

# Banco de Dados



SQL

João Eduardo Ferreira

Osvaldo Kotaro Takai

Marcelo Finger

# Introdução

---

- ❑ SQL - Structured Query Language
  - Linguagem declarativa – não-procedural
  - Desenvolvida pelo laboratório da IBM em San Jose – anos 60/70
  - Inicialmente chamada SEQUEL (Structured English QUery Language)
  - Criada como interface entre usuários e o primeiro SGBDR – SYSTEM R

# Introdução

---

- ❑ SQL - Structured Query Language
  - Padrão industrial que atinge grande parte do mercado de SGBDs
  - Atrativo: pequena quantidade de comandos para realizar uma grande quantidade de operações necessárias para definição e manipulação de relações
  - Simplicidade
  - Grande poder de consulta
  - Padrão facilita migração

# Introdução

---

- ❑ SQL - Structured Query Language
  - Permite otimização
  - Necessita otimização
  - Eficiência em acessos
  - Expressividade limitada, porém cobre a maioria absoluta dos casos práticos
  - Única linguagem verdadeiramente paralelizável.

# Introdução

---

- ❑ O padrão SQL
  - American National Standard Institute (ANSI) e International Organization for Standardization (ISO)
  - Versão mais recente: SQL3 (SQL99)
    - ❑ Principalmente conceitos de orientação a objetos
  - Versões anteriores
    - ❑ SQL92 – SQL2
    - ❑ SQL86 – SQL1

# Introdução

---

## □ Recursos:

- DDL e DML
- Criação de visões (views)
- Especificações de segurança e autorizações
- Definição de restrições de integridade
- Controle de transação
- Regras para integração com linguagens de programação
- Regras e triggers

# Introdução

---

- ❑ Dois conjuntos principais de comandos:
  - DDL – Data Definition Language:
    - ❑ Especificação do esquema da base de dados
  - DML – Data Manipulation Language:
    - ❑ inserção, remoção, alteração e consultas na instância da base de dados

# Introdução

---

- ❑ A aceitação da SQL não é devido as suas qualidades.
  - Alguns conceitos da linguagem são falhos.
  - Seria possível projetar uma linguagem muito melhor e mais fácil de usar.
  - Não é computacionalmente completa: muita programação é necessária.
  - As reais vantagens repousam no Modelo Relacional.



# Introdução

---

- Era para ser + simples do que CRT, CRD, e AR
- Vocabulário diverge do Modelo Relacional:
  - Relação → Tabela
  - Tuple → Linha
  - Atributo → Coluna
  - Maior desvio do Modelo Relacional: Tabelas podem ter linhas idênticas

# DDL

---

## □ Alguns comandos da DDL

- CREATE SCHEMA
- CREATE TABLE
- ALTER TABLE
- DROP TABLE
- CREATE DOMAIN
- DROP DOMAIN
- CREATE VIEW
- DROP VIEW
- CREATE INDEX
- DROP INDEX

# DDL - Criação do Esquema

---

- ❑ CREATE SCHEMA – Comando utilizado para criar esquemas de aplicações.
- ❑ O esquema permite agrupar as tabelas, restrições, visões, domínios e outros construtores (como concessão de autoridade) que descrevem o esquema.
- ❑ Por exemplo:

```
CREATE SCHEMA COMPANHIA AUTHORIZATION MAC0426
```

# DDL – Criação de Domínios

---

- ❑ CREATE DOMAIN – Utilizado para definir domínios de atributos.

*CREATE DOMAIN nome AS tipo [<restrições de coluna>]*

- Facilita a redefinição de tipos de dados de um domínio utilizados por muitos atributos de um esquema, além de melhorar a legibilidade do esquema.

- ❑ Por exemplo:

```
CREATE DOMAIN TIPO_ID AS CHAR(9);
```

# DDL – Criação de Domínios

---

- Pode-se definir um novo domínio com a especificação de uma restrição sobre o tipo de dados.
- Por exemplo:

```
CREATE DOMAIN TIPO_DEPNUM AS INTEGER  
CHECK (TIPO_DEPNUM > 0 AND TIPO_DEPNUM < 21);
```

# DDL – Criação de Tabelas

---

- ❑ CREATE TABLE - cria uma tabela (tabela base), e define colunas e restrições

```
CREATE TABLE [esquema].tabela (  
    atrib1 tipo [<restrições da coluna 1>],  
    atrib2 tipo [<restrições da coluna 2>],  
    ....  
    atribn tipo [<restrições da coluna n>],  
    <restrições da tabela>  
);
```

# DDL – Criação de Tabelas

---

## □ Restrições de colunas

- NOT NULL
- DEFAULT *valor*
- CHECK(*condição*)

```
CREATE TABLE [esquema].tabela (  
    atrib1 tipo [(tamanho)] [NOT NULL | DEFAULT valor]  
        [CHECK (condição)],  
    atrib2 tipo [(tamanho)] [NOT NULL | DEFAULT valor]  
        [CHECK (condição)],
```

...

# DDL – Criação de Tabelas

---

## □ Restrições de tabela

- PRIMARY KEY ( *<atributos chave primária>* )
- UNIQUE ( *<atributos chave candidata>* )
- FOREIGN KEY ( *<atributos chave estrangeira>*  
REFERENCES *tabelaRef* [(*<chave primária>*)] [*<ações>*]

## □ *<ações>*

- ON DELETE | ON UPDATE
- CASCADE | SET NULL | SET DEFAULT
- CHECK(*condição*)



# SQL – Alguns tipos de dado

---

- ❑ INTEGER | SMALLINT
- ❑ DECIMAL [(precision, scale)] - precision é o número total de dígitos total e scale é o número de dígitos depois do ponto
- ❑ DOUBLE PRECISION | FLOAT | REAL
- ❑ CHAR(n) - tamanho fixo - n caracteres
- ❑ VARCHAR(n) - tamanho variável – máximo de n caracteres
- ❑ BLOB – Binary Large Object
- ❑ DATE | TIME | TIMESTAMP
- ❑ ...

# DDL – Criação de Tabelas

## □ Forma geral:

```
CREATE TABLE [esquema].tabela (  
    atrib1 tipo [(tamanho)] [NOT NULL | DEFAULT valor] [CHECK (condição)],  
    atrib2 tipo [(tamanho)] [NOT NULL | DEFAULT valor] [CHECK (condição)],  
    ...  
    [CONSTRAINT nome da restrição  
        PRIMARY KEY (<atributos chave primária>),  
    [CONSTRAINT nome da restrição  
        UNIQUE (<atributos chave candidata>),  
    [CONSTRAINT nome da restrição  
        FOREIGN KEY (<atributos chave estrangeira>  
            REFERENCES tabelaRef [(<chave primária>)]  
            [ON DELETE CASCADE | SET NULL | SET  
DEFAULT]  
            [ON UPDATE CASCADE | SET NULL | SET  
DEFAULT],  
    [CONSTRAINT nome da restrição  
        CHECK (condição)  
    );
```

# DDL – Criação de Tabelas

## Exemplo:

### EMPREGADO

PNome	MNome	SNome	ID	DATANASC	ENDERECO	SEXO	SALARIO	IDSUPER	NDEP
-------	-------	-------	----	----------	----------	------	---------	---------	------

```
CREATE TABLE COMPANHIA.EMPREGADO (  
    PNome          VARCHAR(15)    NOT NULL,  
    MNome          CHAR,            
    SNome          VARCHAR(15)    NOT NULL,  
    ID              INTEGER        NOT NULL,  
    DATANASC       DATE,            
    ENDERECO       VARCHAR(30),     
    SEXO           CHAR,            
    SALARIO        DECIMAL(10,2),   
    IDSUPER        INTEGER        NOT NULL,  
    NDEP           TIPO_DEPNUM    NOT NULL,  
    PRIMARY KEY (ID),  
    FOREIGN KEY (IDSUPER) REFERENCES EMPREGADO(ID),  
    FOREIGN KEY (NDEP) REFERENCES DEPARTAMENTO(DNUMERO)  
);
```

# DDL – Alteração de Tabela

---

- ❑ ALTER TABLE – incluir/alterar/remover definições de colunas e restrições

ALTER TABLE *tabela* <ação>;

- ❑ <ação>:
  - ADD *novoAtrib tipo* [<restrições de coluna>]
  - ADD [CONSTRAIN *nome*] <restrição de tabela>
  - DROP *atributo* [CASCADE | RESTRICT]
  - DROP CONSTRAINT *nome*

# DDL – Alteração de Tabela

---

- ❑ *ADD novoAtrib tipo [<restrições de coluna>]*
  - E o valor do novo atributo nas tuplas já existentes?
    - ❑ Se não for especificada nenhuma cláusula default, então o valor será null. Assim, a cláusula NOT NULL não pode ser aplicada.
- ❑ Por exemplo:

```
ALTER TABLE COMPANHIA.EMPREGADO  
ADD FUNCAO VARCHAR(12);
```

# DDL – Alteração de Tabela

---

- ❑ DROP *atributo* [CASCADE | RESTRICT]
  - CASCADE – todas as visões e restrições (*constrains*) que referenciam o atributo são removidas automaticamente
  - RESTRICT – o atributo só é removido se não houver nenhuma visão ou restrição que o referencie
- ❑ Por exemplo:  

```
ALTER TABLE COMPANHIA.EMPREGADO  
DROP FUNCAO CASCADE;
```

# DDL – Remoção de Tabela

---

- ❑ DROP TABLE - exclui uma tabela do banco de dados

DROP TABLE *tabela* [CASCADE | RESTRICT];

- CASCADE: todas as visões e restrições que referenciam a tabela são removidas automaticamente
  - RESTRICT: a tabela é removida somente se não for referenciada em nenhuma restrição ou visão
- ❑ Por exemplo:

**DROP TABLE COMPANHIA.DEPENDENTE CASCADE**

# DDL – Remoção de Esquema

---

- ❑ DROP SCHEMA - exclui um esquema do banco de dados

DROP SCHEMA *esquema* [CASCADE | RESTRICT];

- CASCADE: todos os elementos do esquema são removidos automaticamente
  - RESTRICT: o esquema só será removido se não existir os elementos
- ❑ Por exemplo:

**DROP SCHEMA COMPANHIA CASCADE;**



# DML

---

- Alguns comandos da DML
  - INSERT
  - UPDATE
  - DELETE
  - SELECT

# DML – Inserção

---

- ❑ INSERT – insere uma ou mais tuplas em uma tabela

- Inserção de 1 tupla:

**INSERT INTO** *tabela* [(*atrib1,atrib2,...*)]  
**VALUES** (*valor1, valor2,...*)

- Inserção de múltiplas tuplas:

**INSERT INTO** *tabela* [(*atrib1,atrib2,...*)]  
*<comando SELECT>*

# DML – Inserção

---

- Exemplo 1 – Inserção de uma única tupla:
  - Inserir 3 as tuplas na relação PROJETO:

PROJETO				ce
PNUMERO	PLOCALIZACAO	DNUM		

```
INSERT INTO PROJETO VALUES ('ProductX', '1', 'Bellaire', '5')
INSERT INTO PROJETO VALUES ('ProductY', '2', 'Sugarland', '5')
INSERT INTO PROJETO VALUES ('ProductZ', '3', 'Houston', '5')
```

- O terceiro valor '5' corresponde ao departamento 5. Logo, o departamento 5 deve existir na relação DEPARTAMENTO para que as inserções tenham sucesso; pois caso contrário, violaria a restrição de integridade referencial.

# DML – Inserção

---

- ❑ Exemplo 2 – Inserção de múltiplas tuplas:
  - Popular uma tabela temporária DEPTS\_INFO:

```
❑ CREATE TABLE DEPTS_INFO (  
    DEPT_NAME      VARCHAR(10),  
    NO_OF_EMPS     INTEGER,  
    TOTAL_SAL      INTEGER  
);  
  
❑ INSERT INTO DEPTS_INFO (DEPT_NAME, NO_OF_EMPS, TOTAL_SAL)  
    SELECT DNAME, COUNT (*), SUM (SALARY)  
    FROM DEPARTMENT, EMPLOYEE  
    WHERE DNUMBER=DNO  
    GROUP BY DNAME;
```

O comando **SELECT** será explicado posteriormente. Por hora, o importante é saber que podemos inserir em uma relação, várias tuplas, resultado de uma consulta **SELECT**.

# DML – Alteração

---

- UPDATE – modifica o valor de um atributo em uma ou mais tuplas da tabela

```
UPDATE tabela SET
    atributo1 = <valor ou expressão>,
    atributo2 = <valor ou expressão>,
    ...
WHERE <condição de localização>;
```

- Por exemplo,

```
UPDATE PROJETO
SET PLOCALIZACAO = 'Bellaire', DNUM = 5
WHERE PNUMERO=10;
```

# DML – Remoção

---

- ❑ DELETE – remove uma ou mais tuplas da tabela.

```
DELETE FROM tabela1 [FROM tabela2]  
[WHERE <condição de localização>];
```

- ❑ Exemplos:

```
DELETE FROM EMPREGADO  
WHERE ID = 1234;
```

```
DELETE FROM EMPREGADO  
WHERE DNUM IN  
    (SELECT DNUMERO  
     FROM DEPARTAMENTO  
     WHERE DNOME='Research');
```

```
DELETE FROM EMPLOYEE;
```

# DML - Consulta

---

- SELECT – Comando de consulta.

- Forma geral:

```
SELECT [ DISTINCT | ALL ] <lista de atributos>  
FROM <lista de tabelas>  
[ WHERE <condições> ]  
[ GROUP BY atributo ]  
[ HAVING <condições> ]  
[ ORDER BY atributo [ ASC | DESC ] ];
```

# DML - Consulta

---

- ❑ **SELECT** – seleciona O QUE se deseja na tabela resultado:
  - **<lista de atributos>** ou **\*** (para todos os atributos)
  - **ALL** – inclui tuplas duplicadas (é o default)
  - **DISTINCT** – elimina tuplas duplicadas
  - **FROM** – DE ONDE retirar os dados necessários
  - **WHERE** – CONDIÇÕES de seleção dos resultados.



# DML – Consulta Simples

- ❑ Consulta 1 – Recuperar a data de aniversário e o endereço do empregado chamado 'John B. Smith'.

## Em SQL:

```
SELECT DATANASC, ENDERECO  
FROM EMPREGADO  
WHERE PNAME='John' AND MNAME='B' AND SNAME='Smith';
```

## Em Álgebra Relacional:

```
EMP_SMITH       $\leftarrow \sigma$  PNAME='John' AND MNAME='B' AND SNAME='Smith' (EMPREGADO)  
RESULTADO       $\leftarrow \pi$  DATANASC, ENDERECO (EMP_SMITH)
```

- Assim, uma consulta com apenas uma relação na cláusula FROM é similar a um par de operações SELECT-PROJECT da Álgebra Relacional.
- A diferença é que podemos obter tuplas repetidas no resultado.

# DML – Consulta Simples

---

- A mesma consulta em CRT e CRD.

## Em SQL:

```
SELECT DATANASC, ENDERECO  
FROM EMPREGADO  
WHERE PNOME='John' AND MNOME='B' AND SNOME='Smith';
```

## Em Cálculo Relacional de Tuplas:

$$\{ e.DATANASC, e.ENDERECO \mid \text{EMPREGADO}(E) \text{ AND } \\ e.PNOME='John' \text{ AND } e.MNOME='B' \text{ AND } e.SNOME='Smith' \}$$

## Em Cálculo Relacional de Domínio:

$$\{ uv \mid (\exists q)(\exists r)(\exists s) (\text{EMPREGADO}(qrstuvwxyz) \text{ AND } \\ q='John' \text{ AND } r='B' \text{ AND } s='Smith') \}$$

# DML – Consulta com um Join

- Consulta 2 – Obter o nome e o endereço dos empregados que trabalham para o departamento de 'Pesquisa'.

## Em SQL:

```
SELECT PNOOME, ENDERECO
FROM EMPREGADO, DEPARTAMENTO
WHERE DNOOME = 'Pesquisa' AND DNUMERO=DNUM;
```

## Em Álgebra Relacional:

```
DEP_PESQUISA(DNUM)  $\leftarrow \pi_{DNUMERO} (\sigma_{DNOOME='Pesquisa'} (DEPARTAMENTO))$ 
RESULT  $\leftarrow \pi_{PNOOME, ENDERECO} (DEP\_PESQUISA * EMPREGADO)$ 
```

## Em Cálculo Relacional de Tuplas:

```
{ t.PNOOME, t.ENDERECO | EMPREGADO( t ) AND (  $\exists d$  ) ( DEPARTAMENTO( d )
AND
d.DNOOME='Pesquisa' AND d.DNUMERO=t.DNUM ) }
```

## Em Cálculo Relacional de Domínio:

```
{ qv | (  $\exists z$  ) (  $\exists l$  ) (  $\exists m$  ) ( EMPREGADO(qrstuvwxyz) AND DEPARTAMENTO(lmno)
AND
l = 'Pesquisa' AND m = z ) }
```

# DML – Consulta com um Join

---

- Similar à seqüência SELECT-PROJECT-JOIN da Álgebra Relacional
- (DNOME='Pesquisa') é uma condição de seleção do operador SELECT da Álgebra Relacional)
- (DNUMERO=DNUM) é uma condição de junção do operador JOIN da Álgebra Relacional.

# DML – Consulta com dois Joins

- Para todo projeto localizado em 'Stafford', listar o número do projeto, o número do departamento responsável, o sobrenome, endereço e data de nascimento do gerente responsável pelo departamento.

## Em SQL:

```
SELECT PNUMERO, DNUM, SNAME, DATANASC, ENDERECO  
FROM PROJETO, DEPARTAMENTO, EMPREGADO  
WHERE DNUM=DNUMERO AND GERID=ID AND PLOCALIZACAO='Stafford';
```

## Em Álgebra Relacional:

```
STAFFORD_PROJS  $\leftarrow \sigma_{PLOCALIZAÇÃO = 'Stafford'} (PROJETO)$   
CONTR_DEPT  $\leftarrow (STAFFORD\_PROJS \times_{DNUM = DNUMERO} DEPARTAMENTO)$   
PROJ_DEPT_MGR  $\leftarrow (CONTR\_DEPT \times_{GERID = ID} EMPREGADO)$   
RESULT  $\leftarrow \pi_{PNUMERO, DNUM, SNAME, ENDERECO, DATANASC} (PROJ\_DEPT\_MGR)$ 
```

# DML – Consulta com dois Joins

---

- Na consulta anterior, existem duas condições Join
- A condição Join  $DNUM=DNUMERO$  relaciona um projeto com o departamento que o controla
- A condição Join  $GERID=ID$  relaciona o departamento com o empregado que o gerencia.

# Qualificando um Atributo

---

- ❑ Em SQL podemos usar o mesmo nome para dois ou mais atributos, desde que os atributos estejam em relações diferentes.
- ❑ Uma consulta que referencia dois ou mais atributos com o mesmo nome deve qualificar o atributo com o nome da relação.
- ❑ Exemplo:

## **Em SQL:**

```
SELECT EMPREGADO.PNOME, PROJETO.PNOME  
FROM PROJETO, DEPARTAMENTO, EMPREGADO  
WHERE DNUM=DNUMERO AND GERID=ID AND PLOCALIZACAO='Stafford';
```

# ALIASES

---

- ❑ Algumas consultas precisam referenciar duas vezes a mesma relação
- ❑ Em alguns casos, pseudônimos (*aliases*) são atribuídos ao nome da relação
- ❑ Por exemplo, considere a seguinte consulta:
  - Para cada empregado, recupere o nome do empregado e o nome de seu supervisor imediato.

## Em SQL:

```
SELECT E.PNOME, E.SNOME, S.PNOME, S.SNOME  
FROM EMPREGADO E S  
WHERE E.IDSUPER=S.ID;
```



# ALIASES

---

- ❑ Na consulta anterior, E e S são chamados de *alias* ou *variáveis de tupla* da relação EMPREGADO.
- ❑ Podemos pensar em E e S como duas cópias distintas de EMPREGADO: E representa os supervisionados e S representa os supervisores.
- ❑ *Aliases* podem ser usados em qualquer consulta SQL.
- ❑ Pode-se, também, usar a palavra-chave AS:

## Em SQL:

```
SELECT E.PNOME, E.SNOME, S.PNOME, S.SNOME  
FROM EMPREGADO AS E, EMPREGADO AS S  
WHERE E.IDSUPER=S.ID;
```

# Cláusula WHERE não especificada

---

- ❑ Uma cláusula WHERE não especificada indica ausência de uma condição.
- ❑ Assim, todas as tuplas das relações da cláusula FROM serão selecionadas.
- ❑ Isso equivale a condição WHERE TRUE.
- ❑ Por exemplo, considere a seguinte consulta:
  - Recupere o ID de todos os empregados.

```
Em SQL:  
SELECT ID  
FROM EMPREGADO;
```

# Cláusula WHERE não especificada

---

- Se mais de uma relação é especificada na cláusula FROM e não existir nenhuma condição de junção, então o resultado será o produto cartesiano.

## Em SQL:

```
SELECT ID, DNOME  
FROM EMPREGADO, DEPARTAMENTO;
```

- É extremamente importante não negligenciar a especificação de qualquer condição de seleção ou de junção na cláusula WHERE; sob a pena de gerar resultados incorretos e volumosos.

# O Uso do \*

---

- Um \* é usado para recuperar todos os valores de atributos da tupla selecionada.
- Exemplos:

## **Em SQL:**

```
SELECT *  
FROM EMPREGADO  
WHERE DNUM=5;
```

---

```
SELECT *  
FROM EMPREGADO, DEPARTAMENTO  
WHERE DNOME='Research' AND DNUM=DNUMERO;
```

# Uso do DISTINCT

---

- ❑ A SQL não trata uma relação como um conjunto; tuplas duplicadas podem ocorrer.
- ❑ Para eliminar tuplas duplicadas no resultado de uma consulta, a palavra **DISTINCT** é usada.
- ❑ A primeira consulta abaixo pode gerar tuplas duplicadas, mas a segunda não:

## Em SQL:

```
SELECT SALARIO  
FROM EMPREGADO;
```

---

```
SELECT DISTINCT SALARIO  
FROM EMPREGADO;
```

# Operação de Conjunto

---

- ❑ Algumas operações de conjunto foram incorporados à linguagem SQL.
- ❑ Existe uma operação de União e, em algumas versões da SQL, existem as operações de Subtração e Intersecção.
- ❑ As relações resultantes dessas operações são sempre conjunto de tuplas; tuplas duplicadas são eliminadas do resultado.
- ❑ O conjunto de operações aplicam-se somente às relações que são compatíveis na união.

# Operação de Conjunto

---

- Listar os números de projetos em que o empregado de sobrenome Smith trabalhe ou que sejam controlados por algum departamento gerenciado pelo empregado de sobrenome Smith:

**Em SQL:**

```
(  
    SELECT PNUMERO  
    FROM PROJETO, DEPARTAMENTO, EMPREGADO  
    WHERE DNUM=DNUMBER AND GERID=ID AND SNOME='Smith' )  
UNION  
(  
    SELECT PNUMERO  
    FROM PROJETO, TRABALHA-PARA, EMPREGADO  
    WHERE PNUMERO=PNO AND EID=ID AND SNAME='Smith' );
```

# Operação de Conjunto

- ❑ A SQL também possui operações sobre multiconjuntos (conjuntos que permitem repetição de elementos)
  - UNION ALL
  - EXCEPT ALL
  - INTERSECT ALL.

R	A	S	A	R UNION ALL S	A	R EXCEPT ALL S	A	R INTERSECT ALL S	A
	a1		a1		a1		a3		a1
	a2		a2		a1				a2
	a3		a4		a2				
			a5		a2				
					a3				
					a4				
					a5				



# Consultas Aninhadas

---

- Uma consulta SELECT completa, chamada de **consulta aninhada**, pode ser especificada dentro da cláusula WHERE de uma outra consulta, chamada **consulta externa**.
- Por exemplo:
  - Recupere o nome e o endereço de todos os empregados que trabalham para o departamento de Pesquisa.

**Em SQL:**

```
SELECT PNAME, SNAME, ENDERECO
FROM EMPREGADO
WHERE DNUM IN (      SELECT DNUMERO
                     FROM DEPARTAMENTO
                     WHERE DNOME='Pesquisa' );
```

# Consultas Aninhadas

---

- ❑ A consulta externa seleciona tuplas de empregados se o valor de seu DNUM pertencer ao resultado da consulta aninhada
- ❑ O operador **IN** é equivalente ao operador pertence da teoria de conjuntos
- ❑ Em geral, podemos ter vários níveis de consultas aninhadas
- ❑ Uma referência a um atributo não qualificado estará se referindo a um atributo da relação declarada na consulta externa mais próxima
- ❑ Neste exemplo, a consulta aninhada não está correlacionado à consulta externa

# Consultas Aninhadas Correlacionadas

- ❑ Se a condição WHERE de uma consulta aninhada referenciar um atributo de uma relação declarada na consulta externa, as consultas estarão correlacionadas
- ❑ Por exemplo:
  - Recupere o nome de cada empregado que tenha um dependente com o mesmo nome do empregado.

**Em SQL:**

```
SELECT  E.PNOME, E.SNAME
FROM    EMPREGADO AS E
WHERE   E.ID      IN (
        SELECT  EID
        FROM    DEPENDENTE
        WHERE   EID=E.ID AND
        E.PNOME=NOMEDEPENDENTE );
```



Atributo referenciado

# Consultas Aninhadas Correlacionadas

---

- ❑ Na consulta anterior, a consulta aninhada tinha um resultado diferente para cada tupla da consulta externa
- ❑ Consultas escritas com blocos SELECT-FROM-WHERE e que utilizem operadores de comparação e **IN** sempre podem ser expressas como uma consulta simples
- ❑ Por exemplo, a mesma consulta pode ser escrita como:

**Em SQL:**

```
SELECT  E.PNOME, E.SNOME  
FROM    EMPREGADO AS E, DEPENDENTE AS D  
WHERE   E.ID=D.EID AND E.PNOME=D.NOMEDEPENDENTE;
```

# Conjuntos Explícitos

---

- É possível utilizar um conjunto de valores explicitamente enumerado na cláusula WHERE ao invés de utilizar uma consulta aninhada.
- Por exemplo:
  - Recupere o ID de todos os empregados que trabalham nos projetos de números 1, 2, ou 3.

**Em SQL:**

```
SELECT DISTINCT EID  
FROM TRABALHA-PARA  
WHERE PNO IN (1, 2, 3);
```

# Divisão em SQL

- ❑ Encontrar os nomes de empregados que trabalham em todos os projetos controlados pelo departamento 5

Em SQL:

```
SELECT PNO, SNO  
FROM EMPREGADO  
WHERE ( ( SELECT PNO  
          FROM TRABALHA-EM  
          WHERE ID=EID )  
        CONTAINS  
        ( SELECT PNUMERO  
          FROM PROJETO  
          WHERE DNUM=5 ) );
```

**CONTAINS** é equivalente ao operador de divisão da Álgebra Relacional.

Infelizmente, a maioria das implementações da SQL não implementam **CONTAINS**.

# Divisão em SQL

- Para fazer essa consulta em SQL que não possua o operador CONTAINS devemos recorrer ao Cálculo Relacional de Tuplas e utilizar cláusulas EXISTS e NOT EXISTS:

**Em Cálculo Relacional de Tuplas:**

$$\{ e.PNOME, e.SNOME \mid EMPREGADO(e) \text{ AND } \\ (\forall x) ( ( PROJETO(x) \text{ AND } x.DNUM=5 ) \Rightarrow \\ (\exists w) (TRABALHA-EM(w) \text{ AND } \\ w.EID=e.ID \text{ AND } \\ x.PNUMERO=w.PNO) ) \}$$

**Ou, transformando o quantificador universal e a implicação utilizando**

**$\text{NOT } (\exists t) ( \text{NOT } F(t) ) \equiv (\forall t) ( F(t) )$     $\text{NOT } (F1 \text{ AND } F2) \equiv \text{NOT } F1 \text{ OR } \text{NOT } F2$**

**$F2 \equiv F1 \Rightarrow F2$**

**temos:**

$$\{ e.PNOME, e.SNOME \mid EMPREGADO(e) \text{ AND } \\ \text{NOT } (\exists x) ( PROJETO(x) \text{ AND } x.DNUM=5 \text{ AND } \\ \text{NOT } (\exists w) (TRABALHA-EM(w) \text{ AND } \\ w.EID=e.ID \text{ AND } \\ x.PNUMERO=w.PNO) ) \}$$

# Divisão em SQL

## Em Cálculo Relacional de Tuplas:

$\{ e.PNOME, e.SNOME \mid EMPREGADO(e) \text{ AND}$   
 $\text{NOT } (\exists x) ( PROJETO(x) \text{ AND } x.DNUM=5 \text{ AND}$   
 $\text{NOT } (\exists w) (TRABALHA-EM(w) \text{ AND}$   
 $w.EID=e.ID \text{ AND}$   
 $x.PNUMERO=w.PNO) ) \}$

## Em SQL:

```
SELECT  E.PNOME, E.SNOME
FROM    EMPREGADO AS E
WHERE NOT EXISTS (
```

```
        FROM
        WHERE
        NOT EXISTS (
```

```
SELECT  *
FROM    PROJETO X
WHERE   X.DNUM=5 AND
```

```
SELECT  *
FROM    TRABALHA-PARA W
WHERE   W.EID = E.ID AND
X.PNUMERO = W.PRNO ) );
```



# Valores Nulos em Consultas SQL

---

- ❑ A SQL permite que consultas verifiquem se um valor é nulo utilizando **IS** ou **IS NOT**
- ❑ Por exemplo:
  - Recupere os nomes de todos os empregados que não possuem supervisores.

## Em SQL:

```
SELECT PNAME, SNAME  
FROM EMPREGADO  
WHERE IDSUPER IS NULL;
```

# Junção de Relações

---

- ❑ Pode se especificar uma “junção de relações” na cláusula FROM.
- ❑ A **junção de relações** é uma relação como outra qualquer, mas é o resultado de uma junção.
- ❑ Permite que o usuário especifique tipos diferentes de junções ("theta" JOIN, NATURAL JOIN, LEFT OUTER JOIN, RIGHT OUTER JOIN, CROSS JOIN, etc).

# Junção de Relações

---

## □ Por exemplo:

**Em SQL:**

```
SELECT PNAME, SNAME, ENDERECO  
FROM EMPREGADO, DEPARTAMENTO  
WHERE DNAME='Pesquisa' AND DNUMERO=DNUM;
```

## □ Pode ser reescrito com 'theta' Join:

**Em SQL:**

```
SELECT PNAME, SNAME, ENDERECO  
FROM EMPREGADO JOIN DEPARTAMENTO ON DNUMERO=DNUM  
WHERE DNAME='Pesquisa';
```

# Junção de Relações

## □ Por exemplo:

**Em SQL:**

```
SELECT DNOME, DLOCALIZACAO  
FROM DEPARTAMENTO AS D, LOCAIS-DEPTO AS L;  
WHERE DNOME = 'Pesquisa' AND D.DNUMERO = L.DNUMERO;
```

## □ Pode ser reescrito com Natural Join:

**Em SQL:**

```
SELECT DNOME, DLOCALIZACAO  
FROM DEPARTAMENTO NATURAL JOIN LOCAIS-DEPTO  
WHERE DNOME='Pesquisa';
```

**LOCAIS-DEPTO**

DNUMERO   DLOCALIZACAO

**DEPARTAMENT**

DNOME DNUMERO GERNNS GERDATINIC

# Junção de Relações

---

## □ Por exemplo:

**Em SQL:**

```
SELECT E.PNOME, E.SNOME, S.PNOME, S.SNOME  
FROM EMPREGADO E S  
WHERE E.IDSUPER=S.ID;
```

## □ Pode ser escrita como:

**Em SQL:**

```
SELECT E.PNOME, E.SNOME, S.PNOME, S.SNOME  
FROM ( EMPREGADO AS E LEFT OUTER JOIN EMPREGADO AS S ON  
                                             E.IDSUPER=S.ID );
```

PNOME de empregados que não possuem um supervisor também serão apresentados, porém com PNOME e SNOME do supervisor com valores nulos.

# Funções Agregadas

---

- ❑ A SQL possui as seguintes funções agregadas:
  - **COUNT, SUM, MAX, MIN e AVG**
- ❑ Exemplo:
  - Encontrar o maior salário, o menor salário, e a média salarial de todos os empregados.

**Em SQL:**

```
SELECT MAX(SALARIO), MIN(SALARIO), AVG(SALARIO)  
FROM EMPREGADO;
```

Algumas implementações da SQL não permitem mais de uma função agregada na cláusula **SELECT**.

# Funções Agregadas

---

## □ Exemplo:

- Recuperar o total de empregados da companhia (Consulta A) e o número de empregados do departamento Pesquisa (Consulta B).

**Em SQL (Consulta A):**

```
SELECT COUNT (*)  
FROM EMPREGADO;
```

**Em SQL (Consulta B):**

```
SELECT COUNT (*)  
FROM EMPREGADO, DEPARTAMENTO  
WHERE DNUM=DNUMBER AND DNOME='Pesquisa';
```

# Agrupamento

---

- ❑ Como na extensão da Álgebra Relacional, a SQL tem uma cláusula GROUP BY para especificar os atributos de agrupamento, que devem aparecer na cláusula SELECT.
- ❑ Por exemplo:
  - Para cada departamento, recuperar o seu número, a quantidade de empregados que possui e a sua média salarial.

## **Em SQL:**

```
SELECT DNUM, COUNT (*), AVG (SALARIO)
FROM EMPREGADO
GROUP BY DNUM;
```



# Agrupamento

---

- Por exemplo:
  - Para cada projeto, recuperar o número do projeto, seu nome e o número de empregados que trabalham no projeto.

**Em SQL:**

```
SELECT PNUMERO, PNOME, COUNT (*)  
FROM PROJETO, TRABALHA-EM  
WHERE PNUMERO=PNO  
GROUP BY PNUMERO, PNOME;
```

- Neste caso, o agrupamento e a função agregada são aplicadas após a junção das duas relações.

# A Cláusula HAVING

---

- ❑ Algumas vezes queremos recuperar os valores das funções agregadas que satisfaçam a certas condições
- ❑ A cláusula HAVING é usada para especificar essa condição
- ❑ Por exemplo:
  - Para cada projeto em que trabalhem mais de dois empregados, recupere o número do projeto, o nome do projeto e o número de empregados que trabalham no projeto.

## Em SQL

```
SELECT PNUMERO, PNOME, COUNT(*)  
FROM PROJETO, TRABALHA-PARA  
WHERE PNUMERO=PNO  
GROUP BY PNUMERO, PNOME  
HAVING COUNT(*) > 2;
```

# Comparação de Substrings

---

- ❑ O operador de comparação **LIKE** é usado para comparar partes de uma string
- ❑ Os dois caracteres reservados são usados:
  - O '%' (ou '\*' em algumas implementações) pesquisa um número arbitrário de caracteres
  - O '\_' pesquisa um único caractere arbitrário.
- ❑ Por exemplo:
  - Recupere todos os empregados que morem em Houston, Texas. Aqui, o valor do atributo endereço deve conter a substring 'Houston,TX'.

## Em SQL

```
SELECT  PNAME, SNAME  
FROM    EMPREGADO  
WHERE   ENDereco LIKE '%Houston,TX%'
```

# Comparação de Substrings

---

- ❑ Encontre todos os empregados que nasceram durante a década de 50.

## Em SQL

```
SELECT PNAME, SNAME  
FROM EMPREGADO  
WHERE DATANASC LIKE '____5_'
```

- ❑ Se o caractere '%' ou '\_' for necessário como um caractere normal de uma string, ele deve ser precedido por um caractere de *escape*.
  - A string 'AB\_CD%EF' deve ser escrita como: 'AB\\_CD\%EF'.

# Operações Aritméticas

---

- ❑ Os operadores aritméticos padrão +, -, \* e / podem ser aplicados à valores numéricos como um resultado de uma consulta SQL.
- ❑ Por exemplo:
  - Recupere todos os empregados (nome e sobrenome) e seus respectivos salários que trabalham no projeto 'ProdutoX' com um aumento de 10%.

## Em SQL

```
SELECT PNO, PNO, 1.1 * SALARIO
FROM EMPREGADO, TRABALHA-EM, PROJETO
WHERE ID=EID AND PNO=PNUMERO AND PNO='ProdutoX';
```

# Order By

---

- ❑ A cláusula ORDER BY é usada para ordenar tuplas resultantes de uma consulta com base nos valores de alguns atributos.
- ❑ Por exemplo:
  - Recuperar a lista de empregados e dos projetos em que eles trabalhem, ordenados pelo departamento do empregado e cada departamento ordenado alfabeticamente pelo sobrenome do empregado.

## Em SQL

```
SELECT D.DNOME, E.SNOME, E.PNOME, P.PNOME
FROM DEPARTAMENTO D, EMPREGADO E, TRABALHA-EM W, PROJETO P
WHERE D.DNUMERO=E.DNUM AND E.ID=W.EID AND W.PNO=P.PNUMERO
ORDER BY D.DNOME, E.SNOME;
```

# Order By

---

- ❑ A ordem padrão é ascendente, mas podemos utilizar a palavra-chave **DESC** para especificar que queremos a ordem decendente.
- ❑ A palavra-chave **ASC** pode ser usada para especificar explicitamente a ordem ascendente.

## Em SQL

```
SELECT D.DNOME, E.SNOME, E.PNOME, P.PNOME
FROM DEPARTAMENTO D, EMPREGADO E, TRABALHA-EM W, PROJETO P
WHERE D.DNUMERO=E.DNUM AND E.ID=W.EID AND W.PNO=P.PNUMERO
ORDER BY D.DNOME ASC, E.SNOME ASC;
```

# Questões

---

- ❑ Refaça os exercícios de álgebra relacional utilizando, agora, a linguagem SQL.

Sugestão: Utilize o WinRDBI para validar as consultas (<http://www.eas.asu.edu/~winrdbi/>).