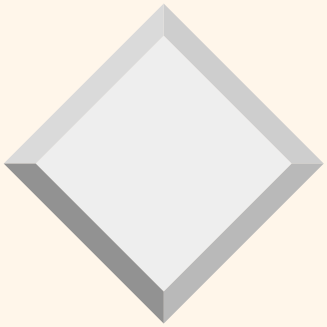


Capítulo 4
A Linguagem SQL

Murilo Silva de Camargo

SQL - Structured Query Language

- **Estrutura básica**
- **Operações de conjunto**
- **Funções agregadas**
- **Valores nulos**
- **Sob consultas aninhadas**
- **Relações derivadas**
- **Visões**
- **Modificação do banco de dados**
- **Junção de relações**
- **Linguagem de definição de dados**
- **SQL embutida**

O Exemplo da Empresa Bancária

agencia (nome_agencia, cidade_agencia, fundos)

cliente (nome_cliente, rua_cliente, cidade_cliente)

conta (nome_agencia, numero_conta, saldo)

emprestimo (nome_agencia, numero_emprestimo, total)

depositante (nome_cliente, numero_conta)

devedor (nome_cliente, numero_emprestimo)

Estrutura Básica

- SQL é baseada em operações em conjuntos e relações com algumas modificações e melhorias. Uma consulta típica da SQL tem a forma:

  select $A_1, A_2, ..., A_n$
  from $r_1, r_2, ..., r_m$
  where P

  - A_is representam atributos
  - r_is representam relações
  - P é um predicado.

- Esta consulta é equivalente à expressão da álgebra relacional:

$$\Pi_{A_1, A_2, ..., A_n}(\sigma_P (r_1 \times r_2 \times ... \times r_m))$$

- O resultado de uma consulta SQL é uma relação.

A Cláusula select

- A cláusula **select** corresponde à operação de projeção da álgebra relacional. Ela é usada para listar os atributos desejados no resultado de uma consulta.
- Achar os nomes de todas as agências na relação **empréstimo**

  select nome_agencia

  from emprestimo
- Na sintaxe da álgebra relacional "pura", esta consulta poderia ser:

  $\Pi_{nome\_agencia}$ (emprestimo)
- Um asterisco na cláusula select denota "todos atributos"

  select *

  from emprestimo

A Cláusula select (Cont.)

- SQL permite duplicatas nas relações, bem como nos resultados das consultas.
- Para forçar a eliminação de duplicatas, deve-se inserir declaração **distinct** depois do **selec**t.

  Achar os nomes de todas as agências na relação empréstimo e remover as duplicatas

  select distinct nome_agencia

  from emprestimo
- A decvlaração **all** especifica que duplicatas **não** devem ser removidas.

  select all nome_agencia

  from emprestimo

A Cláusula select (Cont.)

- A cláusula **select** pode conter expressões aritméticas envolvendo os operadores +, - , * , e / , e operações em constantes ou atributos de tuplas.
- A consulta:

  select nome_agencia, numero_emprestimo, total * 100

  from emprestimo

  retornaria uma relação a qual é a igual a relação empréstimo, exceto que o atributo total é multiplicado por 100.

A Cláusula where

- A cláusula **where** corresponde ao predicado de seleção da álgebra relacional. Ele consiste de um predicado envolvendo atributos das relações que aparecem na cláusula **from**.
- Achar todos os numeros de empréstimos de empréstimos feitos na agência Perryridge com totais maiores que $1200.

  select numero_empustimo

  from emprestimo

  where nome_agencia = 'Perryridge' **and** total > 1200
- SQL usa os conectivos lógicos **an**d, **o**r, e **not**. Ele permite o uso de expressões aritméticas como operandos para os operadores de comparação.

A Cláusula where (Cont.)

- SQL inclui um operador de comparação **between** para simplificar cláusulas **where** que especificam que um valor deva ser menos ou igual a algum valor e maior ou igual que algum outro valor.
- Achar o número do empréstimo dos empréstimos com total entre \$90,000 e \$100,000 (isto é, ≥ \$90,000 and ≤ \$100,000)

```
select numero_emprestimo
from   emprestimo
where  total between 90000 and 100000
```

A Cláusula from

- A cláusula **from** corresponde ao produto cartesiano da álgebra relacional . Lista as relações que serão mapeadas na avaliação da expressão.
- Encontre o produto cartesiano devedor × emprestimo

```
select *
from devedor,emprestimo
```

- Encontre o nome e o número de empréstimo de todos os clientes que possuem um empréstimo na agência Perryridge.

```
select distinct nome_cliente,devedor.numero_emprestimo
from devedor,emprestimo
where devedor.numero_emprestimo =
                        emprestimo.numero_emprestimo
and nome_agencia = 'Perryridge'
```

A Operação Rename

- A SQL possui um mecanismo para renomear tanto relações quanto atributos através da cláusula **as,** da seguinte forma

  nome_antigo **as** novo_nome

- Encontre o nome e o número do empréstimo dos clientes que possuem um empréstimo na agência Perryridge; substitua o nome da coluna numero_empréstimo por "numero_do_emprestimo_do_devedor".

```
select   distinct nome_cliente, devedor.numero_emprestimo as
                                numero_do_emprestimo_do_devedor
from     devedor, emprestimo
where    devedor.numero_emprestimo =
   emprestimo.numero_emprestimo
and      nome_agencia = 'Perryridge'
```

Variáveis Tuplas

- Variáveis tuplas são definidas na cláusula **from** através do uso da cláusula **as** .
- Encontre o nome dos clientes e seus números de empréstimo para todos os clientes que possuem um empréstimo em alguma agência.

```
select distinct  nome_cliente, T.numero_emprestimo
from     devedor as T, emprestimo as S
where    T.numero_emprestimo =S.numero_emprestimo
```

- Encontre o nome de todas as agências que possuam fundos maiores que ao menos uma agência daquelas localizadas no Brooklyn.

```
select distinct T.nome_agencia
from     agencia as T, agencia as S
where    T.fundos > S.fundos and S.cidade_agencia  = 'Brooklyn'
```

Operações com Strings

- SQL inclui operadores de comparação de strings. Padrões são descritos usando dois caracteres especiais:
  - % combina com qualquer substring, independente do tamanho.
  - _ combina caractere a caractere .
- Encontre o nome de todos os clientes cuja rua contenha o substring ‘Main’.

```
select nome_cliente
from cliente
where rua_cliente like '%Main%'
```

- Combine com o substring 'Main%'

```
like 'Main \ %' escape ' \ '
```

Operações em Strings (cont.)

Características:

- É sensível ao tamanho das letras (case sensitive)
- Permite encontrar diferenças, e não coincidências, através do uso de **not like**.
- Possui operações de concatenação, extração de substrings, indicação de tamanhos, conversão de maiúsculas para minúsculas e vice-versa, etc.

Ordenando a Apresentação de Tuplas

- Listar em ordem alfabética os nomes de todos os clientes que têm um empréstimo na agência Perryridge:

```
select distinct nome_cliente
from devedor, emprestimo
where devedor.numero_emprestimo = emprestimo.numero_emprestimo
and nome_agencia = 'Perryridge'
order by nome_cliente
```

- A palavra **desc** indica ordem descendente, e **asc** a ordem ascendente (default) de apresentação dos dados.
- SQL deve executar uma classificação para realizar uma solicitação **order by**. Como classificar uma grande quantidade de dados pode ser demorada, é aconselhável usar o **order by** apenas quando necessário.

Ordenação e apresentação de tuplas (continuação)

- SQL pode executar uma ordenação por diversos atributos.

```
select *
from emprestimo
order by total desc, numero_emprestimo asc
```

Duplicatas

- Em relações com duplicidades, SQL pode definir quantas cópias da tupla aparecem no resultado.
- Em versões multiconjuntos dos operadores da álgebra relacional - dadas as relações r_1 e r_2:
  1. Se existe c_1 cópias da tupla t_1 em r_1, e t_1 satisfaz a seleção σ_θ, então existem c_1 cópias de t_1 em $\sigma_\theta(r1)$.
  2. Para cada cópia da tupla t_1 em r_1, em que existe uma cópia da tupla $\Pi_A(t_1)$ em $\Pi_A(r_1)$, onde $\Pi_A(t_1)$ denota a projeção de uma tupla única t_1.
  3. Se existe c_1 cópias da tupla t_1 em r_1 e c_2 cópias da tupla t_2 em r_2, existe $c_1 \times c_2$ cópias da tupla $t_1.t_2$ em $r_1 \times r_2$.

Duplicatas (Cont.)

- Suponhas que as relações r_1 com o esquema (A,B) e r_2 com esquema (C) sejam os seguintes multiconjuntos:

  $r_1 = \{(1,a),(2,a)\}$ $\quad$ $r_2 = \{(2),(3),(3)\}$

- Então $\Pi_B(r_1)$ poderia ser {(a),(a)}, enquanto $\Pi_B(r_1) \times r_2$ poderia ser

  {(a,2),(a,2),(a,3),(a,3),(a,3),(a,3)}

- A cláusula SQL

  select $A_1, A_2, ..., A_n$
  from $r_1, r_2, ..., r_m$
  where P

  é equivalente à expressão em álgebra relacional:

  $$\Pi_{A_1, A_2, ..., A_n} (\sigma_P (r_1 \times r_2 \times ... \times r_m))$$

Operações de Conjuntos

- As operações de conjuntos **union**, **intersect** e **except** operam em relações e correspondem às operações $\cup$, $\cap$ e - (diferença) da álgebra relacional.
- Estas operações eliminam as duplicatas; se desejarmos obter as repetições, devemos explicitar através da forma **union all**, **intersect all** e **except all**.
- Suponha uma tupla que ocorra "m" vezes em *r* e "n" vezes em *s*, então temos :
  - m + n vezes em *r* **union all** *s*
  - min (m, n) vezes em *r* **intersect all** *s*
  - max (0, m-n) vezes em *r* **except all** *s*

Operações de Conjuntos (cont.)

- Encontre todos os clientes que possuam um empréstimo, uma conta ou ambos:

  (select nome_cliente **from** depositante)

  union (select nome_cliente **from** devedor)

- Encontre todos os clientes que possuem ambos uma conta e um empréstimo:

  (select nome_cliente **from** depositante)

  intersect (select nome_cliente **from** devedor)

- Encontre todos os clientes que possuem uma conta mas não possuem empréstimo;

  (select nome_cliente **from** depositante)

  except (select nome_cliente **from** devedor)

Funções Agregadas

- Essas funções operam nos multi-conjuntos de valores de uma coluna de uma relação e retornam um valor

  avg: média dos valores

  min: valor mínimo

  max: valor máximo

  sum: soma dos valores

  count: número de valores

Funções Agregadas (Cont.)

- Encontre a média dos saldos em contas na agência Perryridge.

  select avg (saldo)

  from contas

  where nome_agencia = 'Perryridge'

- Encontre o número de tuplas na relação clientes:

  select count (*)

  from cliente

- Encontre o número de depositantes no banco:

  select count (distinct nome_cliente)

  from depositante

Funções Agregadas – Group By

- Encontre o número de depositantes em cada agência.

```
select nome_agencia, count (distinct nome_cliente )
from depositante,conta
where depositante.numero_conta =conta.numero_conta
group by nome_agencia
```

Nota: Atributos na cláusula **select** fora das funções agregadas devem aparecer na lista **group by**.

Funções Agregadas – Having

- Encontre o nome de todas as agências onde a média do saldo das contas seja maior que $1,200

```
select nome_agencia, avg (saldo)
from conta
group by nome_agencia
having avg (saldo) > 1200
```

Nota: predicados na cláusula **having** são aplicados após a formação dos grupos.

Valores Null

- É possível para as tuplas ter valor nulo, denotados por null, para alguns de seus atributos. Significa valor desconhecido ou inexistente. O resultado de uma expressão aritmética envolvendo *null* é *null.* As operações envolvendo *null* retornam *false*
  - (*true* **or** *unknown*)= *true*, (false **or** *unknown*)= *unknown* (unknown **or** unknown)= unknown
  - (*true* **and** *unknown*)= *unknown*, (*false* **and** *unknown*)= *false*, (unknown **and** unknown)= unknown
  - Resultado da cláusula **where** é tratado como *false* se a sua avaliação é desconhecida
  - "P **is *unknown***" é *true* se a avaliação do predicado P é = *unknown*.

Valores Null (continuação)

- Encontre todos os números de empréstimo que aparecem na relação empréstimo com valores *null* para o total.

  select numero_emprestimo
  from emprestimo
  where total **is null**

- Total de todos os empréstimos

  select sum (total)
  from emprestimo

  A declaração acima ignora valores *null* no total ; o resultado será *null* somente se não existir nenhuma tupla com o atributo total diferente de *null*

- Todas as operações agregadas, exceto **count(*)** ignoram tuplas com valores *null* nos atributos agregados .

Subconsultas Aninhadas

- SQL provê um mecanismo para aninhamento de subconsultas.
- Uma subconsulta é uma expressão **select-from-where** que é aninhada dentro de uma outra consulta.
- As aplicações mais comuns para as subconsultas são testes para membros de conjuntos, comparação de conjuntos e cardinalidade de conjuntos.

Membro de Conjunto

- F **in** r $\Leftrightarrow \exists t \in r\ (t = F)$

(5 **in** | **0** / **4** / **5**) = true

(5 **in** | **0** / **4** / **6**) = false

(5 **not in** | **0** / **4** / **6**) = true

Exemplo (Consulta Aninhada)

- Encontrar todos os clientes que possuem uma conta e um empréstimo no banco.

```
select distinct nome_cliente
from devedor
where nome_cliente in (select nome_cliente
                       from depositante)
```

- Encontrar todos os clientes que tenham um empréstimo no banco mas não tenham uma conta neste banco.

```
select distinct nome_cliente
from devedor
where nome_cliente not in (select nome_cliente
                           from depositante)
```

Exemplo (Consulta Aninhada)

- Encontrar todos os clientes que tenham uma conta e um empréstimo na agência Perryridge.

```
select distinct nome_cliente
from devedor, emprestimo
where
    devedor.numero_emprestimo=emprestimo.numero_em
    prestimo and nome_agencia = 'Perryridge' and
        (nome_agencia, nome_cliente) in
            (select nome_agencia, nome_cliente
            from depositante, conta
            where depositante.numero_conta =
                conta.numero_conta)
```

Exemplo (Operador **not in***)*

- Conjuntos enumerados:

```
select distinct  nome_cliente
from  devedor
where  nome_cliente  not in  ('Smith', 'Jones')
```

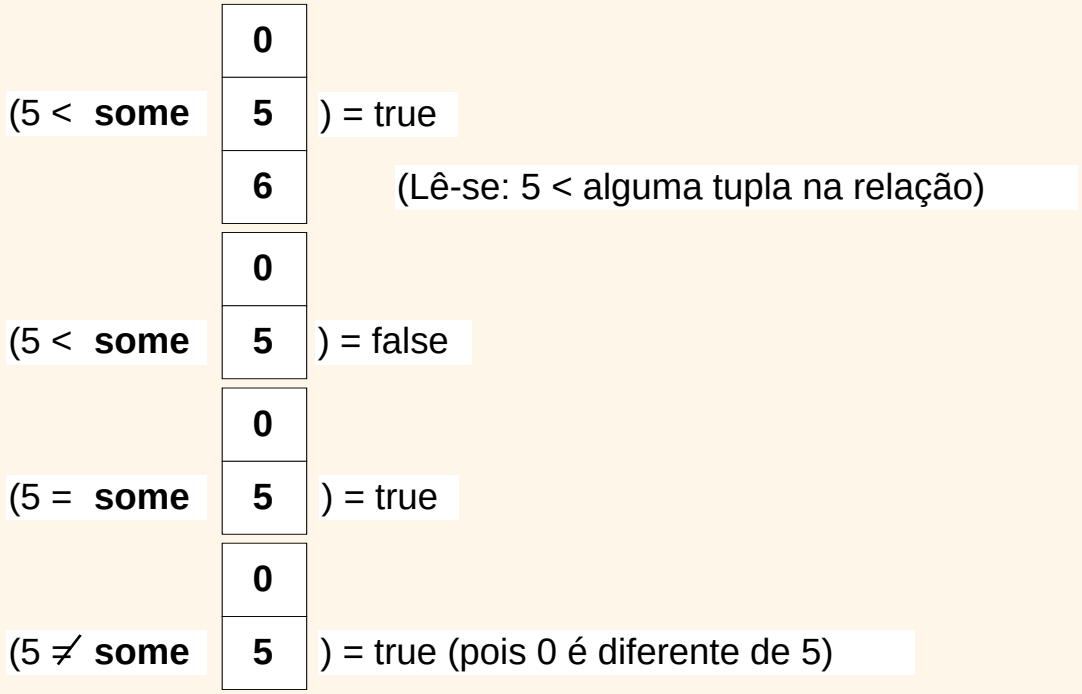
Comparação de Conjuntos

- Encontrar os nomes de todas as agências que tenham fundos maiores que **ao menos uma** agência localizada em Brooklyn

```
select distinct T.nome_agencia
from agencia as T, agencia as S
where T.fundos > S.fundos and
      S.cidade_agencia = 'Brooklyn'
```

A Cláusula **some**

- F <comp> **some** r ⇔∃t(t ∈ r ^ [F <comp> t])
  Onde <comp> pode ser: <, ≤, >, ≥, =, <>



- (= **some**) ≡ **in**
- No entanto, (≠ **some**) ≡ **not in**

Exemplo (Cláusula **some)**

- Encontre todos as agências que têm fundos maiores que **ao menos uma** agência localizada em Brooklyn.

```
select nome_agencia
from agencia
where fundos > some
        (select fundos
        from agencia
        where cidade_agencia = 'Brooklyn')
```

A Cláusula **all**

- F <comp> **all** r $\Leftrightarrow \forall$ t (t $\in$ r $\wedge$ [F <comp> t])

(5 < **all**	**0** **5** **6**	) = false
(5 < **all**	**6** **10**	) = true
(5 = **all**	**4** **5**	) = false
(5 $\neq$ **all**	**4** **6**	) = true (since 5 $\neq$ 4 and 5 $\neq$ 6)

- **($\neq$ all) $\equiv$ not in**
- No entanto, (= **all**) $\not\equiv$ **in**

Exemplo (Cláusula **all)**

- Encontrar os nomes de todas as agências que tenham fundos maiores que **cada uma** das agências localizadas em Brooklyn.

```
select nome_agencia
from agencia
where fundos > all
    (select fundos
     from agencia
     where cidade_agencia = 'Brooklyn')
```

Exemplo (Cláusula **all)**

- Encontrar a agência que tenha o maior saldo médio. Observe que funções agregadas não podem ser novamente agregadas

  OBS.:(max (avg(...)) não é permitido.

```
select nome_agencia
from conta
group by nome_agencia
having avg(saldo) >= all (select avg(saldo)
                          from conta
                          group by nome_agencia
```

Verficação de Relações Vazias

- O construtor **exists** retorna o valor **true** se a sub-consulta usada como argumento é “não-vazia”.

- **exists** r $\Leftrightarrow$ r $\neq \varnothing$
- **not exists** r $\Leftrightarrow$ r = $\varnothing$

Exemplo (Construtor **exists)**

- Encontre todos os clientes que tenham uma conta em todas as agências localizadas no Brooklin

```
select distinct S.nome_cliente
from depositante as S
where not exists (
        (select nome_agencia
         from agencia
         where cidade_agencia = 'Brooklyn')
        except
         (select R.nome_agencia
          from depositante as T, conta as R
          where T.numero_conta = R.numero_conta and
                S.numero_cliente =T.numero_cliente))
```

- Note que $X - Y = \emptyset \Leftrightarrow X \subseteq Y$

Teste para Ausência de Tuplas Repetidas

- O contrutor **unique** testa se a sub-consulta tem alguma tupla repetida no seu resultado.
- Encontre todos os clientes que tenham apenas uma conta na agência Perryridge.

```
select T.nome_cliente
from depositante as T
where unique (
        select R.nome_cliente
        from conta, depositante as R
        where T.nome_cliente = R.nome_cliente and
                R.numero_conta = conta.numero_conta and
                conta.nome_agencia = 'Perryridge')
```

Exemplo (Cláusula **unique)**

- Encontre todos os clientes que tenham pelo menos duas contas na agência Perryridge.

```
select distinct T.nome_cliente
from depositante T
where not unique (
        select R.nome_cliente
        from conta, depositante as R
        where T.nome_cliente = R.nome_cliente and
                R.numero_conta = conta.numero_conta and
                conta.nome_agencia = 'Perryridge')
```

Relações Derivadas

- Encontre a média do balanço de contas das agências onde a média do balanço de contas é maior que $1200.

```
select nome_agencia, saldo_médio
from (select nome_agencia, avg (saldo)
        from conta
        group by nome_agencia)
        as result (nome_agencia,saldo_médio)
where saldo_médio > 1200
```

Note que não é necessário usar a cláusula **having**, já que calculamos na cláusula **from** uma relação temporária e os atributos dessa relação podem ser usados diretamente no cláusula **where**.

Visões

- Fornecem um mecanismo para esconder certos dados do alcance de certos usuários. Para criar uma visão usa-se o comando:

  create view v **as** <expressão query>

  onde:

  - <expressão query> é qualquer expressão correta
  - o nome da visão é representada por “v”

Usando views

- Uma visão constituída de agências e seus clientes

```
create view todos_clientes as
    (select nome_agencia,nome_cliente
    from depositante,conta
    where depositante.numero_conta =
  conta.numero_conta)
  union
    (select nome_agencia,nome_cliente
    from devedor, emprestimo
    where devedor.numero_emprestimo =
          emprestimo.numero_emprestimo)
```

- Encontre todos os clientes da agência Perryridge

```
          select nome_cliente
    from todos_clientes
    where nome_agencia = 'Perryridge'
```

Modificações no Banco de Dados - Remoção (delete)

- Exclua todas os registros de contas da agência Perryridge

```
delete from conta
where nome_agencia = 'Perryridge'
```

Modificações no Banco de Dados - Remoção (delete)

- Exclua todas as contas de todas as agências localizadas em Needham.

```
delete from conta
where nome_agencia in (select nome_agencia
                       from agencia
                       where cidade_agencia = 'Needham')
delete from depositante
where numero_conta in (select numero_conta
        from agencia, conta
        where cidade_agencia = 'Needham'
        and agencia.nome_agencia = conta.nome_agencia)
```

Exemplo (Cláusula **delete)**

- Apague os registros de todas as contas com saldos abaixo da média no banco

delete from conta

where saldo < **(select avg** (saldo)

from conta)

  - Problema: ao apagar tuplas de conta, o saldo médio muda
  - Soluções usadas em SQL:

1. Primeiro, calcula o saldo médio - **avg** (saldo) - e encontre todas as tuplas para exclusão;
2. A seguir, apague todas tuplas encontradas acima (sem recalcular **avg** ou testar novamente as tuplas)

Modificações no Banco de Dados - Inserção (insert)

- Adicionar uma nova tupla em conta

  insert into conta

  values ('Perryridge', A-9732, 1200)

  ou de forma equivalente

  insert into conta (nome_agencia, saldo, numero_conta)

  values ('Perryridge', 1200, A-9732)

- Adicionar uma nova tupla à conta com saldo igual a nulo

  insert into account

  values ('Perryridge', A-777, null)

Modificações no Banco de Dados - Inserção (insert)

- Forneça aos clientes da agência Perryridge uma caderneta de poupança de $200 como brinde para cada emprestimo que eles tenham. O número do emprestimo será usado como número da caderneta de poupança

```
insert into conta
    select nome_agencia, numero_emprestimo,200
    from emprestimo
    where nome_agencia = 'Perryridge'
insert into depositante
    select nome_cliente, numero_emprestimo
    from  emprestimo, devedor
    where nome_agencia = 'Perryridge'
    and emprestimo.numero_conta = devedor.numero_conta
```

Modificações no Banco de Dados – Atualização (update)

- Acrescentar 6% a todas as contas com saldos acima de $10.000. Todas as outras contas recebem 5%.
  - Escreva dois comandos **update**:

update conta
set saldo = saldo * 1.06
where saldo > 10000

update conta
set saldo = saldo * 1.05
where saldo =< 10000

  - Aqui, a ordem dos comandos é importante!
  - SQL-92 fornece um construtor **case** que pode ser usado para realizar ambas as alterações anteriores em um único comando **update**

Atualizações de uma Visão

- Crie uma visão de todos os dados de empréstimos, omitindo o atributo “total”

```
create view agencia_emprestimo as
        select nome_agencia, numero_emprestimo
        from emprestimo
```

- Adicione uma nova tupla à visão agencia-emprestimo

```
insert into agencia_emprestimo
        values ('Perryridge', 'L-307')
```

  Esta inserção deve ser representada pela inserção da tupla

  ('Perryridge', 'L-307',null)

  na relação emprestimo.

- Atualizações em visões mais complexas são difíceis ou podem ser impossíveis de traduzir, e, assim, são proibitivas.

Composição de Relações

- Operações de **Junção** tomam duas relações e retornam como resultado uma outra relação.
- As operações de Junção são normalmente usadas na cláusula **from**.
- Condições de Junção – define quais tuplas das duas relações apresentam correspondência, e quais atributos serão apresentados no resultado de uma junção.
- Tipos de Junção – define como as tuplas em cada relação que não possuam nenhuma correspondência(baseado na condição de junção) com as tuplas da outra relação devem ser tratadas.

Tipos de junção
inner join
left outer join
rigth outer join
full outer join

Condições de junção
natural
on <predicate>
using (A_1, A_2,..., A_n)

Composição de Relações (exemplos)

- Relação emprestimo

nome_agencia	numero_emprestimo	total
Downtown	L-170	3000
Redwood	L-230	4000
Perryridge	L-260	1700

- Relação devedor

nome_cliente	numero_emprestimo
Jones	L-170
Smith	L-230
Hayes	L-155

Composição de Relações (exemplos)

- emprestimo **inner join** devedor **on** emprestimo.numero_emprestimo= devedor.numero_emprestimo

nome_agencia	numero_emprestimo	total	nome_cliente	numero_emprestimo
Downtown	L-170	3000	Jones	L-170
Redwood	L-230	4000	Smith	L-230

- emprestimo **left outer join** devedor **on** emprestimo.numero_emprestimo=devedor.numero_empre stimo

nome_agencia	numero_emprestimo	total	nome_cliente	numero_emprestimo
Downtown	L-170	3000	Jones	L-170
Redwood	L-230	4000	Smith	L-230
Perryridge	L-260	1700	null	null

Composição de Relações (exemplos)

- emprestimo **natural inner join** devedor

nome_agencia	numero_emprestimo	total	nome_cliente
Downtown	L-170	3000	Jones
Redwood	L-230	4000	Smith

- emprestimo **natural right outer join** devedor

nome_agencia	numero_emprestimo	total	nome_cliente
Downtown	L-170	3000	Jones
Redwood	L-230	4000	Smith
null	L-155	null	Hayes

Composição de Relações (exemplos)

- emprestimo **full outer join** devedor **using** (numero_emprestimo)

nome_agencia	numero_emprestimo	total	nome_cliente
Downtown	L-170	3000	Jones
Redwood	L-230	4000	Smith
Perryridge	L-260	1700	null
null	L-155	null	Hayes

- Encontrar todos os clientes que tenham uma conta ou um emprestimo (mas não os dois) no Banco.

  select nome_cliente

  from (depositante **natural full outer join devedor**)

  where numero_conta **is** null **or** numero_emprestimo **is** null

Linguagem de Definição de Dados (DDL)

Permite não só a especificação de um conjunto de relações, como também informações acerca de cada uma das relações, incluindo:

- O esquema de cada relação.
- O domínio dos valores associados a cada atributo.
- Regras de Integridade.
- O conjunto de índices para manutenção de cada relação.
- Informações sobre segurança e autoridade sobre cada relação.
- A estrutura de armazenamento físico de cada relação no disco.

Tipos de Domínios em SQL

- **char(n**). É uma cadeia de caracter de tamanho fixo, com o tamanho n definido pelo usuário.
- **varchar(n**). É uma cadeia de caracter de tamanho variável, como tamanho máximo n definido pelo usuário.
- **in**t. É um inteiro (um subconjunto finito dos inteiros que depende do equipamento).
- **smallin**t. É um inteiro pequeno (um subconjunto do domínio dos tipos inteiros dependente do equipamento).
- **numeric(p,d**). É um numero de ponto fixo cuja precisão é definida pelo usuário. O numero consiste de p dígitos(mais o sinal), sendo que d dos p dígitos estão à direita do ponto decimal.

Tipos de Domínios em SQL (continuação)

- **real, double precisio**n. São numeros de ponto flutuante e ponto flutuante de precisão dupla cuja precisão é dependente do equipamento.
- **float(n**). É um numero de ponto flutuante com a precisão definida pelo usuário em pelo menos *n* dígitos.
- **date**. Datas, contém um ano (com quatro dígitos), mês e dia do mês.
- **time**. Representa horário, em horas, minutos e segundos.
  - Valor nulo é um membro de todos os tipos de domínios. Declarando um um domínio de atributo como sendo **not null** proibi-se, assim, a inserção de valores nulos para esse tipo de atributo.
  - **create domain** em SQL-92 permite definir domínios.

    create domain person-name **cha**r(20) **not null**

Definição de Esquema em SQL

- Definimos uma relação SQL usando o comando **create table**:

  create table r (A_1 D_1, A_2 D_2, ..., A_n D_n,
  ⟨regras de integridade$_1$⟩,
  ...,
  ⟨regras de integridade$_k$ ⟩)

  - r é o nome da relação
  - cada A_i é o nome de um atributo no esquema da relação r
  - D_i é o tipo de domínio dos valores no domínio dos atributos A_i

- Exemplo:

```
create table agencia
    (nome_agencia    char(15) not null,
     cidade_agencia  char(30),
     fundos          integer)
```

Regras de Integridade em create table

- **not null**
- **primary key** (A_1, ..., A_n)
- **check** (P), onde P é um predicado

  Exemplo: declarar nome_agencia como chave primária para a relação agencia e verificar se o valor de fundos não é negativo.

```
create table agencia
    (nome_agencia    char(15) not null,
     cidade_agencia  char(30),
     fundos          integer,
     primary key (nome_agencia),
     check (fundos >=0))
```

- **primary key** são necessariamente declarados como **not null** in SQL-92

Comandos drop *e* alter table

- O comando **drop table** remove todas as informações de uma relação do banco de dados..
- O comando **alter table** é usado para adicionar atributos a uma relação existente. Todas as tuplas da relação recebem valores nulo para seu novo atributo. A forma do comando **alter table** e:
- **alter table** r **add** A D

  onde A é o nome do novo atributo que será adicionado e D é seu domínio.
- O comando **alter table** também pode ser usado para remover atributos de uma relação

  alter table r **drop** A

  onde A é o nome do atributo a ser removido da relação r.

Comandos SQL Embutidos

- O padrão SQL define que a SQL será embutida em uma variedade de linguagens de programação, como Pascal, PL/I, Fortran, C, e Cobol.

- A linguagem na qual são embutidas consultas SQL é chamada linguagem hospedeira, e as estruturas SQL permitidas na linguagem hospedeira são denominadas SQL embutida.

- EXEC SQL é usado para identificar os pedidos em SQL embutida para o pré-processador

  EXEC SQL <comando SQL embutido > END EXEC

Exemplo - SQL Embutido

Suponha que temos na linguagem hospedeira uma variável chamada **total** e que desejamos encontrar os nomes e cidades dos clientes dos clientes que tenham mais de um total em dólares em qualquer conta.

– Podemos escrever essa consulta como segue:

```
EXEC SQL
    declare c cursor for
    select nome_cliente,cidade_cliente
    from depósito,cliente
    where depósito.nome_cliente = cliente.nome_cliente
          and depósito.saldo > :total
END-EXEC
```

Exemplo - SQL Embutido

- O comando **open** faz com que a consulta seja avaliada

  EXEC SQL **open** c END-EXEC

- O comando **fetch** determina os valores de uma tupla que serão colocados em variáveis da linguagem host.

  EXEC SQL **fetch** c **into** :cn :an END-EXEC

  Pode-se utilizar um laço para processar cada tupla do resultado; uma variável na área de comunicação do SQL indica que não há mais tupla a ser processada.

- O comando **close** faz com que o sistema de banco de dados remova a relação temporária mantida para o resultado da consulta.

  EXEC SQL **close** c END-EXEC

SQL Dinâmico

- Permite que programas construam e submetam consultas SQL em tempo de execução.
- Exemplo de uso de SQL dinâmico dentro de um programa C.

```
char *sqlprog ='update conta set saldo = saldo *  1.05
                where numero_conta =?'
EXEC SQL prepare dinprog from :sqlprog;
char conta[10] = 'A-101';
EXEC SQL execute dinprog using :conta;
```

- O programa dinâmico contém uma “?”, que é colocada para manter um valor gerado quando o programa SQL é executado.

Outros Recursos SQL

- Linguagens de Quarta Geração – linguagem especial para apoio aos programadores de aplicação na criação de telas de interface com o usuário e para formatação de dados na criação de relatórios; disponível em muitos produtos comerciais de banco de dados.

- SQL sessions – proporcionam uma abstração de um cliente de um servidor (possivelmente remota)
  - cliente conecta o servidor SQL, estabelecendo uma sessão
  - executa uma série de comandos
  - fecha a sessão (desconecta)
  - oferece comandos de commit para a efetivação do trabalho realizado durante a sessão, ou rollback para não efetivá-los

- Um ambiente SQL contém diversos componentes, inclusive a identificação do usuário e um esquema, que identifica qual dos diversos esquemas a sessão está usando.