

052400-1 VU Information Management and Systems Engineering (2025S)

Milestone 1

*(Please note: submission deadline: Tue 11.04.2025 13:00)*

Group 11

Student 1: Fahimy, Mustafa, matriculation number

Student 2: Nikzad, Khalifa, 12437813

*27.03.2025*, Vienna

**Contents**

[1](#_heading=h.13mf2v5qhhgj) Milestone 1 2

[1.1](#_heading=h.13mf2v5qhhgj) Team - Conceptual Modeling 2

[1.1.1](#_heading=h.s03ro0wt1fa8) Describe the Application Domain 2

[1.1.2](#_heading=h.ea5cq1u61sbz) Logical Design – ER Diagram in Chen Notation 2

[1.1.3](#_heading=h.ac7g6gje2los) Relational Modeling – SQL CREATE Statements 2

[1.2](#_heading=h.4bgm1ge0l1bs) Individual - Student 1 3

[1.2.1](#_heading=h.8yke9dc35bx5) Use Case Definition and Design 3

[1.2.2](#_heading=h.rfrga0qfa54v) Analytics Report 3

[1.2.3](#_heading=h.zfqa0owlzhe0) NoSQL Design 3

[1.3](#_heading=h.tlxymrnp9pth) Individual - Student 2 4

[1.3.1](#_heading=h.tu2pgtpyrzw6) Use Case Definition and Design 4

[1.3.2](#_heading=h.as692n11sj30) Analytics Report 4

[1.3.3](#_heading=h.4f3r7gipw15s) NoSQL Design 4

# Milestone 1

## Team - Conceptual Modeling

### Describe the Application Domain

In our system, **Users[ name,email,phone,password,...]** serves as a **supertype**, representing all individuals who interact with the platform, including both **Customers[Prefered Contact Method, Loyalty Points]** and **Employees[Job title, Salary, Hire Date, Shift]**. **Customers** are **users** who book **repair services[service\_name (e.g., Battery replacement), description,price,time\_taken]**, browse available repairs, and manage their **repair appointments[date\_time,status (Pending, Completed, Cancelled), total\_price (Final price for all selected services)]**.

On the other hand, **Employees** are **users** responsible for managing **repair appointments**, updating repair statuses, and ensuring smooth service execution. **Employees** may include **technicians**, who perform the actual repairs, **customer support staff**, who assist users with inquiries, and **admin** that manage overall the system. While both Customers and Employees share common attributes such as **name, email, phone, and password**, Employees have additional permissions, allowing them to **modify repair details, update repair appointments statuses, and manage payment[amount,status (Paid, Unpaid), payment\_method, payment\_date] processing**. This structured user hierarchy ensures clear role differentiation while maintaining an efficient repair management workflow.

The system supports different **device types[type\_name (e.g., Smartphone, Tablet…),description]**, each associated with **multiple brands[brand\_name (e.g., Samsung, Apple), country (Country of origin), founded\_year]**. Every brand, in turn, offers a variety of **device models[model\_name (e.g., Galaxy A55), release\_year (Year the model was released)]**, ensuring that **customers** can find the correct **repair service** for their specific device. Each **repair service** is tied to a particular **device model** and includes essential details such as **service name, description, price, estimated time for repair**. **Users** can select **multiple repair services** (e.g., battery replacement and screen repair) for the same device in a single **repair appointments**. Additionally, the system allows **users** to choose from different **service methods[method\_name (Visit Store, Send Device, Pickup), estimated\_time, cost],** but only **one service method** should be selected. The estimated time and cost of the service may vary depending on the selected method.

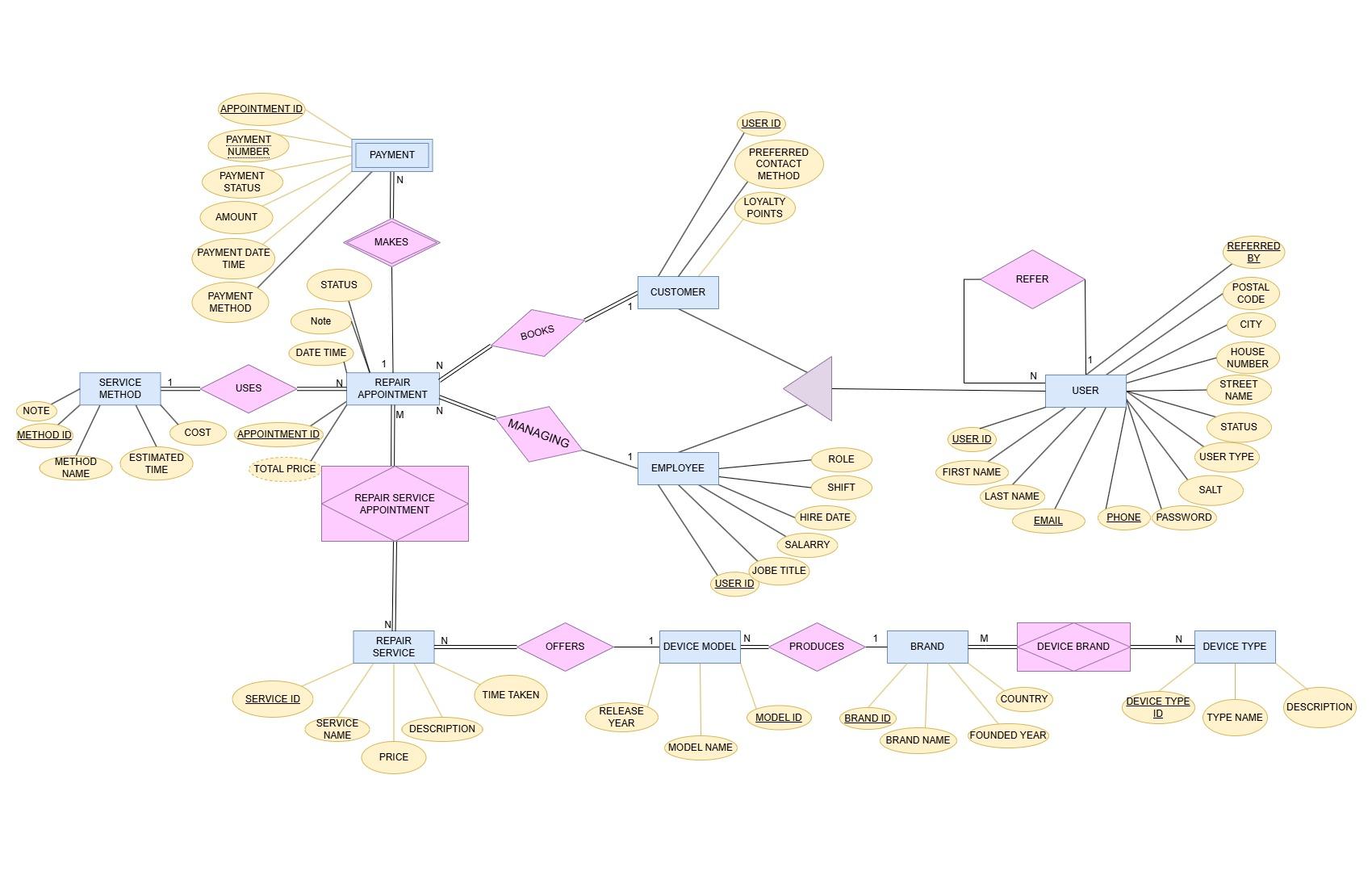
The **repair appointments** process includes selecting a **date and time** for the repair, ensuring convenience for the **users**. Once a **repair appointments** is confirmed, it is assigned a **status** (e.g., Pending, Completed, Canceled) that updates throughout the service process. **Users** also receive information about the **total cost** of the selected services and can proceed with **payment** using various methods such as **credit card, PayPal, or cash on delivery**.

The system maintains an organized structure of its relationships: **Users** can **place multiple repair appointments,** and each repair appointments can include **multiple repair services**. Additionally, the system supports an **administration role** where **admins** can **manage repair services,** update pricing, and oversee **repair appointments**. This structure ensures an efficient, user-friendly experience while allowing service providers to maintain a well-organized repair management system

After the description of the application domain using color-coding (see guidelines):

* List all entities and all relationships (fill in the table on the previous page)
* For the weak entity: List the weak entity, the strong entity, and the identifying relationship between weak and strong entity

### Logical Design – ER Diagram in Chen Notation



### Relational Modeling – SQL CREATE Statements

1. **User**: user\_id, first\_name, last\_name, email, phone, password, salt, user\_type, status, street\_name, house\_number, city, postal\_code, referred\_by
2. **Customer**: user\_id, preferred\_contact\_method, loyalty\_points
3. **Employee**: user\_id, job\_title, salary, hire\_date, shift, role
4. **Repair Appointment**: appointment\_id, customer\_id, employee\_id, method\_id, date\_time, status, total\_price
5. **Repair Service**: service\_id, service\_name, description, price, time\_taken
6. **Repair Service Appointment**: service\_id, appointment\_id
7. **Service Method**: method\_id, method\_name, estimated\_time, cost, note
8. **Payment**: appointment\_id, payment\_number, amount, payment\_status, payment\_method, payment\_date\_time
9. **Device Type**: device\_type\_id, type\_name, description
10. **Brand**: brand\_id, brand\_name, country, founded\_year
11. **Device Type Brand**: device\_type\_id, brand\_id
12. Device Model: model\_id, model\_name, release\_year

## Individual - Student 1

Student 1: last name, first name, matriculation number

### Use Case Definition and Design

State if you are using version 1 (weak entity) or version 2 (IS-A)

#### Textual Description

##### *Title (weak entity/IS-A relationship)*

**Trigger**:

**Preconditions**:

**Main Flow**:

**Postconditions**:

**Entities**:

#### Graphical Representation

<Insert your diagram here>

### Analytics Report

#### Concept

After describing your analytics report, list the three entities involved.

#### Proof of Concept

Add screenshots and descriptions of executing your analytics report.

### NoSQL Design

List entities involved in the use case (use case, not report[!]) in order to improve readability.

#### Design Overview

#### Expected Execution and Possible Changes

#### Five Rules of Thumb

## Individual - Student 2

Student 1: Nikzad, Khalifa, 12437813

### Use Case Definition and Design

I am using version 2 (IS-A).

#### Textual Description

### **Use Case Name:** Book Repair Appointment

### **Trigger**: The customer confirms the repair appointment by clicking the "Confirm Appointment" button.

### **Preconditions:**

1. The customer has selected the device type, brand, and model.
2. The system and database are functioning properly.

### **Main Flow:**

1. **Customer Action:** Selects the repair service and chooses the service method and time slot.
2. **System Action:** Checks if the customer is logged in:
   * If **logged in**, displays stored personal details for review and confirmation.
   * If **not logged in**, prompts the customer:
     + **Has an account:** Directs to login.
     + **Does not have an account:** Asks for personal details and gives options:
       - Create an account or proceed as a guest.
       - If proceeding as a guest, data is processed temporarily, and no account is created.
3. **System Action:** Processes personal details provided, updates the customer record in the database.
4. **Customer Action:** Confirms the appointment.
5. **System Action:** Creates an appointment record and stores all necessary data in the database.
   * Send a confirmation email to the customer.

### **Postconditions:**

1. The customer’s appointment is successfully recorded in the database.
2. A confirmation email is sent to the customer.
3. If the customer opted to create an account, they are now registered in the system.

**Entities**: User (Supertype), Customer (Subtype of User), Repair\_Appointment, Service\_Method, and Repair\_Service

#### Graphical Representation

### Analytics Report

#### Concept

The analytics report provides a comprehensive summary of **completed repair appointments**, capturing detailed insights about customer interactions, repair methods, device details, and services performed. This report is tailored for management, helping them track customer preferences, popular repair services, employee roles, and device trends.

##### Report Objectives

1. **Customer Information:** Displays the full name and contact method of customers who booked repair appointments.
2. **Appointment Details:** Highlights appointment dates, service methods, and repair statuses.
3. **Device Information:** Lists the device type, brand, and specific model linked to the repair.
4. **Repair Service:** Identifies the specific service performed, such as screen replacement or battery replacement.
5. **Employee Details:** Shows the name and role of the employee managing each appointment.

##### Entities Involved in the report:

1. User (Supertype)
2. Customer (Subtype of User)
3. Employee (Subtype of User)
4. Repair Appointment
5. Repair Service
6. Service Method
7. Device Type, Brand, Device Model (Preconditions)

#### Proof of Concept

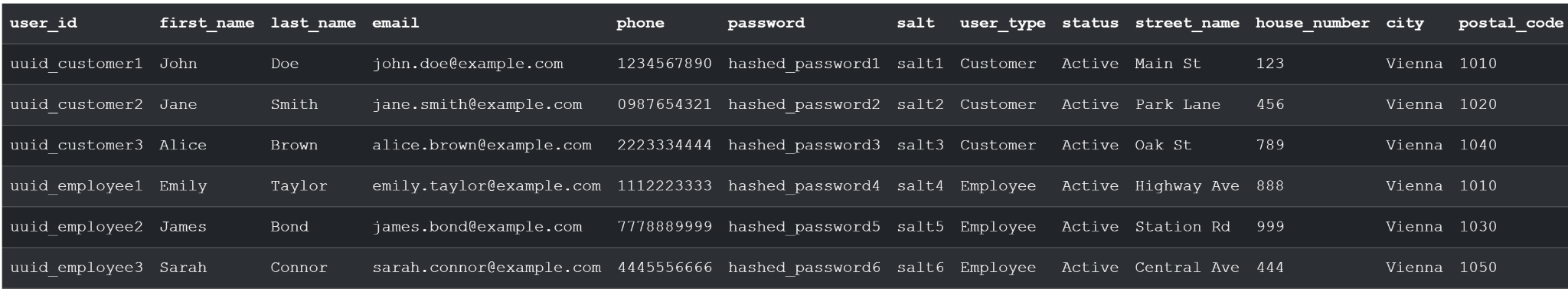
##### 1. Setting Up the Database

The database schema was created to establish relationships between the entities involved in the analytics reprot. The database was established using SQL CREATE TABLE statements, with foreign keys maintaining referential integrity between the entities.

##### 2. Inserting Dummy Data

Dummy data was inserted into the database to simulate real-world scenarios. Key data included:

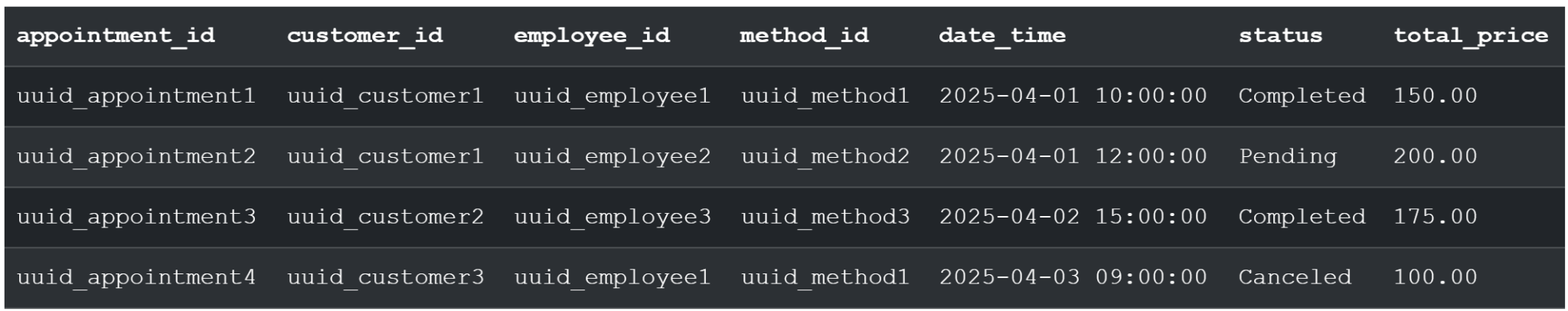
* **Users:** app\_user table entries for customers and employees.



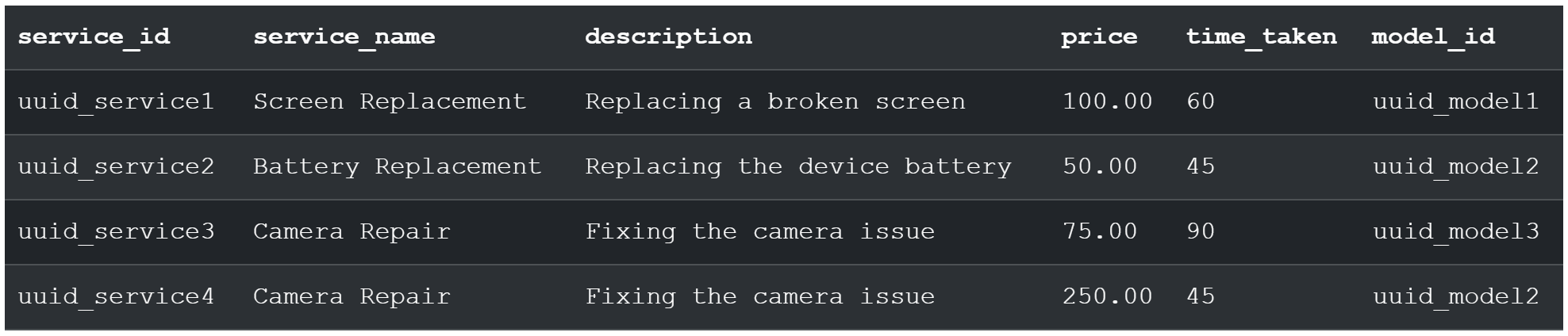
* **Customer:** The subtype of user table

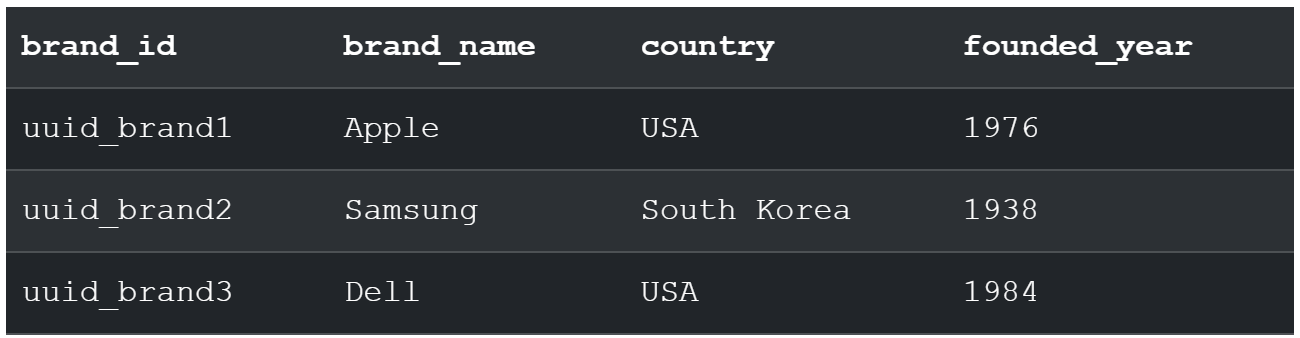
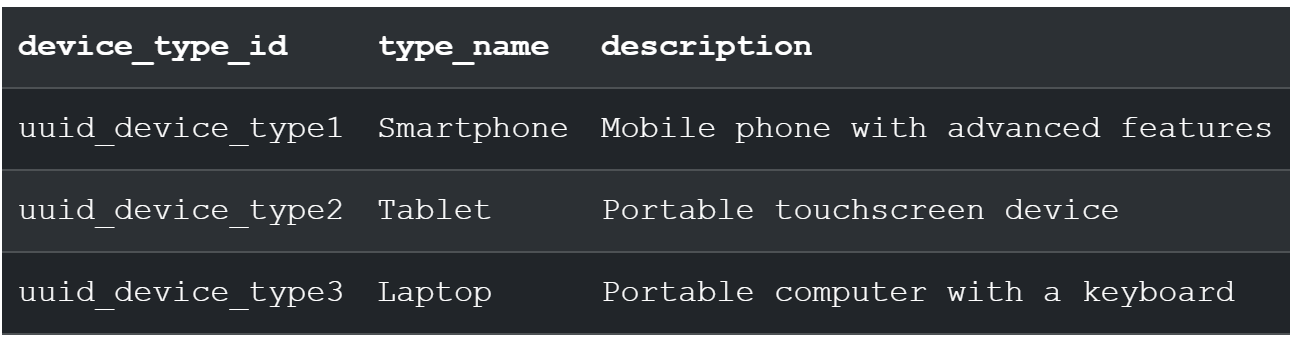
****

* **Appointments:** Completed appointments linking customers, employees, and repair services.



* **Repair Services:** Entries for specific repairs tied to device models (e.g., "Screen Replacement").



* **Devices:** Models such as "iPhone 13" linked to their brands (e.g., Apple) and types (e.g., Smartphone).
  + Device model:
  + Device Brand:
  + Device Type: 

SQL INSERT statements were executed to populate each table. Appointments were carefully linked to customers, employees, and repair services to ensure relational consistency.

##### 3. Executing the Analytics Query

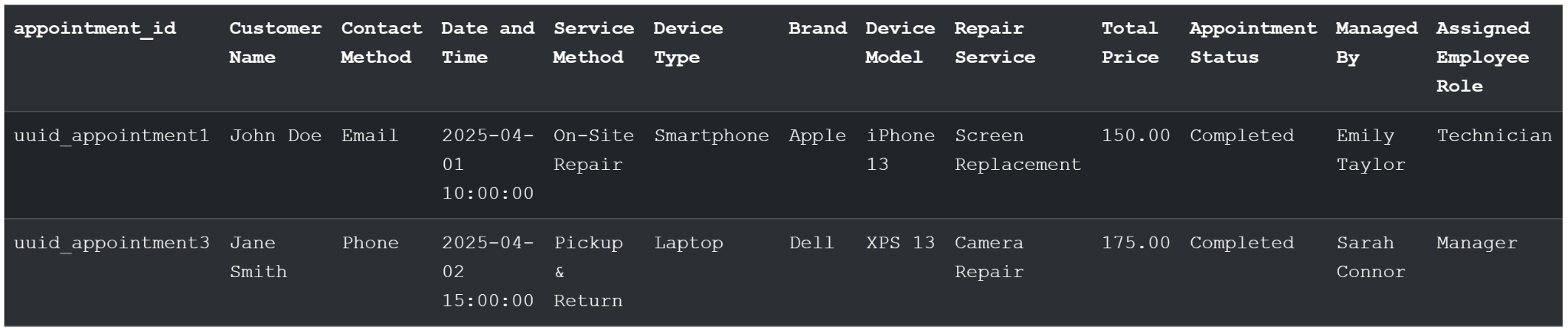
The analytics query brought together data from all involved tables using SQL JOINs. It filtered only **completed repair appointments** and extracted the following details:

1. Customer name and contact preference.
2. Appointment date, time, and total price.
3. Service method used for the repair.
4. Device type, brand, and model being repaired.
5. Repair service performed, such as "Battery Replacement."
6. Employee assigned to manage the appointment.

The query used multiple LEFT JOIN operations to combine information from various tables, leveraging foreign key relationships. A WHERE clause filtered appointments with the status "Completed."

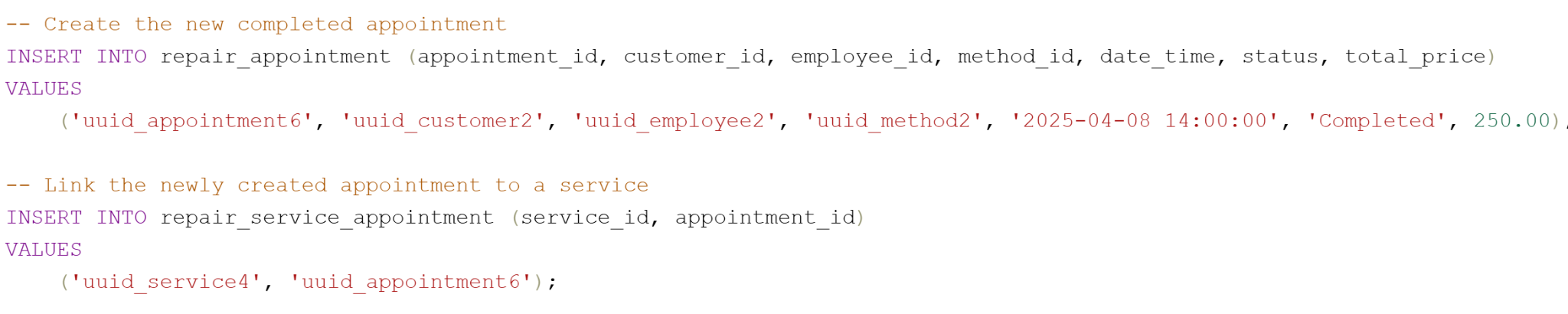
###### Before Simulating the Use Case

The query ran on dummy data, providing details on completed appointments, including customer, device, and repair service information.



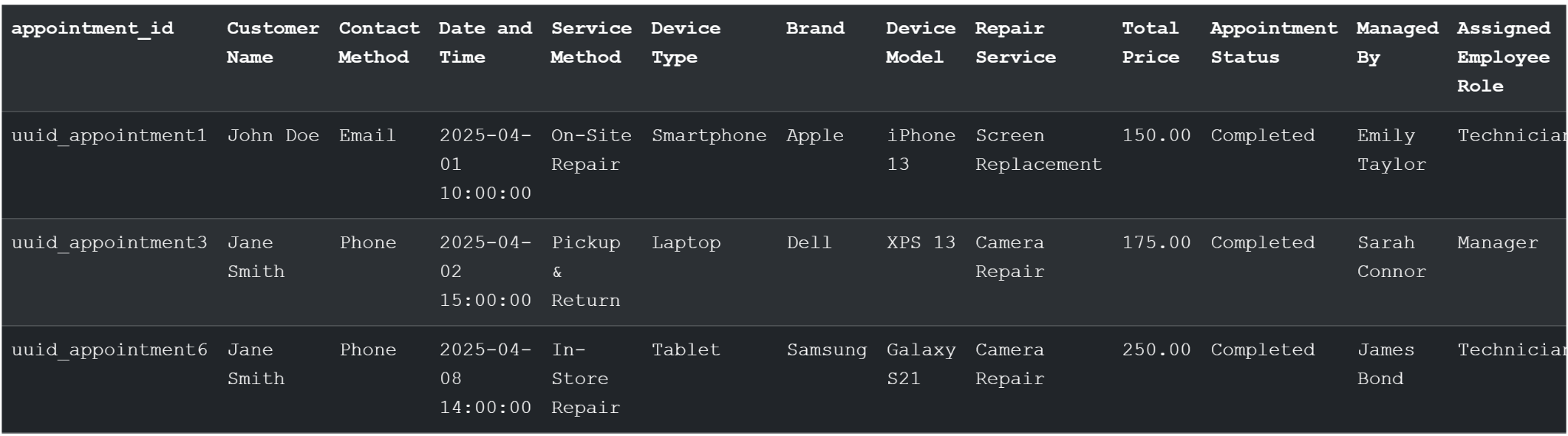
###### Simulating the Use Case:

The "Book Repair Appointment" use case was simulated by inserting a new completed appointment into the database. This included linking the appointment with a repair service, device model, customer, employee, and service method, ensuring realistic data integration for the analytics report.



###### After Simulating the Use Case

After adding a new completed appointment, the query was rerun, reflecting the updated data and successfully integrating the new appointment into the report.



### NoSQL Design

#### Entities Involved in the Use Case

In this design, I use MongoDB to model the following entities that are involved in the “Book Repair Appointment” use case:

* **User** (with Customer as a subtype)
* **Repair\_Appointment**
* **Service\_Method**
* **Repair\_Service**

#### Design Overview

In my NoSQL design, I store a repair appointment as one complete document. This document includes a snapshot of the customer (from the Users collection), the chosen service method, and the repair services selected. This way, all the data needed for an appointment is in one place, making it fast and simple to read. At the same time, I keep separate “master” collections for Users, Service\_Method, and Repair\_Service. These master collections help me update the core data independently of the snapshots stored in the appointment document.

##### Master Collections

###### Users Collection

This collection stores all the common data about a user. For customers, I include extra fields in a subdocument called customerInfo. Here is an example document for a customer in the Users collection:

db.getCollection('users').insertOne({

"\_id": "642f9900f1a1234567890a1",

"first\_name": "Alice",

"last\_name": "Smith",

"email": "alice.smith@example.com",

"phone": "123-456-7890",

"password": "hashedpassword",

"salt": "randomsalt",

"user\_type": "customer",

"status": "active",

"address": {

"street\_name": "Main St",

"house\_number": "100",

"city": "Vienna",

"postal\_code": "1010"

},

"referred\_by": null,

"customerInfo": {

"preferred\_contact\_method": "email",

"loyalty\_points": 150

}

});

###### Service\_Method Collection

This collection stores all the service methods that a customer can choose when booking an appointment. Here is an example for a service method:

db.getCollection('service\_methods').insertOne({

"\_id": "606d20028f1b2c036835b0aa",

"method\_name": "In-Store Dropoff",

"estimated\_time": 60,

"cost": 15,

"note": "Bring warranty documentation if available."

});

###### Repair\_Service Collection

This collection stores the repair services that can be performed. Each repair service is stored as its own document. For example:

db.getCollection('repair\_services').insertOne({

"\_id": "606d204b8f1b2c036835b0ab",

"service\_name": "Screen Replacement",

"description": "Replace a cracked or unresponsive display",

"price": 70,

"time\_taken": 45

});

##### Transactional Collections

###### Repair Appointment Collection

The repair appointment document is a complete record of a booking. I embed a snapshot of the customer, the selected service method, and an array of repair services inside this document. This means that even if the master data in the other collections is updated later, the appointment keeps the original details at the time of booking.

db.getCollection('repair\_appointments').insertOne({

"\_id": "606d1fcf8f1b2c036835b0a9",

"date\_time": "2025-04-15T14:30:00Z",

"status": "confirmed",

"total\_price": 85.50,

"customer": {

"user\_id": "642f9900f1a1234567890a1",

"first\_name": "Alice",

"last\_name": "Smith",

"email": "alice.smith@example.com",

"phone": "123-456-7890",

"address": {

"street\_name": "Main St",

"house\_number": "100",

"city": "Vienna",

"postal\_code": "1010"

},

"customerInfo": {

"preferred\_contact\_method": "email",

"loyalty\_points": 150

}

},

"service\_method": {

"method\_id": "606d20028f1b2c036835b0aa",

"method\_name": "In-Store Dropoff",

"estimated\_time": 60,

"cost": 15,

"note": "Bring warranty documentation if available."

},

"repair\_services": [

{

"service\_id": "606d204b8f1b2c036835b0ab",

"service\_name": "Screen Replacement",

"description": "Replace a cracked or unresponsive display",

"price": 70,

"time\_taken": 45

},

{

"service\_id": "606d205f8f1b2c036835b0ac",

"service\_name": "Diagnostic Check",

"description": "Assessment of overall device issues",

"price": 15,

"time\_taken": 15

}

]

});

#### Expected Execution and Possible Changes

When a customer books a repair appointment, one appointment document is created and stored. This document is read often after booking. Because all necessary data is embedded, the application can quickly show the details without extra queries.

* **If Reads Increase:** I may add indexes on fields like date\_time and status to speed up queries.
* **If Writes Increase:** If the core data such as service method details change frequently, I might consider storing only a reference (ID) in the appointment document instead of a full snapshot. However, for my use case, keeping the snapshot is better because it preserves the details at the time of booking.

#### Five Rules of Thumb

1. **Favor embedding unless there is a compelling reason not to.**

I embed customer data, service method details, and repair services in the appointment document because they are always used together.

1. **The need to access an object on its own is a compelling reason not to embed it.** I keep master collections for Users, Service\_Method, and Repair\_Service so that I can update or manage each entity independently.
2. **High-cardinality arrays are a compelling reason not to embed.**

Since a repair appointment usually includes only a few repair services, it is safe to embed them in an array.

1. **Consider the write/read ratio of a collection/document when denormalizing.** Because an appointment is written once and read many times, embedding helps me get fast reads. If updates were very frequent, I might change the design.
2. **Structure your data to match the ways that your application queries and updates it.**

I have structured the appointment document so that all needed details are in one place, making it easy for my application to retrieve complete booking information quickly and efficiently.