BANCO DE DADOS

AULA 5

Profº Lucas Rafael Filipak

CONVERSA INICIAL

O primeiro objetivo desta aula é aprender a consultar mais de uma tabela dentro da mesma instrução, retornando dados mais íntegros, finalizando assim os comandos DML; entender como funciona as transações e onde elas podem ser utilizadas; aprender a criar um usuário para o banco de dados; fazer a atribuição e a remoção de privilégios de acesso aos usuários; estudar a importância da criação dos índices para melhorar (agilizar) a busca dos registros dentro das tabelas.

TEMA 1 – CONSULTA ENTRE TABELAS

Até agora foi estudado o comando SELECT e alguns outros comandos para filtrar ou refinar a seleção, mas sempre utilizando como base de seleção uma única tabela de cada vez. O comando JOIN é utilizado para recuperar dados entre mais de uma tabela, obtendo assim registros mais completos.

Figura 1 – Tabela alunos e tabela cidade utilizadas para exemplificar

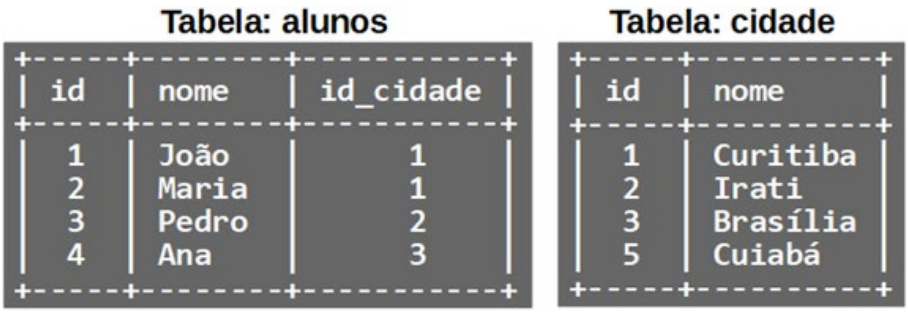


Tabela: alunos

id	nome	id_cidade
1	João	1
2	Maria	1
3	Pedro	2
4	Ana	3

Tabela: cidade

id	nome
1	Curitiba
2	Irati
3	Brasília
5	Cuiabá

Para entender o cenário, analise a Figura 1. O objetivo é mostrar, em uma única consulta, o nome do aluno e o nome da cidade do aluno. Na tabela *Cidade* são cadastradas as cidades que vão ser utilizadas.

Na tabela *Alunos* são cadastrados os dados dos alunos. A coluna *id cidade* contém um valor que representa o id de uma cidade e tem relação com a coluna "id" da tabela *Cidade*. Essa técnica de relacionar dados deixa o banco de dados mais íntegro, pois a cidade de Curitiba foi cadastrada apenas uma vez (tabela *Cidade*) e é utilizado seu id na tabela *Alunos*, diminuindo assim o seu tamanho de armazenamento e de transferência de dados. Imagine que para cada aluno cadastrado na tabela *Alunos*, em vez do id da cidade fosse gravado o nome da cidade. Não é difícil entender que se utiliza menos espaço para armazenar o número 1 (que é o código do cadastro de Curitiba) do que a palavra *Curitiba*. O

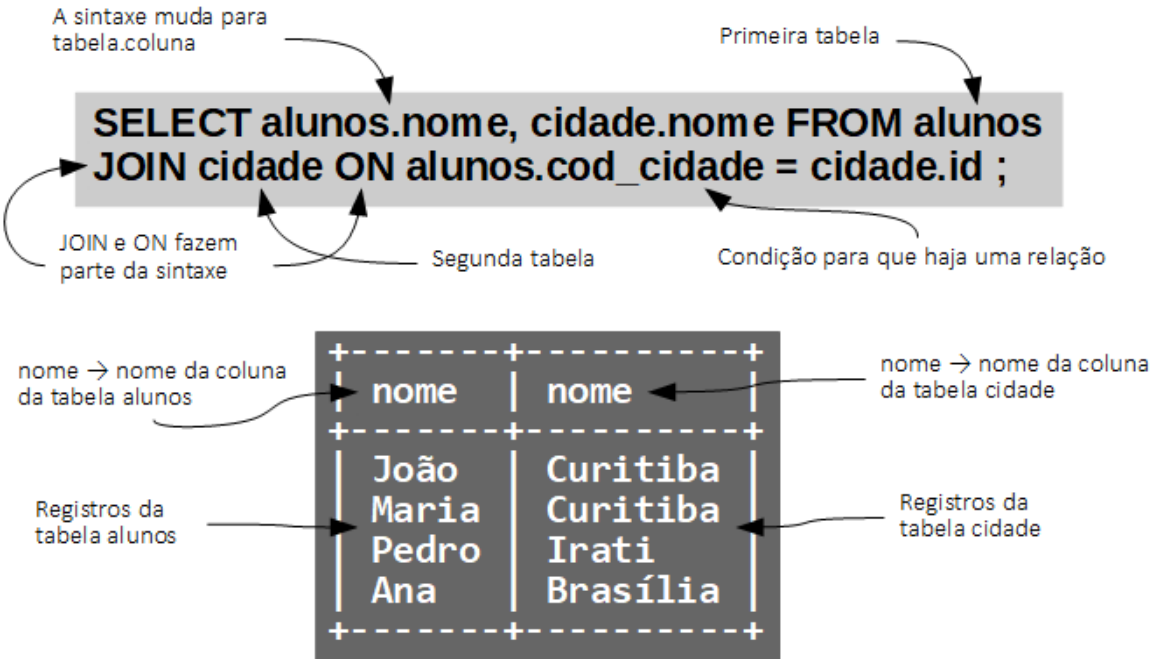
momento é oportuno para relembrar o DER e a cardinalidade utilizada no exemplo da Figura 1. Um aluno tem somente uma cidade cadastrada (1:1), mas uma cidade cadastrada pode ser utilizada por vários alunos (1:N).

Figura 2 – Relação e cardinalidade entre as tabelas

O comando JOIN (ou INNER JOIN) foi utilizado para consultar dados entre as tabelas *Alunos* e *Cidade*. Para informar as colunas de seleção utiliza-se o nome da tabela e depois o nome da coluna. Na Figura 3 foi utilizado a coluna *nome* da tabela *Alunos* (alunos.nome) e a coluna *nome* da tabela *Cidade* (cidade.nome).

O parâmetro ON é a cláusula da condição e pode ser substituído pela palavra *WHERE*. Para que haja resultados na consulta a condição ON ou WHERE deve ser verdadeira. Note que a seleção não retornou o registro da cidade de *Cuiabá*. Isso ocorreu, pois na tabela *alunos* nenhum registro possui o valor da coluna *id_cidade* como 5.

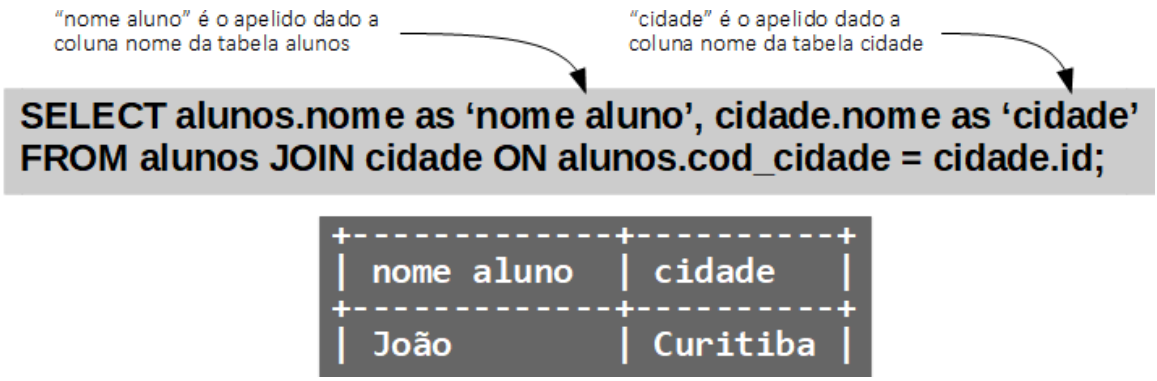
Figura 3 – Exemplo do uso do comando JOIN



No exemplo da Figura 3, as duas colunas utilizadas têm o mesmo nome (*nome*) e isso pode prejudicar na interpretação do resultado. É possível nomear

as colunas ou tabelas, momentaneamente, para facilitar a visualização. O comando para renomear (apelidar) uma coluna ou uma tabela é o *Alias*. Repare na Figura 4, as colunas receberam apelidos.

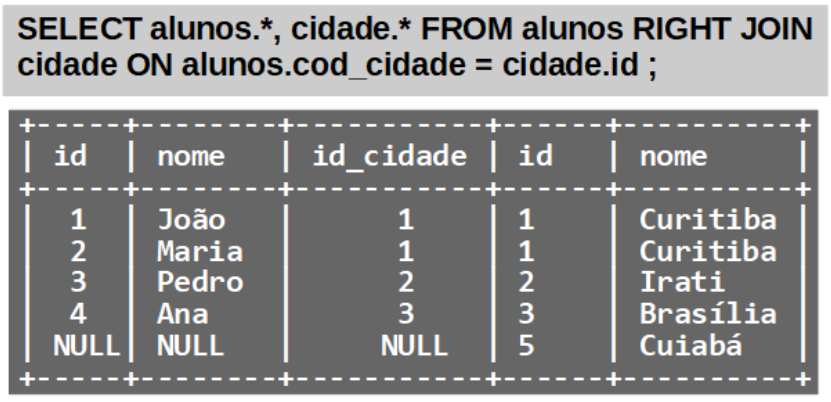
Figura 4 – Exemplo do uso do comando para apelidar as colunas



Repare na Figura 4 que a coluna *nome* da tabela *alunos* recebeu o *alias* nome aluno. Para isso utiliza-se do comando AS logo após a coluna que vai receber o *alias*. Os apelidos (*alias*) não alteram a estrutura da tabela, somente na visualização da consulta.

Existem outros tipos de *JOIN*. Camargo (2010) explica que a cláusula *RIGHT JOIN* ou *RIGHT OUTER JOIN* permite obter não apenas os dados relacionados de duas tabelas, mas também os dados não relacionados encontrados na tabela à direita da cláusula *JOIN*. Repare na Figura 5 que o comando *RIGHT JOIN* retornou todos os registros da tabela à direita do *JOIN* (*cidade*) e os registros que coincidem com a igualdade do *JOIN* na tabela *alunos*.

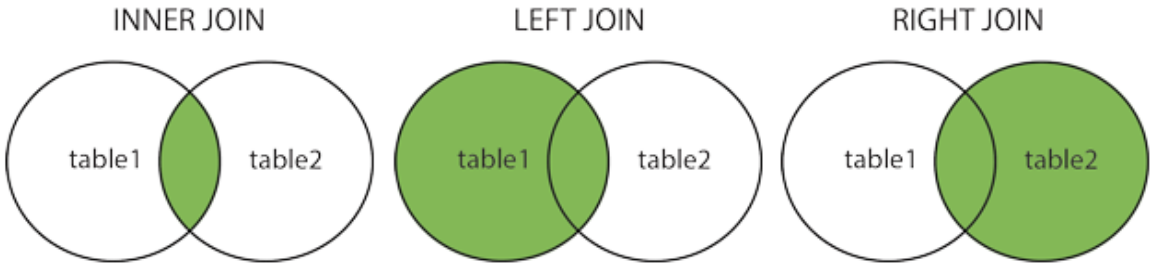
Figura 5 – Exemplo do uso do comando *RIGHT JOIN*



Ao contrário do *RIGHT JOIN*, a cláusula *LEFT JOIN* ou *LEFTOUTER JOIN* retorna todos os dados encontrados na tabela à esquerda de *JOIN* (Camargo, S.d.).

Observe na Figura 6 uma representação gráfica dos 3 tipos de *JOINs* explicados.

Figura 6 – Representação dos diferentes tipos de *JOINs*



Fonte: W3schools, S.d.

TEMA 2 – OUTRAS FUNÇÕES DO *SELECT*

Algumas outras funções podem ser utilizadas com a instrução *SELECT* permitindo enxergar os dados de maneiras mais específicas, ajudando na tomada de decisão. A tabela de Figura 7 será utilizada para os próximos exemplos.

Figura 7 – Tabela utilizada como exemplo nas próximas explicações

cod	nome	sexo	salario
1	Augusto	M	4500.00
2	Paula	F	5300.00
3	Maria	F	3300.00
4	César	M	4800.00

Para retornar a média dos valores de uma coluna, utiliza-se a função AVG. Na Figura 8, a instrução executada retornou a média dos valores dos salários de todos os funcionários.

Figura 8 – Exemplo do uso da função AVG

```
SELECT avg(salario)
FROM funcionarios ;
```

avg(salario)
4475.000000

Essas funções podem compor o comando *SELECT* junto com outras colunas e são interpretados como uma nova coluna, que fica gravada apenas na

memória. Na Figura 9, a consulta retornou duas colunas: *sexo* e a *média dos salários.*

Figura 9 – Exemplo do uso da função AVG agrupando pela coluna *sexo*

```
SELECT sexo, avg(salario)
FROM funcionarios
GROUP BY sexo;
```

```
+--------------------+
| sexo | avg(salario) |
+--------------------+
|  M   |  4650.000000 |
|  F   |  4300.000000 |
+------+-------------+
```

Utiliza-se a função de ordenação GROUP BY para agrupar os registros que possuem valores comuns. A Figura 9 mostra a média de salários por grupo (*sexo*).

A função SUM retorna a soma dos valores de uma coluna e possuí a mesma dinâmica do parâmetro AVG. Observe na figura 10.

Figura 10 – Exemplo do uso da função SUM

```
SELECT sum(salario)
FROM funcionarios ;
```

```
+--------------+
| sum(salario) |
+--------------+
|     17900.00 |
+--------------+
```

Todas as funções apresentadas nesse capítulo também podem utilizar a cláusula WHERE. Além de somar (SUM) e fazer a média (AVG) também se pode contar os registros. Veja na Figura 11.

Figura 11 – Exemplo do uso da função COUNT

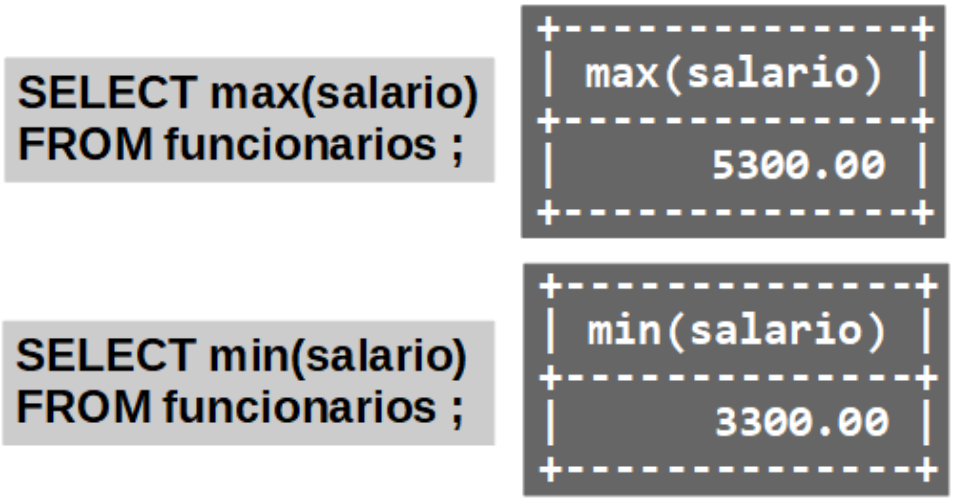
```
SELECT count(*) as 'funcionarios'
FROM funcionarios ;
```

```
+--------------+
| funcionarios |
+--------------+
|            4 |
+--------------+
```

A instrução retornou o número 4, pois existem 4 registros na tabela *Funcionários*. No exemplo da Figura 11, a função *count(*)* foi apelidada de *funcionários* (note na visualização). O apelido é para facilitar a identificação da coluna. Repare na Figura 10 que não foi apelidada a coluna e foi mostrado um(salario) no retorno da consulta. Outras duas funções interessantes são MAX

e MIN, que retornam o valor máximo e o valor mínimo de uma coluna. Na Figura 12 percebe-se que foram retornados o maior e o menor salário encontrado.

Figura 12 – Exemplo do uso das funções MAX e MIX

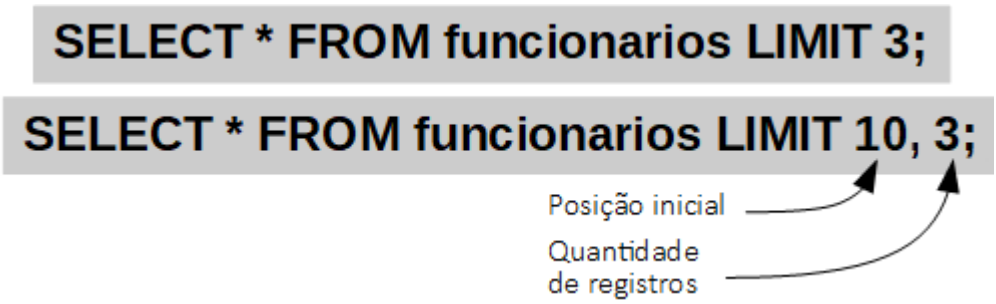


Muitas vezes existem muitos registros dentro das tabelas, sendo necessário limitar o retorno das consultas. A função LIMIT faz isso de duas maneiras:

- LIMIT 3 → retorna os 3 primeiros registros (número máximo);
- LIMIT 10, 3 → retorna 3 registros após o 10º registro (11, 12 e 13 registros).

Visualize na Figura 13 dois exemplos da utilização da função LIMIT.

Figura 13 – Exemplo do uso da função LIMIT



TEMA 3 – *DATA CONTROL LANGUAGE* (DCL)

Os comandos que fazem parte dessa categoria são responsáveis por definir critérios de segurança em relação aos usuários dentro de um banco de dados. A DCL ou Linguagem de Controle de Dados é uma subcategoria da DML, que controla os aspectos de autorização de dados e permissões dos usuários, controlando o acesso, a manipulação e a visualização dos dados.

Antes de iniciar os estudos sobre as permissões, é preciso criar uma nova identificação para o banco de dados. A sintaxe para a criação de uma identificação é bem simples, mas somente uma identificação que tenha permissão para criar uma nova identificação vai conseguir executar o comando. Observe a Figura 14.

Figura 14 – Exemplo da sintaxe do comando CREATE USER

```
CREATE USER 'novo_usuario' IDENTIFIED BY 'senha' ;
```

Na Figura 15 foi criada a identificação *chefe* com a senha 123. A identificação que é recém-criada não tem acesso às databases que existem no banco de dados.

Figura 15 – Exemplo do uso do comando CREATE USER

```
CREATE USER 'chefe' IDENTIFIED BY '123' ;
```

Existem dois comandos principais para fornecer e retirar as permissões das identificações: GRANT e REVOKE.

- GRANT → dá as permissões, autorizando o usuário a executar ou setar operações;
- REVOKE → retira as permissões, removendo ou restringindo a capacidade de um usuário de executar operações.

As sintaxes dos dois comandos são bem parecidas e atuam de maneira opostas. Na Figura 16 a sintaxe do comando GRANT é:

Figura 16 – Exemplo da sintaxe do comando GANT

```
GRANT direitos ON nome_tabela TO identificacao ;
```

Para explicar a sintaxe, é utilizado um fragmento de texto do livro de Alves (2014, p. 132):

- direitos: indica os direitos que podem ser concedidos ao usuário. São eles: ALL PRIVILEGES, SELECT, INSERT, UPDATE e DELETE.
- nome_tabela: é a tabela de dados ou visão na qual será aplicada a concessão dos direitos.
- identificação: é uma identificação da autorização para a qual os privilégios foram concedidos.

O comando da Figura 17 dá permissão para a identificação chefe para executar uma consulta na tabela *Alunos*.

Figura 17 – Exemplo do uso do comando GRANT

```
GRANT select ON alunos TO chefe ;
```

Como só foi dada essa permissão para a identificação chefe, se ele executar o comando *SHOW TABLES* vai aparecer somente a tabela *Alunos* (a única em que ele tem permissão).

A Tabela 1 mostra a lista das principais permissões que se pode atribuir para uma identificação.

Tabela 1 – Lista de permissões

Privilégio	Descrição
CREATE	criar novas tabelas ou bases de dados
DROP	deletar tabelas ou bases de dados
DELETE	deletar registros
INSERT	inserir registros
SELECT	selecionar registros
UPDATE	atualizar/modificar registros

A Figura 18 exemplifica a atribuição das permissões para a identificação chefe de selecionar, inserir, deletar e modificar a tabela *Alunos*.

Figura 18 – Exemplo do uso do comando GRANT

```
GRANT select, insert, delete, update ON alunos TO chefe ;
```

Para fornecer mais de uma permissão para uma identificação, basta colocar os privilégios separados por vírgula. Como a identificação chefe vai ser uma identificação principal, ele receberá todos os privilégios (permissões).

Após a execução do comando da Figura 19, a identificação chefe vai ter um acesso a todos os privilégios em todas as bases de dados.

Figura 19 – Exemplo do uso do comando GRANT

```
GRANT all privileges ON *.* TO chefe ;
```

Existem três níveis de privilégios. Observe a Tabela 2:

Tabela 2 – Nível de privilégio

.	Privilégio global
db.*	Qualquer tabela do banco db
db.tb	Apenas a tabela tb do banco de dados db. Para especificar apenas algumas colunas de uma determinada tabela, estas deverão ser listadas ao lado do privilégio (priv (*colunas*))

Fonte: Duarte, S.d.

O comando *REVOKE* remove os privilégios de uma identificação. A sintaxe para o comando *REVOKE* é um pouco parecida com o comando *GRANT*. Observe a Figura 20.

Figura 20 – Exemplo da sintaxe do comando REVOKE

```
REVOKE direitos ON nome_tabela FROM identificacao ;
```

O comando *REVOKE* utiliza a mesma lista de permissões e os mesmos níveis de privilégios do comando *GRANT*. Pode-se remover uma ou mais permissões em um determinado nível. Ao executar o comando da Figura 21, a identificação chefe não pode mais executar o comando *SELECT* na tabela *Alunos*.

Figura 21 – Exemplo do uso do comando *REVOKE*

```
REVOKE select ON alunos FROM chefe ;
```

Também é possível remover todas as permissões da identificação chefe em um determinado nível, na Figura 22 são removidas as permissões de todas as tabelas do banco sistema.

Figura 22 – Exemplo do uso do comando *REVOKE*

```
REVOKE all privileges ON sistema.* FROM chefe ;
```

Da mesma maneira que se pode criar uma nova identificação, também é possível excluir uma identificação já existente. Essa operação é irreversível e

todos os privilégios são excluídos juntos com a identificação. Observe o comando *DROP USER* na Figura 23.

Figura 23 – Exemplo do uso do comando DROP USER

```
DROP USER 'chefe' ;
```

TEMA 4 – *TRANSACT CONTROL LANGUAGE* (TCL)

Para começar a falar dos comandos de transação, é preciso entender o que são as transações. As transações são um conjunto de operações (comandos) que devem ser executados sem erros para que a transação seja efetivada.

Exemplificando as transações, pode-se pensar em uma transferência bancária entre duas pessoas. O usuário que faz a transferência somente informa os dados do outro usuário e recebe um aviso que a operação foi concluída. Mas para validar tal operação, o sistema verifica os seguintes itens:

- Os dados do destinatário estão corretos;
- Tem dinheiro na conta do remetente;
- Tirar o dinheiro do remetente e passar para o destinatário (atualizando as duas contas).

Olhando os itens da transação nota-se que para realizar a transação são utilizados comandos DML. O *SELECT* é utilizado nos dois primeiros itens e dois *UPDATE* no último item (um para tirar da conta do remetente e outro para colocar na conta do destinatário). As transações devem assegurar:

- Consistência dos dados;
- Atomicidade (todos os comandos devem ser executados com sucesso, se um falhar, nenhuma operação deve ser realizada);
- Integridade do banco de dados.

Uma transação primeiramente é realizada primeira em memória e só são transmitidas fisicamente para o banco de dados após a confirmação de que todas as instruções foram efetuadas com sucesso. São três os comandos principais na categoria TCL.

- *Begin* – indica o início de uma transação e todos os comandos da transação devem vir abaixo de Begin;

- *Commit* – é o fim da transação, executando as instruções no banco de dados (permanente);
- *Rollback* – também é o fim da transação, mas cancela todas as alterações efetuadas pois algo deu errado. Nada será alterado no banco de dados.

A partir do momento em que o cliente pede a transferência bancária, todas as instruções para que a transferência seja efetuada são executadas automaticamente. O comando *Rollback* não significa desfazer a ação, mas sim voltar ao estado original. A maioria dos comandos SQL são transacionais e executam uma transação com efeito imediato no banco de dados. Utilizam-se os comandos da TCL quando é preciso executar vários comandos ao mesmo tempo, de forma automática e todos eles devem ser executados com sucesso.

Observe, na Figura 24, que a primeira instrução a ser executada é um *update* da tabela *Conta_corrente*, retirando o valor desejado da coluna *saldoConta* do usuário remetente. Nesse ponto já é criada uma cópia na memória com a alteração feita pelo *UPDATE*. Em seguida, na outra instrução do UDPATE, é somado o valor na conta do usuário destinatário. Nesse ponto, as duas modificações ficam gravadas apenas na memória. O comando *COMMIT* verifica se todas as instruções foram gravadas com sucesso na memória e efetiva a operação gravando definitivamente no banco de dados.

Figura 24 – Exemplo do uso do comando *TRANSACTION*

```
BEGIN TRANSACTION;

UPDATE CONTA_CORRENTE
  set saldoConta= saldoConta - @Valor
  where numConta = @contaDe;

UPDATE CONTA_CORRENTE
  set saldoConta= saldoConta + @Valor
  where numConta = @contaPara;

COMMIT;
```

É possível nomear as transações. Na Figura 25 a transação *transferencia* tem uma instrução *INSERT* na tabela *Uf*. Da mesma forma do exemplo da Figura 25, quando a instrução é concluída os dados não são inseridos na tabela *Uf*, pois o comando *COMMIT* não foi utilizado, ficando os dados inseridos apenas em memória. O comando *ROLLBACK* no final da transação descarta toda a transação, não inserindo nada no banco de dados e apagando os dados da memória.

Figura 26 – Exemplo do uso do comando *TRANSACTION* sem *COMMIT*

```
BEGIN TRANSACTION 'transferencia'
    INSERT INTO uf VALUES('pr'), ('sc');
ROLLBACK TRANSACTION 'transferencia';
```

Quando todas as instruções SQLs da transação forem realizadas com sucesso, a transação é finalizada e gravada no banco de dados. Caso algo gere uma falha, a transação é desfeita e nada é alterado no banco de dados. As instruções *COMMIT* e *ROLLBACK* controlam as transações.

Para Puga, França e Goya (2013, p. 202)

> Por serem instruções de controle das transações, *COMMIT* e *ROLLBACK* exercem funções determinantes para a confirmação ou descarte da transação. No caso, a instrução *COMMIT* confirma as transações, efetivando as manipulações de dados realizadas nas tabelas fisicamente, ou seja, as operações necessárias para realizar a transação ocorrem em memória e não são registradas no banco de dados até que a instrução *COMMIT* seja acionada. Em contrapartida, ao acionar o *ROLLBACK* as transações que ainda não tenham sido confirmadas e estejam em memória são descartadas.

Após o comando *COMMIT*, as instruções são passadas da memória para o banco de dados, não havendo como reverter a operação realizada.

TEMA 5 – ÍNDICES

Os índices são criados para facilitar e agilizar as consultas dos registros no banco de dados. Antes de começar a falar dos índices é importante saber como a gravação dos dados ocorre de maneira física. A explicação foi retirada do *site* DevMedia:

> Os registros são armazenados em páginas de dados, páginas estas que compõem o que chamamos de pilha, que por sua vez é uma coleção de páginas de dados que contém os registros de uma tabela. Cada página de dados tem seu tamanho definido em até 8 Kb, apresenta um cabeçalho, também conhecido como header, que contém arquivos de links com outras páginas e identificadores (*hash*) que ocupam a nona parte do seu tamanho total (8 Kb) e o resto de sua área é destinado aos dados. Quando são formados grupos de oito páginas (64 Kb), chamamos este conjunto de *extensão*. Os registros de dados não são armazenados em uma ordem específica, e não existe uma ordenação sequente para as páginas de dados. As páginas de dados não estão vinculadas a uma lista, pois implementam diretamente o conceito de pilhas. Quando são inseridos registros em uma página de dados e ela se encontra quase cheia, as páginas de dados são divididas em um link é estabelecido para marcações e ligações entre elas" (DevMedia, S.d.).

Depois da explicação de como os dados são armazenados fisicamente, o próximo passo é analisar os dois métodos de pesquisa dos dados:

- **Exame nas tabelas** → a consulta percorre todos os registros de todas as páginas e seleciona apenas os que são verdadeiros segundo a cláusula *WHERE*;
- **Usando índices** → percorre a estrutura da árvore do índice, comparando e extraindo somente os registros que são verdadeiros segundo a cláusula *WHERE*.

O interpretador SQL tem um otimizador de consultas, que analisa a consulta e direciona para o método mais eficiente. Para ficar mais claro a criação da árvore de índice e sua utilização, será analisado um exemplo não computacional.

O livro é um ótimo exemplo. Imagine que você está pesquisando algum conteúdo, mas para isso tem que olhar em todas as páginas do livro. Se o livro for pequeno, a dificuldade é menor, mas, mesmo assim, se você precisar fazer várias consultas e para cada consulta ter que folhar todas as páginas, o trabalho se torna moroso e cansativo. E se o livro tiver umas 500 páginas? Aí o trabalho se torna impossível de ser realizado manualmente.

A consulta que utiliza o exame nas tabelas é um método manual que percorre todas as páginas que possuam dados gravados. Agora vamos voltar ao exemplo e nos atentar ao índice do livro. Nos livros que contêm índices, essa pesquisa é realizada de maneira muito mais fácil e ágil, pois não é preciso percorrer todas as páginas buscando o conteúdo desejado. A árvore de índice tem a função semelhante à dos índices dos livros físicos.

Como foi visto, os índices são utilizados para melhorar as consultas nos registros, mas é preciso levar em conta que a criação da árvore de índice consome um grande espaço em disco, podendo se tornar um problema se o banco está armazenado em um *storage*.

Saiba mais

Storage é um equipamento que armazena os dados da rede local de uma empresa ou residência, podendo ser um simples HD até um complexo sistema de armazenamento com vários *petabytes*. Além disso, esse tipo de equipamento pode ser implementado como servidor de arquivos, fazer *backup* ou ser uma área para centralizar, processar ou compartilhar dados (*Controle Net*, S.d.).

Outros pontos a se pensar:

- Não utilizar colunas que tenham uma grande quantidade de dados duplicados ou que tenham pouca variação, como a coluna *sexo*.
- O SGBD gasta recursos mantendo os índices sempre atualizados e associados.

A sintaxe é simples e exemplificada na Figura 26.

Figura 26 – Exemplo da sintaxe do comando *CREATE INDEX*

```
CREATE INDEX 'nome do índice'
ON 'nome tabela' (coluna);
```

Para exemplificar será utiliza a tabela *Funcionários* (Figura 27) que possui 5 colunas e a coluna *cod* como chave primária.

Figura 27 – Tabela utilizada nos próximos exemplos

cod	nome	sobrenome	sexo	salario
1	Augusto	Silva	M	4500.00
2	Paula	Peres	F	5300.00
3	Maria	Lima	F	3300.00
4	César	Rosa	M	4800.00

Na Figura 28 é criado um índice para a tabela *Funcionarios*. O nome do índice é importante, pois é possível utilizar um índice específico em uma pesquisa, para isso é necessário saber o seu nome.

Figura 28 – Exemplo do uso do comando *CREATE INDEX*

```
CREATE INDEX 'NomeFuncionario'
ON 'funcionarios' (nome);
```

Os índices podem ser compostos por uma única coluna ou por várias (multicoluna). O multicolunas pode consistir de até 15 colunas (em colunas *CHAR* e *VARCHAR* você também pode utilizar um prefixo da coluna como parte de um índice). Observe a Figura 29.

Figura 29 – Exemplo do uso do comando *CREATE INDEX*

```
CREATE INDEX 'NomeFuncionario'
ON 'funcionarios' (nome, sobrenome);
```

O índice *NomeFuncionario* é multicoluna, composto pelas colunas *nome* e *sobrenome*. Os índices também podem ser definidos na criação da tabela como exemplificado na Figura 30.

Figura 30 – Exemplo para criar um índice na criação da tabela

```
CREATE TABLE funcionarios
(
    cod INT(3) auto_increment,
    nome VARCHAR(30),
    sobrenome VARCHAR(30),
    sexo CHAR,
    salario DECIMAL(6,2),
    PRIMARY KEY (cod),
    INDEX NomeFuncionario (sobrenome)
);
```

Para um melhor desempenho da criação dos índices, é sugerido criá-los nas seguintes colunas:

- Chaves primárias;
- Chaves estrangeiras;
- Colunas acessadas por intervalos (*BETWEEN*);
- Campos utilizados em *GROUP BY* ou *ORDER BY*.

FINALIZANDO

Nesta aula foi apresentado o comando *JOIN*, que é utilizado para retornar dados de mais de uma tabela em uma mesma instrução *SELECT*. Foram exemplificados outros parâmetros que podem ser utilizados em uma consulta ao banco de dados (AVG, SUN, MAX, MIN e COUNT).

Para serem estudados os comandos da categoria DCL foi criado um usuário para o banco de dados e foram explicados os comandos para atribuir e retirar permissões dos usuários. Também foram estudados os comandos da categoria TCL, explicando e exemplificando como funcionam as transações.

Os índices foram o último assunto abordado na aula 5, sendo explicada a sua criação e a sua importância.

REFERÊNCIAS

ALVES, W. P. **Banco de dados.** 1. ed. São Paulo: Érica, 2014.

CAMARGO, W. B. de. Cláusulas INNER JOIN, LEFT JOIN e RIGHT JOIN no SQL Server. **DevMedia**. Disponível em: <https://www.devmedia.com.br/clausulas-inner-join-left-join-e-right-join-no-sql-server/18930>. Acesso em: 5 set. 2019.

DUARTE, E. Gerenciamento de usuários e controle de acessos do MySQL. **DevMedia**. Disponível em: <https://www.devmedia.com.br/gerenciamento-de-usuarios-e-controle-de-acessos-do-mysql/1898>. Acesso em: 5 set. 2019.

ENTENDENDO e usando índices – Parte 1. **DevMedia**. Disponível em: <https://www.devmedia.com.br/entendendo-e-usando-indices-parte-1/6567>. Acesso em: 5 set. 2019.

O QUE É *storage*? NAS, DAS, SAN ou FAS? **Controle Net**. Disponível em: <https://www.controle.net/faq/o-que-e-storage>. Acesso em: 5 set. 2019.

PUGA, S.; FRANÇA, E.; GOYA, M. **Banco de dados**: implementação em SQL, PL/SQL e Oracle 11g. São Paulo: Pearson, 2013.

SQL joins. **W3schools**. Disponível em: <https://www.w3schools.com/sql/sql_join.asp>. Acesso em: 5 set. 2019.