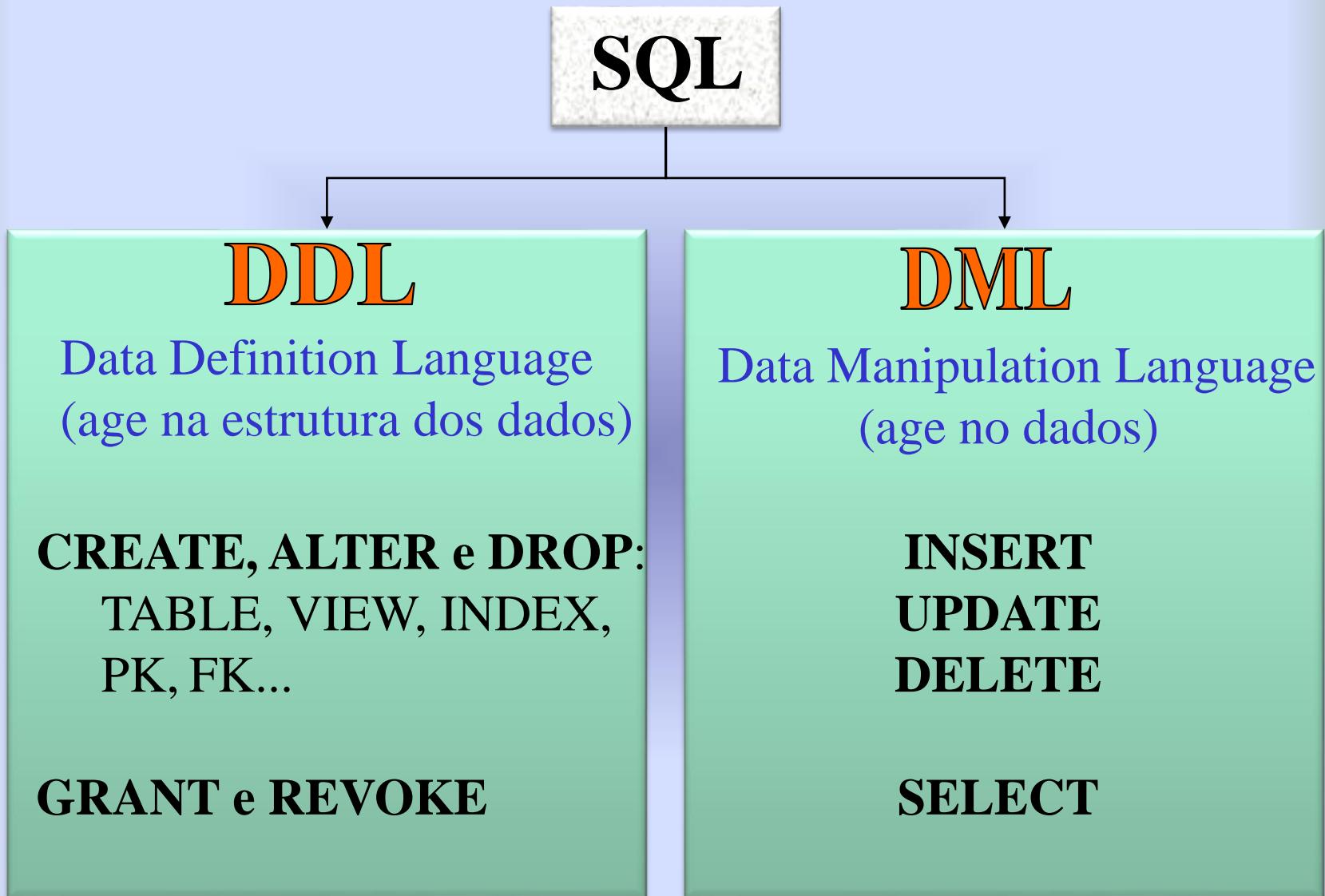




Unidade de Engenharia e Computação
Disciplina Banco de Dados - Prof^a. Eliana Caus Sampaio

Unidade 5

SQL



DDL – CREATE TABLE

Possibilita a criação de um novo esquema de tabela.

CREATE TABLE <table_name>
(<column_name> <datatype> { null | Not null } [,...]);

Onde:

table_name → refere-se ao nome que será dado a tabela

column_name → nome para cada coluna a ser criada para a tabela

datatype → refere-se ao tipo do dado (numeric, char, date, ...) que será atribuído a coluna

DROP TABLE <table_name>;

DDL – CREATE TABLE

Observações Importantes sobre a criação de tabelas:

- ❖ A criação da tabela deve levar em consideração todas as colunas que ela deve possuir no seu estágio final.
- ❖ Características especiais sobre cada coluna (pk, fk, indices) poderá ser feito em momento posterior.
- ❖ Na criação da tabela, algumas restrições de integridade devem ser declaradas, ao passo que outras, podem ser declaradas em momento posterior, via ALTER TABLE.
 - ✓ Restrições obrigatórias no CREATE: Datatype, Null/Not Null
 - ✓ Restrições que podem ser declaradas depois: Primary Key, Foreign Key, Unique, Default, Check

DDL – ALTER TABLE

Possibilita a modificação do esquema de tabela.

ALTER TABLE <table_name>
ADD (<column_name> <datatype> { null | Not null } [,...]);

Onde:

table_name → refere-se ao nome da tabela que sofrerá a alteração
ADD → refere-se ao tipo de modificação que será aplicado a tabela.
É possível acrescentar: Primary Key, Foreign Key, novas colunas,...

ALTER TABLE <table_name>
DROP <column_name>;

DDL – ALTER TABLE

Adicionando restrição de Chave Primária – **Primary Key**

```
ALTER TABLE <table_name>
ADD PRIMARY KEY (<column_name>[,...]);
```

Onde:

table_name → refere-se ao nome da tabela que sofrerá a alteração
column_name → refere-se ao nome da coluna que será transformada em PK.

```
ALTER TABLE <table_name>
DROP PRIMARY KEY ;
```

DDL – ALTER TABLE

Adicionando restrição de Chave Estrangeira – **Foreign Key**

```
ALTER TABLE <table_name>
ADD CONSTRAINT <fk_name> FOREIGN KEY (<column_name>)
REFERENCES <table_refered> (<column_refered>)
ON UPDATE {restrict | cascade | set null }
ON DELETE {restrict | cascade | set null };
```

Onde:

table_name → refere-se ao nome da tabela que sofrerá a alteração

fk_name → nome que será dada a restrição de integridade.

column_name → nome da coluna que sofrerá a restrição

restrict, cascade e set null → Define as Ações Referenciais.

```
ALTER TABLE <table_name>
DROP CONSTRAINT <fk_name>;
```

DDL – CREATE INDEX

Criando Índice

```
CREATE [unique] INDEX <index_name>  
ON <table_name> (<column_name> [...]) ;
```

Onde:

unique → define se os valores a serem colocados nas colunas indexadas aceitam duplicidade ou não

index_name → nome da regra responsável por armazenar o índice

table_name → nome da tabela cuja coluna será indexada

column_name → nome da coluna que será indexada

```
DROP INDEX <index_name>;
```

DDL – CREATE TABLE com todas as RIs

```
CREATE TABLE <table_name>
(
    <column_name> <datatype> { null | Not null } [,...]
    PRIMARY KEY (<column_name> [,...])
    FOREIGN KEY <fk_name> (<column_name>)
        REFERENCES <table_refered>
            (<column_refered>)
        ON UPDATE { restrict | cascade | set null }
        ON DELETE { restrict | cascade | set null }
    INDEX <index_name> (<column_name> [,...])
)
;
```

DDL – CREATE VIEW

```
CREATE TABLE empr  
(matricula numeric(5)  not null,  
 nome      char(30)    not null,  
 salario    numeric (5,3) null);
```

```
SELECT * FROM empr;
```

01 jose 100,00
02 ana 900,00
03 paulo 500,00

```
Insert into empr  
Values (04, 'carlos', 10.00);
```

Externo

```
CREATE VIEW v_empr  
(vmatr, vnome)  
AS  
(SELECT matricula, nome  
FROM empr);
```

```
Select * from v_empr;
```

01 jose
02 ana
03 paulo

```
Insert into v_empr  
Values (04, 'carlos');
```

catálogo
conceitual

interno

DDL – CREATE VIEW

```
CREATE VIEW <view_name> [(<column_name>, ...)]  
AS (SELECT statement))  
[WITH CHECK OPTION];
```

Onde:

view_name → nome que será dado a view

column_name → nome de cada coluna que será exibida pela view

SELECT statement → escrita do comando SELECT que será responsável pelos dados a serem vistos pela view.

With check option → obriga que as regras usadas para o select da view, deverão ser seguidas para os comandos INSERT/UPDATE e DELETE.

```
DROP VIEW <view_name>;
```

DML - INSERT

Possibilita a inclusão de uma nova linha em uma tabela.

```
INSERT [INTO] {table_name| view_name}
    [{(column_list)}]
{VALUES (expression [, expression ] ...)| SELECT statement};
```

DML - INSERT

Onde:

table_nome | view_name → nome da tabela ou da view que receberá os dados

column_list → lista das colunas daquela tabela que receberão dados. Se omitidas, deverão ser passados dados para todas as colunas.

expression → tipos de dados que podem ser atribuidos as colunas.

SELECT statement → comando select responsável por capturar dados oriundos de várias fontes e que serão utilizados para alimentar a tabela.

DML - UPDATE

- ❖ Possibilita a alteração dos dados de uma coluna.
- ❖ A alteração poderá afetar uma ou inúmeras linhas, dependendo do que for definido no critério de seleção (WHERE)

```
UPDATE {table_name|view_name}  
SET column_name = {expression | null | (SELECT statement)}  
      [, column_name = {expression | null | (SELECT statement)}]  
{ FROM {table_name | view_name}  
      [, [{table_name | view_name}]] ...]  
[ WHERE search_conditions ]
```

DML - UPDATE

Onde:

table_name | view_name → nome da tabela ou da view que sofrerá a alteração

column_name → nome da coluna cujo conteúdo será modificado

SELECT statement → comando select responsável por capturar dados oriundos de várias fontes e que serão utilizados para alimentar a tabela.

table_name → na cláusula FROM é usado para indicar outras tabelas que precisa ser acessada para determinar quais linhas da tabela declarada na cláusula UPDATE serão modificadas.

search_condition → indica as restrições que deverão ser aplicadas ao conjunto de linhas das tabelas envolvidas de forma a restringir o universo de ação do comando UPDATE.

DML - DELETE

Possibilita a exclusão de uma linha em uma tabela.

```
DELETE FROM {table_name | view_name}  
[ WHERE search_conditions ]
```

DML - SELECT - Sintaxe

```
SELECT [all | distinct] select_list  
[ INTO table_name]  
[FROM {table_name | view_name} [, {table_name | view_name}] ]  
[WHERE search_conditions]  
[GROUP BY [all] column_name [, column_name] ...]  
[HAVING search_conditions ]  
[ORDER BY {column_name | select list_number | expression }  
                  [ asc | desc ]  
          [,  
          {column_name | select list_number | expression }  
                  [ asc | desc ]  
          ...]
```

DML - SELECT - Explicações adicionais

SELECT → define a forma como os dados serão projetados no resultado.

INTO → define uma tabela para onde os dados gerados serão armazenada. Funciona semelhante ao comando INSERT baseado em SELECT.

FROM → define a fonte de dados, que pode ser a partir de Tabelas, Views ou de outro comando SELECT.

WHERE → define as restrições que serão aplicadas ao conjunto dos dados lidos de forma a restringir o universo de atuação do comando SELECT.

DML - SELECT - Sintaxe – Explicações adicionais

GROUP BY → responsável pelo agrupamento de dados de forma a propiciar a execução de alguma função agregada: SUM, AVG, MIN...

HAVING → define as restrições que serão aplicadas ao resultado gerado por uma função agregada.

ORDER BY → ordena os dados a serem projetados de forma crescente (ASC) ou decrescente (DESC).

DML - SELECT - Sintaxe – Explicações Adicionais

all → indica que todas as linhas serão exibidas, ainda que duplicadas

distinct → indica que linhas duplicadas serão eliminadas do resultado

select_list → lista das colunas que serão projetadas no resultado

table_name → na cláusula INTO, indica a tabela destino para os dados recuperados pelo SELECT. Na cláusula FROM, indica as tabelas que serão usadas para compor o resultado.

search_condition → indica as restrições que deverão ser aplicadas ao conjunto de linhas das tabelas envolvidas de forma a restringir o universo de ação do comando SELECT.

DML - SELECT - Sintaxe – Explicações Adicionais

table_name → na clausula FROM é usado para indicar outras tabelas que precisa ser acessada para determinar quais linhas da tabela declarada na clausula UPDATE serão modificadas.

Condition → indica as restrições que deverão ser aplicadas ao conjunto de linhas das tabelas envolvidas de forma a restringir o universo de ação do comando UPDATE.

DML - SELECT - Operadores

Aritméticos → +, -, /, *, %

Lógicos → NOT, OR, AND

Comparação → =, <>, <=, <, >=, >, IS NULL,

BETWEEN min AND max

IN (valor1, valor2, ...)

LIKE (busca um padrão dentro de campos).

Utiliza 2 Curingas:

% → despreza vários caracteres (de 0 a muitos)

_ → despreza um caracter (obrigatoriamente um)

DML - SELECT - Operadores – Exemplos

- 1) Listar o nome do produto, o preço atual e o preço projetado com 50% de aumento.

```
SELECT PRO_NOME, PRO_PRECO * 1.5 AS "PREÇO PROJETADO"  
FROM PRO;
```

- 2) Listar o nome e o preço dos produtos cujo preço com 50% de desconto sejam menor do que 100 e nome do produto comece com a letra A e tenha 20 caracteres.

```
SELECT PRO_NOME, PRO_PRECO  
FROM PRO  
WHERE (PRO_PRECO / 2) < 100  
      AND (PRO_NOME LIKE 'A%'  
      AND LENGTH(PRO_NOME) = 20);
```

DML - SELECT - Operadores – Exemplos

- 3) Listar o nome e preço dos produtos cujo preço seja 10, 40, 70, 80

```
SELECT PRO_NOME, PRO_PRECO  
FROM PRO  
WHERE PRO_PRECO IN (10, 40, 70, 80);
```

- 4) Listar o nome e o preço dos produtos cujo preço seja menor do que 100 ou maior do que 200.

```
SELECT PRO_NOME, PRO_PRECO  
FROM PRO  
WHERE PRO_PRECO NOT BETWEEN 100 AND 200;
```

DML - SELECT - Junções

- ❖ Muitas consultas podem requerer dados dispersos em mais que uma tabela
- ❖ Pode ocorrer de uma tabela não produzir nada no resultado, mas faz parte do “caminho” a ser percorrido para se chegar ao resultado desejado.
- ❖ Junções(ligações) possibilitam acessar mais que uma tabela em uma consulta
- ❖ As ligações entre as tabelas, dá-se através dos valores de uma ou mais colunas.
- ❖ Existem 2 tipos de junção:
 - ✓ Junção Interna
 - ✓ Junção Externa: Esquerda, Direita e Completa.

DML - SELECT - Junção Interna

- ❖ Busca encontrar linhas em duas tabelas a partir de algum critério de seleção (clausula de junção).
- ❖ Somente as linhas das duas tabelas que atenderem ao critério de seleção participam do resultado, ou melhor, serão projetadas.
- ❖ O critério de seleção baseia-se na comparação de colunas das tabelas envolvidas.
- ❖ As colunas podem ter nomes diferentes e pertencerem a domínios distintos.
- ❖ Linhas que não satisfizerem ao critério de seleção são eliminadas.
- ❖ A palavra JOIN só está presente na sintaxe proposta a partir do padrão ANSI SQL/92.
- ❖ Tomar cuidado com junções internas com mais que 2 tabelas, pois se algum critério de seleção não for atendido, a linha será desprezada.

DML - SELECT - Junção Interna - Exemplos

- 1) Listar o nome do empregado e o nome do dependente de cada empregado.

Sintaxe anterior ao ANSI/SQL92 → Junção pela clausula WHERE

```
SELECT NOME_EMPREGADO, NOME_DEPENDENTE  
FROM EMP, DEP  
WHERE EMP.MATRICULA = DEP.MATRICULA;
```

Sintaxe posterior ao ANSI/sQL92 → Junção pela clausula FROM.

```
SELECT NOME_EMPREGADO, NOME_DEPENDENTE  
FROM EMP INNER JOIN DEP  
ON (EMP.MATRICULA = DEP.MATRICULA);
```

DML - SELECT - Junção Interna - Exemplos

Junção envolvendo mais que duas tabelas.

```
SELECT NOME_EMPREGADO, NOME_CARGO,  
       NOME_DEPENDENTE  
FROM EMP, DEP, CAR  
WHERE EMP.MATRICULA = DEP.MATRICULA  
  AND EMP.COD_CARGO = CAR.COD_CARGO;
```

```
SELECT NOME_EMPREGADO, NOME_CARGO,  
       NOME_DEPENDENTE  
FROM EMP INNER JOIN DEP  
          ON (EMP.MATRICULA = DEP.MATRICULA)  
INNER JOIN CAR  
          ON (EMP.COD_CARGO = CAR.COD_CARGO);
```

DML - SELECT - Junção Externa

Busca encontrar pares de linhas entre duas tabelas e dá liberdade a uma tabela ter sua linha no resultado mesmo que esta linha não forme par com alguma linha da outra tarefa referenciada.

São muito usadas para listar uma tabela que possua relacionamento parcial. Pode ser Left, Right ou Full.

-Left → A tabela a esquerda da clausula de junção está desobrigada de encontrar um PAR para ser listada.

-Right → A tabela a direita da clausula de junção está desobrigada de encontrar um PAR para ser listada.

-Full → Ambas as tabelas, a direita e a esquerda serão totalmente listadas.

DML - SELECT - Junção Externa a Esquerda - Exemplos

- 1) Listar o nome de todos os empregados. Caso ele tenha filhos, listar também o nome do filho.

```
SELECT NOME_EMPREGADO, NOME_DEPENDENTE  
FROM EMP, DEP  
WHERE EMP.MATRICULA = DEP.MATRICULA (+);
```

No exemplo acima, a tabela de empregado será totalmente listada. Se um determinado empregado, possuir dependentes, o dado referente ao dependente será listado também. Caso não encontre, será colocado NULO no campo referente ao dado do dependente.

Essa sintaxe é suportada no Oracle.

DML - SELECT - Junção Externa a Esquerda - Exemplos

- 1) Listar o nome de todos os empregados. Caso ele tenha filhos, listar também o nome do filho.

```
SELECT NOME_EMPREGADO, NOME_DEPENDENTE  
FROM EMP, DEP  
WHERE EMP.MATRICULA *= DEP.MATRICULA ;
```

No exemplo acima, a tabela de empregado será totalmente listada. Se um determinado empregado, possuir dependentes, o dado referente ao dependente será listado também. Caso não encontre, será colocado NULO no campo referente ao dado do dependente.

Essa sintaxe é suportada no Sybase e SQL Server.

DML - SELECT - Junção Externa a Esquerda - Exemplos

- 1) Listar o nome de todos os empregados. Caso ele tenha filhos, listar também o nome do filho.

```
SELECT NOME_EMPREGADO, NOME_DEPENDENTE  
FROM EMP LEFT OUTER JOIN DEP  
      ON EMP.MATRICULA = DEP.MATRICULA ;
```

No exemplo acima, a tabela de empregado será totalmente listada. Se um determinado empregado, possuir dependentes, o dado referente ao dependente será listado também. Caso não encontre, será colocado NULO no campo referente ao dado do dependente.

Essa sintaxe é suportada no em SGBD a partir do ANSI/SQL92.