Capítulo 4: SQL

- Linguagem de Definição de Dados
- Estrutura básica
- Operações com conjuntos
- Funções de agregação
- Valores nulos
- Subconsultas embutidas
- Relações derivadas
- Junções
- Vistas
- Modificação da Base de Dados
- Embedded SQL
- Dynamic SQL
- Funções e Procedimentos
- SQL Recursivo

Ordenando os tuplos

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

  select distinct *customer_name*
  from *borrower, loan*
  where *borrower.loan_number = loan.loan_number* **and**
  branch_name = 'Perryridge'
  order by *customer_name*

- Pode-se especificar **desc** para ordenação descente ou **asc** para ordenação ascendente, para cada atributo; por omissão, assume-se ordem ascendente.
  - E.g. **order by** *customer_name* **desc**
- Pode-se ter mais do que uma chave de ordenação, separando-as com vírgulas

Estrutura Básica

- SQL é baseada em operações de conjuntos e de álgebra relacional com algumas modificações e extensões
- Uma consulta SQL básica tem a forma:

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

  - A_i*s* representam atributos
  - r_i*s* representam relações
  - P é um predicado.
- A consulta é (quase) equivalente à expressão de álgebra relacional (a menos de tratamento de repetições):

$$\prod_{A1,\ A2,\ ...,\ An}(\sigma_P\ (r_1\ \ \text{x}\ r_2\ \ \text{x}\ \ ...\ \ \text{x}\ \ r_m))$$

- O resultado de uma consulta SQL é uma relação.

Duplicados

- Em relações com duplicados, a linguagem SQL especifica quantas cópias dos tuplos aparecem no resultado.
- *Versões Multiconjunto* de alguns operadores da álgebra relacional dadas relações multiconjunto r_1 e r_2:
  1. Se existem c_1 cópias do tuplo t_1 em r_1, e t_1 satisfaz a selecção $\sigma_{\theta,}$, então existem c_1 cópias de t_1 em $\sigma_\theta (r_1)$.
  2. Para cada cópia do tuplo t_1 em r_1, existe uma cópia do tuplo $\Pi_A(t_1)$ em $\Pi_A(r_1)$, onde $\Pi_A(t_1)$ denota a projecção do tuplo t_1.
  3. Se existem c_1 cópias do tuplo t_1 em r_1 e c_2 cópias do tuplo t_2 em r_2, então existem c_1 x c_2 cópias do tuplo t_1. t_2 em r_1 x r_2

Duplicados (cont.)

- Exemplo: supondo que as relações multiconjunto r_1 (A, B) e r_2 (C) são as seguintes:

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

- Então $\Pi_B(r_1)$ devolve {(a), (a)}, enquanto que $\Pi_B(r_1)$ x r_2 é

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

- A semântica de duplicados de SQL:

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

  é equivalente à versão *multiconjunto* da expressão:

  $$\Pi_{A1,, A2, ..., An}(\sigma_P (r_1 \text{ x } r_2 \text{ x } ... \text{ x } r_m))$$

Operações com Conjuntos

- As operações com conjuntos **union, intersect,** e **except (minus** no Oracle11g) operam sobre relações e correspondem aos operadores de álgebra relacional ∪, ∩, -.
- Cada uma das operações anteriores elimina os duplicados automaticamente. Para reter duplicados deve-se utilizar as respectivas versões multiconjunto **union all, intersect all** e **except all.**

  Supondo que um tuplo ocorre *m* vezes em *r* e *n* vezes em *s,* então ele 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 com Conjuntos

- Listar todos os clientes que têm um empréstimo ou uma conta:

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

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

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

- Listar os clientes que têm uma conta mas não têm empréstimos

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

Funções de Agregação

- Estas funções aplicam-se a multiconjuntos de valores de uma coluna de uma relação, devolvendo um valor

  avg: valor médio
  min: valor mínimo
  max: valor máximo
  sum: soma dos valores
  count: número de valores

Funções de Agregação (cont.)

- Determinar o saldo médio das contas na agência de Perryridge.

  select avg *(balance)*
  from *account*
  where *branch_name* = 'Perryridge'

- Calcular o número de clientes.

  select count (*)
  from *customer*

- Encontrar o número de depositantes do banco.

  select count (distinct *customer_name)*
  from *depositor*

Funções de agregação – Group By

- Listar o número de depositantes por agência.

  select *branch_name,* **count (distinct** *customer_name)*
  from *depositor, account*
  where *depositor.account_number = account.account_number*
  group by *branch_name*

  Nota: Atributos na cláusula **select** fora de funções de agregação têm de aparecer na lista **group by**

  Nota: Se aparecer mais do que um atributo em **group by**, então cada grupo é formado pelos tuplo com valores iguais *em todos* esses os atributos

Funções de Agregação – Cláusula Having

- Listar os nomes de todas as agências cujo valor médio dos saldos das contas é superior a $1,200.

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

Funções de Agregação – Cláusula Having

- Predicados na cláusula **having** são aplicados depois da formação dos grupos, enquanto que os predicados na cláusula **where** são aplicados antes da formação dos grupos.
- Encontrar a média dos saldos das contas dos clientes que vivem em Harrison e que têm pelo menos três contas.

```
select d.customer_name, avg (balance)
from depositor as d, account as a, customer as c
where d.account_number = a.account_number and
        d.customer_name = c.customer_name and
        c.city = 'Harrison'
group by d.customer_name
having count (distinct d.account_number) >= 3
```

Funções de agregação e álgebra relacional

- Um comando SQL genérico:

select $Fagr_1$, … $Fagr_n$
from R_1, …, R_m
where $Cond_W$
group by A_1, …, A_g
having $Cond_H$

- corresponde à expressão:

$$\Pi_{Fagr1, \ldots Fagrn} (\sigma_{Cond_H}({}_{A1, \ldots, Ag}\mathcal{g}_{Fagr1, \ldots Fagrn}(\sigma_{Cond_W} (R_1 \times \ldots \times R_m))))$$

Funções de Agregação (Oracle10g)

- AVG
- COLLECT
- CORR
- CORR_*
- COUNT
- COVAR_POP
- COVAR_SAMP
- CUME_DIST
- DENSE_RANK
- FIRST
- GROUP_ID
- GROUPING
- GROUPING_ID
- LAST
- MAX
- MEDIAN
- MIN
- PERCENTILE_CONT
- PERCENTILE_DISC
- PERCENT_RANK
- RANK
- REGR_ (Linear Regression) Functions
- STATS_BINOMIAL_TEST
- STATS_CROSSTAB
- STATS_F_TEST
- STATS_KS_TEST
- STATS_MODE
- STATS_MW_TEST
- STATS_ONE_WAY_ANOVA
- STATS_T_TEST_*
- STATS_WSR_TEST
- STDDEV
- STDDEV_POP
- STDDEV_SAMP
- SUM
- VAR_POP
- VAR_SAMP
- VARIANCE

Valores Nulos

- Os tuplos podem conter valores nulos, denotado por *null*, nalguns dos seus atributos.
- *null* significa um valor desconhecido ou que não existe.
- O predicado **is null** pode ser utilizado para testar a existência de valores nulos.
  - E.g. mostrar todos os números de empréstimos com um valor nulo na coluna *amount.*

    select *loan_number*
    from *loan*
    where *amount* **is null**
- O resultado de uma expressão aritmética com *null* é *null*
  - E.g. 5 + null devolve null
- Contudo, as funções de agregação ignoram os nulos
  - A seguir será analisado este assunto mais detalhadamente

Valores Nulos e Lógica Trivalente

- Qualquer comparação com *null* retorna *unknown*
  - *E.g. 5 < null ou null <> null ou null = null*
- Lógica trivalente usando o valor lógico *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 verdade se o valor de *P* é *unknown*
- Resultado da condição da cláusula **where** é tratado como *false* quando o seu valor é *unknown*
- ***Atenção:*** *no Oracle a cadeia vazia comporta-se como null, mas isto não é assim no SQL Standard!*

Valores Nulos e Agregados

- Calcule o total de todos os montantes dos empréstimos

  select sum (*amount)*
  from *loan*

  - A instrução acima ignora montantes nulos
  - Resultado é null se não existir nenhum montante não-nulo
- Todas as funções de agregação excepto **count(*)** ignoram tuplos com valores nulos nos atributos agregados.

Exemplos com funções de agregação

A	B
1	1
1	
1	3
2	
3	
3	
3	
4	1
4	2
4	2
4	2
4	3

```
SELECT SUM(a), AVG(DISTINCT a) FROM t
```

SUM(A)	AVG(DISTINCTA)
34	2.5

```
SELECT COUNT(b), SUM(b), AVG(b) FROM t
```

COUNT(B)	SUM(B)	AVG(B)
7	14	2

```
SELECT COUNT(*), COUNT(b), COUNT(DISTINCT b)
FROM t
```

COUNT(*)	COUNT(B)	COUNT(DISTINCTB)
12	7	3

Exemplos com funções de agregação

A	B
1	1
1	
1	3
2	
3	
3	
3	
4	1
4	2
4	2
4	2
4	3

```
SELECT COUNT(DISTINCT b), SUM(DISTINCT b),
AVG(DISTINCT b) FROM t
```

COUNT(DISTINCTB)	SUM(DISTINCTB)	AVG(DISTINCTB)
3	6	2

```
SELECT b, COUNT(*), SUM(b) FROM t GROUP BY b
```

B	COUNT(*)	SUM(B)
1	2	2
	5	
2	3	6
3	2	6

```
SELECT COUNT(*), COUNT(b), SUM(b), AVG(b)
FROM t WHERE a = 3;
```

COUNT(*)	COUNT(B)	SUM(B)	AVG(B)
3	0		

Exemplos com funções de agregação

A	B
1	1
1	
1	3
2	
3	
3	
3	
4	1
4	2
4	2
4	2
4	3

```
SELECT a,b,COUNT(*),COUNT(DISTINCT a), COUNT(b)
FROM t GROUP BY a, b ORDER BY a DESC, b
```

A	B	COUNT(*)	COUNT(DISTINCTA)	COUNT(B)
4	1	1	1	1
4	2	3	1	3
4	3	1	1	1
3		3	1	0
2		1	1	0
1	1	1	1	1
1	3	1	1	1
1		1	1	0

```
SELECT a FROM t GROUP BY a
HAVING MAX(b) IS NULL
```

A
2
3

Subconsultas imbricadas

- SQL disponibiliza um mecanismo para imbricar consultas umas dentro de outras.
- Uma subconsulta é uma expressão **select-from-where** que se encontra dentro de uma outra (sub)consulta.
- Subconsultas na clausula **from** são entendidas como cálculo de relações auxiliares.
- As subconsultas na clausula **where** são utilizadas habitualmente para efectuar testes de pertença a conjuntos, comparações entre conjuntos e calcular a cardinalidade de conjuntos.

Pertença a conjunto (in)

- Listar todos os clientes que têm contas e empréstimos no banco.

  select distinct *customer_name*
  from *borrower*
  where *customer_name* **in (select** *customer_name*
  from depositor)

- Encontrar todos os clientes que têm empréstimos mas não possuem contas no banco

  select distinct *customer_name*
  from *borrower*
  where *customer_name* **not in (select** *customer_name*
  from *depositor)*

Consulta de exemplo

- Listar todos os clientes que têm uma conta e empréstimos na agência de Perryride

```
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 )
```

- Nota: A consulta acima pode ser escrita de uma maneira muito mais simples. A formulação utilizada serve apenas para ilustrar as possibilidades da linguagem SQL.

Comparação de conjuntos (some)

- Apresentar todas as agências que têm activos superiores aos de alguma agência localizada em Brooklyn.

```
select distinct  T.branch_name
from branch as T, branch as S
where  T.assets > S.assets and
          S.branch_city = 'Brooklyn'
```

- A mesma consulta recorrendo à cláusula > **some**

```
select branch_name
from branch
where assets > some
          (select assets
           from branch
           where branch_city = 'Brooklyn')
```

Definição da cláusula Some

- F <comp> **some** $r \Leftrightarrow \exists\, t \in r$: (F <comp> t)
  em que <comp> pode ser : <, ≤, >, ≥ =, ≠

(5< **some** | 0 / 5 / 6 |) = true
(ler: 5 menor que algum tuplo na relação)

(5< **some** | 0 / 5 |) = false

(5 = **some** | 0 / 5 |) = true

(5 ≠ **some** | 0 / 5 |) = true (pois 0 ≠ 5)

(= some) ≡ in

No entanto, **(≠ some) ≢ not in**

Cláusula all

- Listar os nomes das agências com activos superiores aos de todas as agências localizadas em Brooklyn.

```
select branch_name
from branch
where assets > all
 (select assets
 from branch
 where branch_city = 'Brooklyn')
```

- Sem o **all**

```
(select branch_name from branch)

        except

(select branch_name
 from branch T,branch S
 where S.branch_city = 'Brooklyn' and T.assets < S.assets)
```

Definição da cláusula all

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

em que <comp> pode ser : <, ≤, >, ,≥, =, ≠

(5< **all** {0, 5, 6}) = false

(5< **all** {6, 10}) = true

(5 = **all** {4, 5}) = false

(5 ≠ **all** {4, 6}) = true (dado que 5 ≠ 4 e 5 ≠ 6)

(≠ all) ≡ not in

Contudo, **(= all) ≢ in**

Teste de Relações Vazias

- A construção **exists** devolve o valor **true** se a subconsulta é não vazia.
- **exists** $r \Leftrightarrow r \neq \emptyset$
- **not exists** $r \Leftrightarrow r = \emptyset$
- Encontrar os clientes que têm uma conta e um empréstimo

```
select customer_name
    from borrower
    where exists (select *
            from depositor
            where depositor.customer_name = borrower.customer_name)
```

Cláusula contains

- Listar todos os clientes que têm uma conta em todas as agências de Brooklyn.

```
select distinct S.customer_name
from depositor as S
where
        (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)
        contains
              (select branch_name
               from branch
               where branch_city = 'Brooklyn')
```

- ***Nota:*** Não existe no Oracle, o que não é grave pois:

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

Consulta de exemplo

- Listar todos os clientes que têm uma conta em todas as agências 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 ))
```

- ***Notas***:
  - Repare que $X - Y = \emptyset \Leftrightarrow X \subseteq Y$
  - Não se pode escrever esta consulta com combinações de = **all** ou de suas variantes.
  - Em álgebra relacional esta consulta escrever-se-ia com uma divisão:

$$\prod_{customer\_name,branch\_name}(\text{depositor} \bowtie \text{account}) \div \prod_{branch\_name}(\sigma_{branch\_city='Brooklyn'}(\text{branch}))$$

Divisão em SQL

- $r \div s = \{ t \mid t \in \prod_{R-S}(r) \wedge \forall u \in s (tu \in r) \}$
- De forma equivalente:
  - Seja $q = r \div s$
  - Então q é a maior relação satisfazendo $q \times s \subseteq r$
- Seja r(A,B) e s(B). Em SQL, $r \div s$ é obtido por:

```
select distinct X.A
from r as X
where  (select Y.B from r as Y where X.A = Y.A)
          contains
           (select B from s)
```

- Como fazer genericamente em Oracle?

Testar ausência de tuplos duplicados

- A construção **unique** verifica se o resultado de uma subconsulta possui tuplos 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')
```

- Esta construção não está disponível no Oracle

Consulta de exemplo

- Listar 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 a utilização de sub-consultas na cáusula **from**.
- Determinar o saldo médio das contas em agências cujo saldo médio é superior a $1200.

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

Repare que neste caso não foi necessário recorrer à cláusula **having**, dado que é calculada a relação temporária *result* na cláusula **from**, e assim os atributos de *result* pode ser utilizado directamente numa cláusula **where**.

Cláusula With

- A cláusula **with** permite a definição temporária de relações, cuja definição está disponível na consulta onde a clausula **with** ocorre. Análogo a procedimentos locais de uma linguagem de programação.
- Encontrar as contas de maior saldo

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

Exemplo com with

- Quais as agências cuja soma de saldos das suas contas é superior à média da soma dos saldos de todas as agências.

```
with branch_total (branch_name, value) as
   ( select branch_name, sum (balance)
   from account
   group by branch_name),
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
```

Particularidades do With em Oracle

- O Oracle 11g não permite “column aliasing” na instrução WITH. O comando do acetato anterior deve ser escrito:

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