Estratégia Concursos

Aula 00

TRT-AP/PA 8ª Região (Analista Judiciário - Tecnologia da Informação) Passo Estratégico de Conhecimentos Específicos

Autor:
Thiago Rodrigues Cavalcanti

26 de Janeiro de 2023

BANCO DE DADOS: 1 COMANDOS SQL. 1.1 DML - LINGUAGEM DE MANIPULAÇÃO DE DADOS. 1.2 DDL – LINGUAGEM DE DEFINIÇÃO DE DADOS. 1.3 DCL - LINGUAGEM DE CONTROLE DE DADOS

Sumário

Apresentação 1
O que é o Passo Estratégico? 2
Análise Estatística 3
Roteiro de revisão e pontos do assunto que merecem destaque 3
SQL 3
Introdução a SQL 3
Conceitos básicos 5
Sublinguagens de SQL 7
Tipos de dados 8
DML – Data Manipulation Language 17
Principais Comandos 26
Aposta estratégica 33
Questões estratégicas 34

APRESENTAÇÃO

Olá Senhoras e Senhores,

Eu me chamo Thiago Cavalcanti. Sou funcionário do Banco Central do Brasil, passei no concurso em 2010 para Analista de Tecnologia da Informação (TI). Atualmente estou de licença, cursando doutorado em

economia na UnB. Também trabalho como professor de TI no Estratégia e sou o analista do Passo Estratégico de Informática.

Tenho graduação em Ciência da Computação pela UFPE e mestrado em Engenharia de Software. Já fui aprovado em diversos concursos tais como ANAC, BNDES, TCE-RN, INFRAERO e, claro, Banco Central. A minha trajetória como concurseiro durou pouco mais de dois anos. Neste intervalo, aprendi muito e vou tentar passar um pouco desta minha experiência ao longo deste curso.

O QUE É O PASSO ESTRATÉGICO?

O Passo Estratégico é um material escrito e enxuto que possui dois objetivos principais:

a) orientar revisões eficientes;

b) destacar os pontos mais importantes e prováveis de serem cobrados em prova.

Assim, o Passo Estratégico pode ser utilizado tanto para **turbinar as revisões dos alunos mais adiantados nas matérias, quanto para maximizar o resultado na reta final de estudos por parte dos alunos que não conseguirão estudar todo o conteúdo do curso regular.**

Em ambas as formas de utilização, como regra, **o aluno precisa utilizar o Passo Estratégico em conjunto com um curso regular completo.**

Isso porque nossa didática é direcionada ao aluno que já possui uma base do conteúdo.

Assim, se você vai utilizar o Passo Estratégico:

a) **como método de revisão**, você precisará de seu curso completo para realizar as leituras indicadas no próprio Passo Estratégico, em complemento ao conteúdo entregue diretamente em nossos relatórios;

b) **como material de reta final**, você precisará de seu curso completo para buscar maiores esclarecimentos sobre alguns pontos do conteúdo que, em nosso relatório, foram eventualmente expostos utilizando uma didática mais avançada que a sua capacidade de compreensão.

Seu cantinho de estudos famoso!

Poste uma foto do seu cantinho de estudos nos stories do Instagram e nos marque:

@passoestrategico

Vamos repostar sua foto no nosso perfil para que ele fique famoso entre milhares de concurseiros!

ANÁLISE ESTATÍSTICA

A análise estatística estará disponível a partir da próxima aula.

ROTEIRO DE REVISÃO E PONTOS DO ASSUNTO QUE MERECEM DESTAQUE

A ideia desta seção é apresentar um roteiro para que você realize uma revisão completa do assunto e, ao mesmo tempo, destacar aspectos do conteúdo que merecem atenção.

Para revisar e ficar bem preparado no assunto, você precisa, basicamente, seguir os passos a seguir:

SQL

Dentro do contexto da Linguagem SQL vamos apresentar as construções (ou comandos) e os conceitos fundamentais. Vamos tentar entender como as bancas abordam este assunto em suas questões. Sabe-se que SQL é, **estatisticamente, o assunto mais cobrado** dentro do rol de conteúdos de banco de dados nas provas de concurso. Logo, essa aula é, sem dúvidas, uma das **mais importantes** do nosso curso.

Ao longo da explicação teórica, teremos algumas questões. Elas foram colocadas balanceando os requisitos de **relevância e frequência**. Ao final da explicação teórica, teremos questões da banca do seu concurso, elas serão as primeiras que aparecerão no bloco de questões comentadas. Vamos que vamos!

Introdução a SQL

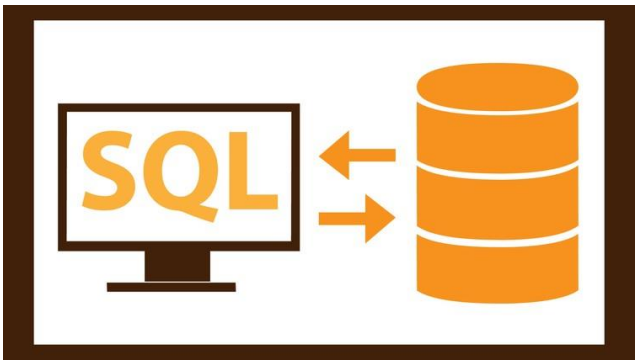
A linguagem SQL foi criada por um grupo de pesquisadores do laboratório da IBM em San Jose, Califórnia, mais especificamente pelos pesquisadores **Donald D. Chamberlin e Reymond Boyce**. Eles publicaram um artigo denominado "**SEQUEL: *A Structured English Query Language***", no qual apresentavam detalhes da linguagem, como sua **sintaxe e operações**. A IBM chegou a implementar, na IBM Research, o SQL como a interface para um sistema de banco de dados relacional experimental, chamado **SYSTEM R**.


Donald D. Chamberlin


Raymond F. Boyce

CURIOSIDADE

Alguns anos depois da publicação do artigo, devido a um conflito de patentes com um projeto secreto de uma empresa inglesa de aviação, SEQUEL mudou de nome para SQL.



A *Structured Query Language* – SQL – pode ser considerada **uma das principais razões para o sucesso dos bancos de dados relacionais**. Ela se provou bastante útil e foi adotada como padrão pela ANSI (*American National Standards Institute*). São várias versões da norma **ANSI/ISO/IEC 9075** publicadas ao longo do tempo: 1986, 1989, 1992, 1999, 2003, 2006, 2008, 2011 e 2016. Nove ao todo! Nosso foco de estudo se baseia na parte da norma que apresenta os fundamentos da linguagem SQL: **ISO/IEC 9075-2 (SQL/Foundation).** Uma norma geralmente possui várias partes, uma numeração separa conteúdos específicos em diversos documentos.

Por exemplo, a norma produzida pelo comitê com a numeração ISO/IEC 9075-**1** descreve a **estrutura conceitual** usada para especificar a **gramática** de SQL e o resultado das instruções de processamento nessa linguagem. A norma também define os termos e a notação utilizados nos demais documentos da série.

Por exemplo, o termo **tabela** é definida na norma ISO/IEC 9075-**1.** "Uma **tabela** possui uma **coleção** ordenada de uma ou mais **colunas** e uma coleção não ordenada de zero ou mais **linhas**. Cada coluna tem um nome e um tipo de dados. Cada linha tem, para cada coluna, exatamente um valor no tipo de dados dessa coluna." (tradução da versão em inglês). Observe a tabela APROVADOS abaixo e preencha a linha em branco com os seus dados 😊

APROVADOS

Identificador	Nome	Data de Nascimento	Concurso Aprovado
1	Thiago	12/02/1983	Banco Central

Pense que programar é semelhante a aprender uma nova língua. Você precisa estudar os elementos da linguagem para conseguir escrever bons textos. Para manter banco de dados relacionais você vai construir tabelas e relacionamentos entre elas. Em seguida, você poderá inserir, consultar, atualizar e remover dados das mesmas. **A linguagem SQL vai viabilizar essas ações**. Antes de colocar a mão na massa, vamos definir alguns conceitos.

Conceitos básicos

Primeiramente, SQL é uma linguagem de programação reconhecida internacionalmente e usada para definição e manutenção de bancos de dados relacionais. A **principal característica da linguagem** é ser **declarativa**[1], ou seja, os detalhes de implementação dos comandos são deixados para os SGBDs relacionais. Não esqueça disso! Já caiu várias vezes em provas anteriores. No SQL você **declara o que você quer e o SGBD vai achar os dados para você no banco, caso existam é claro.**

Vamos tirar do contexto de programação e tentar entender a ideia por trás de linguagem declarativa e da sua "rival" a linhagem procedural.

Imagine que eu te faça a seguinte pergunta: Estou ao lado do supermercado pão de açúcar, como eu faço para chegar na sua casa?

A resposta **procedural** seria: Pegue a direita, faça o retorno na rotatória e volte no sentido do bosque do Sudoeste, em seguida, vire a direita e siga até a primeira avenida, então saia na rotatória na terceira saída. Siga até a entrada da quadra 101/102 e procure pelo bloco C da quadra 101. Veja que eu mostrei o passo a passo da rota até a minha casa.

E resposta **declarativa**!? Como seria? Vejamos: Meu endereço é na Quadra 101 bloco C, Sudoeste Brasília, Apto 304[2].

Percebe a diferença? Linguagem declarativa descrevem **O QUE** você quer, já a linguagem procedural descreve **COMO** fazer o que você quer.

Outro aspecto importante é que SQL **não** existe fora do contexto relacional devido ao fato de ser fundamentada no modelo relacional. A linguagem utiliza o cálculo e a álgebra relacional como base teórica para implementar as operações dentro dos SGBDs. Embora a Linguagem SQL inclua alguns recursos da álgebra relacional, ela é **baseada em grande parte no cálculo relacional de tupla**.

Ficou assuntado com esses termos matemáticos? Não se preocupe! A **sintaxe SQL é mais fácil de ser utilizada do que qualquer uma das duas linguagens formais. Quer um exemplo?** Vamos voltar para a nossa tabela inicial. Favor preencher novamente com seus dados.

---

[1] Linguagens declarativas se contrapõem com as linguagens procedurais. Nestas você precisa descrever o passo a passo para execução de uma determinada tarefa.

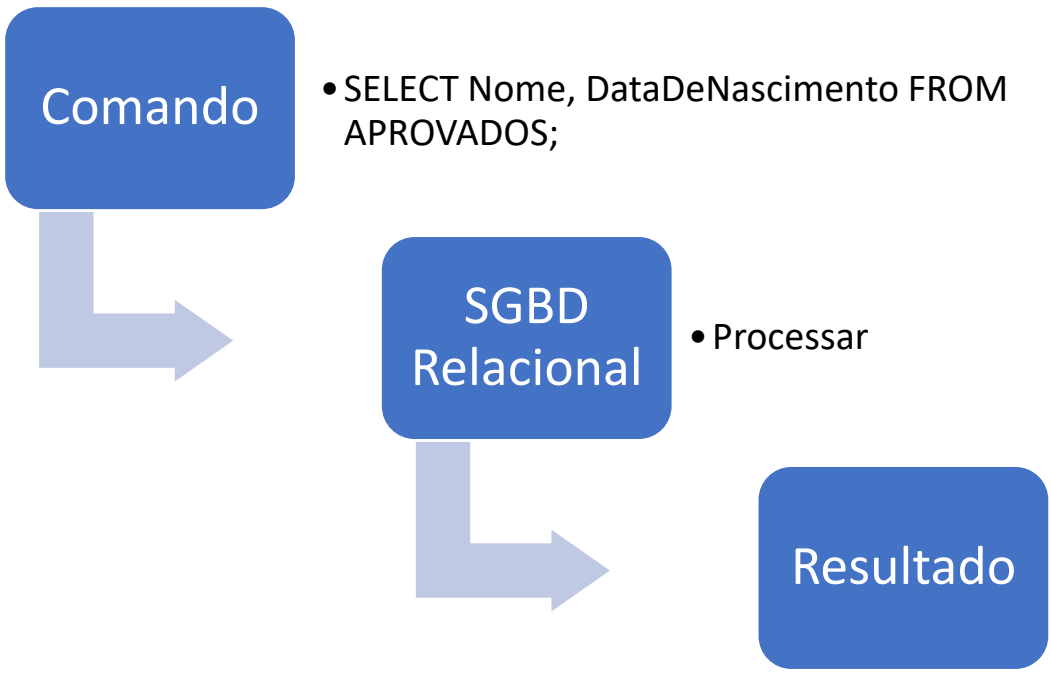
[2] Esse não é meu endereço. Se você estiver pensando em me mandar um presente entre em contato.

APROVADOS

Identificador	Nome	DataDeNascimento	Concurso Aprovado
1	Thiago	12/02/1983	Banco Central
3	Diego	15/03/1984	STN
4	Herbert	05/02/1987	SEFAZ

Vamos fazer uma consulta sobre a tabela aprovados para extrair apenas as colunas Nome e Data de Nascimento.

Nome	DataDeNascimento
Thiago	12/02/1983
Diego	15/03/1984
Herbert	05/02/1987



Seu nome e data de nascimento, devem aparecer no resultado da mesma forma estão escritos na tabela anterior. Percebe que é simples? Dizemos para o SGBD quais as colunas que vamos retornar da cláusula SELECT e a tabela onde estão os dados na cláusula FROM ... e voilà! Eis a nossa resposta em forma de tabela exatamente com os dados que solicitamos.

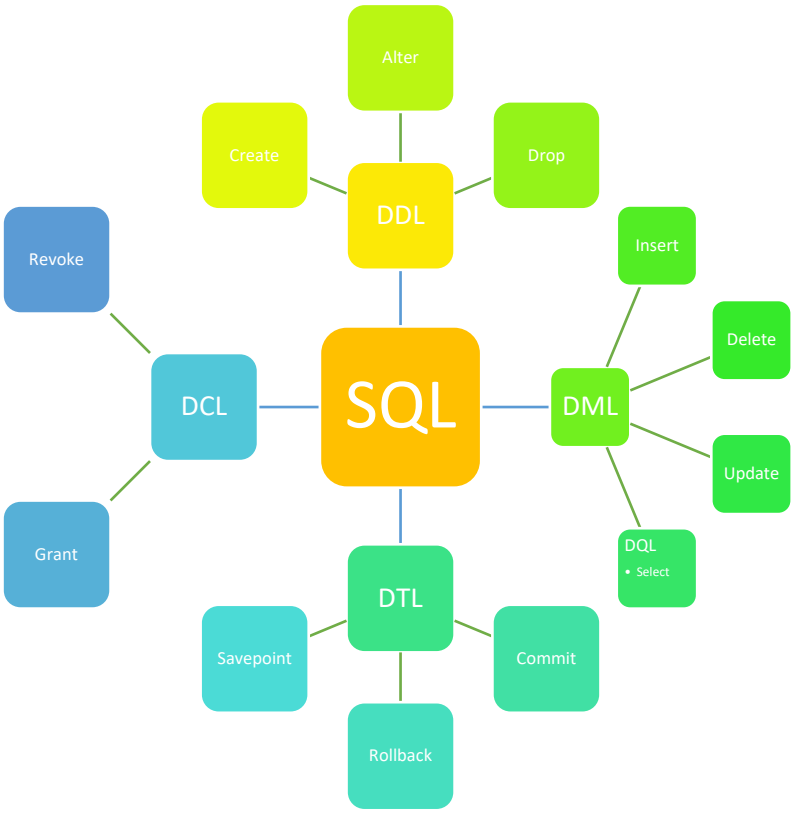
Mas o SELECT não é o único comando presente na linguagem. O SELECT faz parte de um grupo de comandos de manipulação da base de dados. Ele vai procurar os dados em tabelas usando critérios de busca definidos pelo usuário. Vamos conhecer um pouco mais sobre os grupos de comandos da linguagem.

Sublinguagens de SQL

Quando começamos a estudar a linguagem, precisamos entender as subdivisões que existem dentro dos comandos possíveis. SQL é uma linguagem de banco de dados abrangente, tem instruções para **definição** de dados (DDL), consultas e atualizações (DML). Possui ainda facilidades para definir **visões** sobre o banco de dados, para especificar **segurança e autorização**, para definir **restrições de integridade** e para especificar **controles de transação**.

Alguns autores chegam a dividir a linguagem em cinco categorias. As categorias são baseadas nas funcionalidades que cada comando executa sobre o banco de dados. Vejam a lista abaixo:

- **DDL** – Data Definition Language – A linguagem de definição de dados contém comandos que criam, modificam e excluem objetos de banco de dados. São exemplos de comando: CREATE, ALTER, DROP e TRUNCATE.
- **DML** – Data Manipulation Language – A linguagem de manipulação de dados fornece instruções para trabalhar com os dados armazenados como SELECT, INSERT, UPDATE e DELETE.
- **DQL** – Data Query Language – A linguagem de consulta de dados é um subconjunto da DML que possui apenas a instrução de SELECT.
- **DTL** – Data Trasaction Language – Linguagem de transação de dados inclui comandos de COMMIT, ROLLBACK e SAVEPOINT
- **DCL** – Data Control Language – A linguagem de controle de dados contém os comandos relacionados com as permissões de controle de acesso. Garante os privilégios aos usuários para acessar os objetos do banco. Os mais conhecidos comandos são o GRANT e o REVOKE.



Tipos de dados

SQL inclui **diversos** tipos de tipos de dados **pré-definidos**: string (conjunto de caracteres), numéricos, binário, datetime, interval, boolean e XML. Os tipos de dados são usados na instrução CREATE TABLE como parte das definições da coluna:

```
CREATE TABLE <tablename>(
<column_name> <data_type> ... ,
<column_name> <data_type> ... ,
... );
```

Primeiramente, vamos falar dos tipos de dados formados por cadeias de caracteres.

Character

No SQL padrão, esses tipos são divididos em tamanho fixo e variável. Todas as cadeias de caracteres em SQL podem ser de comprimento fixo ou variável. Você tem três tipos principais de cadeias de caracteres:

1. Cadeias de caracteres de tamanho fixo (**CHARACTER** ou **CHAR**)
2. Cadeias de caracteres de tamanho variável (**CHARACTER VARYING** ou **VARCHAR**)

3. Cadeia de caracteres para armazenar grandes objetos (**CHARACTER LARGE OBJECT** ou **CLOB**).

Como você pode imaginar, essa flexibilidade tem um preço de desempenho, pois o SGBD deve executar a tarefa adicional de alocação dinâmica. Um CHAR ou CHARACTER especifica o número exato de caracteres que serão armazenados para cada valor. Por exemplo, se você definir o comprimento de 10 caracteres, mas o valor contém apenas seis caracteres, os quatro caracteres restantes serão completados com espaços em branco. Um exemplo da sintaxe do comando seria: CONCURSO_NOME CHAR (100).

O outro tipo de cadeia de caracteres é o CHARACTER VARYING ou VARYING CHAR ou VARCHAR. Ele especifica o número máximo de caracteres que pode ser incluído em uma variável. O número de caracteres armazenados é exatamente o mesmo número do valor introduzido, de modo que nenhum espaço será adicionado ao valor. Um exemplo da definição de uma coluna deste tipo: CONCURSO_NOME VARCHAR (60). Observem que o comprimento da cadeia é passado entre parênteses.

O tipo de dados CHARACTER LARGE OBJECT (CLOB) foi introduzido juntamente com o SQL:1999. Como o próprio nome sugere, ele é usado para armazenar grandes cadeias de caracteres que são demasiadamente grandes para o tipo CHARACTER. CLOBs se comportam como cadeias de caracteres comuns, mas há uma série de restrições sobre o que você pode fazer com eles.

Por exemplo, um CLOB **não** pode ser usado em uma **chave primária**, **chave estrangeira** ou com predicado **UNIQUE**. Devido ao seu tamanho grande, aplicações geralmente não transferem o CLOB para ou a partir de um banco de dados. Em vez disso, um tipo de dados especial do lado do cliente chamado CLOB *locator* é utilizado para manipular os dados CLOB. É um parâmetro cujo valor identifica um objeto grande.

Vários idiomas têm alguns CHARACTER que diferem de quaisquer caracteres em outro idioma. Por exemplo, o alemão tem alguns caracteres especiais não presentes no conjunto de caracteres do idioma Inglês. Você pode especificar o idioma Inglês definindo-o como o padrão para o seu sistema, mas você pode usar outros conjuntos de caracteres alternativos; para isso existe o NATIONAL CHARACTER, NATIONAL CHARACTER VARYING, e NATIONAL CHARACTER LARGE OBJECT. Eles são tipos de dados com a mesma função que o CHARACTER, CHARACTER VARYING e CHARACTER LARGE OBJECT - a única diferença é que o conjunto de caracteres que você está especificando é diferente do conjunto de caracteres padrão.

Você pode especificar o conjunto de caracteres quando define uma coluna de uma tabela. Se você quiser, cada coluna pode usar um conjunto de caracteres diferente. O exemplo a seguir mostra a criação de uma tabela e usa vários conjuntos de caracteres:

```
CREATE TABLE IDIOMAS (
  LANGUAGE_1 CHARACTER (40),
  LANGUAGE_2 CHARACTER VARYING (40) CHARACTER SET GREEK,
  LANGUAGE_3 NATIONAL CHARACTER (40),
  LANGUAGE_4 CHARACTER (40) CHARACTER SET KANJI
  ) ;
```

Aqui a coluna LANGUAGE_1 contém caracteres em conjunto de caracteres padrão do sistema. A coluna LANGUAGE_3 contém caracteres em conjunto de caracteres nacional do sistema. A coluna LANGUAGE_2

contém caracteres gregos. E a coluna LANGUAGE_4 contém caracteres Kanji. Depois de uma longa ausência, conjuntos de caracteres asiáticos, como Kanji, estão, agora, disponíveis em muitos produtos.

Para finalizar vamos observar um comando com os principais tipos de dados de caracteres associados a colunas de uma tabela. Vejamos:

```
CREATE TABLE teste (
  id    DECIMAL PRIMARY KEY,
  col1  CHAR(8),              --  exatamente 8 caracteres
  col2  VARCHAR(100),         -- até 100 caracteres
  col3  CLOB                  -- strings muito longas
);
```

Binary String

Ainda dentro das strings temos as **strings binárias**. Uma string binária é uma sequência de bytes da mesma forma que uma string de caracteres é uma sequência de caracteres. Ao contrário das strings de caracteres que geralmente contêm informações na forma de texto, ela é usada para armazenar dados não tradicionais, tais como imagens, áudio e arquivos de vídeo, executáveis de programa, e assim por diante.

Os tipos de dados de strings binárias foram introduzidos no SQL:2008. Considerando que os dados binários têm sido fundamentais para computadores digitais desde o Atanasoff-Berry Computer (primeiro computador eletrônico digital) da década de 1930, esse reconhecimento da importância dos dados binários parece vir um pouco tarde para SQL. Antes tarde do que nunca! Existem três tipos diferentes BINARY, BINARY VARYING e BINARY LARGE OBJECT. BINARY LARGE OBJECT é conhecido como BLOB. O BLOB armazena grandes grupos de bytes, até o valor especificado.

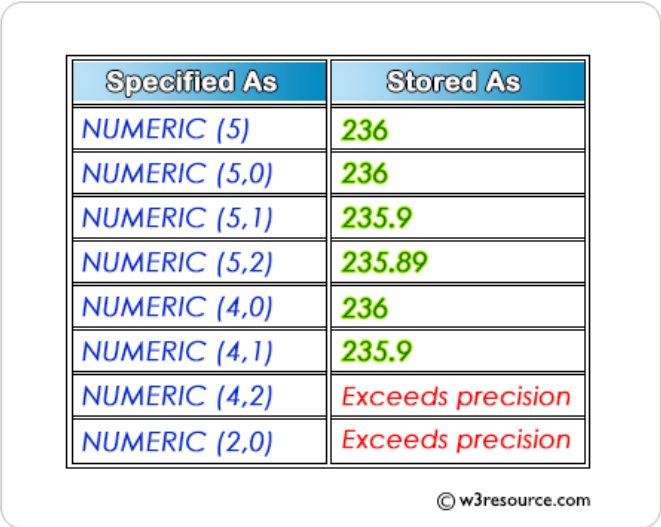
Tipos numéricos

Vamos, agora, tratar dos tipos de dados numéricos. Eles são divididos em duas grandes categorias: **números exatos (*exact numbers*)** e **números aproximados (*approximate*)**.

Os **números exatos** podem ser números **inteiros** (lápis, pessoas ou planetas) ou ter **casas decimais** (preços, pesos ou percentuais). Os números podem ser **positivos ou negativos**. Eles também podem ter **precisão e escala**. O que seria isso? Vejamos ...

A **precisão (p)** determina o número total máximo de dígitos decimais que podem ser armazenados (tanto à esquerda quanto à direita do ponto decimal). E **escala (e)** especifica o número máximo de casas decimais permitidas. SQL permite criação de colunas com os tipos NUMERIC e DECIMAL.

Veja o exemplo abaixo em que o número 235,89 está descrito como NUMERIC(p,e):



Specified As	Stored As
NUMERIC (5)	236
NUMERIC (5,0)	236
NUMERIC (5,1)	235.9
NUMERIC (5,2)	235.89
NUMERIC (4,0)	236
NUMERIC (4,1)	235.9
NUMERIC (4,2)	Exceeds precision
NUMERIC (2,0)	Exceeds precision

Figura 1 - A figura acima representa exemplos de precisão e escala para o número 235,89.

Os números de pontos flutuantes ou aproximados são números que não podem ser representados com precisão absoluta. O tipo de dado *float* é representado por *float(p)*, um valor numérico aproximado **com precisão binária**, ou seja, número de dígitos binários necessários para o armazenamento do valor aproximado. O valor máximo de <precisão> é definido pela implementação e não deve ser maior o valor máximo definido na implementação do SGBD.

Vamos imaginar um exemplo. Suponha que exista uma coluna col_float na nossa base de dados e que essa coluna foi definida com o tipo de dado float(20). O valor 20 nos informa que é possível armazenar seu número em até 20 bits. Vamos então armazenar o número 235,89 nesta coluna. São utilizados oito dígitos binários para armazenar a parte inteira (235 (decimal) = 1110 1011 (binário)) e sete para armazenar a parte fracionária (89 (decimal) = 101 1001 (binário). Assim, temos 8+7=15 bits. Esse valor será inteiramente gravado na coluna do banco.

Agora vamos pensar em um outro número, imagine o saldo na conta do prof. Valley Carvalhus, R$ 116.456,23. Para armazenar a parte inteira precisamos de 17 bits, 116456 (decimal) = 1 1100 0110 1110 1000 (binário). Sobrando, portanto, 3 bits para armazenar o valor fracionário. Como 23 em decimal é equivalente a 10111 em binário, não temos espaço para armazenar esse valor. Assim o SGBD vai arredondar o número 0,23 para 0,2, truncando a segunda casa decimal. Neste caso o número 2, que em binário é 10, pode ser armazenado em 3 bits. Logo, o valor que seja gravado no banco de dados será 116.456,2. Lógico que esses 3 centavos não farão diferença para o professor, mas farão você entender o funcionamento do parâmetro passado na criação da variável float.

Deu para entender? 😊

Esses tipos de dados numéricos estão disponíveis em todos os SGBDs relacionais, por exemplo, no SGBD Oracle. Vejam, abaixo, a relação entre os tipos SQL padrão e o ORACLE, bem como alguns comentários importantes sobre os aspectos numéricos do Oracle.

ANSI SQL Data Type	Oracle Data Type
NUMERIC[(p,s)]	NUMBER(p,s)
DECIMAL[(p,s)]	
INTEGER	NUMBER(p,0)
INT	
SMALLINT	
FLOAT	FLOAT(126)
DOUBLE PRECISION	FLOAT(126)
REAL	FLOAT(63)

Os tipos de dados NUMERIC e DECIMAL do SQL ANSI são representados pelo tipo NUMBER no ORACLE. Eles só podem ser definidos como números de ponto fixo. E, para esses tipos de dados, o valor padrão para a escala é zero (0). O tipo de dado FLOAT é um tipo de ponto flutuante com precisão binária. O valor default para a precisão neste tipo de dados é de 126 casa binárias ou 38 casas decimais.

Perceba que utilizamos o tipo FLOAT do ORACLE para representar os tipos DOUBLE PRECISION e REAL do SQL ANSI. O tipo de dados DOUBLE PRECISON é também um número de ponto flutuante como precisão binária de 126. E, por fim, o tipo REAL é um ponto flutuante com precisão de 63 casas binárias ou 18 casas decimais. Esses valores são informados como parâmetros, como observamos na tabela acima.

Vamos observar agora um exemplo de comando com a descrição de vários atributos do tipo numérico.

```
CREATE TABLE test (

      id    DECIMAL PRIMARY KEY,

      col1  DECIMAL(5,2),     -- 3 dígitos antes e 2 dígitos depois da vírgula
                                decimal

      col2  SMALLINT,         -- Apenas valores inteiros.

      col3  INTEGER,          -- Apenas valores inteiros.

      col4  BIGINT,            -- Apenas valores inteiros.
```

```
    col5  FLOAT(24),          -- pelo SQL/ANSI você vai utilizar 24 bits para
                                representar o número, começando pelos dígitos a
                                esquerda da casa decimal.

    col6  REAL,

    col7  DOUBLE PRECISION

);
```

Para concluir o nosso estudo sobre tipos de dados numéricos, apresentamos uma tabela que relaciona os tipos com o espaço de armazenamento necessário para cada valor associado, e o range, que seria os valores possíveis ou o domínio de um determinado tipo numérico.

DATA TYPE	STORAGE SIZE (BYTES)	RANGE
INTEGER	4	-2,147,483,648 to +2,147,483,647
SMALLINT	2	–32,768 to + 32,768
BIGINT	8	-9,223,372,036,854,775,808 to +9,223,372,036,854,775,808
REAL	*4*	The range is from negative 3.402E + 38 to negative 1.175E – 37, or from positive 1.175E – 37 to 3.402E + 38. It also includes 0.
FLOAT	*4 to 8*	The number can be zero or can range from –1.79769E + 308 to –2.225E – 307, or from 2.225E – 307 to 1.79769E + 308.
DOUBLE	*8*	The number can be zero or can range from –1.79769E + 308 to –2.225E – 307, or from 2.225E – 307 to 1.79769E + 308.

Datetime e interval

Vamos, agora, falar sobre o tipo DATE. DATE é uma estrutura que consiste em três elementos: ano, mês e dia. O ano é um número de quatro dígitos que permite que os valores de 0000 a 9999. O mês é um elemento de dois dígitos com valores de 01 a 12, e dia que tem dois dígitos com um intervalo de 01 a 31. SQL/ANSI define a semântica de data usando a estrutura descrita acima, mas os SGBDs não são obrigados a usar essa abordagem, desde que a implementação produza os mesmos resultados.

O tipo TIME consiste de hora, minuto e segundo. A hora é um número de 00 a 23. O minuto é um número de dois algarismos, de 00 a 59. O segundo é um inteiro entre 00-59 ou um número decimal com uma precisão mínima de cinco e escala mínima de três que pode conter valores de 00.000 para 59.999.

Temos ainda os tipos: **1.** DATETIME que combina data e hora em um único tipo, com intervalo de datas. **2.** TIMESTAMP que engloba os campos DATE e TIME, mais seis posições para a fração decimal de segundos e uma qualificação opcional WITH TIME ZONE e **3.** INTERVAL que serve para calcular o intervalo entre dois objetos que representam tempos. Vejamos um exemplo de criação de uma tabela cujas colunas são dos tipos temporais.

```
CREATE TABLE tempo (

      id   DECIMAL PRIMARY KEY,

      col1 DATE,    -- armazena ano, mês e dia

      col2 TIME,

      col3 TIMESTAMP(9), -- armazena um selo de tempo.

      col4 TIMESTAMP WITH TIME ZONE   -- exemplo de timestamp com timezone

);
```

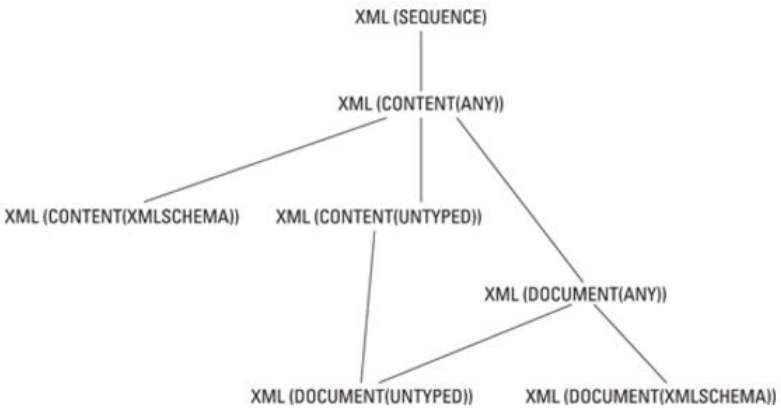
Boolean e XML

O tipo de dados BOOLEAN consiste na verdade dos valores distintos, verdadeiro e falso, assim como o valor desconhecido. O valor verdadeiro ou falso booleano pode ser comparado com um valor NULL.

XML é um acrônimo para eXtensible Markup Language, que define um conjunto de regras para a adição de marcação aos dados. As estruturas de marcação permitem aos dados transmitir o que eles significam. XML permite o compartilhamento de dados entre diferentes plataformas.

O tipo de dados XML tem uma estrutura de árvore, assim, um nó raiz pode ter nós filho, o que pode, por sua vez, ter seus próprios filhos. Introduzido pela primeira vez no SQL:2003, o tipo XML foi concretizado na versão SQL/XML: 2005, e ainda mais melhorado no SQL:2008.

Os modificadores primários do tipo XML são SEQUENCE, CONTENT e DOCUMENT. Os modificadores secundários são UNTYPED, ANY e XMLSCHEMA. A figura mostra a estrutura de árvore que ilustra as relações hierárquicas entre os subtipos.



Por fim, é importante saber que o valor nulo é membro de todos os tipos de domínios. Se você declarar o domínio de um atributo como NOT NULL, o SGBD vai proibir a inserção de valores nulos para essa coluna. Vejamos um exemplo de uma tabela com uma coluna do tipo XML.

```
CREATE TABLE test(
      id    DECIMAL PRIMARY KEY,
      col1  XML
);
```

Tipos ROW, Collections e UDT

Além dos tipos predefinidos, embutidos, SQL suporta tipos de coleção, tipos construídos e tipos definidos pelo usuário (UDT).

Tipos ROW

O tipo de dados de linha foi introduzido com o SQL:1999. Uma coisa notável sobre o tipo de dados ROW é que ela viola as regras de normalização declaradas por Codd nos primeiros dias da teoria de banco de dados relacional. Uma das características que definem a primeira forma normal é que um atributo de uma linha da tabela não pode ter um valor múltiplo. Um campo pode conter um e apenas um valor, ou seja, é atômico. No entanto, o tipo de dados ROW permite que você declare uma linha inteira de dados contida dentro de um único campo em uma única linha de uma tabela, em outras palavras, uma linha dentro de outra.

As formas normais, definidas por Codd, são características que definem a estrutura para bancos de dados relacionais. A inclusão do tipo ROW no padrão SQL foi a primeira tentativa de ampliar SQL além do modelo relacional puro. Considere a seguinte instrução SQL, que define um tipo ROW para informações sobre o endereço de uma pessoa:

```
CREATE ROW TYPE addr_typ (
        Street          CHARACTER VARYING (25),
        City            CHARACTER VARYING (20),
```

```
        State           CHARACTER (2),
        PostalCode      CHARACTER VARYING (9)
) ;
```

Depois que ele é definido, um novo tipo de linha pode ser usado em uma definição de tabela:

```
CREATE TABLE CUSTOMER (
        CustID          INTEGER     PRIMARY KEY,
        LastName        CHARACTER VARYING (25),
        FirstName       CHARACTER VARYING (20),
        Address         addr_typ,
        Phone           CHARACTER VARYING (15)
) ;
```

A vantagem aqui é que se você está mantendo informações de endereço para diferentes entidades - como clientes, fornecedores, colaboradores e acionistas - você tem que definir os detalhes da especificação de endereço apenas uma vez: na definição do tipo ROW.

Tipos de coleção

Após SQL sair da fronteira do modelo relacional com o SQL:1999, os tipos de dados que violam a primeira forma normal tornaram-se possíveis. Um campo pode conter uma coleção de objetos, em vez de apenas um. O tipo ARRAY foi introduzido no SQL:1999, e o tipo MULTISET foi introduzido no SQL:2003.

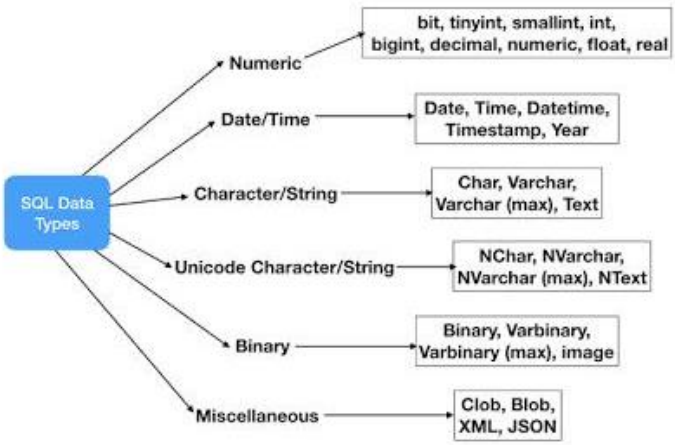
Duas coleções podem ser comparadas entre si somente se ambas são do mesmo tipo, ou ARRAY ou MULTISET, e se os seus tipos de elementos são compatíveis. Como ARRAYs têm uma ordem para os elementos definidos, elementos correspondentes podem ser comparados. MULTISETS não tem nenhuma ordem definida para os elementos, mas você pode compará-los, se (a) existe uma enumeração sobre cada um deles e (b) as enumerações podem ser emparelhadas.

Tipos definidos pelo usuário (UDT – User defined types)

Tipos definidos pelo usuário (UDTs) representam outro exemplo de recursos que chegaram no SQL:1999 que vêm do mundo de programação orientada a objetos. Como um programador de SQL, você não está mais restrito aos tipos de dados definidos na especificação SQL. Você pode definir seus próprios tipos de dados, usando os princípios de tipos de dados abstratos (ADTs) encontrados em tais linguagens de programação orientada a objetos como C++.

Um dos benefícios mais importantes dos UDTs é o fato de que você pode usá-los para eliminar a diferença de impedância entre o SQL e a linguagem de host que está "envolvendo" o SQL. Um problema de longa data com o SQL tem sido o fato de tipos de dados predefinidos do SQL não correspondem aos tipos de dados das linguagens host no qual as instruções SQL são incorporadas. Agora, com UDTs, um programador de banco de dados pode criar tipos de dados dentro do SQL que correspondem aos tipos de dados da linguagem de host.

UDT tem atributos e métodos, que são encapsulados. O mundo exterior pode ver as definições de atributos e os resultados obtidos pelos métodos - mas as implementações específicas dos métodos estão escondidas. O acesso aos atributos e aos métodos de uma UDT pode ser ainda mais restrito, podemos especificá-los como públicos, privados ou protegidos.



Agora que vimos os tipos de dados, vamos passar rapidamente pelas possíveis restrições de integridade que compõem o padrão SQL e vão fazer parte dos comandos de criação de tabelas.

DML – Data Manipulation Language

São quatro os principais comandos DML: SELECT, INSERT, UPDATE e DELETE. Todos os SGBDs relacionais implementam esses comandos. Vamos conhecer a sintaxe de cada um deles. É importante perceber que a maioria das peculiaridades estão descritas para o comando SELECT.

O COMANDO SELECT

A instrução SELECT permite formar consultas complexas que podem retornar exatamente o tipo de dados que você deseja recuperar. Como você pode observar abaixo, as cláusulas exigidas na construção de um comando SELECT são a palavra reservada SELECT e a cláusula FROM. Todas as outras cláusulas são opcionais! É importante relembramos que este comando possui uma correspondência com a álgebra relacional. Assim, a projeção corresponde às colunas definidas no SELECT, e a seleção da álgebra relacional corresponde às condições de busca inseridas na cláusula WHERE. Podemos, ainda, observar um produto cartesiano entre as tabelas na cláusula FROM.

```
SELECT [ DISTINCT | ALL ] { * | <select list> }
FROM <table reference> [ { , <table reference> } . . . ]
[ WHERE <search condition> ]
[ GROUP BY <grouping specification> ]
[ HAVING <search condition> ]
[ ORDER BY <order condition> ]
```

Sobre o comando acima, a primeira informação importante é que o * (asterisco) é utilizado quando queremos extrair na nossa consulta todas as colunas da tabela, ou das relações descritas na cláusula FROM. Quando a nossa intenção for recuperar também todas as linhas, devemos omitir a clausula WHERE. Nela são informadas as restrições que queremos fazer sobre a nossa busca. Podemos, por exemplo, restringir o domínio de um valor inteiro selecionando apenas as tuplas que satisfazem essa restrição.

Quando o SGBD recebe uma consulta com todas as palavras chaves acima, ele segue uma ordem lógica para avaliação do comando. Primeiramente ele analisa as tabelas ou relações envolvidas na consulta que estão presentes no FROM. Em seguida, o SGBD vai verificar as restrições de busca ou condições contidas na cláusula WHERE. Dando prosseguimento, caso exista uma especificação de agrupamento descrita no GROUP BY, ela é verificada.

Essa condição de agrupamento geralmente vem acompanhada de uma função de agregação sobre uma determinada coluna da tabela. Podemos pensar, por exemplo, na operação de soma (SUM) dos valores de um atributo numérico da tabela. Após o agrupamento é possível fazer uma restrição sobre os valores agrupados. Pela descrição da norma SQL/ANSI, você pode comparar o resultado dos agrupamentos e selecionar apenas aqueles que satisfaçam a uma condição de busca.

Após essa etapa, o SGBD vai avaliar quais colunas são descritas na cláusula SELECT do comando. Por fim, é possível ordenar a relação pelos atributos ou colunas listadas no ORDER BY.

No Oracle, por exemplo, é possível definir uma variável prefixada utilizando o & para avisar ao banco de dados que o valor será informado no momento da execução do comando. Veja, nas figuras abaixo, o uso do & para uma variável numérica, e para o caso do valor recebido ser uma data ou uma *string* de caracteres onde devem ser usadas as aspas simples.

```
SELECT employee_id, last_name, salary, department_id
FROM   employees
WHERE  employee_id = &employee_num ;
```

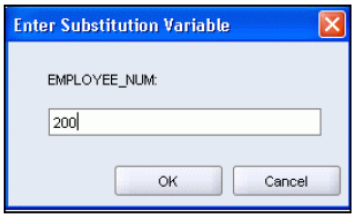
```
SELECT last_name, department_id, salary*12
FROM   employees
WHERE  job_id = '&job_title' ;
```

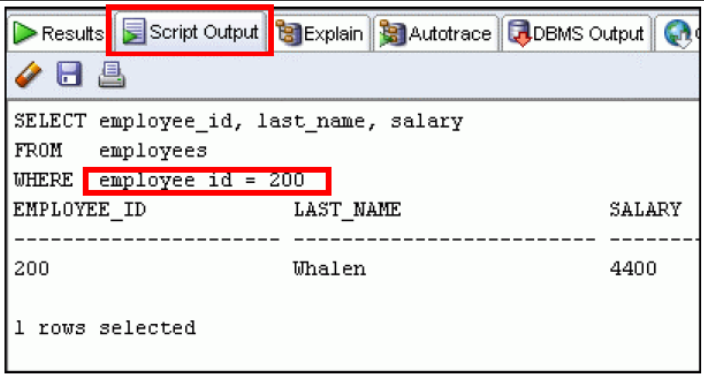
Outro ponto interessante no uso de variáveis é quando você quer reusar o valor da variável sem necessitar de digitar novamente ou repassar um novo valor. Neste caso, você deve usar dois &´s comerciais. Veja a figura abaixo:

```
SELECT   employee_id, last_name, job_id, &&column_name
FROM     employees
ORDER BY &column_name ;
```

Use o comando VERIFY para alternar a exibição da variável de substituição, tanto antes como depois SQL Developer substitui variáveis de substituição com valores:

```
SET VERIFY ON
SELECT employee_id, last_name, salary
FROM   employees
WHERE  employee_id = &employee_num;
```





ALIAS, DISTINCT e operadores

A Linguagem SQL possui um mecanismo para renomear tanto as relações quanto os atributos por meio da cláusula **as**, da seguinte forma: nome_antigo AS novo_nome. Esse mecanismo é conhecido como **alias**. Quando utilizado no contexto de uma tabela na cláusula FROM, os alias são conhecidos como variáveis de tuplas.

Existem diversos modificadores de consulta que nos ajudam a adequar o retorno da nossa consulta. Para forçar a eliminação de tuplas duplicadas, devemos inserir a declaração **DISTINCT** depois do SELECT. A cláusula SELECT pode conter expressões aritméticas envolvendo os operadores +, - , * , e /, em operações que utilizam constantes ou atributos de tabelas.

Não é necessário que os atributos usados na cláusula WHERE estejam listados no SELECT. A cláusula corresponde ao predicado de seleção da álgebra relacional como já falamos. Ela consiste de um predicado envolvendo atributos das relações que aparecem na cláusula FROM. Podemos utilizar operadores aritméticos (+ , - , * , / , % (módulo)), operadores de comparação (<, <=, >, >=, = e <>) e ainda conectivos ou operadores lógicos (AND, OR e NOT).

Outra possibilidade de comparação existente é quando estamos comparando STRINGs. Neste caso, podemos utilizar os operadores LIKE, NOT LIKE, % (que combina qualquer substring, independente do tamanho) e _ (combina caractere a caractere). Basicamente o que fazemos com esses operadores é verificar se um valor de um atributo em uma determinada tupla é semelhante ao de uma constante passada como parâmetro. Vejamos

um exemplo, se você quiser encontrar o nome de todos os clientes cuja rua contenha a substring ‘Professor’, você pode executar o comando abaixo.

```
select nome_cliente

from cliente

where rua_cliente like '% Professor %';
```

Quando construímos um predicado com vários operadores, precisamos entender que existe uma precedência na avaliação dos operadores. Você pode usar parênteses para sobrescrever as regras de precedência. A ordem de avaliação das regras de precedências pode ser vista na lista abaixo:

Operator	Meaning
1	Arithmetic operators
2	Concatenation operator
3	Comparison conditions
4	IS [NOT] NULL, LIKE, [NOT] IN
5	[NOT] BETWEEN
6	Not equal to
7	NOT logical condition
8	AND logical condition
9	OR logical condition

Depois de definir a cláusula WHERE podemos ordenar o resultado da consulta por meio do comando ORDER BY. É possível ordenar por um determinado atributo em ordem crescente e outro em ordem decrescente. Para isso, basta seguir a seguinte sintaxe: ORDER BY <nome(s)_coluna(s)> DESC ou ASC (**default**) e separar por vírgula os nomes das colunas com suas respectivas formas de ordenação. Lembrando que, se não for definida, a ordenação padrão é feita de forma ascendente.

Para executarmos a ordenação, pode-se optar por usar valores numéricos que referenciam a enésima coluna que aparece na cláusula SELECT. Veja o exemplo na figura abaixo, cujo valor **3** referencia a coluna *department_id*:

```
SELECT   last_name, job_id, department_id, hire_date
FROM     employees
ORDER BY 3;
```

Consultas aninhadas (IN, EXISTs, ALL, SOME, ANY)

Algumas consultas precisam que os valores existentes no banco de dados sejam buscados e depois usados em uma condição de comparação. SQL provê um mecanismo para aninhamento de subconsultas. Uma subconsulta é uma expressão do tipo SELECT-FROM-WHERE que é aninhada dentro de outra consulta. As aplicações mais comuns para as subconsultas são testes para membros de conjuntos, comparação de conjuntos e cardinalidade de conjuntos.

Sempre que uma condição na cláusula WHERE de uma consulta aninhada referencia algum atributo de uma relação declarada na consulta externa, as duas consultas são consideradas correlacionadas. Vejamos um exemplo. Vamos encontrar todos os clientes que possuem conta e empréstimo em um banco.


```
SELECT DISTINCT nome_cliente

FROM devedor

WHERE nome_cliente IN (select nome_cliente

    from depositante)
```

Observem que utilizamos acima o construtor **IN**, ele testa se um valor existe no outro subconjunto. Outros construtores importantes são o **UNIQUE** que testa se a subconsulta tem alguma tupla repetida no seu resultado; o **EXISTS** que retorna o valor TRUE se a subconsulta usada como argumento é "não vazia".

É possível, ainda, usar a palavra-chave **NOT** em conjunto com os operadores EXISTS e IN. Nestes casos o operador **NOT IN** verifica se o valor não existe em outro subconjunto. Semelhantemente podemos usar o NOT EXISTS que retorna um valor booleano para aceitação ou não da tupla em questão, caso o argumento da subconsulta seja vazio.

Precisamos ainda entender as palavras chave **ALL**, **ANY** e **SOME**. A norma padrão não trata da palavra SOME, mas ela funciona da mesma forma que a ANY. A utilização dessas palavras serve para comparar o valor da subconsulta externa com todos os valores da subconsulta interna. É possível, por exemplo, saber se um valor é maior que todos os valores da subconsulta. Vejam abaixo.

```
SELECT Name

FROM Product

WHERE ListPrice >= ALL

   (SELECT ListPrice

    FROM Product
```

Uma classificação para as subconsultas é se eles retornam uma ou múltiplas linhas. Algumas dicas devem ser levadas em consideração quando estamos elaborando subconsultas: (1) coloque subconsultas entre parênteses; (2) subconsultas devem ocupar o lado direito das comparações condicionais para facilitar a legibilidade; e (3) use operadores de única linha com subconsultas de uma **única linha** (=, >, >=, <, <=, <>) e operadores de **múltiplas linhas** (ALL e ANY) com subconsultas de várias linhas.

Funções agregadas (GROUP BY e HAVING)

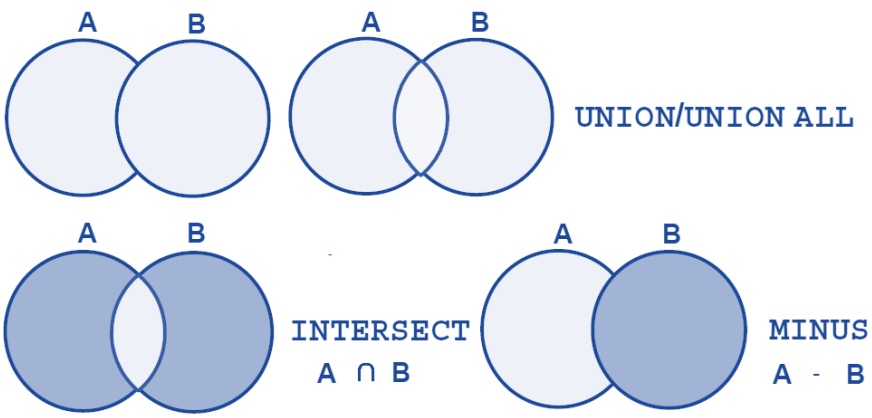
Vamos conhecer agora as funções agregadas que operam sobre conjuntos de valores de uma coluna de uma relação e retornam um valor para cada conjunto, são elas: COUNT (), SUM (), MAX (), MIN (), AVG (). Quando utilizamos funções agregadas, devemos utilizar o GROUP BY para os atributos na cláusula SELECT que não aparecem como parâmetros nas funções agregadas e a opção HAVING para predicados que são aplicados após a formação dos grupos.

Outra opção de subconsultas que retornam uma **única linha** é quando usamos funções de agregação como MIN e MAX. Vejam o exemplo abaixo para entendermos melhor a ideia:

```
SELECT last_name, job_id, salary
FROM   employees
WHERE  salary =              2500
                (SELECT MIN(salary)
                 FROM   employees);
```

Operações de conjuntos

As operações de conjuntos **UNION**, **INTERSECT** e **EXCEPT** operam em relações e correspondem às operações ∪, ∩ e - (diferença) da álgebra relacional. Estas operações eliminam as duplicatas. Se desejarmos obter as repetições, devemos explicitar por meio da forma **UNION ALL**, **INTERSECT ALL** e **EXCEPT ALL**.



É interessante perceber que a união é feita sobre o resultado de duas relações, ou seja, é possível duas consultas sobre tabelas distintas passarem por uma operação de união. É importante entender, porém, que os atributos das duas relações que participam da operação devem operar sobre os mesmos domínios. Vejamos um exemplo. Suponha que você queira encontrar todos os clientes que possuam um empréstimo, uma conta ou ambos.

```
(SELECT nome_cliente FROM depositante ) UNION (select
nome_cliente FROM devedor)
```

Junção

Por fim, vamos falar da composição de relações utilizando junção ou JOIN. As operações de junção tomam duas relações e retornam como resultado outra relação. São normalmente usadas na cláusula ***from*** e devem ser feitas seguindo uma condição e um tipo de junção. A **condição de junção** define quais tuplas das duas relações apresentam correspondência, e quais atributos serão apresentados no resultado de uma junção. O **tipo de junção** define como as tuplas em cada relação que não possuam nenhuma correspondência (baseado na condição de junção) com as tuplas da outra relação devem ser tratadas.

Observem a figura com as condições e os tipos de junção abaixo:

RESUMINDO

Tipos de junção
inner join
left outer join
rigth outer join
full outer join

Condições de junção
natural
on <predicate>
using (A_1, A_2,..., A_n)

Quando utilizamos a condição de junção **natural,** o SQL basicamente vai utilizar os atributos das duas relações que possuem os mesmos nomes. Caso não deseje utilizar todos os atributos com nomes iguais, você pode restringi-los utilizando a condição **using,** passando como argumentos apenas os atributos que você deseja.

Vejam um exemplo abaixo do uso do NATURAL JOIN E do USING:

```
SELECT department_id, department_name,
       location_id, city
FROM   departments
NATURAL JOIN locations ;
```

```
SELECT employee_id, last_name,
       location_id, department_id
FROM   employees JOIN departments
USING (department_id) ;
```

Apenas uma consideração importante: você não deve usar qualificadores para as colunas que serão utilizadas na cláusula USING. Isso pode ocasionar um erro na execução da requisição (no caso do Oracle: ORA-25154).

Após esse breve resumo da operação de SELECT, passaremos agora para analisar os demais comandos DML.

INSERT, UPDATE e DELETE

O comando INSERT INTO é utilizado para inserir novos registros em uma tabela. Sua sintaxe permite que você defina os valores para colunas de duas formas. Na primeira você não especifica os nomes das colunas nas quais os valores serão inseridos. Neste caso, o SGBD vai relacionar os valores na mesma ordem em que foram definidos no comando CREATE. A segunda forma seria definir explicitamente os nomes das colunas e os valores. Vejam as duas opções a seguir:

TOME NOTA!

INSERT INTO *table_name*
VALUES (*value1*,*value2*,*value3*,...);

INSERT INTO *table_name* (*column1*,*column2*,*column3*,...)
VALUES (*value1*,*value2*,*value3*,...);

No ORACLE, se você quiser passar como parâmetro o valor corrente da data e hora é possível fazer uso da função SYSDATE. Outra possibilidade é usar como valores do comando INSERT uma subconsulta. Neste caso, você não precisa usar a cláusula VALUES e o número de coluna da subconsulta deve ser igual ao número de colunas do INSERT. Vejamos um exemplo para ajudar na fixação do assunto:

```
INSERT INTO sales_reps(id, name, salary, commission_pct)
  SELECT employee_id, last_name, salary, commission_pct
  FROM   employees
  WHERE  job_id LIKE '%REP%';

4 rows inserted
```

Outro comando que compõe a linguagem é o UPDATE. Utilizado para atualizar registros em uma tabela, ele basicamente define, na sua cláusula WHERE, quais linhas ou registros devem ser atualizados. Caso nenhuma verificação seja feita, todas as linhas serão atualizadas. As mudanças a serem executas são definidas na cláusula SET. Vejam a sintaxe do comando abaixo:

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE some column=some value;
```

Valores para um conjunto de linhas específicas são atualizados ou modificados se você definir os critérios na cláusula WHERE. Se você quiser atribuir à coluna o valor NULL, basta usar SET nome_da_coluna = NULL. Lembre-se de que, caso nenhum critério for definido na cláusula WHERE, todas as linhas são atualizadas. Veja alguns exemplos do comando UPDATE:

```
UPDATE employees
SET    department_id = 50
WHERE  employee id = 113;
1 rows updated
```

```
UPDATE   copy_emp
SET      department_id = 110;
22 rows updated
```

Vejamos algumas informações sobre o DELETE. A instrução DELETE é usada para deletar linhas de uma tabela. Veja que, se nenhuma condição for definida na cláusula WHERE do comando, todas as linhas serão removidas da tabela. A sintaxe do comando é apresentada a seguir:

```
DELETE FROM table_name;
WHERE some column=some value:
```

Outro comando definido como **DDL,** mas que tem como funcionalidade remover, de forma rápida e fácil, todas as linhas de uma tabela, é o comando TRUNCATE. Ele remove todo conteúdo da tabela, zera os índices e as sequências associadas. Para tal, basta utilizar o comando DDL: TRUNCATE TABLE nome_tab;

```
TRUNCATE [ TABLE ] [ ONLY ] name [ * ] [, ... ]
    [ RESTART IDENTITY | CONTINUE IDENTITY ] [ CASCADE | RESTRICT ]
```

Acima, foi possível observar o comando TRUNCATE com a sintaxe específica do POSTGRES. Nele é possível remover um conjunto de tabelas ao mesmo tempo. O primeiro parâmetro permite reiniciar, ou não, as sequências associadas às colunas das tabelas cujos dados serão removidos, usando, respectivamente, o **restart identity** e **continue identity**. O outro parâmetro, **cascade** ou **restrict**, vai aplicar o comando truncate às tabelas que têm relacionamento ou chaves estrangeiras referenciando uma das tabelas removidas.

Principais Comandos

Chegamos a parte que eu considero mais importante da aula. Enquanto que para os aplicativos de suítes para escritório (Word, Excel, PowerPoint, etc) os pontos mais importantes são os menus e os atalhos, na aula de SQL o ponto principal são os comandos.

Consulta

SELECT

Comando utilizado para selecionar dados de um banco de dados.

Os dados retornados são salvos em uma tabela temporária.

Para selecionar dados de alguns campos da tabela, a sintaxe é:

```
SELECT column1, column2, ...
FROM table_name;
```

Para selecionar todos os campos da tabela, a sintaxe é:

```
SELECT * FROM table_name;
```

SELECT DISTINCT

A tabela, geralmente, contém vários valores duplicados.

O comando lista apenas os valores diferentes, sua sintaxe é:

```
SELECT DISTINCT column1, column2, ...
FROM table_name;
```

INSERT INTO

Comando é utilizado para inserir novos registros na tabela.

Para especificar nomes das colunas e valores a serem inseridos, a sintaxe é:

```
INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);
```

Para adicionar valores a todas as colunas da tabela, é preciso especificar as colunas na consulta SQL. A ordem dos valores deve estar na mesma ordem das colunas da tabela, sua sintaxe é:

```
INSERT INTO table_name
VALUES (value1, value2, value3, ...);
```

UPDATE

Comando é utilizado para modificar registros existentes na tabela, sua sintaxe é:

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

DELETE

Comando é utilizado para deletar registros existentes na tabela, sua sintaxe é:

```
DELETE FROM table_name
WHERE condition;
```

Se a condição WHERE não for especificada, todos os registros da tabela serão apagados.

É possível ainda deletar todos os registros da tabela sem deletar a tabela. Ou seja, a tabela mantém sua estrutura, seus atributos e seus índices. Sua sintaxe é:

```
DELETE FROM table_name;
```

CREATE TABLE

Comando utilizado para criar uma nova tabela em um banco de dados, sua sintaxe é:

```
CREATE TABLE table_name (
   column1 datatype,
   column2 datatype,
   column3 datatype,
  ....
```

```
);
```

Parâmetro das colunas: especifica os nomes das colunas na tabela

Parâmetro tipo de dados: especifica tipo de dado para cada coluna, como varchar, integer, date, etc.

ALTER TABLE

Comando utilizado para adicionar, deletar ou modificar colunas em uma tabela.

Para adicionar uma coluna em uma tabela, a sintaxe é:

```
ALTER TABLE table_name
ADD column_name datatype;
```

Para deletar uma coluna de uma tabela, a sintaxe é:

```
ALTER TABLE table_name
DROP COLUMN column_name;
```

Para alterar o tipo de dados de uma coluna de uma tabela, a sintaxe é:

```
ALTER TABLE table_name
ALTER (ou MODIFY) COLUMN column_name datatype;
```

DROP TABLE

Comando é utilizado para deletar uma tabela, a informação completa da tabela. Sua sintaxe é:

```
DROP TABLE table_name;
```

TRUNCATE TABLE

Comando é utilizado para deletar os dados de dentro da tabela, mas não a tabela. Sua sintaxe é:

```
TRUNCATE TABLE table_name;
```

CREATE INDEX

Comando é utilizado para utilizado para criar índices em uma tabela, valores duplicados são permitidos. Índices são utilizados para melhorar performance de consultas/buscas. Sua sintaxe é:

```
CREATE INDEX index_name
ON table_name (column1, column2, ...);
```

DROP INDEX

Comando utilizado para deletar um índice em uma tabela, sua sintaxe é:

```
DROP INDEX index_name;
```

Cláusula where

A cláusula WHERE é utilizada para filtrar registros, para extrair apenas registros que preencham a determinada condição. Sua sintaxe é:

```
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

A cláusula WHERE também é utilizada nos comandos UPDATE, DELETE, etc.

Operadores condicionais

Lógicos

A cláusula WHERE pode vir combinada dos operadores AND, OR e NOT.

Os operadores AND e OR são utilizados para filtrar registros com mais de uma condição:

O operador AND mostra os registros se todas as condições são verdadeiras, sua sintaxe é:

```
SELECT column1, column2, ...
FROM table_name
WHERE condition1 AND condition2 AND condition3 ...;
```

O operador OR mostra os registros se uma das condições é verdadeira, sua sintaxe é:

```
SELECT column1, column2, ...
FROM table_name
```

```
WHERE condition1 OR condition2 OR condition3 ...;
```

O operador NOT mostra os registros se nenhuma condição é verdadeira, sua sintaxe é:

```
SELECT column1, column2, ...
FROM table_name
WHERE NOT condition;
```

LIKE e NOT LIKE

O operador LIKE é utilizado na cláusula WHERE para pesquisar um padrão específico em uma coluna.

Existem dois curingas utilizados em conjunto com o operador LIKE:

% - o percentual representa zero, um, ou múltiplos caracteres

_ - O underscore representa um simples caracter

O percentual e o underscore podem ser utilizados combinados. Sua sintaxe é:

```
SELECT column1, column2, ...
FROM table_name
WHERE columnN LIKE pattern;
```

Também existe o LIKE utilizado junto ao NOT, veja um exemplo:

```
SELECT * FROM Customers
WHERE CustomerName NOT LIKE 'a%';
```

IN e NOT IN

O operador IN permite especificar múltiplos valores em uma cláusula WHERE, sua sintaxe é:

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (value1, value2, ...);
```

Ou:

```
SELECT column_name(s)
```

```
FROM table_name
WHERE column_name IN (SELECT STATEMENT);
```

O operador IN também pode ser utilizado em associação a NOT, veja um exemplo:

```
SELECT * FROM Customers
WHERE Country NOT IN ('Germany', 'France', 'UK');
```

Ordenação

ORDER BY

A palavra-chave ORDER BY ordena um resultado em ordem ascendente ou descendente.

ORDER BY ordena os registros de forma ascendente por padrão. Para ordenar os registros em ordem descendente, use a palavra-chave DESC. Sua sintaxe é:

```
SELECT column1, column2, ...
FROM table_name
ORDER BY column1, column2, ... ASC|DESC;
```

Agrupamento

GROUP BY

O comando GROUP BY é geralmente usado para agregar funções (COUNT, MAX, MIN, SUM, AVG) para agrupar resultados por uma ou mais colunas. Sua sintaxe é:

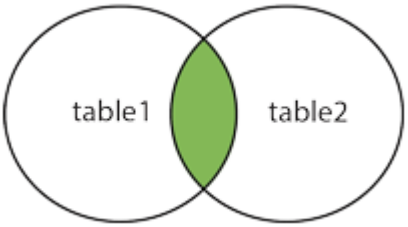
```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
ORDER BY column_name(s);
```

Junções (joins)

Uma cláusula JOIN é utilizada para combinar linhas a partir de duas ou mais tabelas, baseada em colunas relacionadas entre elas.

INNER JOIN

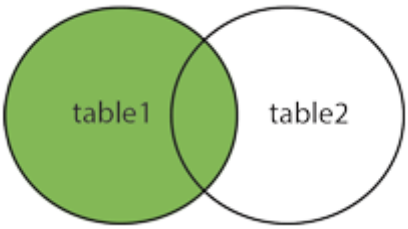
A palavra-chave INNER JOIN seleciona registros com valores correspondentes em ambas as tabelas.




```
SELECT column_name(s)
FROM table1
INNER JOIN table2 ON table1.column_name = table2.column_name;
```

LEFT JOIN

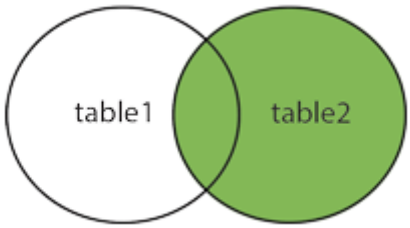
A palavra-chave LEFT JOIN retorna todos os registros da tabela da esquerda (table 1), e os registros correspondentes da tabela da direita (table 2). O resultado é NULL da tabela da direita, se não há registros correspondentes.




```
SELECT column_name(s)
FROM table1
LEFT JOIN table2 ON table1.column_name = table2.column_name;
```

RIGHT JOIN

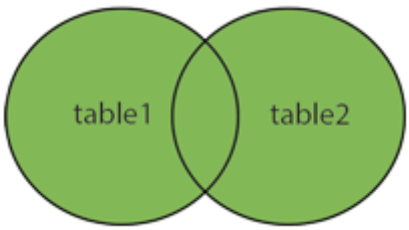
A palavra-chave RIGHT JOIN retorna todos os registros da tabela da direita (table 2), e os registros correspondentes da tabela da esquerda (table 1). O resultado é NULL da tabela da esquerda, se não há registros correspondentes.




```
SELECT column_name(s)
FROM table1
RIGHT JOIN table2 ON table1.column_name = table2.column_name;
```

FULL JOIN

A palavra-chave FULL OUTER JOIN retorna todos os registros quando há qualquer correspondência dos registros das tabelas da esquerda (table1) ou da direita (table2).

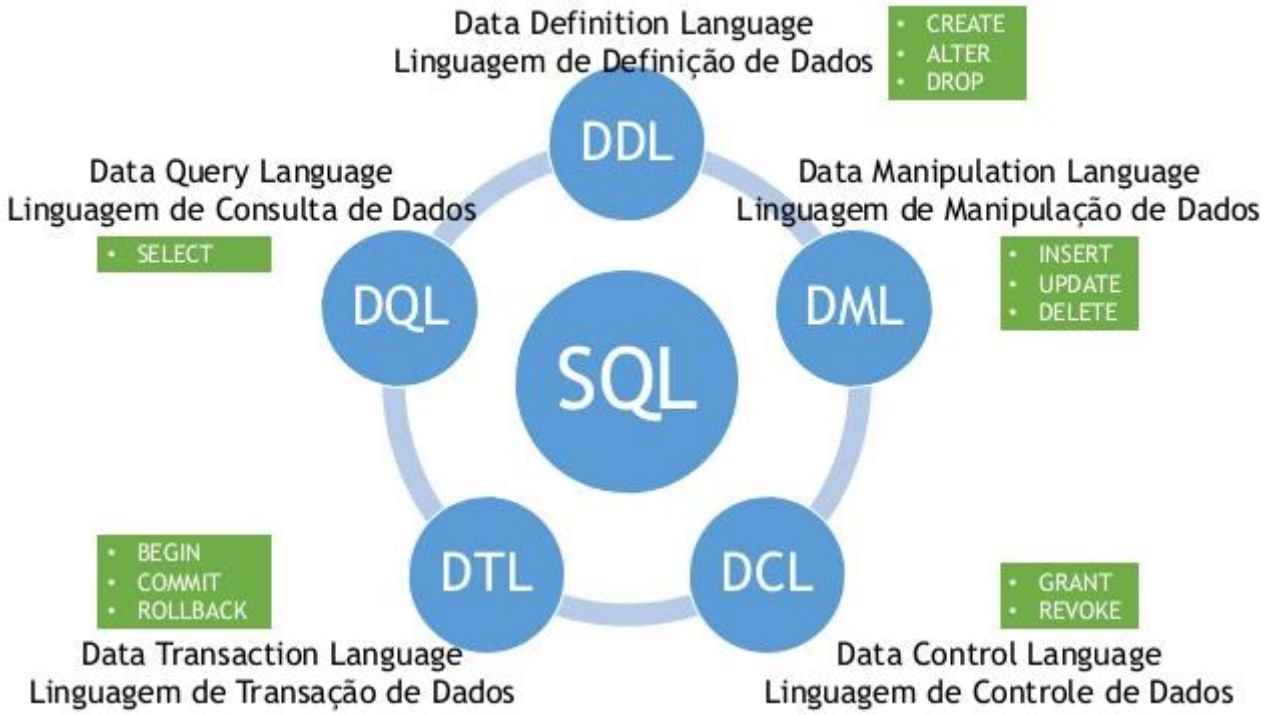



```
SELECT column_name(s)
FROM table1
FULL OUTER JOIN table2 ON table1.column_name = table2.column_name;
```

APOSTA ESTRATÉGICA

A ideia desta seção é apresentar os pontos do conteúdo que mais possuem chances de serem cobrados em prova, considerando o histórico de questões da banca em provas de nível semelhante à nossa, bem como as inovações no conteúdo, na legislação e nos entendimentos doutrinários e jurisprudenciais[3].

[3] Vale deixar claro que nem sempre será possível realizar uma aposta estratégica para um determinado assunto, considerando que às vezes não é viável identificar os pontos mais prováveis de serem cobrados a partir de critérios objetivos ou minimamente razoáveis.



Imprima o capítulo Aposta Estratégica separadamente e dedique um tempo para absolver tudo o que está destacado nessas duas páginas. Caso tenha alguma dúvida, volte ao Roteiro de Revisão e Pontos do Assunto que Merecem Destaque. Se ainda assim restar alguma dúvida, não hesite em me perguntar no fórum.

QUESTÕES ESTRATÉGICAS

Nesta seção, apresentamos e comentamos uma amostra de questões objetivas selecionadas estrategicamente: são questões com nível de dificuldade semelhante ao que você deve esperar para a sua prova e que, em conjunto, abordam os principais pontos do assunto.

A ideia, aqui, não é que você fixe o conteúdo por meio de uma bateria extensa de questões, mas que você faça uma boa revisão global do assunto a partir de, relativamente, poucas questões.

1. Ano: 2016 Banca: CESPE Órgão: TCE-SC Cargo: Auditor de TI

Com relação aos bancos de dados relacionais, julgue os próximos itens.

95 Denomina-se visão uma tabela única derivada de uma ou mais tabelas básicas do banco. Essa tabela existe em forma física e viabiliza operações ilimitadas de atualização e consulta.

Comentário: A definição de visão presente no padrão SQL/ANSI é de uma estrutura temporária que armazena informações advinda de uma ou mais tabelas. A visão não é armazenada fisicamente em disco e é removida ou apagada ao final da sua utilização. Sendo assim, a alternativa 95 encontra-se **incorreta**.

Gabarito: errado.

---

2. CESPE - Auditor Federal de Controle Externo/Controle Externo/Auditoria Governamental/2015

Acerca de bancos de dados relacionais, julgue o item subsequente.

88 Os bancos de dados relacionais são constituídos de três componentes: uma coleção de estrutura de dados (relações ou tabelas), uma coleção de operadores (linguagem SQL) e uma coleção de restrições de integridade (conjunto consistente de estados de base de dados e de alterações de estados).

Comentário: Os bancos de dados relacionais são compostos por estruturas de dados, que representam as informações que se deseja **armazenar e relacionar**. Para **manipular** (**selecionar**, **criar**, **apagar**, **alterar**, etc.) essas informações estruturadas, é necessário que um conjunto de operadores esteja disponível para os programadores e administradores da base de dados.

Por fim, é necessário que os bancos de dados suportem um conjunto de restrições de integridade, que serão responsáveis por **impedir que alterações indevidas** (ex.: apagar um registro sem antes tomar cuidado para as chaves estrangeiras que apontam para esse dado) sejam efetuadas e causem **inconsistências** nas informações estruturas no banco de dados.

Gabarito: certo.

---

3. CEBRASPE (CESPE) - Auditor Municipal de Controle Interno (CGM João Pessoa)/Tecnologia da Informação/Desenvolvimento de Sistemas/2018

A respeito de bancos de dados, julgue o item a seguir.

O MySQL Utilities é um pacote de utilitários voltados à manutenção e à administração de servidores MySQL. Cada um desses utilitários encapsula comandos primitivos e os agrupa para que possam ser utilizados no cumprimento de operações compostas a partir da execução de um único comando.

Comentários

O MySQL Workbench Manual traz a seguinte definição do MySQL UTILITIES: *"MySQL Utilities is a package of utilities that are used for maintenance and administration of MySQL servers. These utilities encapsulate a set of primitive commands, and bundles them so they can be used to perform macro operations with a single command."* Note que a assertiva é basicamente uma tradução dessa definição.

Gabarito: certo.

---

4. CEBRASPE (CESPE) - Auditor Municipal de Controle Interno (CGM João Pessoa)/Tecnologia da Informação/Desenvolvimento de Sistemas/2018

```
SELECT c.Nome
FROM CONTRIBUINTE as c, IMOVEL as i ORDER BY
  i.Valor_IPTU DESC LIMIT 10
WHERE c.CPF_CNPJ=i.CPF_CNPJ_Proprietario
```

Com base no trecho de código apresentado para execução pelo SGBD MySQL, julgue o item a seguir.

Quando executado, o código retornará os nomes dos dez contribuintes com maior valor atribuído de IPTU, considerando a soma dos valores de IPTU de todos os imóveis registrados nos nomes desses contribuintes.

Comentários

O código apresentado retornará os nomes de 10 contribuintes com maior valor atribuído de IPTU. Acontece que não considera a soma dos valores de IPTU por não haver um agrupamento (group by) e uma somatória (sum). Portanto, a assertiva está incorreta.

Gabarito: errado.

---

5. CEBRASPE (CESPE) - Analista Judiciário (STM)/Apoio Especializado/Análise de Sistemas/2018

Um sistema gerenciador de banco de dados (SGBD.) instalado no Linux deve ser configurado de modo a permitir os seguintes requisitos:

I no máximo, 1000 conexões simultâneas;

II somente conexões originadas a partir do servidor de aplicação com IP 10.10.10.2.

Tendo como referência essas informações, julgue o seguinte item.

Caso o SGBD instalado seja o MySQL 5.7, para atendimento dos requisitos I e II, deve-se modificar o arquivo my.cnf, alterando-se os parâmetros max_user_connections para 1000 e connection_source para o IP fornecido; e reiniciar o serviço do SGBD.

Comentários

Não existe o parâmetro connection_source no arquivo my.cnf. É possível restringir os endereços IP que acessam o banco usando o parâmetro bind-address. O parâmetro que limita a quantidade de requisições por usuários é a max_user_connections, mas a questão cobra o número máximo de conexões simultâneas, então o parâmetro correto seria max_connections.

Gabarito: errado.

---

6. CEBRASPE (CESPE) - Analista Judiciário (STM)/Apoio Especializado/Análise de Sistemas/2018

Julgue o item que se segue, a respeito do processamento de transações e otimização de desempenho do SGBD e de consultas SQL.

No MySQL 5.6, o modo padrão de execução das transações é autocommit, o qual faz que as mudanças realizadas se tornem permanentes após a execução bem-sucedida desse comando; entretanto, esse modo será desabilitado implicitamente, se uma série de instruções for iniciada por meio do comando START TRANSACTION.

Comentários

De acordo com o manual do MySQL 5.6, o MySQL inicia a sessão para cada nova conexão com o **autocommit ativado**, então o MySQL faz um COMMIT após cada instrução SQL, se essa instrução não retornar um erro. Com START TRANSACTION, o **autocommmit permanece desativado** até que você finalize a transação com COMMIT ou ROLLBACK. O modo do autocommit reverte para o estado anterior após a transação.

Gabarito: certo.

---

7. CEBRASPE (CESPE) - Analista Judiciário (STM)/Apoio Especializado/Análise de Sistemas/2018

Julgue o item que se segue, a respeito do processamento de transações e otimização de desempenho do SGBD e de consultas SQL.

No MySQL 5.6, o banco de dados information_schema guarda dados estatísticos e eventos para serem utilizados caso se queira encontrar problemas de velocidade de acesso aos dados e(ou) problemas de integridades no SGBD.

Comentários

O information_schema fornece acesso, na forma de readonly, do metadados do banco de dados, informações sobre o servidor MySQL, como o nome de um banco de dados ou tabela, o tipo de dados de uma coluna ou privilégios de acesso.

A questão apresentada está mais relacionada ao banco de dados performance_schema onde você pode, por exemplo, consultar as tabelas para ver informações em tempo real sobre as características de desempenho de seu servidor e os aplicativos que estão sendo executados.

Gabarito: errado.

---

8. CEBRASPE (CESPE) - Auditor de Controle Interno (COGE CE)/Auditoria/Tecnologia da Informação/2019

```
ALTER TABLE dbo.Ordens ADD CONSTRAINT
DFT_Ordens_ordemts DEFAULT(SYSDATETIME()) FOR
ordemts;
```

O código precedente modifica a tabela Ordens (composta por ordens de venda e ordemts – data/hora do pedido), em um banco de dados MS-SQL Server.

Após essa modificação, sempre que for inserida uma nova ordem de venda,

a) se não for definido um valor para o campo ordemts, o sistema invocará a função SYSDATETIME.

b) será obrigatório o preenchimento manual do campo ordemts.

c) se não for definido um valor para o campo ordemts, este ficará Null.

d) se for definido um valor para o campo ordemts, o banco de dados retornará erro de dado não permitido.

e) se não for definido um valor Null para o campo ordemts, o banco de dados retornará erro de dado não permitido.

Comentários

O comando apresentado na questão está alterando a tabela Ordens, adicionando uma nova restrição ao definir que o valor padrão do atributo ordemts seja a data e hora do sistema no momento da inserção, caso não seja informado nenhum valor. Vamos analisar as alternativas:

Na alternativa A temos, *se não for definido um valor para o campo ordemts, o sistema invocará a função SYSDATETIME* e é exatamente isso que o comando faz ao utilizar a cláusula DEFAULT: define um valor padrão para o atributo ordemts, caso ele não seja informado. O valor padrão é o resultado da função SYSDATETIME, que retorna a hora e a data do sistema.

Gabarito: alternativa A.

---

9. CEBRASPE (CESPE) - Analista Judiciário (TJ AM)/Analista de Sistemas/2019

A respeito de bancos de dados relacionais, julgue o item a seguir.

Em SQL, o comando RIGHT OUTER JOIN exibe a união entre duas tabelas, apresentando as linhas da segunda tabela que também existem na primeira tabela, descartando-se as demais situações.

Comentários

Utilizamos o operador JOIN quando temos que juntar tabelas que guardam uma correspondência entre si. Essa correspondência é representada através da chave estrangeira, que é um atributo que aponta para a chave primária de outra tabela.

O comando RIGHT OUTER JOIN (ou simplesmente RIGHT JOIN) retorna as linhas da segunda tabela que também existem na primeira tabela e as linhas da segunda tabela que não existem na primeira tabela.

Gabarito: errado.

---

10. CEBRASPE (CESPE) - Assistente Judiciário (TJ AM)/Suporte ao Usuário de Informática/2019

Acerca de sistema gerenciador de banco de dados, do tuning e da segurança em banco de dados, julgue o item subsequente.

O comando a seguir, após sua execução, restringirá o acesso do usuário tec_infor2_tjam à tabela processo.

```
GRANT update, delete, insert ON processo TO
tec_infor2_tjam;
```

Comentários

O comando acima está garantindo ao usuário tec_infor2_tjam o privilégio de executar as operações UPDATE, DELETE e INSERT na tabela processo.

Percebam que o comando, após sua execução, AMPLIARÁ o acesso do usuário tec_infor2_tjam à tabela processo, já que agora ele pode executar atualizações, remoções e inserções.

Gabarito: errado.

---

11. CEBRASPE (CESPE) - Analista Judiciário (STM)/Apoio Especializado/Análise de Sistemas/2018

```
1. CREATE TABLE IF NOT EXISTS 'software' (
2.   'id' int NOT NULL,
3.   'nome' varchar(70) NOT NULL,
4.   PRIMARY KEY ('id')
5. ) DEFAULT CHARSET=utf8;

6. INSERT INTO 'software' ('id', 'nome') VALUES
7.   ('1', 'Programa ABC'),
8.   ('2', 'Programa WYZ'),
9.   ('3', 'Programa DFG');

10. CREATE TABLE IF NOT EXISTS 'vsoftware' (
11.   'idsoft' int UNSIGNED NOT NULL REFERENCES
    software(id.),
12.   'versao' int(3) NOT NULL,
13.   'descricao' varchar(70) NOT NULL,
14.   PRIMARY KEY ('idsoft','versao'));

15. INSERT INTO 'vsoftware'('idsoft', 'versao',
    'descricao') VALUES
16.    ('1', '1', 'criacao do programa.'),
17.    ('2', '1', '1a versao.'),
18.    ('1', '2', 'atualizacao na tela A.'),
```

19. ('1', '3', 'adicao da tela C.'),
20. ('2', '2', 'adicao da tela D.');

Com base nos comandos MySQL 5.6 precedentes, julgue o item a seguir.

Especialmente devido à expressão na linha 11, o comando a seguir, após executado, retornará três registros.

```
SELECT a.idsoft, a.versao, a.descricao
FROM 'vsoftware' a
INNER JOIN (
  SELECT idsoft, MAX(versao) versao
  FROM 'vsoftware'
  GROUP BY idsoft) b ON a.idsoft = b.idsoft
AND a.versao = b.versao;
```

Comentários

A consulta interna agrupa por idsoft e cada software (1 e 2) vai retornar sua última versão - MAX(versao), ou seja, versão 3 para o idsoft 1 e versão 2 para o idsoft 2.

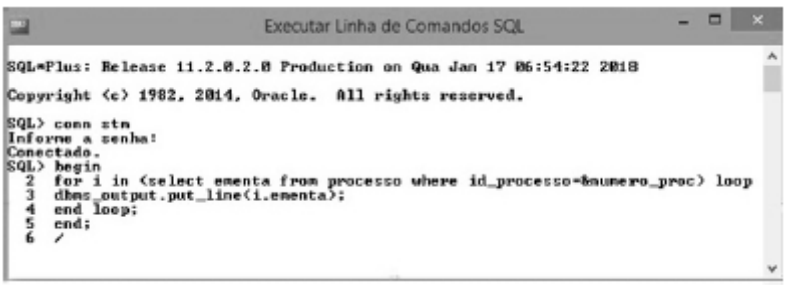
Como há apenas dois softwares, o resultado intermediário apresenta apenas duas tuplas.

O INNER JOIN compara o valor da versão e do id na clausula ON, o que ainda mantém apenas duas tuplas no resultado.

Gabarito: errado.

---

12. CEBRASPE (CESPE) - Técnico Judiciário (STM)/Apoio Especializado/Programação de Sistemas/2018



Considerando as informações apresentadas na figura precedente (captura de tela de uma sessão do SQL*Plus), relativas a comandos SQL, julgue o item que se segue.

A instrução contida na linha 3 possibilita a inserção de informações em um buffer que poderá ser lido por outro procedimento ou pacote.

Comentários

O pacote dbms_output permite que seja enviado mensagens a partir de Stored Procedures, Packages e Triggers. O pacote é especialmente útil para exibir informações de depuração do PL / SQL.

A procedure .put_line permite que seja colocado informações em um buffer que pode ser lido por outra Stored Procedure, Package ou Trigger.

Gabarito: certo.

...

Forte abraço e bons estudos.

"Hoje, o 'Eu não sei', se tornou o 'Eu ainda não sei'"

(Bill Gates)

Thiago Cavalcanti

Face: www.facebook.com/profthiagocavalcanti
Insta: www.instagram.com/prof.thiago.cavalcanti
YouTube: youtube.com/profthiagocavalcanti

ESSA LEI TODO MUNDO CONHECE:
PIRATARIA É CRIME.

Mas é sempre bom revisar o porquê e como você pode ser prejudicado com essa prática.

1. 

Professor investe seu tempo para elaborar os cursos e o site os coloca à venda.

2. 

Pirata divulga ilicitamente (grupos de rateio), utilizando-se do anonimato, nomes falsos ou laranjas (geralmente o pirata se anuncia como formador de "grupos solidários" de rateio que não visam lucro).

3. 

Pirata cria alunos fake praticando falsidade ideológica, comprando cursos do site em nome de pessoas aleatórias (usando nome, CPF, endereço e telefone de terceiros sem autorização).

4. 

Pirata compra, muitas vezes, clonando cartões de crédito (por vezes o sistema anti-fraude não consegue identificar o golpe a tempo).

5. 

Pirata fere os Termos de Uso, adultera as aulas e retira a identificação dos arquivos PDF (justamente porque a atividade é ilegal e ele não quer que seus fakes sejam identificados).

6. 

Pirata revende as aulas protegidas por direitos autorais, praticando concorrência desleal e em flagrante desrespeito à Lei de Direitos Autorais (Lei 9.610/98).

7. 

Concurseiro(a) desinformado participa de rateio, achando que nada disso está acontecendo e esperando se tornar servidor público para exigir o cumprimento das leis.

8. 

O professor que elaborou o curso não ganha nada, o site não recebe nada, e a pessoa que praticou todos os ilícitos anteriores (pirata) fica com o lucro.

Deixando de lado esse mar de sujeira, aproveitamos para agradecer a todos que adquirem os cursos honestamente e permitem que o site continue existindo.