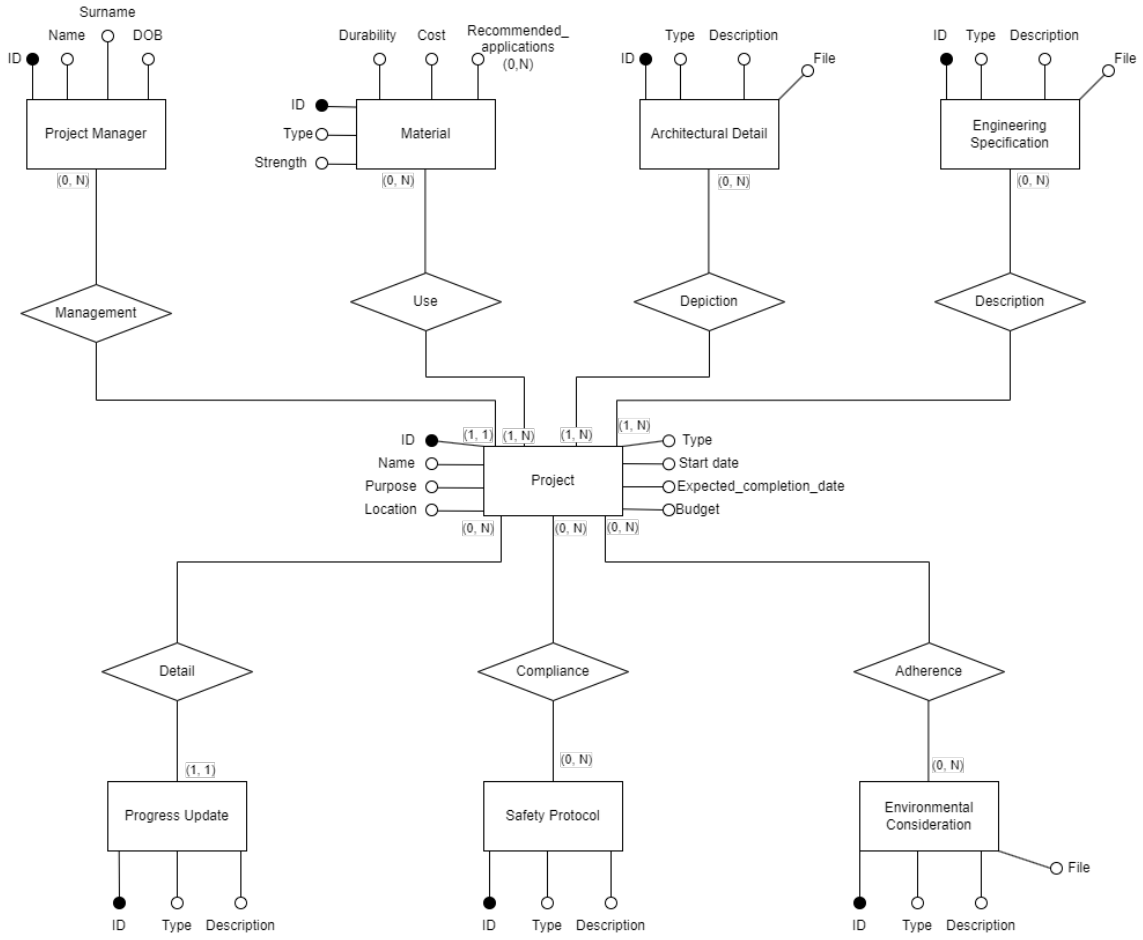

Figure 2: Complete E-R schema.

3.2.3 Business rules

In this section, constraints not expressed by the E-R schema are presented. These rules ensure data integrity and consistency by defining constraints on the values that certain attributes can take and the relationships between different entities.

(BR) The 'Type' attribute of the 'Project' entity must be 'residential', 'commercial' or 'industrial'.

(BR) A new 'Project Manager' can be inserted if their DOB indicates they are 18 years old or older.

(BR) The 'Type' attribute of the 'Architectural Detail' entity must be 'building design, 'blueprint', 'floor plan', or '3D model'

(BR) The 'Type' attribute of the 'Engineering Specification' entity must be 'structural design, 'load-bearing capacity', or 'safety measure'

(BR) The 'Type' attribute of the 'Safety Protocol' entity must be 'standard' or 'requirement'

(BR) The 'Type' attribute of the 'Environmental Consideration' entity must be 'construction practice', 'design', or 'green material'

(BR) A project's expected completion date must always be after the start date.

(BR) The total cost of materials for a project cannot exceed the budget of the project.

4 Logical Design

4.1 Table of volumes

In this section, the table of volumes for the entities and relationships in the defined E-R schema are provided. While some volumes are based on the provided specifications, others are deduced by making some assumptions about the numbers of relevant entities and relationships that are present in the database.

Concept	**Type**	**Volume**	**Explanation**
Project	E	10000	Given by the specifications
Project Manager	E	500	Assumption: Each manager manages more than one project, a reasonable number might be 20 projects per manager
Material	E	100	Assumption: 100 distinct types of construction materials are frequently used
Architectural Detail	E	20000	Assumption: There might be unique and reusable architectural details. Suppose, on average, two architectural details per project
Engineering Specification	E	15000	Assumption: Similar to architectural details, suppose on average, there is 2 engineering specifications per project
Progress Update	E	100000	Assumption: Suppose each project has an average of 10 progress updates
Safety Protocol	E	200	Assumption: Suppose there are 200 different safety protocols and regulations
Management	R	10000	Assumption: Each project has a unique manager at a given time. So, the number of relationships equals the number of projects
Use	R	500000	Assumption: On average, each project uses about 50 different types of materials
Depiction	R	20000	Assumption: Each project has 2 architectural details on average

Description	R	20000	Assumption: Each project has 2 engineering specifications on average
Detail	R	100000	Assumption: Each progress update is associated with a unique project, each project has an average of 10 progress updates
Compliance	R	50000	Assumption: On average, each project might need to comply with about 5 different safety protocols
Adherence	R	20000	Assumption: Each project may have multiple environmental considerations. Suppose an average of 2 per project

4.2 Table of operations

The table below provides an overview of the most relevant operations in the database, with the associated frequencies.

Operation	**Description**	**Frequency**
1	Inserting a new project	1/week
2	Print information about the project with information on milestones	1/week
3	Print information about construction materials total cost for a specific project	1/day
4	Print the projects and its challenges faced. For each project, the challenges are ordered by time	2/week
5	Print the projects ordered by the highest cost in terms of construction materials used	1/month
6	Print the architectural and engineering details and the progresses of the project with the lowest number of milestones	2/month

4.3 Access Tables

Access tables are an integral part of database design, serving as a guide for navigating the data involved in various operations. They delineate the 'logical path' to be followed to retrieve necessary information when navigating the E-R schema.

Access tables can help estimate the cost of an operation on the database. This is achieved by tallying the number of accesses to instances of entities and relationships required to execute the operation.

The type of access, whether read or write, is also documented in the access table, given that write operations are typically more demanding than read ones. This data can prove invaluable when making decisions during the restructuring of Entity-Relationship schemas.

Below, the access tables for the most relevant operations for the database are provided.

4.3.1 Operation 1: Inserting a new project

Concept	Type	Accesses	Type	Explanation
Project	E	1	W	Create a new project
Management	R	1	W	Each project has a manager
Use	R	50	W	Assumption: On average, each project uses about 50 different types of materials
Depiction	R	2	W	Assumption: Each project has 2 architectural details on average
Description	R	2	W	Assumption: Each project has 2 engineering specifications on average
Compliance	R	5	W	Assumption: On average, each project might need to comply with about 5 different safety protocols
Adherence	R	2	W	Assumption: Suppose an average of 2 per project

Cost (write operations are counted double): 126

4.3.2 Operation 2: Print information about the project with information on milestones

Concept	Type	Accesses	Type	Explanation
Project	E	1	R	Read specific project
Detail	R	10	R	Assumption: each project has an average of 10 progress updates
Progress Update	E	10	R	Assumption: each project has an average of 10 progress updates

Cost: 21

4.3.3 Operation 3: Print information about construction materials total cost for a specific project

Concept	Type	Accesses	Type	Explanation
Project	E	1	R	Read specific project
Use	R	50	R	Assumption: On average, each project uses about 50 different types of materials
Material	E	50	R	Assumption: On average, each project uses about 50 different types of materials

Cost: 101

4.3.4 Operation 4: Print the projects and the challenges they faced

Concept	Type	Accesses	Type	Explanation
Project	E	10000	R	Read all projects
Detail	R	100000	R	Assumption: each project has an average of 10 progress updates
Progress Update	E	100000	R	Assumption: each project has an average of 10 progress updates

Cost: 210000

4.3.5 Operation 5: Print the projects ordered by the highest cost in terms of construction materials used

Concept	Type	Accesses	Type	Explanation
Project	E	10000	R	Read all projects
Use	R	500000	R	Assumption: On average, each project uses about 50 different types of materials
Material	E	500000	R	Assumption: On average, each project uses about 50 different types of materials

Cost: 1010000

4.3.6 Operation 6: Print the architectural and engineering details and the progresses of the project with the lowest number of milestones

Concept	Type	Accesses	Type	Explanation
Project	E	10000	R	Read all projects
Detail	R	100000	R	Assumption: each project has an average of 10 progress updates
Progress Update	E	100000	R	Assumption: each project has an average of 10 progress updates, need to access all of them to find the milestones
Depiction	R	2	R	Assumption: Each project has 2 architectural details on average
Architectural Detail	E	2	R	Assumption: Each project has 2 architectural details on average
Description	R	2	R	Assumption: Each project has 2 engineering specifications on average
Engineering Specification	E	2	R	Assumption: Each project has 2 engineering specifications on average

Cost: 210008

4.4 Redundancy Analysis

Taking into consideration one of the operations with the highest cost (operation 6), it may be useful to consider creating a redundant attribute to lower the cost of this operations.

A useful redundancy could be to introduce an attribute 'number_of_milestones' for the Project entity. This would affect operations 1 and 6. For operation 1, the cost would remain the same as this

derived attribute can be calculated during the project creation. But, for operation 6, the cost would be significantly lowered, as the new table of accesses (with redundancy) would be as follows:

Concept	Type	Accesses	Type	Explanation
Project	E	10000	R	Read all projects
Depiction	R	2	R	Assumption: Each project has 2 architectural details on average
Architectural Detail	E	2	R	Assumption: Each project has 2 architectural details on average
Description	R	2	R	Assumption: Each project has 2 engineering specifications on average
Engineering Specification	E	2	R	Assumption: Each project has 2 engineering specifications on average

Thus, introducing this redundant attribute would lower the cost of operation 6 from 210008 to 10008, which is a significant improvement.

4.5 Logical Schema

Given the above considerations, the logical schema for the database is described in the Figure below.

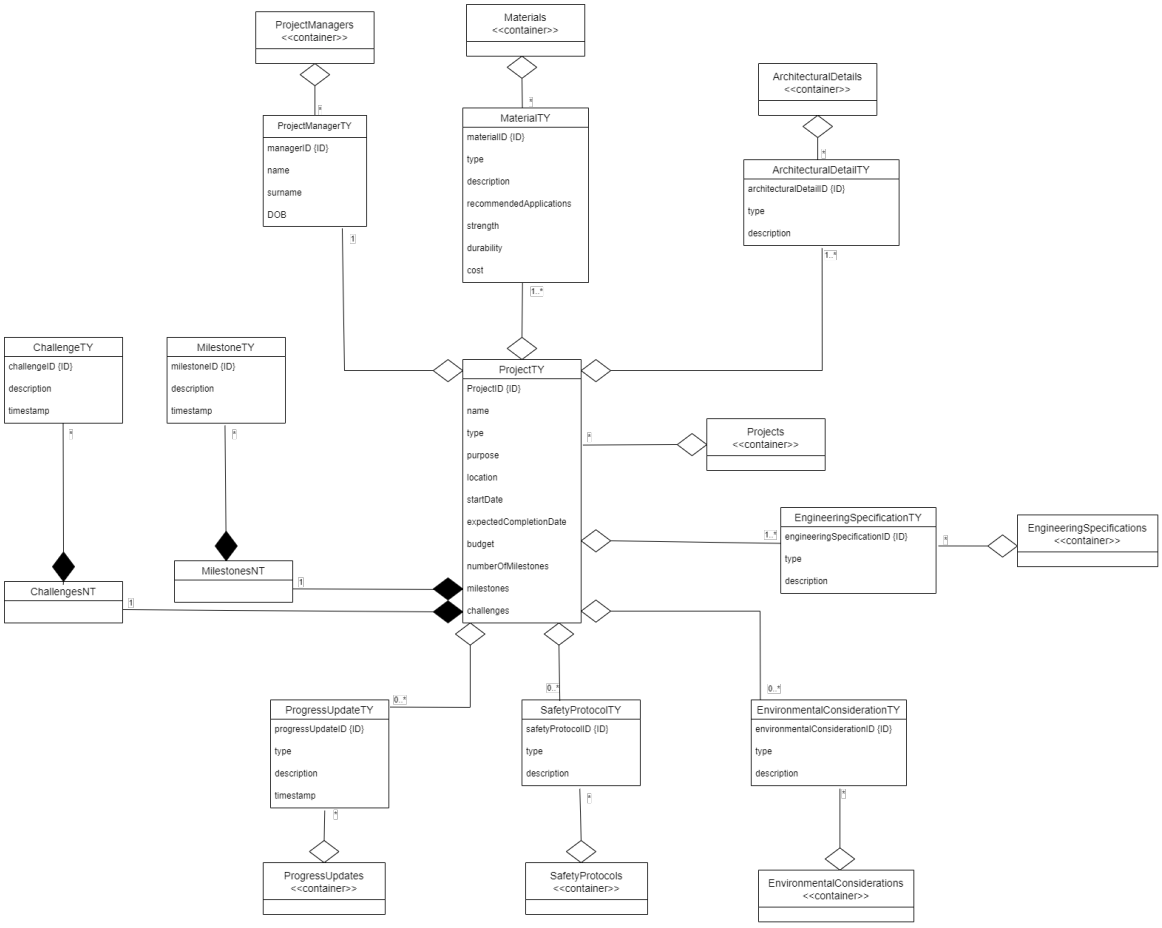

Figure 3: Skeleton schema.

5 Physical Design

5.1 Types and Tables

As a consequence of the logical design phase, the database was structured with the necessary types and tables to capture all the essential data relevant to the business operations. The schema was designed with a careful consideration of the business rules, data relationships, and the specific functionalities of the Oracle Database Management System.

Placed at the heart of the database is the ‘Project‘ table, which is an embodiment of the ‘ProjectType‘ object type. This table is designed to capture all the significant aspects of each construction project. Each project is uniquely identified by a ‘projectID‘, and further described by attributes such as ‘name‘, ‘type‘, ‘purpose‘, ‘location‘, ‘startDate‘, ‘expectedCompletionDate‘, ‘budget‘, and ‘numberOfMilestones‘.

To model the relationship between a project and its manager, the ‘Project‘ table includes a REF column ‘projectManager‘, which references the ‘ProjectManagerType‘ object type. This design decision allows for a more efficient and flexible approach to managing project manager data. The ‘ProjectManagerType‘ encapsulates details such as the ‘managerID‘, ‘name‘, ‘surname‘, and ‘DOB‘.

Furthermore, the ‘Project‘ table incorporates two nested tables: ‘milestones‘ and ‘challenges‘, each based on the ‘MilestoneTypeList‘ and ‘ChallengeTypeList‘ object types. Each entry in these nested tables represents a specific milestone or challenge associated with the project. This design choice was motivated by the inherent one-to-many relationship between a project and its milestones or challenges. Each milestone and challenge is uniquely identified by a ‘milestoneID‘ and ‘challengeID‘ respectively, and further described by a ‘description‘ and ‘timestamp‘.

The data model also includes several other object types, each with a corresponding table, to capture various aspects of the construction process:

- ‘MaterialType‘: This type encapsulates essential data about construction materials, including a unique ‘materialID‘, ‘type‘, ‘strength‘, ‘durability‘, ‘cost‘, and ‘recommendedApplications‘. The corresponding ‘Material‘ table stores instances of this type.

- ‘ArchitecturalDetailType‘ and ‘EngineeringSpecificationType‘: These types hold architectural and engineering details for the project. Both types are associated with a unique identifier and a ‘type‘ attribute. They also include a ‘description‘ attribute to provide further details. The ‘ArchitecturalDetail‘ and ‘EngineeringSpecification‘ tables store instances of these types.

- ‘SafetyProtocolType‘ and ‘EnvironmentalConsiderationType‘: These types capture safety and environmental considerations for each project. They have similar structures, each with a unique identifier, a ‘type‘, and a ‘description‘. The ‘SafetyProtocol‘ and ‘EnvironmentalConsideration‘ tables store instances of these types.

- ‘ProgressUpdateType‘: This type is designed to track progress updates for each project. It includes a unique ‘progressUpdateID‘, ‘type‘, ‘description‘, and ‘timestamp‘. The ‘ProgressUpdate‘ table serves as the storage for instances of this type.

To capture relationships between the ‘Project‘ table and the ‘Material‘, ‘ArchitecturalDetail‘, ‘EngineeringSpecification‘, ‘SafetyProtocol‘, ‘EnvironmentalConsideration‘, and ‘ProgressUpdate‘ tables, junction tables were created: ‘ProjectMaterial‘, ‘ProjectArchitecturalDetail‘, ‘ProjectEngineeringSpecification‘, ‘ProjectSafetyProtocol‘, ‘ProjectEnvironmentalConsideration‘, and ‘ProjectProgressUpdate‘.

Each junction table includes a composite primary key comprising ‘projectID‘ and a unique identifier from the associated table. This design ensures that each record in the junction table is unique and represents a specific relationship between a project and an instance from the associated table.

5.2 Triggers

The process of database design necessitates the establishment of certain constraints to ensure data integrity and consistency. This is enabled by the use of database triggers - programmatic units associated with a specific table which execute automatically when certain conditions are met. They form a critical part of the physical design phase and are employed to enforce business rules that cannot be directly expressed in the schema.

During the conceptual design phase, business rules were specified and subsequently translated into active rules during the physical design phase. Below a description of the specifics of these triggers,

their necessity, and their functional aspects is presented.

5.2.1 Enforcing Project Types

The first trigger, named 'project_type_check', ensures that the 'Type' attribute of the 'Project' entity adheres to the permissible values: 'residential', 'commercial', or 'industrial'. This trigger is invoked before an INSERT or UPDATE operation on the Project table. In case of violation, an error is raised, preventing the transaction from proceeding.

```
CREATE OR REPLACE TRIGGER project_type_check
BEFORE INSERT OR UPDATE ON Project
FOR EACH ROW
BEGIN
  IF :new.type NOT IN ('residential', 'commercial', 'industrial') THEN
    RAISE_APPLICATION_ERROR(-20001, 'Invalid project type. Type must
       be residential, commercial or industrial.');
  END IF;
END;
/
```

5.2.2 Age Constraint on Project Managers

The 'ProjectManagerDOBCheck' trigger enforces the business rule that a new 'Project Manager' can only be added if their age, calculated based on the provided date of birth, is 18 years or older. This row-level trigger executes before an INSERT operation on the ProjectManager table.

```
CREATE OR REPLACE TRIGGER ProjectManagerDOBCheck
BEFORE INSERT ON ProjectManager
FOR EACH ROW
DECLARE
  v_years NUMBER;
BEGIN
  SELECT (SYSDATE - :NEW.DOB) / 365 INTO v_years FROM dual;
  IF v_years < 18 THEN
    RAISE_APPLICATION_ERROR(-20003, 'A project manager must be 18
       years old or older.');
  END IF;
END;
/
```

5.2.3 Enforcing Types for Architectural Detail

The trigger 'architectural_detail_type_check' validates that the 'Type' attribute of the 'Architectural Detail' entity is one of the allowed types: 'building design', 'blueprint', 'floor plan', or '3D model'. This trigger is activated before an INSERT or UPDATE operation on the ArchitecturalDetail table.

```
CREATE OR REPLACE TRIGGER architectural_detail_type_check
BEFORE INSERT OR UPDATE ON ArchitecturalDetail
FOR EACH ROW
BEGIN
  IF :new.type NOT IN ('building design', 'blueprint', 'floor plan',
     '3D model') THEN
    RAISE_APPLICATION_ERROR(-20004, 'Invalid architectural detail
       type. Type must be building design, blueprint, floor plan, or
       3D model.');
  END IF;
END;
/
```

5.2.4 Enforcing Types for Engineering Specification

Similarly, the 'engineering_specification_type_check' trigger ensures that the 'Type' attribute of the 'Engineering Specification' entity is either 'structural design', 'load-bearing capacity', or 'safety measure'. This trigger is invoked before an INSERT or UPDATE operation on the EngineeringSpecification table.

```
CREATE OR REPLACE TRIGGER engineering_specification_type_check
BEFORE INSERT OR UPDATE ON EngineeringSpecification
FOR EACH ROW
BEGIN
  IF :new.type NOT IN ('structural design', 'load-bearing capacity',
     'safety measure') THEN
    RAISE_APPLICATION_ERROR(-20005, 'Invalid engineering
       specification type. Type must be structural design,
       load-bearing capacity, or safety measure.');
  END IF;
END;
/
```

5.2.5 Enforcing Types for Safety Protocol and Environmental Consideration

In the same vein, the 'safety_protocol_type_check' and 'environmental_consideration_type_check' triggers enforce the business rules related to the 'Type' attributes of the 'Safety Protocol' and 'Environmental Consideration' entities respectively. These triggers ensure that the 'Type' attributes contain only the predefined values.

```
CREATE OR REPLACE TRIGGER safety_protocol_type_check
BEFORE INSERT OR UPDATE ON SafetyProtocol
FOR EACH ROW
BEGIN
  IF :new.type NOT IN ('standard', 'requirement') THEN
    RAISE_APPLICATION_ERROR(-20006, 'Invalid safety protocol type.
       Type must be standard or requirement.');
  END IF;
END;
/

CREATE OR REPLACE TRIGGER environmental_consideration_type_check
BEFORE INSERT OR UPDATE ON EnvironmentalConsideration
FOR EACH ROW
BEGIN
  IF :new.type NOT IN ('construction practice', 'design', 'green
     material') THEN
    RAISE_APPLICATION_ERROR(-20007, 'Invalid environmental
       consideration type. Type must be construction practice,
       design, orgreen material.');
  END IF;
END;
/
```

5.2.6 Ensuring an Appropriate Project Duration

The trigger 'project_date_check' ensures that the completion date of a project is specified as being after the start date of a project. This is a row-level trigger and it is activated before an INSERT operation on the 'Project' table.

```
BEFORE INSERT OR UPDATE ON Project
```

```
FOR EACH ROW
BEGIN
  IF :new.startDate >= :new.expectedCompletionDate THEN
    RAISE_APPLICATION_ERROR(-20008, 'Expected completion date must be
       after the start date.');
  END IF;
END;
/
```

5.2.7 Check Total Costs Against Budget

The 'check_budget' trigger is used to enforce the business rule that the total cost of a project, calculated as the sum of all associated costs, cannot exceed the project's budget. This trigger is invoked after the INSERT or UPDATE operation on the 'Cost' table.

```
CREATE OR REPLACE TRIGGER check_budget
AFTER INSERT OR UPDATE ON Cost
FOR EACH ROW
DECLARE
  v_total_cost NUMBER;
  v_budget NUMBER;
BEGIN
  SELECT SUM(amount) INTO v_total_cost FROM Cost WHERE project_id =
     :new.project_id;
  SELECT budget INTO v_budget FROM Project WHERE id = :new.project_id;
  IF v_total_cost > v_budget THEN
    RAISE_APPLICATION_ERROR(-20002, 'Total costs cannot exceed the
       project budget.');
  END IF;
END;
/
```

These triggers are essential for maintaining data integrity and enforcing business rules at the database level. They provide a reliable mechanism for ensuring that only valid data is stored in the database, and that the database remains in a consistent state even as it evolves over time.

5.3 Procedures and functions

Several procedures in the Oracle Database were established to facilitate the most recurrent operations in the management of the construction projects. The implemented operations are described below:

1. Insertion of a new project
2. Rendering of a project's information along with its milestones
3. Determination of the total cost of construction materials for a specific project
4. Presentation of the projects and their pertaining challenges, sorted chronologically
5. Displaying the projects in descending order based on the cost of construction materials used
6. Information retrieval of architectural and engineering features and progress updates for the project with the fewest milestones

Each operation was implemented as a procedure, and it is essential to note that all procedures employ exception handling to manage errors and exceptions effectively.

The following subsections provide a detailed account of each procedure, including its purpose, implementation details, and how exceptions are managed.

5.3.1 Procedure 1: Insert New Project

This procedure is designed to insert a new project into the database. It requires several parameters that correspond to the project's properties, including project ID, name, type, purpose, location, start date, expected completion date, budget, number of milestones, and the manager's ID.

The procedure begins by attempting to insert a new row into the Project table. A ProjectType object is created with the provided parameters, with empty lists for milestones and challenges. The appropriate manager is fetched from the ProjectManager table via a subquery.

In the case of any error during the insertion, the procedure will roll back any changes made during the current transaction and re-raise the exception to be handled by the calling program.

```
CREATE OR REPLACE PROCEDURE insert_new_project(
p_projectID NUMBER,
p_name VARCHAR2,
p_type VARCHAR2,
p_purpose VARCHAR2,
p_location VARCHAR2,
p_startDate DATE,
p_expectedCompletionDate DATE,
p_budget NUMBER,
p_numberOfMilestones NUMBER,
p_managerID NUMBER
) IS
BEGIN
INSERT INTO Project
VALUES (
ProjectType(
p_projectID,
p_name,
p_type,
p_purpose,
p_location,
p_startDate,
p_expectedCompletionDate,
p_budget,
p_numberOfMilestones,
(SELECT REF(p) FROM ProjectManager p WHERE managerID = p_managerID),
MilestoneTypeList(),
ChallengeTypeList())
);
COMMIT;
EXCEPTION
WHEN OTHERS THEN
ROLLBACK;
RAISE;
END insert_new_project;
/
```

5.3.2 Procedure 2: Print Project and Milestones

This procedure retrieves and prints information about a given project and its milestones. It accepts a single parameter: the project's ID. A cursor is declared to fetch the project's ID, name, and milestone descriptions from the Project table, only for the specified project.

The procedure then enters a loop, fetching rows from the cursor and printing the project's ID, name, and the description of each milestone until no more rows are found. The cursor is closed upon completion of the operation.

```
CREATE OR REPLACE PROCEDURE print_project_and_milestones (p_projectID
   IN NUMBER)
IS
CURSOR c_projects IS
SELECT p.projectID, p.name, m.description
FROM Project p, TABLE(p.milestones) m
WHERE p.projectID = p_projectID;
v_project c_projects%ROWTYPE;
BEGIN
OPEN c_projects;
LOOP
FETCH c_projects INTO v_project;
EXIT WHEN c_projects%NOTFOUND;
DBMS_OUTPUT.PUT_LINE('Project ID: ' || v_project.projectID || ',
   Project Name: ' || v_project.name || ', Milestone Description: '
   || v_project.description);
END LOOP;
CLOSE c_projects;
END print_project_and_milestones;
/
```

5.3.3 Procedure 3: Print Total Material Cost

This procedure computes and prints the total cost of construction materials for a specific project. It accepts the project's ID as an input parameter. A variable is declared to hold the total cost of materials, which is computed with a SELECT statement that joins the Project, ProjectMaterial, and Material tables.

If the SELECT statement does not find any matching rows, it raises a NO_DATA_FOUND exception. The procedure handles this by printing a message that no data was found for the given project ID.

```
CREATE OR REPLACE PROCEDURE print_total_material_cost (p_projectID IN
   NUMBER)
IS
  v_totalMaterialCost NUMBER;
BEGIN
  SELECT SUM(m.cost) INTO v_totalMaterialCost
  FROM Project p
  JOIN ProjectMaterial pm ON p.projectID = pm.projectID
  JOIN Material m ON pm.materialID = m.materialID
  WHERE p.projectID = p_projectID;
  DBMS_OUTPUT.PUT_LINE('Project ID: ' || p_projectID || ', Total
     Material Cost: ' || v_totalMaterialCost);
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('No data found for project ID: ' ||
       p_projectID);
END print_total_material_cost;
/
```

5.3.4 Procedure 4: Print Projects and Challenges

This procedure retrieves and prints information about all projects and their associated challenges, sorted by project start date. No input parameters are required.

A cursor is declared to fetch the project's ID, name, start date, and challenge descriptions from the Project table. The cursor fetches rows sorted by start date in ascending order.

The procedure then enters a loop, fetching rows from the cursor and printing the project's ID, name, start date, and the description of each challenge until no more rows are found. The cursor is closed upon completion of the operation.

```
CREATE OR REPLACE PROCEDURE print_projects_and_challenges
IS
 CURSOR c_projects IS
   SELECT p.projectID, p.name, p.start_date, c.description
   FROM Project p, TABLE(p.challenges) c
   ORDER BY p.start_date ASC;
 v_project c_projects%ROWTYPE;
BEGIN
 OPEN c_projects;
 LOOP
   FETCH c_projects INTO v_project;
   EXIT WHEN c_projects%NOTFOUND;
   DBMS_OUTPUT.PUT_LINE('Project ID: ' || v_project.projectID || ',
      Project Name: ' || v_project.name || ', Start Date: ' ||
      v_project.start_date || ', Challenge Description: ' ||
      v_project.description);
 END LOOP;
 CLOSE c_projects;
END print_projects_and_challenges;
/
```

5.3.5 Procedure 5: Print Projects by Material Cost

This procedure retrieves and prints information about all projects, sorted by the total cost of construction materials used in descending order. No input parameters are required.

A cursor is declared to fetch the project's ID, name, and the total cost of materials, which is computed as the sum of the cost of each material multiplied by the quantity of that material used.

The procedure then enters a loop, fetching rows from the cursor and printing the project's ID, name, and the total cost of materials until no more rows are found. The cursor is closed upon completion of the operation.

```
CREATE OR REPLACE PROCEDURE print_projects_by_material_cost
IS
  CURSOR c_projects IS
    SELECT p.projectID, p.name, SUM(m.cost) as totalMaterialCost
    FROM Project p
    JOIN ProjectMaterial pm ON p.projectID = pm.projectID
    JOIN Material m ON pm.materialID = m.materialID
    GROUP BY p.projectID, p.name
    ORDER BY totalMaterialCost DESC;
  v_project c_projects%ROWTYPE;
BEGIN
  OPEN c_projects;
  LOOP
    FETCH c_projects INTO v_project;
    EXIT WHEN c_projects%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE('Project ID: ' || v_project.projectID || ',
       Project Name: ' || v_project.name || ', Total Material Cost: '
       || v_project.totalMaterialCost);
  END LOOP;
  CLOSE c_projects;
END print_projects_by_material_cost;
/
```

5.3.6 Procedure 6: Print Architectural and Engineering Features

This procedure retrieves and prints information about the architectural and engineering features and progress updates for the project with the fewest milestones. It first identifies the project with the fewest milestones using a subquery, then fetches data from the ArchitecturalFeature, EngineeringFeature, and ProgressUpdate tables for that project.

```
CREATE OR REPLACE PROCEDURE print_lowest_milestone_project_details
IS
  v_projectID NUMBER;
  CURSOR c_architecturalDetails IS
    SELECT ad.type, ad.description
    FROM ProjectArchitecturalDetail pad
    JOIN ArchitecturalDetail ad ON pad.architecturalDetailID =
       ad.architecturalDetailID
    WHERE pad.projectID = v_projectID;
  v_architecturalDetail c_architecturalDetails%ROWTYPE;
  CURSOR c_engineeringSpecifications IS
    SELECT es.type, es.description
    FROM ProjectEngineeringSpecification pes
    JOIN EngineeringSpecification es ON
       pes.engineeringSpecificationID = es.engineeringSpecificationID
    WHERE pes.projectID = v_projectID;
  v_engineeringSpecification c_engineeringSpecifications%ROWTYPE;
  CURSOR c_progressUpdates IS
    SELECT pu.type, pu.description, pu.timestamp
    FROM ProjectProgressUpdate ppu
    JOIN ProgressUpdate pu ON ppu.progressUpdateID =
       pu.progressUpdateID
    WHERE ppu.projectID = v_projectID;
  v_progressUpdate c_progressUpdates%ROWTYPE;
BEGIN
  -- find the project with the lowest number of milestones
  SELECT p.projectID INTO v_projectID
  FROM Project p
  ORDER BY p.numberOfMilestones ASC
  FETCH FIRST ROW ONLY;

  -- print architectural details for this project
  OPEN c_architecturalDetails;
  LOOP
    FETCH c_architecturalDetails INTO v_architecturalDetail;
    EXIT WHEN c_architecturalDetails%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE('Architectural Detail Type: ' ||
       v_architecturalDetail.type || ', Description: ' ||
       v_architecturalDetail.description);
  END LOOP;
  CLOSE c_architecturalDetails;

  -- print engineering specifications for this project
  OPEN c_engineeringSpecifications;
  LOOP
    FETCH c_engineeringSpecifications INTO v_engineeringSpecification;
    EXIT WHEN c_engineeringSpecifications%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE('Engineering Specification Type: ' ||
       v_engineeringSpecification.type || ', Description: ' ||
       v_engineeringSpecification.description);
  END LOOP;
```

```
  CLOSE c_engineeringSpecifications;

  -- print progress updates for this project
  OPEN c_progressUpdates;
  LOOP
    FETCH c_progressUpdates INTO v_progressUpdate;
    EXIT WHEN c_progressUpdates%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE('Progress Update Type: ' ||
       v_progressUpdate.type || ', Description: ' ||
       v_progressUpdate.description || ', Timestamp: ' ||
       v_progressUpdate.timestamp);
  END LOOP;
  CLOSE c_progressUpdates;

EXCEPTION
  WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('No project found with the lowest number of
       milestones.');
END print_lowest_milestone_project_details;
/
```

5.4 Query Optimization

Query optimization is a vital aspect of database design and implementation, playing a substantial role in enhancing the overall efficiency and effectiveness of the database system. This process involves tweaking and refining the queries to minimize their execution costs and processing times, thereby facilitating faster operations. Optimized queries are instrumental in reducing resource consumption and improving the response time of the database, which in turn, significantly enhances the application's performance.

In the context of the construction company's database system, considerations were made towards optimizing the most frequent operations. This involved the potential introduction of indexes to enhance the speed of operations. However, after an insightful comparison of execution plans, both with and without the introduction of indexes, it was observed that the Database Management System (DBMS) did not employ the defined indexes during the execution of queries. Consequently, the implementation of indexes was deemed unnecessary and was not included in the final database design. This decision was driven by the understanding that unused indexes would merely increase maintenance costs and consume storage space without contributing to the system's efficiency.

To ascertain the execution plan, the queries were extracted from the procedures, and the built-in "Explain Plan" functionality of Oracle SQL Developer was utilized. This allowed for a comprehensive comparison of how the query execution differs with and without the presence of an index.

Several operations constitute the construction company's database, each requiring a unique approach to optimization. The focus here is on operations 2, 3, 4, 5, and 6, which involve various tables and queries within the database.

Operation 2 involved filtering by the 'projectID' in the 'Project' table. Similarly, Operation 3 encompassed joins on the 'Project', 'ProjectMaterial', and 'Material' tables, with a filter on 'projectID'. It is worth noting that Oracle automatically indexes primary keys, which were used in these operations. Therefore, additional indexing was not required.

For Operation 4, the query sorts the results by 'timestamp' from the 'challenges' nested table. For large tables, sorting could be a slow process. To address this, an index on 'timestamp' was considered. However, Oracle does not permit the direct creation of an index on a nested table's column. A workaround to this limitation was to create a materialized view that flattens the nested table, followed by the creation of an index on that.

```
CREATE MATERIALIZED VIEW mv_project_challenges AS
SELECT p.projectID, p.name, c.description, c.timestamp
FROM Project p, TABLE(p.challenges) c;
```

```
CREATE INDEX idx_mv_challenges_timestamp ON
    mv_project_challenges(timestamp);
```

Once this view was created, the code for the procedure corresponding to Operation 4 was modified to utilize this view. This modification allows for the efficient extraction of project and challenge data through the materialized view mv_project_challenges, thereby reducing the overhead of navigating through the nested table structure for every execution.

However, upon comparing the execution plans with and without the index, it was discovered that the DBMS did not use the introduced index. As a result, the index was not implemented.

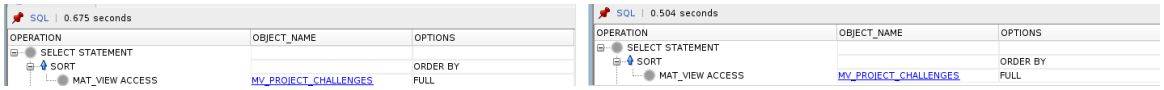

Figure 4: Execution Plan comparison with and without the index for operation 4.

Similarly, for Operation 5, the query involved joins on 'Project', 'ProjectMaterial', and 'Material' tables, with grouping on 'projectID' and 'name' from 'Project' and ordering by a computed sum. Here, indexing the join, group, and order columns could potentially expedite this query.

```
CREATE INDEX idx_project_name ON Project(name);
```

Yet again, the index was not utilized by Oracle, leading to its subsequent removal.

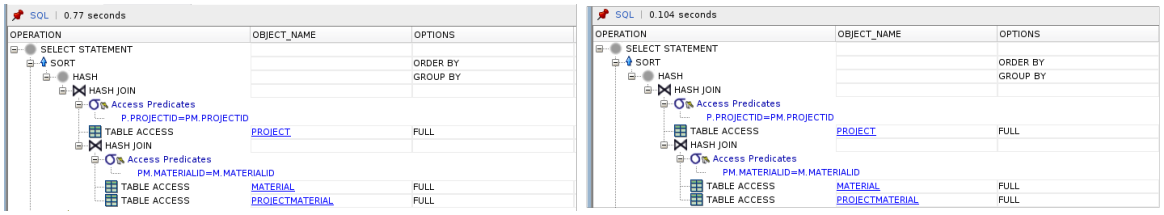

Figure 5: Execution Plan comparison with and without the index defined for operation 5.

In Operation 6, the query sorts results by 'numberOfMilestones' from 'Project' and retrieves the first row. An index on 'numberOfMilestones' was considered to enhance this operation.

```
CREATE INDEX idx_project_numberOfMilestones ON
    Project(numberOfMilestones);
```

However, this index too was not used, leading to its non-implementation.

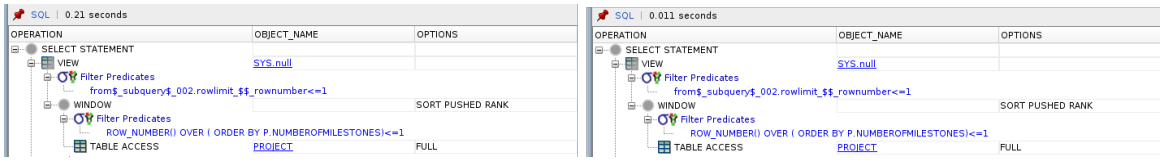

Figure 6: Execution Plan comparison with and without the index defined for operation 6.

The decision of the DBMS to use or disregard user-defined indexes can hinge on multiple factors, such as table size, data distribution, and Oracle's Cost-Based Optimizer decisions, among other aspects. Therefore, it is of paramount importance to test these indexes in a development or staging environment before applying them to production. This approach was adopted for this project, ensuring that the most effective and efficient database design was implemented.

6 Web Application

As a supplement to the primary project, a web application was developed to provide a user-friendly interface for interacting with the database. This web application was implemented using Java servlets and HTML pages, with development facilitated by Eclipse and the Tomcat server.

Servlets are Java objects that can handle network requests and responses. They act as the intermediary layer between a client's HTTP request and the server's database. As such, they were instrumental in implementing the operations outlined in the database specifications for this project.

Each operation was associated with a unique servlet that encapsulated the logic required to execute said operation. The servlets were then mapped to corresponding HTML pages, which served as the user interface for initiating these operations.

`ProjectServlet`

This servlet was responsible for implementing operation number 1—inserting a new project into the database. It achieved this by calling the stored procedure insert_new_project.

`ProjectInfoServlet`

This servlet was used to implement operation number 2. However, unlike the first servlet, it did not directly call a stored procedure. Instead, it had to reproduce the logic of print_project_and_milestones within the servlet itself due to the limitations of the DBMS_OUTPUT package.

`MaterialCostServlet`

This servlet, in conjunction with its corresponding HTML file, implemented operation number 3—calculating and displaying the total material cost for a specified project ID.

`ProjectChallengesServlet`

The purpose of this servlet was to display the challenges associated with each project. It was designed to implement operation number 4.

`HighestCostServlet`

This servlet was responsible for operation number 5—fetching the projects sorted by their total material cost.

`LowestMilestoneProjectServlet`

Finally, this servlet was designed to implement operation number 6—displaying the architectural details, engineering specifications, and progress updates of the project with the fewest milestones.

Each of these servlets followed a similar operational structure. They established a connection with the Oracle database, prepared a SQL query or invoked a stored procedure specific to their operation, executed the query, and retrieved the results. These results were then written to the HTTP response and sent back to the client.

The HTML files associated with each servlet were designed with simplicity and user-friendliness in mind. Each HTML page contained a form that, when submitted, initiated a GET or POST request to the corresponding servlet. The results of the operation were then displayed on the page.

In addition to servlets and HTML files, a CSS stylesheet was developed to enhance the aesthetics of the web pages, thereby improving the user experience.

Illustrative screenshots of the web pages are presented in Figure 7.

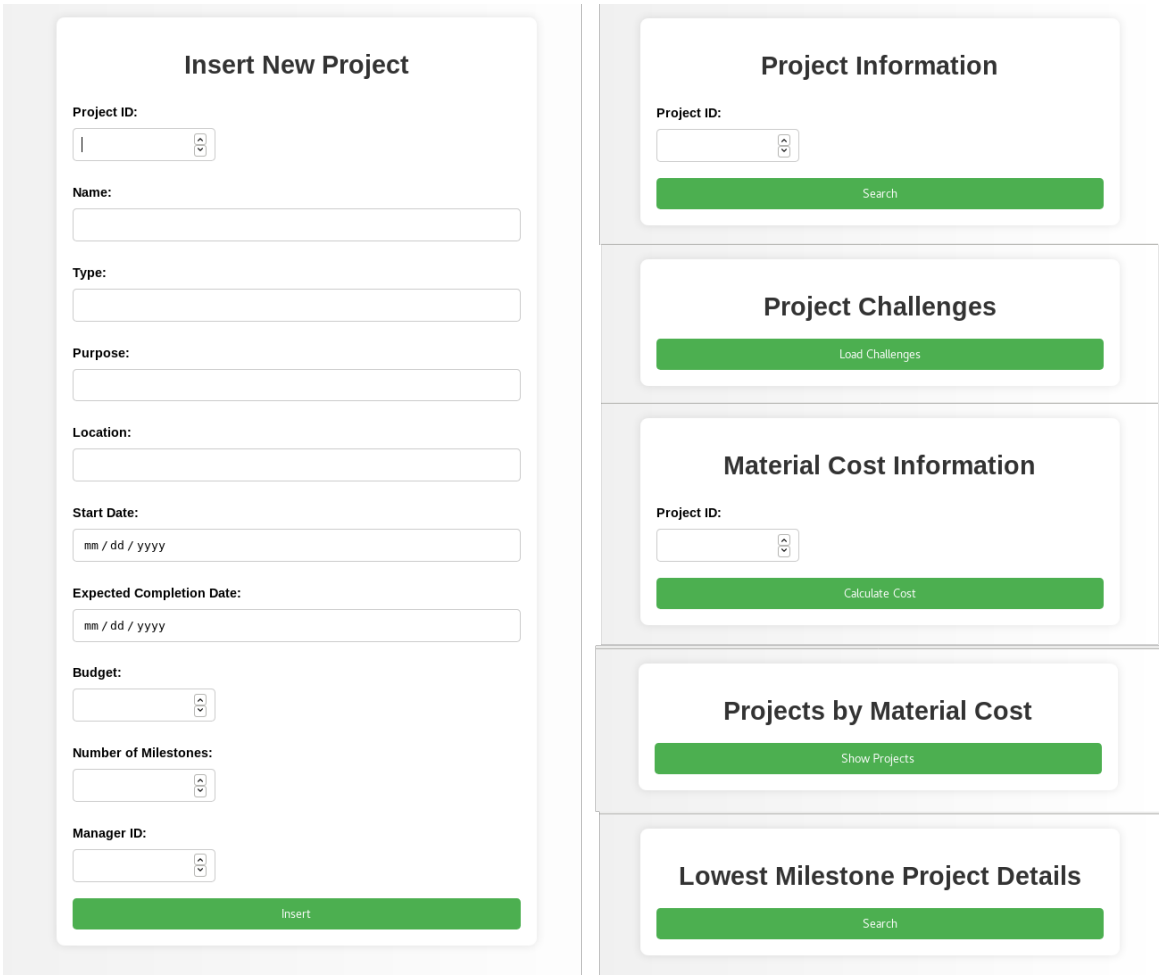

Figure 7: Screenshots of the implemented web pages.