BANCO DE DADOS

O banco de dados é o responsável por armazenar (persistir) os dados das aplicações em seu sistema para posterior consulta e emissão de relatórios. Um gerenciamento de banco de dados eficaz consiste em técnicas e rotinas planejadas para inserção e armazenamento de informações, tanto em banco de dados móveis quanto em banco de dados de aplicativos *desktops*.

Este conhecimento tem o objetivo de ser prático, para que você seja capaz de estruturar e implementar de fato o banco de dados para aplicações *web*, construindo todos os *scripts* SQL necessários para desenvolvimento e finalização.

SISTEMA GERENCIADOR DE BANCO DE DADOS PARA *WEB*

Várias instruções e *scripts* SQL são necessários para que a aplicação obtenha retorno satisfatório do sistema gerenciador do banco de dados para perfeita harmonia entre os dados do banco de dados e o *software* aplicativo.

Muitos conceitos em banco de dados foram apresentados no conhecimento **Modelagem de banco de dados: componentes de sistemas de bancos de dados, modelagem conceitual, modelo relacional**, desta unidade curricular (UC), por isso agora vamos aprofundar os conhecimentos focando em consultas, criações e alterações de *scripts* SQL.

TABELA, RELACIONAMENTO, CONSULTAS E CONSULTAS SIMPLIFICADAS

Vamos começar a criação do banco de dados e das tabelas e fazer o relacionamento (junção de tabelas), mostrando os diversos tipos de consultas (**select**) no banco de dados.

MANIPULAÇÃO DE ARQUIVOS

Criação (create)

Com o comando **create**, podemos desenvolver o banco de dados e as tabelas, conforme sintaxe apresentada a seguir:

Banco de dados:

```
create database seubancodedados;
```

Tabelas com chave primária:

```
CREATE TABLE suatabela(
   codigo int(4) integer PRIMARY KEY AUTO_INCREMENT,
   nome varchar(30) NOT NULL,
   email varchar(50));
```

O comando **AUTO_INCREMENT** pode ser utilizado para automatizar um código que sirva de chave primária para uma tabela.

PRIMARY KEY define a chave primária da tabela, isto é, o campo que serve como chave da tabela e que não pode ser repetido.

NOT NULL, por sua vez, define que um determinado campo seja de preenchimento obrigatório.

Tabelas com chave estrangeira:

```
 CREATE TABLE suatabela(
    codigo int(4) integer PRIMARY KEY AUTO_INCREMENT,
    nome varchar(30) NOT NULL,
    email varchar(50),
    PRIMARY KEY (codigo),
    ID_suatabeladois integer,
    CONSTRAINT      fk_suaTabelaDois      FOREIGN      KEY
(ID_suatabeladois)         REFERENCES         suaTabelaDois
(ID_suaTabelaDois) );
```

A última instrução é a chave estrangeira propriamente dita. Para isso, observe que adicionamos uma *constraint* chamada de **fk_ suaTabelaDois** (nome padrão: misto dos nomes das tabelas relacionadas com o prefixo **fk**) como uma *foreign key* (chave estrangeira) e associamos essa *foreign key* ao campo **ID_ suaTabelaDois** da tabela **suaTabelaDois**. Ainda na mesma linha, definimos a referência propriamente dita (palavra-chave **references**) à tabela **suaTabelaDois**, especificamente ao campo **ID_ suaTabelaDois**.

Atualizar (update)

Com o comando de *update*, atualizamos as tabelas e seus respectivos conteúdos:

```
UPDATE suatabela SET Nomecoluna = novoValorColuna WHERE
identificador = identificador;
```

No comando **UPDATE**, precisamos sempre identificar qual é o registro que queremos editar por meio da cláusula **WHERE** seguida pelo identificador do registro, pois assim se evita que todos os registros sejam editados.

Alteração (alter)

Como os sistemas sofrem mudanças diariamente, também precisamos modificar a estrutura de nossas tabelas. Esta alteração é possível por meio do comando **alter**, como é possível observar a seguir:

```
ALTER TABLE 'suaTabela' ADD COLUMN 'novaColuna'
'tipo_dado';
```

ALTER TABLE 'suaTabela' é o comando que informa ao servidor MySQL para modificar a tabela denominada **'suaTabela'**; e **ADD COLUMN 'novaColuna' 'tipo dado'** é o comando que informa ao servidor MySQL para adicionar uma nova coluna denominada **'novaColuna'** com o tipo de dados **'tipo dado'**.

Seleção (select)

O comando de seleção é usado para buscar os dados no banco de dados e apresentar ao usuário em forma de relatórios ou na aplicação.

Existem vários comandos que podem ser agregados ao **select** para aperfeiçoar a consulta.

```
select * from suaTabela;
```

O asterisco indica que serão selecionados todos os campos da tabela.

```
select id, nome from suaTabela;
```

Neste comando, estão selecionados apenas os campos **id** e o **nome** da tabela **suaTabela**.

Enquanto:

```
select * from suaTabela WHERE id >5;
```

Este comando seleciona todos os campos da tabela enquanto o id do registro for maior do que 5.

Na cláusula **where**, deve-se usar os operadores de comparação (> [maior], < [menor], = [igual] , <> [diferente] , >= [maior igual] ou <= [menor igual]).

Ordenação:

```
select id, nome
from suaTabela order by nome;
```

Neste comando, são selecionados os campos **id** e **nome**, com orientação para ordenação em ordem alfabética pelo campo nome.

```
select id, nome
from suaTabela order by id DESC;
```

Pode-se selecionar por outros campos. Se estes outros campos forem do tipo inteiro, será selecionado do menor registro para o maior registro; se precisarmos selecionar na ordem inversa, é possível acrescentar o comando **DESC**.

Agrupamento:

```
Select id, nome
from suaTabela group by nome
```

O parâmetro **group by** é usado para agrupar todos os clientes que têm o mesmo nome. Este parâmetro é muito utilizado em relatórios quando é necessário fazer o agrupamento de dados, como para saber quantos itens iguais forma vendidos em determinada venda.

Soma (SUM)

A função **SUM** faz a soma de colunas da tabela e por meio dela pode-se fazer a soma de todas as linhas de uma coluna. A sintaxe da função SUM é a seguinte:

```
select SUM(nome_coluna)
from suaTabela
```

Junção de tabelas:

```
select a.idtabela1, a.nome, b.quantidade
from suatabelaum a inner join suatabeladois b
        on (a.idtabelaum = b.idtabeladois)
```

O comando **join** é o responsável pela junção de tabelas, em que criamos alias para identificar as tabelas (a, b). Após a cláusula **on**, devemos sempre identificar os campos comuns nas tabelas unidas.

Subconsultas SQL

As subconsultas SQL são utilizadas para fazer a seleção de itens de mais de uma tabela no banco de dados sem a necessidade de criação de tabelas auxiliares, ou mesmo a consulta otimizada em apenas uma tabela. Sua construção e sintaxe são simples e por isso são muito utilizadas nos *scripts* SQL.
Subconsultas em uma cláusula **WHERE**:

```
SELECT nome
FROM 'suatabela'
WHERE idSuaTabela = (SELECT idSuaTabela FROM suatabela
WHERE sobrenome = 'suaCondição')
```

Estamos selecionando o nome da tabela **suaTabela** enquanto o **idSuaTabela** é igual a outra consulta SQL. Sendo assim, a *script* irá, primeiramente, rodar o último SQL a fim de satisfazer a condição **where**. Nesta consulta, também podemos utilizar todos os operadores de comparação, conforme mencionado no tópico **Enquanto**.