A Linguagem SQL

Prof.: Leandro Clementino Almeida

Introdução

- Desenvolvida pelo depto de pesquisa da IBM na década de 1970 (System R) - Sequel
- Linguagem padrão de BD Relacionais
- Apresenta várias padrões evolutivos: SQL86, SQL89(SQL1), SQL92 (SQL2), SQL99(SQL3)
- A última versão definida pela ANSI/ISO traz características novas como: store procedures, triggers, suporte à programação OO, XML, entre muitas outras (SQL2003)
- Diferentes fornecedores de SGBDS apresentam versões de SQL com algumas particularidades

Características Comuns

- Estilo declarativo, não procedimental
- Permite otimizações
- Utilizadas por várias classes de usuários
- Sintaxe simples e bem definida
- Presente em todos os SGBDs Relacionais
- É incorporada comumente a uma outra linguagem
- Não é uma linguagem completa como C, Java ou Delphi
- Portável entre sistema operacionais

DDL – Definição de Dados

- Para todos comandos da DDL:
  - CREATE- Cria uma definição
  - ALTER - Alterar alguma definição
  - DROP - Exclui uma definição
- Alguns comandos da DDL
  - TABLE - Define uma tabela, seus campos e as restrições de campo
  - INDEX - Define um índice associado a um campo de uma tabela
  - GRANT - Define usuários e autorizações para cada um
  - EXCEPTION - Define uma mensagem de erro

DDL – Definição de Dados

- TRIGGER - Define um conjunto de instruções que são automaticamente executadas antes ou depois de um comando INSERT, UPDADE ou DELETE
- PROCEDURE - Define um conjunto de instruções. Podem receber ou retornar valores. Podem ser executadas através de uma solicitação do usuário ou por um TRIGGER

Comando CREATE DATABASE

- CREATE DATABASE - Cria um novo banco de dados ou esquema

  CREATE {DATABASE | SCHEMA} <nome do banco>

Comando CREATE TABLE

- CREATE TABLE - Cria uma nova tabela com seus campos e define as restrições de campo.

  CREATE TABLE <nome da tabela> (

  <**definição de coluna** 1>

  ...

  <definição de coluna N>

  <restrições de integridade>

  );

- Onde <**definição de Coluna**> pode ser

  <nome atributo> <**tipo de dado**> [NULL | NOT NULL | USER | DEFAULT <default- value> | COMPUTED BY <expression>

Tipos de Dados

- <tipo de dado> pode ser:
  - {SMALLINT | INTEGER | FLOAT | DOUBLE PRECISION}
  - | {DECIMAL | NUMERIC} [(precision [, scale])]
  - | DATE
  - | {CHAR | CHARACTER | CHARACTER VARYING | VARCHAR}
  - [(int)]
  - | BLOB

Tipos de Dados

- CHAR(n):
  - Tamanho de armazenamento n bytes. Para armazenamento de dados textuais (caracter e número). O número máximo de caracteres é definido por n e deve estar entre 1 e 32.767. Reserva previamente o tamanho definido por n, mesmo que o dado armazenado não preencha totalmente o campo.
- VARCHAR(n) ou CHAR VARYING(n) ou CHARACTER VARYING(n).
  - Tamanho de armazemanento n bytes. Para armazenamento de dados textuais (caracter e número). O número máximo de caracteres é definido por n e deve estar entre 1 e 32.765. Armazena somente a quantidade de caracteres que conter o dado, no máximo o valor n.

Tipos de Dados

- DECIMAL (precisão, decimal) ou NUMERIC (precisão, decimal)
  - Tamanho de armazemanento variável. Para armazenamento de valores decimais. Precisão define a quantidade de digitos numérico máximo para o dado e decimal a quantidade de números decimais.
  - Por exemplo: DECIMAL (10,3) indica que o dados pode Ter no máximo 7 dígitos inteiros e 3 dígitos decimais.
- DOUBLE PRECISION
  - Tamanho de armazemanento 64 bits (depende da plataforma). Para armazenamento de números no intervalo de 1.7×10^{-308} até 1.7×10^{308} com precisão de 15 dígitos.
- FLOAT
  - Tamanho de armazemanento 32 bits. Para armazenamento de números no intervalo de 3.4×10^{-38} até 3.4×10^{38} com precisão de 7 dígitos.

Tipos de Dados

- INTEGER
  - Tamanho de armazemanento 32 bits. Para armazenamento de números inteiros no intervalo de – 2.147.483.648 to 2.147.483.647.
- SMALLINT
  - Tamanho de armazemanento 16 bits. Para armazenamento de números inteiros no intervalo de – 32768 to 32767.
- DATE
  - Tamanho de armazenamento 64 bits. Para armazenamento de data e hora. O intervalo de datas válidas é 1/Jan/100 até 11/Jan/5941, incluso tempo.
- BLOB.
  - Tamanho de armazenamento variável. Para armazenamento de grande quantidade de dados como áudio, vídeo, gráficos,etc.

DDL - CREATE TABLE

- **Exemplo:**
  - /* Cria tabela aluno */
  - CREATE TABLE Aluno (
    nome varchar(40) NOT NULL,
    RA decimal(8) NOT NULL,
    idade decimal(2) DEFAULT 18,
    cidade varchar(30),
    ID_Responsavel varchar(20) DEFAULT USER
  -) ;

DDL – Comando CREATE TABLE

- CREATE TABLE - Cria uma nova tabela com seus campos e define as restrições de campo.

```
CREATE TABLE <nome da tabela> (
    <definição de coluna 1>
    ...
    <definição de coluna N>
    <restrições de integridade>
);
```

DDL – Restrições de Integridade

- Valor Nulo
  - Representado por NULL
  - Membro de todos os domínios
- Restrição NOT NULL
  - Especificada quando NULL não é permitido
  - Proíbe que o atributo receba valor nulo
- Comparações
  - Usar IS NULL e IS NOT NULL

DDL – Restrições de Integridade

- Cláusula PRIMARY KEY
  - Identifica os atributos da relação que formam a as chave primária
    - Os atributos devem ser definidos como NOT NULL
  - Sintaxe:
    - PRIMARY KEY (atributo1,atributo2,..,atributoX)
- Cláusula UNIQUE
  - Não permite valores duplicados para um determinado atributo
- Cláusula DEFAULT
  - Associa um valor Default para um atributo, caso nenhum outro valor seja especificado

DDL –CREATE TABLE

- **Exemplo:**
  - /* Cria tabela TURMA */

  - CREATE TABLE Turma (
  - sigla char(7) NOT NULL,
  - numero decimal(2) NOT NULL,
  - codigo decimal(4) NOT NULL PRIMARY KEY,
  - NNalunos decimal(3),
  -);

Chaves Primárias Composta

```
/* Cria tabela ITEM_DE_PEDIDO*/

CREATE TABLE ITEM_DE_PEDIDO (
   numeropedido Integer NOT NULL,
   codigoproduto integer NOT NULL,
   PRIMARY KEY (numeropedido, codigoproduto)
   );
```

DDL – Restrições de Integridade

- Cláusula CHECK
  - Especifica um predicado que precisa ser satisfeito por todas as tuplas de uma relação
  - Exemplos:
    - CHECK (saldo >= 0)
    - CHECK (nivel IN ´Bacharelado´, ´Mestrado´, ´Doutorado´)
- Integridade Referencial
  - Dependência existente entre a chave estrangeira de uma relação e a chave primária da relação relacionada
  - Problemas:
    - Atualização ou exclusão de elementos da chave primária sem fazer um ajuste coordenado nas chaves estrangeiras
    - Inclusão ou alteração de valores não nulos na chave estrangeira que não existam na chave primária.

DDL – Restrições de Integridade

- Cláusula FOREIGN KEY
  - Características:
    - Elimina a possibilidade de violação da integridade referencial
    - Reflete nas chaves estrangeiras todas as alterações na chave primária
  - Sintaxe:
    - FOREIGN KEY (atributos)
      REFERENCES nome relação (atributos)
      ON { UPDATE | DELETE } { CASCADE | SET NULL | SET DEFAULT }

DDL – Restrições de Integridade

- <Restrições de Integridade> podem ser:
  - [CONSTRAINT <nome da restrição>]
  - PRIMARY KEY (<Atributo>,...)
  - | UNIQUE (<Atributo>,...)
  - | FOREIGN KEY (< <Atributo>,... >) REFERENCES <tabela> (<Atributos chaves>, ...) [<**ações**>]
  - | CHECK (<condição>)
  - Onde <**ações**> podem ser:
  - ON { UPDATE | DELETE }
  - { CASCADE | SET NULL | SET DEFAULT }

DDL –CREATE TABLE

- **Exemplo**:
  - /* Cria tabela TURMA */
  - CREATE TABLE Turma (
  - sigla char(7) NOT NULL,
  - numero decimal(2) NOT NULL,
  - codigo decimal(4) NOT NULL PRIMARY KEY,
  - NNalunos decimal(3),
  - FOREIGN KEY (Sigla) REFERENCES Discip (Sigla)
  - ON DELETE CASCADE
  - ON UPDATE CASCADE,
  - CONSTRAINT SiglaNumero UNIQUE (Sigla, Numero),
  - CONSTRAINT LimiteDeVagas CHECK (NNalunos < 50)
  -);

DDL – CREATE TABLE

- **Exemplo**:
  - /* Cria tabela PRODUTOS*/

  - CREATE TABLE Produtos (
  - Codigo Integer NOT NULL PRIMARY KEY,
  - Descricao CHAR(40) NOT NULL,
  - Preco NUMERIC(18,4) NOT NULL,
  - Estoque INTEGER NOT NULL,
  - ValEstoque COMPUTED BY (Preco * Estoque)
  - DataCadastro DATA DEFAULT ‘TODAY’,
  - Fornecedor INTEGER
  -);

DDL – DROP TABLE

- DROP TABLE <nome tabela>
  - Remove uma tabela existente no BD
    - Dados, índices, metadados, gatilhos que referenciam a tabela
  - Usuários autorizados
    - Proprietário do banco de dados
    - DBA ou usuário com privilégio de root
  - Exemplo:

    Drop table Produtos

DDL – ALTER TABLE

- ALTER TABLE <nome tabela>
  - Altera a estrutura de uma tabela
    - Colunas podem ser acrescentadas, modificadas ou excluídas
  - Incluir novo atributo na tabela – novas colunas

    ALTER TABLE PRODUTOS

    ADD Categoria INTEGER NOT NULL
  - Elimina um atributo

    ALTER TABLE PRODUTOS

    DROP Categoria
  - Modifica o nome de um atributo

    ALTER TABLE PRODUTOS

    ALTER COLUMN Categoria TO Categ

DDL – ALTER TABLE

- DROP TABLE <nome tabela>
  - Modifica o tipo de dado

```
ALTER TABLE PRODUTOS
    ALTER COLUMN Categ Type CHAR(15)
```

  - Adição de uma chave estrangeira

```
ALTER TABLE PRODUTOS
ADD CONSTRAINT FK_Fornec FOREIGN KEY
  (fornecedor) REFERENCES Fornecedor(codigo)
```

Chaves Primárias Composta

```
/* Cria tabela ITEM_DE_PEDIDO*/

CREATE TABLE ITEM_DE_PEDIDO (
   numeropedido Integer NOT NULL,
   codigoproduto integer NOT NULL,
   Constraint PK_itemPedido
        PRIMARY KEY (numeropedido, codigoproduto)
   Constraint FK_Pedido FOREIGN KEY (numeropedido)
REFERENCES PEDIDO
   Constraint FK_Produto FOREIGN KEY (codigoproduto)
REFERENCES PRODUTO
   );
```

DDL – CREATE INDEX

- CREATE [UNIQUE] INDEX <nome índice> ON <Tabela> (atributo1, atributo2)
  - Modo comum de melhorar o desempenho do banco de dados
  - Permite ao servidor de banco de dados encontrar e trazer linhas específicas muito mais rápido do que faria sem o índice
  - Produzem trabalho extra para o sistema de banco de dados como um todo devendo, portanto, serem utilizados com sensatez.

CREATE INDEX descricao on produtos (Descricao)

DDL – ALTER INDEX

- ALTER INDEX <nome indice> {Active, Inactive}
  - Torna um índice existente:
    - Ativo
    - Inativo
    - Torna um índice inativo e depois ativá-lo novamente gera a reconstrução e o balanceamento do índice

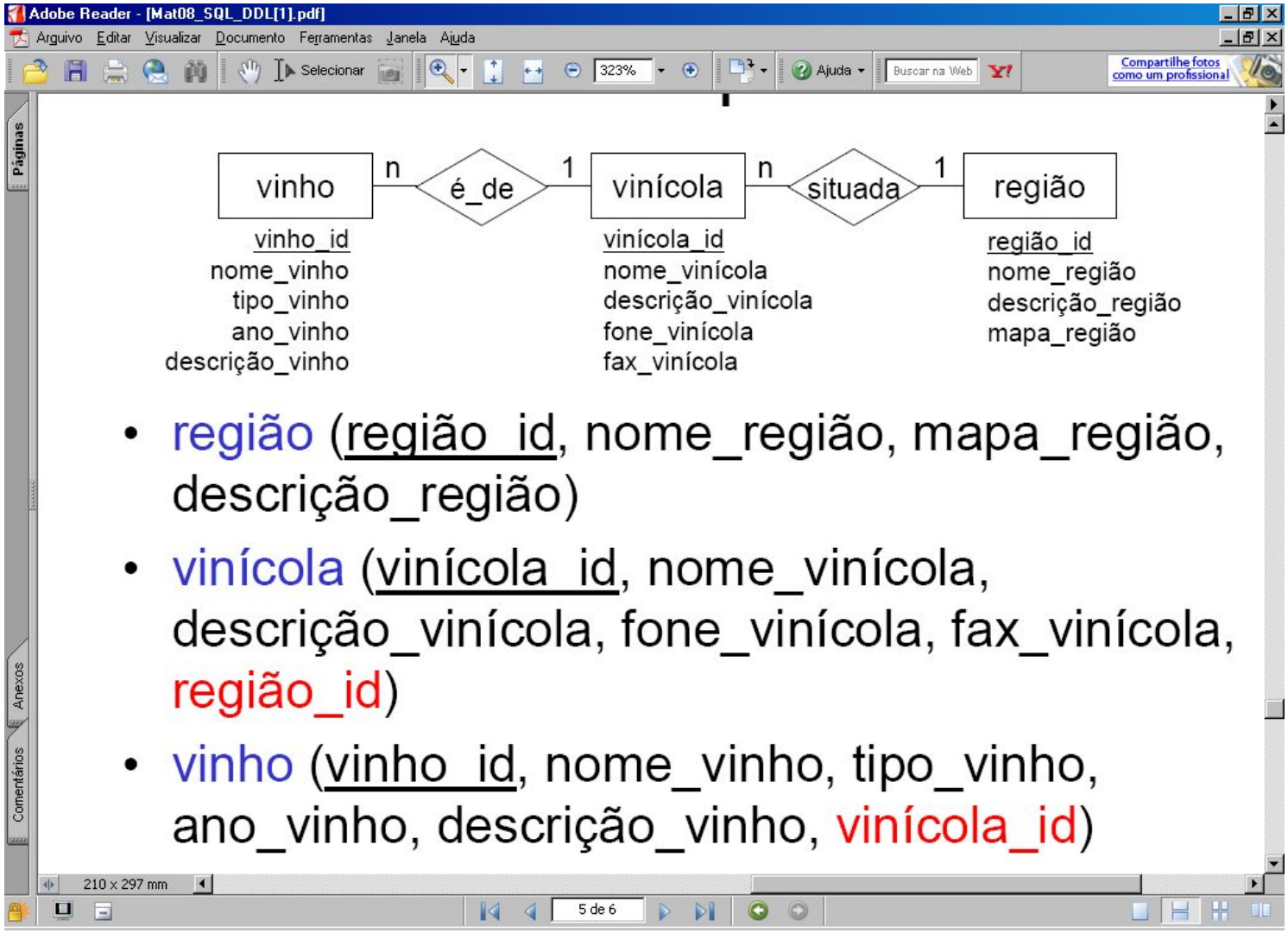
ALTER INDEX descricao Inactive

DDL – DROP INDEX

- DROP INDEX <nome indice>
  - Remove um índice existente no BD
    - Falha caso o domínio esteja definindo o tipo de dado de algum atributo
    - Usuários autorizados:
      - Proprietário do banco de dados
      - DBA ou usuário com privilégio de root

DROP INDEX descricao

Exercício


Adobe Reader - [Mat08_SQL_DDL[1].pdf]
Arquivo Editar Visualizar Documento Ferramentas Janela Ajuda
Selecionar
323%
Ajuda
Buscar na Web
Compartilhe fotos como um profissional
Páginas
Anexos
Comentários
vinho
n
é_de
1
vinícola
n
situada
1
região
vinho_id
nome_vinho
tipo_vinho
ano_vinho
descrição_vinho
vinícola_id
nome_vinícola
descrição_vinícola
fone_vinícola
fax_vinícola
região_id
nome_região
descrição_região
mapa_região
• região (região_id, nome_região, mapa_região, descrição_região)
• vinícola (vinícola_id, nome_vinícola, descrição_vinícola, fone_vinícola, fax_vinícola, região_id)
• vinho (vinho_id, nome_vinho, tipo_vinho, ano_vinho, descrição_vinho, vinícola_id)
210 x 297 mm
5 de 6

DML– INSERT

- INSERT INTO - Insere tuplas em uma relação.

- Formato 1: Insere uma tupla de cada vez.

```
INSERT INTO <Tabela> [( <Atributo>, ... )] VALUES ( expression | DEFAULT, ... );
```

- Formato 2: Insere múltiplas tuplas a partir de uma tabela.

```
INSERT INTO <Tabela> [( <Atributo>, ... )] <Comando SELECT>;
```

DML– INSERT

- Exemplo:

- Formato 1: Insere uma tupla de cada vez.

```
INSERT INTO Professor VALUES ('Antonio', '5656',
'MS-3', 33);
```

```
INSERT INTO Professor ( Nome, Grau, NNfuncional)
VALUES ('Antoninho', 'MS-3', '5757');
```

```
INSERT INTO Professor (Nome, Grau, Nnfuncional,
data_nasc) VALUES ('Antoninho', 'MS-3',
'5757‘,’01/20/1955’);
```

DML– INSERT

- Exemplo:

- Formato 2: Insere múltiplas tuplas a partir de uma tabela.

```
INSERT INTO pessoa ( Nome, Idade ) SELECT
nome, idade from aluno;
```

```
INSERT INTO Prudentinos FROM aluno WHERE
cidade =  “Presidente Prudente”;
```

DML– UPDATE

- UPDATE - Altera o valor de atributos de tuplas de uma relação.

- `UPDATE <tabela> SET <Atributo> = <expressão>, ...`
- `[ WHERE <Condição> ]`

- onde:

- <expressão> = {<Atributo>| <constante> | <expr> | NULL |
- USER | ?}

- <expr> = Qualquer comando SELECT que resulte em apenas
- uma tupla e uma coluna.

DML– UPDATE

- Exemplo:

- /*Todas as tuplas da relação têm idade incrementada de um (1)*/

- `UPDATE Aluno SET Idade=Idade+1;`

DML– DELETE

- DELETE FROM – Apaga tupla(s) de uma relação

- `DELETE FROM <tabela> [WHERE <Condição>]`

DML– DELETE

- Exemplos

- /* Apaga tupla cujo RA vale 1234 */
- `DELETE FROM Aluno WHERE RA=1234;`

- /*Remover todos os Alunos da cidade de São Paulo */
- `DELETE FROM Aluno WHERE Cidade = 'São Paulo'`

- /* Apagar todas as tuplas (linhas) da relação (tabela) */
- `DELETE FROM Aluno;`

DML– SELECT

- SELECT - Procura na tabela ou tabelas especificadas, extrai as colunas escolhidas, seleciona as linhas que atendem aos critérios e classifica ou agrupa as linhas resultantes na ordem especificada.

```
SELECT [ ALL | DISTINCT ] <lista de atributos>
FROM <lista de Tabelas>
[ WHERE <condição> ]
[ GROUP BY <Atributo>, ...
[ HAVING <condição> ] ]
ORDER BY <Lista de atributos> [ ASC | DESC ], ...]
```

DML– SELECT

- SELECT - Procura na tabela ou tabelas especificadas, extrai as colunas escolhidas, seleciona as linhas que atendem aos critérios e classifica ou agrupa as linhas resultantes na ordem especificada.

```
SELECT [ ALL | DISTINCT ] <lista de atributos>
FROM <lista de Tabelas>
[ WHERE <condição> ]
[ GROUP BY <Atributo>, ...
[ HAVING <condição> ] ]
ORDER BY <Lista de atributos> [ ASC | DESC ], ...]
Onde:
<Lista de Tabelas> = <joined_table> | <Tabela>
<joined_table> = <Tabela> <join-type> JOIN <Tabela> ON
<condição> | ( <joined_table>)
<join-type> = {[INNER] | {LEFT | RIGHT | FULL}[OUTER]}
```

DML– SELECT

- <condição> = {
- <val> <operator> <val>
- | <val> [NOT] BETWEEN <val> AND <val>
- | <val> [NOT] LIKE <val> [ESCAPE <val>]
- | <val> [NOT] IN (<val> [, <val> …] | <select_list>)
- | <val> IS [NOT] NULL
- | <val> {[NOT] {= | < | >} | >= | <=} {ALL | SOME | ANY} (<select_list>)
- | EXISTS (<select_expr>)
- | SINGULAR (<select_expr>)
- | <val> [NOT] CONTAINING <val>
- | <val> [NOT] STARTING [WITH] <val>
- | (<CondiçãoJunção>)
- | NOT <CondiçãoJunção>
- | <condição> OR <condição>
- | <condição > AND <condição>

DML– SELECT

- **ALL:** indica se um valor é igual a todos os valores retornados por uma subquery.
- **SOME:** indica se um valor é igual a qualquer valor retornado por uma subquery.
- **ANY:** (idem a SOME)
- **EXISTS:** indica se ao menos um valor é retornado pela subquery.
- **SINGULAR:** indica se UM e SOMENTE UM valor é retornado pela subquery.

DML– SELECT

- Exemplos:

- /* Selecionar todos os campos na tabela ALUNO */
- `SELECT * FROM ALUNO;`

- /* Selecionar todos RA e nomes da tabela ALUNO */
- `SELECT RA, Nome FROM ALUNO;`

- /* Selecionar todos os nomes e RA da tabela ALUNO,
- renomeando a coluna nome para “aluno” */
- `SELECT RA, Nome AS Aluno FROM ALUNO;`

DML– SELECT

- Cláusula WHERE: Especifica quais registros das tabelas listadas na cláusula FROM são afetados por uma instrução SELECT, UPDATE ou DELETE. Se você não especificar uma cláusula WHERE, a consulta retornará todas as linhas da tabela.

- /* Selecionar todos os alunos cuja idade seja maior que 22 */

```
SELECT Nome, Idade FROM ALUNO
  WHERE Idade > 22;
```

DML– SELECT

- /* Selecionar todos alunos matriculados a partir de 30/01/2006 */
- `SELECT NOME FROM MATRICULA`
- `  WHERE Data > '30/01/2006';`

- /* Selecionar todos alunos matriculados a partir de 30/01/2006 */
- `SELECT M.RA, A.Nome`
- `FROM MATRICULA M, ALUNO A`
- `WHERE M.RA = A.RA AND M.Data > '30/01/2006';`

SELECT CLÁUSULA GROUP BY

- Cláusula GROUP BY: Combina registros com valores idênticos na lista de campos especificada em um único registro. Um valor de resumo é criado para cada registro se você incluir uma função agregada SQL, como **`Sum`** ou **`Count`**, na instrução SELECT.

- /* Selecionar todos alunos matriculados em alguma turma */

```
SELECT Aluno.RA, Aluno.Nome
FROM Aluno, Matricula
WHERE Aluno.RA=Matricula.RA
GROUP BY Aluno.Nome;
```

SELECT CLÁUSULA HAVING

- Cláusula HAVING: Especifica quais registros agrupados são exibidos na instrução SELECT com uma cláusula GROUP BY. Depois de GROUP BY combinar os registros, HAVING exibirá qualquer registro agrupado pela cláusula GROUP BY que satisfaça às condições da cláusula HAVING.

- /* Listar todas as turmas que possuem mais do que 20 alunos matriculados*/

```
SELECT Turma.Codigo
FROM Turma, Matricula
WHERE Matricula.CodigoTurma = Turma.Codigo
GROUP BY Turma.Codigo
HAVING COUNT(Matricula.RA)>20;
```

SELECT CLÁUSULA ORDER BY

- Cláusula ORDER BY: Classifica os registros resultantes de uma consulta em um campo ou campos especificados, em ordem crescente ou decrescente. Os registros são classificados pelo primeiro campo listado após ORDER BY. Os registros que têm valores iguais naquele campo serão então classificados pelo valor no segundo campo listado e assim por diante.

- /* Listar todos alunos ordenando-os descendentemente por idade e depois por nome */

```
SELECT Nome, Idade
FROM Aluno
ORDER BY Idade DESC, Nome DESC;
```

SELECT OPERADOR IN

- Operador IN: Determina se o valor de uma expressão é igual a algum dos vários valores em uma lista especificada. Se expr for encontrado na lista de valores, o operador IN retornará True; caso contrário, retornará False.

- /* Listar todos os alunos provenientes de São Paulo, Bauru ou Rio de Janeiro*/

```
SELECT *
FROM ALUNO
WHERE UPPER(Cidade) IN ('SAO PAULO',
'BAURU', 'RIO DE JANEIRO');
```

SELECT OPERADOR BETWEEN

- Operando BETWEEN...AND: Determina se o valor de uma expressão se situa dentro de um intervalo especificado de valores. Se o valor de expr estiver entre <valor1> e <valor2> (inclusive), o operador BETWEEN...AND retornará True; caso contrário, retornará False.

- /* Listar todos alunos com idade entre 20 e 22 anos*/

```
SELECT Nome, Idade
FROM Aluno
WHERE Idade BETWEEN 20 AND 22;
```

SELECT OPERADOR LIKE

- Operando LIKE: Compara uma expressão de seqüência com um padrão em uma expressão SQL. Para padrão, você pode utilizar caracteres curinga (por exemplo, Like ‘Comp%’, para ‘Computação’) ou utilizar caracteres isolados (por exemplo, Like ‘_OSE’, para ‘JOSE’ e ‘ROSE’)

- /* Listar todos os alunos cujo nome termina em ‘ina’, ignorando as 3 primeiras letras */

```
SELECT Nome
FROM Aluno
WHERE UPPER(Nome) LIKE '___INA'; /* São 3 ‘_’ */
```

SELECT OPERADOR LIKE

- LIKE ´%ão_´ qualquer nome que termine em “ão”
- LIKE ´[CM]%´ permite enxergar qualquer nome que comece com ´C´ ou com ´M´
- LIKE ´[C-X]%´qualquer nome que comece cp, ´C´ até ´X´
- LIKE ´M[^o]%´ qualquer nome que comece com ´M´e não tenha ´o´como segunda letra

```
SELECT codigo_vendedor, nome_vendedor
FROM vendedores
WHERE nome_vendedor NOT LIKE ´Jo%´
```

SELECT OPERADOR IS NULL

- Operando IS NULL: Determina se o valor de uma expressão é nulo

- /* Listar todas as disciplinas que NÃO possuem pré-requisito */
- `SELECT Nome`
- `FROM Discip`
- `WHERE SiglaPreReq IS NULL;`

SELECT FUNÇÃO AVG

- Função AVG(): Calcula a média aritmética de um conjunto de valores contido em um campo especificado em uma consulta.
-
- /* Calcular a idade média dos alunos cadastrados */

```
SELECT AVG(Idade) AS Media
FROM ALUNO;
```

SELECT FUNÇÃO COUNT

- Função **COUNT():** Calcula o número de registros retornado por uma consulta. A função COUNT não conta registros que tenham campos Null, exceto quando expr for o caractere curinga asterisco (*).

- /* Contar quantas matrículas existem */
- `SELECT COUNT(RA)`
- `FROM MATRICULA;`

- /* Contar quantos alunos se matricularam em pelo menos uma
- disciplina */
- `SELECT COUNT(DISTINCT RA)`
- `FROM MATRICULA;`

SELECT FUNÇÃO COUNT

- Função **COUNT():** Calcula o número de registros retornado por uma consulta. A função COUNT não conta registros que tenham campos Null, exceto quando expr for o caractere curinga asterisco (*).

- /* Contar quantas disciplinas existem */
- `SELECT COUNT(*)`
- `FROM Discip;`

- /* Contar quantas disciplinas possuem pré-requisito */
- `SELECT COUNT(SiglaPreReq)`
- `FROM Discip;`

- /* Contar quantos alunos existem por turma*/
- `SELECT CodigoTurma, COUNT(RA)`
- `FROM Matricula`
- `GROUP BY CodigoTurma;`

SELECT FUNÇÃO MIN/MAX

- Função MIN(): Retorna o mínimo de um conjunto de valores contido em um campo especificado em uma consulta.
- Função MAX( ): Retorna o máximo de um conjunto de valores contido em um campo especificado em uma consulta.

- /* Verificar qual é a idade do aluno mais velho */
- `SELECT MAX(Idade)`
- `FROM Aluno;`

- /* Verificar o(s) nome(s) e a idade do(s) aluno(s) mais novo(s) */
- `SELECT Nome, Idade`
- `FROM Aluno`
- `WHERE Idade IN (SELECT MIN(Idade) FROM ALUNO);`

SELECT FUNÇÃO SUM

- Função SUM(): Retorna a soma de um conjunto de valores contido em um campo especificado em uma consulta. A função Sum ignora os registros que contenham campos Null.

- /* Calcular a quantidade de créditos oferecidos na Universidade */
- `SELECT SUM(NNCred) AS TotalCreditos`
- `FROM DISCIP;`

- /* Calcular a quantidade de alunos por disciplina */
- `SELECT Sigla, SUM(NNAlunos) AS TotalAlunos`
- `FROM TURMA`
- `GROUP BY Sigla;`