

# Linguagem SQL - DML

Base de Dados - 2023/24

Carlos Costa

# SQL DML - Introdução

- DML - Data Manipulation Language
- Os comandos SQL DML permitem:
  - Inserir, eliminar e atualizar dados
  - Efetuar consultas:
    - Simples
    - Avançadas

# SQL DML

INSERT, DELETE e UPDATE

# Inserção - INSERT INTO

- Utilizado para inserir um novo tuplo numa relação.
  - Sintaxe 1: Não se indicam as colunas, tendo os valores inseridos de respeitar a ordem de criação dos atributos. Podemos utilizar os termos NULL ou DEFAULT:

```
INSERT INTO tablename VALUES (v1,v2,...,vn);
```

```
INSERT INTO EMPLOYEE VALUES  
( 'Richard', 'K', 'Marini', '653298653', NULL, '98  
Oak Forest, Katy, TX', 'M', 37000, '653298653', 4 );
```

- Sintaxe 2: Indicamos as colunas em que queremos inserir os dados. As restantes ficam com o seu valor nulo ou por defeito (caso tenha sido definido):

```
INSERT INTO tablename (A1,A4,A8,...,An) VALUES (v1,v4,v8,...,vn);
```

```
INSERT INTO EMPLOYEE (Dno, Fname, Lname, Ssn) VALUES  
(4, 'Richard', 'Marini', '653298653');
```

# Eliminação - DELETE

- Utilizado para remover um ou mais tuplos de uma relação.

```
DELETE FROM tablename WHERE match_condition;
```

```
-- remoção (potencial) de um tuplo:
```

```
DELETE FROM EMPLOYEE WHERE Ssn='123456789';
```

```
-- remoção (potencial) de n tuplos:
```

```
DELETE FROM EMPLOYEE WHERE Dno = 5;
```

```
-- ou
```

```
DELETE FROM EMPLOYEE WHERE Dno > 5 AND Dno < 8;
```

```
-- remoção de todos os tuplos da relação:
```

```
DELETE FROM EMPLOYEE;
```

Só afecta uma relação. No entanto, a ação pode propagar-se a outras relações<sup>5</sup> devido às definições de integridade referencial (on delete cascade).

# Actualização - UPDATE

- Utilizado para atualizar um ou mais tuplos de uma relação.

```
UPDATE tablename SET A1=v1,...,An=vn WHERE match_condition;
```

```
-- atualiza um tuplo:
```

```
UPDATE    PROJECT  
SET       Plocation = 'Bellaire', Dnum = 5  
WHERE     Pnumber=10;
```

```
-- atualização (potencial) de n tuplos:
```

```
UPDATE    EMPLOYEE  
SET       Salary = Salary * 1.1  
WHERE     Dno = 5;
```

Só afecta uma relação. No entanto, a ação pode propagar-se a outras relações devido às definições de integridade referencial (on update cascade).

# SQL DML

## Consultas Simples

# Operações com Conjuntos

- A linguagem SQL é baseada em operações de conjuntos e de álgebra relacional.
- No entanto, existem particularidades:
  - modificações e extensões
- SQL define formas de lidar com tuplos duplicados
  - Especifica quantas cópias dos tuplos aparecem no resultado.
    - Existem comandos para eliminar duplicados
  - Versões Multiconjunto de operadores (AR)
    - i.e. as relações podem ser multiconjuntos



# Projeção - SELECT FROM

- SELECT FROM

- Permite selecionar um conjunto de atributos (colunas) de uma ou mais tabelas.

$$\Pi_{\langle \text{attribute\_list} \rangle} (R1)$$

-- Forma Básica:

SELECT <attribute\_list> FROM <table\_list>;

SELECT \* FROM EMPLOYEE;

-- Todas as colunas

SELECT Fname, Ssn FROM EMPLOYEE;

-- Duas colunas

EMPLOYEE

Fname	Minit	Lname	<u>Ssn</u>	Bdate	Address	Sex	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	1

EMPLOYEE

Fname	<u>Ssn</u>
John	123456789
Franklin	333445555
Alicia	999887777
Jennifer	987654321
Ramesh	666884444
Joyce	453453453
Ahmad	987987987
James	888665555

# SELECT ALL vs DISTINCT


- Podemos seleccionar todos os tuplos ou eliminar os duplicados.
  - Tendo em atenção que, ao seleccionarmos só algumas colunas da tabela, o resultado pode não ser um conjunto (set) mas um multiconjunto.

```
-- Todos os tuplos (por defeito):
SELECT All <attribute_list> FROM <table_list>;


-- Eliminar tuplos repetidos:
SELECT DISTINCT <attribute_list> FROM
<table_list>;
```

```
SELECT ALL Salary FROM EMPLOYEE;
```

```
SELECT DISTINCT Salary FROM EMPLOYEE;
```



Salary
30000
40000
25000
43000
38000
25000
25000
55000



Salary
30000
40000
25000
43000
38000
55000

10

# Seleção - WHERE

- WHERE permite selecionar um subconjunto de tuplos da(s) tabela(s) de acordo com uma expressão condicional.

$$\Pi_{\langle \text{attribute\_list} \rangle}(\sigma_{\langle \text{condition} \rangle}(R1))$$

```
SELECT <attribute_list> FROM <table_list> WHERE <condition>;
```

```
SELECT Bdate, Address FROM EMPLOYEE
WHERE Fname='John' AND Minit='B' AND Lname='Smith';
```

A condição pode conter operadores de comparação (=, <, <=, >, >=, <>) e ser composta usando AND, OR e NOT.

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	1



<u>Bdate</u>	<u>Address</u>
1965-01-09	731Fondren, Houston, TX

# Renomeação - Relação, Atributo e Aritmética

- Podemos renomear:
  - relações e atributos;
  - resultado de uma operação aritmética.

## -- Renomear

-- Renomear Tabela\*

SELECT E.Fname, E.Ssn FROM EMPLOYEE AS E;

 $\rho_{R2}(R1)$ 

OU

SELECT E.Fname AS Fn, E.Ssn AS Ssname FROM EMPLOYEE AS E;

-- Renomear Atributo

SELECT Dno AS DepNumber FROM EMPLOYEE;

 $\rho_{B1, \dots, Bn}(R1)$ 
 $\rho_{R2(B1, \dots, Bn)}(R1)$ 

-- Renomear Resultado de Operação Aritmética\*\*

SELECT Salary \* 0.35 AS SalaryTaxes FROM EMPLOYEE;

\* ver mais à frente a importância de renomear tabelas em operações de junção.

\*\* qual o resultado de não renomear? Depende de SGBD. SQL Server não dá nome à coluna!!!

# Reunião, Intersecção e Diferença

- Requisitos:
  - as duas relações têm de ter o mesmo número de atributos.
  - o domínio de cada atributo deve ser compatível.
- Operadores SQL:
  - **UNION**, **INTERSECT** e **EXCEPT**
    - devem ser colocados entre duas queries.
    - tuplos duplicados são eliminados.
- Para manter os tuplos duplicados devemos utilizar as suas versões multiconjunto.
  - **UNION ALL**, **INTERSECT ALL\*** e **EXCEPT**

$$R1 \cup R2$$

$$R1 \cap R2$$

$$R1 - R2$$

\* Não disponível em SQL SERVER

# UNION - Exemplo

- Quais os projetos (número) que têm um funcionário **ou** um gestor do departamento que controla o projeto com o último nome Smith?

```
SELECT FROM .....  
UNION (ALL)  
SELECT FROM .....
```

```
(SELECT DISTINCT Pnumber  
FROM    PROJECT, DEPARTMENT, EMPLOYEE  
WHERE   Dnum=Dnumber AND Mgr_ssn=Ssn AND Lname='Smith' )  
UNION  
(SELECT DISTINCT Pnumber  
FROM    PROJECT, WORKS_ON, EMPLOYEE  
WHERE   Pnumber=Pno AND Essn=Ssn AND Lname='Smith' );
```

# Produto Cartesiano

- Podemos utilizar mais do que uma relação na instrução SELECT FROM.
- O resultado é o produto cartesiano dos dois conjuntos.

$R1 \times R2 \times \dots \times R_N$

```
SELECT * FROM table1, table2, ..., tableN;
```

```
-- Exemplo de Produto Cartesiano
```

```
SELECT * FROM EMPLOYEE, DEPARTMENT;
```

```
-- Exemplo de Produto Cartesiano só com dois atributos
```

```
-- >> Pode ser visto com Prod. Cartesiano seguido de Projeção
```

```
SELECT Ssn, Dname FROM EMPLOYEE, DEPARTMENT;
```

# Junção de Relações - WHERE

- O Produto Cartesiano tem pouco interesse prático...
- No entanto, a associação do operador WHERE permite a junção de relações.

```
SELECT <attribute_list> FROM <table_list> WHERE <join_condition>;
```

-- Exemplo de “*select-project-join query*”

```
SELECT Fname, Lname, Address
FROM   EMPLOYEE, DEPARTMENT
WHERE  Dname='Research' AND Dnumber=Dno;
```

ANSI SQL 89

<u>Fname</u>	<u>Lname</u>	<u>Address</u>
John	Smith	731 Fondren, Houston, TX
Franklin	Wong	638 Voss, Houston, TX
Ramesh	Narayan	975 Fire Oak, Humble, TX
Joyce	English	5631 Rice, Houston, TX

EMPLOYEE

<u>Fname</u>	<u>Minit</u>	<u>Lname</u>	<u>Ssn</u>	<u>Bdate</u>	<u>Address</u>	<u>Sex</u>	<u>Salary</u>	<u>Super_ssn</u>	<u>Dno</u>
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

→ Join Condition

DEPARTMENT

<u>Dname</u>	<u>Dnumber</u>	<u>Mgr_ssn</u>	<u>Mgr_start_date</u>
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19



# Junção de 3 Relações - Exemplo

- Caso com três relações e duas *join conditions*:

*/\* Questão: Para cada projeto localizado em 'Stafford', queremos saber o seu número, o número do departamento que o controla e último nome, endereço e data de nascimento do gestor desse departamento. \*/*

```
SELECT Pnumber, Dnum, Lname, Address, Bdate
FROM   EMPLOYEE, DEPARTMENT, PROJECT
WHERE  Dnum=Dnumber AND Mgr_ssn=Ssn AND Plocation='Stafford';
```

<u>Pnumber</u>	<u>Dnum</u>	<u>Lname</u>	<u>Address</u>	<u>Bdate</u>
10	4	Wallace	291 Berry, Bellaire, TX	1941-06-20
30	4	Wallace	291 Berry, Bellaire, TX	1941-06-20



  
 Join Condition 1
  Join Condition 2

EMPLOYEE

Fname	Minit	Lname	<u>Ssn</u>	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

PROJECT

Pname	<u>Pnumber</u>	Plocation	<u>Dnum</u>
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4

DEPARTMENT

<u>Dname</u>	<u>Dnumber</u>	<u>Mgr_ssn</u>	<u>Mgr_start_date</u>
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

# Junção - Ambiguidade de Nomes de Atributos

- Quando existem nomes de atributos iguais em distintas relações da junção, podemos utilizar o *full qualified name (fqn)*:

`relation_name.attribute`

```
/* Exemplo: Vamos pegar num dos exemplos anteriores e imaginar  
que o atributo Dno de EMPLOYEE se chamava Dnumber... */
```

```
SELECT Fname, Lname, Address  
FROM   EMPLOYEE, DEPARTMENT  
WHERE  Dname='Research' AND EMPLOYEE.Dnumber=DEPARTMENT.Dnumber;
```

Podemos também utilizar o fqn em situações em que não há ambiguidade de nomes.

# Junção - Ambiguidade + Renomeação

- Há situações em que ambiguidade de nomes de atributos resulta de termos uma relação recursiva.
- Nesta situação temos de renomeação as relações (*alias*).

```
/* Exemplo: Para cada Funcionário, pretendemos obter o seu primeiro e último nome, assim como do seu supervisor. */
```

```
SELECT E.Fname, E.Lname, S.Fname, S.Lname  
FROM   EMPLOYEE AS E, EMPLOYEE AS S  
WHERE  E.Super_ssn=S.Ssn;
```

Muitas vezes a renomeação envolvendo várias relações ajuda a melhorar a legibilidade da instrução.

# Queries - Comparação de Strings

- Operador LIKE permite comparar *Strings*
- Podemos utilizar wildcards.
  - % - significa zero ou mais caracteres.
  - \_ - significa um qualquer carácter.

## Exemplos:

```
/* Obter o primeiro e último nome dos funcionários cujo endereço contém a substring 'Houston,TX'. */  
SELECT    Fname, Lname  
FROM      EMPLOYEE  
WHERE     Address LIKE '%Houston,TX%';
```

```
/* Obter o primeiro e último nome dos funcionários nascidos nos anos 50 */  
SELECT    Fname, Lname  
FROM      EMPLOYEE  
WHERE     Bdate LIKE '_ _ 5 _ _ _ _ _ _ _';
```

# Queries - Comparação de Strings

- Podemos pesquisar os próprios wildcards na string.
  - Para isso utilizamos um carácter especial a preceder o wildcard
  - Devemos definir esse carácter com a instrução ESCAPE

## LIKE ... ESCAPE

```
/* Nome dos funcionários cujo endereço contém a substring 'Houston%,TX'. */
SELECT    Fname, Lname
FROM      EMPLOYEE
WHERE     Address LIKE '%Houston@%,TX%' ESCAPE '@';
```

- Alguns SGBD permite utilizar outros Wildcards.

Description	SQL Wildcard	MS-DOS Wildcard	Example
Any number (zero or more) of arbitrary characters	%	*	'Able' LIKE 'A%'
One arbitrary character	_	?	'Able' LIKE 'Abl_'
One of the enclosed characters	[ ]	n/a	'a' LIKE '[a-g]' 'a' LIKE '[abcdefg]'
Match not in range of characters	[ ^ ]	n/a	'a' LIKE '[ ^ w-z]' 'a' LIKE '[ ^ wxyz]'

SQL SERVER

# Queries - Operadores Aritméticos e BETWEEN

- Operações Aritméticas:
  - Operadores: adição (+), subtração (-), multiplicação (\*), divisão (/)
  - Operandos: valores numéricos ou atributos com domínio numérico.
- BETWEEN
  - Verificar se um atributo está entre uma gama de valores.

## Exemplos:

```
/* Obter o salário, com um aumento de 10%, de todos os trabalhadores do projeto GalaxyS. */
```

```
SELECT    E.Fname, E.Lname, 1.1 * E.Salary AS Increased_sal
FROM      EMPLOYEE AS E, WORKS_ON AS W, PROJECT AS P
WHERE     E.Ssn=W.Essn AND W.Pno=P.Pnumber AND P.Pname='GalaxyS';
```

```
/* Funcionários do departamento nº 5 com salário entre 3k e 4k */
SELECT * FROM EMPLOYEE
WHERE (Salary BETWEEN 30000 AND 40000) AND Dno = 5;
```

# Queries - Ordenação de Resultados

- Podemos ordenar os resultados segundo uma ou mais colunas.
- Sintaxe: **ORDER BY** A1, ..., Ak
  - A1, ..., Ak - atributos a ordenar.
  - 1,2,...,k - também podemos usar o número da coluna
- Podemos definir se é ascendente (ASC) ou descendente (DESC).
  - Por omissão as colunas são ordenadas ascendentemente.

## Exemplo:

```
/* Lista de funcionários e projetos em que trabalham, ordenado por
departamento e, dentro deste, pelo último nome (descendente) e depois o
primeiro */
```

```
SELECT      D.Dname, E.Lname, E.Fname, P.Pname
FROM        DEPARTMENT AS D, EMPLOYEE AS E, WORKS_ON AS W, PROJECT AS P
WHERE       D.Dnumber= E.Dno AND E.Ssn= W.Essn AND W.Pno= P.Pnumber
ORDER BY   D.Dname, E.Lname DESC, E.Fname;
```

```
/* ... ORDER BY 1, 2 DESC, 3; */
```

# SQL DML

## Consultas Avançadas



# Tratamento dos NULL

- NULL
  - significa um valor desconhecido ou que não existe.
- SQL tem várias regras para lidar com os valores null.
- O resultado de uma expressão aritmética com *null* é *null*:  $5 + null$  é *null*
- Temos possibilidade de verificar se determinado atributo é nulo: IS NULL
- Por norma, as funções de agregação ignoram o null.

# IS (NOT) NULL - Exemplo

- **IS NULL**: selecionar tuplos com determinado atributo a NULL;
- **IS NOT NULL**: selecionar tuplos com determinado atributo diferente de NULL;

## Exemplos:

```
-- IS NOT NULL  
SELECT * FROM EMPLOYEE  
WHERE Super_ssn IS NOT NULL;
```

```
-- IS NULL  
SELECT * FROM EMPLOYEE  
WHERE Super_ssn IS NULL;
```

EMPLOYEE									
Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4

Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5

# Junções - JOIN ON

- WHERE

- Já vimos que o produto cartesiano associado ao operador “where” permite juntar várias relações. (ANSI SQL 89)

- ANSI SQL 92: JOIN ON

utilizar sempre a partir de agora...

- Permite especificar simultaneamente as tabelas a juntar e a condição de junção.

$R \bowtie_c S$

```
SELECT ... FROM (.. [INNER] JOIN .. ON ..) ...;
-- [INNER] é opcional
```

```
-- exemplo de Equi-join:
SELECT   Fname, Lname, Address
FROM     (EMPLOYEE JOIN DEPARTMENT ON Dno=Dnumber)
WHERE    Dname='Research';
```

# NATURAL JOIN

- Junção Natural - os atributos de junção têm todos o mesmo nome nas duas relações.
- Os atributos repetidos são removidos.
- Podemos renomear os atributos de uma relação para permitir a junção natural.

R ⋈ S

```
SELECT ... FROM (... NATURAL JOIN ...) WHERE <condition>;
```

```
-- exemplo de Natural Join com renomeação:  
SELECT  Fname, Lname, Address  
FROM    (EMPLOYEE NATURAL JOIN  
          (DEPARTMENT AS DEPT (Dname, Dno, Mssn, Msdate)))  
WHERE   Dname='Research';
```

Não disponível em  
SQL Server!

# OUTER JOIN

- As junções externas podem ser à esquerda, à direita ou totais (LEFT, RIGHT, FULL).

```
SELECT .. FROM (.. LEFT|RIGHT|FULL [OUTER] JOIN ..) ...;
```

/\* exemplo de Outer Join com renomeação das relações e atributos \*/

```
SELECT E.Lname AS Employee_name, S.Lname AS Supervisor_name
FROM (EMPLOYEE AS E LEFT OUTER JOIN EMPLOYEE AS S
      ON E.Super_ssn=S.Ssn);
```

```
-- RIGHT OUTER JOIN
-- FULL OUTER JOIN
```

$$R \bowtie_{A1=B2} S$$

$$R \bowtie_{A1=B2} S$$

$$R \bowtie_{A1=B2} S$$

# JOIN - Encadeamento

- Podemos ter várias operações JOIN encadeadas envolvendo 3..N relações.
  - uma das relações da junção resulta de outra operação de junção.

```
SELECT .. FROM (.. JOIN .. JOIN .. JOIN ..) ...;
```

```
/* Exemplo do slide 17: Para cada projeto localizado em  
'Stafford', queremos saber o seu número, o número do  
departamento que o controla e último nome, endereço e data de  
nascimento do gestor desse departamento. */
```

```
-- Nota: Neste caso as join conditions estão à frente do ON
```

```
SELECT    Pnumber, Dnum, Lname, Address, Bdate  
FROM      ((PROJECT JOIN DEPARTMENT ON Dnum=Dnumber)  
           JOIN EMPLOYEE ON Mgr_ssn=Ssn)  
WHERE     Plocation='Stafford';
```

# Agregações

- Funções de agregação introduzidas em álgebra relacional.
- Funções de Agregação
  - Exemplos\*: COUNT, SUM, MAX, MIN, AVG
  - Em geral, não são utilizados os tuplos com valor NULL no atributo na função.
- Efetuar agregação por atributos
  - GROUP BY <grouping attributes>
- Efetuar seleção sobre dados agrupados
  - HAVING <condition>

\* Existem outras funções de agregação específicas do SGBD

# Funções de Agregação - Exemplo

## Exemplos... sem agrupamento de atributo(s)

```
/* Exemplo 1: relativamente aos salários dos funcionários, obter  
o valor total, o máximo, o mínimo e o valor médio */  
SELECT  SUM (Salary), MAX (Salary), MIN (Salary), AVG (Salary)  
FROM    EMPLOYEE;
```

```
/* Exemplo 2: N° de funcionários do departamento 'Research' */  
SELECT  COUNT (*)  
FROM    EMPLOYEE JOIN DEPARTMENT ON DNO=DNUMBER  
WHERE   DNAME='Research';
```

```
/* Exemplo 3: N° de vencimentos distintos */  
SELECT  COUNT (DISTINCT Salary)  
FROM    EMPLOYEE;
```

Nota1: O operador COUNT(A1) conta o número de valores não NULL do atributo A1.  
O operador COUNT(\*) conta o número de linhas.

Nota2: Min, Max, Count(...) e Count(\*) podem ser utilizadas com qualquer tipo de dados. SUM e AVG só podem ser aplicadas a campos numéricos.



# Agregação (GROUP BY) - Exemplo

## Exemplos... agregação de atributo(s)

/\* Exemplo 1: para cada departamento, obter o seu número, o número de funcionários e a sua média salarial \*/

```
SELECT    Dno, COUNT(*), AVG(Salary)
FROM      EMPLOYEE
GROUP BY  Dno;
```

Os “grouping attributes” devem aparecer na cláusula SELECT  
Exemplo: Dno

Fname	Minit	Lname	Ssn	...	Salary	Super_ssn	Dno
John	B	Smith	123456789		30000	333445555	5
Franklin	T	Wong	333445555		40000	888665555	5
Ramesh	K	Narayan	666884444		38000	333445555	5
Joyce	A	English	453453453	...	25000	333445555	5
Alicia	J	Zelaya	999887777		25000	987654321	4
Jennifer	S	Wallace	987654321		43000	888665555	4
Ahmad	V	Jabbar	987987987		25000	987654321	4
James	E	Bong	888665555		55000	NULL	1

Dno	Count (*)	Avg (Salary)
5	4	33250
4	3	31000
1	1	55000

Result of Q24

/\* Exemplo 2: agregação com junção de duas relações \*/

```
SELECT    Pnumber, Pname, COUNT(*)
FROM      PROJECT JOIN WORKS_ON ON Pnumber=Pno
GROUP BY  Pnumber, Pname;
```

Nota: Se existirem valores NULL nos “grouping attribute”, então é criado um grupo com todos os tuplos contendo NULL nesses atributos.

# Agregação (GROUP BY.. HAVING) - Exemplo

Exemplo... agregação de atributo(s) com seleção

/\* Exemplo 1: Para cada projeto, com mais de dois funcionários, obter o seu nome e nº de funcionários que trabalham no projeto \*/

```
SELECT      Pname, COUNT(*)
FROM        PROJECT join WORKS_ON
            ON Pnumber=Pno

GROUP BY    Pname
HAVING      COUNT(*) > 2;
```

Junção

Pname	Count (*)
ProductY	3
Computerization	3
Reorganization	3
Newbenefits	3

Pname	Pnumber	...	Essn	Pno	Hours
ProductX	1		123456789	1	32.5
ProductX	1		453453453	1	20.0
ProductY	2		123456789	2	7.5
ProductY	2		453453453	2	20.0
ProductY	2		333445555	2	10.0
ProductZ	3		666884444	3	40.0
ProductZ	3		333445555	3	10.0
Computerization	10	...	333445555	10	10.0
Computerization	10		999887777	10	10.0
Computerization	10		987987987	10	35.0
Reorganization	20		333445555	20	10.0
Reorganization	20		987654321	20	15.0
Reorganization	20		888665555	20	NULL
Newbenefits	30		987987987	30	5.0
Newbenefits	30		987654321	30	20.0
Newbenefits	30		999887777	30	30.0

Nota1: A condição da cláusula WHERE é aplicada antes da criação dos grupos. A condição do HAVING é executada depois da criação dos grupos.

Nota2: Na cláusula HAVING só podemos ter atributos que aparecem em GROUP BY ou funções de agregação.

# Agregação - Resumo

## SQL

```

SELECT      A1,...,An, FAgr1,..Fagrh
FROM        R1,R2,...,Rm
WHERE       <condition_W>
GROUP BY    A1,...,An
HAVING      <condition_H>;
    
```

Expressão equivalente em álgebra relacional

$$\Pi_{A1,...,An, FAgr1,..Fagrh} (\sigma_{<condition_H>} (A1,..., An \Join_{FAgr1,..,Fagrh} (\sigma_{<condition_W>} (R1 \times .. \times Rm))))$$

Importante para se perceber  
a ordem das operações

# SubConsultas (SubQueries)

- É possível usar o resultado de uma query, i.e. uma relação, noutra query.
  - Nested Queries
- Subconsultas podem aparecer na cláusula:
  - FROM - entendidas como cálculo de relações auxiliares.
  - WHERE - efetuar testes de pertença a conjuntos, comparações entre conjuntos, calcular a cardinalidade de conjuntos, etc.

# Cláusula FROM - Subquery como Tabela

- Podemos utilizar o resultado de uma subquery como uma tabela na cláusula FROM, dando-lhe um nome (alias).

## Exemplo... agregação de atributo(s) com selecção

```
/* Exemplo 1: Obter uma lista de funcionários com mais de dois dependentes */  
SELECT      Fname, Minit, Lname, Ssn  
FROM        Employee JOIN (  
    SELECT   Essn  
    FROM     DEPENDENT  
    GROUP BY Essn  
    HAVING   count(Essn)>2) AS Dep  
ON Ssn=Dep.Essn;
```

# Operador IN - Pertença a Conjunto

- WHERE **A1,...,An** IN (SELECT **B1,...,Bn** FROM ...)
  - Permite selecionar os tuplos em que os atributos indicados (A1,...,An) existem na subconsulta.
  - B1,...,Bn são os atributos retornados pela subconsulta
- A1,...,An e B1,...,Bn
  - têm de ter o mesmo número atributos e domínios compatíveis.
- NOT IN
  - permite obter o resultado inverso.

# Operador IN - Exemplo

## Exemplos...

/\* Exemplo 1: Obter o nome de todos os funcionários que não têm dependentes \*/

```
SELECT      Fname, Minit, Lname
FROM        EMPLOYEE
WHERE       Ssn NOT IN (SELECT Essn FROM DEPENDENT);
```

/\* Exemplo 2: Obter o Ssn de todos os funcionários que trabalham no mesmo projeto, e o mesmo número de horas, que o funcionário com o Ssn = '123456789'\*/

```
SELECT      DISTINCT Essn
FROM        WORKS_ON
WHERE       (Pno, Hours) IN ( SELECT      Pno, Hours
                              FROM        WORKS_ON
                              WHERE       Essn='123456789' );
```

SQL Server não  
suporta múltiplas  
colunas!

/\* Exemplo 3: Obter o Ssn de todos os funcionários que trabalham no projeto nº 1, 2 ou 3 \*/

```
SELECT      DISTINCT Essn
FROM        WORKS_ON
WHERE       Pno IN (1, 2, 3);
```

# Comparação de Conjuntos

- Existem **operadores** que pode ser utilizados para **comparar** um **valor simples** (tipicamente um atributo) **com** um **set** ou **multiset** (tipicamente uma subquery).
- **ANY (= CASE)**
  - Permite selecionar os resultados cujos atributos indicados sejam iguais (=), maiores (>), menores(<) ou diferentes (<>) do que pelo menos um tuplo da subquery.
  - =ANY é o mesmo que IN
- **ALL**
  - Também pode ser combinada com os operadores iguais (=), maiores (>), menores(<) ou diferentes (<>).



# ANY e ALL - Exemplos

## Exemplos...

/\* Exemplo 1: Obter o nome dos funcionários cujo salário é maior do que o salário de todos os trabalhadores do departamento 5 \*/

```
SELECT      Lname, Fname
FROM        EMPLOYEE
WHERE       Salary > ALL ( SELECT  Salary
                           FROM    EMPLOYEE
                           WHERE    Dno=5);
```

/\* Exemplo 2: Obter o nome dos funcionários cujo salário é maior do que o salário de algum trabalhador do departamento 5 \*/

```
SELECT      Lname, Fname
FROM        EMPLOYEE
WHERE       Salary > ANY ( SELECT  Salary
                           FROM    EMPLOYEE
                           WHERE    Dno=5);
```

# Teste de Relações Vazias - EXISTS

- O operador EXISTS retorna
  - TRUE, se subconsulta não é vazia.
  - FALSE, se subconsulta é vazia.
- Existe a possibilidade de utilizar o NOT EXISTS

## SQL – (NOT) EXISTS

```
/* Exemplo 1: Nomes dos funcionários que não têm dependentes */  
SELECT      Fname, Lname  
FROM        EMPLOYEE  
WHERE       NOT EXISTS ( SELECT *  
                        FROM   DEPENDENT  
                        WHERE   Ssn=Essn );
```

# Existem Tuplos Duplicados? - UNIQUE

- Unique permite verificar se o resultado de uma subconsulta possui tuplos duplicados.
- Permite verificar se determinado resultado (relação) é um conjunto ou um multiconjunto.

## SQL - (NOT) EXISTS

```
/* Exemplo 1: Nomes dos funcionários que gerem um departamento.  
(supondo que o mesmo funcionário pode gerir mais do que um  
departamento...) */  
SELECT      Fname, Lname  
FROM        EMPLOYEE  
WHERE       UNIQUE      ( SELECT  Mgr_ssn  
                           FROM    DEPARTMENT  
                           WHERE    Ssn=Mgr_ssn );
```

Não disponível em  
SQL Server!


# SubConsultas Não Correlacionadas

- A subquery (query interior) não depende de dados lhe são fornecidos pela query exterior.
  - Nestes casos, a query interior é executada uma única vez e o resultado é utilizado no SELECT exterior.

## SubConsulta Correlacionada

/\* Exemplo 1: Nome dos funcionários que são gestores de departamento \*/

```
SELECT    Fname, Lname
FROM      EMPLOYEE
WHERE     Ssn IN ( SELECT Mgr_ssn
                   FROM    DEPARTMENT
                   WHERE    Mgr_ssn IS NOT NULL);
```



# SubConsultas Correlacionadas


- A subquery (query interior) depende de dados lhe são fornecidos pela query exterior.
  - Nestes casos, a query interior é executada uma vez para cada resultado do SELECT exterior.

## SubConsulta Correlacionada

/\* Exemplo 1: Nome dos funcionários que tem um dependente com o primeiro nome e sexo igual ao próprio funcionário \*/

```

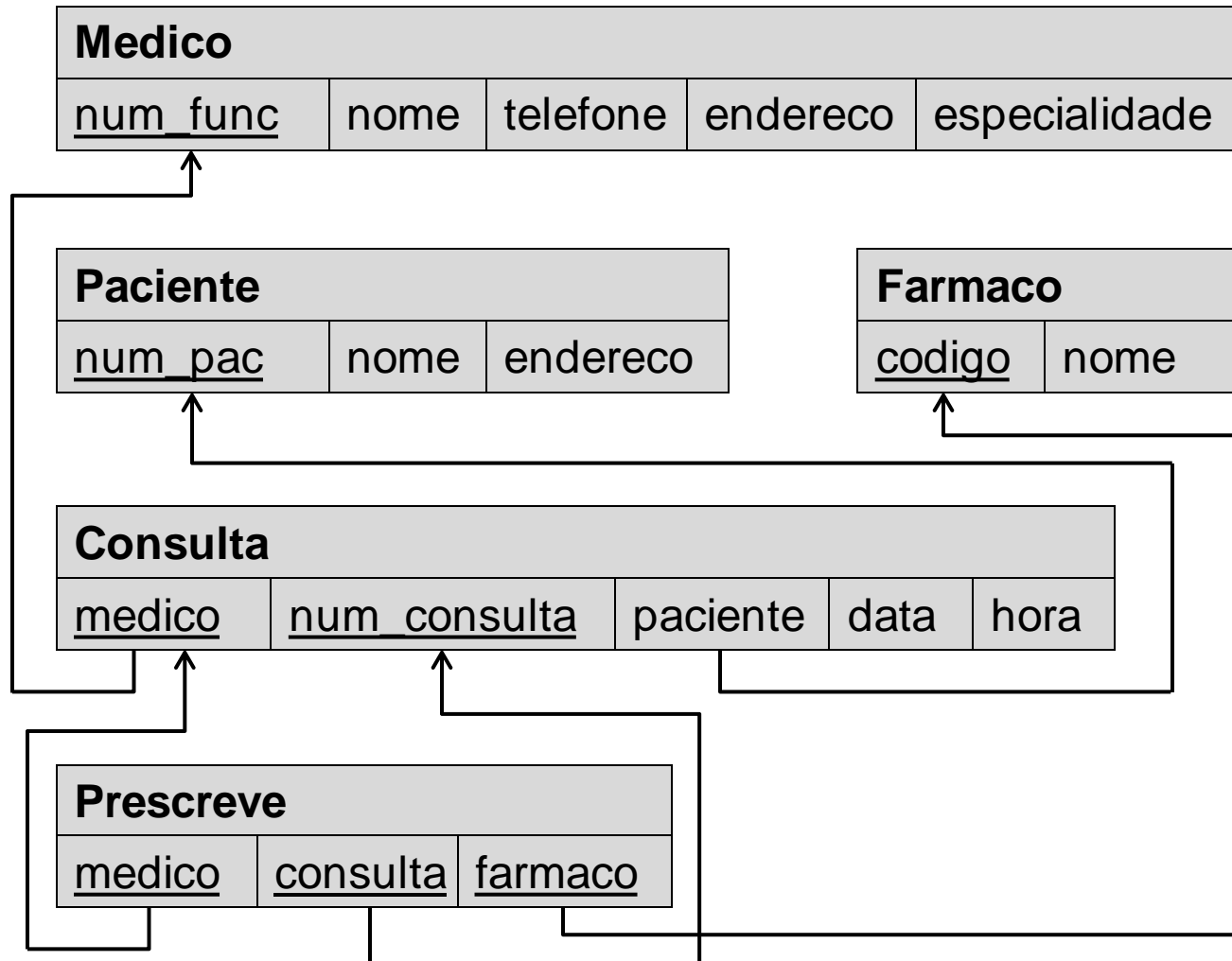
SELECT      E.Fname, E.Lname
FROM        EMPLOYEE AS E
WHERE       E.Ssn IN (
              SELECT Essn
              FROM    DEPENDENT AS D
              WHERE   E.Fname=D.Dependent_name
                    AND E.Sex=D.Sex );
  
```



# SQL DML - Caso de Estudo

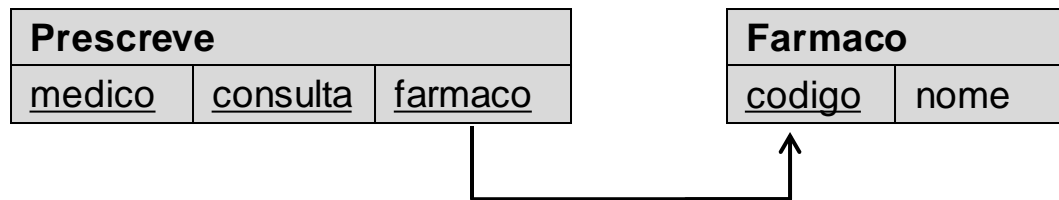
Clínica  
(Conversão das Queries AR para SQL)

# Clínica - Esquema Relacional da BD



# Clínica - Problema 1

- O nome dos fármacos que nunca foram prescritos



$$\Pi_{\text{nome}}(\sigma_{\text{farmaco}=\text{null}}(\text{Prescreve} \bowtie_{\text{farmaco}=\text{codigo}} \text{Farmaco}))$$

SQL...

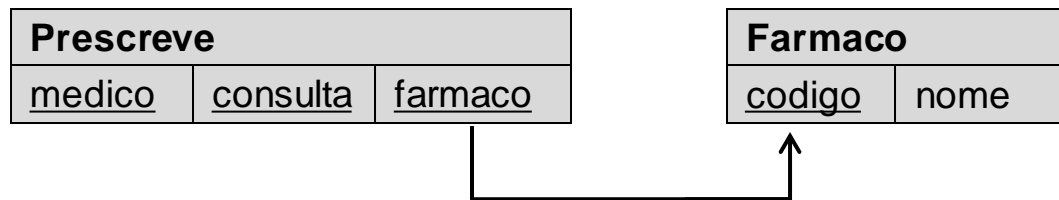
```

SELECT    nome
FROM      Prescreve RIGHT OUTER JOIN Farmaco ON farmaco=codigo
WHERE     farmaco IS NULL;
    
```



# Clínica - Problema 2

- O número de fármacos prescritos em cada consulta



$\Pi_{\text{medico, consulta, num\_farm=count(farmaco)}} (\text{Prescreve})$

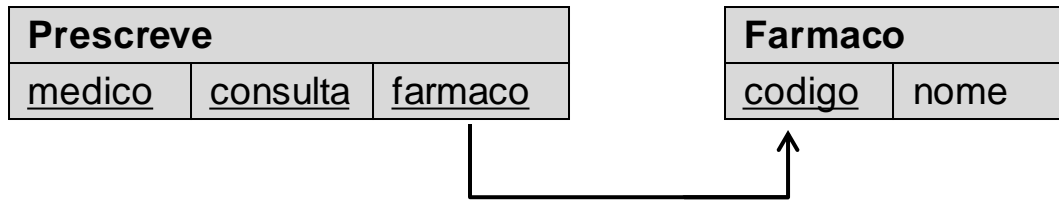
SQL...

```

SELECT    medico, consulta, count(farmaco) AS num_farm
FROM      Prescreve
GROUP BY  medico, consulta;
    
```

# Clínica - Problema 3

- Para cada médico, a quantidade média de fármacos receitados por consulta



$\Pi_{\text{medico, avg\_farmaco=avg(num\_farm)}} (\Pi_{\text{medico, consulta, num\_farm=count(farmaco)}} (\text{Prescreve}))$

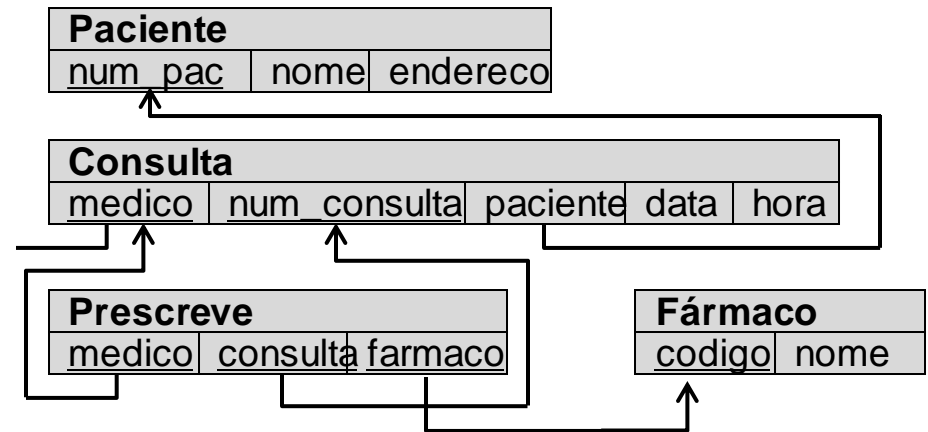
SQL...

```

SELECT      medico, avg(num_farm) AS avg_farmaco
FROM        (SELECT      medico, consulta, count(farmaco) AS num_farm
              FROM        Prescreve
              GROUP BY    medico, consulta) AS T
GROUP BY    medico;
  
```

# Clínica - Problema 4

- O nome de todos os fármacos prescritos, incluindo a quantidade, para o paciente número 35312161



$temp \leftarrow \pi_{medico, num\_consulta} (\sigma_{paciente=35312161} (Consulta))$

$temp2 \leftarrow \pi_{farmaco, quantidade=count(farmaco)} (temp \bowtie_{medico=medico \text{ AND } num\_consulta=consulta} Prescreve)$

$\pi_{nome, quantidade} (temp2 \bowtie_{farmaco=codigo} Farmaco)$

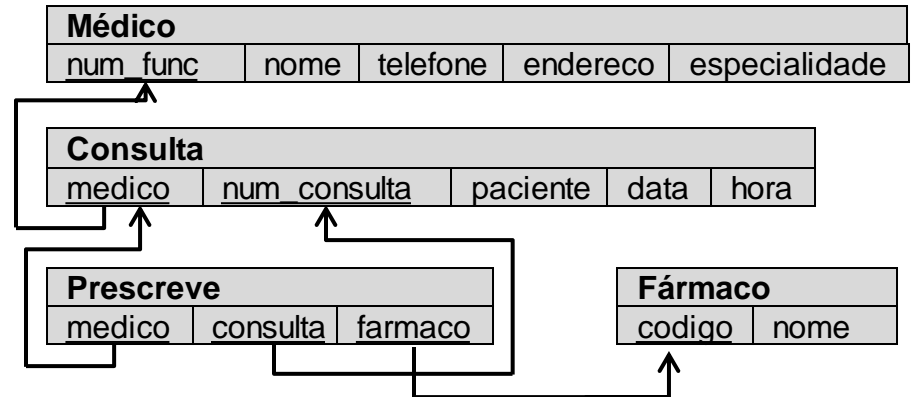
SQL...

```

SELECT nome, quantidade
FROM   Farmaco JOIN (SELECT  farmaco, count(farmaco) AS quantidade
                     FROM    Prescreve AS JOIN (SELECT medico, num_consulta
                                                FROM      Consulta
                                                WHERE     paciente=35312161) AS T
                     ON (P.medico=T.medico AND num_consulta=consulta) AS T2
                     GROUP BY farmaco)
ON     farmaco=codigo;
  
```

# Clínica - Problema 5

- O nome dos fármacos que já foram prescritos por todos os médicos da clínica



$$\text{temp} \leftarrow \rho_{\text{codigo}, \text{num\_func}}(\pi_{\text{farmaco}, \text{medico}}(\text{Prescreve})) \div \pi_{\text{num\_func}}(\text{Medico})$$

$$\pi_{\text{nome}}(\text{temp} \bowtie \text{Farmaco})$$

÷ não existe em SQL

SQL... Uma Implementação Alternativa da Query:

```

SELECT      farmaco, count(DISTINCT medico) as num_medicos
FROM        Prescreve
GROUP BY    farmaco
HAVING      count(DISTINCT medico)=(SELECT count(*) from Medico);
  
```

# A Seguir?

## Data Operations – Relational Algebra

r		s		r ∪ s	
A	B	A	B	A	B
α	1	α	2	α	1
α	2	β	3	α	2
β	1			β	1
				β	3



$\Pi_{\text{nome}}(\sigma_{\text{famaco}=\text{null}}(\text{Prescreve} \bowtie_{\text{farmaco}=\text{codigo}} \text{Farmaco}))$



## SQL – Describe Database Schema

```
CREATE TABLE DEPARTMENT
( Dname          VARCHAR(15)      NOT NULL,
  Dnumber        INT              NOT NULL,
  Mgr_ssn        CHAR(9)         NOT NULL,
  Mgr_start_date DATE,
  PRIMARY KEY (Dnumber),
  UNIQUE (Dname),
  FOREIGN KEY (Mgr_ssn) REFERENCES EMPLOYEE(Ssn) );
```

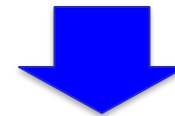
## SQL – Data Manipulation

### SQL query:

```
SELECT  Pnumber, Pname, COUNT(*)
FROM    PROJECT, WORKS_ON
WHERE   Pnumber=Pno
GROUP BY Pnumber, Pname;
```

### SQL query:

```
INSERT INTO EMPLOYEE (Fname,
  Lname, Ssn, Dno) VALUES ('Robert',
  'Hatcher', '980760540', 2);
```



### SQL View:

```
CREATE VIEW EMPLOYEE_DEP5 AS
  SELECT  Fname, Lname, Ssn, Dno
  FROM    EMPLOYEE
  WHERE   Dno=5
  WITH CHECK OPTION;
```

# Resumo

- SQL DML
- Inserir, eliminar e actualizar dados
- Efectuar pesquisas:
  - Simples
  - Avançadas
- Caso de Estudo