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
  - Vistas e relações derivadas
  - Subconsultas
  - Modificações de bases de dados
- Bibliografia:
  - Capítulo 3 e Secções 4.1 e 4.2 do livro recomendado

Subconsultas na cláusula where

- O SQL também permite subconsultas na cláusula **where**
- Estas subconsultas permitem fazer:
  - fazer testes de pertença a conjuntos resultado de consultas
  - fazer comparações entre conjuntos resultado de consultas
  - calcular a cardinalidade de conjuntos
- A tradução deste tipo de consultas para álgebra relacional não é tão direta
  - Em geral, é possível caso a caso traduzir para expressões de álgebra relacional
  - Mas não existe uma tradução direta de cada tipo destas consultas para um operador da álgebra relacional

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 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. A formulação utilizada serve apenas para ilustrar as possibilidades da linguagem SQL.

Redundância do operador in

- Grande parte as vezes não é necessário usar o operador **in**
  - E é muito mais simples, não o usar!
- 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)
```

- Pode ser feito com

```
select distinct customer_name
from borrower inner join depositor using (customer_name)
```

Redundância do operador in

- Grande parte as vezes não é necessário usar o operador **in**
  - E é muito mais simples, não o usar!
- 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)*
- Pode ser feito com

  select distinct *customer_name*
  from *borrower* **left outer join** *depositor* **using** (*customer_name*)
  where *depositor.account_number* **is null**

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* $\Leftrightarrow$ $\forall$ *t* $\in$ *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 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:

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

Divisão em SQL

- $r \div s = \{ t \mid t \in \Pi_{R-S}(r) \wedge \forall u \in s (tu \in r) \}$
- 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 Y.B from r as Y where X.A = Y.A)
                                  except
                                (select B from s))
```

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')
```

- 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 customer_name) > 1
```

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 Perryride, 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 Perryride.

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

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
```

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 11g 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

Integração de SQL com outras linguagens de programação

- Tópicos:
  - Embedded e Dynamic SQL
  - Ligações por ODBC e JDBC
  - Linguagens proprietárias
- Bibliografia:
  - Secções 5.1 e 5.2 do livro recomendado

Embedded SQL

- SQL fornece uma linguagem *declarativa* para manipulação de bases de dados. Facilita a manipulação e permite optimizações muito difíceis se fossem programadas em linguagens imperativas.
- Mas há razões para usar SQL juntamente com linguagens de programação gerais (imperativas):
  - o SQL não tem a expressividade de uma máquina de Turing (há perguntas impossíveis de codificar em SQL – e.g. fechos transitivos)
    - usando SQL juntamente com linguagens gerais é possível suprir esta deficiência
  - nem tudo nas aplicações de bases de dados é declarativo (e.g. ações de afixar resultados, interfaces, etc)
    - Essa parte pode ser programado em linguagens gerais
- O standard SQL define uma série de embeddings, para várias linguagens de programação (e.g. Pascal, PL/I, C, C++, etc).
  - À linguagem na qual se incluem comandos SQL chama-se linguagem *host*. Às estruturas SQL permitidas na linguagem *host* chama-se SQL embutido (ou *embedded* SQL)

Embedded SQL

- Permite acesso a bases e dados SQL, via outra linguagens de programação.
  - Toda a parte de acesso e manipulação da base de dados é feito através de código embutido. Todo o processamento associado é feito pelo sistema de bases de dados. A linguagem *host* recebe os resultados e manipula-os.
  - O código tem que ser pré-processado. A parte SQL é transformada em código da linguagem *host,* mais chamadas a run-time do servidor.
- A expressão EXEC SQL é usado para identificar código SQL embutido

  EXEC SQL <embedded SQL statement > END-EXEC

  Nota: Este formato varia de linguagem para linguagem. E.g. em C usa-se ';' em vez do END-EXEC.

  Em Java usa-se # SQL { …. } ;

Cursores

- Para executar um comando SQL numa linguagem *host* é necessário começar por declarar um cursor para esse comando.
- O comando pode conter variáveis da linguagem *host,* precedidas de :
- E.g. Encontrar os nome e cidades de clientes cujo saldo seja superior a *amount*

EXEC SQL

declare *c* **cursor for**
select *customer-name, customer-city*
from *account* **natural inner join** *depositor*
natural inner join *customer*
where *account.balance > :amount*

END-EXEC

Embedded SQL (Cont.)

- O comando **open** inicia a avaliação da consulta no cursor

  EXEC SQL **open** *c* END-EXEC

- O comando **fetch** coloca o valor de um tuplo em variáveis da linguagem *host*.

  EXEC SQL **fetch** *c* **into** :*cn, :cc* END-EXEC

- Chamadas sucessivas a **fetch** obtêm tuplos sucessivos
- Uma variável chamada SQLSTATE na *SQL communication area* (SQLCA) toma o valor '02000' quando não há mais dados.
- O comando **close** apaga a relação temporária, criada pelo **open**, que contem os resultados da avaliação do SQL.

  EXEC SQL **close** *c* END-EXEC

Modificações com Cursores

- Como não devolvem resultado, o tratamento de modificações dentro doutras linguagens é mais fácil.
- Basta chamar qualquer comando válido SQL de **insert, delete,** ou **update** entre EXEC SQL e END SQL

- Em geral, as váriaveis da linguagem *host* só podem ser usadas em locais onde se poderiam colocar variáveis SQL.
- Não é possível *construir* comandos (ou parte deles) manipulando strings da linguagem *host*

Dynamic SQL

- Permite construir e (mandar) executar comandos SQL, em run-time.
- E.g. (chamando dynamic SQL, dentro de um programa em C)

```
char *  sqlprog = "update account
                   set balance = balance * 1.05
                   where account-number = ?"
EXEC SQL prepare dynprog  from :sqlprog;
char account [10] = "A-101";
EXEC SQL execute dynprog using :account;
```

- A string contém um ?, que indica o local onde colocar o valor a ser passado no momento da chamada para execução.

ODBC

- Standard Open DataBase Connectivity(ODBC)
  - Standard para comunicação entre programas e servidores de bases de dados
  - application program interface (API) para
    - Abrir uma ligação a uma base de dados
    - Enviar consultas e pedidos de modificações
    - Obter os resultados
- Aplicações diversas (e.g. GUI, spreadsheets, etc) podem usar ODBC

ODBC (Cont.)

- Um sistema de bases de dados que suporte ODBC tem uma "driver library" que tem que ser ligada com o programa cliente.
- Quando o cliente faz uma chamada à API ODBC, o código da library comunica com o servidor, que por sua vez executa a chamada e devolve os resultados.
- Um programa ODBC começa por alocar um ambiente SQL, e um *connection handle*.
- Para abrir uma ligação a uma BD, usa-se SQLConnect(). Os parâmetros são:
  - connection handle,
  - servidor onde ligar
  - username,
  - password

Exemplo de código ODBC

```
int ODBCexample()
{
    RETCODE error;
    HENV    env;     /* environment */
    HDBC    conn;  /* database connection */
    SQLAllocEnv(&env);
    SQLAllocConnect(env, &conn);
    SQLConnect(conn, "aura.bell-labs.com", SQL_NTS, "avi", SQL_NTS,
      "avipasswd", SQL_NTS);
    { …. Manipulação propriamente dita … }

    SQLDisconnect(conn);
    SQLFreeConnect(conn);
    SQLFreeEnv(env);
}
```

ODBC (Cont.)

- Os programas enviam comandos SQL à base de dados usando SQLExecDirect
- Os tuplos resultado são obtidos via SQLFetch()
- SQLBindCol() liga variáveis da linguagem a atributos do resultado do SQL
  - Quando um tuplo é obtido com um *fetch*, os valores dos seus atributos são automaticamente guardados nas ditas variáveis.

Exemplo de código ODBC

```
char branchname[80];
float  balance;
int  lenOut1, lenOut2;
HSTMT   stmt;

    SQLAllocStmt(conn, &stmt);
    char * sqlquery = "select branch_name,  sum (balance)
                       from account
                       group by branch_name";

    error = SQLExecDirect(stmt, sqlquery, SQL_NTS);

    if (error == SQL_SUCCESS) {
        SQLBindCol(stmt, 1, SQL_C_CHAR,   branchname  , 80, &lenOut1);
        SQLBindCol(stmt, 2, SQL_C_FLOAT,  &balance,        0 , &lenOut2);

        while (SQLFetch(stmt) >= SQL_SUCCESS) {
            printf (" %s  %g\n", branchname, balance);
        }
    }
    SQLFreeStmt(stmt, SQL_DROP);
```

JDBC

- JDBC é uma API **Java** para comunicar com sistemas de bases de dados que suportam o SQL.
- JDBC suporta várias formas de consulta e modificação de bases de dados
- O modelo de comunicação com a base de dados:
  - Abre uma ligação
  - Cria um objecto “statement”
  - Executa comandos usando esse objecto para enviar os comandos e obter os resultados
  - Usa mecanismos de excepção para lidar com os erros

Exemplo de código JDBC

```
public static void JDBCexample(String dbid, String userid, String passwd)
{
    try {
        Class.forName ("oracle.jdbc.driver.OracleDriver");
        Connection conn = DriverManager.getConnection(
          "jdbc:oracle:thin:@aura.bell-labs.com:2000:bankdb",  userid,
          passwd);
        Statement stmt = conn.createStatement();
            … Do Actual Work ….
        stmt.close();
        conn.close();
    }
    catch (SQLException sqle) {
        System.out.println("SQLException : " + sqle);
    }
}
```

Exemplo de código JDBC (Cont.)

- Actualização

```
try {
    stmt.executeUpdate(  "insert into account values
                                ('A-9732', 'Perryridge', 1200)");
} catch (SQLException sqle) {
    System.out.println("Could not insert tuple. " + sqle);
}
```

- Execução de perguntas

```
ResultSet rset = stmt.executeQuery( "select branch_name,
    avg(balance)
                                        from account
                                        group by branch_name");
while (rset.next()) {
     System.out.println(
                rset.getString("branch_name") + "  " + rset.getFloat(2));

}
```