postilando•com

O maior Site de apostilas da Internet

# Introdução a linguagem SQL

**Autor:**
Jair Daum Franceschini

## Objetivos

Ao término da apresentação, o leitor deve estar apto a usar a linguagem SQL para:

- Fazer um acesso básico (query) de um BD existente;
- Usar funções de construção para realizar manipulação de dados;
- Obter informação derivada do BD;
- Agregar dados que foram recuperados de um BD;
- Agrupar e limitar dados recuperados do BD;
- Unir uma ou mais tabelas para recuperar dados do BD;
- Realizar queries de dados usando subqueries;
- Criar uma tabela simples e incluir, atualizar e deletar dados.

## A linguagem SQL

**Strutured Query Language (Linguagem de consulta estruturada):**

- Linguagem como o Inglês;
- Não incluí nenhuma referência a caminhos de acesso explícitos;
- É um modo de recuperar e manipular dados de um BD;
- Pode ser usado por um terminal ON-LINE;
- Pode ser usado na forma de “EMBEDDED SQL” em um programa de aplicação;
- Possui uma série completa de operações de atualização;
- Usado para administração.

## Database – Banco de Dados (BD)

É um corpo de dados no qual existem relacionamentos entre os elementos de dados.

## Database Management System (Sistema Gerenciador de Dados)

Software que facilita a definição de BDs estruturados e o armazenamento e recuperação dos dados dessas estruturas.

## Terminologia

- **Tabela (relação)**
  - um conjunto de linhas, ou um conjunto de listas de valores, uma “relação”.
- **Coluna (atributo)**
  - Semelhante à campo de um registro;
  - Cada coluna em cada linha tem apenas um conteúdo;
  - Cada coluna é só de um tipo de dados.
- **Linha (dupla)**
  - Análogo a um registro de um arquivo;
  - Todas linhas de uma tabela tem o mesmo conjunto de colunas.
- **Chave Primária (Primary Key)**
  - Uma ou mais colunas com valores que são únicos dentro da tabela e por isso podem ser usados para identificar as linhas dessa tabela.
- **Domínio**
  - Conjunto de valores válidos para uma determinada coluna.

## Tabelas

- **Modelo Relacional**
  - Num BD relacional todos os dados estão em tabelas;
  - A tabela mantém dados relacionados a urna determinada classe de objetos.
- **Tabelas são feitas de linhas e colunas**
  - Existe somente um valor de dado para cada coluna de cada linha.

## Linhas e Colunas

- **Colunas**
  - Cada coluna tem um nome.

        - Cada coluna contém dados sobre um aspecto da tabela;
        - Cada coluna contém dados de um só tipo, por exemplo:
            - número inteiro;
            - string de caracteres;
            - etc.
- **Linhas**
    - Cada linha contém dados relacionados a uma citação da tabela;
    - As linhas não estão em qualquer ordem determinada.

## Acessando os dados / Chaves primárias

- O nome da tabela, nome da coluna e linha determinam um ítem de dado;
- Cada linha pode ser acessada por uma única Chave Primária;
- Em cada linha, alguma coluna ou grupo de colunas identificam a linha.

## Pessoal do Banco de Dados e funções de trabalho

- **Usuário do Banco de Dados**
    - Recuperação dos dados;
    - Atualização dos dados;
    - Inclusão dos dados;
    - Deleção dos dados.
- **Desenhista do Banco de Dados**
    - Desenha o Banco de Dados;
    - Criação das Tabelas;
    - Criação de Regras de Trabalho / Segurança (Integridade Referencial).
- **Programador de Aplicação no Banco de Dados**
    - Utilização do “embeded SQL” na codificação;
    - Desenha e escreve telas para usuários.
- **Administrador do Banco de Dados**
    - Instalação do sistema;
    - Cópias e Recuperações;

- Performance e segurança (Ex: Senha).

## Formando uma tabela

O DBA ou desenhista criará as tabelas usando os seguintes passos:

- **Cria a Tabela**

```
create table stores
        (stor_id        char(4)         not null,
        stor_name       varchar(40)     not null,
        stor_address    varchar(40)     null,
        city            varchar(40)     null,
        state           char(2)         null,
        country         varchar(12)     null,
        postalcode      char(5)         null,
        dayterms        varchar(12)     null)
go
```

Urna vez que a tabela é criada os dados podem ser inseridos.

- **Inclui os dados**

Exemplo:

**insert into** stores (stor_id stor name stor_address, city, state, country, postalcode, dayterrns)
**values** (“7066”, “Barnum’s” , “567 Pasadena Ave”, “Tustin” “CA”,”USA”,”92745”,”Net 30”)

## Recuperando os dados de uma tabela

- **Recuperando o dado**

```
select *
from stores
```

- **Convenção de nomes**

  database.owner.table_name.column_name

  Exemplo:

  pubs2.bob.stores.stor_id;

  pubs2.fred.stores.stor_name.

## O banco de dados Pubs2

Titles

| | |
|---|---|
| Authors | Salesdetail |
| Titleauthor | Royched |
| Publishers | Discounts |
| Stores | Blurbs |
| Sales | Au_pix |

O banco de dados Pubs2 contém 11 tabelas de dados relacionadas a operação e distribuição de livros fictícios.

| | |
|---|---|
| Titles | Informações básicas de cada livro |
| Authors | Informações sobre autores |
| Titleauthor | Tabela relacional usada para ligar as tabelas authors e title |
| Publisher | Informações sobre editores |
| Stores | Informações sobre lojas que vendem nossos livros |
| Sales | Resumo de vendas para as lojas |
| Salesdetail | Vendas detalhadas de títulos para as lojas |
| Roysched | Escala demonstrando o percentual de royalties pagável em cada título |
| Discounts | Uma tabela de descontos variados disponíveis |
| Blurbs | Pequena bibliografia dos autores |
| Au_pix | Imagem dos autores |

## Escolhendo um banco de dados

- **Para conectar-se a um Banco de Dados**

Sintaxe:

**use** nome_do_banco_de_dados

## Simples recuperação – Select / From

- **Usado para recuperar dados de um Banco de Dados**
  - **select**, especifica a coluna que você quer recuperar;
  - **from**, específica a tabela ou as tabelas das quais você deseja recuperar dados.

Sintaxe simplificada:

  - **select:** lista_do_select;
  - **from:** lista_de_tabelas;

- **Todas as colunas**
  - select *

    from stores

    (Lista todas as informações da tabela stores).

- **Uma coluna**
  - select stor_narne

    from stores

    (Lista apenas os nomes das lojas tabela stores)

- **Mais de uma coluna**
  - select stor_name, city, state

    from stores

    (Lista nome da loja, a cidade e o estado da tabela stores)

- **Reordenando colunas**

A ordem das colunas no comando select determina a ordem em que as colunas aparecerão no resultado.

  - select city, stor namc, state

    from stores

    (Lista a cidade, o nome da loja e o estado da tabela stores)

## Eliminando duplicações

- **Distinct:** palavra chave que elimina colunas duplicadas na saída

Sintaxe simplificada:

- select [ distinct ] lista_do_select
  from lista_de_tabelas

Exemplo:

- (sem distinct)
  select state
  from stores
  (Lista os estados da tabela stores)
- (com distinct)
  select distinct state
  from stores
  (Lista uma ocorrência para cada estado existente na tabela stores)

## Recuperação qualificada – Select / From / Where

- **where:** cláusula que determina exatamente que linhas devem ser recuperadas.

Sintaxe simplificada:

- select lista_do_select
  from lista_de_tabelas
  where condições_de_pesquisa

- **Condições na cláusula where**
  - Operadores de comparação (=,>,<);
  - Intervalos (between and not between);
  - Caracteres semelhantes (like and not like);
  - Valores desconhecidos (is null and is not null);
  - Listas(in and not in);
  - Combinações (and,or).
- **not** pode negar qualquer expressão booleana e palavras chave, tais como “like”,

“null”, “between”, e “in”.

- **operadores de comparação**

| | |
|---|---|
| = | Igual a |
| > | Maior que |
| < | Menor que |
| >= , !< | Maior ou igual a |
| <= , !> | Menor ou igual a |
| != , <> | Diferente |

  - Na comparação de datas, “<” significa antes e “>” significa depois;
  - Use aspas simples ou duplas para dados char, varchar e datetime, por exemplo;
  - Alfabéticos de caixa baixa são maiores que alfabéticos de caixa alta e alfabëticos de caixa alta são maiores que números.

Exemplo:

  - (igual)
    select stor_name, state
    from stores
    where state = ‘CA’
    (Lista as lojas que se localizam na Califórnia)

  - (não igual)
    select stor_name, city, state
    from stores
    where state != ‘CA’
    (Lista as lojas que não se localizam na Califórnia)

  - (maior que)
    select stor_id, stor_name
    from stores
    where stor_id > ‘7066’
    (Lista os códigos e nomes das lojas que possuem código maior que ‘7066’)

- (menor que)

  select city, stor_name

  from stores

  where city < ‘Remulade’

  (Lista os nomes das lojas e as cidades das lojas que se localizam em cidades cujo nome é alfabeticamente inferior a ‘Remulade’)

- **between:** palavra chave usada para especificar uma série que compreende, de um valor mais baixo até um valor mais alto, para a pesquisa.

Exemplo:

select stor_id stor_name, city, state

from stores

where stor_id between ‘7067’ and ‘8000’

(Lista as lojas que possuem os códigos entre 7067 e 8000 inclusive)

- **not between:** palavra chave usada para excluir os dados fora da série entre o mais baixo e o mais alto valor especificado (Fora do intervalo).

Exemplo:

select stor_id, stor_name, city, state

from stores

where stor_id between ‘7067’ and ‘8000’

(Lista as lojas que possem os códigos fora do intervalo 7067 e 8000.)

- **like:** palavra chave usada para selecionar linhas que contém campos que se assemelham em uma determinada porção do string.
  - É usado somente com char , varchar e datetime;
  - Pode usar coringas
    - Significado dos coringas
      - % (porcentagem), qualquer string de zero ou mais caracteres;
      - _ (anderscore), qualquer caractere simples;

        - [] (colchetes), qualquer caractere simples dentro da série especificada;
        - [^] (colchetes e acento circunflexo), qualquer caractere simples fora da série especificada.
    - Coloque os coringas e o string de caracteres entre aspas simples.

Exemplo:

- **Like**

  Select stor_name

  From stores

  Where stor_name like ‘B%’

  (lista os nomes das lojas cujos nomes comecem com a letra B)
- **Not Like**

  Select sor_name

  From stores

  Where stor_name not like ‘B%’

  (lista os nomes das lojas cujos nomes não comecem com a letra B)
- **Especificando o número de caracteres**

  Select stor_id, stor_name

  From stores

  Where stor_id like ‘70__’

  (lista as lojas cuja identificação possua nas duas primeiras posições 70)
- **Intervalo de caracteres**

  Select stor_name

  From stores

  Where stor_name like ‘[D-F]%’

  (lista os nomes das lojas cujos nomes comecem com a letra D, E ou F)
- **Fora do intervalo de caracteres**

  Select stor_name

  From stores

  Where stor_name like ‘[^D-F]%’

  (lista os nomes das lojas cujos nomes não comecem com a letra D, E ou F)
- **Dois intervalos**

  Select stor_name

From stores

Where stor_name ‘[A-C, G-Z]%’

(lista os nomes das lojas cujos nomes comecem com a letra A, B, C ou de G à Z)

- **in, not in:** palavra chave que permite que você selecione valores que se assemelham ou não com qualquer valor de uma lista de valores.

Exemplo:

  - **in**

    select stor_name, city, state

    from stores

    where state in (‘CA’, ‘WA’)

    (lista as lojas que se realizam em Washington ou na Califórnia)
  - **not in**

    select stor_name, city, state

    from stores

    where state not in (‘CA’, ’WA’)

    (lista as lojas que se não localizam em Washington e nem na Califórnia)

- **and:** une duas os mais condições. Retorna resultados somente quando todas condições são verdadeiras.

Exemplo:

  - select stor_name, city, state

    from stores

    where state = ‘CA’

    and city = ‘Fremont’

    (lista as lojas localizadas na cidade de Fremont no estado da Califómia)

- **or:** conecta duas ou mais condições. Retoma resultados quando qualquer uma das condições é verdadeira. É compreensivo.

Exemplo:

  - select stor_name, city, state

    from stores

    where state = ‘CA’

or city = ‘Portland’

(lista as lojas localizadas na cidade de Portland ou no estado da Califórnia)

**Quando mais de um operador lógico é usado, a ordem na qual eles são executados é:**

**not**

**and**

**or**

**Parênteses podem alterar o significado de uma pergunta para forçar a ordem de execução**

Exemplos:

- **sem parênteses**

  select title_id, type, advance

  from titles

  where type = ‘business’ or

  type = ‘psychology’ or

  advance> 5500

  (lista os livros da categoria business, ou categoria psychology, ou com adiantamento maior que U$ 5500)

- **com parênteses**

  select title_id, type, advance

  from titles

  where (type = ‘business’ or

  type = ‘psychology’) and

  advance> 5500

  (lista os livros da categoria business, ou categoria psychology, e desses somente os que possuem adiantamento maior que U$ 5500)

## Renomeando Colunas

Permite que o usuário forneça outro nome a ser usado na saída do comando select ao invés do nome da coluna.

Sintaxe simplificada:

- select titulo_de_coluna = nome_de_coluna (,...)

**Atenção:** A falta de uma vírgula na lista de seleção pode causar que a seguinte coluna seja tratada como um título de coluna ao invés de um nome de coluna.

Exemplo:

- select CPF = au_id, SOBRENOME = au_lname
  from authors
  where state = 'CA'

## Strings de caracteres nos resultados da query

Adicionando strings de caracteres à cláusula select.

Exemplo:

- select 'O NOME DA LOJA É', stor_name
  from stores
  where stor_id = '7067'
- select 'TOTAL DE VENDAS', total_sales, 'DO LIVRO', title_id
  from titles
  where type 'psychology'
- select 'O NOME DO EDITOR', pub_id 'É ', pub_name
  from publishers

## Expressões numéricas

- Operadores aritméticos

| Símbolo | Operação |
|---|---|
| + | Adição |
| - | Subtração |
| * | Multiplicação |
| / | Divisão |
| % | Módulo |

- podem ser usados em qualquer coluna numérica;
- usados em qualquer cláusula que permita uma expressão;
- módulo não pode ser usado em colunas de valores monetário.

- **Adição**

  select advance, price, advance + price, title_id

  from titles

  where type ‘business’
- **Subtração**

  select advance, price, advance - price, title_id

  from titles

  where type = ‘business’
- **Multiplicação**

  select title, RECEITA = price * total_sales

  from titles

  where type = ‘business’
- **Multiplicação** (expressão usada na cláusula where)

  select title, RECEITA = price * total_sales

  from titles

  where price * total_sales > 8000
- **Multiplicação** (múltiplas condições)

  select title, RECEITA = price * total_sales

  from titles

  where price * total_sales > 8000 and

  type = ‘business’
- **Divisão**

  select title, VENDAS MENSAIS = total_salles/12

from titles

where type = ‘business’

- **Módulo**

  select total_sales, total_sales % 2, title_id

  from titles

  where total_sales % 2 = 1

- Função Convert
  - Converte expressões de um tipo de dados para outro;
  - Usado na lista_do_select e na cláusula where;
  - Uma expressão “char” que contenha alfabéticos, não pode ser convertida para um “int”.

Sintaxe simplificada:

- convert (tipo_de_dados, expressão)

- erro – tipos de dados incompatíveis
  - select stor_id * 10

    from stores
- correto
  - select convert (int, stor_id) * 10

    from stores

**Com dados monetários e “qualquer outro tipo”, tais como ponto flutuante, um símbolo de dólar ($) pode opcionalmente ser usado na frente do “outro tipo” de modo a executar uma operação. (*3)**

Exemplo:

- select price * $1.10, title

  from titles

  where type = ‘business’

## Valores nulos

- um valor nulo implica em um valor desconhecido
    - um valor nulo não implica em zeros ou brancos, não existe valor designado;
    - is null ( = nul ) pode ser usado para selecionar colunas que contém valores nulos.
- Um valor nulo nunca é igual a outro valor nulo.
- Algumas colunas são definidas para permitir valores nulos.
- Operações envolvendo nulos resultam em nulos.

Exemplo:

- select title, price
  from titles
  where type = ‘popular_comp’ and
  price is null
- select title_id, advance
  from titles
  where type = ‘popular_comp’ and
  price = null
  select title_id, advance
  from titles
  where advance is not null
- select title_id, advance
  from titles
  where advance < 5000
- select title_id, advance
  from titles
  where advance < 5000 or
  advance is null
- operação envolvendo nulos resultando nulos
  select title_id, advance, price, advance / price
  from titles
  where type not in (‘business’, ‘psychology’, ‘trad_cook’)

## Select / order by

A clausula order by sorteia os resultados da pergunta (em ordem ascendente por default)

- itens nomeados nessa cláusula não precisam aparecer no select_list;
- Usando order by, nulos são listados primeiro.

Sintaxe simplificada:

- select [distinct] lista_do_select
  from lista_de_tabelas
  [where condições_de_pesquisa]
  [order by {coluna/expressão} [asc/dsc] [,...]]

Exemplo:

- select stor_name state
  from stores
  order by state, stor_name
- select stor_name, state
  from stores
  order by state, stor_name desc
- ordenando por coluna derivada
  select ((total_sales / 12) * 1.10), title_id
  from titles
  where type = ‘psychology’
  order by ((total_sales / l2) * 1.10)
- ordenando por coluna não constante no select
  select title, price
  from titles
  where pub_id = ‘0736’
  order by pubdate
- nulos aparecendo primeiro
  select title_id, type, price
  from titles

where type = ‘popular_comp’
order by price

## Funções agregadas

| Palavra chave da função | Valor calculado |
|---|---|
| SUM | Total |
| AVG | Valor médio (média) |
| MIN | Valor mínimo |
| MAX | Valor máximo |
| COUNT(*) | Número de linhas |
| COUNT | Número de (únicos) |
| ([DISTINCT] nome_da_coluna) | Valores válidos |

- Agregados ignoram valores nulos (exceto count(*));
- SUM e AVG só trabalham com valores numéricos;
- Só uma coluna é retornada (se a cláusula “group by” não é usada);
- Não podem ser usados na cláusula where;
- Pode ser aplicado a todas as colunas de uma tabela ou subconjunto de tabelas

Sintaxe simplificada:

  - select função_agregada ([distinct] expressão)
    from lista_de_tabelas
    [where ... condições]

- **COUNT:** contém a soma do número de colunas que satisfazem a condição
  - select count(*)
    from titles
  - select count(advance)
    from titles
- **MAX:** determina o maior valor
  - select max (price)
    from titles

- **MIN:** determina o menor valor
    - select min (price)
      from titles
- **SUM:** determina a quantidade total de valores em determinada coluna
    - select sum (total_sales)
      from titles
      where type = ‘psychology’
- **AVG:** determina a média dos valores em determinada coluna
    - select avg (advance)
      from titles
- **mais de uma função agregada pode ser usada na cláusula select**

Exemplo:

    - select min (advance), max (advance)
      from titles

- Palavra chave distinct
    - Opcional com SUM, AVG, e COUNT;
    - Não permitida com MIN, MAX e COUNT(*);
    - Permitida com COUNT(nome_de_coluna);
    - Usado somente com nomes de colunas, e não com expressões aritméticas.

Exemplo:

    - select avg (distinct price)
      from titles
      where type = ‘business’
      Resultado = 11.64
    - select avg (price)
      from titles
      Resultado = 13.73
    - select count (distinct type)
      from titles

## Lidando com valores nulos

Tomando um valor nulo a ser contado numa função agregada.

- isnull (expressão, valor)
    - select avg (price)
      from titles
    - select avg (isnull(price, $9.98))
      from titles
      (para livros que não possuem preço, assumir U$ 9.98)
    - select price * 2
      from titles
    - select isnull (price, 0) * 2
      from titles

## Selecto/Group by

A cláusula group by divide dados em grupos

- Normalmente usado com uma função agregada na lista_de_select.
- Todos os valores nulos na coluna “group by” são tratados como um grupo.

Sintaxe simplificada:

**select** [distinct] lista_de_select

**from** lista_de_tabelas

[**where** condições_de_pesquisa]

[**group by** [all] expressão_agregada [,…]]

[**order by** coluna/expressão [asc|desc] [,...]]

Exemplos:

(uso correto)

select type, avg(price)

from titles

group by type

(uso com order by)

```
select avg(price), type
from titles
group by type
order by (price)
select stor_id, sum(qty)
from salesdetail
groupo by stor_id
```

A cláusula “where” elimina linhas antes de fazer agrupamentos

- Aplica uma condição à tabela antes que os grupos sejam formados;
- Não aceitará uma função agregada.

Sintaxe simplificada:

**select** [distinct] lista_do_select
**from** lista_de_tabelas
[**where** condições_de_pesquisa]
[**group by** [all] expressão_agregada [,…]

Exemplos:

```
select title_id, sum(qty)
from salesdetail
where discount > 50
group by title_id
```

(Quais os livros que foram vendidos com desconto maior que 50%?)

O qualificador “all” pode ser usado com um “group by”

- Inclui no resultado os grupos determinados, mais os grupos que não tem linhas que satisfizeram a condição where;
- O valor agregado será nulo para as linhas que não satisfizeram a condição where.

Exemplo:
select title_id, avg(price)
from titles
where type in (‘business’, ’popularcomp’)
group by type

select type, avg(price)
from titles
where type in (‘business’, ‘popular_comp’)
group by all type

A cláusula “having” determina condições para cláusula”group by”

- Aplica uma condição aos grupos depois que eles foram formados.

Sintaxe simplificada:
**select** [distinct] lista_do_select
**from** lista_de_tabelas
[**where** condições_de_pesquisa]
[**group by** [all] expressão_agregada [,...]]
[**having** condições_de_pesquisa]
[**order by** {coluna|expressão} [asc|desc] [,...]]

select title_id, pubdate, total_sales, price
from titles
where total_sales > 4000
having pubdate > 06/12/85
(Lista todos os livros publicados depois de 06/12/85 e que venderam mais de 4000 cópias)

ou

```
select title_id, pubdate, total_sales, price
from titles
where total_sales > 4000
and pubdate > 06/12/85

select title_id, sum(qty)
from salesdetail
group by title_id
having sum(qty) > 50
```
(Quais os livros que venderam mais de 50 cópias?)

```
select title_id, sum(qty)
from salesdetail
where discount > 50
group by title_id
having sum(qty) > 50
```
(Quais os livros com desconto maior que 50% que venderam mais de 50 cópias?)

## Operação de união “JOIN”

O “join” é uma operação de multi-tabelas

- **Select:** o nome da coluna deve ser precedido pelo nome da tabela, se o nome da coluna for ambíguo, isto é, se mais de uma coluna na tabela especificada na cláusula “from” tiver o mesmo nome.
- **From:** duas ou mais tabelas listadas no comando “from” indicam ao SQL server que uma união é desejada.
    - tabelas podem estar localizadas no mesmo BD ou em diferentes BDs.
- **Where:** colunas são comparadas; elas devem ter valores similares (valores puxados do mesmo domínio).
    - não precisam ter o mesmo tipo de dados; mas tem de ser de um tipo que o SQL server automaticamente converta.
    - (int, smallint, tinyint, decimal real or float);
    - (char, varchar, datetime and smalldatetime)

- Valores nulos nunca se unem
- Colunas na condição de “join” não precisam ser definidas na cláusula “select”.

Sintaxe simplificada:
**select** [tabela], nome_de_coluna, [,...]
**from** lista_de_tabelas
[**where** condições_de_pesquisa]

Exemplo:
select pub_name, publishers.pub_id, titles.title_id
from publishers, titles
where publishers.pub_id = titles.pub_id

## “JOINS” baseados em igualdade

Baseado em uma igualdade, entre os valores nas colunas especificadas

Exemplo:
select stores.stor_id, qty, title_id, stor_name
from salesdetail, stores
where publishers.stor_id = store.stor_id
(Quais as vendas de cada loja?)

(usando função agregada e group by)
select stor_name, salesdetail.stor_id, sum(qty)
from salesdetail, stores
where salesdetail.stor_id = stores.stor_id
group by salesdetail.stor_id, stor_name
(Quantos livros cada loja vendeu?)

(ordenando por coluna derivada)

select titles.title_id, qty, price, “PRECO TOTAL” = price * qty

from titles, salesdetail

where titles.title_id = salesdetail.title_id

order by price * qty

## Aliases

Função que fornece um modo abreviado para referenciar tabelas num simples comando SQL.

Sintaxe simplificada:

**select** lista_de_select

**from** nome_de_tabela **alias_1**, nome_de_tabela **alias_2**

**where alias_1**.nome_de_coluna = **alias_2**.nome_de_coluna

Exemplo:

select t.title_id

from titles t.titleauthor ta

where t.title_id = ta.title_id

and au_id = ‘409-56-7008’

(Quais os livros escritos pelo autor com codigo 409-56-7008?)

## “JOINS” não baseados em igualdade

Operadores de comparação usados.

- > maior que
- < menor que
- >= maior igual que
- <= menor igual que

Exemplo:

select ‘ESTADO DO EDITOR’ = p.state, au_lname, au_fname, a.state

from publishers p,authors a

where a.state > p.state and pub_name = “New Age Books”

(Que autores moram num estado cujo o nome é alfabeticamente maior que o estado da editora “New Age Books”?)

## Self-Joins

Unindo uma tabela á ela mesma

- une linhas de uma tabela a outras (ou à mesma) linhas nessa tabela
- Mais de um par de colunas pode ser usado para especificar a condição “join”.
- Existem autores que tenham o mesmo sobrenome?
- “join” não igual normalmente é feito com um “self_join”
    - Duas colunas são usadas na condição (incompleto)

(incompleto)

select au1.au_lname, au1.au_fname, au1.city

from authors au1, authors au2

where au2.au_lname = ‘Karsen’ and au2.au_fname = ‘Livia’

and au1.city = au2.city

(Que autores moram na mesma cidade que Livia Karsen?)

(incompleto)

select a1.au_lname, a1.au_fname

from authors a1, authors a2

where a1.au_lname = a2.au_lname

select a1.au_lname, a1.au_fname

from authors a1, authors a2

where a1.au_id != a2.au_id

(completo e correto)

```
select a1.au_lname, a1.au_fname
from authors a1, authors a2
where a1.au_lname = a2.au_lname and
a1.au_id != a2.au_id
```

(que autores possuem o mesmo sobrenome?)

## Unindo mais de duas tabelas

- A cláusula from deve listar todas tabelas envolvidas.
- A cláusula where deve listar condições **"joins"** suficientes para reunir todas as tabelas em cadeia.
- Você não precisa mostrar as colunas de cada tabela envolvida no "join"

Exemplo:

```
select au_lnane, au_fnarne, titles.title_id, title
from authors, titleauthor, titles
where authors.au_id = titleauthor.au_id
and titleauthor.title_id = titles.title_id
```

(Liste os autores de cada livro.)

## Outer joins (joins externos)

- Inclui linhas não combinadas no resultado, como também as combinadas.
- Operadores do outer join (*6)
  - *= Inclui no resultado todas as linhas da primeira tabela, e não somente aquelas que combinaram, nas colunas reunidas.
  - =* Inclui no resultado todas as linhas da segunda coluna, e não somente aquelas que combinaram, nas colunas reunidas.

Sintaxe simplificada:

**select** lista_do_select

**from** [[banco_de_dados.] **owner**.] {tabela1}, [[banco_de_dados.] **owner**.] {tabela2} [[...]]

[**where** tabela1.nome_de_coluna {*= | =*} tabela2.nome_de_coluna]

Exemplo:

```
select stor_name, sum(qty)
from salesdetail, stores
where stores.stor_id *= salesdetail.stor_id
group by stor_name
```

(assuma que esta sendo aberta uma nova loja, que ainda não vendeu, liste a venda total de cada loja incluindo esta nova loja)

## Escrevendo subqueries

- Um subquery é um comando de seleção, usado como uma expressão, como parte de outro select, update, insert ou comando delete.
- O subquery é resolvido e os resultados são substituidos para dentro do query mais externo.
- Se a cláusula where do query mais externo inclui o nome da coluna, a coluna deve ser compativel com a coluna nomeada no select do subquery
- Subquery não pode ter “order by” ou “cláusula” compute ou palavra chave “into”

Sintaxe simplificada:

**select** lista_do_select

**from** lista_de_tabelas

[**where** condição_de_pesquísa] =

(**select** lista_do_select

**from** lista_de_tabelas

[**where** condição_de_pesquisa]

[**group by** expressão_agregada]

[**having** condição_de_pesquisa]

[**group by** expressão_agregada]

[**having** condição_de_pesquisa]

[**order by** {{tabela.}coluna | expressão}[asc | desc][,…]

[**by** coluna [,…]]

## Razões para usar subqueries

- Muitas uniões podem ser estabelecidas com um subquery
  - Às vezes, é mais fácil entender do que um join que realiza a mesma proposta.
- Executa algumas tarefas que de outro modo seriam impossíveis.

## Subqueries - restrições

- A lista_do_select do subquery pode incluir somente um nome de coluna (exceto para o subquery existente), isto é, mais de um valor de coluna não pode ser retornado.
- A palavra chave “distinct” não pode ser usada com subqueries que tenham a cláusula “group by”.
- As cláusulas “where”, “having” de um comando “select”, “insert”, “update” ou “delete” podem conter um subquery.
- Somente colunas da lista do select do comando mais atual (último comando) podem ser mostradas.

## Múltiplos níveis de aninhamento

- Um subquery pode conter um ou mais subqueries
  - Não existe nível máximo de aninhamento

Exemplo:

(dois níveis)

```
select title
from titles
where title_id = (select title_id
                  from titleauthor
                  where au_id = (select au_id
                                 from authors
                                 where au_lname = 'Blotchet-Halls'))
```

(Que livros Blotchet-Halls escreveu?)

```
select pub_name, pub_id
from publishers
where pub_id = (select pub_id
                from titles
                where price = (select max(price) from titles)
```

(que editor publicou o livro mais caro?)

## Subqueries retornando múltiplas linhas

- Subqueries usadas com "in" or "not in" ao invés de uma igualdade na cláusula "where", pode retomar zero ou mais valores.

Sintaxe simplificada:

**select** lista_do_select

**from** lista_de_tabelas

[**where** condições_de_pesquisa] [**not**] **in** (subquery)

Exemplo:

```
select distinct stor_id, title_id
from salesdetail
where stor_id in (select stor_id
                  from stores
                  where state = "CA")
```

(Que livros foram vendidos na Califórnia CA?)

```
select distinct stor_name
from stores
where stor_id in (select stor_id
                  from salesdetail
                  where title_id in (select title_id
                                     from titles
                                     where title=
                                     'The Busy Executive's Database Guide'))
```

(Que lojas venderam o livro 'The Busy Executive's Database Guide?)

```
select distinct pub_name
from publishers
where pub_id not in (select pub_id
                     from titles
                     where type = 'business')
```

(Que editores não editaram livros da categoria 'business'?)

## Subqueries com operadores de comparação

- O subquery deve retornar só um valor, ao contrário deve existir algum erro

Sintaxe simplificada:

**select** lista_do_select
**from** lista_de_tabelas
[**where** expressão {= | != | < | > | <= | >= }]
(subquery)

Exemplo:

(maior que)

```
select title_id, price
from titles
where price > (select avg(price)from titles)
```

(Que livros custam mais que a média de todos os livros?)

```
select distinct title, advance
from titles
where advance > (select min(advance)
                from titles, publishers
                where titles.pub_id = publishers.pub_id
                and pub_name = “Algodata Infosystems”)
```

(Que livros tem adiantamento superior ao adiantamento mínimo dado pela editora “Algodata lnfosystems”?)

```
select distinct title, advance
from titles
where advance > (select max(advance)
                from publishers, titles
                where title.pub_ib = publishers.pub_id
                and pub_name = “Algodata Infosystems”)
```

(Que livros tem adiantamento superior ao adiantamento máximo dado pela editora “Algodata lnfosystems”?)

## Exists

- Exists and not exists , usado para dois conjuntos de operações de teoria
  - exists - intersecção, todos elementos que pertencem a ambos os conjuntos de origem.
  - not exists - diferença, os elementos que pertencem somente ao primeiro dos dois conjuntos.
  - Palavra chave exists não é precedida por um nome de coluna.

- Normalmente, a lista_do_select do subquery será “*“, desde que a função exists retome verdadeiro ou falso e não dados reais.

| | | |
|---|---|---|
| * **exists** (subquery) | True | se linhas são retomadas |
| | False | se linhas não são retornadas |
| * **not exists** (subqueries) | True | se linhas não são retornadas |
| | False | se linhas são retornadas |

Sintaxe simplificada:
**select** lista_do_select
**from** lista_de_tabelas
[where {exists | not exists}]
(subquery)
(intersecção entre autores e editores)

```
select distinct city
from authors
where exists (select *
        from publishers
        where authors.city = publisliers.cíty)
```

(Que autores moram na mesma cidade que editores?)

(diferença entre autores e editores)

```
select distinct city
from authors
where not exists (select *
        from publishers
        where authors.cíty = publishers.city)
```

(Que autores não moram na mesma cidade que editores?)

## Unions

- Deixa você combinar os resultados de mais de um select

Exemplo:

```
select city, state from publishers
union
select city, state from stores
union
select city, state from stores
order by state, city
```

- Descarta as linhas duplicadas (a não ser que você use a opção “all “)

Sintaxe geral:

Query 1
[**union** [**all**] Query n]…
[**order by** cláusula]
[**compute** cláusula]
onde Query 1 é: **select** lista_do_select
[**into** cláusula]
**from** lista_de_tabelas
[**where** clausula]
[**group by** cláusula]
[**having** cláusula]
onde Query n é: **select** lista_do_select
**from** lista_de_tabelas
[**wbere** cláusula]
[**group by** cláusula]
[**having** cláusula]

- Qualquer número de operadores de uniões podem aparecer no comando transact-SQL.

- Operadores união são executados da esquerda para direita.

Por exemplo:

X union all (Y union Z)

não é equivalente a

(X union all Y ) union Z

Use parênteses para controlar a ordem de execução

- Todas listas de select no comando “uníon” devem ter o mesmo número de expressões.
- As expressões nas listas de select são arranjadas por ordem.
- Expressões correspondentes nas listas de select devem ser do mesmo tipo de dados, ou uma conversão de dados ímplicita deve ser possível, ou uma conversão de dados explícita deve ser especificada.
- Os nomes de coluna do resultados são obtidas do primeiro query da união.
  - Especifique nomes de coluna somente no primeiro query
- “order by” e “compute” são permitidos somente depois de todas queries, não podem ser dentro de queries individuais.
- group by e having só podem aparecer em queries individuais, não podem ser usados para o resultado final.
- A cláusula “into” só pode ser usada no primeiro query, para criar como resultante, uma nova tabela.

## Criando uma tabela a partir de outra existente

- A palavra chave into é usada para criar uma tabela a partir de outra existente.

Sintaxe simplificada:

**select** lista_do_select

[**into** tabela]

**from** lista_de_tabelas

[**where** condição_de_pesquisa]

Exemplo:

(criação de tabela carregando os dados)

```
select *
into new_titles
from titles
```

(criação de tabela não carregando os dados)

```
select *
into new_pubs
from publishers
where 1 = 2
```

(criação de tabela com carga limitada de dados)

```
select *
into pub_1389_titles
from titles
where pub_id = '1389'
```

(criação de tabela eliminando colunas)

```
select pub_name, pub_id
into short_pubs
frorn publishers
```

## Modificando dados – incluindo dados

- A cláusula insert adiciona dados numa tabela existente
  - O SQL server verifica em tempo de entrada de dados, se o valor inserido é do tipo de dados correto.
    - Se o valor é maior (muito longo) o SQL trunca para o tamanho especificado; não da mensagem de warning (aviso).
  - Algumas colunas são definidas para permitir valores nulos.
    - Para que uma coluna permita valores nulos, especifique null para inserir um valo nulo.

- Entrar com datatime e todos valores de caractere entre aspas simples ou duplas.
    - Para entrar com um string de caracteres maior que uma linha use uma barra invertida (\) antes de ir para próxima linha.
    - Para entrar com aspas como se fossem um literal use o tipo oposto de aspas,isto é, se uma aspa simples é o literal, use aspas duplas para delimitar o valor de dados total a ser inserido.

Sintaxe simplificada:
**insert** [into] tabela
[(lista_de_colunas)]
{**values** (constantes) | comando_select}
(Adiciona uma linha simples.)

Exemplo:
(linha completa)
insert into new_pubs
value (‘9945’,’Mysteries Galore’,’Kansas City’, ‘KS’)

(inserindo uma aspa)
insert into new_pubs
value (‘4444’, “O’bryan Publishing House”, ‘Washington’,’DC’)

- Qualquer coluna não incluída deve permitir nulos.

Exemplo:
(linha parcial)
insert into new_pubs(pub_id,pub_nane)
values (‘3333’,”Jardin’s mc”)

- Se a lista de colunas é omitida, os valores devem estar na mesma ordem que os nomes de coluna no comando create table.

(inserindo dados a partir de outra tabela)

```
insert into new_pubs(pub)id, pub_name, city, state)
select pub_id, pub_name, city, state
from publisher
```

(inserindo usando uma subquerie)

```
insert into new_pubs
select *
from mass_publishers
where pub_id in (select pub_id
                 from mass_publishers
                 where city = “Boston”)
```

## Modificando dados – atualizando dados

- A cláusula update altera o valor de uma ou mais colunas em uma tabela ( altera os valores dos dados nas linhas existentes,coluna por coluna).
  - **set** cláusula que determina as colunas a serem alteradas
  - **from** cláusula que determina de que tabelas os dados se originam, se mais de uma tabela é usada para determinar o valor de uma nova coluna.
  - **where** cláusula que determina que linhas devem ser alteradas.
  - **não podem** ser alteradas várias tabelas no mesmo comando update.

Sintaxe simplificada:

**update** tabela
**set** nome_de_coluna = {expressão | **null**}
[,nome_de_coluna = {expressão | **null**}]...
[**from** tabela1, tabela2, ...]
[**where** condições_de_pesquísa]

(alteração de dados baseado em constantes)

```
update new_ubs
set pub_name = “New Publishers Name”
where pub_name = “New Age Books”
```

(alteração de dados para as linhas que satisfaçam a condição)

```
update new_titles
set contract = 0
where pub_id = '1389'
```

(alteração para todas as linhas incondicionalmente)

```
update new_titles
set advance = 0
```

(alteração dos dados de uma tabela baseados nos dados de outra tabela)

```
update new_titlies
set contract = 0
from new_titles, publishers
where new_titles.pub_id = publishers.pub_id and
pub_name = 'New Age Books'
```

(alteração usando sub quety,)

```
update new_titlies
set total_sales = (select sum(qty)
                   from salesdetail
                   where title_id = 'BU1032')
```

## Modificando dados – deletando dados

- A cláusula "delete" exclui dados selecionados de unia tabela.
  - A palavra chave from (depois do delete) é opcional
  - A palavra chave from (sintaxe de extensão) permite que dados sejam deletados de uma tabela baseados em dados guardados em outras tabelas.
  - A cláusula where determina que linhas serão deletadas; se omitida todas as linhas serão deletadas.

Sintaxe simplificada:

**delete** [from] tabela

[from tabela] [,...]

[wbere condições_de_pesquisa]

Exemplo:

(remove dados de uma tabela baseados em uma constante)

```
delete from new_pubs
where pub_name = ‘Mysteri’

delete from new_titles
where price > $15
```

(remove dados de uma tabela baseados em outra tabela)

```
delete new_titles
from new_titles, new_pubs
where new_titles.pub_id = new_pubs.pub_id and pub_name = “Jardin’s”
```

(remove dados de uma tabela usando subqueries)

```
delete from new_titles
where title)id in (select title_id
                   from titleauthor
                   where au_id in (select au_id
                                   from authors
                                   where state = ‘UT’))
```

## Removendo definições de dados

- Drop table remove a definição de uma tabela e todos os dados do BD

Sintaxe simplificada:

**drop table** tabela