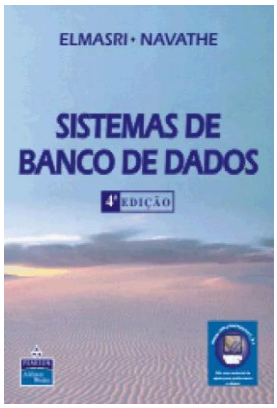
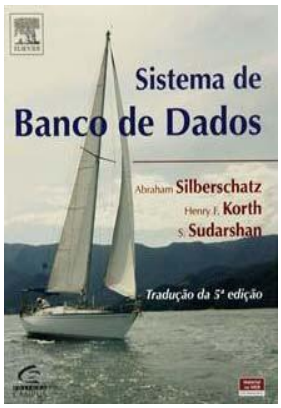
Banco de Dados II

Sistema de Gerenciamento de Banco de Dados - SGBD

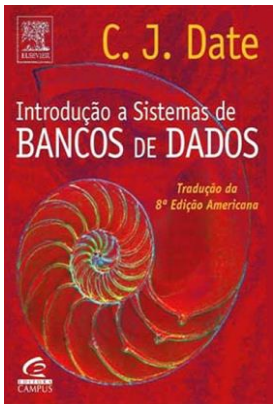
Referências Bibliográficas



NAVATHE, Shamakant; ELMASRI, Ramez.
Sistema de banco de dados.
São Paulo: Pearson Addison-Wesley, 2006.



KORTH, Henry F;
SILBERSCHATZ, Abraham;
SUDARSHAN, S.
Sistema de banco de dados.
São Paulo, Makron, Books, 1999.



DATE, C. J.
Introdução a sistemas de bancos de dados.
Rio de Janeiro, Campus, 1998.

Introdução

3

Introdução

CARACTERÍSTICAS BÁSICAS DE UM BANCO DE DADOS

- **ESTRUTURA**
  - Reflete a Estrutura (Real e Abstrata) dos Objetos do Mini-Mundo correspondente.
- **COMPORTAMENTO**
  - Reflete o Comportamento dos Objetos do Mini-Mundo correspondente.
- **ESTADO**
  - Conjunto dos Dados sobre os Objetos do Mini-Mundo, armazenados do Banco de Dados, num determinado momento do tempo.
- **TRANSAÇÃO**
  - Conjunto de Operações sobre os dados em um Banco de Dados, que levam o Banco de Dados de um Estado Consistente a outro Estado Consistente.
- **CONSISTÊNCIA**
  - Cada Estado do Banco de Dados deve corresponder ao Estado do Mini-Mundo.
- **PERSISTÊNCIA**
  - Capacidade dos Dados continuarem a existir (persistir) após a o término da execução das transações

4

Banco de Dados I

Linguagem SQL

Introdução

- *Structured Query Language*, ou Linguagem de Consulta Estruturada.
- É uma linguagem de pesquisa declarativa para banco de dados relacional (bases de dados relacionais).
- Muitas das características originais do SQL foram inspiradas na álgebra relacional.
- Embora o SQL tenha sido originalmente criado pela IBM, rapidamente surgiram vários dialetos desenvolvidos por outros desenvolvedores.

Introdução

- Essa expansão levou à necessidade de ser criado e adaptado um padrão para a linguagem. Esta tarefa foi realizada pela *American National Standards Institute* (ANSI) em 1986 e ISO em 1987.
- O SQL foi revisto em 1992 e a esta versão foi dado o nome de SQL92.
- Foi revisto novamente em1999 e 2003 para se tornar SQL:1999 (SQL3) e SQL:2003, respectivamente.
- O SQL:1999 usa expressões regulares de emparelhamento, queries recursivas e gatilhos (triggers).

Introdução

- Embora padronizado pela ANSI e ISO, possui muitas variações e extensões produzidos pelos diferentes fabricantes de SGBDs.
- Outra aproximação é permitir para código de idioma processual ser embutido e interagir com o banco de dados.
- Por exemplo, o Oracle e outros incluem Java na base de dados, enquanto o PostgreSQL permite que funções sejam escritas em Perl, Tcl, ou C, entre outras linguagens.

Revisão

- **Tabelas:**
  - Todos os dados de um banco de dados relacional (BDR) são armazenados em tabelas. Uma tabela é uma simples estrutura de linhas e colunas. Cada linha contém um mesmo conjunto de colunas mas as linhas não seguem qualquer tipo de ordem. Em um BD podem existir uma ou centenas de tabelas. O limitador é imposto exclusivamente pela ferramenta de software utilizada.
- **Registros (ou tuplas):**
  - Cada linha, formada por uma lista ordenada de colunas, representa um registro. Estes não precisam necessariamente conter dados em todas as colunas, os seus valores podem ser nulos. Formalmente falando, uma tupla é uma lista ordenada de valores, onde cada valor é do domínio especificado pelo atributo definido no esquema de relação.

Revisão (cont.)

- **Colunas (Atributos ou Campos):**
  - As colunas de uma tabela, são também chamadas de Atributos ou Campos. Cada atributo pertence a um domínio (tipo de campo), que define os valores que podem ser associados aquele atributo.
- **Domínio (Tipos de Dados):**
  - Os domínios possuem características que definem os possíveis valores que serão armazenado em um atributo de uma tupla. Por exemplo. Em um campo do tipo numérico, serão somente armazenados números. Os sistemas de banco de dados possuem regras para consistir os dados que são armazenados.

Revisão (cont.)

- **Chave (Key):**
  - As tabelas relacionam-se umas as outras através de chaves.
  - Uma chave é um conjunto de um ou mais campos que determinam a unicidade de cada registro.
  - Por exemplo, se um banco (base) de dados tem como chave Código do Produto + ID Sistema, sempre que acontecer uma inserção de dados, o SGBD irá fazer uma consulta para identificar se o registro não se encontra gravado na tabela. Caso exista um novo registro não será criado, sendo que apenas a alteração do registro existente será possível.
  - A unicidade dos registros, determinada por sua chave, também é fundamental para a criação dos índices.

Revisão (cont.)

- **Chave (Key)** (cont.)**:**
  - Temos dois tipos de chaves:
    - Chave Primária: (PK - *Primary Key*) é a chave que identifica cada registro dando-lhe unicidade. A chave primária nunca se repetirá. A maioria dos bancos também exige que ela seja NÃO NULA (*NOT NULL*) além de única.
    - Chave Estrangeira: (FK – *Foreign Key*) é uma chave formada pela chave primária de outra tabela e a chave de um campo da tabela que recebe o relacionamento. Define um relacionamento entre as tabelas e pode ocorrer repetidas vezes.
  - **Observação:** Geralmente Chaves Primárias são de apenas um campo, mas podem ser compostas, ou seja, vários campos fazem parte dela. Isto determina que a combinação entre os campos é que precisa ser única, e não os campos que isoladamente a compõe.

Revisão (cont.)

- **Relacionamentos:**
  - Como o próprio nome sugere, um BDR possui diversos relacionamentos entre as tabelas existentes.
  - Um relacionamento é feito ligando-se um campo de uma tabela X com um campo de uma tabela Y ou tabela estrangeira.
  - Por exemplo, se pedidos (em uma tabela) estão relacionados a um cliente (em outra tabela), o SGBD poderá bloquear a remoção deste cliente enquanto ele possuir pedidos registrados.
  - **Observação:** no caso de chaves primárias compostas, a relação se dará entre todos os campos desta chave.

Revisão (cont.)

- **Relacionamentos** (cont.)**:**
  - Existem alguns tipos de relacionamentos possíveis:
  - Um para um (1 para 1)
  - Um para muitos (1 para N)
  - Muitos para muitos (N para N), que não pode ser implementado diretamente no modelo relacional e tem que ser construído com uma tabela auxiliar (associativa).

Linguagem SQL – Funcionalidades

- Existem algumas sub denominações, mas o mais comum é definir em dois grandes grupos:
  - DDL (*Data Definition Language*) Linguagem de Definição de Dados.
  - DML (*Data Manipulation Language*) Linguagem de Manipulação de Dados.
- Definição de visões (*views*).
- Uso de gatilhos (*triggers*) que são disparados automaticamente.
- Definição de tipos de usuários (*roles*).
- Atribuição (*grant*) e remoção (*revoke*) de privilégios de usuários

15

Tipos de Dados

- Variam para cada SGBD.
- No Firebird, são mais usados:
  - Inteiros:

Tipo	Particularidades	Descrição
integer	32 bits	Suportam valores numéricos inteiros com sinal.
smallint	16 bits	
bigint	64 bits	

16

Tipos de Dados (cont.)

- Literais:

Tipo	Particularidades	Descrição
char(T)	retorna os valores com espaço a direita, preenchendo o tamanho T especificado.	São tipos utilizados para armazenar letras, números, caracteres, etc..
Varchar(T)	Retorna da mesma forma em que foram armazenados, ou seja, se o tamanho T não for preenchido, não são adicionados espaços a direita.	

Tipos de Dados (cont.)

- Decimais:

Tipo	Particularidades	Descrição
numeric (T,P)	É necessário especificar o tamanho máximo T (incluindo números após o ponto) e a precisão P (quantas casas decimais).	São utilizados para armazenar valores do tipo fracionário, muito importante quando é necessário precisão no retorno dos dados.
decimal (T,P)		

Tipos de Dados (cont.)

- Ponto Flutuante:

Tipo	Descrição
float	Guarda números de ponto flutuante, porém na maioria dos casos deve ser evitado, pois é possível que a informação armazenada não seja exatamente a mesma quando recuperada.
double	Como o *float*, também armazena valor em ponto flutuante, porém com uma faixa maior de valores, guardando em 64 bits, contudo possui o mesmo problema do *float*, o da precisão na recuperação dos valores.

Tipos de Dados (cont.)

- Temporais:

Tipo	Particularidades
date	Somente datas.
time	Somente horas (tempo).
timestamp	Data + hora.

Tipos de Dados (cont.)

- Binários:

Tipo	Particularidades	Descrição
blob	Dois sub tipos mais usados: 0-Dados binários 1-Informação textual	É utilizado para armazenar desde textos a arquivos binários. Sua capacidade de armazenamento é ilimitada, sendo limitada somente por falta de espaço em disco.

DDL

- Os comandos DDL são usados para definir a estrutura do banco de dados, organizando em tabelas que são compostas por campos (colunas).
- Comandos que compõem a DDL:
  - CREATE
  - ALTER
  - DROP

DDL - Create

- Exemplo de criação de tabela:

```
CREATE TABLE clientes(
cod_cli integer not null primary key,
cpf varchar(11),
nome varchar(100),
data_nascimento date,
sexo char(1),
salario numeric(15,2),
cod_prof integer
);
```

```
CREATE TABLE profissoes(
cod_prof integer not null primary key,
nome varchar(100)
);
```


DDL - Alter

- Criação de *constraints* (chave primária ou estrangeira).
  - Chave Primária:
    - Sintaxe:

```
ALTER TABLE nome_tabela ADD CONSTRAINT PRIMARY KEY
(nome_campo_not_null );
```

    - Exemplo:

```
ALTER TABLE clientes add constraint  PRIMARY KEY (cod_cli);
```

DDL - Alter

- Chave Estrangeira:
  - Sintaxe:

```
ALTER TABLE nome_tabela ADD CONSTRAINT nome_cons TIPO_CONSTRAINT
(nome_campo_a_ser_referenciado)
REFERENCES nome_tabela_estrangeira(campo_chave_primaria) ON UPDATE
regra ON DELETE regra;
```

  - Exemplo:

```
ALTER TABLE clientes ADD CONSTRAINT FK_Cli_Prof FOREIGN KEY (cod_prof)
REFERENCES profissoes(cod_prof)
ON UPDATE cascade ON DELETE set null;
```


DDL – Regras (*Rules*) para Chaves Estrangeiras

- As regras de integridade são acionadas automaticamente em 2 eventos:
  - **ON UPDATE**: Dispara sempre que houver modificações na chave primária referenciada, atuando na estrangeira correspondente, executando uma regra definida.
  - **ON DELETE**: Dispara sempre que houver exclusões na tabela dependente, executando uma regra definida.
- Regras mais utilizadas:
  - **SET NULL**: Atribui nulo na chave estrangeira.
  - **CASCADE**: Exclui todos os registros que possuem como chave estrangeira o mesmo valor da chave primária da tabela referenciada.

Criação de Tabela sem o uso do comando ALTER

- Certifique-se de ter criado todas as estruturas que serão referenciadas.

```
CREATE TABLE clientes(
cod_cli integer not null primary key,
cpf varchar(11),
nome varchar(100),
data_nascimento date,
sexo char(1),
salario numeric(15,2),
cod_prof integer,
foreign key (cod_prof) references profissao (cod_prof)
on update cascade on delete set null
);
```


Auto Incrementando campos do tipo Chaves Primárias (PKs)

- Usando o SGBD Firebird, deve-se criar duas estruturas:
  - ***Generator***: estrutura que armazena o último valor usado. Sempre retorna um valor inteiro.
  - ***Trigger***: evento disparado antes (*before*) ou depois (*after*) de alguma ação (*insert*, *update* ou *delete*).

Auto Incrementando campos do tipo Chaves Primárias (PKs)

- Sintaxes:
  - ***Generator***:
    - **create generator** <nome_do_generator>**;**
  - ***Trigger***:
    - **create trigger** <nome_do_trigger> **for** <tabela>
    - **active** <evento> <ação> **position** <prioridade>
    - **as**
    - **begin**
      - /* Código */
    - **end**

Auto Incrementando campos do tipo Chaves Primárias (PKs)

- Exemplos:
  - ***Generator***:
    - **create generator g_inc_cliente;**
  - ***Trigger***:
    - **Set term^;**
    - **create trigger t_inc_cliente for clientes**
    - **active before insert position 5**
    - **as**
    - **begin**
      - new.COD_CLI=**gen_id**(**g_inc_cliente**,**1**);
    - **End^**
    - **Set term;^**

DDL - Alter

- Alterando Tabelas:
  - Sintaxe:

```
ALTER TABLE nome_tabela
    ADD nome_coluna1 tipo_dado,
    ADD nome_coluna2 tipo_dado,
    ADD nome_coluna3 tipo_dado
;
```

- Exemplo:

```
ALTER TABLE servicos
    ADD valor numeric(15,2),
    ADD descricao varchar(30)
;
```

31

DDL - Alter

- Alterando Tabelas:
  - Sintaxe:

```
ALTER TABLE nome_tabela
    ADD nome_coluna1 tipo_dado,
    ADD nome_coluna2 tipo_dado,
    ADD nome_coluna3 tipo_dado
;
```

- Exemplo:

```
ALTER TABLE servicos
    ADD valor numeric(15,2),
    ADD descricao varchar(30)
;
```

32

DDL - Alter

- Removendo colunas:
  - Sintaxe:

```
ALTER TABLE nome_tabela
    DROP COLUMN nome_coluna1,
    DROP COLUMN nome_coluna2;
```

- Exemplo:

```
ALTER TABLE Pacientes
    DROP COLUMN doenca,
    DROP COLUMN cidade;
```

33

DML

- Define operações de manipulação de dados:
  - INSERT
  - UPDATE
  - DELETE
  - SELECT
  - EXECUTE
- Instruções declarativas manipulação de conjuntos especifica-se o que fazer e não como fazer.

34

DML - Insert

- Sintaxes:

```
INSERT INTO nome_tabela
VALUES (TODA_lista_valores_atributos_ordenados);
```

```
INSERT INTO nome_tabela (lista_atributos_específicos)
VALUES (lista_valores_atributos);
```

- Exemplos:

```
INSERT INTO Ambulatorios VALUES (1, 1, 30);
```

```
INSERT INTO Medicos (codm, nome, idade, especialidade, CPF, cidade)
VALUES (4, 'Carlos', 28,'ortopedia',  11000110000, 'Joinville');
```


DML - Condicional

- Pode-se restringir o resultado das consultas, utilizando a cláusula WHERE.
- Operadores relacionais:
  - Maior: >
  - Menor: <
  - Maior ou igual: >=
  - Menor ou igual: <=
  - Diferente: <>
  - Não nulo: *is not null*
  - Aproximado: *like* (usa-se '**%**' como coringa)
    - Começando com: Ex.: nome_campo like 'a%'
    - Contendo: Ex.: nome_campo like '%a%'
    - Terminando com: Ex.: nome_campo like '%a'

DML - Condicional

- Operadores lógicos:
  - E: *AND*
  - Ou: *OR*
- Apelidos (*Aliases*):
  - É um boa prática usar apelidos em tabelas, a fim de diminuir o tamanho do código.
  - Normalmente usa-se a 1 letra inicial (ou mais) para representar.
  - CUIDADO para não criar apelidos iguais.
  - Exemplo:

```
UPDATE Medicos m
SET m.cidade = 'Jaguari';
```


DML - UPDATE

- Usado para alterar valores de campos em tabelas.
  - Sintaxe:

```
UPDATE nome_tabela
SET nome_atributo_1 = Valor,
    nome_atributo_2 = Valor
WHERE condição;
```

  - Exemplos:

```
UPDATE Medicos
SET cidade = 'Jaguari';
```

```
UPDATE Ambulatorios
SET capacidade = capacidade + 5,
    andar = 3
WHERE nroa = 2;
```

DML - DELETE

- Usado para excluir linhas de tabelas.
  - Sintaxe:

```
DELETE FROM nome_tabela
WHERE condição;
```

  - Exemplos:

```
DELETE FROM Ambulatorios;
```

```
DELETE FROM Medicos
WHERE especialidade = 'cardiologia'
OR cidade < > 'Jaguari';
```

39

DML - SELECT

- Uma consulta é uma recuperação de informações contidas no banco de dados feita através da instrução SELECT.
- Uma consulta é usada para extrair dados do banco de dados em um formato legível de acordo com a solicitação do usuário.
- Por exemplo, se tivermos uma tabela funcionário, poderemos emitir uma instrução SQL que informe qual funcionário tem o maior salário.

40

DML - SELECT

- Existem quatro palavras-chaves, ou cláusulas, que são partes importantes de uma instrução SELECT:
  - SELECT:
    - Tem por finalidade selecionar os dados que deseja-se visualizar de acordo com as colunas em que eles estão armazenados em uma tabela.
  - FROM:
    - Deve ser usada junto com a instrução SELECT, sendo um elemento obrigatório em qualquer consulta.
    - Sua finalidade é informar ao banco de dados que tabelas devem ser acessadas para recuperar os dados desejados da consulta.
    - A cláusula FROM pode conter uma ou mais tabelas e deve sempre listar pelo menos uma tabela.

DML - SELECT

- WHERE:
  - Uma condição é parte de uma consulta que é usada para exibir informações seletivas, conforme especificadas pelo usuário.
  - O valor de uma condição pode ser TRUE ou FALSE, limitando assim, os dados recebidos da consulta.
  - A cláusula WHERE é usada para impor condições a uma consulta, eliminando linhas que normalmente seriam retornadas por uma consulta que não tivesse condições definidas.
  - A cláusula WHERE pode conter mais de uma condição e, nesse caso, as condições são unidas por operadores AND e OR.
- ORDER BY:
  - Esta cláusula organiza os resultados de uma consulta em um formato de listagem que especificamos.
  - A ordenação padrão para a cláusula ORDER BY é a crescente.
  - Para ordenar de forma decrescente usa-se o comando DESC após a indicação dos campos a serem ordenados.

DML - Exemplo

```
SELECT f.nm_funcionario, f.vl_salario
FROM funcionario f
WHERE f.vl_salario > 1200.00
ORDER BY f.nm_funcionario
```


SELECT

- A estrutura básica da instrução de consulta select consiste em três cláusulas:
  - **SELECT**
  - **FROM**
  - **WHERE**

```
SELECT <apelido.coluna>
FROM <tabelas> <apelido>
WHERE <condições>
```

SELECT

- **Cláusula SELECT**
  - Usada para listar os atributos desejados no resultado da consulta, ou seja, as colunas requisitadas como respostas.
  - Esta lista, além de nomes de colunas, pode conter:
    - Asterisco (*), indicando que todas as colunas devem ser retornadas;
    - Expressões;
    - Constantes;
    - Funções;
    - Qualquer combinação destes, conectadas por operadores aritméticos e parênteses.

SELECT

- **Cláusula FROM**
  - Lista as tabelas a serem examinadas na avaliação da expressão.
  - As colunas indicadas na cláusula SELECT devem constar nas tabelas listadas nesta cláusula.

- **Cláusula WHERE**
  - Consiste de uma condição de procura envolvendo atributos das tabelas que aparecem na cláusula FROM.
  - É o lugar onde se restringe ou seleciona quais linhas deverão ser retornadas.

SELECT

- A cláusula **WHERE** pode ser omitida
  - Neste caso, o predicado é tido como sempre verdadeiro e todas as linhas serão retornadas.

- **Exemplo:**
  - Quais os códigos de todos os funcionários que fizeram alguma requisição ao estoque?"
  - Instrução correspondente em SQL:
    - **SELECT r.codFunc FROM requisicao r;**

47

Eliminação de linhas duplicadas

- A eliminação de linhas repetidas ou duplicadas é possível utilizando a palavra-chave **DISTINCT** depois de select.

- Então pode-se reescrever a consulta anterior da seguinte maneira:
  - **SELECT DISTINCT r.codFunc FROM requisicao r;**

48

Constantes e Expressões

- A lista da cláusula **SELECT** também pode conter constantes e expressões.

- **Exemplo utilizando constante:**
  - **SELECT r.codigo, 'DATADA DE', r.dataReq FROM requisicao r;**

- **Exemplo utilizando expressão:**
  - **SELECT p.descricao, p.quantidade, (p.quantidade/2) OPERACAO from produto p;**

Uma expressão pode conter uma combinação de parênteses e das quatro operações aritméticas básicas: adição (+), subtração (-), multiplicação (*) e divisão (/)

Condições de Procura

- Uma condição de procura na cláusula **WHERE** qualifica o escopo de um consulta, pois especifica as condições particulares que devem ser encontradas.

- A cláusula **WHERE** pode ser usada nas instruções **SELECT, DELETE, UPDATE.**

- Cada condição de procura contém um ou mais predicados conectados por operadores lógicos **OR, AND, e NOT.**

Condições de Procura

- Exemplo:
  - **SELECT p.descricao, p.quantidade**
  - **FROM produto p**
  - **WHERE p.quantidade > 100 AND p.quantidade < 400;**

- A seguinte ordem de precedência é utilizada ao avaliar uma expressão numa condição:
  - Primeiro são analisados as expressões dentro de parênteses;
  - O operador **NOT** é analisado antes do **AND;**
  - O operador **AND** é analisado antes do **OR;**
  - Operadores do mesmo nível de precedência são analisados da esquerda para a direita.

51

Predicado Between

- Compara um valor com uma faixa de valores.

- Os limites da faixa estão inclusos.

- Exemplo:
  - **SELECT f.nome, f.salario**
  - **FROM funcionario f**
  - **WHERE f.salario BETWEEN 1000 AND 3000;**

52

Predicado NULL

- O predicado **NULL** testa valores nulos.
- Verifica, por exemplo, se colunas não contém nenhum valor armazenado.
- Exemplo:
  - SELECT f.nome, f.salario FROM funcionario f WHERE f.salario IS NULL;
  - SELECT f.nome, f.salario FROM funcionario f WHERE f.salario IS NOT NULL;

É possível utilizar o operador **NOT** juntamente com a palavra-chave **IS (is not NULL)** para, por exemplo, montar condições de colunas que não possuem valores nulos.

53

Predicado LIKE

- O predicado LIKE procura por strings que se encontram dentro de um determinado padrão.
- O predicado LIKE só pode ser usado com tipos de dados CHAR e VARCHAR.

%	Equivale a zero ou mais caracteres
_	Equivale a um caracter qualquer

- Exemplo:
  - SELECT f.nome FROM funcionario f WHERE f.nome LIKE 'A%';
  - SELECT f.nome FROM funcionario f WHERE f.nome LIKE '%O%';
  - SELECT f.nome FROM funcionario f WHERE f.nome LIKE '_O%';

54

Predicado Exists

- É utilizado para testar se o conjunto de linhas resultantes de uma consulta é vazio.

- Exemplo:

```
SELECT f.nome
FROM funcionario f
WHERE EXISTS
    (SELECT d.*
    FROM devolucao d
    WHERE f.codigo = d.codFunc);
```

55

Predicado IN

- O predicado IN compara um valor com um conjunto de valores.

- Este conjunto de valores pode ser uma lista ou o resultado de um subselect.

- Exemplo:
  - SELECT d.dataDev FROM devolucao d WHERE d.codFunc IN (5, 6, 7);
  - SELECT f.nome FROM funcionario f WHERE f.codigo IN (SELECT d.codFunc from devolucao d);

56

Funções de agregação

- SQL oferece a possibilidade de computar funções em grupos de linhas usando a cláusula **GROUP BY**
- Os atributos, dados na cláusula **GROUP BY,** são usados para formar grupos.
- Linhas com o mesmo valor em todos os atributos na cláusula **GROUP BY** são colocados em um grupo.

57

Funções de agregação

- SQL inclui funções de agregação para computar:
  - Média: **AVG**
  - Mínimo: **MIN**
  - Máximo: **MAX**
  - Total: **SUM**
  - Contar: **COUNT**
- Exemplo: Selecione o salário médio em cada setor da empresa
  - **SELECT f.codSetor, AVG (f.salario)**
  - **FROM funcionario f**
  - **GROUP BY f.codSetor;**

58

Funções de agregação

Função de Agregação	Resultado
SUM (| DISTINCT | expressão)	Soma de valorres DISTINTOS na expressão numérica
AVG (| DISTINCT | expressão)	Média de valores DISTINTOS na espressão numérica
COUNT (| DISTINCT | expressão)	Número de valores DISTINTOS na espressão numérica
COUNT(*)	Número de linhas selecionadas
MAX (expressão)	Marior valor computado para a expressão
MIN (expressão)	Menor valor computado para a expressão

Cláusula HAVING

- Em certas situações, é necessário definir uma condição que se aplique a grupos, ao invés de linhas.

- Exemplo: Selecionar o salário médio apenas daqueles setores cujo salário médio seja maior que 2000

```
SELECT f.codSetor, AVG (f.salario) FROM funcionario f
GROUP BY f.codSetor HAVING AVG (f.salario) > 2000;
```

Junções

- *Para cada embarque com quantidade maior que 300, obter o código da peça e o nome da peça*

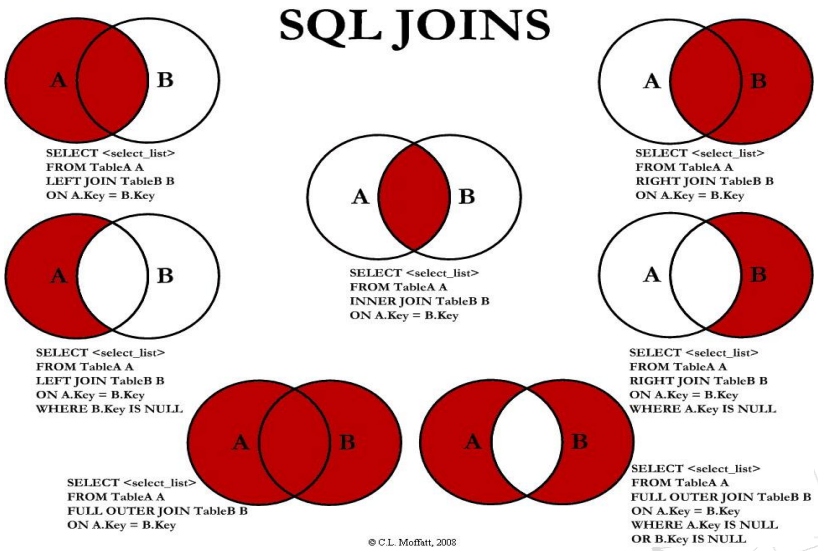

```
Select P.CodPeça, P.Nome
From   Embarq E, Peças P
Where E.Qtde > 300
AND E.CodPeça = P.CodPeça
```

Atributos de junção

```
Select P.CodPeça, P.Nome
From   Embarq E
Inner Join Peças p on E.CodPeça = P.CodPeça
Where E.Qtde > 300
```

Atributos de junção

Junções



Exemplo

Peça

CodPeça	NomePeça	CorPeça	PesoPeça	CidadePeça
P1	Eixo	Cinza	10	PoA
P2	Rolamento	Preto	16	Rio
P3	Mancal	Verde	30	São Paulo

Embarq

CodPeça	CodFornec	QtdeEmbarc
P1	F1	300
P1	F2	400
P1	F3	200
P2	F1	300
P2	F4	350

Fornec

CodFornec	NomeFornec	StatusFornec	CidadeFornec
F1	Silva	5	São Paulo
F2	Souza	10	Rio
F3	Álvares	5	São Paulo
F4	Tavares	8	Rio

```
Select   *
From     Embarq E, Fornec F
Where    E.CodFornec = F.CodFornec
```

CodPeça	CodFornec	QtdeEmbarc	CodFornec	NomeFornec	StatusFornec	CidadeFornec
P1	F1	300	F1	Silva	5	São Paulo
P1	F2	400	F2	Souza	10	Rio
P1	F3	200	F3	Álvares	5	São Paulo
P2	F1	300	F1	Silva	5	São Paulo
P2	F4	350	F4	Tavares	8	Rio

Junções

- Obter os nomes dos fornecedores das peças de cor vermelha

Peça

CodPeça	NomePeça	CorPeça	CidadePeça

Fornec

CodFornec	NomeFornec	StatusFornec	CidadeFornec

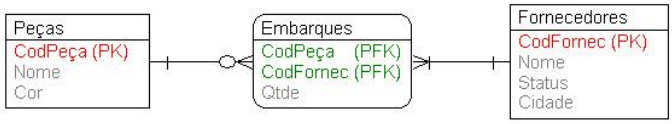
Embarq

CodPeça	CodFornec	QtdeEmbarc

```
Select NomeFornec
From Fornec F, Embarq E, Peca P
Where  CorPeca = 'Verm'
And E.CodPeca = P.CodPeca
And E.CodFornec = F.CodFornec
```

```
Select F.NomeFornec
From Fornec F
Inner Join Embarq E on E.CodFornec = F.CodFornec
Inner Join Peca P on E.CodPeca = P.CodPeca
Where  P.CorPeca = 'Verm
```

Junções na cláusula FROM



- *Obter o código e o nome das peças embarcadas*

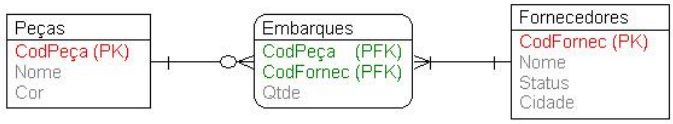
```
Select E.CodPeca, P.NomePeca
From  Embarques E
Inner Join Peças P on E.CodPeça= P.CodPeça
```


Junções Externas

- Junção na qual as linhas de uma ou ambas as tabelas que não são combinadas são mesmo assim preservadas no resultado
- Três Tipos
  - Junção externa à esquerda (*left [outer] join*)
    - Linhas da tabela à esquerda são preservadas
  - Junção externa à direita (*right [outer] join*)
    - Linhas da tabela à direita são preservadas
  - Junção externa completa (*full [outer] join*)
    - Linhas de ambas tabelas são preservadas

Junções Externas

```
select lista_atributos
from tabela1
{left|right|full} join tabela2 on condição_junção
[where condição]
```



- *Obter o nome de todos Fornecedores e, para aqueles que têm peças embarcadas, mostrar o código da peça embarcada*

```
Select f.Nome, e.CodPeça
From  Fornecedores f
left join Embarques e on f.CodFornec = e.CodFornec
```

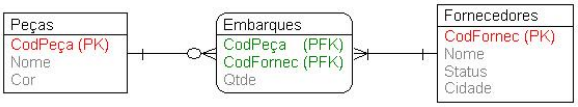

Junções Externas

- *Buscar o nome de todos os médicos e, para aqueles que têm consultas marcadas, mostrar os dados de suas consultas*

- *Buscar o nome de todos os médicos e, para aqueles que têm consultas marcadas, mostrar os nomes dos pacientes da consulta marcada e o horário da consulta*

- *Buscar os números de todos os ambulatórios e, para aqueles ambulatórios nos quais médicos dão atendimento, exibir o CRM e o nome dos médicos associados*

Subconsultas ou Consultas Aninhadas



Obter os códigos dos fornecedores que tem embarques de peças de cor vermelha

```
SELECT CodFornec
FROM   Embarques E,Peças P
WHERE  Cor='Verm' AND E.CodPeca=P.CodPeca
```

Neste caso o resultado da consulta envolve apenas colunas da tabela Embarq, mas a cláusula FROM referencia também a tabela Peça

```
SELECT  CodFornec
FROM    Embarques
WHERE   CodPeça IN
        ( SELECT CodPeça
          FROM Peças
          WHERE Cor= 'Verm' )
```

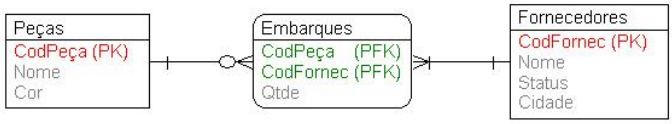
69

Subconsultas ou Consultas Aninhadas

- Filtragens prévias de dados na subconsulta
  - Apenas linhas/colunas de interesse são combinados com dados da(s) tabela(s) da consulta externa

70

Subconsultas – cláusula exist



Obter os nomes dos fornecedores para os quais não há embarques

```
SELECT F.Nome
FROM   Fornecedores F
WHERE  not exists
       (SELECT  *
        FROM  Embarques E
        WHERE E.CodFornec = F.CodFornec)
```

71

Modelo Estendido

- Modelo Mental de execução
  1. É feito o *produto cartesiano* das tabelas envolvidas
  2. São *selecionadas* as linhas da tabela que obedecem ao critério da cláusula WHERE
  3. São *criados grupos* de linhas que contenham valores idênticos nas colunas GROUP BY
  4. São *selecionados os grupos* que obedecem ao critério da cláusula HAVING
  5. É feita a *classificação* do resultado pelos valores das colunas da cláusula ORDER BY
  6. É feita a *projeção* sobre as colunas que vão ao resultado

72

Definição de Grupos

```
select bairro, count(*)
from MEDICOS
group by bairro
```

Bairro	count(*)
Centro	3
Tristeza	1
Moinhos	2

MEDICOS

Codigo	Nome	Bairro
E3	João	Centro
E4	José	Tristeza
E1	Laura	Centro
E6	Carlos	Moinhos
E7	Nilton	Centro
E9	Bianca	Moinhos

Codigo	Nome	Bairro
E3	João	**Centro**
E1	Laura	**Centro**
E7	Nilton	**Centro**

Codigo	Nome	Bairro
E4	José	**Tristeza**

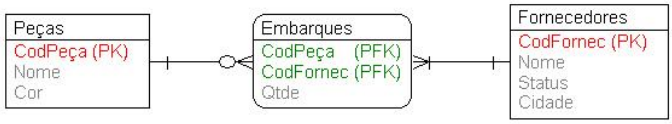
Codigo	Nome	Bairro
E6	Carlos	**Moinhos**
E9	Bianca	**Moinhos**

Condições em grupos

- Cláusula HAVING
  - define condições para que grupos sejam formados
    - condições só podem ser definidas sobre atributos do agrupamento ou serem operações de agregação
  - existe somente associada à cláusula GROUP BY
- Exemplos

```
select especialidade, count(*)
from Médicos
group by especialidade
having count(*) > 1
```

Condições em grupos

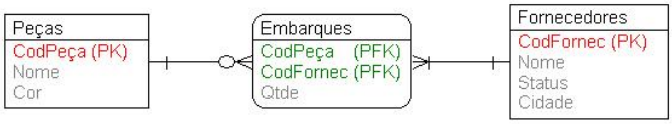


Obter o código dos fornecedores que realizaram mais de 3 embarques totalizando uma quantidade embarcada superior a 200 unidades

```
SELECT E.CodFornec,SUM(E.Qtde)
FROM   Embarques E
GROUP BY E.CodFornec
HAVING COUNT(E.CodPeça) > 3
And SUM(E.Qtde) < 200
```

75

Condições em grupos



Obter os códigos de fornecedores que totalizam mais de 500 unidades de peças vermelhas embarcadas, junto com a quantidade de embarques de peças vermelhas

```
SELECT E.CodFornec,COUNT(E.CodPeça)
FROM Embarques E
WHERE CodPeça IN (SELECT P.CodPeça
                  FROM Peças P
                  WHERE P.Cor='Verm')
GROUP BY P.CodFornec
HAVING SUM(P.Qtde) > 500
```

76