



REPORTE DE PRÁCTICA NO. 2

NOMBRE DE LA PRÁCTICA

ALUMNO: Kevin Badillo Olmos

Dr. Eduardo Cornejo-Velázquez

Introducción

El álgebra relacional es un lenguaje formal y matemático utilizado para manipular y consultar datos en bases de datos relacionales. Fue propuesta por Edgar F. Codd en la década de 1970 como parte de la teoría que sustenta los sistemas de gestión de bases de datos relacionales (RDBMS).

A diferencia de SQL, que es un lenguaje de implementación, el álgebra relacional se enfoca en operaciones conceptuales y precisas sobre las relaciones (tablas) que componen la base de datos. Estas operaciones permiten seleccionar, proyectar, unir, combinar o eliminar información de manera rigurosa y predecible.

El álgebra relacional es fundamental por varias razones:

Base teórica de SQL: todas las consultas SQL se pueden expresar mediante operaciones de álgebra relacional.

Optimización de consultas: los motores de bases de datos utilizan principios del álgebra relacional para mejorar el rendimiento de las consultas.

Rigor matemático: asegura que los resultados sean consistentes y que las operaciones sobre las tablas respeten las reglas de integridad.

Educación y análisis: facilita la comprensión de cómo se combinan y transforman los datos dentro de un RDBMS.

Entre los operadores principales del álgebra relacional se encuentran la selección, proyección, unión, diferencia, producto cartesiano, join y renombrado, cada uno de ellos con un papel específico en la manipulación de los datos.

En síntesis, el álgebra relacional constituye el fundamento lógico y conceptual de las bases de datos relacionales, proporcionando un marco para expresar consultas de manera precisa y estructurada antes de traducirlas a SQL u otro lenguaje de implementación.

Marco Teórico

Fundamentos

Las bases de datos relacionales organizan los datos en tablas (relaciones) compuestas por filas (tuplas) y columnas (atributos). El álgebra relacional define un conjunto de operadores que permiten generar nuevas relaciones a

partir de las existentes, garantizando consistencia y precisión en los resultados.

3. Operadores Básicos del Álgebra Relacional

3.1 Selección (σ)

Extrae tuplas que cumplen una condición específica dentro de una relación. Permite filtrar los datos según criterios determinados, sin modificar la estructura de la tabla.

3.2 Proyección (π)

Extrae atributos específicos de una relación, eliminando duplicados por defecto.

Permite reducir la cantidad de columnas en el resultado, enfocándose únicamente en la información necesaria.

3.3 Unión (\cup)

Combina tuplas de dos relaciones que tienen la misma estructura, eliminando duplicados.

Solo puede aplicarse entre relaciones con el mismo número de atributos y tipos de datos compatibles.

3.4 Diferencia ($-$)

Devuelve las tuplas que están en la primera relación pero no en la segunda. Es útil para identificar elementos que no cumplen ciertas condiciones o no están presentes en otra relación.

3.5 Producto cartesiano (\times)

Combina todas las tuplas de dos relaciones, generando todas las combinaciones posibles.

Generalmente se usa como base para realizar joins, donde se filtran las combinaciones según condiciones.

3.6 Join (\bowtie)

Permite unir dos relaciones mediante una condición de igualdad entre atributos relacionados.

Los tipos de join incluyen: natural, equi-join, theta-join, y outer join.

Es fundamental para integrar información de diferentes relaciones de manera coherente.

3.7 Renombrar (ρ)

Cambia el nombre de una relación o atributo para mejorar la claridad en consultas complejas o para evitar conflictos de nombres.

4. Propiedades del Álgebra Relacional

Cerradura: Todas las operaciones de álgebra relacional producen como resultado otra relación, permitiendo encadenar operaciones.

Composición: Las operaciones pueden combinarse para formar consultas complejas y precisas.

Independencia de datos: El álgebra relacional se centra en la manipulación lógica de datos, sin depender de la forma física de almacenamiento.

5. Importancia

Fundamento teórico de bases de datos: Permite entender cómo se relacionan y manipulan los datos de manera formal.

Optimización de consultas: La teoría del álgebra relacional se usa para analizar y mejorar la eficiencia de las consultas.

Consistencia y precisión: Garantiza que los resultados sean correctos y que las operaciones respeten las reglas de integridad de la base de datos.

Diseño y análisis: Facilita la planificación de estructuras de datos y relaciones entre tablas antes de su implementación física.

Análisis de requerimientos

1. Gestión de empleados

Registrar nuevos empleados con información personal y laboral.

Modificar o actualizar los datos de los empleados existentes.

Consultar la información de todos los empleados.

Eliminar registros de empleados si es necesario.

2. Gestión de recompensas

Registrar recompensas otorgadas a los empleados.

Asociar cada recompensa a un empleado específico.

Consultar todas las recompensas otorgadas y a quiénes se asignaron.

Consultas y reportes

Listar empleados por departamento, salario o criterios específicos.

Mostrar recompensas otorgadas a un empleado determinado.

Generar reportes de empleados sin recompensas.

3. Requerimientos no funcionales

Los requerimientos no funcionales definen cómo debe comportarse el sistema:

Seguridad

Control de acceso mediante usuarios y permisos.

Protección de datos sensibles como salarios o información personal.

Rendimiento

Consultas eficientes, incluso con grandes volúmenes de empleados y recompensas.

Escalabilidad

Posibilidad de ampliar el sistema para soportar más empleados, departamentos y tipos de recompensas.

Integridad de datos

Asegurar que cada recompensa esté asociada a un empleado existente.

Mantener consistencia en los datos de empleados y departamentos.

4. Requerimientos de datos

Se identifican las entidades y atributos principales:

Empleado (Employee)

ID de empleado

Nombre

Apellido

Departamento

Salario

Recompensa (Reward)

ID de recompensa

ID de empleado asociado

Nombre de la recompensa

Monto

5. Requerimientos de relaciones

Cada empleado puede tener cero o más recompensas.

Cada recompensa está asociada exactamente a un empleado.

Se deben respetar las relaciones de integridad referencial entre tablas

Herramientas empleadas

One Compiler: Es un compilador de programas

Algebra relacional

El álgebra relacional es un lenguaje de consulta formal y procedimental que se utiliza para trabajar con datos en bases de datos relacionales. A diferencia de SQL, que es declarativo (le dices al sistema lo que quieres), el álgebra relacional es

procedural (le dices al sistema cómo obtenerlo). Es fundamental porque proporciona la base teórica para el diseño y las operaciones de los sistemas de gestión de bases de datos (DBMS).

Operadores Clave del Álgebra Relacional

El álgebra relacional se basa en un conjunto de operadores que toman una o más relaciones (tablas) como entrada y producen una nueva relación como resultado.

Operadores de Conjunto Tradicionales

Estos operadores son los mismos que se usan en la teoría de conjuntos, pero aplicados a tablas:

Unión (\cup): Combina todas las tuplas (filas) de dos tablas. Las tablas deben tener el mismo número de columnas y tipos de datos compatibles.

Intersección (\cap): Devuelve las tuplas que existen en ambas tablas.

Diferencia ($-$): Devuelve las tuplas que están en la primera tabla, pero no en la segunda.

Producto Cartesiano (\times): Combina cada tupla de la primera tabla con cada tupla de la segunda tabla. El resultado es una tabla con un número de columnas igual a la suma de las columnas de ambas tablas.

Operadores Relacionales Específicos

Estos operadores fueron creados específicamente para bases de datos:

Selección (σ): Filtra las tuplas de una tabla basándose en una condición específica (por ejemplo, Salario > 50000).

Proyección (π): Selecciona solo ciertas columnas de una tabla para crear una nueva tabla más pequeña. Se usa para eliminar columnas duplicadas.

Renombrar (ρ): Cambia el nombre de una tabla o columna.

Join Natural (\bowtie): Combina dos tablas basándose en las columnas que tienen nombres y tipos de datos idénticos. Es una operación muy común para unir datos relacionados.

División (\div): Una operación más compleja que se utiliza para encontrar tuplas en una tabla que se relacionan con todas las tuplas de otra tabla. Por ejemplo, "encontrar los estudiantes que han tomado todos los cursos ofrecidos".

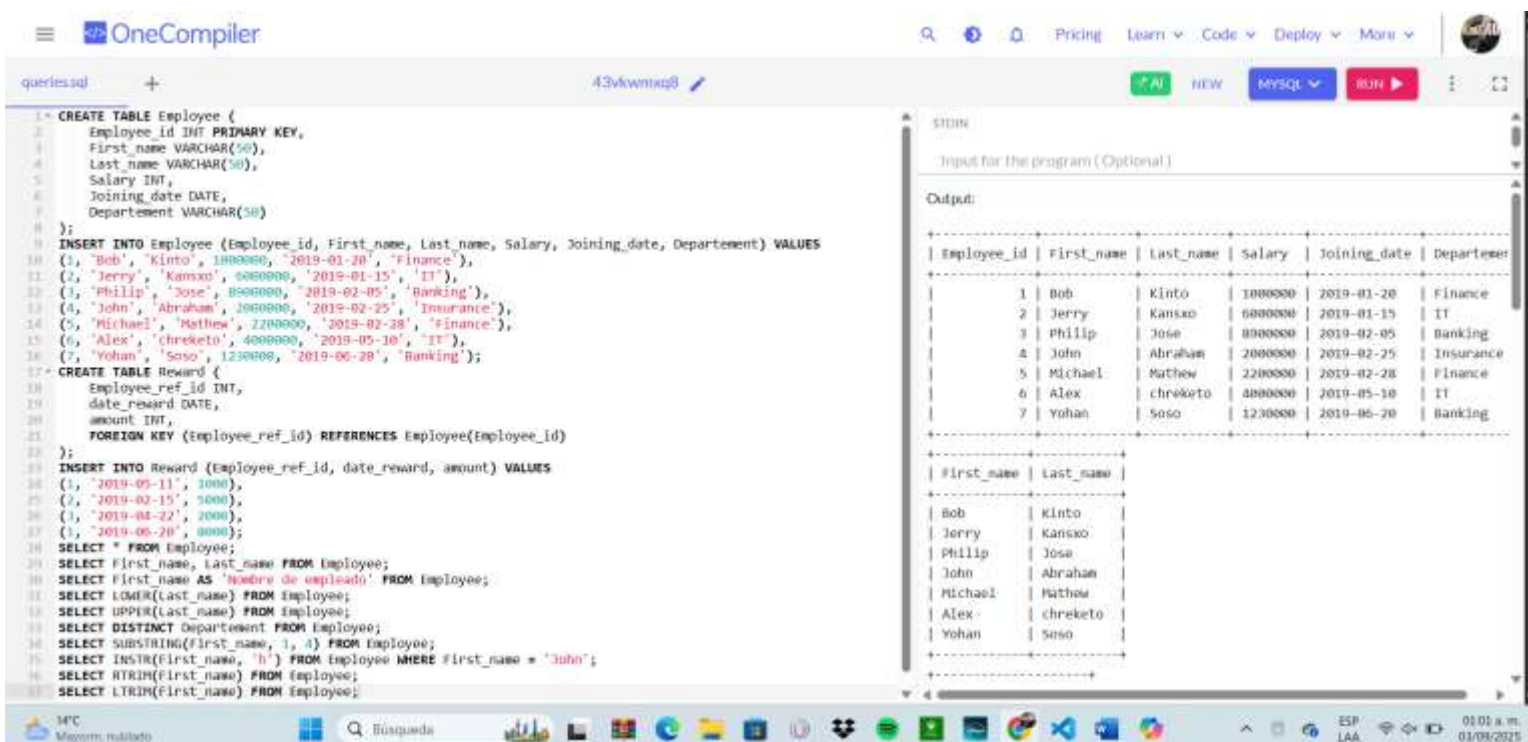
MySQL

MySQL, en resumen, es un sistema de gestión de bases de datos relacionales (RDBMS) de código abierto que utiliza el lenguaje SQL (Structured Query Language) para gestionar, manipular y recuperar datos. Es una de las

soluciones de bases de datos más populares del mundo, conocida por su velocidad, fiabilidad y facilidad de uso, especialmente en el desarrollo web.

Desarrollo

My SQL en One Compiler



The screenshot shows the OneCompiler web interface. The left pane contains a MySQL script with the following code:

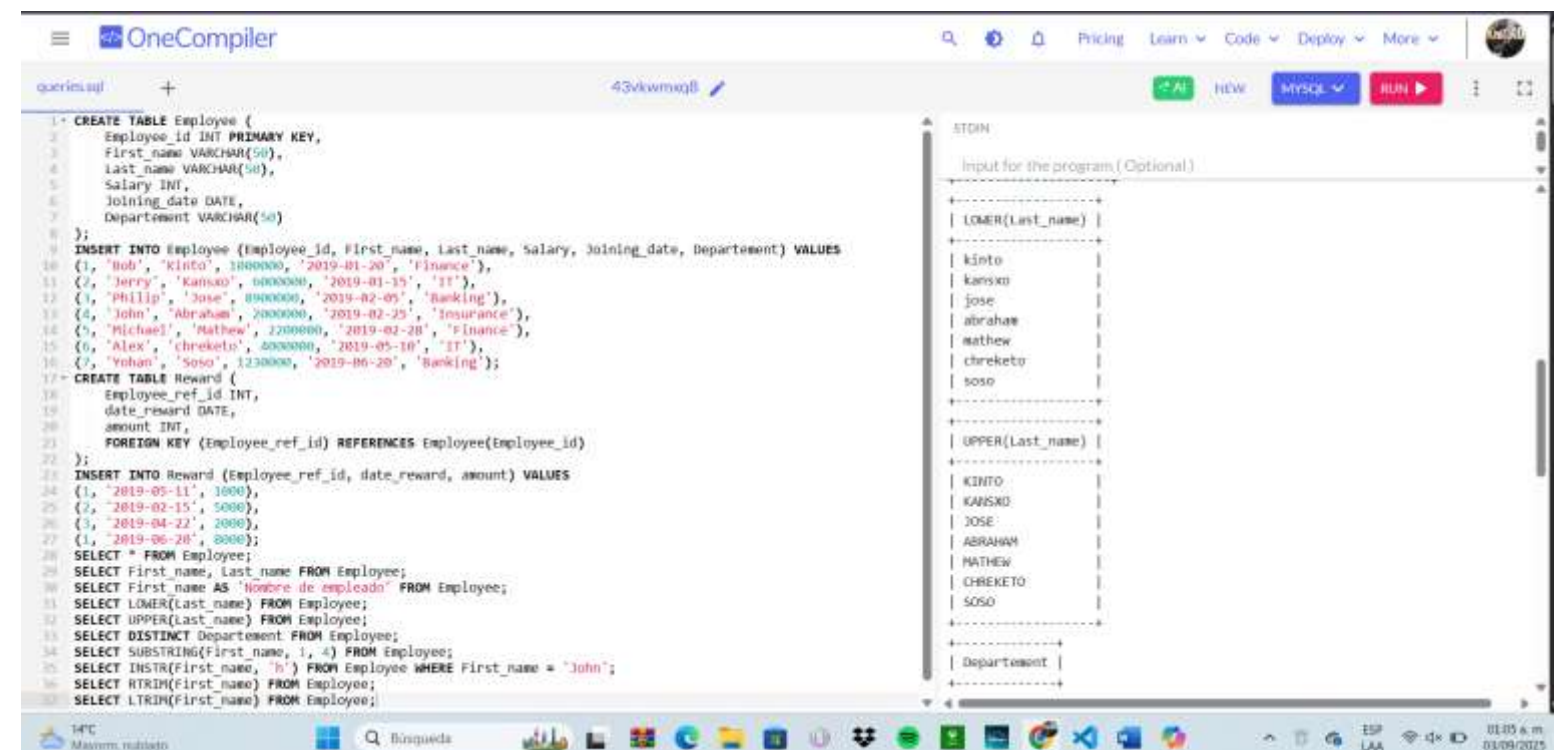
```
1 CREATE TABLE Employee {
2   Employee_id INT PRIMARY KEY,
3   First_name VARCHAR(50),
4   Last_name VARCHAR(50),
5   Salary INT,
6   Joining_date DATE,
7   Departement VARCHAR(50)
8 };
9
10 INSERT INTO Employee (Employee_id, First_name, Last_name, Salary, Joining_date, Departement) VALUES
11 (1, 'Bob', 'Kinto', 1000000, '2019-01-20', 'Finance'),
12 (2, 'Jerry', 'Kansao', 6000000, '2019-01-15', 'IT'),
13 (3, 'Phillip', 'Jose', 8000000, '2019-02-05', 'Banking'),
14 (4, 'John', 'Abraham', 2000000, '2019-02-25', 'Insurance'),
15 (5, 'Michael', 'Mathew', 2200000, '2019-02-28', 'Finance'),
16 (6, 'Alex', 'chreketo', 4000000, '2019-05-10', 'IT'),
17 (7, 'Yohan', 'Soso', 1230000, '2019-06-20', 'Banking');
18
19 CREATE TABLE Reward {
20   Employee_ref_id INT,
21   date_reward DATE,
22   amount INT,
23   FOREIGN KEY (Employee_ref_id) REFERENCES Employee(Employee_id)
24 };
25
26 INSERT INTO Reward (Employee_ref_id, date_reward, amount) VALUES
27 (1, '2019-05-11', 1000),
28 (2, '2019-02-15', 5000),
29 (3, '2019-04-22', 2000),
30 (4, '2019-06-20', 8000);
31
32 SELECT * FROM Employee;
33 SELECT First_name, Last_name FROM Employee;
34 SELECT First_name AS 'Nombre de empleado' FROM Employee;
35 SELECT LOWER(Last_name) FROM Employee;
36 SELECT UPPER(Last_name) FROM Employee;
37 SELECT DISTINCT Departement FROM Employee;
38 SELECT SUBSTRING(First_name, 1, 4) FROM Employee;
39 SELECT INSTR(First_name, 'h') FROM Employee WHERE First_name = 'John';
40 SELECT RTRIM(First_name) FROM Employee;
41 SELECT LTRIM(First_name) FROM Employee;
```

The right pane shows the output of the script, which includes a table of employee data and a list of last names in various cases.

Employee_id	First_name	Last_name	Salary	Joining_date	Departement
1	Bob	Kinto	1000000	2019-01-20	Finance
2	Jerry	Kansao	6000000	2019-01-15	IT
3	Phillip	Jose	8000000	2019-02-05	Banking
4	John	Abraham	2000000	2019-02-25	Insurance
5	Michael	Mathew	2200000	2019-02-28	Finance
6	Alex	chreketo	4000000	2019-05-10	IT
7	Yohan	Soso	1230000	2019-06-20	Banking

Below the table, the last names are listed in various cases:

First_name	Last_name
Bob	Kinto
Jerry	Kansao
Phillip	Jose
John	Abraham
Michael	Mathew
Alex	chreketo
Yohan	Soso



The screenshot shows the OneCompiler web interface. The left pane contains a MySQL script with the following code:

```
1 CREATE TABLE Employee {
2   Employee_id INT PRIMARY KEY,
3   First_name VARCHAR(50),
4   Last_name VARCHAR(50),
5   Salary INT,
6   Joining_date DATE,
7   Departement VARCHAR(50)
8 };
9
10 INSERT INTO Employee (Employee_id, First_name, Last_name, Salary, Joining_date, Departement) VALUES
11 (1, 'Bob', 'Kinto', 1000000, '2019-01-20', 'Finance'),
12 (2, 'Jerry', 'Kansao', 6000000, '2019-01-15', 'IT'),
13 (3, 'Phillip', 'Jose', 8000000, '2019-02-05', 'Banking'),
14 (4, 'John', 'Abraham', 2000000, '2019-02-25', 'Insurance'),
15 (5, 'Michael', 'Mathew', 2200000, '2019-02-28', 'Finance'),
16 (6, 'Alex', 'chreketo', 4000000, '2019-05-10', 'IT'),
17 (7, 'Yohan', 'Soso', 1230000, '2019-06-20', 'Banking');
18
19 CREATE TABLE Reward {
20   Employee_ref_id INT,
21   date_reward DATE,
22   amount INT,
23   FOREIGN KEY (Employee_ref_id) REFERENCES Employee(Employee_id)
24 };
25
26 INSERT INTO Reward (Employee_ref_id, date_reward, amount) VALUES
27 (1, '2019-05-11', 1000),
28 (2, '2019-02-15', 5000),
29 (3, '2019-04-22', 2000),
30 (4, '2019-06-20', 8000);
31
32 SELECT * FROM Employee;
33 SELECT First_name, Last_name FROM Employee;
34 SELECT First_name AS 'Nombre de empleado' FROM Employee;
35 SELECT LOWER(Last_name) FROM Employee;
36 SELECT UPPER(Last_name) FROM Employee;
37 SELECT DISTINCT Departement FROM Employee;
38 SELECT SUBSTRING(First_name, 1, 4) FROM Employee;
39 SELECT INSTR(First_name, 'h') FROM Employee WHERE First_name = 'John';
40 SELECT RTRIM(First_name) FROM Employee;
41 SELECT LTRIM(First_name) FROM Employee;
```

The right pane shows the output of the script, which includes a table of employee data and a list of last names in various cases.

Employee_id	First_name	Last_name	Salary	Joining_date	Departement
1	Bob	Kinto	1000000	2019-01-20	Finance
2	Jerry	Kansao	6000000	2019-01-15	IT
3	Phillip	Jose	8000000	2019-02-05	Banking
4	John	Abraham	2000000	2019-02-25	Insurance
5	Michael	Mathew	2200000	2019-02-28	Finance
6	Alex	chreketo	4000000	2019-05-10	IT
7	Yohan	Soso	1230000	2019-06-20	Banking

Below the table, the last names are listed in various cases:

First_name	Last_name
Bob	Kinto
Jerry	Kansao
Phillip	Jose
John	Abraham
Michael	Mathew
Alex	chreketo
Yohan	Soso


```

4 Last_name VARCHAR(50),
5 Salary INT,
6 Joining_date DATE,
7 Departement VARCHAR(50)
8 );
9 INSERT INTO Employee (Employee_id, First_name, Last_name, Salary, Joining_date, Departement) VALUES
10 (1, 'Bob', 'Kinto', 1800000, '2019-01-20', 'Finance'),
11 (2, 'Jerry', 'Kansko', 6000000, '2019-01-15', 'IT'),
12 (3, 'Phillip', 'Jose', 8000000, '2019-02-05', 'Banking'),
13 (4, 'John', 'Abraham', 2000000, '2019-02-25', 'Insurance'),
14 (5, 'Michael', 'Mathew', 2200000, '2019-02-28', 'Finance'),
15 (6, 'Alex', 'Chreketo', 4000000, '2019-05-10', 'IT'),
16 (7, 'Yohan', 'Soso', 1200000, '2019-06-20', 'Banking');
17 CREATE TABLE Reward (
18 Employee_ref_id INT,
19 date_reward DATE,
20 amount INT,
21 FOREIGN KEY (Employee_ref_id) REFERENCES Employee(Employee_id)
22 );
23 INSERT INTO Reward (Employee_ref_id, date_reward, amount) VALUES
24 (1, '2019-05-11', 1000),
25 (2, '2019-02-15', 5000),
26 (3, '2019-04-22', 2000),
27 (1, '2019-06-20', 8000);
28 SELECT * FROM Employee;
29 SELECT First_name, Last_name FROM Employee;
30 SELECT First_name AS 'Nombre de empleado' FROM Employee;
31 SELECT LOWER(Last_name) FROM Employee;
32 SELECT UPPER(Last_name) FROM Employee;
33 SELECT DISTINCT Departement FROM Employee;
34 SELECT SUBSTRING(First_name, 1, 4) FROM Employee;
35 SELECT INSTR(First_name, 'h') FROM Employee WHERE First_name = 'John';
36 SELECT RTRIM(First_name) FROM Employee;
37 SELECT LTRIM(First_name) FROM Employee;
38

```

Input for the program (Optional)

```

SUBSTRING(First_name, 1, 4)
| Bob |
| Jerry |
| Phil |
| John |
| Mich |
| Alex |
| Yoha |

INSTR(First_name, 'h')
| 3 |

RTRIM(First_name)
| Bob |
| Jerry |
| Phillip |
| John |
| Michael |
| Alex |

```

onecompiler.com/mysql/43vkwmg8

OneCompiler

43vkwmg8

```

2 Employee_id INT PRIMARY KEY,
3 First_name VARCHAR(50),
4 Last_name VARCHAR(50),
5 Salary INT,
6 Joining_date DATE,
7 Departement VARCHAR(50)
8 );
9 INSERT INTO Employee (Employee_id, First_name, Last_name, Salary, Joining_date, Departement) VALUES
10 (1, 'Bob', 'Kinto', 1800000, '2019-01-20', 'Finance'),
11 (2, 'Jerry', 'Kansko', 6000000, '2019-01-15', 'IT'),
12 (3, 'Phillip', 'Jose', 8000000, '2019-02-05', 'Banking'),
13 (4, 'John', 'Abraham', 2000000, '2019-02-25', 'Insurance'),
14 (5, 'Michael', 'Mathew', 2200000, '2019-02-28', 'Finance'),
15 (6, 'Alex', 'Chreketo', 4000000, '2019-05-10', 'IT'),
16 (7, 'Yohan', 'Soso', 1200000, '2019-06-20', 'Banking');
17 CREATE TABLE Reward (
18 Employee_ref_id INT,
19 date_reward DATE,
20 amount INT,
21 FOREIGN KEY (Employee_ref_id) REFERENCES Employee(Employee_id)
22 );
23 INSERT INTO Reward (Employee_ref_id, date_reward, amount) VALUES
24 (1, '2019-05-11', 1000),
25 (2, '2019-02-15', 5000),
26 (3, '2019-04-22', 2000),
27 (1, '2019-06-20', 8000);
28 SELECT * FROM Employee;
29 SELECT First_name, Last_name FROM Employee;
30 SELECT First_name AS 'Nombre de empleado' FROM Employee;
31 SELECT LOWER(Last_name) FROM Employee;
32 SELECT UPPER(Last_name) FROM Employee;
33 SELECT DISTINCT Departement FROM Employee;
34 SELECT SUBSTRING(First_name, 1, 4) FROM Employee;
35 SELECT INSTR(First_name, 'h') FROM Employee WHERE First_name = 'John';
36 SELECT RTRIM(First_name) FROM Employee;
37 SELECT LTRIM(First_name) FROM Employee;
38

```

STDIN

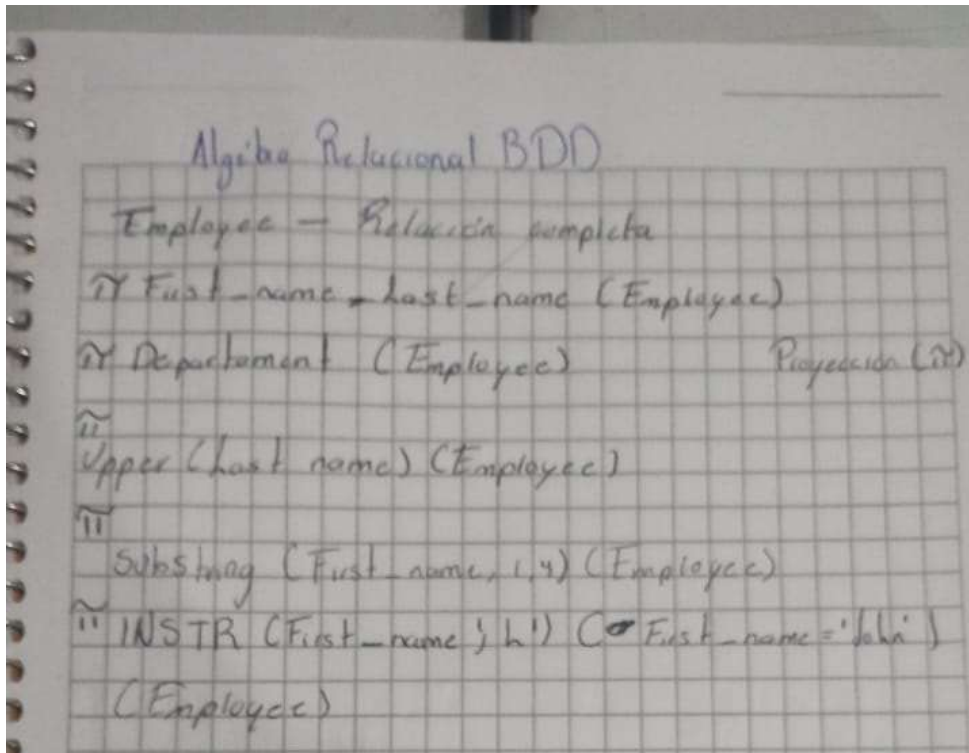
Input for the program (Optional)

```

3
RTRIM(First_name)
| Bob |
| Jerry |
| Phillip |
| John |
| Michael |
| Alex |
| Yohan |

LTRIM(First_name)
| Bob |
| Jerry |
| Phillip |
| John |
| Michael |
| Alex |
| Yohan |

```

Conclusiones

La tarea permitió aplicar los fundamentos del álgebra relacional para representar consultas sobre bases de datos de manera formal y estructurada. A través de operaciones como proyección (π), selección (σ), renombramiento (ρ) y eliminación de duplicados (δ), se logró traducir requerimientos comunes en expresiones precisas que reflejan cómo se manipulan los datos en un entorno relacional.

Además, se identificaron limitaciones del álgebra relacional clásica frente a funciones más avanzadas como el manejo de cadenas o formatos de texto, lo que abre la puerta a explorar lenguajes más expresivos como SQL o extensiones del álgebra relacional.

En resumen, esta práctica fortalece la comprensión de cómo se estructuran las consultas en bases de datos y prepara el terreno para diseñar sistemas más eficientes y robustos en el manejo de la información.

Referencias

Elmasri, R., & Navathe, S. B. (2007). Sistemas de bases de datos (5ª ed., traducción al español). Pearson Educación.

Silberschatz, A., Korth, H. F., & Sudarshan, S. (2011). Fundamentos de bases de datos (6ª ed.). McGraw-Hill Interamericana.

Date, C. J. (2001). Introducción a los sistemas de bases de datos (8ª ed.). Pearson Educación.