Parte 1 – Criando Índices em Banco de Dados

Queries para Responder as Perguntas

1. **Qual o departamento com maior número de pessoas?**

```
SELECT department_id, COUNT(*) AS num_employees
FROM employee
GROUP BY department_id
ORDER BY num_employees DESC
LIMIT 1;
```

7. **Quais são os departamentos por cidade?**

```
SELECT city, department_id
FROM department
 JOIN location ON department.location_id = location.location_id;
```

11. **Relação de empregados por departamento**

```
 SELECT department_id, employee_id, first_name, last_name
 FROM employee
 ORDER BY department_id;
```

Criação dos Índices

1. **Índice para `department_id` na tabela `employee`**

```
CREATE INDEX idx_employee_department_id ON
employee(department_id);
-- Motivo: A consulta para contar o número de empregados por
departamento se beneficiará deste índice.
```

4. **Índice para `city` na tabela `location`**

```
CREATE INDEX idx_location_city ON location(city);
-- Motivo: A consulta para listar departamentos por cidade se
beneficiará deste índice.
```

7. **Índice para `location_id` na tabela `department`**

```
CREATE INDEX idx_department_location_id ON
department(location_id);
-- Motivo: A consulta para listar departamentos por cidade se
beneficiará deste índice.
```

README

```
# Índices Criados para o Cenário de Company

## Descrição do Projeto
Este projeto visa otimizar consultas SQL no cenário de uma empresa,
criando índices apropriados para melhorar a performance das consultas
mais frequentes.

## Índices Criados e Motivos

1. **Índice `idx_employee_department_id` na tabela `employee`**
   - **Motivo**: A consulta para contar o número de empregados por
departamento se beneficiará deste índice, pois ele permite acesso
rápido aos registros de empregados agrupados por departamento.

2. **Índice `idx_location_city` na tabela `location`**
   - **Motivo**: A consulta para listar departamentos por cidade se
beneficiará deste índice, pois ele permite acesso rápido aos registros
de localização agrupados por cidade.
```

3. **Índice `idx_department_location_id` na tabela `department`**
   - **Motivo**: A consulta para listar departamentos por cidade se beneficiará deste índice, pois ele permite acesso rápido aos registros de departamentos associados a uma localização específica.

Queries Utilizadas

1. **Qual o departamento com maior número de pessoas?**
   ```sql
   SELECT department_id, COUNT(*) AS num_employees
   FROM employee
   GROUP BY department_id
   ORDER BY num_employees DESC
   LIMIT 1;
   ```

2. **Quais são os departamentos por cidade?**
```
SELECT city, department_id
FROM department
JOIN location ON department.location_id = location.location_id;
```
6. **Relação de empregados por departamento**
```
SELECT department_id, employee_id, first_name, last_name
FROM employee
ORDER BY department_id;
```

Parte 2 – Utilização de Procedures para Manipulação de Dados em Banco de Dados

Procedure para Manipulação de Dados

```sql
DELIMITER \\

CREATE PROCEDURE manage_employee(
    IN action INT,
    IN emp_id INT,
    IN first_name VARCHAR(50),
    IN last_name VARCHAR(50),
    IN dept_id INT
)
BEGIN
    CASE action
        WHEN 1 THEN
            -- Inserção
            INSERT INTO employee (employee_id, first_name, last_name, department_id)
            VALUES (emp_id, first_name, last_name, dept_id);
        WHEN 2 THEN
            -- Atualização
            UPDATE employee
            SET first_name = first_name, last_name = last_name, department_id = dept_id
            WHERE employee_id = emp_id;
        WHEN 3 THEN
            -- Remoção
            DELETE FROM employee
            WHERE employee_id = emp_id;
        ELSE
            -- Seleção
            SELECT * FROM employee
            WHERE employee_id = emp_id;
    END CASE;
```

```
END \\

DELIMITER ;
```

Chamada da Procedure

```
-- Inserir um novo empregado
CALL manage_employee(1, 101, 'John', 'Doe', 10);

-- Atualizar um empregado existente
CALL manage_employee(2, 101, 'John', 'Smith', 20);

-- Remover um empregado
CALL manage_employee(3, 101, NULL, NULL, NULL);

-- Selecionar um empregado
CALL manage_employee(4, 101, NULL, NULL, NULL);
```

README para Procedures

```
# Procedures para Manipulação de Dados

## Descrição do Projeto
Este projeto visa criar uma procedure para manipulação de dados em um
banco de dados, permitindo inserção, atualização, remoção e seleção de
registros na tabela `employee`.

## Procedure Criada

```sql
DELIMITER \\

CREATE PROCEDURE manage_employee(
    IN action INT,
    IN emp_id INT,
    IN first_name VARCHAR(50),
    IN last_name VARCHAR(50),
    IN dept_id INT
)
BEGIN
    CASE action
        WHEN 1 THEN
            -- Inserção
            INSERT INTO employee (employee_id, first_name, last_name,
department_id)
            VALUES (emp_id, first_name, last_name, dept_id);
        WHEN 2 THEN
            -- Atualização
            UPDATE employee
            SET first_name = first_name, last_name = last_name,
department_id = dept_id
            WHERE employee_id = emp_id;
        WHEN 3 THEN
            -- Remoção
            DELETE FROM employee
            WHERE employee_id = emp_id;
        ELSE
            -- Seleção
            SELECT * FROM employee
            WHERE employee_id = emp_id;
```
```

```
    END CASE;
END \\

DELIMITER ;
```

Chamada da Procedure

```
-- Inserir um novo empregado
CALL manage_employee(1, 101, 'John', 'Doe', 10);

-- Atualizar um empregado existente
CALL manage_employee(2, 101, 'John', 'Smith', 20);

-- Remover um empregado
CALL manage_employee(3, 101, NULL, NULL, NULL);

-- Selecionar um empregado
CALL manage_employee(4, 101, NULL, NULL, NULL);
```