**A Database Fuel Station Network**

**Business Description**

The Fuel Station Network database supports the management of fuel stations across multiple locations. It tracks fuel inventory, sales transactions, employee assignments, and supplier deliveries. Designed for operational efficiency, the system ensures data accuracy, supports quarterly analytics (such as January to March 2025)., and enables informed decision-making through real-time reporting on sales and inventory.

**Key Business Requirements**:

* Store fuel station details (such as name, location).
* Manage fuel types (such as diesel, petrol) with unique names and associated prices.
* Track inventory levels for each fuel type at each station.
* Record customer transactions, including fuel sales, payment methods, and transaction dates.
* Log fuel replenishments, including quantities received and supplier information.
* Maintain employee records, associating them with specific stations.
* Provide quarterly analytics for sales and inventory (such as last 3 months).
* Ensure security with a read-only manager role for reporting purposes.

**Step-by-Step Approach to Modeling**

* 1. **Requirement Analysis**

The first step was to gather and analyze the business requirements for the Fuel Station Network. This involved identifying the core entities, their attributes, and the relationships between them, as well as defining business rules.

* **Entities and Attributes**:
  + **Fuel\_Station**: station\_id, name, location.
  + **Fuel\_Type**: fuel\_type\_id, fuel\_name, price\_per\_liter.
  + **Customer**: customer\_id, first\_name, last\_name, full\_name (generated), contact\_number.
  + **Employee**: employee\_id, full\_name, position, station\_id.
  + **Inventory**: inventory\_id, station\_id, fuel\_type\_id, quantity\_available.
  + **Transaction**: transaction\_id, station\_id, fuel\_type\_id, customer\_id, quantity\_sold, payment\_method, transaction\_date, total\_amount (generated).
  + **Replenishment**: replenishment\_id, station\_id, fuel\_type\_id, quantity\_received, supplier\_name, delivery\_date.
* **Relationships**:
  + Many-to-Many (M:N) between Fuel\_Station and Fuel\_Type via bridge table Inventory.
  + One-to-Many (1:N) from Fuel\_Station to Transaction, Replenishment, and Employee.
  + One-to-Many (1:N) from Customer to Transaction.
* **Business Rules**:
  + Fuel names must be unique (such as no duplicate "Petrol").
  + Quantities (available, sold, received) must be non-negative.
  + Transactions and replenishments must occur after January 1, 2024.
  + Payment methods are restricted to 'cash', 'card', or 'mobile'.
  1. **Conceptual Design**

At this stage, the model does **not include attributes, data types, or constraints**. Instead, it focuses on the **structural relationships** between entities to represent the system at a high level. This abstraction allows stakeholders to focus on business structure and logic before technical implementation details.

* **Entities:**

The conceptual model defines seven core entities: **Fuel\_Station**, **Fuel\_Type**, **Customer**, **Employee**, **Inventory**, **Transaction**, and **Replenishment**.

* **Relationships**:
  + **Fuel\_Station → Employee** (1:N)
  + **Fuel\_Station → Transaction** (1:N)
  + **Fuel\_Station → Replenishment** (1:N)
  + **Customer → Transaction** (1:N)
  + **Fuel\_Station ↔ Fuel\_Type** (M:N via Inventory)
* **Tool**: Used dbdiagram.io to generate the Conceptual ERD.
* **Output**: SQL\_Cristian\_Luca\_FinalTask\_FuelStation\_ConceptualModel.png.
  1. **Logical Design**

The logical design translated the Conceptual ERD into a detailed relational schema in 3rd Normal Form, specifying data types, keys, and constraints.

* **Tables and Attributes**:
  + All 7 entities were converted into tables with appropriate attributes.
  + Data types were assigned (such as SERIAL for IDs, DECIMAL(10,2) for quantities, VARCHAR for names).
* **Constraints**:
  + **Primary Keys**: Used SERIAL for auto-incrementing IDs (such as station\_id, transaction\_id).
  + **Foreign Keys**: Defined for relationships (such as Transaction.station\_id references Fuel\_Station.station\_id).
  + **UNIQUE**: Applied to Fuel\_Type.fuel\_name to prevent duplicates.
  + **CHECK Constraints**:
    - Inventory.quantity\_available: CHECK (quantity\_available >= 0).
    - Transaction.quantity\_sold: CHECK (quantity\_sold > 0).
    - Transaction.payment\_method: CHECK (payment\_method IN ('cash', 'card', 'mobile')).
    - Transaction.transaction\_date: CHECK (transaction\_date > '2024-01-01').
    - Replenishment.quantity\_received: CHECK (quantity\_received > 0).
    - Replenishment.delivery\_date: CHECK (delivery\_date > '2024-01-01').
    - Fuel\_Type.price\_per\_liter: CHECK (price\_per\_liter > 0).
  + **GENERATED Columns**:
    - Customer.full\_name: GENERATED ALWAYS AS (CONCAT(first\_name, ' ', last\_name)) STORED.
    - Transaction.total\_amount: GENERATED ALWAYS AS (quantity\_sold \* (SELECT price\_per\_liter FROM Fuel\_Type WHERE fuel\_type\_id = Transaction.fuel\_type\_id)) STORED.
  1. **Normalization Process:**

**1NF**:

* + Ensured all attributes are atomic (such as no lists like "Petrol, Diesel" in a single column).
  + Eliminated repeating groups by using the Inventory table for the M:N relationship between Fuel\_Station and Fuel\_Type.
  + Defined primary keys for all tables (e.g., station\_id, transaction\_id).

**2NF**:

* + Confirmed that all tables with single-column primary keys (such as Fuel\_Station, Transaction) are automatically in 2NF.
  + For Inventory, which could have a composite key (station\_id, fuel\_type\_id), verified that quantity\_available depends on the entire primary key, not just part of it. With the surrogate key (inventory\_id), 2NF is automatically satisfied.

**3NF**:

* + Eliminated transitive dependencies by:
    - Using generated columns for derived attributes (Customer.full\_name, Transaction.total\_amount).
    - Storing foreign keys instead of redundant data (such as Employee.station\_id instead of station name, Transaction.fuel\_type\_id instead of fuel price).
  + Ensured all non-key attributes depend directly on the primary key and not on other non-key attributes.

**Ensured the schema is in 3NF :**

* **Customer**:
  + full\_name is generated from first\_name and last\_name, avoiding a transitive dependency.
  + All other attributes (first\_name, last\_name, contact\_number) depend directly on customer\_id.
* **Transaction**:
  + total\_amount is generated from quantity\_sold and Fuel\_Type.price\_per\_liter, avoiding a transitive dependency.
  + All other attributes (station\_id, fuel\_type\_id, customer\_id, quantity\_sold, payment\_method, transaction\_date) depend directly on transaction\_id.
* **Employee**:
  + Stores station\_id as a foreign key, avoiding transitive dependencies with station-related data (such as station name, location).
  + All attributes (full\_name, position, station\_id) depend directly on employee\_id.
* **Tool**: Used dbdiagram.io to generate the Logical ERD.
* **Output**:

SQL\_Cristian\_Luca\_FinalTask\_FuelStation\_LogicalModel.png.

*Note that dbdiagram.io does not visually display UNIQUE or CHECK constraints in the PNG format!*

*Table Fuel\_Station {*

*station\_id serial [pk]*

*name varchar(100) [not null]*

*location varchar(150) [not null]*

*}*

*Table Fuel\_Type {*

*fuel\_type\_id serial [pk]*

*fuel\_name (UNIQUE) varchar(50) [not null]*

*price\_per\_liter (CHECK > 0) decimal(10,2) [not null]*

*}*

*Table Customer {*

*customer\_id serial [pk]*

*first\_name varchar(50) [not null]*

*last\_name varchar(50) [not null]*

*full\_name (GENERATED) varchar(100) [not null, note: “GENERATED ALWAYS AS (CONCAT(first\_name, \\, last\_name)) STORED”]*

*contact\_number varchar(20)*

*}*

*Table Employee {*

*employee\_id serial [pk]*

*full\_name varchar(100) [not null]*

*position varchar(50) [not null]*

*station\_id int [not null]*

*}*

*Table Inventory {*

*inventory\_id serial [pk]*

*station\_id int [not null]*

*fuel\_type\_id int [not null]*

*quantity\_available (CHECK >= 0) decimal(10,2) [not null]*

*}*

*Table Transaction {*

*transaction\_id serial [pk]*

*station\_id int [not null]*

*fuel\_type\_id int [not null]*

*customer\_id int [not null]*

*transaction\_date (CHECK > 2024-01-01) date [not null]*

*quantity\_sold (CHECK > 0) decimal(10,2) [not null]*

*payment\_method (CHECK IN (cash,card,mobile)) varchar(20) [not null]*

*total\_amount (GENERATED) decimal(10,2) [not null, note: “GENERATED ALWAYS AS (quantity\_sold \* (SELECT price\_per\_liter FROM Fuel\_Type WHERE fuel\_type\_id = Transaction.fuel\_type\_id)) STORED”]*

*}*

*Table Replenishment {*

*replenishment\_id serial [pk]*

*station\_id int [not null]*

*fuel\_type\_id int [not null]*

*delivery\_date (CHECK > 2024-01-01) date [not null]*

*quantity\_received (CHECK > 0) decimal(10,2) [not null]*

*supplier\_name varchar(100) [not null]*

*}*

*Ref: Employee.station\_id > Fuel\_Station.station\_id*

*Ref: Inventory.station\_id > Fuel\_Station.station\_id*

*Ref: Inventory.fuel\_type\_id > Fuel\_Type.fuel\_type\_id*

*Ref: Transaction.station\_id > Fuel\_Station.station\_id*

*Ref: Transaction.fuel\_type\_id > Fuel\_Type.fuel\_type\_id*

*Ref: Transaction.customer\_id > Customer.customer\_id*

*Ref: Replenishment.station\_id > Fuel\_Station.station\_id*

*Ref: Replenishment.fuel\_type\_id > Fuel\_Type.fuel\_type\_id*

**Ultima versiune**

**A Database Fuel Station Network**

**Business Description**

The Fuel Station Network database supports the management of fuel stations across multiple locations. It tracks fuel inventory, sales transactions, employee assignments, and supplier deliveries. Designed for operational efficiency, the system ensures data accuracy, supports quarterly analytics (such as January to March 2025), and enables informed decision-making through real-time reporting on sales and inventory.

**Key Business Requirements**

* Store fuel station details (such as name, location).
* Manage fuel types (such as diesel, petrol) with unique names and associated prices.
* Track inventory levels for each fuel type at each station.
* Record customer transactions, including fuel sales, payment methods, and transaction dates.
* Log fuel replenishments, including quantities received and supplier information.
* Maintain employee records, associating them with specific stations.
* Provide quarterly analytics for sales and inventory (such as last 3 months).
* Ensure security with a read-only manager role for reporting purposes.

**Step-by-Step Approach to Modeling**

**1. Requirement Analysis**

The first step was to analyze the business goals, identify the core entities, and define the relationships between them.

**Key Entities:**

* Fuel Station
* Fuel Type
* Customer
* Employee
* Inventory
* Transaction
* Replenishment

**Core Relationships:**

* **Fuel Station has many Employees**
* **Fuel Station has many Transactions and Replenishments**
* **Customer makes many Transactions**
* **Fuel Station and Fuel Type** have a **many-to-many** relationship (handled via Inventory)

**Business Rules:**

* Fuel names must be unique
* Quantities (available, sold, received) must be non-negative
* Transactions and replenishments must occur after January 1, 2024
* Payment methods must be one of: 'cash', 'card', or 'mobile'

**2. Conceptual Design**

This high-level Entity-Relationship Diagram (ERD) shows **only the entities and their relationships**, with no attribute or implementation detail.

* Entities: Fuel Station, Fuel Type, Customer, Employee, Inventory, Transaction, Replenishment
* Relationships:
  + Fuel Station → Employee (1:N)
  + Fuel Station → Transaction (1:N)
  + Fuel Station → Replenishment (1:N)
  + Customer → Transaction (1:N)
  + Fuel Station ↔ Fuel Type (M:N via Inventory)

**Note:** Attributes, data types, and constraints are excluded at this stage to focus on system structure and interactions.

**Tool Used:** dbdiagram.io  
**Output:** SQL\_Cristian\_Luca\_FinalTask\_FuelStation\_ConceptModel\_UPDATED.png

**3. Logical Design**

This stage translates the conceptual model into a relational schema using third normal form (3NF). It includes attribute definitions, data types, and integrity constraints.

**Tables:**

All seven entities were converted into normalized tables with the following features:

* SERIAL surrogate keys for uniqueness
* VARCHAR, DATE, and DECIMAL used appropriately
* Foreign key relationships reflecting business logic

**Constraints:**

* **Primary Keys:** Unique IDs for each entity
* **Foreign Keys:** Enforced for proper relational mapping
* **UNIQUE:** Enforced for fuel\_name
* **CHECK:** Applied to ensure valid quantities, payment methods, and dates
* **GENERATED:** For calculated fields (Customer.full\_name, Transaction.total\_amount)

**Normalization Summary:**

* **1NF:** Atomic fields, no repeating groups
* **2NF:** All non-key attributes depend on entire primary key
* **3NF:** No transitive dependencies — derived fields are generated or moved

**Tool Used:** dbdiagram.io  
**Output:** SQL\_Cristian\_Luca\_FinalTask\_FuelStation\_LogicalModel.png

**4. Physical Design**

The physical model specifies how the database is implemented, including foreign keys and relationships in the schema.

**Relationships (with foreign key enforcement):**

* Employee.station\_id → Fuel\_Station.station\_id
* Inventory.station\_id → Fuel\_Station.station\_id
* Inventory.fuel\_type\_id → Fuel\_Type.fuel\_type\_id
* Transaction.station\_id → Fuel\_Station.station\_id
* Transaction.fuel\_type\_id → Fuel\_Type.fuel\_type\_id
* Transaction.customer\_id → Customer.customer\_id
* Replenishment.station\_id → Fuel\_Station.station\_id
* Replenishment.fuel\_type\_id → Fuel\_Type.fuel\_type\_id

**Correction:**

* Arrows in the physical model now show correct **1:N** and **M:N** directions
* **Foreign keys are displayed** visually for clarity

**Tool Used:** dbdiagram.io  
**Output:** SQL\_Cristian\_Luca\_FinalTask\_FuelStation\_PhysicalModel\_UPDATED.png

**Note:** dbdiagram.io does not visually display CHECK or UNIQUE constraints in exported PNG diagrams — these are handled in SQL.

**SQL Summary (from Scripts)**

Your SQL script implements all the business rules and normalization practices correctly:

* Proper surrogate keys (SERIAL)
* Foreign keys enforced
* CHECK constraints for data validation
* GENERATED ALWAYS columns used for calculated fields
* Use of surrogate keys to avoid composite primary keys

**No changes required** — matches the physical model precisely.