

**AULA 03: SQL**

SUMÁRIO	PÁGINA
1. Introdução	1
2. Projeto Físico e DDL	2
2.1. Create	5
2.2. Alter	10
2.3. Drop	11
2.4. Objetos do Banco de Dados	13
3. DML	25
3.1. Insert	25
3.2. Delete	27
3.3. Update	29
3.4. Select	30
3.4.1. In, Between, Like	33
3.4.2. Order By	38
3.4.3. Funções	38
3.4.4. Agrupadores	43
4. Subqueries	54
5. Produto Cartesiano	56
6. Resumo	61
7. Lista das Questões Apresentadas	62
8. Gabaritos	71

**1. INTRODUÇÃO**

Saudações caros(as) amigos(as),

Hoje vamos à nossa quarta aula de **Conhecimentos de Banco de Dados**, tratando principalmente do tema SQL. Na minha opinião é o mais importante dos assuntos cobrados. Digo isso porque acredito que o fato da CGU ter decidido exigir essa matéria é para que seus futuros AFCs dominem o SQL. Com o conhecimento de SQL é possível, entre outras coisas, extrair informações dos Bancos de Dados, de forma a torná-los uma ferramenta útil para a realização de auditorias, inspeções etc.

Se eu fosse contratado para elaborar as questões de Conhecimentos de Banco de Dados para esse concurso, metade delas seriam de SQL. Mas

como não sou eu que vou elaborar a prova, e sabemos que cabeça de examinador ninguém entende, pode ser que tenhamos poucas questões de SQL. Menos do que três, acho muito difícil, mas a ESAF apronta dessas às vezes, faz você estudar um monte de assuntos e na hora da prova não cobra. Bem, mas continuo com a minha “aposta”, pelo menos três questões saem do assunto SQL.

Nossa aula de hoje vai contemplar o projeto físico, e vamos mergulhar de vez no SQL. Se puderem praticar no SGBD que sugeri que instalassem, ótimo, é bom ver a coisa toda funcionando na prática. Se optarem por não executar, tudo bem, acredito que isso não vai impedir o entendimento e a possibilidade de responder as questões da prova.

Por fim, tenho uma notícia para vocês. Modelagem ainda não acabou, por incrível que pareça. Ainda tem um assunto, meio “remoso”, que deixei para o final. Espero até que eles não cobrem esse assunto, mas vamos ver por precaução.

Sem mais delongas, vamos ao trabalho.

## **2. PROJETO FÍSICO E DDL**

Terminamos a aula passada com o nosso Modelo Relacional pronto. Gostaria de esclarecer que aqueles passos do mapeamento do MER para o Modelo Relacional não são regras absolutas, mas são largamente utilizados. O mais importante que vocês devem guardar é que quando temos um relacionamento N:M deve entrar uma nova relação no meio, criando assim duas relações 1:N, e que as chaves estrangeiras devem ser transportadas de uma relação para a outra sempre que temos relacionamento 1:N.

Bem, para construir o nosso Banco de Dados, temos agora que partir para o Projeto Físico. Nessa fase final simplesmente pegamos nosso Modelo Relacional e criamos, através de uma linguagem específica,

comandos que dizem ao SGBD como será o nosso Banco de Dados. Essa linguagem específica é o SQL.

Structured Query Language (SQL), ou Linguagem de Consulta Estruturada, é uma linguagem de pesquisa declarativa para Banco de Dados relacionais. Ela tem diversos propósitos, sendo os dois principais servir como Linguagem de Definição de Dados (DDL), ou seja, uma linguagem que serve para informar ao SGBD qual a estrutura do meu Banco de Dados, dando uma descrição completa dos meus metadados, e também serve como Linguagem de Manipulação de Dados (DML), ou seja, uma linguagem que diz ao SGBD para gravar, alterar, excluir ou atualizar os dados propriamente ditos no meu Banco de Dados.

É importante que vocês saibam que existe um SQL padrão, conhecido como ANSI SQL. Em uma perspectiva ideal, todos os SGBDs do mercado implementariam o ANSI SQL. Mas na prática não é isso que acontece. Cada SGBD implementa um SQL um pouco modificado, ou seja, um SQL que varia um pouco da implementação do ANSI SQL. Mas afinal, qual SQL a ESAF vai cobrar, o ANSI ou aquele implementado em um SGBD específico? Bem, pelas provas anteriores, tenho visto que as bancas em geral cobram aquilo que é comum, ou seja, que a maioria dos SGBDs implementa. Vou mostrar o SQL que funciona no SGBD MySQL, e se tiver algum pedaço que só sirva para esse SGBD específico, eu informo.

Vamos colocar isso na prática? Bem, para começar, vamos lembrar do nosso Modelo Relacional.

**DVD (IdDVD, tituloPort, tituloIng, tipo, gênero, ano)**

**Amigo (CPE, nome, endereço, tel1, tel2, tel3)**

**Empréstimo (IdEmprest, CPF, IdDVD, dataEmprest, dataPrev, dataDevol)**

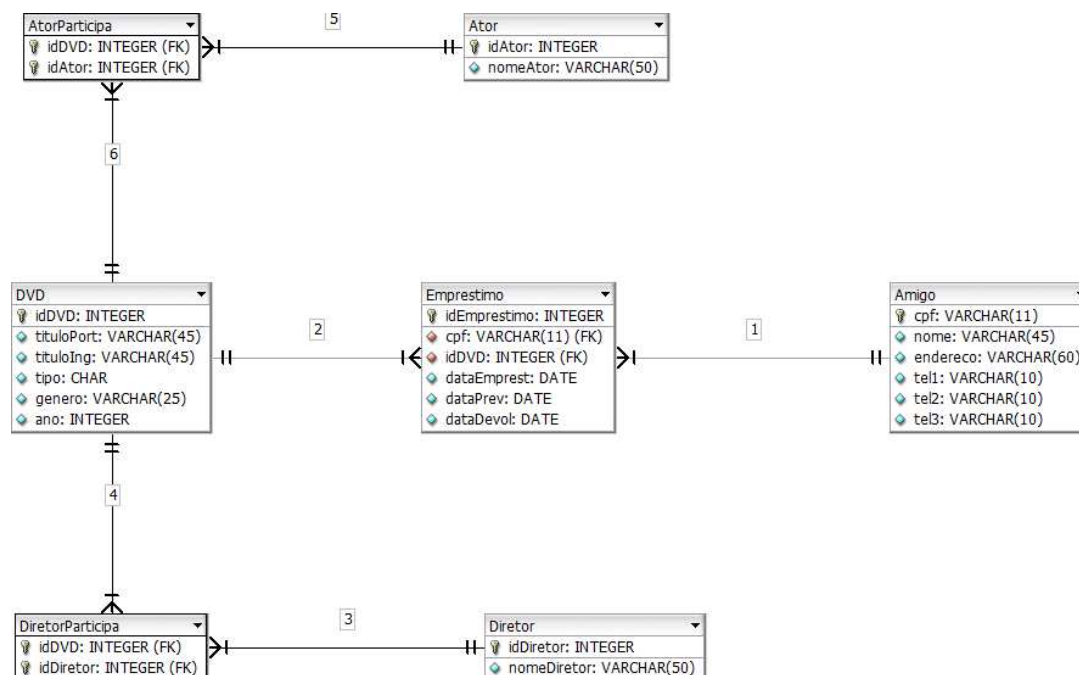
**Ator (IdAtor, nomeAtor)**

**Diretor (IdDiretor, nomeDiretor)**

**AtorParticipa (IdDVD, IdAtor)**

**DiretorDirige (IdDVD, IdDiretor)**

Criei esse modelo em um programa chamado DbDesigner para ficar mais fácil a visualização:



Na figura acima temos as Relações ligadas por linhas. Essa notação é conhecida como modelo do pé-de-galinha, pois podemos ter no final de uma linha que liga duas relações esse ponta de três linhas, parecido um pé de galinha. Ele significa o nosso N. Por exemplo, vejam que a relação entre Amigo e Empréstimo é 1:N, pois um amigo efetua N empréstimos, e 1 empréstimo está relacionado com um amigo. Assim, temos o pé de galinha colado à caixa que representa a relação empréstimo.

Notem também que cada caixinha tem na parte de cima o nome da Relação. Depois vem uma linha de separação e começam os campos. Os campos que fazem parte da chave primária estão separados dos demais por outra linha, além de terem uma figura de chave ao seu lado. Os campos que são chaves estrangeiras tem um FK entre parênteses depois

deles. Do lado de cada campo tem um tipo, que define o **domínio** do campo. Por fim, vocês devem ter notado que próximo às linhas que ligam as relações (indicando relacionamentos) temos números. Esses números estão aí porque esse programa específico (DbDesigner) exige que eu nomeie os relacionamentos. Mas na verdade no Modelo Relacional o que define os relacionamentos são as chaves estrangeiras.

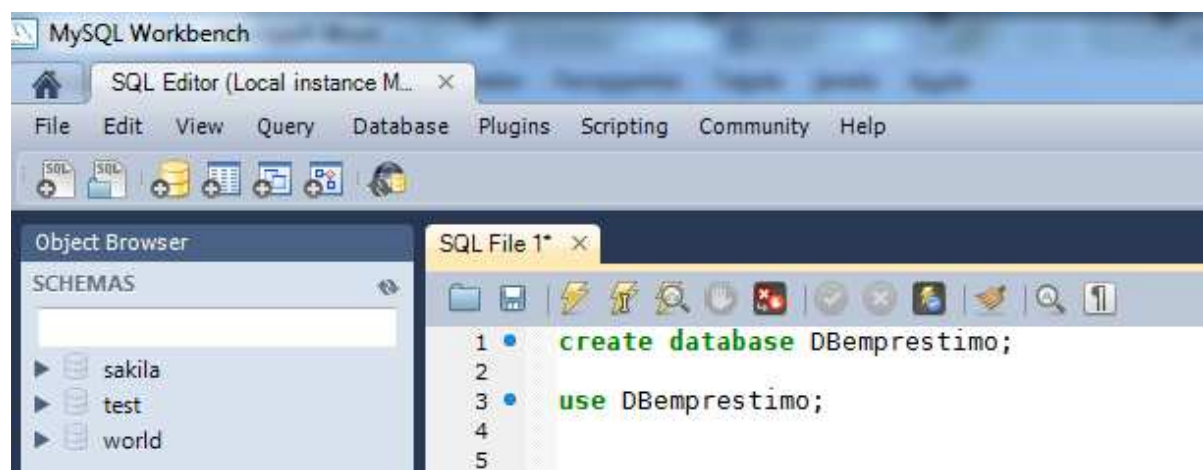
Não se preocupem muito com essa notação, só coloquei porque fica mais fácil de visualizar nosso modelo. Esse diagrama de pé-de-galinha é usado por alguns até para fazer MER, mas isso não é relevante para nosso estudo. O mais comum é termos os MER naquele formato que eu apresentei.

## 2.1. CREATE

O primeiro comando SQL que vamos ver é o **CREATE**. Ele serve para criar um objeto no Banco de Dados. O primeiro uso que vamos fazer dele é para criar o próprio Banco de Dados. Vou chamar nosso Banco de Dados de DBemprestimo, pois ele vai controlar os meus empréstimos. O comando a ser digitado é o seguinte:

**Create database DBemprestimo;**

Vamos ver na tela do SGBD:



No caso do MySQL, basta clicar no símbolo do trovão que esse comando será executado, criando um Banco de Dados vazio. Reparem no lado esquerdo três nome: sakila, test e world. São três Bancos de Dados, ou esquemas de Bancos de Dados, que já existem. O comando que está na sequência (use DBemprestimo;) informa ao SGBD que estamos usando a partir desse momento esse Banco de Dados.

Um esquema do Banco de Dados é uma coleção de objetos de um banco de dados que estão disponíveis para um determinado usuário ou grupo. Os objetos de um esquema são estruturas lógicas que se referem diretamente aos dados do banco de dados. Eles incluem estruturas, tais como tabelas, visões, seqüências, procedimentos armazenados, sinônimos, índices, agrupamentos e links de banco de dados. Quando eu falar em esquema, entendam como sinônimo de Banco de Dados propriamente dito.

O comando Create também serve para criar outros objetos, como as **tabelas**, que será nosso próximo passo. Vamos pegar algumas relações e ver como ficam os comandos Create para criá-las no nosso Banco de Dados chamado DBemprestimo. Primeiro vamos analisar a criação da tabela Diretor.

```
CREATE TABLE Diretor (  
    idDiretor INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,  
    nomeDiretor VARCHAR(50) NOT NULL,  
    PRIMARY KEY(idDiretor)  
);
```

O Create Table serve para criar uma tabela no nosso Banco de Dados. O que estou dizendo nesse comando para o SGBD é o seguinte: crie uma tabela chamada Diretor. O primeiro campo dessa tabela vai se chamar idDiretor. O tipo desse campo (domínio) é INTEGER UNSIGNED, ou seja, ele só vai aceitar valores inteiros sem sinal (positivos). Esse campo não aceita Nulos (NOT NULL). Por fim, os valores desse campo vão ser gerados automaticamente por você, SGBD, na forma de auto-

incremento, ou seja, o primeiro registro gravado vai ter nesse campo o valor 1, o segundo o valor 2 e assim sucessivamente. O segundo campo (as definições de cada campo são separados por vírgula) se chamará nomeDiretor. Seu domínio são sequências de caracteres de até 50 letras, e ele não aceitará valores nulos. Por fim, informo ainda que o campo IdDiretor é a chave primária dessa tabela.

Parece bem complicado, não é? A boa notícia é que ainda não vi uma banca cobrar o conhecimento desse comando, para a criação de tabelas. Até porque isso é trabalho para o super DBA, nem mesmo os desenvolvedores criam esse comandos. Sinceramente não acredito que a ESAF vá cobrar isso de vocês. De qualquer forma, devemos tentar entender esse comando. Existe uma sequência lógica nele que deve ser seguida. Temos as palavras-chave CREATE TABLE, seguida pelo nome da Tabela, depois entre parênteses a definição dos campos, no seguinte formato: nome de cada campo, seguido pelo tipo, definição de NULL, e as vezes algum parâmetro a mais opcional, como AUTO\_INCREMENT. Na última linha, temos a definição da chave primária. Reparem que a definição de cada campo é separada por uma vírgula no final. Por fim, vale a ressaltar que esse AUTO\_INCREMENT é uma característica do MySQL. No Oracle, por exemplo, não existe esse AUTO\_INCREMENT.

É importante notar que os SGBDs, assim como tudo na computação, esperam que quem direcione instruções para o computador obedeça a algumas regras de **sintaxe**. Dessa forma, o CREATE TABLE tem sua sintaxe definida, e se errarmos alguma palavra que ele espera, ou apresentamos as informações na ordem incorreta, o SGBD simplesmente não reconhece o comando e dá uma mensagem de erro. Por exemplo, se colocarmos na terceira linha:

nomeDiretor NOT NULL VARCHAR(50) ,

Esse comando não vai ser reconhecido pelo SGBD, pois a sintaxe da definição dos campos no comando CREATE TABLE começa com o nome do campo, seguido do tipo, e por fim se aceita nulos ou não.

Vou colocar os demais comandos de criação das tabelas do nosso esquema, e analisaremos mais um para fechar essa parte.

```
CREATE TABLE DVD (  
    idDVD INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,  
    tituloPort VARCHAR(45) NULL,  
    tituloIng VARCHAR(45) NULL,  
    tipo CHAR NOT NULL,  
    genero VARCHAR(25) NOT NULL,  
    ano INTEGER UNSIGNED NULL,  
    PRIMARY KEY(idDVD)  
);  
  
CREATE TABLE Amigo (  
    cpf VARCHAR(11) NOT NULL,  
    nome VARCHAR(45) NOT NULL,  
    endereco VARCHAR(60) NOT NULL,  
    tel1 VARCHAR(10) NOT NULL,  
    tel2 VARCHAR(10) NULL,  
    tel3 VARCHAR(10) NULL,  
    PRIMARY KEY(cpf)  
);  
  
CREATE TABLE Ator (  
    idAtor INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,  
    nomeAtor VARCHAR(50) NOT NULL,  
    PRIMARY KEY(idAtor)  
);  
  
CREATE TABLE Emprestimo (  
    idEmprestimo INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,  
    cpf VARCHAR(11) NOT NULL,  
    idDVD INTEGER UNSIGNED NOT NULL,
```



```
dataEmprest DATE NOT NULL,  
dataPrev DATE NOT NULL,  
dataDevol DATE NULL,  
PRIMARY KEY(idEmprestimo),  
INDEX Emprestimo_FKIndex1(idDVD),  
INDEX Emprestimo_FKIndex2(cpf),  
FOREIGN KEY(idDVD)  
REFERENCES DVD(idDVD)  
ON DELETE RESTRICT  
ON UPDATE CASCADE,  
FOREIGN KEY(cpf)  
REFERENCES Amigo(cpf)  
ON DELETE RESTRICT  
ON UPDATE CASCADE  
);  
  
CREATE TABLE AtorParticipa (  
idDVD INTEGER UNSIGNED NOT NULL,  
idAtor INTEGER UNSIGNED NOT NULL,  
PRIMARY KEY(idDVD, idAtor),  
INDEX DVD_has_Ator_FKIndex1(idDVD),  
INDEX DVD_has_Ator_FKIndex2(idAtor),  
FOREIGN KEY(idDVD)  
REFERENCES DVD(idDVD)  
ON DELETE RESTRICT  
ON UPDATE CASCADE,  
FOREIGN KEY(idAtor)  
REFERENCES Ator(idAtor)  
ON DELETE RESTRICT  
ON UPDATE CASCADE  
);  
  
CREATE TABLE DiretorParticipa (  
idDVD INTEGER UNSIGNED NOT NULL,  
idDiretor INTEGER UNSIGNED NOT NULL,
```

```
PRIMARY KEY(idDVD, idDiretor),  
INDEX DVD_has_Diretor_FKIndex1(idDVD),  
INDEX DVD_has_Diretor_FKIndex2(idDiretor),  
FOREIGN KEY(idDVD)  
REFERENCES DVD(idDVD)  
ON DELETE RESTRICT  
ON UPDATE CASCADE,  
FOREIGN KEY(idDiretor)  
REFERENCES Diretor(idDiretor)  
ON DELETE RESTRICT  
ON UPDATE CASCADE  
);
```

Pronto, esses scripts para a criação das tabelas são a base do nosso projeto físico, pois definem como serão minhas tabelas no Banco de Dados. Vamos analisar somente mais um pedaço dessa última tabela, chamada de diretorParticipa. Lembram como ela surgiu? Do relacionamento N:M de DVD a Diretor. Bem, ela tem no início a definição dos seus campos, sua chave primária, que é composta de dois campos, depois vem dois comandos INDEX que vamos ignorar por enquanto. Por fim, temos as definições de chaves estrangeiras, que é o que nos interessa.

Primeiro vem o comando FOREIGN KEY(idDVD), informando que esse campo (idDVD) é uma chave estrangeira. Mas essa chave vem da onde? A sequência do comando informa isso, dizendo que idDVD dessa tabela vem de tabela DVD, no campo IdDVD. Isso é mostrado por REFERENCES DVD(idDVD). Ou seja, estou dizendo ao SGBD que o campo IdDVD da tabela diretorParticipa é uma chave estrangeira da tabela DVD, sendo relacionada na tabela de origem ao campo idDVD. Apesar dos campos terem o mesmo nome nas duas tabelas, **isso não é obrigatório**, e é por isso que no REFERENCES devemos colocar o nome da tabela e do campo de origem.

## 2.2. ALTER

Agora veremos nosso segundo comando de DDL, que é o ALTER. Como o nome sugere, o ALTER serve para alterar determinado objeto. Imaginem que eu resolvo mudar a estrutura de determinada tabela, adicionando um campo extra. O comando para executar essa tarefa é o ALTER.

Por exemplo, imaginem que eu desejo alterar a tabela Ator, colocando lá um campo a mais para armazenar a nacionalidade do ator. Vamos ver como fica o comando que executa essa tarefa.

```
ALTER TABLE Ator ADD COLUMN nacionalidade VARCHAR(25) NULL;
```

Esse comando SQL diz o seguinte para o SGBD: Olha SGBD, altere a tabela Ator, adicionando a coluna nacionalidade. Ela vai ser do tipo sequência de caracteres, com tamanho de 25 posições, e aceita valores nulos. Bem simples, não é? Vamos ver mais um.

```
ALTER TABLE Amigo ADD COLUMN sexo CHAR NOT NULL CHECK (sexo IN ('M','F'));
```

Esse comando acrescenta o campo sexo, na tabela Amigo. Ele será do tipo CHAR (aceita um caractere) e não aceita valores nulos. O que ele tem diferente está no final é a cláusula CHECK. Ela serve para que o SGBD faça uma checagem, restringindo **o domínio desse campo**. O significado da expressão sexo IN ('M','F') é a seguinte: SGBD, só aceite para o campo sexo os valores que estão listados dentro dos parênteses que vêm depois do IN. Ou seja, só aceite os valores 'M' ou 'F', e nenhum outro valor. E essas aspas simples, que envolvem o M e o F, o que significam? É que todas as expressões literais, ou seja, que envolvem campos que aceitam caracteres, como o CHAR e o VARCHAR, devem estar entre aspas simples. Assim, para os SGBD (e na computação em geral), 1 é diferente de '1'. O primeiro representa o valor numérico 1, o segundo representa o literal 1. Por fim, é importante observarmos que o domínio

que eu defini só aceita 'M' ou 'F' maiúsculos. Assim, se eu tentar gravar o literal 'f' ou o literal 'm', o SGBD não vai aceitar.

Alguém pode perguntar, eu posso usar o CHECK lá no CREATE TABLE? Sim, pode sim, vejam que depois do ALTER TABLE ADD COLUMN, o que vem é uma definição de campo, da mesma forma que tinha lá nos comandos CREATE TABLE.

Tudo certo, até o momento vimos o CREATE e o ALTER. Usamos o CREATE para criar o Banco de Dados e para criar tabelas, e o ALTER para alterar tabelas. Mas o que mais podemos criar ou alterar? Existem outros objetos no Banco de Dados, entre eles visões, triggers, índices, procedures etc etc. Os comandos CREATE e ALTER também servem para criar e alterar esses objetos. Mas por enquanto vou me concentrar nas tabelas, depois veremos com mais detalhes e que são esses outros objetos, e como criá-los ou alterá-los.

### **2.3. DROP**

Bem, se podemos criar a alterar, também devemos poder excluir, destruir, deletar, ou como se diz no jargão de Banco de Dados, “dropar”. Para isso existe outro comando na DDL, que é o comando SQL DROP. Por exemplo, quer apagar seu Banco de Dados todo? Simples, é só usar:

**DROP DATABASE DBemprestimo;**

Pronto, já era, foi-se o Banco de Dados!! Lembram que eu citei a grande responsabilidade nas mãos do DBA? Pois é, o cidadão pode simplesmente com 3 palavrinhas mágicas mandar o Banco de Dados para o saco!!!!

Certo, agora sendo menos radical, como fazemos para excluir uma tabela? Simples, o comando é:

**DROP TABLE Diretor;**

Nesse momento era uma vez a tabela Diretor. Se apagar acidentalmente, ou você tem backup, ou já era. Agora, os SGBDs têm mecanismos de integridade, que vimos na aula passada, lembram? Dependendo da configuração da tabela, se ela tiver um campo que é chave estrangeira em outra, o SGBD não vai permitir a exclusão da mesma, porque isso iria ferir a integridade referencial. Nesse caso, se a tabela Diretor tiver registros relacionados com a tabela DiretorParticipa, aí o SGBD não exclui a tabela, e retorna um erro na execução desse comando.

Mas e se eu quiser “dropar” apenas uma coluna de uma tabela, pode? Sim, pode. Usamos o comando ALTER combinado com o DROP. Da seguinte forma:

```
ALTER TABLE Ator DROP COLUMN nacionalidade;
```

O DROP é bem simples, só isso mesmo. Assim, como o ALTER e o CREATE, o DROP pode ser usado naqueles outros objetos que eu citei. Agora sim chegou o momento de vê-los.

## 2.4. OBJETOS DO BANCO DE DADOS

Os objetos disponíveis nos SGBD também variam em cada produto, mas vou citar aqui os mais comuns, que são INDEX (índices), TRIGGER (gatilho), FUNCTION (função) e PROCEDURE (procedimento). Ainda temos um objeto muito importante, a VIEW (visão), mas só veremos depois, pois temos que ver um pouco de DML para entender melhor as VIEWS.

Primeiro veremos **índices**. **Índices** são objetos do Banco de Dados que servem para tornar as consultas mais rápidas. É um processo parecido com o que fazemos quando procuramos uma informação em um livro. Qual o primeiro passo? Procuramos no índice do livro para saber em que página está a informação que queremos. E se o livro não tiver índice?

Bem, ai teremos que folhear página por página até encontrar o conteúdo desejado.

O mesmo processo se dá com um Banco de Dados. Por isso ele permite que se criem índices, de forma a tornar a recuperação da informação mais eficiente. Em primeiro lugar, informo para vocês que toda vez que definimos uma chave primária, o SGBD cria automaticamente um índice para aquele(s) campo (s). Dessa forma, existe um índice para o campo CPF na tabela DVD. Imaginem que essa tabela tenha 5 mil amigos cadastrados (pense numa pessoa popular!!). Se pedirmos para o SGBD procurar um amigo específico, a partir do seu CPF, o SGBD não vai procurar registro por registro da tabela (lembrem-se que os registros são as tuplas, ou linhas). O SGBD ai consultar um índice, e vai direto ao ponto na tabela, encontrando o amigo. Tudo bem, você pode pensar, 5 mil registros para um computador, mesmo procurando um por um, é um processo rápido. Mas acreditem, nas aplicações mundo afora temos tabelas monstruosas. Pensem na tabela da Receita Federal que guarda os CPFs, com mais de cem milhões de registros. Sem índices, os SGBDs não dariam conta.

Então, além daqueles índices que o SGBD cria automaticamente, podemos selecionar determinados campos e criar índices para esses campos. Vamos criar um índice para o campo nome da tabela Amigo, para que quando eu procure um amigo pelo nome, o SGBD use um índice.

```
CREATE INDEX ID_AMIGO_01 ON Amigo (nome);
```

O comando CREATE INDEX cria um índice. O parâmetro que coloquei depois, ID\_AMIGO\_01 é simplesmente o nome que eu escolhi para esse índice (poderia ser qualquer nome). Depois vem a palavra chave ON, e na sequência o nome da tabela e do(s) campo(s) entre parênteses.

Agora vamos ver as **TRIGGERS**. TRIGGER, ou gatilho, são trechos de código (sequência de comando SQL) que são ativados (disparados)

automaticamente. Funciona da seguinte forma: criamos uma trigger informando ao SGBD duas informações principais, um evento e uma ação. O evento é aquilo que vai ativar a trigger, e a ação são os passos que devem ser executados na trigger.

Vamos a um exemplo para clarear as idéias. Imagine que você tem um Banco de Dados que guarde em uma tabela a informação dos produtos vendidos da sua loja. Nessa tabela, além do código, descrição e preço do produto, tem um campo que guarda o valor percentual máximo que um vendedor pode dar de desconto. Ai você, muito precavido, pensa. E se o vendedor entender de SQL, entrar no Banco de Dados, alterar o valor do desconto para maior, e depois voltar o desconto ao patamar normal? Como eu posso me precaver para evitar essa fraude? Uma das maneiras seria a criação de uma trigger para essa tabela. Nessa trigger, você especificaria que o evento que irá disparar a trigger é uma alteração em um registro da tabela de produtos. E o que seria a ação? Bem, poderia ser gravar um registro em outra tabela, registrando a data, hora, funcionário que alterou, valor do desconto original e valor para o qual o desconto foi alterado. Pronto, você acabou de colocar o SGBD para vigiar essas operações para você. Ai não tem jeito, se o espertinho alterou o desconto de 5 para 50%, vendeu o produto para a cunhada e depois voltou para 5%, a trigger vai ser disparada duas vezes, e você vai poder descobrir a fraude.

Outro exemplo é programar uma trigger para que toda vez que um registro fosse excluído, você registrar essa informação em outra tabela, ou mesmo bloquear essa exclusão. Imaginem se o seu gerente pudesse simplesmente apagar um registro da tabela de contas a receber? Muito perigoso, não é?

Então, as triggers servem para isso. Mais importante que aprender a sintaxe de como criar uma trigger é entender o que é uma trigger e

para que ela é usada. Isso costuma ser cobrado nos concursos. Só por curiosidade, vou colocar a criação de uma trigger abaixo:

```
CREATE TRIGGER mytrigger  
BEFORE UPDATE ON emprestimo FOR EACH ROW  
BEGIN  
.....  
END;
```

Descrevendo o comando, temos o CREATE TRIGGER que são as palavras chaves que indicam a criação da trigger. Depois vem o nome da trigger. Na linha de baixo temos o evento que vai disparar a trigger. BEFORE UPDATE ON empréstimo POR EACH ROW significa que essa trigger vai ser executada antes de uma atualização em uma linha da tabela Empréstimo, ou seja, quando uma alteração é feita, antes de ela ser gravada no Banco de Dados, a trigger é disparada. Depois o Begin e End vão demarcar a ação que será tomada. Essa ação é um conjunto de comandos SQL que estarão entre o Begin e o End. Eu não coloquei nenhum ali, porque não é importante para nosso estudo agora, mas deve ter pelo menos um comando SQL entre o Begin e o End. Agora, afirmo com toda tranquilidade, não se preocupem com a sintaxe do CREATE TRIGGER, é muitíssimo improvável que isso seja cobrado. Guardem apenas a seguinte informação: **uma trigger é um objeto do banco de dados que é disparado por um evento, de forma automática, e quando é disparada executa uma série de ações. Esses eventos que disparam as triggers normalmente são manipulações em uma tabela, ou seja, comandos DML.**

O nosso próximo objeto são as **PROCEDURES**. Uma procedure é um conjunto de instruções em SQL que serve para executar uma determinada tarefa. Elas possibilitam que se guarde parte das regras de negócio da sua aplicação dentro do Banco de Dados. Também podem ser usadas pelo DBA para automatizar tarefas rotineiras (ex backups). Enfim, o que precisamos entender das PROCEDURES, ou como alguns SGBDs



chamam, **STORED PROCEDURES**, é que elas são como pequenos trechos de programa, ou seja, sequências de comandos, que realizam uma determinada tarefa. Qual a grande diferença entre PROCEDURES E TRIGGERS? Bem, ambas têm um conjunto de instruções que executam determinada tarefa, mas as TRIGGERS estão associadas a um evento, normalmente a determinada tabela, e são executadas automaticamente quando esse evento ocorre. Já as PROCEDURES não estão associadas a eventos, nem a tabelas específicas, e precisam ser executadas diretamente para surtirem efeito. Ou seja, ou você na linha de comando chama a procedure, da mesma forma que usamos comandos como o CREATE, ALTER etc, ou um determinado programa da sua aplicação faz uma chamada para essa PROCEDURE. Como se cria uma PROCEDURE? Vamos ver.

```
CREATE PROCEDURE myProcedure ()  
BEGIN  
....  
END;
```

Temos como sempre palavras chaves, que nesse caso são CREATE PROCEDURE, e depois vem o nome da PROCEDURE, seguido de parênteses. Dentro desses parênteses podemos passar parâmetros para a PROCEDURE, que servem de insumos para ela executar seus comandos. Imaginem que eu crie uma PROCEDURE que serve para calcular a quantidade de dias que um amigo da nossa aplicação está atrasado na entrega de uma mídia. Essa PROCEDURE, para poder ser genérica e poder ser usada de forma geral, deveria receber como parâmetro o CPF do amigo que eu quero efetuar o cálculo. Bem, não quero me aprofundar nisso, senão vamos entrar na seara da programação, e não é o nosso foco. Guardem como informação somente o essencial: as PROCEDURES são objetos do Banco de Dados que servem para guardar uma sequência de comandos, usados de forma lógica para resolver um determinado problema. Pronto, isso é suficiente.

Por fim, temos as FUNCTIONS. Uma FUNCTION, ou função, é bem semelhante a uma PROCEDURE. Só tem uma diferença. A FUNCTION retorna um valor, ou seja, ela é usada para efetuar determinado conjunto de comando, e no final, ela retorna o resultado. Por exemplo, podemos ter uma FUNCTION que recebe como parâmetro (entrada) um determinado CPF, e retorna o seu dígito verificador. Ela seria mais ou menos assim:

```
CREATE FUNCTION digito (CPF VARCHAR(11))  
RETURNS INTEGER  
Begin  
...  
End;
```

Nessa definição, além das palavras chaves e do nome da FUNCTION, coloquei um parâmetro de entrada (CPF, do tipo VARCHAR), e na linha seguinte coloquei uma cláusula RETURNS INTEGER, indicando que essa FUNCTION vai retornar um valor inteiro. Como ela vai fazer isso, como vai calcular o dígito e retornar? Isso vai ser feito por uma série de comandos que então entre o Begin e o End.

Bem, o essencial sobre objetos dos Bancos de Dados nós vimos. É importante saber quem são eles, e para que servem. Eu coloquei a sintaxe só para vocês não ficarem imaginando como se criam esses objetos. E vale a observação de que, assim como podemos criar esses objetos (CREATE), pode também alterá-los (ALTER) e excluí-los (DROP). Portanto, podemos usar o comando DROP INDEX nomedoindice, DROP PROCEDURE nomedaprocedure, e assim sucessivamente.

Não desanimem, nossa brincadeira com o SQL está apenas começando. Vamos a alguns exercícios:

**1. (ESAF/Analista Sistemas/ANA 2009) Em SQL, a cláusula check aplicada a uma declaração de domínio**

**a) permite especificar um predicado que deve ser satisfeito por qualquer valor atribuído a uma variável de determinado domínio.**

**b) especifica um predicado que deve ser satisfeito por uma tupla em uma relação.**

**c) proíbe a inserção de um valor nulo para as variáveis do domínio.**

**d) verifica se os atributos considerados formam uma chave candidata.**

**e) não tem efeito, pois não se aplica esta cláusula a declarações de domínio.**

Comentários:

Já vimos o CHECK, quando incluímos aquele campo sexo na tabela Amigo, lembram?

a) Um predicado é uma qualidade, uma característica. Na computação, entendam predicado como uma condição que deve ser satisfeita. Assim, quando definimos aquele CHECK, afirmando que seu domínio se limitava aos literais M ou F, definimos um predicado, e se ele não for satisfeito, o atributo (sexo naquele caso) não poderá receber o valor. Dessa forma, qualquer valor atribuído àquele campo deve satisfazer ao predicado. A afirmação é verdadeira. Ela é um pouco confusa, não é? Mas veremos que chegaríamos a ela nem que fosse por eliminação.

b) Esse predicado (condição) não deve ser satisfeito por uma tupla específica, mas sim por todas as tuplas. Afinal de contas, definimos o domínio daquele campo, e todas as tuplas da nossa tabela têm os mesmos campos. Nenhuma tupla pode aceitar valores diferentes daqueles definidos pelo domínio.

c) Não, a cláusula que proíbe a inserção de nulos é o NOT NULL, que também comentamos.

d) Essa é moleza, não é? Nada a ver essa afirmativa. CHECK não tem nenhuma relação com chaves.

e) Muito pelo contrário, só se aplica a declaração de domínios, ou seja, ela é feita mesmo para determinar o domínio de um campo.

**Gabarito: Letra a**

**2. (ESAF/Auditor Informática/Pref Natal 2008) Quanto à estrutura, propriedades e sintaxe da linguagem SQL, é correto afirmar que**

**a) o CREATE e o DROP são comandos básicos da DDL (Data Definition Language - Linguagem de Definição de Dados).**

**b) o SELECT e o INSERT são comandos básicos da DDL ( Data Definition Language - Linguagem de Definição de Dados ).**

**c) o CREATE e o DROP são comandos básicos da DCL ( Data Control Language - Linguagem de Controle de Dados ).**

**d) o GRANT e o REVOKE são comandos básicos da DDL ( Data Manipulation Language - Linguagem de Manipulação de Dados ).**

**e) o SELECT e o INSERT são comandos básicos da DCL ( Data Control Language - Linguagem de Controle de Dados ).**

Comentários:

Ainda não vimos alguns desses comandos que aparecem ai, mas já vimos CREATE e DROP, e sabemos que são comandos da DDL. Assim, facilmente chegamos na letra (a) como resposta.

**Gabarito: Letra a**

**3. (ESAF/Auditor Informática/Pref Natal 2008) Com relação às características da linguagem SQL e dos SGBD, é correto afirmar que**

**a) os "Triggers" são utilizados para auxiliar a manutenção da consistência dos dados. Também podem ser utilizados para**

**propagar alterações ocorridas em um determinado dado de uma tabela para outra tabela.**

**b) considerando o SQL\_ANSI, o código a seguir é um exemplo correto para a criação de um Trigger relacionado a ações na tabela "cadastro": CREATE TRIGGER trigger\_cpf AS Codigo a ser executado FOR UPDATE ON cpf.cadastro**

**c) os "Triggers" são relações que não armazenam dados, mas são definidos dinamicamente por uma consulta a uma tabela previamente analisada e otimizada.**

**d) os "Triggers" são relações que armazenam dados, definidos dinamicamente por uma regra de inserção a uma tabela.**

**e) os "Triggers" são um arquivo auxiliar associado a uma ou mais tabelas. Sua função é acelerar o tempo de acesso aos dados de uma tabela.**

Comentários:

a) Esse item não define TRIGGERS, mas cita duas aplicações da TRIGGER. Auxiliar na manutenção da consistência dos dados. É verdade, já que as TRIGGERS podem ser executadas automaticamente quando queremos inserir, alterar ou excluir um dado, podemos programar a TRIGGER para ela determinar se aquela operação pretendida está correta (naquele exemplo citado, podemos fazer com que a TRIGGER não permita a alteração do valor do desconto). Outra utilidade citada na questão é propagar alterações ocorridas em uma tabela para outra. Também está correto. Podemos, no momento que uma tabela for alterada, através de uma TRIGGER propagar essa alteração para outras tabelas.

b) Bem, para começar a sintaxe mostrada está diferente do que eu mostrei. Mas o grande erro da questão é que não existe a definição de TRIGGERS em ANSI SQL. Cada SGBD tem seu formato para criar e alterar TRIGGERS.

- c) TRIGGERS não são relações. Ponto.
- d) De novo: TRIGGERS não são relações. Ponto.
- e) TRIGGERS não são arquivos.

**Gabarito: Letra a**

**4. (ESAF/Analista de TI/SEFAZ-CE 2007) Na linguagem SQL, os procedimentos gravados para serem executados implicitamente quando ocorrer determinada ação do usuário, como, por exemplo, a modificação de uma tabela, são denominados**

- a) Selects.**
- b) Inserts.**
- c) Views.**
- d) Queries.**
- e) Triggers.**

Comentários:

Admito que as anteriores eram um pouco enroladas, mas vejam que a ESAF já cobrou esse conceito de TRIGGER de maneira bem fácil, como nessa questão.

**Gabarito: Letra e**

**5. (ESAF/Analista de Sistemas/ENAP 2006) A SQL possui recursos para apagar tabelas e bancos de dados a partir do comando**

- a) CREATE TABLE NULL.**
- b) TRUNCATE PRIMARY KEY.**
- c) DROP.**
- d) INSERT INTO TABLE NULL.**

**e) DELETE TABLE NULL.**

Comentários:

Mais uma bem fácil, não é?

**Gabarito: Letra c**

**6. (ESAF/Analista Informática/IRB 2006) Considere um banco de dados com uma tabela com o nome "Carro", onde estão cadastrados os mais de 1000 veículos de uma empresa. Esta tabela contém um campo de nome "Cor", que pode ser nulo, contendo a informação da cor de cada veículo cadastrado. Ao se executar a instrução SQL ALTER TABLE Carro DROP COLUMN Cor**

**a) a tabela Carro será totalmente apagada devido à execução do comando DROP.**

**b) todos os registros preenchidos com o valor Null serão alterados.**

**c) todos os registros preenchidos com o valor diferente de Null serão alterados.**

**d) todas as informações das cores dos veículos contidas neste campo serão perdidas.**

**e) nada irá acontecer com os dados da tabela Carro.**

Comentários:

a) Item errado. O comando em questão apaga apenas uma coluna, e não a tabela inteira.

b) Apagar uma coluna nada tem a ver com os valores NULL.

c) idem

d) Sim, se apagamos a coluna que continha as cores dos veículos, logicamente todas as informações que continham nesse campo serão perdidas. Item correto.

e) Claro que algo vai acontecer, afinal estamos apagando uma coluna.

**Gabarito: Letra d**

**7. (FCC/ACE-TI/TCE-SE 2011) Durante a criação de uma tabela - Create Table, em SQL, deseja-se especificar que uma coluna só possa incluir, por exemplo, valores maiores que zero. Uma constraint utilizada para isso é**

**a) Verify.**

**b) Check.**

**c) Max.**

**d) Avg.**

**e) Having.**

Comentários:

Coloquei essa questão da FCC para chamar a atenção a um termo muito usado em Banco de Dados, que é constraint. Constraints nada mais são que restrições que você estabelece para uma coluna no banco de dados, ou seja, um método para validar a integridade de todos os dados que entram em sua base. Assim, pelo que estudamos, fica claro que o Check é a constraint que pode ser usada para restringir o domínio de determinado valor. Como ficaria essa definição de campo em uma tabela qualquer? Ficaria algo do tipo: nomeColuna INTEGER NOT NULL CHECK (nomeColuna > 0).

**Gabarito: Letra b**



### 3. DML

A Linguagem de manipulação de dados (DML) é a linguagem usada para que possamos fazer as operações básicas com os dados, que é o que realmente nos interessa nos Banco de Dados. Até agora criamos a infraestrutura do nosso Banco de Dados. Usamos a DDL para criar o Banco de Dados e seus objetos, principalmente as tabelas, que são os elementos mais importantes. Claro, dependendo da necessidade, podemos criar também triggers, procedures, function, índices etc. Mas as tabelas que criamos estão vazias, ou seja, não têm dados.

As principais operações feitas nos Bancos de Dados, a partir desse momento são: inclusão de dados, exclusão de dados, alteração de dados e consulta a dados. Tudo isso fazemos com os comandos da DML. Só um parêntese, no mundo real, quem vai alimentar (povoar) esse Banco de Dados são as aplicações, que terão aquelas duas camadas iniciais (interface e negócios) para receber, tratar esses dados e passá-los para a camada de persistência (Banco de Dados propriamente dito). Mas como no nosso curso não vamos criar aplicações, faremos essas operações diretamente nos nossos Bancos de Dados. No fundo, as aplicações vão ter, embutida nelas, os comandos que vamos aprender.

Já que o nosso Banco de Dados DBemprestimo tem um monte de tabelas vazias, vamos “começar do começo”. Primeiro, vamos inserir dados.

#### 3.1. INSERT

O INSERT é o comando SQL que serve para inserirmos dados em determinada tabela do Banco de Dados. Vamos ver como funciona.

```
INSERT INTO Ator (nomeAtor) VALUES ('KEANU REAVES');  
INSERT INTO Ator (nomeAtor) VALUES ('LAURENCE FISHBURNE');
```

### COMMIT;

Lembram da estrutura (metadados) da tabela Ator? Só tinha dois campos, IdAtor e nomeAtor. E lembram que o campo idAtor é AUTO INCREMENT? Pois é, disse para vocês que eu não precisava informar esse campo, já que o SGBD ia preencher sozinho com valores sequenciais. Vamos então analisar o comando.

No início vêm as palavras chaves INSERT INTO, que informam que queremos inserir uma informação. Mas inserir onde? Na sequência temos o nome da tabela, e entre parênteses o nome do campo que queremos preencher. Depois vem mais uma palavra chave, o VALUES, e na sequência, entre parênteses, o(s) valor(es) que queremos inserir na tabela. Nesse caso é apenas a string KEANU REAVES, no primeiro comando, e a string LAURENCE FISHBURNE, no segundo. Entendam por string uma sequência de caracteres. Observem que são valores literais, por isso estão entre aspas simples. Nesse caso, como estão sendo usados dois comandos INSERT para gravar o primeiro e o segundo registro nessa tabela, vai ser gravado o valor 1 em IdAtor, e o valor 'KEANU REAVES' em nomeAtor, isso no primeiro registro. No segundo registro vai ser gravado o valor 2 em IdAtor, e o valor 'LAURENCE FISHBURNE' em nomeAtor. Bem intuitivo, não acham?

Nesse momento devo comentar logo um comando muito importante, que sempre aparece depois de um ou vários comandos DML. É o comando **COMMIT**. Ele deve ser executado depois dos comandos DML para confirmar a operação, ou seja, ele grava definitivamente a operação realizada no comando no Banco de Dados. Se existe um comando que confirma a operação, deve existir um que desfaz, não é? Existe sim, é o comando **ROLLBACK**. Com ele, um comando que acabou de ser executado e você percebeu que estava errado, pode ser revertido. Esses dois comandos servem apenas para isso, e são compostos mesmo apenas por uma palavra. Mas atenção em uma observação, para depois vocês não me chamarem de mentiroso!!! No ambiente de trabalho do MySQL,

que vocês instalaram, existe uma configuração que é AUTO COMMIT, que está setada por padrão para ON, ou seja, no MySQL WorkBench vocês não precisam executar o COMMIT, ele é executado automaticamente. Essa opção está no Menu Query, opção Auto-Commit Transactions. Mas guardem isso: o COMMIT confirma todas as operações DML anteriores à sua execução, e o ROLLBACK desfaz essas operações (na verdade elas não foram nem gravadas no SGBD ainda).

Voltemos ao INSERT. Vamos ver uma variação dele em outra tabela. Vamos considerar a tabela Amigo original, sem o campo sexo (se você criou o campo quando estava aprendendo o CHECK, é só dropar).

```
INSERT INTO Amigo VALUES ('12345678911','ZE','RUA Z NRO 15','2198765432',NULL,NULL);
```

O que esse INSERT tem diferente do outro? Além de eu ter inserido vários campos, eu não listei o nome dos campos, depois do nome da tabela. Isso pode Arnaldo? Pode sim, a regra é clara. Quando queremos inserir todos os campos, não precisamos especificar depois do nome da tabela os nomes dos campos, basta respeitar a ordem em que eles estão criados na tabela. Assim, o INSERT acima é equivalente a este, que especifica os campos:

```
INSERT INTO Amigo (CPF,nome,endereco,tel1,tel2,tel3) VALUES ('12345678911','ZE','RUA Z NRO 15','2198765432',NULL,NULL);
```

Sobre o INSERT é só isso, bem fácil o comando.

### **3.2. DELETE**

Vamos para a segunda operação básica do DML, a exclusão. Para isso, usamos o comando DELETE. Sua forma mais simples é a mais perigosa! Vamos a ela:

```
DELETE FROM Amigo;
```

Sabe o que acabou de acontecer? Simplesmente você excluiu TODAS as tuplas dessa tabela. Se vocês tentarem fazer isso no MySQL, ele não vai deixar, porque ele tem umas configurações de segurança que vêm habilitadas por padrão, e precisam ser desabilitadas. Mas acreditem, o comando DELETE, do jeito que está escrito acima, é totalmente válido, e limpa a tabela.

Então você me pergunta: mas professor, eu tenho que deletar logo tudo de uma vez? Não tem como deletar só um ou alguns registros? Ah meus amigos, como gostam de dizer os nerds, só a matemática salva!!! Lembram da álgebra relacional, que vimos aula passada? Pois é, temos lá a operação de seleção, que faz um particionamento horizontal na tabela, ou seja, seleciona algumas linhas de uma relação. Então, no SQL foi criada uma cláusula que implementa essa operação da álgebra relacional, que é o WHERE. Vamos ver como funciona:

```
DELETE FROM Amigo WHERE cpf = '12345678911';
```

O que tem de diferente agora? Especificamos a cláusula WHERE, que implementa a operação de seleção, e depois dela colocamos uma condição. O que esse comando faz é o seguinte então: exclui as linhas da tabela Amigo que têm o campo CPF com valor igual a '12345678911'. Nesse caso, só tem uma linha que obedece a essa condição. Vamos falar muito do WHERE ainda, porque existem diversas formas de especificar a condição (predicado). Nesse caso só temos uma linha (um registro) que satisfaz essa condição, e ele é excluído (depois do commit, se o SGBD não estiver configurado com auto commit). E se mais de uma linha satisfizer a condição? Todas as linhas que satisfizeram a condição serão deletadas. E se nenhuma linha satisfizer a condição, é retornado um erro? Não, simplesmente o comando não faz nada. Com relação ao DELETE, é só isso.

Temos um outro comando importante, relacionado ao DELETE, que é o TRUNCATE TABLE NomeTabela. Ele limpa toda a tabela, deixando-a

vazia. Parece muito com o DELETE FROM Tabela, mas a diferença é que o TRUNCATE TABLE é um comando considerado de DDL, pois ele deixa a tabela como se estivesse acabado de ser criada. O TRUNCATE TABLE também não aciona triggers. Por fim, o TRUNCATE TABLE não pode ser desfeito (Rollback). O DELETE pode.

### 3.3. UPDATE

Nosso próximo amiguinho é o comando UPDATE. Ele é usado para atualizar um ou mais campos de uma ou mais linhas de determinada tabela. Vamos ver como funciona.

```
UPDATE Ator SET nomeAtor = 'KEANU REEVES'
```

Opa, assim como no DELETE, eu coloquei esse primeiro comando sem o WHERE, ou seja, sem a operação de seleção. Se eu executar esse comando do jeito que está, vou atualizar o nome de todos os atores que estão na minha tabela para a string KEANU REEVES. Tudo bem, o cara é o The One, mas assim já é demais! Na verdade, quero atualizar apenas a tupla que inseri lá atrás, porque o nome estava errado. Então o comando correto ficaria assim:

```
UPDATE Ator SET nomeAtor = 'KEANU REEVES' WHERE nomeAtor =  
'KEANU REAVES';
```

Agora sim, como eu tinha cadastrado esse ator com o nome errado, eu uso esse nome como referência de pesquisa, e informo ao SGBD exatamente qual tupla eu quero atualizar. Dessa forma, no UPDATE temos o a palavra chave UPDATE, depois vem o nome da tabela, depois temos a palavra chave SET, seguida pelo(s) campo(s) que eu quero atualizar, e por fim devo ter um WHERE com minha operação de seleção, para informar em que linhas (ou registros, ou tuplas) serão feitas as atualizações. Vamos ver mais um exemplo?

```
UPDATE Amigo SET tel2 = '6198789876', tel3 = '9298709862'  
WHERE cpf = '12345678911';
```

Observem que agora eu atualizei dois campos de uma só vez (tel2 e tel3), apenas separando eles e seus valores por vírgula.

### 3.4. SELECT

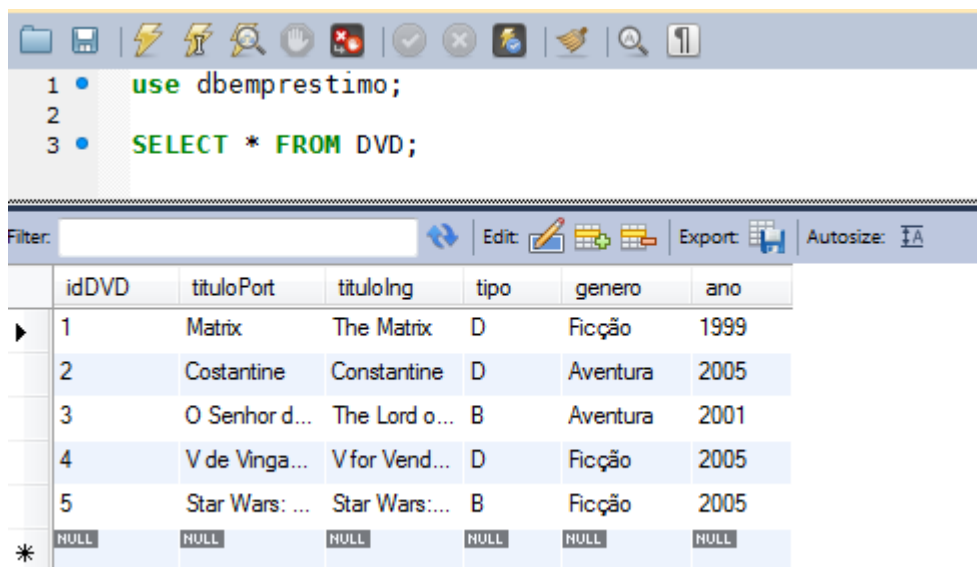
Agora chegamos no mais importante dos comandos do SQL. Sim, isso mesmo, o mais importante, para a maioria dos usuários de Banco de Dados. Isso porque é esse comando que recupera as informações do Banco de Dados. Ele vem nos modelos do popular ao luxo, ou seja, ele tem versões bem simples, e outras bem complexas. Acredito que se realmente a CGU colocar Bancos de Dados para vocês analisarem, vão usar muito o comando SELECT. Agora que já fiz o comercial dele, vamos com calma ver algumas de suas diversas aplicações.

Em primeiro lugar, disponibilizei para vocês dois scripts. Um que exclui o Banco de Dados e o recria limpo. O segundo povoa as tabelas do nosso Banco de Dados (menos a empréstimo), para que possamos usá-las em nossos testes com o comando SELECT.

Vamos começar com o uso mais básico do SELECET (sem ar, nem direção nem trava elétrica). Em seu uso mais básico, utilizamos a operação de projeção da álgebra relacional, seria algo assim:

```
SELECT * FROM DVD;
```

Esse comando seleciona todas as tuplas da tabela DVD. Isso porque eu não especifiquei o WHERE, ou seja, a seleção. Mas o que significa aquele \* depois do SELECT? É a nossa projeção. O \* informa ao banco de dados que eu quero todas as colunas dessa relação. O resultado deve ser parecido com a tela abaixo:



```
1 • use dbemprestimo;  
2  
3 • SELECT * FROM DVD;
```

	idDVD	tituloPort	tituloIng	tipo	genero	ano
▶	1	Matrix	The Matrix	D	Ficção	1999
	2	Costantine	Constantine	D	Aventura	2005
	3	O Senhor d...	The Lord o...	B	Aventura	2001
	4	V de Vinga...	V for Vend...	D	Ficção	2005
	5	Star Wars: ...	Star Wars:...	B	Ficção	2005
*	NULL	NULL	NULL	NULL	NULL	NULL

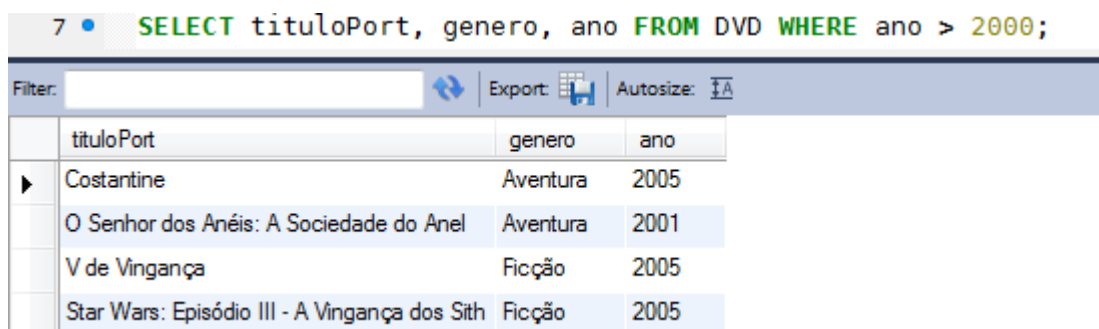
Bem, vamos avançar. Agora quero projetar somente alguns campos, mas quero ver todas as tuplas, como faço? O SELECT a seguir:

```
SELECT tituloPort, genero, ano FROM DVD;
```

Executando esse comando vocês verão que somente as colunas que eu listei depois do SELECT irão aparecer. Bem esse é o básico, usando apenas a projeção. Agora mais fazer um SELECT usando projeção e seleção.

```
SELECT tituloPort, genero, ano FROM DVD WHERE ano > 2000;
```

Vejam a diferença agora. Eu usei a projeção (listei os campos que eu queria depois do SELECT) e usei a seleção (coloquei um predicado depois do WHERE. Nessa query (consulta), eu queria três campos, mas somente das tuplas em que o campo ano fosse maior que 2000. Olhem o resultado, que legal:



```
7 • SELECT tituloPort, genero, ano FROM DVD WHERE ano > 2000;
```

	tituloPort	genero	ano
▶	Costantine	Aventura	2005
	O Senhor dos Anéis: A Sociedade do Anel	Aventura	2001
	V de Vingança	Ficção	2005
	Star Wars: Episódio III - A Vingança dos Sith	Ficção	2005

Percebam também que antes usamos somente o igual, como operador do predicado. Agora utilizamos o operador maior, e temos como resultado as tuplas em que o campo ano é maior que 2000. Temos diversos outros operadores, como < (menor), >= (maior ou igual), <= (menor ou igual), <> (diferente).

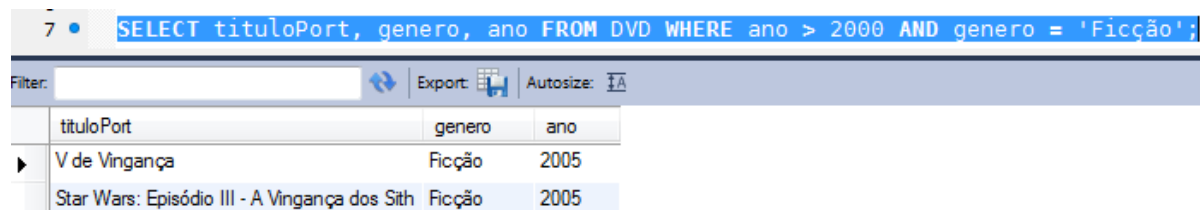
Agora, se eu quiser recuperar as tuplas em que os filmes tiverem o campo ano maior que 2000, e ao mesmo tempo sejam filmes de ficção, como eu faço?

Bem, existem alguns operadores lógicos que podem ser usados para eu elaborar condições mais sofisticadas. Eles são o operador AND, OR e NOT. Se você já estudou raciocínio lógico, já viu esses operadores. Se não, vai ver agora.

O operador AND serve para encadear duas condições. E as duas devem ser verdadeiras para satisfazer a condição. Então, respondendo a pergunta, quais os filmes com ano maior que 2000 e que são de ficção, a query fica assim:

```
SELECT tituloPort, genero, ano FROM DVD WHERE ano > 2000 AND genero = 'Ficção';
```

Bem intuitivo, não é? Eu coloquei duas condições, ano > 2000 AND gênero = 'Ficção'. O filme Matrix não entra porque, apesar de ser ficção, tem ano menor que 2000. O resultado fica da seguinte forma:



tituloPort	genero	ano
V de Vingança	Ficção	2005
Star Wars: Episódio III - A Vingança dos Sith	Ficção	2005

Legal, estamos avançando! Agora, eu quero outra consulta, outra query. Quero os filmes que sejam ou de ficção, ou que sejam do tipo B (Blu-ray), e quero todos os campos. Vamos ver como fica a query:

```
SELECT * FROM DVD WHERE genero = 'Ficção' OR tipo = 'B';
```



Vamos ver como fica nosso resultado:

```
9 • |SELECT * FROM DVD WHERE genero = 'Ficção' OR tipo = 'B';
```

Filter:	idDVD	tituloPort	tituloIng	tipo	genero	ano
	1	Matrix	The Matrix	D	Ficção	1999
	3	O Senhor dos Anéis: A Sociedade do Anel	The Lord of the Rings: The Fellowship of the Ring	B	Aventura	2001
	4	V de Vingança	V for Vendetta	D	Ficção	2005
	5	Star Wars: Episódio III - A Vingança dos Sith	Star Wars: Episode III - Revenge of the Sith	B	Ficção	2005
*	NULL	NULL	NULL	NULL	NULL	NULL

Temos nesse resultado filmes que obedecem a pelo menos uma das condições, ou seja, eles podem ser tanto do gênero ficção, como do tipo B, ou mesmo as duas coisas ao mesmo tempo (caso do filme Star Wars).

Vamos agora testar o operador NOT. Ele é uma negação, pode ser usado no seguinte contexto: imagine que você quer recuperar todos os filmes que não são do gênero aventura. Temos duas opções, que retornam as mesmas tuplas:

```
SELECT * FROM DVD WHERE genero <> 'Aventura';
```

```
SELECT * FROM DVD WHERE NOT genero = 'Aventura';
```

No primeiro comando temos o operador diferente (<>), ou seja, queremos tuplas com o gênero diferente de Aventura. Na segunda linha temos o operador NOT antes da expressão, ou seja, estamos negando a expressão. Aquilo que é não igual, é o que? Diferente, correto? Então os dois comandos se equivalem. Mas depois veremos que existem outras aplicações para o NOT.

### 3.4.1. IN, BETWEEN, LIKE

Vamos agora conhecer mais três operadores que podem ser usados nos nossos predicados. Primeiro, o operador IN. Já vimos ele sendo usado lá no CHECK. Ele serve para indicar um dos valores de uma lista deles. Por exemplo, quero descobrir o código dos atores que atuaram nos filmes

Matrix e V de vingança. Preciso primeiro descobrir qual o código desses filmes.

```
SELECT * FROM DVD WHERE tituloPort = 'Matrix' OR tituloPort = 'V de Vingança';
```

Temos o seguinte resultado:

```
14
15 • SELECT * FROM DVD WHERE tituloPort = 'Matrix' OR tituloPort = 'V de Vingança';
```

	idDVD	tituloPort	tituloIng	tipo	genero	ano
▶	1	Matrix	The Matrix	D	Ficção	1999
	4	V de Vingança	V for Vendetta	D	Ficção	2005
*	NULL	NULL	NULL	NULL	NULL	NULL

Então, sabendo que os códigos desses filmes são 1 e 4, faço a consulta na tabela que relaciona DVDs e Atores:

```
SELECT idAtor FROM AtorParticipa WHERE idDVD IN (1,4);
```

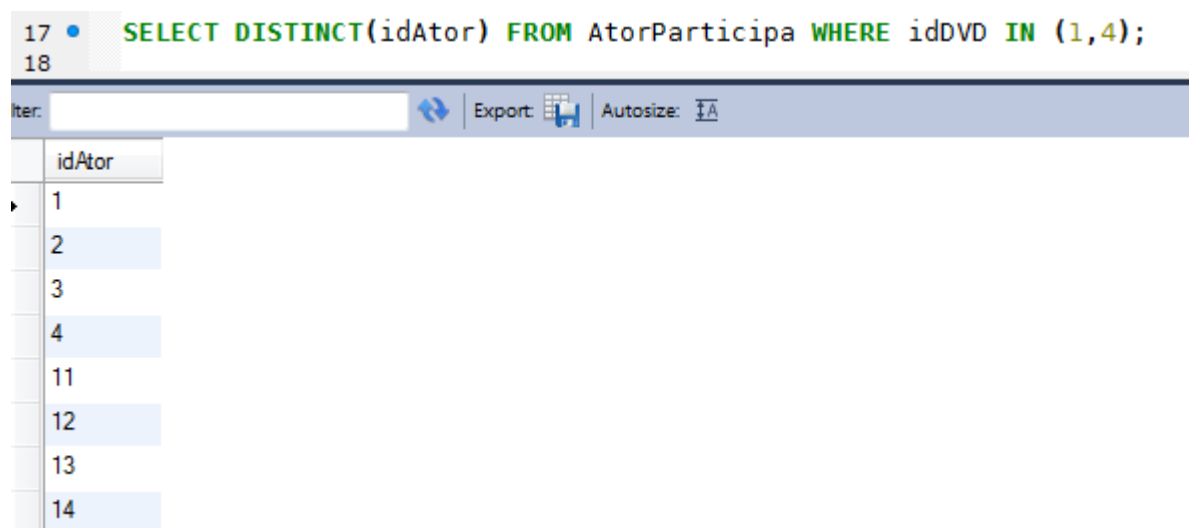
```
17 • SELECT idAtor FROM AtorParticipa WHERE idDVD IN (1,4);
18
```

	idAtor
▶	1
	2
	3
	4
	4
	11
	12
	13
	14

Pronto, temos o idAtor dos atores que participam dos filmes 1 ou 4. Usamos o IN para selecionar os IdDVD que queremos encontrar nessa tabela. Note, que o idAtor 4 aparece duas vezes, pois o mesmo participa dos dois filmes. Existe uma forma de aparecer só uma vez cada código, ou seja, de tirar os códigos repetidos? Sim, temos outra cláusula, na parte da projeção, que é o **DISTINCT**. Ele faz com que elementos

repetidos apareçam só uma vez. Vamos ver como fica nosso comando reescrito, usando o DISTINCT:

```
SELECT DISTINCT(idAtor) FROM AtorParticipa WHERE idDVD IN (1,4);
```



The screenshot shows a SQL query editor with the following query:

```
17 • SELECT DISTINCT(idAtor) FROM AtorParticipa WHERE idDVD IN (1,4);  
18
```

Below the query, there is a table with the following data:

idAtor
1
2
3
4
11
12
13
14

Perceberam que o idAtor 4 aparece agora apenas uma vez? Isso foi por causa do DISTINCT.





Vamos ao operador BETWEEN. Ele checa um intervalo de valores no predicado, sendo verdadeiro quando o valor testado está nesse intervalo. Suponha que eu quero encontrar os filmes lançados entre os anos de 1990 e 2001. Posso escrever a seguinte query:


```
SELECT * FROM DVD WHERE ano BETWEEN 1990 AND 2001;
```


Como resultado tenho os filmes que obedecem a esse critério, ou seja, lançados nesse período. Observem dois detalhes. Primeiro, o operador BETWEEN é usado da seguinte forma: BETWEEN limite\_inferior AND limite\_superior. Segundo, vejam que esses limites são incluídos no teste, ou seja, quero filmes entre 1990 e 2001, inclusive. Vamos ver o resultado:

19 • `SELECT * FROM DVD WHERE ano BETWEEN 1990 AND 2001;`

Filter:

 Edit:   

Export: 

Autosize: 

	idDVD	tituloPort	tituloIng	tipo	genero	ano
▶	1	Matrix	The Matrix	D	Ficção	1999
	3	O Senhor dos Anéis: A Sociedade do Anel	The Lord of the Rings: The Fellowship of the Ring	B	Aventura	2001
*	NULL	NULL	NULL	NULL	NULL	NULL

Agora vamos ao operador LIKE. Ele serve para comparar strings. Funciona da seguinte forma: Imaginem que eu desejo obter da minha tabela DVD os filmes cujos campos tituloPort possuam a palavra Vingança. Com o operador igual eu não consigo fazer essa comparação. Para isso surgiu o operador LIKE. A sintaxe para essa query é a seguinte:

```
SELECT * FROM DVD WHERE tituloPort LIKE '%Vingança%';
```

Vamos analisar o nosso LIKE. Depois dele, veio uma string com o seguinte formato: %Vingança%. O que significa esse % afinal? Ele é o que chamamos de curinga (wildcard). Significa que estou procurando filmes cujo título têm, em qualquer posição, a string Vingança. E se eu quiser os filmes que começam com a string Vingança? Ai não usaria o curinga no começo, só no final, assim: Vingança%. E se eu quiser filmes que terminam com a string Vingança? Bem, meu curinga seria apenas no inicio, dessa forma: %Vingança. Dessa forma, o nosso % substitui de 0 a N caracteres. Revisando:

%Vingança%: filmes que tem a string Vingança em qualquer parte do campo.





Vingança%: filmes que começam com a string Vingança.


%Vingança: filmes que terminam com a string Vingança.


Vamos ver como ficou a execução da nossa query?

21 • `SELECT * FROM DVD WHERE tituloPort LIKE '%Vingança%';`

ter:

 Edit:   

Export: 

Autosize: 

	idDVD	tituloPort	tituloIng	tipo	genero	ano
	4	V de Vingança	V for Vendetta	D	Ficção	2005
	5	Star Wars: Episódio III - A Vingança dos Sith	Star Wars: Episode III - Revenge of the Sith	B	Ficção	2005
*	NULL	NULL	NULL	NULL	NULL	NULL

Pronto, temos dois filmes em que a string Vingança aparece em alguma posição do título em português. Testem as outras variações e vejam o resultado.





Amigos, todos esses operadores podem ser combinados, formando consultas das mais simples às mais complexas. Tudo depende da criatividade de quem escreve, e da necessidade de informação. O importante para vocês resolverem a prova é entender o que faz cada operador, como age, e ver se as consultas estão escritas corretamente. A melhor maneira de aprender isso é praticando, mas é claro que até a prova não dá para dominar totalmente todos esses comandos e operadores do SQL. Olhem esse exemplo abaixo, que eu uso alguns desses operadores combinados:


```
SELECT * FROM DVD WHERE (tituloPort BETWEEN 'A' AND 'T') AND (genero LIKE 'F%');
```


Eu tenho duas condições, ligadas por um AND. Assim, somente filmes que atendem às duas condições serão retornados. A primeira é um uso do BETWEEN não com números, mas com literais. Ela vai retornar todos os filmes que estão entre A e T, ou seja, que começam da letra A em diante, até a letra T. Depois vem o AND e a segunda condição, quero filmes que o campo gênero comece com F. Vamos ver o resultado desse pagode?

23 • **SELECT \* FROM DVD WHERE (tituloPort BETWEEN 'A' AND 'T') AND (genero LIKE 'F%');**

Filter:

Edit:    

Export: 

Autosize: 

	idDVD	tituloPort	tituloIng	tipo	genero	ano
▶	1	Matrix	The Matrix	D	Ficção	1999
	5	Star Wars: Episódio III - A Vingança dos Sith	Star Wars: Episode III - Revenge of the Sith	B	Ficção	2005
*	NULL	NULL	NULL	NULL	NULL	NULL

Podem ver que as tuplas retornadas obedecem às duas condições, o títuloPort está entre A e T, e o gênero começa com a letra F.

### 3.4.2. ORDER BY

Agora vamos estudar uma cláusula que não faz nenhum filtro no resultado, ou seja, não faz parte do predicado, mas serve para ordenar nossa resposta. A resposta que uma query recebe o nome de ResultSet, que é aquela planilhinha que o SGBD retorna às nossas consultas.

Bem, o ORDER BY é usado depois do predicado, e serve para ordenar nossa resposta segundo algum critério. Por exemplo, se eu quero todos os filmes com ano de lançamento maior que 2000, mas quero minha resposta ordenada pelo título em português, uso o seguinte comando:

```
SELECT * FROM DVD WHERE ano > 2000 ORDER BY tituloPort;
```

Vejam que o ORDER BY não altera as tuplas que serão retornadas, altera apenas a ordem de apresentação delas, o resultado segue abaixo:

27 • `SELECT * FROM DVD WHERE ano > 2000 ORDER BY tituloPort;`

idDVD	tituloPort	tituloIng	tipo	genero	ano
2	Constantine	Constantine	D	Aventura	2005
3	O Senhor dos Anéis: A Sociedade do Anel	The Lord of the Rings: The Fellowship of the Ring	B	Aventura	2001
5	Star Wars: Episódio III - A Vingança dos Sith	Star Wars: Episode III - Revenge of the Sith	B	Ficção	2005
4	V de Vingança	V for Vendetta	D	Ficção	2005
*	NULL	NULL	NULL	NULL	NULL

Observem que os filmes retornados estão agora em ordem alfabética, considerando o campo tituloPort. Eu posso ordenar por mais de um campo, basta separar os campos por vírgula.

Por fim, a ordenação por padrão é na ordem ascendente (do menor para o maior), mas posso ordenar também na ordem descendente. Basta colocar a palavra chave DESC. Vamos ver como funciona.

```
SELECT DISTINCT(idAtor) FROM AtorParticipa  
WHERE idDVD IN (1,4) ORDER BY idAtor DESC;
```

```
29 • SELECT DISTINCT(idAtor) FROM AtorParticipa WHERE idDVD IN (1,4) ORDER BY idAtor DESC;
```

idAtor
14
13
12
11
4
3
2
1

Peguei uma query que já tínhamos usado, e coloquei no final um ORDER BY idAtor DESC. Isso fez com que os idAtor fossem retornados do maior para o menor.

Quando escolhemos mais de um campo para ordenar, podemos definir para cada um se a ordem é ascendente (ASC) ou descendente (DESC). Se eu não informar nada, é considerada a ordem ascendente. Olhem esse exemplo:

```
SELECT * FROM DVD ORDER BY genero DESC, tituloPort ASC;
```

Aqui nos mostramos todos os filmes (não fiz a operação de seleção, ou seja, não tem WHERE) ordenando primeiro pelo campo gênero em ordem descente, depois por tituloPort, em ordem ascendente. Como vocês devem ter notado, o ASC não precisa ser declarado, pois ele é o padrão, ou como dizemos em Computação, é o default.

### 3.4.3. FUNÇÕES

Cansados??? A brincadeira ainda está no início ainda. Vamos agora a um grupo de cláusulas muito usadas nos SELECT. Vejam que só o assunto SELECT dá um curso completo. Vamos ver o que é mais comum, mas nem de perto dá para ver tudo.

Temos algumas funções que ajudam a extrair informações das tabelas. A primeira que veremos é o COUNT(). Ela serve para contar quantas tuplas são retornadas em uma query.

Primeiro caso: quantidade de tuplas de uma tabela.

Podemos contar quantas tuplas tem uma tabela com o seguinte comando:

```
SELECT COUNT(*) FROM DVD;
```

Nesse caso, não coloquei nenhum predicado, e vai ser retornada a quantidade de linhas (registros) que a tabela tem, que é 5.

Segundo caso: quantidade de tuplas de uma seleção.

Agora vamos usar o COUNT par determinar quantas tuplas de uma tabela obedecem a uma determinada condição:

```
SELECT COUNT(*) FROM DVD WHERE ano = 2005;
```

Temos agora como resultado o valor 3, que é o número de filmes lançados no ano de 2005.

Vamos usar mais um exemplo, agora com uma tabela maior? Vou usar a tabela Country, do esquema World, que está disponível no MySQL como exemplo. Quantas linhas têm essa tabela?

```
SELECT COUNT(*) NumeroDeLinhas FROM WORLD.COUNTRY;
```

Pronto, lembrem-se que nós estamos usando o esquema DBemprestimo, que é o nosso Banco de Dados. Para fazer uma query em uma tabela de outro esquema, basta colocar o nome do esquema na frente. Assim tenho nessa query a quantidade de tuplas que a tabela Country do esquema World possui. Aproveitei para mostrar mais um segredinho do SQL. O que é aquele NumeroDeLinhas que tem depois do COUNT(\*). É simplesmente um apelido (alias) que usamos para ser exibido como nome do campo na resposta. Vejam como fica:



```
35 • SELECT COUNT(*) NumeroDeLinhas FROM WORLD.COUNTRY;
```

Filter:	Export:	Autosize:
NumeroDeLinhas		
▶ 239		

Nossa próxima função é o SUM(). Serve para somar os valores de determinada coluna. Para exemplificar seu uso, devemos ter um campo numérico, que faça sentido somar. Temos na tabela Country um campo chamado population, que tem o número de habitantes de um país, e temos também o campo continent. Vou somar as populações dos países que estão na Ásia:

```
SELECT SUM(population) SOMA FROM WORLD.COUNTRY WHERE continent = 'Asia';
```

É somente isso que o SUM() faz. Vamos ver o resultado:

```
39 • SELECT SUM(population) SOMA FROM WORLD.COUNTRY WHERE continent = 'Asia';
```

Filter:	Export:	Autosize:
SOMA		
▶ 3705025700		

Nossa próxima função é o AVG(). Ele tira a média aritmética de um conjunto de valores. Vamos usar no mesmo contexto que usamos o SUM, mas ao invés da soma da população dos países da Ásia, vamos encontrar a população média:

```
SELECT AVG(population) MEDIA FROM WORLD.COUNTRY WHERE continent = 'Asia';
```

Temos o seguinte resultado:

```
41 • SELECT AVG(population) MEDIA FROM WORLD.COUNTRY WHERE continent = 'Asia';
```

Filter:	Export:	Autosize:
MEDIA		
▶ 72647562.7451		

Vamos agora à função MAX(). Ela retorna o maior dos valores. Vamos descobrir qual a maior população que um país da Europa tem:

```
SELECT MAX(population) FROM WORLD.COUNTRY WHERE continent = 'Europe';
```

Como resultado temos o maior valor que a query encontrar para o campo population, entre as tuplas selecionadas (países da Europa). Assim como temos o máximo, podemos obter também o mínimo, com a função MIN(). Criei uma query que mostra os dois valores logo de uma vez.

```
SELECT MAX(population) MAIOR, MIN(population) MENOR FROM WORLD.COUNTRY WHERE continent = 'Europe';
```

Vamos ver o resultado:

```
45 • SELECT MAX(population) MAIOR, MIN(population) MENOR FROM WORLD.COUNTRY
46 WHERE continent = 'Europe';
```

er:	Export:	Autosize:
MAIOR	MENOR	
146934000	1000	

Mais uma função. Bem fácil. NOW() retorna o dia e hora atuais. Executem o seguinte comando:

```
SELECT NOW();
```

Vejam o resultado. Mas professor, pode fazer isso, um SELECT sem FROM? Pode sim, se fizermos SELECT sem FROM simplesmente o SGBD resolve as funções ou operações (aritméticas, por exemplo). Vamos ver o que acontece quando executo o seguinte SELECT:

```
SELECT now() DATA_HORA, (150 * 6) Multiplicação, (1975 - 520) Subtração;
```

```
48 • SELECT now() DATA_HORA, (150 * 6) Multiplicação, (1975 - 520) Subtração;
```

Filter:	Export:	Autosize:
DATA_HORA	Multiplicação	Subtração
2012-05-27 10:22:24	900	1455

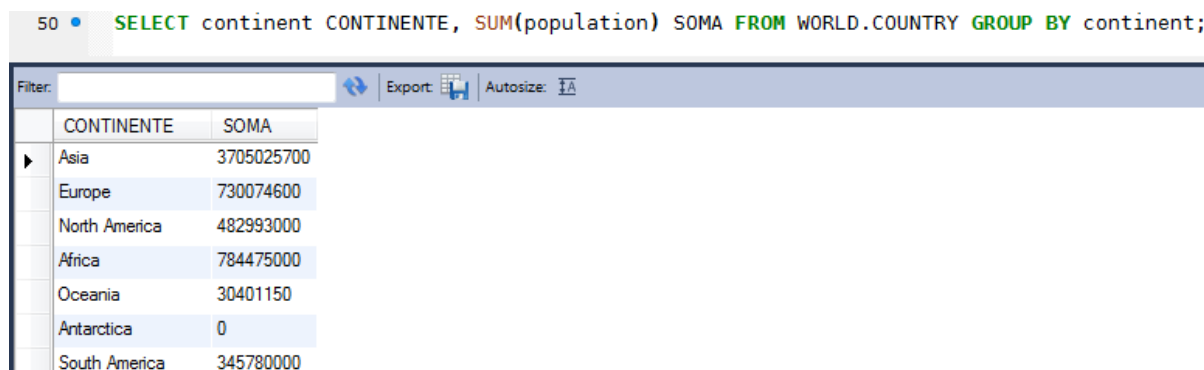
### 3.4.4. AGRUPADORES

Agora vamos ver uma cláusula muito importante, que é usada geralmente em conjunto com as nossas funções. É a cláusula GROUP BY. Ela funciona como um agrupador. Vamos logo ver um exemplo, que fica mais fácil de entender.

Imaginem que eu quero agora saber a soma da população de cada continente. Posso fazer isso da seguinte forma:

```
SELECT continent CONTINENTE, SUM(population) SOMA  
FROM WORLD.COUNTRY GROUP BY continent;
```

Olha que legal, esse comando agrupa as tuplas pelo campo continent (GROUP BY), e faz a soma da população de cada continente. Vejam o resultado:



CONTINENTE	SOMA
Asia	3705025700
Europe	730074600
North America	482993000
Africa	784475000
Oceania	30401150
Antarctica	0
South America	345780000

Beleza, agora quero refinar minha query. Quero fazer a mesma operação, mas agora só me interessam continentes que a soma da população seja maior que zero. Como faço isso? Bem, sempre que usamos o GROUP BY e queremos fazer um teste com nossa função que conta, soma, calcula média etc etc, usamos a cláusula HAVING, da seguinte forma:

```
SELECT continent CONTINENTE, SUM(population) SOMA  
FROM WORLD.COUNTRY GROUP BY continent  
HAVING SUM(population) > 0;
```



Executem esse comando e vejam que o resultado é quase igual ao anterior, mas sem o Antarctica (nem Brahma, nem SKOL, muito menos Kaiser), pois esse continente não passou na condição do HAVING.

Vamos a mais um exemplo, voltando ao nosso esquema. Quero agora a quantidade de filmes, em cada gênero, mas só quero saber dos filmes no século XXI. Como fica essa query?

```
SELECT genero, COUNT(*) FROM DVD WHERE ano > 2000
GROUP BY genero ORDER BY 1;
```

Nessa query primeiro seleciono as tuplas que tem ano maior que 2000. Depois agrupo por gênero e conto (COUNT). No final, mostro o resultado ordenado pelo primeiro campo (por isso o 1, mas poderia também colocar o nome do campo, como fizemos antes). Vamos ver o resultado:

55 **SELECT** genero, **COUNT(\*)** **FROM** DVD **WHERE** ano > 2000  
56 **GROUP BY** genero **ORDER BY** 1;

Filter:  Export:  Autosize: 

	genero	COUNT(*)
▶	Aventura	2
	Ficção	2

Mas vocês podem me perguntar, quando eu uso a condição no WHERE e quando eu uso a condição no HAVING. De repente você pode usar até nos dois, mas a regra é o seguinte. Sempre que eu me referir a uma função dessas que contam, somam, tiram média etc, eu uso a condição que faz referência a essa função no HAVING. Já as condições que dizem respeito à seleção das tuplas que entrarão no meu agrupamento, eu uso no WHERE. Olhem essa query, que usa as duas coisas ao mesmo tempo.

```
SELECT continent CONTINENTE, SUM(population) SOMA
FROM WORLD.COUNTRY
WHERE GovernmentForm like '%Republic%'
```

GROUP BY continent

HAVING SUM(population) > 0;

Agora estou agrupando os países por continente, mas só quero países que a forma de governo tem a string Republic na sua descrição. Depois eu somo suas populações e só considero os resultados que a soma da população é maior que zero. Digam a verdade: é lindo ou não esse SQL?

```
58 • SELECT continent CONTINENTE, SUM(population) SOMA FROM WORLD.COUNTRY
59 WHERE GovernmentForm like '%Republic%'
60 GROUP BY continent HAVING SUM(population) > 0;
```

ter:		Export	Autosize
CONTINENTE	SOMA		
Asia	3388296700		
Europe	578424000		
North America	442621000		
Africa	746211000		
Oceania	1304000		
South America	345597000		

Amigos, tudo isso que aprendemos até agora pode ser usado junto. Agrupadores, funções, predicados com AND, OR, NOT e por ai vai. As possibilidades são imensas.

Vamos a alguns exercícios para distrair:

**8. (ESAF/Analista Infraestrutura/CVM 2010) Assinale a opção correta.**

**a) A linguagem de definição de dados permite expressar as consultas e atualizações do banco de dados.**

**b) A linguagem de manipulação de dados permite a especificação do esquema do banco de dados.**

**c) A linguagem de manutenção de dados permite expressar as consultas e atualizações do banco de dados.**

**d) A linguagem de definição de dados permite a especificação do esquema do banco de dados.**

**e) A linguagem de manipulação de consultas permite a atualizações do banco de dados.**

Comentários:

a) DDL é utilizada para definir o Banco de Dados. As consultas e atualizações são efetuadas pela DML.

b) Inverteu novamente os conceitos. DDL é que especifica o esquema do BD.

c) Linguagem de manutenção de dados não existe, o que existe é linguagem de manipulação de dados.

d) Correto, conforme já comentei, a DDL especifica o esquema do Banco de Dados. Exemplo de comandos de DDL são Create Table e Drop Table.

e) Linguagem de manipulação de consultas não é um termo utilizado em BD. O que ainda pode ser aceito é Linguagem de Recuperação de Dados (DRL). E ela não permite atualizações no Banco de Dados.

**Gabarito: Letra d**

### **9. (ESAF/APOTI/MPOG 2010) Em uma SQL**

**a) a Linguagem de Manipulação de Relacionamentos compreende os comandos para construir tabelas em um banco de dados.**

**b) a Linguagem de Definição de Dados fornece tabelas para criação e modificação de comandos.**

**c) os comandos básicos da Linguagem de Definição de Dados são Select, Insert, Update e Delete.**

**d) a Linguagem de Manipulação de Dados compreende os comandos para inserir, remover e modificar informações em um banco de dados.**

**e) os comandos básicos da Linguagem de Definição de Dados são Sort, Insert, Undo e Store.**

Comentários:

a) A ESAF é bem criativa, não é? Agora colocou ai linguagem de manipulação de relacionamentos. A construção de tabelas no Banco de Dados é feita pela DDL. Item incorreto.

b) Inverteu os conceitos, a DDL não fornece tabelas para a construção de comandos, fornece comandos para a construção de tabelas.

c) Os comandos Select, Insert, Update e Delete são os comandos básicos de DML, e não de DDL. Afinal de contas, eles servem para manipular os dados, e não para criar a estrutura do BD. Item Incorreto.

d) Exatamente como já vimos. Item Correto.

e) Entre os principais comandos da DDL estão Create Table, Create View, Create Index, Alter Table, Alter Index, Drop View etc etc. Item incorreto.

**Gabarito: Letra d**

**10. (ESAF/PSS TI 2008) Na linguagem SQL, o uso da cláusula WHERE, quando apropriado,**

**a) agrupa o resultado em subconjuntos que possuem valores correspondentes em uma ou mais colunas. Em cada grupo não há duas linhas com o mesmo valor na(s) coluna(s) de agrupamento.**

**b) permite selecionar linhas baseadas em uma expressão booleana. Somente as linhas para as quais a expressão é avaliada como TRUE são retornadas no resultado.**

**c) especifica se o ResultSet de uma instrução SELECT simples, que atende aos requisitos para um cursor, é atualizável ou não.**

**d) permite selecionar colunas baseado em uma expressão numérica. Somente as colunas para as quais a expressão é avaliada como TRUE são retornadas no resultado. O uso desta cláusula obriga que os valores nulos sejam considerados diferentes nos agrupamentos.**

**e) cria uma ou mais linhas e as armazena na tabela especificada. O número de valores especificados na instrução WHERE deve ser idêntico ao número de colunas especificadas ou implícitas.**

Comentários:

a) Quem faz agrupamento é o group by. Item incorreto.

b) Exatamente, o where é seguido por uma expressão lógica (booleana) que é o critério de seleção das tuplas em uma tabela. É a operação de seleção em álgebra relacional. Dizer que a operação é booleana é simplesmente dizer que ela pode ser avaliada como verdadeiro ou falso.

c) O ResultSet é tudo aquilo que é retornado de uma consulta. Um cursor é uma referência a determinada consulta, assunto que não veremos. Por fim, o where não determina se os dados retornados são ou não atualizáveis.

d) Quem seleciona as consultas é o Select (operação de projeção), e para essa seleção não é necessária a especificação de uma expressão lógica. Item totalmente incorreto.

e) Where não cria linhas, apenas determina o critério de seleção das mesmas.

**Gabarito: Letra b**



**11. (ESAF/Auditor Informática/Pref Natal 2008) Quanto à estrutura, propriedades e sintaxe da linguagem SQL, é correto afirmar que**

**a) o CREATE e o DROP são comandos básicos da DDL ( Data Definition Language - Linguagem de Definição de Dados ).**

**b) o SELECT e o INSERT são comandos básicos da DDL ( Data Definition Language - Linguagem de Definição de Dados ).**

**c) o CREATE e o DROP são comandos básicos da DCL ( Data Control Language - Linguagem de Controle de Dados ).**

**d) o GRANT e o REVOKE são comandos básicos da DDL ( Data Manipulation Language - Linguagem de Manipulação de Dados ).**

**e) o SELECT e o INSERT são comandos básicos da DCL ( Data Control Language - Linguagem de Controle de Dados ).**

Comentários:

a) Conforme já estudamos, está correto.

b) Select e Insert são da DML. Item incorreto.

c) Conceito novo, DCL (Data Control Language) controla os aspectos de autorização dos dados e níveis de acesso aos objetos de um Banco de Dados. No fundo, é uma especialização da DDL. Seus principais comandos são Grant e Revoke. Veremos isso em outra oportunidade.

d) Como já sabemos, os comandos básicos da DDL são Create, Alter e Drop. Item incorreto.

e) Select e Insert são comandos da DML. Item incorreto.

**Gabarito: Letra a**

**12. (ESAF/AFC/CGU 2008) Em um banco de dados que utiliza a linguagem SQL para definição, manipulação e controle de dados, é correto afirmar que os comandos**

**a) CREATE, DROP e INSERT fazem parte da DML (Linguagem de Manipulação de Dados).**

**b) GRANT e REVOKE fazem parte da DCL (Linguagem de Controle de Dados).**

**c) INSERT, UPDATE e SELECT fazem parte da DDL (Linguagem de Definição de Dados).**

**d) ALTER, DELETE e REVOKE fazem parte da DML (Linguagem de Manipulação de Dados).**

**e) CREATE, GRANT e DROP fazem parte da DCL (Linguagem de Controle de Dados).**

Comentários:

a) Create e Drop fazem parte da DDL, e Insert da DML. Item incorreto.

b) Item correto. Temos na DCL o controle de acesso dos usuários aos objetos. Isso é feito pelos comandos Grant (que dá uma permissão) e Rvoke (revoga uma permissão).

c) Fazem parte da DML.

d) Entre os citados, só Delete faz parte da DML.

e) Entre os citados, só Grant faz parte da DCL. Create e Drop fazem parte da DDL.

**Gabarito: Letra b**

**13. (ESAF/Analista de TI/SEFAZ-CE 2007) No SQL, considerando o uso das cláusulas SELECT, GROUP BY e HAVING, indique a opção que apresenta uma sintaxe correta.**

**a) SELECT Ano, COUNT(\*) AS 'Total' FROM Matricula WHERE Cod = 171 GROUP BY Ano HAVING count(\*) > 5**

**b) SELECT Ano, COUNT(\*) AS 'Total' FROM Matricula HAVING count(\*) > 5 WHERE Cod = 171 GROUP BY Ano**

**c) SELECT Ano, COUNT(\*) AS 'Total' WHERE Cod = 171 FROM Matricula GROUP BY Ano HAVING count(\*) > 5**

**d) SELECT Ano, COUNT(\*) AS 'Total' WHERE Cod = 171 GROUP BY Ano HAVING count(\*) > 5 FROM Matricula**

**e) SELECT Ano WHERE Cod = 171 GROUP BY Ano HAVING count(\*) > 5 COUNT(\*) AS 'Total' FROM Matricula**

#### Comentários

Para resolvermos essa questão, devemos conhecer a sintaxe do Select com o uso do Group By e Having. Funciona assim:

Select <campos/expressões> from <tabelas/subquery> where <condição> group by <campos de agrupamento> Having <condição>

Agora, podemos facilmente identificar que a letra (a) está com a sintaxe correta, e as demais estão com cláusulas na ordem incorreta. Vejam uma pequena diferença daquilo que estamos acostumado, com o uso do AS antes da definição do apelido (ALIAS) do campo. Esse uso também é possível no SQL.

#### **Gabarito: Letra a**

**14. (ESAF/Analista Técnico/SUSEP 2006) Analise as seguintes afirmações relacionadas a conceitos básicos sobre Banco de Dados.**

**I. O comando SQL responsável por fechar uma transação confirmando as operações feitas é o INSERT.**

**II. O comando SQL responsável por fechar uma transação e desfazer todas as operações é o COMMIT.**

**III. Quando uma transação ainda está aberta para um usuário, enquanto não é executado um comando COMMIT, o**

**próprio usuário pode ver as suas alterações, mas outros usuários não podem vê-las.**

**IV. Uma transação assegura um espaço de trabalho que contém várias alterações, inclusões e exclusões de dados em uma ou mais tabelas, com a possibilidade de confirmação ou cancelamento das operações sem comprometimento dos dados.**

**Indique a opção que contenha todas as afirmações verdadeiras.**

- a) I e II**
- b) II e III**
- c) III e IV**
- d) I e III**
- e) II e IV**

Comentários:

I. O comando que fecha a transação e confirma os valores é o Commit. Item incorreto. Lembram que eu falei do COMMIT? Uma transação é simplesmente um conjunto de operações de DML que devem ser executadas juntas. É a máxima dos três mosqueteiros. Ou todas as operações são feitas juntas e confirmadas por um COMMIT, ou tudo é desfeito por um RollBack.

II. O Commit não desfaz as operações. Quem desfaz é o RollBack. Item incorreto.

III. Exato, as alterações não são propagadas até que o Commit feche a transação, mas se o usuário que fez uma atualização, por exemplo, antes do Commit executar um Select, vai ver os dados já alterados. Os demais usuários não vêem a alteração.

IV. É uma boa definição para transação, correta em todos os sentidos.

**Gabarito: Letra c**

**15. (ESAF/AFC/CGU 2006) Um procedimento armazenado (stored procedure) é uma coleção de comandos em SQL que**

**a) provoca um aumento no tráfego na rede e reduz a performance do sistema, mas continua sendo largamente utilizado para criar mecanismos de segurança em bancos de dados relacionais.**

**b) encapsula tarefas repetitivas, aceita parâmetros de entrada e pode retornar um valor de status para indicar sucesso ou falha na execução.**

**c) estão armazenados no banco de dados e que são executadas diretamente na máquina do usuário.**

**d) estão armazenados na máquina do usuário e que são executadas diretamente no servidor do banco de dados.**

**e) são utilizados unicamente para autenticar um usuário, dando a ele direitos de acesso a escrita/alteração em tabelas do banco de dados.**

Comentários:

a) Não se pode dizer que as stored procedures aumentam o tráfego na rede e reduzem a performance do sistema. Na verdade, o que acontece normalmente é exatamente o contrário. Por estarem armazenadas no próprio Banco de Dados, os códigos executados pelas Stored Procedure evitam que esse dados brutos percorram a rede. A performance pode até ser melhorada, se esses dados forem processados na próprio servidor do SGBD ao invés de serem transportados a um servidor de aplicação para processamento. O grande problema de se fazer isso em larga escala é que as regras de negócio do seu sistema ficariam presas ao Banco de Dados, indo de encontro ao projeto em camadas, e fazendo sua aplicação mais dependente daquela plataforma. Por exemplo,

imaginem se toda a regra de negócio de sua aplicação estiver armazenada em Stored Procedures de uma SGBD SQL Server, por exemplo, e sua empresa decide que agora vai utilizar somente o MySQL. O que acontece? As procedures deverão ser reescritas para rodar no outro banco de dados, já que muitas vezes as implementações de SQL são bem diferentes. Então, esse item está incorreto.

b) Exatamente, as Stored Procedures podem encapsular tarefas repetitivas. Elas podem aceitar parâmetros de entradas e devolver o status de sua execução (ou seja, se executaram com sucesso ou não).

c) Estão armazenadas no Banco de Dados e são executadas no próprio servidor do SGBD, e não na máquina do usuário.

d) Como já vimos, item incorreto.

e) Podem ser usadas para autenticar usuários, mas não somente para isso. Podem executar em princípio qualquer comando SQL, e executar uma infinidade de funções.

**Gabarito: Letra b**

## 4. SUBQUERIES

Às vezes, para conseguirmos obter o resultado de uma query, precisamos de um valor que vem de outra query. Assim, o SQL permite que definamos subqueries, que nada mais são queries dentro de outras queries. Vamos a um exemplo.

Lembram que procuramos o id de todos os atores que participam dos filmes Matrix e V de Vingança. Para isso, usamos duas queries:

```
SELECT * FROM DVD WHERE tituloPort = 'Matrix' OR tituloPort = 'V de Vingança';
```

```
SELECT DISTINCT(idAtor) FROM AtorParticipa WHERE idDVD IN (1,4);
```

Existe uma forma de eu juntar essas duas operações em uma query só, utilizando subqueries. Fica assim:

```
SELECT DISTINCT(idAtor) FROM AtorParticipa  
WHERE idDVD IN  
(SELECT idDVD FROM DVD WHERE tituloPort = 'Matrix'  
OR tituloPort = 'V de Vingança');
```

Essa query é resolvida da seguinte forma: Primeiro é executada a query que está dentro dos parênteses (em vermelho). Essa query retorna os IdDVD dos filmes Matrix e V de Vingança (os valores 1 e 4). Esse resultado é usado pela query principal (em verde), que fica idêntica à query anterior, ou seja, retorna de forma distinta (única) os id dos atores que participam dos filmes citados.

Vamos a outro exemplo. Quero agora os filmes que o ator KEANU REEVES participa. Não sei o idAtor dele, e quero fazer só uma query. Fica assim:

```
SELECT * FROM DVD  
WHERE idDVD IN  
(SELECT idDVD FROM AtorParticipa WHERE idAtor =  
(SELECT idATOR FROM Ator WHERE nomeATOR =  
'KEANU REAVES'));
```

Olhem que coisa linda mais linda, mais cheia de graça. Tenho uma query principal com uma subquery. Esta subquery também tem uma subquery. Como isso é resolvido?

A primeira query resolvida é aquela que está mais interna nos parênteses (SELECT idATOR FROM Ator WHERE nomeATOR = 'KEANU REAVES'). O seu resultado é usado um nível acima, em outra query (SELECT idDVD FROM AtorParticipa WHERE idAtor = <resultado subquery>). Por fim, é executada a query fora dos parênteses (SELECT \* FROM DVD WHERE idDVD IN <resultado subquery>).

```
67 SELECT * FROM DVD
68 WHERE idDVD IN (SELECT idDVD FROM AtorParticipa WHERE idAtor =
69 (SELECT idATOR FROM Ator WHERE nomeATOR = 'KEANU REAVES'));
70
```

idDVD	tituloPort	tituloIng	tipo	genero	ano
1	Matrix	The Matrix	D	Ficção	1999
2	Constantine	Constantine	D	Aventura	2005
*	NULL	NULL	NULL	NULL	NULL

Amigos, repito mais uma vez, as possibilidades são imensas. Vamos ver o máximo que for possível, e torcer por questões dentro desse conteúdo. Um curso completo de SQL pode durar um semestre, estamos fazendo o módulo ultra hiper rápido.

## 5. PRODUTO CARTESIANO

Vamos ver agora a forma mais simples de fazer a operação de álgebra relacional Produto Cartesiano. Fizemos algumas consultas que voltaram os Id dos atores. Tudo bem, pelo Id vamos lá na tabela e achamos o nome do Ator. Mas não seria legal ter logo o nome do ator direto? Vamos ver como fazer isso. Quero o nome dos atores que participaram do filme Senhor dos Anéis.

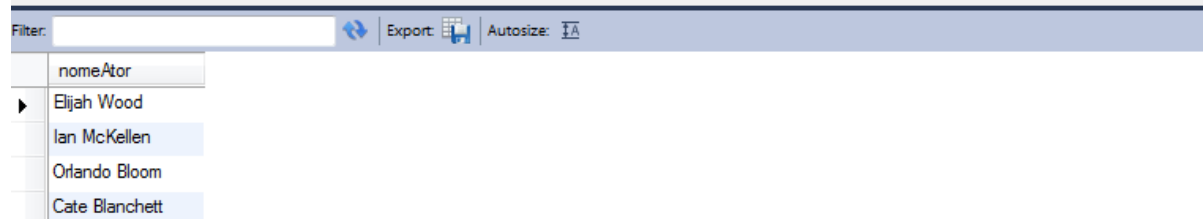
```
SELECT ATOR.nomeAtor FROM ATORPARTICIPA, ATOR
WHERE ATORPARTICIPA.idAtor = ATOR.idAtor AND
ATORPARTICIPA.idDVD =
(SELECT idDVD FROM DVD
WHERE tituloPort LIKE 'O Senhor dos An%');
```

Bem, em primeiro lugar, notem que depois do FROM colocamos o nome de duas tabelas, separadas vírgula. Isso faz o produto cartesiano entre as duas tabelas, ou seja, cada linha da tabela AtorParticipa é juntada a cada linha da tabela Ator. Na primeira parte do predicado dizemos que os campos idAtor das duas tabelas tem que ser iguais, ou seja, tenho que relacionar a chave primária de ator com a chave



extrangeira de AtorParticipa. Depois, determinamos que idDVD da tabela AtorParticipa tem que ser igual ao idDVD do filme procurado (usamos uma subquery). Como resultado, mostramos apenas o nome (Ator.nomeAtor) das tuplas que satisfazem às condições. Vejam o resultado abaixo.

```
71 SELECT ATOR.nomeAtor FROM ATORPARTICIPA, ATOR
72 WHERE ATORPARTICIPA.idAtor = ATOR.idAtor AND
73 ATORPARTICIPA.idDVD = (SELECT idDVD FROM DVD WHERE tituloPort LIKE 'O Senhor dos An%');
74
```



Pronto, agora ai invés de mostrar o id dos atores, mostramos o nome. Isso foi possível pela operação de produto cartesiano entre as tabelas Ator e AtorParticipa. Vejam que quando usamos produto cartesiano, devemos colocar o nome da tabela antes de cada campo, para dizer ao SGBD a qual campo estamos nos referindo. Vamos ver mais alguns exercícios:

**16. (ESAF/AFRF/SRF 2005) Com relação ao uso da SQL na manipulação de dados, caso se queira eliminar linhas repetidas do conjunto resultado, deve-se utilizar a palavra-chave DISTINCT, da seguinte forma:**

- a) **SELECT {colunas} FROM {tabelas} DISTINCT.**
- b) **DISTINCT SELECT {colunas} FROM {tabelas}.**
- c) **SELECT FROM {tabelas} DISTINCT {colunas}.**
- d) **SELECT DISTINCT {colunas} FROM {tabelas}.**
- e) **FROM {tabelas} SELECT DISTINCT {colunas}.**

Comentários:

Para resolver essa questão, da Receita Federal, basta saber a sintaxe do operador DISTINCT. Bem fácil essa, não?

**Gabarito: Letra d**

**17. (CEPS-UFPA/Analista de TI/UFPA 2011) O comando SQL Select é utilizado para**

- a) atualizar um valor de um atributo.**
- b) excluir uma linha de uma tabela.**
- c) calcular um valor de uma expressão aritmética.**
- d) incluir uma linha em uma tabela.**
- e) remover um atributo de uma tabela.**

Comentários

- a) Não, quem faz isso é o Update.
- b) Quem exclui é o Delete
- c) Sim, o comando Select pode ser usado para calcula o valor de uma expressão aritmética, sem consultar tabela nenhuma. Por exemplo, em MySql podemos usar:

Select 130 \* 100 from dual;

O dual é o que se chama de tabela dummy, utilizada quando nenhuma tabela do seu esquema for referenciada. No MySQL, pode ser usada a partir da versão 4.1.0. No SQL Server, utilizaríamos apenas:

Select 130 \* 100;

- d) Quem inclui é o Insert
- e) Para remover um atributo de uma tabela, ou usa Alter ou dropa a tabela e a recria

**Gabarito: Letra c**

**18. (FCC/Analista Judiciário - TI/TRT23 2011) As operações da álgebra relacional Seleção, Projeção e Produto Cartesiano são**

**implementadas na linguagem SQL, respectivamente, pelas cláusulas**

- a) Select, From e Where.**
- b) Select, Where e From.**
- c) Where, Select e From.**
- d) Where, From e Select.**
- e) Select, Select e Join.**

Comentários:

A operação de Seleção é implementada pelo Where.

A operação de Projeção é implementada pelo Select.

A operação de Produto Cartesiano é implementada quando se informa mais de uma tabela após o From.

Cuidado com a pegadinha! O Select não implementa seleção, mas sim a projeção!!!

**Gabarito: Letra c**

**19. (FCC/Analista Judiciário - TI/TRT24 2011) Um wildcard ( curinga ) SQL deve ser usado com um operador LIKE. Assim,**

- a) \_ ( símbolo underline ) é um substituto para zero ou mais caracteres.**
- b) % (símbolo percentual) é um substituto para zero ou mais caracteres.**
- c) % (símbolo percentual) é um substituto para exatamente um caracter.**
- d) [!lista] indica qualquer caracter simples da lista.**
- e) [lista] indica nenhum caracter simples da lista.**

Comentários:

O operador LIKE é utilizado para comparar strings com um determinado padrão. Assim, utilizamos o curinga % para substituir no padrão zero ou mais caracteres. Por exemplo, se quisermos procurar em determinada tabela todas as pessoas que tem MARIA no nome, utilizaríamos algo do tipo:

```
Select * from Tabela where nome like '%MARIA%'
```

Assim, todas as tuplas em que o campo Nome de Tabela tiver a sequência MARIA serão retornadas, como por exemplo: JOSÉ MARIA, MARIANA, ANAMARIA, e assim sucessivamente.

Observem que poderíamos usar somente um operador %, como por exemplo:

```
Select * from Tabela where nome like 'MARIA%' : retornaria todos os nomes que começam por MARIA.
```

```
Select * from Tabela where nome like '%MARIA': retornaria todos os nomes que terminam por MARIA.
```

### **Gabarito: Letra b**

Amigos, nessa aula ficamos por aqui. Precisamos de mais uma aula sobre SQL, e depois entraremos em Mineração de Dados, que é um assunto mais teórico. Talvez tenhamos uma aula a mais. Mas entrego todas as aulas dentro do prazo final do curso, estou me esforçando para isso. Fico aguardando as dúvidas. Essa aula está um pouco pesada, mas o assunto é muito importante, e acredito que testando um pouco os comando aqui apresentados vocês vão entender o SQL de forma mais natural. Nossa parte de Banco de Dados propriamente dita está perto do fim. Que a força esteja com vocês!

Um grande abraço.

Leonardo Lima

leonardolima@estrategiaconcursos.com.br

## 6. RESUMO

Chegamos nessa aula entramos no mundo SQL, percorrendo as duas principais linguagens para o SGBD, DDL e DML.

Com a DDL, criamos nossas estruturas, o nosso esquema, ou seja, criamos um Banco de Dados vazio. Seus principais comandos são:

Create: Usado para criar objetos (TABLE, INDEX, DATABASE, TRIGGER, PROCEDURE etc)

Drop: Usado para excluir os objetos.

Alter: Usado para alterar os objetos.

Entre os objetos importantes, as triggers têm um destaque especial. São trechos de código que são executados automaticamente, quando um evento ocorre em uma tabela ou outro objeto de Banco de Dados. Exemplo de um evento pode ser a alteração de um registro.

Outros objetos importantes são PROCEDURES E FUNCTIONS, que são conjuntos de comandos SQL que resolvem determinado problema, sendo sua principal diferença o fato de que as FUNCTIONS sempre retornam valores.

A DML tem comandos que servem para manipular os dados do Banco de Dados. Basicamente podemos incluir, excluir, alterar e consultar os dados com os comandos da DML.

O comando INSERT serve para inserir um novo registro em uma tabela.

O comando DELETE exclui uma ou mais tuplas de uma tabela, dependendo de uma condição (operação de seleção).

O Comando UPDATE atualiza uma ou mais colunas, afetando uma ou mais linhas de uma vez, dependendo de uma condição (operação de seleção).

O Comando SELECT é a principal ferramenta do SQL, e serve para selecionar uma ou mais linhas, dependendo de uma condição (operação de seleção). Pode retornar, dentro dessa seleção, uma ou mais colunas (operação de projeção). Pode ainda implementar o produto cartesiano, quando colocamos mais de uma tabela na query (após o FROM).

Diversos operadores podem ser usados no SELECT. Entre eles temos o IN, que testa se um campo está contido em uma lista de valores, o BETWEEN, que testa um intervalo de valores, e o LIKE, que testa strings com o uso de curingas.

Diversas funções também são usadas em queries. Entre elas temos o SUM(), que soma os valores de uma coluna, o AVG(), que tira a média, e o COUNT(\*), que conta quantas linhas foram retornadas no ResultSet.

Os agrupadores são usados com as instruções GROUP BY e HAVING. Elas servem para agrupar os resultados de uma consulta, de forma a podermos fazer cálculos em grupos específicos de uma tabela.

As queries admitem subqueries, que são queries dentro de queries, retornando um determinado resultado para ser usado pela query principal. Seu uso possibilita que possamos aninhar diversas consultas, uma dentro da outra, e dessa forma consigamos elaborar consultas mais complexas.

## **7. QUESTÕES APRESENTADAS NESTA AULA**

**1. (ESAF/Analista Sistemas/ANA 2009) Em SQL, a cláusula check aplicada a uma declaração de domínio**

**a) permite especificar um predicado que deve ser satisfeito por qualquer valor atribuído a uma variável de determinado domínio.**

**b) especifica um predicado que deve ser satisfeito por uma tupla em uma relação.**

c) proíbe a inserção de um valor nulo para as variáveis do domínio.

d) verifica se os atributos considerados formam uma chave candidata.

e) não tem efeito, pois não se aplica esta cláusula a declarações de domínio.

**2. (ESAF/Auditor Informática/Pref Natal 2008) Quanto à estrutura, propriedades e sintaxe da linguagem SQL, é correto afirmar que**

a) o CREATE e o DROP são comandos básicos da DDL (Data Definition Language - Linguagem de Definição de Dados).

b) o SELECT e o INSERT são comandos básicos da DDL ( Data Definition Language - Linguagem de Definição de Dados ).

c) o CREATE e o DROP são comandos básicos da DCL ( Data Control Language - Linguagem de Controle de Dados ).

d) o GRANT e o REVOKE são comandos básicos da DDL ( Data Manipulation Language - Linguagem de Manipulação de Dados ).

e) o SELECT e o INSERT são comandos básicos da DCL ( Data Control Language - Linguagem de Controle de Dados ).

**3. (ESAF/Auditor Informática/Pref Natal 2008) Com relação às características da linguagem SQL e dos SGBD, é correto afirmar que**

a) os "Triggers" são utilizados para auxiliar a manutenção da consistência dos dados. Também podem ser utilizados para propagar alterações ocorridas em um determinado dado de uma tabela para outra tabela.

b) considerando o SQL\_ANSI, o código a seguir é um exemplo correto para a criação de um Trigger relacionado a ações

na tabela "cadastro": **CREATE TRIGGER trigger\_cpf AS** Código a ser executado **FOR UPDATE ON cpf.cadastro**

c) os "Triggers" são relações que não armazenam dados, mas são definidos dinamicamente por uma consulta a uma tabela previamente analisada e otimizada.

d) os "Triggers" são relações que armazenam dados, definidos dinamicamente por uma regra de inserção a uma tabela.

e) os "Triggers" são um arquivo auxiliar associado a uma ou mais tabelas. Sua função é acelerar o tempo de acesso aos dados de uma tabela.

**4. (ESAF/Analista de TI/SEFAZ-CE 2007)** Na linguagem SQL, os procedimentos gravados para serem executados implicitamente quando ocorrer determinada ação do usuário, como, por exemplo, a modificação de uma tabela, são denominados

- a) Selects.
- b) Inserts.
- c) Views.
- d) Queries.
- e) Triggers.

**5. (ESAF/Analista de Sistemas/ENAP 2006)** A SQL possui recursos para apagar tabelas e bancos de dados a partir do comando

- a) **CREATE TABLE NULL.**
- b) **TRUNCATE PRIMARY KEY.**
- c) **DROP.**
- d) **INSERT INTO TABLE NULL.**



**e) DELETE TABLE NULL.**

**6. (ESAF/Analista Informática/IRB 2006) Considere um banco de dados com uma tabela com o nome "Carro", onde estão cadastrados os mais de 1000 veículos de uma empresa. Esta tabela contém um campo de nome "Cor", que pode ser nulo, contendo a informação da cor de cada veículo cadastrado. Ao se executar a instrução SQL ALTER TABLE Carro DROP COLUMN Cor**

**a) a tabela Carro será totalmente apagada devido à execução do comando DROP.**

**b) todos os registros preenchidos com o valor Null serão alterados.**

**c) todos os registros preenchidos com o valor diferente de Null serão alterados.**

**d) todas as informações das cores dos veículos contidas neste campo serão perdidas.**

**e) nada irá acontecer com os dados da tabela Carro.**

**7. (FCC/ACE-TI/TCE-SE 2011) Durante a criação de uma tabela - Create Table, em SQL, deseja-se especificar que uma coluna só possa incluir, por exemplo, valores maiores que zero. Uma constraint utilizada para isso é**

**a) Verify.**

**b) Check.**

**c) Max.**

**d) Avg.**

**e) Having.**

**8. (ESAF/Analista Infraestrutura/CVM 2010) Assinale a opção correta.**

**a) A linguagem de definição de dados permite expressar as consultas e atualizações do banco de dados.**

**b) A linguagem de manipulação de dados permite a especificação do esquema do banco de dados.**

**c) A linguagem de manutenção de dados permite expressar as consultas e atualizações do banco de dados.**

**d) A linguagem de definição de dados permite a especificação do esquema do banco de dados.**

**e) A linguagem de manipulação de consultas permite a atualizações do banco de dados.**

**9. (ESAF/APOTI/MPOG 2010) Em uma SQL**

**a) a Linguagem de Manipulação de Relacionamentos compreende os comandos para construir tabelas em um banco de dados.**

**b) a Linguagem de Definição de Dados fornece tabelas para criação e modificação de comandos.**

**c) os comandos básicos da Linguagem de Definição de Dados são Select, Insert, Update e Delete.**

**d) a Linguagem de Manipulação de Dados compreende os comandos para inserir, remover e modificar informações em um banco de dados.**

**e) os comandos básicos da Linguagem de Definição de Dados são Sort, Insert, Undo e Store.**

**10. (ESAF/PSS TI 2008) Na linguagem SQL, o uso da cláusula WHERE, quando apropriado,**

**a) agrupa o resultado em subconjuntos que possuem valores correspondentes em uma ou mais colunas. Em cada grupo não há duas linhas com o mesmo valor na(s) coluna(s) de agrupamento.**

**b) permite selecionar linhas baseadas em uma expressão booleana. Somente as linhas para as quais a expressão é avaliada como TRUE são retornadas no resultado.**

**c) especifica se o ResultSet de uma instrução SELECT simples, que atende aos requisitos para um cursor, é atualizável ou não.**

**d) permite selecionar colunas baseado em uma expressão numérica. Somente as colunas para as quais a expressão é avaliada como TRUE são retornadas no resultado. O uso desta cláusula obriga que os valores nulos sejam considerados diferentes nos agrupamentos.**

**e) cria uma ou mais linhas e as armazena na tabela especificada. O número de valores especificados na instrução WHERE deve ser idêntico ao número de colunas especificadas ou implícitas.**

**11. (ESAF/Auditor Informática/Pref Natal 2008) Quanto à estrutura, propriedades e sintaxe da linguagem SQL, é correto afirmar que**

**a) o CREATE e o DROP são comandos básicos da DDL ( Data Definition Language - Linguagem de Definição de Dados ).**

**b) o SELECT e o INSERT são comandos básicos da DDL ( Data Definition Language - Linguagem de Definição de Dados ).**

**c) o CREATE e o DROP são comandos básicos da DCL ( Data Control Language - Linguagem de Controle de Dados ).**

**d) o GRANT e o REVOKE são comandos básicos da DDL ( Data Manipulation Language - Linguagem de Manipulação de Dados ).**

**e) o SELECT e o INSERT são comandos básicos da DCL ( Data Control Language - Linguagem de Controle de Dados ).**

**12. (ESAF/AFC/CGU 2008) Em um banco de dados que utiliza a linguagem SQL para definição, manipulação e controle de dados, é correto afirmar que os comandos**

**a) CREATE, DROP e INSERT fazem parte da DML (Linguagem de Manipulação de Dados).**

**b) GRANT e REVOKE fazem parte da DCL (Linguagem de Controle de Dados).**

**c) INSERT, UPDATE e SELECT fazem parte da DDL (Linguagem de Definição de Dados).**

**d) ALTER, DELETE e REVOKE fazem parte da DML (Linguagem de Manipulação de Dados).**

**e) CREATE, GRANT e DROP fazem parte da DCL (Linguagem de Controle de Dados).**

**13. (ESAF/Analista de TI/SEFAZ-CE 2007) No SQL, considerando o uso das cláusulas SELECT, GROUP BY e HAVING, indique a opção que apresenta uma sintaxe correta.**

**a) SELECT Ano, COUNT(\*) AS 'Total' FROM Matricula WHERE Cod = 171 GROUP BY Ano HAVING count(\*) > 5**

**b) SELECT Ano, COUNT(\*) AS 'Total' FROM Matricula HAVING count(\*) > 5 WHERE Cod = 171 GROUP BY Ano**

**c) SELECT Ano, COUNT(\*) AS 'Total' WHERE Cod = 171 FROM Matricula GROUP BY Ano HAVING count(\*) > 5**

**d) SELECT Ano, COUNT(\*) AS 'Total' WHERE Cod = 171 GROUP BY Ano HAVING count(\*) > 5 FROM Matricula**

**e) SELECT Ano WHERE Cod = 171 GROUP BY Ano HAVING count(\*) > 5 COUNT(\*) AS 'Total' FROM Matricula**

**14. (ESAF/Analista Técnico/SUSEP 2006) Analise as seguintes afirmações relacionadas a conceitos básicos sobre Banco de Dados.**

**I. O comando SQL responsável por fechar uma transação confirmando as operações feitas é o INSERT.**

**II. O comando SQL responsável por fechar uma transação e desfazer todas as operações é o COMMIT.**

**III. Quando uma transação ainda está aberta para um usuário, enquanto não é executado um comando COMMIT, o próprio usuário pode ver as suas alterações, mas outros usuários não podem vê-las.**

**IV. Uma transação assegura um espaço de trabalho que contém várias alterações, inclusões e exclusões de dados em uma ou mais tabelas, com a possibilidade de confirmação ou cancelamento das operações sem comprometimento dos dados.**

**Indique a opção que contenha todas as afirmações verdadeiras.**

**a) I e II**

**b) II e III**

**c) III e IV**

**d) I e III**

**e) II e IV**

**15. (ESAF/AFC/CGU 2006) Um procedimento armazenado (stored procedure) é uma coleção de comandos em SQL que**

**a) provoca um aumento no tráfego na rede e reduz a performance do sistema, mas continua sendo largamente utilizado para criar mecanismos de segurança em bancos de dados relacionais.**

**b) encapsula tarefas repetitivas, aceita parâmetros de entrada e pode retornar um valor de status para indicar sucesso ou falha na execução.**

**c) estão armazenados no banco de dados e que são executadas diretamente na máquina do usuário.**

**d) estão armazenados na máquina do usuário e que são executadas diretamente no servidor do banco de dados.**

**e) são utilizados unicamente para autenticar um usuário, dando a ele direitos de acesso a escrita/alteração em tabelas do banco de dados.**

**16. (ESAF/AFRF/SRF 2005) Com relação ao uso da SQL na manipulação de dados, caso se queira eliminar linhas repetidas do conjunto resultado, deve-se utilizar a palavra-chave DISTINCT, da seguinte forma:**

**a) SELECT {colunas} FROM {tabelas} DISTINCT.**

**b) DISTINCT SELECT {colunas} FROM {tabelas}.**

**c) SELECT FROM {tabelas} DISTINCT {colunas}.**

**d) SELECT DISTINCT {colunas} FROM {tabelas}.**

**e) FROM {tabelas} SELECT DISTINCT {colunas}.**

**17. (CEPS-UFPA/Analista de TI/UFPA 2011) O comando SQL Select é utilizado para**

**a) atualizar um valor de um atributo.**

**b) excluir uma linha de uma tabela.**

**c) calcular um valor de uma expressão aritmética.**

**d) incluir uma linha em uma tabela.**

**e) remover um atributo de uma tabela.**

**18. (FCC/Analista Judiciário - TI/TRT23 2011) As operações da álgebra relacional Seleção, Projeção e Produto Cartesiano são implementadas na linguagem SQL, respectivamente, pelas cláusulas**

- a) Select, From e Where.**
- b) Select, Where e From.**
- c) Where, Select e From.**
- d) Where, From e Select.**
- e) Select, Select e Join.**

**19. (FCC/Analista Judiciário - TI/TRT24 2011) Um wildcard ( curinga ) SQL deve ser usado com um operador LIKE. Assim,**

- a) \_ ( símbolo underline ) é um substituto para zero ou mais caracteres.**
- b) % (símbolo percentual) é um substituto para zero ou mais caracteres.**
- c) % (símbolo percentual) é um substituto para exatamente um caracter.**
- d) [!lista] indica qualquer caracter simples da lista.**
- e) [lista] indica nenhum caracter simples da lista.**

## **8. GABARITO**

1	a	2	a	3	a	4	e	5	c
6	d	7	b	8	d	9	d	10	b
11	a	12	b	13	a	14	c	15	b
16	d	17	c	18	c	19	b		