Structured Query Language - SQL

- Tópicos:
  - Linguagem de definição de dados
  - Estrutura básica de perguntas em SQL
  - Operações com conjuntos
  - Funções de agregação
  - Junções
  - Valores nulos
  - Subconsultas
  - Vistas
  - Modificações de bases de dados
- Bibliografia:
  - Capítulo 3 e Secções 4.1, 4.2, 4.4 e 4.5 do livro recomendado

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
```

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 cláusula **from** são entendidas como cálculo de relações auxiliares.
- As subconsultas na cláusula **where** são utilizadas habitualmente para fazer testes de pertença a conjuntos, comparações entre conjuntos e calcular a cardinalidade de conjuntos.

Relações Derivadas

- O SQL permite a utilização de sub-consultas na cláusula **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 diretamente numa cláusula **where**.

Exemplo com with (revisto)

- 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),
select branch_name
from branch_total, ( select avg (value) as tAvg
                                        from branch_total )
where branch_total.value >= tAvg
```

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)
```

- Encontrar todas as agências de Lisboa ou de Almada

```
select *
from branch
where branch_city in ('Lisboa','Almada')
```

Consulta de exemplo

- Listar todos os clientes que têm conta(s) e empréstimo(s) na agência de Perryridge

```
select distinct customer_name
from borrower inner join loan using (loan_number)
where 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. (Como?) 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** é o mesmo que **in**

No entanto, ≠ **some** não é o mesmo que **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 T.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* ⇔ ∀*t* ∈ *r* : (F <comp> *t*)
  em que <comp> pode ser: <, >, ≤, ≥, =, ≠
  - Se r for o conjunto vazio então F <comp> **all** *r* devolve **true**

(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**) é o mesmo que **not in**

Contudo, (= **all**) não é o mesmo que **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 "exists"

- Encontrar todas as cadeiras leccionadas no 1º semestre de 2021/22 e no 2º semestre de 2022/23:

```
select cod_cadeira
from cadeiras as S
where semestre = 1 and ano = 2021 and
        exists  (select *
                 from cadeiras as T
                 where semestre = 2 and ano = 2022
                       and S.cod_cadeira = T.cod_cadeira);
```

- **Nome de correlação** – variável S na consulta externa
- **Subconsulta correlacionada** – a consulta no interior

Cláusula contains

- Listar todos os clientes que têm 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, mas não é grave:

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

Consulta de exemplo

- Listar todos os clientes que têm 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 $X \supseteq Y \Leftrightarrow Y - X = \emptyset$
  - 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:

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

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)
```

- Ou então (o que já funciona no Oracle):

```
select distinct X.A from r as X
where  not exists ((select B from s)
                              except
                     (select Y.B from r as Y where X.A = Y.A))
```

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
  - Também não é problemático:

unique (select As **from** ...**)**

é equivalente a

1 = **all (select count(**As) **from** ... **group by** As**)**

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')
```

- Ou então (de forma bem mais simples!):

```
select customer_name
from depositor inner join account using (account_number)
where branch_name = 'Perryridge'
group by customer_name
having count(distinct account_number) > 1
```

Subconsulta escalar

- Uma subconsulta escalar é aquela em que apenas se espera obter um único valor
- Listar todos os departamentos juntamente com o número de docentes de cada departamento

```
select dept_name,
        ( select count(*)
          from instructor
          where department.dept_name = instructor.dept_name)
        as num_instructors
from department;
```

- Ocorre um erro em tempo de execução se a subconsulta devolver mais do que um tuplo

Vistas

- Em certas circunstâncias, não é desejável que todos os utilizadores possam aceder a todo o modelo lógico (i.e. a todas as relações armazenadas na base de dados).
- Considere o caso de um empregado que necessita de saber o número de empréstimo de um cliente, mas que não precisa de saber o montante desse empréstimo. Este empregado deverá ver apenas a relação descrita por

  select *costumer_name, loan_number*
  from *borrower* **natural inner join** *loan*

Vistas

- Mesmo que não seja por razões de segurança, pode ser útil apresentar um conjunto de relações personalizada que se adapte melhor às intuições de um dado utilizador do que o modelo lógico.
- Considere o caso de um empregado do departamento de marketing que poderá preferir ver uma relação contendo os clientes que têm uma conta ou um empréstimo no banco, e as respectivas agências. Tal relação seria:

  (**select** *branch_name, customer_name*
  from *depositor* **natural inner join** *account*)

  union

  (**select** *branch_name, customer_name*
  from *borrower* **natural inner join** *loan*)

- Qualquer relação que não pertença ao modelo lógico mas que se torne visível ao utilizador como uma “relação virtual” é designada por *vista*.

Vistas

- Mecanismo que permite apresentar um modelo diferente do modelo lógico, e.g. ocultando (escondendo) informação de certos utilizadores. Para criar uma vista utilizamos o comando:

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

  em que:

  - <query expression> é qualquer expressão SQL válida
  - O nome da vista é *v*

- Em SQL do Oracle pode-se escrever

  create or replace view *v* **as** <query expression>

  para criar ou substituir uma vista já existente, evitando a utilização do comando **drop view**.

Exemplo

- Uma vista contendo todas as agências e respetivos clientes

create view *all_customer* **as**
(**select** *branch_name, customer_name*
from *depositor* **natural inner join** *account)*
union
(**select** *branch_name, customer_name*
from *borrower* **natural inner join** *loan)*

- Listar todos os clientes da agência de Perryridge

select *customer_name*
from *all_customer*
where *branch_name* = 'Perryridge'

- Uma definição de uma vista não é o mesmo que a criação duma nova relação a partir da avaliação da sua expressão. Em vez disso, a definição da vista permite guardar a expressão que depois é substituída nas consultas que utilizam a vista
  - Mas a vista pode depois ser usada como se fosse uma relação – uma relação que é recalculada em cada momento de utilização!

Definição de vistas

- Uma vista pode ser utilizada na expressão de definição de outra vista. Por exemplo:

  create view *perryridge_customer* **as**
  select *customer_name*
  from *all_customer*
  where *branch_name* = 'Perryridge'

- Uma vista v_1 *depende directamente* de uma vista v_2 se v_2 é utilizada na expressão que define v_1
- Uma vista v_1 *depende de* uma vista v_2 se v_1 depende directamente de v_2 ou se existe um caminho de dependências entre v_1 e v_2
- Uma vista *v* diz-se *recursiva* se depender dela própria.
- Em SQL não são permitidas vistas recursivas!

Expansão de vistas

- Forma de atribuir significado a vistas definidas em termos de outras vistas.
- Seja a vista v_1 definida em termos de uma expressão e_1 que pode ela própria conter vistas.
- Para expandir as vistas numa expressão repete-se sucessivamente o seguinte passo:

  repeat
  Encontrar uma vista v_i em e_1
  Substituir a vista v_i pela expressão que a define
  until não ocorram mais vistas em e_1

- Desde que as definições das vistas não sejam recursivas, este ciclo terminará sempre.

Expansão de vistas (exemplo)

- A expressão:

```
select *
from perryridge_customer
where customer_name = 'John'
```

- ... é inicialmente expandida para:

```
select *
from (select customer_name
        from all_customer
        where branch_name = 'Perryridge')
where customer_name = 'John'
```

- ... que por sua vez é expandida para:

```
select *
from (select customer_name
        from (    (select branch_name, customer_name
                   from depositor natural inner join account)
                   union
                   (select branch_name, customer_name
                   from borrower natural inner join loan))
                   where branch_name = 'Perryridge')
        where customer_name = 'John'
```

Modificação da base de Dados – Remoção

- A remoção de tuplos de uma tabela (ou vista) é feita em SQL com a instrução

  delete from *<tabela ou vista>*
  where *<Condição>*

- Apagar todas as contas da agência de Perryridge

  delete from *account*
  where *branch-name* = ‘Perryridge’

Exemplo de remoção

- Apagar todas as contas de todas as agências na cidade de Needham.

```
delete from depositor
where account-number in
                (select account-number
                  from branch natural inner join account
                  where branch-city = 'Needham')

delete from account
where branch-name in (select branch-name
                              from branch
                              where branch-city = 'Needham')
```

Exemplo de remoção

- Remover todas as contas com saldos inferiores aos da média do banco.

  delete from *account*
  where *balance* < **some** (**select avg** *(balance)*
  from *account)*

  - Problema:
    - à medida que removemos tuplos de *account,* o saldo médio altera-se
  - Solução utilizada no standard SQL:
    1. Primeiro, calcula-se o saldo médio (**avg**) e determinam-se quais os tuplos a apagar
    2. Seguidamente, removem-se todos os tuplos identificados anteriormente (sem recalcular **avg** ou testar novamento os tuplos)

Modificação da base de dados – Inserção

- A inserção de tuplos numa tabela (ou vista) é feita em SQL com a instrução

  insert into *<tabela ou vista>*
  values *<Conjunto de tuplos>*

ou

  insert into *<tabela ou vista>*
  select *...*

- Adicionar um novo tuplo a *account*

  insert into *account*
  values ('A-9732', 'Perryridge',1200)

  ou equivalentemente

  insert into *account (branch-name, balance, account-number)*
  values ('Perryridge', 1200, 'A-9732')

- Adicionar um novo tuplo a *account* em que *balance* é null

  insert into *account*
  values ('A-777','Perryridge', *null*)

Exemplo de Inserção

- Dar como bónus a todos os mutuários da agência de Perryridge, uma conta de poupança de €200. O número do empréstimo servirá de número de conta de poupança

```
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 natural inner join borrower
    where branch-name = 'Perryridge'
```

- A instrução **select-from-where** é avaliada antes da inserção de tuplos na relação (caso contrário consultas como
  insert into *table*1 **select** * **from** *table*1
  causariam problemas)

Modificação da base de dados – Actualização

- A actualização de tuplos duma tabela (ou vista) é feita em SQL com a instrução

  update *<tabela ou vista>*
  set *<Atributo> = <Expressão>,*
  <Atributo> = <Expressão>, ...
  where *<Condição>*

- Pagar juros de 1% a todas as contas da agência Perryridge.

  update *account*
  set *balance = balance* * 1.01
  where *branch_name* = 'Perryridge'

Modificação da base de dados – Actualização

- Pagar juros de 6% a todas as contas com saldos superiores a €10,000, e juros de 5% às restantes contas.
  - Escrever duas instruções de **update**:

update *account*
set *balance* = *balance* * 1.06
where *balance* > 10000

update *account*
set *balance* = *balance* * 1.05
where *balance* ≤ 10000

  - A ordem é importante!
    - Porquê?
  - Pode ser feito de forma mais “limpa” recorrendo à instrução **case**

Instrução Case para Actualizações Condicionais

- Pagar juros de 6% a todas as contas com saldos superiores a €10,000, e juros de 5% às restantes contas.

```
update account
set balance =  case
                  when balance <= 10000 then balance *1.05
                  else   balance * 1.06
               end
```

Atualizações com subconsultas escalares

- Atualizar o total de créditos para todos os estudantes

```
update alunos
set tot_cred = (select sum(creditos)
                from inscricoes inner join cadeiras using(cod_cadeira)
                 where alunos.numero = inscricoes.numero and
                       inscritos.nota <> 'R' and
                       inscritos.nota is not null );
```

- Coloca *tot_creds* a null para estudantes que não tenham realizado qualquer cadeira
- Em vez de **sum**(*credits*), escrever:

```
case
   when sum(creditos) is not null then sum(creditos)
   else 0
end
```

Atualização de uma vista

- Modificações nas bases de dados através de vistas devem ser traduzidas para modificações das verdadeiras relações presentes na base de dados.
  - E.g com uma vista com a informação sobre empréstimos, escondendo o atributo *amount*

    create view *branch-loan* **as**
    select *branch-name, loan-number*
    from *loan*

  - A adição de um novo tuplo em *branch-loan*

    insert into *branch-loan* **values** (‘Perryridge’, ‘L-307’)

    causa problemas pois terá que ser traduzido em adições de tuplos em tabelas que existam na base de dados.
  - Duas hipóteses:
    - Rejeitar a inserção e devolver uma mensagem de erro
    - Traduzir na inserção, na relação *loan,* do tuplo

      (‘L-307’, ‘Perryridge’, *null*)

Atualização de uma vista (cont.)

- Outro problema ocorre quando temos, por exemplo, a vista:

  create view *info_empréstimos* **as**
  select *customer_name, amount*
  from *borrower* **natural inner join** *loan*

e pretendemos fazer a seguinte inserção:

insert into *info_empréstimos* **values** *('Johnson',1900)*

- *A única forma seria inserir ('Johnson',null) na tabela* borrower *e (null,null,1900) na tabela* loan, não tendo o efeito desejado.

loan_number	*branch_name*	*amount*
L-11	Round Hill	900
L-14	Downtown	1500
L-15	Perryridge	1500
L-16	Perryridge	1300
L-17	Downtown	1000
L-23	Redwood	2000
L-93	Mianus	500
null	*null*	1900

loan

customer_name	*loan_number*
Adams	L-16
Curry	L-93
Hayes	L-15
Jackson	L-14
Jones	L-17
Smith	L-11
Smith	L-23
Williams	L-17
Johnson	*null*

borrower

Atualização de uma vista (cont)

- Outras não têm tradução única, como por exemplo:

  create view *all_costumers* **as**
  (select * **from** *depositor)*
  union
  *(***select** * **from** *borrower)*

- Toda a adição em *all_costumers* não tem tradução única:
  - Deve introduzir-se em *depositor* ou em *borrower*???

Atualização de vistas

- Uma view em SQL é atualizável (updatable) se todas as seguintes condições se verificam:
  - A clausula **from** só contém uma relação da base de dados;
  - A clausula **select** apenas contém nomes de atributos da relação, não contendo expressões, agregados, ou especificação de **distinct**;
  - Qualquer atributo que não aparece na clausula **select** deve poder tomar o valor null;
  - A consulta não contém nenhuma clausula **group by** nem **having**.
- A view

  create view *downtown_account* **as**
  select *acount_number,branch_name,balance*
  from *account*
  where *branch-name* = 'Downtown'

... é atualizável. No entanto, a inserção

insert into *downtown_account* **values** ('L-307','Perryridge',1000)

apesar de ser efetuada, não produziria efeitos na view.

Atualização de vistas

- Em Oracle é possível impedir as situações anteriores por intermédio da cláusula WITH CHECK OPTION na criação da vista.

  create view *downtown_account* **as**
  select *acount_number,branch_name,balance*
  from *account*
  where *branch-name* = ‘Downtown’
  with check option

- Para impedir a atualização de vistas utiliza-se a cláusula WITH READ ONLY