

Banco de Dados - SQL

Prof. Carlos Bazilio

<http://www.ic.uff.br/~bazilio>

carlosbazilio@id.uff.br

O que é?

- SQL = DDL + DML
- Structured Query Language = Data *Definition* Language + Data *Manipulation* Language
- Implementa operadores de Algebra Relacional (AR) + as seguintes operações:
 - Inserção, Atualização e Remoção
 - CRUD (CReate, Update, Delete)

Versões

- Ansi SQL
- SQL 92 (SQL 2)
- SQL 99 (SQL 3)
 - Características Objeto-Relacionais e outras capacidades
 - Será o alvo do nosso estudo

Definindo Esquemas de Tabelas em SQL

- DDL (Data Definition Language)
- Tipos básicos: CHAR, CHAR(n), VARCHAR, VARCHAR(n), BIT, BIT(n), BOOLEAN, INTEGER, SHORTINT, FLOAT, DATE, TIME, TIMESTAMP, ...

- Criação de tabelas:

```
CREATE TABLE Product (  
    maker CHAR (1) DEFAULT 'A',  
    model INTEGER,  
    type VARCHAR (20) );
```

Definindo Esquemas de Tabelas em SQL

- Exemplos de Remoção e Alteração de tabelas:

`DROP TABLE Product;`

`ALTER TABLE Product ADD date DATE;`

`ALTER TABLE Product DROP type;`

Restrições (Constraints)

- Uma forma de definição de elementos ativos em Bancos de Dados
- Utilizados para garantir integridade do sistema (tanto na visão do modelo relacional quanto na de regras de negócio)
- Um exemplos de restrições são as chaves primárias e estrangeiras

Chave Primária

- Exemplo:

```
CREATE TABLE PC (  
    model INTEGER PRIMARY KEY,  
    speed INTEGER,  
    ... ); ou
```

```
CREATE TABLE PC (  
    model INTEGER,  
    speed INTEGER,  
    PRIMARY KEY (model), ...);
```

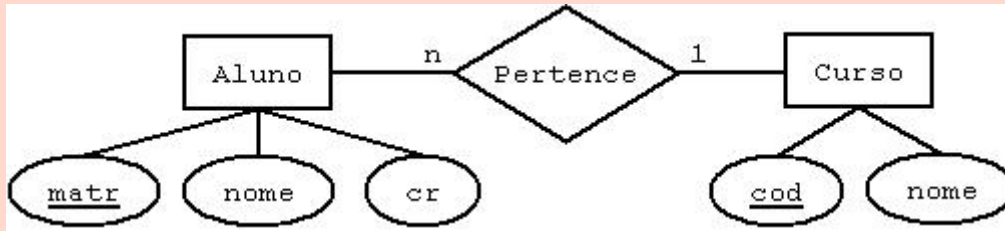
Campo UNIQUE

- Além da chave primária, com o modificador UNIQUE podemos garantir também que outros campos tenham valores distintos
- Diferentemente de chave primária, um campo UNIQUE permite valores NULL

```
CREATE TABLE Printer (  
    model INTEGER PRIMARY KEY,  
    color BOOLEAN,  
    type VARCHAR(50) UNIQUE, ...);
```


Chaves Estrangeiras

- Possibilitam a associação entre tabelas



```
CREATE TABLE Aluno (  
    matr INTEGER,  
    nome VARCHAR (100) NOT NULL,  
    cr FLOAT,  
    codCurso INTEGER,  
    PRIMARY KEY (matr),  
    FOREIGN KEY (codCurso) REFERENCES Curso(cod);
```

Manutenção da Integridade Referencial

- Na realização de modificações sobre tuplas que são referenciadas em outras tabelas (chaves estrangeiras), as seguintes ações podem ser definidas:
 - Rejeição das modificações (operação padrão em BDs)
 - Modificações em cascata (altera também as referências)
 - Modificações gerando NULL (no caso de remoções, é atribuído NULL à referência)

Sintaxe de Ações sobre Chaves Estrangeiras

```
CREATE TABLE Aluno (  
    matr INTEGER,  
    nome VARCHAR (100) NOT NULL,  
    cr FLOAT,  
    codCurso INTEGER,  
    PRIMARY KEY (matr),  
    FOREIGN KEY (codCurso) REFERENCES Curso(cod)  
    ON (DELETE | UPDATE) (CASCADE | SET NULL |  
    REJECT) *;
```

* Sintaxe de expressões regulares: () encapsula termos e | é um OU de termos

Exemplo de Consulta

Select model

From PC

Where speed > 1000 AND rd = '16xDVD';

Select model: atributos selecionados (π)

From PC: tabelas afetadas

Where ...: condição de filtro (σ), que é opcional

Expressão em AR:

$\pi_{\text{model}}(\sigma_{\text{speed} > 1000 \text{ AND RD} = '16\text{xDVD}'}(PC))$

Projeção

- Lista de atributos do *select*
- Podem ser renomeados (ρ em AR)
- Podem conter expressões e constantes
- Curinga '*' lista todos os atributos
- Pode conter o modificador *DISTINCT*

Select DISTINCT ram AS memória, price*1.2,
 'reais' AS moeda

From ...

Where ...;

Seleção

- Conteúdo do *where*
- Pode conter qualquer expressão condicional usual de Ling's de Programação

Select *

From PC

Where speed > 100 AND hd <> 40 AND
NOT (ram < 512);

Ordenação

- Cláusula *Order By*
- Ordena as tuplas de uma tabela segundo valores de um ou mais campos
- Select model
- From PC
- Where ...
- Order By price, ram DESC;

Strings

- Representadas entre ' '
- Podem ter tamanho fixo ou variável (CHAR e VARCHAR)
- Podem ser comparadas lexicograficamente (=, <, >, ...)
- Pode-se utilizar também o operador LIKE, que compara strings baseadas num padrão:

Select model

From Printer

Where type LIKE 'las___' OR type LIKE '%jet%';

Data e Hora

- Implementações de SQL usualmente suportam tipos específicos para data (DATE) e hora (TIME)
 - Exemplo: '1975-07-07' e '20:05:00.5'
- Timestamp é um tipo comum que combina data e hora
 - Exemplo: '1975-07-07 20:05:00'

Null

- Valor especial em SQL
- Entretanto, não há como utilizá-los explicitamente em expressões SQL (um teste, por exemplo)
- Em cláusulas *Where*, precisamos nos preparar para a ocorrência de valores nulos:
 - Operações aritméticas envolvendo NULL retornam NULL
 - Operações de comparação envolvendo NULL retornam UNKNOWN (terceiro valor-verdade)

Unknown

- Terceiro valor-verdade
- Pode ser interpretado como $\frac{1}{2}$, com TRUE=1 e FALSE=0
- Fórmulas
 - $A \text{ AND } B = \text{MIN}(A, B)$
 - $A \text{ OR } B = \text{MAX}(A, B)$
 - $\text{NOT } A = |1 - A|$
- Caso atípico:
Select * From Laptop
Where screen > 14.0 OR screen <= 14.0;

Consultas com + de 1 Relação

- Produtos e Junções

Select maker

From Product, Printer

Where Product.model = Printer.model AND
Printer.type = 'laser';

- *From Product, Printer* realiza o produto cartesiano entre as 2 tabelas

Variáveis de Tupla

- Imagine uma consulta que retorne os modelos de PC's com a mesma quantidade de hd

Select p1.model, p2.model

From PC p1, PC p2

Where p1.hd = p2.hd AND p1.model < p2.model;

União, Interseção e Diferença de Consultas

- UNION, INTERSECT e EXCEPT
- Modificador ALL considera duplicatas

(Select model
From Product
Where maker = 'A') UNION
(Select model
From PC
Where price < 1000)

Sub-consultas Aninhadas

- Produzindo um valor atômico:

Select maker

From Product

Where type = 'printer' AND Product.model =

(Select model

From Printer

Where price < 500)

Condições envolvendo Relações (1/2)

- Operadores existentes: EXISTS, IN, ALL, ANY e combinações com NOT

Select maker

From Product

Where type = 'printer' AND Product.model IN

(Select model

From Printer

Where price < 500)

Condições envolvendo Relações (2/2)

- Significado dos operadores:
 - EXISTS R: TRUE se $R \neq \{\}$
 - s IN R: TRUE se $s \in R$
 - s > ALL R: TRUE se $s > t, \forall t \in R$
 - s > ANY R: TRUE se $s > t, \exists t \in R$

O que esta consulta retorna?

Select p1.make

From Product p1

Where p1.type = 'printer' AND EXISTS (

Select *

From Product p2

Where p2.model = p1.model AND

p2.make <> p1.make);

E esta?

```
Select l.model  
From Laptop l  
Where l.price < ANY (  
    Select p.price  
    From PC p);
```

Sub-consultas Relacionadas

- Consultas que obrigam a avaliação de consultas aninhadas diversas vezes

Select model

From Laptop l

Where price <= ANY

(Select price

From PC

Where l.hd >= hd);

Sub-consultas em Cláusulas *From*

```
Select distinct p1.make  
From Product p1,  
( Select *  
From Printer  
Where color = 'true' ) p2  
Where p1.model = p2.model;
```

Expressões de Junção SQL (Joins)

- Sejam R e S relações (tabelas)
 - R cross join S; (produto cartesiano)
 - R join S on CONDIÇÃO; (junção theta)
 - R natural join S; (junção natural)
 - R natural {full | left | right} outer join (outer join)

Agregação – Operadores

- SUM, AVG, MIN, MAX e COUNT
- Tipicamente utilizados com expressões escalares com uma coluna numérica

Select AVG (price)

From Laptop;

- Exceção é o COUNT (*), que conta o número de tuplas de uma relação
- DISTINCT também pode ser utilizado:
 - COUNT (DISTINCT x)

Agrupamento

- Cláusula *Group By*

Select type, COUNT(DISTINCT maker)

From Product

Group By type;

- Somente atributos citados na cláusula *Group By* podem aparecer sem operadores no *Select*

Agrupamento

- Para a consulta anterior:

Select type, COUNT(DISTINCT maker)

From Product

Group By type;

- Como reformulá-la para retornar produtos diferentes de PC?

Agrupamento

```
Select type, COUNT(DISTINCT maker)  
From Product  
Where type <> 'PC'  
Group By type;
```

Agrupamento

E para retornar apenas produtos com número ≥ 3 fabricantes?

Cláusulas HAVING

- Condições aplicada a grupos

Select type, COUNT(DISTINCT maker)

From Product

Group By type

HAVING COUNT(DISTINCT maker) >= 3;

Exercícios

- 6.1.3, 6.2.2, 6.3.1 e 6.4.6 do livro do Molina

Atualização em Bancos de Dados

- Inserção, Remoção e Alteração
- Inserção:

INSERT INTO R ($A_1, \dots, A_n$) VALUES ($v_1, \dots, v_n$);

Onde:

R: Relação

A_k : Atributos

v_k : valores

INSERT INTO R VALUES ($v_1, \dots, v_n$);

- Usado quando há valores v_k para todos os atributos de R

Atualização em Bancos de Dados

- Remoção

DELETE FROM R WHERE <condição>;

- Atualização

UPDATE R SET <atribuições> WHERE
<condição>;

- Exemplo:

UPDATE Laptop

SET price = price * 1.1

WHERE speed >= 600;

Exercícios

- 6.5.1 do livro do Molina

Índices

- São criados baseados em 1 ou + campos
- São estruturas de dados que tornam mais eficientes as consultas sobre esse atributos

CREATE INDEX velocIndex ON PC(speed);

DROP INDEX velocIndex;

- 2 fatores a se considerar:
 - Tornam eficientes as consultas que utilizam este atributos
 - Tornam mais custosas as operações de inserção, remoção e atualização

Visões

- São relações abstratas (virtuais)
- `CREATE TABLE` cria relações que existem fisicamente, enquanto que `CREATE VIEW` cria visões, as quais não existem fisicamente e tipicamente expressam o resultado de um consulta
- Diferentemente do resultado das consultas, visões podem ser consultadas como tabelas (como se existissem fisicamente)

Exemplo de Definição e Uso de Visão

```
CREATE VIEW Barganha(modelo, ram, hd,  
preço) AS  
SELECT model, ram, hd, price  
FROM Laptop  
WHERE price < 1300;
```

```
SELECT Min(preco)  
FROM Barganha;
```

Modificação de Visões (Visões Atualizáveis)

- Nem sempre são possíveis
- A operação que mais restringe é a inclusão:
 - Esta não pode infringir a integridade do banco (os atributos não citados na inclusão precisam poder admitir NULL ou valores padrão)
 - Por exemplo, se o campo *speed* da relação Laptop não admitisse NULL
 - Uma tupla inserida numa visão precisa ser inserida na tabela base e constar na visão resultante
 - No exemplo citado, não podemos inserir um laptop com *preço* ≥ 1300

Modificação de Visões (Visões Atualizáveis)

- Remoções e atualizações numa visão são traduzidas para operações equivalentes na tabela base:

```
DELETE FROM Barganha  
WHERE ram < 512;
```

```
DELETE FROM Laptop  
WHERE ram < 512 AND price < 1300;
```

Modificação de Visões (Visões Atualizáveis)

- O comando `DROP TABLE` remove a tabela base e inutiliza (e eventualmente também remove) as visões baseadas nesta tabela
- `DROP VIEW` remove apenas a visão, ou seja, não remove as tuplas da tabela base

Asserções (Assertions)

- Uma asserção é uma expressão em SQL de valor booleano que precisa ser verdadeira sempre

- Formato:

```
CREATE ASSERTION <nome> CHECK  
  <condição>;
```

```
DROP ASSERTION <nome>;
```

- Qualquer modificação que invalide uma asserção será rejeitada

Asserções

Exemplo

```
CREATE ASSERTION
    statusHistoricoIncorreto CHECK (
NOT EXISTS (
    SELECT *
    FROM Aluno a
    WHERE a.cr > 0 AND NOT EXISTS (
        SELECT *
        FROM Historico h
        Where a.mat = h.mat );
```


Gatilhos (Triggers)

- Um gatilho é uma série de ações que são associadas a eventos num BD, como inserções, atualizações e remoções
- Regras ECA (Evento – Condição – Ação)

```
CREATE TRIGGER Atualizar
AFTER INSERT ON Historico
REFERENCING NEW ROW AS NovaTupla
FOR EACH ROW BEGIN
    UPDATE Aluno a
    SET cr = (SELECT AVG(nota) FROM Historico      WHERE
a.mat = NovaTupla.mat)
    WHERE a.mat = NovaTupla.mat;
...
END;
```

Gatilhos

- Opções:
 - AFTER / BEFORE / INSTEAD OF
 - INSERT / DELETE / UPDATE
 - WHEN (condição para disparo do gatilho)
 - BEGIN .. END; (+ de 1 expressão SQL)
 - OLD/NEW ROW AS (Atualização)
 - NEW ROW AS (Inserção)
 - OLD ROW AS (Remoção)
 - FOR EACH ROW / STATEMENT
 - OLD TABLE AS
 - NEW TABLE AS

Programação com Banco de Dados

- 2 alternativas:
 - Codificação utilizando uma biblioteca de acesso ao banco de dados
 - Exemplo: C, C++, ...
 - Utilizar os recursos existentes na linguagem (se for o caso) para acesso ao banco de dados
 - Java, PHP, ...

Procedimentos e Funções Armazenados (Stored Procedures)

- São módulos (procedimentos ou funções) pré-compilados e armazenados no SGBD; O termo padrão em SQL é PSM (*Persistent Stored Modules*)
- Permitem o encapsulamento de definições do esquema de um BD e melhoram o desempenho
- Dispondo de recursos de linguagem de programação (variáveis, condicionais, repetições, etc), estes módulos podem realizar diversas operações para uma única chamada
- Estes procedimentos podem conter parâmetros com tipos de SQL e modos de entrada (IN), saída (OUT) e entrada juntamente com saída (INOUT)

Stored Procedures

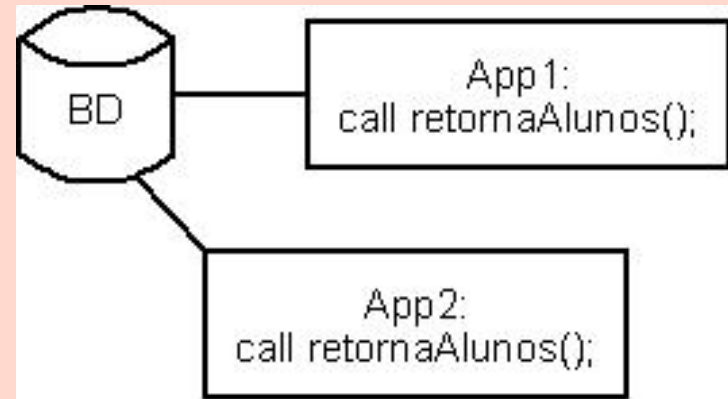
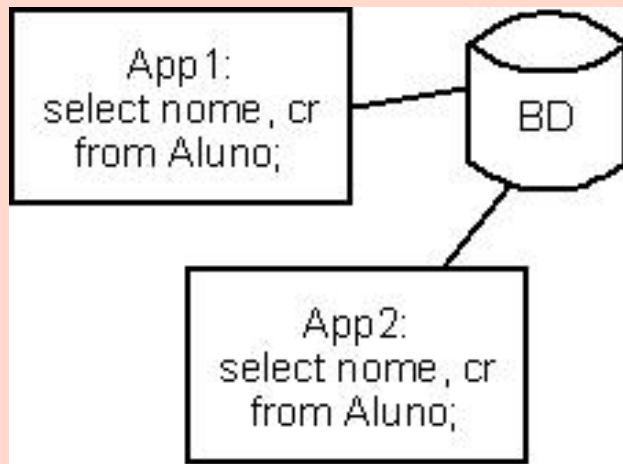
Exemplo

```
CREATE PROCEDURE Bonifica (  
    IN n FLOAT,  
    IN m VARCHAR (20)  
)  
    IF n <= 1 (  
        UPDATE Aluno  
        SET nota = nota + n  
        WHERE mat = m; );
```

Stored Procedures

Situações Úteis

- Se um BD precisa ser chamado em vários aplicativos, estes módulos podem ser armazenados no servidor
- Do contrário, teríamos instruções SQL repetidas entre estes aplicativos
- Observe os diagramas abaixo sem e com stored procedures e imagine a modificação do esquema da tabela Aluno



Transações

- São uma unidade lógica de trabalho
- Necessárias quando se deseja que operações sejam executadas como uma única, de forma indivisível
 - Por exemplo, uma operação de transferência bancária pode ser traduzida como uma operação de débito e outra de crédito
- Transações se justificam pelo fato de um SGBD ser multiusuário e multitarefa
- Ou seja, vários processos pode estar executando sobre a mesma base de dados e inconsistências podem ocorrer

Transações

- Outra situação que justifica o uso de transações é na ocorrência de falhas
- O SGBD garante que, dada uma transação com n operações, ou as n operações são executadas ou nenhuma delas
- O estado consistente existente antes da execução de uma transação deve existir após a execução desta
- Com isso, numa falha (disco, quebra no sistema, exceção detectada pela transação, etc), o SGBD deve ser capaz de se recuperar da situação errônea

Transações – Pseudo-código

BEGIN TRANSACTION;

UPDATE ACC 123 { Saldo := Saldo - \$100 };

IF *ocorrer_algum_erro* GO TO UNDO; END IF;

UPDATE ACC 456 { Saldo := Saldo + \$100 };

IF *ocorrer_algum_erro* GO TO UNDO; END IF;

COMMIT; GO TO FINISH;

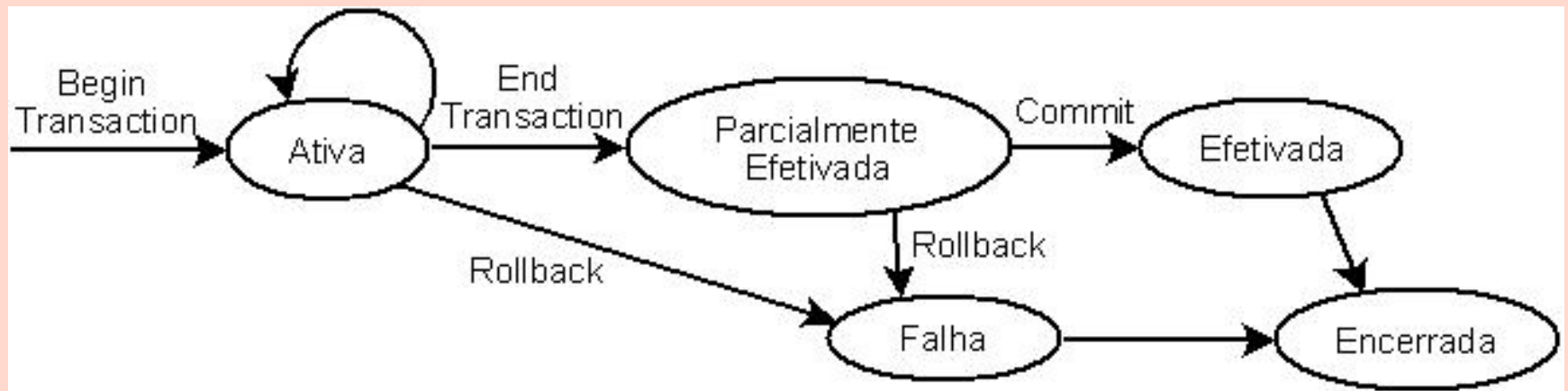
UNDO: ROLLBACK;

FINISH: RETURN;

Transações – Pontos Importantes

- COMMIT/ROLLBACK implícito
- Tratamento de mensagens
- Log de recuperação
- Atomicidade de instruções
- A execução de um programa é uma seqüência de transações
- Transações aninhadas
- Corretude
- Múltiplas associações

Transações



Transações

- Tipos de transações
 - Read-only
 - Writable
- Propriedades de transações (ACID)
 - Atomicidade: todas operações ou nenhuma
 - Correção: somente geram estados válidos
 - Isolamento: atualizações percebidas entre diferentes transações somente após o *commit*
 - Durabilidade: Atualizações depois do *commit* persistem após possível queda do sistema

Transações

- Recuperação do sistema
 - Buffers de BDs
 - BDs físicos
 - Checkpoints

Consultas vistas anteriormente

```
select th.nome, tm.nome  
from TabHomem as th, TabMulher as tm  
where th.mulher = tm.id;
```

Retornar os nomes dos casais

Consultas vistas anteriormente

```
select distinct tm.cidade  
from TabHomem as th, TabMulher as tm  
where th.mulher = tm.id AND  
tm.nome like '%Brunet%'  
order by tm.cidade desc;
```

Retornar os nomes das cidades em ordem decrescente que possuem casais cuja mulher tem Brunet no nome

Consultas Complexas

Teste de Campos Nulos

```
select nome  
from TabHomem as t  
where t.mulher is NULL;
```

Retornar os nomes dos homens que não são casados

Consultas Complexas

Teste de Pertinência

```
select tm.nome, tm.tel  
from TabMulher as tm  
where tm.cidade in ('Petropolis',  
'Teresopolis', 'Friburgo');
```

*Retornar os nomes e telefones das
mulheres que moram na região serrana*

Consultas Complexas

Consultas Aninhadas

```
select nome  
from TabHomem  
where nome in (  
    select th.nome  
    from TabHomem as th, TabMulher as tm  
    where th.mulher = tm.id AND  
    th.cidade = tm.cidade );
```

Retornar os nomes dos homens que são casados e moram na mesma cidade que a sua mulher

Consultas Complexas

Consultas Aninhadas

```
select t.nome, t.tel  
from TabHomem as t  
where (t.nome, t.tel) in (  
    select th.nome, th.tel  
    from TabHomem as th, TabMulher as tm  
    where th.mulher = tm.id AND  
    th.cidade = tm.cidade );
```

Retornar os nomes e telefones dos homens que são casados e moram na mesma cidade que a sua mulher

Consultas Complexas

Teste de Existência

```
select th.nome, th.tel  
from TabHomem as th  
where exists (  
    select *  
    from TabMulher as tm  
    where th.cidade = tm.cidade );
```

Retornar os nomes e telefones dos homens moram na mesma cidade que alguma mulher

Consultas Complexas

Teste de Existência

```
select th.nome, th.tel  
from TabHomem as th  
where th.mulher is NULL AND  
      not exists (  
        select *  
        from TabMulher as tm  
        where th.cidade = tm.cidade );
```

Retornar os nomes e telefones dos homens que são solteiros e moram numa cidade que não tem mulheres

Consultas Complexas

Comparações Avançadas

```
select tm.nome, tm.tel  
from TabMulher as tm  
where tm.salario > ALL (  
    select th.salario  
    from TabHomem as th );
```

Retornar os nomes e telefones das mulheres que possuem salários maiores que todos os homens (mulheres independentes)

Consultas Complexas

Variações de Pertinência

```
select tm.nome, tm.tel  
from TabMulher as tm  
where not exists (  
    ( select th.id  
      from TabHomem as th  
      where th.cidade = tm.cidade )  
  except  
    ( select th.id  
      from TabHomem as th  
      where th.mulher is not Null ) );
```

Retornar os nomes e telefones das mulheres que moram numa cidade que não possua homens solteiros

Consultas Complexas

Variações de Consultas Aninhadas

```
select count(*)  
from TabMulher as tm  
where tm.salario >  
      ( select avg(th.salario)  
        from TabHomem as th );
```

Retornar a quantidade mulheres que possuem salário maior que a média masculina

Consultas Complexas

Agrupamento

```
select tm.cidade, count(*)  
from TabMulher as tm  
group by tm.cidade;
```

Retornar a cidade e a respectiva quantidade de mulheres

Consultas Complexas

Variações com Agrupamento

```
select tm.cidade, count(*)  
from TabMulher as tm  
where tm.salario >  
      ( select avg(th.salario)  
        from TabHomem as th )  
group by tm.cidade;
```

*Retornar a cidade e a quantidade das mulheres
que possuem salário maior que a média masculina*

Exercícios

- Retornar uma tabela que contenha o nome do professor, ano e a quantidade de disciplinas ministradas neste ano.
- Retornar uma tabela que contenha o nome do professor, ano, quantidade de alunos que cursaram as disciplinas e a quantidade de disciplinas ministradas neste ano.

Junção de Tabelas

```
select tm.nome, tm.tel  
from (TabMulher tm join TabHomem th on  
id=mulher)  
where tm.cidade='Petrópolis';
```

Retornar nome e telefone das mulheres casadas que moram em Petrópolis

Junção de Tabelas

Natural Join

```
select tm.nome, tm.tel  
from TabMulher tm natural join (TabHomem as  
th (idh, nome, tel, end, id))  
where tm.cidade='Petrópolis';
```

Retornar nome e telefone das mulheres casadas que moram em Petrópolis (renomeia a tabela TabHomem para que a junção natural possa ser executada)

Junção de Tabelas

Inner Join

```
select tm.nome, tm.tel  
from TabMulher as tm, TabHomem as th  
where tm.id=th.mulher;
```

*Retornar nome e telefone das mulheres
casadas que moram em Petrópolis
(Somente retorna a lista das casadas)*

Junção de Tabelas

Outer Join

```
select tm.nome, tm.tel  
from TabMulher as tm left outer join  
TabHomem as th on tm.id=th.mulher;
```

Retornar nome e telefone das mulheres casadas que moram em Petrópolis (Forçar retorno da lista de todas as mulheres)