LINGUAGEM SQL

Structured Query Language (SQL)

PÓS GRADUAÇÃO EM BANCO DE DADOS

Introdução

- Linguagem de consulta mais usada em SGBDs relacionais.
  - Dois tipos principais de instruções:
    - *Data Definition Language* (DDL)
      - Criação de tabelas, restrições de integridade, etc.
    - *Data Manipulation Language* (DML)
      - Consulta, inserção, remoção, alteração de registros.

Criação de tabelas (*DDL*)

- Os passos principais na criação de uma relação (tabela) envolvem a escolha do seu nome, e dos campos (nomes e tipo de dados associados).

- Tipos de dados principais do ANSI SQL99:
  - Cadeias de caracteres *(string*s)
    - Sequências de caracteres englobados por aspas simples (ex. ‘carlos‘).
    - strings em SQL são de tamanho máximo fixo.
    - CHAR(n) denota uma string de tamanho fixo n.
    - VARCHAR2(n) denota uma string com no máximo *n* caracteres.
  - Números inteiros
    - NUMBER(n,d) representa os números inteiros e reais que correspondem a números com no máximo *n* dígitos, sendo *d* casas decimais.
  - Datas e horas
    - DATE representa dados do tipo data (inclusive hora)
    - Os valores concretos deste tipo são representados usando a sintaxe definida pelo banco de dados (NLS_DATE_FORMAT)

Criar uma tabela

- A criação de tabelas em SQL faz- se com a instrução:

  CREATE TABLE *nome_ da_ tabela* (lista de declarações de campos)

Exemplos:

```
CREATE TABLE VENDA
(CODVENDA          NUMBER(9),
 DATAVENDA         DATE,
 DATAENTREGA       DATE,
 VALORTOTAL        NUMBER(10,2),
 CODCLIENTE        NUMBER(6),
 CODVENDEDOR       NUMBER(6));

CREATE TABLE CLIENTE
(CODCLIENTE        NUMBER(6),
 NOME          VARCHAR2(40),
 ENDERECO          VARCHAR2(80),
 TIPOCLIENTE       NUMBER(1));
```

Excluir e alterar tabelas

- Alterar o esquema relacional de uma tabela

  ALTER TABLE *R* [ADD *campo* | MODIFY *campo* | DROP COLUMN *campo*]

  ALTER TABLE venda ADD (observacao VARCHAR2(255));
  ALTER TABLE venda MODIFY (observacao DATE);
  ALTER TABLE venda DROP COLUMN observacao;

- Excluir uma tabela

  DROP TABLE *R*

  Exclue a relação *R* da base de dados
  Exemplo:

  DROP TABLE VENDA;

Valores default em campos

• Existem situações em que podemos ter campos de uma linha (registro) sem valor (ex. uma inserção de um registro)

• Nestas situações o SQL coloca o valor NULL nos campos. É possível alterar este valor por outro (default) para os campos.

• Na declaração do tipo de uma campo podemos indicar o seu valor "default" usando a palavra DEFAULT.

```
CREATE TABLE VENDA
(CODVENDA          NUMBER(9),
 DATAVENDA         DATE DEFAULT '01-JAN-1901',
 DATAENTREGA       DATE,
 VALORTOTAL        NUMBER(10,2),
 CODCLIENTE        NUMBER(6) DEFAULT 999,
 CODVENDEDOR       NUMBER(6),
 OBSERVACAO        VARCHAR2(255) DEFAULT 'NENHUMA');
```

Criação de índices

- Um índice é uma estrutura de dados que permite melhorar o tempo de acesso a registros através de condições que envolvem valores do índice.

- Exemplo: Se é bastante comum fazer consultas que envolvem algo do tipo “procurar as vendas do cliente com codcliente= 345663” então poderá fazer sentido criar um índice no atributo *codcliente* para melhorar a eficiência computacional destas consultas.

CREATE [UNIQUE] INDEX *nome do índice* ON R(*campos*)

CREATE INDEX IX_VENDA_CLIENTE ON VENDA
(CODCLIENTE);

CREATE INDEX IX_VENDA_VENDEDOR ON VENDA
(CODVENDEDOR);

DROP INDEX IX_VENDA_VENDEDOR;

Restrições de integridade das Relações
Restrições de chave

- As restrições de chave são declaradas na criação de tabelas.

- Duas formas alternativas (mas não equivalentes) :
  - PRIMARY KEY e UNIQUE

```
ALTER TABLE VENDA ADD
(CONSTRAINT PK_VENDA PRIMARY KEY
(CODVENDA));
```

A construção UNIQUE usa a mesma sintaxe que a PRIMARY KEY.

Chaves externas
Restrições de integridade de referência

- Um atributo de uma relação pode ser declarado como uma chave externa no caso de se “referir” a uma entidade de uma outra relação.

  Implicações:

  – O atributo referido tem que ser chave primária na outra tabela.

  – Um valor que apareça na chave externa da tabela inicial tem que existir na outra tabela na respectiva chave primária.

```
ALTER TABLE VENDA ADD
(CONSTRAINT FK_VENDA_CLIENTE FOREIGN KEY
(CODCLIENTE)
REFERENCES CLIENTE
(CODCLIENTE));
```

Políticas de manutenção da integridade de referência

```
CREATE TABLE VENDA
(CODVENDA          NUMBER(9),
 DATAVENDA         DATE,
 DATAENTREGA       DATE,
 VALORTOTAL        NUMBER(10,2),
 CODCLI            NUMBER(6) REFERENCES CLIENTE (CODCLIENTE),
 CODVENDEDOR       NUMBER(6));
```

- Ações rejeitadas pela política de integridade:
  - Inserção de uma nova venda cujo valor de *codcli* não é NULL e não existe na tabela *cliente*.
  - Alteração do valor do campo *codcli* de uma venda para um novo valor que não se encontra na tabela *cliente*.
  - Remoção de um registro da tabela *cliente* cujo *codcliente* é referido num registro da tabela *venda* através do campo *codcli*.
  - Alteração do valor do campo *codcliente* de um registro de *clientes*, sendo que o valor anterior era referido por algum registro de *vendas* através do campo *codcli*.

As políticas *CASCADE* e *SET NULL*

• Estas políticas são iguais à política de integridade referencial, exceto na forma como lidam com ações sobre a tabela que é referenciada.

CASCADE

– A ação de apagar um *cliente* é permitida.
  • Todas as vendas que referenciem o cliente (como seu comprador), são também apagados.

– Alterações no *codcliente* de um cliente são permitidas.
  • Todas as vendas que referenciem o valor anterior de *codcliente* (através do campo *codcli*), serão também atualizados com a alteração feita no campo *codcliente* da tabela *cliente* de modo a manter a integridade das referências.

Restrições nos valores dos campos

- Restrições *NOT NULL*
  - Indicam que o valor de um campo numa relação não pode ser NULL.

```
CREATE TABLE VENDA
(CODVENDA          NUMBER(9) NOT NULL PRIMARY KEY,
 DATAVENDA         DATE,
 DATAENTREGA       DATE,
 VALORTOTAL        NUMBER(10,2),
 CODCLIENTE        NUMBER(6) NOT NULL,
 CODVENDEDOR       NUMBER(6) NOT NULL);

CREATE TABLE ITEMVENDA
(CODVENDA          NUMBER(9) NOT NULL REFERENCES VENDA (CODVENDA),
 CODITEM           NUMBER(3) NOT NULL,
 VALOR             NUMBER(9,2),
 QUANTIDADE        NUMBER(5),
 CODPRODUTO        NUMBER(6) NOT NULL,
 PRIMARY KEY (CODVENDA,CODITEM));
```

Restrições nos atributos usando *CHECK*

- Expressões que podem ser associadas a campos e que têm que ser verificadas para atender a restrição.

- Implica numa verificação sempre que existe algum novo valor nesse campo (inserção, alteração).

  ...

  sexo VARCHAR2(1) CHECK (sexo IN (‘ F’, ‘M’)),

- A condição a colocar a frente do CHECK pode incluir referências a outros campos da tabela.

Dar nomes às restrições

- Para serem alteradas e também para melhor entendimento de eventuais mensagens de violação de restrições.
- O nome atribui-se precedendo a restrição pela palavra CONSTRAINT seguida do nome.

Exemplos :

PRIMARY KEY

```
CREATE TABLE CLIENTE
(CODCLIENTE NUMBER(6) NOT NULL CONSTRAINT PK_CLIENTE PRIMARY KEY,...

ALTER TABLE CLIENTE ADD
(CONSTRAINT PK_CLIENTE PRIMARY KEY
(CODCLIENTE)
USING INDEX TABLESPACE …
```

Dar nomes às restrições (continuação)

FOREIGN KEY

```
ALTER TABLE FISICO ADD
(CONSTRAINT FK_FISICO_CLIENTE FOREIGN KEY
(CODCLIENTE)
REFERENCES CLIENTE
(CODCLIENTE)
```

CHECK

```
CREATE TABLE CLIENTE
(CODCLIENTE NUMBER(6) NOT NULL CONSTRAINT PK_CLIENTE PRIMARY KEY,
 TIPOCLIENTE NUMBER(1) CONSTRAINT CK_TIPOCLIENTE
                    CHECK (TIPOCLIENTE IN (1,2)),...

ALTER TABLE CLIENTE
ADD
(CONSTRAINT CK_TIPOCLIENTE CHECK (TIPOCLIENTE IN (1,2)))
```

Alterar restrições

- Eliminar restrições é a única opção para mudá-la.

```
ALTER TABLE CLIENTE DROP PRIMARY KEY;

ALTER TABLE FISICO DROP CONSTRAINT FK_FISICO_CLIENTE;

ALTER TABLE FISICO ADD
(CONSTRAINT PK_FISICO PRIMARY KEY
(CODCLIENTE);
```

Consultas

Forma básica de uma consulta em SQL

SELECT <lista de campos>
FROM <lista de tabelas>
WHERE <condição>

Ex: ***Cliente***=*{CodCliente, Nome, Endereco, TipoCliente}*

```
SELECT nome, endereco
FROM cliente
WHERE tipocliente = 1

SELECT *
FROM cliente
WHERE nome = 'Charles Carvalho Santos'
```

Projeção em SQL

- A cláusula SELECT é usada para selecionar os campos pretendidos.

```
SELECT codcliente, nome
FROM cliente
```

```
SELECT cpf
FROM físico
```

```
SELECT DISTINCT codcliente
FROM venda
```

- Pode-se mudar o nome das colunas da tabela resultante da execução da consulta (operação de mudança de nome).

```
SELECT codcliente Cliente, tipocliente Tipo
FROM cliente
```

- Podem-se usar expressões em vez de campos.

```
SELECT codproduto Item, valor*quantidade Total
FROM itemvenda
```

Seleção em SQL

- A cláusula WHERE estabelece uma condição que determina os registros que irão aparecer no resultado da consulta.

```
SELECT *
FROM produto
WHERE estoquemin > 4000 AND estoquemax < 8000
```

– Operadores Lógicos AND, OR, NOT.
– Operadores Relacionais >, <, >=, <=, <>, =.

```
SELECT codvenda
FROM venda
WHERE dataentrega > ’12-Ago-2004’
```

Ordenação dos resultados

• A cláusula ORDER BY estipula um critério de ordenação dos resultados da consulta.

```
SELECT *
FROM cliente
ORDER BY nome
```

```
SELECT codvenda
FROM itemvenda
WHERE codproduto = 1
ORDER BY quantidade DESC
```

```
SELECT codvendedor, nome
FROM vendedor
ORDER BY 2, 1
```

Consultas envolvendo mais do que uma relação

- Todas operações envolvendo várias relações da álgebra relacional estão implementadas no SQL.
  - Junções, produtos, interseções, uniões e diferenças.

- operações sobre conjuntos (interseção, união e diferença) estão implementadas através de operadores específicos.
  - UNION, INTERSECT, MINUS.

- Para os produtos e junções existem duas alternativas:
  - O uso da instrução SELECT- FROM- WHERE.
  - O uso das instruções JOIN
    - A partir do standard SQL2 (92).

União, interseção e diferença

- A união de relações

  Q1: Todos os clientes físicos e jurídicos.

  (SELECT codcliente FROM fisico)
  UNION
  (SELECT codcliente FROM juridico)

- A interseção de relações

  Q2: Clientes Cadastrados e que são pessoa jurídica.

  (SELECT codcliente FROM cliente)
  INTERSECT
  (SELECT codcliente FROM juridico)

- A diferença de relações

  Q3 : Todos os clientes que são pessoa física.

  (SELECT codcliente FROM cliente)
  MINUS
  (SELECT codcliente FROM juridico)

Produtos e Junções com a instrução SELECT...

- Incluindo várias relações na cláusula FROM pode-se dizer ao SQL para juntar a informação das respectivas tabelas.
- Produto Cartesiano

  Álgebra relacional : *cliente* $\times$ *fisico*

  SQL:
  ```
  SELECT *
  FROM cliente, fisico
  ```

- Junção Natural

  Álgebra relacional : *cliente x físico*

  SQL:
  ```
  SELECT *
  FROM cliente, fisico
  WHERE cliente.codcliente = fisico.codcliente
  ```

Produtos e Junções com a instrução SELECT... - II

- Junção Condicional

  Álgebra relacional: *cliente* >|<$_{dataentrega>'12\text{-}ago\text{-}2004'}$ *venda*

  SQL: SELECT *

  FROM cliente,venda

  WHERE dataentrega > ‘12-Ago-2004’

- *EquiJoin*

  Álgebra relacional : *cliente* >|<$_{cliente.codcliente = venda.codcliente}$ *venda*

  SQL: SELECT *

  FROM cliente,venda

  WHERE cliente.codcliente = venda.codcliente

Produtos e Junções com a instrução SELECT...

Exemplos

```
SELECT cliente.codcliente, nome, cpf
FROM cliente,fisico
WHERE cliente.codcliente = fisico.codcliente

SELECT a.codfornecedor, a.nome, b.codproduto, b.estoque,
            c.estoquemin, c.estoquemax
FROM fornecedor a, produtoxfornecedor b, produto c
WHERE a.codfornecedor = b.codfornecedor AND
          b.codproduto = c.codproduto
```

OUTER JOINs

• Tipos especiais de junções que também integram no resultado da junção *R* >< *S* os registros de uma das relações que não possuem equivalência de linhas (registros) na outra.

• Várias variantes de OUTERJOINs

– *R.COD = S.COD (+)*

• Junção natural que inclui todos os registros de R mesmo os que não possuem equivalência nos campos comuns de *R* e S.

– *R.COD (+) = S.COD*

• Junção natural que inclui os registro de *S* que não possuem equivalência com registros em *R*.

Valores NULL

- Como resultado de um OUTER JOIN entre duas relações vamos ter registros em que o valor de alguns campos não é conhecido (por não terem equivalência).
- O SQL usa o valor NULL para estas situações.
- O valor NULL também é usado em outras ocasiões, como em inserções incompletas de registros.
- Regras com valores NULL :
  - Quando um dos operandos de uma expressão aritmética é NULL o resultado será sempre NULL.
  - Quando um dos operandos de uma expressão relacional é NULL o resultado é indeterminado
- O valor NULL não é uma constante, portanto não pode ser usado de maneira explícita.
  - Pode-se testar se um valor é NULL com a expressão “IS NULL”.

Consultas aninhadas

- As cláusulas WHERE até agora analisadas comparam valores escalares.
- No SQL é possível ter cláusulas WHERE que comparem registros completos de tabelas ou mesmo tabelas inteiras.
- Uma consulta aninhada *(subquery*) é uma expressão que fornece como resultado uma relação.
- Assim uma instrução SELECT- FROM- WHERE pode ser vista como uma consulta aninhada.
- O SQL possui uma série de operadores para comparar registros e tabelas no contexto das cláusulas WHERE.

Consultas aninhadas que produzem valores escalares

- Uma expressão SELECT- FROM- WHERE pode produzir uma relação com qualquer número de atributos e de registros.
- No entanto, por vezes podemos estar interessados num único atributo como resultado da consulta.
- Além disso, poderemos saber pelas restrições de chave que a consulta irá resultar num único valor para esse atributo.
- Nesses casos a expressão SELECT pode ser rodeada de parênteses e tratada como um número numa comparação.

```
SELECT cliente.nome, cgc, inscestadual
FROM cliente, juridico
WHERE cliente.codcliente=juridico.codcliente
    AND tipocliente=2
```

```
SELECT nome
FROM cliente
WHERE codcliente IN
      (SELECT codcliente
       FROM juridico)
```

Condições envolvendo relações

- Existem uma série de operadores do SQL que podem ser aplicados a relações produzindo um valor booleano (true/false).
- Alguns destes operadores também envolvem valores escalares.
  - Nestes casos a relação envolvida tem que ter um único campo (coluna).
- EXISTS *R* : verdadeiro se a relação *R* não for vazia.
- *s* IN *R* : verdadeiro se *s* é igual a um dos valores em *R.* É assumido que *R* é uma relação unária.
- Todos os operadores (EXISTS, IN) podem ser negados colocando-se “NOT” antes.

Condições envolvendo registros

• Se um registro tem os mesmo campos que uma relação, pode-se usar o operador IN para comparar o registro com a relação.

Quais os vendedores que realizaram vendas de produtos acima de R$ 5.000,00?

Vendedor=(codvendedor,nome,endereço,telefone,comissão)

Venda=(codvenda,datavenda,dataentrega,valortotal,codcliente,codvendedor)

ItemVenda=(codvenda,coditem,valor,quantidade,codproduto)

```
SELECT nome
FROM vendedor
WHERE codvendedor IN
  (SELECT codvendedor
   FROM venda
   WHERE codvenda IN
      (SELECT codvenda
       FROM itemvenda
       WHERE valor*quantidade > 5000))
```

```
SELECT a.nome
FROM vendedor a,venda b,itemvenda c
WHERE a.codvendedor=b.codvendedor
      AND b.codvenda=c.codvenda
      AND c.valor * c.quantidade > 5000
```

Consultas aninhadas correlacionadas

- Consultas aninhadas que são avaliadas para cada registro da consulta “superior” por dependerem dela.

Quais itens foram vendidos que ultrapassam o estoque máximo?

```
SELECT codvenda, coditem, quantidade
FROM itemvenda iv
WHERE quantidade >
    (SELECT estoquemax
    FROM produto
    WHERE codproduto = iv.codproduto)
```

Agregação - operadores

- O SQL tem 5 operadores que são aplicáveis a um campo de uma tabela produzindo uma espécie de agregação dos valores desse campo.
- SUM produz a soma dos valores do campo nos diferentes registros.
- AVG produz a média dos valores do campo.
- MAX dá como resultado o valor máximo no campo.
- MIN dá como resultado o valor mínimo no campo.
- COUNT indica o número de valores do campo incluindo duplicados, a não ser que estes sejam eliminados com DISTINCT.

Agregação – alguns exemplos

```
SELECT SUM(quantidade)
    FROM itemvenda
    WHERE codproduto = 1
```

```
SELECT AVG(valor)
    FROM itemvenda
    WHERE codproduto IN (
                    SELECT codproduto
                    FROM produto
                    WHERE nome = 'ARRUELA METALICA 10')
```

```
SELECT codvenda, coditem, valor, quantidade
    FROM itemvenda
    WHERE quantidade= (
        SELECT MAX(quantidade)
        FROM itemvenda)
```

~~SELECT codproduto,~~
~~Max(estoquemax)~~
~~FROM produto~~

Agregação – alguns exemplos

Quantos fornecedores existem?

```
SELECT COUNT(*)
    FROM fornecedor
```

Quantos produtos são comercializados do fornecedor MOTOROLA ?

```
SELECT COUNT(DISTINCT codproduto)
    FROM produtoxfornecedor
    WHERE codfornecedor IN (
        SELECT codfornecedor
        FROM fornecedor
        WHERE nome = 'MOTOROLA')
```

Agrupamento – a cláusula GROUP BY

- Por vezes não estamos somente interessados na agregação de uma coluna, mas sim em agrupar registros da tabela e agregar colunas para cada grupo encontrado.

  – *Qual a quantidade média em estoque para cada produto?*

```
SELECT codproduto, AVG(estoque)
  FROM produtoxfornecedor
  GROUP BY codproduto
```

  – *Quantas vendas cada vendedor realizou?*

```
SELECT codvendedor, COUNT(*) QtdVendas
  FROM venda
  GROUP BY codvendedor
```

Especificação dentro dos grupos
-a cláusula HAVING

• Esta cláusula permite escolher os grupos usando uma condição qualquer para filtrá-los.

– Qual a quantidade média de itens vendidos para os produtos que foram vendidos mais de 3 vezes?

```
SELECT codproduto, AVG(quantidade)
    FROM itemvenda
    GROUP BY codproduto
    HAVING COUNT(*) > 3
```

Modificações no Banco de Dados (*DML*)
Inserção

• A forma básica da instrução de inserção de registros em SQL é:
INSERT INTO R(A_1 , A_2 , ..., A_n) VALUES (v_1 , v_2 , ..., v_n)

– É acrescentado um registro à relação *R* com o valor v_1 no campo A_1 , e assim sucessivamente.

– Não é necessário incluir todos os campos do esquema da relação R.
  • Os campos que não forem incluídos na inserção ficarão com o valor por default do campo no registro que é introduzido. O valor por default mais comum é o valor NULL.

– Ao introduzir um registro com valores para todos os campos da relação pode-se omitir os nomes dos mesmos.
INSERT INTO *R* VALUES (v_1 , v_2 , ..., v_n)
  • Nestas situações deve-se ter cuidado para a ordem dos valores ser exatamente a ordem dos campos na relação.

Inserção – alguns exemplos

- *Inserir um novo fornecedor na relação de fornecedor*

```
INSERT INTO FORNECEDOR (codfornecedor, nome, cgc, contato, telefone)
    VALUES (2, 'MOTOROLA', 8430375732419, 'MARCOS LIMA', '15-34114467')
```

Ou

```
INSERT INTO FORNECEDOR
    VALUES (2, 'MOTOROLA', 8430375732419, 'MARCOS LIMA', '15-34114467')
```

- *Inserir um vendedor contratado pela empresa*

```
INSERT INTO VENDEDOR (CODVENDEDOR, NOME)
    VALUES (4, 'Lucas Mello Braga')
```

- *Incluir à relação físico todos os clientes que são pessoas físicas.*

```
INSERT INTO fisico (codcliente)
    (SELECT codcliente
    FROM cliente
    WHERE codcliente=1)
```

Exclusão

- A forma básica da instrução de exclusão de registros em SQL é:

  DELETE FROM *R* WHERE *condição*

  – Todos os registro da relação *R* que verifiquem a condição à frente do WHERE serão eliminados.

  – Como no SQL podem existir registros “repetidos” não existe maneira de apagar um só dos repetidos.

- Exemplos:

```
DELETE FROM juridico
WHERE codcliente = 6

DELETE FROM juridico
```

Alterações

- A forma básica da instrução de alteração de registros em SQL é:

  UPDATE *R* SET *novos valores* WHERE *condição*

  – Todos os registros da relação *R* que satisfazem a condição vêm os campos incluídos nas expressões à frente da palavra SET alterados.

- Exemplos:

```
UPDATE fornecedor
    SET telefone = '2432-1501',
        contato = 'Ubirajara'
    WHERE codfornecedor = 1

UPDATE vendedor
    SET comissão = comissao * 1.1
```

Views

• Uma view (visão) define uma tabela virtual obtida por uma expressão aplicada a tabelas reais.

• A declaração de uma view é feita com a instrução...

CREATE VIEW *nome da view* AS *definição da view*

```
CREATE VIEW clientefisico AS
    SELECT cliente.codcliente, nome, cpf
    FROM cliente,fisico
    WHERE cliente.codcliente = fisico.codcliente

CREATE VIEW portfoliofornecedor AS
    SELECT fornecedor.codfornecedor, fornecedor.nome fornecedor,
                produto.codproduto, produto.nome produto, produto.descricao
    FROM produto, fornecedor, produtoxfornecedor
    WHERE produto.codproduto=produtoxfornecedor.codproduto
        AND fornecedor.codfornecedor=produtoxfornecedor.codfornecedor
```

Consultas a views

- As views podem ser consultadas como se fossem uma tabela. Antes de responder a consulta a view é “calculada” usando a consulta incluída na sua definição.

```
SELECT nome
    FROM clientefisico
    WHERE codcliente IN (select codcliente from venda)
```

```
SELECT *
    FROM portfoliofornecedor a, produtoxfornecedor b
    WHERE a.codfornecedor = b.codfornecedor
    AND a.codproduto=b.codproduto
```