[redacted]

Boston University

MET CS669 **Database Design and Implementation for Business**

Qiyuan Hu [redacted]

[redacted]

Table of Contents

Project Direction Overview

Who the database will be for?

I want to create a web application for **individual users** to query the lifespan of businesses.

What kind of data it will contain?

My database will contain the name of the business, name of the country, name of the continent that the country is in, business category, and which year the business was founded.

How do I envision it will be used?

What is the oldest business, and which continent/country it locates in?
Which categories of business have a longer lifespan?
Which continent has the eldest business?
Which are the most common categories for the oldest businesses on each continent?
Individual users can easily filter counts of old businesses by continent and category. Then, they can have the results compared with the ease of doing business-standard created by the world bank to check if any policies in some particular continents will benefit start-ups.

Why am I interested in it?

I was taking Entrepreneurship and leadership as my electives in my undergraduate. According to the book named zero to one written by Peter Thiel, most start-ups cannot survive one year. Especially during pedantic, it's such a tough life for a business owner. However, some businesses last for hundreds of years. What category of these businesses? Which country do these businesses locate in?

Use Cases and Fields

Every individual user needs to log in then purchase the information from the web application.

Use case/ Account Signup

- The user visits Businesslifespan's website and starts to search for information.
- The website asks him/her to create an account when its first run.
- The user enters his/her information, and the account is created in the database.
- The web application asks him/her to allow follow and track their view history.

Field	What it stores	Why does it be need
Account	This field stores a summary name	Sometimes the same person will have multiple accounts, they can select

	associated with each account.	the correct one from a dropdown list.
FirstName	This field stores the first name of the account holder.	It is essential for displaying the person's name on screens and addressing them when sending them emails or other communications.
LastName	This field stores the last name of the account holder.	It is essential for displaying the person's name on screens and addressing them when sending them emails or other communications.
UserDate	This is the date the account when was created	It would be used to track user stickiness and making marketing campaigns such as emailing discounts to heavy users, etc.
AccountBalance	This field stores the balance owed by the user.	It is useful to track of users that owe money to use the website.

Purchase detailed information use case

i. The user logs in to businesslifespan.
ii. The user selects the option to search the name of the business.
Businesslifespan pulls a form including the name of business, name of the country, name of the continent that the country is in, business category, and which year the business was founded.
iii. The user can filter, count, or search based on a specific continent or category, which causes a database search.
iv. The web application pulls all values matching the criteria from the database.
v. The users select the business they are interested in.
vi. Businesslifesspan shows all recorded information about each business.
vii. In the end, the user can close the website or share the information they purchased by clicking the share option.

The database contains five tables.

Countries

Column	Type	Meaning
Country	Varchar	Name of the country
Continent	Varchar	Name of the continent that the country in
Country_Code	Varchar	ISO 3166-1 3-letter country code

Categories

Column	Type	Meaning
Category	Varchar	Business category
Category_Code	Varchar	Code for the business category

Business

Column	Type	Meaning
Business_Name	Varchar	Name of the business
Year	Int	Which year Business was founded
Category_Code	Varchar	Code for the business category
Country_Code	Varchar	ISO 3166-1 3-letter country code

Account

Column	Type	Meaning
Account_Id	Decimal	Id number for individual
First_Name	Varchar	First Name of the user
Last_Name	Varchar	Last Name of the user
User_Date	Date	Account created date

Orders

Column	Type	Meaning
Orders_Id	Decimal	Number for each order
Account_Id	Decimal	Id number for individual
Spend_Date	Date	Date for creating order
Spend_Amount	Decimal	The amount of order

FreeAccount

Column	Type	Meaning
Account_Id	Decimal	Id number for individual

PaidAccount

Column	Type	Meaning
Account_Id	Decimal	Id number for individual
Account_Balance	Decimal	Balance in account
Renewal_date	Date	When to renew the account

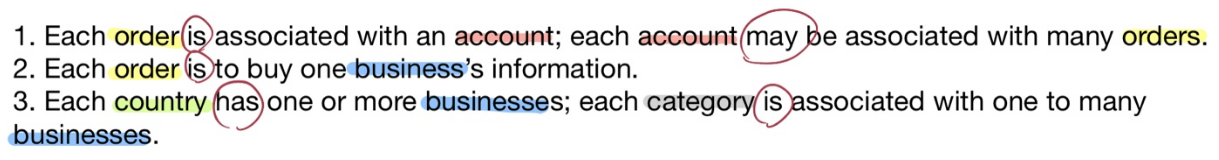
BalanceChange

Column	Type	Meaning
balancechange_id	Decimal	Id number for individual
prev_balance	Decimal	Previous balance amount
Current_balance	Decimal	Previous current amount
Paid_account_id	Decimal	Paid account id
Change_Date	Date	When to change the account

Structural Database Rules

Business rules are focused on the general operation of the business but are not design or implementation details for the application or database. Business rules along with other items are the foundation for creating rules and constraints for each application or I.T. system component. Application programmers create **UML diagrams** such as class diagrams and sequence diagrams to help describe how the application will be designed. Database designers create **structural database rules** and **entity-relationship diagrams** to describe how the database will be designed.

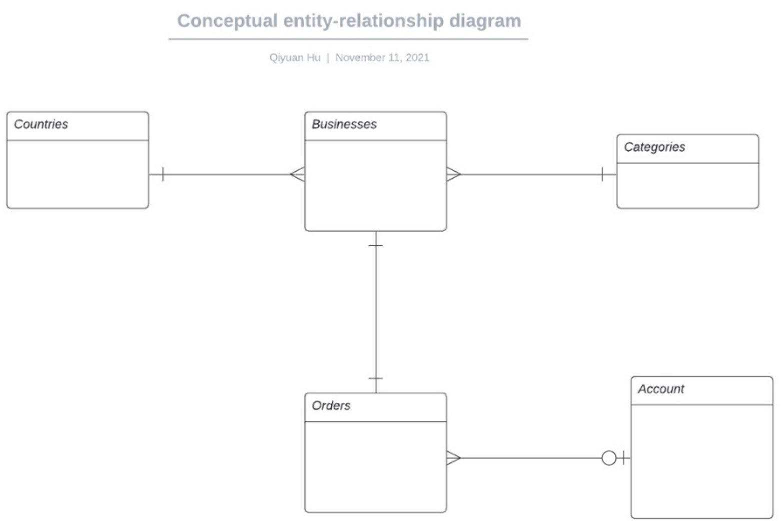
- An entity is a blueprint for a data set and each item in the dataset is termed an entity instance.
- A database relationship is an association between two entities.



1. Each order is associated with an account; each account may be associated with many orders.
2. Each order is to buy one business's information.
3. Each country has one or more businesses; each category is associated with one to many businesses.

- *From the perspective of account, it may or may not participate in the relationship with orders (optional participation). From the perspective of account, it may be associated to many orders (plural).*
- *From the perspective of order, it must participate in the relationship with account and business (mandatory participation). From the perspective of order, it must be associated to one account and one business (singular).*
- *From the perspective of country, it must participate in the relationship with business (mandatory participation). From the perspective of country, it may be associated to many businesses (plural).*
- *From the perspective of category, it must participate in the relationship with business (mandatory participation). From the perspective of category, it may be associated to many businesses (plural).*

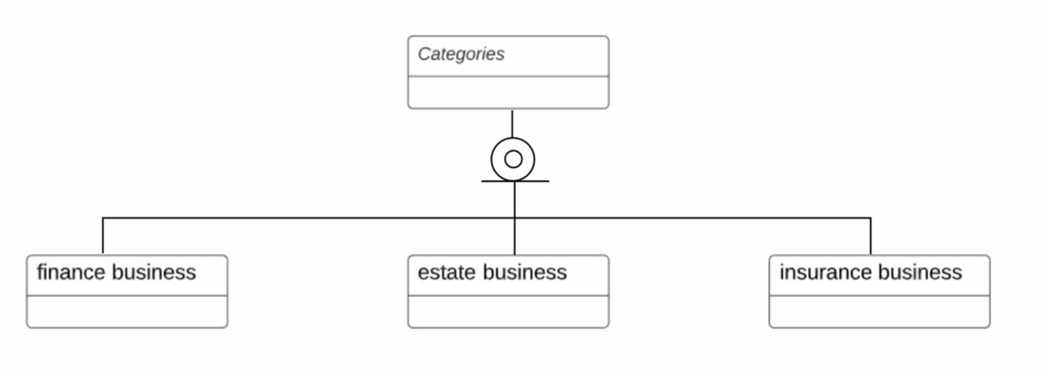
Conceptual entity-relationship diagram


Conceptual entity-relationship diagram
Qiyuan Hu | November 11, 2021
Countries
Businesses
Categories
Orders
Account

Initial DBMS Physical ERD

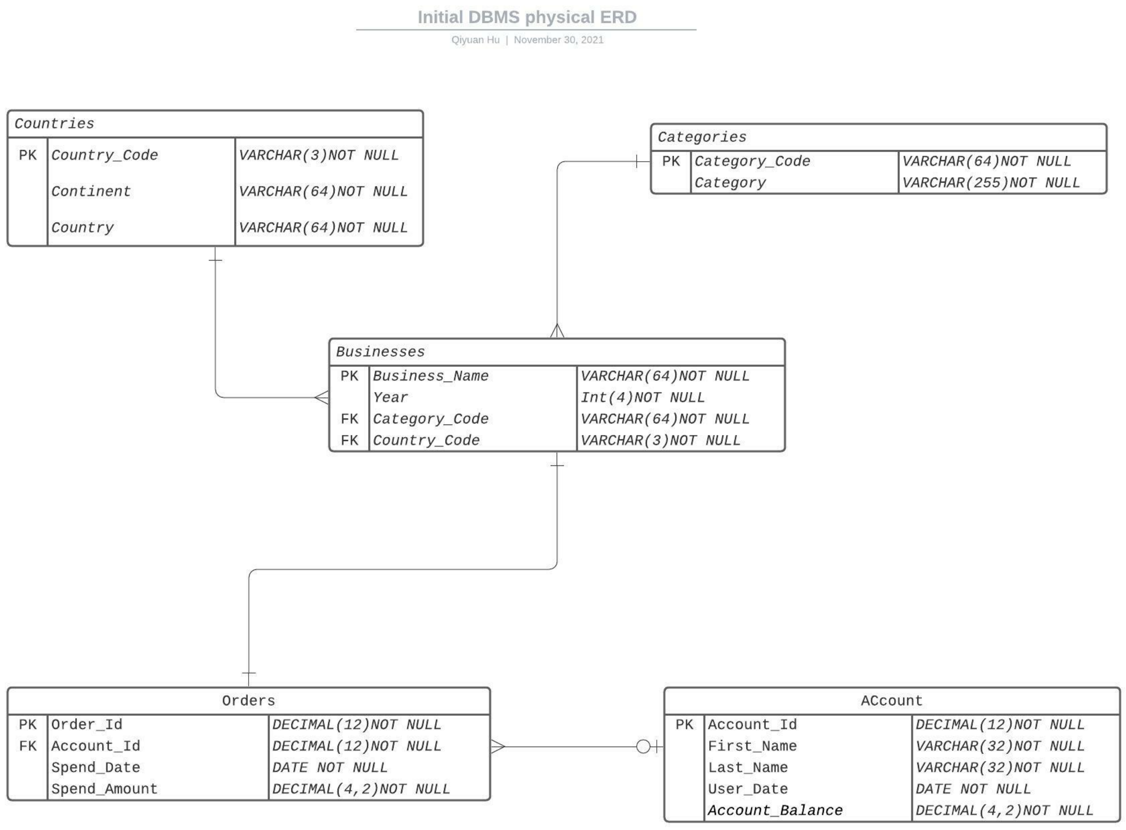
Specialization-Generalization relationships

A category has finance business, estate business, insurance business, **several of these, or none of these**



Using Crow's Foot, we use the "O" to indicate that the relationship is **overlapping**, since the same businesses can be in multiple business. We use the single bar to indicate the relationship is **partially complete**, since there are other kinds of businesses are categories of other than those listed.

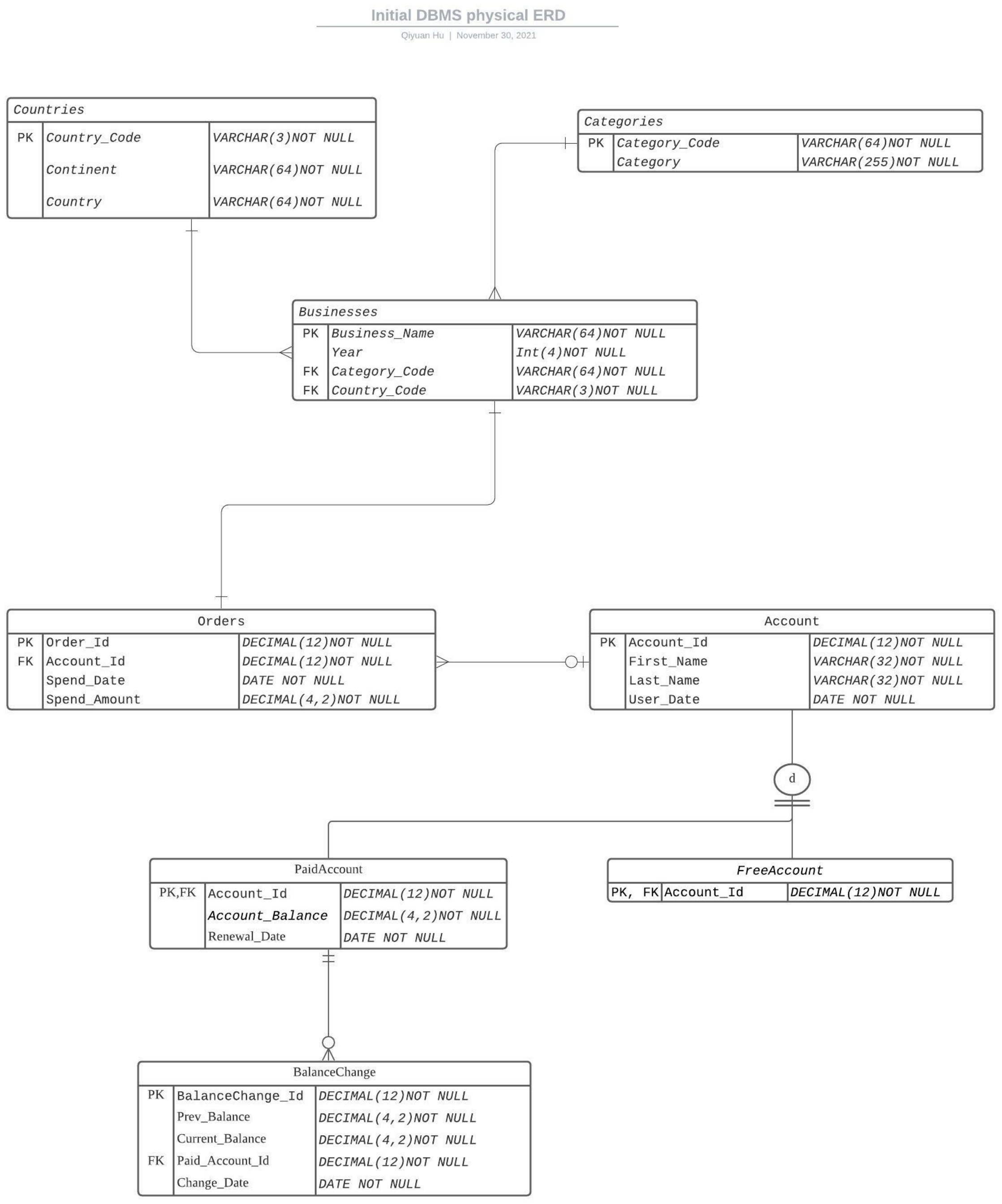
ERD


Initial DBMS physical ERD
Qiyuan Hu | November 30, 2021
Countries
PK Country_Code VARCHAR(3)NOT NULL
Continent VARCHAR(64)NOT NULL
Country VARCHAR(64)NOT NULL
Categories
PK Category_Code VARCHAR(64)NOT NULL
Category VARCHAR(255)NOT NULL
Businesses
PK Business_Name VARCHAR(64)NOT NULL
Year Int(4)NOT NULL
FK Category_Code VARCHAR(64)NOT NULL
FK Country_Code VARCHAR(3)NOT NULL
Orders
PK Order_Id DECIMAL(12)NOT NULL
FK Account_Id DECIMAL(12)NOT NULL
Spend_Date DATE NOT NULL
Spend_Amount DECIMAL(4,2)NOT NULL
ACcount
PK Account_Id DECIMAL(12)NOT NULL
First_Name VARCHAR(32)NOT NULL
Last_Name VARCHAR(32)NOT NULL
User_Date DATE NOT NULL
Account_Balance DECIMAL(4,2)NOT NULL

ERD after normalization

The BalanceChange entity is present and linked to PaidAccount.
The new ERD will show in the next stage with added attributes.

[redacted]



Countries

PK	Country_Code	VARCHAR(3)NOT NULL
	Continent	VARCHAR(64)NOT NULL
	Country	VARCHAR(64)NOT NULL

Categories

PK	Category_Code	VARCHAR(64)NOT NULL
	Category	VARCHAR(255)NOT NULL

Businesses

PK	Business_Name	VARCHAR(64)NOT NULL
	Year	Int(4)NOT NULL
FK	Category_Code	VARCHAR(64)NOT NULL
FK	Country_Code	VARCHAR(3)NOT NULL

Orders

PK	Order_Id	DECIMAL(12)NOT NULL
FK	Account_Id	DECIMAL(12)NOT NULL
	Spend_Date	DATE NOT NULL
	Spend_Amount	DECIMAL(4,2)NOT NULL

Account

PK	Account_Id	DECIMAL(12)NOT NULL
	First_Name	VARCHAR(32)NOT NULL
	Last_Name	VARCHAR(32)NOT NULL
	User_Date	DATE NOT NULL

PaidAccount

PK,FK	Account_Id	DECIMAL(12)NOT NULL
	Account_Balance	DECIMAL(4,2)NOT NULL
	Renewal_Date	DATE NOT NULL

FreeAccount

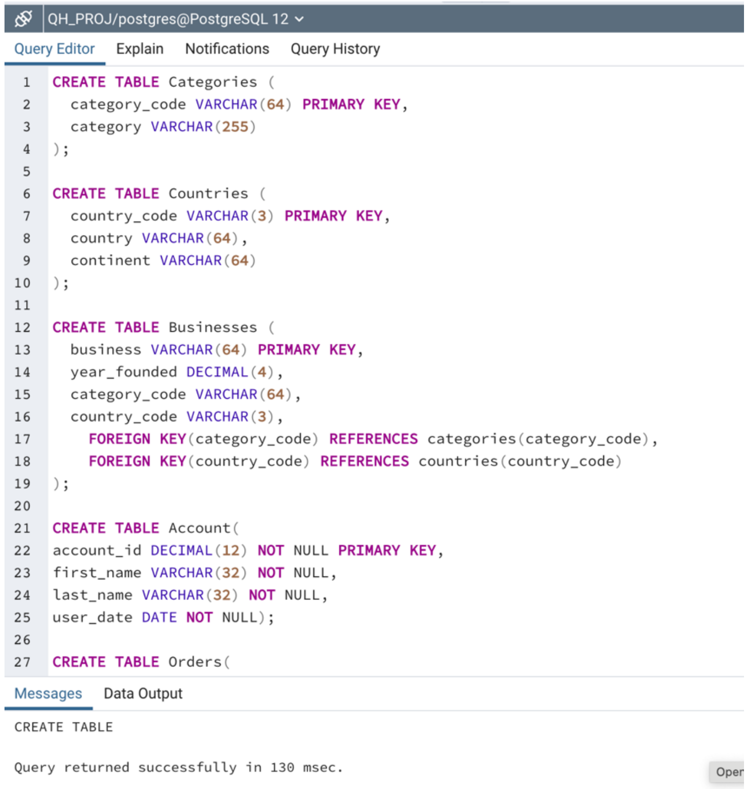
PK, FK	Account_Id	DECIMAL(12)NOT NULL

BalanceChange

PK	BalanceChange_Id	DECIMAL(12)NOT NULL
	Prev_Balance	DECIMAL(4,2)NOT NULL
	Current_Balance	DECIMAL(4,2)NOT NULL
FK	Paid_Account_Id	DECIMAL(12)NOT NULL
	Change_Date	DATE NOT NULL

[redacted]

Index Identification and Creations



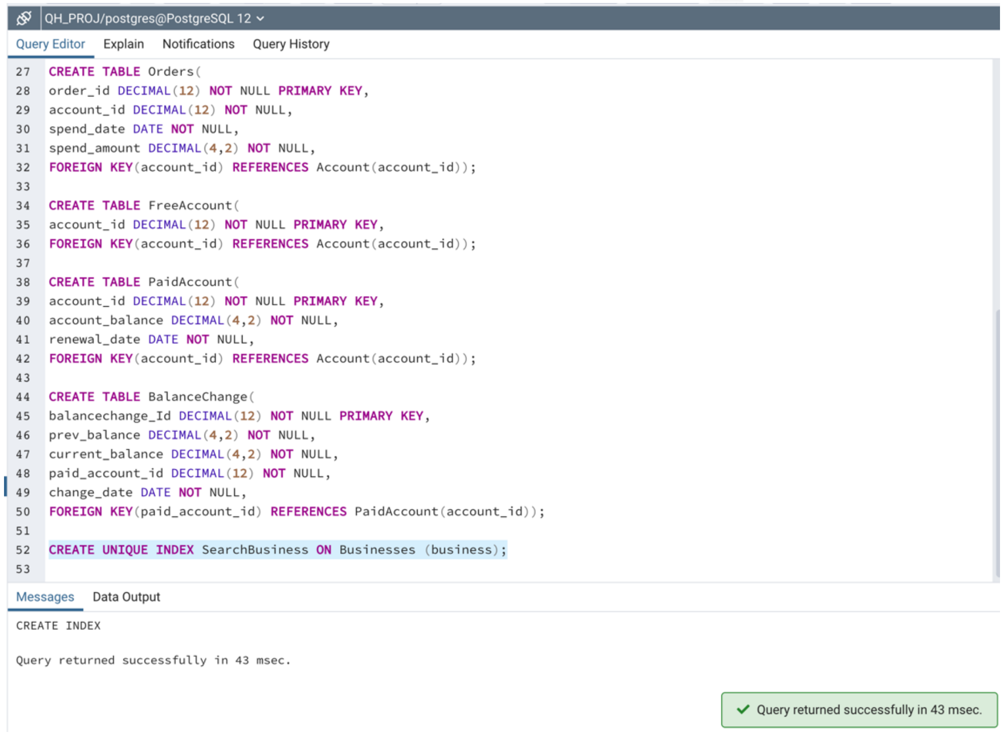
QH_PROJ/postgres@PostgreSQL 12

Query Editor Explain Notifications Query History

```
CREATE TABLE Categories (
  category_code VARCHAR(64) PRIMARY KEY,
  category VARCHAR(255)
);

CREATE TABLE Countries (
  country_code VARCHAR(3) PRIMARY KEY,
  country VARCHAR(64),
  continent VARCHAR(64)
);

CREATE TABLE Businesses (
  business VARCHAR(64) PRIMARY KEY,
  year_founded DECIMAL(4),
  category_code VARCHAR(64),
  country_code VARCHAR(3),
    FOREIGN KEY(category_code) REFERENCES categories(category_code),
    FOREIGN KEY(country_code) REFERENCES countries(country_code)
);

CREATE TABLE Account(
account_id DECIMAL(12) NOT NULL PRIMARY KEY,
first_name VARCHAR(32) NOT NULL,
last_name VARCHAR(32) NOT NULL,
user_date DATE NOT NULL);

CREATE TABLE Orders(
```

Messages Data Output

```
CREATE TABLE

Query returned successfully in 130 msec.
```



QH_PROJ/postgres@PostgreSQL 12

Query Editor Explain Notifications Query History

```
CREATE TABLE Orders(
order_id DECIMAL(12) NOT NULL PRIMARY KEY,
account_id DECIMAL(12) NOT NULL,
spend_date DATE NOT NULL,
spend_amount DECIMAL(4,2) NOT NULL,
FOREIGN KEY(account_id) REFERENCES Account(account_id));

CREATE TABLE FreeAccount(
account_id DECIMAL(12) NOT NULL PRIMARY KEY,
FOREIGN KEY(account_id) REFERENCES Account(account_id));

CREATE TABLE PaidAccount(
account_id DECIMAL(12) NOT NULL PRIMARY KEY,
account_balance DECIMAL(4,2) NOT NULL,
renewal_date DATE NOT NULL,
FOREIGN KEY(account_id) REFERENCES Account(account_id));

CREATE TABLE BalanceChange(
balancechange_Id DECIMAL(12) NOT NULL PRIMARY KEY,
prev_balance DECIMAL(4,2) NOT NULL,
current_balance DECIMAL(4,2) NOT NULL,
paid_account_id DECIMAL(12) NOT NULL,
change_date DATE NOT NULL,
FOREIGN KEY(paid_account_id) REFERENCES PaidAccount(account_id));

CREATE UNIQUE INDEX SearchBusiness ON Businesses (business);
```

Messages Data Output

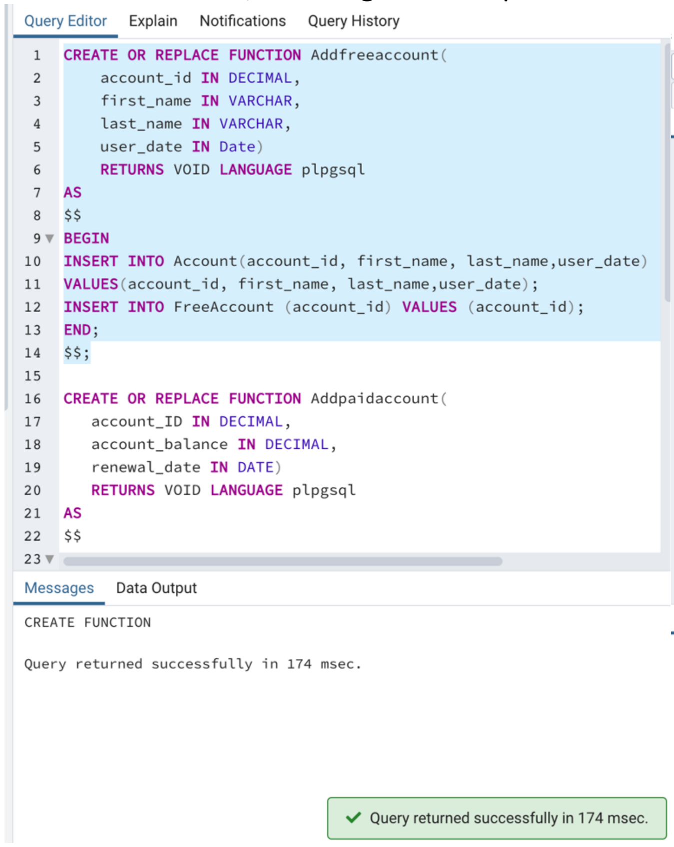
```
CREATE INDEX

Query returned successfully in 43 msec.
```

Query returned successfully in 43 msec.

[redacted]

Stored Procedure Execution and Explanations

It’s not necessary for the application to connect over the network repeatedly to execute SQL, resulting in better performance.



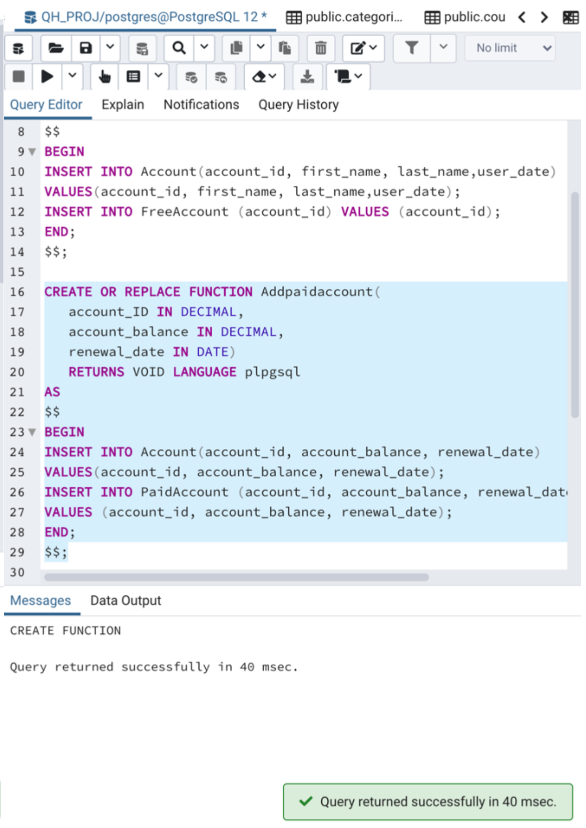
Query Editor Explain Notifications Query History

```
CREATE OR REPLACE FUNCTION Addfreeaccount(
    account_id IN DECIMAL,
    first_name IN VARCHAR,
    last_name IN VARCHAR,
    user_date IN Date)
    RETURNS VOID LANGUAGE plpgsql
AS
$$
BEGIN
INSERT INTO Account(account_id, first_name, last_name,user_date)
VALUES(account_id, first_name, last_name,user_date);
INSERT INTO FreeAccount (account_id) VALUES (account_id);
END;
$$;

CREATE OR REPLACE FUNCTION Addpaidaccount(
   account_ID IN DECIMAL,
   account_balance IN DECIMAL,
   renewal_date IN DATE)
   RETURNS VOID LANGUAGE plpgsql
AS
$$
```

Messages Data Output

```
CREATE FUNCTION

Query returned successfully in 174 msec.
```

Query returned successfully in 174 msec.



QH_PROJ/postgres@PostgreSQL 12 * public.categori... public.cou

No limit

Query Editor Explain Notifications Query History

```
$$
BEGIN
INSERT INTO Account(account_id, first_name, last_name,user_date)
VALUES(account_id, first_name, last_name,user_date);
INSERT INTO FreeAccount (account_id) VALUES (account_id);
END;
$$;

CREATE OR REPLACE FUNCTION Addpaidaccount(
   account_ID IN DECIMAL,
   account_balance IN DECIMAL,
   renewal_date IN DATE)
   RETURNS VOID LANGUAGE plpgsql
AS
$$
BEGIN
INSERT INTO Account(account_id, account_balance, renewal_date)
VALUES(account_id, account_balance, renewal_date);
INSERT INTO PaidAccount (account_id, account_balance, renewal_dat
VALUES (account_id, account_balance, renewal_date);
END;
$$;
```

Messages Data Output

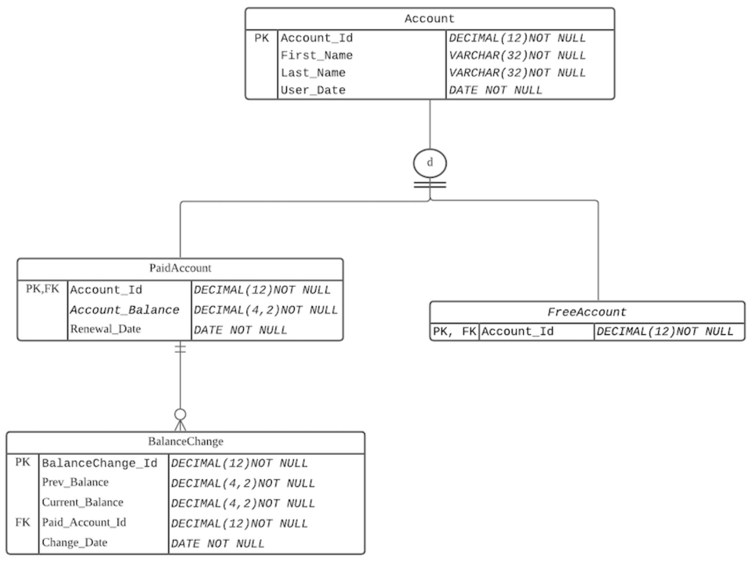
```
CREATE FUNCTION

Query returned successfully in 40 msec.
```

Query returned successfully in 40 msec.

Trigger Creation and Use

Maintaining History Tables with Triggers

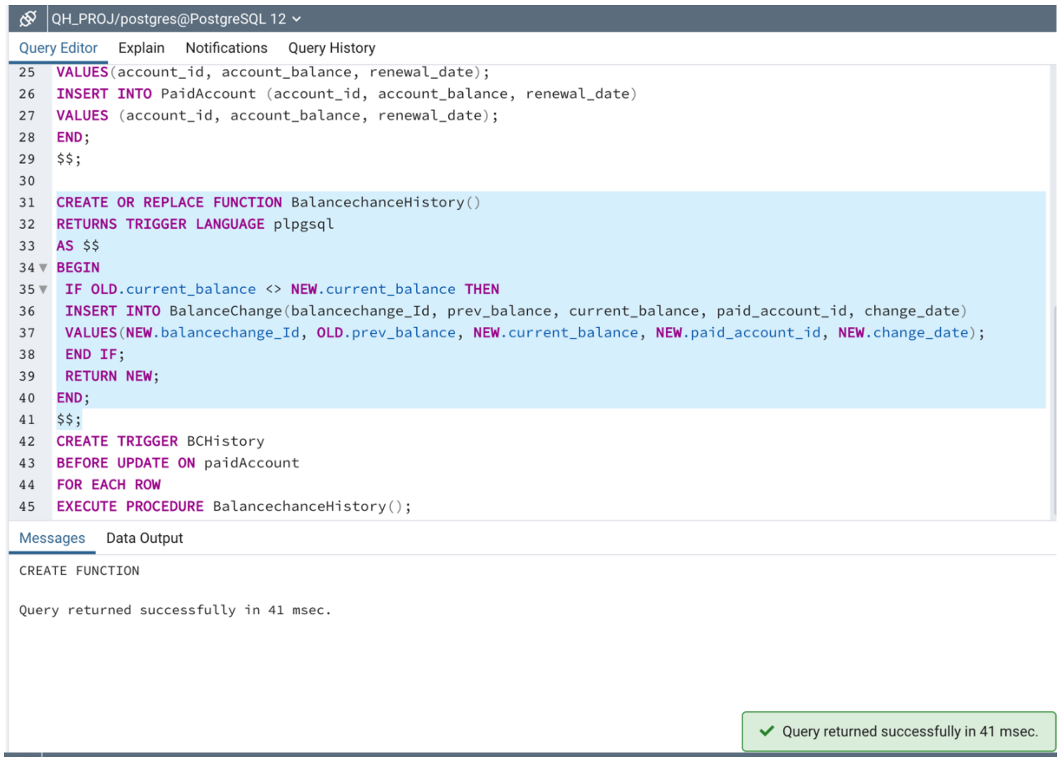
For some data, we’re only concerned with its current value. A standard history table contains the old and new value(s) for the column(s) being tracked, a foreign key to the table being tracked, and the date of the change.


Account
PK Account_Id DECIMAL(12)NOT NULL
First_Name VARCHAR(32)NOT NULL
Last_Name VARCHAR(32)NOT NULL
User_Date DATE NOT NULL
d
PaidAccount
PK,FK Account_Id DECIMAL(12)NOT NULL
Account_Balance DECIMAL(4,2)NOT NULL
Renewal_Date DATE NOT NULL
FreeAccount
PK, FK Account_Id DECIMAL(12)NOT NULL
BalanceChange
PK BalanceChange_Id DECIMAL(12)NOT NULL
Prev_Balance DECIMAL(4,2)NOT NULL
Current_Balance DECIMAL(4,2)NOT NULL
FK Paid_Account_Id DECIMAL(12)NOT NULL
Change_Date DATE NOT NULL

The added trigger is explained step by step in the following table.

CREATE OR REPLACE FUNCTION BalancechanceHistory() RETURNS TRIGGER LANGUAGE plpgsql AS $$	The BalancechanceHistory() is created and links to PaidAccount table, which specifically indicates that the trigger will run before any update on table.
BEGIN IF OLD.current_balance <> NEW.current_balance THEN	When current balance is different from old one, we execute the following query.
INSERT INTO BalanceChange(balancechange_Id, prev_balance, current_balance, paid_account_id, change_date) VALUES(NEW.balancechange_Id, OLD.prev_balance, NEW.current_balance, NEW.paid_account_id, NEW.change_date);	This is the insert statement that records the balance change by adding a row into the balancechange table. The old and new balance as already saved in the variables are used.
END IF; RETURN NEW; END; $$;	This ends the trigger definition.

[redacted]



QH_PROJ/postgres@PostgreSQL 12

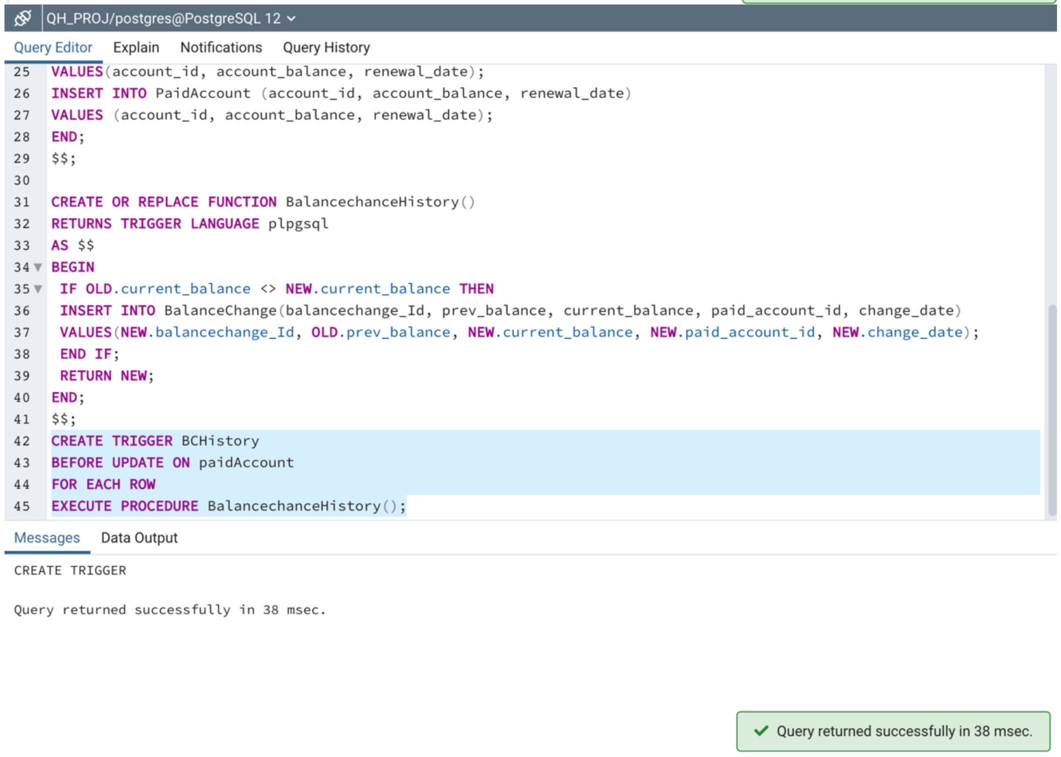
Query Editor Explain Notifications Query History

```
VALUES(account_id, account_balance, renewal_date);
INSERT INTO PaidAccount (account_id, account_balance, renewal_date)
VALUES (account_id, account_balance, renewal_date);
END;
$$;

CREATE OR REPLACE FUNCTION BalancechanceHistory()
RETURNS TRIGGER LANGUAGE plpgsql
AS $$
BEGIN
 IF OLD.current_balance <> NEW.current_balance THEN
 INSERT INTO BalanceChange(balancechange_Id, prev_balance, current_balance, paid_account_id, change_date)
 VALUES(NEW.balancechange_Id, OLD.prev_balance, NEW.current_balance, NEW.paid_account_id, NEW.change_date);
 END IF;
 RETURN NEW;
END;
$$;
CREATE TRIGGER BCHistory
BEFORE UPDATE ON paidAccount
FOR EACH ROW
EXECUTE PROCEDURE BalancechanceHistory();
```

Messages Data Output

```
CREATE FUNCTION

Query returned successfully in 41 msec.
```

✓ Query returned successfully in 41 msec.



QH_PROJ/postgres@PostgreSQL 12

Query Editor Explain Notifications Query History

```
VALUES(account_id, account_balance, renewal_date);
INSERT INTO PaidAccount (account_id, account_balance, renewal_date)
VALUES (account_id, account_balance, renewal_date);
END;
$$;

CREATE OR REPLACE FUNCTION BalancechanceHistory()
RETURNS TRIGGER LANGUAGE plpgsql
AS $$
BEGIN
 IF OLD.current_balance <> NEW.current_balance THEN
 INSERT INTO BalanceChange(balancechange_Id, prev_balance, current_balance, paid_account_id, change_date)
 VALUES(NEW.balancechange_Id, OLD.prev_balance, NEW.current_balance, NEW.paid_account_id, NEW.change_date);
 END IF;
 RETURN NEW;
END;
$$;
CREATE TRIGGER BCHistory
BEFORE UPDATE ON paidAccount
FOR EACH ROW
EXECUTE PROCEDURE BalancechanceHistory();
```

Messages Data Output

```
CREATE TRIGGER

Query returned successfully in 38 msec.
```

✓ Query returned successfully in 38 msec.

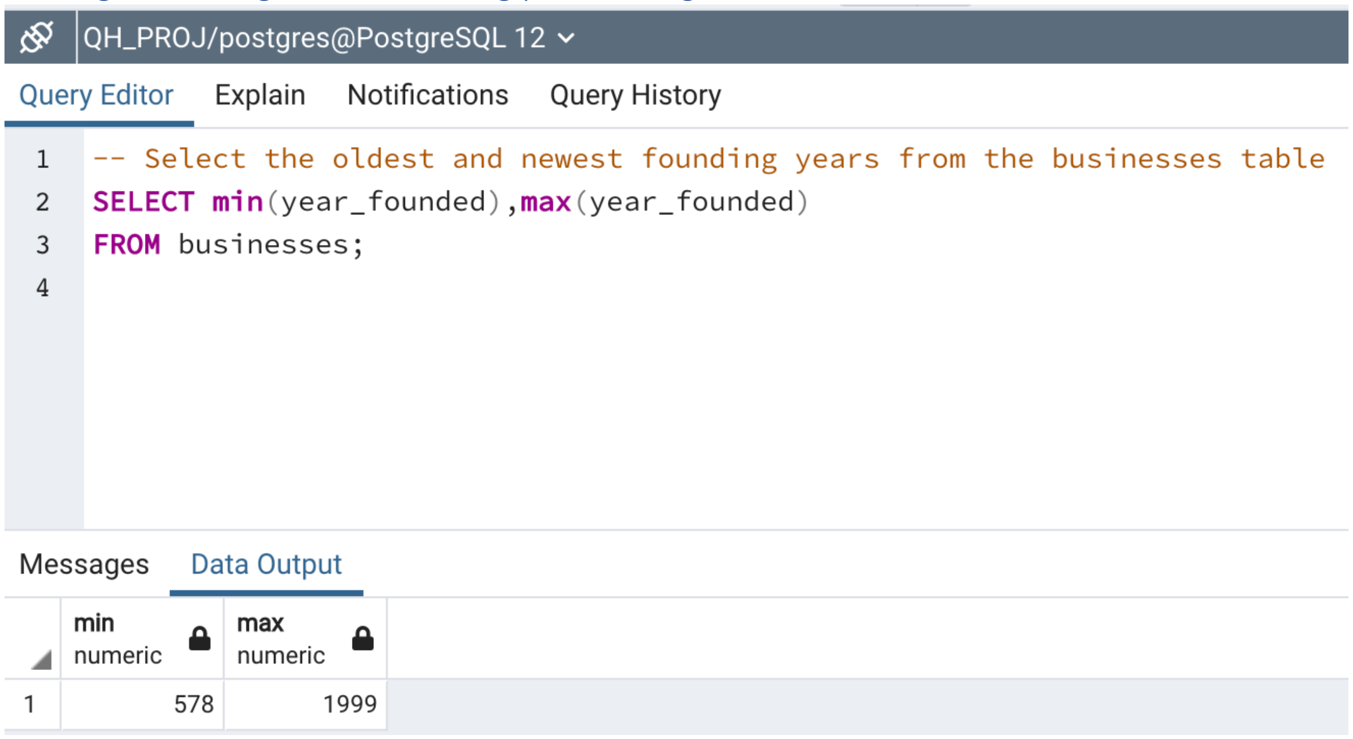
[redacted]

Question Identification and Explanations

An important part of business is planning for the future and ensuring that the company survives changing market conditions. Some businesses do this really well and last for hundreds of years. he oldest company that is still in business in (almost) every country and compiled the results into a dataset. In this project, I'll explore that dataset to see what they found.

Query Executions and Explanations

Looking at the range of the founding years throughout the world.



```sql
-- Select the oldest and newest founding years from the businesses table
SELECT min(year_founded),max(year_founded)
FROM businesses;
```

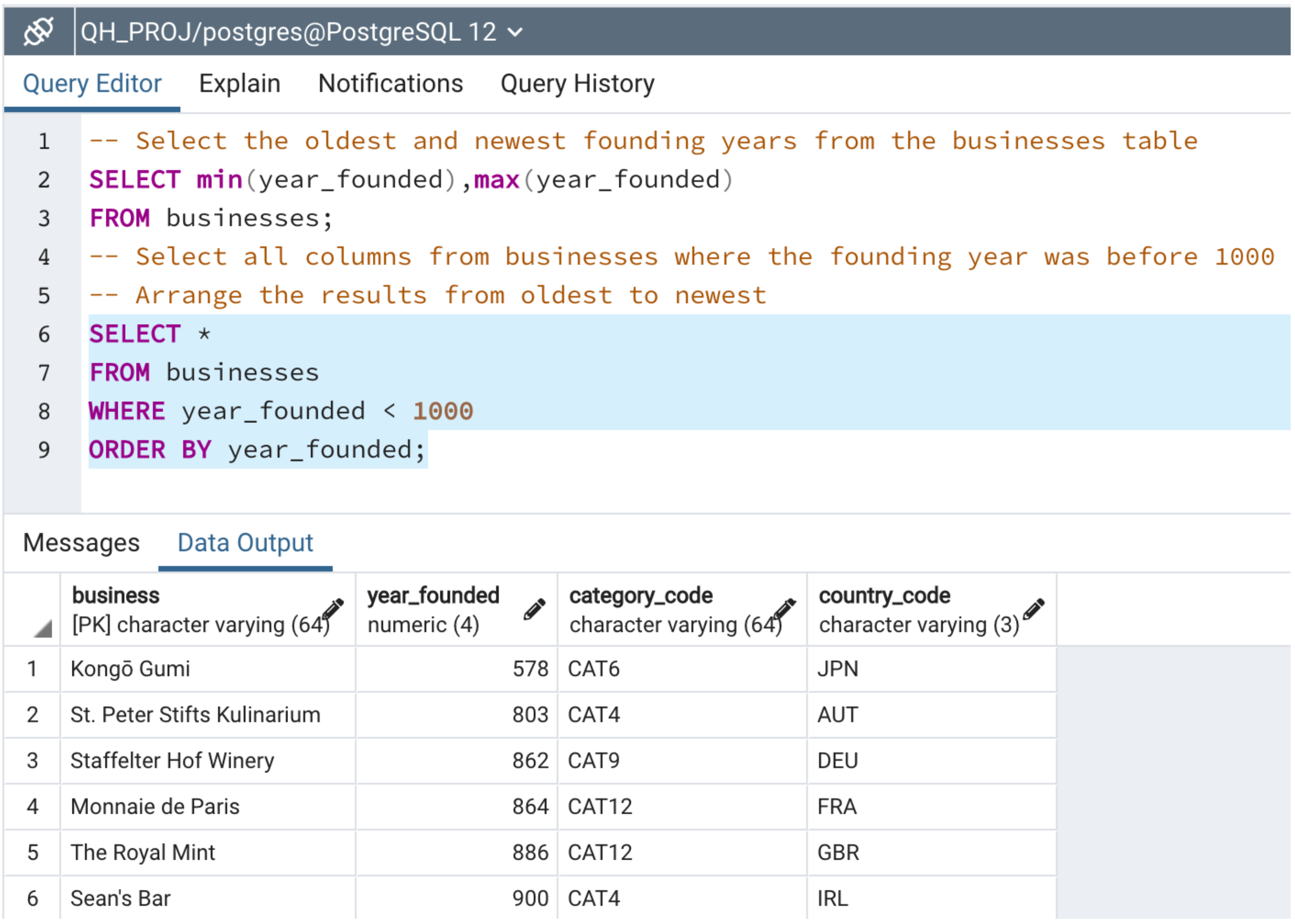
	min numeric	max numeric
1	578	1999

That's a lot of variation between countries. In one country, the oldest business was only founded in 1999. By contrast, the oldest business in the world was founded back in 578. That's pretty incredible that a business has survived for more than a millennium.

Which businesses were founded before 1000?

Which businesses have been around for more than a millennium?

[redacted]



QH_PROJ/postgres@PostgreSQL 12

Query Editor | Explain | Notifications | Query History

```sql
-- Select the oldest and newest founding years from the businesses table
SELECT min(year_founded),max(year_founded)
FROM businesses;
-- Select all columns from businesses where the founding year was before 1000
-- Arrange the results from oldest to newest
SELECT *
FROM businesses
WHERE year_founded < 1000
ORDER BY year_founded;
```

Messages | Data Output

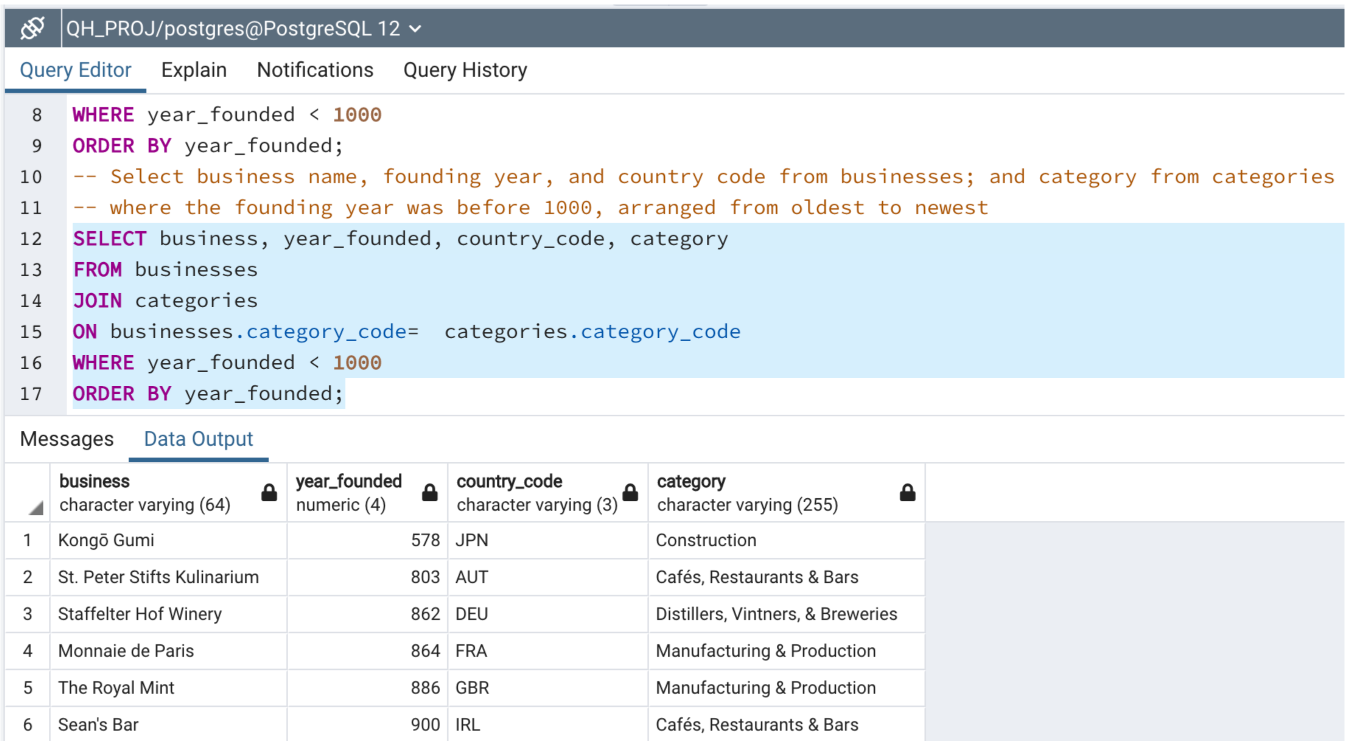
	business [PK] character varying (64)	year_founded numeric (4)	category_code character varying (64)	country_code character varying (3)
1	Kongō Gumi	578	CAT6	JPN
2	St. Peter Stifts Kulinarium	803	CAT4	AUT
3	Staffelter Hof Winery	862	CAT9	DEU
4	Monnaie de Paris	864	CAT12	FRA
5	The Royal Mint	886	CAT12	GBR
6	Sean's Bar	900	CAT4	IRL

Exploring the categories

Now we know that the oldest, continuously operating company in the world is called Kongō Gumi. But was does that company do? The category codes in the businesses table aren't very helpful: the descriptions of the categories are stored in the categories table.

This is a common problem: for data storage, it's better to keep different types of data in different tables, but for analysis, I want all the data in one place. To solve this, I'll have to join the two tables together.

[redacted]


QH_PROJ/postgres@PostgreSQL 12

Query Editor Explain Notifications Query History

```
WHERE year_founded < 1000
ORDER BY year_founded;
-- Select business name, founding year, and country code from businesses; and category from categories
-- where the founding year was before 1000, arranged from oldest to newest
SELECT business, year_founded, country_code, category
FROM businesses
JOIN categories
ON businesses.category_code=  categories.category_code
WHERE year_founded < 1000
ORDER BY year_founded;
```

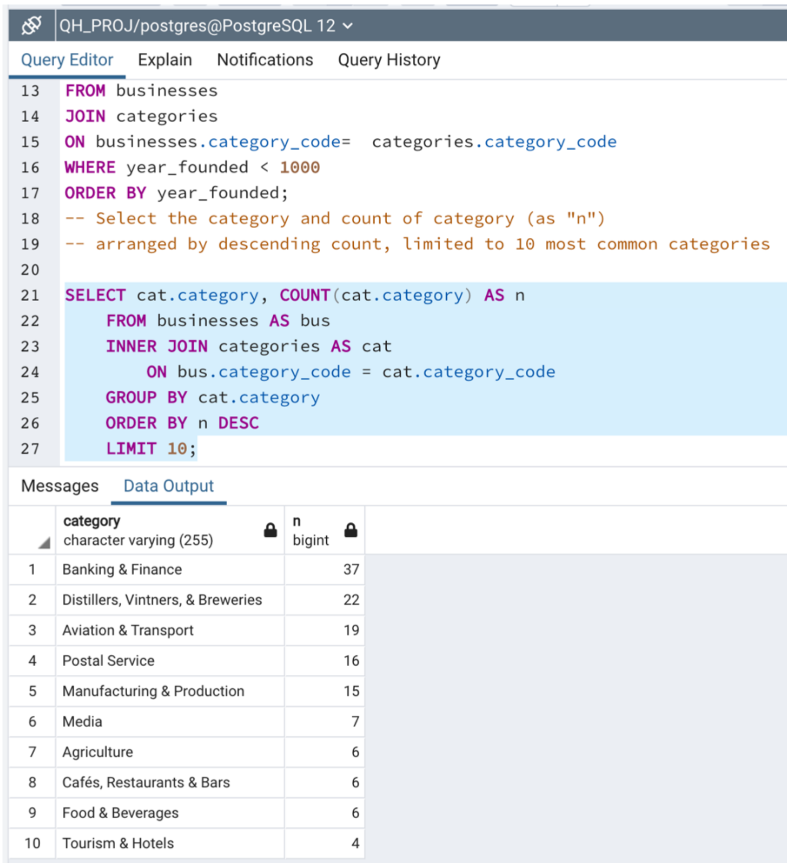
Messages Data Output

	business character varying (64)	year_founded numeric (4)	country_code character varying (3)	category character varying (255)
1	Kongō Gumi	578	JPN	Construction
2	St. Peter Stifts Kulinarium	803	AUT	Cafés, Restaurants & Bars
3	Staffelter Hof Winery	862	DEU	Distillers, Vintners, & Breweries
4	Monnaie de Paris	864	FRA	Manufacturing & Production
5	The Royal Mint	886	GBR	Manufacturing & Production
6	Sean's Bar	900	IRL	Cafés, Restaurants & Bars

Counting the categories

With that extra detail about the oldest businesses, we can see that Kongō Gumi is a construction company. In that list of six businesses, we also see a café, a winery, and a bar. The two companies recorded as "Manufacturing and Production" are both mints. That is, they produce currency.
I'm curious as to what other industries constitute the oldest companies around the world, and which industries are most common.

[redacted]



QH_PROJ/postgres@PostgreSQL 12

Query Editor | Explain | Notifications | Query History

```
FROM businesses
JOIN categories
ON businesses.category_code=  categories.category_code
WHERE year_founded < 1000
ORDER BY year_founded;
-- Select the category and count of category (as "n")
-- arranged by descending count, limited to 10 most common categories

SELECT cat.category, COUNT(cat.category) AS n
    FROM businesses AS bus
    INNER JOIN categories AS cat
        ON bus.category_code = cat.category_code
    GROUP BY cat.category
    ORDER BY n DESC
    LIMIT 10;
```

Messages | Data Output

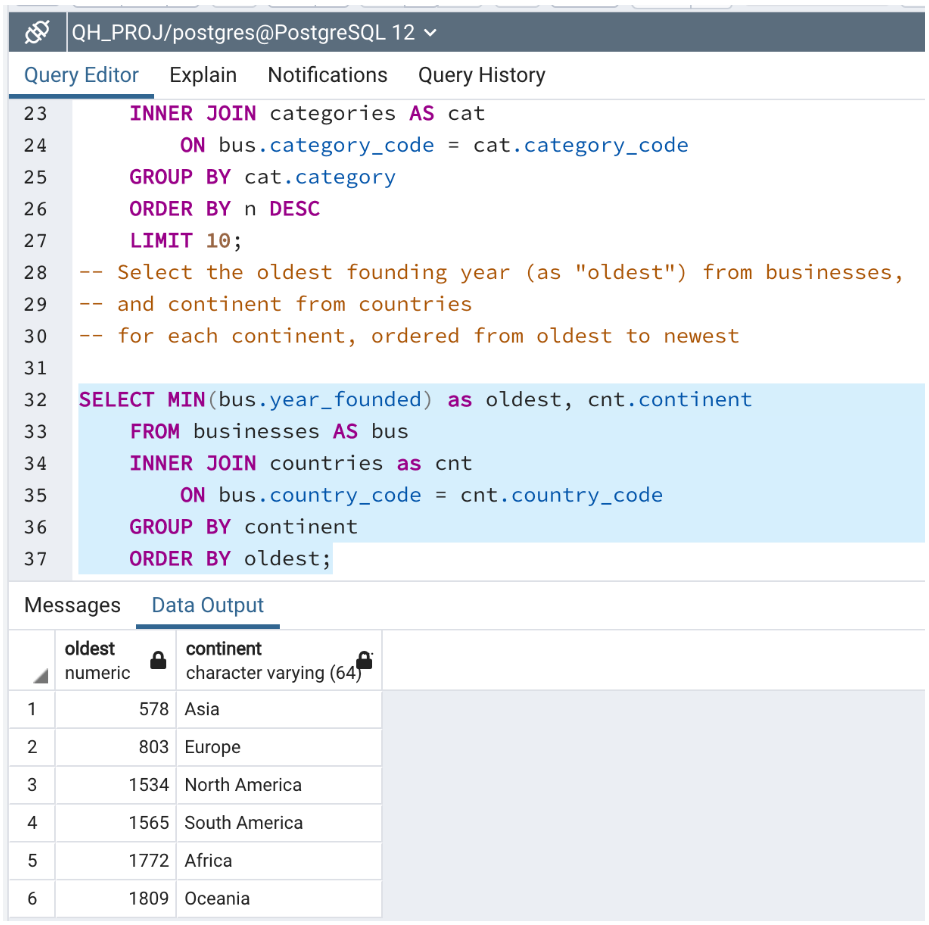
	category character varying (255)	n bigint
1	Banking & Finance	37
2	Distillers, Vintners, & Breweries	22
3	Aviation & Transport	19
4	Postal Service	16
5	Manufacturing & Production	15
6	Media	7
7	Agriculture	6
8	Cafés, Restaurants & Bars	6
9	Food & Beverages	6
10	Tourism & Hotels	4

Oldest business by continent

It looks like "Banking & Finance" is the most popular category. Maybe that's where the client should aim if client wants to start a thousand-year business.

One thing I haven't looked at yet is where in the world these really old businesses are. To answer these questions, I'll need to join the businesses table to the countries table. Let's start by asking how old the oldest business is on each continent.

[redacted]


QH_PROJ/postgres@PostgreSQL 12

Query Editor | Explain | Notifications | Query History

```
    INNER JOIN categories AS cat
        ON bus.category_code = cat.category_code
    GROUP BY cat.category
    ORDER BY n DESC
    LIMIT 10;
-- Select the oldest founding year (as "oldest") from businesses,
-- and continent from countries
-- for each continent, ordered from oldest to newest

SELECT MIN(bus.year_founded) as oldest, cnt.continent
    FROM businesses AS bus
    INNER JOIN countries as cnt
        ON bus.country_code = cnt.country_code
    GROUP BY continent
    ORDER BY oldest;
```

Messages | Data Output

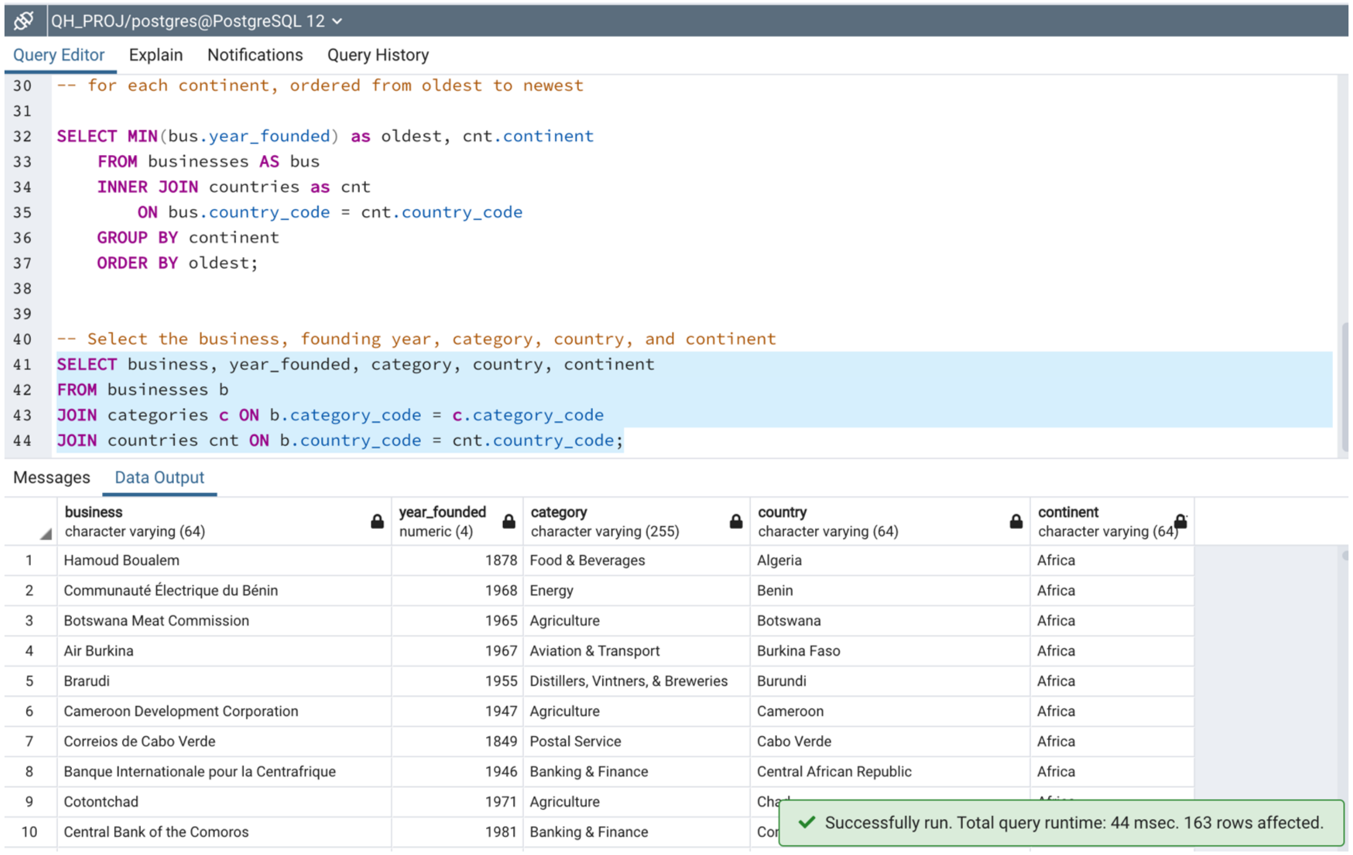
	oldest numeric	continent character varying (64)
1	578	Asia
2	803	Europe
3	1534	North America
4	1565	South America
5	1772	Africa
6	1809	Oceania

Joining everything for further analysis

There's a jump in time from the older businesses in Asia and Europe to the 16th Century oldest businesses in North and South America, then to the 18th and 19th Century oldest businesses in Africa and Oceania.

As mentioned earlier, when analyzing data, it's often really helpful to have all the tables I want access to joined together into a single set of results that can be analyzed further. Here, that means I need to join all three tables.

[redacted]



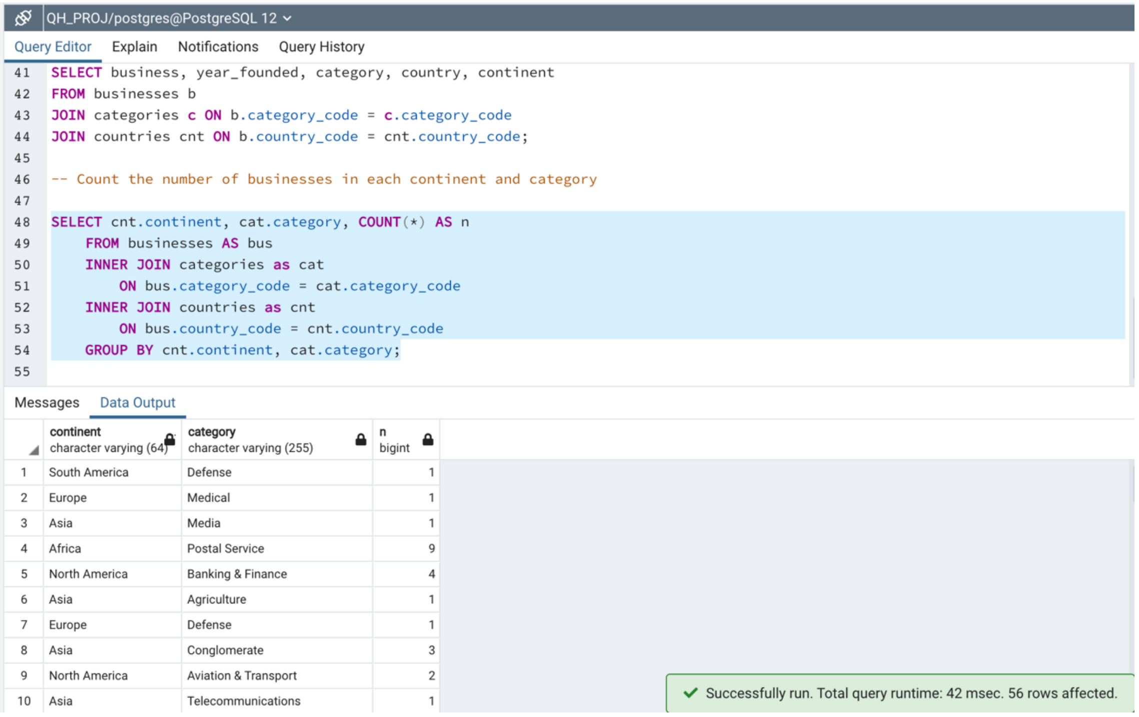
```
-- for each continent, ordered from oldest to newest

SELECT MIN(bus.year_founded) as oldest, cnt.continent
    FROM businesses AS bus
    INNER JOIN countries as cnt
        ON bus.country_code = cnt.country_code
    GROUP BY continent
    ORDER BY oldest;


-- Select the business, founding year, category, country, and continent
SELECT business, year_founded, category, country, continent
FROM businesses b
JOIN categories c ON b.category_code = c.category_code
JOIN countries cnt ON b.country_code = cnt.country_code;
```

	business character varying (64)	year_founded numeric (4)	category character varying (255)	country character varying (64)	continent character varying (64)
1	Hamoud Boualem	1878	Food & Beverages	Algeria	Africa
2	Communauté Électrique du Bénin	1968	Energy	Benin	Africa
3	Botswana Meat Commission	1965	Agriculture	Botswana	Africa
4	Air Burkina	1967	Aviation & Transport	Burkina Faso	Africa
5	Brarudi	1955	Distillers, Vintners, & Breweries	Burundi	Africa
6	Cameroon Development Corporation	1947	Agriculture	Cameroon	Africa
7	Correios de Cabo Verde	1849	Postal Service	Cabo Verde	Africa
8	Banque Internationale pour la Centrafrique	1946	Banking & Finance	Central African Republic	Africa
9	Cotontchad	1971	Agriculture	Cha	[illegible]
10	Central Bank of the Comoros	1981	Banking & Finance	Cor	[illegible]

✓ Successfully run. Total query runtime: 44 msec. 163 rows affected.

Counting categories by continent

Having businesses joined to categories and countries together means I can ask questions about both these things together. For example, which are the most common categories for the oldest businesses on each continent?



```
SELECT business, year_founded, category, country, continent
FROM businesses b
JOIN categories c ON b.category_code = c.category_code
JOIN countries cnt ON b.country_code = cnt.country_code;

-- Count the number of businesses in each continent and category

SELECT cnt.continent, cat.category, COUNT(*) AS n
    FROM businesses AS bus
    INNER JOIN categories as cat
        ON bus.category_code = cat.category_code
    INNER JOIN countries as cnt
        ON bus.country_code = cnt.country_code
    GROUP BY cnt.continent, cat.category;

```

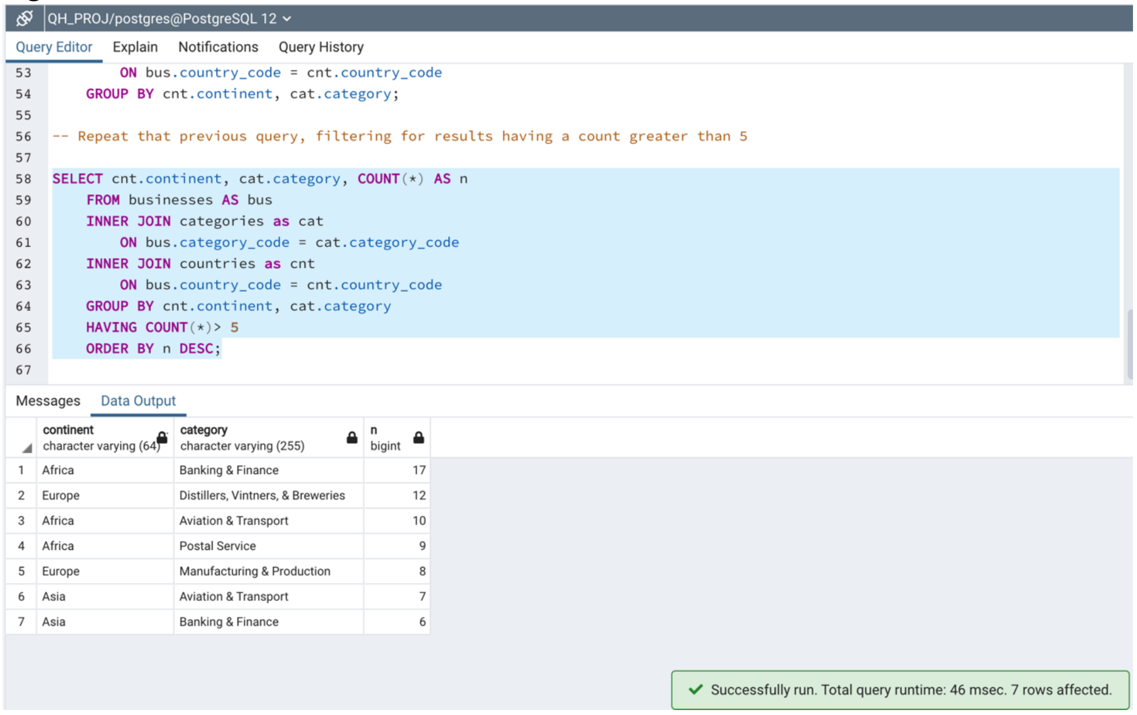
	continent character varying (64)	category character varying (255)	n bigint
1	South America	Defense	1
2	Europe	Medical	1
3	Asia	Media	1
4	Africa	Postal Service	9
5	North America	Banking & Finance	4
6	Asia	Agriculture	1
7	Europe	Defense	1
8	Asia	Conglomerate	3
9	North America	Aviation & Transport	2
10	Asia	Telecommunications	1

✓ Successfully run. Total query runtime: 42 msec. 56 rows affected.

[redacted]

Filtering counts by continent and category

Combining continent and business category led to a lot of results. It's difficult to see what is important. To trim this down to a manageable size, let's restrict the results to only continent/category pairs with a high count.



QH_PROJ/postgres@PostgreSQL 12

Query Editor | Explain | Notifications | Query History

```
        ON bus.country_code = cnt.country_code
    GROUP BY cnt.continent, cat.category;

-- Repeat that previous query, filtering for results having a count greater than 5

SELECT cnt.continent, cat.category, COUNT(*) AS n
    FROM businesses AS bus
    INNER JOIN categories as cat
        ON bus.category_code = cat.category_code
    INNER JOIN countries as cnt
        ON bus.country_code = cnt.country_code
    GROUP BY cnt.continent, cat.category
    HAVING COUNT(*)> 5
    ORDER BY n DESC;
```

Messages | Data Output

	continent character varying (64)	category character varying (255)	n bigint
1	Africa	Banking & Finance	17
2	Europe	Distillers, Vintners, & Breweries	12
3	Africa	Aviation & Transport	10
4	Africa	Postal Service	9
5	Europe	Manufacturing & Production	8
6	Asia	Aviation & Transport	7
7	Asia	Banking & Finance	6

Successfully run. Total query runtime: 46 msec. 7 rows affected.

Summary and Reflection

The work I have completed thus far

I understand the stored procedures and triggers deals with the way the database, how long that will take, and how much work the database must perform to do so. Creating triggers and thinking in terms of implementation. From this CS669, I will know how to create the proper procedures and make query safer and faster.

My questions, concerns, and observations

My database is going to be intact with a web application named business lifespan. It will have all the lifespan of business worldwide. Having businesses joined to categories and countries together means we can ask questions about both these things together. An index can be implemented to allow the key values to repeat or can be implemented to disallow repeating keys. The design also contains a hierarchy of PaidAccount and FreeAccount to reflect the fact that people can sign up for a free account or a paid account for web application named business lifespan.

From 5 iterations, I understood language and diagrams, design and build database from scratch to my own relational database. The SQL script that creates all tables follows the specification from the DBMS physical ERD exactly. Important indexes have been created to help speed up access to my database and are also available in an index script.

From reviewing my ERD, after normalization, I should have at least 8 entities in my DBMS physical ERD to support the minimal complexity requirements for the term project. Then I simply add FreeAccount, PaidAccount, and BalanceChange. Then carry the impact through my design into the structural database rules, conceptual ERD, and DBMS physical ERD. An organization needs queries that get it the data it needs. It's hard to believe how much I learn and be able to establish an individual project. There is still more to explore and develop in my database.

I'd appreciate any correction on improving this.