***SQL VISÃO GERAL***

SQL – STRUCTURED QUERY LANGUAGE

SQL É UMA LINGUAGEM DE PAGINAÇÃO USADA PARA ARMAZENAR, MANIPULAR E RECUPERAR DADOS ARMAZENADOS EM BANCO DE DADOS RELACIONAL.

SQL É A LINGUAGEM PADRÃO PARA BANCOS DE DADOS RELACIONAIS.

***HISTÓRIA***

1970 – Dr E.F. “TED” DA IBM

DESCREVEU O MODELO RELACIONAL PARA BANCOS DE DADOS

1974 – SURGIU A SQL

1978 – IBM DESENVOLVEU E LANÇOU UM PRODUTO CHAMADO SYSTEM / R

1986 – IBM DESENVOLVEU O PRIMEIRO PROTOTIPO DE BANCO RELACIONAL E PADRONIZOU PELA ANSI

***ARQUITETURA SQL***

BD FÍSICO

PARSER + OTIMIZADOR

GERENCIADOR DE TRANSAÇÕES

ENGINE SGBD

QUERY LANGUAGE PROCESSOR

QUERY SQL

***COMANDO SQL***

* **DDL – DATA DEFINITION LANGUAGE**
  + CREATE – CRIA TABELAS VIEWS OU OUTROS OBJETOS NO BD
  + ALTER – MODIFICA UM OBJETO EXISTENTE NO BD. e.g: TABELAS
  + DROP – DELETA UMA TABELA INTEIRA, VIEW OU OBJETO DO BD
* **DML – DATA MANIPULATION LANGUAGE**
  + INSERT – INSERE UM REGISTRO
  + DELEETE – DELETA UM REGISTRO
  + UPDATE – MODIFICA UM REGISTRO
* **DCL – DATA CONTROL LANGUAGE**
  + GRANT – DÁ PERMISSÃO A UM USUARIO
  + REVOKE – RETIRA PERMISSÃO DE UM USUARIO
* **DQL – DATA QUERY LANGUAGE**
  + SELECT – RECUPERA REGISTROS DE UMA OU MAIS TABELAS

***CONCEITOS DE SGBD***

* SGBD – SISTEMA GERENCIADOR DE BANCO DE DADOS

1. **TABELAS**
   1. OS DADOS NOS SGBDs SÃO ARMAZENADOS EM TABELAS
   2. TABELAS SÃO COLEÇÕES DE DADOS RELACIONAIS
   3. SÃO COMPOSTAS POR LINHAS E COLUNAS

|  |  |  |  |
| --- | --- | --- | --- |
| ID | NOME | IDADE | SALARIOS |
| 1 | PEDRO | 32 | 2000 |
| 2 | ZÉ | 25 | 1500 |
| 3 | ANA | 27 | 6500 |

* 1. TODA TABELA É COMPOSTA POR ENTIDADES MENORES CHAMADAS CAMPOS
  2. UM CAMPO É UMA COLUNA DESIGNADA PARA MANTER UMA INFORMAÇÃO DE UM DETERMINADO TIPO

1. **REGISTROS**
   1. UM REGISTRO É UMA DAS LINHAS DA TABELA
   2. CADA LINHA REPRESENTA UM INDIVIDUO ÚNICO
2. **ATRIBUTOS**
   1. CADA COLUNA REPRESENTA TODA INFORMAÇÃO RELACIONADA COM UM DETERMINADO CAMPO
3. **VALOR NULL**
   1. É UM VALOR NA TABELA QUE INDICA QUE AQUELE CAPO PARECE ESTAR VAZIO
   2. OU SEJA, NULL É UM VALOR QUE INDICA QUE O CAMPO NÃO TEM VALOR

**AULA 2**

Atributos

***SGBD***

Tabela Cliente

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | NOME | IDADE | SALARIO | CIDADE |
| 1 | Ana | 22 | 1600 | Curitiba |
| 2 | Pedro | 25 | 1500 | Porto Alegre |
| 3 | Jose | 30 | 3000 | Curitiba |
| 4 | Julia | 28 | 3000 | São Paulo |

**Restrições** – são regras aplicadas nas colunas de dados na tabela. São usadas para limitar os tipos de dados que podem ser armazenados na tabela.

Garantem a precisão e confiabilidade dos dados.

As restrições mais comuns em SQL são:

* NOT NULL: Garantem que a coluna não vai permitir valores nulos;
* DEFAULT: Definem um valor padrão para a coluna caso nenhum seja fornecido;
* UNIQUE: Garantem que os valores na coluna devem ser diferentes;
* PRIMARY KEY: Identifica unicamente as linhas/registros de uma tabela;
* FOREIGN KEY: Identifica unicamente as linhas/registros em outra tabela;
* CHECK: Garantem que todos os valores de uma coluna satisfazem uma condição;
* INDEX: Usado para recuperar dados rapidamente do BD.

Exemplos feitos no SQL FIDDLE

CREATE TABLE cliente

(

id INT NOT NULL,

nome VARCHAR(50) NOT NULL,

idade INT NOT NULL CHECK (idade >= 18),

salario DECIMAL(10, 2)DEFAULT 5000,

cidade VARCHAR(50) UNIQUE,

PRIMARY KEY (id)

);

INSERT INTO cliente VALUES

(

1,

'Ana',

20,

2000,

'Curitiba'

);

INSERT INTO cliente(id, nome, idade) VALUES

(

2,

'Pedro',

25

);

INSERT INTO cliente VALUES

(

3,

'José',

17,

NULL,

'Curitibo'

);

-- DROP TABLE cliente;

CREATE TABLE funcionarios

(

id int NOT NULL,

PRIMARY KEY (id),

nome VARCHAR(50) NOT NULL,

idade int NOT NULL,

salario DECIMAL(10,2) NOT NULL DEFAULT 2000,

area VARCHAR(30)

);

insert into funcionarios values

(

1, 'Ramesh', 32, 4000, 'TI'

);

insert into funcionarios values

(

2, 'Khilan', 25, 5000, 'Financeiro'

);

insert into funcionarios values

(

7, 'Muffy', 24, 7000, 'Gerente'

);

insert into funcionarios (id, nome, idade) values

(

3, 'Kaushik', 23

);

insert into funcionarios (id, nome, idade) values

(

4, 'Chaitali', 25

);

insert into funcionarios (id, nome, idade, salario) values

(

5, 'Hardik', 27, 1500

);

insert into funcionarios (id, nome, idade, area) values

(

6, 'Komal', 22, 'TI'

);

set SQL\_SAFE\_UPDATES=0;

-- 8) Modifique os funcionários sem area para área de Marketing

update funcionarios

set area = 'Marketing' where area is NULL;

delete from funcionarios

where idade = 25;

-- 3) Exiba todos os funcionários com mais de 25 anos

select \* from funcionarios;

-- 4) Exiba todos os funcionários com salario acima de 3000

select \* from funcionarios

where idade > 25;

-- 5) Exiba todos os funcionários sem salario

select \* from funcionarios

where salario > 3000;

-- 6) Exiba todos os funcionários da area de TI

select \* from funcionarios

where salario is NULL;

-- 7) Exiba todos os funcionários cujos nomes começam com K

select \* from funcionarios

where area = 'TI';

--

select \* from funcionarios

where nome like 'K%';

-- 10) Exiba os 3 funcionários com maiores salários

select \* from funcionarios

order by salario;

-- 11) Exiba todos os funcionários por ordem alfabética

select \* from funcionarios

order by salario desc LIMIT 3;

-- 12) Exiba todas as áreas e exclua as repetidas

select \* from funcionarios

order by nome;

select distinct area from funcionarios;

select sum(salario) from funcionarios;

select sum(salario), area from funcionarios

group by area;

TESTES EM MY SQL

SELECT 'Hello, World!';

SELECT 'oi', 123, 20130831, '2013-08-31';

SELECT CURRENT\_DATE, CURRENT\_TIME, CURRENT\_TIMESTAMP, UNIX\_TIMESTAMP() FROM DUAL;

SELECT DATE\_FORMAT(CURRENT\_TIMESTAMP - 1, '%d/%M/%Y %H:%i'), DATE\_SUB(CURRENT\_DATE, INTERVAL 10 DAY),

DATE\_ADD(CURRENT\_DATE, INTERVAL 10 DAY);

SELECT EXTRACT(MONTH FROM CURRENT\_DATE);

SELECT \* FROM aluno

WHERE ID = CAST(1 AS CHAR);

SELECT 'banana', INSTR('banana', 'ana'), REPLACE('banana', 'ana', 'marcela'),

SUBSTRING('banana',3), SUBSTRING('banana',3,2), CONCAT('Jaque, ', 'vc ', 'é ', 'Incrivelmente ', 'linda!'),

TRIM(' OI '),

LTRIM(' OI '),

RTRIM(' OI '),

UPPER('oi'), LOWER('OI');

-- %d - 01,02... 31 (dia)

-- %H - 00...23 (hora)

-- %h - 00..12 (hora)

-- %i - minutos

-- %S - segundos

-- %Y - Ano (2017)

-- y - ano (17)

-- %m - 01...12 (mês)

-- %M - Jan, Fev...Dez (mês)

15/08/17

**JOIN**

|  |
| --- |
| PEDIDOS |
| num\_pedido\* |
| data |
| status |
| comentário |
| Id\_cliente |

|  |
| --- |
| DETALHES\_PEDIDO |
| num\_pedido\* |
| Cod\_produto |
|  |
|  |
|  |

|  |  |
| --- | --- |
| 0 | padrão |
| 1 |  |
| 2 | back |
| 3 | Grid |

|  |  |
| --- | --- |
| ID | padrão |
| A | brick |
| B | grid |
| C | diamond |

**CROSS JOIN**

- Realiza o produto cartesiano das linhas de várias tabelas

SELECT T1 ID, T2 ID FROM T1

CROSS JOIN T2;

|  |  |
| --- | --- |
| T1 ID | T2 ID |
| 1 | A |
| 1 | B |
| 1 | C |
| 2 | A |
| 2 | B |
| 2 | C |
| 3 | A |
| 3 | B |
| 3 | C |

**INNER JOIN**

- As linhas de duas tabelas são casadas se e somente se tiverem colunas com valores idênticos

- A condição é informada no predicado do join

SELECT T1 ID, T2 ID FROM T2

INNER JOIN T2 ON T1.PADRAO=T2.PADRAO

|  |  |
| --- | --- |
| T1 ID | T2 ID |
| 2 | A |
| 3 | B |

**LEFT JOIN**

- Similar ai inner join, e também requer um predicado

- Retorna todas as linhas da tabela da esquerda, incluindo as linhas que casarem com o predicado

SELECT T1 ID, T2 ID FROM T1

LEFT JOIN T2 ON T1.PADRAO = T2.PADRAO

|  |  |
| --- | --- |
| T1 ID | T2 ID |
| 1 | NULL |
| 2 | A |
| 3 | B |

**RIGHT JOIN**

- Análogo ao left join, no entanto retorna todas as linhas da tabela da direita

SELECT T1 ID, T2 ID FROM T1 RIGHT JOIN T2 ON T1.PADRAO = T2.PADRAO

|  |  |
| --- | --- |
| T1.ID | T2.ID |
| 2 | A |
| 3 | B |
| NULL | C |

**APLICAÇÕES COM BD**

* Ativo: BD contém as regras de negócio (DBA)
* Passivo: BD funciona como um repositório (PROGRAMADOR)
* Misto: regras de negócios estão espalhadas (?)

Operações + rápidas;

Mais difícil manutenção;

**STORED PROCEDURES**

- Um stored procedure (procedimento armazenado - SP) é um segmento de código SQL declarativo armazenado dentro do BD;

- Os SP’s podem ser invocados por outros SP’s triggers ou aplicações como java, php, python, etc.

a

get\_produtos

SELECT \* FROM PRODUTOS

b

get\_produtos

*VANTAGENS*

* Aumentam o desempenho de aplicações. São compiladas e armazenadas dentro do BD;
* Diminuem o tráfego entre as aplicações e BD;
* Reutilizáveis e transparentes para as aplicações;
* São seguras. O DBA pode gerenciar as permissões de acesso às tabelas;

*DESVANTAGENS*

* Muitos SP’s aumentam o consumo de memória;
* Uso de operações lógicas aumentam o uso de CPU;
* Difícil criar regras de negócios muito complexas;
* Difícil realizar depuração dos SP’s;
* Difícil de dar manutenção.

**CRIANDO UM SP**

delimiter //

create procedure alo\_mamae()

begin

select ‘alo\_mamae’;

end //

delimiter ;

* Delimiter: não está relacionado à sintase do SP. Este comando altera o delimitador ‘ ; ’ do sql para algum outro. Por que é necessário mudar o delimitador? Queremos passar código do SP completo para o BD, no entanto o interpretador do MySQL interpreta uma sentença de cada vez.
* CREATE PRCEDURE: Cria um novo SP. Os parâmetros podem ser passados pelos parênteses.
* BEGIN / END: Corpo do SP. Onde são colocadas todas as instruções declarativas;

Para testar a SP:

call alo\_mamae();

**PARÂMETROS**

- A maioria das SP’s necessitam de parâmetros. Tornam as SP’s mais flexíveis;

- Os parâmetros tem 3 modos:

\* IN: É o modo padrão. O pgma que chama deve passar um valor, e o valor fica protegido;

\* OUT: O valor pode ser modificado no SP. O SP não pode acessar o valor inicial;

\* INOUT: Combinação de IN e OUT.

*SINTAXE*

<modo> <parâmetro> <tipo>

*Exemplo:*

delimiter //

create procedure get\_escritorio\_por\_pais (in nome\_pais varchar(255))

begin

SELECT \* FROM ESCRITORIO

WHERE PAIS = nome\_pais;

end //

delimiter ;

//call get\_escritorio\_por\_pais(‘BR’);

*Exemplo:*

delimiter $$

create procedure conta\_pedidos(in status varchar(25), out total int)

begin

SELECT count (num\_pedido)

INTO TOTAL FROM pedidos

Where status\_pedido = status;

End $$

delimiter ;

call conta\_pedidos(‘enviado’, @total);

SELECT @total;

@total: é uma variável que é passada como argumentos para o SP

*Exemplo:*

delimiter //

create procedure contador(inout cont int, in incr int)

begin

SET cont = cont + incr;

End //

delimiter ;

SET @cont = 1

Call contador(@cont, 1); --2

Call contador(@cont, 1); --3

Call contador(@cont, 5); --8

SET @cont FROM DUAL;

**ESTRUTURAS DE DECISÃO (IF/ELSE/ELSEIF)**

if <expressão> then

<sentenças>;

end if;

//if / else

if <expressão> then

<sentenças>;

else

<sentenças else>;

end if;

//if /elseif

if <expressão> then

<sentenças>;

elseif <expressão> then

<senteças>;

else

<sentenças>;

end if;

*Exemplo:*

DELIMIER $$

CREATE PROCEDURE

Get Tipo Cliente(IN cliente\_id INT, OUT cliente\_tipo VARCHAR(10))

BEGIN

DECLARE limite DOUBLE DEFAULT 0; //inicia varia de escopo local

SET limite = 0; //insere na variável

SELECT c.cliente\_limite INTO limite

FROM cliente.c WHERE c.cliente\_id=cliente\_id;

if limite = 50000 then

SET cliente\_tipo = ‘SUPER VIP’;

elseif limite <= 50000 and limite > 30000 then

SET cliente\_tipo = ‘GOLD VIP’;

else

SET cliente\_tipo = ‘NORMAL’;

end if;

END $$

**ESTRUTURAS DE DECISÃO (CASE / WHEN)**

case <expressão>

when <valor1> then

<sentença1>;

when <valor2> then

<sentença2>;

.

.

.

else

<sentençaN>;

end case;

*Exemplo:*

case pais

when ‘USA’ then

set taxa = 1.5;

when ‘CAN’ then

set taxa = 1.3;

when ‘BRA’ then

set taxa = 2.5;

else

set taxa = 0;

end case;

/////////////////////////////////////

case

when limite > 50000 then

set cliente\_tipo = ‘SUPER VIP’

when limite <= 50000 and limite >30000 then

set cliente\_tipo = ‘SUPER GOLD’;

else

set cliente\_tipo = ‘NORMAL’;

end case;

**VARIAVEIS LOCAIS**

- Usadas para armazenar valores imediatos;

DECLARE <nome\_variavel> <tipo> DEFAULT <valor>;

<tipo> = QUALQUER TIPO DE DADOS VÁLIDO EM SQL: DATE, INT, VARCHAR...

- O valor inicial das variáveis é NULL caso não sejam inicializadas;

DECLARE saldo INT;

DECLARE nome VARCHAR(100);

DECLARE data DATE DEFAULT 20171010;

- Atribuição é feita pelo comando SET;

SET nome = ’joão’;

SET saldo = 1000;

SET saldo = saldo + 20;

Atividade

1 Criar um SP para cadastrar alunos

2 Criar um SP para cadastrar disciplinas

3 Criar um SP para atribuir notas aos alunos

|  |  |  |  |
| --- | --- | --- | --- |
| ALUNOS | | | |
| ID | **NOME** | **DATA\_BASE** | **CURSO** |
|  |  |  |  |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| NOTA | | | |
| ID | **ID\_ALUNO** | **ID\_DISCIPLINA** | **NOTA** |
|  |  |  |  |
|  |  |  |  |

|  |  |
| --- | --- |
| DISCIPLINA | |
| ID | **NOME** |
|  |  |
|  |  |

***LOOP’s***

while <condição> do

end while;

loop...

end loop;

repeat...

end repeat;

*Exemplo:*

//WHILE

CREATE PROCEDURE P1

BEGIN

DECLARE V INT DEFAULT 0;

WHILE V < 5 DO

INSERT INTO TABELA VALUES V;

SET V += 2;

END WHILE;

END;

//REPEAT

CREATE PROCEDURE P1

BEGIN

DECLARE V INT DEFAULT 0;

REPEAT

INSERT INTO TABELA VALUES V;

UNTIL V >= 5;

END REPEAT;

END;

//LOOP

CREATE PROCEDURE P1

BEGIN

DECLARE V INT DEFAULT 0;

Loop\_label:

LOOP

INSERT INTO TABELA VALUES V;

SET V += 2;

IF V >= 5 THEN

LEAVE loop\_label;

END IF;

END LOOP;

END;

***CURSORES***

- São ponteiros para as linhas de uma consulta;

- Permite o processamento linha a linha;

\*\* Devemos evitar cursores por questões de desempenho \*\*

\*\* Não pode atualizar os valores enquanto o cursor percorrer a consulta \*\*