UniFil
EDUCAÇÃO A DISTÂNCIA

Unidade 2

Modelagem de Banco de Dados, Recursos avançados e alguns conceitos

Abertura

Olá, amigo(a) discente! Seja bem-vindo(a)!

Com a existência de uma grande demanda por desenvolvimento de softwares nas empresas, é necessário compreender a forma que é necessário armazenar e manipular os dados. Seguindo essa visão, nesta unidade vamos compreender e discutir sobre modelagem de banco de dados e alguns recursos existentes.

Na aula 01, vamos estudar os conceitos e práticas da modelagem de banco de dados e como esses conhecimentos podem auxiliar durante o desenvolvimento de softwares.

Na aula 02, vamos entrar em detalhes sobre o funcionamento dos recursos avançados de banco de dados e alguns conceitos que são importantes conhecer para aplicar um banco de dados forte e definido.

Objetivos

- Conhecer os conceitos e a aplicação da modelagem de banco de dados.
- Conhecer recursos avançados de banco de dados.

Conteúdo Programático

Aula 01 – Modelagem de Banco de Dados e conceitos

Aula 02 – Recursos avançados.

Modelagem de Banco de Dados

Olá, estudante, como vai? Vamos começar agora um estudo muito importante sobre banco de dados. Iniciaremos os estudos sobre como funciona a modelagem de um banco de dados e a sua importância.

A boa modelagem de um banco de dados pode modificar totalmente a experiência de construção de um software. Com a modelagem correta sendo aplicada as chances de entregar um software com qualidade e que gere valor para o usuário é muito maior.

Assim, nesta aula, vamos conhecer melhor como funciona a modelagem de banco de dados.

Videoaula 1

Utilize o QR Code para assistir!

Assista ao vídeo para entender melhor o assunto.

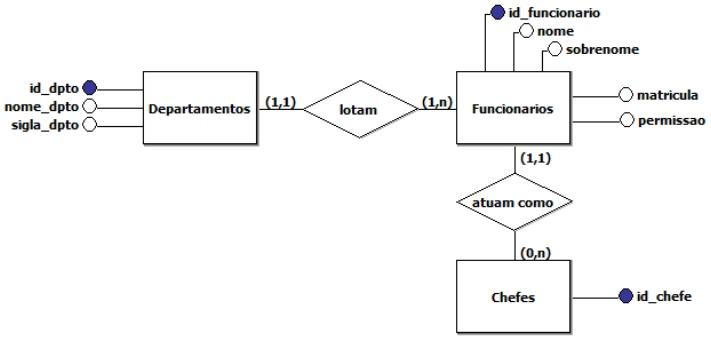
Ao se modelar um banco de dados, busca-se explorar as estruturas de dados e o conjunto de informações que serão necessárias para conseguir desenvolver um determinado sistema. É por isso que durante a etapa de modelagem de um banco de dados que se explora diversos modelos de modelagem de banco de dados.

Nesta etapa, as tabelas e seus relacionamentos são modelados como entidades e o relacionamento entre essas entidades, para dessa forma atribuir melhor conhecimento e funcionalidade de negócio dentro do banco de dados.

Modelo Conceitual

O modelo conceitual consiste em conhecer o negócio, rascunhar as entidades, porém ainda não se dá tanta preocupação sobre os relacionamentos de muitas entidades para muitas entidades. Nesta etapa é que se constrói o Modelo de Entidade e Relacionamento (MER).

Figura 1. MER



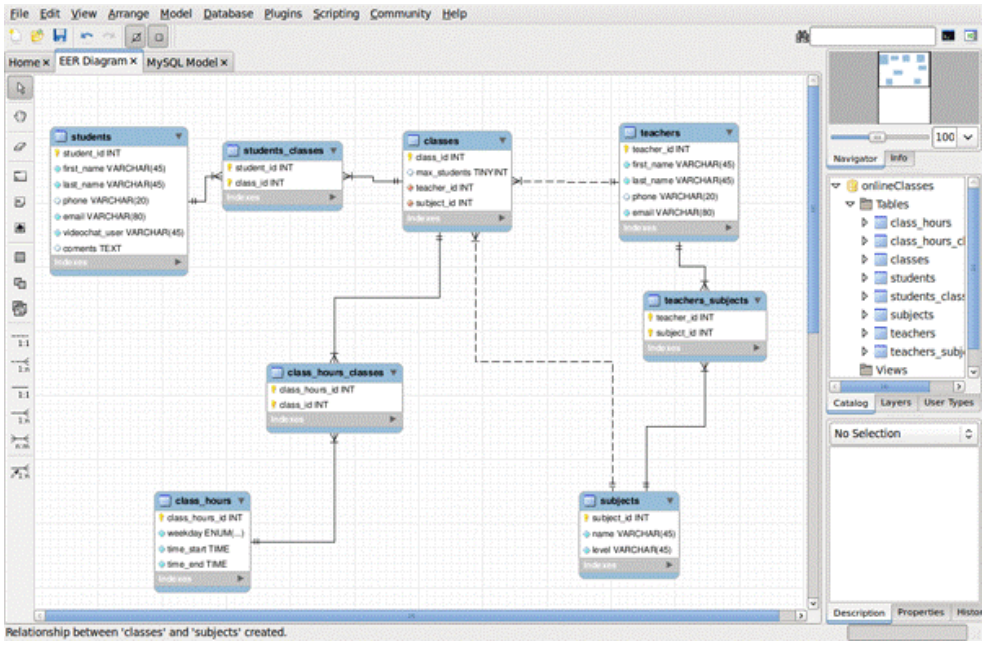
A figura 1 demonstra como é realizada a modelagem conceitual de um banco de dados em um modelo de entidade e relacionamento, onde cada caixa é uma entidade e, claro, demonstrando também os seus relacionamentos entre as entidades, com seus atributos.

Modelo Lógico

Na modelagem lógica é que documenta a estrutura do banco de dados, para isso busca-se possuir uma visão de negócio sobre o armazenamento dos dados necessários, aqui é aplicado a normalização das tabelas, definição das chaves e dos relacionamentos.

Vale ressaltar que no caso de tabelas com relacionamento múltiplo é criado uma tabela auxiliar para tratar desse relacionamento.

Figura 2. DER com relacionamento múltiplo.



Observando a figura 2 se pode analisar um Diagrama de Entidade e Relacionamento (DER) onde existem relacionamentos múltiplos, ou seja, uma entidade pode estar presente mais de uma vez em outra entidade, dessa forma, gerando entidades auxiliares, como a ‘students_classes’.

Videoaula 2

Utilize o QR Code para assistir!

Assista ao vídeo para entender melhor o assunto.

Modelo Físico

No modelo físico de banco de dados é quando é necessário tomar conhecimento das limitações do banco de dados escolhidos, considerar os requisitos de sistemas para o banco em questão e assim criar fisicamente o banco de dados com suas: tabelas, colunas, chaves e relacionamentos.

É nessa modelagem que se constituem os scripts em SQL de criação do banco de dados e suas tabelas.

A boa aplicação de uma modelagem de banco de dados favorece o desenvolvimento de um sistema, além de manter os dados organizados e bem armazenados para quando for necessário utilizá-los.

Para melhor aplicar os conhecimentos em banco de dados relacionais é preciso se atentar para com alguns conceitos e recursos. Sendo eles:

Transação

A transação é um termo que se relata como uma coleção de operações que juntas são uma unidade lógica de trabalho, dessa maneira, uma tarefa única e lógica.

Quando é realizada uma transação bancária, o dinheiro precisa sair de uma conta e ir para outra conta, essa operação é denominada na linguagem de banco de dados como transação.

Figura 3. Transação Exemplo Postgres.

```
BEGIN;
UPDATE accounts SET balance = balance - 100.00
    WHERE name = 'Alice';
SAVEPOINT my_savepoint;
UPDATE accounts SET balance = balance + 100.00
    WHERE name = 'Bob';
-- oops ... forget that and use Wally's account
ROLLBACK TO my_savepoint;
UPDATE accounts SET balance = balance + 100.00
    WHERE name = 'Wally';
COMMIT;
```

Fonte: PostgreSql.Org, 2021.

A figura 3 demonstra um exemplo de como é construído um script de transação dentro de um banco de dados relacional Postgresql, a transação relata a transferência entre contas bancárias.

Ao final da transação ele deve ser submetido para o banco de dados, isso ocorre, pois existe a possibilidade de concorrência entre os usuários. A concorrência pode acarretar problemas, porque pode ocorrer de atualizar o mesmo item no banco de dados, o que irá gerar uma falha na transação ou em outras operações do banco de dados.

Figura 3.1. Transação Exemplo Postgres, destacando operações

```
BEGIN;
UPDATE accounts SET balance = balance - 100.00
    WHERE name = 'Alice';
SAVEPOINT my_savepoint;
UPDATE accounts SET balance = balance + 100.00
    WHERE name = 'Bob';
-- oops ... forget that and use Wally's account
ROLLBACK TO my_savepoint;
UPDATE accounts SET balance = balance + 100.00
    WHERE name = 'Wally';
COMMIT;
```

Fonte: PostgreSql.Org, 2021.

Na figura 3.1, se demonstra uma transação e dentro de cada transação podem existir uma ou mais operações e em vermelho, na figura em questão, estão destacadas as operações que serão realizadas ao executar o script dessa transação.

É preciso se atentar a alguns pontos dentro de uma transação no banco de dados relacional, são eles:

- As operações foram realizadas corretamente.
- As alterações executadas foram para o banco de dados e gravadas

Uma transação pode possuir três estados diferentes:

- Transaction Begin: estado inicial da transição
- Transaction Commit: após a execução com sucesso, a transação é submetida ao banco de dados e assim as operações são gravadas permanentemente no BD.
- Transaction Rollback: ocorreu um erro com a transação, podendo assim retornar os dados ao estado anterior a transação.

Por esses motivos, é necessário sempre se atentar às operações que serão executadas dentro de uma transação, pois isso, pode acabar confundindo e assim quebrando a integridade dos dados armazenados.

Sempre que rodar em um banco de dados uma transação, é preciso verificar se ela foi executada com sucesso e se os dados foram alterados de estado da forma que era esperada.

Caso ocorra uma falha na transação, dependendo das operações realizadas, pode ocorrer que uma transação foi realizada pela metade. Isso ocorre quando a transação falha entre uma operação e outra e não acontece o *rollback* direto de toda a transação.

Para auxiliar a tratar essas situações, alguns SGBDs já possuem aspectos implementados para facilitar a execução de transações e cuidar de possíveis falhas, o versionamento das transações e operações juntamente com os logs de verificação e validação ajudam diariamente desenvolvedores a solucionarem problemas com transações.

Concorrência

A concorrência ocorre quando mais de um usuário executa operações na mesma tabela dentro do mesmo período. Sendo assim, é preciso analisar o seguinte cenário:

Tabela 1. Transações concorrentes

TRANSAÇÃO 1	TRANSAÇÃO 2
SELECT – Tabela Alunos	
|	

| | SELECT – Tabela Alunos
| | |
UPDATE – Tabela Alunos | |
| | |
| UPDATE – Tabela Alunos
| | |
| | |
FIM TRANSAÇÃO | |
| |
FIM TRANSAÇÃO

Fonte: O autor, 2021.

A tabela 1 demonstra a concorrência entre transações no banco de dados relacional, no caso da transação 1 ela foi perdida, pois logo após ela realizar o update no banco de dados Alunos a transação 2 realizou a mesma alteração.

A concorrência é um grande problema quando não tratado ou até mesmo reconhecido pelos desenvolvedores ou gestores de bancos de dados.

Uma possível correção é fechar uma transação e deixar com que ela seja isolada de outras transações, assim, não danificando o armazenamento dos dados e a sua integridade.

Com isso, para que as transações não se quebrem e falhem existem alguns princípios e recursos que podem ser utilizados são eles:

- Escalonamento
- Timbre de hora
- Bloqueio
- Técnica otimista (entendem que operações conflitantes são exceção)

O recurso mais utilizado é o bloqueio, que pode ocorrer nas transações e nas operações a fim de manter sempre a integridade e maturidade do banco de dados.

Tipos de bloqueios:

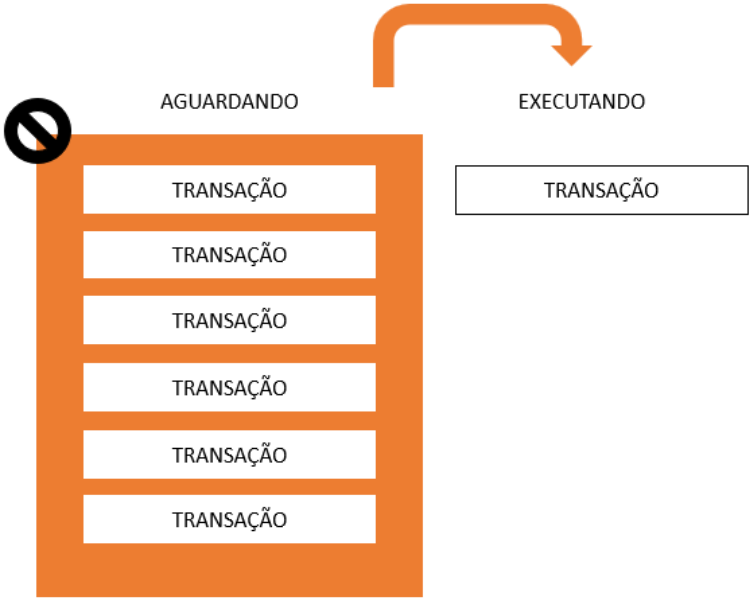
- XLOCK: serve para registros e é utilizado para atualizações.
- SLOCK: é compartilhado sobre os registros e é utilizado para consultas.

Os bloqueios podem ocorrer em diversos pontos de um banco de dados: BD inteiro, registro, tabela, disco.

Cada SGBD tem em sua definição qual o melhor bloqueio a ser utilizado e o nível desse bloqueio, em alguns casos, é possível configurar também o nível que cada bloqueio irá ocorrer.

Durante esse bloqueio as outras transações são escalonadas, desse modo sendo adicionadas em um Buffer onde cada transação aguarda a próxima ser executada, porém essa espécie de fila também pode resultar em alguns problemas.

Figura 4. Bloqueio de transações com Buffer

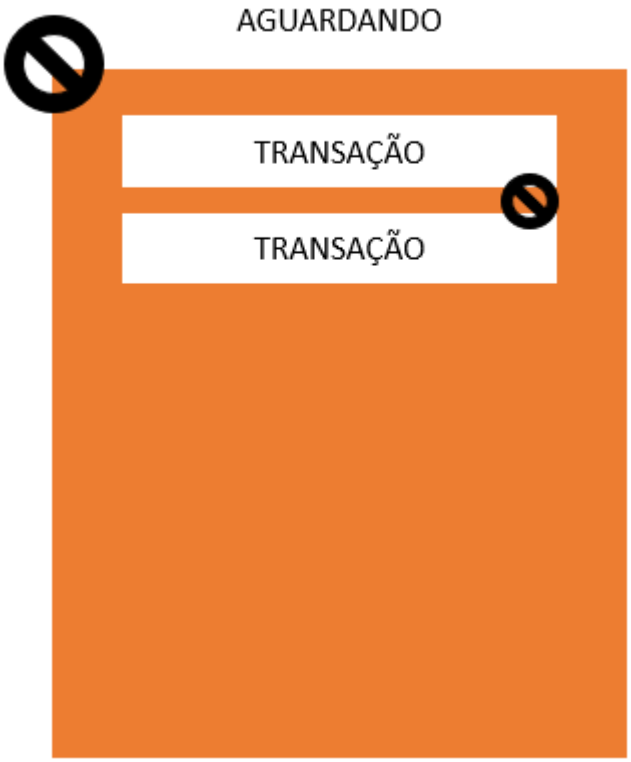


Fonte: O autor, 2021.

Observando a figura 4 pode-se analisar graficamente a estrutura de *Buffer* para o controle e bloqueio de transações, onde na esquerda diversas transações estão armazenadas em memória e esperando para serem executadas, e só serão executadas quando a transação que está sendo executada a direita liberar para a execução.

Porém, em alguns casos, pode ocorrer que duas transações fiquem bloqueadas uma esperando a outra, situações como essa podem acabar acontecendo e são denominadas como Deadlock.

Figura 5. Bloqueio Deadlock



Fonte: O autor, 2021.

A figura 5 demonstra graficamente um bloqueio de Deadlock, onde é possível processar uma das transações, porém como uma depende da outra o banco de dados está com as 2 transações travadas.

Quando um SGBD identifica essas situações a sua decisão deve sempre ser quebrar o Deadlock para que não fique desse modo acumulando transações. Na maioria dos casos, o Sistema Gerenciador do Banco de Dados decide matar uma das transações para que assim a outra possa ser executada com sucesso.

Videoaula 3

Utilize o QR Code para assistir!

Assista ao vídeo para entender melhor o assunto.

Recursos avançados

Olá, aluno, tudo bem? Nesta aula vamos conhecer melhor alguns *scripts* de banco de dados relacionais, vamos descobrir juntos seus princípios e como a aplicação desses comandos com os conhecimentos de transações e concorrência podem melhorar o desempenho de um banco de dados.

Durante a construção de um software pode-se fazer necessário atribuir alguns recursos dentro do próprio banco de dados para desse modo facilitar o funcionamento do sistema que está sendo desenvolvido.

Com isto conheceremos alguns recursos de banco de dados como: Trigger, View, Stored Procedure e o funcionamento dos Joins.

Cada banco de dados possui uma sintaxe para a construção e execução desses recursos, porém todos utilizam do SQL como estrutura principal. Para os exemplos demonstrados nesta disciplina será utilizado o banco de dados PostgreSQL.

Videoaula 1

Utilize o QR Code para assistir!

Assista ao vídeo para entender melhor o assunto.

Trigger

A Trigger é um recurso armazenado dentro do próprio SQL, como o nome diz, é uma espécie de gatilho que é definido para ser disparado quando ocorre algum evento no banco de dados.

Em sua maioria, esses eventos são comandos DML onde quando ocorre um INSERT se deve atribuir um novo valor para um cache ou outra operação deve ser realizada no banco de dados. Essas Triggers podem ser programadas e definidas para ocorrerem antes ou depois dos eventos.

Figura 6. Exemplo Trigger PostreSQL

```
CREATE TABLE emp (
    empname text,
    salary integer,
    last_date timestamp,
    last_user text
);

CREATE FUNCTION emp_stamp() RETURNS trigger AS $emp_stamp$
    BEGIN
        -- Check that empname and salary are given
        IF NEW.empname IS NULL THEN
            RAISE EXCEPTION 'empname cannot be null';
        END IF;
        IF NEW.salary IS NULL THEN
            RAISE EXCEPTION '% cannot have null salary', NEW.empname;
        END IF;

        -- Who works for us when she must pay for it?
        IF NEW.salary < 0 THEN
            RAISE EXCEPTION '% cannot have a negative salary', NEW.empname;
        END IF;

        -- Remember who changed the payroll when
        NEW.last_date := current_timestamp;
        NEW.last_user := current_user;
        RETURN NEW;
    END;
$emp_stamp$ LANGUAGE plpgsql;

CREATE TRIGGER emp_stamp BEFORE INSERT OR UPDATE ON emp
    FOR EACH ROW EXECUTE PROCEDURE emp_stamp();
```

Fonte: PostgreSQL.Org, 2021.

Na figura 6 observa-se uma Trigger construída em um banco de dados PostgreSQL, onde é criada a tabela **emp** e na sequência é criada a Trigger **emp_stamp** que valida as informações e caso a validação ocorra com sucesso é atribuída a data e o usuário que realizou a operação nos campos **last_date** e **last_user** da tabela emp.

Ao final da figura, também pode-se analisar que a Trigger deve ser executada antes de qualquer comando **INSERT** e **UPDATE** na tabela **emp**.

O funcionamento de uma Trigger é descrito dessa forma, lembrando que isso ocorre dentro de uma configuração do banco de dados e não no sistema que irá utilizá-lo.

Alguns SGBDs disponibilizam recursos para construção de triggers de modo gráfico, assim facilitando a compreensão para pessoas que têm dificuldades nas construções deste recurso.

Views

Uma View é uma estrutura de tabela virtual que é resultado de uma consulta SQL, sendo assim construída por intermédio de um **SELECT** com seus **WHERE** e **JOINs**.

Quando existe a necessidade de acessar dados de forma rápida e que estão em tabelas distintas ao qual sempre executar um JOIN pode acarretar o desempenho do BD, a proposta é criar uma **view** para assim facilitar e não perder a performance do banco de dados.

Figura 7. Exemplo View PostreSQL

```
CREATE VIEW comedies AS
    SELECT *
    FROM films
    WHERE kind = 'Comedy';
```

Fonte: PostgreSQL.Org, 2021.

A figura 7 demonstra como funciona a criação de uma view em um banco de dados, onde a **view** se chama **comedies** e nela se encontram os dados da tabela **films** onde o **kind** (tipo) for **'Comedy'**.

Então, na **view** comedies irão se encontrar todos os atributos dos filmes que são do tipo comédia. A view fica armazenada dentro do banco de dados e sempre retorna os dados mais atualizados, pois sempre recria os dados toda vez que o usuário consulta a view.

Este é um exemplo claro e simples de uma view, porém pode-se construir views com muito mais dados e recursos, tudo depende da forma que o SELECT é construído e qual a necessidade de disponibilização dos dados.

Porque a construção de uma view está totalmente ligada a disponibilização de dados para o consumidor do banco de dados.

Esta é uma view simples, onde as operações que são realizadas são somente **INSERT**, porém em alguns casos é preciso realizar outras operações dentro de uma procedure, como **subquerys**, **JOIN**, e até mesmo **UPDATE** ou **DELETE** de alguns dados.

Stored Procedure

É um recurso nativo do banco de dados que permite armazenar no servidor um conjunto de instruções e operações para serem realizadas.

Figura 8. Exemplo Procedure PostreSQL

```
CREATE PROCEDURE insert_data(a integer, b integer)
LANGUAGE SQL
AS $$
INSERT INTO tbl VALUES (a);
INSERT INTO tbl VALUES (b);
$$;

CALL insert_data(1, 2);
```

Fonte: PostgreSQL.Org, 2021.

Observando a figura 8 pode-se analisar a criação de uma procedure **insert**_data, onde esta recebe dois valores inteiros **a** e **b** e então adiciona os valores a e b na tabela **tbl**. Logo em seguida, é chamada a procedure passando os valores utilizando o comando **CALL**.

Videoaula 2

Utilize o QR Code para assistir!

Assista ao vídeo para entender melhor o assunto.

JOIN

O JOIN é uma cláusula no SQL, serve para unir duas tabelas durante uma consulta, sendo assim utilizado para obter uma consulta em conjunto de outras tabelas. Quando a quantidade de dados é muito massiva, utilizar do JOIN pode resultar em perda no desempenho do banco de dados, além de manter tabelas ou até mesmo o BD bloqueado por um tempo elevado devido ao risco da concorrência.

Em um JOIN, para facilitar a compreensão, é preciso analisar que cada tabela está em uma ponta da execução, sendo assim, cada JOIN tem uma tabela à direita e outra à esquerda e isso fica definido de acordo com o comando escrito.

Existem vários tipos de JOIN sendo eles:

INNER JOIN

É o comando mais comum entre os desenvolvedores e administradores de banco de dados, ele retorna o que é comum entre 2 tabelas.

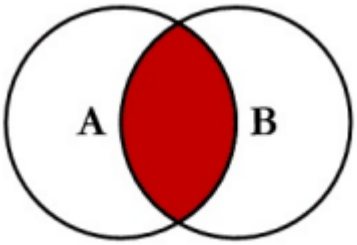
Figura 9. Exemplo INNER JOIN PostreSQL.

```
SELECT *
    FROM weather INNER JOIN cities ON (weather.city = cities.name);
```

Fonte: PostgreSQL.Org, 2021.

A figura 9 relata como é construído um **INNER JOIN** entre as tabelas **weather** e **cities**, onde deve retornar os dados comuns entre as duas tabelas onde o **weather.city** for igual ao **cities.name**. Sendo no exemplo a tabela **weather** à esquerda e **cities** à direita.

Figura 10. INNER JOIN.



Fonte: Devmedia, 2021.

Na figura 10, analisa-se como é o funcionamento de um INNER JOIN, sendo cada círculo uma tabela, o resultado da consulta é somente o que as duas tabelas possuem em comum que na figura estão demonstradas em vermelho claro no meio da imagem.

LEFT JOIN

O LEFT JOIN é utilizado quando é necessário que o retorno possua todos os dados da tabela à direita juntamente com os dados em comum da tabela à esquerda.

Figura 11. Exemplo LEFT JOIN.

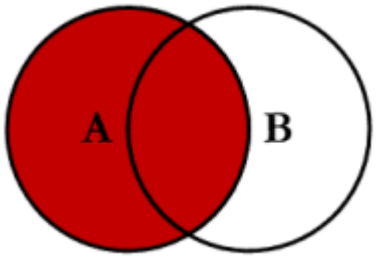
```
1 SELECT
2 *
3 FROM
4     filme
5 LEFT JOIN produtora
6     ON produtora.id = filme.produtora_id;
```

Fonte: O autor, 2021.

Na figura 11, observa-se a construção de um LEFT JOIN onde irá retornar todos os dados presentes na tabela filme, pois está à esquerda e todos os dados em comum com a tabela

produtora que está à direita. Podendo retornar dados que não tem vínculo nenhum com a tabela produtora.

Figura 12. LEFT JOIN.



Fonte: Devmedia, 2021.

A figura 12 demonstra de forma gráfica como ocorre o retorno de uma LEFT JOIN, onde todos os dados da tabela A irão retornar estando estes presentes na tabela B ou não.

RIGHT JOIN

O RIGHT JOIN é exatamente o inverso do LEFT como o próprio nome condiz, nesta consulta são retornados todos os dados presentes na tabela à direita e os dados em comum na tabela à esquerda.

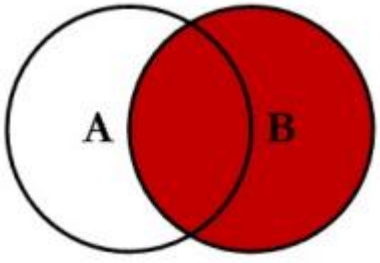
Figura 13. Exemplo RIGHT JOIN.

```
1 SELECT
2 *
3 FROM
4     filme
5 RIGHT JOIN produtora
6     ON produtora.id = filme.produtora_id;
```

Fonte: O autor, 2021.

Na figura 13, observa-se script de um **RIGHT JOIN** onde irão retornar todos os dados da tabela **produtora** e os dados comuns com a tabela **filme**.

Figura 14. RIGHT JOIN.



Fonte: Devmedia, 2021.

Observando a figura 14, constata-se como funciona o retorno de um RIGHT JOIN onde todos os dados da tabela B irão retornar estando estes em comum com a tabela A ou não.

FULL JOIN

Nessa situação, utilizando este comando é retornado todos os dados presentes nas duas tabelas do JOIN sendo seus dados em comum ou não uma com a outra.

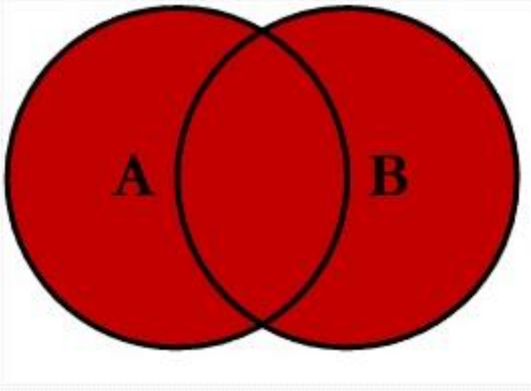
Figura 15. Exemplo FULL JOIN.

```
1 SELECT
2 *
3 FROM
4     filme
5 FULL JOIN produtora
6     ON produtora.id = filme.produtora_id;
```

Fonte: O autor, 2021.

No FULL JOIN tanto os dados comuns quanto todos os outros dados da tabela à direita e à esquerda serão retornados na consulta.

Figura 16. FULL JOIN.



Fonte: Devmedia, 2021.

Observando a figura 16, analisa-se como irá ocorrer o retorno de dados no caso de um FULL JOIN onde todos os dados são retornados. Vale lembrar que na utilização de LEF, RIGHT e FULL JOIN podem ocorrer de alguns atributos/colunas retornarem com valor NULL devido existir valores em uma das tabelas e não existir na outra tabela.

Os JOINs facilitam a recuperação de dados e até mesmo na validação de informações e consistência das tabelas, pois realizam a junção dos dados de forma clara e de fácil interpretação.

Porém, como a utilização pode impactar de forma negativa o banco de dados, em sua maioria, as informações necessárias são sempre especificadas no comando SELECT.

Figura 17. Exemplo INNER JOIN com filtro SELECT.

```
SELECT
filme.produtora_id,
filme.nome,
filme.data_lancamento,
produtora.id
FROM
    filme
LEFT JOIN produtora
    ON produtora.id = filme.produtora_id;
```

Fonte: O autor, 2021.

Pode-se observar na figura 17 como é o funcionamento de um SELECT com INNER JOIN onde irá retornar os dados especificados dentro do SELECT, que são eles:

- filme.produtora_id
- filme.nome
- filme.data_lancamento
- produtora.id

Isso faz com que o desempenho do BD seja maior, pois não irá buscar todas as informações para retorno, somente as especificadas. Essa boa prática pode ser aplicada em qualquer SELECT, quando a quantidade de dados é muito grande e seguir práticas como essa são extremamente importantes para o bom funcionamento do banco de dados.

Videoaula 3

Utilize o QR Code para assistir!

Assista ao vídeo para entender melhor o assunto.

Fórum Avaliativo

Prezado(a) aluno(a)!
Agora convido você para realizar a primeira atividade avaliativa da disciplina: **Fórum de Discussões**.

Para participar, você deverá clicar em “**Discussões**” no “**Menu Lateral**” e acessar.
Lembre-se: Após a data estipulada o Fórum será encerrado e não será mais permitido participar.

Sua contribuição é muito importante. Bons estudos!

Encerramento

Chegamos ao final da nossa segunda unidade, onde você teve contato com os conceitos de modelagem de banco de dados e alguns recursos dos bancos de dados relacionais. Com certeza, muito do que foi mostrado aqui você já tinha uma noção.

No mundo dos softwares, há muitas formas de tratar e armazenar os dados que podem ser melhores ou piores, no final tudo depende da demanda e do produto que deve ser entregue. Na aula 1, visualizamos alguns conceitos de modelagem de banco de dados e noções sobre transações e concorrência.

Na aula 2, foi discutido e apresentado alguns recursos presentes nos bancos de dados relacionais além de práticas essas que podem auxiliar na construção e vida de um banco de dados.

Juntando as duas aulas foi possível se aproximar mais dos bancos de dados relacionais e seus conceitos e características marcantes, além dos recursos nativos que podem ser utilizados. Que você possa aplicar esses conhecimentos obtidos em seu cotidiano. Até a próxima.

Referências

HEUSER, Carlos Alberto. **Projeto de banco de dados: Volume 4 da Série Livros didáticos informática UFRGS**. Bookman Editora, 2009.

Dataversity. What Is a Wide Column Database. Disponível em: < https://www.dataversity.net/wide-column-database/>. Acesso em 18 de dez. de 2021.


UniFil
EDUCAÇÃO A DISTÂNCIA
UNIFIL.BR