

Bases de Dados

T14 - SQL Avançado Parte I

Prof. Daniel Faria

Prof. Flávio Martins

Sumário

- Recapitulação Breve
- SQL Avançado
 - Agregação Simples
 - Agregação com Agrupamento
 - Selects Encadeados

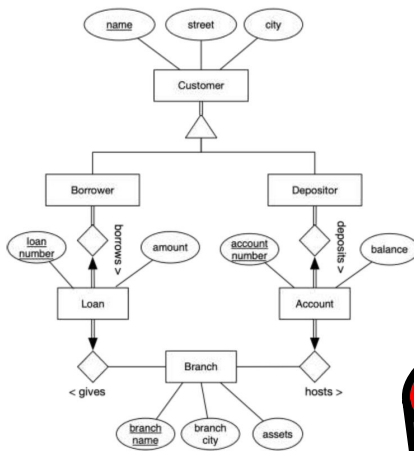
Recapitulação Breve

Concepção de Bases de Dados

Especificação de Requisitos

- requisito funcional 1:
- requisito funcional 2:
- ...
- restrição de integridade 1
- restrição de integridade 2
- ...

Modelo Conceptual



Modelo Relacional

R1 (x, y)
R2 (x, z)

Esquema Relacional (SQL)

branch			account			depositor	
branch_name	branch_city	assets	account_number	branch_name	balance	customer_name	account_number
Brighton	Brooklyn	710000	A-101	Downtown	500	Hayes	A-102
Downtown	Brooklyn	900000	A-102	Perryridge	400	Johnson	A-101
Mannus	Horseneck	400000	A-201	Brighton	900	Jones	A-201
North Town	Rye	370000	A-215	Mannus	700	Lindsay	A-222
Perryridge	Horseneck	1700000	A-217	Brighton	750	Smith	A-215
Normal	Horseneck	300000	A-322	Redwood	798	Turner	A-305
Redwood	Palo Alto	2100000	A-305	Redwood	350		
Round Hill	Horseneck	800000					

loan			borrower			customer		
loan_number	branch_name	amount	customer_name	loan_number		customer_name	customer_city	customer_zip
L-11	Round Hill	900	Adams	L-16	Curry	Spring	Pittsfield	
L-14	Downtown	1500	Hayes	L-15	Hayes	Senator	Brooklyn	
L-15	Perryridge	1500	Jackson	L-14	Glenn	North	Rye	
L-16	Perryridge	1500	Jones	L-17	Glenn	Sand Hill	Woodside	
L-17	Downtown	1000	Smith	L-11	Hayes	Walnut	Stanford	
L-23	Redwood	2000	Smith	L-23	Johnson	Main	Harrison	
L-25	Mannus	500	Williams	L-17	Jones	Main	Harrison	
					Lindsay	Park	Pittsfield	
					Smith	North	Rye	
					Turner	Putnam	Stanford	
					Williams	Nassau	Princeton	

Normalização

Teoria da Normalização

Conceitos Fundamentais:

- Dependências Funcionais: $X \rightarrow Y$



- Superchaves e Chaves Candidatas



- Formas Normais: 1FN, 2FN, 3FN e FNBC



- Decomposição de Relações (sem perda de informação ou de dependências funcionais)

Interrogação de Bases de Dados

SQL ← Álgebra Relacional

```
[WITH with_query [, ...]]
```

```
SELECT [ALL | DISTINCT [ON (expression [, ...])]]
```

```
  [* | expression [[AS] output_name] [, ...]]
```

```
  [FROM from_item [, ...]]
```

```
  [WHERE condition]
```

```
  [GROUP BY [ALL | DISTINCT] grouping_element [, ...]]
```

```
  [HAVING condition]
```

```
  [{UNION | INTERSECT | EXCEPT} [ALL | DISTINCT] select]
```

```
  [ORDER BY expression [ASC | DESC | USING operator]
```

```
    [NULLS { FIRST | LAST}] [, ...]]
```

```
  [LIMIT {count | ALL}]
```

Project, Rename, Aggregation

Cartesian Product, Joins

Select

Aggregation w/ Grouping

Select (after Aggregation)

Union, Intersection, Difference

Base de Dados Exemplo

pig(id, name, DOB, job)

nonpig(id, name, DOB, species)

RI-1: id cannot be in pig

RI-2: species must be 'chicken', 'cow', 'goat' or 'sheep'

Produce(code, producer, date, amount, type)

producer: FK(nonpig.id)

UNIQUE(producer, date)

RI-3: type must be 'eggs' or 'milk'

merchant(SSN)

buys(SSN, code, seller, price)

SSN: FK(merchant)

code: FK(produce)

seller: FK(pig.id)

Agregação Simples

Agregação Simples

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

$G_{\text{COUNT}()}(\text{buys})$

+-----+
COUNT
+-----+
7
+-----+

```
SELECT SUM(price), AVG(price) FROM buys;
```

$G_{\text{SUM}(\text{price}),\text{AVG}(\text{price})}(\text{buys})$

+-----+-----+
SUM AVG
+-----+-----+
430.00 61.43
+-----+-----+

- Funções que agregam valores para toda a tabela, devolvendo uma única linha

<https://www.postgresql.org/docs/current/functions-aggregate.html>

Agregação Simples: Exercícios

- Quantos porcos existem?

Agregação Simples: Exercícios

- Quantos porcos existem?

$G_{\text{COUNT}()}(\text{pig})$

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

Agregação Simples: Exercícios

- Quantos animais produziram algum produto?

Agregação Simples: Exercícios

- Quantos animais produziram algum produto?

$$G_{\text{COUNT}()}(\prod_{\text{producer}}(\text{produce})) = G_{\text{COUNT-DISTINCT}(\text{producer})}(\text{produce})$$

```
SELECT
    COUNT(DISTINCT producer)
FROM
    produce;
```

- Sem o DISTINCT não teríamos a contagem certa!

Agregação Simples: Exercícios

- Quantos animais existem (porcos e não-porcos)?

Agregação Simples: Exercícios

- Quantos animais existem (porcos e não-porcos)?

mais intuitiva: $G_{\text{COUNT}()}(\Pi_{id}(pig) \cup \Pi_{id}(nonpig))$

mais simples: $G_{\text{COUNT}()}(pig \bowtie nonpig)$

```
SELECT
    COUNT(*)
FROM
    pig FULL JOIN nonpig USING (id);
```

Agregação Simples: Exercícios

- Qual foi o volume de vendas de leite de vaca em 2022?

Agregação Simples: Exercícios

- Qual foi o volume de vendas de leite de vaca em 2022?

$G_{SUM(price)}(\sigma_{species='cow' \wedge date > '12-31-2021' \wedge date < '01-01-2023'}(buys \bowtie_{producer=id} produce \bowtie nonpig))$

```
SELECT
    SUM(price)
FROM
    buys JOIN produce USING (code) JOIN nonpig ON (producer=id)
WHERE
    species='cow' AND EXTRACT(YEAR FROM date)=2022;
```

Agregação com Agrupamento

Agregação com Agrupamento

seller $\overset{G}{\text{SUM}}(\text{price}) \rightarrow \text{total}$ (*buys*)

```
SELECT seller, SUM(price) AS total
FROM buys
GROUP BY seller;
```

+	-----+	-----+
	seller	total
+	-----+	-----+
	101001	1650.00
	100100	700.00
	110011	350.00
+	-----+	-----+

- Cláusula **GROUP BY** permite fazer agregação com agrupamento
- O output é sempre uma linha por cada instância do grupo

Agregação com Agrupamento

- **GROUP BY**

- Necessário quando se retorna colunas agregadas e não agregadas, mas pode ser usado mesmo quando se retorna apenas colunas agregadas ou apenas não agregadas
- Pode agrupar-se por várias colunas (separadas por vírgulas)
 - Nesse caso o agrupamento será para combinações de valores diferentes das várias colunas (tal como em Álgebra Relacional)
- Não faz sentido agrupar por chaves candidatas: não haverá agregação!

Agregação com Agrupamento

- **GROUP BY**

- Deve-se quase sempre retornar a(s) coluna(s) usada(s) para agrupar, ou os resultados não farão sentido
 - Pode retornar-se mais colunas, não usadas para agrupar, mas devem ser funcionalmente dependentes das colunas a agrupar
- Não faz sentido agrupar por chaves candidatas: não haverá agregação!

Agregação com Agrupamento: Exercícios

- Quanto produziu cada animal em Janeiro de 2023?

Agregação com Agrupamento: Exercícios

- Quanto produziu cada animal em Janeiro de 2023?

$producer \overset{G}{SUM}(amount) (\sigma_{date > '12-31-2022' \wedge date < '02-01-2023'}(produce))$

```
SELECT producer, SUM(amount)
FROM produce
WHERE
    EXTRACT(YEAR FROM date)=2023 AND
    EXTRACT(MONTH FROM date)=1
GROUP BY producer
ORDER BY producer DESC;
```

Agregação com Agrupamento: Exercícios

- Qual o total de vendas dos produtos de cada espécie?

Agregação com Agrupamento: Exercícios

- Qual o total de vendas dos produtos de cada espécie?

species $\sum(price)$ (*buys* \bowtie *produce* $\bowtie_{producer=id}$ *nonpig*)

```
SELECT
    species, SUM(price)
FROM
    buys JOIN produce USING (code) JOIN nonpig ON (producer=id)
GROUP BY
    species;
```

Agregação com Agrupamento: Seleção

$\sigma_{total > 1000}(\text{seller } G_{SUM(price) \rightarrow total}(buys))$

```
SELECT seller, SUM(price) AS total
FROM buys
GROUP BY seller HAVING total > 1000;
```

+	-----	+	-----	+
	seller		total	
+	-----	+	-----	+
	101001		1650.00	
+	-----	+	-----	+

- Cláusula **HAVING** permite fazer seleção após a agregação

Agregação com Agrupamento: Seleção

- **HAVING**

- Semelhante ao **WHERE** mas a seleção é feita após a agregação
- Tem de ser sempre precedido por **GROUP BY**
- Podemos ter **WHERE** e **HAVING** na mesma query se quisermos filtrar linhas antes e após a agregação
- Permite fazer agregação sem retornar o agregado (i.e., podemos listar a agregação apenas no **HAVING** se só quisermos usar o agregado para filtrar a tabela)

Agregação com Agrupamento: Exercício

- Quais os porcos com total de vendas superior a €500?

Agregação com Agrupamento: Exercício

- Quais os porcos com total de vendas superior a €500?

$\Pi_{seller}(\sigma_{total > 500}(\rho_{seller} G_{SUM(price) \rightarrow total}(buys)))$

```
SELECT seller
FROM buys
GROUP BY seller HAVING SUM(price) > 500;
```

- Não precisamos de devolver os agregados, podemos apenas indicar/utilizar a agregação no HAVING!

Selects Encadeados

Selects Encadeados

- Tal como em Álgebra Relacional, o resultado de um **SELECT** em SQL é também uma tabela e como tal pode ser usado noutro **SELECT**
 - Na cláusula **FROM**
 - Para fazer queries complexas que requerem computação de tabelas intermédias (e.g. agregação de agregação)
 - Nas cláusulas **WHERE** ou **HAVING**
 - Para filtrar linhas (antes ou após agregação) por comparação de conjuntos
- As operações de conjuntos (**UNION**, **INTERSECT** e **EXCEPT**) são sequenciais, não encadeadas, mas vamos cobri-las também

Operações de Conjuntos

- Interseção:
 - Quais os nomes dos porcos que também são nomes de vacas?

$$\Pi_{name}(pig) \cap \Pi_{name}(\sigma_{species='cow'}(nonpig))$$

```
SELECT name FROM pig
      INTERSECT
SELECT name FROM nonpig WHERE species='cow' ;
```

$$\Pi_{name}(pig \bowtie_{p.name=np.name} (\sigma_{species='cow'}(nonpig)))$$

```
SELECT DISTINCT name
FROM pig INNER JOIN nonpig USING (name)
WHERE species='cow' ;
```


Operações de Conjuntos

- Quais os nomes de todos os animais?

$$\Pi_{name}(pig) \cup \Pi_{name}(nonpig)$$

```
SELECT name FROM pig
      UNION
SELECT name FROM nonpig
```

$$\Pi_{name}(pig) \bowtie \Pi_{name}(nonpig)$$

```
SELECT DISTINCT name FROM pig FULL JOIN nonpig USING (name);
```

Operações de Conjuntos

- Diferença:
 - Quais os nomes dos porcos que não são nomes de cabras?

$$\Pi_{name}(pig) - \Pi_{name}(\sigma_{species='goat'}(nonpig))$$

```
SELECT name FROM pig
EXCEPT
SELECT name FROM nonpig WHERE species='goat' ;
```

$$\Pi_{p.name}(\sigma_{species=NULL}(pig \bowtie_{p.name=np.name}(\sigma_{species='goat'}(nonpig))))$$

- **Requer SELECTs encadeados (queremos filtrar cabras antes do join e NULLs depois do join)**

Selects Encadeados no FROM

- Queries complexas:
 - Quais os nomes dos porcos que não são nomes de cabras?

$$\Pi_{p.name} (\sigma_{species=NULL} (pig \bowtie_{p.name=np.name} (\sigma_{species='goat'} (nonpig))))$$

```
SELECT
    name
FROM
    pig LEFT JOIN
        (SELECT name FROM nonpig WHERE species='goat') AS k USING (name)
WHERE
    k.species IS NULL;
```

Selects Encadeados no FROM

- Agregação de agregação:
 - Qual o volume máximo de vendas de entre todos os porcos?

$G_{MAX(total)}(seller \rightarrow G_{SUM(price) \rightarrow total}(buys))$

```
SELECT
    MAX(total)
FROM
    (SELECT SUM(price) AS total FROM buys GROUP BY seller) AS k;
```

- **Não precisamos de devolver a coluna do agrupamento dado que queremos apenas agregar a coluna agregada!**

Selects Encadeados no FROM

- Qual o volume de vendas de cada porco cujo nome é partilhado por vacas mas não por cabras?

$$\text{seller} \sum(\text{price}) (\text{buys} \bowtie (\pi_{p.\text{id} \rightarrow \text{seller}} (\sigma_{\text{species}='cow'} (\text{pig} \bowtie_{p.\text{name}=n.\text{name}} \text{nonpig})) - \pi_{p.\text{id} \rightarrow \text{seller}} (\sigma_{\text{species}='goat'} (\text{pig} \bowtie_{p.\text{name}=n.\text{name}} \text{nonpig}))))$$

```
SELECT seller, SUM(price)
FROM buys NATURAL JOIN (
    SELECT p.id AS seller FROM pig p JOIN nonpig USING (name)
    WHERE species = 'cow' EXCEPT
    SELECT p.id AS seller FROM pig p JOIN nonpig USING (name)
    WHERE species = 'goat'
) AS k
GROUP BY seller;
```

Atribuição

- Para simplificar queries complexas com **SELECTs** encadeados, a cláusula **WITH** permite pré-computar esse(s) **SELECT(s)** e declará-los à partida
- Corresponde à atribuição em Álgebra Relacional
- É particularmente útil quando teríamos de usar o mesmo SELECT encadeado várias vezes
 - Não porque poupe processamento (o query engine dos SGBD SQL é “esperto” o suficiente para não processar duas vezes a mesma sub-query dentro de uma query) mas porque poupa linhas de código e torna a query mais legível

Atribuição

- Qual o porco que realizou a venda mais cara?

$comparison \leftarrow \rho_{s1}(seller) \bowtie_{s1.price > s2.price} \rho_{s2}(seller)$
 $\Pi_{s1.id}(comparison) - \Pi_{s2.id}(comparison)$

```
WITH comparison AS (  
    SELECT s1.seller AS expensive, s2.seller AS cheap  
    FROM seller s1, seller s2 WHERE s1.price > s2.price  
)  
SELECT expensive AS seller FROM comparison EXCEPT  
SELECT cheap AS seller FROM comparison;
```

Selects Encadeados no WHERE/HAVING

- Comparação de Arrays (não há em Álgebra Relacional)
 - **[NOT] IN**: operador de comparação que verifica se um valor existe num array de valores: uma lista manual ou um select encadeado que devolve uma única coluna
 - **ANY & ALL**: modificadores de operadores de comparação (usados após o operador: =, >, <, !=, ...) para comparar um valor contra um array de valores
 - **IN(array) \Leftrightarrow ANY(array)**
 - **NOT IN(array) \Leftrightarrow != ALL(array)**

Selects Encadeados no WHERE/HAVING

- Com **IN**:
 - Qual o volume de vendas de cada porco cujo nome é partilhado por vacas mas não por cabras?

```
SELECT seller, SUM(price) FROM buys
WHERE seller IN (
    SELECT p.id FROM pig p JOIN nonpig USING (name) WHERE species = 'cow'
)
AND seller NOT IN (
    SELECT p.id FROM pig p JOIN nonpig USING (name) WHERE species = 'goat'
)
GROUP BY seller;
```

Selects Encadeados no WHERE/HAVING

- Com **ALL**:
 - Qual o porco que realizou a venda mais cara?

```
SELECT seller
FROM buys
WHERE price >= ALL(
    SELECT price
    FROM buys
);
```

Selects Encadeados no WHERE/HAVING

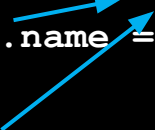
- Com **ALL**:
 - Qual o porco que tem o maior volume de vendas?

```
SELECT seller
FROM buys
GROUP BY seller HAVING SUM(price) >= ALL (
    SELECT SUM(price) FROM buys GROUP BY seller
);
```

Selects Encadeados no WHERE/HAVING

- **EXISTS**: operador que verifica se uma subquery (tipicamente relacionada com a query externa) retorna um conjunto não vazio
 - Qual o volume de vendas de cada porco cujo nome é partilhado por vacas mas não por cabras?

```
SELECT seller, SUM(price) FROM buys JOIN pig p ON (seller=id) WHERE EXISTS(  
    SELECT * FROM nonpig n WHERE p.name = n.name AND species = 'cow')  
AND NOT EXISTS(  
    SELECT * FROM nonpig n WHERE p.name = n.name AND species = 'goat')  
GROUP BY seller;
```



Selects Encadeados: Exercícios

- Qual espécie de animal mais produtiva?

Selects Encadeados: Exercícios

- Qual espécie de animal mais produtiva?

```
SELECT species
FROM produce JOIN nonpig ON (producer=id)
GROUP BY species HAVING SUM(amount) >= ALL(
    SELECT SUM(amount)
    FROM produce JOIN nonpig ON (producer=id)
    GROUP BY species
);
```

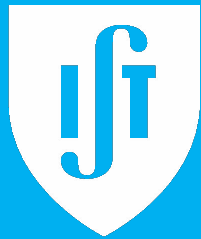
Selects Encadeados: Exercícios

- Qual a diferença de volume de vendas entre 2023 e 2022 para cada animal que produziu produtos em ambos os anos?

Selects Encadeados: Exercícios

- Qual a diferença de volume de vendas entre 2023 e 2022 para cada animal que produziu produtos em ambos os anos?

```
WITH sales23 AS (  
    SELECT producer, SUM(price) AS vol FROM produce JOIN buys USING (code)  
    WHERE EXTRACT(YEAR FROM date)=2023 AND producer IN (  
        SELECT producer FROM produce WHERE EXTRACT(YEAR FROM date)=2022)  
    GROUP BY producer),  
sales22 AS (  
    SELECT producer, SUM(price) AS vol FROM produce JOIN buys USING (code)  
    WHERE EXTRACT(YEAR FROM date)=2022 AND producer IN (  
        SELECT producer FROM produce WHERE EXTRACT(YEAR FROM date)=2023)  
    GROUP BY producer)  
SELECT producer, s3.vol-s2.vol FROM sales23 s3 JOIN sales22 s2 USING (producer);
```

TÉCNICO LISBOA