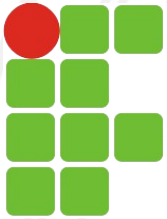

INSTITUTO FEDERAL
PIAUÍ
Campus Parnaíba

SQL

Prof. Msc Denival A. dos Santos

Histórico

- A versão original foi desenvolvida pela IBM no laboratório de pesquisa de San José;
- Originalmente chamada de **Sequel**, foi implementada como parte do projeto do **Sistema R** no início dos anos 70;
- Inúmeros produtos dão suporte atualmente para a linguagem SQL;
- SQL – Linguagem de consulta estruturada;
- A linguagem SQL pode ser considerada uma das maiores razões para o sucesso dos bancos de dados relacionais no mundo comercial;
- Um esforço conjunto da ANSI (American National Standards Institute – Instituto nacional americano de padrões) e a ISO (International Standards Organization – Organização Internacional de padrões) chegou a versão-padrão da SQL (ANSI, 1986), chamada SQL-86 ou SQL1;
- Como a especificação do padrão SQL está em expansão, com mais funcionalidades a cada versão, o último padrão é a **SQL-99 e SQL-2003**.

Partes da Linguagem SQL

- **Linguagem e definição de dados (DDL)** – proporcional comandos para a definição de esquemas de relação, exclusão de relações, criação de índices e modificação nos esquemas de relações;
- **Linguagem interativa de manipulação de dados (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** – foi projeta para aplicações e linguagens de programação de uso geral, como PL/I, Delphi, Java, C#, C++, etc.
- **Definição de visões** – comandos para definição de visões;
- **Autorização** – comandos para especificação de direitos de acesso a relações e visões;
- **Integridade** – comandos para especificação de regras de integridade que os dados que serão armazenados no banco de dados devem satisfazer;
- **Controle de transações** – comandos para a especificação de inicialização e finalização de transações.

DDL - 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 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, etc.
- Principais comandos
  - **Create** – criar base de dados ou tabelas;
  - **Drop** – remove base de dados ou tabelas;
  - **Alter** – altera a estrutura de uma tabela existente.

Base de Dados

- Para criarmos uma base de dados no MySQL Server, utilizamos:

```
mysql> CREATE DATABASE livraria;
Query OK, 1 row affected (0.02 sec)
```

- Para selecionar uma bases de dados existente, utilizamos USE:

```
mysql> USE livraria;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
```

Base de Dados

- Para listar as bases de dados existentes, utilizamos:

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| livraria           |
| mysql              |
| test               |
+--------------------+
4 rows in set (0.03 sec)
```

- Para remover uma base de dados existente, utilizamos:

```
mysql> DROP DATABASE livraria;
Query OK, 0 rows affected (0.08 sec)
```

Definição de esquema em SQL

- Definimos uma relação SQL usando o comando CREATE TABLE.
- Sua sintaxe é:

```
CREATE TABLE nome_tabela (
  nome_atributo_1 tipo_1 [[NOT]NULL][UNIQUE]
  [{, nome_atributo_n    tipo_n}]
  [, PRIMARY KEY (nome(s)_atributo(s))]
  [{, FOREIGN KEY (nome_atributo)
        REFERENCES nome_tabela}]
)
```

Tipos de domínios em SQL

- O padrão SQL-92 aceita uma variedade de tipos de domínios embutidos, incluindo os seguintes:
  - **Char(n)** – é uma cadeia de caracteres de tamanho fixo, com tamanho n definido pelo usuário;
  - **Varchar(n)** – é uma cadeia de caracteres de tamanho variável, com o tamanho n máximo definido pelo usuário;
  - **Int** – é um número inteiro;
  - **Smallint** – é um número inteiro pequeno;
  - **Numeric(p,d)** – é um número de ponto fixo cuja precisão é definida pelo usuário;
  - **Date** – é um calendário contendo um ano (dia, mês e ano);
  - **Time** – representa horário (hora, minuto e segundo)

Obs.: **Timestamp** – engloba os campos DATE e TIME

Varchar2(n) – utilizado em ORACLE

Definição de esquema em SQL

- Para criarmos uma tabela no MySQL Server, utilizamos:

```
MariaDB [livraria]> create table livro(
    ->  idLivro int not null,
    -> titulo varchar(100),
    -> preco numeric(9,2),
    -> primary key(idLivro)
    -> );
Query OK, 0 rows affected (0.44 sec)
```

- Para listar as tabelas existentes, utilizamos:

```
mysql> SHOW TABLES;
+---------------------+
| Tables_in_livraria |
+---------------------+
| Livro               |
+---------------------+
1 row in set (0.00 sec)
```

- Para remover uma tabela existente, utilizamos:

```
mysql> DROP TABLE Livro;
Query OK, 0 rows affected (0.00 sec)
```

Exemplo

```
MariaDB [livraria]> create database hospital;
Query OK, 1 row affected (0.00 sec)

MariaDB [livraria]> use hospital;
Database changed
MariaDB [hospital]> create table ambulatorio(
    -> idAmbulatorio int not null,
    -> andar int not null,
    -> capacidade int,
    -> primary key(idAmbulatorio)
    -> );
Query OK, 0 rows affected (0.26 sec)

MariaDB [hospital]> create table medico(
    -> idMedico int not null,
    -> nome varchar(70) not null,
    -> idade int,
    -> especialidade varchar(40),
    -> idAmbulatorio int not null,
    -> primary key(idMedico),
    -> foreign key(idAmbulatorio) references ambulatorio(idAmbulatorio)
    -> );
Query OK, 0 rows affected (0.21 sec)

MariaDB [hospital]> show tables;
+--------------------+
| Tables_in_hospital |
+--------------------+
| ambulatorio        |
| medico             |
+--------------------+
2 rows in set (0.00 sec)
```

Restrições

- Foram pensadas para permite definir restrições em colunas de tabelas.
- As restrições são úteis para impedir o armazenamento de dados que possam gerar erros.
- Exemplos:
  - Uma coluna contendo preços de produtos provavelmente só poderá aceitar valores positivos.
  - Em uma tabela que possua informações sobre produtos, o campo nome do produto não poderá aceitar valor NULL.
  - Se for um cadastro de uma pessoa, por exemplo, a idade não poderá ser nula, e dependendo do caso não poderá ser menor que 18 ou 23.

Restrições

- Restrição de valores nulos
  - 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 atributo;
  - Qualquer modificação que possa resultar na inserção de um valor nulo em um domínio 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 uma **chave primária** de um esquema de relação.

```
Create table Pessoa(
    cpf varchar(11) not null, // Não aceita valor nulo.
    nome varchar(50) null,    // Aceita valor nulo.
    endereco varchar(50),     // Por default aceita valor nulo.
    primary key(cpf)
)
```

Restrições

- Restrição de valores duplicados
  - Há situações onde o valor armazenado em um campo deve ter um registro em relação aos outros registros da tabela.
  - Para isso utilizamos a cláusula **UNIQUE**.

```
Create table Pessoa(
   cpf varchar(11) unique not null, // O numero de cpf não deve se repetir.
   nome varchar(50) null,
   endereco varchar(50),
   primary key(cpf)
)
```

Restrições

- Restrição de verificação
  - Há situações onde o valor de um campo deve obedecer a uma regra.
  - Para que o valor desse campo fique restrito a um conjunto de valores, utiliza-se a cláusula **CHECK**.

```
Create table Pessoa(
   cpf varchar(11) not null,
   nome varchar(50) null,
   endereco varchar(50),
   idade int check(idade > 0 && idade < 18),  // A idade só pode ser entre 1 e 18.
   sexo varchar(1) check (sexo in ("M", "F")),// Obriga usar M ou F
   primary key(cpf)
)
```

Restrições

- Restrição de chave primária
  - A chave primária tem como função identificar univocamente uma linha do registro da tabela.
  - Toda tabela deve possuir um campo chave, e quando ele é definido, ficam implícitas as cláusulas **UNIQUE** e **NOT NULL** para este campo, não sendo necessário a especificação delas.
  - Da mesma forma, a cláusula NULL fica implícita quando não se digita nada.

```
Create table Pessoa(
    cpf varchar(11) not null,
    nome varchar(50) null,
    endereco varchar(50),
    primary key(cpf) // Define o cpf como chave primária
)
```

Restrições

- Restrição de chave estrangeira
  - A chave estrangeira *(Foreign key)* especifica que o valor do atributo deve corresponder a algum valor existente em um atributo de outra entidade. A chave estrangeira mantém a integridade referencial entre duas entidades relacionadas. Ela é a chave de uma relação 1 para muitos onde precisa-se de uma chave de identificação da tabela pai na tabela filho.
  - **Observação:** No MySQL uso de FOREIGN KEYS só é suportado pelo *engine InnoDB*.

```
Create table Pai(
   id_pai int not null,
   nome varchar(50) null,
   primary key(cpf)
)
```

```
Create table Filho(
   id_filho int not null,
   nome varchar(50) null,
   id_pai int,
   primary key(id_filho),
   foreign key(id_pai) references Pai(id_pai)
)
```

Restrições

- Restrição de valores padrão
  - Pode-se declarar, ao criar uma tabela, que determinado campo já venha com um valor padrão, isso significa que, se nada for inserido nesse campo, ele terá aquele valor padrão armazenado - **DEFAULT**.

```
-- Criação da tabela
Create table Cidade(
   id_cidade int not null,
   nome varchar(50) null,
   uf varchar(2) Default "PI",
   primary key(id_cidade)
)
-- Povoa a tabela
insert into Cidade values(1,"Timon","MA");
insert into Cidade(id_cidade, nome) values(2,"Parnaíba");
-- Saída
```

id_cidade	nome	uf
1	Timon	MA
2	Parnaíba	PI

Comando Alter Table

- As definições de uma tabela básica ou de outros elementos do esquema que possuírem denominação poderão se alteradas pelo comando **ALTER**. Para as tabelas básicas, as ações de alteração possíveis compreendem: adicionar ou remover uma coluna(atributo), alterar a definição de uma coluna e adicionar ou eliminar restrições na tabela.
- Sintaxe:

```
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 de alteração de Tabelas

```
ALTER TABLE Ambulatórios
ADD nome VARCHAR(30)

ALTER TABLE Médicos
DROP PRIMARY KEY

ALTER TABLE Pacientes
DROP COLUMN doenca

ALTER TABLE Funcionários
ADD FOREIGN KEY (nroa) REFERENCES Ambulatórios

ALTER TABLE Funcionarios
ADD CONSTRAINT  fk_nroa FOREIGN KEY(nroa) REFERENCES Ambulatorios(nroa)

ALTER TABLE  Cidade
DROP FOREIGN KEY fk_cliente

ALTER TABLE  Pacientes
MODIFY COLUMN nome_pac varchar(50)
```

DML – Linguagem de manipulação de dados

- É um subconjunto de instruções SQL que é utilizada para realizar inserções, alterações, exclusões e extração de dados presentes em registros de uma tabela.
- Principais comandos
  - **Insert** – inserir novo registro em uma tabela.
  - **Update** – atualizar registros existentes em uma tabela.
  - **Delete** – exclusão de registros existentes.
  - **Select** – realizar a busca de dados em tabelas.

Comando INSERT

- Em sua forma mais simples, o **INSERT** é usado para adicionar uma única tupla em uma relação;
- Uma segunda forma do comando INSERT permite ao usuário especificar explicitamente os nomes dos atributos que receberão os valores fornecidos por esse comando. Essa maneira é útil se uma relação possuir muitos atributos, mais somente para alguns poucos serão designados valores na nova tupla. Desta forma os atributos não relacionados serão registrados com valor NULL ou DEFAULT;
- Sintaxe quando se conhece a ordem dos atributos e deseja atribuir valor a todos.

  INSERT INTO nome_tabela

  VALUES (lista_valores_atributos) [, (lista_valores_atributos)]

- Sintaxe quando não se conhece a ordem dos atributos ou não deseja atribuir valor a todos.

  INSERT INTO nome_tabela [(lista_atributos)]

  VALUES (lista_valores_atributos) [, (lista_valores_atributos)]

Exemplo de inserção de tuplas

```
INSERT INTO Cidade
VALUES (1, “TERESINA”)

INSERT INTO Ambulatorios
VALUES (1, 1, 30)

INSERT INTO Medicos (codm, nome, idade,  especialidade, CPF, cidade)
VALUES (4, “Carlos”, 28,”ortopedia”,  11000110000, “Joinville”)

INSERT INTO Cidade(codigo, nome)
VALUES (34, “PARNAIBA”)
```

Comando DELETE

- O comando **DELETE** remove tuplas de uma relação. Se incluir a cláusula **WHERE**, similar à que é usada nas consultas SQL, serão selecionadas as tuplas que serão deletadas;
- As tuplas serão explicitamente removidas de uma única tabela de cada vez. Entretanto, as remoções poderão propagar-se nas tuplas de outras relações, se forem definidas ações engatilhadas nas restrições de integridade;
- **Obs.:** A omissão da cláusula WHERE determina que todas as tuplas da relação serão excluídas, entretanto, a definição da tabela vazia, permanecerá no banco de dados.
- Sintaxe:

DELETE FROM nome_tabela

[**WHERE** condição]

Operadores

Tipo	Operador	Descrição
Operadores lógicos	AND	Conjunção lógica
	OR	Disjunção lógica
	NOT	Negação lógica
Operadores de comparação	=	Igual a
	< >	Diferente de
	>	Maior que
	<	Menor que
	>=	Maior ou igual a
	<=	Menor ou igual a
	BETWEEN	Especifica um intervalo de valores
	LIKE	Especifica um padrão de comparação
	IN	Especifica registros dentro de um banco de dados

Exemplos de exclusões

```
DELETE FROM Ambulatorios

DELETE FROM Medicos
WHERE especialidade = “cardiologia”
          OR cidade < > “Florianopolis”

DELETE FROM Cidade
WHERE nome = “Parnaiba”

DELETE FROM Empregado
WHERE departamento = “010” AND salario < 150.00
```

Comando UPDATE

- O comando **UPDATE** é usado para modificar os valores dos atributos de uma ou mais tuplas;
- Como o comando **DELETE**, as cláusulas **WHERE** em um comando **UPDATE** seleciona as tuplas de uma única relação que serão modificadas. Entretanto uma atualização no valor da chave primária pode propagar-se para os valores das chaves estrangeiras, nas tuplas de outras relações, se essa ação engatilhada for especificada nas restrições de integridade referencial;
- Sintaxe:

```
UPDATE nome_tabela
SET nome_atributo_1 = Valor
    [{, nome_atributo_n = Valor}]
[WHERE condição]
```

Exemplo de atualizações

```
UPDATE Medicos
SET cidade = “Florianopolis”;

UPDATE Ambulatorios
SET capacidade = capacidade + 5,
      andar = 3
WHERE nroa = 2

UPDATE Cidade
SET nome = “Timon City”,
      populacao = 120000
WHERE codigo = 12
```

Exercício - Popular as tabelas

Ambulatorios

nroa	andar	capacidade
1	1	30
2	1	50
3	2	40
4	2	25
5	2	55

Medicos

codm	nome	idade	especialidade	CPF	cidade	nroa
1	Joao	40	ortopedia	10000100000	Florianopolis	1
2	Maria	42	traumatologia	10000110000	Blumenau	2
3	Pedro	51	pediatria	11000100000	São José	2
4	Carlos	28	ortopedia	11000110000	Joinville	
5	Marcia	33	neurologia	11000111000	Biguacu	3

Pacientes

codp	nome	idade	cidade	CPF	doenca
1	Ana	20	Florianopolis	20000200000	gripe
2	Paulo	24	Palhoca	20000220000	fratura
3	Lucia	30	Biguacu	22000200000	tendinite
4	Carlos	28	Joinville	11000110000	sarampo

Funcionarios

codf	nome	idade	cidade	salario	CPF
1	Rita	32	Sao Jose	1200	20000100000
2	Maria	55	Palhoca	1220	30000110000
3	Caio	45	Florianopolis	1100	41000100000
4	Carlos	44	Florianopolis	1200	51000110000
5	Paula	33	Florianopolis	2500	61000111000

Consultas

codm	codp	data	hora
1	1	2006/06/12	14:00
1	4	2006/06/13	10:00
2	1	2006/06/13	9:00
2	2	2006/06/13	11:00
2	3	2006/06/14	14:00
2	4	2006/06/14	17:00
3	1	2006/06/19	18:00
3	3	2006/06/12	10:00
3	4	2006/06/19	13:00
4	4	2006/06/20	13:00
4	4	2006/06/22	19:30

Comando SELECT - Estrutura Básica

- Um banco de dados relacional consiste de uma coleção de relações, cada uma designada por um único nome;
- A SQL permite o uso de valores nulos para indicar valores desconhecidos ou inexistentes;
- A estrutura básica de uma expressão em SQL consiste em três cláusulas: **Select**, **From** e **Where**.
  - A cláusula **SELECT** corresponde à operação de projeção da álgebra relacional. Ela é usada para relacionar os atributos desejados no resultado de uma consulta;
  - A cláusula **FROM** corresponde à operação de produto cartesiano da álgebra relacional. O comando FROM define que tabelas serão utilizadas em uma consulta, ou seja, de quais tabelas devemos buscar os dados;
  - A cláusula **WHERE** corresponde à seleção do predicado da álgebra relacional. O comando WHERE sempre estará associado a ele uma condição que determina quais registros deverão ser retornados pela consulta.

Comando SELECT

- O resultado de uma consulta de SQL é, naturalmente, uma relação. Consideremos uma consulta simples:

  "*Selecione todas as informações da tabela medico*"

```
Select *
From medico
```

- O resultado é uma relação consistindo de um atributo simples intitulado *nome_medico*;
- A SQL permite duplicidade nas relações, assim como no resultado de expressões SQL. Dessa forma, a consulta precedente listará uma vez cada *nome_medico* em todas as tuplas nas quais ela aparece dentro da relação medico;

Cláusula FROM

- A cláusula **FROM** por si só define um produto cartesiano das relações na cláusula. Esta é uma cláusula obrigatória na expressão **SELECT**;
- Especifica as tabelas das quais as outras cláusulas da consulta podem acessar as colunas a serem utilizadas nas expressões;
- Sintaxe:

  SELECT * FROM nome_das_Tabelas
- Exemplos:

  SELECT cod_especialidade, nome_especialidade

  FROM especialidade

  SELECT * FROM consulta

Cláusula WHERE

- A cláusula **WHERE** é uma parte opcional das expressões de (Select, Delete e Update). Ela permite selecionar linhas baseadas em uma expressão booleana, ou seja, somente as linhas para as quais a expressão é avaliada como **TRUE** são retornadas no resultado, ou no caso da instrução **DELETE**, excluídas ou no caso da instrução **UPDATE** atualizadas;
- Sintaxe: **WHERE expressão_booleana**;
- Exemplo:

```
SELECT * FROM medico
WHERE cod_especialidade = 2

SELECT * FROM doenca
WHERE cod_doenca = 1
```

Distinct e All

- Nos casos em que desejamos forçar a eliminação de duplicidade, podemos inserir a palavra chave **DISTINCT** depois do **SELECT**;
- Podemos reescrever a consulta anterior da seguinte forma caso desejemos suprimir duplicidades:

```
Select distinct nome_medico
From medico
```

- Note que a SQL nos permite usar a palavra-chave **ALL** para especificar explicitamente que as duplicidades não serão eliminada:

```
Select All nome_medico
From medico
```

- Uma vez que a manutenção de duplicidade é padrão, não usaremos **ALL** em nossos exemplos. Para assegurar a eliminação da duplicidade em nossas consultas de exemplo, usaremos **DISTINCT** sempre que for necessário.

Operação de renomeação AS

- A SQL proporciona um mecanismo para rebatizar tanto relações quanto atributos, usando a cláusula **AS**. As cláusulas AS pode aparecer tanto na cláusula SELECT quanto na cláusula FROM;
- Os nomes dos atributos nos resultados são derivados dos nomes desses atributos nas relações indicadas pela cláusulas FROM;
- Sintaxe:

  nome_antigo AS novo_nome;
- Exemplos:

```
Select cod_paciente as paciente, cod_medico as medico
From consulta;

Select *
From consulta as TbConsultas
```

Ordenação e apresentação de Tuplas

- A SQL oferece ao usuário algum controle sobre a ordenação por meio da qual as tuplas de uma relação são apresentadas. A cláusula **ORDER BY** faz com que as tuplas do resultado de uma consulta apareçam em uma determinada ordem;
- Sintaxe: **ORDER BY campos [ASC/DESC];**
- Exemplos:

```
-- Ascendente
Select * from Vendedor
Order by vendedor asc
```

id_vendedor	vendedor
5	Cunha
1	Denival
4	Ely
2	Gerson
3	Regis

```
-- Descendente
Select * from Vendedor
Order by vendedor desc
```

id_vendedor	vendedor
3	Regis
2	Gerson
4	Ely
1	Denival
5	Cunha

Comparações envolvendo NULL

- A SQL permite o uso do valor **NULL** (nulo) para indicar a ausência de informação sobre o valor de um atributo;
- Usamos a palavra especial **NULL** como predicado para testar a existência de valores nulos;
- O predicado **IS NOT NULL** testa a ausência de valores nulos;
- A existência de valores nulos complica o processamento de operadores agregados;
- **Observação:** Em geral, as funções podem tratar nulos usando a seguinte regra: todas as funções agregadas, excerto COUNT(*), ignoram os valores nulos de seus conjuntos de valores de entrada;
- Exemplo:

```
Select *
From pacientes
Where idade is null
```

Ligação entre tabelas - JOIN

- Consultas que envolvam mais de uma tabela que estejam relacionadas (se tiverem campos comuns – em uma tabela chave primária e em outra chave estrangeira), utilizam como soluções a utilização de junções (**JOIN**);
- O feito do JOIN é a criação de uma tabela temporária em que cada par de linhas (de tabelas diferentes) que satisfaça a condição de ligação, seja interligada para forma uma única linha;
- A ligação é sempre estabelecida na cláusula WHERE através da igualdade de campos de tabelas diferentes, tabelas essas que precisam ter sido especificadas na cláusula FROM que estabelece o produto cartesiano entre as tabelas listadas e a cláusula WHERE filtra as linhas úteis segundo a condição especificada.

Ligação entre tabelas - JOIN

Departamento

cod_dep	descricao	localizacao
1	Desenvolvimento	Sala C3-10
2	Análise	Sala B2-30
3	Teste	Sala C1-10
4	Contabilidade	Sala A1-20

Funcionário

cod_func	nome	data_nasc	cod_dep
1	João	1980-01-02	1
2	José	1981-02-03	2
3	Maria	1982-05-04	1
4	Antônio	1983-07-06	3

Consulta - Junção Simples

```
Select f.cod_func, f.nome, f.data_nasc, d.descricao
From departamento as d, funcionario as f
where d.cod_dep = f.cod_dep
```

Saída

cod_func	nome	data_nasc	descricao
1	João	1980-01-02	Desenvolvimento
2	José	1981-02-03	Análise
3	Maria	1982-05-04	Desenvolvimento
4	Antônio	1983-07-06	Teste

Ligação entre tabelas - JOIN

- **Junção interna (Inner Join)**
  - A junção interna entre tabelas é a modalidade de junção que faz com que somente participem da relação resultante as linhas das tabelas de origem que atenderem à cláusula de junção.

Consulta - Junção interna

```
Select f.cod_func, f.nome, f.data_nasc, d.descricao
From funcionario as f INNER JOIN departamento as d
                      ON f.cod_dep = d.cod_dep
```

Saída

cod_func	nome	data_nasc	descricao
1	João	1980-01-02	Desenvolvimento
3	Maria	1982-05-04	Desenvolvimento
2	José	1981-02-03	Análise
4	Antônio	1983-07-06	Teste

Ligação entre tabelas - JOIN

- **Junção Externa (Outer Join)**
  - Na junção externa, os registros que participam do resultado da junção não obrigatoriamente obedecem à condição de junção, ou seja, a não inexistência de valores correspondentes não limita a participação de linhas no resultado de uma consulta.
  - Existem tipos diferentes de junção externa. São elas:
    - Junção Externa à Esquerda (Left Outer Join)
    - Junção Externa à Direita (Right Outer Join)

Ligação entre tabelas - JOIN

- **Junção Externa à Esquerda (Left Outer Join)**
  - Suponha que desejemos uma listagem com os nomes de todos os departamentos cadastrados no nosso banco de dados e, para aqueles que possuam funcionários lotados nele, apresente os respectivos nomes. Para isso, teremos que utilizar a junção externa à esquerda.

Consulta - Junção externa à esquerda

```
Select d.descricao, f.nome
From departamento as d LEFT OUTER JOIN funcionario as f
                       ON f.cod_dep = d.cod_dep
```

Saída

descricao	nome
Desenvolvimento	João
Desenvolvimento	Maria
Análise	José
Teste	Antônio
Contabilidade	(NULL)

Ligação entre tabelas - JOIN

- **Junção Externa à Direita (Right Outer Join)**
  - A junção externa à direita é muito parecida com a junção externa à esquerda. A única diferença está no fato de que a tabela da qual todas as linhas constarão no resultado está posicionada à direita do termo *Right Outer Join* no comando.

Consulta - Junção externa à direita

```
Select d.descricao, f.nome
From departamento as d RIGHT OUTER JOIN funcionario as f
                       ON f.cod_dep = d.cod_dep
```

Saída

descricao	nome
Desenvolvimento	João
Análise	José
Desenvolvimento	Maria
Teste	Antônio

Operações com cadeias de caracteres

- O SQL inclui um mecanismo de concordância de padrões para comparações envolvendo cadeia de caracteres;
- Os padrões são descritos recorrendo a dois caracteres especiais:
  - **Percentagem (%)**: concorda com qualquer subcadeia.
  - **Sublinhado (_)**: concorda com qualquer carácter.
- As operações mais usadas são as checagens para verificação de coincidência de pares, usando o operador **LIKE**. Exemplos.
- 1. listar todos os clientes que começam com a letra “A”.

  Select * from cliente

  where nome like “A%”
- 2. listar todos os clientes que terminam com o nome “Silva”.

  Select * from cliente

  where nome like “%Silva”
- 3. listar todos os clientes que possuam o sobrenome “Araujo”.

  Select * from cliente

  where nome like “%Araujo%”

Operações com cadeias de caracteres

Outros exemplos:

- **Like "A%"** – todas as palavras que iniciam com a letra A.
- **Like "%A"** – todas as palavras que terminam com a letra A.
- **Like "%A%"** – todas que tenham a letra A em qualquer posição.
- **Like "A_"** – string de 2 caracteres que tenha a letra A e o segundo caráter qualquer outro.
- **Like "_A"** – string de 2 caracteres cujo primeiro caráter seja qualquer um e a última seja a letra A.
- **Like "%A_"** – todos que tenham a letra A na penúltima posição e última seja qualquer outro caráter;
- **Like "_A%"** – todos que tenham a letra A na segunda posição e o primeiro caráter seja qualquer um.

Funções de agregação

Funções de Agregação	
AVG	Calcula a média dos valores selecionados
MIN	Calcula o menor valor entre os selecionados
MAX	Calcula o maior valor entre os selecionados
COUNT	Conta quantos valores foram selecionados
SUM	Calcula o somatório dos valores selecionados

- Funções agregadas são funções que tomam uma coleção de valores como entrada, retornando um valor simples. A SQL oferece cinco funções agregadas pré-programadas:
- **Observação:**
  - A entrada para SUM e AVG precisam ser um conjunto de números, mas as outras operações podem operar com conjuntos de tipos de dados não numéricos, como Strings e semelhantes;
  - O SQL não permite o uso de DISTINCT com COUNT. No entanto permite usar DISTINCT com MAX e MIN;

Funções agregadas

cod_produto	produto	unidade	valor_unit
1	Arroz	LT	2,00
2	Feijão	LT	3,00
3	Sabão	UN	1,00
4	Farinha	LT	1,50
5	Milho	LT	0,50
6	Leite	LT	2,90
7	Cafe	KG	3,00
8	Azeite	LT	3,50

- **AVG (média)**
  - Utilizado para obtenção de médias;
  - Exemplo: Exibir o preço médio dos produtos.

```
select avg(valor_unit) as média from produto
```

média
2,175000

Funções agregadas

cod_produto	produto	unidade	valor_unit
1	Arroz	LT	2,00
2	Feijão	LT	3,00
3	Sabão	UN	1,00
4	Farinha	LT	1,50
5	Milho	LT	0,50
6	Leite	LT	2,90
7	Cafe	KG	3,00
8	Azeite	LT	3,50

- **Min (Mínimo)**
  - Utilizado para a obtenção do mínimo valor de um atributo;
  - Exemplo: Exibir o preço mínimo dos produtos.

```
select min(valor_unit) as mínimo from produto
```

mínimo
0,50

Funções agregadas

cod_produto	produto	unidade	valor_unit
1	Arroz	LT	2,00
2	Feijão	LT	3,00
3	Sabão	UN	1,00
4	Farinha	LT	1,50
5	Milho	LT	0,50
6	Leite	LT	2,90
7	Cafe	KG	3,00
8	Azeite	LT	3,50

- **Max (Máximo)**
  - Utilizado para se obter o máximo valor de um atributo;
  - Exemplo: Exibir o máximo valor dos produtos.

```
select max(valor_unit) as maximo from produto
```

maximo
3,50

Funções agregadas

cod_produto	produto	unidade	valor_unit
1	Arroz	LT	2,00
2	Feijão	LT	3,00
3	Sabão	UN	1,00
4	Farinha	LT	1,50
5	Milho	LT	0,50
6	Leite	LT	2,90
7	Cafe	KG	3,00
8	Azeite	LT	3,50

- **Count (Contar)**
  - Usamos a função agregada COUNT com muita freqüência para contar o número de tuplas em uma relação;
  - Exemplo: Exibir a quantidade de produtos cadastrados.

```
select count(cod_produto) as quantidade from produto
```

quantidade
8

Funções agregadas

cod_produto	produto	unidade	valor_unit
1	Arroz	LT	2,00
2	Feijão	LT	3,00
3	Sabão	UN	1,00
4	Farinha	LT	1,50
5	Milho	LT	0,50
6	Leite	LT	2,90
7	Cafe	KG	3,00
8	Azeite	LT	3,50

- **Sum (Soma)**
  - Utilizado para se obter a soma dos valor de um atributo;
  - Exemplo: Exibir a soma do valores do produtos.

```
select sum(valor_unit) as soma from produto
```

soma
17,40

Group by (Grupamento)

- O atributo ou atributos fornecidos em uma cláusula GROUP BY são usados para formar grupos. Tuplas com os mesmos valores em todos os atributos da cláusula **GROUP BY** são colocados em um grupo.
- As funções de agregação não podem ser combinadas a outros atributos da tabela no resultado da consulta. Nestes casos a apresentação dos resultados da consulta com a organização dos dados em grupos. Para isso é utilizada a cláusula **GROUP BY**. Esta cláusula agrupa os resultados por valores idênticos. Ela é usada, muitas vezes, com as funções de agregação, mas pode ser utilizada, também, sem estas.
- Observação: Atributos nas cláusula SELECT fora das funções agregadas devem aparecer na lista **GROUP BY**.

Group by (Grupamento)

Funcionário

cod_func	nome	data_nasc	cod_dep
1	João	1980-01-02	1
2	José	1981-02-03	2
3	Maria	1982-05-04	1
4	Antônio	1983-07-06	3

Consulta - Grupamento

```
Select cod_dep, count(cod_dep)
from funcionario
group by cod_dep
```

Saída

cod_dep	count(cod_dep)
1	2
2	1
3	1

Having (grupamento condicional)

- Os predicados **HAVING** são aplicados depois da formação dos grupos, assim poderão ser usadas funções agregadas. Utilizado quando uma condição não se aplica a uma única tupla, mas a cada grupo determinado pela cláusula GROUP BY;
- Se uma cláusula WHERE e uma cláusula HAVING aparecem na mesma consulta, o predicado que aparece em WHERE é aplicado primeiro;
- Exemplos:

```
Select cod_paciente, max(data)
  From Consultas
  Group by cod_paciente
  Having max(data) < “2012-12-10”
```

```
select cod_paciente, max(data)
From Consultas
Where cod_medico = 1
Group by cod_paciente
Having max(data) < “2012-12-10”
```

Consultas aninhadas (Subconsultas)

- Uma consulta SQL é aninhada quando ela está dentro de outra consulta SQL;
- Na manipulação de um Banco de Dados algumas consultas são resultantes de uma série de outras subconsultas, ou seja, o resultado esperado é obtido a partir de uma consulta final sobre o resultado de uma subconsulta;
- Para podermos realizar subconsultas precisamos do auxílio de cláusulas ou verificadores que facilitam o acesso de uma consulta mais externa aos resultados das subconsultas. As cláusulas auxiliadoras são **in** e **not in**, **any**, **all**, **exists** e **not exists** .
- **Observação:** Consultas aninhadas levam em consideração a precedência dos comandos. Apesar dos comandos serem lidos e reconhecidos na ordem que estão, a subconsulta mais interna será a primeira a ser executada e a consulta mais externa será a última, visto que a mais interna necessita dos resultados das outras de hierarquia mais baixa.

Consultas aninhadas (Subconsultas)

Clientes

num_conta	nome_cliente
1	Ana Oliveira
2	Bianca Xavier
3	João Nunes
4	José Camargo
5	Maria Alencar

Empréstimo

num_emprestimo	num_conta
101	1
809	1
433	2
203	4
211	5

- **Conectivo IN**
  - Testa os membros de um conjunto, no qual o conjunto é a coleção de valores produzidos pela cláusula SELECT;
  - O conectivos **NOT IN** verifica a ausência de membros de um conjunto.

```
Select *
From clientes
Where num_conta in (Select num_conta From emprestimo)
```

num_conta	nome_cliente
1	Ana Oliveira
2	Bianca Xavier
4	José Camargo
5	Maria Alencar

Consultas aninhadas (Subconsultas)

- **Conectivo EXISTS**
  - A cláusula **exists** é uma cláusula de teste, podendo estar ligada a um valor *TRUE* ou *FALSE* . Ela gera *TRUE* se no resultado de uma subconsulta existir pelo menos uma tupla. Isso quer dizer que a tarefa estabelecida pelo *select* mais externo será realizada se e somente se o *where exists* confirmar a existência de pelo menos uma tupla na pseudotabela resultante da subconsulta.

```
Select * from Clientes
Where exists (Select * From Clientes
              Where num_conta not in ( Select num_conta
                                       From Emprestimo))
```

num_conta	nome_cliente
1	Ana Oliveira
2	Bianca Xavier
3	João Nunes
4	José Camargo
5	Maria Alencar

Consultas aninhadas (Subconsultas)

- **Conectivo ANY**
  - A cláusula **any** permite outras formas de comparação entre elementos e conjuntos através do uso de condicionais. As condicionais que auxiliam o uso de **any** são os sinais de =, >, < e <>, que são utilizados, como exemplo, da seguinte forma:
    - **= any** (subconsulta) - se houver algum elemento **igual a algum** da subconsulta ;
    - **> any** (subconsulta) - se houver algum elemento **maior a algum** da subconsulta .

```
Select num_conta from clientes
Where num_conta <> ANY (Select num_conta from emprestimo)
```

num_conta
1
2
3
4
5

Consultas aninhadas (Subconsultas)

- **Conectivo ALL**
  - A cláusula **ALL** só funcionará se a condição for satisfeita por **TODOS** os elementos do resultado de uma subconsulta. As condicionais que auxiliam o uso de **ALL** são os sinais de =, >, < e <>, da mesma forma que no conectivo any.

```
Select num_conta from Clientes
Where num_conta <> ALL (Select num_conta from Emprestimo)
```

num_conta
3