Bases de Dados

AULA 10: Consultas Básicas em SQL com a Instrução SELECT

- **Conhecendo a Instrução SELECT**
- **Restringindo e Ordenando o Conjunto de Resultados**
- **Operações de Conjunto**
- **Trabalhando com o Valor NULO**
- **Produto Cartesiano**
- **Junção Natural e Junção Theta**

Prof. Eduardo Corrêa Gonçalves
21/09/2022

I. Introdução

Conforme sabemos, os SGBD's relacionais são manipulados através de uma **linguagem padrão**, desenvolvida especialmente para o ambiente relacional, denominada **SQL** (*Structured Query Language*). Muitos pesquisadores da área de banco de dados consideram-na a principal responsável pela imensa popularidade conquistada pelos SGBD's relacionais nos últimos 30 anos. A SQL é oferecida em praticamente todos os ambientes de programação (Python, Java, PHP, C#, etc.) e está disponível até mesmo em linguagens especialmente direcionadas para estatística e ciência de dados como R e SAS.

A linguagem SQL é composta por um reduzido conjunto de **instruções** que permitem manipular um banco de dados com diferentes finalidades, conforme apresenta a Tabela I.1.

Tabela I.1 – Instruções SQL e sua diferentes finalidades

Finalidade	Instruções	Descrição
Recuperação de Dados	SELECT	Recupera registros armazenados em tabelas do banco de dados.
Manipulação de Dados	INSERT UPDATE DELETE	Inserção, alteração e remoção de registros de tabelas do banco de dados. Este subconjunto de instruções da SQL é conhecido como **DML** (*Data Manipulation Language*).
Definição de Dados	CREATE ALTER DROP	Criação, alteração e exclusão de objetos do banco de dados (ex.: tabelas, índices, etc.). Este subconjunto de instruções é conhecido como **DDL** (*Data Definition Language*).
Controle de Transações	COMMIT ROLLBACK	Gerenciam as modificações realizadas pelos comandos DML. Permitem agrupar as alterações dos dados em transações lógicas.

Esta aula aborda a utilização básica da instrução SELECT, que na prática representa a instrução SQL mais utilizada pelos profissionais da área de estatística / ciência de dados. O objetivo central desta aula é apresentar a forma de realizar o mapeamento ou "tradução" direta das instruções da Álgebra Relacional que aprendemos nas últimas aulas para comandos SELECT da SQL. Em aulas posteriores (principalmente na segunda parte do curso), serão apresentados outros recursos e formas de utilização desta poderosa instrução SQL.

II. Banco de Dados "Products.db"

O banco de dados utilizado nesta aula é chamado "products.db" e consiste na implementação em SQLite da base de dados utilizada na **aula 09** (Execícios de Álgebra Relacional). Essa base de dados−originalmente introduzida no Capítulo 2 do livro "Database Systems: The Complete Book"[1] − é formada pelas quatro tabelas apresentadas na Figura II.1.

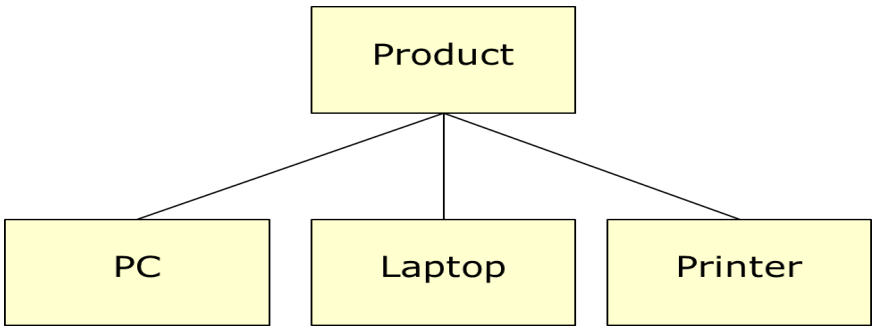

Figura II.1 – Tabelas da base de dados "Product.db"

A especificação completa das tabelas é apresentada a seguir. Para cada tabela, inicialmente apresenta-se a descrição da mesma e, a seguir, a relação de atributos (colunas) que a compõem. As seguintes informações são apresentadas para cada atributo:

- Atributo = nome do atributo;
- Tipo = tipo de dado (TEXT, INT ou NUM);
- PK? = se estiver marcado com X, a coluna é chave primária
- NOT NULL? = se estiver marcado com X, a coluna é do tipo NOT NULL;
- FK? = se estiver marcado com X, a coluna é chave estrangeira.
- Descrição / Comentários = descrição do atributo e informações adicionais.

Product

Tabela que contém as informações gerais sobre diversos produtos. Fornece o fabricante (maker), número do modelo (model) e tipo (type) de aparelho ('PC', 'laptop' ou 'printer') de cada produto. Assumiremos por conveniência que os números de modelo são únicos entre todos os fabricantes e tipos de produto. Todas as demais tabelas possuem chave estrangeira para esta tabela.

Atributo	Tipo	PK?	NOT NULL?	FK?	Descrição / Comentários
maker	TEXT		X		Código do fabricante.
model	INT	X	X		Número do modelo
type	TEXT		X		Tipo de produto ('pc', 'laptop' ou 'printer').

[1] Hector Garcia-Molina, Jeffrey D. Ullman e Jennifer Widom. "Database Systems: The Complete Book", 2nd ed. Pearson, 2009.

PC

Fornece, para cada número de modelo que corresponde a um 'pc' (computador desktop), a velocidade do processador em giga-hertz (speed), a quantidade de memória RAM em megabytes (ram), o tamanho do hard disk em gigabytes (hd) e o preço (price).

Atributo	Tipo	PK?	NOT NULL?	FK?	Descrição / Comentários
model	INT	X	X	X	Número do modelo. (FK ref. "model" em Product)
speed	NUM		X		Velocidade do processador em giga-hertz.
ram	INT		X		Quantidade de memória RAM em megabytes
hd	INT		X		Tamanho do hard disk em gigabytes
price	NUM				Preço do produto.

Laptop

Fornece, para cada número de modelo que corresponde a um 'laptop', as mesmas informações da tabela *PC* (speed, ram, hd e price) e mais um campo adicional para armazenar tamanho da tela em polegadas (screen).

Atributo	Tipo	PK?	NOT NULL?	FK?	Descrição / Comentários
model	INT	X	X	X	Número do modelo. (FK ref. "model" em Product)
speed	NUM		X		Velocidade do processador em giga-hertz.
ram	INT		X		Quantidade de memória RAM em megabytes
hd	INT		X		Tamanho do hard disk em gigabytes
screen	NUM		X		Tamanho da tela em polegadas.
price	NUM				Preço do produto.

Printer

Fornece, para cada número de modelo que corresponde a uma 'printer' (impressora), se ela é do tipo colorida (color, valor 1 caso seja colorida e 0 caso contrário), o tipo (type, 'laser' ou 'ink-jet') e também o preço (price).

Atributo	Tipo	PK?	NOT NULL?	FK?	Descrição / Comentários
model	INT	X	X	X	Número do modelo. (FK ref. "model" em Product)
color	BOOLEAN		X		Indica se impressora é colorida (1=SIM, 0=NÃO)
type	INT		X		Tipo de impressora ('laser' ou 'ink-jet').
price	NUM				Preço do produto.

A seguir apresenta-se o *script* SQL executado para a criar e preencher as tabelas. Note que primeiro é preciso criar *Product* no SQLite, já que as demais tabelas possuem chave estrangeira referenciando essa tabela. De maneira análoga, para inserir os dados de um

produto é necessário primeiro inserir um registro na tabela "geral" *Product* e só depois na tabela "específica" associada ao registro (*PC*, *Laptop* ou *Printer*).

```
CREATE TABLE Product
(    maker TEXT NOT NULL,
    model INT  NOT NULL,
    type  TEXT NOT NULL,
    PRIMARY KEY (model)
);

CREATE TABLE PC
(
    model INT  NOT NULL,
    speed NUM  NOT NULL,
    ram   INT  NOT NULL,
    hd    INT  NOT NULL,
    price NUM,
    PRIMARY KEY (model),
    FOREIGN KEY (model) REFERENCES Product(model)
);

CREATE TABLE Laptop
(
    model  INT  NOT NULL,
    speed  NUM  NOT NULL,
    ram    INT  NOT NULL,
    hd     INT  NOT NULL,
    screen NUM  NOT NULL,
    price NUM,
    PRIMARY KEY (model),
    FOREIGN KEY (model) REFERENCES Product(model)
);

CREATE TABLE Printer
(
    model  INT      NOT NULL,
    color  BOOLEAN  NOT NULL,
    type   TEXT     NOT NULL,
    price NUM,
    PRIMARY KEY (model),
    FOREIGN KEY (model) REFERENCES Product(model)
);

INSERT INTO Product VALUES('A',1001,'pc');
INSERT INTO Product VALUES('A',1002,'pc');
INSERT INTO Product VALUES('A',1003,'pc');
INSERT INTO Product VALUES('A',2004,'laptop');
INSERT INTO Product VALUES('A',2005,'laptop');
INSERT INTO Product VALUES('A',2006,'laptop');
INSERT INTO Product VALUES('B',1004,'pc');
INSERT INTO Product VALUES('B',1005,'pc');
INSERT INTO Product VALUES('B',1006,'pc');
INSERT INTO Product VALUES('B',2007,'laptop');
INSERT INTO Product VALUES('C',1007,'pc');
INSERT INTO Product VALUES('D',1008,'pc');
```

```
INSERT INTO Product VALUES('D',1009,'pc');
INSERT INTO Product VALUES('D',1010,'pc');
INSERT INTO Product VALUES('D',3004,'printer');
INSERT INTO Product VALUES('D',3005,'printer');
INSERT INTO Product VALUES('E',1011,'pc');
INSERT INTO Product VALUES('E',1012,'pc');
INSERT INTO Product VALUES('E',1013,'pc');
INSERT INTO Product VALUES('E',2001,'laptop');
INSERT INTO Product VALUES('E',2002,'laptop');
INSERT INTO Product VALUES('E',2003,'laptop');
INSERT INTO Product VALUES('E',3001,'printer');
INSERT INTO Product VALUES('E',3002,'printer');
INSERT INTO Product VALUES('E',3003,'printer');
INSERT INTO Product VALUES('F',2008,'laptop');
INSERT INTO Product VALUES('F',2009,'laptop');
INSERT INTO Product VALUES('G',2010,'laptop');
INSERT INTO Product VALUES('H',3006,'printer');
INSERT INTO Product VALUES('H',3007,'printer');

INSERT INTO PC VALUES(1001,2.66,1024,250,2114);
INSERT INTO PC VALUES(1002,2.1,512,250,995);
INSERT INTO PC VALUES(1003,1.42,512,80,478);
INSERT INTO PC VALUES(1004,2.8,1024,250,649);
INSERT INTO PC VALUES(1005,3.2,512,250,630);
INSERT INTO PC VALUES(1006,3.2,1024,320,1049);
INSERT INTO PC VALUES(1007,2.2,1024,200,510);
INSERT INTO PC VALUES(1008,2.2,2048,250,770);
INSERT INTO PC VALUES(1009,2,1024,250,650);
INSERT INTO PC VALUES(1010,2.8,2048,300,770);
INSERT INTO PC VALUES(1011,1.86,2048,160,959);
INSERT INTO PC VALUES(1012,2.8,1024,160,649);
INSERT INTO PC VALUES(1013,3.06,512,80,529);

INSERT INTO Laptop VALUES(2001,2,2048,240,20.1,3673);
INSERT INTO Laptop VALUES(2002,1.73,1024,80,17,949);
INSERT INTO Laptop VALUES(2003,1.8,512,60,15.4,549);
INSERT INTO Laptop VALUES(2004,2,512,60,13.3,1150);
INSERT INTO Laptop VALUES(2005,2.16,1024,120,17,2500);
INSERT INTO Laptop VALUES(2006,2,2048,80,15.4,1700);
INSERT INTO Laptop VALUES(2007,1.83,1024,120,13.3,1429);
INSERT INTO Laptop VALUES(2008,1.6,1024,100,15.4,900);
INSERT INTO Laptop VALUES(2009,1.6,512,80,14.1,680);
INSERT INTO Laptop VALUES(2010,2,2048,160,15.4,2300);

INSERT INTO Printer VALUES(3001,1,'ink-jet',99);
INSERT INTO Printer VALUES(3002,0,'laser',239);
INSERT INTO Printer VALUES(3003,1,'laser',899);
INSERT INTO Printer VALUES(3004,1,'ink-jet',120);
INSERT INTO Printer VALUES(3005,0,'laser',120);
INSERT INTO Printer VALUES(3006,1,'ink-jet',100);
INSERT INTO Printer VALUES(3007,1,'laser',200);
```

III. Introdução à Instrução SELECT

1 – Conhecendo a Instrução SELECT

Exemplo 1: SELECT * – seleção de todas as colunas e linhas de uma tabela.

```
SELECT *
FROM Product;
```

maker	model	type
A	1001	pc
A	1002	pc
A	1003	pc
A	2004	laptop
A	2005	laptop
A	2006	laptop
B	1004	pc
B	1005	pc
B	1006	pc
B	2007	pc
C	1007	pc
D	1008	pc
D	1009	pc
D	1010	pc
D	3004	printer
D	3005	printer
E	1011	pc
E	1012	pc
E	1013	pc
E	2001	laptop
E	2002	laptop
E	2003	laptop
E	3001	printer
E	3002	printer
E	3003	printer
F	2008	laptop
F	2009	laptop
G	2010	laptop
H	3006	printer
H	3007	printer

O Exemplo 1 mostra o comando SQL para retornar todas as linhas e colunas da tabela *Product*. Esta tabela contém 3 colunas: “maker”, “model” e “type”. Para exibir todas as colunas de uma tabela, deve-se utilizar a palavra-chave **SELECT** com um “*” (asterisco). A tabela a ser consultada deve ser indicada após a palavra chave **FROM**.

Expressão equivalente em Álgebra Relacional - simplesmente escrever o nome da relação

Product

Exemplo 2: Seleção de colunas específicas de uma tabela.

```
SELECT model, hd
FROM PC;
```

model	hd
1001	250
1002	250
1003	80
1004	250
1005	250
1006	320
1007	200
1008	250
1009	250
1010	300
1011	160
1012	160
1013	80

Para selecionar colunas específicas de uma tabela (operação de projeção), uma lista com os nomes das colunas deve ser especificada logo após a palavra-chave SELECT (os nomes das colunas devem ser separados por vírgulas). O Exemplo 2 ilustra o comando SQL para recuperar o modelo ("model") e o tamanho do hard disk ("hd") de todos os registros da tabela *PC*. No conjunto de resultados retornados, observe que o primeiro registro corresponde ao modelo de número 1001 e possui hd de 250GB. O segundo registro corresponde ao modelo de número 1002 também com hd de 250GB. E assim por diante.

Expressão equivalente em Álgebra Relacional

$\pi_{model,hd}(PC)$

DEFINIÇÃO I: Sintaxe da instrução SELECT básica

Conforme foi visto nos exemplos anteriores, uma instrução SELECT básica deve ser composta por ao menos duas cláusulas (partes):

1. Uma cláusula SELECT que especifica as colunas a serem exibidas (* ou lista de colunas)

2. Uma cláusula FROM que especifica a tabela que contém as colunas listadas na cláusula SELECT.

A sintaxe de uma instrução SELECT básica é apresentada no quadro a seguir (o símbolo "|" significa "ou").

```
SELECT * | col1, ..., coln
FROM tabela
```

As instruções SQL não fazem distinção entre letras maiúsculas e minúsculas. Muitas vezes as cláusulas costumam ser colocadas em linhas separadas (como na definição acima e nos exemplos apresentados) com a única finalidade de melhorar a legibilidade (ou seja: não há problema em colocar a cláusula FROM na mesma linha da cláusula SELECT).

Exemplo 3: Explorando uma tabela com a palavra-chave DISTINCT – supressão de linhas duplicadas.

Uma diferença fundamental entre a Álgebra Relacional e a SQL está no fato de que na Álgebra Relacional **tuplas repetidas são sempre suprimidas da relação resultante** de uma operação. Por exemplo, a expressão $\pi_{hd}(PC)$ retorna os valores distintos do atributo "hd" da relação *PC*:

$\pi_{hd}(PC)$

```
250
80
320
200
300
160
```

Ou seja: embora a relação PC possua 13 tuplas, existem apenas 6 valores distintos para o atributo "hd"; por isso $\pi_{hd}(PC)$ retorna uma relação com 6 tuplas. Isto ocorre porque a Teoria Relacional estabelece que não devem existir tuplas repetidas em uma relação (já que uma relação é um conjunto de tuplas e, por definição, conjuntos não podem ter elementos repetidos).

Por questões práticas, o comportamento padrão da instrução SQL é exatamente o oposto: **nenhuma linha é suprimida**[2] do resultado, mesmo que existam linhas com valores repetidos. Isso porque os fabricantes de SGBD's preferiram definir a tabela como uma estrutura do tipo *multiset* ou *bag* ("saco"). Melhor explicando: uma tabela é um *multiset* de tuplas enquanto uma relação é um conjunto de tuplas. E, conceitualmente, um *multiset* nada mais é do que um conjunto que permite exemplos repetidos[3]. Para que isso fique claro, observe o resultado do comando SQL da página a seguir e compare com os resultados da expressão da Álgebra Relacional acima.

[2]Apenas as operações de conjunto suprimem linhas repetidas na SQL como veremos na Seção IV.

[3] Mais informações em: https://en.wikipedia.org/wiki/Multiset e https://en.wikipedia.org/wiki/Table_(database)

```
SELECT hd
FROM PC;

hd
250
250
80
250
250
320
200
250
250
300
160
160
80
```

Para suprimir valores repetidos, é preciso utilizar a palavra-chave DISTINCT:

```
SELECT DISTINCT hd
FROM PC;

hd
250
80
320
200
300
160
```

A palavra-chave DISTINCT é utilizada para fazer com que linhas duplicadas apareçam somente uma vez na resposta ao SELECT. Na prática, a principal utilização do comando é permitir com que seja possível **"estudar" uma coluna** de uma tabela, já que o DISTINCT permite **identificar todos os valores distintos armazenados** na mesma. No exemplo acima, o resultado da consulta nos ajuda a identificar, de forma rápida que o(s) PC(s) com menor e maior capacidade de hard disk possuem, respectivamente, 80 e 320 gigabytes.

No Exemplo 2, dissemos que a instrução `SELECT model, hd FROM PC` é equivalente à expressão da Álgebra Relacional $\pi_{\texttt{model,hd}}(PC)$ mesmo sem a presença do comando DISTINCT na expressão SQL. Isto porque, neste caso, **a chave primária faz parte da consulta** (campo "model"). Como uma PK nunca possui valores repetidos, por consequência uma instrução SELECT que a inclua na lista de campos nunca trará resultados repetidos.

DISTINCT: remoção de linhas duplicadas

É importante reforçar que o objetivo do comando DISTINCT é **remover linhas duplicadas** de um conjunto de resultados, reproduzindo assim o comportamento equivalente ao padrão da Álgebra Relacional. Caso haja alguma dúvida sobre este conceito, compare os resultados retornados pelas duas seguintes instruções:

```
1. SELECT type FROM Product

2. SELECT DISTINCT type FROM Product
```

A primeira instrução SELECT retorna o tipo de todos os 30 produtos da tabela *Product* e, por consequência, produz um conjunto resposta de 30 linhas. Já na segunda instrução SELECT, o comando DISTINCT é utilizado para forçar com que todas as linhas duplicadas sejam desconsideradas. Por este motivo, apenas 3 linhas são mantidas no resultado final.

Exemplo 4: Utilizando a palavra-chave DISTINCT com várias colunas.

```
SELECT DISTINCT speed, ram FROM PC;
```

speed	ram
2.66	1024
2.1	512
1.42	512
2.8	1024
3.2	512
3.2	1024
2.2	1024
2.2	2048
2	1024
2.8	2048
1.86	2048
3.06	512

A palavra-chave DISTINCT também pode ser utilizada com várias colunas. Com isto, todas as **combinações de valores distintas** das colunas podem ser identificadas. Na consulta acima, o comando DISTINCT foi empregado para determinar todas combinações de valores existentes de duas colunas da tabela *PC*: "speed" e "ram". Analisando os resultados desta consulta, é possível observar que embora existam 13 modelos gravados na tabela *PC*, a consulta retornou 12 linhas (uma linha foi suprimida do resultado final). Isto quer dizer que existem dois modelos de PC com os mesmos valores para "speed" e "ram" (se você verificar o conteúdo completo da tabela, verá que se trata dos modelos 1004 e 1012).

DISTINCT *versus* Tabelas de Frequência

O comando DISTINCT é útil em situações onde estamos "explorando" uma base de dados e desemos identificar todos os diferentes valores de uma ou mais colunas. No entanto, ele **não** serve para computar tabelas de frequência. Ou seja: o comando não determina a frequência com que cada valor ou combinação de valores ocorre. Para computar tabelas de frequência, é preciso trabalhar com **funções de grupo –** tema muito importante, mas que será abordado na segunda parte de nosso curso

Exemplo 5: Definição de rótulos (ou "apelidos") para colunas.

```
SELECT model AS Modelo, maker AS Fabricante, type AS Tipo
FROM Product;

Modelo  Fabricante  Tipo
1001    A           pc
1002    A           pc
1003    A           pc
2004    A           laptop
...     ...         ...
3007    H           printer
```

Ao exibir os resultados de uma consulta, a SQL utiliza automaticamente o nome de cada coluna no cabeçalho. No entanto, também é possível definir rótulos ou "apelidos" para as colunas (função similar ao operador de renomeação da Álgebra Relacional). Os rótulos devem ser indicados na cláusula SELECT, imediatamente após o nome da coluna. A palavra-chave **AS** pode ser colocada entre o nome da coluna e o rótulo (ex: `maker AS Fabricante`); no entanto, na maioria dos SGBD's o uso de AS é opcional (por exemplo, basta especificar: `maker Fabricante`). Alguns SGBD's permitem a definição de rótulos que contenham espaços ou caracteres especiais (tais como "*", "#" ou "$"). Nesse caso, é preciso definir o nome do rótulo entre aspas duplas.

DEFINIÇÃO II: Modelo estendido para a sintaxe da instrução SELECT básica.

Conforme foi visto nos exemplos, as instruções SQL podem, **opcionalmente**, conter rótulos; Também de maneira opcional, uma instrução SQL pode fazer uso da palavra-chave DISTINCT para suprimir linhas repetidas. Com isto, o quadro a seguir apresenta o modelo estendido para a sintaxe de uma instrução SELECT (as partes opcionais são exibidas entre colchetes)

```
SELECT [DISTINCT] * | col1 [AS apelido1],..., coln  [AS apelidon]
FROM tabela
```

Exercícios Propostos 1

1. Crie e execute uma consulta para recuperar todas as linhas e colunas da tabela *Printer*.

2. Crie e execute uma consulta para recuperar todas as linhas e colunas da tabela *Printer* Nesta consulta, utilize o rótulo “COLORIDA” para a coluna “color”.

3. Por que a coluna “model” não foi incluída no comando SELECT apresentado na primeira instrução SQL apresentada na página 11?

4. Crie uma consulta para recuperar as colunas “ram” e “hd” (nesta ordem) da tabela *Laptop*.

5. Crie uma consulta para recuperar todas as combinações distintas de valores das colunas “ram” e “hd” armazenados em *Laptop*.

2 – Restringindo o Conjunto de Resultados

Ao recuperar informações de uma tabela, pode haver a necessidade de utilizar algum critério para que apenas algumas de suas linhas sejam recuperadas (ou seja: para selecionar as linhas de interesse). Esta lição aborda a cláusula WHERE, utilizada como operador de seleção na SQL.

Exemplo 6: Utilizando a cláusula WHERE – um exemplo simples.

```
SELECT * FROM Product
WHERE type = 'printer';

maker model type
D     3004  printer
D     3005  printer
E     3001  printer
E     3002  printer
E     3003  printer
H     3006  printer
H     3007  printer
```

No Exemplo 6, a instrução SELECT recupera os produtos cujo valor da coluna "type" é igual a 'printer' (ou seja: os produtos do tipo impressora). Para tal, foi preciso utilizar a cláusula WHERE com a condição type = 'printer'. Mais detalhadamente, a instrução SQL solicita com que o banco de dados recupere **todas as colunas da tabela *Product* considerando apenas as linhas onde a coluna "type" possui valor igual a 'printer'**. Como só existem 7 linhas que atendem a esta condição, apenas sete resultados são retornados no conjunto resposta.

A cláusula **WHERE** é utilizada na SQL para implementar a operação de **seleção** da Álgebra Relacional. É com o uso desta cláusula que são especificados os critérios para definir os registros que farão parte de um resultado. Veja no quadro a seguir a mesma consulta do Exemplo 6 expressa em Álgebra Relacional.

Expressão equivalente em Álgebra Relacional

$\sigma_{type='printer'}(Product)$

Exemplo 7: WHERE – Teste em uma coluna não recuperada.

A instrução SQL a seguir recupera os valores distintos de velocidade de PCs considerando *apenas os PCs que possuem preço inferior a US$ 900*. Note que a instrução recupera o campo "speed", mas faz a seleção dos registros a serem exibidos através de um teste na coluna "price".

```
SELECT DISTINCT speed
FROM PC
WHERE price < 900;
```

speed
1.42
2.8
3.2
2.2
2
3.06

Expressão equivalente em Álgebra Relacional

$\pi_{speed}(\sigma_{price < 900}(PC))$

Colunas Exibidas *versus* Colunas Testadas

Toda instrução SELECT básica começa com a palavra-chave SELECT seguida do nome de uma ou mais colunas que serão exibidas para o usuário. Depois vem a palavra FROM seguida do nome de uma tabela (as colunas exibidas devem pertencer a esta tabela).

Caso seja necessário restringir linhas, utiliza-se a cláusula WHERE associada a um ou mais testes sobre colunas da tabela indicada na cláusula FROM

É importante deixar claro que existe uma **independência entre as colunas exibidas e as colunas testadas**. Em outras palavras: é possível envolver colunas em testes da cláusula WHERE (seleção) mesmo que elas não façam parte da lista de colunas a serem recuperadas no SELECT (projeção). A única exigência é que todas as colunas façam parte da(s) tabela(s) indicada(s) na cláusula FROM.

Por exemplo, se uma tabela *X* possui as colunas “a”, “b”, “c”, é possível montar uma instrução que recupere as colunas “a” e “b”, baseada em um teste na coluna “c” (ex.: SELECT a, b FROM *X* WHERE c = 1000).

Exemplo 8: WHERE – Utilizando operadores lógicos

A Tabela III.1 apresenta os **operadores lógicos** disponíveis na Linguagem SQL. Estes operadores combinam o resultado de duas ou mais condições especificadas na cláusula WHERE. Após a tabela, são apresentados e comentados diversos exemplos de utilização prática para os operadores lógicos.

Tabela III.1 – Operadores Lógicos

Operador	Significado
AND	Retorna TRUE se todas as condições avaliadas forem verdadeiras.
OR	Retorna TRUE se ao menos uma das condições avaliadas for verdadeira.
NOT	Retorna TRUE se a condição seguinte for FALSE.

Exemplo 8.1: Recupera todos os laptops que possuem velocidade de 2.0, memória de 2048 e hard disk com capacidade superior a 100. Todas as colunas são retornadas (SELECT *).

```
SELECT * FROM Laptop
WHERE speed = 2.0 AND ram = 2048 AND hd > 100;

model  speed  ram   hd    screen  price
2001   2      2048  240   20.1    3673
2010   2      2048  160   15.4    2300
```

Exemplo 8.2: Recupera o número do modelo de todas as impressoras laser coloridas.

```
SELECT model FROM Printer
WHERE color = 1 AND type = 'laser';

model
3003
3007
```

Exemplo 8.3: Recupera os PC's com memória de 2048 ou hard disk de 250

```
SELECT *
FROM PC
WHERE ram = 2048 OR hd = 250;

model  speed  ram   hd    price
1001   2.66   1024  250   2114
1002   2.1    512   250   995
1004   2.8    1024  250   649
1005   3.2    512   250   630
1008   2.2    2048  250   770
1009   2      1024  250   650
1010   2.8    2048  300   770
1011   1.86   2048  160   959
```

Exemplo 8.4: Recupera todos os produtos que não foram produzidos pelo fabricante 'A'.

```
SELECT * FROM Product
WHERE NOT (maker = 'A');
```

maker	model	type
B	1004	pc
B	1005	pc
B	1006	pc
B	2007	pc
C	1007	pc
D	1008	pc
D	1009	pc
D	1010	pc
D	3004	printer
D	3005	printer
E	1011	pc
E	1012	pc
E	1013	pc
E	2001	laptop
E	2002	laptop
E	2003	laptop
E	3001	printer
E	3002	printer
E	3003	printer
F	2008	laptop
F	2009	laptop
G	2010	laptop
H	3006	printer
H	3007	printer

Exemplo 8.5: Recupera todos os produtos que não foram produzidos pelos fabricante 'A', 'E' e 'F'.

```
SELECT * FROM Product
WHERE NOT (maker = 'A' OR maker = 'E' OR maker = 'F');
```

maker	model	type
B	1004	pc
B	1005	pc
B	1006	pc
B	2007	pc
C	1007	pc
D	1008	pc
D	1009	pc
D	1010	pc
D	3004	printer
D	3005	printer
G	2010	laptop
H	3006	printer
H	3007	printer

DEFINIÇÃO III: Cláusula WHERE

A cláusula WHERE é utilizada para restringir as linhas que serão retornadas ou analisadas por uma consulta, de acordo com um ou mais critérios. Trata-se de uma cláusula opcional, ao contrário de SELECT e FROM, que são cláusulas obrigatórias. Dentro de um comando SQL, a palavra-chave WHERE deve ser colocada logo após a lista de tabelas associadas à cláusula FROM (Obs.: a cláusula FROM conterá mais de uma tabela sempre que for necessário realizar uma operação de junção; porém este assunto será abordado apenas na Seção VI). Além disso, a palavra-chave WHERE deve possuir uma ou mais condições associadas.

```
SELECT [DISTINCT] * | col1 [AS apelido1],..., coln  [AS apelidon]
FROM tabela
[WHERE condição(ões)]
```

Exemplo 9: WHERE – Utilizando operadores de comparação

A Tabela III.2 apresenta alguns dos operadores de comparação que podem ser utilizados em conjunto com a cláusula WHERE. Note que alguns operadores já foram utilizados em exemplos previamente apresentados.

Tabela III.2 – Operadores de Comparação

Operador	Significado
=	Igual a
<>	Diferente
>	Maior do que
>=	Maior ou igual a
<	Menor do que
<=	Menor ou igual a
BETWEEN	Verifica se valor está dentro de uma determinada faixa de valores
IN	Verifica se valor pertence a um conjunto de valores
LIKE	Verifica se texto (valor alfanumérico) "casa" com um determinado padrão (*veremos apenas na segunda parte da matéria)*

Exemplo 9.1: Recupera todos os produtos que não foram produzidos pelo fabricante 'A'. OBS: o resultado é idêntico ao do Exemplo 8.4.

```
SELECT *
FROM Product
WHERE maker <> 'A';

maker model type
B     1004  pc
B     1005  pc
B     1006  pc
B     2007  pc
C     1007  pc
D     1008  pc
```

```
D     1009   pc
D     1010   pc
D     3004   printer
D     3005   printer
E     1011   pc
E     1012   pc
E     1013   pc
E     2001   laptop
E     2002   laptop
E     2003   laptop
E     3001   printer
E     3002   printer
E     3003   printer
F     2008   laptop
F     2009   laptop
G     2010   laptop
H     3006   printer
H     3007   printer
```

Exemplo 9.2: Recupera os PCs que possuem velocidade de **pelo menos** 3.00 giga-hertz.

```
SELECT * FROM PC
WHERE speed >= 3;

model  speed  ram   hd    price
1005   3.2    512   250   630
1006   3.2    1024  320   1049
1013   3.06   512   80    529
```

Exemplo 9.3: Recupera os números de modelo dos PC's com preço **igual ou inferior a 900** e que possuem memória de **pelo menos 1024**.

```
SELECT model FROM PC
WHERE price <= 900 AND ram >= 1024;

model
1004
1007
1008
1009
1010
1012
```

<u>Exemplo 10</u>: IMPORTANTE → Regras de precedência para operadores lógicos

A Tabela III.3 apresenta a ordem de precedência aplicada quando vários operadores lógicos estão presentes numa mesma instrução SQL. O operador NOT tem precedência maior, seguido do operador AND e do operador OR, respectivamente. Estas regras de precedência

podem ser sobrepostas através do uso de parênteses (isto é válido também para a Álgebra Relacional).

Tabela III.3 – Regras de Precedência para Operadores Lógicos

Operador	Ordem de Avaliação
NOT	1
AND	2
OR	3

Considere o seguinte exemplo: suponha que um usuário deseja obter uma lista de PCs que possuam memória de 1024 ou 2048 e que tenham preço inferior a 900. Neste caso, o SELECT abaixo **não atenderia** ao desejo do usuário, pois retornaria o conjunto de modelos de PCs errado.

```
SELECT * FROM PC
WHERE ram = 1024 OR ram = 2048 AND price < 900;

model  speed  ram   hd    price
1001   2.66   1024  250   2114
1004   2.8    1024  250   649
1006   3.2    1024  320   1049
1007   2.2    1024  200   510
1008   2.2    2048  250   770
1009   2      1024  250   650
1010   2.8    2048  300   770
1012   2.8    1024  160   649
```

O problema está no seguinte fato: como o operador AND tem precedência maior do que a do operador OR, o SGBD testa a condição especificada em vermelho negrito primeiro. Ou seja, o SELECT será interpretado pelo SGBD da seguinte maneira: "selecione a linha de PC caso a memória seja de 2048 (ram=2048) e o preço for menor do que 900 (price < 900); ou se a memória for de 1024 (ram=1024)". Por isto, o SELECT acaba retornando todos os modelos que possuem memória de 1024, independente do preço ser inferior a 900 ou não (por esta razão, os modelos 1001 e 1006 foram retornados pela consulta).

Observe agora a consulta a seguir, onde os parênteses são empregados para alterar a precedência e, desta forma, recuperar o conjunto de PC's correto:

```
SELECT * FROM PC
WHERE (ram = 1024 OR ram = 2048) AND price < 900;

model  speed  ram   hd    price
1004   2.8    1024  250   649
1007   2.2    1024  200   510
1008   2.2    2048  250   770
1009   2      1024  250   650
1010   2.8    2048  300   770
1012   2.8    1024  160   649
```

Desta vez, o SELECT será interpretado pelo SGBD da seguinte maneira: "selecione a linha se o PC possuir memória de 1024 ou 2048 e se o mesmo tiver preço inferior a 900".

Exemplo 11: Operador BETWEEN (parte 1)

```
SELECT * FROM PC
WHERE price BETWEEN 700 AND 1000
```

model	speed	ram	hd	price
1002	2.1	512	250	995
1008	2.2	2048	250	770
1010	2.8	2048	300	770
1011	1.86	2048	160	959

O operador BETWEEN serve para que sejam recuperadas apenas as linhas cujo o conteúdo do campo avaliado esteja **contido numa determinada faixa de valores**. A instrução SELECT acima recupera os dados dos PC's cujo preço se encontra situado entre US$ 700 e US$ 1000 (inclusive 700 e 1000).

Exemplo 12: Operador BETWEEN (parte 2)

É importante esclarecer que o **limite inferior deve ser especificado inicialmente**. Outra observação importante é que, embora o operador BETWEEN seja mais utilizado para a avaliação de campos numéricos (inteiros ou reais), ele pode ser utilizado também para o teste de campos alfanuméricos. Neste caso, ele realiza uma comparação lexicográfica (mais detalhes no Exemplo 16). No exemplo abaixo, o operador BETWEEN é utilizado para retornar todos os números de modelos de laptops dos fabricantes 'E', 'F' e 'G'.

```
SELECT * FROM Product
WHERE maker BETWEEN 'E' AND 'G'
AND type='laptop'
```

maker	model	type
E	2001	laptop
E	2002	laptop
E	2003	laptop
F	2008	laptop
F	2009	laptop
G	2010	laptop

Exemplo 13: Operador IN (parte 1)

```
SELECT * FROM Product
WHERE maker IN ('E','F','G')
AND type='laptop';
```

maker	model	type
E	2001	laptop
E	2002	laptop
E	2003	laptop

```
F     2008   laptop
F     2009   laptop
G     2010   laptop
```

O operador IN permite identificar se o valor do campo avaliado **pertence a um determinado conjunto**, que deve ser especificado entre parênteses. No exemplo acima a consulta retorna os produtos do tipo 'laptop' produzidos pelos fabricantes 'E', 'F' e 'G' (OBS: retorna os mesmos registros do Exemplo 12). Este operador pode ser utilizado tanto para avaliar campos alfanuméricos, como para a avaliação de campos numéricos do **tipo inteiro**. A instrução abaixo recupera os modelos de PC com hard disk de 80, 200 ou 300 GB.

```
SELECT model FROM PC
WHERE hd IN (80, 200, 300);

model
1003
1007
1010
1013
```

Exemplo 14: Operador NOT IN

```
SELECT * FROM Product WHERE maker NOT IN ('A','E','F');

maker model type
B     1004   pc
B     1005   pc
B     1006   pc
B     2007   pc
C     1007   pc
D     1008   pc
D     1009   pc
D     1010   pc
D     3004   printer
D     3005   printer
G     2010   laptop
H     3006   printer
H     3007   printer
```

O operador NOT IN permite identificar se o valor do campo avaliado **não pertence a um determinado conjunto**. No exemplo acima a consulta retorna os produtos que não são produzidos pelos fabricantes 'A', 'E' e 'F' (OBS: retorna os mesmos registros do Exemplo 8.5).

Exemplo 15: Comparação Lexicográfica

```
SELECT * FROM Product WHERE maker > 'F';

maker model type
G     2010   laptop
```

```
H       3006   printer
H       3007   printer
```

A consulta acima é interpretada pelo SGBD da seguinte forma: “recuperar todos os produtos cujo fabricante é maior do que 'F' ”. Observe que o campo “maker”, utilizado na cláusula WHERE, é um atributo do tipo alfanumérico. Quando um campo deste tipo é avaliado pelos operadores “<”, “<=”, “>=” ou “>” e também pelo operador BETWEEN, o SGBD executará uma **comparação lexicográfica** (ou seja, uma avaliação que é baseada em ordenação alfabética). No caso da consulta do Exemplo 16, são retornados todos os produtos cujo nome é “lexicograficamente superior” à F (ou seja: G e H!).

Apesar do exemplo apresentado parecer não muito útil, existem muitas situações práticas em que a comparação lexicográfica pode ser útil. Considere por exemplo uma tabela que armazena os alunos do curso de banco de dados. Neste caso, seria possível obter a relação de todos os alunos cujo nome começa com a letra “A” utilizando-se o seguinte comando SQL:

```
SELECT * FROM t_aluno WHERE nome < 'B';
```

<u>Exemplo 16</u>: Executando operações aritméticas na cláusula WHERE

```
SELECT model, price FROM PC
WHERE (price * 3.30) <= 2500;


model price
1003  478
1004  649
1005  630
1007  510
1009  650
1012  649
1013  529
```

É possível utilizar operações aritméticas sobre colunas na cláusula WHERE. O exemplo acima retorna todos os modelos e preços dos PC's cujo preço seja inferior a R$ 2500. Neste exemplo, considere que “price” armazena os preços em dólar e que o valor de 1 dólar no dia em que a consulta foi executada equivalia a R$ 3,30. Em SQL, o símbolo “*” é utilizado como operador de multiplicação. Já os símbolos “+”, “-” e “/” são utilizados como operadores de adição, subtração e divisão, respectivamente.

<u>Exemplo 17</u>: Executando operações aritméticas na cláusula SELECT

```
SELECT model, price AS preco_dolar, (price * 3.30) as preco_real FROM PC
WHERE (price * 3.30) <= 2500;

model preco_dolar  preco_real
1003  478          1577.4
1004  649          2141.7
1005  630          2079
```

```
1007  510          1683
1009  650          2145
1012  649          2141.7
1013  529          1745.7
```

Também é possível utilizar operações aritméticas junto à cláusula SELECT. No SQL acima, o preço é convertido para valor em R$ na última coluna retornada.

A Cláusula WHERE nas Instruções DELETE e UPDATE

Além de ser utilizada com a instrução SELECT a cláusula WHERE também é utilizada nas instruções DELETE e UPDATE. Quando utilizada junto à instrução DELETE, a condição especificada na WHERE determinará as linhas que serão removidas. Já na instrução UPDATE, a condição determina as linhas que serão atualizadas.

Não é possível utilizar a cláusula WHERE em nenhuma outra instrução da SQL que não seja SELECT, UPDATE ou DELETE.

Exercícios Propostos 2

1. Crie uma consulta para recuperar todas as impressoras laser coloridas.

2. Crie uma consulta para recuperar todas as impressoras laser coloridas com preço igual ou inferior US$ 200.

3. Cria uma consulta para recuperar todos os laptops com as seguintes características:
   - Com preço igual ou inferior a US$ 1000,00 (independente das demais características).
   - Com tamanho de tela igual ou superior a 15 polegadas e preço igual ou inferior a US$ 2000,00 (independente das demais características).

4. Crie duas consultas distintas, uma utilizando o operador IN e outra sem utilizar este operador, para recuperar todos os modelos produzidos pelos fabricantes 'B', 'E' e 'H'.

3 – Ordenando o Conjunto de Resultados

Em muitas situações práticas pode existir a necessidade de especificar a ordem de exibição dos registros (ex.: em um site de compras, exibir os laptops por ordem crescente de preço se o usuário assim o desejar). Esta seção apresenta a cláusula ORDER BY, que pode ser intruduzida no final de um comando SELECT para ordenar os resultados.

Exemplo 19: Utilizando a cláusula ORDER BY

```
SELECT * FROM Laptop
WHERE screen > 15
ORDER BY price;

model speed  ram   hd     screen price
2003  1.8    512   60     15.4   549
2008  1.6    1024  100    15.4   900
2002  1.73   1024  80     17     949
2006  2      2048  80     15.4   1700
2010  2      2048  160    15.4   2300
2005  2.16   1024  120    17     2500
2001  2      2048  240    20.1   3673
```

Por definição, os resultados retornados após a execução de um comando SELECT **não são ordenados por nenhum critério** (ou seja: nenhuma coluna específica… o SGBD entrega os resultados na ordem que for "mais rápida" de buscá-los). No entanto, é possível especificar que os resultados sejam ordenados por uma ou mais colunas, através do uso da cláusula ORDER BY. O exemplo acima mostra uma consulta que retorna os laptops com tela acima de 15 polegadas, apresentando os resultados ordenados pela coluna "price" em ordem ascendente.

ORDER BY e Álgebra Relacional Básica

Na álgebra relacional "básica", apresentada nas últimas aulas, não existe o operador equivalente ao ORDER BY. Ele existe apenas na álgebra relacional "estendida", que não é coberta em nosso curso.

Exemplo 20: Utilizando a cláusula ORDER BY com mais de uma coluna

```
SELECT * FROM Laptop
ORDER BY ram, hd;

model speed  ram   hd     screen price
2003  1.8    512   60     15.4   549
2004  2      512   60     13.3   1150
2009  1.6    512   80     14.1   680
```

```
2002  1.73   1024   80     17     949
2008  1.6    1024   100    15.4   900
2005  2.16   1024   120    17     2500
2007  1.83   1024   120    13.3   1429
2006  2      2048   80     15.4   1700
2010  2      2048   160    15.4   2300
2001  2      2048   240    20.1   3673
```

Esta consulta retorna todos os registros de *Laptop* ordenados de maneira ascendente pela quantidade de memória (primeiro critério) e de maneira ascendente pelo tamanho do hard disk (como segundo critério). Não há limite para o número de colunas a serem especificadas na cláusula ORDER BY.

Exemplo 21: Ordenação descendente

```
SELECT * FROM Laptop
ORDER BY price DESC;

model  speed  ram   hd     screen price
2001  2      2048   240    20.1   3673
2005  2.16   1024   120    17     2500
2010  2      2048   160    15.4   2300
2006  2      2048   80     15.4   1700
2007  1.83   1024   120    13.3   1429
2004  2      512    60     13.3   1150
2002  1.73   1024   80     17     949
2008  1.6    1024   100    15.4   900
2009  1.6    512    80     14.1   680
2003  1.8    512    60     15.4   549
```

Esta consulta retorna todos os registros da tabela *Laptop* por ordem **descendente** de preço. A palavra-chave DESC é utilizada na cláusula ORDER BY para esta finalidade.

Exemplo 22: Combinando as ordenações ascendente e descendente

```
SELECT * FROM Laptop
ORDER BY ram DESC, hd ASC;

model  speed  ram   hd     screen price
2006  2      2048   80     15.4   1700
2010  2      2048   160    15.4   2300
2001  2      2048   240    20.1   3673
2002  1.73   1024   80     17     949
2008  1.6    1024   100    15.4   900
2005  2.16   1024   120    17     2500
2007  1.83   1024   120    13.3   1429
2003  1.8    512    60     15.4   549
2004  2      512    60     13.3   1150
2009  1.6    512    80     14.1   680
```

A consulta retorna todos os registros de *Laptop* ordenados de maneira descendente pela quantidade de memória como primeiro critério e de maneira ascendente pelo tamanho do hard disk como segundo critério. A palavra-chave ASC ao lado de "hd" poderia ser omitida, já que a cláusula ORDER BY classifica os resultados de maneira ascendente por padrão.

Ordenação por Posição Relativa de um Campo

Os SGBDs também permitem utilizar a cláusula ORDER BY para ordernar resultados pela posição relativa em um resultado, onde o primeiro campo do resultado tem a posição relativa 1, o campo seguinte 2, e assim por diante. Por exemplo:

```
SELECT model, hd, price FROM Laptop ORDER BY 2
```

Esta consulta ordena os resultados por "hd", pois esse é o segundo campo especificado no SELECT (de maneira análoga ORDER BY 3 ordenaria os resultados pelo campo "price" e ORDER BY 1 por "model").

Exemplo 23: Ordenação por um campo não-selecionado

```
SELECT model, speed, ram FROM Laptop ORDER BY price;

model speed ram
2003  1.8   512
2009  1.6   512
2008  1.6   1024
2002  1.73  1024
2004  2     512
2007  1.83  1024
2006  2     2048
2010  2     2048
2005  2.16  1024
2001  2     2048
```

Esta consulta retorna o modelo, velocidade e memória de todos os laptops classificados de maneira ascendente pelo preço. Observe, entretanto, que o campo "price" (critério de ordenação) não é uma coluna retornada pela instrução SQL. (Dica: execute novamente o SQL selecionando também o campo "price", para confirmar que a ordenação foi realizada de forma correta).

DEFINIÇÃO IV: Cláusula ORDER BY

A ordem de exibição dos resultados de uma consulta pode ser especificada pela cláusula ORDER BY. Quando esta cláusula for utilizada, ela deverá ser **sempre a última da instrução SQL**.

```
SELECT [DISTINCT] * | col1 [AS apelido1],..., coln  [AS apelidon]
FROM tabela
[WHERE condição(ões)]
[ORDER BY lista de campos]
```

Referências a Apelidos de Colunas

Na maioria dos SGBDs, os apelidos de colunas podem ser referenciados em uma cláusula ORDER BY, mas não podem ser referenciados em uma cláusula WHERE.

4 – O Valor Nulo (NULL) e os Operadores IS NULL e IS NOT NULL

O conceito de valor **Nulo** (ou *NULL*) é utilizado em banco de dados relacionais para representar a **ausência de informação** sobre um determinado campo. Detalhando melhor, se um campo de uma linha contém valor nulo, isto indica que seu conteúdo é **desconhecido**, **inexistente**, ou **não-aplicável**.

É importante entender que nulo **não é** a mesma coisa que zero. Zero é um número! Considere, por exemplo, uma tabela que armazene os dados de funcionários de uma empresa. Se a coluna "número de filhos" de um funcionário armazena o valor 0, isso indica que este não possui filhos (possui 0 filhos). Porém, se a coluna armazena o valor NULL, isto significa que não é sabido se o funcionário possui filhos ou não. De maneira análoga, também é importante ressaltar que o valor nulo **não é** a mesma coisa que espaço em branco, já que um espaço em branco representa um caractere.

Esta seção apresentará os operadores IS NULL e IS NOT NULL, utilizados em condições associadas em cláusulas WHERE para tratar valores nulos. Para poder executar os exemplos, inseriremos um produto do tipo "printer" (impressora) na base de dados com o preço nulo. Para tal, execute os comandos INSERT abaixo. Primeiro é preciso inserir na tabela *Product* e depois em *Printer*, para que não ocorra violação de chave estrangeira.

```
INSERT INTO Product VALUES ('H',3010,'printer');

INSERT INTO Printer VALUES (3010,0,'laser',NULL);
```

Exemplo 24: Operador IS NULL

```
SELECT * FROM Printer
WHERE price IS NULL;

model color type   price
3010  0     laser  NULL
```

O operador IS NULL deve ser utilizado sempre que se desejar avaliar se um campo é nulo. **Não funciona fazer o teste utilizando o operador "=" você precisa usar IS NULL**. No exemplo acima, estão sendo selecionados apenas os dados das impressoras cujo valor do campo "price" seja nulo. A consulta retornará apenas um registro, exatamente correspondente ao modelo 3010, recém-inserido.

Exemplo 25: Operador IS NOT NULL

```
SELECT * FROM Printer
WHERE price IS NOT NULL;

model color type    price
3001  1     ink-jet  99
3002  0     laser    239
3003  1     laser    899
3004  1     ink-jet  120
3005  0     laser    120
3006  1     ink-jet  100
3007  1     laser    200
```

O operador IS NOT NULL é utilizado quando deseja-se avaliar se um campo **não é** nulo. No exemplo acima, estão sendo selecionadas as impressoras cuja coluna "price" não tem valor nulo. **Não funciona fazer o teste utilizando o operador "<>" você precisa usar IS NOT NULL.**

Lógica de 3 Valores

Considere o conteúdo da tabela *Printer*. Ela contém 8 registros, sendo que o recém-inserido modelo 3010 está com valor NULL para o campo "price".

A instrução SELECT abaixo retorna o modelo e preço de todas as impressoras cujo preço é superior a US$ 200,00.

```
SELECT model, price FROM Printer WHERE price > 200

model  price
3002   239
3003   899
```

Já a instrução SELECT abaixo retorna o modelo e preço de todas as impressoras cujo preço é igual ou inferior a US$ 200,00.

```
SELECT model, price FROM Printer WHERE price <= 200

model  price
3001   99
3004   120
3005   120
3006   100
3007   200
```

Observe que nenhuma das duas consultas retornou a impressora com o número de modelo 3010. Por que? A resposta é a seguire: na teoria dos bancos relacionais, a semântica para o valor NULL é um tanto quanto curiosa. Se um campo possui valor NULL isto não deve ser interpretado como valor "branco" ou valor zero, e sim como valor "desconhecido" (unknown), "inexistente" ou "não aplicável".

Por este motivo, as avaliações booleanas em um SGBD possuem comportamento diferente da maioria das linguagens de programação: além de retornar os tradicionais valores TRUE e FALSE,

uma avaliação booleana também podem resultar em um terceiro valor, que é exatamente o valor NULL. Isto é conhecido como **lógica de três valores** (*three-valued logic*).

Na prática a lógica de 3 valores representa o seguinte: **toda comparação envolvendo uma coluna que contenha o valor NULL, resultará em NULL**. Sendo assim, o teste NULL > 200 (o valor do preço da impressora 3010 é superior a 200?) resulta em NULL e não em FALSE. De maneira análoga, o teste NULL <= 200 (o valor da impressora 3010 é igual ou inferior a 200?) também resulta em NULL e não em TRUE.

Isto porque NULL significa desconhecido ("eu não sei"); então o SGBD não pode responder nem "não" (FALSE) e nem "sim" (TRUE) para ambos os testes. Nos dois casos, ele responde "eu não sei" (NULL).

Exercícios Propostos 3

1. Crie uma consulta para recuperar a chave primária, tipo e preço de todas as impressoras com preço nulo ou preço abaixo de US$ 200,00.

2. Modifique a consulta acima para retornar os resultados ordenados por preço (descendente) e tipo (ascendente), excluindo as impressoras com valor nulo no preço.

IV. Operações de Conjunto

Esta seção aborda a utilização dos operadores de UNIÃO, INTERSEÇÃO e DIFERENÇA na Linguagem SQL.

5 – UNION, UNION ALL, INTERSECT e EXCEPT

Exemplo 26: Operador UNION (parte 1)

```
SELECT model, price FROM PC
UNION
SELECT model, price FROM Laptop

model price
1001  2114
1002  995
1003  478
1004  649
1005  630
1006  1049
1007  510
1008  770
1009  650
1010  770
1011  959
1012  649
1013  529
2001  3673
2002  949
2003  549
2004  1150
2005  2500
2006  1700
2007  1429
2008  900
2009  680
2010  2300
```

No exemplo acima, o operador UNION foi utilizado para retornar os modelos e preços dos computadores que estão na tabela *PC* adicionados aos modelos e preços dos computadores da tabela *Laptop*. Assim como ocorre na Álgebra Relacional, a utilização do operador UNION requer que os resultados dos dois (ou mais) SELECTs envolvidos estejam selecionando campos do mesmo tipo (neste caso "model" é do tipo INT e "price" do tipo NUM em ambas as tabelas *PC* e *Laptop*). Diferentemente do que ocorre na Álgebra Relacional, os nomes das colunas podem ser diferentes, contanto que o tipo seja o mesmo (caso isto aconteça, o SGBD atribuirá os nomes das colunas do primeiro SELECT para os nomes das colunas do conjunto de resultados).

É também possível fazer o UNION de mais de duas tabelas, como mostra o exemplo a seguir:

```
SELECT model, price FROM PC
UNION
SELECT model, price FROM Laptop
UNION
SELECT model, price FROM Printer
```

model	price
1001	2114
1002	995
1003	478
1004	649
1005	630
1006	1049
1007	510
1008	770
1009	650
1010	770
1011	959
1012	649
1013	529
2001	3673
2002	949
2003	549
2004	1150
2005	2500
2006	1700
2007	1429
2008	900
2009	680
2010	2300
3001	99
3002	239
3003	899
3004	120
3005	120
3006	100
3007	200
3010	*NULL*

Expressão equivalente em Álgebra Relacional

$\pi_{model,price}(PC) \cup \pi_{model,price}(Laptop) \cup \pi_{model,price}(Printer)$

<u>Exemplo 27</u>: Operador UNION (parte 2)

As cláusulas WHERE e ORDER BY podem ser utilizadas em SELECTs que contém o operador UNION. No entanto existe uma regra: o WHERE deve ser feito para cada tabela e o

ORDER BY apenas uma vez, como última linha do comando (servirá para ordenar os resultados como um todo).

```
SELECT model, price FROM PC WHERE price < 800
UNION
SELECT model, price FROM Laptop WHERE price < 1000
UNION
SELECT model, price FROM Printer WHERE price < 200
ORDER BY price

model price
3001  99
3006  100
3004  120
3005  120
1003  478
1007  510
1013  529
2003  549
1005  630
1004  649
1012  649
1009  650
2009  680
1008  770
1010  770
2008  900
2002  949
```

O exemplo acima recupera todos os PC's com preço inferior a US$ 800,00, todos os laptops com preço inferior a US$ 1000 e todas as impressoras com preço inferior a US$ 200. O resultado final é ordenado por preço.

Exemplo 28: Operador UNION ALL

Por padrão, o conjunto de resultados retornados por um SELECT com UNION removerá tuplas repetidas. Veja o exemplo abaixo:

```
SELECT ram, hd FROM PC
UNION
SELECT ram, hd FROM Laptop
ORDER BY ram, hd

ram  hd
512   60
512   80
512   250
1024  80
1024  100
1024  120
1024  160
1024  200
```

```
1024  250
1024  320
2048  80
2048  160
2048  240
2048  250
2048  300
```

Embora existam 20 registros em *PC* e 13 em *Laptop*, a consulta acima retornou apenas 15 registros ao invés de 23, pois linhas com o mesmo valor de "ram" e "hd" foram suprimidas. Para que as linhas repetidas sejam retornadas, é preciso utilizar o operador UNION ALL.

```
SELECT ram, hd FROM PC
UNION ALL
SELECT ram, hd FROM Laptop
ORDER BY ram, hd

ram  hd
512   60
512   60
512   80
512   80
512   80
512   250
512   250
1024  80
1024  100
1024  120
1024  120
1024  160
1024  200
1024  250
1024  250
1024  250
1024  320
2048  80
2048  160
2048  160
2048  240
2048  250
2048  300
```

UNION ALL e Álgebra Relacional

Na álgebra relacional nenhuma relação pode ter tuplas repetidas. Por este motivo, ela não oferece um operador equivalente ao UNION ALL.

Exemplo 29: Operador INTERSECT

```
SELECT ram, hd FROM PC
INTERSECT
SELECT ram, hd FROM Laptop

ram   hd
512   80
2048  160
```

O operador INTERSECT retorna a **interseção** de dois ou mais conjuntos de resultados. Assim como na união, cada conjunto de resultados é definido por um comando SELECT. Se um registro existe em ambos os conjuntos ele será incluído no resultado final. No exemplo acima, apenas as combinações de valores de "ram" e "hd" comuns às tabelas *PC* e *Laptop* são levadas para o resultado final.

Expressão equivalente em Álgebra Relacional

$\pi_{ram,hd}(PC) \cap \pi_{ram,hd}(Laptop)$

Exemplo 30: Operador EXCEPT

```
SELECT ram, hd FROM PC
EXCEPT
SELECT ram, hd FROM Laptop

ram   hd
512   250
1024  160
1024  200
1024  250
1024  320
2048  250
2048  300
```

O operador EXCEPT é utilizado para retornar todas as linhas do primeiro conjunto de resultados (definido pelo primeiro SELECT) e então remover deste resultado todas as linhas do segundo conjunto de resultados (definido pelo segundo SELECT). Equivale ao operador de **diferença** da Álgebra Relacional. **IMPORTANTE**: em alguns SGBDs, o operador EXCEPT possui outro nome, como por exemplo MINUS.

Expressão equivalente em Álgebra Relacional

$\pi_{ram,hd}(PC) - \pi_{ram,hd}(Laptop)$

Exercícios Propostos 4

- Elabore instruções SELECT com os operadores UNION, UNION ALL, INTERSECT ou EXCEPT para resolver as seguintes consultas:

  1. Encontrar todos os tamanhos de hard disk distintos, considerando as tabelas *Laptop* e *PC*. Eliminar os valores repetidos do resultado final.

  2. Encontrar todos os tamanhos de hard disk comuns às tabelas *Laptop* e *PC*. Ordenar os resultados de forma descendente.

  3. Encontrar todos os fabricantes que produzem laptops, mas não PC's.

V. Combinando Linhas de Duas ou Mais Tabelas

Na maioria das situações práticas, será necessário trabalhar com informações contidas em **mais de uma** tabela de um SGBD relacional. Por exemplo: uma consulta onde deseja-se recuperar todos os fabricantes que vendem laptops com hard disk acima de 100GB. Como a informação sobre o fabricante está na tabela *Product* e o tamnho do hard disk na tabela *Laptop*, é preciso elaborar um comando SELECT que realize o **casamento de linhas** de *Product* com as linhas de *Laptop*.

Conforme apresentado na aula 08, existem dois tipos de operações que combinam linhas de tabelas:

- **Produto Cartesiano**: combina as linhas de duas tabelas de todas as formas possíveis.
- **Junções**: combinam de forma seletiva as linhas de duas tabelas. A **junção natural** realiza a combinação baseada em atributos que são comuns às duas tabelas, enquanto a **junção theta** realiza a combinação baseado em qualquer outro tipo de critério.

O objetivo principal desta seção é mostrar como criar consultas que realizam as operações acima na linguagem SQL. Em uma aula posterior voltaremos a ao tema "junção de tabelas" com o objetivo de examinar técnicas de junção mais popularmente utilizadas na SQL , como a junção interna (INNER JOIN) e a junção externa (OUTER JOIN).

6 – Produto Cartesiano

Exemplo 31: Produto Cartesiano

O produto cartesiano é a operação que combina todos os registros de uma tabela T1 com todos de outra tabela T2 (ver páginas 6 a 9 da aula08).

```
SELECT a.*, b.*
FROM Product a, PC b

maker model type        model:1 speed ram  hd   price
A     1001  pc          1001    2.66  1024 250  2114
A     1001  pc          1002    2.1   512  250  995
A     1001  pc          1003    1.42  512  80   478
...
H     3010  printer     1013    3.06  512  80   529
```

Em SQL, um produto cartesiano entre duas ou mais tabelas é definido pela especificação **do nome das tabelas separados por vírgula na cláusula FROM**. O exemplo acima ilustra o produto cartesiano entre as tabelas *Product* e *PC*. O resultado final combina todas as linhas de *Product* com todas as linhas de *PC*. Ou seja: todas as 31 linhas[4] contidas em *Product* foram combinadas com todas as 13 linhas de *PC* gerando um resultado final formado

[4]No exemplo foi considerada a linha inserida em *Product* na página 28.

por 31 x 13 = 403 linhas. O esquema (definição) do conjunto resultante é igual à união dos esquemas de *Product* e *PC*. Ou seja: número de colunas resultante é igual a soma do número de colunas de cada tabela (3 + 5 = 8).

Observe que em nosso exemplo a tabela *Product* foi apelidada como "a" e *PC* apelidada como "b". Desta forma, torna-se possível referenciar qualquer coluna de *Product* pelo prefixo "a." (ex: a.maker); analogamente, qualquer coluna de *PC* pode ser referenciada como "b." (ex: b.price). No exemplo acima, utilizou-se `SELECT a.*, b.*`, para recuperar todas as colunas de *Product* e *PC*. Como uma coluna de nome "model" existe nas duas tabelas (coluna comum aos dois esquemas), o SQLite exibiu o rótulo "model:1" para identificar a coluna "model" da tabela *PC* (na prática, cada SGBD utiliza uma forma diferente para remover a ambiguidade de nomes).

Expressão equivalente em Álgebra Relacional

$Product \times PC$

Embora o produto cartesiano represente um recurso teórico interessante para resolver alguns problemas de Álgebra Relacional, ele é raramente útil em situações práticas. Isto se deve principalmente ao fato de que quando as tabelas envolvidas possuem muitos registros (ex: milhões), o resultado do produto cartesiano pode "lotar" o *buffer* (área de trabalho) do SGBD, degradando o desempenho do sistema.

7 – Junção Natural

Assim como o produto cartesiano, a junção natural também serve para combinar linhas de duas tabelas T1 e T2. Porém, a junção natural faz um "casamento inteligente", combinando apenas as linhas de T1 e T2 que **coincidem** em **quaisquer atributos que são comuns** aos esquemas de T1 e T2 (ver páginas 10 a 18 da aula08).

Exemplo 32: Junção Natural

```
SELECT a.maker, a.model, a.type, b.speed, b.ram, b.hd, b.price
FROM Product a NATURAL JOIN PC b
```

maker	model	type	speed	ram	hd	price
A	1001	pc	2.66	1024	250	2114
A	1002	pc	2.1	512	250	995
A	1003	pc	1.42	512	80	478
B	1004	pc	2.8	1024	250	649
B	1005	pc	3.2	512	250	630
B	1006	pc	3.2	1024	320	1049
C	1007	pc	2.2	1024	200	510
D	1008	pc	2.2	2048	250	770
D	1009	pc	2	1024	250	650
D	1010	pc	2.8	2048	300	770
E	1011	pc	1.86	2048	160	959

```
E       1012    pc      2.8     1024    160     649
E       1013    pc      3.06    512     80      529
```

O comando SQL acima utiliza a cláusula `NATURAL JOIN` para implementar a **junção natural** entre as tabelas *Product* e *PC*. Ela recupera o fabricante ("maker"), modelo ("model"), tipo de produto ("type"), velocidade ("speed"), memória RAM ("ram"), hard disk ("hd") e preço ("price") de todos os PC's da base de dados. As colunas "maker", "model" e "type" são recuperadas da tabela *Product* e as demais colunas de *PC*. Ao contrário do que ocorre no produto cartesiano, uma linha de *Product* só será "casada" com uma linha de *PC* caso elas coincidam no valor do atributo em comum (neste caso, "model", que é chave primária em *Product* e chave estrangeira em *PC*). A resposta da consulta retorna 13 linhas.

Expressão equivalente em Álgebra Relacional

Product |X| *PC*

IMPORTANTE: Se você modificar a consulta do Exemplo 32 para **`SELECT * FROM Product NATURAL JOIN PC`**, poderá observar que o **SQLite irá retornar o campo "model" duas vezes** no conjunto de resultados (como "model" e "model:1"). Esse comportamento difere do comportamento da Álgebra Relacional, onde a relação resultante da operação *Product* |X| *PC* conteria apenas um atributo "model".

<u>Exemplo 33</u>: Junção Natural – uso com WHERE e ORDER BY (parte 1)

```
SELECT a.model, b.price
FROM Product a NATURAL JOIN PC b
WHERE a.maker IN ('E')
ORDER BY b.price;


model  price
1013   529
1012   649
1011   959
```

As cláusulas WHERE e ORDER BY podem ser utilizadas normalmente em instruções SQL que possuem produto cartesiano ou junção de tabelas (seja junção natural ou qualquer outro tipo). No exemplo acima, a consulta anterior foi modificada para listar apenas o modelo e preço dos PC's produzidos pelo fabricante 'E' (cláusula WHERE) e ordenar os resultados de forma ascendente pelo preço do produto (cláusula ORDER BY). As cláusula WHERE e ORDER BY podem ser utilizadas com colunas de ambas as tabelas, sem qualquer tipo de limitação.

Exemplo 34: Junção Natural – uso com WHERE e ORDER BY (parte 2)

```
SELECT DISTINCT a.maker, b.screen
FROM Product a NATURAL JOIN Laptop b
WHERE b.price < 2000
ORDER BY b.screen DESC
```

maker	screen
E	17
E	15.4
A	15.4
F	15.4
F	14.1
A	13.3
B	13.3

Este exemplo envolve uma operação de junção entre as tabelas *Product* e *Laptop*. Para cada fabricante que produz um laptop, apresenta-se os diferentes tamanhos de tela disponíveis em seus modelos de laptop com preço inferior a US$ 2000. Os resultados são ordenados pelo tamanho da tela em ordem decrescente.

IMPORTANTE: observe que nesse exemplo, o campo "model" - justamente o campo que liga as duas tabelas - deixou de ser selecionado pela consulta. Isto foi feito de forma proposital, para demonstrar que não há obrigatoriedade em selecionar a coluna responsável pela junção.

Exemplo 35: Junção Natural com Subconsulta

```
SELECT a.model, a.maker, a.type, b.price
FROM Product a NATURAL JOIN (
      SELECT model, price FROM PC
      UNION
      SELECT model, price FROM Laptop
      UNION
      SELECT model, price FROM Printer
      ) b
```

model	maker	type	price
1001	A	pc	2114
1002	A	pc	995
1003	A	pc	478
1004	B	pc	649
1005	B	pc	630
1006	B	pc	1049
1007	C	pc	510
1008	D	pc	770

```
1009  D     pc          650
1010  D     pc          770
1011  E     pc          959
1012  E     pc          649
1013  E     pc          529
2001  E     laptop      3673
2002  E     laptop      949
2003  E     laptop      549
2004  A     laptop      1150
2005  A     laptop      2500
2006  A     laptop      1700
2007  B     pc          1429
2008  F     laptop      900
2009  F     laptop      680
2010  G     laptop      2300
3001  E     printer     99
3002  E     printer     239
3003  E     printer     899
3004  D     printer     120
3005  D     printer     120
3006  H     printer     100
3007  H     printer     200
3010  H     printer     NULL
```

A consulta acima retorna o número do modelo, fabricante, tipo de produto e preço de todos os produtos. Ela demonstra um dos mais poderosos recursos da SQL (também presente na Álgebra Relacional): a elaboração de instruções SELECT de complexidade arbitrária através da **aplicação de consultas sobre resultados obtidos por outras consultas**. Neste caso, parênteses são utilizados para indicar as **subconsultas** (consultas internas) que serão executadas inicialmente e terão os seus resultados "consumidos" pelas consultas externas (responsáveis por prover o conjunto de resultados final). No exemplo acima, a consulta interna é:

```
(
SELECT model, price FROM PC
UNION
SELECT model, price FROM Laptop
UNION
SELECT model, price FROM Printer
) b
```

Ela gera como resultado uma relação (resultado intermediário) contendo os valores das colunas "model" e "price" das tabelas *PC*, *Laptop* e *Printer*. Este resultado intermediário é apelidado de "b".

A consulta externa realiza a junção natural de *Product* com o resultado da consulta interna, retornando uma nova relação (resultado final) que recupera o modelo, fabricante, tipo de produto e preço de todos os produtos cadastrados no BD.

Expressão equivalente em Álgebra Relacional

$b := \pi_{model,price}(PC) \cup \pi_{model,price}(Laptop) \cup \pi_{model,price}(Printer)$

$Resposta := \pi_{model,maker,type,price}(Product \ |X| \ b)$

IMPORTANTE: diferentemente da Álgebra Relacional, a SQL **não** permite a resolução de uma consulta utilizando uma sequência de operações. As consultas SQL precisam sempre ser elaboradas como expressão única, mesmo que possuam uma ou mais subconsultas.

8 – Junção Theta

Este tipo de junção permite combinar registros de duas tabelas utilizando uma condição arbitrária (ver páginas 19 a 28 da aula 07).

Exemplo 36: Junção Theta

O exemplo abaixo mostra o SQL que identifica os tamanhos de hard disk que ocorrem em dois ou mais PC's.

```
SELECT DISTINCT a.hd
FROM PC a, PC b
WHERE a.model < b.model AND  a.hd = b.hd

hd
250
80
160
```

O resultado de uma operação de junção theta entre duas tabelas é construído em duas etapas: (i) realizar o produto cartesiano entre as tabelas; e (ii) selecionar apenas as tuplas que satisfaçam à condição especificada. O exemplo acima realiza o produto cartesiano da tabela *PC* com ela mesma no trecho:

```
FROM PC a, PC b
```

Observe, que foi necessário apelidar uma instância de *PC* como "a" e a outra como "b" para viabilizar o produto cartesiano[5].

[5] Os apelidos não precisam ser necessariamente "a" e "b". Poderiam ser "x", "y", "t1", "t2", "aux", etc.

Em seguida, a condição:

```
WHERE a.model < b.model AND  a.hd = b.hd
```

É utilizada para selecionar apenas os registros em que os modelos são diferentes e os tamanhos de hard disk são iguais (qualquer tupla que não obedeça a essa condição é eliminada).

Expressão equivalente em Álgebra Relacional

a := *PC*
b := *PC*
Resposta := $\pi_{a.hd}$(a |X|$_{a.model < b.model\ AND\ a.hd = b.hd}$ *b*)

Exercícios Propostos 5

1. Crie uma consulta para recuperar os fabricantes que produzem impressoras coloridas. CUIDADO: as tabelas *Product* e *Printer* possuem dois atributos com nomes iguais: "model" e "type".

2. Modifique a consulta acima da seguinte forma: retornar o fabricante, número do modelo e preço de todas as impressoras coloridas.

3. Crie uma consulta para recuperar todos os dados (modelo, velocidade, ram, hd e preço) dos PC's que não são produzidos pelos fabricantes A e B. Ordene o resultado por velocidade de modo descendente (primeiro critério) e modelo em ordem ascendente (segundo critério).

4. Encontre os fabricantes e modelos de computadores (PC's ou laptops) com velocidade igual ou superior a 2.00 giga-hertz.

5. Encontre o preço do laptop mais barato.

6. Encontre todas as combinações diferentes de pares de modelos de PC. Um par deve ser listado apenas uma vez; ex: listar (*i*,*j*) mas não (*j*,*i*).

7. Modifique o exercício acima para exibir apenas os pares formados por dois modelos que custem menos de US$ 600,00.