Universidade Presbiteriana Mackenzie

Banco de Dados – Aula 14

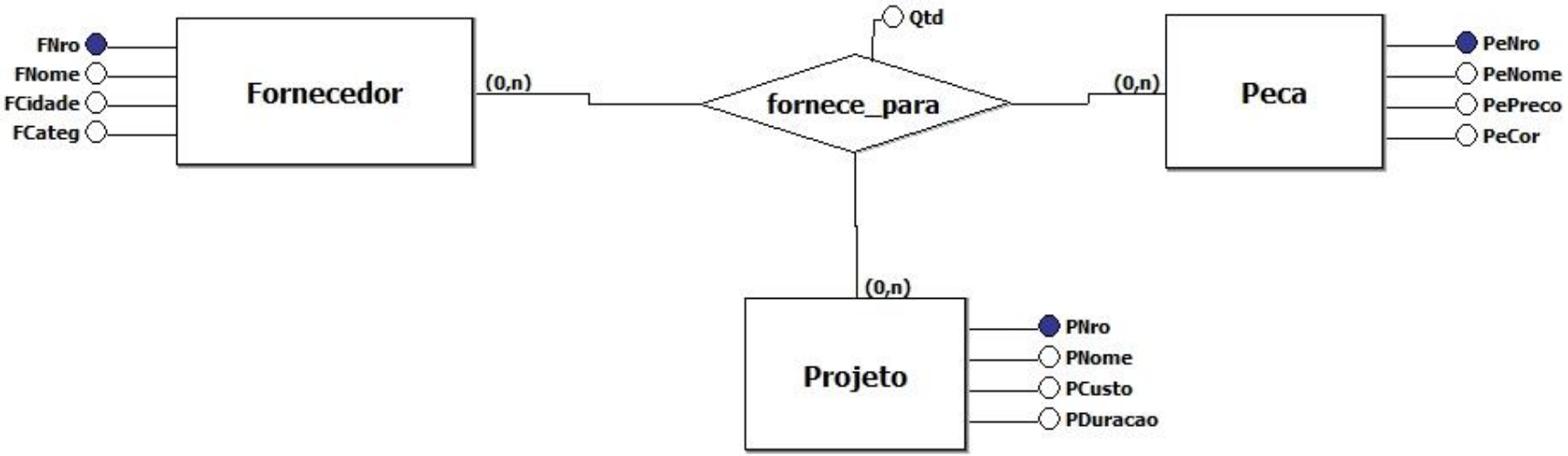
Linguagem SQL - SELECT com GROUP BY e HAVING

Profa. Elisângela Botelho Gracias

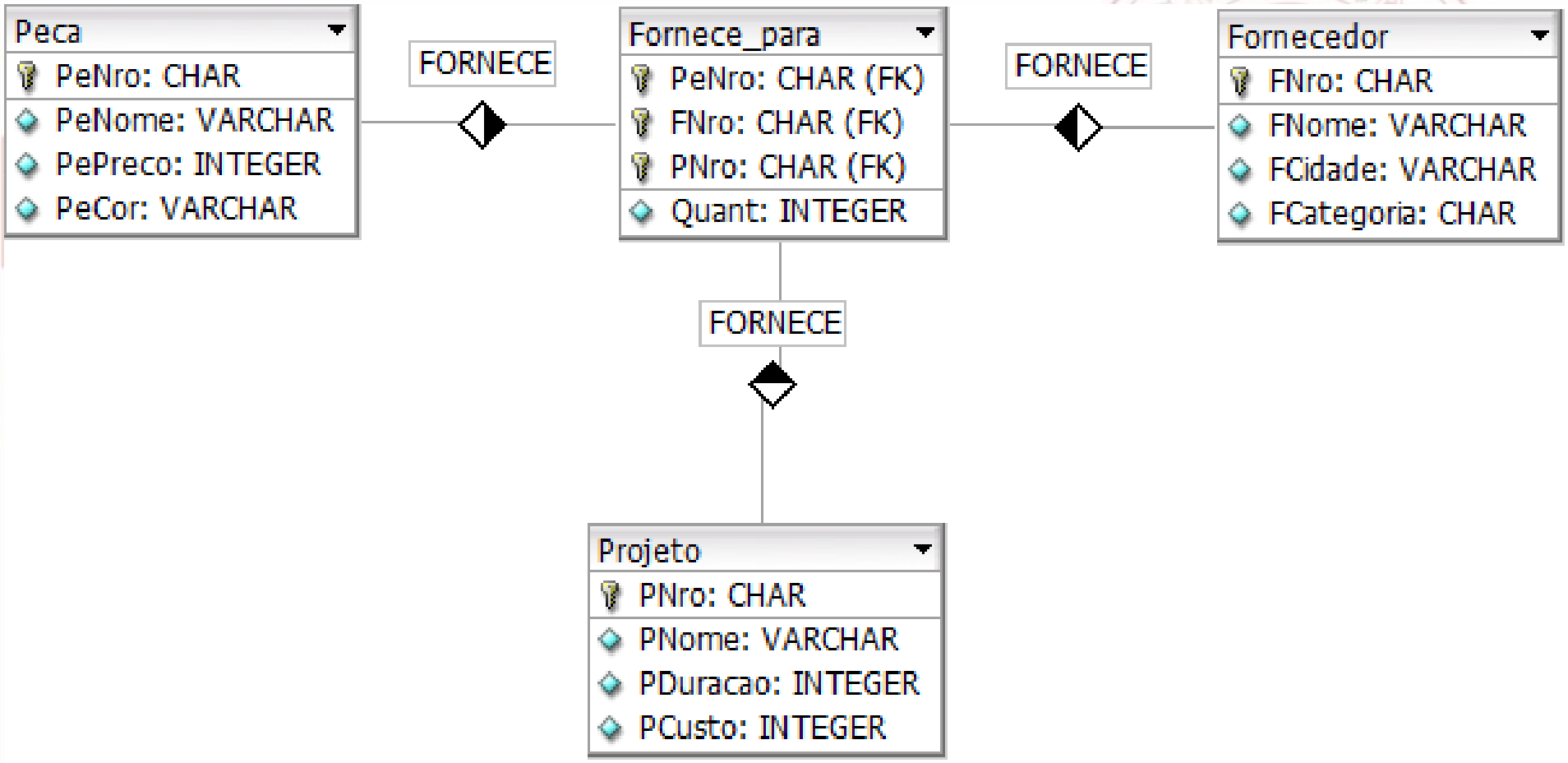
Faculdade de Computação e Informática

Banco de Dados Exemplo

Modelo Entidade-Relacionamento



Modelo Relacional



Exemplo de Banco de Dados

Considere o seguinte modelo relacional (chaves primárias estão sublinhadas:

PECA = {PeNro, PeNome, PePreco, PeCor}

FORNECEDOR = {FNro, FNome, FCidade, FCateg}

PROJETO = {PNro, PNome, PDuracao, PCusto}

FORNECE_PARA = {PeNro, FNro, PNro, Quant}

- **PeNro é chave estrangeira que referencia a tabela Peca**
- **FNro é chave estrangeira que referencia a tabela Fornecedor**
- **PNro é chave estrangeira que referencia a tabela Projeto**

5

Peca

PeNro	PeNome	PePreço	PeCor
PE1	Cinto	22	Azul
PE2	Volante	18	Vermelho
PE3	Lanterna	14	Preto
PE4	Limpador	09	Amarelo
PE5	Painel	43	Vermelho

Fornecedor

FNro	FNome	FCidade	FCateg
F1	Plastec	Campinas	B
F2	CM	São Paulo	D
F3	Kirurgic	Campinas	A
F4	Piloto	Piracicaba	A
F5	Equipament	São Carlos	C

Projeto

PNro	PNome	PDuração	PCusto
P1	Detroit	5	43000
P2	Pegasus	3	37000
P3	Alfa	2	26700
P4	Sea	3	21200
P5	Paraiso	1	17000

Fornece_para

PeNro	FNro	PNro	Quant
PE1	F5	P4	5
PE2	F2	P2	1
PE3	F3	P4	2
PE4	F4	P5	3
PE5	F1	P1	1
PE2	F2	P3	1
PE4	F3	P5	2

--**Script de criação do Banco de Dados:**

```
DROP TABLE Fornece_Para CASCADE CONSTRAINT;
DROP TABLE Projeto CASCADE CONSTRAINT;
DROP TABLE Fornecedor CASCADE CONSTRAINT;
DROP TABLE Peca CASCADE CONSTRAINT;

CREATE TABLE Peca (
PeNro CHAR(4),
PeNome VARCHAR(30) NOT NULL,
PePreco INTEGER NOT NULL,
PeCor VARCHAR(20) NOT NULL,
PRIMARY KEY(PeNro));

CREATE TABLE Fornecedor (
FNro CHAR(4),
FNome VARCHAR(30) NOT NULL,
FCidade VARCHAR(30) NOT NULL,
FCategoria CHAR(1) NOT NULL,
PRIMARY KEY(FNro));

CREATE TABLE Projeto (
PNro CHAR(4),
PNome VARCHAR(30) NOT NULL,
PDuracao INTEGER NOT NULL,
PCusto INTEGER NOT NULL,
PRIMARY KEY(PNro));

CREATE TABLE Fornece_para (
PeNro CHAR(4),
FNro CHAR(4),
PNro CHAR(4),
Quant INTEGER,
PRIMARY KEY(PeNro,FNro,PNro),
FOREIGN KEY(PeNro) REFERENCES Peca(PeNro),
FOREIGN KEY(FNro) REFERENCES Fornecedor(FNro),
FOREIGN KEY(PNro) REFERENCES Projeto(PNro));
```

```
INSERT INTO Peca VALUES ('PE1', 'Cinto', 22, 'Azul');
INSERT INTO Peca VALUES ('PE2', 'Volante', 18, 'Vermelho');
INSERT INTO Peca VALUES ('PE3', 'Lanterna', 14, 'Preto');
INSERT INTO Peca VALUES ('PE4', 'Limpador', 9, 'Amarelo');
INSERT INTO Peca VALUES ('PE5', 'Painel', 43, 'Vermelho');

INSERT INTO Fornecedor VALUES ('F1', 'Plastec', 'Campinas', 'B');
INSERT INTO Fornecedor VALUES ('F2', 'CM', 'Sao Paulo', 'D');
INSERT INTO Fornecedor VALUES ('F3', 'Kirurgic', 'Campinas', 'A');
INSERT INTO Fornecedor VALUES ('F4', 'Piloto', 'Piracicaba', 'A');
INSERT INTO Fornecedor VALUES ('F5', 'Equipament', 'Sao Carlos', 'C');

INSERT INTO Projeto VALUES ('P1', 'Detroit', 5, 43000);
INSERT INTO Projeto VALUES ('P2', 'Pegasus', 3, 37000);
INSERT INTO Projeto VALUES ('P3', 'Alfa', 2, 26700);
INSERT INTO Projeto VALUES ('P4', 'Sea', 3, 21200);
INSERT INTO Projeto VALUES ('P5', 'Paraiso', 1, 17000);

INSERT INTO Fornece_para VALUES ('PE1', 'F5', 'P4', 5);
INSERT INTO Fornece_para VALUES ('PE2', 'F2', 'P2', 1);
INSERT INTO Fornece_para VALUES ('PE3', 'F3', 'P4', 2);
INSERT INTO Fornece_para VALUES ('PE4', 'F4', 'P5', 3);
INSERT INTO Fornece_para VALUES ('PE5', 'F1', 'P1', 1);
INSERT INTO Fornece_para VALUES ('PE2', 'F2', 'P3', 1);
INSERT INTO Fornece_para VALUES ('PE4', 'F3', 'P5', 2);
COMMIT;
```

SELECT

- SELECT
  - possibilita a consulta de uma ou mais tabelas de acordo com os critérios estabelecidos e com as necessidades

SELECT

- Sintaxe do comando SELECT

SELECT [DISTINCT] nome_atributo1,... nome_atributoN

FROM nome_tabela1, ... nome_tabelaN

[WHERE (condições**)]**

[GROUP BY nome_atributo1,... nome_atributoN**]**

[HAVING (condições**)]**

[ORDER BY nome_atributo1 **{ASC | DESC}, ...**

nome_atributoN **{ASC | DESC}] ;**

Obs: *tudo que está entre [ ] é opcional, mas se for utilizar tire o [ ]*

SELECT

- Onde:
  - SELECT: o que se deseja no resultado da consulta
  - DISTINCT: não permite repetição de valores no resultado
  - FROM: de onde buscar os dados necessários
  - WHERE: condições para busca dos resultados

SELECT

- Onde (continuação):
  - GROUP BY: agrupamento de dados
  - HAVING: condições para a definição de grupos no resultado
  - ORDER BY: estabelece a ordenação lógica do resultado

Linguagem SQL

Comando SELECT com utilização de GROUP BY e HAVING

SELECT

- Quando deseja-se **aplicar as funções agregadas a vários grupos** de uma tabela (ou várias), deve-se utilizar o agrupamento – **GROUP BY**
- Neste caso, é necessário **particionar a tabela em grupos que possuem o mesmo valor de atributo**

SELECT

- A cláusula **GROUP BY especifica o(s) atributo(s) de agrupamento**
- Se uma consulta contiver o GROUP BY, então todos os atributos no SELECT também **devem aparecer no GROUP BY**
- Para **cada grupo** deve-se especificar a **função agregada** (ou as funções agregadas) desejada

SELECT

- Por exemplo, para saber a média de idade dos alunos de cada curso da universidade, deve-se utilizar o **GROUP BY**, sendo que o **grupo é o atributo curso** e a informação de **cada curso** é a **média de idade** dos alunos
- Neste caso, para cada **valor do atributo curso** é criado **um grupo**, e sobre **cada grupo é calculada a média de idade**

SELECT

- É possível saber, também, para cada curso, além da média de idade, a **quantidade de alunos** de cada curso, a **maior idade** de cada curso, a **menor idade** de cada curso

SELECT

- Se for necessário **impor condições ao grupos criados**, deve-se utilizar a cláusula **HAVING**, que é a condição para que um grupo apareça no resultado do SELECT
- Por exemplo, se deseja retornar, para cada curso da universidade, a média de idade dos alunos deste curso, mas somente se a **média de idade do curso for superior a 21 anos**, deve-se utilizar a cláusula **HAVING**

SELECT

- A cláusula **HAVING** é a condição de um agrupamento, portanto, ela só existe se a consulta tiver a cláusula **GROUP BY**

- Lembre-se que a cláusula **WHERE** de uma consulta é **condição de cada linha da tabela** e **NÃO do agrupamento**, portanto, em uma consulta com agrupamento é possível ter condições de cada linha da tabela – **WHERE** – e condições de um agrupamento - **HAVING**

SELECT

- **Ou seja:**
  - na cláusula **WHERE não** pode-se utilizar **funções agregadas**, pois ela só pode ser aplicada a linhas individuais
  - enquanto a cláusula **HAVING** só pode ser aplicada a **condições um agrupamento**.

SELECT

- Poderia-se obter o total de peças utilizadas no total, com a seguinte consulta:

```
SELECT SUM(Quant) AS Soma
FROM Fornece_para;
```

- Agora, se deseja-se ter a **quantidade total de cada peça** utilizada nos projetos, deve-se utilizar o **GROUP BY** (que é o próximo exemplo)

SELECT

- **Exemplo1** (GROUP BY): Obtenha o número de cada peça e a quantidade total de cada peça utilizada em todos os projetos, em ordem crescente do número da peça.

```
SELECT PeNro, SUM(Quant) AS Soma

FROM Fornece_para

GROUP BY PeNro

ORDER BY PeNro ASC;
```

PeNro	Soma
PE1	5
PE2	2
PE3	2
PE4	5
PE5	1

SELECT

- **Exemplo2** (GROUP BY): Obtenha o número de cada fornecedor, em ordem crescente, obtendo a quantidade total de peças fornecidas por cada um deles nos diversos projetos

```
SELECT FNro, SUM(Quant) AS Soma

FROM Fornece_para

GROUP BY FNro

ORDER BY FNro ASC ;
```

FNro	Soma
F1	1
F2	2
F3	4
F4	3
F5	5

SELECT

- **Exemplo3** (GROUP BY e HAVING): Obtenha o número de cada peça e a quantidade total de cada peça utilizada em todos os projetos, mas desde que esse total seja menor que 3. Retorne primeiro em ordem decrescente deste total e, depois, em ordem crescente do número da peça.

```
SELECT PeNro, SUM(Quant) AS Soma

FROM Fornece_para

GROUP BY PeNro

HAVING SUM(Quant) < 3

ORDER BY SUM(Quant) DESC, PeNro ASC;
```

PeNro	Soma
PE2	2
PE3	2
PE5	1

SELECT

- **Exemplo3** (GROUP BY e HAVING): Obtenha o número de cada peça e a quantidade total de cada peça utilizada em todos os projetos, mas desde que esse total seja menor que 3. Retorne primeiro em ordem decrescente deste total e, depois, em ordem crescente do número da peça.

```
SELECT PeNro, SUM(Quant)
FROM Fornece_para
GROUP BY PeNro
HAVING SUM(Quant) < 3
ORDER BY SUM(Quant) DESC, PeNro ASC;
```

Observe que é possível ORDENAR o resultado de uma consulta com MAIS de UMA informação e/ou atributo

SELECT

- Observe que no Exemplo3 só foram retornadas as peças **PE2**, **PE3** e **PE5**, pois a quantidade total de cada peça (utilizada em todos os projetos) **atendeu à condição da cláusula HAVING**
- As peças **PE1 e PE4 não apareceram no resultado**, pois elas foram utilizadas em uma quantidade maior ou igual a 3, **não atendendo à condição da cláusula HAVING**

SELECT

- **Exemplo4** (GROUP BY): Obtenha, para cada cor, a quantidade de peças, em ordem decrescente desta quantidade e, depois, em ordem crescente da cor.

```
SELECT PeCor, COUNT(PeCor) AS Total

FROM Peca

GROUP BY PeCor

ORDER BY COUNT(PeCor) DESC, PeCor ASC;
```

PeCor	Total
Vermelho	2
Amarelo	1
Azul	1
Preto	1

SELECT

- **Exemplo5** (WHERE e HAVING): Obtenha a quantidade de fornecedores de cada cidade cujo nome da cidade se inicie com a letra C, mas somente para aquelas cidades que, além de iniciarem com a letra C, possuirem mais de um fornecedor.

```
SELECT FCidade, COUNT(FCidade) AS Total
FROM Fornecedor
WHERE (FCidade LIKE 'C%')
GROUP BY FCidade
HAVING COUNT(FCidade) > 1;
```

FCidade	Total
Campinas	2

SELECT

- Observe que no Exemplo5 foram utilizadas as cláusulas **WHERE** e **HAVING**
- A cláusula **WHERE**, que é **condição de cada linha da tabela**, selecionou somente as linhas cuja cidade se iniciava com a letra C
- Depois desta seleção é que foi realizado o agrupamento, mas obedecendo a condição da cláusula **HAVING**

SELECT

- **Exemplo6** (GROUP BY): Obtenha somente o nome das cidades que tem somente um único fornecedor, em ordem crescente do nome da cidade.

```
SELECT FCidade
FROM Fornecedor
GROUP BY FCidade
HAVING COUNT(FCidade) = 1
ORDER BY FCidade ASC ;
```

FCidade
Piracicaba
São Carlos
São Paulo

SELECT

- Observe que no Exemplo6 a função agregada referente ao grupo – que é o atributo FCidade – NÃO apareceu no SELECT, pois não foi pedido para aparecer.
- Mas a função agregada apareceu na cláusula HAVING, de acordo com o que foi pedido.

SELECT

- **Exemplo7** (GROUP BY com 2 atributos): Obtenha a quantidade de fornecedores de **cada peça em cada projeto**, em ordem decrescente desta quantidade e, depois, em ordem crescente do projeto.

```
SELECT PeNro, PNro, COUNT(FNro) AS Total
FROM Fornece_para
GROUP BY PeNro, PNro
ORDER BY COUNT(FNro) DESC,
         PNro ASC;
```

PeNro	PNro	Total
PE4	P5	2
PE5	P1	1
PE2	P2	1
PE2	P3	1
PE1	P4	1
PE3	P4	1

SELECT

- **Exemplo7** (GROUP BY com 2 atributos): Obtenha a quantidade de fornecedores de em ordem decrescente desta ordem crescente do projeto.

```
SELECT PeNro, PNro, COUNT
FROM Fornece_para
GROUP BY PeNro, PNro
ORDER BY COUNT(FNro) DESC,
         PNro ASC;
```

Observe que o agrupamento foi feito utilizando DOIS atributos

PE5	P1	1
PE2	P2	1
PE2	P3	1
PE1	P4	1
PE3	P4	1

SELECT

- Observe que no Exemplo7 foram utilizados dois atributos no agrupamento – **PeNro e PNro**
- No resultado dessa consulta, pode-se observar que:
  - somente a peça PE4 no projeto P5 teve dois fornecedores diferentes
  - a peça PE2 apareceu em duas linhas do resultado, pois ela foi utilizada em dois projetos diferentes

Obrigado

Elisângela Botelho Gracias
elisangela.botelho@mackenzie.br