

Bases de Dados

Módulo 13a: View, Funções de Grupo e Cláusula GROUP BY

Prof. André Bruno de Oliveira

26/04/24 11:01

Tópicos

- Introdução para uso de Views
- Introdução + Base de Dados Exemplo
 - Funções de Grupo
 - Cláusula GROUP BY
 - A filosofia Split-Apply-Combine
 - Cláusula HAVING
 - Uso da base de dados [Censo_escolar.db](#).

Introdução – Uso de Views

- **O que é uma View?**
 - Uma view é uma maneira alternativa de observação de dados de uma ou mais entidades (tabelas), que compõem uma base de dados. Pode ser considerada como uma tabela virtual ou uma consulta armazenada. **Pode ser compreendida como um conjunto de resultados.**
- **Como se utiliza view?**
 - Uma view é uma implementação que encapsula uma instrução de seleção (SELECT), guardando os dados em uma “tabela virtual”.
 - Pode ser mais rápido executar uma consulta armazenada em forma de view do que executar uma instrução de SELECT nova, pois após a execução de uma consulta os resultados ficam armazenadas em cache.

Introdução – Uso de Views

- O uso de views possibilita implementar também restrição de acesso:

Exemplo:

- Seu departamento de vendas não precisa ter acesso a coluna que contenham valores (dados) referentes aos salários dos desenvolvedores, mas precisa acessar alguns dados da tabela *Salario*.
- Para isso, basta encapsular dentro da view uma seleção dos dados (SELECT) desejados da tabela salário sem que a coluna referente ao valor dos salários seja incluída.

Introdução – Uso de Views

- O uso de views possibilita associar várias tabelas e representar o resultado como uma “tabela virtual”:

Exemplo:

- Podemos ter várias "JOIN" encapsulados em uma view, formando somente uma “tabela virtual”.

Introdução – Uso de Views

- O uso de views possibilita agregar informações em vez de fornecer detalhes:

Exemplo:

- Podemos apresentar um somatório de despesas em ligações de um determinado usuário, restringindo acesso aos detalhes da conta.
- A agregação de dados será visto mais a frente neste módulo.

Introdução – Vantagens das Views

- As views possuem muitas vantagens, podemos citar 5 delas:
 - i) Economiza tempo com retrabalho
 - Você não precisar ficar reescrevendo aquela instrução enorme. É possível escrever uma vez e depois só utilizar;
 - ii) Favorece a velocidade de acesso às informações

Uma vez executada o resultado fica armazenado em uma tabela temporária (virtual).
 - iii) Permite mascarar complexidade do banco de dados

É possível aplicar alterações nas tabelas (nome de atributo, exclusão de atributo, criar variáveis derivadas) e esconder a complexidade da consulta sem afetar a interação com o usuário.

Introdução – Vantagens das Views

- As views possuem muitas vantagens, podemos citar 5 delas:

iv) Simplifica o gerenciamento de permissão de usuários;

Em vez de conceder permissão para que os usuários acessem diretamente as tabelas, os responsáveis pelos bancos de dados podem conceder permissões para que os usuários consultem dados somente através de views. Isso também protege as alterações na estrutura das tabelas.

Introdução – Vantagens das Views

- As views possuem muitas vantagens, podemos citar 5 delas:

- v) Preparação de dados para serem exportados

É possível criar uma view baseada em uma consulta complexa, que associe até 32 tabelas e depois exportar dados para outro aplicativo com a finalidade aplicar análises adicionais.

Introdução – Criação de Views

- A criação de uma view é algo bem simples. A complexidade fica por conta das operações existentes no comando SELECT.
- CREATE VIEW nome AS instrução select.
 - Vamos usar como exemplo query do exercício 8 da biblioteca: Recuperar a matrícula e nome do usuário, id do empréstimo, data do empréstimo, número da cópia, ISBN do livro, título e categorias dos livros da editora 'Alfa' que não foram escritos pelo autor 'Mário Quintana'.

CREATE VIEW *VperfilAssociado* AS

```
SELECT a.matricula, a.nome AS nome_associado, e.id, e.datEmprestimo,  
ecl.numCopia, l.isbn, l.titulo, c.nome AS desc_categoria  
FROM T_ASSOCIADO a  
INNER JOIN T_EMPRESTIMO e ON (a.matricula = e.matAssociado)  
INNER JOIN T_EMPRESTIMO_COPIA_LIVRO ecl ON (e.id = ecl.idEmprestimo)  
INNER JOIN T_COPIA_LIVRO c ON  
(c.isbnLivro = ecl.isbnLivro AND c.numCopia = ecl.numCopia)  
INNER JOIN T_LIVRO l ON (c.isbnLivro = l.isbn)  
INNER JOIN T_EDITORA edi ON (l.idEditora = edi.id)  
INNER JOIN T_LIVRO_CATEGORIA lc ON (l.isbn = lc.isbnLivro)  
INNER JOIN T_CATEGORIA c ON (lc.idCategoria = c.id)  
INNER JOIN T_LIVRO_AUTOR la ON (l.isbn = la.isbn)  
INNER JOIN T_AUTOR au ON (la.idAutor = au.id)  
WHERE (edi.nome = 'Alfa') AND (au.nome != 'Mario Quintana');
```

Introdução – Criação de Views

- A criação de uma view é algo bem simples.
 - Após a criação da view é possível usá-la como uma tabela no comando SELECT. Assim, é possível aplicar a mesma sintaxe utilizada para consultas de recuperação de dados em tabelas. Grosso modo é possível usar a view com junções e operações de conjunto.
 - A única exceção é que em alguns bancos de dados não aceitam encapsular o ORDER BY do final do SELECT na construção da view, mas é possível usar o ORDER BY após a view ser criada.

SELECT * FROM VperfilAssociado

	matricu	nome_a	id	datEmprest	numCo	isbn	titulo	desc_categoria
1	M987	A. Silva	1	2016-12-05	1	500-X	Homage to Catalonia	Biografia
2	M987	A. Silva	1	2016-12-05	1	500-X	Homage to Catalonia	Literatura Estrangeira
3	M987	A. Silva	3	2017-03-22	1	700-X	O Conde de Monte Cristo	Literatura Estrangeira

Introdução – Criação de Views

- Exemplo de view com order bay

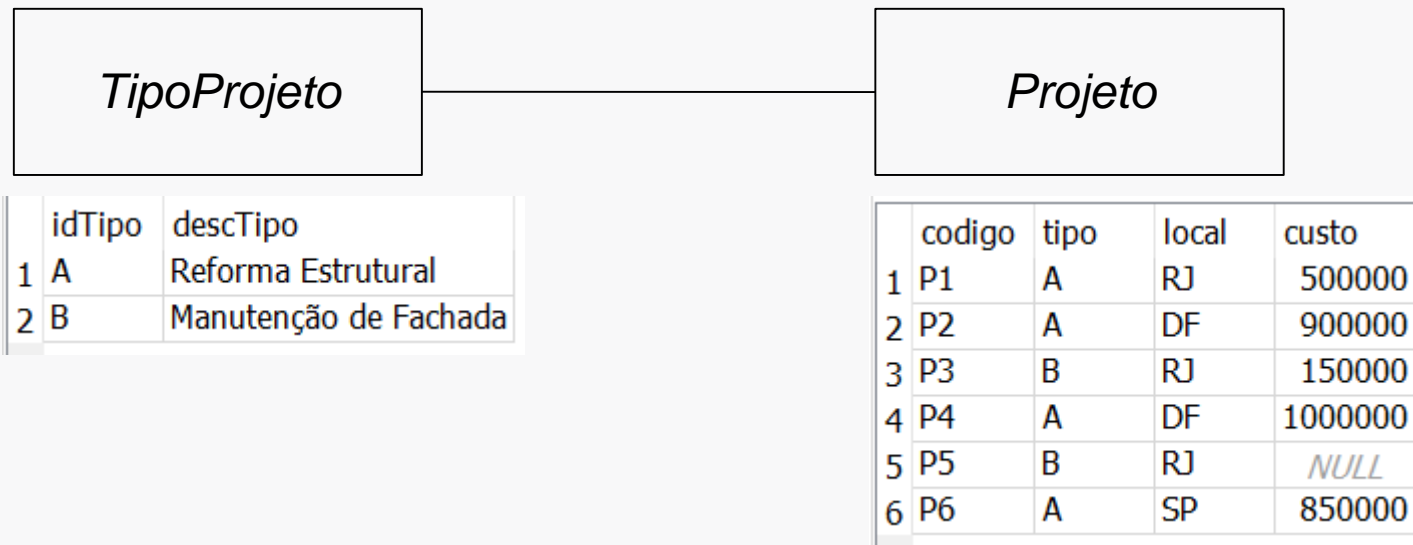
```
SELECT *  
FROM VperfilAssociado  
ORDER BY desc_categoria
```

	matricu	nome_a	id	datEmpresti	numCo	isbn	titulo	desc_categoria
1	M987	A. Silva	1	2016-12-05	1	500-X	Homage to Catalonia	Biografia
2	M987	A. Silva	1	2016-12-05	1	500-X	Homage to Catalonia	Literatura Estrangeira
3	M987	A. Silva	3	2017-03-22	1	700-X	O Conde de Monte Cristo	Literatura Estrangeira

FUNÇÕES DE GRUPO

Introdução

- A instrução SELECT oferece funções para a produção de resultados agregados.
- Elas são conhecidas como funções de grupo e serão apresentadas nesta aula com o uso de exemplos baseados nas tabelas Projeto e TipoProjeto.



Introdução

Script SQL para a criar e popular as tabelas:

```
CREATE TABLE TipoProjeto
(idTipo CHAR(1) NOT NULL,
descTipo VARCHAR(30) NOT NULL,
PRIMARY KEY(idTipo));
CREATE TABLE Projeto
(codigo CHAR(2) NOT NULL,
tipo CHAR(1) NOT NULL,
local CHAR(2) NOT NULL,
custo NUM,
PRIMARY KEY(codigo),
FOREIGN KEY (tipo) REFERENCES TipoProjeto(idTipo));
INSERT INTO TipoProjeto VALUES('A', 'Reforma Estrutural');
INSERT INTO TipoProjeto VALUES('B', 'Manutenção de Fachada');
INSERT INTO Projeto VALUES('P1','A','RJ',500000);
INSERT INTO Projeto VALUES('P2','A','DF',900000);
INSERT INTO Projeto VALUES('P3','B','RJ',150000);
INSERT INTO Projeto VALUES('P4','A','DF',1000000);
INSERT INTO Projeto VALUES('P5','B','RJ',NULL);
INSERT INTO Projeto VALUES('P6','A','SP',850000);
```

Introdução

- A tabela *Projeto* armazena os projetos correntemente conduzidos por uma empresa de reforma de edifícios.
- Atributos:
 - código (código do projeto);
 - tipo (FK para TipoProjeto);
 - local (UF onde obra está sendo realizada); e
 - custo (orçamento da obra em R\$).
- Observe que o projeto ‘P5’ ainda não tem o orçamento definido (está com valor NULL armazenado na coluna “custo”).

TipoProjeto

	idTipo	descTipo
1	A	Reforma Estrutural
2	B	Manutenção de Fachada

Projeto

	código	tipo	local	custo
1	P1	A	RJ	500000
2	P2	A	DF	900000
3	P3	B	RJ	150000
4	P4	A	DF	1000000
5	P5	B	RJ	NULL
6	P6	A	SP	850000

Função de Grupo

- Função COUNT(a)
- Determina o número de ocorrências de valores não-nulos para o atributo a de uma tabela.

```
SELECT  
  
    COUNT(codigo),  
    COUNT(custo)  
FROM Projeto;
```

Projeto

	codigo	tipo	local	custo
1	P1	A	RJ	500000
2	P2	A	DF	900000
3	P3	B	RJ	150000
4	P4	A	DF	1000000
5	P5	B	RJ	NULL
6	P6	A	SP	850000



	COUNT(codigo)	COUNT(custo)
1	6	5

Função de Grupo

- Função COUNT(a)
- Utilizada para determinar o número total de linhas de uma tabela.

SELECT

COUNT(*)


FROM *Projeto*;

Este tipo de sintaxe `select COUNT(*) from tabela` Pega todas as colunas da tabela durante a consulta, assim todos os dados estarão disponíveis na memória para o SGDB fazer a operação de contagem, ou seja pode gastar mais recursos do que especificar uma única coluna, caso o SGBD não faça alguma otimização.

Uma forma muito utilizada para contagem de linhas nas tabelas é `SELECT COUNT(0) FROM Projeto` por sugerir que o BD carregue uma quantidade menor de informações para a memória.

Projeto

	codigo	tipo	local	custo
1	P1	A	RJ	500000
2	P2	A	DF	900000
3	P3	B	RJ	150000
4	P4	A	DF	1000000
5	P5	B	RJ	NULL
6	P6	A	SP	850000



	COUNT(*)
1	6

Funções de Grupo

- Funções MAX (a), MIN(a), AVG(a), SUM(a)
- MIN(a): retorna o valor mínimo do atributo **a** de uma tabela.
- MAX(a): retorna o valor máximo do atributo **a** de uma tabela.
- AVG(a): retorna o valor médio do atributo **a** de uma tabela.
- SUM(a): retorna o somatório dos valores de **a** em todas as linhas da tabela.

SELECT


MIN(custo), MAX(custo),

AVG(custo), SUM(custo)

FROM *Projeto*;

Projeto

	codigo	tipo	local	custo
1	P1	A	RJ	500000
2	P2	A	DF	900000
3	P3	B	RJ	150000
4	P4	A	DF	1000000
5	P5	B	RJ	NULL
6	P6	A	SP	850000



	MIN(custo)	MAX(custo)	AVG(custo)	SUM(custo)
1	150000	1000000	680000	3400000

Cláusula GROUP BY

- A cláusula GROUP BY pode ser combinada com as funções de grupo para permitir a produção de resultados agregados por uma ou mais variáveis.

SELECT

Tipo, MIN(custo), MAX(custo),


AVG(custo), SUM(custo)

FROM *Projeto*

GROUP BY tipo;

Projeto

	codigo	tipo	local	custo
1	P1	A	RJ	500000
2	P2	A	DF	900000
3	P3	B	RJ	150000
4	P4	A	DF	1000000
5	P5	B	RJ	NULL
6	P6	A	SP	850000



	Tipo	MIN(custo)	MAX(custo)	AVG(custo)	SUM(custo)
1	A	500000	1000000	812500	3250000
2	B	150000	150000	150000	150000

Cláusula GROUP BY

- ATENÇÃO:** A variável categórica pela qual os resultados serão agregados (discriminados) deve ser indicada na lista do SELECT e ao lado da cláusula GROUP BY:.

SELECT

Tipo, MIN(custo), MAX(custo),


AVG(custo), SUM(custo)

FROM *Projeto*

GROUP BY *tipo*;

Projeto

	codigo	tipo	local	custo
1	P1	A	RJ	500000
2	P2	A	DF	900000
3	P3	B	RJ	150000
4	P4	A	DF	1000000
5	P5	B	RJ	NULL
6	P6	A	SP	850000



	Tipo	MIN(custo)	MAX(custo)	AVG(custo)	SUM(custo)
1	A	500000	1000000	812500	3250000
2	B	150000	150000	150000	150000

Cláusula GROUP BY com + 1 de um atributo

- É possível compor grupos formados por mais de um atributo.
- O exemplo a seguir, produz o somatório do orçamento por tipo de projeto e local.
- Os resultados são ordenados por “tipo” e “local” (o uso de ORDER BY não é requerido – ele foi utilizando nesse exemplo apenas para melhorar a apresentação dos resultados).

```
SELECT tipo, local, SUM(custo)
```

```
FROM Projeto
```

```
GROUP BY tipo, local
```

```
ORDER BY tipo, local;
```

Projeto

	codigo	tipo	local	custo
1	P1	A	RJ	500000
2	P2	A	DF	900000
3	P3	B	RJ	150000
4	P4	A	DF	1000000
5	P5	B	RJ	NULL
6	P6	A	SP	850000



	tipo	local	SUM(custo)
1	A	DF	1900000
2	A	RJ	500000
3	A	SP	850000
4	B	RJ	150000

Cláusula GROUP BY com WHERE

- A cláusula GROUP BY pode ser utilizada em conjunto com o WHERE.
- É importante saber que o WHERE é executado antes do GROUP BY, para definir as linhas que, de fato, serão levadas em conta para produzir a tabulação.
- Veja que a posição do GROUP BY está após o WHERE e antes do ORDER BY.

SELECT tipo, AVG(custo)

FROM *Projeto*

WHERE local <> 'SP'

GROUP BY tipo

ORDER BY tipo desc;

Projeto

	codigo	tipo	local	custo
1	P1	A	RJ	500000
2	P2	A	DF	900000
3	P3	B	RJ	150000
4	P4	A	DF	1000000
5	P5	B	RJ	NULL
6	P6	A	SP	850000



	tipo	AVG(custo)
1	B	150000
2	A	800000

Cláusula GROUP BY com WHERE

- A execução do SELECT abaixo será processada da seguinte forma:
 - (1) O SGBD faz a execução do FROM trazendo as informações para a memória (Buffer de trabalho).
 - (2) Executa o WHERE para selecionar as linhas desejadas, neste caso descarta o local 'SP'.
 - (3) O GROUP BY divide os grupos pela chave fornecida (a chave pode ter 1 ou mais atributos), neste caso dividido por tipo.
 - (4) A função de agregação AVG é usada para calcular a média de cada grupo formado somente pelas linhas selecionadas pelo WHERE.
 - (5) Combina os resultados intermediários.
 - (6) Ordena o resultado intermediário para montar a resposta.

```
SELECT tipo, AVG(custo)FROM Projeto
```

```
WHERE local <> 'SP'
```

```
GROUP BY tipo
```

```
ORDER BY tipo desc;
```


Cláusula GROUP BY com WHERE

Etapas de processamento da consulta:

SELECT tipo, AVG(custo) FROM *Projeto*

WHERE local <> 'SP'

GROUP BY tipo

ORDER BY tipo desc;

(1)-From e (2)-where

	tipo	custo
1	A	500000
2	A	900000
3	B	150000
4	A	1000000
5	B	<i>NULL</i>

Cláusula GROUP BY com WHERE

Etapas de processamento da consulta:

SELECT tipo, AVG(custo) FROM *Projeto*

WHERE local <> 'SP'

GROUP BY tipo

ORDER BY tipo desc;

Resultados (1) E (2) (3)- Group by

	tipo	custo
1	A	500000
2	A	900000
3	B	150000
4	A	1000000
5	B	NULL

	tipo	custo
1	A	500000
2	A	900000
3	A	1000000

	tipo	custo
1	B	150000
2	B	NULL

Cláusula GROUP BY com WHERE

Etapas de processamento da consulta:
 SELECT tipo, AVG(custo) FROM *Projeto*
 WHERE local <> 'SP'
 GROUP BY tipo
 ORDER BY tipo desc;

Resultados (1) E (2)

	tipo	custo
1	A	500000
2	A	900000
3	B	150000
4	A	1000000
5	B	NULL

Resultados (3)

	tipo	custo
1	A	500000
2	A	900000
3	A	1000000
1	B	150000
2	B	NULL

(4) - AVG

	tipo	avg(custo)
1	A	800000
1	B	150000

Cláusula GROUP BY com WHERE

Etapas de processamento da consulta:

SELECT tipo, AVG(custo) FROM *Projeto*

WHERE local <> 'SP'

GROUP BY tipo

ORDER BY tipo desc;

Resultados (1) E (2)

	tipo	custo
1	A	500000
2	A	900000
3	B	150000
4	A	1000000
5	B	NULL

Resultados (3)

	tipo	custo
1	A	500000
2	A	900000
3	A	1000000
1	B	150000
2	B	NULL

Resultados (4)

	tipo	avg(custo)
1	A	800000
1	B	150000

(5) - combina

	tipo	avg(custo)
1	A	800000
2	B	150000

Cláusula GROUP BY com WHERE

Etapas de processamento da consulta:

SELECT tipo, AVG(custo) FROM *Projeto*

WHERE local <> 'SP'

GROUP BY tipo

ORDER BY tipo desc;

Resultados (1) E (2)

	tipo	custo
1	A	500000
2	A	900000
3	B	150000
4	A	1000000
5	B	NULL

Resultados (3)

	tipo	custo
1	A	500000
2	A	900000
3	A	1000000
1	B	150000
2	B	NULL

Resultados (4)

	tipo	avg(custo)
1	A	800000
1	B	150000

Resultado (5)

	tipo	avg(custo)
1	A	800000
2	B	150000

(6) - Ordena

	tipo	AVG(custo)
1	B	150000
2	A	800000

Cláusula GROUP BY com WHERE

Etapas de processamento da consulta:

SELECT tipo, AVG(custo) FROM *Projeto*

WHERE local <> 'SP'

GROUP BY tipo

ORDER BY tipo desc;

Resultados (1) E (2)

	tipo	custo
1	A	500000
2	A	900000
3	B	150000
4	A	1000000
5	B	NULL

Resultados (3)

	tipo	custo
1	A	500000
2	A	900000
3	A	1000000
1	B	150000
2	B	NULL

Resultados (4)

	tipo	avg(custo)
1	A	800000
1	B	150000

Resultado (5)

	tipo	avg(custo)
1	A	800000
2	B	150000

Resposta

	tipo	AVG(custo)
1	B	150000
2	A	800000

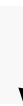
Cláusula HAVING

- Caso você queira eliminar algum grupo do resultado final, torna-se necessário utilizar a cláusula HAVING.
- O exemplo a seguir encontra um resultado com a soma de custo por tipo e local de projeto.

```
SELECT tipo, local, SUM(custo)  
FROM Projeto  
GROUP BY tipo, local  
ORDER BY tipo, local;
```

Projeto

	codigo	tipo	local	custo
1	P1	A	RJ	500000
2	P2	A	DF	900000
3	P3	B	RJ	150000
4	P4	A	DF	1000000
5	P5	B	RJ	NULL
6	P6	A	SP	850000



	tipo	local	SUM(custo)
1	A	DF	1900000
2	A	RJ	500000
3	A	SP	850000
4	B	RJ	150000


Cláusula HAVING

- Com o uso da cláusula **HAVING**, pode-se remover alguns resultados da tabulação final, ou seja, dos resultados produzidos após a tabulação.
- Veja que são eliminados todos os grupos em que o somatório do custo é inferior à R\$ 600.000,00, o que diferencia do exemplo anterior.

```
SELECT tipo, local, SUM(custo)
FROM Projeto
GROUP BY tipo, local
HAVING SUM(custo) > 600000
ORDER BY tipo, local;
```

Projeto

	codigo	tipo	local	custo
1	P1	A	RJ	500000
2	P2	A	DF	900000
3	P3	B	RJ	150000
4	P4	A	DF	1000000
5	P5	B	RJ	NULL
6	P6	A	SP	850000



	tipo	local	SUM(custo)
1	A	DF	1900000
2	A	SP	850000

Cláusula HAVING

- **ATENÇÃO:** WHERE e HAVING são de propósitos diferentes.
- O WHERE aplica uma restrição para selecionar as linhas que devem ser usadas pela agregação para tabular.
- O HAVING aplica restrição com uso de função de agregação para selecionar linhas após a agregação.

SELECT tipo, local, SUM(custo)

FROM Projeto

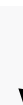
GROUP BY tipo, local

HAVING SUM(custo) > 600000

ORDER BY tipo, local;

Projeto

	codigo	tipo	local	custo
1	P1	A	RJ	500000
2	P2	A	DF	900000
3	P3	B	RJ	150000
4	P4	A	DF	1000000
5	P5	B	RJ	NULL
6	P6	A	SP	850000



	tipo	local	SUM(custo)
1	A	DF	1900000
2	A	SP	850000

Cláusula HAVING

- **ATENÇÃO.**
- O modo correto de usar HAVING é quando há GROUP BY, já que o HAVING serve exatamente para filtrar resultados produzidos pelo GROUP BY..
- O campo HAVING no SELECT deve vir após o GROUP BY e antes do ORDER BY.
- A cláusula HAVING pode envolver qualquer atributo do conjunto de tabelas do resultado final, conforme os exemplo de consultas abaixo.

```
SELECT tipo, local, SUM(custo)
FROM Projeto
GROUP BY tipo, local
HAVING SUM(custo) > 600000
ORDER BY tipo, local;
```

```
SELECT tipo, local, SUM(custo)
FROM Projeto
GROUP BY tipo, local
HAVING tipo='A'
ORDER BY tipo, local;
```

```
SELECT tipo, local, SUM(custo)
FROM Projeto
GROUP BY tipo, local
HAVING local= 'RJ'
ORDER BY tipo, local;
```

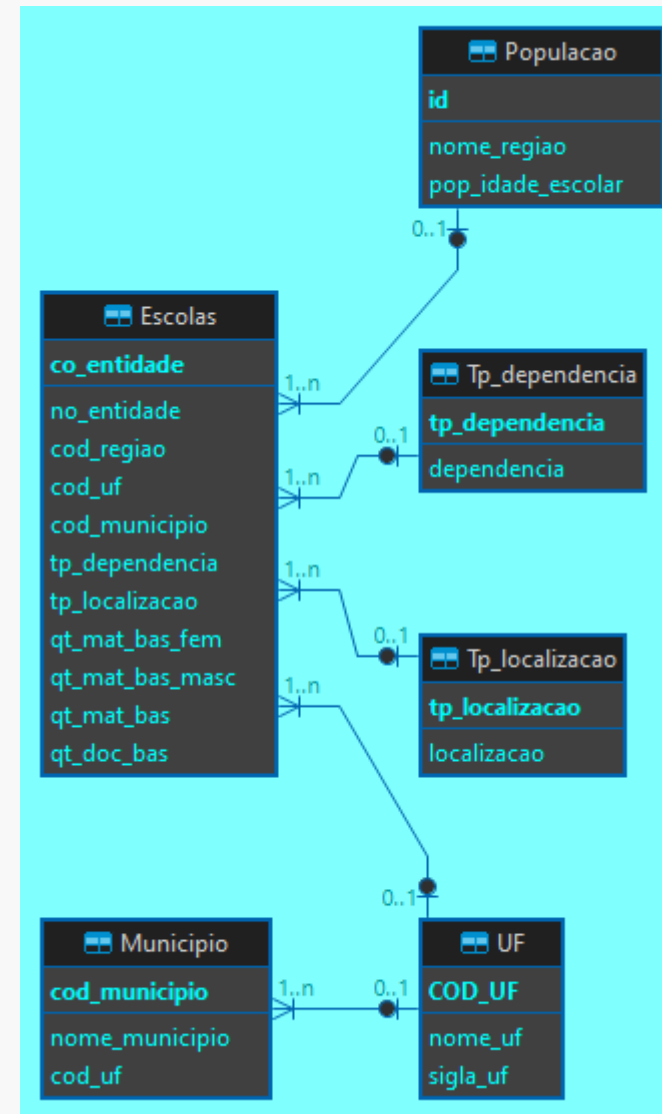
Comentários finais

- As funções de grupo e as cláusulas GROUP BY e HAVING podem ser usadas normalmente em consultas com junção de tabelas e visões, incluindo cláusulas como WHERE e ORDER BY.
- Assim com a combinação de diversas cláusulas no SELECT torna-se possível produzir tabulações com SQL.

Exercícios Propostos

Considere o banco de dados reduzido do Censo_escolar.db.

- As chaves primárias estão em destaque logo abaixo do nome da tabela.
- A tabela *Escola* tem 4 FK conforme é visto no diagrama: `cod_região` com `id` de *Populacao*, `cod_uf`, `cod_municipio`, `tp_dependencia`, `tp_localizacao.`
- A tabela *Municipio* tem uma FK com UF.
- A base de dados no formato do banco Sqlite está disponível na pasta de material do curso em **bases**.



Exercícios Propostos

Considere o banco de dados reduzido do Censo_escolar.db.

Escolas

	co_entida	no_entidade	cod_re	cod_uf	cod_mur	tp_dep	tp_loca	qt_mat	qt_mat	qt_mat	qt_doc
1	11001100	EMEF GEDOCY RUAS WOLFF	1	11	1100205	3	2	NULL	NULL	NULL	
2	11002158	EMEF SAO FRANCISCO DE ASSIS	1	11	1100205	3	1	70	58	128	7
3	11003391	EMEF MANOEL MACIEL NUNES	1	11	1100205	3	2	21	22	43	2
4	11003901	EMEF ALUIZIO FERREIRA	1	11	1100338	3	2	NULL	NULL	NULL	
5	11004428	EEEFM BURITI	1	11	1100452	2	1	337	293	630	22
6	11005360	FFFFM PAULJO FRFTRF	1	11	1101104	2	1	500	524	1024	58

Municipio

	cod_municipio	nome_municipio	cod_uf
1	1100015	Alta Floresta D'Oeste	11
2	1100023	Ariquemes	11
3	1100031	Cabixi	11
4	1100049	Cacoal	11

Tp_localizacao

	tp_localizacao	localizacao
1	1	Urbana
2	2	Rural

Uf

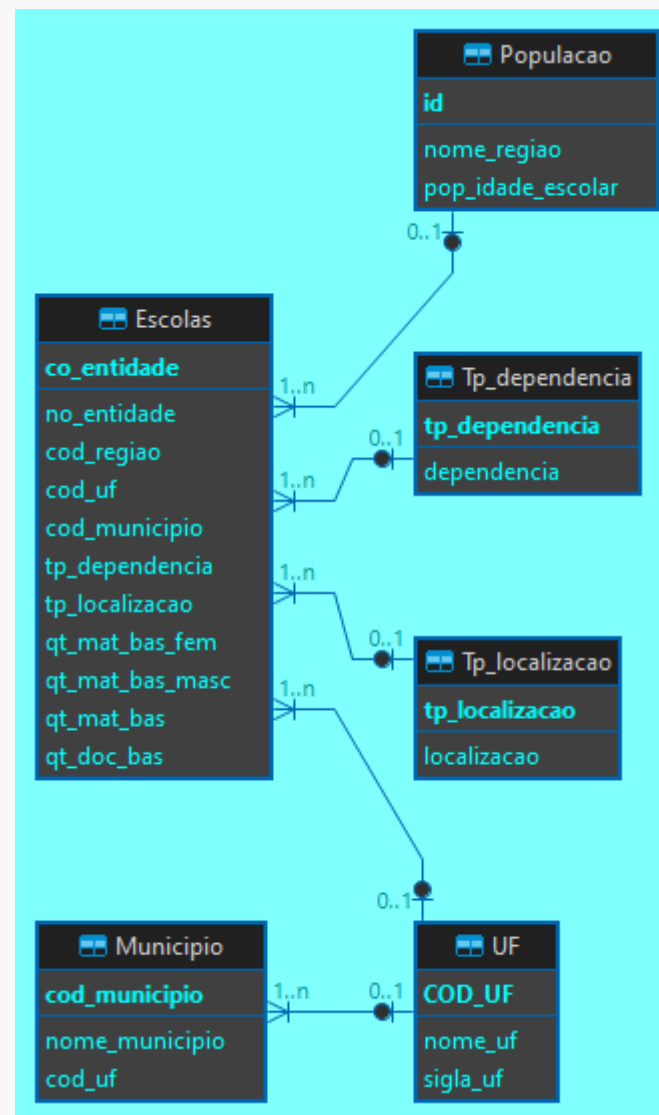
	COD_UF	nome_uf	sigla_uf
1	11	RONDÔNIA	RO
2	12	ACRE	AC
3	13	AMAZONAS	AM

Populacao

	id	nome_regiao	pop_idade_escolar
1	1	Norte	5305101
2	2	Nordeste	14105336
3	3	Sudeste	18550814
4	4	Sul	6665363
5	5	Centro Oeste	4107944

Tp_dependencia

	tp_dependencia	dependencia
1	1	Federal
2	2	Estadual
3	3	Municipal
4	4	Privada



Exercícios Propostos

- (1) Obtenha uma lista com nome do região e total de escolas de cada região. Classifique o resultado pela ordem descendente do total de escolas.
- (2) Construa uma query que retorne para cada região o total de alunas do sexo feminino. Com base neste resultado encontre a região com menor número de alunas do sexo feminino matriculadas no ensino básico. Atenção o resultado deve exibir uma única linha com região de menor número de matrículas femininas.
- (3) Encontre o total de Municípios de São Paulo que não estão contidos na tabela Escolas.
- (4) Encontre o total de escolas públicas (Federal, Estadual, Municipal) em cada unidade federativa. Sabe-se que tp_dependencia 1, 2 e 3 corresponde a Federal, Estadual e Municipal.
- (5) Monte uma lista com o nome da UF, o total de escolas públicas e o total de alunas da região rural por UF. Organize o resultado pelo total de matrículas de mulheres nas ordem decrescente e como segundo critério o total de escolas públicas na região na ordem crescente. (Sabe-se que TP_localizacao=2 refere-se a região rural).

View, Funções de Grupo e Cláusula GROUP BY

- (1) (1) Obtenha uma lista com nome da região e total de escolas de cada região.
Classifique o resultado pela ordem descendente do total de escolas.

```
SELECT P.nome_regiao, COUNT(P.nome_regiao) as total_escolas  
FROM ESCOLAS E INNER JOIN Populacao P on E.cod_regiao=P.id  
GROUP BY nome_regiao  
ORDER BY 2 DESC;
```

	nome_regiao	total_escolas
1	Nordeste	1517
2	Sudeste	1498
3	Sul	591
4	Norte	517
5	Centro Oeste	266

View, Funções de Grupo e Cláusula GROUP BY

(2) Construa uma query que retorne para cada região o total de alunas do sexo feminino. Com base neste resultado encontre a região com menor número de alunas do sexo feminino matriculadas no ensino básico. Atenção o resultado deve exibir uma única linha com região de menor número de matrículas femininas.

```
SELECT nome_regiao, min(Alunas) FROM  
(SELECT P.nome_regiao, SUM(QT_MAT_BAS_FEM) as Alunas  
FROM ESCOLAS E INNER JOIN Populacao P on E.cod_regiao=P.id  
GROUP BY P.nome_regiao) V ;
```

OU

```
CREATE VIEW V_TOTAL_ALUNAS_REGIAO AS  
(SELECT P.nome_regiao, SUM(QT_MAT_BAS_FEM) as Alunas  
FROM ESCOLAS E INNER JOIN Populacao P on E.cod_regiao=P.id  
GROUP BY P.nome_regiao);  
  
SELECT nome_regiao, min(Alunas) FROM V_TOTAL_ALUNAS_REGIAO  
GROUP BY P.nome_regiao) V ;
```

	nome_regiao	min(Alunas)
1	Centro Oeste	34432

View, Funções de Grupo e Cláusula GROUP BY

- (2) Construa uma query que retorne para cada região o total de alunas do sexo feminino. Com base neste resultado encontre a região com menor número de alunas do sexo feminino matriculadas no ensino básico. Atenção o resultado deve exibir uma única linha com região de menor número de matrículas femininas.

```
SELECT nome_regiao, min(Alunas) FROM  
(SELECT P.nome_regiao, SUM(QT_MAT_BAS_FEM) as Alunas  
FROM ESCOLAS E INNER JOIN Populacao P on E.cod_regiao=P.id  
GROUP BY P.nome_regiao) V ;
```

Primeiro, é preciso encontrar o total de matrícula de mulheres para cada região. Isso, pode ser feito somando-se os valores contidos no atributo qt_mat_bas_fem (quantidade de matrículas femininas do ensino básico em cada escola).

O raciocínio é semelhante aos anteriores, sempre que se deseja calcular um total agrupando para um conjunto de atributo, usa-se este conjunto no SELECT e na cláusula GROUP BY. Desta vez foi usada a função SUM para somar qt_mat_bas_fem de todas as escolas agregadas por região.

Numa segunda etapa usa-se o resultado desta query mais interna no FROM para encontrar a região com menor número de alunas. A função min (mínimo) encontra a linha o valor menor, o que torna possível listar o nome da região associado a este valor. (Query internas ainda serão estudadas mais detalhamento).

View, Funções de Grupo e Cláusula GROUP BY

(2) Construa uma query que retorne para cada região o total de alunas do sexo feminino. Com base neste resultado encontre a região com menor número de alunas do sexo feminino matriculadas no ensino básico. Atenção o resultado deve exibir uma única linha com região de menor número de matrículas femininas.

```
CREATE VIEW V_TOTAL_ALUNAS_REGIAO AS  
SELECT P.nome_regiao, SUM(QT_MAT_BAS_FEM) as Alunas  
FROM ESCOLAS E INNER JOIN Populacao P on E.cod_regiao=P.id  
GROUP BY P.nome_região;  
  
SELECT nome_regiao, min(Alunas) FROM V_TOTAL_ALUNAS_REGIAO ;
```

Veja que a construção da VIEW inclui a complexidade de encontrar o total de matrículas para cada região. Assim, o SEELCT para encontrar o resultado faz uso da visão de dados como se fosse uma tabela com os atributos nome_regiao e alunas.

Em situações do cotidiano, a *view* é usada para guardar o SQL mais complexo e facilitar a obtenção de resultados através de SELECT com código mais legível.

Álgebra Relacional – Casamentos de Tuplas

(3) Encontre o total de Municípios de São Paulo que não estão contidos na tabela Escolas.

```
select count(1) from (  
  SELECT cod_municipio  
  FROM Municipio M  
  WHERE M.cod_uf=35  
except  
  SELECT cod_municipio FROM Escolas);
```

OU

```
SELECT COUNT(1) FROM (  
  SELECT distinct M.nome_municipio  
  FROM Municipio M  
  WHERE M.cod_municipio NOT IN (SELECT E.cod_municipio From Escolas E)  
  AND M.COD_UF=35);
```

	count(1)
1	422

Álgebra Relacional – Casamentos de Tuplas

(3) Encontre o total de Municípios de São Paulo que não estão contidos na tabela Escolas.

```
select count(1) from (  
  SELECT cod_municipio  
  FROM Municipio M  
  WHERE M.cod_uf=35  
  except  
  SELECT cod_municipio FROM Escolas);
```

	count(1)
1	422

Como se deseja encontrar um total do conjunto final resultado de uma diferença entre dois conjuntos, ou seja encontrar o que há na relação município e que não tem na relação Escolas. Pode-se aplicar a operação diferença.

Um cuidado fundamental para se aplicar uma diferença entre duas relações é que elas devem ser compatíveis, mesmo número de atributos e tipos. Neste caso, aplicou-se um seleção em Municipio para retornar somente o códigos de municípios do Estado de São Paulo cujo o código da UF é 35. No caso da tabela Escolas, fez-se somente a seleção do código de município.

Com as relações novas estão compatíveis, pode-se usar o comando EXCEPT que faz o mesmo papel da diferença entre relações, só que no SQL. Com o resultado da diferença, usou-se o COUNT para contar o número de linhas.

Álgebra Relacional – Casamentos de Tuplas

(3) Encontre o total de Municípios de São Paulo que não estão contidos na tabela Escolas.

```
SELECT COUNT(1) FROM (  
  SELECT distinct M.nome_municipio  
  FROM Municipio M  
  WHERE M.cod_municipio NOT IN (SELECT E.cod_municipio From Escolas E)  
  AND M.COD_UF=35);
```

	count(1)
1	422

Há uma segunda solução para este tipo de query, que é verificar quais código de municípios existem na tabela Municipio e não estão constidos da tabela Escolas.

Quando se usa tipo solução com a cláusula NOT IN é preciso a lista de atributos de ambos as consultas sejam compatíveis em número de atributos e tipo.

Ainda vamos ver mais exemplos sobre o uso do NOT IN e do EXISTS que também poderia ter sido usado.

View, Funções de Grupo e Cláusula GROUP BY

- (4) Encontre o total de escolas públicas (Federal, Estadual, Municipal) em cada unidade federativa e ordene para que as UF com menores total de escolas públicas sejam listadas primeiro. Sabe-se que tp_dependencia 1, 2 e 3 -- corresponde a Federal, Estadual e Municipal.

```
SELECT UF.COD_UF,NOME_UF,  
COUNT(1) as total_Escolas_publicas  
from Uf  
INNER JOIN Escolas E ON E.cod_uf=Uf.cod_uf  
WHERE tp_dependencia in (1,2,3) -- Federal Estadual Municipal  
GROUP BY UF.COD_UF,UF.NOME_UF  
Order by total_Escolas_publicas;
```

View, Funções de Grupo e Cláusula GROUP BY

- (5) Monte uma lista com o nome da UF, o total de escolas públicas e o total de alunas da região rural por UF. Organize o resultado pelo total de matrículas de mulheres nas ordem decrescente e como segundo critério o total de escolas públicas na região na ordem crescente. (Sabe-se que TP_localizacao=2 refere-se a região rural).

```
SELECT UF.COD_UF, NOME_UF
, sum(qt_mat_bas_fem) as total_Feminina_rural
, COUNT(1) as Total_escolas_publicas
from Uf
INNER JOIN Escolas E ON E.cod_uf=Uf.cod_uf
WHERE E.tp_dependencia in (1,2,3) and E.TP_localizacao=2
GROUP BY UF.COD_UF,UF.NOME_UF
Order by total_Feminina_rural desc, Total_escolas_publicas asc;
```

A diferente desta query para a query do exercício anterior está na região rural que afeta o numerador e denominador.

View, Funções de Grupo e Cláusula GROUP BY

(5)

```
SELECT UF.COD_UF, NOME_UF
, sum(qt_mat_bas_fem) as total_Feminina_rural
, COUNT(1) as Total_escolas_publicas
from Uf
INNER JOIN Escolas E ON E.cod_uf=Uf.cod_uf
WHERE E.tp_dependencia in (1,2,3)
      E.TP_localizacao=2
GROUP BY UF.COD_UF,UF.NOME_UF
Order by total_Feminina_rural
        Total_escolas_publicas asc;
```

	COD_UF	NOME_UF	total_Feminina_rural	Total_escolas_publicas
2	29	BAHIA	5870	187
3	15	PARÁ	4534	146
4	26	PERNAMBUCO	3647	79
5	23	CEARÁ	3422	79
6	13	AMAZONAS	3036	88
7	31	MINAS GERAIS	2551	152
8	27	ALAGOAS	2232	34
9	35	SÃO PAULO	2169	30
10	51	MATO GROSSO	2024	24
11	22	PIAUÍ	1924	63
12	33	RIO DE JANEIRO	1639	42
13	41	PARANÁ	1575	28
14	43	RIO GRANDE DO SUL	1422	45
15	25	PARAÍBA	1124	50
16	50	MATO GROSSO DO SUL	841	7
17	28	SERGIPE	767	21
18	42	SANTA CATARINA	737	23

Obrigado