UDESC

Banco de Dados I

Prof. Diego Buchinger
diego.buchinger@outlook.com
diego.buchinger@udesc.br

Profa. Rebeca Schroeder Freitas
Prof. Fabiano Baldo

PostgreSQL

SQL – Structured Query Language

Microsoft® SQL Server®

UDESC

SQL – O que é

- Utiliza uma base Formal combinando álgebra relacional e cálculo relacional

- Linguagem comercial para BD relacional
  - Desenvolvida pela IBM no início dos anos 70
  - Tornou-se padrão ISO desde a década de 80
    - SQL-1 (86), SQL-2 (92), SQL-3 (99)

- Se estabeleceu como a linguagem padrão para banco de dados relacionais

UDESC

SQL – Do que é composta

Apesar de ser chamada de “linguagem de consulta” possui outras funcionalidades também:

- **Linguagem de Definição de Dados (DDL)**
  - Comandos para definição, remoção e modificação de tabelas, índices, chaves *etc*.
- **Linguagem de Manipulação de Dados (DML)**
  - Comandos de consulta, inserção, exclusão e modificação de dados
- **DML Embutida:**
  - Projetada para utilização em linguagens de programação de uso geral (Cobol, C, JAVA *etc*.)

UDESC

SQL – Do que é composta

- **Definição de Visões:** comandos para a definição de visões
- **Autenticação:** comandos para especificação de autorização de acesso as tabelas e as visões
- **Integridade:** comandos para especificação de regras de integridade
- **Controle de Transações:** comandos para especificação de início e fim de transações, e bloqueios de dados para controle de concorrência

Qual tecnologia vamos usar?

- Vamos utilizar o **Postgre SQL [**ou **pgSQL**]

- Vamos acessar o SGBD através de um programa: **pgAdmin**

  Mas ... Quem quiser pode usar o modo terminal /prompt
  SQL Shell (psql)

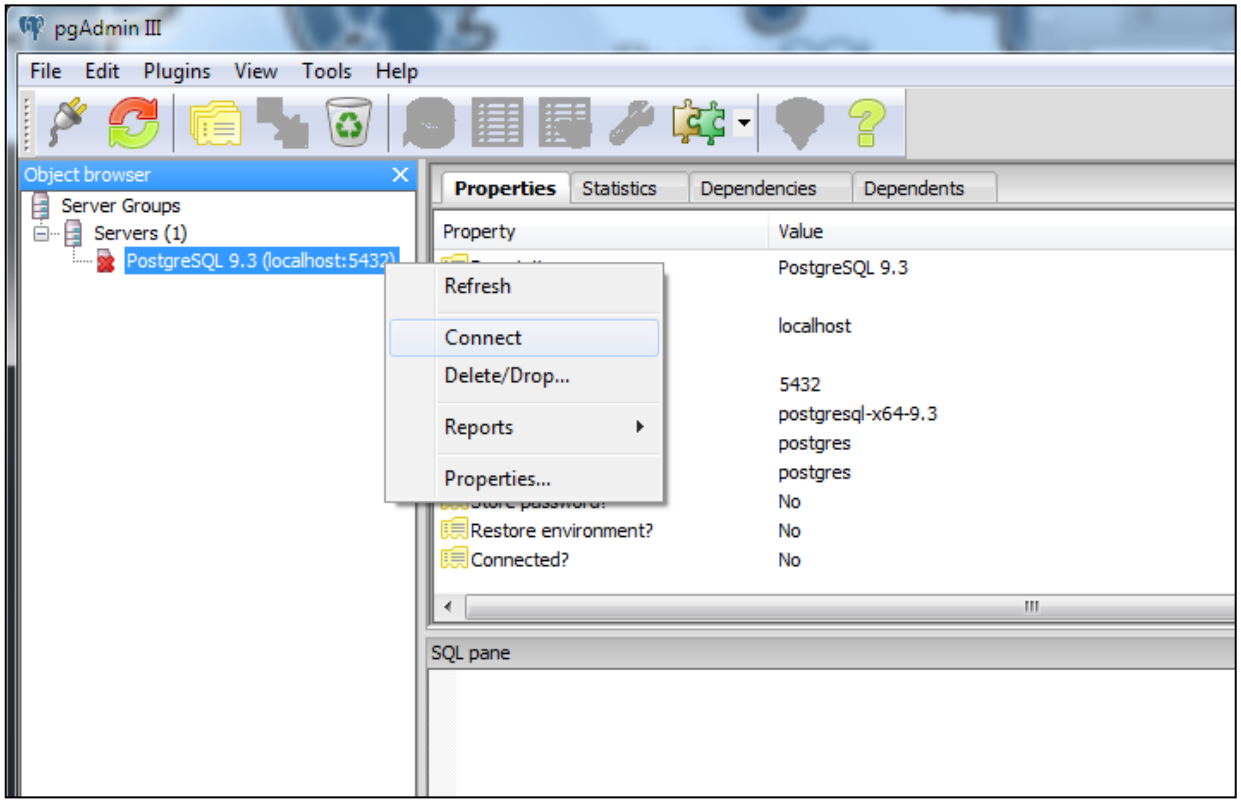
  – Qual é a senha? udesc

- Mais para frente, vamos acessar o banco de dados com a linguagem de programação JAVA

PostgreSQL

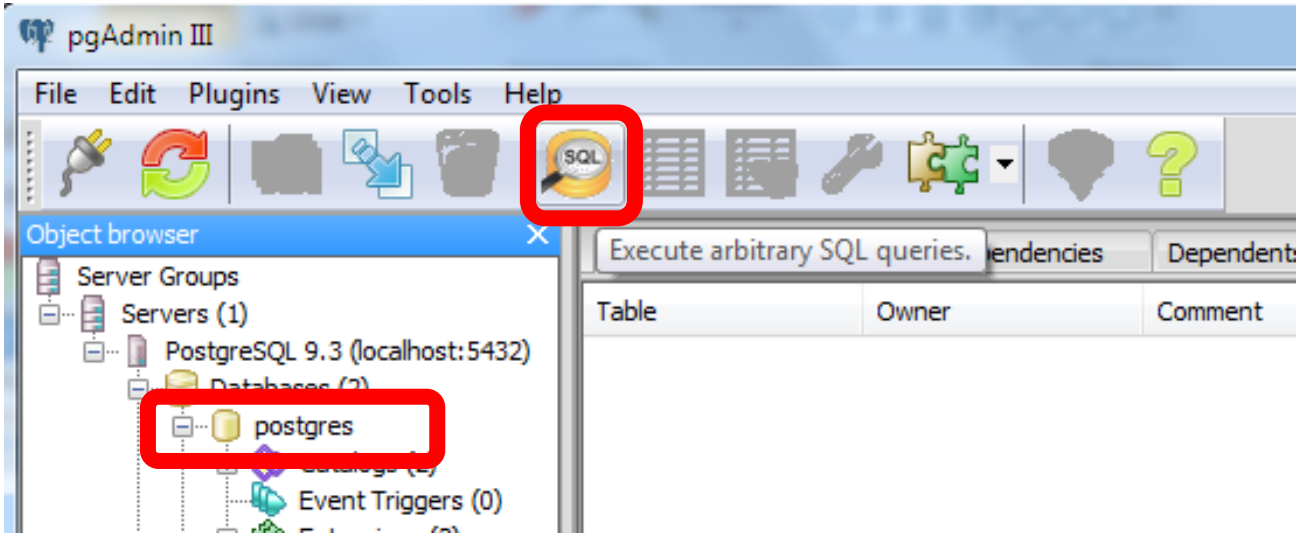
Qual tecnologia vamos usar?

- Instruções iniciais:
  - Conectar ao BD



Qual tecnologia vamos usar?

- Instruções iniciais:
  - Vamos escrever e executar scripts SQL na área própria: Selecione a base de dados que deseja utilizar e clique no botão destacado na figura



OBS: a conexão com o banco é feita sobre uma base de dados. Para executar scripts sobre outra base é necessário fazer nova conexão

UDESC

DDL – *Data Definition Language*

Mantendo Bases e Esquemas

- Especifica uma nova base de dados

```
CREATE DATABASE exemplo
    [WITH OWNER = postgres]
      [ENCODING = 'UTF8'];
```

[...] = opcional

- Especifica um novo esquema de tabelas (em uma base)

```
CREATE SCHEMA teste
       AUTHORIZATION nome;
```

- Exclui uma base de dados: `DROP DATABASE exemplo;`
- Exclui uma base de dados: `DROP SCHEMA teste;`

Criando Tabelas

- Para definir uma nova tabela e sua estrutura:

```
CREATE TABLE nome_tabela(
    nome_atributo_1 tipo_1 [*RESTRIÇÕES*],
    [{nome_atributo_n tipo_n [*RESTRIÇÕES*]},]
    [PRIMARY KEY (nome(s)_atributo(s)),]
    [FOREIGN KEY (nome_atributo)
       REFERENCES nome_tabela (nome_atributo)]
  );
```

Criando Tabelas

Exemplo

```
CREATE TABLE Cursos(
    id INT,
    nome VARCHAR(30) NOT NULL,
    PRIMARY KEY (id)
);
```

```
CREATE TABLE Alunos(
    matricula INT,
    nome VARCHAR(50) NOT NULL,
    curso_id INT,
    PRIMARY KEY (matricula),
    FOREIGN KEY (curso_id)
        REFERENCES Cursos (id)
  );
```

UDESC

Criando Tabelas

- Principais tipos de dados:
  - Numéricos: englobam números inteiros de vários tamanhos e pontos flutuantes (ver tabela a seguir)
  - Cadeia de Caracteres: podem ser de tamanho fixo [`char(n)`] ou variável [`varchar(n)`, `text`]
  - Booleano: assume valores TRUE e FALSO, mas pode ser também um valor NULL
  - Date: formato `aaaa-mm-dd`
  - Time: formato `hh:mm:ss`
  - Timestamp: formado por data e hora

UDESC

Criando Tabelas

- Detalhes sobre dados numéricos:

nome	tamanho armazenamento	faixa de valores
tinyint	1 bytes	-128 a 127
smallint	2 bytes	-32.768 a +32.767
int	4 bytes	-2.147.483.648 a +2.147.483.647
bigint	8 bytes	-9.223.372.036.854.775.808 a ...
decimal	Variável	Sem limite
numeric	Variável	Sem limite
real	4 bytes	Precisão de 6 dígitos decimais
double precision	8 bytes	Precisão de 15 dígitos decimais

Criando Tabelas

- É possível utilizar atributos opcionais para dados numéricos:
  - `UNSIGNED`: indica que um tipo numérico não terá sinal e por consequência a faixa de valores é alterada
  - `ZEROFILL`: indica que os espaços vazios do campo serão preenchidos com zeros [PostgreSQL não implementa]
  - `(n)`: É possível especificar o "tamanho de apresentação", ex: `int`(5) => 00032 (OBS: faixa de valores não muda)
  - `(i,d)`: para os tipos `decimal` e `numeric` é possível especificar o número de dígitos totais e decimais, ex: `decimal`(6,2) => -9.999,99 até +9.999,99

Criando Tabelas

- O “tamanho de representação” para dados numéricos não é o número de bits utilizados para representar o número!

MySQL Docs, Numeric Type Attributes

MySQL supports an extension for optionally specifying the display width of integer data types in parentheses following the base keyword for the type. For example, **INT(4)** specifies an INT with a **display width of four digits**. This optional display width **may be used by applications to display** integer **values** having a width less than the width specified for the column **by left-padding them with spaces.** (That is, this width is present in the metadata returned with result sets. Whether it is used or not is up to the application.)

Criando Tabelas

– Cada SGBD possui as suas peculiaridades, podendo ter notações ou tipos de dados diferentes (específicos)

PostgreSQL disponibiliza tipos de dados para armazenamento de endereços IPv4, IPv6 e MAC:

`cidr` e `inet` – 12 ou 24 bytes
`macaddr` – 6 bytes

PostgreSQL não implementa o tamanho de apresentação para atributos numéricos

(…)

Criando Tabelas

- Restrições / Atributos Opcionais
  - `NOT NULL`: indica que o campo não pode ser nulo, ou seja, precisa necessariamente ter um valor
  - `UNIQUE`: indica que um campo deve ser único para cada registro / linha da tabela (não pode haver duplicatas
  - `AUTO_INCREMENT`: indica que será utilizado um valor incrementável automaticamente para o campo

    OBS: no PostgreSQL não há está restrição, mas sim um tipo de dado serial que possui este comportamento

https://www.postgresql.org/docs/9.4/static/ddl-constraints.html

Criando Tabelas

- Restrições / Atributos Opcionais
  - `DEFAULT valor`: indica um valor padrão fixo para o campo caso um valor não seja especificado
  - `CHECK (predicado)`: indica que deverá ser realizar verificação de integridade de acordo com o predicado;
  - `PRIMARY KEY`: indica que um campo é chave primária;
  - `REFERENCES tabela (campo)`: indica que um campo é chave estrangeira

OBS: estas restrições podem ser definidas também no final

https://www.postgresql.org/docs/9.4/static/ddl-constraints.html

Criando Tabelas

- Exemplos com Restrições

```
CREATE TABLE Materiais(
     id SERIAL PRIMARY KEY,
     nome VARCHAR(50) UNIQUE NOT NULL,
     preco NUMERIC(6,2)
);
```

```
CREATE TABLE Disciplinas(
       sigla CHAR(3) PRIMARY KEY,
       curso_id INT REFERENCES cursos (id),
       ch SMALLINT
);
```

UDESC

Criando Tabelas

- Exemplos com Restrições (forma alternativa)

```
CREATE TABLE Materiais(
     id SERIAL,
     nome VARCHAR(50) NOT NULL,
     preco NUMERIC(6,2) CHECK (preco > 0),
     preco_vista NUMERIC(6,2),
     PRIMARY KEY (id),
--   CONSTRAINT pk PRIMARY KEY (id),
     UNIQUE (nome),
--   CONSTRAINT nome_diferente UNIQUE (nome),
     CHECK (preco_vista <= preco)
--   CONSTRAINT avista CHECK (preco_vista <= preco)
);
```

Criando Tabelas

- Exemplos com Restrições (forma alternativa)

```
CREATE TABLE Disciplinas(
        sigla CHAR(3),
        curso_id INT,
        ch SMALLINT,
        PRIMARY KEY (sigla),
        CONSTRAINT fk FOREIGN KEY (curso_id)
           REFERENCES cursos (id)
);
```

➢ Múltiplas chaves primárias, ou campos únicos podem ser definidos separando os nomes das colunas com vírgula

Criando Tabelas

- Para as `FOREIGN KEYS`, pode-se especificar o que deve ser feito caso uma operação sobre suas referencias seja realizada:
  - Definir qual a operação considerada:
    - `ON DELETE ; ON UPDATE`
  - Definir o comportamento:
    - NO ACTION: padrão, previne a alteração do registro se este estiver referenciado por outro registro.
    - RESTRICT: similar ao NO ACTION
    - SET NULL / SET DEFAULT: altera os registros que estiverem o referenciando para um valor nulo ou padrão;
    - CASCADE: propaga a alteração ou remoção aos registros que o estiverem referenciando

Criando Tabelas

- ***Storage Engine*** **ou** ***Engine****:* são componentes do SGBD que são responsáveis pela realização das operações SQL

- PostgreSQL suporta e utiliza apenas um único e*ngine* que recebe o seu nome: PostgreSQL
- MySQL suporta e possibilita o uso de diversos *engines*, entre eles: InnoDB, MyISAM, MEMORY, CSV, ARCHIVE, BLACKHOLE, MERGE, FEDERATED, EXAMPLE

```
CREATE TABLE Disciplinas(
      [...]
) engine=MyISAM;
```

```
CREATE TABLE Disciplinas(
      [...]
) engine=InnoDB;
```

UDESC

Criando Tabelas

- No SQL os *engines* mais comuns são MyISAM e InnoDB:
  - *MyISAM*: modelo simples e eficiente
    - ✓ **contras**: não garante restrições de integridade referenciais
    - ✓ **prós**: simples e bom desempenho;
      bom para muitas leituras e poucas escritas;
      bom quando o controle de restrições está no código
  - *InnoDB:* modelo mais robusto
    - ✓ **prós**: garante restrições de integridade referenciais;
      bom para escritas concorrentes;
    - ✓ **contras**: um pouco mais lento do que o MyISAM

Apagando Tabelas

- Para excluir uma tabela existente pode-se utilizar:

```
DROP TABLE nome_tabela;
```

Alterando Tabelas

- Existem diversas alterações possíveis que podem ser feitas em uma tabela já existente:

```
ALTER TABLE nome_tabela
  RENAME TO novo_nome
  SET SCHEMA new_schema
  ADD [COLUMN] nome tipo [{Ris}]
  RENAME [COLUMN] nome_antigo TO nome_novo
  DROP [COLUMN] nome
  ALTER [COLUMN] nome [{SET|DROP} DEFAULT
      | {SET|DROP} NOT NULL | SET DATA TYPE tipo|...]
  ADD CONSTRAINT ri [PK|FK|CHECK|...]
  DROP CONSTRAINT nome_ri
  ...
```

https://www.postgresql.org/docs/9.1/static/sql-altertable.html

Alterando Tabelas

- Alguns exemplos

```
ALTER TABLE alunos
  RENAME TO graduandos

ALTER TABLE graduandos
  SET SCHEMA public

ALTER TABLE graduandos
  ADD COLUMN dt_nasc
  DATE NOT NULL

ALTER TABLE graduandos
  RENAME COLUMN dt_nasc
  TO data_nascimento
```

```
ALTER TABLE graduandos
  DROP COLUMN data_nascimento

ALTER TABLE graduandos
  ADD CONSTRAINT nroMat
  CHECK (matricula > 10000)

ALTER TABLE graduandos
  DROP CONSTRAINT nroMat

ALTER TABLE disciplinas
ALTER COLUMN ch
  SET DEFAULT (32)
```

Criando Índices

- **Índices** são utilizados para acelerar consultas a dados (assim como ocorre em um livro). Entretanto, índices ocupam espaço e devem ser utilizados com parcimônia
  - Índices são definidos automaticamente para PKs

```
CREATE [UNIQUE] INDEX nome_indice
  ON nome_tabela (nome_atrib [{, nome_atrib_n}])

DROP INDEX nome_indice ON nome_tabela
```

UDESC

Criando Sequencias

- O PostgreSQL utiliza **sequências** para valores do tipo serial (auto incrementáveis). As sequências podem ser alteradas com os seguintes comandos

```
ALTER SEQUENCE nome_sequencia
  INCREMENT BY valor -- escolhe o valor de inc.
  RESTART WITH valor -- reinicia com um valor
  MINVALUE valor -- define o valor mínimo
  MAXVALUE valor -- define o valor máximo
```

Atividade

1. Crie um BD com o nome Clinica
2. Crie as seguintes tabelas neste BD:
   - **Ambulatorios:** #nroa (int), andar (numeric(3), não nulo)
   - **Médicos:** #codm (int), nome (varchar(50), não nulo), idade (smallint, não nulo), especialidade (char(20)), CPF (numeric(11), único), cidade (varchar(30)), nroa (int)
   - **Pacientes:** #codp (int), nome (varchar(40), não nulo), idade (smallint), CPF (numeric(11), único), doenca (varchar(40), não nulo)
   - **Funcionarios**: #codf (int), nome (varchar(40), não nulo), idade (smallint), CPF (numeric(11), único), cidade (varchar(30)), salario (numeric(10)), cargo (varchar(20))
   - **Consultas**: #&*codm* (int), &*codp* (int), #data (date), #hora (time),

Guarde/salve os comandos SQL. Vamos reutilizar esses dados!

Atividade

3. Altere a tabela **Ambulatorios** criando a coluna `capacidade (smallint)` e a tabela **Pacientes** criando a coluna `cidade (varchar(30));`
4. Altere a tabela **Funcionarios** removendo a coluna `cargo`
5. Altere a tabela **Medicos** criando uma FK para a coluna `nroa` que referencia a coluna `nroa` de **Ambulatórios**, e crie uma nova coluna `ativo (bool)` com valor padrão inicial verdadeiro (`true`)
6. O campo `doenca` foi mal interpretado e está no lugar errado. Corrija removendo este campo na tabela **Pacientes** e adicione-o na tabela **Consultas**
7. Crie uma restrição de integridade de verificação que garanta que não possa ser registrada uma consulta antes das 06:00 nem depois das 21:00.

Guarde/salve os comandos SQL.
Vamos reutilizar esses dados!

UDESC

DML – *Data Manipulation Language*
Parte I

UDESC

DML

- A **DML** define operações de manipulação de dados:
  - Inserção, atualização, exclusão e seleção

- As instruções são declarativas, ou seja, especificam o que deve ser feito mas não como fazer

- **ATENÇÃO:** deve-se ter muito cuidado com as operações de atualização e exclusão pois podem causar danos irreversíveis nos registros das tabelas!

Inserção de Registros

- Para inserir valores podem-se utilizar dois modelos:
  - ❑ Apresentando todos os valores explicitamente:

```
INSERT INTO nome_tabela
  VALUES ( valores_separados_por_virgula )
```

  - ❑ Identificando quais colunas estão sendo listadas (demais colunas recebem valor padrão ou valor nulo)

```
INSERT INTO nome_tabela ( lista_colunas )
  VALUES ( valores_na_mesma_ordem )
```

OBS: caso algum campo com restrição não nulo não apareça na operação, o SGBD acusará um erro

Inserção de Registros

- Exemplos:

```
INSERT INTO materiais
   VALUES (1, 'livro BAN', 75.00, 70.00);
```

```
INSERT INTO materiais (nome, preco, preco_vista)
  VALUES ('teste', 125.50, 100.00);
```

OBS: caso o campo de sequencia não tenha sido atualizado, a primeira tentativa resultará em erro pois o SGBD tentará utilizar id=1, mas este valor já foi utilizado

```
INSERT INTO materiais (nome, preco, preco_vista)
  VALUES ('mouse', 45.00, 39.99),
         ('teclado', 75.50, 65.00);
```

OBS: nem todo SGBD permite a inserção de múltiplos registros em uma única cláusula

Inserção de Registros

- Exemplos das Restrições de Integridade na inserção:

OBS: o campo nome é obrigatório (NOT NULL). Vamos adicionar um novo registro sem nome e ver o que acontece

```
INSERT INTO materiais (preco, preco_vista)
  VALUES (99.90, 95.00);
```

OBS: a coluna nome foi marcada como UNIQUE. Vamos adicionar um novo registro com o mesmo nome e ver o que acontece

```
INSERT INTO materiais (nome, preco, preco_vista)
  VALUES ('teste', 175.50, 150.00);
```

OBS: preco_vista tem uma restrição assertiva (check) que verifica se este valor é menor do que o preço. E se não for, o que acontece?

```
INSERT INTO materiais (nome, preco, preco_vista)
  VALUES ('caneta atômica', 4.50, 5.00);
```

Alteração/Atualização de Registros

- Para alterar / atualizar registros utiliza-se:

```
UPDATE nome_tabela
   SET nome_atributo = valor
       [{, nome_atributo_n = valor}]
   [WHERE condição]
```

CUIDADO: caso um condicional WHERE não for definido, todos os registros da tabela serão alterados!!!

- Exemplos:

```
UPDATE materiais
 SET valor_vista = 65.00
 WHERE id = 1;
```

```
UPDATE materiais
 SET nome = 'livro EDA',
  preco_vista = preco * 0.9
 WHERE id = 2;
```

Exclusão de Registros

- Para excluir registros utiliza-se:

```
DELETE FROM nome_tabela
   [WHERE condição]
```

CUIDADO: caso um condicional WHERE não for definido, todos os registros da tabela serão excluidos!!!

- Exemplos:

```
DELETE FROM Ambulatorios
```

```
DELETE FROM materiais
WHERE id = 1
```

```
DELETE FROM materiais
WHERE nome = 'livro EDA'
   OR nome = 'livro BAN'
```

Operadores SQL

- Para as expressões condicionais é possível utilizar os seguintes operadores:
  - Operadores de comparação

    `= ; <> ; != ; > ; >= ; < ; <=`
  - Operadores lógicos

    `and; or ; not`

Atividade

8. Popular as tabelas

Ambulatorios

nroa	andar	capacidade
1	1	30
2	1	50
3	2	40
4	2	25
5	2	55

Funcionarios

codf	nome	idade	cidade	salario	CPF
1	Rita	32	Sao Jose	1200	20000100000
2	Maria	55	Palhoca	1220	30000110000
3	Caio	45	Florianopolis	1100	41000100000
4	Carlos	44	Florianopolis	1200	51000110000
5	Paula	33	Florianopolis	2500	61000111000

Atividade

8. Popular as tabelas

Pacientes				
codp	**nome**	**idade**	**cidade**	**cpf**
1	Ana	20	Florianopolis	20000200000
2	Paulo	24	Palhoca	20000220000
3	Lucia	30	Biguacu	22000200000
4	Carlos	28	Joinville	11000110000
5	Denise	26	Joinville	30120120231
6	Marcos	42	Bombinhas	12347860123
7	Debora	19	Florianopolis	56312456478

UDESC

Atividade

Medicos

codm	nome	idade	especialidade	CPF	cidade	nroa
1	Joao	40	ortopedia	10000100000	Florianopolis	1
2	Maria	42	traumatologia	10000110000	Blumenau	2
3	Pedro	51	pediatria	11000100000	Sao Jose	2
4	Carlos	28	ortopedia	11000110000	Joinville	
5	Marcia	33	neurologia	11000111000	Biguacu	3

UDESC

Atividade

Consultas				
codm	codp	data	hora	doenca
1	1	2006 / 06 / 12	14:00	gripe
1	4	2006 / 06 / 13	10:00	tendinite
2	1	2006 / 06 / 13	09:00	fratura
2	2	2006 / 06 / 13	11:00	fratura
2	3	2006 / 06 / 14	14:00	traumatismo
2	4	2006 / 06 / 14	17:00	checkup
3	1	2006 / 06 / 19	18:00	gripe
3	3	2006 / 06 / 12	10:00	virose
3	4	2006 / 06 / 19	13:00	virose
4	4	2006 / 06 / 20	13:00	tendinite
4	4	2006 / 06 / 22	19:30	dengue

Atividade

9. Realize as seguintes atualizações no BD:
   a. O paciente Paulo mudou-se para Ilhota
   b. A consulta do médico 1 com o paciente 4 foi remarcada para às 12:00 do dia 4 de Julho de 2006
   c. A paciente Ana fez aniversário
   d. A consulta do médico Pedro (codm=3) com o paciente Carlos (codp=4) foi postergada para uma hora e meia depois
   e. O funcionário Carlos (codf=4) deixou a clínica
   f. As consultas marcadas após as 19 horas foram canceladas
   g. Os médicos que residem em Biguacu e Joinville foram transferidos para outra clínica. Registrar como inativos.

UDESC

DML – *Data Manipulation Language*
Parte II

Consultas Básicas

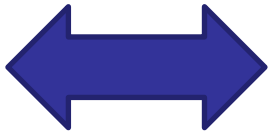
- Para consultar dados de uma tabela utiliza-se:

```
SELECT lista_de_colunas
FROM tabela
[WHERE condição]
```

  - *lista_de_colunas* pode ser substituida por asterisco (*) que representa todos os atributos da tabela

- Mapeamento entre SQL e álgebra relacional:

```
SELECT a1, ..., an
FROM t
WHERE c
```



$$\pi_{a_1,\ldots,an}(\sigma_c(t))$$

Consultas Básicas

- Exemplos:

Álgebra Relacional	SQL
$(Pacientes)$	`SELECT * FROM pacientes`
$\sigma_{idade>18}(pacientes)$	`SELECT * FROM pacientes WHERE idade > 18`
$\pi_{cpf,nome}(t)$	`SELECT cpf, nome FROM pacientes`
$\pi_{cpf,nome}(\sigma_{idade>18}(pacientes))$	`SELECT cpf, nome FROM pacientes WHERE idade > 18`

UDESC

Consultas Básicas
Particularidade na projeção

- Não há eliminação automática de duplicatas
  - Tabela ≡ coleção
  - Para eliminar duplicatas deve-se usar o termo `distinct`

```
SELECT DISTINCT doenca FROM consultas
```

- É possível renomear os campos / colunas (`AS`)
  - Operador ρ (*rho*) na álgebra relacional

```
SELECT codp AS codigo_paciente,
codm AS codigo_medico, data
FROM consultas
```

Consultas Básicas
Particularidade na projeção

- É possível utilizar operadores aritméticos e funções
  - Quantos grupos de 5 leitos podem ser formados em cada ambulatório?

```
SELECT nroa, capacidade/5 AS cap5
FROM ambulatorios
```

  - Qual o salário líquido dos funcionários sabendo que há um desconto de 12,63% sobre o salário base?

```
SELECT nome, ROUND( salario * 0.8737 , 2) AS
    salario_liquido FROM funcionarios
```

Função **ROUND**: parâmetros (valor : numeric , casas : int | numeric)

Consultas Básicas
Particularidade na projeção

- Pode-se usar funções de agregação / agrupamento
  - ➢ COUNT: contador de ocorrências [registros ou atributos]

OBS: Conta valores não nulos

```
SELECT COUNT(*) FROM medicos
WHERE especialidade = 'ortopedia'
```

  - ➢ MAX / MIN: valor máximo / mínimo de um atributo

```
SELECT MAX(salario) AS maior_salario FROM funcionarios
```

  - ➢ SUM: soma os valores de um dado atributo
    - ❖ Qual é o gasto total com a folha de pagamento dos funcionários?
  - ➢ AVG: contabiliza a média dos valores de um dado atributo
    - ❖ Qual é a média de idade dos funcionários de Florianópolis?

Consultas Básicas
Particularidade na projeção

- Pode-se misturar funções de agregação com `distinct`

```
SELECT COUNT(DISTINCT especialidade)
FROM medicos
```

- Não podem ser combinados outros campos junto com funções de agregação

```
SELECT andar, COUNT(andar)
FROM ambulatorios
```

- Pode-se realizar *casting* de tipos usando: `campo::tipo`

```
SELECT nome, cpf::text, idade::numeric(3,1)
FROM pacientes
```

UDESC

Consultas Básicas
Particularidade na seleção

- Procurar por valores nulos ou não nulos
  - cláusula `IS [NOT] NULL`

```
SELECT cpf, nome
FROM medicos WHERE nroa IS NULL
```

- Procurar por intervalos de valores
  - Cláusula `[NOT] BETWEEN valor1 AND valor2`

```
SELECT * FROM consultas
WHERE hora BETWEEN '13:00' AND '18:00'
```

Consultas Básicas
Particularidade na seleção

- Procurar por pertinência em conjunto ou coleção
  - cláusula `[NOT] IN`

```
SELECT * FROM medicos
WHERE especialidade IN
    ('ortopedia', 'traumatologia')
```

```
SELECT codm, codp, data FROM consultas
WHERE codm IN (
    SELECT codm FROM medicos
    WHERE idade > 40
)
```

Consultas Básicas
Particularidade na seleção

- Procurar por padrões
  - Cláusula `[NOT] LIKE`
  - Pode receber os seguintes padrões

    `%` casa com qualquer cadeia de caracteres

    _ casa com um único caractere

    [a-d] casa com qualquer caractere entre as letras apresentadas (SQL-Server)

❖ Buscar médicos que nome iniciando por ‘M’

❖ Buscar médicos com um número exato de décadas de vida

```
SELECT * FROM medicos
WHERE nome LIKE 'M%'
```

```
SELECT * FROM medicos
WHERE idade::text LIKE '_0'
```

Consultas Básicas
Particularidade na seleção

- Procurar por padrões

  ❖ Buscar por consultas marcadas para o mês de julho

```
SELECT * FROM consultas
WHERE data::text LIKE '%/07/%'
```

  ❖ Buscar por pacientes cujo CPF termina com 20000 ou 30000

```
SELECT * FROM pacientes
WHERE cpf::text LIKE '%20000'
   OR cpf::text LIKE '%30000'
```

União de Tabelas

- SQL implementa a união da álgebra relacional
  - Lembre-se, as tabelas devem ser compatíveis!

Álgebra Relacional

```
relação-1   ∪   relação-2
```

SQL

```
SQL-1   UNION   SQL-2
```

❖ Buscar o nome e o CPF dos médicos e funcionários

```
SELECT cpf, nome FROM medicos
  UNION
SELECT cpf, nome FROM pacientes
```

Atividade

1. Realize as seguintes consultas no BD:
   a. Buscar o nome e o CPF dos médicos com menos de 40 anos ou com especialidade diferente de traumatologia
   b. Buscar todos os dados das consultas marcadas no período da tarde após o dia 19/06/2006
   c. Buscar o nome e a idade dos pacientes que não residem em Florianópolis
   d. Buscar a hora das consultas marcadas antes do dia 14/06/2006 e depois do dia 20/06/2006
   e. Buscar o nome e a idade (em meses) dos pacientes
   f. Buscar o menor e o maior salário dos funcionários de Florianópolis

UDESC

Atividade

g. Qual o horário da última consulta marcada para o dia 13/06/2006?

h. Qual a média de idade dos médicos e o total de ambulatórios atendidos por eles?

i. Buscar o código, o nome e o salário líquido dos funcionários que recebem mais do que $ 1200. Para quem ganha acima de $1200, há um desconto de 12% do salário bruto

j. Buscar o nome dos funcionários que terminam com a letra ‘a’

k. Buscar o nome e idade dos funcionários que possuem o número 6 em seu CPF

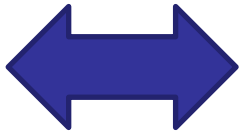
l. Em quais cidades residem os funcionários e médicos da clínica

Consultas Envolvendo Múltiplas Tabelas

- SQL implementa a operação de **PRODUTO CARTESIANO**
  - Relaciona todos os registros de uma tabela com todos os registros de outra tabela
  - Geralmente é necessário vincular as duas tabelas através de uma condição

- Mapeamento entre SQL e álgebra relacional:

```
SELECT a1, ..., an
FROM t1, ..., tm
WHERE c
```



$$\pi_{a_1,\ldots,an}(\sigma_c (t_1 \times \cdots \times t_m))$$

Consultas Envolvendo Múltiplas Tabelas

- Trazer o CPF e nome dos pacientes e suas respectivas datas de consultas agendadas para o período da tarde

```
SELECT cpf, nome, data
FROM pacientes, consultas
WHERE hora > '12:00' AND
  pacientes.codp = consultas.codp
```

- Trazer os nomes dos médicos que tem a mesma especialidade do médico de nome ‘João’
  [neste caso é necessário renomear uma tabela]

OBS: Quando as colunas são iguais é necessário usar o nome da tabela como prefixo. Assim, é comum renomear as tabelas com nomes mais enxutos

Consultas Envolvendo Múltiplas Tabelas

- SQL implementa as operações de **JUNÇÃO**

```
SELECT a1, ..., an
FROM t1 [INNER] JOIN t2
  ON condição_junção
WHERE c
```

❖ Trazer o CPF e nome dos pacientes e suas respectivas datas de consultas agendadas para o período da tarde

```
SELECT p.cpf, p.nome, c.data
FROM pacientes AS p JOIN consultas AS c
  ON p.codp = c.codp
WHERE hora > '12:00'
```

Consultas Envolvendo Múltiplas Tabelas

- **JUNÇÃO NATURAL**
  - Junção utilizando as colunas que possuem mesmo nome entre as tabelas relacionadas. Assim, a condição não é declarada

```
SELECT a1, ..., an
FROM t1 NATURAL JOIN t2
WHERE c
```

❖ Trazer o CPF e nome dos médicos e suas respectivas datas de consultas agendadas para o período da manhã

```
SELECT m.cpf, m.nome, c.data
FROM medicos AS m NATURAL JOIN consultas AS c
WHERE hora <= '12:00'
```

Consultas Envolvendo Múltiplas Tabelas

- **JUNÇÕES EXTERNAS**
  - Junções que mantêm os elementos (linhas) não relacionados de uma ou mais tabelas no resultado

```
SELECT a1, ..., an
FROM t1 LEFT|RIGHT|FULL [OUTER] JOIN t2
  ON condição_junção
WHERE c
```

❖ Listar todos os pacientes e, caso existam, as datas e horarios de suas consultas.

```
SELECT p.cpf, p.nome, c.data
FROM pacientes AS p LEFT JOIN consultas AS c
   ON p.codp = c.cod
```

Consultas Envolvendo Múltiplas Tabelas

- **JUNÇÕES EXTERNAS**

❖ Listar todos os médicos e funcionários (nome e cidade), vinculando aqueles que moram em uma mesma cidade

```
SELECT f.nome, f.cidade, m.nome, m.cidade
FROM funcionarios AS f FULL JOIN medicos AS m
   ON f.cidade = m.cidade
```

Consultas Aninhadas

- Já vimos que é possível realizar consultas alinhadas usando a cláusula [NOT] IN

```
SELECT codm, codp, data FROM consultas
WHERE codm IN (
    SELECT codm FROM medicos
    WHERE idade > 40
)
```

  - É possível realizar operações de diferença e interseção

$\pi_{CPF}(Funcionarios) - \pi_{CPF}(Pacientes)$

```
SELECT cpf FROM funcionarios
WHERE cpf NOT IN (
    SELECT cpf FROM pacientes
)
```

$\pi_{CPF}(Medicos) \cap \pi_{CPF}(Pacientes)$

```
SELECT cpf FROM medicos
WHERE cpf NOT IN (
    SELECT cpf
    FROM pacientes
)
```

Consultas Aninhadas

- Pode-se fazer uso também de **Subconsultas unitárias**
  - Cardinalidade da subconsulta = 1
  - Neste caso não é necessário utilizar cláusula de subconsulta

❖Buscar o nome e CPf dos médicos que possuem a mesma especialidade do que o médico de CPF 10000100000

```
SELECT nome, cpf FROM medicos
WHERE cpf != 10000100000 AND
   especialidade = (
      SELECT especialidade FROM medicos
      WHERE cpf = 10000100000
)
```

Consultas Aninhadas

- Existem ainda as cláusulas ANY , ALL e EXISTS (Cálculo Relacional)

  – **ANY**: testa se uma dada condição é verdadeira para pelo menos um valor da consulta aninhada

  ❖Buscar o nome e a idade dos médicos que são mais velhos do que pelo menos um funcionário

```
SELECT nome, idade FROM medicos
WHERE idade > ANY (
    SELECT idade FROM funcionarios
)
```

Consultas Aninhadas

- Existem ainda as cláusulas ANY , ALL e EXISTS (Cálculo Relacional)

  – **ALL**: testa se uma dada condição é verdadeira para todos os valores de uma consulta aninhada

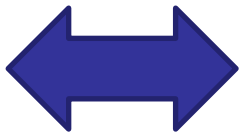
❖Buscar o nome e a idade dos médicos que são mais velhos do que todos os funcionários de Florianópolis

```
SELECT nome, idade FROM medicos
WHERE idade > ALL (
    SELECT idade FROM funcionarios
    WHERE cidade = 'Florianopolis'
)
```

Consultas Aninhadas

- Existem ainda as cláusulas ANY , ALL e EXISTS (Cálculo Relacional)
  - **EXISTS**: testa se um predicado é verdadeiro ou falsa; a subconsulta é executada para cada linha da consulta externa
- Mapeamento entre SQL e álgebra relacional:

```
SELECT a1, ..., an
FROM t1
WHERE EXISTS
  (SELECT * FROM t2
   WHERE d > 5 AND
   t2.x = t1.c)
```



$$\{t_1.a_1, \ldots, t_1.an \mid t_1 \in T_1 \wedge \exists\, t_2 \in T_2 \; (t_2.d > 5 \wedge t_2.x = t_1.c)\}$$

Consultas Aninhadas

- Existem ainda as cláusulas ANY , ALL e EXISTS (Cálculo Relacional)

  – **EXISTS**: testa se um predicado é verdadeiro ou falsa; a subconsulta é executada para cada linha da consulta externa

❖Buscar o nome dos médicos que possuem uma consulta para o dia 06 de novembro de 2013

$$\{ m.nome \mid m \in medicos \ \wedge \ \exists c \in consultas \ (c.data = '2013/11/06' \ \wedge c.codm = m.codm)\}$$

```
SELECT nome FROM medicos AS m
WHERE EXISTS (
    SELECT * FROM consultas
    WHERE data = '2013/11/06'
      AND codm = m.codm )
```

Consultas Aninhadas

- Existem ainda as cláusulas ANY , ALL e EXISTS (Cálculo Relacional)

  – **EXISTS**: testa se um predicado é verdadeiro ou falsa; a subconsulta é executada para cada linha da consulta externa

❖Buscar o nome dos funcionários de Florianópolis que nunca se consultaram como pacientes na clínica

$$\{ f.nome \mid f \in funcionarios \wedge f.cidade = \text{'Florianopolis'} \neg\exists p \in pacientes (p.cpf = f.cpf)\}$$

?

Consultas Aninhadas

- Subconsulta na cláusula **FROM / JOIN**
  - Consulta externa é feita sobre um subconjunto resposta
  - Útil para otimização filtrando linhas e colunas antecipadamente

❖Buscar os dados dos médicos e a hora das consultas que estão agendadas para o dia 06/11/2013

```
SELECT medicos.*, c.hora
FROM medicos JOIN
   (SELECT codm, hora FROM consultas
    WHERE data = '2013/11/06')
AS c ON medicos.codm = c.codm
```

Consultas Aninhadas

- Subconsulta na cláusula **FROM / JOIN**
  - Consulta externa é feita sobre um subconjunto resposta
  - Útil para otimização filtrando linhas e colunas antecipadamente

❖Buscar os números e andares dos ambulatórios em que médicos de Florianopolis dão atendimento

```
SELECT amb.* FROM
   (SELECT nroa, andar FROM ambulatorios)
AS amb JOIN
   (SELECT nroa FROM medicos
    WHERE cidade = 'Florianopolis')
AS mflo ON amb.nroa = mflo.nroa
```

UDESC

Atividade

1. Realize as seguintes consultas usando produto cartesiano ou junção (quando possível use junção natural):
   a. Buscar nome e CPF dos médicos que são pacientes do hospital
   b. Buscar os nomes dos funcionários e médicos que residem numa mesma cidade; mostrar também qual é a cidade
   c. Buscar código e nome dos pacientes com consulta marcada para horários após às 14 horas
   d. Buscar o número e andar dos ambulatórios utilizados por médicos ortopedistas
   e. Buscar nome e CPF dos pacientes que têm consultas marcadas entre os dias 14 e 16 de junho de 2006.
   f. Buscar o nome e a idade dos médicos que têm consulta com a paciente Ana

Atividade

g. Buscar o código e nome dos médicos que atendem no mesmo ambulatório do médico Pedro
h. Buscar o nome, cpf e idade dos pacientes que tem consultas marcadas com ortopedistas para antes do dia 16/06/2006
i. Nome e salário dos funcionários que moram na mesma cidade do funcionário Carlos e possuem salário superior ao dele
j. Buscar os dados de todos os ambulatórios e, para aqueles onde médicos dão atendimento, exibir também os seus nomes.
k. Para cada consulta listar o nome do médico, o nome do paciente, a data, o horário e o ambulatório utilizado.

Atividade

2. Realize as seguintes consultas usando subconsultas com IN, ANY, ALL e/ou EXISTS, ou subconsultas na cláusula FROM:
   a. Buscar nome e CPF dos médicos que são pacientes do hospital
   b. Buscar código e nome dos pacientes com consulta marcada para horários após às 14 horas.
   c. Buscar o número e andar dos ambulatórios onde nenhum médico dá atendimento.
   d. Buscar o número e o andar de todos os ambulatórios, exceto o de menor capacidade.
   e. Buscar nome e CPF dos médicos que não atendem em ambulatórios com capacidade superior à capacidade dos ambulatórios do segundo andar.
   f. Buscar nome e CPF dos médicos que têm consultas marcadas com todos os pacientes.

Ordenar Tuplas Resultantes

- Resultados podem ser ordenadas pela cláusula **ORDER BY**

```
SELECT lista_atributos
FROM lista_tabelas
[WHERE condições]
[ORDER BY nome_atrib_1 [ASC|DESC]
  {[, nome_atrib_n [ASC|DESC]]}
```

- Mapeamento entre SQL e álgebra relacional:

$$\tau_{idade\ asc,\ nome\ desc}(\pi_{nome,idade}(\ Funcionarios\ ))$$

```
SELECT nome, idade FROM pacientes
ORDER BY idade ASC, nome DESC
```

Limitar Tuplas Resultantes

- Resultados podem ser limitados pela cláusula **LIMIT**

```
SELECT lista_atributos
FROM lista_tabelas
[WHERE condições]
[ORDER BY regras]
[LIMIT v1 [,v2]]
```

```
SELECT lista_atributos
FROM lista_tabelas
[WHERE condições]
[ORDER BY regras]
[LIMIT qtd OFFSET ini]
```

- Se apenas v_1 é utilizado ele representa o número de tuplas
- Se v_1 e v_2 forem utilizados, v_1 representa quantos registros iniciais devem ser pulados e v_2 representa o número de tuplas

- No **PostgreSQL** deve-se utilizar **LIMIT** para indicar o número de tuplas e **OFFSET** para indicar quantos registros iniciais devem ser pulados.

Limitar Tuplas Resultantes

- Resultados podem ser limitados pela cláusula **LIMIT**

  ❖ Retornar o nome e a idade dos 3 pacientes mais velhos

```
SELECT nome, idade FROM pacientes
ORDER BY idade DESC LIMIT 3
```

  ❖ Retornar o nome, a idade e o salario dos funcionários que recebem o segundo e terceiro maior salário

```
SELECT nome, salario
FROM funcionarios
ORDER BY salario DESC
LIMIT 1, 2
```

```
SELECT nome, salario
FROM funcionarios
ORDER BY salario DESC
LIMIT 2 OFFSET 1
```

UDESC

Agrupar Tuplas

- Tuplas podem ser agrupados pela cláusula **GROUP BY**
  - Agrupa as tuplas que possuem mesmo valor nas colunas especificadas para o agrupamento
  - Apenas os atributos de agrupamento podem aparecer no resultado final da consulta
  - Geralmente é utilizada alguma **função de agregação** (ex: contagem, somatório) sobre o resultado da consulta

```
SELECT lista_atributos
FROM lista_tabelas
[WHERE condições]
[GROUP BY lista_atributos_agrupamento
  [HAVING condição_para_agrupamento]]
```

Agrupar Tuplas

- Tuplas podem ser agrupados pela cláusula **GROUP BY**
  - Funções de agregação: COUNT, SUM, AVG, MIN, MAX

❖ Listar quantos médicos existem por especialidade

```
SELECT especialidade, COUNT(*)
FROM medicos
GROUP BY especialidade
```

❖ Qual é a média de salários pagos aos funcionários por sua cidade de origem?

```
SELECT cidade, AVG(salario)
FROM funcionarios
GROUP BY cidade
```

Agrupar Tuplas

- Tuplas podem ser agrupados pela cláusula **GROUP BY**
  - Opcionalmente pode-se utilizar a cláusula **HAVING** para aplicar condições sobre os grupos que são formados
  - As condições só podem ser definidas sobre atributos do agrupamento ou sobre funções de agregação

❖ Listar as cidades que são origem de pelo menos mais do que um paciente e informar quantos pacientes são dessas cidades

```
SELECT cidade, COUNT(*)
FROM pacientes
GROUP BY cidade
HAVING COUNT(*) > 1
```

Atualização com Consultas

- Comandos de atualização (INSERT, UPDATE e DELETE) podem incluir comandos de consulta

  - Ex1: a médica Maria pediu para cancelar todas as suas consultas após as 17:00

```
DELETE FROM consultas
WHERE hora > '17:00'
AND codm IN
   (SELECT codm FROM medicos
    WHERE nome = 'Maria')
```

Atualização com Consultas

- Comandos de atualização (INSERT, UPDATE e DELETE) podem incluir comandos de consulta

  - Ex2: a direção da clínica determinou que deve haver sempre dois médicos por ambulatório, caso contrário o médico deve ter um ambulatório definido/fixo

```
UPDATE medicos
SET nroa = NULL
WHERE NOT EXISTS
   (SELECT * FROM medicos AS m
    WHERE m.codm != medicos.codm
      AND m.nroa = medicos.nroa)
```

Atualização com Consultas

- Comandos de atualização (INSERT, UPDATE e DELETE) podem incluir comandos de consulta

  - Ex3: os leitos do ambulatório 4 foram transferidas para o leito de número 2.

?

UDESC

Atividade

1. Mostre os dados de todos os funcionários ordenados pelo salário (decrescente) e pela idade (crescente). Buscar apenas os três primeiros funcionários nesta ordem.
2. Mostre o nome dos médicos, o número e andar do ambulatório onde eles atendem, ordenado pelo número do ambulatório
3. Mostre o nome do médico e o nome dos pacientes com consulta marcada, ordenado pela data e pela hora. Buscar apenas as tuplas 3 a 5, nesta ordem.
4. Mostre as idades dos médicos e o total de médicos com a mesma idade
5. Mostre as datas e o total de consultas em cada data, para horários após às 12 horas

UDESC

Atividade

6. Mostrar os andares onde existem ambulatórios e a média de capacidade por andar
7. Mostrar os andares onde existem ambulatórios e a média de capacidade no andar seja maior ou igual a 40
8. Mostrar o nome dos médicos que possuem mais de uma consulta marcada
9. Passar todas as consultas da paciente Ana para às 19:00
10. Excluir os pacientes que não possuem consultas marcadas
11. Passar todas as consultas do médico Pedro marcadas para o período da manhã para o dia 21/11/2006, no mesmo horário
12. O ambulatório 4 foi transferido para o mesmo andar do ambulatório 1 e sua capacidade é agora o dobro da capacidade do ambulatório de maior capacidade da clínica