Fundamentos de Banco de Dados

SQL - Structured Query Language

Cleyton Carvalho da Trindade

SQL - Structured Query Language

- Linguagem comercial para BD relacional
  - Padrão ISO desde a década de 80
- A SQL vai além de realizar consultas em um banco de dados.
  - Possui recursos para definição da estrutura de dados, para modificar dados no banco de dados e recursos para especificar restrições de segurança e integridade.

Classificação dos Comandos

- **Data Definition Language (DDL)** (Linguagem de Definição de Dados). A SQL DDL fornece comandos para definição e modificação de esquemas de relação, remoção de relações e criação de índices. Os principais comandos que fazem parte da DDL são: CREATE, ALTER, DROP.

- **Data Manipulation Language (DML)** (Linguagem de Manipulação de Dados). A SQL DML inclui uma linguagem de consulta baseada na álgebra relacional e no cálculo relacional. Compreende também comandos para inserir, remover e modificar informações em um banco de dados. Os comandos básicos da DML são: SELECT, INSERT, UPDATE, DELETE

Classificação dos Comandos

- **Vision Definition Language (VDL)** (Linguagem de Definição de Visões). É o conjunto de comandos que fazem o cadastramento de usuários e determina seu nível de privilégio para os objetos do banco de dados. Os principais comandos são: GRANT, REVOKE.

- **Transactions control (Controle de Transações)**. A SQL inclui comandos para especificação do início e fim das transações. Diversas implementações permitem o trancamento explícito de dados para o controle de concorrência. (COMMIT, ROLLBACK, SAVEPOINT)

SQL
Comandos DDL

Criar um Banco de Dados

- SQL padrão não oferece tal comando
  - BDs são criados via ferramentas do SGBD
- Alguns SGBDs (SQL Server, DB2, MySQL) oferecem este comando
  - **create database** nome_BD
  - **drop database** nome_BD

Criar Tabelas

- Uma tabela (ou relação) SQL é definida usando o comando create table:

```
create table r (
        A1 D1,
        A2 D2,
        ...,
        An Dn)
```

onde r é o nome da relação, Ai é o nome de um atributo no esquema da relação r e Di é o tipo do atributo Ai.

Exemplo Create Table

```
create table cliente(
   nome      char(30),
   sexo      char(1),
   CPF       number(11),
   endereco        char(30),
   cidade          char(30) )
```

Criar Tabelas

- **Sintaxe Completa**

  CREATE TABLE <nome_da_tabela> (

  Atributo1 tipo [(tamanho)] [Null| Not Null] [Índice] ...,

  Atributo2 tipo [(tamanho)] [Nill| Not Null] [Índice] ...,

  CONSTRAINT Nome <restrições>)

- Os elementos em parêntesis são opcionais. O elemento Atributo1 representa o nome do atributo da tabela. O elemento tipo representa o domínio de cada atributo

Exemplo Create Table

```
CREATE TABLE Alunos (
    Codigo INT NOT NULL AUTO_INCREMENT,
    Nome VARCHAR(250) NOT NULL,
    Idade INT,
    Cod_Curso INT,
    PRIMARY KEY (Codigo),
    FOREIGN KEY (Cod_Curso) REFERENCES Curso (Cod_Curso)
    ON UPDATE CASCADE
    ON DELETE SET NULL,
    CHECK (Idade BETWEEN 17 AND 70)
)
```

Restrições de integridade

- As restrições de integridade servem para garantir as regras inerentes ao sistema que está sendo implementado, prevenindo a entrada de informações inválidas pelos usuários desse sistema.
- Valores nulos
  - Para evitar que em algum momento uma coluna de uma tabela possa conter valor nulo (ou não conter valor algum) deve-se utilizar a cláusula NOT NULL após a definição da coluna.

Restrições de integridade

- Valores duplicados
  - Existir situações onde o valor armazenado em um atributo de um registro deve ser único em relação a todos os registros da tabela. Isto é, não pode haver dois registros com o mesmo valor para um determinado atributo.

- Para implementar esta restrição de integridade deve-se utilizar a cláusula **UNIQUE** após a especificação de uma coluna.

Exemplo

```
CREATE TABLE Alunos (
    Codigo INT NOT NULL AUTO_INCREMENT,
    Nome VARCHAR(250) NOT NULL,
    CPF VARCHAR(11) UNIQUE,
    Cod_Curso INT,
    PRIMARY KEY (Codigo),
    FOREIGN KEY (Cod_Curso) REFERENCES Curso (Cod_Curso)
    ON UPDATE CASCADE
    ON DELETE SET NULL,
    CHECK (Idade BETWEEN 17 AND 70)
)
```

Restrições de integridade

- Definindo valores default
  - Pode-se definir um valor padrão para uma coluna acrescentando à sua definição a cláusula DEFAULT. Esta cláusula permite substituir automaticamente os valores nulos por um valor inicial desejado.

Exemplo

```
CREATE TABLE Alunos (
    Codigo INT NOT NULL AUTO_INCREMENT,
    Nome VARCHAR(250) NOT NULL,
    CPF VARCHAR(11) UNIQUE,
    Cod_Curso INT,
    cidade          char(20) DEFAULT ‘Maceio’,
    PRIMARY KEY (Codigo),
    FOREIGN KEY (Cod_Curso) REFERENCES Curso (Cod_Curso)
    ON UPDATE CASCADE
    ON DELETE SET NULL,
    CHECK (Idade BETWEEN 17 AND 70)
)
```

Restrições de integridade

- Evitar valores inválidos
  - Existem situações onde o valor de um atributo deve pertencer a um determinado domínio. Para que o valor de um atributo fique restrito a um conjunto de valores, utiliza-se a cláusula CHECK.
  - A finalidade da cláusula CHECK é especificar uma condição, isto é, uma limitação de integridade.

Exemplo

```
CREATE TABLE Alunos (
    Codigo INT AUTO_INCREMENT,
    Nome VARCHAR(250) NOT NULL,
    CPF VARCHAR(11) UNIQUE,
    Cod_Curso INT,
    Idade INT,
    Cidade VARCHAR(20) DEFAULT ‘Maceio’,
    PRIMARY KEY (Codigo),
    FOREIGN KEY (Cod_Curso) REFERENCES Curso (Cod_Curso)
    ON UPDATE CASCADE
    ON DELETE SET NULL,
    CHECK (Idade BETWEEN 17 AND 70)
)
```

Restrições de integridade

- **Chave Primária**
  - A função da chave primária é identificar univocamente uma linha da tabela. Cada tabela deve possuir uma chave primária. Quando se define um atributo como chave primaria, fica implícito as cláusulas UNIQUE e NOT NULL para este atributo, não sendo necessário a especificação destas.

- **ATENÇÃO**: Quando uma tabela possui uma chave primária composta por mais de um atributo, esta forma é OBRIGATÓRIA.

```
create table movimento(
        agencia INT, conta INT, valor DECIMAL(10,2),
        primary key (agencia, conta) )
```

Restrições de integridade

- Integridade referencial
  - Assegura que o valor de um determinado campo de uma tabela está presente na chave primária de outra tabela. Este campo é chamado chave estrangeira (FOREIGN KEY).
  - Este recurso permite a especificação de chaves primárias e estrangeiras como parte da instrução create table.

Exemplo

```
CREATE TABLE Alunos (
    Codigo INT NOT NULL AUTO_INCREMENT,
    Nome VARCHAR(250) NOT NULL,
    CPF VARCHAR(11) UNIQUE,
    Cod_Curso INT,
    Cidade VARCHAR(20) DEFAULT 'Maceio',
    PRIMARY KEY (Codigo),
    FOREIGN KEY (Cod_Curso) REFERENCES Curso (Cod_Curso)
    ON UPDATE CASCADE
    ON DELETE SET NULL,
    CHECK (Idade BETWEEN 17 AND 70)
)
```

Remover Tabela

- Para remover uma relação de um banco de dados SQL, usa-se o comando drop table.
  - O comando drop table remove todas as informações sobre a relação.

drop table r

onde **r** é o nome de uma relação (tabela) existente.

Alterar tabela

- O comando **alter table** é usado para adicionar, excluir ou alterar atributos em uma relação existente.
- Para inserir um novo atributo em uma tabela é usada a cláusula **add**. Todas as tuplas na relação são assinaladas como null para o valor do novo atributo.

```
alter table r add (
    A1 D1,
    A2 D2,
    ... )
```

Onde **r** é o nome de uma relação existente e **A1 D1, A2 D2, ...** é uma lista contendo nome do atributo (A) a ser adicionado e o tipo do atributo (D), respectivamente.

Alterar tabela

- Para excluir colunas de uma tabela utiliza-se a cláusula **drop**.

  alter table r drop A1, A2, ...

  Onde r é o nome de uma relação existente e A1, A2, ... é uma lista dos atributos a serem removidos.

- Para alterar o nome de um atributo de uma tabela utiliza-se a cláusula **rename**.

  alter table r rename A NA

  Onde r é o nome de uma relação existente, A é o nome do atributo a ter o seu nome alterado para NA. Para alterar o tipo de um atributo utiliza-se a cláusula modify.

Alterar Tabela

- **Sintaxe Completa**

```
ALTER TABLE nome_tabela
ADD [COLUMN] nome_atributo_1 tipo_1 [{RIs}]
  [{, nome_atributo_n  tipo_n [{RIs}]}]
|
MODIFY [COLUMN] nome_atributo_1 tipo_1 [{RIs}]
  [{, nome_atributo_n  tipo_n [{RIs}]}]
|
DROP COLUMN nome_atributo_1
  [{, nome_atributo_n }]
|
[ADD|DROP] [PRIMARY KEY ...|FOREIGN KEY ...]
```

Exemplos Alter Table

```
ALTER TABLE Alunos
   ADD nomePai VARCHAR(30)

ALTER TABLE Alunos DROP PRIMARY KEY

ALTER TABLE Alunos DROP COLUMN idade

ALTER TABLE Alunos
ADD FOREIGN KEY(codigo_Dis) REFERENCES
    disciplina (codigo)
```

SQL – Índices

- Definidos sobre atributos para acelerar consultas a dados
- Índices são definidos automaticamente para chaves primárias

SQL – Índices

- **Sintaxe**

CREATE [UNIQUE] INDEX nome_índice ON nome_tabela
(nome_atributo_1[{, nome_atributo_n }])

DROP INDEX nome_índice ON nome_tabela

- **Exemplos**

CREATE UNIQUE INDEX indAluno_CPF ON Aluno (CPF)

DROP INDEX indAluno_CPF ON Aluno

SQL
Comandos DML

Inserir Dados

- **INSERT INTO** é o comando do SQL que permite a introdução de dados nas tabelas.
- Sintaxe

  INSERT INTO nome_tabela [(lista_atributos)]
  VALUES (lista_valores_atributos) [, (lista_valores_atributos)]

- Exemplos

INSERT INTO Alunos VALUES (1, ‘André’, 20, 1)

INSERT INTO Alunos
(Codigo, Nome, Idade, Cod_Curso)
VALUES (4, ‘Carlos’, 28,2);

Alterar Dados

- UPDATE - Alteração de valores em um ou mais atributos numa tabela e com critérios específicos
- Sintaxe

UPDATE <nome_tabela>

SET <atributo> = <expressão>,

...

[WHERE <condição>]

Alterar Dados

- SET define quais são os atributos que se pretende atualizar e os novos valores para esse atributo.
- WHERE é opcional e é utilizada quando se pretende condicionar as tuplas.
- Exemplo:

```
UPDATE Alunos
SET Idade = Idade + 1
WHERE Codigo = 1
```

Excluir Dados

- DELETE - Eliminação de valores em uma ou mais tuplas numa tabela e com critérios específicos.
- Sintaxe:

  DELETE FROM <nome_tabela>
  [WHERE <condição>]

- Exemplo:

  DELETE FROM Alunos
  WHERE codigo=1

SQL
Comandos DML
(Select)

Consulta de dados

- SELECT – seleciona um conjunto de atributos de uma(s) tabela(s) dada(s) pelo comando FROM.
- Sintaxe

select lista_atributos

from lista_tabelas

[**where** condição]

[**order by** nome_atributo 1 [desc] {[, nome_atributo n [desc]]}]

[**group by** lista_atributos_agrupamento

[**having** condição_para_agrupamento]]

Exemplo

```
SELECT Codigo, Nome
FROM Alunos
```

```
SELECT *
FROM Alunos
```

Select

- O comando SELECT permite incluir expressões aritméticas e modificar o nome dos atributos. Uma expressão pode ser uma combinação de valores, operadores e funções que produzem um valor. Os operadores aritméticos que podemos incluir são:

Operadores	**Descrição**
+	Soma
-	Subtração
*	Multiplicação
/	Divisão
+	Soma

Exemplo

```
SELECT Codigo, Nome, Mensalidade * 12
FROM Alunos
```

Select (Where)

- O comando WHERE é utilizado a seguir ao comando FROM e contém uma(s) condição(s) que as tuplas têm que satisfazer para que sejam visualizados. A palavra WHERE deverá possuir três elementos:
  1. O nome do atributo.
  2. O operador de comparação.
  3. O nome de um atributo, uma constante ou uma lista de valores.

Exemplo

```
SELECT Codigo, Nome
FROM Alunos
WHERE Nome = ‘Cleyton’

SELECT Codigo, Nome
FROM Alunos
WHERE Cod_Curso = 1
```

Select (Where)

- Os operadores de comparação podem ser divididos em duas categorias: **lógicos** e **SQL**.
- Os **operadores lógicos** testam as seguintes condições:

Operador	**Significado**
=	Igual a
>	Maior que
>=	Maior ou igual que
<	Menor que
<=	Menor ou igual que
<>	Diferente

Select (Where)

- Operadores SQL, existem quatro, que operam sobre todos os tipos de dados:

Operador	**Significado**
BETWEEN ..AND..	Entre dois valores
IN(lista)	Corresponde a qualquer valor da lista
LIKE	Cadeia de caracteres que satisfaz uma condição
IS NULL	É um valor nulo

Select (Where)

- Ainda sobre os operadores de comparação utilizados no comando WHERE, existem as respectivas expressões de negação:

Operador	**Significado**
NOT BETWEEN ..AND..	Não entre dois valores
NOT IN(lista)	Corresponde a nenhum valor da lista
NOT LIKE	Cadeia de caracteres que não satisfaz uma condição
IS NOT NULL	É um valor não nulo
<>	Diferente

Exemplo

```
SELECT Codigo, Nome
FROM Alunos
WHERE Cod_Curso IN (1, 3, 5)

SELECT Codigo, Nome
FROM Alunos
WHERE Nome LIKE 'C%';
```

Exemplo

```
SELECT Codigo, Nome
FROM Alunos
WHERE Cod_Curso IS NULL;

SELECT Codigo, Nome
FROM Alunos
WHERE Idade NOT BETWEEN 10 AND 18;
```

Exemplo

```
SELECT Codigo, Nome, idade
FROM Alunos
WHERE idade BETWEEN 20 AND 30
AND Cidade='Recife';
```

```
SELECT Codigo, Nome, idade
FROM Alunos
WHERE idade BETWEEN 20 AND 30
OR Cidade='Recife';
```

Select (Order By)

- O comando ORDER BY é utilizado para ordenar tuplas. Neste exemplo, As tuplas são ordenadas de forma ascendente pelo nome do aluno:
- Exemplo

```
SELECT Codigo, Nome, CPF
FROM Alunos
ORDER BY Nome
```

Select (Distinct e Where)

- A utilização da cláusula DISTINCT permite eliminar linhas repetidas

```
SELECT DISTINCT Nome, Cod_Curso
FROM Alunos
ORDER BY Nome
```

Select (Group by)

- Na linguagem SQL é possível obter resultados baseados em grupos de tuplas ao contrario daquilo que temos feito até agora.
- Assim, existem funções de grupo que operam sobre conjuntos de tuplas.
- O comando GROUP BY é utilizado para dividir as tuplas de uma tabela em grupos mais pequenos.

Select (Group by)

Função	**Valor Produzido**
AVG(n)	Valor médio de n
COUNT(expr)	Número de vezes que a expr toma um valor
MAX(expr)	Valor máximo de expr
MIN(expr)	Valor mínimo de expr
SUM(n)	Soma dos valores de n

Select (Group by)

- Expr indica os argumentos que podem ser do tipo CHAR, INT ou DATE.
- Todas as funções de grupo, à excepção de COUNT(*), ignoram os valores nulos.
- Torna-se ainda importante destacar que as funções de grupo por si só tratam todos as tuplas de uma tabela como um grupo.

Exemplo

```
SELECT AVG(idade)
FROM Alunos;

SELECT MIN(idade)
FROM Alunos
WHERE Cod_Curso = 1;

SELECT Cod_Curso, AVG(idade) AS MediaDeIdade
FROM Alunos
GROUP BY Cod_Curso;
```

Group by (Having)

- Há situações em que não só queremos poder filtrar os registros, mas também filtrar certos grupos baseando-nos nos valores dos totais gerados para cada grupo.
- Isto pode ser feito usando-se a cláusula HAVING. Esta cláusula nos permite informar critérios de filtragem baseados nos valores das próprias totalizações geradas para os grupos.

Exemplo

```
SELECT Cod_Curso, AVG(idade) AS
   MediaDeIdade
FROM Alunos
GROUP BY Cod_Curso;
ORDER BY MediaDeIdade

SELECT Cod_Curso, MAX(idade) AS MaximoIdade
FROM Alunos
GROUP BY Cod_Curso;
HAVING MAX(idade) >= 18
```