

# SELECTS AVANÇADOS

## JOINS E SUBCONSULTAS

# Buscando Informações no Banco de Dados

- Uma das principais atividades de um *Desenvolvedor de Software* é gerar **consultas personalizadas** sobre os registros disponíveis no banco de dados;
- A razão de utilizar sistemas comerciais vai além do controle de processos. Um sistema deve **gerar informação e conhecimento** para os usuários;
- Esse conhecimento é gerado através de **consultas** (*selects*) na base de dados;

# Regras Básicas da Consulta

- O usuário do sistema não sabe e não precisa saber o que é Chave Primária, Estrangeira, Atributo, Tipo de dados, entre outros;
- As informações devem ser mostradas na mesma **linguagem** do usuário, para que ele compreenda o conhecimento gerado pelo sistema;
- Uma consulta **jamaís** deve:
  - ▣ Mostrar número de Chave Estrangeira;
  - ▣ Mostrar nome de Atributo;

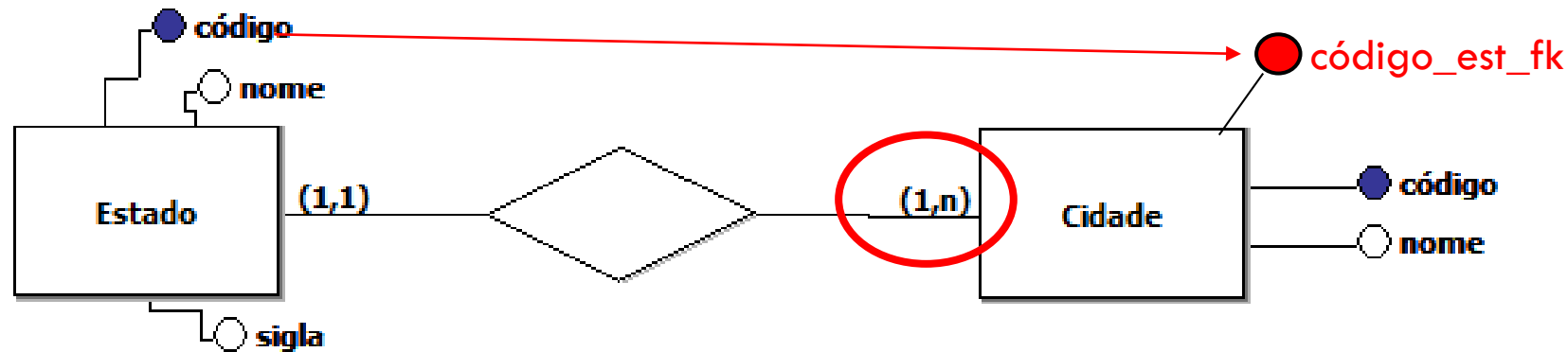
# Tipos de Consultas

- Basicamente existem **dois tipos** de consultas:
  - ▣ **Consulta Regular:** Por comparação direta de chaves;
  - ▣ **Consulta por Junção:** Com utilização de Joins;
- Vamos lembrar a consulta com a comparação direta de chaves. Utilize o Script SQL - BD Mecânica 5.0 disponível no AVA;

# Select com Múltiplas Tabelas

- ❑ Duas tabelas estão relacionadas quando possuem um relacionamento N para 1. Assim a tabela com o N recebe a chave primária (FK) da tabela com o 1.
- ❑ No SELECT por comparação direta devemos **comparar** o atributos comum entre as **02 tabelas** relacionadas;
- ❑ Ou seja, comparamos a **chave primária** da tabela com o 1 com chave estrangeira que está na tabela com o N;

# Exemplo



**Podemos afirmar que:**

- ❑ **Um** (1) Estado está relacionado a **muitas** (N) Cidades;
- ❑ **Uma** (1) Cidade está relacionada a **Um** (1) Estado;

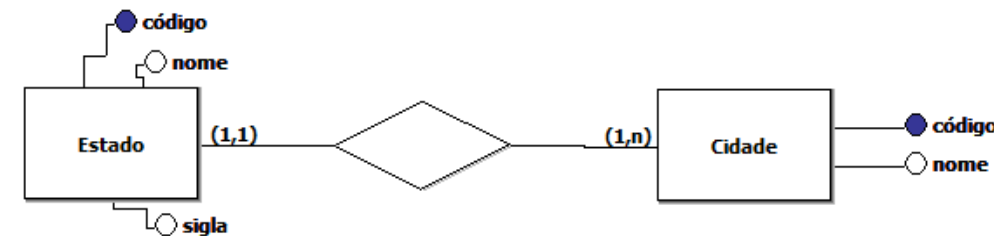
**Logo:**

- ❑ Cidade está com o N, então recebe a Chave Primária de Estado que está com o 1;

# Sintaxe

- ❑ **SELECT**
- ❑ tabela1.atributo1,
- ❑ tabela2.atributo2
- ❑ **FROM**
- ❑ tabela1, tabela2
- ❑ **WHERE**
- ❑ (tabela1.atributoPK = tabela2.atributoFK)

# Exemplo com 02 Tabelas



- ❑ **Tarefa:** Selecione todos os registros de Cidade, porém, substitua a Chave Estrangeira de Estado pelo Nome do Estado.

- ❑ **Solução:**

- **SELECT**

- CIDADE.cod\_cid as 'ID Cidade',
- CIDADE.nome\_cid as 'Nome Cidade',
- ESTADO.nome\_est as 'Nome Estado'

Selecione os atributos a serem mostrados, mas adicione o **nome da tabela** de origem do atributo

- **FROM**

- CIDADE, ESTADO

Informe quais tabelas estão sendo selecionadas

- **WHERE**

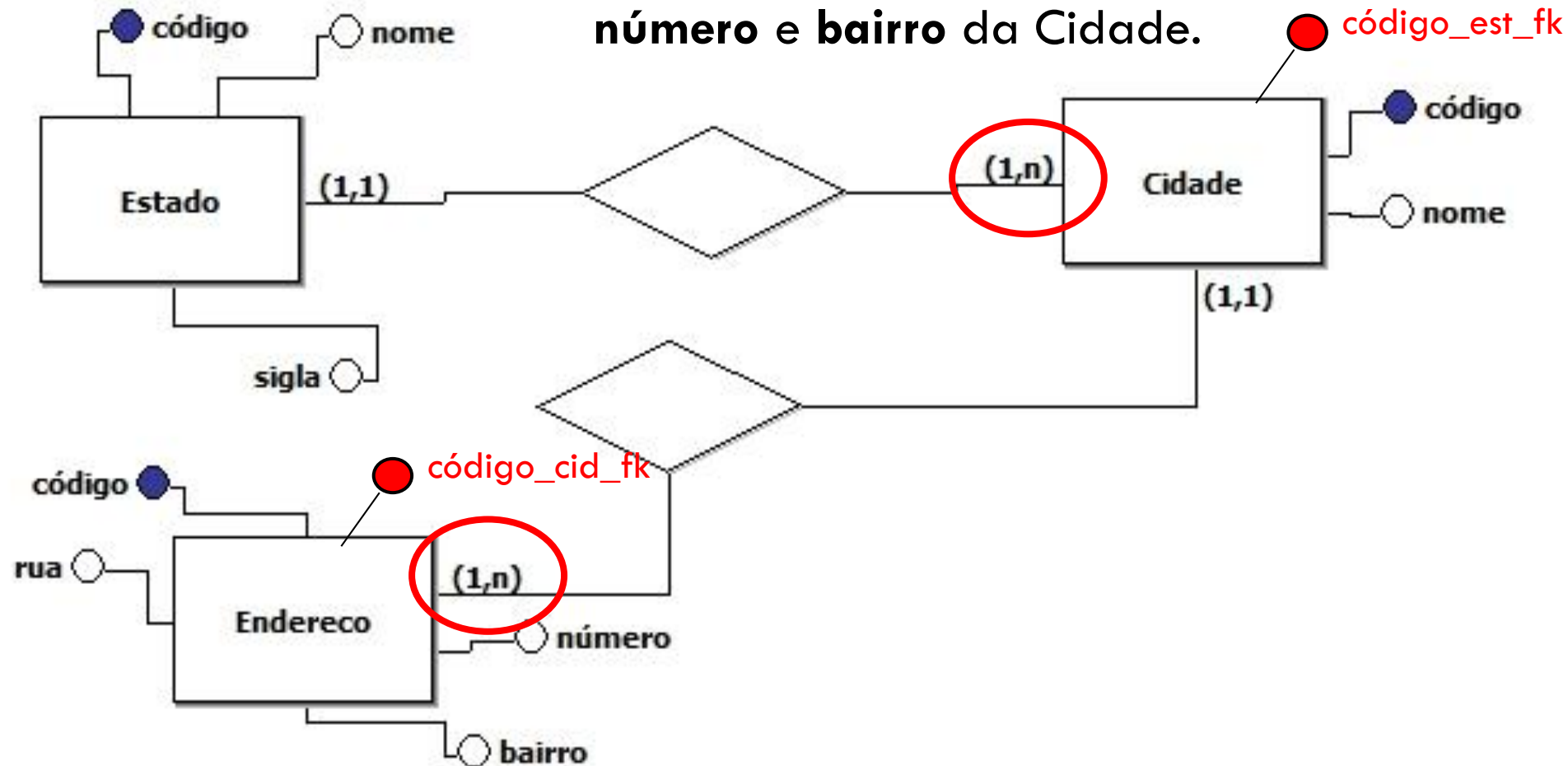
- (ESTADO.cod\_est = CIDADE.cod\_est\_fk);

Compare as **chaves comuns** entre as tabelas, ou seja, a chave **estrangeira** com a chave **primária**



# Exemplo com 03 Tabelas

**Tarefa:** Selecione o **nome** do Estado, **nome** da Cidade e a **rua**, **número** e **bairro** da Cidade.



# Exemplo com 03 Tabelas

- **SELECT**

- CIDADE.nome\_cid as 'Nome Cidade',
- ESTADO.nome\_est as 'Nome Estado',
- ENDERECO.rua\_end as 'Rua',
- ENDERECO.numero\_end as 'Nº',
- ENDERECO.bairro\_end as 'Bairro'

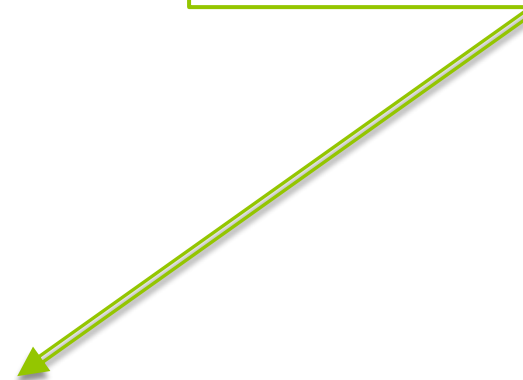
- **FROM**

- CIDADE, ESTADO, ENDERECO

- **WHERE**

- (ESTADO.cod\_est = CIDADE.cod\_est\_fk) **AND**
- (CIDADE.cod\_cid = ENDERECO.cod\_cid\_fk);

Comparação entre as FK e PK  
que une as 03 tabelas



# Exercícios de Fixação I

1. Faça um consulta que mostre todos os dados de **Funcionário**, mas substitua as FK por informações compreensíveis;
2. Faça um consulta que mostre todos os dados de **Cliente**, mas substitua as FK por informações compreensíveis;
3. Faça um consulta que mostre todos os dados de uma **Venda**, mas substitua as FK por informações compreensíveis;
4. Faça um consulta que mostre todos os dados de uma **Compra**, mas substitua as FK por informações compreensíveis;

# SELECTS AVANÇADOS

## JOINS

# JOINS

- ❑ **Joins** (junções) são um recurso presente nos bancos de dados relacionais, através da qual é possível juntar o conteúdo de duas tabelas através de um critério;
- ❑ É um conceito que muitas vezes quem está iniciando no mundo dos bancos de dados relacionais tem dificuldade de entender;
- ❑ Os Joins mais utilizadas são:  
**INNER, LEFT, RIGHT, CROSS**

# JOINS

## ➤ Sintaxe:

- ❑ **SELECT**

- ❑ Tabela1.atributo1,

- ❑ Tabela2.atributo1,

- ❑ **FROM**

- ❑ Tabela1 **TIPO JOIN** Tabela2

- ❑ **ON**

- ❑ (tabela1.atributoPK = tabela2.atributoFK);

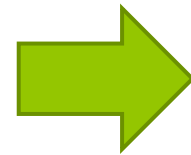
# Usaremos o **BD Heróis** para Exemplos

```
1 • create database Herois;
2 • use Herois;
3
4 • create table Origem (
5     cod_ori int primary key not null,
6     nome_ori varchar (200) not null
7 );
8
9 • insert into Origem values (1, 'Marvel');
10 • insert into Origem values (2, 'DC Comics');
11 • insert into Origem values (3, 'TV');
12 • insert into Origem values (4, 'Internet');
13
```

```
14
15 • create table Heroi (
16     cod_hero int primary key not null,
17     nome_hero varchar (200) not null,
18     arma_hero varchar (100),
19     cod_ori_fk int,
20     foreign key (cod_ori_fk) references Origem (cod_ori)
21 );
22
23 • insert into heroi values (1, 'Capitão America', 'Escudo', 1);
24 • insert into heroi values (2, 'Homem de Ferro', 'Armadura', 1);
25 • insert into heroi (cod_hero, nome_hero) values (3, 'Huck');
26 • insert into heroi (cod_hero, nome_hero, cod_ori_fk)
27     values (4, 'Power Rangers', 3);
28 • insert into heroi (cod_hero, nome_hero) values (5, 'Big Hero');
29 • insert into heroi values (6, 'Homem-Aranha', 'Teia', 1);
30 • insert into heroi values (7, 'Super Homem', 'Força', 2);
31 • insert into heroi values (8, 'Batman', 'Dinheiro', 2);
32 • insert into heroi values (9, 'Seya de Pegasus', 'Cosmo', 3);
33
```

# Cross Join

- Quando queremos juntar duas ou mais tabelas por cruzamento. Ou seja, para cada linha da tabela ORIGEM queremos todos os HEROIS ou vice-versa;
- #Mostra um cruzamento de todos os registros da tabela
- **SELECT**
- origem.nome\_ori as 'Origem',
- heroi.nome\_hero as 'Nome Heroi'
- **FROM**
- origem **CROSS JOIN** heroi;



Result Grid			Filter Rows:	Export:
	nome_ori	nome_hero		
	Marvel	Capitão America		
	DC Comics	Capitão America		
	TV	Capitão America		
	Internet	Capitão America		
	Marvel	Homem de Ferro		
	DC Comics	Homem de Ferro		
	TV	Homem de Ferro		
	Internet	Homem de Ferro		

Result 33 x



# INNER JOIN

- Usado quando queremos juntar **duas ou mais tabelas** por coincidência e tem o mesmo efeito do SELECT por comparação direta de chaves;
- No INNER JOIN são selecionados **apenas os registros que possuem relação**, ou seja, que possuem a FK preenchida. Já os registros com a **FK Nula são descartados**.
- No caso de HEROI e ORIGEM os atributos internos coincidentes são cod\_ori na tabela **ORIGEM** e cod\_ori\_fk na tabela **HEROI**;

# Exemplo Inner Join

- ❑ #mostra os valores vinculados através da chave estrangeira (cod\_ori). Os registros não vinculados SÃO DESCARTADOS.
- ❑ **SELECT**
- ❑ origem.nome\_ori as 'Origem',
- ❑ heroi.nome\_hero as 'Nome Heroi'
- ❑ **FROM**
- ❑ origem **INNER JOIN** heroi
- ❑ **ON** (origem.cod\_ori = heroi.cod\_ori\_fk);



Result Grid		Filter Rows:
	nome_ori	nome_hero
▶	Marvel	Capitão America
	Marvel	Homem de Ferro
	Marvel	Homem-Aranha
	DC Comics	Super Homem
	DC Comics	Batman
	TV	Power Rangers
	TV	Seya de Pegasus

Result 34 x

Output

# LEFT JOIN

- Observando o resultado do SELECT com o INNER que os heróis *Huck* e *Big Hero* não apareceram porque não possuíam relacionamento.
- Percebe-se também a origem *Internet* também não apareceu, porque não possui nenhum herói relacionado a ela;
- Se desejarmos listar **todos os Heróis com suas respectivas Origens, incluindo os heróis sem origem**, a exemplo de *Huck* e *Big Hero*, devemos o **LEFT** no lugar INNER, assim, todos os registros **à esquerda** do JOIN serão mostrados, independente se possuem ou não relação;

# Exemplo Left Join

- ❑ #busca tudo que esta na tabela da **esquerda** da **comparação** (join) e vincula com a tabela direita (join) inclusive os registros sem relação;
- ❑ **SELECT**
- ❑ origem.nome\_ori as 'Origem',
- ❑ heroi.nome\_hero as 'Nome Heroi'
- ❑ **FROM**
- ❑ heroi **LEFT JOIN** origem
- ❑ **ON** (heroi.cod\_ori = origem.cod\_ori);



Result Grid		Filter Rows:	Export
	nome_hero	nome_ori	
▶	Capitão America	Marvel	
	Homem de Ferro	Marvel	
	Huck	NULL	
	Power Rangers	TV	
	Big Hero	NULL	
	Homem-Aranha	Marvel	
	Super Homem	DC Comics	
	Batman	DC Comics	
	Seya de Pegasus	TV	

# RIGHT JOIN

- ❑ Observando o resultado da consulta com o LEFT JOIN notamos que a origem *Internet* não apareceu, pois ela não possui relacionamentos e a tabela Origem estava a direita da comparação (JOIN);
- ❑ Se desejarmos listar todas as ORIGENS e seus respectivos HEROIS, incluindo as ORIGENS sem HEROIS, poderíamos usar o **RIGHT JOIN**;

# Exemplo de Righth Join

- ❑ #busca tudo que esta na tabela da **direita** da **comparação** (antes do join) e vincula com a tabela esquerda (depois do join) inclusive os registros null da tabela a esquerda;
- ❑ **SELECT**
- ❑ origem.nome\_ori as 'Origem',
- ❑ heroi.nome\_hero as 'Nome Heroi'
- ❑ **FROM**
- ❑ heroi **RIGHT JOIN** origem
- ❑ **ON** (heroi.cod\_ori = origem.cod\_ori);



Result Grid		Filter Rows:	Ex
	nome_hero	nome_ori	
▶	Capitão America	Marvel	
	Homem de Ferro	Marvel	
	Homem-Aranha	Marvel	
	Super Homem	DC Comics	
	Batman	DC Comics	
	Power Rangers	TV	
	Seya de Pegasus	TV	
	NULL	Internet	

Result 44 x

# Full Join vs Union All





- ❑ Na linguagem SQL existe o **Full Join** que tem a função de vincular todos os registros da direita com os da esquerda incluindo os registro null e vice-versa;
- ❑ Entretanto no **MySQL Workbench** esse comando **não é reconhecido**;
- ❑ Usamos então o **Union all** para **simular** o comando **full join**;

# Exemplo Union All

- ❑ #alternativa do FULL JOIN usando a solução com o UNION, mostra tudo a **direita** com a **esquerda**
- ❑ **SELECT**
- ❑ origem.nome\_ori as 'Origem',
- ❑ heroi.nome\_hero as 'Nome Heroi'
- ❑ **FROM** origem **LEFT JOIN** heroi **ON** (heroi.cod\_ori = origem.cod\_ori)
- ❑ **UNION ALL**
- ❑ **SELECT**
- ❑ origem.nome\_ori as 'Origem',
- ❑ heroi.nome\_hero as 'Nome Heroi'
- ❑ **FROM** origem **RIGHT JOIN** heroi **ON** (heroi.cod\_ori = origem.cod\_ori);



# Exemplo Resultado com Union All

Result Grid     Filter Rows: <input type="text"/>   Export:    Wrap Cell Content: 		
	nome_ori	nome_hero
▶	Marvel	Capitão America
	Marvel	Homem de Ferro
	Marvel	Homem-Aranha
	DC Comics	Super Homem
	DC Comics	Batman
	TV	Power Rangers
	TV	Seya de Pegasus
	Internet	NULL
	Marvel	Capitão America
	Marvel	Homem de Ferro
	NULL	Huck
	TV	Power Rangers
	NULL	Big Hero
	Marvel	Homem-Aranha
	DC Comics	Super Homem
	DC Comics	Batman
	TV	Seya de Pegasus

Result 41 x

# Exercício de Fixação II

1. Refaça as consultas do Exercício de Fixação I. Mas agora utilize a comparação com JOIN;
2. Faça uma consulta que mostre o nome do cliente e os nome dos produtos comprados por ele entre 2018 e 2020 (*escolha entre usar o JOIN e a Comparação Direta*);

# SELECTS AVANÇADOS

## SUBCONSULTAS

Prof. Jackson Henrique

# Subconsultas Simples

- O tipo mais comum de subconsulta é a simples, ou seja, quando **unimos duas consultas**;
- A subconsulta simples utiliza uma subconsulta na cláusula WHERE (condição) de uma consulta normal;
- Essa subconsulta tem como objetivo retornar **um valor**, que será utilizado na comparação de valores na condição da consulta principal;
- **LEMBRE-SE:** A subconsulta **não pode** retornar mais de um registro ou mais de um atributo;

# Subconsultas Simples

- Normalmente a subconsulta utiliza **funções** (MAX, MIN, AVG, SUM e COUNT) para **retornar uma valor** desejado;
- Mas também é possível buscar **um valor** dentro de um atributo em um **registro específico**, desde que **não retorne mais de um valor**;
- **LEMBRE-SE:** As funções MAX e MIN **podem retornar** mais de um registro, então teste a subconsulta de forma separada antes;
- A subconsulta pode **ser independente** da consulta principal, ou seja, a consulta principal pode consultar uma **tabela A** e a subconsulta consultar a **tabela B**;

# Subconsultas Simples

- ❑ **Sintaxe da Subconsulta Simples:**
- ❑ **SELECT**
- ❑ atributo1,
- ❑ atributo2
- ❑ **FROM**
- ❑ Tabela1
- ❑ **WHERE**
- ❑ (atributo3 = (**SELECT** atributo1 **FROM** Tabela2 **WHERE** condição));

# Exemplos Subconsultas Simples

- ❑ Utilize o **Script de BD Agência Bancária 5.0.**
- ❑ **Objetivo:** Selecione o nome e a renda do cliente que possui a maior renda.
  - **SELECT**
  - nome\_cli as 'Cliente',
  - renda\_cli as 'Maior Renda'
  - **FROM**
  - Cliente
  - **WHERE**
  - (renda\_cli = (**SELECT MAX**(renda\_cli) **FROM** cliente));

# Exemplos Subconsultas Simples

- **Objetivo:** Selecione o nome e a renda do cliente que possui renda superior a menor renda.
- **SELECT**
- nome\_cli as 'Cliente',
- renda\_cli as 'Maior Renda'
- **FROM**
- Cliente
- **WHERE**
- (renda\_cli > (SELECT **MIN**(renda\_cli) **FROM** cliente);



# Exemplos Subconsultas Simples

- ❑ **Objetivo:** Selecione o nome e a renda do cliente que possui a renda menor do que a soma dos saldos das conta correntes.
- **SELECT**
- nome\_cli as 'Cliente',
- renda\_cli as 'Renda'
- **FROM**
- Cliente
- **WHERE**
- (renda\_cli > (SELECT **SUM**(saldo\_cc) **FROM** conta\_corrente));

# Subconsultas Avançadas

- Porém existem formas mais eficazes de aplicar as subconsultas;
- Podemos utilizar uma **subconsulta no lugar de uma atributo** ao invés de utiliza-la somente na cláusula WHERE (condição);
- Chamamos esse tipo de **subconsulta de avançada**;
- Neste tipo de consulta a subconsulta pode buscar uma informação em **uma tabela diferente** da tabela onde esta sendo feita a consulta, **desde que** essa subconsulta possua **uma tabela de conexão** com a consulta principal;

# Subconsultas Avançadas

## ➤ **Sintaxe:**

### ❑ **SELECT**

❑ tabela1.atributo1,

❑ tabela1.atributo2,

❑ (SELECT atributo1 FROM tabela2 WHERE (tabela1.atributoFK =  
tabela2.atributoFK))

### ❑ **FROM**

❑ Tabela1

### ❑ **WHERE**

❑ (condição);

# Exemplo de Subconsultas Avançadas

- ❑ Busca as informações da **conta corrente** na consulta principal (azul). Já na subconsulta (vermelho) busca a **soma dos saques** realizados na tabela Saque de acordo com cada conta;
- ❑ Observe a **comparação da chave primária com a estrangeira** na cláusula WHERE da subconsulta;

- ❑ **EXEMPLO:**

1. SELECT
2. conta\_corrente.cod\_cc AS 'Código da Conta Corrente',
3. conta\_corrente.numero\_cc AS 'Número da Conta Corrente',
4. (SELECT SUM(valor\_saq) FROM saque WHERE (conta\_corrente.cod\_cc = saque.cod\_cc\_fk)) as Saques
5. FROM
6. Conta\_corrente;

# Exemplo de Subconsultas Avançadas

- ❑ Subconsultas podem ser utilizadas **em conjunto** com **consultas em múltiplas tabelas**;
- ❑ Observe a consulta abaixo, ela busca dados na tabela Cliente e Conta\_Corrente na consulta principal utilizando o INNER JOIN e na subconsulta ela verifica os saques realizados pela conta correta;

```
1.  SELECT
2.  cliente.nome_cli AS 'Nome do Cliente',
3.  conta_corrente.numero_cc AS 'Número da Conta Corrente',
4.  conta_corrente.saldo_cc AS 'Saldo da Conta Corrente',
5.  (SELECT SUM(valor_saq) FROM saque WHERE (conta_corrente.cod_cc = saque.cod_cc_fk))
   as Saques
6.  FROM
7.  Cliente INNER JOIN Conta_corrente
8.  ON (cliente.cod_cli = conta_corrente.cod_cli_fk);
```

# Exemplo de Subconsultas Avançadas

- ❑ Observe a consulta abaixo, ela busca os me dados da consulta anterior, mas utiliza agora uma segunda subconsulta.
- 1.     SELECT
- 2.     cliente.cod\_cli AS 'Código do Cliente',
- 3.     cliente.nome\_cli AS 'Nome do Cliente',
- 4.     conta\_corrente.numero\_cc AS 'Número da Conta',
- 5.     conta\_corrente.saldo\_cc AS 'Saldo em Conta',
- 6.     (SELECT SUM(valor\_saq) FROM saque WHERE (conta\_corrente.cod\_cc = saque.cod\_cc\_fk))  
      as 'Total de Saques',
- 7.     (SELECT SUM(valor\_dep) FROM deposito WHERE (conta\_corrente.cod\_cc =  
      deposito.cod\_cc\_fk)) as 'Total de Depósitos'
- 8.     FROM
- 9.     cliente LEFT JOIN conta\_corrente
- 10.    ON (cliente.cod\_cli = conta\_corrente.cod\_cli\_fk);

# Exemplo de Subconsultas Avançadas

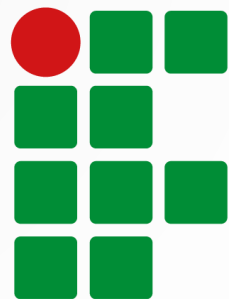
- Observe a consulta abaixo, ela busca os mesmos dados da anterior e acrescenta as informações sobre o banco e agência do cliente.

```
1.  SELECT
2.  banco.nome_ban AS 'Banco',
3.  Agencia.numero_ag AS 'Número da Agência',
4.  conta_corrente.numero_cc AS 'Número da Conta',
5.  cliente.cod_cli as 'Código do Cliente',
6.  cliente.nome_cli as 'Nome do Cliente',
7.  conta_corrente.saldo_cc as 'Saldo em Conta',
8.  (select sum(valor_saq) from saque where (conta_corrente.cod_cc = saque.cod_cc_fk)) as 'Total de Saques',
9.  (select sum(valor_dep) from deposito where (conta_corrente.cod_cc = deposito.cod_cc_fk)) as 'Total de Depósitos'
10. FROM
11. cliente INNER JOIN conta_corrente ON (cliente.cod_cli = conta_corrente.cod_cli_fk)
12. INNER JOIN agencia ON (agencia.cod_ag = conta_corrente.cod_ag_fk)
13. INNER JOIN banco ON (agencia.cod_ban_fk = banco.cod_ban);
```

# Exercício de Fixação III

1. Teste todas as subconsultas simples e avançadas demonstradas no slide;
2. Adicione novas informações as consultas avançadas apresentadas no slide, como por exemplo a soma e média dos pagamentos e transferência, contagem de saques e depósitos, entre outros;





**INSTITUTO  
FEDERAL**  
Rondônia

**Jackson Henrique**

Professor de Informática

**[Jackson.henrique@ifro.edu.br](mailto:Jackson.henrique@ifro.edu.br)**