Banco de Dados

Banco de Dados
Aula 2

Linguagem SQL

Conceito

SQL é um acrônimo para Structured Query Language ou Linguagem de Consulta Estruturada. Consiste em uma linguagem de pesquisa declarativa padrão para acesso a banco de dados relacionais. A base da linguagem está relacionada à álgebra relacional.

Historicamente, a SQL foi desenvolvida pela IBM, nos anos 70, dentro do Projeto R, como parte da pesquisa do F. Codd, que visava demonstrar a viabilidade da implementação do modelo relacional. Originalmente, a linguagem se chamava SEQUEL, acrônimo para “Structured English Query Language” (Linguagem de Consulta Estruturada), e foi desenvolvida para acesso ao System R, sistema de banco de dados construído pela IBM no seu Centro de Pesquisas de Almaden, em San Jose, Califórnia.

Linguagem SQL

Em 1986, o ANSI (American National Standarts Institute) publicou um padrão SQL, denominado SQL-86. Seguiram-se outras publicações posteriores: SQL-89 (padrão estendido), SQL-92 (SQL2), SQL:1999 (SQL3), SQL:2003, SQL:2006, SQL:2008, SQL:2011, SQL:2016 (última versão).

SQL: 2016 ou ISO / IEC 9075: 2016 é a oitava revisão do padrão ISO e ANSI para a linguagem de consulta do banco de dados SQL. Foi formalmente adotado em dezembro de 2016. O padrão consiste em 9 partes, descritas em detalhes no SQL.

SQL: 2016 introduziu 44 novos recursos opcionais. 22 deles pertencem à funcionalidade JSON, mais dez estão relacionados a funções de tabela polimórfica.

Linguagem SQL

As adições ao padrão incluem:

1 - JSON → Funções para criar documentos JSON, acessar partes de documentos JSON e verificar se uma sequência contém dados JSON válidos;
2 - Reconhecimento de padrão de linha → Combinando uma sequência de linhas com um padrão de expressão regular;
3 - Formatação e análise de data e hora;
4 - LISTAGG → Uma função para transformar valores de um grupo de linhas em uma sequência delimitada;
5 - Funções de tabela polimórfica → funções de tabela sem tipo de retorno predefinido;
e
6 - Novo tipo de dados DECFLOAT.

Linguagem SQL

Principais características da Linguagem SQL

Manipulação e controle de bancos de dados relacionais.

O acesso ao banco de dados pode ser feito das seguintes formas:

1 - Através de ambiente interativo de consultas;
2 - Embutida em linguagens hospedeiras (por exemplo, linguagens C e Java).

Recursos:

1 - Alto poder de consulta;
2 - Gerenciamento de índices;
3 - Construção de visões; e
4 - Execução de instruções em blocos.

Linguagem SQL

Grupos de Comandos com SQL

A linguagem SQL pode ser dividida em subconjuntos de comandos conforme as operações que se deseja efetuar em um banco de dados.

DDL - Data Definition Language
Linguagem de Definição de Dados

Comandos para definir, alterar e remover tabelas, visões e índices:

1 – CREATE DATABASE → cria o banco de dados

2 - CREATE TABLE → cria a tabela no banco de dados;

3 – DROP TABLE → exclui a tabela do banco de dados;

4 – TRUNCATE TABLE → exclui informações da tabela, mas não exclui a tabela do banco de dados.

Linguagem SQL

DDL - Data Definition Language
Linguagem de Definição de Dados

5 - ALTER TABLE → altera informações de uma tabela;
6 - ALTER INDEX → altera o índice de uma tabela;
7 - CREATE INDEX → cria um índice na tabela;
8 - DROP INDEX → remove um índice da tabela;
9 - CREATE VIEW → cria uma visão no BD; e
10 - DROP VIEW → remove uma visão do BD.

Linguagem SQL

DML - Data Manipulation Language
Linguagem de Manipulação de Dados

Comandos para inserir, remover, atualizar e consultar os dados armazenados nas tabelas:

1 – INSERT, inserir informações na tabela;

2 – DELETE, excluir informações na tabela; e

3 – UPDATE, alterar informações na tabela.

DQL - Data Query Language
Linguagem de Consulta de Dados

Comando para especificar uma consulta (“query”) como uma descrição do resultado desejado:

1 – SELECT, selecionar informações de uma ou mais tabelas.

Linguagem SQL

DCL - Data Control Language
Linguagem de Controle de Dados

Comandos para trabalhos em ambiente multiusuário, que permitem estabelecer níveis de segurança e manipular transações:

1 – GRANT, autoriza ao usuário executar ou setar operações; e

2 – REVOKE, remove ou restringe a capacidade de um usuário de executar operações.

Linguagem SQL

DTL - Data Transaction Language
Linguagem de Transação de Dados

Comandos para interagir com áreas de controle de transação:

1 - BEGIN WORK - (ou BEGIN TRANSACTION, dependendo do dialeto SQL) - pode ser usado para marcar o começo de uma transação de banco de dados que pode ser completada ou não.

2 – COMMIT, finaliza uma transação dentro de um sistema de gerenciamento de banco de dados ; e

3 – ROLLBACK, faz com que as mudanças nos dados existentes desde o último COMMIT ou ROLLBACK sejam descartadas.

Linguagem SQL

Tipos de Dados Numéricos

1- TINYINT — número inteiro muito pequeno (*tiny*);

2 - SMALLINT — número inteiro pequeno;

3 - MEDIUMINT — número inteiro de tamanho médio;

4 - INT — número inteiro de tamanho comum;

5 - BIGINT — número inteiro de tamanho grande;

6 - DECIMAL — número decimal, de ponto fixo;

7 - FLOAT — número de ponto flutuante de precisão simples (32 bits);

8 - DOUBLE — número de ponto flutuante de precisão dupla (64 bits);

9 - BIT — um campo de um bit.

Linguagem SQL

Tipos de Dados Strings

1 - CHAR — uma cadeia de caracteres (*string*), de tamanho fixo e não-binária;

2 - VARCHAR — uma string de tamanho variável e não-binária;

3 - BINARY — uma string binária de tamanho fixo;

4 - VARBINARY — uma string binária de tamanho variável;

5 - BLOB — um BLOB (*Binary Large OBject* – OBjeto Grande Binário) pequeno;

6 - TINYBLOB — um BLOB muito pequeno;

7 - MEDIUMBLOB — um BLOB de tamanho médio;

8 - LONGBLOB — um BLOB grande;

9 - TINYTEXT — uma string não-binária e de tamanho bem reduzido;

Linguagem SQL

Tipos de Dados Strings

10 - TEXT — uma string não-binária e pequena;

11 - MEDIUMTEXT — uma string de tamanho comum e não-binária;

12 - LONGTEXT — uma string não-binária de tamanho grande;

13 - ENUM — de acordo com o manual do MySQL, é uma string, com um valor que precisa ser selecionado de uma lista predefinida na criação da tabela;

14 - SET — é um objeto que pode ter zero ou mais valores – cada um dos quais precisa ser escolhido de uma lista de valores predeterminados quando da criação da tabela.

Linguagem SQL

Tipos de Dados Data

1 - DATE — o valor referente a uma data no formato 'CCYY-MM-DD'. Por exemplo 1985-11-25 (ano-mês-dia). O 'CC' se refere aos dois dígitos do século (*Century*, em inglês);

2 - TIME — um valor horário no formato 'hh:mm:ss' (hora:minutos:segundos);

3 - TIMESTAMP — *timestamp* é uma sequência de caracteres ou informação codificada que identifica uma marca temporal ou um dado momento em que um evento ocorreu. No MySQL, ele tem o formato 'CCYY-MM-DD hh:mm:ss' – neste caso, seguem a padronização ISO 8601;

4 - YEAR — armazena um ano no formato 'CCYY' ou 'YY';

DDL - Data Definition Language

SQL Constraints

NOT NULL → Indica que uma coluna não pode armazenar valores nulos.
UNIQUE → Garante que uma linha de uma coluna contém valores únicos.
PRIMARY KEY → Garante que uma coluna (ou combinação de duas ou mais colunas) tem uma identidade única, que permite encontrar uma determinada linha da tabela.
FOREIGN KEY → Garante a integridade referencial dos dados em uma tabela para coincidir com os valores da tabela referenciada.
CHECK → Garante que o valor em uma coluna atende a uma condição específica.
DEFAULT → Especifica um valor padrão quando não há valor para a coluna.

DDL - Data Definition Language

Criação de Uma Tabela (CREATE TABLE)

Sintaxe:

```
CREATE TABLE
nome_tabela
(nome_coluna tipo [NOT NULL] [SET DEFAULT valor], ....,
PRIMARY KEY (nome_colunas),
[UNIQUE (nome_coluna)],
[FOREIGN KEY (nome_coluna))
```

DDL - Data Definition Language

Criação de Uma Tabela

Exemplo 1: Criação da tabela disciplina.

```
CREATE TABLE DISCIPLINA
(CodD CHAR(5),
NomeD VARCHAR(20) NOT NULL,
CargaD INTEGER NOT NULL,
AreaD VARCHAR(20),
PreReqD CHAR(5),
UNIQUE (NomeD),
PRIMARY KEY (CodD))
```

Ao criar uma tabela podemos prever que o banco exija que o valor de um campo satisfaça uma expressão.

DDL - Data Definition Language

Exemplo 2: Criação da tabela grade.

```
CREATE TABLE GRADE
(CodC CHAR(5),
CodD CHAR(5),
CodP CHAR(5),
Sala INTEGER,
PRIMARY KEY (CodC, CodD, CodP),
FOREIGN KEY (CodC) REFERENCES CURSO (CodC),
FOREIGN KEY (CodD) REFERENCES DISCIPLINA (CodD),
FOREIGN KEY (CodP) REFERENCES PROFESSOR (CodP))
```

DDL - Data Definition Language

Exemplo 3: Criação da tabela com o uso do UNIQUE.

```
CREATE TABLE produtos
( cod_prod integer UNIQUE,
nome char(80) UNIQUE,
preco numeric );
```

Valores exclusivos para cada campo em todos os registros;

Obs.: nulos não são checados. UNIQUE não aceita valores repetidos, mas aceita vários nulos (já que estes não são checados).

DDL - Data Definition Language

Exemplo 4: Criação da tabela com uso do CHECK.

```
CREATE TABLE produtos
( produto_no integer,
descricao text,
preco numeric
CONSTRAINT preco_positivo CHECK (preco > 0) );

INSERT INTO produtos (produto_no, descricao, preco) values (45, 'qquer', 0);
```

- ERRO: novo registro da relação "produtos" viola restrição de verificação "preco_positivo“

DDL - Data Definition Language

Alteração de Uma Tabela (ALTER TABLE)

Sintaxe

```
ALTER TABLE nome_tabela
[ADD nome_coluna tipo [NOT NULL]]
[DROP COLUMN nome_coluna [CASCADE/RESTRICT]]
[MODIFY nome_coluna]
```

Exemplo 1: Adição da coluna MensC à tabela curso.

```
ALTER TABLE CURSO
ADD MensC NUMERIC(6,2);
```

DDL - Data Definition Language

Exemplo 2: Modificação do tamanho da coluna NomeD da tabela disciplina para varchar(50).

```
ALTER TABLE DISCIPLINA
MODIFY NomeD VARCHAR(50);
```

DDL - Data Definition Language

Remoção de Uma Tabela (DROP TABLE)

Remove a tabela e seus dados.

Sintaxe:

```
DROP TABLE nome_tabela;
```

Exemplo: Remoção da tabela grade.

```
DROP TABLE GRADE;
```

DDL - Data Definition Language

Truncando Uma Tabela (TRUNCATE TABLE)

Remove os dados, mas mantém a estrutura da tabela.

Sintaxe:

```
TRUNCATE TABLE nome_tabela;
```

Exemplo: Truncando a tabela grade.

```
TRUNCATE TABLE GRADE;
```

DDL - Data Definition Language

Criação e Remoção de Índices

Os índices são estruturas de dados que contêm os valores de uma ou mais colunas e são utilizadas para obter mais rapidamente os dados das tabelas. O tema será objeto de uma aula a seguir.

Sintaxe:

```
CREATE INDEX nome_índice
ON nome_tabela ( nome_coluna [ASC|DESC],...);
```

Exemplo: Criação de índice sobre a coluna CidadeP da tabela Professor.

```
CREATE INDEX Xcidadeprof
ON PROFESSOR (CidadeP ASC);
```

DDL - Data Definition Language

Sintaxe:

```
DROP INDEX nome_índice
ON nome_tabela;
```

Exemplo: Remoção de índice sobre a coluna CidadeP da tabela Professor.

```
DROP INDEX Xcidadeprof
ON PROFESSOR;
```

DML - Data Manipulation Language

Conceito

A linguagem de manipulação de dados é usada para modificar registros em um banco de dados. As seguintes tabelas exemplo serão utilizadas.

Inclusão de tuplas/registros em uma tabela

Inserir dados em uma tabela significa preencher uma linha de determinada tabela com dados correspondentes aos tipos determinados naquela tabela. Essa inserção de dados deve seguir as regras de integridade da tabela, assim como respeitar as regras de chave primária e chaves estrangeiras estabelecidas na tabela.

DML - Data Manipulation Language

Sintaxe:

INSERT INTO nome_tabela (coluna1, coluna2, ... , colunaN)
VALUES (valor1, valor2, ... , valorN)

Os valores valor1 , valor2 etc. seguem a ordem dos campos da tabela, sendo utilizado valor vazio (' ') ou a sentença NULL para campos que não necessitem de preenchimento. Dados de tipo numérico podem ser escritos sem a necessidade de aspas simples. Dados do tipo caractere (como char e varchar) devem ser escritos entre aspas simples.

DML - Data Manipulation Language

Exemplo: Inserir uma linha na tabela Professor

Professor (tabela)

CodP	NomeP	CidadeP	TituloP
P1	Joaquim	Rib. Preto	Mestre
P2	Paulo	Batatais	Espec.

```
INSERT INTO Professor (CodP, NomeP, CidadeP, TituloP)
VALUES ('P1', 'Joaquim', 'Rib Preto', 'Mestre')

INSERT INTO Professor (CodP, NomeP, CidadeP, TituloP)
VALUES ('P2', 'Paulo', 'Batatais', 'Espec.')
```

DML - Data Manipulation Language

Atualização dos Dados de uma Tabela

Alterar dados em uma tabela significa atualizar um dado de uma determinada tabela por outro dado do mesmo tipo.
O comando UPDATE pode ser realizado sem o WHERE. Neste caso todas as linhas da tabela serão atualizadas com o valor determinado no comando. Para os casos onde se necessite atualizar apenas linhas que cumpram determinada condição, essa condição é estabelecida com a inclusão do comando WHERE.

DML - Data Manipulation Language

Sintaxe:

UPDATE nome_tabela
SET nome_coluna = valor, ...
WHERE (condição de localização)

Exemplo: Alterar o valor da mensalidade do curso de Ciência da Computação para 650,00.

UPDATE Curso
SET MensC = 650
WHERE NomeC = ‘Ciência Comp’

DML - Data Manipulation Language

Remoção de Dados de uma Tabela

Apagar dados em uma tabela significa eliminar uma ou mais linhas de uma determinada tabela. Para isso utilizamos a instrução DELETE.
O comando Delete pode ser realizado sem o WHERE. Neste caso todas as linhas da tabela determinada serão excluídas. Utilizamos WHERE quando desejamos eliminar os registros que obedeçam a certa condição.

DML - Data Manipulation Language

Sintaxe:

DELETE FROM nome_tabela
WHERE (condição de localização)

Exemplo: Remover da tabela Professor todos os professores que têm título de Doutor.

DELETE FROM Professor
WHERE TituloP = ‘Doutor’

DQL - Data Query Language

SELECT - Operadores

A tabela a seguir mostra os principais operadores utilizados na sintaxe das cláusulas que envolvem condições.
Observe as várias funções em SQL que permitem operar sobre os dados resultantes da consulta.

GRUPO	FUNÇÃO	DESCRIÇÃO
Agregação	AVG (col)	Média dos valores da coluna
	SUM (col)	Soma de valores da coluna
	MAX (col)	Valor máximo da coluna
	MIN (col)	Valor mínimo da coluna
	COUNT	Total de tuplas

DQL - Data Query Language

GRUPO	FUNÇÃO	DESCRIÇÃO
Caracter	UPPER (col)	Converte caracteres minúsculos em maiúsculos
	LOWER (col)	Converte caracteres maiúsculos em minúsculos
	SUBSTR (col, pos, n)	Substring da coluna, iniciando em pos, com n caracteres
Números	ROUND (col/const, n)	Arredondamento em n da coluna (ou da constante)
	TRUNC (col/const, n)	Truncamento em n da coluna (ou da constante)
	ABS (col/constr)	Valor absoluto da coluna ou da constante

DQL - Data Query Language

GRUPO	FUNÇÃO	DESCRIÇÃO
Data/Hora	MONTH (data)	Mês da data
	YEAR (data)	Ano da data
	MINUTE (hora)	Minuto da hora
Conversão	TO_CHAR (num data)	Número (ou data) para caracter
	TO_NUMBER (char)	Minuto da caracter para número
	TO_DATE (char)	Minuto da caracter para data

DQL - Data Query Language

Curso

CodC	NomeC	DuracaoC	MensC
C1	Análise Sist.	4	400
C2	Eng. Mecatrônica	5	600
C3	Ciência Comp.	4	450
C4	Eng. Elétrica	4	600
C5	Turismo	3	350

Disciplina

CodD	NomeD	CargaD	AreaD	PreReqD
D1	TLP1	2	Computação	D2
D2	Cálculo 1	4	Matemática	null
D3	Inglês	2	Humanas	null
D4	Ed. Física	3	Saúde	null
D5	G. Analítica	5	Matemática	D2
D6	Projeto Final	6	null	D1

DQL - Data Query Language

Professor

CodP	NomeP	CidadeP	TituloP
P1	Joaquim	Rib. Preto	Mestre
P2	Paulo	Batatais	Espec.
P3	André	Rib. Preto	Doutor
P4	Gil	S. Carlos	Doutor
P5	Juliana	S. Carlos	Pós Doc

Grade

CodC	CodD	CodP	Sala
C1	D6	P1	305
C2	D2	P2	305
C3	D2	P2	305
C4	D1	P5	201
C4	D3	P3	204
C5	D4	P3	204
C5	D4	P4	207

DQL - Data Query Language

SELECT – consulta simples

O comando **SELECT** deve conter o nome do campo (ou campos) que deve ser retornado e qual tabela (ou tabelas) se referencia.

Exemplo 1: Nomes das disciplinas.

```
SELECT NOMED
FROM DISCIPLINA
```

Resultado

NOMED
TLP1
Cálculo 1
Inglês
Ed Física
G analítica
Projeto Final

DQL - Data Query Language

DISTINCT

A cláusula **DISTINCT** é usada para suprimir linhas duplicadas no resultado.

Exemplo 2: Salas onde as aulas serão ministradas, sem repetição.

```
SELECT DISTINCT SALA
FROM GRADE
```

Resultado

SALA
305
201
204

DQL - Data Query Language

WHERE

A cláusula **WHERE** permite aplicar condições para filtrar as linhas que retornam da consulta. A condição deve respeitar o tipo de dado da coluna. Para colunas com conteúdo do tipo caractere devem ser usadas aspas simples (‘ ‘) no objeto de comparação. Para colunas com tipo numérico basta colocar o valor sem aspas na comparação.

Exemplo 3: Nome e código dos professores de Ribeirão Preto.

```
SELECT NOMEP, CODP
FROM PROFESSOR
WHERE CIDADEP = 'Rib Preto'
```

Resultado

NOME	CODP
Joaquim	P1
André	P3

DQL - Data Query Language

Quando desejamos trazer todos os campos utilizamos o asterisco (*).

Exemplo 4: Todas as colunas da grade do curso C4.

```
SELECT *
FROM GRADE
WHERE CODC = 'C4‘
```

Resultado

CODC	CODD	CODP	SALA
C4	D1	P5	201
C4	D3	P3	204

DQL - Data Query Language

Operações no Comando SELECT

É possível efetuar uma operação aritmética sobre o dado de uma coluna no comando **SELECT**.

Exemplo 5: Nome e duração em meses de cada curso.

```
SELECT NOMEC, (DURACAOC * 12) FROM CURSO
```

Resultado

NOMEC	DURACAOC * 12
Análise Sist	48
Eng Mecatrônica	60
Ciência Comp	60
Eng Elétrica	60
Turismo	36

DQL - Data Query Language

Operadores Lógicos

Critérios combinados podem ser especificados utilizando operadores lógicos **AND/OR**.

Exemplo 6: Código e carga horária das disciplinas da área de Matemática, com carga horária maior ou igual a 5.

```
SELECT CODD, CARGAD
FROM DISCIPLINA
WHERE AREAD = 'Matemática'
AND CARGAD >= 5
```

Resultado

CODD	CARGAD
D5	5

DQL - Data Query Language

Operador LIKE

Utilizado em consultas em atributos do tipo caractere para filtrar conteúdos onde ocorrem sequências de strings. O caractere porcentagem (%) indica a posição em que o conteúdo será procurado e o caractere sublinhado (_) indica o número de caracteres envolvidos na pesquisa. Veja os exemplos:

Expressão	Resultado
LIKE `Maria%`	Qualquer string que inicia com Maria
LIKE `%Pedro`	Qualquer string que termina com Pedro
LIKE `%Paulo%`	Qualquer string que tenha Paulo em qualquer posição
LIKE `Z_`	String de dois caracteres onde o primeiro caractere seja Z e o segundo caractere qualquer outro
LIKE `_Z_`	String de três caracteres onde a segunda letra é Z

DQL - Data Query Language

Exemplo 7: O código e o nome de todos os cursos de engenharia.

```
SELECT CODC, NOMEC
FROM CURSO
WHERE NOMEC LIKE 'Eng%'
```

Resultado

CODC	NOMEC
C2	Eng Mecatrônica
C4	Eng Elétrica

DQL - Data Query Language

Exemplo 8: As salas do segundo andar (número fica na casa dos 200) onde serão ministradas aulas.

```
SELECT DISTINCT SALA
FROM GRADE
WHERE TO_CHAR(SALA) LIKE '2__‘
```

Resultado

SALA
201
204
207

DQL - Data Query Language

Operador BETWEEN

Utilizado quando é necessário recuperar linhas entre valores de um intervalo.
Os valores contidos no comando fazem parte do intervalo.

Exemplo 9: Código dos cursos cuja mensalidade está entre 400 e 550 reais.

```
SELECT CODC
FROM CURSO
WHERE MENSC BETWEEN 400 AND 550
```

Resultado

CODC
C1
C3

DQL - Data Query Language

Operador IN

Utilizado quando para recuperar linhas onde os valores a serem comprados estão em uma lista.

Exemplo 10: Nome das disciplinas que são da área de Computação ou de Humanas ou de Saúde.

```
SELECT NOMED
FROM DISCIPLINA
WHERE AREAD IN ('Computação', 'Humanas', 'Saúde')
```

Resultado

NOMED
TLP1
Inglês
Ed Física

DQL - Data Query Language

Operador NULL/NOT NULL

Verifica se o valor de uma coluna é nulo ou não (**IS NULL/IS NOT NULL**).

Exemplo 11: Nome das disciplinas que não pertencem a nenhuma área específica.

```
SELECT NOMED
FROM DISCIPLINA
WHERE AREAD IS NULL
```

Resultado

NOMED
Projeto Final

DQL - Data Query Language

ALIAS

O **alias** é usado para substituir nomes na consulta. Existem dois tipos de **alias**:

1 - de coluna; e

2 - de tabela.

O **alias de coluna** é aplicado na lista de colunas do comando **SELECT** através da cláusula AS e usado para alterar o nome da coluna na apresentação do resultado da consulta.

O **alias de tabela** é usado na cláusula **FROM** após o nome da tabela e serve para substituir o nome da tabela dentro da consulta. Muito utilizado no comando de junção que será visto mais adiante.

DQL - Data Query Language

Exemplo 12: Nome e duração em meses de cada curso.

```
SELECT NOMEC AS CURSO,
(DURACAOC * 12) AS DURACAO_MESES
FROM Curso
```

Resultado

CURSO	DURACAO_MESES
Análise Sist	48
Eng Mecatrônica	60
Ciência Comp	60
Eng Elétrica	60
Turismo	36

DQL - Data Query Language

Concatenação de Campos

Utilizamos concatenação de campos quando é necessário combinar vários campos diferentes em uma coluna de saída da consulta.

Exemplo 12: Códigos dos cursos com seu nome e mensalidade concatenados.

```
SELECT CODC, NOMEC || ' ' || MENSC || ' reais' AS INFO_CURSO
FROM CURSO
```

Resultado

CODC	INFO_CURSO
C1	Análise Sist 400 reais
C2	Eng Mecatrônica 600 reais
C3	Ciência Comp 450 reais
C4	Eng Elétrica 600 reais
C5	Turismo 350 reais

DQL - Data Query Language

Ordenação do Resultado

Para ordenar o resultado da consulta utilizamos a cláusula **ORDER BY**. É possível ordenar o resultado em ordem crescente ou decrescente utilizando a cláusula ASC ou DESC.

Exemplo 13: Nomes dos cursos ordenados de forma ascendente.

```
SELECT NOMEC
FROM CURSO
ORDER BY NOMEC ASC
```

Resultado

NOMEC
Análise Sist
Ciência Comp
Eng Elétrica
Eng Mecatrônica
Turismo

DQL - Data Query Language

Também é possível utilizar a posição da coluna para indicar a que coluna será aplicada a cláusula **ORDER BY**.

Exemplo 14: Código e nomes dos cursos ordenados por nome de forma decrescente.

```
SELECT CODC, NOMEC
FROM CURSO
ORDER BY 2 DESC
```

Resultado

CODC	NOMEC
C5	Turismo
C4	Eng Elétrica
C2	Eng Mecatrônica
C3	Ciência Comp
C1	Análise Sist

Linguagem SQL – Material Complementar

Stored Procedure

A Stored Procedure, que traduzido, significa Procedimento Armazenado, é uma conjunto de comandos em **SQL** que podem ser executados de uma só vez, como em uma função. Ele armazena tarefas repetitivas e aceita parâmetros de entrada para que a tarefa seja efetuada de acordo com a necessidade individual.

Um Stored Procedure pode reduzir o tráfego na rede, melhorar a performance de um banco de dados, criar tarefas agendadas, diminuir riscos, criar rotinas de processsamento.

Linguagem SQL

Quando utilizar procedures?

Quando temos várias aplicações escritas em diferentes linguagens, ou rodam em plataformas diferentes, porém executam a mesma função.
Quando damos prioridade à consistência e segurança.

Por que é mais seguro?

Seguindo a linha de raciocínio dos bancos, **utilizando stored procedures outras aplicações e usuários não conseguiriam nenhum tipo de acesso às tabelas do banco de dados** de forma direta.

Eles poderiam apenas executar os stored procedures, que rodam ações específicas e determinadas pelos DBAs e desenvolvedores.

Linguagem SQL

View

É uma maneira alternativa de observação de dados de uma ou mais **entidades** (tabelas), que compõem uma base de dados. Pode ser considerada como uma tabela virtual ou uma consulta armazenada.

É recomendável que uma view seja implementada encapsulando uma instrução **SELECT** (busca de dados para exposição), guarda os dados em uma tabela virtual, armazenando também em cache, pois todas as consultas ao banco, encapsuladas ou não, ao serem executadas, são armazenadas em cache. Por este motivo, pode ser mais rápido ter uma consulta armazenada em forma de view, em vez de ter que retrabalhar uma instrução.

Linguagem SQL

Exemplo:

```
USE NOME_DO_BANCO
GO
CREATE VIEW dbo.viewAniversariante
(nome, sobrenome, data_nascimento)
AS
SELECT nome, sobrenome, data_nascimento)
FROM usuario
GO
```

Dando tudo certo com as consistências, execute a view da seguinte maneira:

```
SELECT * FROM viewAniversariante
GO
```

Linguagem SQL

As views nos possibilitam mais que simplesmente visualizar dados. Elas podem ser implementadas também com algumas aplicações de restrição:

1 - Restrição usuário x dados:

Exemplo: Seu departamento de vendas não precisa saber ou ter acesso a uma coluna que contém valores (dados) referentes aos salários dos desenvolvedores.

2 - Restrição usuário x domínio:

Exemplo: Podemos restringir o acesso de um usuário específico a colunas (domínios) específicas (os) de uma tabela.

Linguagem SQL

3 - Associar vários domínios formando uma única entidade:

Exemplo: Podemos ter várias "JOIN" encapsuladas em uma view, formando somente uma tabela arbitrariamente.

4 - Agregar informações, em vez de fornecer detalhes:

Exemplo: Podemos apresentar um somatório de despesas em ligações de um determinado usuário, restringindo acesso aos detalhes da conta.

Linguagem SQL

As vantagens de se usar views

1 - Economizar tempo com retrabalho:

Exemplo: Você não precisar escrever aquela instrução enorme. Escreva uma vez e armazene!

2 - Velocidade de acesso às informações:

Exemplo: Uma vez compilada, o seu recordset (conjunto de dados) é armazenado em uma tabela temporária (virtual).

3 - Mascarar complexidade do banco de dados:

Exemplo: isolam do usuário a complexidade do banco de dados. Nomes de domínios podem ser referenciados com literais e outros recursos. Isso proporciona aos desenvolvedores a capacidade de alterar a estrutura sem afetar a interação do usuário com o banco de dados.

Linguagem SQL

Triggers

Um TRIGGER ou gatilho é um objeto de banco de dados, associado a uma tabela, definido para ser disparado, respondendo a um evento em particular. Tais eventos são os comandos da DML (Data Manipulation Language):

1 - INSERT;
2 - DELETE; e
3 - UPDATE.

Podemos definir inúmeros TRIGGERS em uma base de dados baseados diretamente em qual dos comandos acima irá dispará-lo, sendo que, para cada um, podemos definir apenas um TRIGGER. Os TRIGGERS poderão ser disparados para trabalharem antes ou depois do evento.

Linguagem SQL

Triggers

Um trigger é um tipo especial de procedimento armazenado, que é executado sempre que há uma tentativa de modificar os dados de uma tabela que é protegida por ele. Por isso temos:

1 - Associados a uma tabela → os TRIGGERS são definidos em uma tabela específica, que é denominada tabela de TRIGGERS;

2 - Chamados Automaticamente → quando há uma tentativa de inserir, atualizar ou excluir os dados em uma tabela, e um TRIGGER tiver sido definido na tabela para essa ação específica, ele será executado automaticamente, não podendo nunca ser ignorado.

Linguagem SQL

3 - Não podem ser chamados diretamente → ao contrário dos procedimentos armazenados do sistema, os disparadores não podem ser chamados diretamente e não passam nem aceitam parâmetros.

4 - É parte de uma transação → o TRIGGER e a instrução que o aciona são tratados como uma única transação, que poderá ser revertida em qualquer ponto do procedimento, caso você queria usar “ROLLBACK”, conceitos que veremos mais a frente.

Linguagem SQL

Usos e Aplicabilidade dos TRIGGERS

1 - Impor uma integridade de dados mais complexa do que uma restrição CHECK;

2 - Definir mensagens de erro personalizadas;

3 - Manter dados desnormalizados;

4 - Comparar a consistência dos dados – posterior e anterior – de uma instrução UPDATE;

Os TRIGGERS são usados com enorme eficiência para impor e manter integridade referencial de baixo nível, e não para retornar resultados de consultas. A principal vantagem é que eles podem conter uma lógica de processamento complexa.

Linguagem SQL

Os TRIGGERS são usados com enorme eficiência para impor e manter integridade referencial de baixo nível, e não para retornar resultados de consultas. A principal vantagem é que eles podem conter uma lógica de processamento complexa.

Você pode usar TRIGGERS para atualizações e exclusões em cascata através de tabelas relacionadas em um banco de dados, impor integridades mais complexas do que uma restrição CHECK, definir mensagens de erro personalizadas, manter dados desnormalizados e fazer comparações dos momentos anteriores e posteriores a uma transação.

Linguagem SQL

Como Funcionam os TRIGGERS

Quando incluímos, excluímos ao alteramos algum registro em um banco de dados, são criadas tabelas temporárias que passam a conter os registros excluídos, inseridos e também o antes e depois de uma atualização.

Quando você exclui um determinado registro de uma tabela, na verdade você estará apagando a referência desse registro, que ficará, após o **DELETE**, numa tabela temporária de nome DELETED. Um TRIGGER implementado com uma instrução SELECT poderá lhe trazer todos ou um número de registro que foram excluídos.

Assim como acontece com DELETE, também ocorrerá com inserções em tabelas, podendo obter os dados ou o número de linhas afetadas buscando na tabela INSERTED.

Linguagem SQL

Já no **UPDATE** ou atualização de registros em uma tabela, temos uma variação e uma concatenação para verificar o antes e o depois.

De fato, quando executamos uma instrução UPDATE, a “engine” de qualquer banco de dados tem um trabalho semelhante, primeiro exclui os dados tupla e posteriormente faz a inserção do novo registro que ocupará aquela posição na tabela, ou seja, um DELETE seguido por um INSERT. Quando então, há uma atualização, podemos buscar o antes e o depois, pois o antes estará na tabela DELETED e o depois estará na tabela INSERTED.

Linguagem SQL

Opções de eventos disponíveis para uso das triggers:

1 - Before insert (Antes de inserir);
2 - After insert (Depois de inserir);
3 - Before update (Antes de atualizar);
4 - After update (Depois de atualizar);
5 - Before delete (Antes de apagar); e
6 - After delete (Depois de apagar).

Os registros NEW e OLD

Como os *triggers*, são executados em conjunto com operações de inclusão e exclusão, é necessário poder acessar os registros que estão sendo incluídos ou removidos. Isso pode ser feito através das palavras NEW e OLD.

Em gatilhos executados após a inserção de registros, a palavra reservada NEW dá acesso ao novo registro. Pode-se acessar as colunas da tabela como atributo do registro NEW, como veremos nos exemplos.

O operador OLD funciona de forma semelhante, porém em gatilhos que são executados com a exclusão de dados, o OLD dá acesso ao registro que está sendo removido.