Exercises: PostgreSQL Table Relations

This document defines the **exercise assignments** for the PostgreSQL course @ Software University.

Submit your solutions in the SoftUni Judge Contest.

For the upcoming assignments, let us direct our attention toward the implementation of **PRIMARY KEY**.

1. PRIMARY KEY

a. Create a table

Create a new table called **"products"** that includes a column named **"product_name"** which has **a maximum character limit of 100**.

Insert the following values into the **"products"** table: **'Broccoli'**, **'Shampoo'**, **'Toothpaste'**, and **'Candy'**.

Example

product_name character varying (100)
Broccoli
Shampoo
Toothpaste
Candy

b. Define the primary key when changing the existing table structure

The newly created table **does not have a unique identifier**. To add a **PRIMARY KEY**, use the **ALTER TABLE** statement.

Submit your queries for the two-step task in the Judge system.

Example

product_name character varying (100)	id [PK] integer
Broccoli	1
Shampoo	2
Toothpaste	3
Candy	4

2. Remove Primary Key

Write an SQL statement to **DROP CONSTRAINT** from the **"products"** table.

*** Note, if a name is not specified explicitly for the primary key constraint, PostgreSQL will assign a default name to it. The default name for the primary key constraint is **"table-name_pkey"**.

Submit your query for this task in the Judge system.

Example

product_name character varying (100)	id integer
Broccoli	1
Shampoo	2
Toothpaste	3
Candy	4

⚠ In preparation for the upcoming task, we will adopt the subsequent **naming standard** for our **foreign keys**:

`fk_<referencing_table>_<referenced_table>`

Here's the breakdown of each element:

- **`"fk"`**: serves as an indicator that the constraint relates to a foreign key
- **`"<referencing_table>"`**: represents the table containing the column designated as the foreign key
- **`"<referenced_table>"`**: signifies the table to which the foreign key refers

*Let's redirect our attention to the **One-To-One Relationship** for the tasks ahead.*

3. Customs

To fulfill this assignment, you are required to establish a new **database** named **`"customs_db"`** and produce two tables inside it, named **`"passports"`** and **`"people"`**.

a. Create and Insert Passports Table

Wright a SQL statement to create a new table called **`"passports"`** with two columns, **`"id"`** and **`"nationality"`**. The **`"id"`** column should be an **automatically incremented primary key, starting at 100 and incrementing by 1**. The **`"nationality"`** column should have a **maximum character limit of 50**.

Then, insert three rows into the **`"passports"`** table with values **`'N34FG21B'`**, **`'K65LO4R7'`**, and **`'ZE657QP2'`** for the **`"nationality"`** column.

*** Note, when using the **`GENERATED AS IDENTITY`** constraint, a **SEQUENCE** object is utilized, which allows for the **specification of sequence options for system-generated values**. The following syntax can be used to specify these options: **`(START WITH start_from_number INCREMENT BY increment_value)`**.

Example

id	nationality
100	N34FG21B
101	K65LO4R7
102	ZE657QP2

b. Create and Insert People Table

In the next step of this task, your objective is to create a new table called **`"people"`** which includes the following columns:

- **"id"** column that is an **automatically incremented SERIAL PRIMARY KEY**;
- **"first_name"** column with a **maximum length of 50** characters and is of type **VARCHAR**;
- **"salary"** column which is specified to the **second decimal place** and has a **maximum of 10 digits**;
- **"passport_id"** column of type **INT** which is established as a **FOREIGN KEY** constraint and **refers** to the **"id"** column of the **"passports"** table (the name of a foreign key constraint is **"fk_people_passports"**).

After creating the **"people"** table, you need to insert three rows into it, each row should have values for the **"first_name"**, **"salary"**, and **"passport_id"** columns:

- **('Roberto', 43300.0000, 101)**
- **('Tom', 56100.0000, 102)**
- **('Yana', 60200.0000, 100)**

Submit your queries for the two-step task in the Judge system.

Example

id	first_name	salary	passport_id
1	Roberto	43300.00	101
1	Tom	56100.00	102
2	Yana	60200.00	100

*It is time to focus our attention on the **One-To-Many/Many-To-One Relationship** for the upcoming assignments.*

4. Car Manufacture

To complete this task, you will need to create a **database** called **"car_manufacture_db"** that consists of **three tables**: **"manufacturers"**, **"models"**, and **"production_years"**.

a. Create Tables

The **"manufacturers"** table should have a column, which is **"name"**. In the **"models"** table, you should include columns for **"model_name"** and **"manufacturer_id"**. The **"production_years"** table should contain information about **"established_on"** and **"manufacturer_id"**. You are free to **select the data type** for each column, but it is crucial to ensure that each column has a **unique identifier**. Additionally, it is important to **correctly set up the foreign keys**. Note that the **"models"** identifier should **start at 1000 and increment by 1**.

b. Insert Data

Add data to the tables in the following manner:

manufacturers
name
BMW
Tesla
Lada

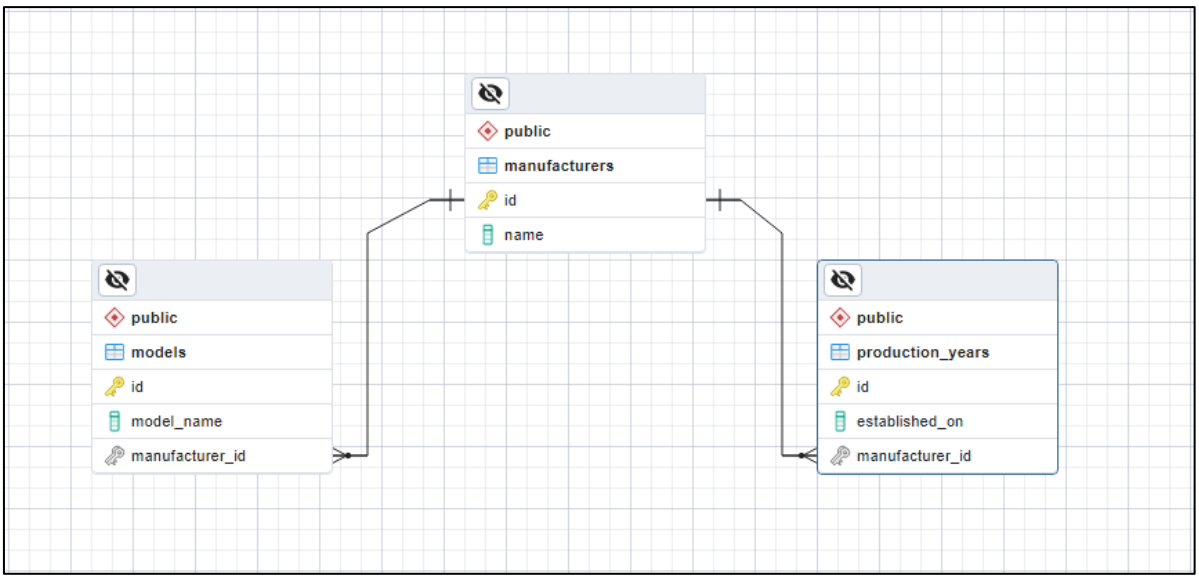
models	
model_name	**manufacturer_id**
X1	1
i6	1
Model S	2
Model X	2
Model 3	2
Nova	3

production_years	
established_on	**manufacturer_id**
1916-03-01	1
2003-01-01	2
1966-05-01	3

Submit your queries for the two-step task in the Judge system.

5. Car Manufacture E/R Diagram**

Generate **Entity/Relationship Diagram** for the three tables created in the **Car Manufacture** task.



6. Photo Shooting

To finish this assignment, you need to create a database called **"photo_shooting_db"** that comprises two tables: **"customers"** and **"photos"**.

The **"customers"** table should contain two columns - **"name"** and **"date"**. In the **"photos"** table, you should include columns for **"url"** and **"place"**. The **data type** for each column can be **chosen according to your preference**, but it is essential to ensure that each column has a **unique identifier**. Moreover, correctly setting up the **foreign keys** is crucial.

Insert data into the tables in the format shown below:

customers	
name	**date**
Bella	2022-03-25
Philip	2022-07-05

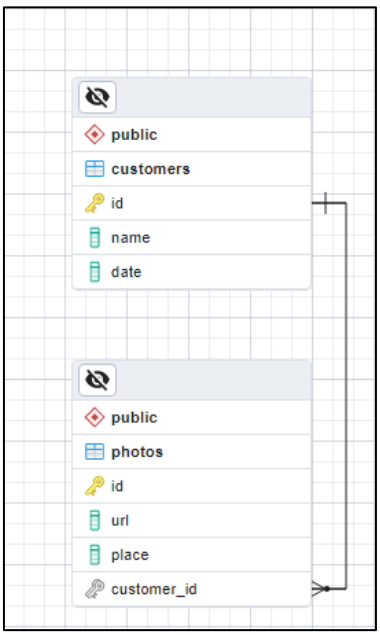
photos		
url	**place**	**customer_id**
bella_1111.com	National Theatre	1
bella_1112.com	Largo	1
bella_1113.com	The View Restaurant	1

philip_1121.com	Old Town	2
philip_1122.com	Rowing Canal	2
philip_1123.com	Roman Theater	2

Submit your query for this task in the Judge system.

7. Photo Shooting E/R Diagram**

Create **Entity/Relationship Diagram** for the **"customers"** and **"photos"** tables that were created in the previous task.



*Let's focus on the **Many-To-Many Relationship** for the upcoming assignments.*

8. Study Session

To complete this task, create a database named **"study_session_db"** that includes the following tables: **"students"**, **"exams"**, **"study_halls"** and **"students_exams"**.

The **"students"** table should have a column for **"student_name"**. In the **"exams"** table, include a column for **"exam_name"**. The **"study_halls"** table should contain columns for **"study_hall_name"** and **"exam_id"**. The **"students_exams"** table should have columns for **"student_id"** and **"exam_id"**. You are free to choose the **appropriate data type** for each column but ensure that each column has a **unique identifier**. Note that the **"exams"** identifier should **start at 101 and increment by 1**. It is important to correctly set up the **foreign keys**.

Insert data into the tables in the format shown below:

students
student_name
Mila
Toni

exams
exam_name
Python Advanced
Python OOP

study_halls	
study_hall_name	exam_id
Open Source Hall	102
Inspiration Hall	101

Ron

PostgreSQL

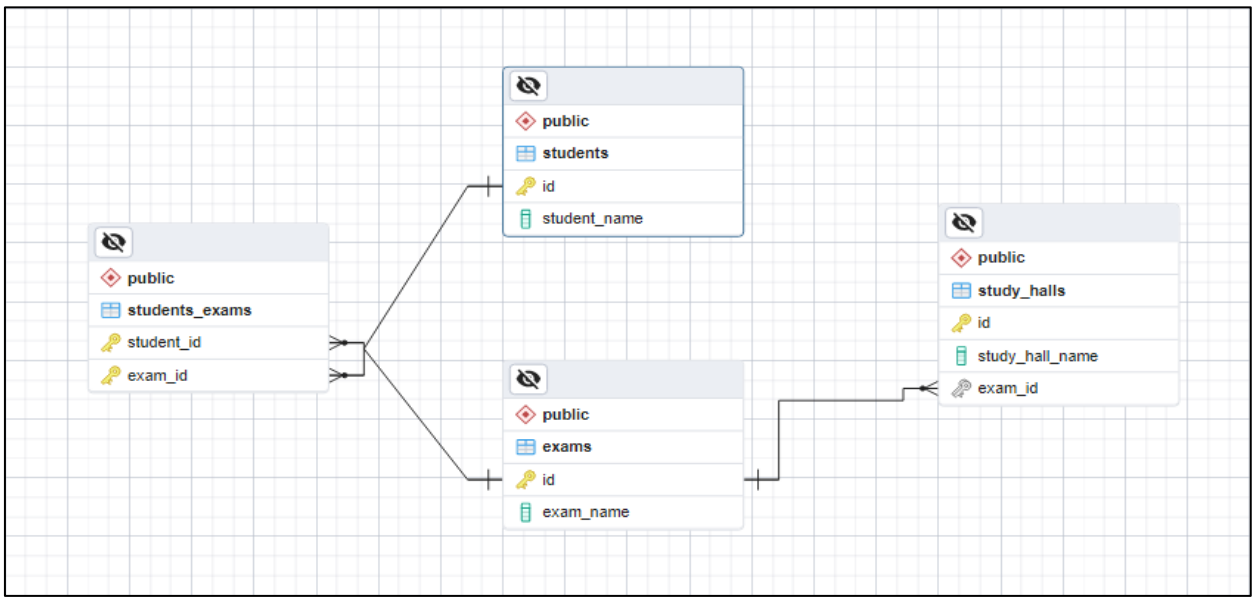
Creative Hall	103
Masterclass Hall	103
Information Security Hall	103

students_exams	
student_id	**exam_id**
1	101
1	102
2	101
3	103
2	102
2	103

Submit your query for this task in the Judge system.

9. Study Session E/R Diagram**

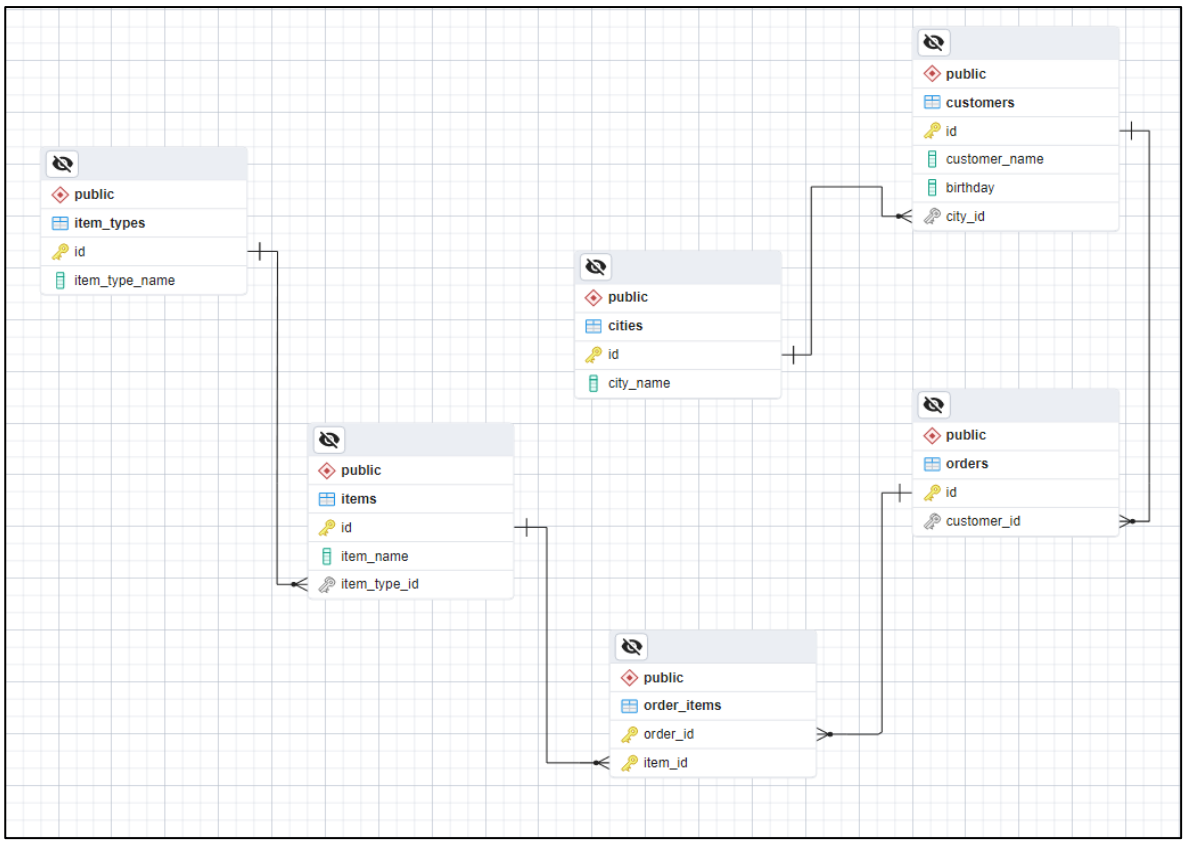
Create **Entity/Relationship** Diagram for the **"study_session_db"** database, which includes four tables: **"students"**, **"exams"**, **"study_halls"**, and **"students_exams"**.



10. Online Store

Create a database called **"online_store_db"** using the provided **E/R Diagram**. Set up the necessary tables and ensure that their **relationships are properly defined**.

! When creating your tables, arrange the columns and foreign key constraints as shown in the diagram below.



Submit your query for this task in the Judge system.

*For the upcoming tasks, let's pay more attention to **FOREIGN KEY** and its **Cascade Operations** and ensure that they are correctly implemented. We will be using a database that you are already familiar with, but for the purpose of these tasks, the data has been modified. Therefore, create a new database named **`table_relations_geography_db`**. Download the **`05-Exercises-Table-Relations-geography_db.sql`** file from the course instance, import it into your database's query tab, and execute the queries provided in the file.*

11. Delete Cascade

Let's proceed to establish the relationships and update the **`"countries"`** table. Locate the columns that correspond to the **`"continents"`** and **`"currencies"`** tables, and ensure that when a **record in the parent table is deleted**, corresponding **records in the child table are also deleted**.

Submit your query for this task in the Judge system.

12. Update Cascade

Revise the **`"countries_rivers"`** table by ensuring that its relationship with the **`"rivers"`** and **`"countries"`** tables is properly updated. When a row in the **parent tables is updated**, ensure that matching rows in the **child table are also updated**.

Submit your query for this task in the Judge system.

13. SET NULL

Write SQL queries to create two new tables: **`"customers"`** and **`"contacts"`**. The **`"customers"`** table should have a column for **`"customer_name"`**, and the **`"contacts"`** table should contain columns for **`"contact_name"`**, **`"phone"`**, **`"email"`**, and **`"customer_id"`**. You can choose the data type for each column based on your preference, but it's important to ensure that each table has a **unique identifier**.

To set up the **foreign keys** correctly, make sure that when a record in the **"customers"** table is **deleted**, the corresponding records in the **"contacts"** table have their **"customer_id"** value set to **NULL**. Additionally, when a row in the **"customers"** table is **updated**, ensure that matching rows in the **"contacts"** table are also **updated accordingly.**

Insert data into the tables in the format shown below:

customers
customer_name
BlueBird Inc
Dolphin LLC

contacts			
contact_name	phone	email	customer_id
John Doe	(408)-111-1234	john.doe@bluebird.dev	1
Jane Doe	(408)-111-1235	jane.doe@bluebird.dev	1
David Wright	(408)-222-1234	david.wright@dolphin.dev	2

Finally, remove the row from the **"customers"** table where the value of the **"id"** column matches **1**.

Submit your query for this task in the Judge system.

Example

customers	
id	customer_name
2	Dolphin LLC

contacts				
id	contact_name	phone	email	customer_id
1	John Doe	(408)-111-1234	john.doe@bluebird.dev	[null]
2	Jane Doe	(408)-111-1235	jane.doe@bluebird.dev	[null]
3	David Wright	(408)-222-1234	david.wright@dolphin.dev	2

14. * Peaks in Rila

Retrieve data from the **" table_relations_geography_db"** database by joining the **"mountains"** and **"peaks"** tables using their common data. Then, display all peaks for the **"Rila"** mountain, including the **"mountain_range"**, **"peak_name"**, and **"elevation"**. Finally, sort the results in descending order by the "elevation".

Submit your query for this task in the Judge system.

Example

mountain_range	peak_name	elevation
Rila	Musala	2925
Rila	Malka Musala	2902
Rila	Malyovitsa	2729
Rila	Orlovets	2685

15. * Countries Without Any Rivers

Create an SQL query that retrieves data from the **"table_relations_geography_db"** database by joining the **"countries"** and **"countries_rivers"** tables based on their common data. Then, calculate the total number of countries that do not have any rivers.

*** Note, using a **LEFT JOIN** will ensure that all records from the **"countries"** table are included in the result set, and a **WHERE** clause will filter out rows where the **"countries_rivers"** table has no corresponding records.

Submit your query for this task in the Judge system.

Example

countries_without_rivers
184

** This task is not required to be submitted to the Judge system and will not be considered in the final result.