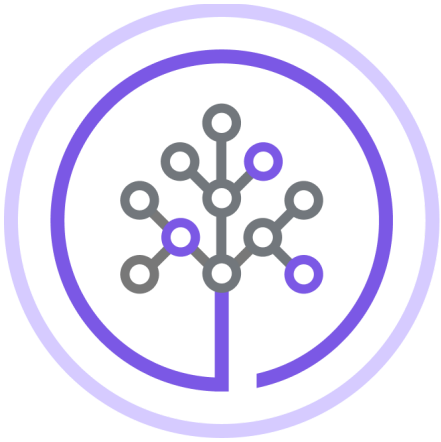


Project : Database Design and Implementation

Solution by Julia Villalba

Introduction

This project is a solution to the final assignment for the "Introduction to Relational Databases" course, part of the IBM Data Engineering Professional Certificate. This project involves designing a relational database system for a New York-based coffee shop chain that is looking to expand nationally by opening several franchise locations. Our role as Data Engineers is to streamline the coffee shop's operations by revamping their data infrastructure and improving operational efficiencies.

Currently, the coffee shop's data is spread across multiple systems, including accounting software, suppliers' databases, POS systems, and spreadsheets. Our objective is to review the data in all of these systems, design a centralized database to house all of the data, and create subsets of data that the coffee shop's executives require. We will export these subsets and load them into staging databases.

To accomplish this, we will use PostgreSQL Database, a RDBMS designed to efficiently store, manipulate, and retrieve data. We will be working with a subset of data from the coffee shop sample data, including staff and sales outlet information, sales data from the POS system, customer data from a bespoke CRM system, and product information from the supplier's database.

Our objectives for this project include identifying entities and attributes, creating an entity relationship diagram (ERD) using the pgAdmin ERD Tool, normalizing tables, defining keys and relationships, and creating database objects by generating and running SQL scripts from the

ERD Tool. By achieving these objectives, we will create a streamlined, centralized database system that will allow the coffee shop chain to make data-driven decisions and achieve their expansion goals.

Important note: In designing this database, I made an effort to separate every table that had string repetitions in order to achieve a normalized and efficient database structure. However, it's important to note that there is no one-size-fits-all solution when it comes to database design, as it heavily depends on the specific use case and the requirements of the users. While in some cases having fewer tables with string repetitions might be more efficient, in other cases a more normalized approach might be better suited. IBM's solution takes a different approach, addressing the normalization procedure by adding only a few more tables.

Definitions

In the context of relational databases, normalization is a technique that reduces redundancy and increases the consistency of data. There are two primary forms of normalization:

1. First Normal Form (1NF): This form requires a table to contain only single values and no repeating groups. By following this form, data can be organized more efficiently and queried more effectively.
2. Second Normal Form (2NF): This form involves splitting data into multiple tables to further reduce redundancy. By doing so, each table can be designed to represent a single entity or concept, leading to more efficient storage and more meaningful queries.

Data used in this project

In this project, we will be working with a subset of data from the Coffee shop sample data. We will be working with data from the following sources:

- Staff information held in a spreadsheet at HQ
- Sales outlet information held in a spreadsheet at HQ
- Sales data output as a CSV file from the POS system in the sales outlets
- Customer data output as a CSV file from a bespoke customer relationship management system
- Product information maintained in a spreadsheet exported from your supplier’s database

Objectives

1. Identify entities and attributes.
2. Normalize tables and define keys and relationships.
3. Create an entity relationship diagram (ERD) using the pgAdmin ERD Tool.
4. Create database objects by generating and running the SQL script from the ERD Tool.

Solution

The following image shows sample data from each of the data sources that we are working with to design the new central database.

staff

staff_id	first_name	last_name	position	start_date	location
1	Sue	Tindale	CFO	08/03/2001	HQ
2	Ian	Tindale	CEO	3/8/2001	HQ
3	Marny	Hermione	Roaster	10/24/2007	WH
4	Chelsea	Claudia	Roaster	3/7/2003	WH
5	Alec	Isadora	Roaster	2/4/2008	WH
6	Xena	Rahim	Store Manager	7/24/2016	3
7	Kelsey	Cameron	Coffee Wrangler	10/18/2003	3
8	Hamilton	Emi	Coffee Wrangler	9/2/2005	3
9	Caldwell	Veda	Coffee Wrangler	9/9/2013	3
10	Ima	Winifred	Coffee Wrangler	10/12/2016	3

sales_outlet

sales_outlet_id	sales_outlet_type	address	city	telephone	postal_code	manager
2	warehouse	164-14 Jamaica Ave	Jamaica	972-871-0402	11432	
3	retail	32-20 Broadway	Long Island City	777-718-3190	11106	6
4	retail	604 Union Street	Brooklyn	619-347-5193	11215	11
5	retail	100 Church Street	New York	343-212-5151	10007	16

sales_transaction

transaction_id	transaction_date	transaction_time	sales_outlet_id	staff_id	customer_id	product_id	quantity	price
1	27/04/2019	09:53:55	8	42	0	38	2	3.75
1	27/04/2019	09:53:55	8	42	0	84	1	0.8
2	27/04/2019	08:00:34	8	42	0	51	2	3
3	27/04/2019	09:04:58	8	42	0	33	1	3.5
4	27/04/2019	08:48:32	8	42	8232	27	1	3.5
5	27/04/2019	09:21:40	8	45	8223	24	1	3

customer

customer_id	customer_name	customer_email	customer_since	customer_card_number	birthdate	gender
3001	Kelly Key	Venus@adipiscing.edu	04/01/2017	908-424-2890	29/05/1950	M
3002	Clark Schroeder	Nora@fames.gov	07/01/2017	032-732-6308	30/07/1950	M
3003	Elvis Cardenas	Brianna@tellus.edu	10/01/2017	459-375-9187	30/09/1950	M
3004	Rafael Estes	Ina@non.gov	13/01/2017	576-640-9226	01/12/1950	M
3005	Colin Lynn	Dale@Integer.com	15/01/2017	344-674-6569	01/02/1951	M

product

product_id	product_category	product_type	product_name	description	price
1	Coffee beans	Organic Beans	Brazilian - Organic	It's like Carnival in a cup. Clean and smooth.	18
2	Coffee beans	House blend Beans	Our Old Time Diner Blend	Our packed blend of beans that is reminiscent of the cup of coffee you used to get at a diner.	18
3	Coffee beans	Espresso Beans	Espresso Roast	Our house blend for a good espresso shot.	14.75
4	Coffee beans	Espresso Beans	Primo Espresso Roast	Our premium single source of hand roasted beans.	20.45
5	Coffee beans	Gourmet Beans	Columbian Medium Roast	A smooth cup of coffee any time of day.	15
6	Coffee beans	Gourmet Beans	Ethiopia	From the home of coffee.	21

To ensure the efficiency and consistency of our database, we will analyze each source of information and define the relationships between them. We will employ normalization, a best practice in database design, which will prevent redundancy by avoiding the repetition of data across multiple rows and ensuring that each piece of data is stored only in one place. Normalization will also enable us to retrieve and analyze data from multiple tables with ease, using a single query. The sources of information we will be analyzing are: "staff", "staff_outlet", "sales_transaction", "customer", and "product". We will examine each of these sources in detail.

Staff: The table "staff" contains the following columns: staff_id, first_name, last_name, position, start_date, and location. Two of these columns, "position" and "location", contain repetitive strings. To normalize this table, we will create two additional tables: "position" and "location". These tables will define primary and foreign keys to link them together. For example, the "position" table might have a primary key of "position_id", while the "staff" table might have a foreign key of "position_id" that links each staff member to their position. Similarly, the "location" table might have a primary key of "location_id", while the "staff" table might have a foreign key of "location_id" that links each staff member to their location.

Sales_outlet: The table "sales_outlet" contains the following columns: sales_outlet_id, sales_outlet_type, address, city, telephone, postal_code, and manager. Two of these columns, "sales_outlet_type" and "city", contain repetitive strings. To normalize this table, we will create two additional tables: "sales_outlet_type" and "city". These tables will have their corresponding primary keys. The "sales_outlet" table will have two foreign keys that reference their corresponding primary keys.

Sales_transaction: The table "sales_transaction" contains the following columns: transaction_id, transaction_date, transaction_time, sales_outlet_id, staff_id, customer_id, product_id, quantity, and price. This table is already normalized, as several columns such as "staff_id", "customer_id", and "product_id" are foreign keys that match the primary keys in their corresponding tables.

Customer: The customer table has columns such as customer_id, customer_name, customer_email, customer_since, customer_card_number, birthdate, and gender. While it is generally considered good practice to normalize data by creating separate lookup tables, in the case of columns like "gender" with a small number of distinct values (such as "M" and "F"), it may not be worth the overhead of creating a separate lookup table to avoid string repetition. Therefore, we can leave the "gender" column as a string column in the customer table.

Product: The product table contains columns such as product_id, product_category, product_type, product_name, description, and price. To normalize this table, we can create separate tables for product_category and product_type, each with its corresponding primary and foreign keys. By doing so, we can avoid string repetition and ensure that each piece of data is stored in one place. This will also make it easier to retrieve and analyze data from multiple tables in a single query.

Now that we have a plan for normalizing the tables, we have the possible entities for our database design. An entity is typically a real-world object, concept, or event that can be uniquely identified and represented within the database. Each entity will correspond to a separate table in our database. The columns within each table correspond to the attributes of the entity. Attributes are the specific characteristics or properties that describe the entity.

Here we share the list of all the entities, their attributes and their relationships:

Entities	**Attributes**
Staff	● staff_id (integer) ● first_name (string) ● last_name (string) ● position_id (integer, foreign key) ● start_date (date) ● location_id (integer, foreign key)
Position	● position_id (integer, primary key) ● position_name (string)
Location	● location_id (integer, primary key) ● location_name (string)
Sales Outlet	● sales_outlet_id (integer) ● sales_outlet_type_id (integer, foreign key) ● address (string) ● city_id (integer, foreign key) ● telephone (string) ● postal_code (string) ● manager (string)
Sales Outlet Type	● sales_outlet_type_id (integer, primary key) ● sales_outlet_type_name (string)
City	● city_id (integer, primary key) ● city_name (string)

Customer	• customer_id (integer) • customer_name (string) • customer_email (string) • customer_since (date) • customer_card_number (string) • birthdate (date) • gender (string)
Product	• product_id (integer) • product_category_id (integer, foreign key) • product_type_id (integer, foreign key) • product_name (string) • description (string) • price (decimal)
Product Category	• product_category_id (integer, primary key) • product_category_name (string)
Product Type	• product_type_id (integer, primary key) • product_type_name (string)
Sales Transaction	• transaction_id (integer) • transaction_date (date) • transaction_time (time) • sales_outlet_id (integer, foreign key) • staff_id (integer, foreign key) • customer_id (integer, foreign key) • product_id (integer, foreign key) • quantity (integer) • price (decimal)

Script for generating the tables

```
-- This script was generated by a beta version of the ERD tool in pgAdmin 4.

-- Please log an issue at https://redmine.postgresql.org/projects/pgadmin4/issues/new
if you find any bugs, including reproduction steps.

BEGIN;

CREATE TABLE IF NOT EXISTS public.city

(

    city_id integer NOT NULL,

    city_name character varying(255) COLLATE pg_catalog."default",

    CONSTRAINT city_pkey PRIMARY KEY (city_id)

);

CREATE TABLE IF NOT EXISTS public.customer

(

    customer_id integer NOT NULL,

    customer_name character varying(255) COLLATE pg_catalog."default",

    customer_email character varying(255) COLLATE pg_catalog."default",

    customer_since date,

    customer_card_number character varying(255) COLLATE pg_catalog."default",

    birthdate date,

    gender character varying(255) COLLATE pg_catalog."default",

    CONSTRAINT customer_pkey PRIMARY KEY (customer_id)

);

CREATE TABLE IF NOT EXISTS public.location

(

    location_id integer NOT NULL,

    location_name character varying(255) COLLATE pg_catalog."default",
```

```
    CONSTRAINT location_pkey PRIMARY KEY (location_id)
);
CREATE TABLE IF NOT EXISTS public."position"
(
    position_id integer NOT NULL,
    position_name character varying(255) COLLATE pg_catalog."default",
    CONSTRAINT position_pkey PRIMARY KEY (position_id)
);
CREATE TABLE IF NOT EXISTS public.product
(
    product_id integer NOT NULL,
    product_category_id integer,
    product_type_id integer,
    product_name character varying(255) COLLATE pg_catalog."default",
    description character varying(255) COLLATE pg_catalog."default",
    price numeric,
    CONSTRAINT product_pkey PRIMARY KEY (product_id)
);
CREATE TABLE IF NOT EXISTS public.productcategory
(
    product_category_id integer NOT NULL,
    product_category_name character varying(255) COLLATE pg_catalog."default",
    CONSTRAINT productcategory_pkey PRIMARY KEY (product_category_id)
);
CREATE TABLE IF NOT EXISTS public.producttype
(
```

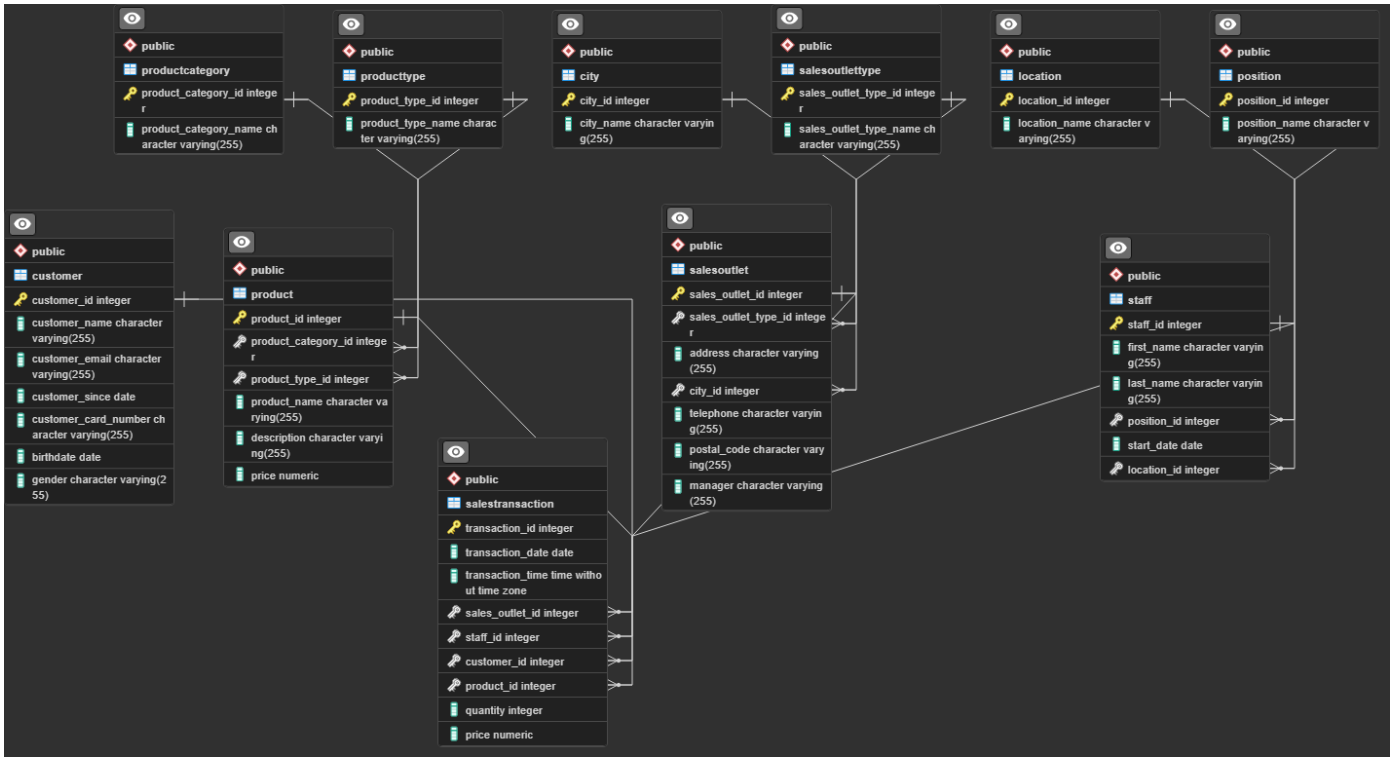
```
    product_type_id integer NOT NULL,
    product_type_name character varying(255) COLLATE pg_catalog."default",
    CONSTRAINT producttype_pkey PRIMARY KEY (product_type_id)
);
CREATE TABLE IF NOT EXISTS public.salesoutlet
(
    sales_outlet_id integer NOT NULL,
    sales_outlet_type_id integer,
    address character varying(255) COLLATE pg_catalog."default",
    city_id integer,
    telephone character varying(255) COLLATE pg_catalog."default",
    postal_code character varying(255) COLLATE pg_catalog."default",
    manager character varying(255) COLLATE pg_catalog."default",
    CONSTRAINT salesoutlet_pkey PRIMARY KEY (sales_outlet_id)
);
CREATE TABLE IF NOT EXISTS public.salesoutlettype
(
    sales_outlet_type_id integer NOT NULL,
    sales_outlet_type_name character varying(255) COLLATE pg_catalog."default",
    CONSTRAINT salesoutlettype_pkey PRIMARY KEY (sales_outlet_type_id)
);
CREATE TABLE IF NOT EXISTS public.salestransaction
(
    transaction_id integer NOT NULL,
    transaction_date date,
    transaction_time time without time zone,
```

```
    sales_outlet_id integer,
    staff_id integer,
    customer_id integer,
    product_id integer,
    quantity integer,
    price numeric,
    CONSTRAINT salestransaction_pkey PRIMARY KEY (transaction_id)
);
CREATE TABLE IF NOT EXISTS public.staff
(
    staff_id integer NOT NULL,
    first_name character varying(255) COLLATE pg_catalog."default",
    last_name character varying(255) COLLATE pg_catalog."default",
    position_id integer,
    start_date date,
    location_id integer,
    CONSTRAINT staff_pkey PRIMARY KEY (staff_id)
);
ALTER TABLE IF EXISTS public.product
    ADD CONSTRAINT product_product_category_id_fkey FOREIGN KEY (product_category_id)
    REFERENCES public.productcategory (product_category_id) MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION;
ALTER TABLE IF EXISTS public.product
    ADD CONSTRAINT product_product_type_id_fkey FOREIGN KEY (product_type_id)
    REFERENCES public.producttype (product_type_id) MATCH SIMPLE
```

```
    ON UPDATE NO ACTION

    ON DELETE NO ACTION;

ALTER TABLE IF EXISTS public.salesoutlet

    ADD CONSTRAINT salesoutlet_city_id_fkey FOREIGN KEY (city_id)

    REFERENCES public.city (city_id) MATCH SIMPLE

    ON UPDATE NO ACTION

    ON DELETE NO ACTION;

ALTER TABLE IF EXISTS public.salesoutlet

    ADD CONSTRAINT salesoutlet_sales_outlet_type_id_fkey FOREIGN KEY
(sales_outlet_type_id)

    REFERENCES public.salesoutlettype (sales_outlet_type_id) MATCH SIMPLE

    ON UPDATE NO ACTION

    ON DELETE NO ACTION;

ALTER TABLE IF EXISTS public.salestransaction

    ADD CONSTRAINT salestransaction_customer_id_fkey FOREIGN KEY (customer_id)

    REFERENCES public.customer (customer_id) MATCH SIMPLE

    ON UPDATE NO ACTION

    ON DELETE NO ACTION;

ALTER TABLE IF EXISTS public.salestransaction

    ADD CONSTRAINT salestransaction_product_id_fkey FOREIGN KEY (product_id)

    REFERENCES public.product (product_id) MATCH SIMPLE

    ON UPDATE NO ACTION

    ON DELETE NO ACTION;

ALTER TABLE IF EXISTS public.salestransaction

    ADD CONSTRAINT salestransaction_sales_outlet_id_fkey FOREIGN KEY
(sales_outlet_id)

    REFERENCES public.salesoutlet (sales_outlet_id) MATCH SIMPLE
```

```
    ON UPDATE NO ACTION

    ON DELETE NO ACTION;

ALTER TABLE IF EXISTS public.salestransaction

    ADD CONSTRAINT salestransaction_staff_id_fkey FOREIGN KEY (staff_id)

    REFERENCES public.staff (staff_id) MATCH SIMPLE

    ON UPDATE NO ACTION

    ON DELETE NO ACTION;

ALTER TABLE IF EXISTS public.staff

    ADD CONSTRAINT staff_location_id_fkey FOREIGN KEY (location_id)

    REFERENCES public.location (location_id) MATCH SIMPLE

    ON UPDATE NO ACTION

    ON DELETE NO ACTION;

ALTER TABLE IF EXISTS public.staff

    ADD CONSTRAINT staff_position_id_fkey FOREIGN KEY (position_id)

    REFERENCES public."position" (position_id) MATCH SIMPLE

    ON UPDATE NO ACTION

    ON DELETE NO ACTION;

END;
```

Creating an ERD



Conclusions

In this project, we have designed a relational database schema that aims to reduce data redundancy and improve data consistency. By normalizing the data into separate tables and linking them through foreign keys, we have avoided storing duplicate data and improved data integrity. However, it's important to note that there's no one-size-fits-all solution for designing databases, as the optimal design will depend on the specific requirements of the application and the preferences of the users. In some cases, denormalization may be preferred if it allows for faster query performance or simpler code implementation. Nonetheless, following the principles of normalization is generally recommended for creating well-structured, maintainable databases.

Overall, this project has provided a solid foundation for building a functional database that can store and manage data effectively. Future work could involve further refinement of the schema based on additional requirements, as well as implementation of the database in a specific programming language or framework.