Universidade Federal do Ceará
Centro de Ciências/Departamento de Computação
Código da Disciplina: CK0084 Ano: 2021
Professor: Ismayle de Sousa Santos

Aula 11

Sistemas de Informações e Banco de Dados

Cláusulas SQL e Funções de Agregação

IsmayleSantos
ismayle@ufc.br
@IsmayleSantos

Funções de Agregação

Funções de Agregação

- Funções de agregação são usadas para sumarizar informações de múltiplas tuplas em uma única-tupla sumário
- Existem várias funções de agregação como por exemplo:
  - Contagem de elementos (**COUNT**), o mínimo (**MIN**) e máximo (**MAX**) de uma série de valores, a soma (**SUM**) e a média (**AVG**) de um conjunto de dados
- Há ainda funções de agregação para cálculos estatísticos mais avançados
- Essas funções podem ser usadas em uma cláusula **SELECT**

Funções de Agregação

- Exemplo:
  - Recuperar a soma dos salários de todos os professores, o salário máximo, mínimo e médio

```
SELECT    SUM (Salario), MIN (Salario),
          MAX (Salario), AVG (Salario)
FROM      PROFESSOR;
```

  - Pode-se usar a palavra-chave AS para criar renomear os nomes das colunas resultantes da consulta

```
SELECT    SUM (Salario) AS TotalSal,
          MIN (Salario) AS MinSal,
          MAX (Salario) AS MaxSal,
          AVG (Salario) AS MedSal
FROM      PROFESSOR;
```

Funções de Agregação

- A função de agregação **COUNT** é responsável por trazer o total de ocorrências da coluna informada
- Exemplo:
  - Recuperar o número total de professores que trabalham no departamento 'Computacao'

```
SELECT  COUNT (*)
FROM    PROFESSOR AS P, DEPARTAMENTO AS D
WHERE   D.Codigo=P.DCodigo AND D.Nome='Computacao';
```

Cláusulas em SQL

Cláusulas Comuns

- Cláusulas são condições de modificação utilizadas para definir os dados que deseja selecionar ou modificar:
  - **FROM** – utilizada para especificar a tabela, que contém os campos listados na cláusula SELECT
  - **WHERE** – utilizada para especificar as condições que devem reunir os registros que serão selecionados
  - **GROUP BY** – utilizada para separar os registros selecionados em grupos específicos
  - **HAVING** – utilizada para expressar a condição que deve satisfazer cada grupo

Cláusulas Comuns

- ORDER BY – utilizada para ordenar os registros selecionados com uma ordem específica
- DISTINCT – utilizada para selecionar dados sem repetição
- UNION – combina os resultados de duas consultas SQL em uma única tabela para todas as linhas correspondentes

Estrutura Geral de uma Consulta SQL

- Em qualquer consulta SQL as cláusulas SELECT e FROM são obrigatórias
- As cláusulas WHERE, GROUP BY, ORDER BY, HAVING e LIMIT são opcionais, porém, caso presentes, devem ocorrer nessa ordem:

```
SELECT    <lista de atributos>
FROM      <lista de tabelas>
 |WHERE     <condições>
 |GROUP BY <atributos>
 |ORDER BY <atributos>
 |HAVING    <condições>
 |LIMIT     <número de linhas>;
```

Agrupamento: Cláusulas GROUP BY e HAVING

Cláusulas GROUP BY

- Muitas vezes deseja-se aplicar funções de agregação a subgrupos de tuplas em uma relação, onde esses subgrupos são baseados em alguns valores de atributos
  - Por exemplo, se desejarmos encontrar o valor médio da bolsa do aluno em cada curso
- Nesses casos precisa-se particionar as tabelas em conjuntos sem sobreposição (ou grupos) de tuplas
- Cada grupo consistirá das tuplas que têm o mesmo valor para um determinado atributo de agrupamento

Cláusulas GROUP BY

- Exemplo:
  - Para cada curso, recuperar o código do curso, o número de alunos no departamento e o valor médio da bolsa

Tabela: Aluno

	mat	nome	telefone	cursoAluno	valorBolsa
	Filtro	Filtro	Filtro	Filtro	Filtro
1	1	Marc...	859656...	1	350,00
2	2	Ana	598556...	1	350,00
3	3	Joana	895656...	2	400,00
4	4	Mich...	895632...	3	500,00
5	5	João	*NULL*	3	750,00
6	6	Carlos	*NULL*	3	550,00

```
SELECT cursoAluno, COUNT (*), AVG (valorBolsa)
FROM Aluno
GROUP BY cursoAluno;
```

	cursoAluno	COUNT (*)	AVG (valorBolsa)
1	1	2	350.0
2	2	1	400.0
3	3	3	600.0

Cláusulas HAVING

- Algumas vezes deseja-se recuperar valores para funções de agregação apenas para grupos que satisfazem determinada condição
  - Isso pode ser feito pela cláusula **HAVING**
  - A cláusula HAVING determina uma condição de busca para um grupo ou um conjunto de registros, definindo critérios para limitar os resultados obtidos a partir do agrupamento de registros
  - **É importante lembrar** que essa cláusula só pode ser usada em parceria com GROUP BY

Cláusulas HAVING

- Exemplo:
  - Para cada curso, no qual dois ou mais alunos estudam, recuperar o código do curso, o nome do curso e o número de alunos

Tabela: Aluno

	mat	nome	telefone	cursoAluno	valorBolsa
	Filtro	Filtro	Filtro	Filtro	Filtro
1	1	Marc...	859656...	1	350,00
2	2	Ana	598556...	1	350,00
3	3	Joana	895656...	2	400,00
4	4	Mich...	895632...	3	500,00
5	5	João	*NULL*	3	750,00
6	6	Carlos	*NULL*	3	550,00

```
SELECT cursoAluno, nomeCurso, COUNT (*)
FROM Aluno, Curso
WHERE Curso.id = Aluno.cursoAluno
GROUP BY cursoAluno
HAVING COUNT(*) >= 2
```

	cursoAluno	nomeCurso	COUNT (*)
1	1	Computação	2
2	3	Matemática	3

Cláusula ORDER BY

Ordem dos Resultados da Consulta

- A SQL permite que o usuário ordene as tuplas no resultado de uma consulta pelos valores de um ou mais dos atributos que aparecem, usando a cláusula **ORDER BY**
  - A ordem padrão é a **ordem ascendente** dos valores
  - Podemos especificar a palavra chave **DESC** se quisermos ver o resultado em uma ordem decrescente de valores
  - A palavra-chave **ASC** pode ser usada para especificar a ordem crescente explicitamente

Ordem dos Resultados da Consulta

- Exemplo:
  - Recuperar uma lista dos Alunos ordenado pelo nome

```
SELECT * FROM ALUNO ORDER BY nome;
```

mat	nome	telefone	cursoAluno	valorBolsa
2	Ana	5985569623	1	350,00
6	Carlos	*NULL*	3	550,00
3	Joana	8956563212	2	400,00
5	João	*NULL*	3	750,00
1	Marcelo	8596563323	1	350,00
4	Michelle	8956325896	3	500,00

```
SELECT * FROM ALUNO ORDER BY nome DESC;
```

mat	nome	telefone	cursoAluno	valorBolsa
4	Michelle	8956325896	3	500,00
1	Marcelo	8596563323	1	350,00
5	João	*NULL*	3	750,00
3	Joana	8956563212	2	400,00
6	Carlos	*NULL*	3	550,00
2	Ana	5985569623	1	350,00

Cláusula DISTINCT

Tabelas como Conjuntos em SQL

- A SQL normalmente trata uma tabela não como um conjunto, mas como um multiconjunto
  - Tuplas duplicadas podem aparecer mais de uma vez em uma tabela, e no resultado de uma consulta
- A SQL não elimina automaticamente tuplas duplicadas:
  - A eliminação de duplicatas é uma operação dispendiosa
  - O usuário pode querer ver as tuplas duplicadas no resultado de uma consulta
  - Queremos, na maioria das vezes, ver duplicatas em função agregada a tuplas

Tabelas como Conjuntos em SQL

- Exemplo:
  - Recuperar os valores de bolsas

```
SELECT DISTINCT valorBolsa  FROM Aluno;
```

	valorBolsa
1	350,00
2	400,00
3	500,00
4	750,00
5	550,00

- A palavra-chave DISTINCT na cláusula SELECT, significando que apenas as tuplas distintas deverão permanecer no resultado

Cláusulas UNION, EXCEPT e INTERSECT

Tabelas como Conjuntos em SQL

- Existem operações de:
  - **UNION** - união de conjunto (A ∪ B)
  - **EXCEPT** - diferença de conjunto (A - B)
  - **INTERSECT** - interseção de conjunto (A ∩ B)
- As relações resultantes dessas operações de conjunto são conjuntos de tuplas
  - Tuplas duplicadas são eliminadas do resultado
- Essas operações de conjunto se aplicam apenas a relações compatíveis com união em que as duas relações (da operação) tenham os mesmos atributos e que os atributos apareçam na mesma ordem no SELECT

UNION

- Exemplo:
  - Listar todos os contratados e dependentes de uma empresa
    - **SELECT** nome,cpf **FROM** Contratado **UNION SELECT** nome,cpf **FROM** Dependente;

cpf	nome
Filtro	Filtro
123	Ana Maria
234	Rafael
456	Amanda

Tabela Contratado

cpf	nome	cpfContratado
Filtro	Filtro	Filtro
890	Carlos	123
415	Gabriel	123
767	José	234
455	Rafael	123

Tabela Dependente

	nome	cpf
1	Amanda	456
2	Ana Maria	123
3	Carlos	890
4	Gabriel	415
5	José	767
6	Rafael	455
7	Rafael	234

EXCEPT

- Exemplo:
  - Listar Contratado que não tem dependente
  - SELECT Contratado.cpf FROM Contratado EXCEPT SELECT cpfContratado FROM Dependente;

cpf	nome
Filtro	Filtro
123	Ana Maria
234	Rafael
456	Amanda

Tabela Contratado

cpf	nome	cpfContratado
Filtro	Filtro	Filtro
890	Carlos	123
415	Gabriel	123
767	José	234
455	Rafael	123

Tabela Dependente

cpf
456

INTERSECT

- Exemplo:
  - Listar os nomes dos dependentes que tem o mesmo nome do contratado
    - SELECT Contratado.nome FROM Contratado INTERSECT SELECT nome FROM Dependente;

cpf	nome
Filtro	Filtro
123	Ana Maria
234	Rafael
456	Amanda

Tabela Contratado

cpf	nome	cpfContratado
Filtro	Filtro	Filtro
890	Carlos	123
415	Gabriel	123
767	José	234
455	Rafael	123

Tabela Dependente

nome
Rafael

Cláusulas Join

Métodos de Junção - Join

- Associações de tabelas podem ser utilizadas para diversas finalidades, como converter em informação os dados encontrados em duas ou mais tabelas
- Essa operação pode ser feito por meio das cláusulas JOIN
  - As tabelas podem ser combinadas por meio de uma condição ou um grupo de condições de junção
    - Deve haver duas tabelas em que haja algum relacionamento para "cruzar" os dados
      - Por exemplo, usar as chaves estrangeiras como condição para relacionar as tabelas

Join

- A cláusula JOIN permite que os dados de várias tabelas sejam combinados com base na relação existente entre elas
  - A cláusula JOIN é usada quando se quer recuperar dados em mais de uma tabela através da igualdade de suas foreign keys
- Os valores pertencentes às colunas das tabelas associadas podem ser comparados entre si por meio de um operador lógico definido pela cláusula JOIN e usada pelo operador ON, como o sinal de igual (=)

Exemplo Join

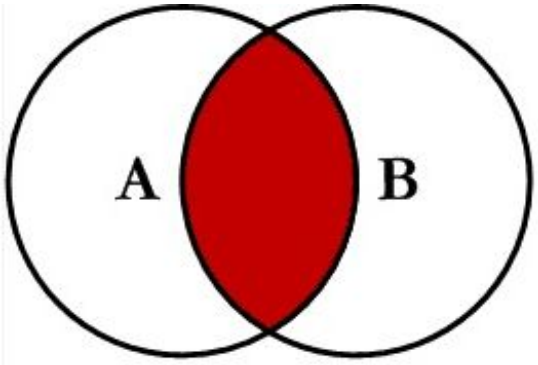
- A forma mais indicada para a especificação de associações é a cláusula FROM, que permite que as condições JOIN sejam identificadas em relação às condições de busca referenciadas na cláusula WHERE

```
1  SELECT *
2  FROM CLIENTES AS C
3  JOIN PEDIDOS AS P ON C.IDCLIENTE = P.IDPEDIDO
```

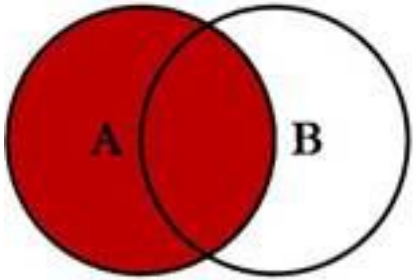
INNER JOIN

Inner Join

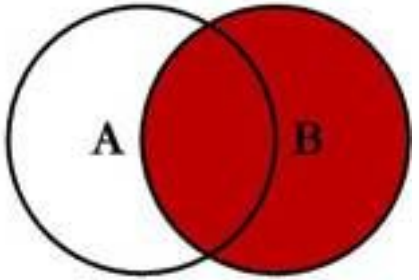
- O Inner Join é o método de junção mais conhecido e, retorna os registros que são comuns às duas tabelas
- A cláusula **INNER JOIN** permite usar um operador de comparação para comparar os valores de colunas associadas
- Usamos então a cláusula **INNER JOIN** para obtermos os dados relacionados das duas tabelas



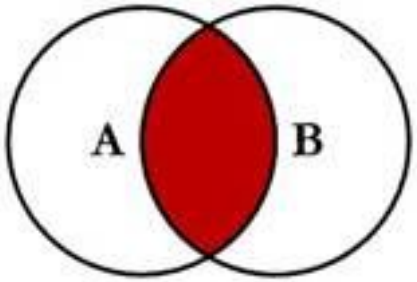
SQL JOINS



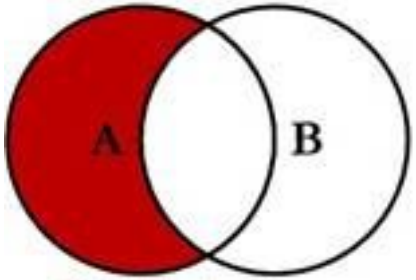

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```



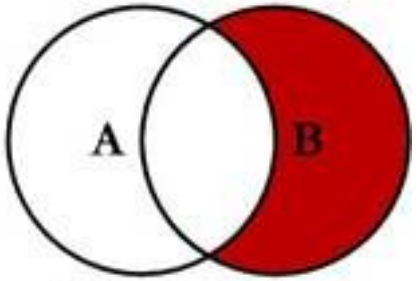

```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```



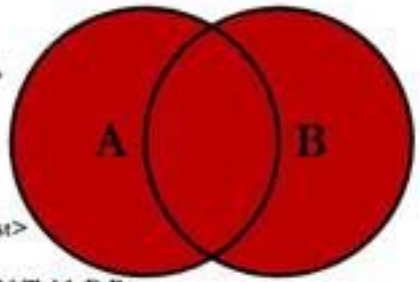

```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```



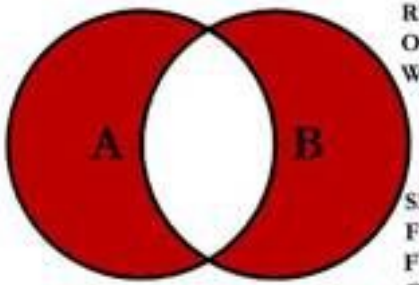

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```




```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
```




```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```




```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
OR B.Key IS NULL
```

Revisão dos Conceitos

Revisão dos Conceitos de BD

- Estrutura Relacional
- Modelo Conceitual
  - Diagrama Entidade Relacionamento
    - Entidade
    - Atributo
    - Relacionamento
- Modelo Lógico
  - Tabelas, Tuplas
- Modelo Físico
  - Esquema interno
- SQL
  - Integridade Referencial
  - Java + SQL
  - Cláusulas

Lembrando... Regras para Mapeamentos de Entidades

1. Toda entidade vira uma relação (Tabela)
2. Atributo identificador se torna chave primária na relação
3. Atributos simples se tornam colunas (campos)
4. Atributos compostos (e.g. *endereço*) **tornam-se atributos simples**, mapeados em colunas, uma coluna para cada atributo
5. Atributos derivados (e.g. *idade*) **não** são mapeados
6. Atributos multivalorados (e..g, *telefone*) podem ser mapeados de duas formas:
   - Como **n colunas**, onde n é o número máximo de valores do atributo
   - Criando-se **uma nova relação**

Obrigado!

Por hoje é só pessoal...

Dúvidas?


IsmayleSantos


ismayle@.ufc.br


@IsmayleSantos