

Mini Project Report

of

**Database Systems Lab (CSE 2262)**

ROS Bag Manager

Presented

BY

Rudra Patel : **210905324**

Section A : **52**

Department of Computer Science and Engineering

**Manipal Institute of Technology, Manipal.**

**April 2023**

****

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

CERTIFICATE

This is to certify that the project titled **StockPile : A Comprehensive Inventory Management System** is a record of the bonafide work done by Rudra Patel (210905324) submitted in partial fulfilment of the requirements for the award of the Degree of Bachelor of Technology (B.Tech.) in COMPUTER SCIENCE & ENGINEERING of Manipal Institute of Technology, Manipal, Karnataka, (A Constituent Institute of Manipal Academy of Higher Education), during the academic year 2022-2023.

Name and Signature of Examiners:

1. Dr. Anup Bhat B., AssociateProfessor, CSE Dept

TABLE OF CONTENTS

ABSTRACT

**CHAPTER 1: INTRODUCTION**

**CHAPTER 2: PROBLEM STATEMENT & OBJECTIVES**

**CHAPTER 3: METHODOLOGY**

**CHAPTER 4: RESULTS & SNAPSHOTS**

**CHAPTER 5: CONCLUSION**

**CHATER 6: LIMITATIONS & FUTURE WORK**

**CHAPTER 7: REFERENCES**

**Introduction**

Stockpile is a database management system designed to provide an efficient and comprehensive solution for managing inventory. The system is implemented using POSTGRES and provides a relational database that allows users to store and retrieve information about inventory items, suppliers, customers, orders, and sales. The main goal of this project is to streamline the inventory management process by providing a user-friendly interface for managing inventory data, generating reports, and automating common inventory-related tasks. With the help of Stockpile, businesses can improve their inventory management efficiency, reduce costs, and boost profitability.

**Problem Statement and Objectives**

Inefficient inventory management can lead to increased costs and reduced profitability for businesses. Without a proper system in place, inventory data can be disorganized and difficult to manage.

The aim of this project is to provide a comprehensive inventory management system Stockpile, which automates inventory-related tasks, improves data accuracy, and enables businesses to make informed decisions about their inventory.

 Stockpile aims to have the following features:

* Inventory tracking: track all inventory items, including stock level and location.
* Automated inventory management: generate purchase orders and update stock levels automatically.
* Reporting and analytics: provide detailed reports and analytics on inventory levels, sales, and other metrics.
* Customer data tracking: track customer data, including purchase history and contact information.

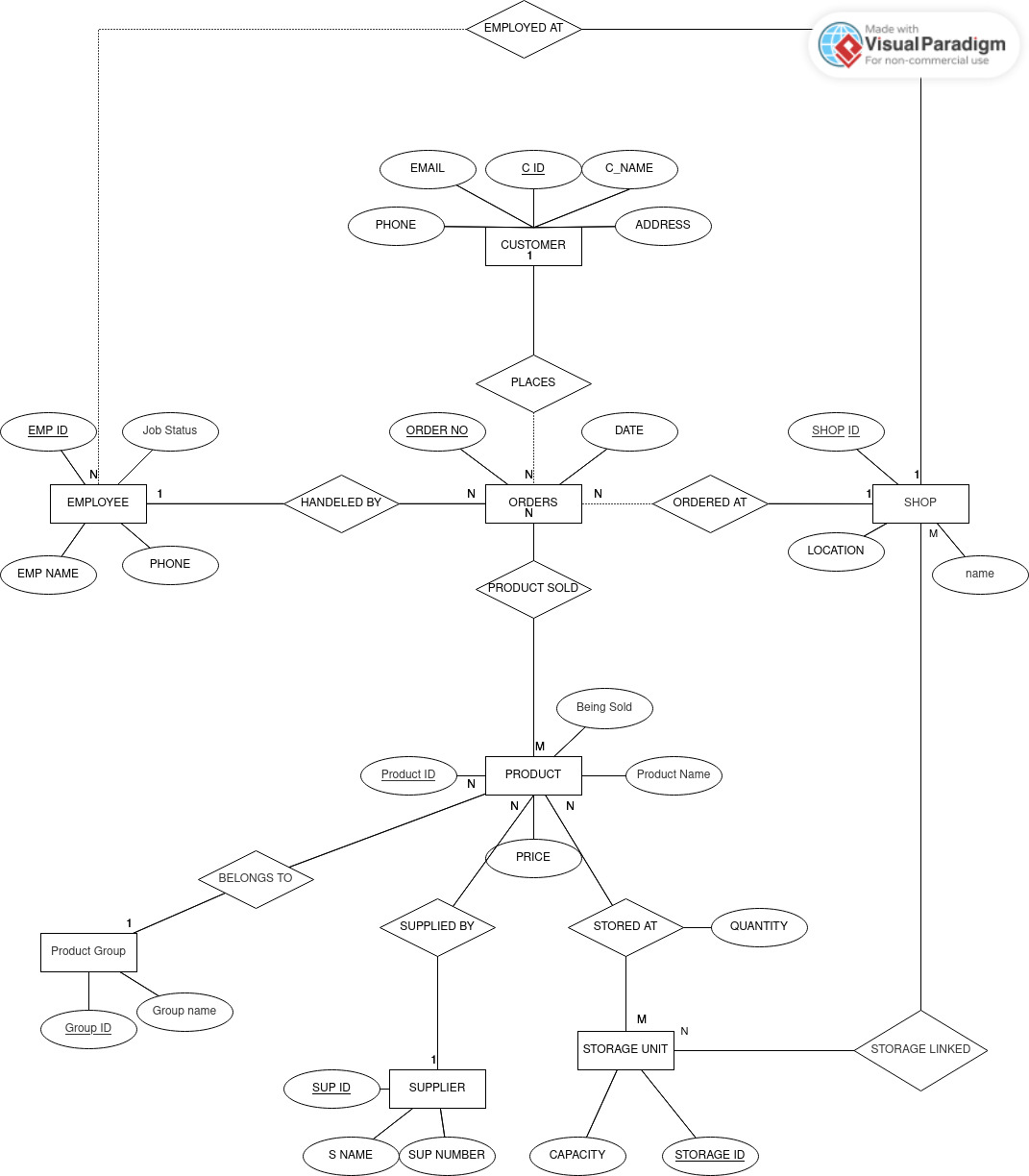
**Methodology**

* The first step was to identify the constraints and requirements for this manager.
* Then we identified our entities such as Products, Suplier, Shop etc.
* After which we constructed the ER diagram.
* Using the ER diagram, we reduced it to a relational schema.

Assumptions:

* A product can belong to multiple storage units
* A storage unit can belong to multiple shops
* A shop can have multiple storage units linked to it.
* When ever a product is sold its taken from the storageunit linked to that shop with the most amout of that product.
* A suplier can only suply one product

**ER Diagram**:



**ER to Relational Schema**

**Step1:** Identify strong entities.

**CUSTOMER** (CUSTOMER\_ID, C\_NAME, ADDRESS, PHONE, EMAIL)

**ORDERS** (ORDER\_ID, CREATED\_AT)

**EMPLOYEE** (EMPLOYEE\_ID, EMPLOYEE\_NAME, JOB\_STATUS,PHONE)

**SHOP** (SHOP\_ID, NAME, LOCATION)

**STORAGE UNIT** (STORAGE\_ID, UNIT KEY, CAPACITY)

**PRODUCT** (PRODUCT\_ID, NAME,BEING SOLD)

**PRODUCT GROUP** (GROUP\_ID, GROUP NAME)

**Step2:** Identify weak entities.

**No weak entities.**

**Step3:** Identify relationships.

**STORAGELINKED** (SHOPID, STORAGEID)

**STOREDAT** (PRODUCTID, STORAGEID, QUANTITY)

**PRODUCTSOLD** (ORDERID, PRODUCTID, QUANTITY)

**Step4:** Merging relations with entities

**employee**(id,name,phone,created\_at,jobStatus,branch\_id)

**orders**(id,created\_at,branch\_id,customer\_id,emp\_id)

**products**(id,photo,name,price,created\_at,productsBeingSold,groupId,supId)

**Normalization**

In the database used each table has a primary key, and all other columns in the table are functionally dependent on that primary key or dont have any dependencies at all. Therefore, each table is in Boyce-Codd Normal Form (BCNF), and normalization is not needed.

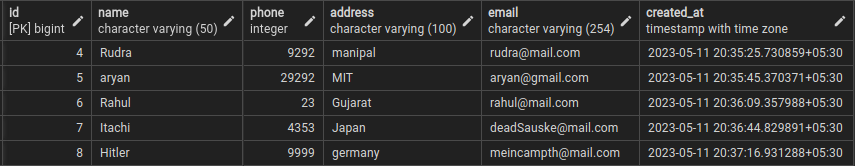
To be more specific:

* ProductsGroup table has a primary key (id), and name column is functionally dependent on that primary key.
* Suplier table has a primary key (id), and name and phone columns are functionally dependent on that primary key.
* Products table has a primary key (id), and all other columns (photo, group, name, price, supId, created\_at, updated\_at, and productBeingSold) are functionally dependent on that primary key.
* Shop table has a primary key (id), and name, location, created\_at, and updated\_at columns are functionally dependent on that primary key.
* Customer table has a primary key (id), and name, phone, address, email, and created\_at columns are functionally dependent on that primary key.
* Employee table has a primary key (id), and name, phone, branchId, created\_at, and jobStatus columns are functionally dependent on that primary key.
* Orders table has a primary key (id), and customerId, branchId, empId, and created\_at columns are functionally dependent on that primary key.
* Invoice table has a primary key (id), and orderId, amount, and products columns are functionally dependent on that primary key.
* ProductSold table has a composite primary key (productId, orderId), and quantity and created\_at columns are functionally dependent on that composite primary key.
* StorageUnit table has a primary key (id), and unitKey and capacity columns are functionally dependent on that primary key.
* StoredAt table has a composite primary key (productId, storageId), and quantity column is functionally dependent on that composite primary key.
* StorageLinked table has a composite primary key (storageId, ShopId), and no other columns are present.

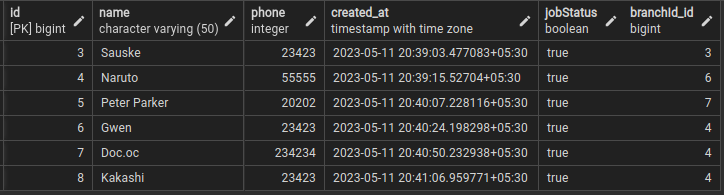
Therefore, every table in this database is in BCNF.

**Relational Tables with sample data**

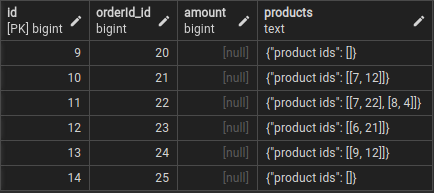
Customer:



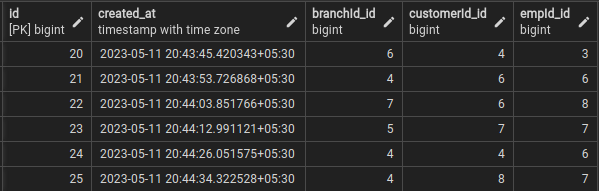
Employee:



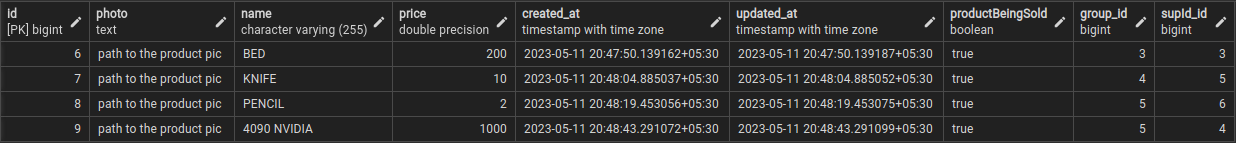
Invoices:



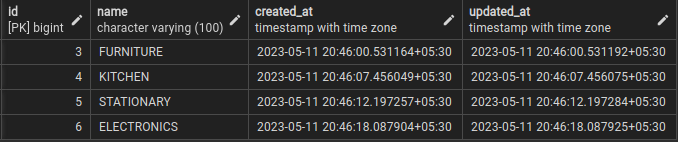
Orders:



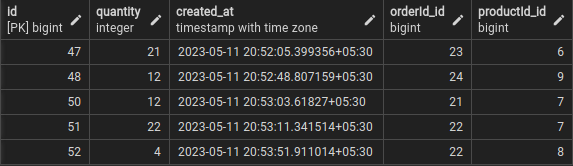
Products:



Product Group:



Products Sold:



## Shop:

## 

## Storage Linked:

## 

## Stored At:

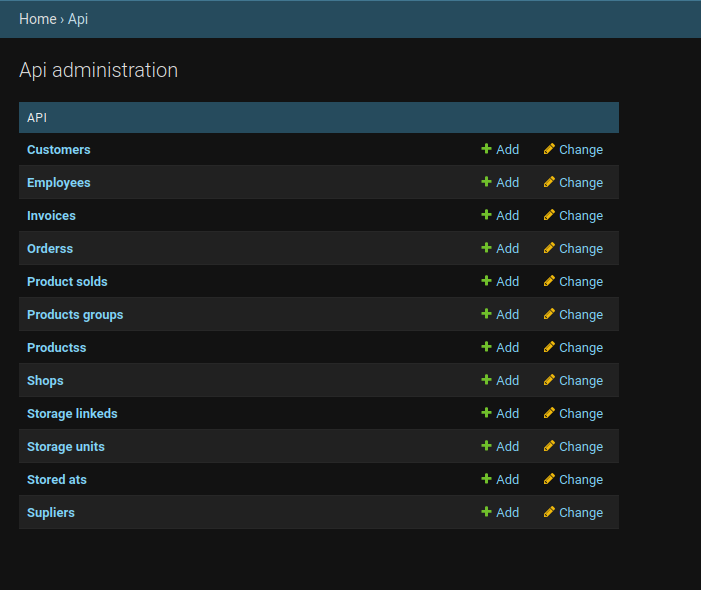
## 

## Supliers:

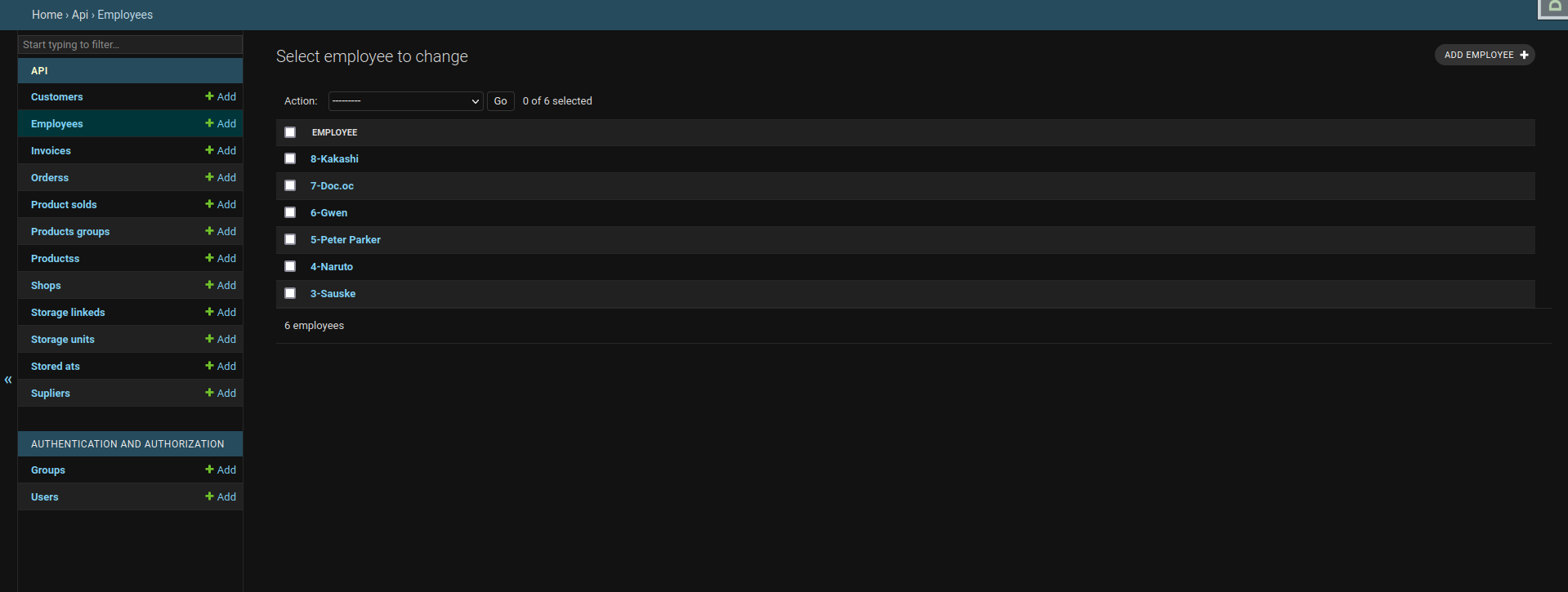
## 

UI :

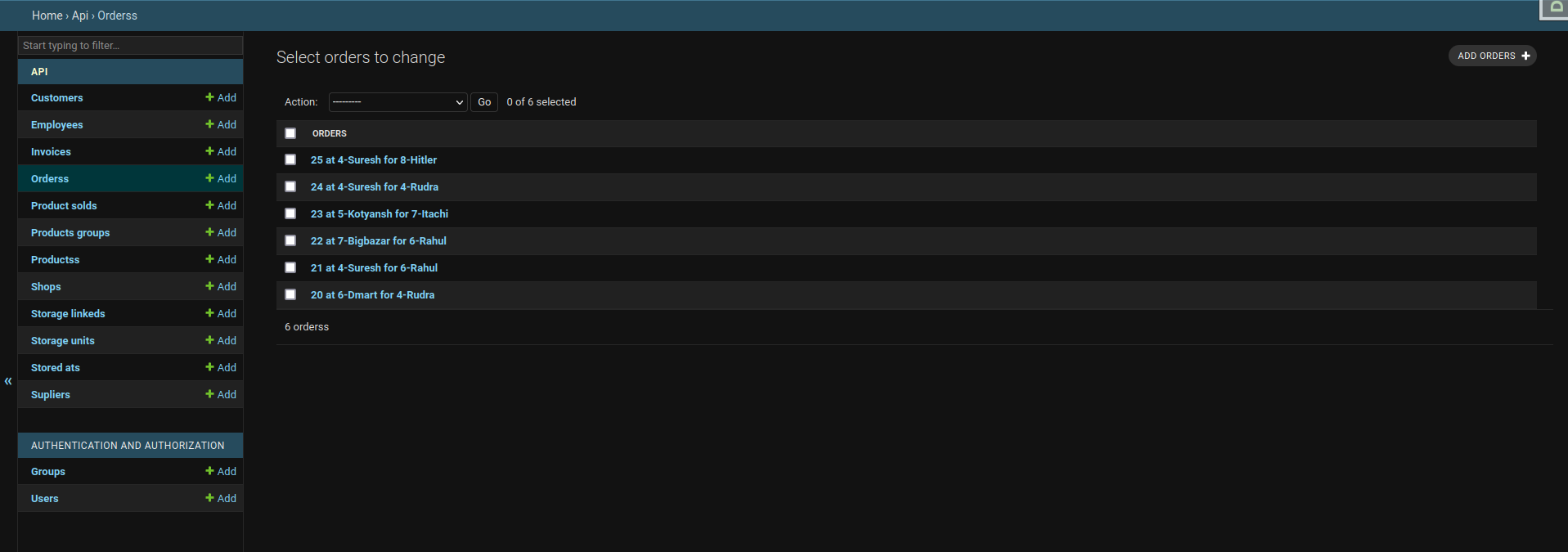
ADMIN PAGE:



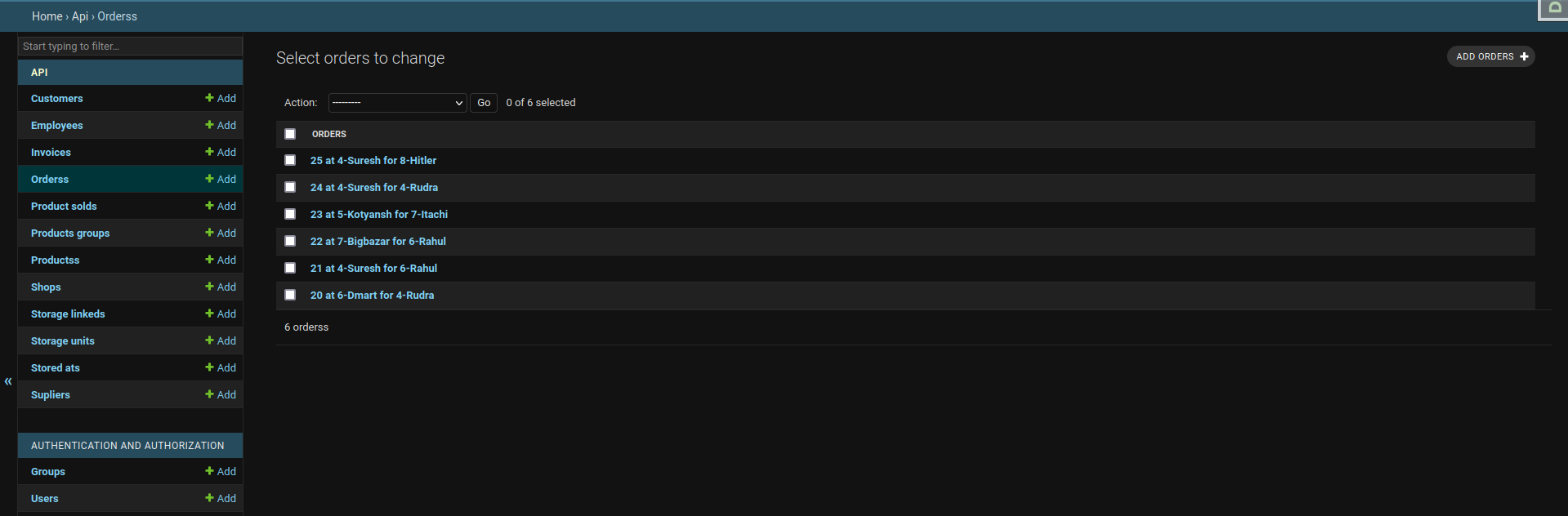
EMPLOYEE :



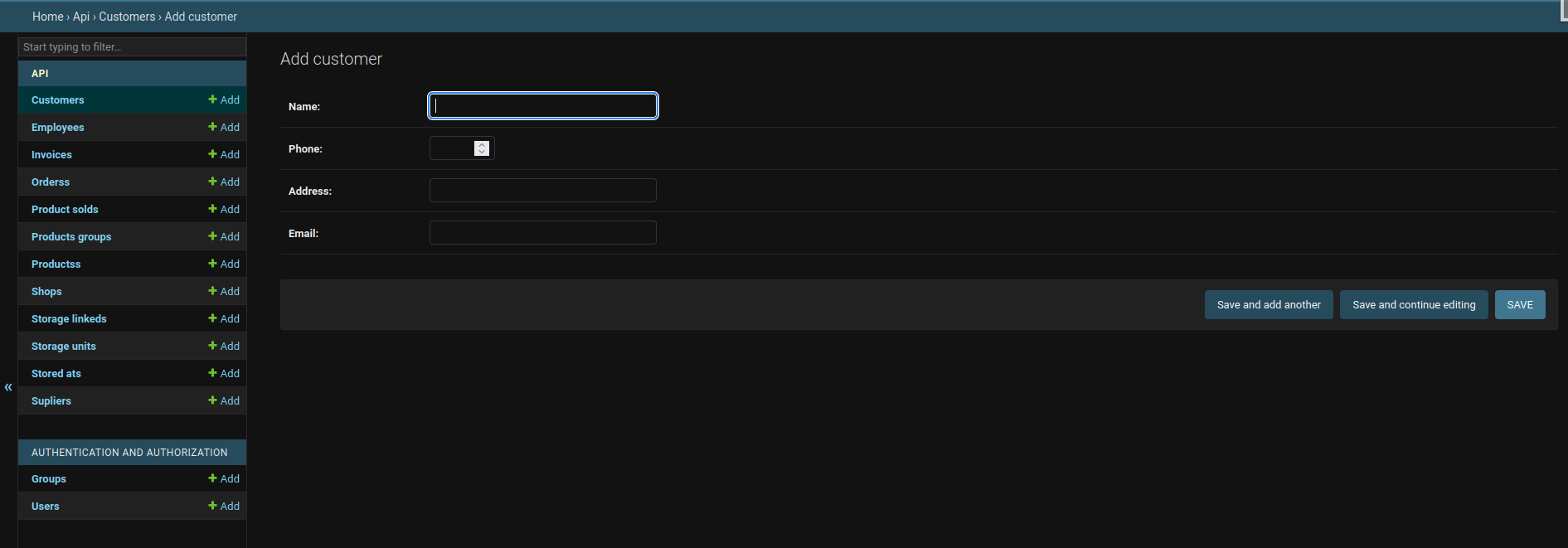
ORDERS :



Products Sold :



Adding Customer:



**DDL COMMANDS:**

CREATE TABLE IF NOT EXISTS public.api\_customer

(

id bigint NOT NULL DEFAULT nextval('api\_customer\_id\_seq'::regclass),

name character varying(50) COLLATE pg\_catalog."default" NOT NULL,

phone integer NOT NULL,

address character varying(100) COLLATE pg\_catalog."default" NOT NULL,

email character varying(254) COLLATE pg\_catalog."default" NOT NULL,

created\_at timestamp with time zone NOT NULL,

CONSTRAINT api\_customer\_pkey PRIMARY KEY (id),

CONSTRAINT api\_customer\_name\_key UNIQUE (name),

CONSTRAINT api\_customer\_phone\_check CHECK (phone >= 0)

)

CREATE TABLE IF NOT EXISTS public.api\_employee

(

id bigint NOT NULL DEFAULT nextval('api\_employee\_id\_seq'::regclass),

name character varying(50) COLLATE pg\_catalog."default" NOT NULL,

phone integer NOT NULL,

created\_at timestamp with time zone NOT NULL,

"jobStatus" boolean NOT NULL,

"branchId\_id" bigint,

CONSTRAINT api\_employee\_pkey PRIMARY KEY (id),

CONSTRAINT api\_employee\_name\_key UNIQUE (name),

CONSTRAINT "api\_employee\_branchId\_id\_487f0a03\_fk" FOREIGN KEY ("branchId\_id")

REFERENCES public.api\_shop (id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION

DEFERRABLE INITIALLY DEFERRED,

CONSTRAINT api\_employee\_phone\_check CHECK (phone >= 0)

)

CREATE TABLE IF NOT EXISTS public.api\_invoice

(

id bigint NOT NULL DEFAULT nextval('api\_invoice\_id\_seq'::regclass),

"orderId\_id" bigint NOT NULL,

amount bigint,

products text COLLATE pg\_catalog."default",

CONSTRAINT api\_invoice\_pkey PRIMARY KEY (id),

CONSTRAINT "api\_invoice\_orderId\_id\_43d1731f\_fk\_api\_orders\_id" FOREIGN KEY ("orderId\_id")

REFERENCES public.api\_orders (id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION

DEFERRABLE INITIALLY DEFERRED

)

CREATE TABLE IF NOT EXISTS public.api\_orders

(

id bigint NOT NULL DEFAULT nextval('api\_orders\_id\_seq'::regclass),

created\_at timestamp with time zone NOT NULL,

"branchId\_id" bigint,

"customerId\_id" bigint,

"empId\_id" bigint,

CONSTRAINT api\_orders\_pkey PRIMARY KEY (id),

CONSTRAINT "api\_orders\_branchId\_id\_6ce32572\_fk" FOREIGN KEY ("branchId\_id")

REFERENCES public.api\_shop (id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION

DEFERRABLE INITIALLY DEFERRED,

CONSTRAINT "api\_orders\_customerId\_id\_e18f35aa\_fk\_api\_customer\_id" FOREIGN KEY ("customerId\_id")

REFERENCES public.api\_customer (id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION

DEFERRABLE INITIALLY DEFERRED,

CONSTRAINT "api\_orders\_empId\_id\_3c4bfbda\_fk\_api\_employee\_id" FOREIGN KEY ("empId\_id")

REFERENCES public.api\_employee (id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION

DEFERRABLE INITIALLY DEFERRED

)

CREATE TABLE IF NOT EXISTS public.api\_products

(

id bigint NOT NULL DEFAULT nextval('api\_products\_id\_seq'::regclass),

photo text COLLATE pg\_catalog."default",

name character varying(255) COLLATE pg\_catalog."default" NOT NULL,

price double precision NOT NULL,

created\_at timestamp with time zone NOT NULL,

updated\_at timestamp with time zone NOT NULL,

"productBeingSold" boolean NOT NULL,

group\_id bigint,

"supId\_id" bigint,

CONSTRAINT api\_products\_pkey PRIMARY KEY (id),

CONSTRAINT api\_products\_group\_id\_9afd0da8\_fk\_api\_productsgroup\_id FOREIGN KEY (group\_id)

REFERENCES public.api\_productsgroup (id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION

DEFERRABLE INITIALLY DEFERRED,

CONSTRAINT "api\_products\_supId\_id\_eb592123\_fk\_api\_suplier\_id" FOREIGN KEY ("supId\_id")

REFERENCES public.api\_suplier (id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION

DEFERRABLE INITIALLY DEFERRED

)

CREATE TABLE IF NOT EXISTS public.api\_productsgroup

(

id bigint NOT NULL DEFAULT nextval('api\_productsgroup\_id\_seq'::regclass),

name character varying(100) COLLATE pg\_catalog."default" NOT NULL,

created\_at timestamp with time zone NOT NULL,

updated\_at timestamp with time zone NOT NULL,

CONSTRAINT api\_productsgroup\_pkey PRIMARY KEY (id),

CONSTRAINT api\_productsgroup\_name\_key UNIQUE (name)

)

CREATE TABLE IF NOT EXISTS public.api\_productsold

(

id bigint NOT NULL DEFAULT nextval('api\_productsold\_id\_seq'::regclass),

quantity integer NOT NULL,

created\_at timestamp with time zone NOT NULL,

"orderId\_id" bigint NOT NULL,

"productId\_id" bigint NOT NULL,

CONSTRAINT api\_productsold\_pkey PRIMARY KEY (id),

CONSTRAINT "api\_productsold\_productId\_id\_orderId\_id\_cfa75181\_uniq" UNIQUE ("productId\_id", "orderId\_id"),

CONSTRAINT "api\_productsold\_orderId\_id\_7368b26a\_fk\_api\_orders\_id" FOREIGN KEY ("orderId\_id")

REFERENCES public.api\_orders (id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION

DEFERRABLE INITIALLY DEFERRED,

CONSTRAINT "api\_productsold\_productId\_id\_adc34ff7\_fk\_api\_products\_id" FOREIGN KEY ("productId\_id")

REFERENCES public.api\_products (id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION

DEFERRABLE INITIALLY DEFERRED,

CONSTRAINT api\_productsold\_quantity\_check CHECK (quantity >= 0)

)

CREATE TABLE IF NOT EXISTS public.api\_shop

(

id bigint NOT NULL DEFAULT nextval('api\_shop\_id\_seq'::regclass),

name character varying(50) COLLATE pg\_catalog."default" NOT NULL,

location character varying(50) COLLATE pg\_catalog."default" NOT NULL,

created\_at timestamp with time zone NOT NULL,

updated\_at timestamp with time zone NOT NULL,

CONSTRAINT api\_shop\_pkey PRIMARY KEY (id),

CONSTRAINT api\_shop\_name\_05fa0f8b\_uniq UNIQUE (name)

)

CREATE TABLE IF NOT EXISTS public.api\_storagelinked

(

id bigint NOT NULL DEFAULT nextval('api\_storagelinked\_id\_seq'::regclass),

"ShopId\_id" bigint NOT NULL,

"storageId\_id" bigint NOT NULL,

CONSTRAINT api\_storagelinked\_pkey PRIMARY KEY (id),

CONSTRAINT "api\_storagelinked\_storageId\_id\_ShopId\_id\_e6c8950b\_uniq" UNIQUE ("storageId\_id", "ShopId\_id"),

CONSTRAINT "api\_storagelinked\_ShopId\_id\_bc1ef242\_fk" FOREIGN KEY ("ShopId\_id")

REFERENCES public.api\_shop (id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION

DEFERRABLE INITIALLY DEFERRED,

CONSTRAINT "api\_storagelinked\_storageId\_id\_bb77b01e\_fk\_api\_storageunit\_id" FOREIGN KEY ("storageId\_id")

REFERENCES public.api\_storageunit (id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION

DEFERRABLE INITIALLY DEFERRED

)

CREATE TABLE IF NOT EXISTS public.api\_storageunit

(

id bigint NOT NULL DEFAULT nextval('api\_storageunit\_id\_seq'::regclass),

capacity integer NOT NULL,

"unitKey" character varying(50) COLLATE pg\_catalog."default" NOT NULL,

CONSTRAINT api\_storageunit\_pkey PRIMARY KEY (id),

CONSTRAINT "api\_storageunit\_unitKey\_7f75ff50\_uniq" UNIQUE ("unitKey"),

CONSTRAINT api\_storageunit\_capacity\_check CHECK (capacity >= 0)

)

CREATE TABLE IF NOT EXISTS public.api\_storedat

(

id bigint NOT NULL DEFAULT nextval('api\_storedat\_id\_seq'::regclass),

quantity integer NOT NULL,

"productId\_id" bigint NOT NULL,

"storageId\_id" bigint NOT NULL,

CONSTRAINT api\_storedat\_pkey PRIMARY KEY (id),

CONSTRAINT "api\_storedat\_productId\_id\_storageId\_id\_39d34b6b\_uniq" UNIQUE ("productId\_id", "storageId\_id"),

CONSTRAINT "api\_storedat\_productId\_id\_ae824ca8\_fk\_api\_products\_id" FOREIGN KEY ("productId\_id")

REFERENCES public.api\_products (id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION

DEFERRABLE INITIALLY DEFERRED,

CONSTRAINT "api\_storedat\_storageId\_id\_ed78eb1c\_fk\_api\_storageunit\_id" FOREIGN KEY ("storageId\_id")

REFERENCES public.api\_storageunit (id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION

DEFERRABLE INITIALLY DEFERRED,

CONSTRAINT api\_storedat\_quantity\_check CHECK (quantity >= 0)

)

CREATE TABLE IF NOT EXISTS public.api\_suplier

(

id bigint NOT NULL DEFAULT nextval('api\_suplier\_id\_seq'::regclass),

name character varying(50) COLLATE pg\_catalog."default" NOT NULL,

phone integer NOT NULL,

created\_at timestamp with time zone NOT NULL,

updated\_at timestamp with time zone NOT NULL,

CONSTRAINT api\_suplier\_pkey PRIMARY KEY (id),

CONSTRAINT api\_suplier\_name\_key UNIQUE (name),

CONSTRAINT api\_suplier\_phone\_check CHECK (phone >= 0)

)

SQL QUERIES :

1. createProductSold:

INSERT INTO "api\_productsold" ("name", "jobStatus", "phone", "branchId")

VALUES ('value\_of\_name', 'value\_of\_jobStatus', 'value\_of\_phone', 'value\_of\_branchId');

1. getProductSolds:

sql

SELECT "api\_productsold"."id", "api\_productsold"."name", "api\_productsold"."jobStatus", "api\_productsold"."phone", "api\_productsold"."branchId", "api\_productsold"."created\_at"

FROM "api\_productsold";

1. getProductSold:

sql

SELECT "api\_productsold"."id", "api\_productsold"."name", "api\_productsold"."jobStatus", "api\_productsold"."phone", "api\_productsold"."branchId", "api\_productsold"."created\_at"

FROM "api\_productsold"

WHERE "api\_productsold"."id" = pk;

1. updateProductSold:

sql

UPDATE "api\_productsold"

SET

"name" = 'value\_of\_name',

"jobStatus" = 'value\_of\_jobStatus',

"phone" = 'value\_of\_phone',

"branchId" = 'value\_of\_branchId'

WHERE "api\_productsold"."id" = pk;

1. deleteProductSold:

sql

DELETE FROM "api\_productsold" WHERE "api\_productsold"."id" = pk;

SAME BASIC QUERY FOR EACH TABLE

**COMPLEX SQL QUERIES :**

**PRODUCT SOLD BY ALL EMPLOYEES DURING A DURATION**

'SELECT "api\_productsold"."id", "api\_productsold"."productId\_id", "api\_productsold"."orderId\_id", "api\_productsold"."quantity", "api\_productsold"."created\_at" FROM "api\_productsold"

**PRODUCT SOLD BY A SPECIFIC EMPLOYEES DURING A DURATION**

'SELECT "ps"."id", "ps"."productId\_id", "ps"."orderId\_id", "ps"."quantity", "ps"."created\_at" FROM "api\_orders" "o" JOIN "api\_productsold" "ps" ON "o"."id" = "ps"."orderId\_id" WHERE "o"."empId\_id" = %s AND "o"."created\_at" BETWEEN %s AND %s'

**GET SPACE LEFT IN A PARTICULAR STORAGE UNIT**

SELECT ("su"."capacity" - COALESCE(SUM("sd"."quantity"), 0)) AS "space\_left"

FROM "api\_storageunit" "su"

LEFT JOIN "api\_storedat" "sd" ON "su"."id" = "sd"."storageId\_id"

WHERE "su"."id" = %s

GROUP BY "su"."id", "su"."capacity"

**GET PRODUCTS WHOSE QUANTITY IS BELOW A CERTAIN THRESHOLD IN A ALL OR A SPECIFIC STORAGE UNIT**

'SELECT "api\_products"."id", "api\_products"."name", "api\_storedat"."quantity" FROM "api\_storedat" INNER JOIN "api\_products" ON ("api\_storedat"."productId\_id" = "api\_products"."id") WHERE "api\_storedat"."quantity" < %s'

**GET PRODUCTS LINKED TO A STORAGE UNIT**

SELECT "p"."id", "p"."name","s"."quantity"

FROM "api\_products" "p"

INNER JOIN "api\_storedat" "s" ON "p"."id" = "s"."productId\_id"

WHERE "s"."storageId\_id" = %s;

**GET STORAGE UNIT USED MY MOST SHOPS**

SELECT "storageId\_id","api\_storageunit"."unitKey", COUNT(\*) AS "num\_shops"

FROM "api\_storagelinked"

INNER JOIN "api\_storageunit" ON ("api\_storageunit"."id" = "api\_storagelinked"."storageId\_id")

GROUP BY "storageId\_id","api\_storageunit"."unitKey"

ORDER BY "num\_shops" DESC

**GET CUSTOMERS WHO BOUGHT A SPECIFIC PRODUCT**

SELECT "name"

FROM "api\_products"

WHERE %s IN (

SELECT "productId\_id"

FROM "api\_productsold"

WHERE "orderId\_id" IN (

SELECT "id"

FROM "api\_orders"

WHERE "api\_orders"."created\_at" BETWEEN NOW() - INTERVAL '1 WEEK' AND NOW()

)

);

**GET THE NUMBER OF A SPECIFIC PRODUCT LEFT THROUGH OUT ALL STORAGE UNITS**

SELECT "api\_products"."name",SUM("quantity") as total\_quantity

FROM "api\_storedat"

INNER JOIN "api\_products" on ("api\_storedat"."productId\_id" = "api\_products"."id")

WHERE "productId\_id" = %s

GROUP BY "api\_products"."name"

**GET ALL THE PRODUCTS LINKED TO A PARTICULAR ORDER**

SELECT "api\_products"."name", "api\_productsold"."quantity"

FROM "api\_productsold"

INNER JOIN "api\_products" ON "api\_productsold"."productId\_id" = "api\_products"."id"

WHERE "api\_productsold"."orderId\_id" = %s;

**GET EMPLOYEE WHO HAS CONDUCTED THE MOST SALES**

SELECT "emp"."id", "emp"."name", "emp"."phone", COUNT(\*) as "sales"

FROM "api\_employee" "emp"

JOIN "api\_orders" "o" ON "o"."empId\_id" = emp."id"

GROUP BY "emp"."id", "emp"."name", "emp"."phone"

ORDER BY "sales" DESC

LIMIT 1

**GET ALL STORAGE UNITS LINKED TO A PARTICULAR SHOP**

SELECT "su"."id", "su"."unitKey"

FROM "api\_storageunit" "su"

INNER JOIN "api\_storagelinked" "sd" ON "sd"."id" = "su"."id"

WHERE "sd"."ShopId\_id" = %s

**GET TOTAL REVENUE GENERATED BY A SHOP IN A PER MONTH BASIS**

SELECT

TO\_CHAR(DATE\_TRUNC('month', "api\_orders"."created\_at"), 'YYYY-MM') AS month,

"api\_orders"."branchId\_id" AS shop\_id,

SUM("api\_products"."price" \* "api\_productsold"."quantity") AS revenue

FROM "api\_orders"

INNER JOIN "api\_productsold"

ON "api\_orders"."id" = "api\_productsold"."orderId\_id"

INNER JOIN "api\_products"

ON "api\_productsold"."productId\_id" = "api\_products"."id"

GROUP BY

month, shop\_id;

**GET SHOP WHICH HAS MOST ORDERS DURING A DURATION**

SELECT "branchId\_id" as "shop\_id", COUNT(\*) AS "order\_count"

FROM "api\_orders" WHERE

"created\_at" BETWEEN %s AND %s

GROUP BY "branchId\_id"

ORDER BY "order\_count" DESC

LIMIT 1;

**GET ALL EMPLOYEES WORKING AT A SHOP**

SELECT "e"."name", "e"."phone", "e"."jobStatus"

FROM "api\_employee" AS "e"

INNER JOIN "api\_shop" AS "s" ON "e"."branchId\_id" = "s"."id"

WHERE "s"."id" = %s;

**GET ALL THE PRODUCTS AVAILABLE AT A SHOP**

SELECT "api\_products"."id", "api\_products"."name", "api\_productsold"."quantity"

FROM "api\_products"

INNER JOIN (

SELECT "api\_storedat"."productId\_id", SUM("api\_storedat"."quantity") AS "quantity"

FROM "api\_storedat"

INNER JOIN "api\_storageunit" ON "api\_storedat"."storageId\_id" = "api\_storageunit"."id"

INNER JOIN "api\_shop" ON "api\_storageunit"."id" = "api\_shop"."id"

INNER JOIN "api\_storagelinked" ON "api\_storageunit"."id" = "api\_storagelinked"."storageId\_id"

WHERE "api\_shop"."id" = %s

GROUP BY "api\_storedat"."productId\_id"

) AS "api\_productsold" ON "api\_products"."id" = "api\_productsold"."productId\_id"

**GET AVG NUMBERS OF ORDERS PER MONTH**

SELECT

AVG(num\_orders) AS avg\_orders\_per\_month

FROM (

SELECT

DATE\_TRUNC('month', "api\_orders"."created\_at") AS month,

COUNT(\*) AS num\_orders

FROM "api\_orders"

WHERE "api\_orders"."branchId\_id" = %s

GROUP BY month

) AS orders\_per\_month;

**GET AVG REVENUE PER MONTH FROM A SHOP**

SELECT

AVG(revenue\_per\_month) AS avg\_revenue\_per\_month

FROM (

SELECT

TO\_CHAR("api\_orders"."created\_at", 'YYYY-MM') AS year\_month,

"api\_orders"."branchId\_id" AS shop\_id,

SUM("api\_products"."price" \* "api\_productsold"."quantity") AS revenue\_per\_month

FROM "api\_orders"

INNER JOIN "api\_productsold"

ON "api\_orders"."id" = "api\_productsold"."orderId\_id"

INNER JOIN "api\_products"

ON "api\_productsold"."productId\_id" = "api\_products"."id"

WHERE "api\_orders"."branchId\_id" = %s

GROUP BY year\_month, shop\_id

) AS revenue\_per\_month\_by\_shop;

**GET ALL ORDERS DONE BY A CUSTOMER**

SELECT "o"."id","cus"."name", o."amount", "o"."created\_at" FROM "api\_orders" "o" INNER JOIN "api\_customer" "cus" ON ("o"."customerId\_id" = "cus"."id") WHERE "o"."customerId\_id" = %s

**SOME QUERIES TO GET STATS OF THE INVENTORIES**

sales\_query = """

SELECT "api\_shop"."location" AS store\_location, SUM("api\_orders"."amount") AS total\_sales

FROM "api\_orders"

INNER JOIN "api\_shop" ON "api\_orders"."branchId\_id" = "api\_shop"."id"

GROUP BY "api\_shop"."location"

ORDER BY total\_sales DESC;

"""

quantity\_query = """

SELECT "api\_shop"."location" AS store\_location, SUM("api\_storedat"."quantity") AS total\_quantity

FROM "api\_storedat"

INNER JOIN "api\_storageunit" ON "api\_storedat"."storageId\_id" = "api\_storageunit"."id"

INNER JOIN "api\_storagelinked" ON "api\_storageunit"."id" = "api\_storagelinked"."storageId\_id"

INNER JOIN "api\_shop" ON "api\_storagelinked"."ShopId\_id" = "api\_shop"."id"

GROUP BY "api\_shop"."location"

ORDER BY total\_quantity DESC;

"""

orders\_query = """

SELECT EXTRACT(YEAR FROM "api\_orders"."created\_at") AS order\_year, COUNT(\*) AS total\_orders

FROM "api\_orders"

GROUP BY order\_year

ORDER BY order\_year ASC;

"""

product\_query = """

SELECT "api\_products"."name" AS product\_name, SUM("api\_productsold"."quantity") AS total\_sold

FROM "api\_productsold"

INNER JOIN "api\_products" ON "api\_productsold"."productId\_id" = "api\_products"."id"

GROUP BY "api\_products"."name"

ORDER BY total\_sold DESC;

"""

**GET MOST POPULAR GROUP IN A DURATION OR OVERALL**

SELECT "api\_productsgroup"."id", "api\_productsgroup"."name", SUM("api\_productsold"."quantity") AS "total\_quantity"

FROM "api\_productsgroup"

INNER JOIN "api\_products" ON ("api\_productsgroup"."id" = "api\_products"."group\_id")

INNER JOIN "api\_productsold" ON ("api\_products"."id" = "api\_productsold"."productId\_id")

WHERE "api\_products"."productBeingSold" = TRUE AND "api\_productsold"."created\_at" BETWEEN %s AND %s

GROUP BY "api\_productsgroup"."id"

ORDER BY "total\_quantity" DESC

LIMIT 1;

PL/SQL

-- Function and Cursor

**A PL/SQL function block that takes in customer id as input and outputs the most ordered product group of the customer:**

CREATE OR REPLACE FUNCTION get\_most\_ordered\_group(customer\_id bigint) RETURN NUMBER IS

DECLARE

group\_key bigint;

BEGIN

SELECT p.group\_id into group\_key

FROM api\_orders o

INNER JOIN "api\_productsold" "ps" ON (o.id = "ps"."orderId\_id")

INNER JOIN "api\_products" "p" ON "ps"."productId\_id" = "p"."id"

WHERE "o"."customerId\_id" = customer\_id

GROUP BY p.group\_id,p.name

ORDER BY sum(quantity) DESC

FETCH FIRST 1 ROW ONLY;

RETURN group\_key;

END;

**A PL/SQL function block that takes in a refcursor as input and returns a ref cursor with the most ordered groups of every customer**

CREATE OR REPLACE FUNCTION get\_most\_ordered\_group\_for\_all(ref1 refcursor) RETURNS refcursor IS

DECLARE

BEGIN

OPEN ref1 FOR

SELECT c.id, c.name, get\_most\_ordered\_group(c.id)

FROM api\_customer c;

return NEXT ref1;

END;

**A PL/SQL function block that takes in a refcursor as input and returns a ref cursor with the most ordered product of every customer**

CREATE OR REPLACE FUNCTION get\_most\_ordered\_product(ref1 refcursor) RETURNS refcursor IS

BEGIN

OPEN ref1 FOR

SELECT c.id, c.name, get\_most\_ordered\_products(c.id)

FROM api\_customer c;

return NEXT ref1;

END;

**A PL/SQL function block that takes in customer id as input and outputs the most ordered product group of the customer:**

CREATE OR REPLACE FUNCTION get\_most\_ordered\_product\_for\_all(customer\_id bigint) RETURN NUMBER IS

DECLARE

product\_name bigint;

BEGIN

SELECT p.id

INTO product\_name

FROM api\_orders o

INNER JOIN "api\_productsold" "ps" ON (o.id = "ps"."orderId\_id")

INNER JOIN "api\_products" "p" ON "ps"."productId\_id" = "p"."id"

WHERE "o"."customerId\_id" = customer\_id

GROUP BY "p"."id", "p"."name"

ORDER BY sum(quantity) DESC

FETCH FIRST 1 ROW ONLY;

RETURN product\_name;

END;

**A PL/SQL function block that takes in Storage unit id and returns the space left**

CREATE OR REPLACE FUNCTION get\_space\_left(unit\_id bigint) RETURNS bigint IS  
DECLARE

space\_left bigint;

BEGIN

SELECT ("su"."capacity" - COALESCE(SUM("sd"."quantity"), 0)) into space\_left

FROM "api\_storageunit" "su"

LEFT JOIN "api\_storedat" "sd" ON "su"."id" = "sd"."storageId\_id"

WHERE "su"."id" = unit\_id

GROUP BY "su"."id", "su"."capacity";

RETURN space\_left;

END;

**A PL/SQL function block that takes in Storage unit id and returns the space left**

CREATE OR REPLACE FUNCTION get\_storage\_unit\_from\_shop\_with\_highest\_quantity(shop\_id bigint,product\_id bigint) RETURNS bigint IS

DECLARE

storage\_unit\_id bigint;

max\_quantity bigint := 0;

current\_quantity bigint;

max\_storage\_id bigint := 1;

BEGIN

FOR storage\_unit\_id IN

SELECT "storageId\_id"

FROM api\_storagelinked

WHERE "ShopId\_id" = shop\_id

LOOP

SELECT quantity INTO current\_quantity

FROM api\_storedat

WHERE "storageId\_id" = storage\_unit\_id AND "productId\_id" = product\_id;

IF current\_quantity > max\_quantity THEN

max\_quantity := current\_quantity;

max\_storage\_id := storage\_unit\_id;

END IF;

END LOOP;

RETURN max\_storage\_id;

END;

**A PL/SQL function block that takes in Shop id and returns all the storage units linked with it**

CREATE OR REPLACE FUNCTION get\_storage\_units\_for\_shop(shop\_id bigint,ref1 refcursor) RETURNS refcursor IS

BEGIN

OPEN ref1 FOR

SELECT s.id

FROM api\_storageunit s

INNER JOIN api\_storagelinked sl ON s.id = sl."storageId\_id"

WHERE sl."ShopId\_id" = shop\_id;

RETURN NEXT ref1;

END;

-- Trigger:

**On entering a tuple in storedat table it checks if there is enough space left in the unit to contain the incoming products**

CREATE OR REPLACE TRIGGER check\_space\_left  
AFTER INSERT ON storedat  
FOR EACH ROW  
BEGIN

IF NEW.quantity > get\_space\_left(NEW."storageId\_id") THEN

RAISE EXCEPTION 'Not enough space available in storage unit';

END IF;

RETURN NEW;

END;

**On entering a tuple in storedat table it checks if there is enough space left in the unit to contain the incoming products**

CREATE OR REPLACE TRIGGER check\_space\_left  
AFTER INSERT ON storedat  
FOR EACH ROW  
BEGIN

IF NEW.quantity > get\_space\_left(NEW."storageId\_id") THEN

RAISE EXCEPTION 'Not enough space available in storage unit';

END IF;

RETURN NEW;

END;

**On entering a tuple in productsold it subtracts the respective quantity from the storage unit linked to the particular order if multiple units have the same products the one with greater quantity will be selected**

DECLARE

shop\_id INTEGER;

storage\_unit\_id INTEGER;

product\_id INTEGER;

quantity\_out INTEGER;

BEGIN

SELECT "o"."branchId\_id", "ps"."productId\_id", "ps"."quantity"

INTO shop\_id, product\_id, quantity\_out

FROM "api\_productsold" "ps"

INNER JOIN "api\_orders" "o" ON "ps"."orderId\_id" = "o"."id"

WHERE "ps"."id" = NEW."id";

SELECT get\_storage\_unit\_from\_shop\_with\_highest\_quantity(shop\_id, NEW."productId\_id") INTO storage\_unit\_id;

IF storage\_unit\_id IS NULL THEN

RAISE EXCEPTION 'No storage units found for product % in shop %', NEW."productId\_id", shop\_id;

END IF;

UPDATE "api\_storedat"

SET "quantity" = "quantity" - quantity\_out

WHERE "storageId\_id" = storage\_unit\_id AND "productId\_id" = product\_id;

RETURN NEW;

END;

**On entering a tuple in storedat table it updates the invoice related to that specific order to have the correct total cost and also add all the products linked to the specific order in a array for reference**

DECLARE

invoice\_id INTEGER;

product\_price NUMERIC;

BEGIN

-- get the invoice id for the current order

SELECT id INTO invoice\_id FROM api\_invoice WHERE "orderId\_id" = NEW."orderId\_id";

-- get the price of the product

SELECT price INTO product\_price FROM api\_products WHERE id = NEW."productId\_id";

-- update the invoice with the new amount

UPDATE api\_invoice

SET amount = amount + NEW.quantity \* product\_price,

products = jsonb\_set(products::jsonb, '{product ids}',

(products::jsonb->>'product ids')::jsonb ||

jsonb\_build\_array(jsonb\_build\_array(NEW."productId\_id", NEW.quantity)))

WHERE id = invoice\_id;

RETURN NEW;

END;