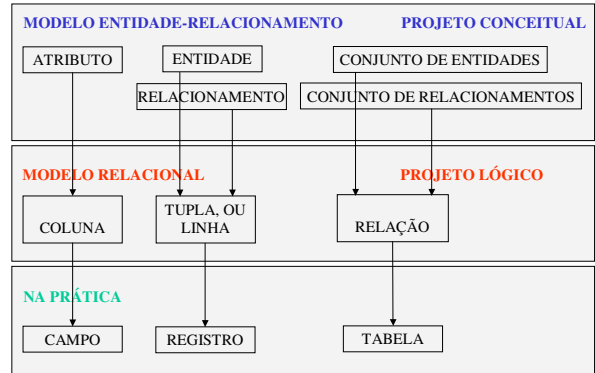


Prof. Omero Francisco Bertol
(omero@pb.cefetpr.br)

SQL- *Structured Query Language*
(Linguagem de Consulta Estruturada)

<http://pessoal.pb.cefetpr.br/omero/index.htm>

Sinônimos



SQL - Introdução

A álgebra relacional e o cálculo relacional são linguagens formais que proporcionam uma notação concisa para a representação de consultas. Entretanto, sistemas de banco de dados comerciais precisam de uma linguagem de consulta mais fácil para o usuário.

Embora seja definida como uma “linguagem de consulta” a **linguagem SQL** possui muitos outros recursos além de consulta ao banco de dados, como meios para a definição da estrutura de dados (esquema), para modificação de dados no banco de dados e para a especificação de restrições de segurança.

SQL - Histórico

Certamente a SQL tem representado o padrão para linguagens de banco de dados relacionais. Existem diversas versões de SQL. A versão original foi desenvolvida pela IBM no início dos anos 70.

Em 1986, o *American National Standards Institute* (ANSI) e a *International Standards Organization* (ISO) publicaram os padrões para a SQL, chamada de SQL-86. A IBM publicou seus próprios padrões para a SQL, a *Systems Application Architecture Database Interface* (SAA-SQL) em 1987. Uma extensão para o padrão SQL, a **SQL-89**, foi publicada em 1989. A versão em uso do padrão ANSI/ISO SQL é o padrão **SQL-92**.

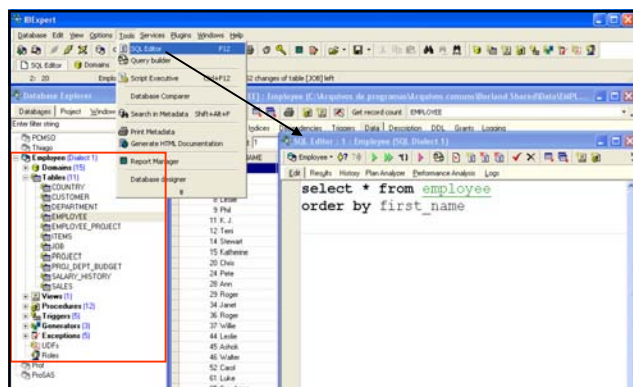
É importante lembrar que algumas implementações da SQL podem dar suporte “somente” à **SQL-89** e assim não aceitar a **SQL-92**.

As “Partes” da Linguagem SQL (1/2)

- **Linguagem de definição de dados (DDL- Data Definition Language):** comandos para a definição de esquemas de relações, exclusão de relações, modificação nos esquemas de relações e criação de domínios.
 - *create table, alter table, drop table e create domain*
- **Linguagem interativa de manipulação de dados (DML- Data Manipulation Language):** 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.
 - *select, insert, delete e update*

As “Partes” da Linguagem SQL (2/2)

- **Incorporação DML (Embedded DML):** comandos SQL incorporados foi projetada para aplicação em linguagens de programação de uso geral, como PL/I, Cobol, Pascal, Fortran e C.
- **Definição de visões:** a SQL DDL possui comandos para definição de visões (relação virtual).
- **Autorização:** a SQL DDL engloba comandos para especificação de direitos de acesso a relações e visões.
- **Integridade:** a SQL DDL possui comandos para especificação de regras de integridade que os dados armazenados no banco de dados devem satisfazer. Atualizações que violarem as regras de integridade serão desprezadas.
- **Controle de transações:** a SQL inclui comandos para a especificações de início e fim de transações. Algumas implementações também permitem explicitar bloqueios de dados para controle de concorrência.



Os exemplos do uso da linguagem SQL a seguir serão demonstrados, usando o banco de dados Firebird “Employee.gdb”, através do recurso SQL Editor disponível na ferramenta administrativa IBEExpert. O Firebird é um Sistema Gerenciador de Banco de Dados Cliente/Servidor Relacional que está baseado no padrão SQL ANSI-92.

DML- Data Manipulation Language

A “Linguagem interativa de manipulação de dados” abrange:

- **Select, From e Where:**
 - uma linguagem de consulta baseada tanto na álgebra relacional quanto no cálculo relacional de tuplas.
- **Insert, Delete e Update:**
 - engloba também comandos para inserção, exclusão e modificação de tuplas no banco de dados.

Consulta SQL Básica

A estrutura básica de uma consulta em SQL consiste em três cláusulas: **select**, **from** e **where**.

- A cláusula **select** corresponde à operação de projeção (π) 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 (\times) da álgebra relacional. Ela associa as relações que serão pesquisadas durante a evolução de uma expressão.
- A cláusula **where** corresponde à seleção (σ) do predicado da álgebra relacional. Ela consiste em um predicado envolvendo atributos da relação que aparece na cláusula **from**.

SELECT *versus* π (projeção) (1/2)

select A_1, A_2, \dots, A_n
from r_1, r_2, \dots, r_m
where P

Onde, cada A_i representa um atributo e cada r_i , uma relação. P é um predicado. A consulta acima é equivalente à seguinte expressão em álgebra relacional:

$\pi_{A_1, A_2, \dots, A_n}(\sigma_P(r_1 \times r_2 \times \dots \times r_m))$

SELECT *versus* π (projeção) (2/2)

Se a cláusula **where** for omitida, o predicado P é **verdadeiro**. Entretanto, de modo diferente das expressões em álgebra relacional, o resultado de uma consulta em SQL pode conter cópias múltiplas de algumas tuplas.

Semântica de uma consulta SQL:

A SQL forma um **produto cartesiano** das relações indicadas na cláusula **from**, executa uma **seleção** em álgebra relacional usando o predicado da cláusula **where** e, então, **projeta** o resultado sobre os atributos da cláusula **select**.

A Cláusula SELECT (1/6)

Encontre todas as tuplas da relação “employee”, ordenando o resultado pelo atributo “first_name”.

select * **from** employee
order by first_name

EMP_ID	FIRST_NAME	LAST_NAME	PHONE_EXT	HIRE_DATE	DEPT_ID	JOB_CODE	JOB_GRADE	JOB_COUNTRY	SALARY	FULL_NAME
20	John	Bennet	95	01.02.1991 00:00	120	Adm	15	England	22.535.000	Bennet, John
40	John	Panamathan	209	01.08.1991 00:00	821	Eng	20	USA	88.695.500	Panamathan, John
11	John	Patel	247	01.06.1993 00:00	823	Eng	25	USA	35.000.000	Patel, John
48	Bruce	Young	233	28.12.1988 00:00	821	Eng	25	USA	57.500.000	Young, Bruce
52	Carol	Neerstrom	420	02.10.1991 00:00	1180	Phel	40	USA	42.742.500	Neerstrom, Carol
23	Joe	Panathoulas	887	01.01.1990 00:00	871	Meq	30	USA	89.625.000	Panathoulas, Joe
72	Claudia	Sutherland		20.04.1992 00:00	1340	SRop	40	Canada	100.914.000	Sutherland, Claudia
60	Dana	Bishop	280	01.06.1992 00:00	821	Eng	30	USA	62.550.000	Bishop, Dana
134	Jacques	Gren		23.08.1993 00:00	122	Chop	40	France	296.500.000	Gren, Jacques
34	Jane	Balaban	2	21.03.1991 00:00	110	Sales	30	USA	61.637.800	Balaban, Jane
71	Jennifer M.	Balaban	289	15.04.1992 00:00	822	Eng	30	USA	53.167.500	Balaban, Jennifer M.
144	John	Montgomery	820	30.03.1994 00:00	812	Eng	50	USA	35.000.000	Montgomery, John

O asterisco “*” pode ser usado para denotar “todos os atributos” da relação selecionada. O resultado de uma consulta SQL é, naturalmente, uma relação.

A Cláusula SELECT

(2/6)

Encontre todas as tuplas da relação “employee”, com dupla ordenação: primeiro pelo atributo “dept_no” e as tuplas com valor igual para o atributo “dept_no” ficam ordenadas pelo atributo “first_name”.

```
select * from employee
order by dept_no, first_name
```

EMP_ID	FIRST_NAME	LAST_NAME	PHONE_EXT	HIRE_DATE	DEPT_NO	JOB_CODE	JOB_GRADE	JOB_COUNTRY	SALARY	FULL_NAME
105	Oliver H.	Blender	295	08.10.1992 00:00	000	CEO	10USA		212.950,00	Oliver H. Blender
12	Tim	Lee	296	01.05.1990 00:00	000	Adm	40USA		53.793,00	Tim Lee
80	May S.	MacDonald	407	01.06.1992 00:00	100	REP	20USA		111.262,50	MacDonald May S.
127	Michael	Fayeno	452	09.08.1993 00:00	100	REP	40USA		44.000,00	Fayeno Michael
34	Janet	Bladen	12	21.03.1991 00:00	110	Sal	30USA		63.637,87	Bladen Janet
63	Luke	Leung	1	18.02.1992 00:00	110	REP	40USA		68.805,00	Leung Luke
110	Leah	Vanaman	23	01.07.1993 00:00	115	REP	40USA		7.440,00	Vanaman Leah
110	Viki	Schala	32	04.02.1993 00:00	115	Rep	30USA		6.000,00	Schala Viki
20	Ann	Bennet	5	01.02.1991 00:00	120	Adm	50England		22.935,00	Bennet Ann
3	Steven	Peres		25.04.1991 00:00	120	Adm	50England		33.635,00	Peres Steven
37	Julie	Stambur	17	05.04.1991 00:00	120	Rep	40England		36.204,80	Stambur Julie

A Cláusula SELECT

(3/6)

Encontre todas as tuplas da relação “employee”, ordenando o resultado pelo atributo “first_name” e apresentando (ou projetando) somente os atributos: “first_name”, “last_name”, “dept_no”.

```
select first_name, last_name, dept_no
from employee
order by first_name
```

FIRST_NAME	LAST_NAME	DEPT_NO
Ann	Bennet	120
Ashuk	Bennethan	621
Bill	Pallas	623
David	Truong	621
Carol	Nordstrom	180
Chris	Papadopoulos	671
Claudia	Sutherland	140
David	Burns	621
Janet	Giles	123
Janet	Bladen	110
Jennifer M.	Richard	620
John	Montgomery	672

π first_name, last_name, dept_no (employee)

A Cláusula SELECT

(4/6)

Encontre todas as tuplas da relação “employee”, ordenando o resultado pelo atributo “dept_no” e apresentando (ou projetando) somente o atributo “dept_no”.

```
select dept_no
from employee
order by dept_no
```

DEPT_NO
000
100
110
115
120
123
125
126
140
180

```
select distinct dept_no
from employee
order by dept_no
```

DEPT_NO
000
100
110
115
120
123
125
126
140
180

π dept_no (employee)

Linguagens formais de consulta apóiam-se na noção matemática de uma relação ser um conjunto. Assim, tuplas duplicadas nunca aparecem nas relações. Na prática, a eliminação da duplicidade consome um tempo relativo. Portanto, a SQL (como a maioria das linguagens comerciais de consulta) permite duplicidade nas relações. Para eliminar as duplicidades deve-se inserir a palavra chave **distinct** depois da cláusula **select**.

A Cláusula SELECT

(5/6)

Encontre todas as tuplas da relação “employee”, ordenando o resultado pelo atributo “dept_no” e apresentando (ou projetando) somente o atributo “dept_no”.

```
select all dept_no
from employee
order by dept_no
```

DEPT_NO
000
100
110
115
120
123
125
126
140
180

A SQL permite o uso da palavra-chave **all** para especificar “explicitamente” que as duplicidades não serão eliminadas.

A Cláusula SELECT

(6/6)

Encontre todas as tuplas da relação “employee”, ordenando o resultado pelo atributo “first_name” e apresentando (ou projetando) somente os atributos: “first_name”, “last_name”, “salary” e “salary / 12”.

```
select first_name, last_name, salary, salary / 12
from employee
order by first_name
```



$\pi_{\text{first_name, last_name, salary, salary / 12}}(\text{employee})$

FIRST_NAME	LAST_NAME	SALARY	P.3
Ann	Bennet	22.935,00	1.911,250
Ashok	Ramanathan	60.689,50	6.724,125
Bil	Patel	35.000,00	2.916,667
Bruce	Young	37.500,00	3.125,000
Carol	Hodkinson	42.742,50	3.561,875
Chris	Papadopoulos	88.659,00	7.471,250
Charles	Fulmerland	102.514,00	8.485,500
Dana	Johnson	62.550,00	5.212,500
Jacqueline	Dean	390.000,00	32.541,667
Jane	Adams	61.637,00	5.136,416
Jennifer M.	Stark	53.167,50	4.430,625
John	Montgomery	35.000,00	2.916,667

A cláusula select pode conter expressões aritméticas envolvendo os operadores +, -, *, e /, e operandos constantes ou atributos das tuplas.

A Cláusula WHERE

(1/5)

Encontre todas as tuplas da relação “employee” para as quais o valor do atributo “dept_no” seja igual a 120.

```
select * from employee
where dept_no = 120
```



$\sigma_{\text{dept_no} = 120}(\text{employee})$

EMP_NO	FIRST_NAME	LAST_NAME	PHONE_EXT	HIRE_DATE	DEPT_NO	JOB_CODE	JOB_GRADE	JOB_COUNTRY	SALARY
20	Ann	Bennet	5	01.02.1991 00:00	120	Adm.	5	England	22.935,00
36	Roger	Fleaves	6	25.04.1991 00:00	120	Sales	3	England	33.620,63
37	Willie	Stansbury	7	25.04.1991 00:00	120	Eng.	4	England	39.224,00

A SQL usa conectores lógicos **and**, **or** e **not**, em vez dos símbolos matemáticos \wedge , \vee e \neg , na cláusula where. Os operandos dos conectivos lógicos podem ser expressões envolvendo operadores de comparação $<$, $<=$, $>$, $>=$, $=$ e $<>$ (diferente de).

A Cláusula WHERE

(2/5)

Encontre todas as tuplas da relação “employee” para as quais o valor do atributo “salary” esteja no intervalo fechado de 80000.00 até 90000.00.

```
select * from employee
where (salary >= 80000.00) and
(salary <= 90000.00)
```



$\sigma_{\text{salary} \geq 80000.00 \wedge \text{salary} \leq 90000.00}(\text{employee})$

EMP_NO	FIRST_NAME	LAST_NAME	PHONE_EXT	HIRE_DATE	DEPT_NO	JOB_CODE	JOB_GRADE	JOB_COUNTRY	SALARY
11	J.	Isrenson	14	17.01.1990 00:00	110	Emp.	4	USA	86.252,94
20	Chris	Papadopoulos	367	01.01.1990 00:00	671	Mng.	3	USA	88.659,00
24	Pete	Fisher	988	12.09.1990 00:00	671	Eng.	3	USA	81.810,15
45	Ashok	Ramanathan	109	01.08.1991 00:00	621	Eng.	3	USA	80.689,50

A Cláusula WHERE

(3/5)

Encontre todas as tuplas da relação “employee” para as quais o valor do atributo “salary” esteja no intervalo fechado de 80000.00 até 90000.00.

```
select * from employee
where salary between 80000.00 and 90000.00
```

EMP_NO	FIRST_NAME	LAST_NAME	PHONE_EXT	HIRE_DATE	DEPT_NO	JOB_CODE	JOB_GRADE	JOB_COUNTRY	SALARY
11	J.	Isrenson	14	17.01.1990 00:00	110	Emp.	4	USA	86.252,94
20	Chris	Papadopoulos	367	01.01.1990 00:00	671	Mng.	3	USA	88.659,00
24	Pete	Fisher	988	12.09.1990 00:00	671	Eng.	3	USA	81.810,15
45	Ashok	Ramanathan	109	01.08.1991 00:00	621	Eng.	3	USA	80.689,50

A SQL possui o operador de comparação **between** para simplificar a cláusula where que especifica que um atributo possa ter um valor maior ou igual a algum valor e menor ou igual a algum outro valor.

A Cláusula WHERE (4/5)

Encontre todas as tuplas da relação “employee” para as quais o valor do atributo “job_country” seja igual a Canada ou igual a England e apresentando (ou projetando) somente os atributos: “last_name”, “first_name”, “job_country”.

```
select last_name, first_name, job_country
from employee
where job_country in ('Canada', 'England')
```

LAST_NAME	FIRST_NAME	JOB_COUNTRY
Bennet	Ann	England
Reeves	Roger	England
Stambury	Julie	England
Stewart	Claudia	Canada

A SQL possui o operador de teste de pertinência in que verifica se um dado valor é membro (ou pertence) a um conjunto de valores. No exemplo acima as tuplas que aparecerão no resultado da consulta são aquelas cujo valor do atributo “job_country” pertença ao conjunto (“Canada”, “England”).

A Cláusula WHERE (5/5)

Encontre todas as tuplas da relação “employee” para as quais o valor do atributo “hire_date” seja um valor de data pertencente ao ano de 1991, ou seja, de 01/01/1991 até 31/12/1991, ordenando o resultado pelo atributo “hire_date” e apresentando (ou projetando) somente os atributos: “full_name”, “salary”, “hire_date”.

```
select first_name, salary, hire_date
from employee
where hire_date between '01/01/1991' and '31/12/1991'
order by hire_date
```

FIRST_NAME	SALARY	HIRE_DATE
Ann	22.935.00	01.02.1991 00:00
Roger	63.482.63	18.02.1991 00:00
Janet	61.637.81	21.03.1991 00:00
Julie	39.224.06	25.04.1991 00:00
Roger	33.620.63	26.04.1991 00:00
Leslie	56.034.36	03.06.1991 00:00
Ashok	80.689.50	01.08.1991 00:00
Walter	116.100.00	09.08.1991 00:00
Carol	42.742.50	02.10.1991 00:00

← hire_date >= '01/01/1991' AND hire_date <= '31/12/1991' (employee)
 * full_name, salary, hire_date (A)

A Cláusula FROM

A cláusula from por si só define um produto cartesiano das relações na cláusula. Encontre todas as tuplas da relação “customer” relacionadas (ou juntadas) com todas as tuplas correspondentes na relação “sales” ordenando pelo atributo “cust_no”.

```
select customer.cust_no, customer, po_number,
       sales.cust_no, total_value
from customer, sales
where customer.cust_no = sales.cust_no
order by cust_no
```

CUST_NO	CUSTOMER	PO_NUMBER	CUST_NO	TOTAL_VALUE
1.001	Signature Design	V1924100	1.001	6.00
1.002	Signature Design	V1940009	1.001	3.399.15
1.001	Signature Design	V1933636	1.001	60.000.00
1.001	Signature Design	V19427029	1.001	422.219.51
1.002	Signature Design	V19324200	1.001	560.000.00
1.002	Idalia Technologies	V1933005	1.002	600.50
1.002	Idalia Technologies	V1936170	1.002	14.050.00
1.002	Idalia Technologies	V1933006	1.002	20.000.00
1.003	Buttle, Griffin and Co.	V1938200	1.003	6.00
1.003	Buttle, Griffin and Co.	V1949139	1.003	12.562.12
1.003	Buttle, Griffin and Co.	V1949200	1.003	27.000.00

A SQL usa a notação: *NomeDaRelação.NomeDoAtributo*, como na álgebra relacional, para evitar ambigüidades nos casos em que um atributo aparece no esquema de mais de uma relação.

A mesma consulta em álgebra relacional:

$\pi_{customer.cust_no, customer, po_number, sales.cust_no, total_value} (customer \bowtie sales)$

A Operação Rename (as)

A SQL proporciona um mecanismo para rebatizar tanto relações (variáveis tuplas) quanto atributos, usando a cláusula as da seguinte forma:

nome_antigo as nome_novo

```
select salary
from employee
where dept_no = 120
```

SALARY
22.935.00
63.482.63
61.637.81

```
select sum(salary) as tot_salary
from employee
where dept_no = 120
```

TOT_SALARY
95.779.63

$\pi_{tot_salary} (sum(salary)) (\sigma_{dept_no = 120} (employee))$

Variáveis Tuplas

A cláusula **as** é particularmente útil na definição do conceito de variável tupla. Uma variável tupla em SQL precisa estar associada a uma relação da cláusula **from** em particular. Encontre todas as tuplas da relação "customer" relacionadas (ou juntadas) com todas as tuplas correspondentes na relação "sales" ordenando pelo atributo "cust_no".

```
select c.cust_no, customer, po_number,  
       s.cust_no, total_value  
from customer c, sales s  
where c.cust_no = s.cust_no  
order by cust_no
```

CUST_NO	CUSTOMER	PO_NUMBER	CUST_NO3	TOTAL_VALUE
1.001	Signature Design	15934330	1.001	0.00
1.001	Signature Design	15403039	1.001	3.399.14
1.001	Signature Design	15933063	1.001	60.000.04
1.001	Signature Design	154027029	1.001	432.219.31
1.001	Signature Design	15934330	1.001	960.000.04
1.002	id allen Technologies	15933006	1.002	600.50
1.002	id allen Technologies	15936130	1.002	14.050.00
1.002	id allen Technologies	15933006	1.002	20.000.00
1.003	Butler, Griffin and Co.	15936320	1.003	679
1.003	Butler, Griffin and Co.	159349139	1.003	12.562.12
1.003	Butler, Griffin and Co.	15934530	1.003	27.000.00

Define-se a variável tupla, neste caso, as variáveis **c** e **s**, na cláusula **from** colocando-a depois do nome da relação à qual está associada. Atenção: o uso da palavra-chave **as** é opcional.

Operações em Strings (1/3)

As operações em strings mais usadas são as checagens para verificação de coincidências de pares, usando o operador **like** combinado com os caracteres especiais: porcentagem (%) e sublinhado (_).

Expressão	Resultado
LIKE 'A %'	Qualquer string que iniciem com a letra A.
LIKE '%A'	Qualquer string que terminem com a letra A.
LIKE '%A %'	Qualquer string que tenha a letra A em qualquer posição.
LIKE 'A _'	String de dois caracteres que tenham a primeira letra A e o segundo caractere seja qualquer outro.
LIKE '_ A'	String de dois caracteres cujo primeiro caractere seja qualquer um e a última letra seja a letra A.
LIKE '_ _ A'	String de três caracteres cuja segunda letra seja A, independentemente do primeiro ou do último caractere.
LIKE '%A _'	Qualquer string que tenha a letra A na penúltima posição e a última seja qualquer outro caractere.
LIKE '% _ A'	Qualquer string que tenha a letra A na segunda posição e o primeiro caractere seja qualquer outro caractere.

Operações em Strings (2/3)

Expressão	Resultado
LIKE ' _ _'	Qualquer string com exatamente três caracteres.
LIKE ' _ _ %'	Qualquer string com pelo menos três caracteres.

Comparações com strings são sensíveis ao tamanho da letra; isto é, minúsculas não são iguais a maiúsculas, e vice-versa.

```
select full_name  
from employee  
where full_name like 'Johnson%'
```

FULL_NAME
Johnson, Leslie
Johnson, Scott

```
select full_name  
from employee  
where full_name like '%Le%'
```

FULL_NAME
Johnson, Leslie
Lee, Terri
Phong, Leslie
Leung, Luke

Operações em Strings (3/3)

Para comparações que envolvam caracteres especiais (isto é, % e _), a SQL permite o uso de um caractere de escape (\). Esse caractere é usado imediatamente antes do caractere especial que deverá ser tratado como um caractere normal.

Expressão	Resultado
LIKE 'ab\%cd'	Qualquer string que comece por ab%cd.
LIKE 'ab\\cd'	Qualquer string que comece por ab\cd.
LIKE '%\"%'	Qualquer string que tenha o caractere ' em qualquer posição.

A SQL permite pesquisar diferenças em vez de coincidências, por meio do uso do operador de comparação **not like**.

A SQL também permite uma variedade de funções com strings de caracteres, como concatenação (usando "||"), extração de substrings, indicação de tamanhos de strings, conversão de minúsculas para maiúsculas (**upper**) e assim por diante.

```
select full_name  
from employee  
where upper(full_name) like '%LE%'
```

FULL_NAME
Johnson, Leslie
Lee, Terri
Phong, Leslie
Leung, Luke

Operações em Datas (1/2)

Utiliza-se a função **extract** com as palavras-chave: **day**, **month** e **year**.

Encontre todas as tuplas da relação "employee" para as quais o valor do atributo "hire_date" seja um valor de data pertencente ao **mês de agosto** de qualquer ano e apresentando (ou projetando) somente os atributos: "full_name", "salary", "hire_date".

```
select full_name, salary, hire_date
from employee
where extract(month from hire_date) = 8
```

FULL_NAME	SALARY	HIRE_DATE
Ramanathan, Ashok	80.689,50	01.08.1991 00:00
Steadman, Walter	116.100,00	09.08.1991 00:00
Williams, Randy	56.295,00	08.08.1992 00:00
Yanowski, Michael	44.000,00	09.08.1993 00:00
Glor, Jacques	390.500,00	23.08.1993 00:00

Operações em Datas (2/2)

Encontre todas as tuplas da relação "employee" para as quais o valor do atributo "hire_date" seja um valor de data pertencente ao **ano de 1991**, ordenando o resultado pelo atributo "full_name" e apresentando (ou projetando) somente os atributos: "full_name", "salary", "hire_date".

```
select full_name, salary, hire_date
from employee
where extract(year from hire_date) = 1991
order by full_name
```

FULL_NAME	SALARY	HIRE_DATE
Balkester, Janet	61.637,81	21.03.1991 00:00
Bennet, Ann	22.935,00	01.02.1991 00:00
De Souza, Roger	69.482,63	18.02.1991 00:00
Nordstrom, Carol	42.742,50	02.10.1991 00:00
Phong, Leslie	56.034,38	03.06.1991 00:00
Ramanathan, Ashok	80.689,50	01.08.1991 00:00
Reeves, Roger	33.620,63	25.04.1991 00:00
Stansbury, Willie	39.224,06	25.04.1991 00:00
Steadman, Walter	116.100,00	09.08.1991 00:00

Atenção:

No banco de dados MS Access **day**, **month** e **year** são funções. E deverão ser usadas como no exemplo a seguir:

```
select * from employee
where year(hire_date) = 1991
```

Ordenação de Tuplas

A SQL oferece ao usuário algum controle sobre a ordenação por meio da qual as tuplas de uma relação serão apresentadas. A cláusula **order by** faz com que as tuplas do resultado de uma consulta apareçam em uma determinada ordem. Por "padrão", a relação ordenada é apresentada em "ordem ascendente". Para especificação da forma de ordenação, deve-se indicar **desc** para ordem descendente e **asc** para ordem ascendente. Além disso, a ordenação pode ser realizada por diversos atributos.

```
select full_name, salary
from employee
order by salary asc
```

FULL_NAME	SALARY
Bennet, Ann	22.935,00
Brown, Kelly	27.000,00
Chen, Sue Anne	35.275,00
Goldstein, Mark	35.000,00
Reeves, Roger	33.620,63
Stark, Ed	35.000,00

```
select full_name, salary
from employee
order by salary desc
```

FULL_NAME	SALARY
Foran, Roberto	99.000.000,00
Tamamoto, Takashi	7.480.000,00
Uchida, Yuki	6.000.000,00
Glor, Jacques	390.500,00
Bennet, Ann	22.935,00
Steadman, Walter	116.100,00

Funções Agregadas (1/3)

As "funções agregadas" são funções que tomam uma coleção (um conjunto ou um subconjunto) de valores como entrada, retornando um **valor simples**.

- Média (average): **avg**

```
select avg(salary) as salario_avg
from employee
```

SALARIO_AVG
2.750.534,95

- Mínimo valor: **min**

```
select min(salary) as salario_min
from employee
```

SALARIO_MIN
22.935,00

- Máximo valor: **max**

```
select max(salary) as salario_max
from employee
```

SALARIO_MAX
99.000.000,00

Funções Agregadas (2/3)

- Soma Total: **sum**

```
select sum(salary) as salario_total
from employee
```

SALARIO_TOTAL
115.522.468,00

```
select c.cust_no,
       sum(total_value) as tot
from customer c, sales s
where c.cust_no = s.cust_no
group by c.cust_no
order by c.cust_no
```

CUST_NO	TOT
1.001	1.045.610,12
1.002	25.450,50
1.003	39.502,12
1.004	75.000,00
1.005	14.900,00
1.006	400.000,00
1.007	3.999,99
1.008	25.000,00
1.009	490,89
1.010	21.195,40
1.011	121.980,72
1.012	463.000,47
1.013	2.893,00
1.014	100,02
1.015	1.500,00

Funções Agregadas (3/3)

- Contagem: **count**

```
select count(*) as ct
from employee
```

CT
42

```
select c.cust_no,
       sum(total_value) as tot,
       count(*) as ct
from customer c, sales s
where c.cust_no = s.cust_no
group by c.cust_no
order by c.cust_no
```

CUST_NO	TOT	CT
1.001	1.045.610,12	5
1.002	25.450,50	2
1.003	39.502,12	3
1.004	75.000,00	2
1.005	14.900,00	2
1.006	400.000,00	2
1.007	3.999,99	1
1.008	25.000,00	2
1.009	490,89	1
1.010	21.195,40	2
1.011	121.980,72	2
1.012	463.000,47	4
1.013	2.893,00	1
1.014	100,02	1
1.015	1.500,00	1

```
select count(distinct cust_no) as ct
from sales
```

CT
15

A Cláusula Group By (1/2)

Existem circunstâncias em que seria necessário aplicar uma função agregada (*count*, *sum*, *avg*, ...) não somente a um conjunto de tuplas, mas também a um **grupo** de conjunto de tuplas o que é possível usando a cláusula SQL **group by**.

O atributo ou atributos fornecidos na cláusula *group by* são usados para formar grupos. Tuplas com os mesmos valores em todos os atributos da cláusula *group by* são colocadas em um grupo.

```
\* aplicando a função agregada a um
conjunto de tuplas *\
select avg(salary) as salario_avg
from employee
```

SALARIO_AVG
2.750.534,95

```
\* aplicando a função agregada a um
grupo de conjunto de tuplas *\
select dept_no, avg(salary) as salario_avg
from employee
group by dept_no
```

DEPT_NO	SALARIO_AVG
000	133.321,50
100	77.631,25
110	65.221,41
115	6.740.000,00
120	31.926,56
121	110.000,00

A Cláusula Group By (2/2)

Às vezes, é mais interessante definir **condições** e aplicá-las a **grupos** de que aplicá-las a tuplas. Por exemplo, encontrar quais "dept_no" possuem média (avg) do atributo "salary" maior que a média de todas as tuplas da relação "employee".

Essa condição não se aplica a uma única tupla, mas em cada grupo determinado pela cláusula *group by*.

Para exprimir tal consulta, deve-se usar a cláusula **having** da SQL. Os predicados da cláusula *having* são aplicados depois da formação dos grupos, assim poderão ser usadas funções agregadas.

```
select dept_no, avg(salary) as salario_avg
from employee
group by dept_no
having avg(salary) > (select avg(salary) from employee)
```

DEPT_NO	SALARIO_AVG
115	6.740.000,00
121	110.000,00

Operações de Conjuntos

Os operadores SQL-92 **union**, **intersect** e **except** operam relações e correspondem às operações de união (\cup), interseção (\cap) e diferença ($-$) da álgebra relacional, e portanto, as relações participantes devem ser compatíveis, ou seja, apresentar o mesmo conjunto de atributos (ou esquema).

A SQL-89 possui diversas restrições para o uso de **union**, **intersect** e **except**.

Certos produtos não oferecem suporte para essas operações.

A Operação de União (\cup)

Una todas as tuplas da relação "employee" para as quais o valor do atributo "dept_no" seja igual a 120 com as tuplas da relação "employee" cujo o valor do atributo "dept_no" seja igual a 600.

```
select full_name, salary, dept_no from employee
where dept_no = 120
union
select full_name, salary, dept_no from employee
where dept_no = 600
```

FULL_NAME	SALARY	DEPT_NO
Bennet, Ann	22.935,00	120
Brown, Kelly	27.000,00	600
Nelson, Robert	105.900,00	600
Roeves, Roger	33.620,63	120
Stansbury, Willie	38.224,06	120

A ← π full_name, salary, hire_date ($\sigma_{dept_no = 120}(employee)$)
B ← π full_name, salary, hire_date ($\sigma_{dept_no = 600}(employee)$)
A \cup B

Valores Nulos

O valor **null** indica a ausência de informação sobre o valor de um atributo. Sendo assim, pode-se usar a palavra-chave **null** como predicado para testar a existência de valores nulos.

```
select * from customer
inner join sales
on customer.cust_no = sales.cust_no
where phone_no is null
```

CUST_NO	CUSTOMER	CONTACT_FIRST	CONTACT_LAST	PHONE_NO
1.007	Mrs. Beauvais	null	Mrs. Beauvais	null

O predicado **not null** testa a ausência de valores nulos.

Composição de Relações

Além de fornecer o mecanismo básico do produto cartesiano para a composição das tuplas de uma relação disponível nas primeiras versões da SQL, a SQL-92 também oferece diversos outros mecanismos para composição de relações como as junções condicionais e as junções naturais, assim como várias formas de junções externas.

- junção interna (ou junção condicional): **inner join**
- junção externa à esquerda: **left outer join**
- junção externa à direita: **right outer join**
- junção externa total: **full outer join**

Junção Interna (inner join)

Relaciona (ou junta) através do atributo “cust_no” cada tupla da relação “customer” com as suas tuplas correspondentes na relação “sales”. Cada tupla resultante dessa primeira relação é juntada com a tupla correspondente na relação “employee” através do predicado “on s.sales_rep = e.emp_no”.

```
select c.cust_no, customer,
       po_number, ship_date, total_value, sales_rep,
       full_name
from customer c
inner join sales s on c.cust_no = s.cust_no
inner join employee e on s.sales_rep = e.emp_no
```

CUST_NO	CUSTOMER	PO_NUMBER	SHIP_DATE	TOTAL_VALUE	SALES_REP	FULL_NAME
1.001	Signature Design	V9324200	09.08.1990 00:00	560.000.00	127	Yanowski, Michael
1.001	Signature Design	V9324200	16.08.1990 00:00	0.00	127	Yanowski, Michael
1.001	Signature Design	V9320630		60.000.00	127	Yanowski, Michael
1.001	Signature Design	V9420099		3.289.15	127	Yanowski, Michael
1.001	Signature Design	V9421023	10.02.1994 00:00	422.210.91	127	Yanowski, Michael
1.002	Dallas Technologies	V9333005	20.03.1990 00:00	600.50	11	Weston, K. J.
1.002	Dallas Technologies	V9333006	02.05.1990 00:00	20.000.00	11	Weston, K. J.
1.002	Dallas Technologies	V9336100	01.01.1994 00:00	14.850.00	11	Weston, K. J.

Junção Externa à Esquerda (left outer join)

As operações de “junção externa” são uma extensão da operação de junção interna (inner join) para tratar informações omitidas.

empregado			empregado_depto		
NOME_EMP	RUA	CIDADE	NOME_EMP	DEPTO	SALARIO
Alzemi	Rua A	Pato Branco	Amêda	X	1.500.00
Amêda	Rua B	Maripol	Juca	X	1.300.00
Juca	Rua C	Vitório	Maria	X	5.300.00
Maria	Rua D	Vest	Tadeu	X	1.500.00

```
select * from empregado e
inner join empregado_depto d
on e.nome_emp = d.nome_emp
```

Junção Interna

Os empregados “Alzemi” e “Tadeu” na participam da relação resultado porque não possuem valores nas duas relações envolvidas.

NOME_EMP	RUA	CIDADE	NOME_EMP	DEPTO	SALARIO
Amêda	Rua B	Maripol	Amêda	X	1.500.00
Juca	Rua C	Vitório	Juca	X	1.300.00
Maria	Rua D	Vest	Maria	X	5.300.00

```
select * from empregado e
left outer join empregado_depto d
on e.nome_emp = d.nome_emp
```

Junção Externa à Esquerda

Acrescenta a relação resultado “todas” as tuplas da relação à esquerda que não encontram par entre as tuplas da relação à direita, preenchendo com valores null todos os outros atributos da relação à direita.

NOME_EMP	RUA	CIDADE	NOME_EMP	DEPTO	SALARIO
Alzemi	Rua A	Pato Branco	Amêda	X	1.500.00
Amêda	Rua B	Maripol	Juca	X	1.300.00
Juca	Rua C	Vitório	Maria	X	5.300.00
Maria	Rua D	Vest	Maria	X	5.300.00

Junção Externa à Direita (right outer join)

A “junção externa à direita” acrescenta a relação resultado “todas” as tuplas da relação à direita que não encontram par entre as tuplas da relação à esquerda, preenchendo com valores null todos os outros atributos da relação à esquerda.

empregado			empregado_depto		
NOME_EMP	RUA	CIDADE	NOME_EMP	DEPTO	SALARIO
Alzemi	Rua A	Pato Branco	Amêda	X	1.500.00
Amêda	Rua B	Maripol	Juca	X	1.300.00
Juca	Rua C	Vitório	Maria	X	5.300.00
Maria	Rua D	Vest	Tadeu	X	1.500.00

```
select * from empregado e
inner join empregado_depto d
on e.nome_emp = d.nome_emp
```

NOME_EMP	RUA	CIDADE	NOME_EMP	DEPTO	SALARIO
Amêda	Rua B	Maripol	Amêda	X	1.500.00
Juca	Rua C	Vitório	Juca	X	1.300.00
Maria	Rua D	Vest	Maria	X	5.300.00

Junção Interna

Os empregados “Alzemi” e “Tadeu” na participam da relação resultado porque não possuem valores nas duas relações envolvidas.

```
select * from empregado e
right outer join empregado_depto d
on e.nome_emp = d.nome_emp
```

NOME_EMP	RUA	CIDADE	NOME_EMP	DEPTO	SALARIO
Amêda	Rua B	Maripol	Amêda	X	1.500.00
Juca	Rua C	Vitório	Juca	X	1.300.00
Maria	Rua D	Vest	Maria	X	5.300.00
Alzemi	Rua A	Pato Branco	Tadeu	X	1.500.00

Junção Externa Total (full outer join)

A “junção externa total” acrescenta a relação resultado as tuplas da relação à esquerda que não encontram par entre as tuplas da relação à direita, assim como as tuplas da relação à direita que não encontram par entre as tuplas da relação à esquerda.

empregado			empregado_depto		
NOME_EMP	RUA	CIDADE	NOME_EMP	DEPTO	SALARIO
Alzemi	Rua A	Pato Branco	Amêda	X	1.500.00
Amêda	Rua B	Maripol	Juca	X	1.300.00
Juca	Rua C	Vitório	Maria	X	5.300.00
Maria	Rua D	Vest	Tadeu	X	1.500.00

```
select * from empregado e
inner join empregado_depto d
on e.nome_emp = d.nome_emp
```

NOME_EMP	RUA	CIDADE	NOME_EMP	DEPTO	SALARIO
Amêda	Rua B	Maripol	Amêda	X	1.500.00
Juca	Rua C	Vitório	Juca	X	1.300.00
Maria	Rua D	Vest	Maria	X	5.300.00

Junção Interna

Os empregados “Alzemi” e “Tadeu” na participam da relação resultado porque não possuem valores nas duas relações envolvidas.

```
select * from empregado e
full outer join empregado_depto d
on e.nome_emp = d.nome_emp
```

NOME_EMP	RUA	CIDADE	NOME_EMP	DEPTO	SALARIO
Alzemi	Rua A	Pato Branco	Amêda	X	1.500.00
Amêda	Rua B	Maripol	Amêda	X	1.500.00
Juca	Rua C	Vitório	Juca	X	1.300.00
Maria	Rua D	Vest	Maria	X	5.300.00
Alzemi	Rua A	Pato Branco	Tadeu	X	1.500.00

Modificações no Banco de Dados (1/7)

As instruções para modificar a instância de banco de dados serão demonstradas, basicamente, sobre o seguinte esquema:

Modelo Relacional (modelo lógico):

cargo

<u>CdCargo</u>	NmCargo	VrSalario
----------------	---------	-----------

depto

<u>CdDepto</u>	NmDepto	Ramal
----------------	---------	-------

funcionario

<u>NrMatric</u>	NmFunc	DtAdm	Sexo	CdCargo	CdDepto
-----------------	--------	-------	------	---------	---------

Modificações no Banco de Dados (2/7)

Inserção (insert)

Para inserir dados em uma relação podemos especificar uma tupla a ser inserida ou escrever uma consulta cujo resultado é um conjunto de tuplas a inserir. Obviamente, os valores dos atributos para as tuplas a inserir devem pertencer ao domínio desses atributos. Similarmente, tuplas a inserir devem possuir a ordem correta (mesmo esquema).

```
insert into NomeDaRelação
values (ValorDoAtributo1, ValorDoAtributo2, ...,
        ValorDoAtributoN)
```

Modificações no Banco de Dados (3/7)

Inserção (insert)

Inserindo uma tupla.

```
insert into cargo
values(1, 'Programador Analista', 2500.00)
```

```
insert into cargo
values(2, 'DBA', 4700.00)
```

```
insert into cargo
values(3, 'Suporte', 800.00)
```

```
select * from cargo
```

CDCARGO	NMCARGO	VRSALARIO
1	Programador Analista	2.500,00
2	DBA	4.700,00
3	Suporte	800,00

Inserindo um conjunto de tuplas.

```
insert into newcargo
select * from cargo
```

Modificações no Banco de Dados (4/7)

Inserção (insert)

Inserindo uma tupla e usando apenas alguns atributos.

```
insert into newcargo(cdcargo, nmcargo)
values(1, 'Programador Analista')
```

Seria equivalente a seguinte instrução.

```
insert into newcargo
values(1, 'Programador Analista', null)
```

Inserindo uma tupla e usando apenas alguns atributos.

```
insert into newcargo(cdcargo, vrsalario)
values(2, 5000.00)
```

Seria equivalente a seguinte instrução.

```
insert into newcargo
values(2, null, 5000.00)
```

```
select * from newcargo
```

CDCARGO	NMCARGO	VRSALARIO
1	Programador Analista	2.500,00
2	DBA	5.000,00

Modificações no Banco de Dados (5/7)

Remoção (*delete*)

Um pedido de remoção de dados é expresso muitas vezes do mesmo modo que uma consulta. Pode-se remover somente tuplas inteiras; não é possível, por exemplo, excluir valores de um atributo em particular.

```
delete from r
where P
```

em que P representa um predicado e r , uma relação. O comando **delete** encontra primeiro todas as tuplas t em r para as quais $P(t)$ é verdadeira e então remove-as de r . A cláusula **where** pode ser omitida nos casos de remoção de todas as tuplas de P .

Modificações no Banco de Dados (6/7)

Remoção (*delete*)

```
Remove todos os funcionários com CdCargo = 1.
delete from funcionario
where CdCargo = 1

Remove todos os funcionários com CdCargo = 1 e
com CdDepto = 1.
delete from funcionario
where (CdCargo = 1) and (CdDepto = 1)

Remove todos os funcionários.
delete from funcionario

O pedido delete por conter um select aninhado, como por
exemplo, para remover todos os funcionários com o valor do
atributo Salary maior que a média do próprio atributo.
delete from employee
where salary > (select avg(salary) from employee)
```

Modificações no Banco de Dados (7/7)

Atualizações (*update*)

```
Aumenta 10% os salário de todos os cargos.
update cargo
set vrSalario = vrSalario * 1.10

Aumenta em R$ 50,00 os salário inferiores a R$ 1.000,00.
update cargo
set vrSalario = vrSalario + 50.00
where vrSalario < 1000.00

Modifica o nome e o salário do CdCargo = 1.
update cargo
set nmCargo = 'Programador Analista Senior',
   vrSalario = 4500.00
where cdCargo = 1

Aumenta 5% os salário dos cargos com salário abaixo da média.
update cargo
set vrSalario = vrSalario * 1.05
where salary < (select avg(salary) from employee)
```

DDL- Data Definition Language

A “Linguagem de definição de dados” abrange comandos para:

- **Create table**, **Alter table** e **Drop table**:
 - comandos para a definição de esquemas de relações (criação e modificação) e exclusão de relações.
- **Create domain**:
 - criação de domínio dos valores associados a atributos.

Tipos de Domínios em SQL (1/3)

- `char(n)` é uma cadeia de caracteres de tamanho fixo
- `varchar(n)` é uma cadeia de caracteres e tamanho variável
- `integer` é um inteiro (4 bytes)
- `smallint` é um inteiro pequeno (2 bytes)
- `numeric(p, d)` é um número de ponto fixo cuja precisão é definida. Onde, p indica a quantidade de dígitos (incluindo o ponto decimal e o sinal) e d dos p dígitos estão à direita do ponto decimal. Por exemplo, para representar o valor -3500,75 seria necessário a seguinte definição: `numeric(8,2)`

Tipos de Domínios em SQL (2/3)

- `real` e `double precision` são números de ponto flutuante e ponto flutuante de precisão dupla
- `float(n)` é um número de ponto flutuante com a precisão definida pelo usuário em pelo menos n dígitos
- `date` é um calendário contendo um ano (com 4 dígitos), mês e dia do mês.
- `time` representa horário, em horas, minutos e segundos

Tipos de Domínios em SQL (3/3)

- O valor nulo (`null`) é um membro de todos os domínios. Para certos atributos, entretanto, valores nulos podem ser inadequados.
- Como por exemplo, no valor de chaves primárias ou de um atributo como o CPF.
- A SQL permite que a declaração de domínios de um atributo inclua a especificação de `not null`, proibindo, assim, a inserção de valores nulos para esse tipo de atributo. Qualquer modificação que possa resultar na inserção de um valor nulo em um domínio `not null` gera um diagnóstico de erro.

Criando Domínios

A SQL-92 permite a definição de domínios usando a cláusula `create domain`:

```
create domain nomeDoDominio tipoDoDominio
```

```
create domain mes as smallint  
check(value between 1 and 12)
```

```
create domain boolean as char(1)  
check(value in('T', 'F'))
```

```
create domain sal_emp as numeric(8, 2)  
default 350.00  
check(value >= 350.00)
```

onde:
`value` representa o valor atribuído ao atributo
`default` define o valor padrão para o atributo
`check` verifica se o valor informado para o atributo satisfaz a condição especificada

para apagar um domínio criado:
`drop domain nomeDoDominio`

Criando Relações (1/2)

Definição de Esquema em SQL:

```
create table r
(A1D1, A2D2, ..., AnDn,
<regras de integridade1>
...,
<regras de integridadek>)
```

onde: r é o nome da relação, cada A_i é o nome de um atributo no esquema da relação r e D_i é o tipo de domínio dos valores no domínio dos atributos A_i .

As regras de integridade (*constraint*) permitidas:

- primary key
- foreign key

Criando Relações (2/2)

```
create table empregado (
  nome_emp varchar(20),
  rua       varchar(20),
  cidade    varchar(20)
)
```

```
create table empregado_depto (
  nome_emp varchar(20),
  depto     char(1),
  salario   sal_emp
)
```

```
create table empreg (
  codigo integer not null,
  nome    varchar(35),
  casado  boolean,
  sal     sal_emp,
  cdCargo integer,
  cdDepto integer,

  constraint pf_codigo primary key (codigo),
  constraint fk_cdCargo foreign key (cdCargo)
    references cargo(cdCargo),
  constraint fk_cdDepto foreign key (cdDepto)
    references depto(cdDepto)
)
```

Removendo Relações

Para remoção de uma relação deve-se usar o comando *drop table*.

O comando: **drop table** r

é uma ação mais drástica que: **delete from** r

onde:

“delete from r ” mantém a relação r , mas remove todas as suas tuplas.

“drop table r ” não remove apenas todas as tuplas de r , mas também seu esquema.

Modificando Relações (1/2)

Para modificar uma relação deve-se usar o comando *alter table*.

para adicionar atributos a uma relação existente:

alter table r **add** A D

onde r é o nome de uma relação existente, A é o nome do novo atributo que será adicionado e D é seu domínio. Atenção: todas as tuplas da relação recebem valores *null* para seu novo atributo.

para remover atributos de uma relação existente:

alter table r **drop** A

onde r é o nome de uma relação existente e A , o nome do atributo da relação que será removido.

Modificando Relações (2/2)

Acrescenta o atributo "nomeChefe" do tipo varchar(30) a relação "depto".

```
alter table depto  
add nomeChefe char(35)
```

Remove o atributo "sexo" da relação "funcionario".

```
alter table funcionario  
drop sexo
```

Adicionando e Removendo restrições (constraint):

Remove a restrição "fk_cddepto" da relação "empreg".

```
alter table empreg  
drop constraint fk_cddepto
```

Acrescenta a relação "empreg" a restrição "fk_cddepto" de chave estrangeira (foreign key) usando o atributo "cddepto" fazendo a ligação com a relação "depto".

```
alter table empreg  
add constraint fk_cddepto  
foreign key(cddepto) references depto(cddepto)
```

Referências

- Sistema de Banco de Dados.
 - Abraham Silberschatz; Henry F. Korth; S. Sudarshan.
 - Capítulo 4: SQL
 - São Paulo: Makron Books, 3ª ed., 1999.
- Prof. Francisco Reverbel
 - <http://www.ime.usp.br/~reverbel/>