Proyecto Final Databases

Alberto van Oldenbarneveld genaamd Tullingh

CUESTIONARIO

1.1. Explica la diferencia entre Base de Datos Relacional y SQL:

Una base de datos relacional es una estructura compuesta por tablas relacionadas, mientras que SQL es el lenguaje que se utiliza para interactuar con estas bases de datos.

1.2. ¿Por qué es necesario para las tablas definir una primary key?

Las Primary Keys aseguran que cada registro sea único en una tabla, lo que facilita la relación entre tablas y evita los duplicados.

1.3. ¿Cómo se denomina la relación que se hace entre una columna de una tabla y la primary key de otra tabla?

Se importa la primary key a la tabla de destino con la forma de un foreign key. Es decir, reconociendo que es una llave primaria de otra tabla.

1.4. ¿Qué es lo que necesitamos hacer para poder tener una relación N:M entre dos tablas?

Dos tablas N:M se relacionan a través de una tabla intermedia que contenga las primary keys de ambas tablas. Así se relacionan cada una de las entradas de las dos tablas.

CONSULTAS SQL SOBRE UNA BASE DE DATOS

2.1 Buscar todos los clientes (`customers`) con el código postal 1010.

Query:

```
SELECT *
FROM customers
WHERE postal_code = '1010';
```

Respuesta:

```
 customer_id |        company_name        |   contact_name   |    contact_title     |               add
-------------+----------------------------+------------------+----------------------+------------------
 CACTU       | Cactus Comidas para llevar | Patricio Simpson | Sales Agent          | Cerrito 333
 OCEAN       | Océano Atlántico Ltda.     | Yvonne Moncada   | Sales Agent          | Ing. Gustavo Monc
 RANCH       | Rancho grande              | Sergio Gutiérrez | Sales Representative | Av. del Libertado
```

2.2 Buscar el número de teléfono que tiene el proveedor (`supplier`) con id 11:

Query:

```
SELECT phone
FROM suppliers
WHERE supplier_id = 11;
```

Respuesta:

```
     phone
---------------
 (010) 9984510
(1 row)
```

2.3. Listar los primeros 10 pedidos (`orders)` ordenados de manera descendente por la fecha de pedido.

Query:

```
SELECT *
FROM orders
ORDER BY order_date DESC
LIMIT 10;
```

Respuesta:

```
    ship_name           |                 ship_address            |   ship_city    | ship_region | shi
----------+-------------+-------------+------------+---------------+--------------+----------+--------
    11074 | SIMOB       |           7 | 1998-05-06 | 1998-06-03    |              |        2 |   18.44
    11075 | RICSU       |           8 | 1998-05-06 | 1998-06-03    |              |        2 |    6.19
    11077 | RATTC       |           1 | 1998-05-06 | 1998-06-03    |              |        2 |    8.53
    11076 | BONAP       |           4 | 1998-05-06 | 1998-06-03    |              |        2 |   38.28
    11072 | ERNSH       |           4 | 1998-05-05 | 1998-06-02    |              |        2 |  258.64
    11070 | LEHMS       |           2 | 1998-05-05 | 1998-06-02    |              |        1 |     136
    11073 | PERIC       |           2 | 1998-05-05 | 1998-06-02    |              |        2 |   24.95
    11071 | LILAS       |           1 | 1998-05-05 | 1998-06-02    |              |        1 |    0.93
    11067 | DRACD       |           1 | 1998-05-04 | 1998-05-18    | 1998-05-06   |        2 |    7.98
    11068 | QUEEN       |           8 | 1998-05-04 | 1998-06-01    |              |        2 |   81.75
(10 rows)
```

2.4. Buscar todos los clientes (customers) que vivan en London, Madrid o Brazil

Query:

```
SELECT *
FROM customers
WHERE city IN ('London', 'Madrid')
   OR country = 'Brazil';
```

Respuesta:

```
 customer_id |             company_name             |   contact_name    |     contact_title     |
-------------+--------------------------------------+-------------------+-----------------------+------
 AROUT       | Around the Horn                      | Thomas Hardy      | Sales Representative  | 120 H
 BOLID       | Bólido Comidas preparadas            | Martín Sommer     | Owner                 | C/ Ar
 BSBEV       | B's Beverages                        | Victoria Ashworth | Sales Representative  | Faunt
 COMMI       | Comércio Mineiro                     | Pedro Afonso      | Sales Associate       | Av. d
 CONSH       | Consolidated Holdings                | Elizabeth Brown   | Sales Representative  | Berke
 EASTC       | Eastern Connection                   | Ann Devon         | Sales Agent           | 35 Ki
 FAMIA       | Familia Arquibaldo                   | Aria Cruz         | Marketing Assistant   | Rua O
 FISSA       | FISSA Fabrica Inter. Salchichas S.A. | Diego Roel        | Accounting Manager    | C/ Mo
 GOURL       | Gourmet Lanchonetes                  | André Fonseca     | Sales Associate       | Av. B
 HANAR       | Hanari Carnes                        | Mario Pontes      | Accounting Manager    | Rua d
 NORTS       | North/South                          | Simon Crowther    | Sales Associate       | South
 QUEDE       | Que Delícia                          | Bernardo Batista  | Accounting Manager    | Rua d
 QUEEN       | Queen Cozinha                        | Lúcia Carvalho    | Marketing Assistant   | Alame
 RICAR       | Ricardo Adocicados                   | Janete Limeira    | Assistant Sales Agent | Av. C
 ROMEY       | Romero y tomillo                     | Alejandra Camino  | Accounting Manager    | Gran
 SEVES       | Seven Seas Imports                   | Hari Kumar        | Sales Manager         | 90 Wa
 TRADH       | Tradição Hipermercados               | Anabela Domingues | Sales Representative  | Av. I
 WELLI       | Wellington Importadora               | Paula Parente     | Sales Manager         | Rua d
(18 rows)
```

2.5 Añadir un nuevo registro en la tabla clientes (customers) con la siguiente información (indicada en el mismo orden de las columnas): "`XYZ`", "`The Shire`", "`Bilbo Baggins`", "`1 Hobbit-Hole`", "`Bag End`", "`111`" y "`Middle Earth`"

Query:

```
INSERT INTO customers (customer_id, company_name, contact_name, address, city, postal_code, country)
VALUES ('XYZ', 'The Shire', 'Bilbo Baggins', '1 Hobbit-Hole', 'Bag End', '111', 'Middle Earth');
```

Comprobación:

```
 customer_id | company_name | contact_name  | contact_title |    address    |  city   | region | postal
-------------+--------------+---------------+---------------+---------------+---------+--------+-------
 XYZ         | The Shire    | Bilbo Baggins |               | 1 Hobbit-Hole | Bag End |        | 111
(1 row)
```

2.6: Actualizar el código postal a "`11122`" del cliente "`Bilbo Baggins`".

Query:

```
UPDATE customers
SET postal_code = '11122'
WHERE customer_id = 'XYZ';
```

Comprobación:

```
 customer_id | company_name | contact_name  | contact_title |    address    |  city   | region | postal
-------------+--------------+---------------+---------------+---------------+---------+--------+-------
 XYZ         | The Shire    | Bilbo Baggins |               | 1 Hobbit-Hole | Bag End |        | 11122
(1 row)
```

2.7: Mostrar "`ProductName`" y "`CategoryName`" de todos los productos (`products`):

Query:

```
SELECT p.product_name, c.category_name
FROM products p
JOIN categories c ON p.category_id = c.category_id;
```

Explicación: Primero seleccionamos los campos `product_name` y `category_name`, y especificamos la tabla base, `products`. Luego la combinamos con `categories`, usando `JOIN` usando la key `category_id` como vínculo entre las dos tablas.

Resultado:

```
       product_name               | category_name
----------------------------------+----------------
 Chai                             | Beverages
 Chang                            | Beverages
 Aniseed Syrup                    | Condiments
 Chef Anton's Cajun Seasoning     | Condiments
 Chef Anton's Gumbo Mix           | Condiments
 [...]
 (77 rows)
```

2.8 Mostrar "`OrderID`" y "`CompanyName`" del expedidor (`shippers`) de todos los pedidos (`orders`) realizados antes del 9 de agosto de 2012:

query:

```
SELECT o.order_id, s.company_name
FROM orders o
JOIN shippers s ON o.ship_via = s.shipper_id
WHERE o.order_date < '2012-08-09';
```

Explicación: Seleccionamos las columnas `order_id` de la tabla `orders` y el nombre de la empresa de la tabla de expedidores, `shippers`. Después indicamos que la tabla base es `orders`, y la unimos con la tabla `shippers` con la key `ship_via` , que es la columna `shipper_id` en `shippers` . Finalmente filtramos solo para los pedidos realizados antes del 9 de agosto de 2012 con `WHERE o.order_date < '2012-08-09'`

Resultado:

```
order_id  |   company_name
----------+------------------
    10248 | Federal Shipping
    10249 | Speedy Express
    10250 | United Package
    10251 | Speedy Express
    10252 | United Package
    [...]
    (830 rows)
```

2.9: Mostar el número de pedidos (`orders`) realizados por cada expedidor (`shipper`):

```
SELECT s.company_name, COUNT(o.order_id) AS total_orders
FROM shippers s
JOIN orders o ON s.shipper_id = o.ship_via
GROUP BY s.company_name;
```

Explicación: seleccionamos el nombre del expedidor, `company_name` y contamos el número de pedidos `order_id`, y lo nombramos `total_orders`. Indicamos que `shippers` como la tabla base, y la combinamos con `orders` usando las keys `shipper_id` de `shippers` y `ship_via` de `orders`. Finalmente, agrupamos los resultados por `company_name` para obtener el número total de pedidos de cada expedidor.

Resultado:

```
company_name      | total_orders
------------------+--------------
Federal Shipping  |          255
Speedy Express    |          249
United Package    |          326
(3 rows)
```

3. MODELADO BASE DE DATOS

Caso de Uso: Aplicación de Todo List

Descripción del Caso de Uso

El objetivo es diseñar una base de datos para una aplicación de lista de tareas (todo list) que permita a los usuarios gestionar sus tareas diarias de manera eficiente. La base de datos debe almacenar información sobre los usuarios, las tareas que crean y las categorías utilizadas para organizar dichas tareas. Además, debe ser capaz de registrar el estado de cada tarea (pendiente, en progreso, completada) y permitir clasificar las tareas bajo varias categorías.

Requisitos

1. **Usuarios**:
   - Cada usuario puede tener varias tareas asociadas.
   - Los usuarios deben estar registrados con un nombre y un correo electrónico único.
2. **Tareas**:
   - Cada tarea pertenece a un usuario.
   - Las tareas tienen un título, una descripción, una fecha de creación y un estado.
   - Una tarea puede clasificarse bajo múltiples categorías.
3. **Categorías**:
   - Una categoría puede asociarse a varias tareas.
   - Las categorías tienen un nombre único.
4. **Relaciones**:
   - Relación 1:N: Un usuario puede tener varias tareas.
   - Relación N:M: Una tarea puede pertenecer a varias categorías, y una categoría puede incluir varias tareas.

Entidades y Atributos

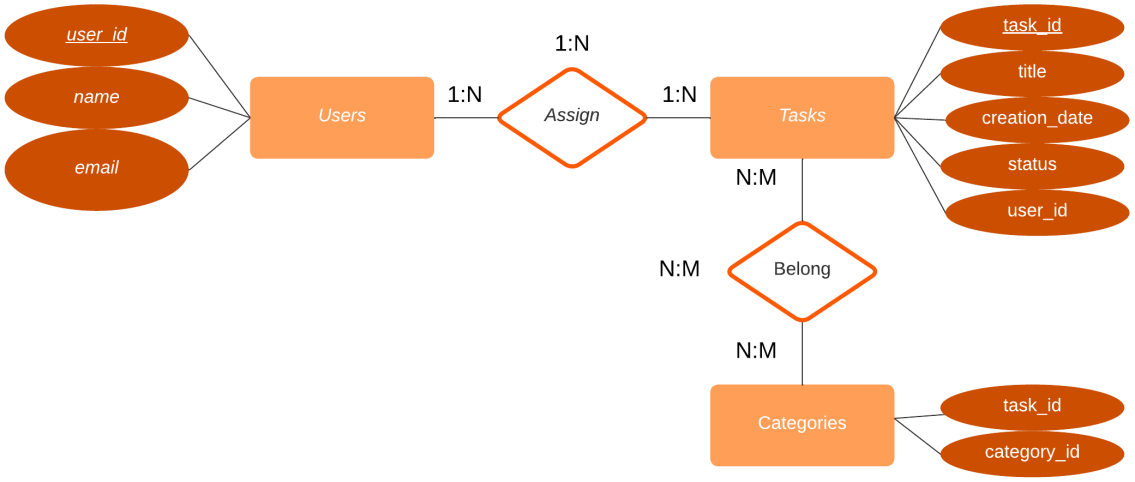
1. **Usuarios**:
   - `user_id` (PK): Identificador único del usuario.
   - `name`: Nombre del usuario.
   - `email`: Correo electrónico del usuario (debe ser único).
2. **Tareas**:
   - `task_id` (PK): Identificador único de la tarea.
   - `title`: Título de la tarea.
   - `description`: Descripción de la tarea.
   - `creation_date`: Fecha de creación de la tarea.
   - `status`: Estado de la tarea (pendiente, en progreso, completada).
   - `user_id` (FK): Identificador del usuario que creó la tarea.
3. **Categorías**:

- `category_id` (PK): Identificador único de la categoría.
- `name`: Nombre de la categoría.

4. **Relación Tareas-Categoría** (Tabla Intermedia):
   - `task_id` (FK): Identificador de la tarea.
   - `category_id` (FK): Identificador de la categoría.

Relaciones

- **Usuarios - Tareas (1:N)**:
  - Un usuario puede tener varias tareas.
  - Cada tarea pertenece a un único usuario.
- **Tareas - Categorías (N:M)**:
  - Una tarea puede pertenecer a varias categorías.
  - Una categoría puede contener varias tareas.



Creamos la base de datos con sus tablas y relaciones entre ellas:

1. Creamos la base de datos:

```
CREATE DATABASE todo_list_app;
```

Nos conectamos a ella:

```
\c todo_list_app
```

Creamos la tabla Users, con la restricciones de que ni el nombre ni el email pueden estar vacíos, y el email debe ser único, y el user_id como primary key:

```
CREATE TABLE Users (
    user_id SERIAL PRIMARY KEY,
    name TEXT NOT NULL,
    email TEXT UNIQUE NOT NULL
);
```

Después creamos la tabla Tasks, con task_id como primary key. El título no podrá estar vacío, y si la fecha no se indica, se toma la fecha de la creación de la entrada por defecto. Por otro lado, establecemos que el estado de la tarea únicamente podrá ser pendiente, en proceso o finalizado. el user_id se establece como foreign key de la tabla users:

```
CREATE TABLE Tasks (
task_id SERIAL PRIMARY KEY,
title TEXT NOT NULL,
```

```
description TEXT,
creation_date DATE DEFAULT CURRENT_DATE,
status TEXT CHECK (status IN ('pending', 'in progress', 'completed')),
user_id INT NOT NULL,
FOREIGN KEY (user_id) REFERENCES Users (user_id)
);
```

Seguidamente, creamos la tabla categories, con category_id como primary key, y con la restricción de que el número de la categoría debe ser único:

```
CREATE TABLE Categories (
    category_id SERIAL PRIMARY KEY,
    name TEXT UNIQUE NOT NULL,
    description TEXT,
    created_at DATE DEFAULT CURRENT_DATE
);
```

Creamos la tabla intermedia para establecer las relaciones entre Tasks y Categories, indicando cuáles son las foreign keys, y estableciendo su propia primary key:

```
CREATE TABLE Task_Category (
    task_id INT NOT NULL,
    category_id INT NOT NULL,
    PRIMARY KEY (task_id, category_id),
    FOREIGN KEY (task_id) REFERENCES Tasks (task_id),
    FOREIGN KEY (category_id) REFERENCES Categories (category_id)
);
```

Comprobamos que hemos creado las tablas correctamente:

si hacemos `\d Users` :

```
                            Table "public.users"
 Column  |  Type   | Collation | Nullable |               Default
---------+---------+-----------+----------+--------------------------------------
 user_id | integer |           | not null | nextval('users_user_id_seq'::regclass)
 name    | text    |           | not null |
 email   | text    |           | not null |
Indexes:
    "users_pkey" PRIMARY KEY, btree (user_id)
    "users_email_key" UNIQUE CONSTRAINT, btree (email)
Referenced by:
    TABLE "tasks" CONSTRAINT "tasks_user_id_fkey" FOREIGN KEY (user_id) REFERENCES users(user_id)
```

`\d Tasks`

```
                                Table "public.tasks"
    Column     |  Type   | Collation | Nullable |                Default
---------------+---------+-----------+----------+---------------------------------------
 task_id       | integer |           | not null | nextval('tasks_task_id_seq'::regclass)
 title         | text    |           | not null |
 description   | text    |           |          |
 creation_date | date    |           |          | CURRENT_DATE
 status        | text    |           |          |
 user_id       | integer |           | not null |
Indexes:
    "tasks_pkey" PRIMARY KEY, btree (task_id)
Check constraints:
    "tasks_status_check" CHECK (status = ANY (ARRAY['pending'::text, 'in progress'::text, 'completed'::
Foreign-key constraints:
    "tasks_user_id_fkey" FOREIGN KEY (user_id) REFERENCES users(user_id)
Referenced by:
    TABLE "task_category" CONSTRAINT "task_category_task_id_fkey" FOREIGN KEY (task_id) REFERENCES task
```

`\d Categories`

```
                                  Table "public.categories"
   Column    |  Type   | Collation | Nullable |                     Default
-------------+---------+-----------+----------+-------------------------------------------------
 category_id | integer |           | not null | nextval('categories_category_id_seq'::regclass)
 name        | text    |           | not null |
 description | text    |           |          |
 created_at  | date    |           |          | CURRENT_DATE
Indexes:
    "categories_pkey" PRIMARY KEY, btree (category_id)
    "categories_name_key" UNIQUE CONSTRAINT, btree (name)
Referenced by:
    TABLE "task_category" CONSTRAINT "task_category_category_id_fkey" FOREIGN KEY (category_id) REFEREN
```

`\d Task_Category`

```
              Table "public.task_category"
   Column    |  Type   | Collation | Nullable | Default
-------------+---------+-----------+----------+---------
 task_id     | integer |           | not null |
 category_id | integer |           | not null |
Indexes:
    "task_category_pkey" PRIMARY KEY, btree (task_id, category_id)
Foreign-key constraints:
    "task_category_category_id_fkey" FOREIGN KEY (category_id) REFERENCES categories(category_id)
    "task_category_task_id_fkey" FOREIGN KEY (task_id) REFERENCES tasks(task_id)
```

Poblamos la base de datos con una serie de registros:

En la tabla `Users` :

```
INSERT INTO Users (name, email) VALUES
('Alberto van Oldenbarneveld', 'alberto@gmail.com'),
('Lorena García', 'lorena@outlook.com'),
('Maria Rodríguez', 'maria@yahoo.com'),
('Juan Pérez', 'juan@gmail.com'),
('Ana Fernández', 'ana@outlook.com');
```

En la tabla `Categories`

```
INSERT INTO Categories (name, description, created_at) VALUES
('Trabajo', 'Categoría relacionada con el trabajo', CURRENT_DATE),
('Personal', 'Categoría para asuntos personales', CURRENT_DATE),
('Salud', 'Categoría sobre salud y bienestar', CURRENT_DATE),
('Educación', 'Categoría relacionada con la educación', CURRENT_DATE),
('Hogar', 'Categoría para tareas del hogar', CURRENT_DATE);
```

En la tabla `Tasks` :

```
INSERT INTO Tasks (title, description, status, user_id) VALUES
('Terminar el proyecto de databases', 'Completar la última parte del proyecto final.', 'in progress', 1
('Enviar correo al cliente', 'Notificar sobre los avances en el proyecto.', 'pending', 2),
('Comprar materiales', 'Ir a la tienda y comprar herramientas para el hogar.', 'completed', 3),
('Leer un libro', 'Avanzar con el libro de educación financiera.', 'pending', 4),
('Organizar escritorio', 'Limpiar y organizar documentos del trabajo.', 'in progress', 5);
```

Asociamos tareas con categorías en la tabla `Task_Category`:

```
INSERT INTO Task_Category (task_id, category_id) VALUES
(1, 1), -- Tarea 1 ("Terminar el proyecto de databases") está en la categoría "Trabajo"
(1, 4), -- Tarea 1 también está en la categoría "Educación"
(2, 1), -- Tarea 2 ("Enviar correo al cliente") está en la categoría "Trabajo"
(3, 5), -- Tarea 3 ("Comprar materiales") está en la categoría "Hogar"
(4, 4), -- Tarea 4 ("Leer un libro") está en la categoría "Educación"
```

```
(5, 1), -- Tarea 5 ("Organizar escritorio") está en la categoría "Trabajo"
(5, 5); -- Tarea 5 también está en la categoría "Hogar"
```

Realizamos las consultas para demostrar que el modelo cumple con los requisitos:

1. Listar las tareas de un usuario específico.
2. Mostrar las categorías asociadas a una tarea.
3. Contar tareas pendientes por usuario.
4. Mostrar tareas asociadas a una categoría.
5. Contar el número de tareas por categoría.

Consulta 1: Listar las tareas de un usuario específico:

```
SELECT t.title, t.description, t.status, t.creation_date
FROM Tasks t
JOIN Users u ON t.user_id = u.user_id
WHERE u.name = 'Alberto van Oldenbarneveld';
```

Seleccionamos los campos `title`, `description`, `status` y `creation_date` de la tabla `Tasks`, y juntamos `Users` con `Tasks` con la foreign key `user_id`, y filtramos por `name`, `'Alberto van Oldenbarneveld'`

Respuesta:

```
           title                  |                  description                  |   status    | creat
----------------------------------+-----------------------------------------------+-------------+------
Terminar el proyecto de databases | Completar la última parte del proyecto final. | in progress | 2024-
(1 row)
```

Consulta 2: Mostrar todas las categorías asociadas a una tarea específica para validar la relación **N:M** entre tareas y categorías.

```
SELECT t.title AS task_title, c.name AS category_name
FROM Tasks t
JOIN Task_Category tc ON t.task_id = tc.task_id
JOIN Categories c ON tc.category_id = c.category_id
WHERE t.title = 'Terminar el proyecto de databases';
```

Seleccionamos el título de la tarea `task_title` y los nombres de la categoría `category_name` y consultamos en la tabla `tasks` . Unimos `Tasks` con la tabla intermedia `Task_Category` a través de `task_id` y unimos `Task_Category` con `Categories` mediante `category_id`. Después, filtramos para mostrar solo las categorías asociadas.

Resultado:

```
            task_title             | category_name
-----------------------------------+---------------
 Terminar el proyecto de databases | Trabajo
 Terminar el proyecto de databases | Educación
(2 rows)
```

Consulta 3: Contar el número de tareas pendientes por usuario para validar la relación **1:N** entre `Users` y `Tasks`.

```
SELECT u.name AS user_name, COUNT(t.task_id) AS pending_tasks
FROM Users u
JOIN Tasks t ON u.user_id = t.user_id
WHERE t.status = 'pending'
GROUP BY u.name;
```

Explicación: Seleccionamos el nombre de usuario `user_name` y contamos con `COUNT` el número de tareas pendientes `pending_tasks` . Utilizamos como bse la tabla `Users` . Unimos las tablas de `Users` con `Tasks` mediante `user_id` ,filtramos por `status` = `pending` y agrupamos con `GROUP` por nombre.

Resultado:

```
   user_name   | pending_tasks
---------------+---------------
 Juan Pérez    |             1
 Lorena García |             1
(2 rows)
```

Consulta 4: Mostrar todas las tareas asociadas a una categoría específica para validar la relación **N:M** entre `Tasks` y `Categories` :

```
SELECT t.title AS task_title, t.status, t.creation_date
FROM Tasks t
JOIN Task_Category tc ON t.task_id = tc.task_id
JOIN Categories c ON tc.category_id = c.category_id
WHERE c.name = 'Trabajo';
```

Seleccionamos el título de la tarea `task_title`, el estado `status` y la fecha de creación `creation_date` desde la tabla `Tasks`. Unimos `Tasks` con la tabla intermedia `Task_Category` mediante `task_id`, y luego unimos `Task_Category` con la tabla `Categories` a través de `category_id`. Finalmente, filtramos las tareas que pertenecen a la categoría cuyo nombre es `Trabajo`.

Resultado:

```
            task_title             |   status    | creation_date
-----------------------------------+-------------+---------------
 Terminar el proyecto de databases | in progress | 2024-12-28
 Enviar correo al cliente          | pending     | 2024-12-28
 Organizar escritorio              | in progress | 2024-12-28
(3 rows)
```

Consulta 5: Contar el número de tareas por categoría.

```
SELECT c.name AS category_name, COUNT(tc.task_id) AS total_tasks
FROM Categories c
JOIN Task_Category tc ON c.category_id = tc.category_id
GROUP BY c.name;
```

Seleccionamos el nombre de la categoría `category_name` y contamos con `COUNT` el número total de tareas `total_tasks`. Utilizamos la tabla `Categories` como base. Unimos `Categories` con la tabla intermedia `Task_Category` mediante `category_id`, y agrupamos con `GROUP BY` por el nombre de la categoría.

Resultado:

```
 category_name | total_tasks
---------------+-------------
 Trabajo       |           3
 Educación     |           2
 Hogar         |           2
 (3 rows)
```