SQL

Database System Concepts, 5th Ed.
©Silberschatz, Korth and Sudarshan

2018/2019

Linguagem de Definição de Dados

- Permite especificar o conjunto de relações e a informação sobre cada relação, incluindo:
  - O esquema de cada relação
  - O Domínio dos valores associados a cada atributo.
  - Restrições de integridade
  - O conjunto de índices a ser mantido para cada relação.
  - Informação sobre a segurança e autorizações para cada relação.
  - As estruturas de armazenamento de cada relação no disco.

Tipos dos Domínios no SQL

- **char(n)**. Um string de tamanho fixo (n caracteres com n a ser especificado pelo utilizador).
- **varchar(n)**. Uma string de tamanho máximo n.
- **int**. Inteiro (um subconjunto finito dos inteiros, depende da máquina).
- **smallint**. Um inteiro pequeno (um subconjunto dos inteiros, depende da máquina).
- **numeric(p,d)**. p dígitos inteiros e d casas decimais.
- **real**, double precision.
- **float(n)**.
  Ver mais no capítulo 4.

Construtor: Create Table

- Uma relação em SQL é definida usando o comando create table:

  create table $r(A_1 D_1, A_2 D_2, \ldots, A_n D_n$
  $(integrity - constraint_1),$
  $\ldots,$
  $(integrity - constraint_k))$

  - r é o nome da relação
  - cada A_i é o nome de um atributo do esquema de r
  - D_i é o tipo de dados dos valores do domínio do atributo A_i
  - $(integrity - constraint_i$ é uma restrição que deve ser verificada pelos valores do atributo A_i.

- Exemplo:

  create table $branch(branch\_name\ char(15)\ not\ null,$
  $branch\_city\ char(30),$
  $assets\ integer)$

Restrições de Integridade no Create Table

- **not null**
- **primary key** $(A_1, A_2, \ldots, A_n)$
- Exemplo:
  Declarar branch_name como chave primária de branch

  create table *branch*(
  branch_name char(15),
  branch_city char(30),
  assets integer
  primary key (*branch_name*))

Construtores: **Drop** e **Alter Table**

- O comando **drop table** remove toda a informação sobre a relação na base de dados.
  drop table r
- O comando **alter table** é usado para adicionar atributos a uma relação que já existe:
  - **alter table** r add A D
  - com **A** é o nome do atributo a ser adicionado à relação **r**
  - e **D** o domínio de **A**.
  - Todos os tuplos da relação ficam com o valor null no novo atributo.
  - O comando alter table também pode ser usado para retirar atributos de uma relação:
    alter table r **drop** A
    com **A** o nome de um atributo de **r**
  - Alguns sistemas de gestão bases de dados não permitem o drop de atributos

Estrutura básica de uma interrogação (Query)

- O SQL é baseado em operações sobre relações e conjuntos com algumas modificações
- Uma query SQL típica tem a forma:

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

  A_i é um atributo
  R_i é uma relação
  P é um predicado.
  - Esta query é equivalente à expressão de algebra relacional: $\pi_{A_1,A_2,...,A_n}\sigma_P(r_1 \times r_2 \times \cdots \times r_n)$
  - O resultado da query SQL é uma relação.

A clausula **select**

- A clausula **select** lista os atributos que devem estar no resultado da query
- Corresponde à operação projecção na álgebra relacional
- Exemplo: Encontrar os nomes de todas as agências na relação empréstimo:
  select branch_name **from** loan
- Na álgebra relacional a expressão seria:
  $\pi_{Branch\_name}(loan)$
- NOTA: os nomes em SQL não são sensíveis ao caso (letra maiúscula ou letra minúscula) E.g. Branch_Name = BRANCH_NAME = branch_name

A clausula **select**

- O SQL permite duplicados nas relações e nos resultados das queries.
- Para forçar a eliminação dos duplicados usa-se a palavra distinct depois do **select**
- Encontrar os nomes das agências na relação empréstimo removendo os duplicados
  select distinct branch_name
  from loan
- A palavra chave all especifica que os duplicados não devem ser removidos.
  select all branch_name
  from loan

A clausula **select**

- Um asterisco * na clausula select denota todos os atributos
  select *
  from loan
- A clausula **select** pode ter expressões aritméticas envolvendo os operadores, $+$, -, *, e $/$, operando sobre constantes ou atributos
  A query:
  select loan_number, branch_name, amount * 100
  from loan
  Retorna a relação que tem os mesmo tuplos da relação loan com os mesmos valores excepto no atributo amount que tem o valor multiplicado por 100.

A clausula **where**

- A clausula **where** especifica as condições que o resultado deve satisfazer
- Corresponde ao predicado do operador selecção da álgebra relacional.
- Para encontrar todos os números dos empréstimos feitos na agência de Perryridge com valores maiores do que $1200.
  select loan_number
  from loan
  where branch_name = Perryridge and amount > 1200
  Os resultados das comparações podem ser combinados usando as conectivas lógicas: e (and), ou (or) e não (not).

A clausula **where**

- O SQL tem a com o operador de comparação between
- Exemplo: Encontrar os números dos empréstimos com valores entre \$90,000 e \$100,000 (i.e. $>$ \$90,000 and $\leq$ \$100,000)
  select loan_number
  from loan
  where amount **between** 90000 **and** 100000

A clausula **from**

- A clausula **from** lista as relações envolvidas na query
- Corresponde ao operador produto cartesiano da álgebra relacional
- Ex: Encontrar o produto cartesiano de *borrower* $\times$ *loan*
  select *
  from borrower **,** loan
- Encontrar o nome, número do empréstimo e valor de todos os clientes que têm um empréstimo na agência de Perryridge.
  select customer_name, borrower.loan_number, amount
  from borrower, loan
  where borrower.loan_number = loan.loan_number and
  branch_name = 'Perryridge'

A clausula **Rename** - Renomear relações

- O SQL permite renomear as relações e atributos usando a clausula **as**:
  old-name **as** new-name
- Ex: Encontrar o nome, o número e o valor dos empréstimos de todos os clientes; dê o nome loan_id à coluna loan_number.

  select customer_name, borrower.loan_number **as** loan_id, amount
  from borrower, loan
  where borrower.loan_number = loan.loan_number

A clausula **Rename** - Renomear relações

- As relações podem ser renomeadas na clausula **from** usando a clausula **as**.
- Encontrar os nomes dos clientes e os números dos seus empréstimos para todos os clientes que têm um empréstimo no banco.

  select customer_name, T.loan_number, S.amount
  from borrower **as** T, loan **as** S
  where T.loan_number = S.loan_number
- Encontrar os nomes de todas as agências que têm mais negócios do que as agências da cidade de Brooklyn.

  select distinct T.branch_name
  from branch **as** T, branch **as** S
  where T.assets > S.assets and S.branch_city = 'Brooklyn'
- A palavra chave **as** é opcional, pode não aparecer
  borrower **as** T = borrower T

Operações sobre Strings

- O SQL tem o operador "like" para comparar strings.
  Este operador pode ser usado com dois caracteres especiais nas strings:
  - (%). O caractere % substitui qualquer substring.
  - (_). O caractere _ substitui qualquer caractere.
- Encontrar nomes de clientes que na rua têm a substring "Main".

```
select  customer_name
from customer
where customer_street like '%Main%'
```

- SQL tem um conjunto de operações sobre strings, ex:
  - concatenar (usar "||")
  - Converter de maiúscula em minúscula
  - Comprimento da string, extrair substring, etc

Ordem de listagem dos tuplos

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

  select distinct customer_name
  from borrower, loan
  where borrower.loan_number = loan.loan_number and branch_nam
- Pode-se especificar **desc** para a ordem ser descendente ou **asc** para a ordem ser ascendente, a ordem ascendente é o que o SQL faz se não se indicar **asc** ou **desc**.
  Exemplo:
  order by customer_name **desc**

Duplicados

- Em relações com duplicados, o SQL pode definir quantas cópias do tuplo aparecem no resultado de uma operação.
- **Multiset** versão de alguns operadores da álgebra relacional com multiconjuntos – dadas as versões multiconjuntos de r_1 e r_2:
  1. $\sigma_\theta(r_1)$: se há c_1 copias do tuplo t_1 em r_1, e t_1 satisfaz θ, então há c_1 copias de t_1 em $\sigma_\theta(r_1)$.
  2. $\pi_A(r)$: para cada cópia do tuplo t_1 em r_1, existe a cópia do tuplo $\pi_A(t_1)$ onde $\pi_A(t_1)$ significa a projecção de t_1 em A.
  3. $r_1 \times r_2$: se há c_1 cópias do tuplo t_1 em r_1 e c_2 copias do tuplo t_2 em r_2, então existem $c_1 * c_2$ copias do tuplo $t_1.t_2$ em $r_1 \times r_2$

Duplicados

- Exemplo:
  Considere as relações multiconjunto $r_1(A, B)$ e $r_2(C)$:
  $r_1 = \{(1, a)(2, a)\}$ e $r_2 = \{(2), (3), (3)\}$
  $\pi_B(r_1)$ deve ser $\{(a), (a)\}$, e $\pi_B(r_1) \times r_2$
  $\{(a, 2), (a, 2), (a, 3), (a, 3), (a, 3), (a, 3)\}$
- Semântica dos duplicados no SQL :
  select $A_1, , A_2, \ldots, A_n$
  from $r_1, r_2, \ldots, r_m$
  where P
  É equivalente à versão multiconjunto da expressão:
  $\pi_{A_1, A_2, \ldots, A_n}(\sigma_P(r_1 \times r_2 \times \cdots \times r_n))$

Operações sobre conjuntos

- As operações **union**, **intersect**, e **except** operam sobre relações e correspondem às operações da álgebra relacional: $\cup, \cap, -$.
- Cada um destes operadores elimina automaticamente os duplicados;para manter os duplicados é necessário usar: **union all**, **intersect all** e **except all**.
- Suponha que um tuplo ocorre **m** vezes em r e **n** vezes em s, então ocorre:
  - **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 sobre conjuntos

- Encontrar todos os clientes que têm um empréstimo, uma conta ou ambos:

  (**select** customer_name
  from depositor)
  union
  (**select** customer_name
  from borrower)

- Encontrar todos os clientes que têm um empréstimo e uma conta.

  (**select** customer_name
  from depositor)
  intersect
  (**select** customer_name
  from borrower)

Operações sobre conjuntos

- Encontrar todos os clientes que têm uma conta mas não têm um empréstimo.

  (**select** customer_name
  from depositor)
  except
  (**select** customer_name
  from borrower)

Funções de agregação

- Estas funções operam sobre valores multiconjunto de uma coluna de uma relação e retornam um valor
  - avg: valor médio
  - min: valor minimo
  - max: valor máximo
  - sum: soma dos valores
  - count: numero de valores

Funções de agregação

- Encontrar o saldo médio das contas da agência de Perryridge.
  select avg (balance)
  from account
  where branch_name = 'Perryridge'
- Encontrar o número de tuplos na relação cliente.
  select count (*)
  from customer
- Encontrar o número de depositantes no banco.
  select count (distinct customer_name)
  from depositor

Operador de agregação - Group By

- Encontrar o número de depositantes em cada agência.
  select branch_name, count (distinct customer_name) **from** deposito
  where depositor.account_number = account.account_number
  group by branch_name
  Nota: Os atributos da clausula **select** fora da função de agregação devem aparecer na lista do **group by**

Funções de agregação – Having Clause

- Encontrar os nomes de todas as agências que têm o saldo médio das contas maior do que $1.200.

  select branch_name, avg (balance)
  from account
  group by branch_name
  having avg (balance) $>$ 1200

  Nota: Os predicados na clausula having são aplicados depois da formação dos grupos mas os predicados da clausula **where** são aplicados antes da formação dos grupos

Valores Null

- É possível que alguns tuplos tenham o valor null nalguns atributos
- null significa que o valor é desconhecido ou que não existe.
- O predicado is null pode ser usado para verificar se um valor é null.

  Exemplo: Encontrar todos os números de empréstimos que aparecem na relação empréstimo com null no valor.

  select loan_number
  from loan
  where amount is null
- O resultado de uma expressão aritmética que envolva nulls é null

  Exemplo: 5 + null retorna null
- As funções de agregação ignoram os nulls

Valores Null e Lógica a três valores

- Qualquer comparação com null retorna unknown Exemplo: 5 ¡ null or null ¡¿ null or null = null
- Lógica a três valores com o unknown:
  OR: (unknown or true) = true,
  (unknown or false) = unknown
  (unknown or unknown) = unknown
  AND: (true and unknown) = unknown,
  (false and unknown) = false,
  (unknown and unknown) = unknown
  NOT: (not unknown) = unknown
- "P is unknown" é avaliado como true se o predicado P é avaliado como unknown
- O resultado do predicado da clausula **where** é tratado como False se for avaliado como unknown

Valores Null e agregação

- Qual é o total do valor de todos os empréstimos
  select sum (amount)
  from loan
- A query acima ignora os valores null
- O resultado é null se não existir nenhum valor diferente de null
- Todas as operações de agregação (excepto count(*)) ignoram os tuplos com valor null no atributo de agregação.

Perguntas imbricadas

- O SQL tem um mecanismo para lidar com queries imbricadas.
- Uma subquery é uma expressão **select-from-where** que está imbricada noutra.
- Um uso comum de perguntas imbricadas é para testar comparação de conjuntos, cardinalidade de conjuntos e pertença a conjunto.

Exemplo de Query

- Encontrar todos os clientes que têm uma conta e um empréstimo no banco.
  select distinct customer_name
  from borrower
  where customer_name in (**select** customer_name **from** depositor)
- Encontrar todos os clientes que têm um empréstimo mas não têm uma conta
  select distinct customer_name
  from borrower
  where customer_name not in (**select** customer_name **from** depositor)

Exemplo

- Encontrar todos os clientes que têm uma conta e um empréstimo na agência de Perryridge
  select distinct customer_name
  from borrower, loan
  where borrower.loan_number = loan.loan_number and branch_name = 'Perryridge' and
  (branch_name, customer_name) in (**select** branch_name, customer_name **from** depositor, account **where** depositor.account_number = account.account_number)

Comparação de conjuntos

- Encontrar todas as agências que têm maior volume de negócios do que alguma agência da cidade de Brooklyn.
  select distinc T.branch_name
  from branch as T, branch as S
  where T.assets $>$ S.assets and
  S.branch_city $=$ 'Brooklyn'
- A mesma query usando a clausula $>$ some
  select branch_name
  from branch
  where assets $>$ some (**select** assets **from** branch **where** branch_city $=$ 'Brooklyn')

Definição da clausula Some

- $F < comp >$ some $r \Leftrightarrow \exists t \in r$ tal que $F < comp > t$. Com $< comp > \in \{\leq, \geq, <, >, =\}$

ex:

$$(5 < some \left\{ \begin{array}{c} 0 \\ 5 \\ 6 \end{array} \right\}) = true$$

$$(5 < some \left\{ \begin{array}{c} 0 \\ 5 \end{array} \right\}) = false$$

$$(5 = some \left\{ \begin{array}{c} 0 \\ 5 \end{array} \right\}) = true$$

$$(5 \neq some \left\{ \begin{array}{c} 0 \\ 5 \end{array} \right\}) = true$$

$(= some) \equiv$ in

$(\neq some) \not\equiv$ not in

Exemplo de Query

- Encontrar os nomes de todas as agências que têm maior volume de negócios que todas as agências da cidade de Brooklyn.
  select branch_name
  from branch
  where assets $>$ all (**select** assets **from** branch **where** branch_city $=$ 'Brooklyn')

Definição da clausula ALL

- $F <comp>$ all $r \Leftrightarrow \forall t \in r$ tal que $F <comp> t$. Com $<comp> \in \{\leq, \geq, <, >, =\}$

ex:

$$(5 < all \left\{ \begin{array}{c} 0 \\ 5 \\ 6 \end{array} \right\}) = false$$

$$(5 < all \left\{ \begin{array}{c} 6 \\ 10 \end{array} \right\}) = true$$

$$(5 = all \left\{ \begin{array}{c} 4 \\ 5 \end{array} \right\}) = false$$

$$(5 \neq all \left\{ \begin{array}{c} 4 \\ 6 \end{array} \right\}) = true$$

$(\neq all) \equiv$ not in

$(= all) \not\equiv$ in

Testar relações vazias

- O construtor exists retorna o valor true se o argumento da subquery não é vazio.
- exists r $\Leftrightarrow$ r $\neq \emptyset$
- not exists r $\Leftrightarrow$ r $= \emptyset$

Exemplo de uma Query

- Encontrar todos os clientes que têm uma conta em todas as agências da cidade de Brooklyn.

```
select distinct S.customer_name
from depositor as S
where not exists ( (select branch_name
                    from branch
                    where branch_city = 'Brooklyn')
                    except
                    (select R.branch_name
                     from depositor as T, account as R
                    where T.account_number = R.account_number
and S.customer_name = T.customer_name ))
```

- Note que $X – Y = \emptyset \Leftrightarrow X \subseteq Y$
- Nota: não se pode escrever a query usando **= all** e as suas variantes
- $\pi_{Custumer\_name, Branch\_name}(depositer \bowtie account) \div \pi_{Branch\_name}(\sigma_{Branch\_city = Brooklyn}(branch))$

Testar a existência de duplicados

- O constructor **unique** testa se uma subquery tem tuplos duplicados no seu resultado, **unique** sucede se não há duplicados.
- Encontrar todos os clientes que têm uma só conta na agência de Perryridge.

```
select T.customer_name
from depositor as T
where unique (select R.customer_name
              from account, depositor as R
              where T.customer_name = R.customer_name
and R.account_number = account.account_number and
account.branch_name = 'Perryridge')
```

Testar a existência de duplicados

- Encontrar todos os clientes que têm pelo menos duas contas na agência de Perryridge.

```
select distinct T.customer_name
from depositor as T
where not unique ( select R.customer_name
                   from account, depositor as R
                   where T.customer_name = R.customer_name
and R.account_number = account.account_number and
account.branch_name = 'Perryridge')
```

Relações derivadas

- O SQL permite usar uma expressão (subquery) na clausula **from**
- Encontrar o saldo médio das contas das agências que têm um saldo médio das contas superior a $1200.

```
select branch_name, avg_balance
from (select branch_name, avg (balance)
            from account
            group by branch_name ) as branch_avg (
            branch_name, avg_balance )
where avg_balance > 1200
```

Clausula WITH

- A clausula with permite definir uma vista temporária que está disponível só para a query onde a clausula with ocorre,
- Encontrar todas as contas com o saldo máximo.

  with max_balance (value) as **select** max (balance)
  from account
  select account_number
  from account, max_balance
  where account.balance = max_balance.value

Queries complexas com a clausula With

- Encontrar todas as agências onde o valor total dos depósitos nas contas é superior à média do total dos depósitos de todas as agências.

```
with branch_total (branch_name, value) as
        select branch_name, sum (balance)
        from account
        group by branch_name
with branch_total_avg (value) as
        select avg (value)
        from branch_total
select branch_name
from branch_total, branch_total_avg
where branch_total.value ≥ branch_total_avg.value
```

Vistas (View)

- Nalgumas situações não é conveniente que todos os utilizadores vejam o modelo lógico inteiro (i.e. todas as relações que estão na bases dados).
- Considere um utilizador que precisa de saber o nome dos clientes, os números dos empréstimos e os nomes das agências mas não precisa (nem deve) saber o valor do empréstimo. Este utilizador deve ver a seguinte relação:
  (**select** customer_name, borrower.loan_number, branch_name
  from borrower, loan
  where borrower.loan_number = loan.loan_number)
- A vista (view) fornece o mecanismo para esconder parte dos dados a certos utilizadores.
- Uma relação que não está no modelo conceptual mas é visível para um utilizador como uma 'relação virtual' chama-se vista(view).

Definição de uma vista (View)

- Uma vista é definida com clausula create view
- **create view** v **as** <expressão da query>
- onde <expressão da query> é uma expressão SQL.
- O nome da view é representado por v.
- Depois de uma view estar definida o seu nome é usado para referir a relação virtual gerada pela view.
- Quando a view é criada, a expressão da query é armazenada na base de dados e será usada nas queries que referem a view.

Exemplos

- Criar a vista (view) que consiste nas agências e seus clientes.
  create view all_customer as (**select** branch_name, customer_name
  from depositor, account
  where depositor.account_number = account.account_number)
  union
  (**select** branch_name, customer_name
  from borrower, loan
  where borrower.loan_number = loan.loan_number)
- Encontrar todos os cliente da agência de Perryridge
  select customer_name
  from all_customer
  where branch_name = 'Perryridge'

Definição de Views usando outras Views

- Uma view pode ser usada numa expressão para definir outra view
- Uma view relation v_1 diz-se dependente directamente de uma view relation v_2 se v_2 é usada na expressão que define v_1
- Uma view relation v_1 diz-se dependente de uma view relation v_2 se v_1 depende directamente de v_2 ou se existe caminho de dependências de v_1 para v_2
- Uma view relation v diz-se recursiva se depende de si mesma.

Expansão de Views

- Uma forma de definir o significado das views usando outras views.
- Considere a view v1 definida pela expressão e1 que pode conter outras views.
- A expansão da view repete o seguinte passo de substituição:
  Repeat Encontrar uma view relation v_i in e_1
  Substituir a view relation v_i pela expressão que define v_i **until** não existam mais view relations em e_1
- Desde que as definições das views não sejam recursivas o loop termina.

Modificação da base de dados – (remover) Deletion

- Remova todos os tuplos das contas da agência de Perryridge.
  delete from account **where** branch_name = 'Perryridge'
- Remova todas as contas de todas as agências da cidade de 'Needham'
  delete from account **where** branch_name in (**select** branch_name **from** branch **where** branch_city = 'Needham')

Modificação da base de dados – (remover) Deletion

Exemplo

- Remova todas as contas que têm um saldo inferior ao saldo médio das contas do banco.
  delete from account **where** balance $<$ (**select** avg (balance) **from** account)
- Problema: à medida que removemos tuplos da conta o saldo médio das contas altera-se.
- Solução:
  1. Calcular, primeiro, avg balance e depois procurar os tuplos para remover
  2. A seguir, remover todos os tuplos encontrados sem recalcular o saldo médio.

Modificação da base de dados – Inserção (Insertion)

- Adicionar um tuplo novo em account
  insert into account **values** ('A-9732', 'Perryridge', 1200)
- Ou de forma equivalente
  insert into account (branch_name, balance, account_number)
  values ('Perryridge', 1200, 'A-9732')
- Adicionar um tuplo novo em account com o saldo null
  insert into account **values** ('A-777','Perryridge', null)

Modificação da base de dados – Inserção (Insertion)

- Ofereça de presente a todos os clientes com empréstimo na agência de Perryridge, $200 numa nova conta. Use o número do empréstimo como número da nova conta.

insert into account **select** loan_number, branch_name, 200 **from** loan **where** branch_name = 'Perryridge'
insert into depositor **select** customer_name, loan_number **from** loan, borrower **where** branch_name = 'Perryridge' and loan.account_number = borrower.account_number

- A clausula **select from where** é avaliada completamente antes do seu resultado ser inserido na relação.

Modificação da base de dados – Actualização (Updates)

- Acrescente a todas as contas com saldos superiores a \$10,000, 6% do saldo, e a todas as outras contas adicione 5%.
- Em dois passos
- **update** account **set** balance = balance * 1.06 **where** balance $>$ 10000
- **update** account **set** balance = balance * 1.05 **where** balance $\leq$ 10000
- A ordem é importante
- Pode ser feito usando a clausula case

Clausula Case para actualizações condicionais

- Acrescente a todas as contas com saldos superiores a \$10,000, 6% do saldo, e a todas as outras contas adicione 5%.
- **update** account **set** balance = **case**
- **when** balance $\leq$ 10000 **then** balance * 1.05
- **else** balance * 1.06
- **end**

Actualização de uma View

- Criar uma view com os dados da relação loan, escondendo o atributo amount
- **create view** loan_branch **as select** loan_number, branch_name **from** loan
- Adicionar um novo tuplo a branch_loan
- **insert into** branch_loan values ('L-37', 'Perryridge')
- Esta inserção obriga à inserção do tuplo ('L-37', 'Perryridge', null) na relação loan

Actualizações nas Views

- Algumas actualizações nas Views são impossíveis de traduzir em actualizações nas relações da base de dados
  - **create view** v **as select** loan_number, branch_name, amount
    from loan
    where branch_name = 'Perryridge'
  - **insert into** v **values** ('L-99','Downtown', '23')
- Outras não podem ser traduzidas de uma só forma
  - **insert into** all_customer **values** ('Perryridge', 'John')
  - Tem que se escolher a relação loan ou account, e criar um novo número de loan ou account!
- Algumas implementações de SQL só permitem actualizações em vistas simples definidas com uma única relação.

Junção de relações - Join

- As operações **Join** recebem duas relações e retornam uma nova relação.
- Estas operações usam-se na clausula **from**
- Condição do **Join** — define que atributos dos tuplos das duas relações devem ser iguais e que atributos ficam no resultado.
- Tipo de **Join** — Define como é que os tuplos de uma relação que não encaixam na outra relação são tratados

Join Types	Join Conditions
inner join left outer join right outer join left outer join	natural on <predicate> using $A_1, A_2, \ldots, A_n$

Junção de relações — Exemplos

- Relação loan e borrower

loan_number	*branch_name*	*amount*
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700

loan

customer_name	*loan_number*
Jones	L-170
Smith	L-230
Hayes	L-155

borrower

- Na informação do borrower falta o L-260 e na do loan falta o L-155
- loan **inner join** borrower **on** loan.loan_number = borrower.loan_number

loan_number	*branch_name*	*amount*	*customer_name*	*loan_number*
L-170	Downtown	3000	Jones	L-170
L-230	Redwood	4000	Smith	L-230

Junção de relações — Exemplos

- Relação loan e borrower

loan_number	*branch_name*	*amount*
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700

loan

customer_name	*loan_number*
Jones	L-170
Smith	L-230
Hayes	L-155

borrower

- Na informação do borrower falta o L-260 e na do loan falta o L-155
- loan **left outer join** borrower **on** loan.loan_number = borrower.loan_number

loan_number	*branch_name*	*amount*	*customer_name*	*loan_number*
L-170	Downtown	3000	Jones	L-170
L-230	Redwood	4000	Smith	L-230
L-260	Perryridge	1700	*null*	*null*

Junção de relações — Exemplos

- Relação loan e borrower

loan_number	*branch_name*	*amount*
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700

loan

customer_name	*loan_number*
Jones	L-170
Smith	L-230
Hayes	L-155

borrower

- Na informação do borrower falta o L-260 e na do loan falta o L-155
- loan **natural inner join** borrower

loan_number	*branch_name*	*amount*	*customer_name*	*loan_number*
L-170	Downtown	3000	Jones	L-170
L-230	Redwood	4000	Smith	L-230

Junção de relações — Exemplos

- Relação loan e borrower

loan_number	*branch_name*	*amount*
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700

loan

customer_name	*loan_number*
Jones	L-170
Smith	L-230
Hayes	L-155

borrower

- Na informação do borrower falta o L-260 e na do loan falta o L-155
- loan **natural right outer join** borrower

loan_number	*branch_name*	*amount*	*customer_name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-155	*null*	*null*	Hayes

Junção de relações — Exemplos

- Relação loan e borrower

loan_number	*branch_name*	*amount*
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700

loan

customer_name	*loan_number*
Jones	L-170
Smith	L-230
Hayes	L-155

borrower

- Na informação do borrower falta o L-260 e na do loan falta o L-155
- loan **full outer join** borrower **using** (loan_number)

loan_number	*branch_name*	*amount*	*customer_name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-260	Perryridge	1700	*null*
L-155	*null*	*null*	Hayes

Junção de relações — Exemplos

- Todos os clientes que têm ou uma conta ou um empréstimo (mas não os dois) no banco.
  select customer_name
  from (depositor **natural full outer join** borrower)
  where account_number is null or loan_number is null

Chaves estrangeiras

```
CREATE TABLE products (
                product_no integer PRIMARY KEY,
                name text,
                price numeric );
CREATE TABLE orders (
                order_id integer PRIMARY KEY,
                shipping_address text,
...
);
CREATE TABLE order_items (
                product_no integer REFERENCES products ON
DELETE RESTRICT,
                order_id integer REFERENCES orders ON DELETE
CASCADE,
                quantity integer,
                PRIMARY KEY (product_no, order_id)
);
```