3. Fundamentos da SQL

O nome SQL significa *Structured Query Language* (Linguagem Estruturada de Pesquisa). A versão original foi desenvolvida pela IBM. Inicialmente, a linguagem foi denominada SEQUEL (*Structured English Query Language*), implementada como parte do projeto do *Sistema R* no início do anos 70. Em 1975, foi implementado um protótipo de aplicação dessa nova linguagem. Entre 1976 e 1977, a linguagem foi revisada e ampliada.

Devido ao sucesso dessa nova forma de consulta e manipulação de dados, dentro de um ambiente de banco de dados, a utilização da SQL foi se tornando cada vez maior. Desta forma, a SQL tornou-se um padrão de fato no ambiente dos bancos de dados realacionais.

Em 1986, o *American National Standards Institute* (ANSI) e a *International Standards Organization* (ISO), publicaram os padrões para a SQL, chamada SQL-86. Uma extensão para o padrão SQL, a SQL-89, foi publicada em 1989, e um sistema de banco de dados típico atualmente dá suporte ao menos aos recursos dispostos na SQL-89. A versão em uso do padrão ANSI/ISO SQL é o padrão SQL-92.

A linguagem SQL possui diversas partes:

- **Linguagem de definição de dados (*Data Definition Language* - DDL)**: proporciona comandos para a definição de esquemas de relações, exclusão de relações, criação de índices e modificação nos esquemas de relações.
- **Linguagem interativa de manipulação de dados (*Data-manipulation Language* - DML)**: abrange uma linguagem de consulta baseada tanto na álgebra relacional quanto no cálculo relacional de tuplas. Engloba também comandos para inserção, exclusão e modificação de tuplas no banco de dados.
- **Incorporação DML (*Embedded* DML)**: a forma de comandos SQL incorporados foi projetada para aplicação em linguagens de programação de uso geral, como *PL/I, Cobol, Pascal, Fortran e C.*
- **Definição de visões**: a SQL DDL possui comandos para a definição de visões.
- **Autorização**: a SQL DDL engloba comandos para especificação de direitos de acesso a relações e visões.
- **Integridade**: a SQL DDL possui comandos para especificação de regras de integridade que os dados que serão armazenados no banco de dados devem satisfazer. Atualizações que violarem as regras de integridade serão desprezadas.
- **Controle de transações**: a SQL inclui comandos para a especificação de iniciação e finalização de transações. Algumas implementações também permitem explicitar bloqueios de dados para controle de concorrência.

3.1. Vantagens e desvantagens da SQL

Com o uso e a padronização da SQL, algumas vantagens são diretas:

- **Independência de fabricante** – a SQL é oferecida em praticamente todos os SGBDs;
- **Portabilidade entre computadores** – a SQL pode ser utilizada desde um computador pessoal, passando por uma estação de trabalho, até um computador de grande porte;
- **Redução dos custos com treinamento** – as aplicações podem se movimentar de um ambiente para outro sem que seja necessária uma reciclagem da equipe de desenvolvimento;
- **Inglês estruturado de alto nível** – a SQL é formada por um conjunto bem simples de sentenças em inglês, oferecendo um rápido e fácil entendimento;
- **Consulta interativa** – a SQL provê um acesso rápido aos dados, fornecendo respostas ao usuário, a questões complexas, em minutos ou segundos;
- **Múltiplas visões dos dados** – a SQL permite ao criador do banco de dados levar diferentes visões dos dados a diferentes usuários;
- **Definição dinâmica dos dados** – por meio da SQL, podem-se alterar, expandir ou incluir, dinamicamente, as estruturas dos dados armazenados com a máxima flexibilidade;

Apesar de todas essas vantagens, algumas críticas são dirigidas à SQL:

- A padronização leva à uma natural inibição da criatividade, pois quem desenvolve aplicações fica preso a soluções padronizadas, não podendo sofrer melhorias ou alterações;
- A SQL está longe de ser uma linguagem relacional ideal (*C.J. Date*), e algumas críticas são feitas à linguagem:
  - Falta de ortogonalidade nas expressões, funções embutidas, variáveis indicadoras, referência a dados correntes, constante NULL, conjuntos vazios, etc.;
  - Definição formal da linguagem após sua criação;
  - Discordância com as linguagens hospedeiras;
  - Falta de algumas funções;
  - Erros (valores nulos, índices únicos, cláusula FROM, etc.);
  - Não dá suporte a alguns aspectos do modelo relacional (atribuição de relação, join explícito, domínios, etc.).

3.2. Linguagem de Definição de Dados

O conjunto das relações em um banco de dados deve ser especificado para o sistema por meio de uma linguagem de definição de dados (DDL). A SQL DDL permite não só a especificação de um conjunto de relações, como também informações acerca de cada uma das relações, incluindo:

- O esquema de cada relação.
- O domínio dos valores associados a cada atributo.
- As regras de integridade.
- O conjunto de índices para manutenção de cada relação.
- Informações sobre segurança e autoridade sobre cada relação.
- A estrutura de armazenamento físico de cada relação no disco.

3.2.1. Tipos de domínios em SQL

O padrão SQL-92 aceita uma variedade de tipos de domínios embutidos, incluindo os seguintes:

- ***char(n)*** é uam cadeia de caracteres de tamanho fixo, com o tamanho n definido pelo usuário. A forma completa, ***character***, também pode ser usada.
- ***varchar(n)*** é uma cadeia de caracteres de tamanho variável, com o tamanho *n* máximo definido pelo usuário. A forma completa, ***character varying***, é equivalente.
- ***int*** é um inteiro (um subconjunto finito dos inteiros que depende do equipamento). A forma completa, ***integer***, também é válida.
- ***smallint*** é um inteiro pequeno (um subconjunto do domínio dos inteiros).
- ***numeric(p,d)*** é um número de ponto fixo cuja precisão é definida pelo usuário. O número consiste de *p* dígitos (mais o sinal), sendo que *d* dos *p* dígitos estão à direita do ponto decimal. Assim, *numeric(3,1)* permite que 44,5 seja armazenado de modo exato, mas nem 444,5 nem 0,32 podem ser armazenados corretamente em um campo deste tipo.
- ***real, double precision*** são números de ponto flutuante e ponto flutuante de precisão dupla cuja precisão é dependente do equipamento.
- ***float(n)*** é um número de ponto flutuante com a precisão definida pelo usuário em pelo menos *n* dígitos.
- ***date*** é um calendário contendo um ano (com quatro dígitos), mês e dia do mês.

- ***time*** representa horário, em horas, minutos e segundos.

A SQL permite operações aritméticas e comparações em vários domínios numéricos e operações de comparação em todos os domínios relacionados. A SQL fornece também um tipo de dado chamado ***interval***, que permite processamento de datas e horários em intervalos.

A SQL permite que a declaração de domínio de um atributo inclua a especificação de ***not null***, proibindo, assim, a inserção de valores nulos para esse tipo de atributo. Qualquer modificação que possa resultar na inserção de um valor nulo em um domínio *not null* gera um diagnóstico de erro. Há muitas situações em que a proibição de valores nulos é desejável. Um caso em particular no qual é imprescindível a proibição de valores nulos é a chave primária de um esquema de relação.

3.3. Definição de Dados Utilizando SQL

3.3.1. Comando CREATE TABLE

O comando **create table** permite ao usuário criar uma nova tabela (ou relação). Para cada atributo da relação é definido um nome, um tipo, máscara e algumas restrições. Os tipos de uma coluna são:

- **char(*n*)**: caracteres e strings onde ***n*** é o número de caracteres;
- **integer**: inteiros
- **float**: ponto flutuante;
- **decimal(*m*,*n*)**: onde ***m*** é o número de casas inteiras e ***n*** o número de casas decimais.

A restrição **not null** indica que o atributo deve ser obrigatoriamente preenchido; se não for especificado, então o "default" é que o atributo possa assumir o valor nulo.

A forma geral do comando **create table** então é:

create table <nome_tabela> (*<nome_coluna$_1$>* *<tipo_coluna$_1$>* *<NOT NULL>,*
<nome_coluna$_2$> *<tipo_coluna$_2$>* *<NOT NULL>,*
:
<nome_coluna$_n$> *<tipo_coluna$_n$>* *<NOT NULL>);*

Por exemplo, para criar a tabela EMPREGADOS do apêndice A, teríamos o seguinte comando:

```
create table EMPREGADOS    (  nome          char (30)        NOT NULL,
                              rg            integer          NOT NULL,
                              cic           integer,
                              depto         integer          NOT NULL,
                              rg_supervisor integer,
                              salario,      decimal (7,2)    NOT NULL );
```

3.3.2. Comando DROP TABLE

O comando **drop table** permite a exclusão de uma tabela (relação) em um banco de dados.

A forma geral para o comando **drop table** é:

drop table <nome_tabela>;

Por exemplo, para eliminar a tabela EMPREGADOS do apêndice A teríamos o seguinte comando:

drop table EMPREGADOS;

Observe que neste caso, a chave da tabela EMPREGADOS, (rg) é utilizada como chave estrangeira ou como chave primária composta em diversos tabelas que devem ser devidamente corrigidas.

Este processo não é assim tão simples pois, como vemos neste caso, a exclusão da tabela EMPREGADOS implica na alteração do projeto físico de diversas tabelas. Isto acaba implicando na construção de uma nova base de dados.

3.3.3. Comando ALTER TABLE

O comando **alter table** permite que o usuário faça a inclusão de novos atributos em uma tabela. A forma geral para o comando **alter table** é a seguinte:

alter table <nome_tabela> add <nome_coluna> <tipo_coluna>;

No caso do comando **alter table**, a restrição NOT NULL não é permitida pois assim que se insere um novo atributo na tabela, o valor para o mesmo em todas as tuplas da tabela receberão o valor NULL.

3.4. Consultas em SQL

3.4.1. O comando SELECT

O comando **select** permite a seleção de tuplas e atributos em uma ou mais tabelas. A forma básica para o uso do comando **select** é:

select *<lista de atributos>*
from *<lista de tabelas>*
where *<condições>;*

Por exemplo, para selecionar o nome e o rg dos funcionários que trabalham no departamento número 2 na tabela EMPREGADOS utilizamos o seguinte comando:

select nome, rg

from EMPREGADOS
where depto = 2;

obteremos então o seguinte resultado:

Nome	***RG***
Fernando	*20202020*
Ricardo	*30303030*
Jorge	*40404040*

A consulta acima é originária da seguinte função em álgebra relacional:

$$\pi_{\text{nome, rg}} (\sigma_{\text{depto = 2}}^{\text{(EMPREGADOS)}});$$

Em SQL também é permitido o uso de condições múltiplas. Veja o exemplo a seguir:

select nome, rg, salario
from EMPREGADOS
where depto = 2 AND salario > 2500.00;

que fornece o seguinte resultado:

Nome	***RG***	***Salário***
Jorge	*40404040*	*4.200,00*

e que é originária da seguinte função em álgebra relacional:

$$\pi_{\text{nome, rg, salario}} (\sigma_{\text{depto = 2 .and. salario > 3500.00}}^{\text{(EMPREGADOS)}});$$

A operação *select-from-where* em SQL pode envolver quantas tabelas forem necessárias. Leve em consideração a seguinte consulta:

selecione o número do departamento que controla projetos localizados em Rio Claro:

select t1.numero_depto
from departamento_projeto t1, projeto t2
where t1.numero_projeto = t2.numero;

Na expressão SQL acima, *t1* e *t2* são chamados *"alias"* (apelidos) e representam a mesma tabela a qual estão referenciando. Um "alias" é muito importante quando há redundância nos nomes das colunas de duas ou mais tabelas que estão envolvidas em uma expressão. Ao invés de utilizar o "alias", é possível utilizar o nome da tabela, mas isto pode ficar cansativo em consultas muito complexas além do que, impossibilitaria a utilização da mesma tabela mais que uma vez em uma expressão SQL. Considere a seguinte consulta:

selecione o nome e o rg de todos os funcionários que são supervisores;
select e1.nome, e1.rg

```
from empregado e1, empregado e2
where e1.rg = e2.rg_supervisor;
```

que gera o seguinte resultado:

Nome	***RG***
João Luiz	*10101010*
Fernando	*20202020*

A consulta acima é decorrente da seguinte expressão em álgebra relacional:

$\pi_{nome, rg}(\quad EMPREGADOS \ _{tg\_t1 = rg\_supervisor\_t2} \ EMPREGADOS);$

O operador ***** dentro do especificador *select* seleciona todos os atributos de uma tabela, enquanto que a exclusão do especificador *where* faz com que todas as tuplas de uma tabela sejam selecionadas. Desta forma, a expressão:

```
select *
from empregados;
```

gera o seguinte resultado:

Nome	***RG***	***CIC***	***Depto.***	***RG Supervisor***	***Salário***
João Luiz	*10101010*	*11111111*	*1*	***NULO***	*3.000,00*
Fernando	*20202020*	*22222222*	*2*	*10101010*	*2.500,00*
Ricardo	*30303030*	*33333333*	*2*	*10101010*	*2.300,00*
Jorge	*40404040*	*44444444*	*2*	*20202020*	*4.200,00*
Renato	*50505050*	*55555555*	*3*	*20202020*	*1.300,00*

Diferente de álgebra relacional, a operação *select* em SQL permite a geração de tuplas duplicadas como resultado de uma expressão. Para evitar isto, devemos utilizar o especificador **distinct**. Veja a seguir os exemplos com e sem o especificador **distinct**.

```
select depto
from empregado;
```

```
select distinct depto
from empregado;
```

que gera os seguintes resultados:

Depto.
1
2
2
2
3

Depto.
1
2
3

Podemos gerar consultas aninhadas em SQL utilizando o espeficicador **in**, que faz uma comparação do especificador **where** da consulta mais externa com o resultado da consulta mais interna. Considere a consulta a seguir:

selecione o nome de todos os funcionários que trabalham em projetos localizados em Rio Claro;

```
select e1.nome, e1.rg, e1.depto
from empregado e1, empregado_projeto e2
where e1.rg = e2.rg_empregado
    and e2.numero_projeto in (  select numero
                                from projeto
                                where localizacao = 'Rio Claro');
```

Para selecionar um conjunto de tuplas de forma ordenada devemos utilizar o comando **order by**. Leve em consideração a seguinte consulta:

selecione todos os empregados por ordem alfabética:

```
select nome, rg, depto
from empregado
order by nome;
```

3.4.2. Inserções e Atualizações

Para elaborar inserções em SQL, utiliza-se o comando **insert into**. A forma geral para o comando **insert into** é:

```
insert into <nome da tabela> <(lista de colunas)>
values      <(lista de valores)>;
```

Considere a seguinte declaração:

insira na tabela empregados, os seguintes dados:

nome: *Jorge Goncalves*

rg: *60606060*

cic: *66666666*

departamento: *3*

rg_supervisor: *20202020*

salário: *R$ 4.000,00*

```
insert into     empregados
values      ('Jorge Goncalves', '60606060', '66666666', 3, '20202020', 4000,00);
```

ou ainda:

insira na tabela empregados os seguintes dados:

nome: *Joao de Campos*

rg: *70707070*

cic: *77777777*

departamento: *3*

salário: *R$2.500,00*

```
insert into     empregados (nome, rg, cic, depto, salario)
```

values ('Joao de Campos, '70707070', '77777777', 3, 2500,00);

Como na primeira inserção todos os campos foram inseridos, então não foi necessário especificar o nome das colunas. Porém, na segunda inserção, o campo *rg_supervisor* não foi inserido, então especificou-se as colunas. Outra forma de se elaborar esta inserção seria:

```
insert into    empregados
values         ('Joao de Campos, '70707070', '77777777', 3, '', 2500,00);
```

Neste caso, utilizou-se os caracteres **''** para se declarar que um valor nulo seria inserido nesta coluna.

Para se efetuar uma alteração em uma tabela, é utilizado o comando **update**. A forma geral do comando **update** é:

```
update <tabela>
set    <coluna> = <expressão>
where  <condição>
```

Considere a seguinte declaração:

atualize o salário de todos os empregados que trabalham no departamento 2 para R$ 3.000,00;

```
update empregado
set    salario = 3.000,00
where  depto = 2;
```

Para se eliminar uma tupla de uma tabela, utiliza-se o comando **delete**. A forma geral do comando **update** é:

```
delete from   <tabela>
where         <condição>;
```

Leve em consideração a seguinte expressão:

elimine os registros nos quais o empregado trabalhe no departamento 2 e possua salário maior que R$ 3.500,00;

```
delete from   empregado
where  salario > 3.500,00 and depto = 2;
```

Nos casos de atualização que foram vistos, todas as *<condições>* podem ser uma consulta utilizando o comando **select**, onde o comando será aplicado sobre todos os registros que satisfizerem as condições determinadas pelo comando de seleção.