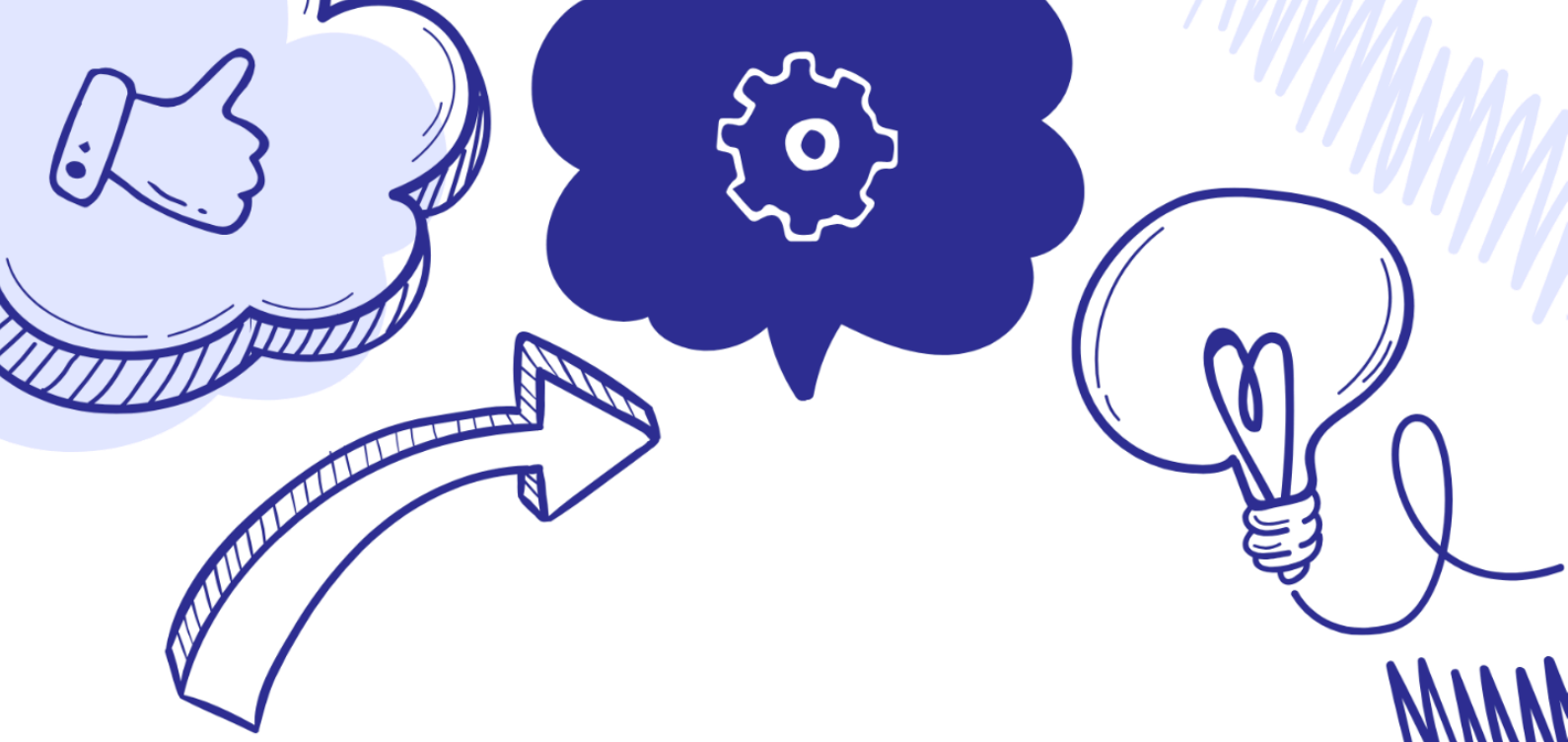


FACULDADE IMPACTA

LINGUAGEM SQL

PROGRAMAÇÃO ORIENTADA A OBJETOS

ALEX SOUSA

SÃO PAULO - 05/2024

SUMÁRIO

SUMÁRIO 2
Linguagem SQL 3
Projetista de banco de dados 3
Projeto Conceitual 3
Projeto Lógico 4
Projeto Físico 4
Sub Linguagens do SQL 4
Artigo - Projeto Físico de Banco de Muito além do CREATE TABLE 4
MER - Modelo Entidade Relacionamento 5
Diagrama Entidade Relacionamento (Diagrama ER ou ainda DER) 5
Tipos de Dados 5
Pontos Principais 8
Data Definition Language (DDL) - Apresentação 8
DML vs. DDL 10
Pontos Principais 10
Data Definition Language DDL - Comandos 10
Pontos Principais 11
Data Manipulation Language (DML) - Apresentação 11
Pontos Principais 12
Data Manipulation Language (DML) - Comandos 12
Pontos Principais 13
Data Query Language (DQL) - Apresentação 14
Data Definition Language (DDL) - Apresentação 19
Data Manipulation Language (DML) - Apresentação 19
Data Query Language (DQL) - Apresentação 19
Cláusulas do Comando SELECT e Ordem de Execução 19

Linguagem SQL

A linguagem SQL foi projetada e implementada no departamento de pesquisa da IBM como a interface para um sistema gerenciador de banco de dados (SGBD) relacional experimental chamado SYSTEM R. Originalmente era chamado de SEQUEL (Structured English Query Language), um acrônimo para linguagem de consulta em inglês estruturado.

Importante: A linguagem SQL é uma linguagem declarativa de alto nível escrita em inglês estruturado que facilita a aprendizagem e uso da linguagem.

Um dos objetivos em seu desenvolvimento era que a linguagem fosse simples de aprender e utilizar. O uso de inglês estruturado na definição da linguagem auxilia a atingir esse objetivo. Por motivos de patentes de nomes não foi possível usar SEQUEL e a linguagem passou a se chamar SQL.

A linguagem SQL se tornou um padrão para banco de dados relacionais, independente de fornecedor, o que facilita a migração de aplicações entre diferentes fornecedores de SGBDs relacionais. A migração é facilitada pois o código SQL escrito seguem os mesmos padrões da linguagem. Além disso podemos escrever programas que acessam mais de um SGBD relacional sem alterar a linguagem de consulta a esses bancos de dados.

A padronização da linguagem SQL é realizado pelos institutos ANSI e ISSO. Os fornecedores de SGBDs relacionais implementam esse padrão e adicionam ao padrão funcionalidades específicas. Ou seja, em um SGBD relacional em particular temos dois conjuntos: as funcionalidades que seguem o padrão estabelecido pelo ANSI/ISO e de funcionalidades particulares desse fornecedor. Quando podemos escolher entre duas funcionalidades em que uma seja padrão e a outra seja particular, uma boa prática é usar a padrão pois facilita uma migração futura e é mais conhecida pelos programadores. O uso de uma particular é feita por motivos de desempenho ou de facilitação de escrita e código, mas deve ser feita de modo consciente.

A linguagem SQL é uma linguagem declarativa de alto nível. Escrevemos o que queremos de resultado e não como obtemos o resultado.

Projetista de banco de dados

Um projetista de banco de dados é um profissional responsável por planejar, projetar e implementar bancos de dados que atendam às necessidades de uma organização ou projeto específico. Ele atua como um arquiteto de dados, definindo a estrutura e a organização dos dados para garantir a eficiência, confiabilidade e segurança das informações.

Projeto Conceitual

O projeto conceitual é o ponto de partida, onde a visão geral do banco de dados é definida. Nessa etapa, o foco está em compreender as necessidades do negócio e dos usuários, identificando as entidades (coisas que queremos representar), seus atributos (características das entidades) e os relacionamentos entre elas.

Ferramentas: Diagramas Entidade-Relacionamento (ER), Linguagem Natural

Objetivo: Descrever o "o quê" do banco de dados, sem se preocupar com "como" será implementado.

Exemplo: Em um sistema de biblioteca, as entidades podem ser "Livros", "Autores" e "Empréstimos". Os atributos de "Livros" podem incluir título, autor, ISBN e gênero. Já os relacionamentos podem indicar que um livro é escrito por um autor e que um livro pode ser emprestado a um usuário.

Projeto Lógico

O projeto lógico leva a modelagem um passo adiante, refinando o modelo conceitual e o traduzindo para uma representação abstrata que não depende de um SGBD específico. Nessa etapa, são definidas as tabelas, colunas, tipos de dados, chaves primárias, chaves estrangeiras e constraints de integridade.

Ferramentas: Diagramas de Classes, Linguagem UML

Objetivo: Descrever o "quê" e o "como" do banco de dados de forma abstrata, sem se preocupar com a implementação física.

Exemplo: No sistema de biblioteca, as tabelas podem ser "Livros", "Autores" e "Empréstimos". As colunas de "Livros" podem incluir "livro_id", "titulo", "autor_id", "ISBN" e "genero". As chaves primárias seriam "livro_id" em "Livros" e "autor_id" em "Autores". Já a chave estrangeira "autor_id" em "Livros" referenciaria a tabela "Autores".

Projeto Físico

Chegamos à etapa final, onde o modelo lógico é adaptado às características específicas do SGBD escolhido. Aqui, são definidas as estruturas de armazenamento, os índices, as partições e outras características físicas que garantem a eficiência e o desempenho do banco de dados.

Ferramentas: Ferramentas do SGBD, Scripts SQL

Objetivo: Descrever o "quê", o "como" e o "onde" do banco de dados, considerando as características do SGBD escolhido.

Exemplo: No SGBD SQL Server, podemos escolher o tipo de dados "varchar(50)" para o título dos livros, criar um índice na coluna "autor_id" da tabela "Livros" para otimizar consultas por autor e configurar o armazenamento em tablespaces específicos para diferentes tipos de dados.

Sub Linguagens do SQL

As instruções para definições de dados são classificadas como pertencentes a sub linguagem DDL (Data Definition Language – Linguagem de Definição dos Dados). Já as instruções de manipulação de dados são classificadas como pertencentes a sub linguagem DML (Data Manipulation Language – Linguagem de Manipulação de Dados) e por fim classificamos as instruções para consulta de dados na sub linguagem DQL (Data Query Language – Linguagem de Consulta de Dados).

Importante: Podemos classificar as instruções da linguagem em sub linguagens, como a linguagem de definição de dados – DDL, a linguagem de manipulação de dados – DML e a linguagem de consulta de dados – DQL;

Artigo - Projeto Físico de Banco de Muito além do CREATE TABLE

https://www.devmedia.com.br/projeto-fisico-de-banco-de-muito-alem-do-create-table/3581

MER - Modelo Entidade Relacionamento

Modelo conceitual utilizado para descrever objetos (entidades) envolvidos em um domínio de negócios, com suas características (atributos) e como elas se relacionam entre si (relacionamentos).

Diagrama Entidade Relacionamento (Diagrama ER ou ainda DER)

Diagrama Entidade Relacionamento DER é a representação gráfica do MER, Em situações práticas, o diagrama é tido muitas vezes como sinônimo de modelo, uma vez que sem uma forma de visualizar as informações, o modelo pode ficar abstrato demais para auxiliar no desenvolvimento do sistema. Dessa forma, quando se está modelando um domínio, o mais comum é criar sua representação gráfica, seguindo algumas regras.

Tipos de Dados

Bancos de dados associam tipos de dados a colunas, expressões, variáveis e parâmetros. Os tipos de dados determinam quais os tipos de valores serão permitidos no armazenamento. Todos os dados são armazenados nos bancos de dados em formato de Bytes. Essa é a forma como os computadores trabalham, ou seja, quando irão armazenar a letra A, na realidade, armazenam o código binário “01000001” que a representa. Quando esse dado precisa ser mostrado, o banco de dados traduz o formato binário gravado, na letra A.

A tabela abaixo mostra a tabela ASCII que é uma das tabelas que traduz números binários em caracteres.

Os tipos de dados podem ser agrupados em categorias:

- Numéricos exatos
    - Temos três sub categorias de tipos numéricos exatos:
    - **inteiro**
        - Existem 4 tipos de dados: tinyint, smallint, int e bigint
    - **decimal**
        - Os tipos de dados decimal e numeric são sinônimos. Para declarar esses tipos de dados podemos fazer de algumas formas:
          DECIMAL – Sem especificar precisão e escala são utilizados valores padrões. O padrão é precisão ter o valor 18 e para a escala é 0. Ou seja, é equivalente a DECIMAL (18) ou a DECIMAL (18,0).
          DECIMAL (4) – Ao declarar dessa forma, a escala utiliza o padrão 0. É equivalente a DECIMAL (4,0).
          DECIMAL (4,2) – Dessa forma temos precisão 4 e escala 2. Informamos tanto a precisão quanto escala. Veja que os números nesse exemplo possuem quatro dígitos no total e duas casas após a vírgula, como em 99,12 e -15,45. Não confundir com 1234,12, que tem 6 dígitos no total e duas casas depois da vírgula. Nesse caso o tipo de dado deve ser pelo menos DECIMAL(6,2).
    - **monetário**
        - money e smallmoney

Tipo de dado	Intervalo	Armazenamento (Bytes)
money	-922.337.203.685.477,5808 a 922.337.203.685.477,50807	8
smallmoney	-214.748,3648 a 214.748,3647	4

- Numéricos aproximados

- Os tipos de dados numéricos aproximados armazenam valores aproximados de números. Tome cuidado para não utilizar esses tipos de dados quando é necessário exatidão nos números, como em valores monetários, por exemplo. Geralmente esses tipos de dados são utilizados quando são provenientes de medições como peso e distância em que o último número da medida já possui uma aproximação pelo próprio aparelho que realizou a medida. Podemos escolher entre os tipos de dados float e real para números aproximados. A tabela abaixo detalha esses dois tipos.

Tipo de dado	Intervalo	Armazenamento (Bytes)
float(n)	$-1{,}79 \times 10^{308}$ a $-2{,}23 \times 10^{-308}$, 0 e $2{,}23 \times 10^{-308}$ a $1{,}79 \times 10^{308}$	Depende do valor de n, 4 ou 8 bytes.
real(n)	$-3{,}40 \times 10^{38}$ a $-1{,}18 \times 10^{-38}$, 0 e $1{,}18 \times 10^{-38}$ a $3{,}40 \times 10^{38}$	4

-

- Data e hora
  - Para armazenarmos data, hora ou data e hora temos alguns tipos de dados, são eles: date, datetime, datetime2, datetimeoffset, smalldatetime e time. Além do intervalo permitido e do armazenamento em bytes temos também a acurácia de cada tipo. A acurácia é a menor unidade de tempo que um tipo de dados de data e hora pode armazenar.

Tipo de dado	Armazenamento (Bytes)	Intervalo de data	Acurária	Formato recomendado
datetime	8	1 de janeiro de 1753 a 31 de dezembro de 9999	3 – 1/3 de milissegundos	'YYMMDD hh:mm:ss:nnn'
smalldatetime	4	1 de janeiro de 1900 a 6 junho de 2079	1 minuto	'YYMMDD hh:mm:ss:nnn'
datetime2	6 a 8	1 de janeiro de 0001 a 31 de dezembro de 9999	100 nano segundos	'YYMMDD hh:mm:ss:nnnnnn'
date	3	1 de janeiro de 0001 a 31 de dezembro de 9999	1 dia	'YY-MM-DD'
time	3 a 5	Não se aplica	100 nano segundos	'hh:mm:ss:nnnnnn'
datetimeoffset	8 a 10	1 de janeiro de 0001 a 31 de desembro de 9999	100 nano segundos	'YY-MM-DD hh:mm:ss:nnnnnnn [+|-] hh:mm'

Tipo de dado	Formatos neutros de linguagem	Exemplos
datetime	'YYYYMMDD hh:mm:ss.nnn' 'YYYY-MM-DDThh:mm:ss.nnn' 'YYYYMMDD'	'20120212 12:30:15.123' '2012-02-12T12:30:15.123' '20120212'
smalldatetime	'YYYYMMDD hh:mm' 'YYYY-MM-DDThh:mm' 'YYYYMMDD'	'20120212 12:30' '2012-02-12T12:30' '20120212'

datetime2	'YYYY-MM-DD'	'20120212 12:30:15.1234567'
	'YYYYMMDD hh:mm:ss.nnnnnnn'	'2012-02-12 12:30:15.1234567'
	'YYYY-MM-DD hh:mm:ss.nnnnnnn'	'2012-02-12T12:30:15.1234567'
	'YYYY-MM-DDThh:mm:ss.nnnnnnn'	'20120212'
	'YYYYMMDD'	'2012-02-12'
	'YYYY-MM-DD'	
date	'YYYYMMDD'	'20120212'
	'YYYY-MM-DD'	'2012-02-12'
time	'hh:mm:ss.nnnnnnn'	'12:30:15.1234567'
datetimeof-fset	'YYYYMMDD hh:mm:ss.nnnnnnn [+|-] hh:mm'	'20120212 12:30:15.1234567 +02:00'
	'YYYY-MM-DD hh:mm:ss.nnnnnnn [+|-] hh:mm'	'2012-02-12 12:30:15.1234567 +02:00'
	'YYYYMMDD'	'20120212'
	'YYYY-MM-DD'	'2012-02-12'

- Cadeias de caracteres (strings)
    - Existem duas subcategorias de tipos de dados categorizados como cadeias de caracteres. Elas são **não-Unicode e Unicode**. Cadeias de caracteres não Unicode utilizam a codificação de caracteres definida na colação do banco de dados. A codificação basicamente traduz uma sequencia de bytes em um caractere específico, como na tabela ASCII mostrada anteriormente. Codificações comuns são UTF-8 e a ISO-8859-1 (conhecida como Latin-1). Você já deve ter entrado em uma página da Internet e visto caracteres estranhos como e no lugar principalmente de caracteres acentuados. Isso provavelmente se deve aos bytes serem armazenados no servidor como ISO-8859-1 e o navegador utilizou outra codificação como UTF-8 para interpretar os bytes. Como as codificações são diferentes um caractere como à foi lido como .
    - **Cadeias de caracteres não Unicode**
        - Os tipos de dados nesta subcategoria são char e varchar. Podemos especificar o tamanho de cada tipo deles como char(10) ou varchar(10). O tipo de dado char(n) armazena sempre n caracteres e cada caracteres ocupa 1 byte. Por exemplo, char(10) sempre irá armazenar 10 caracteres, a cadeia 'Ana' é armazenada como ' Ana' e ocupa exatamente 10 bytes, ou seja, é preenchido com brancos no início da cadeia. Já o tipo de dados varchar(n) armazena o número de caracteres que a cadeia possui até o máximo de n. Por exemplo, varchar(10) irá armazenar no máximo 10 caracteres, ou seja, a cadeia 'Ana' será armazenada como 'Ana', sem preencher os 7 espaços até o máximo de 10 caracteres. No varchar, como a cadeia é variável o SQL precisa de 2 bytes extras para marcar o início e o fim da cadeia.

Tipo de dado	Intervalo	Armazenamento (Bytes)
char(n)	1 a 8000 caracteres	n bytes sempre
varchar(n)	1 a 8000 caracteres	Número de caracteres da cadeia + 2 bytes (Máximo de n + 2 bytes)
varchar(max)	1 a $2^{31} - 1$ caracteres	Número de caracteres da cadeia + 2 bytes

    - **Cadeias de caracteres Unicode**
        - Ao trabalhar com aplicações que podemos armazenar cadeias de caracteres de várias línguas podemos utilizar os tipos de dados dessa subcategoria. Cadeias de caractere Unicode permitem aos bancos de dados representar e manipular de forma **consistente** cadeias de caracteres de qualquer sistema de escrita existente.
        - Atenção: Ao escrevermos uma cadeia de caractere como 'Ana' o SQL interpreta como não Unicode. Para especificar que a cadeia é Unicode temos que prefixar a cadeia com um N (sempre maiúsculo) como N'Ana' (repare que o N está fora das aspas

simples). Sem N no início, o banco irá utilizar a codificação padrão e pode não reconhecer certos caracteres.

Tipo de dado	Intervalo	Armazenamento (Bytes)
nchar(n)	1 a 4000 caracteres	n bytes sempre
nvarchar(n)	1 a 4000 caracteres	(Número de caracteres da cadeia * 2) + 2 bytes (Máximo de n + 2 bytes)
nvarchar(max)	1 a $2^{31} - 1$ caracteres	Número de caracteres da cadeia + 2 bytes

- Binários
    - Os tipos de dados binários armazenam em cada posição dois valores possíveis: 0 ou 1. Por exemplo 0110011. A tabela a seguir detalha os tipos binary e varbinary.

Tipo de dado	Intervalo	Armazenamento (Bytes)
binary(n)	1 a 8000 bytes	n bytes
varbinary(n)	1 a 8000 bytes	n bytes + 2
varbinary(max)	1 a 2,1 bilhões de bytes (aproximadamente)	tamanho atual + 2

- Outros tipos
    - Existem outros tipos de dados como ilustrado na tabela a seguir.

Tipo de dado	Intervalo	Armazenamento (Bytes)	Observações
rowversion	Gerado automaticamente	8	Sucessor do tipo timestamp
uniqueidentifier	Gerado automaticamente	16	Identificador único global (GUID)
xml	0 a 2GB	0 a 2GB	Armazena XML na estrutura hierárquica nativa
cursor	Não aplicável	Não aplicável	Não é um tipo de dado de armazenamento
hierarchyid	Não aplicável	Depende do conteúdo	Representa posição em uma hierarquia
sql_variant	0 a 8000 bytes	Depende do conteúdo	Pode armazenar dados de vários tipos de dados
table	Não aplicável	Não aplicável	Não é um tipo de dado de armazenagem, usado para consulta e operações programáticas

Pontos Principais

- Para armazenar os dados os bancos de dados utilizam tipos de dados para codificar e decodificar corretamente os bytes armazenados no disco;
- A escolha de um tipo de dado deve ser realizada de modo que o dado seja representável nesse tipo e que diminua a quantidade de bytes necessário para armazenamento e manipulação;
- Um tipo de dado define quais operações podemos realizar nos dados, ao usar um tipo numérico podemos realizar operações como soma e subtração. Já ao escolher um tipo que seja uma cadeia de caracteres podemos realizar operações como concatenação e transformar todos os caracteres em maiúsculo.

Data Definition Language (DDL) - Apresentação

Data Definition Language (DDL), ou Linguagem de Definição de Dados, é um subconjunto do SQL usado para **criar, modificar e controlar a estrutura de objetos** em um banco de dados. Esses objetos podem incluir tabelas,

views, índices, usuários e muito mais. Instruções DDL são essenciais para definir o schema do seu banco de dados, que funciona basicamente como um projeto de como seus dados são organizados e armazenados.

Para persistir os dados em um banco de dados, precisamos armazená-los em objetos do tipo tabela. Comandos de criação, alteração ou eliminação de objetos fazem parte da categoria de comandos DDL (Data Definition Language – Linguagem de definição de dados).

- **Criar objetos**: Você pode usar instruções DDL para criar novas tabelas, views, índices, usuários e outros objetos do banco de dados. Isso permite definir a estrutura e os relacionamentos entre seus dados.
- **Modificar objetos**: O DDL também permite modificar a estrutura de objetos existentes. Por exemplo, você pode adicionar ou remover colunas de uma tabela, alterar tipos de dados ou modificar restrições.
- **Controlar acesso**: Instruções DDL podem ser usadas para conceder ou revogar permissões em objetos do banco de dados para diferentes usuários. Isso garante a segurança dos dados e controla quem pode acessá-los e modificá-los.

Aqui estão algumas das instruções DDL mais comuns no SQL:

CREATE: Usada para criar novos objetos de banco de dados, como tabelas, views, índices e usuários.
ALTER: Usada para modificar a estrutura de objetos existentes.
DROP: Usada para excluir ou remover objetos do banco de dados.
GRANT: Usada para conceder permissões a usuários em objetos do banco de dados.
REVOKE: Usada para revogar permissões de usuários em objetos do banco de dados.

Exemplo de uma instrução DDL para criar uma tabela:

```
CREATE TABLE Clientes (
  IDCliente INT PRIMARY KEY,
  NomeCliente VARCHAR(50) NOT NULL,
  Email VARCHAR(100) UNIQUE NOT NULL
);
```

CREATE DATABASE <nome do banco de dado> - cria um banco de dados
USE <nome do banco de dado> - coloca um banco de dados em uso
CREATE TABLE <nome da tabela> - cria tabelas em um banco de dados
GO - executar próxima linha
NOT NULL - define quais campos deve ser de preenchimento obrigatório na tabela
IDENTITY(seed,increment) - função para geração de números automático
exemplo:
IDENTITY ou IDENTITY(1,1) - gera número a partir do 1 e incrementa de 1 em 1.
IDENTITY(0,1) - gera número a partir do 0 e incrementa de 1 em 1.
IDENTITY(0,10) - gera número a partir do 0 e incrementa de 10 em 10. Inicia em 0, 10, 20, ...
PRIMARY KEY
CONSTRAINT pkAluno PRIMARY KEY (Matricula)

FOREIGN KEY
CONSTRAINT fkProva PRIMARY KEY (Matricula)
REFERENCES Aluno(Matrícula)

DML vs. DDL

Ao contrário do DDL, as instruções de Data Manipulation Language (DML) são usadas para interagir com os dados reais armazenados no banco de dados. Instruções DML permitem que você:

INSERT: Inserir novos dados em tabelas.
UPDATE: Modificar dados existentes em tabelas.
DELETE: Remover dados de tabelas.
SELECT: Recuperar dados de tabelas.

Pontos Principais

- CREATE TABLE é o comando que usamos para a criação de tabelas, temos que fornecer um nome para a tabela e nomes de colunas e tipos de dados para o comando executar corretamente;
- Quando precisamos que uma coluna numérica tenha seus valores gerados automaticamente pelo banco de dados usamos IDENTITY. Podemos configurar o número inicial e o incremento de valores.
- A chave primária determina quais colunas determinam que não tenhamos linhas duplicadas em tabelas. Já a restrição de unicidade possui a mesma propriedade que a da chave primária, mas adicionamos as outras chaves da tabela. Chaves estrangeiras são determinadas para relacionarmos as colunas de uma tabela com outra.
- Em relação aos valores podemos determinar que uma coluna tenha preenchimento obrigatório usando a cláusula NOT NULL. Para fornecer um valor padrão em caso de ausência na inserção usamos DEFAULT. Para garantir valores válidos como maior que zero usamos CHECK.

Data Definition Language DDL - Comandos

restrição de dados
Unique - chave única, campo semelhante a chave primária, campo com valores únicos.

alteração de dados
alter table <nome da tabela> add
alter table <nome da tabela> add constraint
alter table <nome da tabela> alter column
alter table <nome da tabela> drop column
alter table <nome da tabela> drop constraint

apagar tabela
drop table <nome da tabela>
drop column <nome da coluna>
drop database <nome da banco>

Note que se uma tabela possui uma chave primária e essa chave participa de relacionamentos com outras tabelas como chave estrangeira, não será permitida a eliminação da tabela que contém a chave primária, ou seja, neste caso, precisamos eliminar as tabelas que mencionam essa chave primária, para depois, conseguirmos eliminar a tabela de chave primária.

valores default
DEFAULT <valor, texto, data, função>

validação dos dados, criação de regras de negócios.
CHECK (AND ou OR)

Pontos Principais

- Para alterar tabelas usamos ALTER TABLE. Na sequência especificamos o tipo de alteração a ser realizado: adição, remoção e alteração de colunas. Também podemos inserir e remover restrições como chave primária, chave estrangeira, unicidade, verificação, não nulo e valor padrão;
- Para remover uma tabela do banco de dados usamos DROP TABLE. A execução desse comando apaga tanto a tabela quanto os dados que estão na tabela. Ao remover várias tabelas, precisamos nos atentar a ordem da remoção. Primeiro removemos as tabelas que contém as chaves estrangeiras para em seguida a tabela com a chave primária referenciada;

Data Manipulation Language (DML) - Apresentação

Linguagem para manipulação de dados dentro da tabela. As cláusulas que tratam a inserção, remoção e eliminação de registros dentro de tabelas são INSERT, UPDATE e DELETE, respectivamente.

Inserir dados na tabela

INSET INTO <tabela>
(lista_colunas)
VALUES
(valores a ser inserido)

Exemplo:
INSERT INTO mytable
(Prikey, description)
VALUES
(1, 'TPX450'),
(2, 'TPX600');

Inserindo dados de uma tabela em outra

Exemplo:
INSERT INTO Mytable (Prikey, description)
    SELECT ForeignKey, description
    FROM SomeView

INSERT usando TOP (números de linhas que eu quero inserir/copiar)

Exemplo:
INSERT TOP (10) INTO Mytable (Prikey, description)
    SELECT ForeignKey, description
    FROM SomeView

https://www.w3schools.com/sql/sql_insert.asp

Pontos Principais

- Para inserir uma ou várias linhas em uma tabela usamos INSERT ... VALUES.;
- Não passamos valores para uma coluna auto numerada (com IDENTITY) pois é o banco de dados que administra os valores dessa coluna;
- Durante a inserção de dados podem ocorrer erros quando os valores da inserção violam restrições definidas na criação da tabela, como chave primária que já existe.

Data Manipulation Language (DML) - Comandos

Comando DELETE removendo linhas (tupla) de uma tabela.

DELETE FROM nome_tabela - Neste caso apagará todos os dados da tabela
DELETE TOP (1) FROM nome_tabela
DELETE TOP (12.5) PERCENT FROM nome_tabela

DELETE utilizando subquery

```
DELETE FROM nome_tabela
WHERE alguma_coluna IN
        (subquery definition);
```

Exemplo:

```
DELETE FROM
sales.salespersonquotaHistory
WHERE salespersonId IN
        (SELECT salespersonId
         FROM sales.salesperson
        WHERE salesYTD > 25000000.00);
```

Truncar tabela, quebrar tabela em parte. Não conseguimos utilizar WHERE

```
TRUNCATE TABLE nome_table
```

Para deletar os dados primeiro aplica se o comando select para verificar se os dados retornados são os que de fato querem ser excluídos.

Exemplo:

```
SELECT name FROM Cliente
WHERE name LIKE 'marcelo%';

DELETE FROM Cliente
WHERE name LIKE 'marcelo%';
```

Comando UPDATE serve para alterar o valor inserido na tabela.

```
UPDATE tabela_ou_visao
```

SET nome_da_coluna = expressao
FROM fontes_tabelas
WHERE Condicao_e_busca;

- SET – lista de colunas, separados por vírgula, que serão alterados;
- FROM – fornece objetos fonte para a cláusula SET;
- WHERE – Especifica a condição de procura para aplicar as alterações com a cláusula SET.

UPDATE sales.salesperson
set bonus = 6000;

UPDATE sales.salesperson
set bonus = 6000 * 2;

UPDATE com a cláusula WHERE

UPDATE alguma_tabela
SET coluna = novo_valor
WHERE valor ou expressão que busca valor que é para ser alterado

UPDATE Vendedor
SET Salario = Salario * 1.1
WHERE Salario < 10000.00;

UPDATE production.product
SET color = N'metallic red'
WHERE name LIKE 'road-250%' AND color = 'Red';

O UPDATE também pode ser utilizado com subquary.

Pontos Principais

- Para remoção de linhas em uma tabela usamos DELETE. Esse comando permite especificar quais linhas serão removidas na cláusula WHERE. Se não especificarmos uma cláusula WHERE todas as linhas da tabela serão removidas;
- A cláusula WHERE recebe uma expressão. Essa expressão deve ser avaliada para verdadeira ou falsa. É aplicada a cada linha da tabela e aquelas que forem avaliadas como verdadeiras serão removidas;
- Usamos TOP(N) para remover as N primeiras linhas da tabela;
- TRUNCATE remove todas as linhas de uma tabela reiniciando propriedades da tabela como IDENTITY. Não é possível especificar a cláusula WHERE nesse comando.
- Para atualizar dados em uma tabela usamos UPADTE. Esse comando além da cláusula WHERE possui a cláusula SET. Nessa cláusula é que especificamos quais colunas terão seu valor atualizado e qual valor devem ter;
- Se não especificarmos a cláusula WHERE todas as linhas da tabela terão seus valores atualizados de acordo com a especificação no SET;
- Podemos utilizar uma sub consulta para determinar quais linhas serão atualizados na cláusula WHERE;
- Podemos usar diversas funções do SQL Server na atualização e na seleção de linhas, como MAX, MIN, AVG, UPPER, etc.

Data Query Language (DQL) - Apresentação

Categoria que envolve a declaração de recuperação de dados.

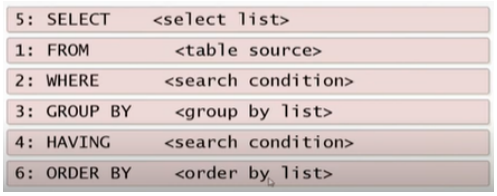
Select é uma declaração sql que retorna um conjunto de resultados registrados de uma ou mais tabelas.

DQL é Sub linguagem de consulta de dados – DQL (Data Query Language) em que o comando principal é o SELECT. O SELECT é uma declaração SQL que retorna um conjunto de resultados de linhas de uma ou mais tabelas. Ele recupera zero ou mais linhas de uma ou mais tabelas-base, tabelas temporárias, funções ou visões em um banco de dados. Também retorna valores únicos de configurações do sistema de banco de dados ou de variáveis de usuários ou do sistema.

Na maioria das aplicações, SELECT é o comando mais utilizado. Como SQL é uma linguagem não procedural, consultas SELECT especificam um conjunto de resultados, mas não especificam como calculá-los, ou seja, a consulta em um plano de consulta é deixada para o sistema de banco de dados, mais especificamente para o otimizador de consulta, escolher a melhor maneira de retorno das informações que foram solicitadas. Ou seja, escrevemos o que queremos que seja devolvido.

SELECT - colunas que quero extrair os dados
FROM tabela de origem
WHERE condição da pesquisa
GROUP BY agrupar a seleção
HAVING condição de agrupamento
ORDER BY ordem da lista retornada

O banco executa as consultas na ordem abaixo.



Formas mais simples da declaração SELECT:

--(*) - Retorna todas as colunas da tabela exemploSQL
SELECT * FROM exemploSQL

SELECT * FROM CLiente;

SELECT Nome, Sobrenome FROM CLiente;

Podemos utilizar operações matemáticas para retornar na consulta.

SELECT preco, qtd, (preco * qtd) FROM detalhesdopedido;

Também podemos ajustar o nome da coluna

SELECT preco, qtd AS Quantidade FROM detalhesdopedido;

Select com Linhas Repetidas.

Ao executar uma consulta como:

```
SELECT pais
FROM Cliente;
```

Um resultado possível de ser obtido é o seguinte:

```
pais
--------------
Argentina
Argentina
Áustria
Áustria
Bélgica
Bélgica
```

Como país é uma coluna que não é chave primária e nem única, várias linhas podem ter o mesmo valor para essa coluna. Então ao realizar a consulta acima é esperado que tenhamos linhas duplicadas. Se for necessário eliminar as linhas repetidas podemos aplicar a cláusula DISTINCT, que retira repetições de linhas para todas as colunas descritas na declaração SELECT:

```
SELECT DISTINCT pais
FROM Cliente;
```

E a saída será:

```
pais
--------------
Argentina
Áustria
Bélgica
```

Select com algumas linhas

```
/* Devolve 10 linhas da tabela exemploSQL */
SELECT top 10 * FROM exemploSQL;

/* Devolve 10% das linhas da tabela exemploSQL */
SELECT top 10 percent * FROM exemploSQL;
```

Selecionando as Linhas a Serem Devolvidas

A cláusula WHERE faz o filtro horizontal em uma consulta, ou seja, permite uma redução do número de linhas que retornarão na consulta. Operadores são utilizados para avaliar uma ou mais expressões que retornam os

valores possíveis: TRUE, FALSE ou UNKNOWN. A devolução de dados se dará em todas as linhas onde a combinação das expressões retornarem TRUE

Operadores de comparação escalar: =, <>, >, >=, <, <=, !=.

Exemplo:
SELECT PrimeiroNome, NomeMeio, UltimoNome
FROM Pessoa
WHERE DataNascimento >= '20040101'

Outros exemplos usando cláusula WHERE simples são dados abaixo:

```
SELECT IdEntidadeNegocio AS 'Número Identificação Empegado',
        DataContratacao,
        HorasDeFerias,
        HorasDoente
FROM RecursosHumanos.Empregado
WHERE IdEntidadeNegocio <= 1000
SELECT
        PrimeiroNome,
        SobreNome,
        Telefone
FROM
        Pessoa.Pessoa
WHERE
        PrimeiroNome = 'Jhon'
```

Podemos usar operadores lógicos para combinar condições na declaração:

```
/* Retorna somente registros onde o primeiro nome for 'John' E o sobrenome for 'Smith' */
WHERE PrimeiroNome = 'John' AND UltimoNome = 'Smith'

/* Retorna todos as linhas onde o primeiro nome for 'John' OU todos onde o sobrenome for 'Smith' */
WHERE PrimeiroNome = 'John' OR UltimoNome = 'Smith'
```

Nem sempre usamos operadores de comparação. Em algumas situações podemos usar outros operadores que são chamados de predicados, simplificando a escrita do código. Alguns exemplos de predicados em SQL são: IN, BETWEEN, ANY, SOME, IS, ALL, OR, AND, NOT e EXISTS.

Por exemplo, se quisermos devolver todas as linhas da tabela Pessoa onde endereço de email não seja nulo, utilizamos o predicado IS NOT NULL (é não nulo):

```
SELECT
        PrimeiroNome,
        SobreNome,
        Telefone
FROM
        Pessoa.Pessoa
WHERE
```

EnderecoEmail IS NOT NULL

O predicado BETWEEN restringe os dados por meio de uma faixa de valores possíveis especificada pelo valor inicial e o valor final. Para devolver todos os pedidos entre as datas de 01 de janeiro de 2011 e 31 de agosto de 2011 podemos escrever:

```
SELECT
        DataPedido,
        NumeroConta,
        SubTotal,
        Impostos
FROM
        Pedidos
WHERE
        DataPedido BETWEEN '20110801' AND '20110831'
```

É equivalente a substituir o BETWEEN pela expressão:
DataPedido >= '20110801' AND DataPedido <= '20110831'

O predicado IN usa uma lista de possibilidades de valores que podem atender a consulta. Se quisermos devolver os pedidos que tenham o valor de IdProduto igual a 750, 753, 765 ou 770 podemos escrever a consulta como:

```
SELECT
        DataPedido,
        NumeroConta,
        SubTotal,
        Impostos
FROM
        Pedidos
WHERE
        IdProduto IN (750, 753, 765, 770)
```

É equivalente a substituir a linha do WHERE usando IN por:
IdProduto = 750 OR IdProduto = 753 OR IdProduto = 765 OR IdProduto = 770

Usando LIKE para colunas de cadeias de caracteres

O predicado LIKE permite realizar consultas mais refinadas em colunas do tipo cadeia de caracteres (char, varchar, ...). Usamos para verificar padrões dentro de campos cadeia de caracteres e utiliza símbolos, chamados de coringas, para permitir a busca desses padrões. Os principais tipos coringa são:

- % (Porcentagem) representa qualquer cadeia de caracteres e qualquer quantidade de caracteres. Exemplo: LIKE 'Carol%' é verdadeira para cadeias como Carolina, Caroline e Carola;
- _ (Underscore) representa qualquer caractere, mas apenas um caractere. Exemplo: LIKE 'Carol_' é verdadeira para cadeias como Carola, mas não Carolina nem Caroline;
- [<List of characters>] representa possíveis caracteres que atendam a cadeia procurada. Exemplo: LIKE 'Carol[ao]' é verdadeira para as cadeias Carola e Carolo, somente.

- [<Character> - <character>] representa a faixa de caracteres, em ordem alfabética, para a string procurada. Exemplo: LIKE ‘Carol[a-e]’ é verdadeira para as cadeias Carola, Carolb, Carolc, Carold e Carole somente.
- [^<Character list or range>] representa o caractere que não queremos na pesquisa. Exemplo: LIKE ‘Carol[^o]’ é verdadeira para todas as cadeias que tenham Carol no início e mais um caractere exceto ‘o’, ou seja, é falso para Carolo e verdadeiro para Carola, Carolb, etc

Utilização do NULL

O NULL representa ausência de valor ou valor desconhecido. Nenhuma das sentenças abaixo é verdadeira porque o banco de dados não pode comparar um valor desconhecido com outro valor que ele também não conhece:

NULL = 0 -- Resultado é desconhecido (Não é verdadeiro!)
NULL = ‘’ (branco ou vazio) -- Resultado é desconhecido (Não é verdadeiro!)
NULL = ‘NULL’ (cadeia NULL) -- Resultado é desconhecido (Não é verdadeiro!)
NULL = NULL -- Resultado é desconhecido (Não é verdadeiro!)

Para trabalhar com valores NULL, temos que utilizar os predicados IS NULL e IS NOT NULL para a lógica da consulta estar correta. Predicados retornam o valor desconhecido quando comparados com valores desconhecidos (valores faltando), ou seja, não são retornados na consulta. Por exemplo:

```
SELECT IdConsumidor, Cidade, Estado, Pais
FROM Vendas.Consumidor
WHERE Estado IS NOT NULL;
```

Ordenação dos Resultados da Consulta

Por padrão ao realizarmos uma consulta não existe garantia da ordem de devolução. Por mais que possa parecer que ao executarmos a consulta repetida vezes o resultado seja ordenado não há essa garantia. Ao adicionarmos um novo índice a uma tabela, por exemplo, a ordem de devolução da consulta pode se alterar. Se precisamos garantir uma ordem específica na devolução de uma consulta, por exemplo, listar os consumidores por estado em ordem alfabética crescente, temos que especificar a cláusula ORDER BY:

```
SELECT IdConsumidor, Cidade, Estado, Pais
FROM Vendas.Consumidor
WHERE Estado IS NOT NULL
ORDER BY Estado; -- Poderíamos adicionar ASC ao final
```

Usamos as cláusula ASC e DESC após cada campo do comando ORDER BY. A ordenação ASCendente é a padrão quando não mencionamos explicitamente. Quando é necessário a ordenação DESCendente, usamos a cláusula DESC:

```
SELECT IdConsumidor, Cidade, Estado, Pais
FROM Vendas.Consumidor
WHERE Estado IS NOT NULL
ORDER BY Estado DESC;
```

Data Definition Language (DDL) - Apresentação

Data Definition Language (DDL), ou Linguagem de Definição de Dados, é um subconjunto do SQL usado para **criar, modificar e controlar a estrutura de objetos** em um banco de dados. Esses objetos podem incluir tabelas, views, índices, usuários e muito mais. Instruções DDL são essenciais para definir o schema do seu banco de dados, que funciona basicamente como um projeto de como seus dados são organizados e armazenados.

Para persistir os dados em um banco de dados, precisamos armazená-los em objetos do tipo tabela. Comandos de criação, alteração ou eliminação de objetos fazem parte da categoria de comandos DDL (Data Definition Language – Linguagem de definição de dados).

Data Manipulation Language (DML) - Apresentação

Linguagem para manipulação de dados dentro da tabela. As cláusulas que tratam a inserção, remoção e eliminação de registros dentro de tabelas são INSERT, UPDATE e DELETE, respectivamente.

Data Query Language (DQL) - Apresentação

Categoria que envolve a declaração de recuperação de dados.

Select é uma declaração sql que retorna um conjunto de resultados registrados de uma ou mais tabelas.

DQL é Sub linguagem de consulta de dados – DQL (Data Query Language) em que o comando principal é o SELECT. O SELECT é uma declaração SQL que retorna um conjunto de resultados de linhas de uma ou mais tabelas. Ele recupera zero ou mais linhas de uma ou mais tabelas-base, tabelas temporárias, funções ou visões em um banco de dados. Também retorna valores únicos de configurações do sistema de banco de dados ou de variáveis de usuários ou do sistema

Cláusulas do Comando SELECT e Ordem de Execução

As cláusulas do comando SELECT são as seguintes:

- SELECT: Define quais as colunas que serão retornadas;
- FROM: Define a(s) tabela(s) envolvidas na consulta;
- WHERE: Filtra as linhas requeridas;
- GROUP BY: Agrupa a lista requerida (utiliza colunas);
- HAVING: Filtra as linhas requeridas, pelo agrupamento;
- ORDER BY: Ordena o retorno da lista.