PROVA DO SEMESTRE PASSADO

Exercício 1

1. (1.0) Apresente:

   (a) quais são e detalhe a fundamentação dos componentes que envolvem a teoria de *Backup*.

   (b) quatro tipos de *Backup* juntamente com sua fundamentação.

a. Tempos observados no BD
   i. T = {t0, t1, t2 ... tn}
   ii. Cada T menor é uma “visita” ao banco de dados em um (t)empo, surgindo B(t)
   iii. O backup só é realizado quando há alteração de Tx para Tx + 1
   iv. A diferença entre os tempos não precisa ser uniforme
   v. B(t), t E T.
      i. B(t) = 0 ou banco de dados no tempo inicial
   vi. Define-se como diferencial absoluto do BD a estrutura que representa as atualizações conduzidas sobre o mesmo até um tempo T
      i. **Notação:** Δt := Bt - B0, t E T
   vii. Define-se como diferencial incremental, diferencial relativo ou ainda diferencial de segunda ordem, as atualizações que ocorrem no BD entre dois diferenciais consecutivos em determinado tempo T.
      i. **Notação:** Δ^2t = Δt = Δt-1, t E T \ {t0}. ou Δ^2t = Bt - Bt-1

---

b. Ao longo do semestre aprendemos 4 tipos de backup em T-SQL
   i. Full backup: Faz o backup completo do banco de dados, desde o momento de sua criação até o seu estado atual

ii. Backup diferencial: Salva o estado inicial do banco e seus diferenciais absolutos

iii. Backup incremental: Salva o estado inicial do BD e seus diferenciais incrementais

iv. Backup sintético: Salva o estado inicial, salva os incrementais e, em cada um destes salvamentos, atualiza-se B0 (estado inicial do banco)

Exercício 2

2. (1.0) Julgue como verdadeiro (V) ou falso (F) e apresente a devida justificativa para tal com respeito aos itens a seguir:

   (a) ( ) Independente de configuração, um *backup* do tipo incremental é menor do que um *backup* do tipo diferencial.

   (b) ( ) B_0 é o estado inicial do banco de dados, ou seja, é apenas a estrutura do banco de dados sem nenhum registro tabular.

   (c) ( ) Sejam t_1 e t_2 dois instantes de tempos do banco de dados tais que $t_1 < t_2$. Então, é sabido que o número de registros de B_{t_1} é inferior ao número de registros de B_{t_2}.

   (d) ( ) O número de registros de qualquer estado do banco B_t é sempre maior que o número de registros de um *backup* diferencial, mesmo que este seja referente a um estado posterior a t.

a. F. Pois tamanho de um backup incremental depende das alterações. Em um backup incremental, apenas os dados que foram alterados desde o último backup são copiados, o que pode resultar em backups menores.

b. F. B0 realmente é o estado inicial do banco de dados, mas não necessariamente sem nenhum registro

c. F. Não necessariamente, pode ser que na transação t1 tenham sido inseridos registros e na transação t2 tenham sido deletados registros, o que faz com que a quantidade de registro de t1 é maior que t2

d. F. Um backup completo captura todos os dados do banco, enquanto um backup diferencial armazena apenas as alterações desde o último backup completo. O número de registros no backup diferencial pode exceder o do backup completo, dependendo da quantidade de alterações realizadas.

Exercício 3

3. (1.5) Escreva um procedimento que conduza um *backup* sobre o banco de dados `BD_prova`. Especificamente, esse procedimento deve receber dois parâmetros literais:

   - `@tipo_de_backup`: parâmetro literal que pode assumir os valores "full" e "diff". Caso assuma o primeiro, o procedimento deve conduzir um *backup* completo. Caso assuma o segundo, deve conduzir um *backup* diferencial.
   - `@diretorio`: parâmetro literal que indica o endereço e o nome do diretório em que o banco de dados deve ser salvo. Por exemplo, esta variável pode assumir o valor "C:/Backup".

   Além disso, o arquivo final de backup deve seguir a respectiva nomenclatura:

   `BD_bak_<tipo>_<ano>_<mes>_<dia>_<hora>_<minuto>.bak`,

   na qual,

   - `<tipo>`: é o conteúdo da variável `@tipo_de_backup`;
   - `<ano>`: é o ano em que o *backup* será feito;
   - `<mes>`: é o mês em que o *backup* será feito;
   - `<dia>`: é o dia em que o *backup* será feito;
   - `<hora>`: é a hora em que o *backup* será feito;
   - `<minuto>`: é o minuto em que o *backup* será feito.

```
CREATE OR ALTER PROCEDURE make_backup
@type INT,
@path VARCHAR(MAX)
AS
BEGIN
    DECLARE
        @file_name VARCHAR(MAX) =
            CONCAT(
                'BD_BAK',
                '_',
                @type,
                '_',
```

```
                CONVERT(VARCHAR(MAX), DATEPART(YEAR, GETDATE(
                '_',
                CONVERT(VARCHAR(MAX), DATEPART(MONTH, GETDATE
                '_',
                CONVERT(VARCHAR(MAX), DATEPART(DAY, GETDATE()
                '_',
                CONVERT(VARCHAR(MAX), DATEPART(HOUR, GETDATE(
                '_',
                CONVERT(VARCHAR(MAX), DATEPART(MINUTE, GETDAT
                '_',
                CONVERT(VARCHAR(MAX), DATEPART(SECOND, GETDAT
                '.bak'
            )
    DECLARE
        @full_path VARCHAR(MAX) = CONCAT(@path , @file_name);
    IF @type = 2
        BACKUP DATABASE @dbname
        TO DISK = @full_path
        WITH DIFFERENTIAL
    ELSE
        BACKUP DATABASE @dbname
        TO DISK = @full_path;
END;
```

Exercício 4

4. (1.5) Suponha que o banco de dados `BD_prova` tenha uma tabela `tb_prova`. Altere o procedimento definido no exercício anterior para que o mesmo execute o *backup* apenas se a tabela `tb_prova` teve, pelo menos, 10000 atualizações desde o último *backup* independente do tipo.

```
CREATE TABLE log_tb(
    quantidade INT
);

INSERT INTO log_tb VALUES (0)
```

```
CREATE OR ALTER TRIGGER handle_insert_data
ON tb
AFTER INSERT, DELETE, UPDATE
AS
BEGIN
    DECLARE
        @qtd_inseridos INT = (SELECT COUNT(*) FROM inserted),
        @qtd_removidos INT = (SELECT COUNT(*) FROM deleted),
        @incremento INT = 0;

        IF @qtd_inseridos > 0 AND @qtd_removidos > 0
            SET @incremento = 1;
        ELSE
            SET @incremento = @qtd_inseridos + @qtd_removidos

    UPDATE log_tb
    SET quantidade = quantidade + @incremento;

END;

INSERT INTO tb
VALUES
('gap', 25);

EXEC add_linhas 9998

SELECT * FROM log_tb

CREATE OR ALTER PROCEDURE make_backup_10k
AS
BEGIN
    DECLARE
        @qtd_registros INT = (SELECT quantidade FROM log_tb),
        @file_name VARCHAR(MAX) =
            CONCAT(
                DB_NAME(),
                '_',
```

```
                'completo',
                '_',
                CONVERT(VARCHAR(MAX), DATEPART(YEAR, GETDATE(
                '_',
                CONVERT(VARCHAR(MAX), DATEPART(MONTH, GETDATE
                '_',
                CONVERT(VARCHAR(MAX), DATEPART(DAY, GETDATE()
                '_',
                CONVERT(VARCHAR(MAX), DATEPART(HOUR, GETDATE(
                '_',
                CONVERT(VARCHAR(MAX), DATEPART(MINUTE, GETDAT
                '_',
                CONVERT(VARCHAR(MAX), DATEPART(SECOND, GETDAT
                '.bak'
            );
        DECLARE
            @full_path VARCHAR(MAX) = CONCAT('C:/Backup/', @f

    IF @qtd_registros >= 10000
        BACKUP DATABASE db06112024
        TO DISK = @full_path
        TRUNCATE TABLE log_tb
END;


INSERT INTO tb
VALUES
('gap', 25);

EXEC make_backup_10k
```

Exercício 5

5. (1.0) Descreva o significado de operações, de transação e de concorrência sobre um banco de dados.

Operações: São movimentações de ações ou transformações de acesso que ocorrem diretamente no BD

- Leitura: Vai usar o buffer para procurar o valor informado no banco e jogar na memória
- Escrita: É utilizado para modificar uma linha da tabela, seja inserindo, alterando ou excluindo
    - Vai usar o buffer para procurar o valor informado no banco, e após isso colocar o valor informado no item
    - É possível recuperar registros excluídos do banco

Transações: Uma transação em BD é um conjunto de operações

Concorrência: Gerenciamento de múltiplas operações simultâneas para garantir a integridade dos dados, evitando conflitos e mantendo a consistência durante transações paralelas. Para definir a prioridade de execução, é necessário definir um plano de execução **(schedule)**

Exercício 6

6. (1.0) Explique o que significa a estratégia ACID para processamento de transações em um banco de dados. Além disso, apresente quatro tipos de isolamentos existentes no dialeto T-SQL destacando as diferenças e as semelhanças de cada um com respeito a gravações e a leituras no banco de dados.

A estratégia ACID consiste nos seguintes pontos

1. **Atomicidade:** ou todas as operações são exercitadas ou nenhuma
2. **Consistência:** BD consistente deve gerar BD também consistente
3. **Isolamento:** uma transação não pode interferir em outra
4. **Durabilidade:** os efeitos das transações são permanentes

Níveis de isolamento:

READ COMMITTED → Permite a leitura de itens confirmados apenas
READ UNCOMMITTED → Permite a leitura de todos os itens
REPEATABLE READ → Garante leitura repetível
SERIALIZABLE → Garante leitura serializavel
SNAPSHOT → Cria uma visão do banco

Exercício 7

7. (1.5) Suponha que em no `BD_prova` exista uma tabela `tb_cliente` com a coluna `saldo` do tipo numérico real e com a chave primária `id` do tipo numérico inteiro. Crie um procedimento que efetue da **maneira mais segura possível** a transferência de crédito de um cliente para o outro.
Dica: utilize o conceito de transações.

```
CREATE OR ALTER PROCEDURE update_valor
    @valor FLOAT,
    @id1 INT,
    @id2 INT
AS
BEGIN
    IF @valor <= 0
    BEGIN
        PRINT ('O valor de transferência deve ser maior que z
        RETURN;
    END

    BEGIN TRY
        BEGIN TRANSACTION;

        DECLARE @saldoAtual FLOAT;
        SELECT @saldoAtual = saldo FROM tb_cliente WHERE id =

        IF @saldoAtual IS NULL
        BEGIN
            PRINT ('Usuário 1 não encontrado.');
            ROLLBACK TRANSACTION;
            RETURN;
        END

        IF @saldoAtual < @valor
        BEGIN
            PRINT ('Saldo insuficiente para realizar a transf
            ROLLBACK TRANSACTION;
            RETURN;
```

```
        END

        IF NOT EXISTS (SELECT 1 FROM tb_cliente WHERE id = @i
        BEGIN
            PRINT ('Usuário 2 não encontrado.');
            ROLLBACK TRANSACTION;
            RETURN;
        END

        UPDATE tb_cliente
        SET saldo = saldo - @valor
        WHERE id = @id1;

        UPDATE tb_cliente
        SET saldo = saldo + @valor
        WHERE id = @id2;

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        ROLLBACK TRANSACTION;
    END CATCH;
END;
```

Exercício 8

8. (1.5) No código a seguir, o uso do conceito de *savepoints* é aplicado no dialeto T-SQL. Indique quais devem ser as respostas das seleções em forma tabular em cada uma das indicações de comentários (*), (**), (***) e (****).

```
DROP TABLE IF EXISTS tb_ex_save_point;

CREATE TABLE tb_ex_save_point(
    id INT PRIMARY KEY IDENTITY(1, 1),
    valor INT
);
```

```
INSERT INTO
    tb_ex_save_point
VALUES
(0);

SELECT
    *
FROM
    tb_ex_save_point;

-- (*)
BEGIN TRANSACTION tr1
INSERT INTO
    tb_ex_save_point
VALUES
(1);

SAVE TRANSACTION tr2
SELECT
    *
FROM
    tb_ex_save_point;

-- (**)
INSERT INTO
    tb_ex_save_point
VALUES
(2);

ROLLBACK TRANSACTION tr2 SAVE TRANSACTION tr3
INSERT INTO
    tb_ex_save_point
VALUES
(3);

DECLARE @count INT = (
    SELECT
```

```
        COUNT(*)
    FROM
        tb_ex_save_point
);

IF @count <= 2 BEGIN COMMIT TRANSACTION tr3;

END
ELSE BEGIN ROLLBACK TRANSACTION tr3;

INSERT INTO
    tb_ex_save_point
VALUES
(4);

END;

SELECT
    *
FROM
    tb_ex_save_point;

-- (***)
COMMIT TRANSACTION tr1;

SELECT
    *
FROM
    tb_ex_save_point;

-- (****)
```

Comentário	Id	Valor
(*)	1	0
(**)	1 2	0 1
(***)	1 2	0 1

Comentário	Id	Valor
	5	4
(****)	1 2 5	0 1 4