Estudo de SQL
Por: Rafael, O Amaldiçoado

Em primeiro lugar se faz necessário entender o grupo de comandos existentes dentro da linguagem SQL, sendo eles:

1 - DDL (Data Definition Language)

É utilizado para a criação de alteração de tabelas existentes, sendo exemplos destes grupos comandos como:

I - CREATE : Utilizado para para criar uma nova tabela ou um novo objeto

Ex:
```
CREATE DATABASE database_name
```
ou

```
CREATE TABLE employee
  ( id number(5),
  name char(20),
  dept char(10),
  age number(2),
  salary number(10),
  location char(10)
  );
```

II- ALTER: Utilizado para modificar a tabela ou banco de dados, adicionando ou removendo dados da mesma (caso seja a tabela), além disso, nos casos necessários há a possibilidade de modificar o collation do banco de dados em questão, permitindo o uso de letras de alfabetos diferentes (caso queira modificar o banco de dados) .

Ex:
```
ALTER DATABASE database_name

[COLLATE collation_name ]
```
ou

```
ALTER TABLE table_name

ADD column_name datatype;
```

III - DROP - Usado para deletar todo o conteúdo de um Banco de Dados (tabelas, views, Procedures, etc), é importante notar que esta Query deleta permanentemente todas as informações de banco de dados, sendo necessário que haja um backup prévio de todo o banco de dados para que possa haver a recuperação dos dados (CUIDADO COM O ESTE COMANDO)

Ex: `DROP DATABASE database_name;`

IV - RENAME - Utilizado para renomear um tabela, contudo nem todos os sistemas de gerenciamento de banco de dados aceita este comando, pois ele não é um comando padrão do SQL, por exemplo no MS SQL é utilizado o comando (SP_RENAME), ao invés do RENAME normal.

Ex: `RENAME TABLE {tbl_name} TO {new_tbl_name};`

ou podemos utilizar o método Oracle:

```
ALTER TABLE {tbl_name} RENAME TO {new_tbl_name};
```

Obs: Em sistemas de gerenciamento mais antigos será necessário modificar todas as instâncias da tabela ao modificar o nome, contudo em sistemas mais modernos e atualizados, como por exemplo, o Oracle 8i todas as dependências serão modificados automaticamente.

V- TRUNCATE - É utilizado para poder deletar todas as linhas de uma tabela, contudo não deleta a tabela em si, muito menos suas colunas, views, constraints e indexes.

Ex: `TRUNCATE TABLE table_name;`

2- DML (Data Manipulation Language)

É utilizado para manipulação de dados dentro do banco de dados, são exemplos desse grupo comandos como:

I - INSERT - Utilizado para inserir novos dados em uma linha dentro de uma tabela, podendo ser utilizado com o método de identificar as colunas as quais os novos dados serão inseridos, como no primeiro exemplo, contudo se os dados a serem inseridos forem para todas as tabelas, não se faz necessário exemplificar qual coluna os dados devem ser inseridos, como no segundo exemplo.

Ex:
```
INSERT INTO TABLE_NAME
[ (col1, col2, col3,...colN)]
VALUES (value1, value2, value3,...valueN);
```

ou

Ex:
```
INSERT INTO TABLE_NAME
VALUES (value1, value2, value3,...valueN);
```

II - UPDATE - É utilizado para modificar as linhas existentes dentro de uma tabela, é preciso lembrar que a condição WHERE deve ser utilizada para identificar quais linhas de quais colunas devem ser afetadas, caso a cláusula WHERE não seja implementada todas as colunas serão afetadas.

Ex:
```
UPDATE employee
SET location ='Mysore'
WHERE id = 101;
```

ou (sem WHERE)

Ex:
```
UPDATE employee
SET salary = salary + (salary * 0.2);
```

III- DELETE - É utilizado para deletar linhas dentro de uma tabela, tendo como a cláusula WHERE o ponteiro para registro de qual linha será deletada com o comando, como poderá ser visto no primeiro exemplo, caso o WHERE não seja adicionado, todas as linhas da tabela serão deletadas.

Ex:
```
DELETE FROM employee WHERE id = 100;
```

ou (sem WHERE)

Ex:
```
DELETE FROM employee;
```

IV - SELECT - É utilizado para selecionar e obter informações de dados de tabelas específicas do banco de dados, sendo necessário, para se fazer um SELECT simples, em primeiro lugar o nome da coluna a ser buscada a informação e em segundo lugar o nome da tabela dentro do banco de dados, mas poderá haver a necessidade de se tentar selecionar todas as informações desta tabela, com isto, utilizaremos o símbolo de (*), que no SQL signifca todos, como poderá ser visto no segundo exemplo :

Ex: `SELECT first_name FROM student_details;`

ou (para selecionar todos)

Ex: `SELECT first_name FROM student_details;`

É preciso que se atente ao fato de que o uso do SELECT simples nem sempre será o necessário para a Query a ser realizada, sendo, desta forma, necessário o uso de cláusulas como WHERE e ORDER BY, para que o comando seja mais claro ao receber as informações desejadas.

Em outros casos, se fará necessário o uso de agentes de conexão como por exemplo o símbolo de soma (+), para que os dados obtidos sejam o mais limpo possível para a leitura, como demonstrado no exemplo a seguir:

Ex: `SELECT first_name + ' ' + last_name FROM employee;`

O exemplo abaixo terá uma saída no seguinte modelo:

Rafael Galvao

Onde caso a formatação não tivesse sido implementada, o resultado sairia no seguinte formato:

Ex: `SELECT first_name + last_name FROM employee;`

Com uma saída de :

RafaelGalvao

Um outro comando que pode ser realizado com o SELECT, sendo este o SELECT INTO, que tem a função de copiar as informações de uma tabela e inserir estes dados em outra, contudo se faz necessário que o número de colunas e o tipo de dado da informação selecionada seja igual. Podemos ver o comando evidenciado a seguir:

Ex:
```
SELECT column_name(s)
INTO new_table
FROM table_name;
```

Assim como o método simples, há a possibilidade de se obter este comando se utilizando do comando (*), para obter todas as colunas da tabela que se quer copiar, como evidenciado pelo comando a seguir:

Ex:
```
SELECT *
INTO Copy_Employee
FROM Employee;
```

Além disso, há a possibilidade de que esta tabela seja copiada para outro banco de dados, com o uso da cláusula IN, como visto no exemplo abaixo:

Ex:
```
SELECT *
INTO Copy_Employee IN 'Backup.mdb'
FROM Employee;
```

Há, também, como selecionar colunas específicas as quais se deseja copiar, como pode ser percebido no exemplo abaixo:

Ex:
```
SELECT EmployeeID, EmployeeName
INTO Copy_Employee
FROM Employee;
```

Por fim, temos com o comando SELECT INTO, a possibilidade de especificar com a cláusula WHERE quais os dados a serem copiados, como por exemplo somente os funcionários do sexo masculino:

Ex:
```
SELECT EmployeeID, EmployeeName
INTO Copy_Employee
FROM Employee
WHERE Gender='Male';
```

O terceiro e último método de SELECT que temos dentro do SQL é o SELECT TOP, sendo este comando utilizado para retornar uma Query com o número máximo X e/ou uma porcentagem N da tabela, contudo é preciso entender que nem todos os sistemas de gerenciamento de banco de dados suportam este comando, sendo os únicos o MSSQL server e o MS Access database, para acessar um número limite de informações em outros sistemas como o MySQL se utiliza a cláusula LIMIT e no sistema da Oracle se utiliza o ROWNUM .

Ex:
```
SELECT TOP 3 * FROM Employee;

      or

SELECT TOP 50 PERCENT * FROM Employee;
```

Utilizando a cláusula LIMIT no MySQL:

Ex:
```
SELECT * FROM Employee
LIMIT 3;
```

Utilizando a cláusula ROWNUM no Oracle:

Ex:
```
SELECT * FROM Employee
WHERE ROWNUM <= 3;
```

3 - TCL (Transaction Control Language)

É utilizado para gerenciar mudanças que afetam os dados dentro de uma transação, sendo estes comandos:

I - ROLLBACK - Utilizado quando se há a necessidade de cancelar a transação em andamento, contudo é preciso notar que não são todos os comandos que permitem o retorno ao estado anterior com o ROLLBACK, sendo um dos comandos que permitem o uso o DELETE

Ex:
```
SELECT * FROM employees;
     DELETE FROM employees
     WHERE employee_id = 130;

ROLLBACK;
```

II - COMMIT - O comando COMMIT serve o propósito de confirmar a mudança e rejeitar um possível ROLLBACK do comando executado previamente, sendo assim, confirma de forma “permanente” o comando executado, sendo ele implementado da seguinte forma:

Ex:
```
SELECT * FROM employees;
DELETE FROM employees
WHERE employee_id = 130;

COMMIT;
```

III - SAVEPOINT - Cria um ponto de retorno que permite a utilização de um ROLLBACK para que caso seja necessário, haja um retorno dos comandos executados para ajuste. Para cada argumento é necessário que o nome dado ao SAVEPOINT seja diferente, caso o nome de um SAVEPOINT seja igual a um anterior, o primeiro SAVEPOINT é apagado. A semântica utilizada para a criação de um savepoint é denominada no exemplo abaixo:

Ex:
```
UPDATE employees
    SET salary = 7000
    WHERE last_name = 'Banda';
 SAVEPOINT banda_sal;

SELECT SUM(salary) FROM employees;

      Algo deu errado ou deve ser modificado

ROLLBACK TO SAVEPOINT banda_sal;
```

4 - DCL (Data Control Language)

É utilizado para criar uma segurança para objetos dentro do banco de dados

I - GRANT - É Utilizado para prover acesso ou privilégios dentro do banco de dados para usuários, sendo o nome destes acessos/privilégios dados como: ALL, EXECUTE e SELECT, se faz necessário aferir qual o objeto ao qual este privilégio será dado, bem como, a quem o privilégio será concedido, podendo conter o nome de Usuário, Role do usuário ou PUBLIC (para dar acesso a todos), além disso, poderá haver uma opção de GRANT onde o usuário escolhido poderá dar esta permissão para outros . A sintaxe do GRANT é dada pelo exemplo abaixo:

Ex:
```
GRANT privilege_name
ON object_name
TO {user_name |PUBLIC |role_name}
[WITH GRANT OPTION];
```

II - REVOKE - É a antítese do GRANT,, onde com este comando será revogado os privilégios de um usuário ou role para determinado privilégio ou objeto, sendo utilizada a sintaxe a seguir:

Ex:
```
REVOKE privilege_name
ON object_name
FROM {user_name |PUBLIC |role_name}
```

Tendo visto os grupos de comando existentes dentro da linguagem SQL, se faz necessário, também, entendermos as cláusulas que foram assinaladas anteriormente em vários comandos vistos acima, desta forma, serão enumeradas as diversas cláusulas e suas funções dentro do SQL.

I - WHERE - A cláusula WHERE tem a função de indicar ou sinalizar qual o apontamento que se fará necessário para o comando esperado retirando informações desnecessárias do Query a ser realizado. Pelo fato de WHERE ser uma cláusula sua sintaxe está ligada à um comando, como evidenciado pelo exemplo com SELECT a seguir:

Ex: `SELECT first_name, last_name FROM student_details`

`WHERE id = 100;`

É preciso salientar que há a possibilidade de utilizar expressões com a cláusula WHERE, como no exemplo a seguir:

Ex: `SELECT name, salary, salary*1.2 AS new_salary FROM employee`

`WHERE salary*1.2 > 30000;`

II - AND - É utilizada quando queremos especificar múltiplas condições juntas dentro de um QUERY com a cláusula WHERE. A sintaxe ao qual ela é utilizada está evidenciada no Exemplo abaixo:

Ex : `SELECT * FROM store_employee WHERE age > 21 AND age < 30 AND salary > 2000`

III - OR - É uma cláusula utilizada quando precisamos passar múltiplas condições e necessitamos que os dados satisfaçam uma destas condições. A sintaxe é dada como a seguinte:

Ex: `Select * from dataflair_employee where salary > 30000 OR age > 26`

IV - LIKE - É utilizado para encontrar diferentes padrões dentro do dado que se quer obter, como por exemplo somente nomes que comecem com a letra a ou terminam com a letra a, sendo esta opção identificada por meio do símbolo (%), como pode ser vist pela sintaxe abaixo:

Ex:
```
Select * from dataflair_employee
    where name_emp LIKE 'A%' ;
```

V - ORDER BY - É utilizado para organizar os dados de forma ascendente ou descendente, sendo o default a sua forma ascendente. Essa cláusula é identificada na forma abaixo:

Ex: `Select * from dataflair_employee Order by salary desc ;`

VI - GROUP BY - É uma cláusula utilizada, principalmente, para sumarizar das linhas as informações, normalmente se é utilizada funções agregativas, como por exemplo, COUNT e SUM. A sintaxe desta cláusula é dada como:

Ex: `SELECT * FROM table_name WHERE condition GROUP BY column1 ;`

Agora que passamos pelo grupo de comandos e cláusulas existentes dentro da linguagem SQL, podemos iniciar o estudo sobre cláusulas dentro da linguagem, além de operadores lógicos e matemáticos, constrain, views, join e function. Mesmo parecendo ser muitos pontos a serem vistos dentro deste “artigo”, se faz necessário ter um entendimento de cada um deles, contudo entraremos por partes, sendo a primeira as Constraints.

CONSTRAINT

I - PRIMARY KEY- Pode ser posta tanto em nível de coluna, onde o comando será realizado imediatamente após a criação daquela coluna, como em nível de tabela, sendo esta última normalmente postulada subsequentemente da criação da tabela. A sintaxe da Primary Key dentro da linguagem de SQL é dada como no exemplo a seguir:

Ex: Ao nível de coluna

```
column name datatype [CONSTRAINT constraint_name] PRIMARY KEY
```

ou

```
CREATE TABLE employee
( id number(5) PRIMARY KEY,
```

```
name char(20),
dept char(10),
age number(2),
salary number(10),
location char(10)
);
```

Ou

```
CREATE TABLE employee
( id number(5) CONSTRAINT emp_id_pk PRIMARY KEY,
name char(20),
dept char(10),
age number(2),
salary number(10),
location char(10)
);
```

Ex: Em nível de Tabela

```
CREATE TABLE employee
( id number(5), NOT NULL,
name char(20),
```

```
dept char(10),
age number(2),
salary number(10),
location char(10),
ALTER TABLE employee ADD CONSTRAINT PK_EMPLOYEE_ID
PRIMARY KEY (id)
);
```

II - FOREIGN KEY- É utilizada para conectar duas colunas ou duas tabelas entre si, sendo ela uma “cópia” de uma primary key já existente que reside em outro local. A FK pode ser, assim como a PK, em nível de coluna e nível da tabela. A sintaxe da Foreign Key em SQL é dada como sendo:

Ex: Em nível de coluna

```
[CONSTRAINT constraint_name] REFERENCES
Referenced_Table_name(column_name)
```

Ou

```
CREATE TABLE product
( product_id number(5) CONSTRAINT pd_id_pk
PRIMARY KEY,
product_name char(20),
supplier_name char(20),
unit_price number(10)
);

CREATE TABLE order_items
( order_id number(5) CONSTRAINT od_id_pk
PRIMARY KEY,
product_id number(5) CONSTRAINT pd_id_fk
REFERENCES, product(product_id),
product_name char(20),
supplier_name char(20),
unit_price number(10)
);
```

Ex: Em nível da tabela:

```
CREATE TABLE order_items
( order_id number(5) ,
product_id number(5),
product_name char(20),
supplier_name char(20),
unit_price number(10)
CONSTRAINT od_id_pk PRIMARY KEY(order_id),
CONSTRAINT   pd_id_fk   FOREIGN   KEY(product_id)
REFERENCES product(product_id)
);
```

OBS: Quando tivermos a necessidade de referenciar a própria tabela com um auto-relacionamento (Caso do Gerente), deveremos seguir a seguinte sintaxe abaixo:

```
CREATE TABLE employee
( id number(5) PRIMARY KEY,
name char(20),
dept char(10),
age number(2),
mgr_id number(5) REFERENCES employee(id),
salary number(10),
location char(10)
);
```

III - NOT NULL - Faz com que as colunas propostas com esta constraint não possam ter um dado NULL inserido, com isso assegura a necessidade de que um valor ou dado seja incluído dentro da coluna. A Sintaxe para o uso de NOT NULL no SQL é dada como a seguinte:

Ex: `[CONSTRAINT constraint name] NOT NULL`

Ou

```
CREATE TABLE employee
( id number(5),
name char(20) CONSTRAINT nm_nn NOT NULL,
dept char(10),
age number(2),
salary number(10),
location char(10)
);
```

IV - UNIQUE - Permite que uma coluna ou grupo de colunas possam ter valores únicos e diversos entre si (não pode ser duplicado). É importante salientar, que o UNIQUE ainda pode ser utilizado no nível de coluna e no nível de tabela. A sintaxe da UNIQUE KEY é dado como:

Ex: Nível de Coluna

```
[CONSTRAINT constraint_name] UNIQUE
```

Ou

```
CREATE TABLE employee
( id number(5) PRIMARY KEY,
name char(20),
dept char(10),
age number(2),
salary number(10),
location char(10) UNIQUE
);
```

Ex: Nível Tabela

```
[CONSTRAINT constraint_name] UNIQUE(column_name)
```

Ou

```
CREATE TABLE employee
( id number(5) PRIMARY KEY,
name char(20),
dept char(10),
age number(2),
salary number(10),
location char(10),
CONSTRAINT loc_un UNIQUE(location)
);
```

V - CHECK - Aplica uma regra de negócio para uma ou um grupo de colunas, fazendo com que todas as linhas satisfaçam essa regra. Tendo como sintaxe:

Ex: `[CONSTRAINT constraint_name] CHECK (condition)`

A nível de coluna:

```
CREATE TABLE employee
( id number(5) PRIMARY KEY,
name char(20),
dept char(10),
age number(2),
gender char(1) CHECK (gender in ('M','F')),
salary number(10),
location char(10)
);
```

A nível de tabela:

```
CREATE TABLE employee
( id number(5) PRIMARY KEY,
name char(20),
dept char(10),
age number(2),
gender char(1),
salary number(10),
location char(10),
CONSTRAINT gender_ck CHECK (gender in ('M','F'))
);
```