Estratégia Concursos

Aula 02

TJ-RO (Analista Judiciário - Analista de Sistemas) Passo Estratégico de Conhecimentos Específicos

Autor:
Fernando Pedrosa Lopes

13 de Outubro de 2024

LINGUAGEM SQL

Sumário

CONTEÚDO

Conceitos de SQL. Linguagem de Definição de Dados (DDL). Tipos de Dados. Restrições de Integridade. Linguagem de Manipulação de Dados (DML). Views. Índices. Segurança (DCL).

ANÁLISE ESTATÍSTICA

Inicialmente, convém destacar o percentual de incidência do assunto, dentro da disciplina **Banco de Dados e Business Intelligence** em concursos/cargos similares. Quanto maior o percentual de cobrança de um dado assunto, maior sua importância.

Obs.: *um mesmo assunto pode ser classificado em mais de um tópico devido à multidisciplinaridade de conteúdo*.

Assunto	Relevância na disciplina em concursos similares
SQL	**21.6 %**
BI (Business Intelligence)	**9.0 %**
DW - Data Warehouse	**7.2 %**
SQL Server	**7.2 %**
Oracle	**6.3 %**
Banco de Dados Multidimensionais	**5.4 %**
Data Mining	**5.4 %**
Administração de banco de dados	**3.6 %**
Banco de Dados	**2.7 %**
Formas normais	**2.7 %**
ETL (Extract Transform Load)	**2.7 %**
Banco de Dados Relacionais	**2.7 %**
Arquitetura de Banco de Dados	**1.8 %**
SGBD - Sistema de Gerenciamento de Banco de Dados	**1.8 %**
OLAP (On-line Analytical Processing)	**1.8 %**
Segurança	**1.8 %**

MS-Access	**1.8 %**
Modelo relacional	**1.8 %**
Metadados e Metainformação	**1.8 %**
Álgebra relacional	**0.9 %**
Banco de Dados Paralelos e Distribuídos	**0.9 %**
Gerência de Transações	**0.9 %**
Modelagem de dados	**0.9 %**
Gatilhos (Triggers)	**0.9 %**
DER - Diagrama de Entidade e Relacionamento	**0.9 %**
Visão (View)	**0.9 %**
Banco de Dados Textuais	**0.9 %**
Índices	**0.9 %**
PostgreSQL	**0.9 %**
MySQL	**0.9 %**
Big Data	**0.9 %**

GLOSSÁRIO DE TERMOS

Faremos uma lista de termos que são relevantes ao entendimento do assunto desta aula. Caso tenha alguma dúvida durante a leitura, esta seção pode lhe ajudar a esclarecer.

SQL: Linguagem de Consulta Estruturada, usada para gerenciar e manipular bancos de dados relacionais.

DDL: Linguagem de Definição de Dados, uma sublinguagem de SQL usada para definir estruturas de dados e objetos de banco de dados.

DML: Linguagem de Manipulação de Dados, uma sublinguagem de SQL usada para manipular dados dentro de objetos de banco de dados.

DCL: Linguagem de Controle de Dados, uma sublinguagem de SQL usada para controlar o acesso a dados no banco de dados.

DTL: Linguagem de Transação de Dados, uma sublinguagem de SQL usada para gerenciar transações de banco de dados.

Dialeto SQL: Uma variante de SQL que possui características e funcionalidades adicionais que não são padronizadas pela SQL padrão.

Tabela: Um objeto de banco de dados que organiza os dados em linhas e colunas.

Linha: Um registro único de dados em uma tabela. Também conhecido como tupla ou registro.

Coluna: Uma categoria vertical de dados em uma tabela. Também conhecida como campo.

CREATE (DDL): Um comando DDL usado para criar novos objetos de banco de dados.

DROP (DDL): Um comando DDL usado para excluir objetos existentes do banco de dados.

ALTER (DDL): Um comando DDL usado para modificar a estrutura de um objeto existente no banco de dados.

RENAME (DDL): Um comando DDL usado para renomear um objeto existente no banco de dados.

TRUNCATE (DDL): Um comando DDL usado para remover todos os registros de uma tabela, mas mantém a estrutura da tabela para uso futuro.

View: Uma representação virtual de uma tabela que é baseada em uma consulta SQL.

Índice: Uma estrutura de dados que melhora a velocidade das operações de recuperação de dados em um banco de dados.

Tipo Texto: Um tipo de dado SQL usado para armazenar cadeias de caracteres.

Tipo Numérico: Um tipo de dado SQL usado para armazenar valores numéricos.

Tipo de Tempo: Um tipo de dado SQL usado para armazenar valores de data e hora.

Tipo Binário: Um tipo de dado SQL usado para armazenar dados binários.

Restrição de Integridade: Regras aplicadas a tabelas para preservar a precisão e a consistência dos dados.

Chave Primária: Uma coluna ou conjunto de colunas cujos valores identificam exclusivamente cada linha em uma tabela.

Chave Estrangeira: Uma coluna ou conjunto de colunas em uma tabela que é usado para estabelecer e impor um link entre os dados em duas tabelas.

Restrição de Unicidade: Uma regra imposta em uma coluna ou conjunto de colunas para garantir que os valores sejam únicos entre todas as linhas da tabela.

Restrição de não nulidade: Uma regra imposta em uma coluna para garantir que não possa conter valores NULL.

SELECT (DML): Um comando DML usado para selecionar dados de uma ou mais tabelas.

INSERT (DML): Um comando DML usado para inserir novos registros em uma tabela.

UPDATE (DML): Um comando DML usado para modificar registros existentes em uma tabela.

DELETE (DML): Um comando DML usado para excluir registros existentes de uma tabela.

Group By: Uma cláusula SQL usada para agrupar registros com valores semelhantes em resultados de consultas.

Alias: Um nome temporário dado a uma tabela ou coluna para a duração de uma consulta SQL.

Distinct: Uma cláusula SQL usada para remover valores duplicados de um conjunto de resultados.

Operadores: Símbolos ou palavras-chave usados para especificar operações a serem realizadas em dados.

Union: Operação de conjunto SQL que combina os resultados de duas ou mais consultas SELECT e remove duplicatas.

Intersect: Operação de conjunto SQL que retorna os registros comuns de duas ou mais consultas SELECT.

Except: Operação de conjunto SQL que retorna registros de uma consulta SELECT que não existem na outra consulta SELECT.

Inner Join: Operação de junção SQL que retorna apenas registros que têm correspondências em ambas as tabelas.

Left Join: Operação de junção SQL que retorna todos os registros da tabela esquerda e as correspondências da tabela direita.

Right Join: Operação de junção SQL que retorna todos os registros da tabela direita e as correspondências da tabela esquerda.

Full Join: Operação de junção SQL que retorna todos os registros quando há uma correspondência em uma das tabelas.

Função agregada: Funções SQL que operam em um conjunto de valores para fornecer um único valor resumido.

Stored Procedure: Um grupo pré-compilado de declarações SQL que são armazenadas no banco de dados.

Function: Um conjunto de instruções SQL que realiza uma tarefa específica e retorna um valor.

Trigger: Um tipo de procedimento armazenado que é executado automaticamente quando ocorre um evento específico no banco de dados.

Grant: Um comando DCL usado para conceder permissões a usuários ou roles.

Revoke: Um comando DCL usado para remover permissões concedidas a usuários ou roles.

Role: Um identificador de um conjunto específico de permissões que pode ser concedido a usuários ou outros roles.

ROTEIRO DE REVISÃO

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.

Introdução a SQL

Bem-vindos à nossa aula sobre SQL, uma habilidade essencial para qualquer pessoa que deseja trabalhar com bancos de dados ou se tornar um especialista em ciência de dados.

SQL, que significa "*Structured Query Language*" ou "**Linguagem de Consulta Estruturada**" em português, é uma linguagem de programação usada para se comunicar e manipular bancos de dados. Ela permite que você execute tarefas como criar, alterar, e excluir tabelas; inserir, atualizar, e excluir dados; e consultar dados de um ou mais bancos de dados.

SQL foi criado nos anos 70 pela IBM e tem sido um padrão na indústria de tecnologia da informação desde então. A maioria dos sistemas de gerenciamento de banco de dados relacional (RDBMS) usa SQL, incluindo MySQL, Oracle, Microsoft SQL Server, PostgreSQL, entre outros.

No mundo atual de Big Data e Análise de Dados, ter um conhecimento sólido de SQL é altamente benéfico. SQL permite que você extraia, analise, e manipule dados, habilidades que são altamente procuradas em muitas áreas, desde o desenvolvimento web até a ciência de dados.

Nesta aula, vamos aprender SQL, desde os conceitos básicos até tópicos mais avançados. O objetivo é que você tenha uma boa compreensão de SQL e seja capaz resolver questões de prova sobre este assunto.

Histórico e Importância de SQL

SQL foi criada no início dos anos 70 na IBM por Donald D. Chamberlin e Raymond F. Boyce. A linguagem foi originalmente chamada de SEQUEL (Structured English Query Language), mas devido a problemas de marca registrada, acabou sendo reduzida para SQL.

Os primeiros sistemas de gerenciamento de banco de dados relacional (RDBMS) surgiram no final dos anos 70 e início dos anos 80, e o SQL se tornou a linguagem padrão para tais sistemas. Em 1986, o Instituto Nacional Americano de Padrões (ANSI) e a Organização Internacional de Normalização (ISO) adotaram oficialmente a SQL como norma.

Apesar de sua longa história, SQL permanece extremamente relevante hoje. Ele é a linguagem padrão para interagir com bancos de dados, seja para consultar dados, atualizar registros, ou gerenciar estruturas de banco de dados.

Em termos de importância no cenário atual de gestão de bancos de dados, a SQL desempenha vários papéis fundamentais:

Ubiquidade: A maioria dos bancos de dados relacionais suporta SQL, tornando-se uma habilidade universalmente útil para profissionais de tecnologia. Além disso, muitos bancos de dados NoSQL também suportam variações de SQL.

Manipulação de Dados: SQL é uma ferramenta muito útil para manipular dados. Você pode inserir novos dados, atualizar e excluir dados existentes, bem como manipular a estrutura dos bancos de dados.

Recuperação de Informações: SQL permite consultar grandes volumes de dados de maneira eficiente. As consultas podem ser usadas para extrair exatamente os dados necessários a partir de um ou mais bancos de dados.

Análise de Dados: SQL tem uma gama de funções de agregação e operações de conjunto que a tornam útil para análises de dados. A linguagem é frequentemente usada em conjunto com outras ferramentas de análise de dados.

Integração com outras linguagens: SQL pode ser facilmente integrada com outras linguagens de programação, tornando-a uma ferramenta útil para desenvolvedores de software.

SQL e Bancos de Dados Relacionais

SQL e bancos de dados relacionais são intimamente conectados. Na verdade, SQL foi criado especificamente para interagir com bancos de dados relacionais. Então, para entender a conexão entre os dois, primeiro vamos relembrar o que é um banco de dados relacional.

Um banco de dados relacional é um tipo de banco de dados que organiza os dados em uma ou mais tabelas, ou "relações". Cada tabela é composta por colunas (campos) e linhas (registros). As tabelas são conectadas através de chaves primárias e estrangeiras, permitindo que os dados sejam combinados de formas complexas.

Aqui estão alguns pontos importantes sobre como o SQL se relaciona com bancos de dados relacionais:

Manipulação e Definição de Dados: SQL é usado para criar, alterar e excluir tabelas em um banco de dados relacional, operações conhecidas como Data Definition Language (DDL). SQL também é usado para inserir, atualizar, e deletar registros em uma tabela, operações conhecidas como Data Manipulation Language (DML).

Consultas: SQL é a linguagem padrão para extrair informações de um banco de dados relacional. Usando a instrução SELECT, você pode buscar registros específicos de uma ou mais tabelas.

Relações: Em um banco de dados relacional, as tabelas estão conectadas por meio de chaves. SQL é usado para definir essas conexões, e para realizar operações em múltiplas tabelas simultaneamente. Por exemplo, a operação JOIN em SQL permite combinar registros de duas ou mais tabelas.

Controle de Acesso: SQL também fornece comandos para gerenciar a segurança de um banco de dados relacional. Você pode usar SQL para criar contas de usuários, definir permissões e realizar outras tarefas de segurança.

Linguagem Declarativa versus Linguagem Procedural

SQL é classificada como uma linguagem declarativa. Isso significa que, ao usar SQL, você especifica **o que você deseja fazer, não como fazer**. Em outras palavras, você descreve o resultado desejado, e o sistema de banco de dados descobre como alcançar esse resultado.

Por exemplo, se você quiser selecionar todos os registros de uma tabela que atendem a uma determinada condição em SQL, você poderia escrever uma consulta como esta:

```sql
SELECT * FROM funcionarios WHERE salario > 50000;
```

Nesse caso, você está dizendo ao banco de dados que você deseja todos os registros da tabela "funcionarios" onde o "salario" é maior que 50000. Você não precisa especificar como o banco de dados deve encontrar esses registros. Isso é o que torna SQL uma linguagem declarativa.

Em contraste, linguagens procedurais, também conhecidas como linguagens imperativas, exigem que você especifique explicitamente como alcançar o resultado desejado, passo a passo.

Por exemplo, em uma linguagem procedural como Python, se você tivesse uma lista de funcionários representados como dicionários e quisesse filtrar aqueles com um salário maior que 50000, você teria que escrever um loop para iterar sobre a lista e uma instrução if para verificar a condição para cada funcionário, algo assim:

```python
funcionarios_com_salario_alto = []
for funcionario in funcionarios:
    if funcionario['salario'] > 50000:
        funcionarios_com_salario_alto.append(funcionario)
```

Nesse caso, você está explicitamente dizendo ao programa como realizar a tarefa: percorrer a lista de funcionários um por um e verificar o salário de cada um.

Cada abordagem tem suas vantagens. Linguagens declarativas, como SQL, tendem a ser mais concisas e expressivas para tarefas específicas, como consulta e manipulação de dados. Linguagens procedurais podem ser mais flexíveis e poderosas para tarefas gerais de programação.

Sublinguagens de SQL

SQL é composto por várias sublinguagens, cada uma destinada a um tipo específico de operação em bancos de dados. As quatro principais sublinguagens do SQL são DDL, DML, DCL e DTL:

DDL (Data Definition Language): DDL é usada para definir e gerenciar estruturas de dados em um banco de dados. Instruções DDL incluem:

- CREATE**:** Usado para criar novas tabelas e bancos de dados.
- ALTER**:** Usado para modificar a estrutura de tabelas e bancos de dados existentes. Por exemplo, você pode adicionar ou remover colunas de uma tabela.
- DROP**:** Usado para excluir tabelas e bancos de dados.
- TRUNCATE**:** Usado para remover todos os registros de uma tabela, mas mantém a estrutura da tabela para uso futuro.

DML (Data Manipulation Language): DML é usada para manipular os dados armazenados nas estruturas definidas pela DDL. Instruções DML incluem:

- SELECT: Usado para buscar e recuperar dados de uma ou mais tabelas.
- INSERT: Usado para adicionar novos registros a uma tabela.
- UPDATE: Usado para modificar registros existentes em uma tabela.
- DELETE: Usado para remover registros de uma tabela.

DCL (Data Control Language): DCL é usada para controlar o acesso aos dados no banco de dados. Instruções DCL incluem:

- GRANT: Usado para conceder privilégios de acesso a usuários.
- REVOKE: Usado para remover privilégios de acesso de usuários.

DTL (Data Transaction Language): Também conhecida como TCL (Transaction Control Language), a DTL é usada para gerenciar transações dentro do banco de dados. Instruções DTL incluem:

- BEGIN TRANSACTION: Usado para iniciar uma transação.
- COMMIT: Usado para salvar as alterações feitas em uma transação.
- ROLLBACK: Usado para desfazer as alterações se algo der errado na transação.
- SAVEPOINT: Usado para criar pontos na transação onde você pode rolar de volta se algo der errado.

Resumo das sublinguagens:

Sublinguagem	Propósito	Exemplos de comandos
DDL (Data Definition Language)	**Define e gerencia estruturas de dados em um banco de dados.**	**CREATE, ALTER, DROP, TRUNCATE**

DML (Data Manipulation Language)	**Manipula os dados armazenados nas estruturas definidas pela DDL.**	**SELECT, INSERT, UPDATE, DELETE**
DCL (Data Control Language)	**Controla o acesso aos dados no banco de dados.**	**GRANT, REVOKE**
DTL (Data Transaction Language)	**Gerencia transações dentro do banco de dados.**	**BEGIN TRANSACTION, COMMIT, ROLLBACK, SAVEPOINT**

Extensões (Dialetos) de SQL

Embora o SQL padrão seja uma linguagem declarativa, existem várias **extensões do SQL**, chamadas de **dialetos**, que adicionam funcionalidades procedurais. Esses dialetos permitem que você escreva blocos de código que incluem variáveis, loops, condicionais, exceções e outras construções de programação padrão. Duas das extensões mais populares são o **Transact-SQL (T-SQL) e o PL/SQL**.

Transact-SQL (T-SQL) é a extensão procedural do SQL usada pelo Microsoft SQL Server e pelo Sybase ASE. T-SQL amplia a funcionalidade do SQL padrão adicionando construções procedurais, como variáveis, controle de fluxo, loops e tratamento de erros. Também adiciona algumas extensões ao SQL para consultas e manipulação de dados.

Por exemplo, você pode usar o T-SQL para escrever uma *stored procedure* que calcula a média de salários para cada departamento em uma empresa, algo que seria difícil de fazer com o SQL padrão:

```
CREATE PROCEDURE CalcularMediaSalario
AS
BEGIN
    SELECT d.Nome, AVG(e.Salario) as MediaSalario
    FROM Departamento d
    JOIN Empregado e ON d.DepartamentoId = e.DepartamentoId
    GROUP BY d.Nome;
END;
```

PL/SQL (Procedural Language/SQL) é o dialeto procedural do SQL usado pelo Oracle Database. Assim como o T-SQL, o PL/SQL adiciona construções procedurais ao SQL, permitindo que você escreva blocos de código completos com variáveis, controle de fluxo, loops, e tratamento de erros. Além disso, o PL/SQL também suporta orientação a objetos.

Por exemplo, você pode usar PL/SQL para escrever uma *stored procedure* que aumenta o salário de um funcionário por uma certa porcentagem:

```
CREATE OR REPLACE PROCEDURE AumentarSalario (idFuncionario IN NUMBER, porcentagem IN NUMBER)
IS
   salarioAtual NUMBER(8,2);
BEGIN
   SELECT salario INTO salarioAtual FROM Funcionarios WHERE id = idFuncionario;
   salarioAtual := salarioAtual * (1 + porcentagem / 100);
   UPDATE Funcionarios SET salario = salarioAtual WHERE id = idFuncionario;
   COMMIT;
END;
```

Tanto T-SQL quanto PL/SQL são poderosos e permitem realizar tarefas que seriam difíceis ou impossíveis com o SQL padrão. No entanto, cada um é específico para o sistema de banco de dados que o usa, então o código escrito em T-SQL ou PL/SQL não é portável para outros sistemas de banco de dados.

Veja uma lista mais completa de outros dialetos procedurais de SQL específicos para alguns SGBD's:

Banco de Dados	Dialeto	Nome Completo
Oracle Database	PL/SQL	Procedural Language/SQL
Microsoft SQL Server	T-SQL	Transact-SQL
PostgreSQL	PL/pgSQL	Procedural Language/PostgreSQL
IBM DB2	SQL PL	SQL Procedural Language
MySQL	SQL/PSM	SQL/Persistent Stored Modules

Elementos Fundamentais de um Banco Relacional (revisão)

Antes de entrarmos nos detalhes de SQL, ou seja, suas sublinguagens, instruções, tipos de dado, operadores etc., convém lembrarmos os elementos fundamentais de um banco de dados relacional, isto é, os termos básicos: tabela, linha (registro) e coluna.

Banco de Dados

Um banco de dados é uma coleção organizada de dados. Ele é projetado para armazenar, gerenciar e recuperar informações de maneira eficiente. Um banco de dados pode ser simples, contendo apenas uma única tabela de dados, ou pode ser complexo, contendo centenas de tabelas inter-relacionadas.

Por exemplo, uma livraria pode ter um banco de dados que contém informações sobre livros, autores, clientes, pedidos, etc.

Tabela

Uma tabela é uma estrutura dentro de um banco de dados que organiza os dados em linhas e colunas, de forma semelhante a uma planilha. Cada tabela tem um nome único dentro do banco de dados e contém informações sobre uma entidade específica.

Por exemplo, no banco de dados da livraria, pode haver uma tabela chamada "Livros", que armazena informações sobre todos os livros disponíveis para venda.

Linha (Registro)

Uma linha, também chamada de registro, é uma única entrada em uma tabela de banco de dados. Cada linha contém um conjunto de valores relacionados que correspondem às colunas da tabela.

Por exemplo, na tabela "Livros", cada linha pode representar um livro específico, com valores para o título do livro, o autor, o preço, etc.

Coluna

Uma coluna em uma tabela de banco de dados representa um tipo específico de dado dentro da tabela. Cada coluna tem um nome único dentro da tabela e um tipo de dado específico.

Por exemplo, na tabela "Livros", pode haver colunas chamadas "Titulo", "Autor" e "Preco". A coluna "Titulo" pode ser do tipo texto, a coluna "Autor" pode ser do tipo texto e a coluna "Preco" pode ser do tipo numérico.

Aqui está um exemplo de como a tabela "Livros" pode parecer:

ID	Titulo	Autor	Preco
1	O Senhor dos Anéis	J.R.R. Tolkien	50
2	1984	George Orwell	40
3	Moby Dick	Herman Melville	45

Neste exemplo, "ID", "Titulo", "Autor" e "Preco" são colunas. Cada linha representa um livro diferente.

Linguagem de Definição de Dados (DDL)

A Linguagem de Definição de Dados, ou DDL (Data Definition Language), é um subconjunto da linguagem SQL que é usada para **definir e gerenciar as estruturas que contêm os dados**, como tabelas e índices, em um banco de dados. DDL é responsável pela "estrutura" do banco de dados.

Os comandos DDL permitem que você crie, altere e exclua estruturas de banco de dados. Os principais comandos DDL são descritos a seguir.

CREATE

CREATE é o comando usado para criar a estrutura de um banco de dados, uma tabela, índice, ou outros objetos. Neste primeiro exemplo, o comando cria um novo banco de dados chamado "Biblioteca":

```
CREATE DATABASE Biblioteca;
```

Já no próximo exemplo, comando SQL cria uma nova tabela chamada "Funcionarios":

```
CREATE TABLE Funcionarios (
    ID INT PRIMARY KEY,
    Nome VARCHAR(100),
    Salario DECIMAL(10,2)
);
```

Neste último exemplo, o comando cria um esquema chamado "Publicacoes":

```
CREATE SCHEMA Publicacoes;
```

Lembrando que um esquema é essencialmente um namespace: ele contém nomes de objetos (como nomes de tabelas) que são únicos dentro desse esquema, mas não necessariamente únicos dentro do banco de dados.

Depois de criar o esquema, você pode criar tabelas dentro dele usando a sintaxe <schema>.<table>. Por exemplo:

```
CREATE TABLE Publicacoes.Livros (
    ID INT PRIMARY KEY,
    Titulo VARCHAR(100),
    Autor VARCHAR(100)
);
```

DROP

O comando **DROP** é uma operação da Linguagem de Definição de Dados que permite a exclusão de um objeto existente no banco de dados. Este objeto pode ser uma tabela, um banco de dados ou um esquema. Aqui estão mais detalhes e exemplos de uso para cada caso:

DROP TABLE: Este comando é usado para excluir uma tabela existente. Quando uma tabela é excluída, todos os dados armazenados na tabela e a estrutura da tabela são removidos permanentemente do banco de dados. Por exemplo, para excluir a tabela "Funcionarios", você usaria:

```
DROP TABLE Funcionarios;
```

DROP DATABASE: Este comando é usado para excluir um banco de dados existente. Quando um banco de dados é excluído, todas as tabelas, esquemas e outros objetos no banco de dados são excluídos, e o próprio banco de dados é removido do sistema de gerenciamento de banco de dados (SGBD). Por exemplo, para excluir o banco de dados "Biblioteca", você usaria:

```
DROP DATABASE Biblioteca;
```

DROP SCHEMA: Este comando é usado para excluir um esquema existente. Um esquema é um conjunto de tabelas e outros objetos de banco de dados que são logicamente agrupados juntos. Normalmente, você precisa excluir todas as tabelas e outros objetos no esquema antes de poder excluir o próprio esquema. Por exemplo, para excluir o esquema "Publicacoes", você usaria:

```
DROP SCHEMA Publicacoes;
```

Esses comandos DROP devem ser usados com cautela, pois uma vez que um objeto é excluído com DROP, não pode ser recuperado. É importante se certificar sempre de ter backups adequados antes de executar comandos DROP.

ALTER

ALTER é o comando usado para alterar a estrutura de um objeto de banco de dados existente, como uma tabela. Você pode adicionar, modificar ou excluir colunas em uma tabela existente. Veja exemplos de uso a seguir:

Adicionar uma coluna à tabela:

Para adicionar uma nova coluna a uma tabela existente, use a cláusula ADD. Por exemplo, para adicionar uma coluna "Email" à tabela "Funcionarios", você usaria:

```
ALTER TABLE Funcionarios
ADD Email VARCHAR(255);
```

Modificar uma coluna existente:

Para modificar o tipo de dados de uma coluna existente, use a cláusula ALTER COLUMN. Por exemplo, para alterar o tipo de dados da coluna "Email" para um campo de texto de maior tamanho, você usaria:

```
ALTER TABLE Funcionarios
ALTER COLUMN Email VARCHAR(500);
```

Excluir uma coluna existente:

Para excluir uma coluna de uma tabela existente, use a cláusula DROP COLUMN. Por exemplo, para excluir a coluna "Email" da tabela "Funcionarios", você usaria:

```
ALTER TABLE Funcionarios
DROP COLUMN Email;
```

Renomear uma tabela:

Para renomear uma tabela existente, use a cláusula RENAME TO. Por exemplo, para renomear a tabela "Funcionarios" para "Empregados", você usaria:

```
ALTER TABLE Funcionarios
RENAME TO Empregados;
```

Nem todos os sistemas de gerenciamento de banco de dados (SGBD) suportam todas as variantes do comando ALTER. Por exemplo, alguns SGBD não suportam a cláusula ALTER COLUMN, ou possuem restrições para o uso do DROP COLUMN. Sempre verifique a documentação do SGBD para obter detalhes sobre o suporte ao comando ALTER.

RENAME

O comando **RENAME** é usado para renomear um objeto existente no banco de dados. O objeto pode ser uma tabela, coluna, índice, entre outros, dependendo do suporte do Sistema de Gerenciamento de Banco de Dados (SGBD).

Vamos ver como você pode usar o comando RENAME com alguns exemplos.

Renomear uma tabela:

Se você quiser renomear uma tabela, você pode fazer isso com o comando RENAME. Por exemplo, se você quiser renomear a tabela "Funcionarios" para "Empregados", você pode usar o seguinte comando:

```
ALTER TABLE Funcionarios
RENAME TO Empregados;
```

Renomear uma coluna:

Da mesma forma, você pode renomear uma coluna de uma tabela. Por exemplo, se você quiser renomear a coluna "Salario" para "Rendimento" na tabela "Empregados", você pode usar o seguinte comando:

```
ALTER TABLE Empregados
RENAME COLUMN Salario TO Rendimento;
```

Note que nem todos os SGBDs suportam a operação de renomear coluna diretamente. No SQL Server, por exemplo, você usaria a stored procedure sp_rename:

```
EXEC sp_rename 'Empregados.Salario', 'Rendimento', 'COLUMN';
```

Como sempre, a sintaxe e a funcionalidade dos comandos SQL podem variar entre diferentes SGBDs, então é sempre uma boa ideia consultar a documentação do SGBD específico que você está usando.

TRUCANTE

TRUNCATE é o comando usado para remover todos os registros de uma tabela, mas mantendo a estrutura da tabela para uso futuro. É uma forma mais eficiente de limpar os dados de uma tabela se você planeja reutilizar a estrutura da tabela.

```
TRUNCATE TABLE Funcionarios;
```

Todos os comandos DDL são "commitados" (instrução Commit) automaticamente, o que significa que eles não podem ser revertidos. Uma vez que você executa um comando DDL, as alterações feitas no banco de dados são permanentes.

A tabela a seguir resume os comandos DDL e seus significados:

Comando DDL	Propósito
CREATE	Usado para criar um novo objeto no banco de dados. Este objeto pode ser um banco de dados, tabela, índice, esquema, entre outros.
ALTER	Usado para modificar a estrutura de um objeto existente no banco de dados, como uma tabela ou esquema, sem precisar excluí-lo e recriá-lo.
DROP	Usado para excluir um objeto existente no banco de dados, como um banco de dados, tabela, índice, ou esquema.
TRUNCATE	Usado para remover todos os registros de uma tabela, mantendo sua estrutura para uso futuro. É uma forma eficiente de limpar os dados de uma tabela se você planeja reutilizar a estrutura da tabela.
RENAME	Usado para renomear um objeto existente no banco de dados. Este objeto pode ser uma tabela, coluna, índice, entre outros.

Criando Visões (VIEW)

Uma **VIEW** em SQL é uma **tabela virtua**l baseada no conjunto de resultados de uma instrução SQL. Ela contém linhas e colunas, assim como uma tabela real. As colunas em uma VIEW são colunas de uma ou mais tabelas reais no banco de dados.

Você pode adicionar funções SQL, WHERE e JOIN a uma VIEW e apresentá-la como se fosse uma tabela única. Geralmente uma VIEW é utilizada para:

- Reutilizar instruções SQL;
- Simplificar instruções SQL complexas;
- Proteger certos dados, permitindo que usuários acessem apenas um conjunto específico de dados;

- Fornecer uma visão personalizada de acordo com as necessidades dos usuários.

Exemplo de criação de uma VIEW

Suponha que temos uma tabela Funcionarios com as colunas ID, Nome, Cargo, Departamento, Salario.

Agora, queremos criar uma VIEW que contenha apenas Nome, Cargo e Departamento de todos os funcionários que trabalham no departamento de "Marketing". A instrução SQL para criar essa VIEW seria assim:

```
CREATE VIEW Marketing AS
SELECT Nome, Cargo, Departamento
FROM Funcionarios
WHERE Departamento = 'Marketing';
```

Depois de criar a VIEW, podemos consultá-la como faríamos com qualquer outra tabela:

```
SELECT * FROM Marketing;
```

Este comando retornará o Nome, Cargo e Departamento de todos os funcionários que trabalham no departamento de "Marketing".

Atualizando uma VIEW

Para atualizar ou modificar uma VIEW, usamos a instrução CREATE OR REPLACE VIEW. Por exemplo, se quisermos adicionar a coluna Salario à nossa VIEW de "Marketing", poderíamos fazer o seguinte:

```
CREATE OR REPLACE VIEW Marketing AS
SELECT Nome, Cargo, Departamento, Salario
FROM Funcionarios
WHERE Departamento = 'Marketing';
```

Excluindo uma VIEW

Para excluir uma VIEW, usamos a instrução DROP VIEW. Por exemplo, para excluir a VIEW de "Marketing", poderíamos fazer o seguinte:

```
DROP VIEW Marketing;
```

Criando Índices

Índices em SQL são estruturas de dados que melhoram a velocidade das operações de recuperação de dados em um banco de dados. Eles são usados para melhorar o desempenho das consultas e são semelhantes aos índices em livros: permitem ao banco de dados encontrar dados sem ter que examinar todas as linhas de uma tabela, da mesma forma que um índice de livro permite que você encontre informações sem ter que ler o livro inteiro.

Índices são criados em colunas específicas de uma tabela. Essas colunas geralmente contêm valores únicos ou uma lista de valores que aparecem com frequência.

Aqui está um exemplo de como criar um índice em SQL:

```
CREATE INDEX idx_nome
ON Clientes (Nome);
```

Neste exemplo, um índice chamado idx_nome é criado na coluna Nome da tabela Clientes. Agora, quando você realiza uma consulta que busca clientes pelo nome, o banco de dados pode usar o índice para encontrar os dados mais rapidamente.

Veja que, embora os índices acelerem as operações de leitura, eles podem desacelerar as operações de gravação (como INSERT, UPDATE e DELETE), pois o banco de dados precisa manter o índice atualizado. Assim, o uso de índices é uma questão de equilíbrio: eles devem ser usados quando o benefício na velocidade de leitura supera o custo adicional na velocidade de gravação.

Além disso, os índices ocupam espaço de armazenamento. É recomendável criar índices apenas para colunas que serão frequentemente usadas em consultas e evitar a criação de índices desnecessários.

Tipos de Dados

Em SQL, tipos de dados definem o que pode ser armazenado em uma coluna quando criamos uma tabela. Cada coluna em uma tabela de banco de dados SQL tem um tipo de dados associado que restringe ou permite específicos tipos de dados que podem ser inseridos na coluna.

Na maioria dos Sistemas de Gerenciamento de Banco de Dados (SGBDs) temos os seguintes tipos de dados:

- **Tipos de Texto:** São usados para armazenar caracteres alfanuméricos. Alguns exemplos incluem:
    - CHAR: Usado para caracteres de comprimento fixo.

  - VARCHAR: Usado para caracteres de comprimento variável.
  - TEXT: Usado para strings de texto de grande tamanho.
- **Tipos Numéricos:** São usados para armazenar dados numéricos. Alguns exemplos incluem:
  - INT: Usado para números inteiros.
  - FLOAT ou REAL: Usado para números de ponto flutuante.
  - DECIMAL ou NUMERIC: Usado para números de precisão fixa.
- **Tipos de Tempo:** São usados para armazenar valores de data e hora. Alguns exemplos incluem:
  - DATE: Usado para armazenar datas.
  - TIME: Usado para armazenar horários.
  - DATETIME: Usado para armazenar tanto data quanto hora.
- **Tipos Binários:** São usados para armazenar dados binários, como imagens, arquivos de áudio ou vídeo. Alguns exemplos incluem:
  - BINARY: Usado para dados binários de tamanho fixo.
  - VARBINARY: Usado para dados binários de tamanho variável.
  - BLOB: Usado para armazenar dados binários de grande tamanho.
- **Outros tipos:** Existem também outros tipos de dados em SQL que não se enquadram nas categorias acima. Alguns exemplos incluem:
  - BOOLEAN: Usado para armazenar valores verdadeiro/falso.
  - ENUM: Usado para armazenar um valor a partir de um conjunto predefinido de valores.

Os tipos de dados específicos disponíveis podem variar entre diferentes SGBDs, então é sempre bom consultar a documentação do SGBD específico que você está usando para obter a lista completa de tipos de dados suportados.

Tipos de Texto

Os tipos de dados de texto em SQL são usados para **armazenar valores alfanuméricos**. Eles permitem que você insira qualquer combinação de letras, números e símbolos especiais. Os tipos de texto mais comuns em SQL são CHAR, VARCHAR e TEXT.

CHAR:

O tipo de dados CHAR é usado para armazenar caracteres alfanuméricos em uma string de **comprimento fixo**. O tamanho da string é definido quando a tabela é criada.

Por exemplo, se você declarar uma coluna como CHAR(10), poderá armazenar exatamente 10 caracteres nela, não mais, não menos. Se a string que você está armazenando tiver menos de 10 caracteres, o espaço extra será preenchido com espaços em branco. Se a string for mais longa, ela será cortada para caber.

Exemplo:

```
CREATE TABLE Pessoa (
    Nome CHAR(50),
    Sobrenome CHAR(50)
);
```

VARCHAR:

O tipo de dados VARCHAR é semelhante ao CHAR, mas armazena strings de comprimento variável. Ao contrário do CHAR, ele não preenche o espaço extra com espaços em branco. Assim, se você declarar uma coluna como VARCHAR(100), ela pode armazenar qualquer string com até 100 caracteres.

Exemplo:

```
CREATE TABLE Pessoa (
    Nome VARCHAR(100),
    Sobrenome VARCHAR(100)
);
```

TEXT:

O tipo de dados TEXT é usado para armazenar strings de caracteres muito grandes. Ele pode armazenar até 65,535 caracteres, tornando-o útil para armazenar informações como parágrafos de texto ou até mesmo o conteúdo de um arquivo.

Exemplo:

```
CREATE TABLE Artigo (
    Titulo VARCHAR(100),
    Conteudo TEXT
);
```

Tipos Numéricos

Os tipos de dados numéricos em SQL são usados para armazenar dados numéricos, que podem ser usados em cálculos matemáticos. Os principais tipos de dados numéricos são INT, FLOAT (ou REAL) e DECIMAL (ou NUMERIC).

INT:

O tipo de dados INT é usado para armazenar números inteiros, ou seja, números sem uma fração decimal. Os valores podem ser tanto positivos quanto negativos. Por exemplo, você pode usar INT para armazenar a idade de uma pessoa, o número de itens em estoque, ou o número de cliques em um site.

Exemplo:

```
CREATE TABLE Produtos (
    ID INT,
    Nome VARCHAR(100),
    QuantidadeEmEstoque INT
);
```

FLOAT (REAL):

O tipo de dados FLOAT ou REAL é usado para armazenar números de ponto flutuante, ou seja, números que têm uma fração decimal. A diferença entre FLOAT e REAL geralmente reside na precisão e na gama de valores que podem ser armazenados.

Estes tipos são adequados quando precisamos armazenar valores que não são inteiros, como peso, altura, ou a média de avaliações de um produto.

Exemplo:

```
CREATE TABLE Produtos (
    ID INT,
    Nome VARCHAR(100),
    Preco FLOAT
);
```

DECIMAL (NUMERIC):

Os tipos de dados DECIMAL ou NUMERIC são usados para armazenar números de precisão fixa. Isso significa que o número de dígitos à esquerda e à direita do ponto decimal permanece constante.

Estes tipos são frequentemente usados em colunas que armazenam valores monetários, onde a precisão é crucial e não se pode correr o risco de ter erros de arredondamento, que podem ocorrer com FLOAT e REAL.

Exemplo:

```
CREATE TABLE Produtos (
    ID INT,
    Nome VARCHAR(100),
    Preco DECIMAL(5,2)
);
```

Neste exemplo, o preço do produto será armazenado com um total de 5 dígitos, dos quais 2 estão após o ponto decimal.

Tipos de Tempo (Datas e Horas)

Os tipos de dados de tempo em SQL são usados para armazenar valores de data e hora. Eles permitem que você armazene informações de data e hora em suas tabelas, que podem ser usadas para rastrear eventos, registrar marcas de tempo e realizar cálculos de data e hora. Os principais tipos de dados de tempo são DATE, TIME e DATETIME.

DATE:

O tipo de dados DATE é usado para armazenar datas. Ele armazena a data no formato AAAA-MM-DD (ano-mês-dia). Por exemplo, você pode usar DATE para armazenar a data de nascimento de uma pessoa ou a data de uma transação.

Exemplo:

```
CREATE TABLE Clientes (
    ID INT,
    Nome VARCHAR(100),
    DataNascimento DATE
);
```

TIME:

O tipo de dados TIME é usado para armazenar horários. Ele armazena o tempo no formato HH:MM:SS (horas:minutos:segundos).

Por exemplo, você pode usar TIME para armazenar o horário de uma reunião ou o tempo que um evento ocorreu.

Exemplo:

```sql
CREATE TABLE Reunioes (
    ID INT,
    Assunto VARCHAR(100),
    HoraInicio TIME,
    HoraFim TIME
);
```

DATETIME:

O tipo de dados DATETIME é usado para armazenar tanto a data quanto a hora. Ele combina DATE e TIME em um único tipo de dados, permitindo que você registre a data e a hora de um evento específico.

Por exemplo, você pode usar DATETIME para registrar a data e hora exatas em que um registro foi criado ou modificado.

Exemplo:

```sql
CREATE TABLE Pedidos (
    ID INT,
    ClienteID INT,
    DataHoraPedido DATETIME
);
```

Tipos Binários

Os tipos de dados binários em SQL são usados para armazenar dados binários, como imagens, arquivos de áudio ou vídeo, ou qualquer outro tipo de dados que não se enquadram diretamente em categorias alfanuméricas ou numéricas. Os principais tipos de dados binários são BINARY, VARBINARY e BLOB.

BINARY:

O tipo de dados BINARY é usado para armazenar dados binários de tamanho fixo. Quando você define uma coluna como BINARY, você especifica um comprimento para os dados. Por exemplo, BINARY(10) armazena exatamente 10 bytes de dados binários.

Exemplo:

```
CREATE TABLE Arquivos (
    ID INT,
    NomeArquivo VARCHAR(100),
    Dados BINARY(5000)
);
```

VARBINARY:

O tipo de dados VARBINARY é semelhante ao BINARY, mas é usado para armazenar dados binários de tamanho variável. Quando você define uma coluna como VARBINARY, você especifica o comprimento máximo dos dados. Por exemplo, VARBINARY(1000) pode armazenar até 1000 bytes de dados binários.

Exemplo:

```
CREATE TABLE Arquivos (
    ID INT,
    NomeArquivo VARCHAR(100),
    Dados VARBINARY(5000)
);
```

BLOB:

O tipo de dados BLOB (Binary Large Object) é usado para armazenar grandes quantidades de dados binários, como imagens ou arquivos de vídeo. BLOB pode armazenar até 65,535 bytes de dados, mas alguns Sistemas de Gerenciamento de Banco de Dados (SGBDs) permitem BLOBs de tamanho muito maior.

Exemplo:

```
CREATE TABLE Arquivos (
    ID INT,
    NomeArquivo VARCHAR(100),
    Dados BLOB
);
```

Dados binários são úteis quando você precisa armazenar dados que não se encaixam facilmente em outros tipos de dados, como imagens ou arquivos de áudio. No entanto, eles

também podem ser mais difíceis de trabalhar do que tipos de dados mais simples, e nem todos os SGBDs tratam dados binários da mesma maneira

Outros Tipos de Dados

Além dos tipos de dados fundamentais em SQL, há uma variedade de outros tipos que podem ser usados para atender a requisitos mais específicos ou complexos. Vamos abordar alguns deles.

BOOLEAN:

O tipo de dados BOOLEAN em SQL é usado para armazenar valores verdadeiro ou falso. Isso pode ser útil para indicar a presença ou ausência de uma condição particular.

Exemplo:

```
CREATE TABLE Usuarios (
    ID INT,
    Nome VARCHAR(100),
    Ativo BOOLEAN
);
```

XML:

O tipo de dados XML é usado para armazenar dados no formato XML. Isso é útil quando você está trabalhando com dados que são mais facilmente representados ou manipulados como documentos XML.

Exemplo:

```
CREATE TABLE Documentos (
    ID INT,
    Dados XML
);
```

UDT (User-Defined Types):

UDTs permitem que você defina seus próprios tipos de dados. Isso pode ser útil quando os tipos de dados padrão não atendem às suas necessidades. Por exemplo, você pode criar um UDT para representar um ponto em um sistema de coordenadas.

Exemplo:

```
CREATE TYPE Point AS (
    X INT,
    Y INT
);
```

ROW:

O tipo de dados ROW permite que você agrupe uma série de campos em uma única estrutura, similar a uma linha em uma tabela.

Exemplo:

```
CREATE TYPE Pessoa AS ROW (
    Nome VARCHAR(100),
    Sobrenome VARCHAR(100),
    Idade INT
);
```

ARRAY:

O tipo de dados ARRAY permite que você armazene uma lista de valores do mesmo tipo. Por exemplo, você pode usar um ARRAY para armazenar uma lista de números de telefone para uma pessoa.

Exemplo:

```
CREATE TABLE Pessoas (
    ID INT,
    Nome VARCHAR(100),
    NumerosTelefone ARRAY<INT>
);
```

MULTISET:

O tipo de dados MULTISET é semelhante a um ARRAY, mas permite duplicatas e não preserva a ordem dos elementos.

Exemplo:

```
CREATE TABLE Pedidos (
    ID INT,
    ItensPedido MULTISET<VARCHAR(100)>
);
```

A tabela a seguir resume os principais tipos de dados em SQL:

Categoria	Subtipo	Descrição
Numérico	INT	Usado para armazenar números inteiros.
	FLOAT/REAL	Usado para armazenar números de ponto flutuante.
	DECIMAL/NUMERIC	Usado para armazenar números de precisão fixa, útil para valores monetários onde a precisão é crítica.
Texto	CHAR	Usado para armazenar caracteres alfanuméricos de tamanho fixo.
	VARCHAR	Usado para armazenar caracteres alfanuméricos de tamanho variável.
	TEXT	Usado para armazenar grandes quantidades de texto.
Binário	BINARY	Usado para armazenar dados binários de tamanho fixo.
	VARBINARY	Usado para armazenar dados binários de tamanho variável.
	BLOB	Usado para armazenar grandes quantidades de dados binários.
Tempo	DATE	Usado para armazenar datas.
	TIME	Usado para armazenar horários.
	DATETIME	Usado para armazenar a data e a hora.
Outros	BOOLEAN	Usado para armazenar valores verdadeiro ou falso.
	XML	Usado para armazenar dados no formato XML.
	UDT (User-Defined Types)	Usado para definir seus próprios tipos de dados.
	ROW	Usado para agrupar uma série de campos em uma única estrutura.
	ARRAY	Usado para armazenar uma lista de valores do mesmo tipo.
	MULTISET	Semelhante a um ARRAY, mas permite duplicatas e não preserva a ordem dos elementos.

Restrições de Integridade

Restrições de integridade em SQL são um conjunto de regras que impõem condições específicas nos dados que podem ser inseridos ou atualizados em um banco de dados. Elas garantem a precisão e confiabilidade dos dados em um banco de dados. Além disso, as restrições de integridade são usadas para garantir que os dados se enquadram em um determinado conjunto de regras predefinidas para manter a consistência e evitar a inserção de dados inválidos ou inconsistentes.

Vejamos os diferentes tipos de restrições de integridade e como elas são implementadas em SQL.

Restrição de Chave Primária (PRIMARY KEY)

Uma restrição de chave primária é uma regra que é aplicada a uma coluna (ou um conjunto de colunas) em uma tabela de banco de dados. Ela serve para garantir que cada valor na coluna seja único e não nulo. Em outras palavras, não podem existir duas linhas na tabela com o mesmo valor de chave primária, e nenhuma linha pode ter um valor de chave primária nulo.

A chave primária é fundamental para a identificação única de registros em uma tabela. Cada tabela deve ter exatamente uma chave primária.

Veja um exemplo de como você pode criar uma tabela com uma restrição de chave primária:

```
CREATE TABLE Estudantes (
    ID INT PRIMARY KEY,
    Nome VARCHAR(100),
    Idade INT
);
```

Neste exemplo, a coluna "ID" é a chave primária. Isso significa que cada estudante deve ter um ID único, e nenhum estudante pode ter um ID nulo.

Se você quiser criar uma chave primária em várias colunas, pode fazer isso da seguinte maneira:

```
CREATE TABLE Matriculas (
    ID_Estudante INT,
    ID_Curso INT,
    Data_Matricula DATE,
    PRIMARY KEY (ID_Estudante, ID_Curso)
);
```

Neste exemplo, a chave primária é uma combinação das colunas "ID_Estudante" e "ID_Curso". Isso significa que, enquanto um estudante pode estar matriculado em vários cursos, e um curso pode ter vários estudantes, a mesma combinação de estudante e curso não pode aparecer na tabela mais de uma vez.

Restrição de Chave Estrangeira (FOREIGN KEY)

A restrição de chave estrangeira é uma regra aplicada a uma coluna (ou conjunto de colunas) em uma tabela de banco de dados. Essa restrição é usada para garantir a integridade referencial entre duas tabelas. Basicamente, a restrição de chave estrangeira garante que o valor em uma coluna de chave estrangeira corresponda a um valor existente na coluna de chave primária de outra tabela.

A chave estrangeira é uma forma de criar uma ligação entre os dados de duas tabelas, estabelecendo uma relação entre elas.

Exemplo de como criar uma tabela com uma restrição de chave estrangeira:

```
CREATE TABLE Estudantes (
    ID INT PRIMARY KEY,
    Nome VARCHAR(100),
    Idade INT
);

CREATE TABLE Matriculas (
    ID INT PRIMARY KEY,
    ID_Estudante INT,
    ID_Curso INT,
    Data_Matricula DATE,
    FOREIGN KEY (ID_Estudante) REFERENCES Estudantes(ID)
);
```

Neste exemplo, a tabela "Matriculas" tem uma coluna "ID_Estudante", que é uma chave estrangeira para a coluna "ID" na tabela "Estudantes". Isso significa que cada valor na coluna "ID_Estudante" deve corresponder a um valor existente na coluna "ID" da tabela "Estudantes". Se tentarmos inserir uma linha na tabela "Matriculas" com um valor de "ID_Estudante" que não existe na tabela "Estudantes", o banco de dados retornará um erro.

Isso garante a integridade dos dados, evitando que você matricule um estudante que não existe na tabela "Estudantes". Além disso, algumas configurações de restrição de chave estrangeira podem prevenir ou limitar a exclusão ou atualização de um registro na tabela primária que está sendo referenciado por registros em outras tabelas.

Restrição de Unicidade (UNIQUE)

A restrição de unicidade (UNIQUE) em SQL é usada para garantir que os dados armazenados em uma coluna, ou em um conjunto de colunas, sejam únicos entre todos as linhas ou registros da tabela. Ou seja, a restrição UNIQUE impede a entrada de valores duplicados nesses campos específicos da tabela.

Esse tipo de restrição é importante para manter a integridade dos dados e pode ser aplicada tanto no momento da criação da tabela quanto posteriormente, por meio de uma alteração de estrutura da tabela.

Aqui está um exemplo de como você pode criar uma tabela com uma restrição de unicidade:

```
CREATE TABLE Estudantes (
    ID INT PRIMARY KEY,
    Nome VARCHAR(100),
    Email VARCHAR(100) UNIQUE,
    Idade INT
);
```

Neste exemplo, a coluna "Email" tem uma restrição UNIQUE. Isso significa que cada estudante deve ter um endereço de email único; não é possível inserir dois estudantes com o mesmo endereço de email.

Se você quiser criar uma restrição de unicidade em várias colunas, você pode fazer isso da seguinte maneira:

```
CREATE TABLE Matriculas (
    ID INT PRIMARY KEY,
    ID_Estudante INT,
    ID_Curso INT,
    Data_Matricula DATE,
    UNIQUE (ID_Estudante, ID_Curso)
);
```

Neste exemplo, a restrição UNIQUE é uma combinação das colunas "ID_Estudante" e "ID_Curso". Isso significa que, embora um estudante possa estar matriculado em vários cursos, e um curso pode ter vários estudantes, a mesma combinação de estudante e curso não pode aparecer na tabela mais de uma vez.

Restrição de não nulidade (NOT NULL)

A restrição NOT NULL em SQL é usada para garantir que uma coluna não possa ter um valor nulo. Em outras palavras, essa restrição exige que um valor seja inserido em uma coluna sempre que uma nova linha (registro) for criada.

Esse tipo de restrição é importante quando você quer garantir que uma coluna específica sempre contenha informações. Se você tentar inserir um novo registro sem fornecer um valor para uma coluna com restrição NOT NULL, o banco de dados retornará um erro e a operação de inserção falhará.

Aqui está um exemplo de como você pode criar uma tabela com uma restrição NOT NULL:

```
CREATE TABLE Estudantes (
    ID INT PRIMARY KEY,
    Nome VARCHAR(100) NOT NULL,
    Email VARCHAR(100),
    Idade INT
);
```

Neste exemplo, a coluna "Nome" tem uma restrição NOT NULL. Isso significa que cada estudante deve ter um nome e não é possível inserir um estudante sem fornecer um valor para o nome.

Nota: Enquanto as chaves primárias são implicitamente NOT NULL (ou seja, não podem ser nulas), nem todas as colunas NOT NULL são chaves primárias. Uma coluna NOT NULL pode conter valores duplicados, desde que cada entrada na coluna contenha algum valor não nulo.

Linguagem de Manipulação de Dados (DML)

A Linguagem de Manipulação de Dados (DML, Data Manipulation Language) é um subconjunto da linguagem SQL (Structured Query Language) usada para manipular os dados armazenados em um banco de dados. O DML inclui comandos para **inserir, selecionar, atualizar e excluir dados**.

Aqui está uma visão geral dos principais comandos DML:

INSERT: este comando é usado para inserir dados em uma tabela. Você pode especificar valores para todas as colunas, ou apenas para colunas específicas.

SELECT: Este comando é usado para selecionar dados de uma ou mais tabelas. Você pode selecionar todas as colunas ou apenas colunas específicas, e pode usar critérios para limitar os registros retornados.

UPDATE: Este comando é usado para atualizar dados em uma tabela. Você pode atualizar todas as linhas ou apenas linhas que atendem a critérios específicos.

DELETE: Este comando é usado para excluir dados de uma tabela. Você pode excluir todas as linhas ou apenas linhas que atendem a critérios específicos.

Vamos estudar cada um em maior profundidade.

SELECT

O comando SELECT é um dos comandos mais fundamentais e comuns na linguagem SQL. Ele é usado para **consultar e extrair dados de uma ou mais tabelas** de um banco de dados. Os dados retornados são armazenados em uma tabela de resultados, chamada de conjunto de resultados.

Aqui está a sintaxe básica do comando SELECT com as cláusulas obrigatórias (veremos cláusulas opcionais mais à frente):

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

Nesta sintaxe, você substitui "column1", "column2", etc. pelos nomes das colunas que deseja selecionar. E substitui "table_name" pelo nome da tabela de onde deseja extrair os dados.

Por exemplo, se tivermos uma tabela chamada "Estudantes" com as colunas "ID", "Nome", "Idade" e "Curso", poderíamos usar o comando SELECT da seguinte maneira:

```
SELECT Nome, Idade
FROM Estudantes;
```

Este comando selecionará todas as informações nas colunas "Nome" e "Idade" de todos os registros na tabela "Estudantes".

Se quisermos selecionar todas as colunas, podemos usar o caractere asterisco (*) em vez de listar todas as colunas:

```
SELECT *
FROM Estudantes;
```

Além disso, você pode adicionar uma cláusula opcional WHERE ao comando SELECT para filtrar os resultados. Por exemplo:

```
SELECT Nome, Idade
FROM Estudantes
WHERE Idade >= 18;
```

Este comando selecionará o "Nome" e a "Idade" de todos os estudantes que têm 18 anos ou mais. A cláusula WHERE é muito flexível e pode incluir vários critérios, que podem ser combinados com operadores lógicos como AND e OR (veremos mais à frente).

Cláusulas opcionais

Além do WHERE, é possível utilizar as cláusulas opcionais GROUP BY, HAVING e ORDER BY com o comando SELECT para agrupar, filtrar e ordenar os resultados da consulta, respectivamente.

GROUP BY e HAVING:

A cláusula GROUP BY é usada para agrupar linhas que têm os mesmos valores em colunas específicas em grupos. Geralmente é usada com funções agregadas (COUNT, MAX, MIN, SUM, AVG) para agrupar os resultados por um ou mais colunas.

Vamos supor que temos uma tabela chamada "Vendas" com as seguintes colunas:

- ID (identificador único da venda)
- Data (data da venda)
- Vendedor (nome do vendedor)
- Valor (valor da venda)

Se quisermos saber o total de vendas realizadas por cada vendedor, poderíamos usar o comando GROUP BY em conjunto com a função SUM para somar o valor das vendas para cada vendedor. Veja como seria o comando:

```
SELECT Vendedor, SUM(Valor) AS TotalVendas
FROM Vendas
GROUP BY Vendedor;
```

Este comando retornará uma lista de vendedores e o valor total das vendas realizadas por cada um.

Se quisermos ver apenas os vendedores que venderam mais de R$5000, podemos adicionar a cláusula HAVING para filtrar os resultados:

```
SELECT Vendedor, SUM(Valor) AS TotalVendas
FROM Vendas
GROUP BY Vendedor
HAVING SUM(Valor) > 5000;
```

Este comando retorna apenas os vendedores que têm um total de vendas superior a R$5000.

Note que a cláusula HAVING é aplicada após o agrupamento ter sido feito, e é capaz de filtrar os grupos de linhas de acordo com critérios que se aplicam ao grupo como um todo, e não às linhas individuais. Isso a distingue da cláusula WHERE, que filtra as linhas antes de elas serem agrupadas.

ORDER BY:

Vamos considerar a mesma tabela "Vendas" do exemplo anterior. Se quisermos exibir todas as vendas realizadas, mas ordenadas pela data da venda, poderíamos usar a cláusula ORDER BY. Veja como seria o comando:

```
SELECT *
FROM Vendas
ORDER BY Data;
```

Este comando retorna todas as vendas, ordenadas pela coluna "Data" em ordem crescente. Em outras palavras, as vendas mais antigas aparecerão primeiro.

Se quisermos ver as vendas mais recentes primeiro, poderíamos alterar a ordenação para decrescente, usando a palavra-chave DESC:

```
SELECT *
FROM Vendas
ORDER BY Data DESC;
```

Este comando retorna todas as vendas, ordenadas pela coluna "Data" em ordem decrescente, ou seja, as vendas mais recentes aparecerão primeiro.

A cláusula ORDER BY também pode ser usada para ordenar por mais de uma coluna. Por exemplo, se quisermos ordenar as vendas primeiro por "Vendedor" e depois por "Data", poderíamos usar o seguinte comando:

```
SELECT *
FROM Vendas
ORDER BY Vendedor, Data DESC;
```

Este comando retorna todas as vendas, ordenadas primeiro pelo nome do vendedor (em ordem alfabética crescente) e, em seguida, pela data da venda (em ordem decrescente para cada vendedor). Esta é uma maneira útil de ordenar os resultados quando há vários critérios de ordenação.

Alias e Distinct

Um alias em SQL é um **nome temporário dado a uma tabela ou coluna** para a duração de uma consulta SQL. Os aliases são úteis para simplificar as consultas e torná-las mais legíveis, especialmente quando as consultas envolvem várias tabelas ou subconsultas complexas.

A sintaxe para criar um alias para uma coluna ou tabela é simplesmente colocar o nome do alias depois do nome da coluna ou tabela, geralmente após a palavra-chave AS. No entanto, o AS é opcional e pode ser omitido.

Exemplo:

```
SELECT v.Nome AS Vendedor, v.Data AS DataVenda
FROM Vendas AS v;
```

Neste exemplo, 'v' é um alias para a tabela 'Vendas' e 'Vendedor' e 'DataVenda' são aliases para as colunas 'Nome' e 'Data' respectivamente.

O comando DISTINCT em SQL é usado para **retornar apenas valores distintos** (únicos) em uma consulta SELECT. Em outras palavras, ele elimina as duplicatas dos resultados.

Aqui está a sintaxe básica do DISTINCT:

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

Por exemplo, se quisermos encontrar todos os vendedores únicos na tabela 'Vendas', poderíamos usar o seguinte comando:

```
SELECT DISTINCT Vendedor
FROM Vendas;
```

Este comando retorna uma lista de nomes de vendedores, sem duplicatas. Isso é útil quando queremos saber quais são os valores únicos em uma coluna, por exemplo, para saber quais categorias de produtos temos, quais regiões estão representadas em nossos dados, etc.

Operadores

Operadores em SQL são símbolos especiais ou palavras-chave que são usados para executar operações específicas entre variáveis e valores. Essas operações podem ser de natureza matemática, comparativa ou lógica. Eles são usados principalmente nas cláusulas WHERE e HAVING de consultas SQL, onde ajudam a definir condições específicas para filtrar os dados.

Existem várias categorias de operadores em SQL, incluindo:

Operadores Aritméticos: São usados para realizar operações matemáticas, como adição (+), subtração (-), multiplicação (*) e divisão (/).

Operadores de Comparação: São usados para comparar um valor com outro. Os operadores de comparação em SQL incluem igual a (=), diferente de (<> ou !=), maior que (>), menor que (<), maior ou igual a (>=), menor ou igual a (<=).

Operadores Lógicos: São usados para combinar ou modificar condições. Os operadores lógicos em SQL incluem AND, OR e NOT.

Operadores de Concatenação: São usados para combinar duas ou mais strings em uma.

Operadores Especiais: Existem alguns operadores adicionais em SQL que não se enquadram nas categorias acima, como BETWEEN, LIKE, IN, IS NULL, e outros. Esses operadores têm usos especializados para situações específicas.

Observação: para verificar se um valor em uma coluna é null, não utilize "Coluna = NULL", mas sim "Coluna is NULL".

O operador LIKE em SQL é usado em uma cláusula WHERE para procurar um padrão específico em uma coluna. Há dois curingas usados em conjunto com o operador LIKE:

% - O sinal de percentagem representa zero, um ou vários caracteres.

_ - O sublinhado representa um único caractere.

Exemplos de uso:

1. Encontrar todas as pessoas cujos nomes começam com "A":

```
SELECT * FROM Persons
WHERE FirstName LIKE 'A%';
```

Neste exemplo, o banco de dados retornará todas as pessoas cujos nomes começam com "A". O "%" após o "A" serve como um curinga que corresponde a qualquer sequência de caracteres que segue o "A".

2. Encontrar todas as pessoas cujos nomes terminam com "son":

```
SELECT * FROM Persons
WHERE LastName LIKE '%son';
```

Neste exemplo, o banco de dados retornará todas as pessoas cujos sobrenomes terminam com "son". O "%" antes do "son" serve como um curinga que corresponde a qualquer sequência de caracteres que precede o "son".

3. Encontrar todas as pessoas cujos nomes têm "an" no terceiro e quarto lugares:

```
SELECT * FROM Persons
WHERE FirstName LIKE '__an%';
```

Neste exemplo, o banco de dados retornará todas as pessoas cujos nomes têm "an" no terceiro e quarto lugares. Os sublinhados "__" representam exatamente dois caracteres, e o "%" corresponde a qualquer sequência de caracteres que segue o "an". Portanto, nomes como "Alan", "Diana" e "Ryan" seriam retornados por essa consulta.

A tabela a seguir traz mais exemplos de como aplicar esses operadores:

Categoria	Operador	Exemplo
Aritméticos	+	SELECT Salario + Bonus AS RendaTotal FROM Funcionarios;
	-	SELECT Salario - Descontos AS RendaLiquida FROM Funcionarios;
	*	SELECT Preco * Quantidade AS Total FROM Produtos;
	/	SELECT Salario / MesesTrabalhados AS MediaSalario FROM Funcionarios;
Comparação	=	SELECT * FROM Clientes WHERE Idade = 30;
	<> or !=	SELECT * FROM Clientes WHERE Idade <> 30;
	>	SELECT * FROM Clientes WHERE Idade > 30;
	<	SELECT * FROM Clientes WHERE Idade < 30;
	>=	SELECT * FROM Clientes WHERE Idade >= 30;
	<=	SELECT * FROM Clientes WHERE Idade <= 30;
Lógicos	AND	SELECT * FROM Clientes WHERE Idade >= 30 AND Regiao = 'Norte';
	OR	SELECT * FROM Clientes WHERE Idade >= 30 OR Regiao = 'Norte';
	NOT	SELECT * FROM Clientes WHERE NOT (Regiao = 'Norte');
Especiais	BETWEEN	SELECT * FROM Clientes WHERE Idade BETWEEN 30 AND 40;
	LIKE	SELECT * FROM Clientes WHERE Nome LIKE 'Jo%';

	IN	SELECT * FROM Clientes WHERE Regiao IN ('Norte', 'Sul');
	IS NULL	SELECT * FROM Clientes WHERE Email IS NULL;

Consultas Aninhadas

Consultas aninhadas, também conhecidas como subconsultas, são consultas SQL que estão embutidas dentro de outra consulta. Uma subconsulta é uma consulta que é executada dentro de outra consulta para retornar um resultado que é usado pela consulta externa.

Vamos agora ver exemplos de consultas aninhadas utilizando IN, EXISTS, ALL, SOME e ANY.

IN:

O operador IN permite especificar múltiplos valores em uma cláusula WHERE.

Exemplo:

```
SELECT *
FROM Clientes
WHERE ID_Cliente IN (SELECT ID_Cliente FROM Pedidos WHERE Valor > 500);
```

Este comando seleciona todos os clientes que fizeram pedidos com valor superior a 500.

EXISTS:

O operador EXISTS é usado para testar a existência de qualquer registro em uma subconsulta.

Exemplo:

```
SELECT *
FROM Produtos p
WHERE EXISTS
(SELECT * FROM Pedidos po WHERE po.ID_Produto = p.ID_Produto AND Quantidade > 10);
```

Este comando seleciona todos os produtos que têm uma quantidade superior a 10 em algum pedido.

ALL:

O operador ALL é usado para comparar um valor com todos os valores retornados por uma subconsulta.

Exemplo:

```
SELECT *
FROM Pedidos
WHERE Valor > ALL (SELECT Valor FROM Pedidos WHERE ID_Cliente = 5);
```

Este comando seleciona todos os pedidos cujo valor é superior a todos os pedidos feitos pelo cliente de ID 5.

SOME ou ANY:

Os operadores SOME e ANY são usados para comparar um valor com qualquer valor retornado por uma subconsulta. SOME e ANY têm o mesmo comportamento.

Exemplo:

```
SELECT *
FROM Pedidos
WHERE Valor > ANY (SELECT Valor FROM Pedidos WHERE ID_Cliente = 5);
```

Operações de Conjuntos

As operações de conjunto em SQL são operações que permitem a combinação de linhas de duas ou mais tabelas com base em regras semelhantes às da teoria dos conjuntos na matemática. As três operações básicas de conjunto em SQL são UNION, INTERSECT e EXCEPT.

UNION:

A operação UNION combina os resultados de duas ou mais consultas SELECT em um único conjunto de resultados. Os duplicados são removidos na operação UNION. Se você quiser manter os duplicados, pode usar UNION ALL.

```
SELECT Nome FROM Funcionarios
UNION
SELECT Nome FROM Gerentes;
```

Este comando retornará os nomes dos funcionários e dos gerentes. Se um nome aparecer em ambas as tabelas, ele aparecerá apenas uma vez no conjunto de resultados.

INTERSECT:

A operação INTERSECT retorna os registros comuns a ambos os SELECTs.

Exemplo:

```
SELECT Nome FROM Funcionarios
INTERSECT
SELECT Nome FROM Gerentes;
```

Este comando retornará apenas os nomes que estão presentes tanto na tabela de funcionários quanto na de gerentes.

EXCEPT:

A operação EXCEPT retorna todos os registros do primeiro SELECT que não existem no segundo SELECT.

Exemplo:

```
SELECT Nome FROM Funcionarios
EXCEPT
SELECT Nome FROM Gerentes;
```

Este comando retornará os nomes que estão presentes na tabela de funcionários, mas que não estão na tabela de gerentes.

Junções (JOINS)

As operações de junção, ou "joins", em SQL são usadas para combinar linhas de duas ou mais tabelas com base em uma coluna relacionada entre elas. Existem vários tipos de junções em SQL: INNER JOIN, LEFT JOIN, RIGHT JOIN e FULL JOIN.

INNER JOIN:

O INNER JOIN seleciona registros que têm valores correspondentes em ambas as tabelas. É o tipo de junção mais comum.

Exemplo:

```
SELECT Pedidos.ID_Pedido, Clientes.Nome
FROM Pedidos
INNER JOIN Clientes ON Pedidos.ID_Cliente = Clientes.ID_Cliente;
```

Este comando retorna a lista de pedidos junto com os nomes dos clientes que fizeram esses pedidos.

LEFT JOIN (ou LEFT OUTER JOIN):

O LEFT JOIN retorna todos os registros da tabela à esquerda (primeira tabela), e os registros correspondentes da tabela à direita (segunda tabela). Se não houver correspondência, o resultado é NULL do lado direito.

Exemplo:

```
SELECT Clientes.Nome, Pedidos.Produto
FROM Clientes
LEFT JOIN Pedidos ON Clientes.ID_Cliente = Pedidos.ID_Cliente;
```

Este comando retorna todos os clientes e os produtos que eles pediram. Se um cliente não fez um pedido, ainda assim, o nome do cliente aparecerá na lista junto com um NULL para o produto.

RIGHT JOIN (ou RIGHT OUTER JOIN):

O RIGHT JOIN retorna todos os registros da tabela à direita (segunda tabela), e os registros correspondentes da tabela à esquerda (primeira tabela). Se não houver correspondência, o resultado é NULL do lado esquerdo.

Exemplo:

```
SELECT Clientes.Nome, Pedidos.Produto
FROM Clientes
RIGHT JOIN Pedidos ON Clientes.ID_Cliente = Pedidos.ID_Cliente;
```

Este comando retorna todos os pedidos e os clientes que os fizeram. Se um pedido foi feito por um cliente não listado na tabela de clientes, ainda assim o pedido será listado com um NULL para o nome do cliente.

FULL JOIN (ou FULL OUTER JOIN):

O FULL JOIN retorna registros quando há uma correspondência em uma das tabelas. Ou seja, se houver uma correspondência na tabela à esquerda, a linha será retornada e, se houver uma correspondência na tabela à direita, essa linha será retornada. Se não houver correspondência, o resultado é NULL em ambos os lados.

Exemplo:

```
SELECT Clientes.Nome, Pedidos.Produto
FROM Clientes
FULL JOIN Pedidos ON Clientes.ID_Cliente = Pedidos.ID_Cliente;
```

Este comando retorna todos os pedidos e os clientes que os fizeram, além de todos os clientes que não fizeram pedidos e todos os pedidos que não têm um cliente correspondente.

Funções Agregadas

As funções agregadas em SQL são usadas para realizar cálculos em um conjunto de valores e retornar um único valor. Essas funções, muitas vezes, são usadas em conjunto com os comandos GROUP BY e HAVING para criar subconjuntos de dados e realizar cálculos sobre esses subconjuntos.

Aqui estão alguns exemplos das funções agregadas mais comumente usadas em SQL:

COUNT:

A função COUNT() retorna o número de linhas que correspondem a um critério específico.

Por exemplo:

```
SELECT COUNT(ID_Cliente)
FROM Pedidos;
```

Este comando retorna o número total de pedidos na tabela Pedidos.

SUM:

A função SUM() retorna a soma total de uma coluna numérica.

Por exemplo:

```
SELECT SUM(Quantidade)
FROM Pedidos;
```

Este comando retorna a soma total de todas as quantidades na tabela Pedidos.

AVG:

A função AVG() retorna a média dos valores em uma coluna numérica.

Exemplo:

```
SELECT AVG(Preco)
FROM Produtos;
```

MAX:

A função MAX() retorna o valor máximo em uma coluna numérica.

Exemplo:

```
SELECT MAX(Preco)
FROM Produtos;
```

Este comando retorna o preço mais alto na tabela Produtos.

MIN:

A função MIN() retorna o valor mínimo em uma coluna numérica.

Exemplo:

```
SELECT MIN(Preco)
FROM Produtos;
```

Este comando retorna o preço mais baixo na tabela Produtos.

INSERT, UPDATE e DELETE

Os comandos INSERT, UPDATE e DELETE são parte essencial da linguagem SQL e estão entre as operações mais comuns realizadas em um banco de dados. Esses comandos fazem parte da sublinguagem DML (Linguagem de Manipulação de Dados) do SQL.

INSERT:

O comando INSERT é utilizado para inserir novos dados em uma tabela. Com ele, você pode adicionar um novo registro contendo valores para algumas ou todas as colunas da tabela.

Sintaxe básica:

```
INSERT INTO nome_tabela (coluna1, coluna2, coluna3, ...)
VALUES (valor1, valor2, valor3, ...);
```

Exemplo:

```
INSERT INTO Clientes (Nome, Endereco, Email)
VALUES ('João Silva', 'Rua das Flores, 100', 'joao.silva@email.com');
```

Neste exemplo, um novo cliente chamado "João Silva" é inserido na tabela Clientes.

UPDATE:

O comando UPDATE é utilizado para modificar os dados já existentes em uma ou mais linhas de uma tabela. Com ele, você pode alterar os valores de uma ou mais colunas de uma ou várias linhas. É muito utilizado em cenários onde se faz necessário alterar informações já salvas, como por exemplo, atualizar o endereço de um cliente.

Sintaxe básica:

```
UPDATE nome_tabela
SET coluna1 = valor1, coluna2 = valor2, ...
WHERE condição;
```

Exemplo:

```
UPDATE Clientes
SET Endereco = 'Rua das Rosas, 200'
WHERE Nome = 'João Silva';
```

Neste exemplo, o endereço do cliente "João Silva" é atualizado na tabela Clientes.

DELETE:

O comando DELETE é utilizado para remover uma ou mais linhas de uma tabela. A operação de DELETE é bastante crítica, pois a remoção de dados é muitas vezes uma operação irreversível. É usado quando se deseja remover registros que não são mais necessários, mas deve ser utilizado com cautela para evitar a perda de dados acidental.

Sintaxe básica:

```
DELETE FROM nome_tabela
WHERE condição;
```

Exemplo:

```
DELETE FROM Clientes
WHERE Nome = 'João Silva';
```

Neste exemplo, o cliente "João Silva" é excluído da tabela Clientes.

Um ponto importante a ser lembrado é que é necessário ter cuidado ao usar o comando DELETE sem uma cláusula WHERE. Isso resultará na exclusão de todos os registros da tabela.

Extensões Procedurais

As extensões procedurais em SQL são recursos que permitem aos desenvolvedores de banco de dados ir além das capacidades básicas de consulta e manipulação de dados do SQL padrão. Elas permitem a criação de rotinas mais complexas e de maior desempenho que são armazenadas diretamente no banco de dados. As três extensões procedurais mais comuns são Stored Procedures, Functions e Triggers.

Stored Procedures:

Stored Procedures são um grupo de comandos SQL que podem ser compilados e armazenados no banco de dados. Uma vez criados, eles podem ser chamados pelo seu nome sempre que necessário. Stored Procedures podem aceitar parâmetros de entrada e retornar múltiplos valores de saída.

Exemplo de Stored Procedure:

```
CREATE PROCEDURE GetTotalOrders @CustomerId INT
AS
BEGIN
    SELECT COUNT(*)
    FROM Orders
    WHERE CustomerId = @CustomerId
END
```

Este Stored Procedure retorna o número total de pedidos de um determinado cliente.

Functions:

As Functions em SQL são semelhantes às Stored Procedures, mas são projetadas para retornar um único valor. Elas também podem aceitar parâmetros.

Exemplo de Function:

```
CREATE FUNCTION GetTotalOrderAmount(@OrderId INT)
RETURNS INT
AS
BEGIN
    DECLARE @Total INT;

    SELECT @Total = SUM(Quantity * Price)
    FROM OrderDetails
    WHERE OrderId = @OrderId;

    RETURN @Total;
END;
```

Esta Function retorna o valor total de um determinado pedido.

Triggers:

Triggers são procedimentos especiais que são executados automaticamente em resposta a determinados eventos no banco de dados. Eles são geralmente usados para manter a integridade dos dados.

Exemplo de Trigger:

```
CREATE TRIGGER UpdateCustomerOrderCount
AFTER INSERT ON Orders
FOR EACH ROW
BEGIN
    UPDATE Customers
    SET OrderCount = OrderCount + 1
    WHERE CustomerId = NEW.CustomerId;
END
```

Este Trigger incrementa a contagem de pedidos de um cliente sempre que um novo pedido é inserido.

Essas extensões procedurais oferecem maior flexibilidade e controle sobre as operações de banco de dados, permitindo a criação de lógica de negócios complexa diretamente dentro do banco de dados. No entanto, elas também devem ser usadas com cautela, pois podem complicar o design do banco de dados e criar problemas de desempenho se não forem projetadas corretamente.

Linguagem de Controle de Dados (DCL) e Segurança de Dados

Segurança e o controle de acesso são aspectos essenciais do gerenciamento de bancos de dados. O Data Control Language (DCL) é uma parte do SQL que lida com direitos, permissões e outros controles do sistema de gerenciamento de banco de dados.

Os comandos DCL são usados pelos administradores do banco de dados para limitar o acesso a objetos no banco de dados, prevenir a modificação não autorizada de dados, proteger os dados confidenciais e manter a integridade geral do banco de dados.

Essa forma de controle de acesso é essencial para manter a segurança do banco de dados, principalmente em ambientes onde muitos usuários têm acesso ao sistema, cada um com

diferentes níveis de privilégios. DCL permite que os administradores do banco de dados tenham um controle refinado sobre quem pode fazer o quê no banco de dados.

Os comandos DCL mais comuns são GRANT, REVOKE e DENY.

GRANT:

Este comando é usado para dar permissões de usuário ou grupos de usuários. As permissões incluem SELECT, INSERT, UPDATE, DELETE e outras. Por exemplo, se quisermos dar ao usuário 'Jose' permissões para ler e atualizar a tabela 'Clientes', o comando seria assim:

```
GRANT SELECT, UPDATE ON Clientes TO Jose;
```

REVOKE:

Este comando é usado para retirar permissões concedidas anteriormente a usuários ou grupos de usuários. Usando o mesmo exemplo, se quisermos retirar as permissões de 'Jose' na tabela 'Clientes', o comando seria:

```
REVOKE SELECT, UPDATE ON Clientes FROM Jose;
```

DENY:

Esse comando é usado para negar permissões específicas a um usuário ou grupo. Este comando é usado principalmente em sistemas de gerenciamento de banco de dados que suportam controle de acesso baseado em função, como o Microsoft SQL Server.

Roles

Roles, ou papéis, são um conceito-chave em muitos sistemas de gerenciamento de banco de dados. No contexto do DCL (Data Control Language), um role é uma forma de gerenciar as permissões que são concedidas aos usuários.

Um role é, essencialmente, um **nome para um conjunto específico de permissões**. Em vez de conceder ou revogar permissões diretamente a um usuário individual, o administrador do banco de dados pode simplesmente criar um role com o conjunto necessário de permissões e, em seguida, atribuir esse role a um ou mais usuários. Isso pode facilitar muito o gerenciamento de permissões, especialmente em grandes organizações onde muitos usuários podem precisar de conjuntos semelhantes de permissões.

Por exemplo, em um banco de dados de uma empresa, pode haver um role chamado 'Gerente', que tem permissões para SELECT, UPDATE e DELETE em várias tabelas. Se um

novo gerente for contratado, o administrador do banco de dados pode simplesmente atribuir o role 'Gerente' a esse usuário, em vez de ter que conceder todas as permissões individuais.

Os roles também podem ser usados para implementar um controle de acesso baseado em função, onde as permissões são concedidas com base na função de trabalho de um usuário, em vez de sua identidade individual.

Aqui está um exemplo de como criar um role e atribuir permissões a ele em SQL:

```
CREATE ROLE Gerente;
GRANT SELECT, UPDATE, DELETE ON Clientes TO Gerente;
```

E aqui está como você atribuiria esse role a um usuário:

```
GRANT Gerente TO Jose;
```

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[1].

Um dos aspectos mais cruciais do estudo de SQL para qualquer aluno é a compreensão profunda das operações de junção, ou "joins", que são fundamentais para o trabalho com bancos de dados relacionais. As junções permitem que dados de diferentes tabelas sejam combinados com base em condições relacionais, criando uma ponte entre conjuntos de dados que, de outra forma, permaneceriam isolados. O entendimento de como diferentes tipos de junções — como INNER JOIN, LEFT JOIN, RIGHT JOIN e FULL JOIN — afetam os resultados de uma consulta é essencial, pois cada tipo de junção serve a propósitos distintos e pode significativamente alterar os dados que são recuperados em uma consulta.

[1] 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.

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. **(FGV / Receita Federal - 2023)** Considere um banco de dados relacional em que as operações de insert e update efetuadas numa certa tabela devem ser monitoradas e anotadas, como subsídio aos procedimentos de auditoria da empresa. Essa tabela é utilizada por uma série de aplicações, em diferentes tipos de transações, e iniciadas por um número considerável de usuários.

   Nesse cenário, assinale o mecanismo mais adequado para a implementação desse monitoramento:

   a) Cursores.

   b) Stored procedures.

   c) Triggers.

   d) Utilitários de exportação de dados.

   e) Views.

   Comentários:

   No cenário descrito, o mecanismo mais adequado para implementar o monitoramento de operações de INSERT e UPDATE na tabela é o uso de Triggers. Elas são procedimentos armazenados que são executados automaticamente em resposta a determinados eventos, como inserção, atualização ou exclusão de dados

em uma tabela. No caso em questão, é possível criar um trigger que seja executado sempre que uma operação de INSERT ou UPDATE for realizada na tabela monitorada. Esse trigger pode inserir uma linha de log em uma outra tabela, que registre a operação realizada e outras informações relevantes para fins de auditoria.

Gabarito: C

2. **(FGV / Receita Federal - 2023)** Os principais Sistemas Gerenciadores de Bancos de Dados oferecem total suporte à linguagem SQL. Um aspecto importante da implementação do SQL é o tratamento para valores nulos quando esses são considerados como unknown values.

Nesse contexto, considere uma tabela T com colunas A e B, que podem conter valores nulos. T possui 100 registros e, em 50% das linhas, há pelo menos uma coluna preenchida com o valor NULL. Considere a consulta a seguir:

```
SELECT * FROM T t1 WHERE t1.A = NULL or t1.B = NULL
```

O número máximo de linhas de resultados que seriam retornadas pela consulta é igual a:

a) 0.

b) 25.

c) 50.

d) 75.

e) 100.

Comentários:

A FGV toda empolgada achando que você vai cair nessa pegadinha sorrateira. Quando queremos comparar um valor com NULL, utilizamos a cláusula “IS NULL” ou “IS NOT NULL”, em vez dos operadores “=” ou “<>”. Isso porque o valor NULL é considerado desconhecido, o que significa que não é igual nem diferente de qualquer outro valor, incluindo ele mesmo. Logo, essa consulta ão irá retornar nenhuma linha de resultado. Para corrigi-la, teríamos que fazer:

SELECT * FROM T t1 WHERE t1.A IS NULL or t1.B IS NULL

Gabarito: A

3. **(FGV / TRT-MA – 2022)** Assinale a afirmativa correta a respeito do esquema relacional apresentado.

```
create table X(A int not null primary key,
        B int)
create table Y(A int not null UNIQUE,
        constraint fk
  foreign key (A) references X(A)
  on delete cascade)
```

a) a tabela X admite linhas duplicadas.

b) a tabela X não pode ter mais linhas que a tabela Y

c) a tabela Y admite linhas duplicadas.

d) a tabela Y não pode ter mais linhas que a tabela X.

e) as tabelas X e Y não podem ter o mesmo número de linhas.

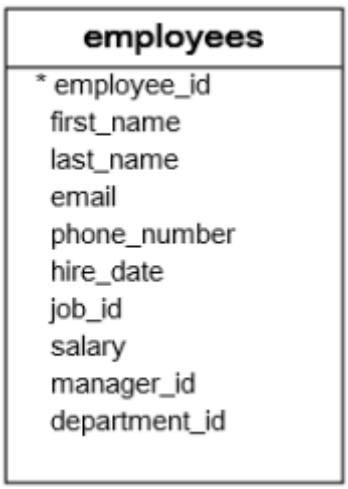
Comentários:

Observem que a Tabela X tem dois atributos: A, que é chave primária; e B, que é um inteiro. Já a Tabela Y tem somente um atributo: A, que é também chave primária (dado que é único e não nulo) e esse atributo é uma chave estrangeira que referencia o atributo A da Tabela X.

Se o atributo A da Tabela Y referencia o atributo A da Tabela X, então a Tabela Y jamais poderá ter mais linhas que a Tabela X. Já a Tabela X pode ter mais (ou iguais) linhas que a Tabela Y sem nenhum problema. E nenhuma das tabelas admite linhas duplicadas porque possuem atributos únicos e não nulos.

Gabarito: D

4. **(FGV / SEFAZ-BA – 2022)** Considere a seguinte tabela em um banco de dados relacional.



Assinale a opção que indica o comando SQL utilizado para localizar todos os nomes completos dos *employees*, cujos primeiros nomes começam com as letras Ma.

a) SELECT

first_name;

last_name;

FROM

employees;

b) SELECT *

FROM

employees

WHERE

first_name = 'Ma';

c) SELECT *

FROM

employees

WHERE

first_name = 'Ma*';

d) SELECT

employee_id,

```
    first_name,
    last_name
FROM
employees
    WHERE
first_name LIKE 'Ma%';
```

e) SELECT

```
employee_id,
    first_name,
    last_name
FROM
employees
WHERE
first_name IN 'Ma_';
```

Comentários:

(a) Errado. Esse comando está retornando o nome completo (primeiro e último nome) de todos os empregados;

(b) Errado. Esse comando está retornando todas as informações (e não apenas o nome completo) de todos os funcionários cujo primeiro nome é "Ma"

c) Errado. Esse comando está retornando todas as informações (e não apenas o nome completo) de todos os funcionários cujo primeiro nome é "Ma";

(d) Correto. Traz o primeiro e último nome (além do ID) de todos os funcionários cujo primeiro nome começa com "Ma". O operador correto é o LIKE e o % indica uma cadeia de caracteres.

(e) Errado. IN é um operador que permite especificar múltiplos valores dentro de uma cláusula. O operador correto seria o LIKE e mesmo assim o underline (_) indica um único caractere e a questão pede qualquer funcionário cujo primeiro nome comece com "Ma" mais qualquer cadeia de caracteres, logo deveria ser LIKE 'Ma%'

Gabarito: D

5. **(FGV / FUNSÚDE-CE – 2021)** Atenção: na próxima questão, considere a definição e as instâncias das tabelas de bancos de dados CLUBE e JOGO exibidas a seguir.

CLUBE

nome
Barcelona
Boca Juniors
The Strongest

JOGO

mandante	visitante	golsM	golsV

Barcelona	Boca Juniors	1	0
Barcelona	The Strongest	NULL	NULL
Boca Juniors	Barcelona	0	0
Boca Juniors	The Strongest	3	0
The Strongest	Barcelona	2	0
The Strongest	Boca Juniors	2	0

Cada clube deve jogar quatro vezes, duas como mandante e duas como visitante. As colunas golsM e golsV registram o número de gols dos times mandantes e visitantes, respectivamente, em cada jogo. Ambas são nulas enquanto o jogo não for realizado.

Analise o comando SQL a seguir, à luz das definições e instâncias das tabelas CLUBE e JOGO, apresentadas anteriormente.

```
select distinct mandante, visitante from JOGO, CLUBE
```

Assinale o número de linhas, sem incluir os títulos, produzidas pela execução desse comando:

a) 4.

b) 6.

c) 10.

d) 24.

e) 48.

A	B	C	D
12	2	3	1
14	3	8	2
18	2	9	3
21	5	4	4

Comentários:

Vamos analisar a consulta apresentada na questão: (1) ela possui um distinct, logo registros repetidos (duplicatas) serão eliminados; (2) ela possui um produto cartesiano, dado que há uma vírgula separando as tabelas JOGO e CLUBE. Dito isso, sabemos que a primeira tabela possui três linhas e a segunda possui seis linhas, logo teríamos 3x6 = 18 linhas. No entanto, há uma pegadinha na questão: ela deseja retornar apenas as colunas mandante e visitante, e essas colunas pertencem somente à tabela JOGO. Como há um distinct na consulta, é irrelevante o produto cartesiano, visto que as repetições serão ignoradas. Logo, ela retornará apenas seis linhas:

mandante	visitante
Barcelona	Boca Juniors
Barcelona	The Strongest
Boca Juniors	Barcelona
Boca Juniors	The Strongest
The Strongest	Barcelona
The Strongest	Boca Juniors

Gabarito: B

6. **(FGV / TCE-AM – 2021)** Considerando-se a instância da tabela T (descrita anteriormente), analise o comando SQL abaixo.

```
delete from T where b + d = c
```

O número de registros da tabela T afetados pela execução desse comando é:

a) zero;

b) um;

c) dois;

d) três;

e) quatro.

Comentários:

Linha 1: b = 2, d = 1, c = 3; 2 + 1 = 3? Sim, portanto deleta a primeira linha;

Linha 2: b = 3, d = 2, c = 8; 3 + 2 = 8? Não, portanto não deleta a segunda linha;

Linha 3: b = 2, d = 3, c = 9; 2 + 3 = 9? Não, portanto não deleta a terceira linha;

Linha 4: b = 5, d = 4, c = 4; 5 + 4 = 4? Não, portanto não deleta a quarta linha.

Gabarito: B

7. **(FGV / IMBEL – 2021)** Considere o comando SQL a seguir, executado num banco de dados relacional com duas tabelas, R1 e R2, contendo 2.000 e 5.000 registros, respectivamente. R1 e R2 possuem chaves primárias definidas.

```
SELECT DISTINCT * FROM A, B
```

Assinale o número de linhas produzidas na execução:

a) 1.

b) 2.000.

c) 5.000.

d) 7.000.

e) 10.000.000

Comentários:

Trata-se de um produto cartesiano, logo teremos 2.000 x 5.000 = 10.000.000 linhas.

Gabarito: E

8. **(FGV / Prefeitura de Niterói-RJ – 2018)** A otimização de consultas em gerenciadores de bancos de dados é fundamental para o desempenho do sistema. Consultas escritas em SQL são particularmente propícias à otimização, porque essa linguagem:

a) não é procedural, permitindo a criação de diferentes planos de execução.

b) tem uma sintaxe simples e é largamente utilizada.

c) suporta todas as operações da Álgebra Relacional.

d) permite o uso de subconsultas, facilitando os processos de busca.

e) permite a criação de camadas de software de persistência.

Comentários:

(a) Correto, ela não é procedural – ela é declarativa. Os diferentes planos de execução permitem chegar ao mesmo resultado com pior ou melhor desempenho; (b) Errado, isso não é relevante para otimização; (c) Errado, isso é um requisito de um SGBD e não se relaciona com a otimização; (d) Errado, isso também não é particularmente relevante para otimização de consultas; (e) Errado, isso é uma característica do SGBD e não tem relação com otimização de consultas.

Gabarito: A

9. **(FGV / MPE-AL – 2018)** Considere o comando SQL a seguir.

```
select * from teste where nota <> null
```

Sabendo-se que a tabela teste tem 100 registros, dos quais 15 estão com a coluna nota ainda não preenchida (null), o comando acima, executado no MS SQL Server ou no Oracle, retorna

a) um erro, porque o literal null não pode ser usado diretamente numa comparação.

b) zero linhas.

c) quinze linhas.

d) oitenta e cinco linhas.

e) cem linhas.

Comentários:

Operadores relacionais não podem ser utilizados para comparar valores nulos, portanto essa consulta não retornará nenhuma linha.

Gabarito: B

10. (FGV / BANESTES – 2018) Considere um banco de dados com duas tabelas, R e S, contendo 4 e 3 registros, respectivamente. Em R, os valores da coluna A são 1, 2, 5 e 7. Em S, os valores da coluna B são 2, 4 e 7. Excetuando-se a linha de títulos, o número de linhas no resultado do comando SQL

```
select * from R full outer join S on A=B
```

É:

a) 3

b) 4

c) 5

d) 7

e) 12

Comentários:

O FULL OUTER JOIN retorna todos os registros mesmo quando não há uma correspondência da tabela esquerda com a direita ou da direita com a esquerda. Portanto, se houver linhas em "Clientes" que não tenham correspondências em "Pedidos", ou se houver linhas em "Pedidos" que não tenham correspondências em "Clientes", essas linhas também serão listadas. Logo, ele retornaria algo como:

(1, null)

(2, 2)

(null, 4)

(5, null)

(7, 7)

Isso ocorre pois, quando nenhuma linha da tabela da esquerda corresponde a uma linha da tabela da direita, retorna-se NULL; quando nenhuma linha da tabela da direita corresponde a uma linha da tabela da esquerda, também se retorna NULL; e quando há uma correspondência, retorna os dois valores.

Gabarito: C

11. (FGV / IBGE – 2017) A linguagem mais comum para elaboração de consultas em bancos de dados é a SQL. Ao elaborar uma consulta nessa linguagem, emprega-se a cláusula WHERE quando se deseja:

a) especificar a tabela onde será realizada a consulta;

b) especificar o diretório onde os dados estão armazenados;

c) especificar o endereço IP onde os dados estão armazenados;

d) especificar condições a que as instâncias selecionadas devem atender;

e) extrair a geometria do objeto selecionado na consulta.

R	
a	b
1	2
2	3
4	5

S	
c	d
3	2
4	2
6	1

Comentários:

A cláusula WHERE é responsável por permitir a filtragem dos registros de uma tabela por meio de uma ou mais condições a que as instâncias selecionadas devem atender.

Gabarito: D

12. (FGV / MPE-BA – 2017) Considerando as tabelas R e S apresentadas anteriormente, o resultado

a	b
1	2
2	3

seria obtido pela execução do comando SQL:

a) select * from R

where not exists (select * FROM S where a=c)

b) select * from R

where not exists (select * FROM S where a=d)

c) select * from S

where not exists (select * FROM R where a=c)

d) select * from S

where not exists (select * FROM R where a=d)

e) select * from R

where exists (select * FROM S where b=d)

Comentários:

(a) Correto. Essa consulta basicamente seleciona todas as linhas de R nas quais a é diferente de c (dado que temos um NOT EXISTS). 1 e 2 da tabela R não existem em S, logo retorna {(1,2), (2,3)}; (b) Errado, isso retornaria apenas {(4,5)}; (c) Errado, isso retornaria {(3,2), (6,1)}; (d) Errado, isso retornaria nulo; (e) Errado, isso retornaria {((1,2)}.

Gabarito: A

13. **(FGV / PGE-RO – 2015)** No SQL, a outorga de privilégios sobre objetos de um banco de dados é efetuada por meio do comando:

a) CREATE;

b) GRANT;

c) LICENSE;

d) PERMIT;

e) REVOKE.

Comentários:

O comando DCL responsável por outorgar privilégios sobre objetos de um banco de dados é o comando GRANT.

Gabarito: B

14. (FGV / DPE-RO – 2015) Observe o comando SQL a seguir.

```
update X set Y = 'Z'
```

Para que esse comando esteja corretamente formulado, quando analisado isoladamente, pressupõe-se que:

a) Y seja uma coluna da tabela X;

b) X seja uma coluna da tabela Y;

c) X e Y sejam tabelas;

d) X seja um banco de dados e Y seja uma tabela;

e) X seja um banco de dados e Y seja uma tabela;

Comentários:

Para esse comando funcionar corretamente, assume-se que X é uma tabela e que Y é uma de suas colunas. Vamos lembrar da sintaxe:

UPDATE NOME_DA_TABELA

SET NOME_DA_COLUNA_1 = VALOR_1, NOME_COLUNA2 = VALOR_2 ...

WHERE LISTA_DE_CONDICOES

Gabarito: A

15. (FGV / TJ-BA – 2015) Considere que as instâncias das tabelas T1, T2 e T3 têm, respectivamente, 1.000, 10.000 e 100.000 registros. O comando SQL:

```
select 1 from t1

union

select 2 from t2

union

select 3 from t3
```

produz um resultado com:

a) 3 linhas;

b) 1.000 linhas;

c) 10.000 linhas;

d) 100.000 linhas;

e) 111.000 linhas.

Comentários:

Pegadinha clássica: UNION basicamente une tabelas que tenham estruturas idênticas. Logo, a tabela resultante teria 1.000 + 10.000 + 100.000 = 111.000 registros. No entanto, observe que há um número após o select, em vez do nome de uma coluna. Quando há uma constante em vez do nome de uma coluna, o comando retorna esse número para cada um dos registros. Logo, ele retorna 1.000 vezes o valor 1; depois retorna 10.000 vezes o valor 2; e depois retorna 100.000 vezes o valor 3.

select 1 from t1 - retorna as 1.000 linhas contendo o número 1.

select 2 from t2 - retorna as 10.000 linhas contendo o número 2.

select 3 from t3 - retorna as 100.000 linhas contendo o número 3.

Professor, isso não daria 111.000 registros do mesmo jeito? Não, porque o UNION elimina duplicadas, então o comando apresentado no enunciado retornaria apenas três linhas: 1, 2, 3. Para retornar as 111.000 linhas, deveria ser utilizado o comando UNION ALL.

Gabarito: A

16. (FGV / TJ-AM – 2013) A SQL é constituída pela Data Control Language (DCL), a Data Definition Language (DDL) e a Data Manipulation Language (DML).

Assinale a alternativa que apresenta os três comandos que são parte integrante da DDL:

a) REVOKE, GRANT e DELETE

b) GRANT, DELETE e UPDATE

c) DELETE, UPDATE e CREATE

d) CREATE, ALTER e DROP

e) ALTER, DROP e REVOKE

Comentários:

(a) REVOKE é DCL; GRANT é DCL; e DELETE é DML; (b) GRANT é DCL, DELETE é DML; e UPDATE é DML; (c) DELETE é DML; UPDATE é DML; e CREATE é DDL; (d) CREATE é DDL; ALTER é DDL; e DROP é DDL; (e) ALTER é DDL; DROP é DDL; e REVOKE é DCL;

Gabarito: D

17. (FGV / MPE-MS – 2013) Observe o comando SQL a seguir:

```
SELECT nome, sobrenome, PIS, anos_de_servico FROM Empregados
```

A cláusula que deve ser adicionada ao comando acima para ordenar os registros por anos de serviço, com os empregados que estão há mais tempo na empresa aparecendo primeiro na listagem, é:

a) ORDER 'anos_de_servico' BY ASC

b) ORDER BY 'anos_de_servico'

c) ORDER BY anos_de_servico DESC

d) SORTED BY anos_de_servico DESC

e) ORDER BY anos_de_servico ASC

Comentários:

Para ordenar os registros por anos de serviço (ORDER BY), com os empregados que estão há mais tempo na empresa aparecendo primeiro na listagem (DESC). Logo, temos:

ORDER BY anos_de_servico DESC.

Gabarito: C

18. (VUNESP - 2022 - UNICAMP - Analista de Desenvolvimento de Sistemas)
Considere a seguinte tabela de um banco de dados relacional: Cliente (ID, Nome,

Agencia, Saldo) A consulta SQL para obter o nome dos clientes que possuam saldo maior do que o saldo do cliente de ID igual a 200 é

A) SELECT T.Nome FROM Cliente S, Cliente T WHERE S.ID = 200 AND T.Saldo > S.Saldo

B) SELECT Cliente.Nome FROM Cliente WHERE Cliente.ID = 200

C) SELECT Cliente.Nome FROM Cliente.ID = 200

D) SELECT Cliente1.Nome FROM Cliente WHERE Cliente1.Saldo > Cliente2.Saldo AND Cliente1.ID > 200

E) SELECT Cliente1.Nome FROM Cliente1, Cliente2 WHERE Cliente2.Saldo > Cliente1.Saldo AND Cliente1.ID > 200

Comentários:

A consulta SQL correta para obter o nome dos clientes que possuem saldo maior do que o saldo do cliente de ID igual a 200 é:

SELECT T.Nome FROM Cliente S, Cliente T WHERE S.ID = 200 AND T.Saldo > S.Saldo

Nessa consulta, a tabela "Cliente" é referenciada duas vezes, sendo uma vez como "S" e outra como "T". A condição "S.ID = 200" seleciona o cliente com ID igual a 200 como ponto de referência para comparação. Em seguida, a condição "T.Saldo > S.Saldo" filtra os clientes que possuem saldo maior que o saldo desse cliente de ID 200. O resultado final é o nome dos clientes que atendem a essa condição.

Portanto, a alternativa A é a correta.

Gabarito: A

19. (VUNESP - 2022 - Prefeitura de Sorocaba - SP - Analista de Sistemas I)

Considere a seguinte tabela de um banco de dados relacional: Medicamento (ID, Nome, Indicação). A consulta SQL para obter o Nome e a Indicação dos medicamentos, ordenada pelo atributo Indicação, e cujo Nome termine com a sequência ‘gem’ é

A) SELECT Nome AND Indicação FROM Medicamento HAVING Nome IN ‘%gem’ ORDER BY Indicação Asc;

B) SELECT Nome, Indicação FROM Medicamento WHERE Nome SIMILAR ‘%gem’ GROUP BY Nome, Indicação;

C) SELECT Nome, Indicação FROM Medicamento WHERE Nome EQUAL ‘%gem’ GROUP BY Indicação;

D) SELECT Nome, Indicação FROM Medicamento WHERE Nome LIKE ‘%gem’ ORDER BY Indicação;

E) SELECT Nome AND Indicação FROM Medicamento HAVING Nome IN ‘%gem’ ORDER BY 2;

Comentários:

A consulta SQL correta para obter o Nome e a Indicação dos medicamentos, ordenada pelo atributo Indicação, e cujo Nome termine com a sequência 'gem' é:

SELECT Nome, Indicação FROM Medicamento WHERE Nome LIKE '%gem' ORDER BY Indicação;

Nessa consulta, utilizamos a cláusula WHERE para filtrar os medicamentos cujo Nome termina com 'gem'. O operador LIKE é utilizado com o curinga '%' antes de 'gem', indicando que o Nome pode ter qualquer sequência de caracteres antes da sequência 'gem'. Em seguida, utilizamos a cláusula ORDER BY para ordenar os resultados pelo atributo Indicação.

Gabarito: D

20. (VUNESP - 2021 - Semae de Piracicaba - SP - Programador Junior) Considere a seguinte tabela de um banco de dados relacional: Funcionário (CPF, Nome, Sobrenome, Função, Salário) O comando SQL para obter Nome e Função dos funcionários de sobrenome Silva, em qualquer parte do sobrenome, ordenados, de forma ascendente, pelo nome, é:

A) SELECT Nome, Função FROM Funcionário WHERE Sobrenome IN “#Silva#” SET BY Nome;

B) SELECT Nome, Função FROM Funcionário WHERE Sobrenome SIMILAR “@Silva@” SET BY Nome;

C) SELECT Nome, Função FROM Funcionário WHERE Sobrenome LIKE “%Silva%” ORDER BY Nome;

D) SELECT Nome, Função FROM Funcionário WHERE Sobrenome = &Silva& Nome ↑;

E) SELECT Nome, Função FROM Funcionário WHERE Sobrenome JUST LIKE “$Silva%” PUT Nome ↑;

Comentários:

O comando SQL correto para obter o Nome e a Função dos funcionários com sobrenome Silva, em qualquer parte do sobrenome, ordenados de forma ascendente pelo nome, é:

SELECT Nome, Função FROM Funcionário WHERE Sobrenome LIKE '%Silva%' ORDER BY Nome;

Nessa consulta, utilizamos a cláusula WHERE para filtrar os funcionários cujo Sobrenome contém a sequência 'Silva'. O operador LIKE é utilizado com os curingas '%' antes e depois de 'Silva', indicando que o Sobrenome pode ter qualquer sequência de caracteres antes e depois da sequência 'Silva'. Em seguida, utilizamos a cláusula ORDER BY para ordenar os resultados pelo atributo Nome, de forma ascendente.

Gabarito: C

21. (VUNESP - 2021 - Semae de Piracicaba - SP - Programador Junior) Considere a seguinte tabela de um banco de dados relacional: Item (Código, Tipo, Nome, Quantidade)

O comando SQL para obter o tipo e o valor médio da quantidade, por tipo de itens, apenas para valores médios superiores a 500, é:

A) SELECT Tipo, Quantidade-Média FROM Item HAVING Quantidade-Média > 500;

B) SELECT Tipo, LIMIT (Quantidade) > 500 FROM Item GROUP BY Tipo

C) SELECT Tipo, AVG (Quantidade) FROM Item GROUP BY Tipo HAVING AVG (Quantidade) > 500;

D) SELECT Tipo, MED (Quantidade) FROM Item GROUP BY Tipo WHERE MED (Quantidade) > 500;

E) SELECT Tipo, MAX - MIN (Quantidade) FROM Item GROUP BY Tipo FOR MAX – MIN (Quantidade) > 500;

Comentários:

O comando SQL correto para obter o tipo e o valor médio da quantidade, por tipo de itens, apenas para valores médios superiores a 500, é:

SELECT Tipo, AVG(Quantidade) FROM Item GROUP BY Tipo HAVING AVG(Quantidade) > 500;

Nessa consulta, utilizamos a cláusula GROUP BY para agrupar os itens pelo tipo. Em seguida, utilizamos a função AVG(Quantidade) para calcular a média da quantidade para cada tipo de item. A cláusula HAVING é utilizada para filtrar apenas os grupos cuja média da quantidade é maior que 500.

Gabarito: C

22. (VUNESP - 2021 - TJM-SP - Técnico em Comunicação e Processamento de Dados Judiciário (Desenvolvedor)) Deseja-se criar uma visão denominada Glass, que deve

conter os campos Ball e Paper presentes na tabela Agency. O comando SQL para criar tal visão é:

A) CREAT VIEW Glass AS

(SELECT Ball, Paper

FROM Agency)

B) CREAT VIEW Glass

(SELECT Ball UNION Paper

FROM Agency)

C) CREAT VIEW Glass

(SELECT Ball AND Paper

FROM Agency)

D) CREAT VIEW Glass

(FROM Agency.Ball, Agency.Paper)

E) CREAT VIEW AS Glass

(SELECT Agency.Ball, Agency.Paper)

Comentários:

O comando SQL correto para criar uma visão denominada Glass, que contém os campos Ball e Paper presentes na tabela Agency, é:

CREATE VIEW Glass AS

(SELECT Ball, Paper

FROM Agency)

Nesse comando, utilizamos a cláusula CREATE VIEW para criar a visão chamada Glass. Em seguida, utilizamos a sintaxe "(SELECT Ball, Paper FROM Agency)" para selecionar os campos Ball e Paper da tabela Agency e definir os dados que serão exibidos na visão.

Gabarito: A

23. (FCC / Copergás-PE - 2023) Em uma tabela chamada user de um banco de dados aberto e em condições ideais, para selecionar todos os registros que possuem nomes (campo nome) iniciados com a letra E e terminados com a letra l utiliza-se a instrução

SQL SELECT * FROM user

a) LIKE = 'E*l';

b) WHERE nome = 'E%l';

c) LIKE nome CONTAINS 'E%l';

d) WHERE nome LIKE 'E*l';

e) WHERE nome LIKE 'E%l';

Comentários:

(a) Errado. O operador LIKE deve ser usado com o símbolo de porcentagem (%) para representar zero ou mais caracteres;

(b) Errado. O operador de igualdade (=) não permite a utilização de caracteres curinga para representar múltiplos caracteres desconhecidos;

(c) Errado. A sintaxe utilizada está incorreta – o operador LIKE deve ser usado diretamente no filtro da cláusula WHERE;

(d) Errado. O operador LIKE deve ser usado com o símbolo de porcentagem (%) para representar zero ou mais caracteres;

(e) Correto. Utiliza o operador LIKE com o padrão 'E%l' para selecionar todos os registros que possuem nomes iniciados com a letra E e terminados com a letra l.

Gabarito: Letra E

24. (FCC / Copergás-PE - 2023) O comando SQL que está correto, sem erros de sintaxe, é:

a) SELECT Servicos.ServicoID, Clientes.NomeCliente FROM Servicos INNER JOIN Clientes ON Servicos.ClienteID = Clientes.ClienteID;

b) SELECT COUNT(DISTINCT Cidade) FROM TABLE Clientes;

c) SELECT * FROM Clientes WHERE Cidade IS LIKE 'Recife';

d) UPDATE Clientes TO NomeCliente = 'Maria da Silva' AND City= 'Caruaru' WHERE ClienteID IS 11234;

e) DELETE FROM Clientes WHERE NomeCliente=*.*;

Comentários:

(a) Correto, sem erros; (b) Errado, não é necessário inserir a palavra TABLE; (c) Errado, não é necessário inserir a palavra IS – apenas LIKE; (d) Errado, não é necessário inserir a palavra TO; (e) Errado, para especificar o ponto, utiliza-se aspas (“.”) e, não, asteriscos (*.*).

Gabarito: Letra A

25. (FCC / SEFAZ-PE – 2022)

CREATE TABLE nfe (

Numero_NFe VARCHAR(9) NOT NULL,

Modelo_NFe VARCHAR(2) NULL,

Serie_NFe VARCHAR(3) NULL,

codigo_UF VARCHAR(2) NULL,

ano_Emissao VARCHAR(2) NULL,

mes_Emissao VARCHAR(2) NULL,

CNPJ_Emitente VARCHAR(14) NULL,

Codigo_Chave VARCHAR(8) NULL,

Digito_Chave VARCHAR(1) NULL,

PRIMARY KEY (Numero_NFe));

Para inserir um registro com valores de teste na tabela nfe, utiliza-se a instrução SQL:

a) INSERT INTO nfe VALUES('123456789','55','1','RJ','22','10','15145076000106','87654321','6');

b) APPEND TO nfe VALUES('123456789','55','1','RJ','22','10','15145076000106','87654321','6');

c) INSERT INTO nfe('123456789','55','1','RJ','22','10','15145076000106','87654321','6');

d) INSERT TO nfe VALUES('123456789','55','1','RJ','22','10','15145076000106','87654321','6');

e) ADD INTO nfe('123456789','55','1','RJ','22','10','15145076000106','87654321','6');

Comentários:

A instrução utilizada para inserir registros em uma tabela do banco de dados segue um dos seguintes formatos:

INSERT INTO Nome_Tabela (Coluna1, Coluna2, ...)

VALUES (Valor1, Valor2, ...)

Ou (quando se está adicionando valores para todas as colunas da tabela):

INSERT INTO Nome_Tabela

VALUES (Valor1, Valor2, ...)

A única alternativa que segue o modelo apresentado é a letra (a):

INSERT INTO nfe

VALUES ('123456789','55','1','RJ','22','10','15145076000106','87654321','6');

Gabarito: Letra A

25. (CESGRANRIO - 2023 - Banco do Brasil - Agente de Tecnologia - Microrregião 158 - TI)

A CNAE (Classificação Nacional de Atividades Econômicas), de responsabilidade do IBGE, possui códigos que são utilizados para caracterizar as atividades econômicas das empresas no Brasil. Por exemplo: empresas da área de construção de edifícios utilizam o CNAE de código 4120-4/00 para caracterizar a sua atividade econômica principal. Considere que existe um banco de dados em uma empresa, que concede empréstimos a outras empresas, com as seguintes tabelas:

Empresa (CNPJ, razaoSocial, endereco, atividade) CNAE (codigo, descricao)

A coluna "atividade" da tabela Empresa é uma chave estrangeira que referencia a coluna "codigo" da tabela CNAE. Que comando SELECT do SQL retorna apenas o CNPJ e a razão social das empresas cuja atividade econômica principal é a construção de edifícios (código 4120-4/00)?

A) SELECT * FROM Empresa E, CNAE C WHERE E.atividade = C.codigo

B) SELECT * FROM Empresa WHERE atividade = 'construção de edifícios'

C) SELECT CNPJ, razaoSocial FROM CNAE WHERE codigo = '4120-4/00'

D) SELECT CNPJ, razaoSocial FROM Empresa E, CNAE C WHERE E.atividade = C.codigo AND C.codigo = 'construção de edifícios'

E) SELECT CNPJ, razaoSocial FROM Empresa WHERE atividade = '4120-4/00'

Comentários:

A justificativa para o gabarito E ser correto é que a consulta SQL especificada seleciona as colunas CNPJ e razão social da tabela Empresa com base em uma condição específica relacionada à atividade econômica principal. A condição "WHERE atividade = '4120-4/00'" filtra as empresas cuja atividade econômica principal corresponde ao código '4120-4/00', que é a construção de edifícios. Este comando SQL é direto e eficiente, pois ele consulta diretamente a tabela Empresa e usa a coluna de atividade (que é uma chave estrangeira referenciando a tabela CNAE) para filtrar os resultados. As outras opções são incorretas porque não especificam corretamente a condição (alternativas A e D), referem-se à tabela errada para a consulta desejada (alternativa C), ou usam uma descrição textual errada da atividade econômica principal (alternativa B) em vez do código CNAE correspondente.

Gabarito: E

26. (CESGRANRIO - 2023 - Banco do Brasil - Agente de Tecnologia - Microrregião 158 - TI)

O banco de dados de uma empresa de investimentos financeiros possui as seguintes tabelas:

Empresa (CNPJ, razaoSocial, endereco) UF (sigla, nome)

O que o comando SQL "SELECT CNPJ, sigla FROM Empresa, UF" recupera desse banco de dados?

A) A sigla da UF das sedes das empresas cadastradas.

B) Alguns pares de CNPJ e sigla, onde o nome da UF é igual à razão social da empresa.

C) O CNPJ das empresas cadastradas cuja sigla de UF esteja na tabela UF.

D) Pares de CNPJ e sigla de todas as empresas cadastradas com as UFs de suas respectivas sedes.

E) Todos os pares de CNPJ e sigla possíveis, de todas as empresas e de todas as UFs cadastradas.

Comentários:

O gabarito correto para essa questão é a opção E. O comando SQL "SELECT CNPJ, sigla FROM Empresa, UF" executa um produto cartesiano entre as tabelas Empresa e UF. Isso significa que ele combinará cada linha da tabela Empresa com cada linha da tabela UF, resultando em todos os pares possíveis de CNPJ (de todas as empresas) e sigla (de todas as unidades federativas). O comando não especifica uma condição de junção ou relacionamento entre as duas tabelas, portanto, não está limitando a consulta a uma correspondência específica entre as sedes das empresas e suas respectivas UFs. Em vez disso, ele simplesmente gera uma lista de todas as combinações possíveis entre os CNPJs das empresas e as siglas das UFs, sem levar em conta qualquer relação lógica ou real entre esses dois elementos. Isso resulta em uma quantidade significativamente grande de combinações, a maioria das quais não possui um significado prático dentro do contexto das informações armazenadas no banco de dados.

Gabarito: E

27. (CESGRANRIO - 2022 - Banco da Amazônia - Técnico Científico - Tecnologia da Informação)

Considere que em um banco de dados de um banco comercial há duas tabelas:

PESSOA_FISICA (CPF, nome, email, telefone)

CLIENTE (CPF, nome, email, telefone).

Um funcionário de TI recebeu a tarefa de identificar corretamente quais pessoas físicas, cadastradas na tabela PESSOA_FISICA, ainda não eram clientes, pois não estavam cadastradas na tabela CLIENTE. Para isso, ele utilizou um comando SELECT em SQL.

Que trecho, em SQL, faz parte de uma das possíveis soluções para essa tarefa?

A) ... WHERE PESSOA_FISICA.CPF NOT IN (SELECT CPF FROM CLIENTE...

B) ... HAVING PESSOA_FISICA.CPF != CLIENTE.CPF...

C) ... WHERE PESSOA_FISICA.CPF <> CLIENTE.CPF...

D) ... DISTINCT PESSOA_FISICA.CPF FROM CLIENTE WHERE ...

E) ... IN PESSOA_FISICA BUT NOT IN CLIENTE...

Comentários:

O gabarito correto para essa questão é a opção A. Esta opção utiliza a cláusula WHERE em conjunto com NOT IN para identificar as pessoas físicas que não são clientes. A subconsulta SELECT CPF FROM CLIENTE gera uma lista de todos os CPFs presentes na tabela CLIENTE. A cláusula WHERE PESSOA_FISICA.CPF NOT IN compara cada CPF na tabela PESSOA_FISICA com essa lista e seleciona apenas aqueles CPFs que não estão presentes na tabela CLIENTE, satisfazendo assim o objetivo de identificar pessoas físicas que ainda não são clientes. As outras opções são incorretas porque não estabelecem a comparação correta entre as duas tabelas para a tarefa especificada. A opção B é inadequada porque a cláusula HAVING é usada para filtrar grupos de registros e não para comparar individualmente registros de duas tabelas. A opção C não funciona sem uma junção adequada entre as tabelas, e as opções D e E não representam comandos SQL válidos ou logicamente corretos para a tarefa em questão.

Gabarito: A

QUESTIONÁRIO DE REVISÃO E APERFEIÇOAMENTO

A ideia do questionário é elevar o nível da sua compreensão no assunto e, ao mesmo tempo, proporcionar uma outra forma de revisão de pontos importantes do conteúdo, a partir de perguntas que exigem respostas subjetivas.

São questões um pouco mais desafiadoras, porque a redação de seu enunciado não ajuda na sua resolução, como ocorre nas clássicas questões objetivas.

O objetivo é que você realize uma auto explicação mental de alguns pontos do conteúdo, para consolidar melhor o que aprendeu ;)

Além disso, as questões objetivas, em regra, abordam pontos isolados de um dado assunto. Assim, ao resolver várias questões objetivas, o candidato acaba memorizando pontos isolados do conteúdo, mas muitas vezes acaba não entendendo como esses pontos se conectam.

Assim, no questionário, buscaremos trazer também situações que ajudem você a conectar melhor os diversos pontos do conteúdo, na medida do possível.

É importante frisar que não estamos adentrando em um nível de profundidade maior que o exigido na sua prova, mas apenas permitindo que você compreenda melhor o assunto de modo a facilitar a resolução de questões objetivas típicas de concursos, ok?

Nosso compromisso é proporcionar a você uma revisão de alto nível!

Vamos ao nosso questionário:

Perguntas

1. O que é SQL?
2. O que significa DDL?
3. O que é DML?
4. O que DCL representa?
5. O que é DTL?
6. O que é um dialeto SQL?
7. O que é uma tabela em SQL?
8. O que é uma linha em uma tabela de banco de dados?
9. O que é uma coluna em uma tabela de banco de dados?
10. O que o comando CREATE faz em SQL?
11. O que o comando DROP faz em SQL?
12. O que o comando ALTER faz em SQL?
13. O que o comando RENAME faz em SQL?
14. O que o comando TRUNCATE faz em SQL?
15. O que é uma view em SQL?
16. O que é um índice em SQL?
17. O que é um tipo de dados de texto em SQL?
18. O que é um tipo de dados numéricos em SQL?
19. O que é um tipo de dados de tempo em SQL?
20. O que é um tipo de dados binários em SQL?
21. O que é uma restrição de integridade em SQL?
22. O que é uma chave primária em SQL?
23. O que é uma chave estrangeira em SQL?

24. O que é uma restrição de unicidade em SQL?
25. O que é uma restrição de não nulidade em SQL?
26. O que o comando SELECT faz em SQL?
27. O que o comando INSERT faz em SQL?
28. O que o comando UPDATE faz em SQL?
29. O que o comando DELETE faz em SQL?
30. O que é a cláusula GROUP BY em SQL?
31. O que é um alias em SQL?
32. O que é a cláusula DISTINCT em SQL?
33. O que são operadores em SQL?
34. O que a operação UNION faz em SQL?
35. O que a operação INTERSECT faz em SQL?
36. O que a operação EXCEPT faz em SQL?
37. O que é uma junção interna (INNER JOIN) em SQL?
38. O que é uma junção à esquerda (LEFT JOIN) em SQL?
39. O que é uma junção à direita (RIGHT JOIN) em SQL?
40. O que é uma junção completa (FULL JOIN) em SQL?
41. O que é uma função agregada em SQL?
42. O que é uma Stored Procedure em SQL?
43. O que é uma Function em SQL?
44. O que é um Trigger em SQL?
45. O que é o comando GRANT em SQL?
46. O que é o comando REVOKE em SQL?
47. O que é um Role em SQL?

Perguntas e Respostas

1. O que é SQL?
   Resposta: SQL, ou Linguagem de Consulta Estruturada, é uma linguagem usada para gerenciar e manipular bancos de dados relacionais.
2. O que significa DDL?
   Resposta: DDL, ou Linguagem de Definição de Dados, é uma sublinguagem de SQL usada para definir estruturas de dados e objetos de banco de dados.
3. O que é DML?
   Resposta: DML, ou Linguagem de Manipulação de Dados, é uma sublinguagem de SQL usada para manipular dados dentro de objetos de banco de dados.

4. O que DCL representa?
   Resposta: DCL, ou Linguagem de Controle de Dados, é uma sublinguagem de SQL usada para controlar o acesso a dados no banco de dados.
5. O que é DTL?
   Resposta: DTL, ou Linguagem de Transação de Dados, é uma sublinguagem de SQL usada para gerenciar transações de banco de dados.
6. O que é um dialeto SQL?
   Resposta: Um dialeto SQL é uma variante de SQL que possui características e funcionalidades adicionais que não são padronizadas pela SQL padrão.
7. O que é uma tabela em SQL?
   Resposta: Uma tabela em SQL é um objeto de banco de dados que organiza os dados em linhas e colunas.
8. O que é uma linha em uma tabela de banco de dados?
   Resposta: Uma linha, também conhecida como registro ou tupla, é um único conjunto de campos em uma tabela. Cada linha contém um conjunto único de dados.
9. O que é uma coluna em uma tabela de banco de dados?
   Resposta: Uma coluna, também conhecida como campo, é uma categoria vertical de dados em uma tabela.
10. O que o comando CREATE faz em SQL?
    Resposta: O comando CREATE é usado para criar novos objetos de banco de dados, como tabelas, índices e bancos de dados.
11. O que o comando DROP faz em SQL?
    Resposta: O comando DROP é usado para excluir objetos existentes do banco de dados, como tabelas, índices ou o próprio banco de dados.
12. O que o comando ALTER faz em SQL?
    Resposta: O comando ALTER é usado para modificar a estrutura de um objeto existente no banco de dados, como adicionar ou remover colunas de uma tabela ou alterar o tipo de dados de uma coluna.
13. O que o comando RENAME faz em SQL?
    Resposta: O comando RENAME é usado para renomear um objeto existente no banco de dados, como uma tabela ou uma coluna.
14. O que o comando TRUNCATE faz em SQL?
    Resposta: O comando TRUNCATE é usado para remover todos os registros de uma tabela, mas mantém a estrutura da tabela para uso futuro.
15. O que é uma view em SQL?
    Resposta: Uma view é uma representação virtual de uma tabela que é baseada em uma consulta SQL. As views são usadas para simplificar consultas complexas e garantir a segurança dos dados.
16. O que é um índice em SQL?
    Resposta: Um índice é uma estrutura de dados que melhora a velocidade das operações

de recuperação de dados em um banco de dados. Ele funciona como o índice de um livro, permitindo

ao banco de dados encontrar dados mais rapidamente.

17. O que é um tipo de dados de texto em SQL?
Resposta: Um tipo de dados de texto é um tipo de dados usado para armazenar cadeias de caracteres de qualquer comprimento. Exemplos incluem CHAR, VARCHAR e TEXT.

18. O que é um tipo de dados numéricos em SQL?
Resposta: Um tipo de dados numérico é um tipo de dados usado para armazenar valores numéricos. Exemplos incluem INT, FLOAT, DECIMAL e NUMERIC.

19. O que é um tipo de dados de tempo em SQL?
Resposta: Um tipo de dados de tempo é um tipo de dados usado para armazenar informações de data e hora. Exemplos incluem DATE, TIME e DATETIME.

20. O que é um tipo de dados binários em SQL?
Resposta: Um tipo de dados binários é um tipo de dados usado para armazenar dados binários, como imagens ou arquivos. Exemplos incluem BINARY, VARBINARY e BLOB.

21. O que é uma restrição de integridade em SQL?
Resposta: Uma restrição de integridade é uma regra que é usada para manter a precisão e a integridade dos dados em um banco de dados.

22. O que é uma chave primária em SQL?
Resposta: Uma chave primária é uma coluna ou conjunto de colunas que identifica exclusivamente cada linha em uma tabela.

23. O que é uma chave estrangeira em SQL?
Resposta: Uma chave estrangeira é uma coluna ou conjunto de colunas que é usado para estabelecer e impor um link entre os dados em duas tabelas.

24. O que é uma restrição de unicidade em SQL?
Resposta: Uma restrição de unicidade é uma regra que impede que duas linhas tenham o mesmo valor na mesma coluna.

25. O que é uma restrição de não nulidade em SQL?
Resposta: Uma restrição de não nulidade é uma regra que impede que uma coluna tenha um valor NULL.

26. O que o comando SELECT faz em SQL?
Resposta: O comando SELECT é usado para selecionar dados de uma ou mais tabelas.

27. O que o comando INSERT faz em SQL?
Resposta: O comando INSERT é usado para inserir novos registros em uma tabela.

28. O que o comando UPDATE faz em SQL?
Resposta: O comando UPDATE é usado para modificar registros existentes em uma tabela.

29. O que o comando DELETE faz em SQL?
Resposta: O comando DELETE é usado para excluir registros existentes de uma tabela.

30. O que é a cláusula GROUP BY em SQL?
Resposta: A cláusula GROUP BY é usada para agrupar linhas que têm os mesmos valores em colunas específicas em conjunto.

31. O que é um alias em SQL?
Resposta: Um alias é um nome temporário dado a uma tabela ou coluna para a duração de uma consulta SQL.

32. O que é a cláusula DISTINCT em SQL?
Resposta: A cláusula DISTINCT é usada para remover valores duplicados de um conjunto de resultados.

33. O que são operadores em SQL?
Resposta: Operadores são símbolos ou palavras-chave usados para especificar operações a serem realizadas em dados.

34. O que a operação UNION faz em SQL?
Resposta: A operação UNION é usada para combinar os conjuntos de resultados de duas ou mais consultas SELECT em um único conjunto de resultados.

35. O que a operação INTERSECT faz em SQL?
Resposta: A operação INTERSECT é usada para retornar apenas as linhas que aparecem em ambos os conjuntos de resultados de duas ou mais consultas SELECT.

36. O que a operação EXCEPT faz em SQL?
Resposta: A operação EXCEPT é usada para retornar apenas as linhas que aparecem no primeiro conjunto de resultados de duas consultas SELECT.

37. O que é uma junção interna (INNER JOIN) em SQL?
Resposta: Uma junção interna é usada para combinar linhas de duas ou mais tabelas com base em uma condição relacionada entre eles.

38. O que é uma junção à esquerda (LEFT JOIN) em SQL?
Resposta: Uma junção à esquerda é usada para retornar todas as linhas da tabela esquerda e as linhas correspondentes da tabela direita. Se não houver correspondência, o resultado é NULL do lado direito.

39. O que é uma junção à direita (RIGHT JOIN) em SQL?
Resposta: Uma junção à direita é usada para retornar todas as linhas da tabela direita e as linhas correspondentes da tabela esquerda. Se não houver correspondência, o resultado é NULL do lado esquerdo.

40. O que é uma junção completa (FULL JOIN) em SQL?
Resposta: Uma junção completa é usada para retornar todas as linhas quando há uma correspondência em qualquer das tabelas esquerda ou direita.

41. O que é uma função agregada em SQL?
Resposta: Uma função agregada é uma função que realiza um cálculo em um conjunto de valores e retorna um único valor. Exemplos incluem COUNT, SUM, AVG, MAX, MIN.

42. O que é uma Stored Procedure em SQL?
Resposta: Uma Stored Procedure é um grupo salvo de comandos SQL que pode ser

reutilizado. As Stored Procedures podem aceitar parâmetros de entrada e retornar vários valores.

43. O que é uma Function em SQL?
Resposta: Uma Function é um tipo de procedimento armazenado que retorna um único valor. As Functions aceitam parâmetros de entrada, mas diferem das Stored Procedures porque não podem alterar o estado do banco de dados.

44. O que é um Trigger em SQL?
Resposta: Um Trigger é um procedimento armazenado que é automaticamente executado em resposta a eventos específicos em uma tabela ou view, como a inserção, atualização ou exclusão de registros.

45. O que é o comando GRANT em SQL?
Resposta: O comando GRANT é usado para conceder privilégios a usuários, grupos de usuários ou funções.

46. O que é o comando REVOKE em SQL?
Resposta: O comando REVOKE é usado para remover privilégios concedidos a usuários, grupos de usuários ou funções.

47. O que é um Role em SQL?
Resposta: Um Role é um conjunto de permissões que podem ser concedidas a usuários ou outras roles. Roles são usadas para gerenciar facilmente as permissões em um banco de dados.

LISTA DE QUESTÕES ESTRATÉGICAS

1. **(FGV / Receita Federal - 2023)** Considere um banco de dados relacional em que as operações de insert e update efetuadas numa certa tabela devem ser monitoradas e anotadas, como subsídio aos procedimentos de auditoria da empresa. Essa tabela é utilizada por uma série de aplicações, em diferentes tipos de transações, e iniciadas por um número considerável de usuários.

   Nesse cenário, assinale o mecanismo mais adequado para a implementação desse monitoramento:

   a) Cursores.

b) Stored procedures.

c) Triggers.

d) Utilitários de exportação de dados.

e) Views.

2. **(FGV / Receita Federal - 2023)** Os principais Sistemas Gerenciadores de Bancos de Dados oferecem total suporte à linguagem SQL. Um aspecto importante da implementação do SQL é o tratamento para valores nulos quando esses são considerados como unknown values.

Nesse contexto, considere uma tabela T com colunas A e B, que podem conter valores nulos. T possui 100 registros e, em 50% das linhas, há pelo menos uma coluna preenchida com o valor NULL. Considere a consulta a seguir:

```
SELECT * FROM T t1 WHERE t1.A = NULL or t1.B = NULL
```

O número máximo de linhas de resultados que seriam retornadas pela consulta é igual a:

a) 0.

b) 25.

c) 50.

d) 75.

e) 100.

3. **(FGV / TRT-MA – 2022)** Assinale a afirmativa correta a respeito do esquema relacional apresentado.

```
create table X(A int not null primary key,
        B int)
create table Y(A int not null UNIQUE,
        constraint fk
  foreign key (A) references X(A)
  on delete cascade)
```

a) a tabela X admite linhas duplicadas.

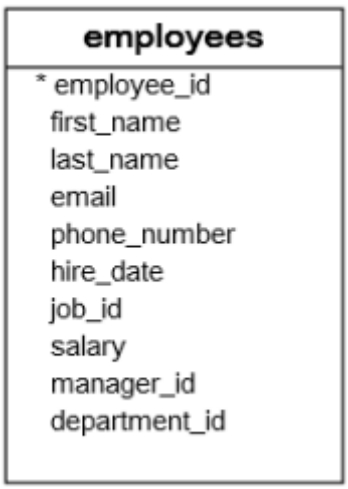
b) a tabela X não pode ter mais linhas que a tabela Y

c) a tabela Y admite linhas duplicadas.

d) a tabela Y não pode ter mais linhas que a tabela X.

e) as tabelas X e Y não podem ter o mesmo número de linhas.

4. **(FGV / SEFAZ-BA – 2022)** Considere a seguinte tabela em um banco de dados relacional.



Assinale a opção que indica o comando SQL utilizado para localizar todos os nomes completos dos *employees*, cujos primeiros nomes começam com as letras Ma.

a) SELECT

first_name;

last_name;

FROM

employees;

b) SELECT *

FROM

employees

WHERE

first_name = 'Ma';

c) SELECT *

FROM

employees

WHERE

first_name = 'Ma*';

d) SELECT

employee_id,

first_name,

last_name

FROM

employees

WHERE

first_name LIKE 'Ma%';

e) SELECT

employee_id,

first_name,

last_name

FROM

employees

WHERE

first_name IN 'Ma_';

5. **(FGV / FUNSÚDE-CE – 2021)** Atenção: na próxima questão, considere a definição e as instâncias das tabelas de bancos de dados CLUBE e JOGO exibidas a seguir.

CLUBE

nome

Barcelona

Boca Juniors

The Strongest

JOGO

mandante	visitante	golsM	golsV
Barcelona	Boca Juniors	1	0
Barcelona	The Strongest	NULL	NULL
Boca Juniors	Barcelona	0	0
Boca Juniors	The Strongest	3	0
The Strongest	Barcelona	2	0
The Strongest	Boca Juniors	2	0

Cada clube deve jogar quatro vezes, duas como mandante e duas como visitante. As colunas golsM e golsV registram o número de gols dos times mandantes e visitantes, respectivamente, em cada jogo. Ambas são nulas enquanto o jogo não for realizado.

Analise o comando SQL a seguir, à luz das definições e instâncias das tabelas CLUBE e JOGO, apresentadas anteriormente.

```
select distinct mandante, visitante from JOGO, CLUBE
```

Assinale o número de linhas, sem incluir os títulos, produzidas pela execução desse comando:

a) 4.

b) 6.

c) 10.

d) 24.

e) 48.

A	B	C	D
12	2	3	1
14	3	8	2
18	2	9	3
21	5	4	4

6. (FGV / TCE-AM – 2021) Considerando-se a instância da tabela T (descrita anteriormente), analise o comando SQL abaixo.

```
delete from T where b + d = c
```

O número de registros da tabela T afetados pela execução desse comando é:

a) zero;

b) um;

c) dois;

d) três;

e) quatro.

7. **(FGV / IMBEL – 2021)** Considere o comando SQL a seguir, executado num banco de dados relacional com duas tabelas, R1 e R2, contendo 2.000 e 5.000 registros, respectivamente. R1 e R2 possuem chaves primárias definidas.

```
SELECT DISTINCT * FROM A, B
```

Assinale o número de linhas produzidas na execução:

a) 1.

b) 2.000.

c) 5.000.

d) 7.000.

e) 10.000.000

8. **(FGV / Prefeitura de Niterói-RJ – 2018)** A otimização de consultas em gerenciadores de bancos de dados é fundamental para o desempenho do sistema. Consultas escritas em SQL são particularmente propícias à otimização, porque essa linguagem:

a) não é procedural, permitindo a criação de diferentes planos de execução.

b) tem uma sintaxe simples e é largamente utilizada.

c) suporta todas as operações da Álgebra Relacional.

d) permite o uso de subconsultas, facilitando os processos de busca.

e) permite a criação de camadas de software de persistência.

9. **(FGV / MPE-AL – 2018)** Considere o comando SQL a seguir.

```
select * from teste where nota <> null
```

Sabendo-se que a tabela teste tem 100 registros, dos quais 15 estão com a coluna nota ainda não preenchida (null), o comando acima, executado no MS SQL Server ou no Oracle, retorna

a) um erro, porque o literal null não pode ser usado diretamente numa comparação.

b) zero linhas.

c) quinze linhas.

d) oitenta e cinco linhas.

e) cem linhas.

10. **(FGV / BANESTES – 2018)** Considere um banco de dados com duas tabelas, R e S, contendo 4 e 3 registros, respectivamente. Em R, os valores da coluna A são 1, 2, 5 e 7. Em S, os valores da coluna B são 2, 4 e 7. Excetuando-se a linha de títulos, o número de linhas no resultado do comando SQL

```
select * from R full outer join S on A=B
```

É:

a) 3

b) 4

c) 5

d) 7

e) 12

11. (FGV / IBGE – 2017) A linguagem mais comum para elaboração de consultas em bancos de dados é a SQL. Ao elaborar uma consulta nessa linguagem, emprega-se a cláusula WHERE quando se deseja:

a) especificar a tabela onde será realizada a consulta;

b) especificar o diretório onde os dados estão armazenados;

c) especificar o endereço IP onde os dados estão armazenados;

d) especificar condições a que as instâncias selecionadas devem atender;

e) extrair a geometria do objeto selecionado na consulta.

R	
a	b
1	2
2	3
4	5

S	
c	d
3	2
4	2
6	1

12. (FGV / MPE-BA – 2017) Considerando as tabelas R e S apresentadas anteriormente, o resultado

a	b
1	2
2	3

seria obtido pela execução do comando SQL:

a) select * from R

where not exists (select * FROM S where a=c)

b) select * from R

where not exists (select * FROM S where a=d)

c) select * from S

where not exists (select * FROM R where a=c)

d) select * from S

where not exists (select * FROM R where a=d)

e) select * from R

where exists (select * FROM S where b=d)

13. (FGV / PGE-RO – 2015) No SQL, a outorga de privilégios sobre objetos de um banco de dados é efetuada por meio do comando:

a) CREATE;

b) GRANT;

c) LICENSE;

d) PERMIT;

e) REVOKE.

14. **(FGV / DPE-RO – 2015)** Observe o comando SQL a seguir.

```
update X set Y = 'Z'
```

Para que esse comando esteja corretamente formulado, quando analisado isoladamente, pressupõe-se que:

a) Y seja uma coluna da tabela X;

b) X seja uma coluna da tabela Y;

c) X e Y sejam tabelas;

d) X seja um banco de dados e Y seja uma tabela;

e) X seja um banco de dados e Y seja uma tabela;

15. **(FGV / TJ-BA – 2015)** Considere que as instâncias das tabelas T1, T2 e T3 têm, respectivamente, 1.000, 10.000 e 100.000 registros. O comando SQL:

```
select 1 from t1
```

```
union

select 2 from t2

union

select 3 from t3
```

produz um resultado com:

a) 3 linhas;

b) 1.000 linhas;

c) 10.000 linhas;

d) 100.000 linhas;

e) 111.000 linhas.

16. (FGV / TJ-AM – 2013) A SQL é constituída pela Data Control Language (DCL), a Data Definition Language (DDL) e a Data Manipulation Language (DML).

Assinale a alternativa que apresenta os três comandos que são parte integrante da DDL:

a) REVOKE, GRANT e DELETE

b) GRANT, DELETE e UPDATE

c) DELETE, UPDATE e CREATE

d) CREATE, ALTER e DROP

e) ALTER, DROP e REVOKE

17. (FGV / MPE-MS – 2013) Observe o comando SQL a seguir:

```
SELECT nome, sobrenome, PIS, anos_de_servico FROM Empregados
```

A cláusula que deve ser adicionada ao comando acima para ordenar os registros por anos de serviço, com os empregados que estão há mais tempo na empresa aparecendo primeiro na listagem, é:

a) ORDER 'anos_de_servico' BY ASC

b) ORDER BY 'anos_de_servico'

c) ORDER BY anos_de_servico DESC

d) SORTED BY anos_de_servico DESC

e) ORDER BY anos_de_servico ASC

18. (VUNESP - 2022 - UNICAMP - Analista de Desenvolvimento de Sistemas) Considere a seguinte tabela de um banco de dados relacional: Cliente (ID, Nome, Agencia, Saldo) A consulta SQL para obter o nome dos clientes que possuam saldo maior do que o saldo do cliente de ID igual a 200 é

A) SELECT T.Nome FROM Cliente S, Cliente T WHERE S.ID = 200 AND T.Saldo > S.Saldo

B) SELECT Cliente.Nome FROM Cliente WHERE Cliente.ID = 200

C) SELECT Cliente.Nome FROM Cliente.ID = 200

D) SELECT Cliente1.Nome FROM Cliente WHERE Cliente1.Saldo > Cliente2.Saldo AND Cliente1.ID > 200

E) SELECT Cliente1.Nome FROM Cliente1, Cliente2 WHERE Cliente2.Saldo > Cliente1.Saldo AND Cliente1.ID > 200

19. (VUNESP - 2022 - Prefeitura de Sorocaba - SP - Analista de Sistemas I)

Considere a seguinte tabela de um banco de dados relacional: Medicamento (ID, Nome, Indicação). A consulta SQL para obter o Nome e a Indicação dos medicamentos, ordenada pelo atributo Indicação, e cujo Nome termine com a sequência 'gem' é

A) SELECT Nome AND Indicação FROM Medicamento HAVING Nome IN '%gem' ORDER BY Indicação Asc;

B) SELECT Nome, Indicação FROM Medicamento WHERE Nome SIMILAR '%gem' GROUP BY Nome, Indicação;

C) SELECT Nome, Indicação FROM Medicamento WHERE Nome EQUAL '%gem' GROUP BY Indicação;

D) SELECT Nome, Indicação FROM Medicamento WHERE Nome LIKE '%gem' ORDER BY Indicação;

E) SELECT Nome AND Indicação FROM Medicamento HAVING Nome IN '%gem' ORDER BY 2;

20. (VUNESP - 2021 - Semae de Piracicaba - SP - Programador Junior)

Considere a seguinte tabela de um banco de dados relacional: Funcionário (CPF, Nome, Sobrenome, Função, Salário) O comando SQL para obter Nome e Função dos funcionários de sobrenome Silva, em qualquer parte do sobrenome, ordenados, de forma ascendente, pelo nome, é:

A) SELECT Nome, Função FROM Funcionário WHERE Sobrenome IN "#Silva#" SET BY Nome;

B) SELECT Nome, Função FROM Funcionário WHERE Sobrenome SIMILAR "@Silva@" SET BY Nome;

C) SELECT Nome, Função FROM Funcionário WHERE Sobrenome LIKE "%Silva%" ORDER BY Nome;

D) SELECT Nome, Função FROM Funcionário WHERE Sobrenome = &Silva& Nome ↑;

E) SELECT Nome, Função FROM Funcionário WHERE Sobrenome JUST LIKE "$Silva%" PUT Nome ↑;

21. (VUNESP - 2021 - Semae de Piracicaba - SP - Programador Junior) Considere a seguinte tabela de um banco de dados relacional: Item (Código, Tipo, Nome, Quantidade)

O comando SQL para obter o tipo e o valor médio da quantidade, por tipo de itens, apenas para valores médios superiores a 500, é:

A) SELECT Tipo, Quantidade-Média FROM Item HAVING Quantidade-Média > 500;

B) SELECT Tipo, LIMIT (Quantidade) > 500 FROM Item GROUP BY Tipo

C) SELECT Tipo, AVG (Quantidade) FROM Item GROUP BY Tipo HAVING AVG (Quantidade) > 500;

D) SELECT Tipo, MED (Quantidade) FROM Item GROUP BY Tipo WHERE MED (Quantidade) > 500;

E) SELECT Tipo, MAX - MIN (Quantidade) FROM Item GROUP BY Tipo FOR MAX – MIN (Quantidade) > 500;

22. **(VUNESP - 2021 - TJM-SP - Técnico em Comunicação e Processamento de Dados Judiciário (Desenvolvedor))** Deseja-se criar uma visão denominada Glass, que deve

conter os campos Ball e Paper presentes na tabela Agency. O comando SQL para criar tal visão é:

A) CREAT VIEW Glass AS

(SELECT Ball, Paper

FROM Agency)

B) CREAT VIEW Glass

(SELECT Ball UNION Paper

FROM Agency)

C) CREAT VIEW Glass

(SELECT Ball AND Paper

FROM Agency)

D) CREAT VIEW Glass

(FROM Agency.Ball, Agency.Paper)

E) CREAT VIEW AS Glass

(SELECT Agency.Ball, Agency.Paper)

23. **(FCC / Copergás-PE - 2023)** Em uma tabela chamada user de um banco de dados aberto e em condições ideais, para selecionar todos os registros que possuem nomes (campo nome) iniciados com a letra E e terminados com a letra l utiliza-se a instrução

SQL SELECT * FROM user

a) LIKE = 'E*l';

b) WHERE nome = 'E%l';

c) LIKE nome CONTAINS 'E%l';

d) WHERE nome LIKE 'E*l';

e) WHERE nome LIKE 'E%l';

24. (FCC / Copergás-PE - 2023) O comando SQL que está correto, sem erros de sintaxe, é:

a) SELECT Servicos.ServicoID, Clientes.NomeCliente FROM Servicos INNER JOIN Clientes ON Servicos.ClienteID = Clientes.ClienteID;

b) SELECT COUNT(DISTINCT Cidade) FROM TABLE Clientes;

c) SELECT * FROM Clientes WHERE Cidade IS LIKE 'Recife';

d) UPDATE Clientes TO NomeCliente = 'Maria da Silva' AND City= 'Caruaru' WHERE ClienteID IS 11234;

e) DELETE FROM Clientes WHERE NomeCliente=*.*;

25. (FCC / SEFAZ-PE – 2022)

CREATE TABLE nfe (

Numero_NFe VARCHAR(9) NOT NULL,

Modelo_NFe VARCHAR(2) NULL,

Serie_NFe VARCHAR(3) NULL,

codigo_UF VARCHAR(2) NULL,

ano_Emissao VARCHAR(2) NULL,

mes_Emissao VARCHAR(2) NULL,

CNPJ_Emitente VARCHAR(14) NULL,

Codigo_Chave VARCHAR(8) NULL,

Digito_Chave VARCHAR(1) NULL,

PRIMARY KEY (Numero_NFe));

Para inserir um registro com valores de teste na tabela nfe, utiliza-se a instrução SQL:

a) INSERT INTO nfe VALUES('123456789','55','1','RJ','22','10','15145076000106','87654321','6');

b) APPEND TO nfe VALUES('123456789','55','1','RJ','22','10','15145076000106','87654321','6');

c) INSERT INTO nfe('123456789','55','1','RJ','22','10','15145076000106','87654321','6');

d) INSERT TO nfe VALUES('123456789','55','1','RJ','22','10','15145076000106','87654321','6');

e) ADD INTO nfe('123456789','55','1','RJ','22','10','15145076000106','87654321','6');

26. (CESGRANRIO - 2023 - Banco do Brasil - Agente de Tecnologia - Microrregião 158 - TI)

A CNAE (Classificação Nacional de Atividades Econômicas), de responsabilidade do IBGE, possui códigos que são utilizados para caracterizar as atividades econômicas das empresas no Brasil. Por exemplo: empresas da área de construção de edifícios utilizam o CNAE de código 4120-4/00 para caracterizar a sua atividade econômica principal. Considere que existe um banco de dados em uma empresa, que concede empréstimos a outras empresas, com as seguintes tabelas:

Empresa (CNPJ, razaoSocial, endereco, atividade) CNAE (codigo, descricao)

A coluna "atividade" da tabela Empresa é uma chave estrangeira que referencia a coluna "codigo" da tabela CNAE. Que comando SELECT do SQL retorna apenas o CNPJ e a razão social das empresas cuja atividade econômica principal é a construção de edifícios (código 4120-4/00)?

A) SELECT * FROM Empresa E, CNAE C WHERE E.atividade = C.codigo

B) SELECT * FROM Empresa WHERE atividade = 'construção de edifícios'

C) SELECT CNPJ, razaoSocial FROM CNAE WHERE codigo = '4120-4/00'

D) SELECT CNPJ, razaoSocial FROM Empresa E, CNAE C WHERE E.atividade = C.codigo AND C.codigo = 'construção de edifícios'

E) SELECT CNPJ, razaoSocial FROM Empresa WHERE atividade = '4120-4/00'

27. (CESGRANRIO - 2023 - Banco do Brasil - Agente de Tecnologia - Microrregião 158 - TI)

O banco de dados de uma empresa de investimentos financeiros possui as seguintes tabelas:

Empresa (CNPJ, razaoSocial, endereco) UF (sigla, nome)

O que o comando SQL "SELECT CNPJ, sigla FROM Empresa, UF" recupera desse banco de dados?

A) A sigla da UF das sedes das empresas cadastradas.

B) Alguns pares de CNPJ e sigla, onde o nome da UF é igual à razão social da empresa.

C) O CNPJ das empresas cadastradas cuja sigla de UF esteja na tabela UF.

D) Pares de CNPJ e sigla de todas as empresas cadastradas com as UFs de suas respectivas sedes.

E) Todos os pares de CNPJ e sigla possíveis, de todas as empresas e de todas as UFs cadastradas.

28. (CESGRANRIO - 2022 - Banco da Amazônia - Técnico Científico - Tecnologia da Informação)

Considere que em um banco de dados de um banco comercial há duas tabelas:

PESSOA_FISICA (CPF, nome, email, telefone)

CLIENTE (CPF, nome, email, telefone).

Um funcionário de TI recebeu a tarefa de identificar corretamente quais pessoas físicas, cadastradas na tabela PESSOA_FISICA, ainda não eram clientes, pois não estavam cadastradas na tabela CLIENTE. Para isso, ele utilizou um comando SELECT em SQL.

Que trecho, em SQL, faz parte de uma das possíveis soluções para essa tarefa?

A) ... WHERE PESSOA_FISICA.CPF NOT IN (SELECT CPF FROM CLIENTE...

B) ... HAVING PESSOA_FISICA.CPF != CLIENTE.CPF...

C) ... WHERE PESSOA_FISICA.CPF <> CLIENTE.CPF...

D) ... DISTINCT PESSOA_FISICA.CPF FROM CLIENTE WHERE ...

E) ... IN PESSOA_FISICA BUT NOT IN CLIENTE...

Gabaritos

1. C
2. A
3. D
4. D
5. B
6. B
7. E
8. A
9. B
10. C
11. D
12. A
13. B
14. A
15. A
16. D
17. C
18. A
19. D
20. C
21. C
22. A
23. E
24. A
25. A
26. E
27. E
28. A

Questões Adicionais

As questões apresentadas a seguir integram o Banco de Questões do Passo Estratégico. Recomenda-se utilizá-las como um recurso complementar para a prática e consolidação dos conhecimentos adquiridos no material teórico, de acordo com o estilo adotado pela banca organizadora.

Bom estudo!

1. Qual operador SQL é usado para buscar um padrão em uma string?

A) IN
B) LIKE
C) CONTAINS
D) EQUALS
E) MATCHES

2. Suponha que você esteja trabalhando com um banco de dados de uma universidade que possui uma tabela chamada 'Estudantes' e outra chamada 'Cursos'. Cada registro na tabela 'Estudantes' possui um identificador único de estudante (id_estudante) e cada registro na tabela 'Cursos' possui um identificador de curso (id_curso). Você deseja criar uma consulta SQL que retorne o nome de cada estudante junto com o nome do curso em que ele está matriculado. Qual comando SQL realiza essa tarefa corretamente, considerando que a tabela 'Estudantes' tem uma coluna 'curso_id' que corresponde à coluna 'id_curso' na tabela 'Cursos'?

A) SELECT Estudantes.nome, Cursos.nome FROM Estudantes INNER JOIN Cursos ON Estudantes.id_estudante = Cursos.curso_id;
B) SELECT Estudantes.nome, Cursos.nome FROM Estudantes LEFT JOIN Cursos ON Estudantes.id_estudante = Cursos.id_curso;
C) SELECT Estudantes.nome, Cursos.nome FROM Estudantes INNER JOIN Cursos ON Estudantes.curso_id = Cursos.id_curso;
D) SELECT Estudantes.nome, Cursos.nome FROM Estudantes RIGHT JOIN Cursos ON Estudantes.curso_id = Cursos.id_curso;
E) SELECT Estudantes.nome, Cursos.nome FROM Estudantes FULL JOIN Cursos ON Estudantes.id_estudante = Cursos.curso_id;

3. Suponha que você tenha uma tabela chamada 'Pedidos' que registra todas as vendas de uma loja, com as colunas 'id_pedido', 'id_cliente', 'valor_total', e 'data_pedido'. Você deseja criar uma consulta SQL para retornar o total de vendas por cliente, apenas para os clientes que realizaram pedidos após 1º de janeiro de 2022. Qual comando SQL realiza essa tarefa corretamente?

A) SELECT id_cliente, SUM(valor_total) FROM Pedidos GROUP BY id_cliente HAVING data_pedido > '2022-01-01';
B) SELECT id_cliente, SUM(valor_total) FROM Pedidos WHERE data_pedido > '2022-01-01' GROUP BY id_cliente;
C) SELECT id_cliente, SUM(valor_total) FROM Pedidos WHERE data_pedido > '2022-01-01'

GROUP BY data_pedido;
D) SELECT id_cliente, COUNT(valor_total) FROM Pedidos WHERE data_pedido > '2022-01-01' GROUP BY id_cliente;
E) SELECT id_cliente, SUM(valor_total) FROM Pedidos WHERE data_pedido < '2022-01-01' GROUP BY id_cliente;

4. Complete o comando SQL para criar uma nova tabela chamada 'Vendas' que inclui as colunas 'id_venda' como chave primária, 'id_cliente' como chave estrangeira que referencia a tabela 'Clientes', e 'valor_total' como um número decimal com duas casas decimais:

CREATE TABLE Vendas (
_____ INT PRIMARY KEY,
_____ INT,
_____ DECIMAL(10,2)
FOREIGN KEY (id_cliente) REFERENCES Clientes(id_cliente)
);

A) id_venda, cliente_id, total
B) venda_id, cliente_id, valor
C) id_venda, id_cliente, valor_total
D) id_venda, id_cliente, valor
E) id_venda, id_cliente, valor_bruto

5. Qual comando SQL é usado para conceder permissões específicas a um usuário ou grupo?

A) PERMIT
B) ALLOW
C) ENABLE
D) AUTHORIZE
E) GRANT

6. Em SQL, qual operador é utilizado para combinar os resultados de duas ou mais consultas SELECT e inclui todas as linhas duplicadas?

A) UNION ALL
B) INTERSECT
C) EXCEPT
D) UNION
E) JOIN

7. Em um banco de dados, qual é a função da cláusula GROUP BY em uma consulta SQL?

A) Excluir grupos de dados repetidos
B) Ordenar o resultado da consulta
C) Agrupar linhas que têm os mesmos valores em colunas especificadas
D) Conceder permissões de grupo
E) Limitar o número de grupos retornados

8. Suponha que você precisa escrever uma consulta SQL para retornar todos os clientes que não têm pedidos registrados na tabela 'Pedidos'. As tabelas são 'Clientes' e 'Pedidos', onde a tabela 'Pedidos' tem uma coluna 'cliente_id' que corresponde à coluna 'id' na tabela 'Clientes'. Qual comando SQL realiza essa tarefa corretamente?

A) SELECT Clientes.nome FROM Clientes FULL JOIN Pedidos ON Clientes.id = Pedidos.cliente_id WHERE Pedidos.id IS NULL;
B) SELECT Clientes.nome FROM Clientes RIGHT JOIN Pedidos ON Clientes.id = Pedidos.cliente_id WHERE Pedidos.id IS NULL;
C) SELECT Clientes.nome FROM Clientes INNER JOIN Pedidos ON Clientes.id = Pedidos.cliente_id WHERE Pedidos.id IS NULL;
D) SELECT Clientes.nome FROM Clientes LEFT JOIN Pedidos ON Clientes.id = Pedidos.cliente_id WHERE Pedidos.id IS NULL;
E) SELECT Clientes.nome FROM Clientes JOIN Pedidos ON Clientes.id = Pedidos.cliente_id WHERE Pedidos.id IS NULL;

9. Qual comando SQL é usado para retornar registros que existem em ambas as consultas SELECT?

A) UNION
B) INTERSECT
C) EXCEPT
D) INNER JOIN
E) FULL JOIN

10. Em uma instrução SQL, qual cláusula é usada para ordenar o conjunto de resultados de acordo com uma ou mais colunas?

A) GROUP BY
B) SORT BY
C) ORGANIZE BY
D) FILTER BY
E) ORDER BY

GABARITOS E COMENTÁRIOS

1. Qual operador SQL é usado para buscar um padrão em uma string?

A) IN
B) LIKE
C) CONTAINS
D) EQUALS
E) MATCHES

Gabarito: B

Comentários: O operador LIKE é usado em SQL para buscar um padrão em uma string. Ele pode ser utilizado com curingas, como '%' para representar qualquer sequência de caracteres, e '_' para representar um único caractere. IN é usado para especificar múltiplos valores em uma cláusula WHERE, enquanto CONTAINS, EQUALS e MATCHES não são operadores padrão para busca de padrão em strings em SQL.

2. Suponha que você esteja trabalhando com um banco de dados de uma universidade que possui uma tabela chamada 'Estudantes' e outra chamada 'Cursos'. Cada registro na tabela 'Estudantes' possui um identificador único de estudante (id_estudante) e cada registro na tabela 'Cursos' possui um identificador de curso (id_curso). Você deseja criar uma consulta SQL que retorne o nome de cada estudante junto com o nome do curso em que ele está matriculado. Qual comando SQL realiza essa tarefa corretamente, considerando que a tabela 'Estudantes' tem uma coluna 'curso_id' que corresponde à coluna 'id_curso' na tabela 'Cursos'?

A) SELECT Estudantes.nome, Cursos.nome FROM Estudantes INNER JOIN Cursos ON Estudantes.id_estudante = Cursos.curso_id;
B) SELECT Estudantes.nome, Cursos.nome FROM Estudantes LEFT JOIN Cursos ON Estudantes.id_estudante = Cursos.id_curso;
C) SELECT Estudantes.nome, Cursos.nome FROM Estudantes INNER JOIN Cursos ON Estudantes.curso_id = Cursos.id_curso;
D) SELECT Estudantes.nome, Cursos.nome FROM Estudantes RIGHT JOIN Cursos ON Estudantes.curso_id = Cursos.id_curso;
E) SELECT Estudantes.nome, Cursos.nome FROM Estudantes FULL JOIN Cursos ON Estudantes.id_estudante = Cursos.curso_id;

Gabarito: C

Comentários: A opção correta é B. A consulta SQL correta usa uma junção interna (INNER JOIN) para combinar as tabelas 'Estudantes' e 'Cursos' com base na correspondência entre a coluna 'curso_id' na tabela 'Estudantes' e a coluna 'id_curso' na tabela 'Cursos', retornando o nome do estudante e o nome do curso em que ele está matriculado.

3. Suponha que você tenha uma tabela chamada 'Pedidos' que registra todas as vendas de uma loja, com as colunas 'id_pedido', 'id_cliente', 'valor_total', e 'data_pedido'. Você deseja criar uma consulta SQL para retornar o total de vendas por cliente, apenas para os clientes que realizaram pedidos após 1º de janeiro de 2022. Qual comando SQL realiza essa tarefa corretamente?

A) SELECT id_cliente, SUM(valor_total) FROM Pedidos GROUP BY id_cliente HAVING data_pedido > '2022-01-01';
B) SELECT id_cliente, SUM(valor_total) FROM Pedidos WHERE data_pedido > '2022-01-01' GROUP BY id_cliente;
C) SELECT id_cliente, SUM(valor_total) FROM Pedidos WHERE data_pedido > '2022-01-01' GROUP BY data_pedido;
D) SELECT id_cliente, COUNT(valor_total) FROM Pedidos WHERE data_pedido > '2022-01-01' GROUP BY id_cliente;
E) SELECT id_cliente, SUM(valor_total) FROM Pedidos WHERE data_pedido < '2022-01-01' GROUP BY id_cliente;

Gabarito: B

Comentários: A opção correta é A. O comando SQL utiliza a cláusula WHERE para filtrar os pedidos realizados após 1º de janeiro de 2022 e a cláusula GROUP BY para agrupar os resultados por cliente, somando o valor total das vendas para cada cliente.

4. Complete o comando SQL para criar uma nova tabela chamada 'Vendas' que inclui as colunas 'id_venda' como chave primária, 'id_cliente' como chave estrangeira que referencia a tabela 'Clientes', e 'valor_total' como um número decimal com duas casas decimais:

```
CREATE TABLE Vendas (
_____ INT PRIMARY KEY,
_____ INT,
_____ DECIMAL(10,2)
FOREIGN KEY (id_cliente) REFERENCES Clientes(id_cliente)
);
```

A) id_venda, cliente_id, total
B) venda_id, cliente_id, valor
C) id_venda, id_cliente, valor_total
D) id_venda, id_cliente, valor
E) id_venda, id_cliente, valor_bruto

Gabarito: C

Comentários: A resposta correta é: CREATE TABLE Vendas (id_venda INT PRIMARY KEY, id_cliente INT, valor_total DECIMAL(10,2) FOREIGN KEY (id_cliente) REFERENCES Clientes(id_cliente)). As colunas devem ser nomeadas de acordo com as instruções para que a criação da tabela seja bem-sucedida.

5. Qual comando SQL é usado para conceder permissões específicas a um usuário ou grupo?

A) PERMIT
B) ALLOW
C) ENABLE
D) AUTHORIZE
E) GRANT

Gabarito: E

Comentários: O comando GRANT é utilizado em SQL para conceder permissões específicas a um usuário ou grupo, permitindo-lhes realizar operações específicas em objetos do banco de dados.

6. Em SQL, qual operador é utilizado para combinar os resultados de duas ou mais consultas SELECT e inclui todas as linhas duplicadas?

A) UNION ALL
B) INTERSECT
C) EXCEPT
D) UNION
E) JOIN

Gabarito: A

Comentários: O operador UNION ALL é utilizado para combinar os resultados de duas ou mais consultas SELECT, incluindo todas as linhas duplicadas no conjunto de resultados.

7. Em um banco de dados, qual é a função da cláusula GROUP BY em uma consulta SQL?

A) Excluir grupos de dados repetidos
B) Ordenar o resultado da consulta
C) Agrupar linhas que têm os mesmos valores em colunas especificadas
D) Conceder permissões de grupo
E) Limitar o número de grupos retornados

Gabarito: C

Comentários: A cláusula GROUP BY em SQL é usada para agrupar linhas que têm os mesmos valores em colunas especificadas. Isso é útil para realizar operações agregadas em cada grupo, como somar, contar, ou calcular médias. Não é usada para excluir grupos de dados, ordenar resultados (essa é a função da cláusula ORDER BY), conceder permissões ou limitar o número de grupos retornados.

8. Suponha que você precisa escrever uma consulta SQL para retornar todos os clientes que não têm pedidos registrados na tabela 'Pedidos'. As tabelas são 'Clientes' e 'Pedidos', onde a tabela 'Pedidos' tem uma coluna 'cliente_id' que corresponde à coluna 'id' na tabela 'Clientes'. Qual comando SQL realiza essa tarefa corretamente?

A) SELECT Clientes.nome FROM Clientes FULL JOIN Pedidos ON Clientes.id = Pedidos.cliente_id WHERE Pedidos.id IS NULL;
B) SELECT Clientes.nome FROM Clientes RIGHT JOIN Pedidos ON Clientes.id = Pedidos.cliente_id WHERE Pedidos.id IS NULL;
C) SELECT Clientes.nome FROM Clientes INNER JOIN Pedidos ON Clientes.id = Pedidos.cliente_id WHERE Pedidos.id IS NULL;
D) SELECT Clientes.nome FROM Clientes LEFT JOIN Pedidos ON Clientes.id = Pedidos.cliente_id WHERE Pedidos.id IS NULL;
E) SELECT Clientes.nome FROM Clientes JOIN Pedidos ON Clientes.id = Pedidos.cliente_id WHERE Pedidos.id IS NULL;

Gabarito: D

Comentários: A opção correta é A. A consulta usa uma junção à esquerda (LEFT JOIN) para combinar as tabelas 'Clientes' e 'Pedidos', e a cláusula WHERE verifica quais registros na tabela 'Pedidos' não têm correspondência, indicando que esses clientes não têm pedidos registrados.

9. Qual comando SQL é usado para retornar registros que existem em ambas as consultas SELECT?

A) UNION
B) INTERSECT
C) EXCEPT
D) INNER JOIN
E) FULL JOIN

Gabarito: B

Comentários: O comando INTERSECT é utilizado para retornar registros que existem em ambas as consultas SELECT, retornando apenas as linhas comuns às duas consultas.

10. Em uma instrução SQL, qual cláusula é usada para ordenar o conjunto de resultados de acordo com uma ou mais colunas?

A) GROUP BY
B) SORT BY
C) ORGANIZE BY
D) FILTER BY
E) ORDER BY

Gabarito: E

Comentários: A cláusula ORDER BY é usada para ordenar o conjunto de resultados de uma consulta SQL de acordo com uma ou mais colunas. Pode ser usada para ordenar em ordem crescente ou decrescente.

1.B	2.C	3.B	4.C	5.E
6.A	7.C	8.D	9.B	10.E

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.