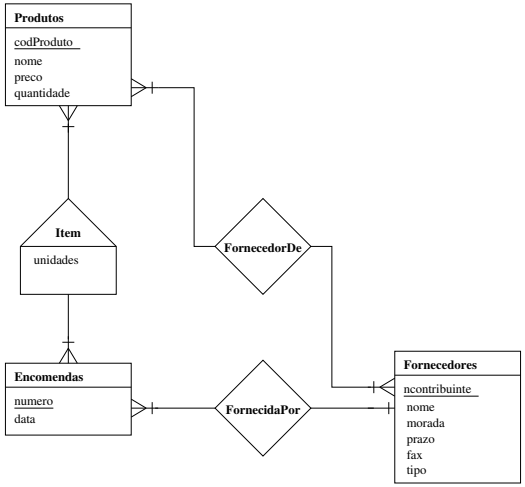
Esquema Utilizado em Exemplos



Pesquisa — 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 A1,A2,...,An
  FROM r1,r2,...,rm
 WHERE P
```

  - os A_is representam atributos;
  - os r_is representam relações;
  - P é um predicado, nos atributos dos r_is.
- A leitura desta instrução é a seguinte:
  Seleccionar (`SELECT`) os atributos $A_1, A_2, \ldots, A_n$ de entre (`FROM`) os atributos do produto cartesiano $r_1 \times r_2 \times \cdots \times r_m$, sujeitos (`WHERE`) ao predicado P.
- Corresponde à expressão da álgebra relacional:

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

A componente SELECT

- É utilizado para listar os atributos pretendidos no resultado da consulta. Por exemplo, listar o nome e o endereço dos Fornecedores:

```
mysql> SELECT nome,morada FROM Fornecedores;
+---------------+--------------------------+
| nome          | morada                   |
+---------------+--------------------------+
| Futaba        | MARYLAND                 |
| Great Planes  | Champaign, Illinois, USA |
| Tower Hobbies | Champaign, Illinois, USA |
+---------------+--------------------------+
```

- Um asterisco na cláusula `SELECT` denota “todos os atributos”, por exemplo:

```
SELECT * FROM loan
```

- NOTA: O SQL não permite o carácter ’-’ (hífen) nos identificadores, portanto deverá utilizar, por exemplo, `balcaoNome` em vez de `balcao-nome` num sistema existente.
- NOTA: As maiúsculas e minúsculas não são distinguidas em identificadores da linguagem SQL.

A componente SELECT (cont.)

- O SQL permite duplicados nas relações e nos resultados de consultas (que não chaves primárias).
- Para forçar a eliminação de duplicados, inserir a palavra-chave `DISTINCT` após o `SELECT`.
  Por exemplo: apresentar os nomes de todos os fornecedores sem repetições (o mesmo fornecedor pode fornecer vários tipos de artigos).

```
SELECT DISTINCT nome
  FROM Fornecedores
```

- A palavra-chave `ALL` (valor por omissão) indica que os duplicados não devem ser removidos.

```
SELECT ALL nome
  FROM Fornecedores
```

A componente SELECT (cont.)

- A cláusula `SELECT` pode conter expressões aritméticas envolvendo as operações, `+,-,*,/` com argumentos constantes e/ou atributos.
- Existem funções de agregação que permitem, entre outras, achar o valor mínimo contido num dado atributo (ver-se-à mais à frente).
- Dependendo das implementações, encontram-se normalmente definidas bibliotecas de funções.

Por exemplo:

```
mysql> SELECT nome,preco*1.21 FROM Produtos;
+--------------------------+-----------------+
| nome                     | preco*1.21      |
+--------------------------+-----------------+
| Tower Hobbies .46 ABC BB | 90.737897415161 |
| Tower Hobbies .61ABC BB  | 102.83789741516 |
| Futaba 9CAFS             | 417.43788818359 |
| Piper J-3 Cub 40         | 145.18789741516 |
+--------------------------+-----------------+
```

dá-nos o preço dos produtos já com o IVA aplicado.

A componente FROM

- A cláusula `FROM` corresponde à operação de produto cartesiano da álgebra relacional. Indica as relações a consultar na avaliação da expressão.

Por exemplo: listar as encomendas existentes e (produto cartesiano) os fornecedores.

```
mysql> SELECT numero,data,nome,Fornecedores.nContribuinte
         FROM Encomendas,Fornecedores;
+--------+---------------------+--------------+---------------+
| numero | data                | nome         | nContribuinte |
+--------+---------------------+--------------+---------------+
|      1 | 2007-07-04 23:42:00 | Futaba       |     123456780 |
|      2 | 2007-07-04 23:43:00 | Futaba       |     123456780 |
|      1 | 2007-07-04 23:42:00 | Great Planes |     123456781 |
|      2 | 2007-07-04 23:43:00 | Great Planes |     123456781 |
```

Ou mais útil, listar as encomendas e respectivos fornecedores

```
   mysql> SELECT numero,data,nome,Fornecedores.nContribuinte
            FROM Encomendas,Fornecedores
           WHERE Encomendas.nContribuinte=Fornecedores.nContribuinte;
+--------+---------------------+--------------+---------------+
| numero | data                | nome         | nContribuinte |
+--------+---------------------+--------------+---------------+
|      2 | 2007-07-04 23:43:00 | Futaba       |     123456780 |
|      1 | 2007-07-04 23:42:00 | Great Planes |     123456781 |
+--------+---------------------+--------------+---------------+
```


A componente WHERE

- A cláusula `WHERE` corresponde ao predicado de selecção da álgebra relacional. É formada por um predicado envolvendo atributos de relações que aparecem na cláusula `FROM`.
- Por exemplo: seleccionar todos os motores do tipo ABC de entre os produtos existentes em armazem:

```
mysql> SELECT * FROM Produtos WHERE nome LIKE '%ABC%' AND quantidade > 0;
+------------+--------------------------+-------+------------+
| codProduto | nome                     | preco | quantidade |
+------------+--------------------------+-------+------------+
|          1 | Tower Hobbies .46 ABC BB | 74.99 |         10 |
|          2 | Tower Hobbies .61ABC BB  | 84.99 |         10 |
|          3 | Tower Hobbies .75 ABC BB | 89.99 |          5 |
+------------+--------------------------+-------+------------+
```

  - Os resultados de comparações podem ser combinados por intermédio dos conectivos lógicos `AND`, `OR`, e `NOT`.
  - Os operadores relacionais habituais estão disponíveis.
  - A comparação entre sequências de caracteres pode ser feita através de expressões regulares (simples, ou complexas).

Operações com Sequências de Caracteres

- O SQL inclui um mecanismo de concordância de padrões para comparações envolvendo cadeias de caracteres. Os padrões são descritos recorrendo a dois caracteres especiais:
  - percentagem(%): O carácter % concorda com qualquer sub-cadeia.
  - sublinhado (_): O carácter _ concorda com qualquer carácter.
- O SQL suporta uma variedade de operações com cadeias de caracteres, tais como:
  - concatenação (utilizando "||");
  - conversão de maiúsculas para minúsculas (e vice-versa);
  - calcular o comprimento, extracção de subcadeias, etc.
- O MySQL suporta (versão 5) expressões regulares como mecanismo de comparação de padrões.

Expressões Regulares

Uma expressão regular sobre um alfabeto finito V é definida de modo indutivo como se segue:

- ϵ (sequência vazia) é uma expressão regular;
- a é uma expressão regular, para todo o $a \in V$;
- se R é uma expressão regular sobre V, então também o é $(R)^*$ (iteração);
- se Q e R são expressões regulares sobre V, então também $(Q)(R)$ e $(Q)|(R)$ (concatenação e união respectivamente) o são.

Expressões Regulares

O MySQL usa (versão 5) expressões regulares como mecanismo de comparação de padrões.

```
SELECT <seq_caracteres> REGEXP <expressão_regular>;
```

- `^` — associa com o inicio da seq. de caracteres.
- `$` — associa com o fim da seq. de caracteres.
- `.` — associa com um qualquer carácter, inclusive a mudança de linha.
- `a*` — associa com zero ou mais 'a'
- `a+` — associa com um ou mais 'a'
- `a?` — associa com zero ou um 'a'
- `de|abc` — associa com 'de' ou com 'abc'
- `[a-dX], [^a-dX]` — associa (ou não) com os caracteres 'a' a 'd' e 'X'.

Expressões Regulares — Exemplos

- Para achar nomes que começam por 'b'

```
mysql> SELECT * FROM pet WHERE name REGEXP '^b';
```

- Para achar nomes contendo um 'w'

```
mysql> SELECT * FROM pet WHERE name REGEXP 'w';
```

- Para achar nomes contendo 'Carlos Seixas'

```
mysql> SELECT * FROM endereços WHERE rua REGEXP 'Carlos.+Seixas';
```

No MySQL, os padrões SQL são, por omissão, insensíveis ao tipo do carácter (maiúscula ou minúscula). Se se pretende que a associação seja feita tendo em conta o tipo do carácter, então é necessário usar a palavra chave "Binary".

- Para achar nomes começando por um 'b' (e não por um 'B')

```
mysql> SELECT * FROM pet WHERE name REGEXP BINARY '^b';
```


Ordenando os tuplos

Listar em ordem alfabética os nomes de todos os fornecedores de motores do tipo "ABC"

```
  SELECT Fornecedores.nome
    FROM Fornecedores,Produtos
   WHERE Produtos.nome LIKE '%ABC%'
ORDER BY Fornecedores.nome
```

- Pode-se especificar "DESC" para ordenação descente ou "ASC" para ordenação ascendente, para cada atributo; por omissão, assume-se ordem ascendente.

```
ORDER BY Fornecedores.nome DESC
```

- Pode-se ter mais do que uma chave de ordenação, separando-as com vírgulas.

```
SELECT * FROM Produtos ORDER BY quantidade,preco
```

Funções de Agregação

Estas funções aplicam-se a multi-conjuntos 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

Por exemplo:

- Determinar o preço médio dos produtos:
  `SELECT AVG(preco) FROM Produtos`
- Calcular o número de produtos:
  `SELECT COUNT(*) FROM Produtos`
- Encontrar o preço máximo dos produtos:
  `SELECT MAX(preco) FROM Produtos`

Funções de agregação - GROUP BY

Podemos agrupar os atributos que pretendemos agregar em diferentes grupos, através do comando GROUP BY.

Listar o número de encomendas por fornecedor:

```
  SELECT nome,count(*)
    FROM Encomendas,Fornecedores
   WHERE Encomendas.nContribuinte=Fornecedores.nContribuinte
GROUP BY nome
```

- Os atributos na cláusula SELECT fora de funções de agregação, têm de aparecer na lista GROUP BY.
- Se aparecer mais do que um atributo em GROUP BY, então cada grupo é formado pelos tuplo com valores iguais em todos esses atributos.

Funções de Agregação - HAVING

Podemos limitar os elementos que vão ser agregados através da inclusão de um predicado.

Listar os nomes de todas os fornecedores cujo valor médio dos preços dos seus produtos é superior a 100€.

```
  SELECT Fornecedores.nome,AVG(preco)
    FROM FornecedorDe,Produtos,Fornecedores
   WHERE FornecedorDe.nContribuinte=Fornecedores.nContribuinte
     AND FornecedorDe.codProduto=Produtos.codProduto
GROUP BY Fornecedores.nome
  HAVING AVG(preco)>100
```

- Os predicados no comando HAVING são aplicados depois da formação dos grupos, enquanto que os predicados no comando WHERE são aplicados antes da formação dos grupos.

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.
Por exemplo: mostrar todos os fornecedores que possuem FAX.

```
SELECT nome
  FROM Fornecedores
 WHERE NOT(fax IS NULL)
```

- O resultado da aplicação de uma operação aritmética ao valor NULL é NULL. Por exemplo: 5 + NULL é igual a NULL.
- Qualquer comparação com NULL retorna UNKNOWN.
- As funções de agregação, com excepção de COUNT(*), ignoram os nulos.

Valores Nulos e Lógica Trivalente

Qualquer comparação com NULL retorna UNKNOWN.
Lógica trivalente usando o valor lógico UNKNOWN.

OR	TRUE	FALSE	UNKNOWN
UNKNOWN	TRUE	UNKNOWN	UNKNOWN

AND	TRUE	FALSE	UNKNOWN
UNKNOWN	UNKNOWN	FALSE	UNKNOWN

NOT	UNKNOWN
UNKNOWN	UNKNOWN

- O predicado IS UNKNOWN pode ser usado para testar se o valor de um dado predicado *P* é, ou não, definido. P IS UNKNOWN é verdadeiro se o valor de *P* não está definido (é UNKNOWN).
- O resultado da condição do comando WHERE é tratado como falso quando o seu valor é UNKNOWN.

Valores Nulos e Agregados

- Todas as funções de agregação excepto COUNT(*) ignoram tuplos com valores nulos nos atributos agregados.
  Por exemplo: calcule a média do prazo de entrega dos fornecedores.

```
SELECT AVG(prazo)
  FROM Fornecedores
```

  - os valores NULL serão ignorados no cálculo da média.
  - o resultado é NULL se não existir nenhum montante não-nulo
- A função de agregação COUNT também ignora os valores nulos, por exemplo:

```
SELECT COUNT(prazo)
  FROM Fornecedores
```

  não irá contabilizar os valores nulos.
- No entanto COUNT(*) irá contabilizar todos os tuplos, mesmo aqueles com todos os atributos a NULL.

Operações de Junção

- As operações de junção retornam uma relação como resultado da combinação de duas outras relações.
- Estas operações adicionais são utilizadas habitualmente em consultas na componente FROM.
- Tipo de junção: define como tratar os tuplos que não estão relacionados entre si (basedados na condição de junção).
- Condição de junção: define quais os tuplos que são combinados nas duas relações, assim como quais os atributos que aparecem no resultado da junção.

Tipos de Junção
inner join
left outer join
right outer join
full outer join

Condições de Junção
natural
on <predicado>
using $(A_1, A_2, \ldots, A_n)$

Junções — Relações de Exemplo

- Relação Emprestimo

numero	balcaoNome	quantia
L-170	Central	3000
L-230	Pólo I	4000
L-260	Pólo II	1700

- Relação ClienteEmp

nome	numero
João	L-170
Paulo	L-230
Raquel	L-155

Junção Interna (ou interior)

- As junções internas devem ser usadas conjuntamente com uma expressão condicional através dos comandos `ON` e `USING`.
- A junção interna (`INNER JOIN`) e um produto cartesiano das relações envolvidas considerando somente os tuplos que verificam a expressão condicional especificada.
- Exemplos:

```
SELECT * FROM table1, table2 WHERE table1.id=table2.id
SELECT * FROM table1 INNER JOIN table2 ON table1.id=table2.id
```

- O condicional `ON` é uma qualquer expressão condicional tais como as que são usadas na componente `WHERE`. Deve-se usar a componente `ON` para especificar a forma de junção das tabelas, enquanto a componente `WHERE` deve ser usada para restringir os tuplos que vão ser seleccionados.
- Numa junção `LEFT JOIN`, se não existe na tabela da direita uma coluna que associe através da condição `ON`, ou `USING` então todos os valores correspondentes no resultado da consulta (referente à tabela da direita) serão preenchidos com o valor `NULL`.

Junções — Exemplos I

-
```
SELECT *
  FROM Emprestimo
INNER JOIN ClienteEmp
   ON Emprestimo.numero = ClienteEmp.numero
```

numero	balcaoNome	quantia	nome	numero
L-170	Central	3000	João	L-170
L-230	Pólo I	4000	Paulo	L-230

-
```
SELECT *
  FROM Emprestimo
LEFT OUTER JOIN ClienteEmp
   ON Emprestimo.numero = ClienteEmp.numero
```

numero	balcaoNome	quantia	nome	numero
L-170	Central	3000	João	L-170
L-230	Pólo I	4000	Paulo	L-230
L-260	Pólo II	1700	NULL	NULL

Operação de Junção Natural

- Sejam r e s relações nos esquemas R e S respectivamente. O resultado é uma relação no esquema $R \bigcup S$ que é obtido considerando cada par de tuplos t_r de r e t_s de s.
- Se t_r e t_s têm o mesmo valor em cada um dos atributos em $R \bigcap S$, um tuplo t é adicionado ao resultado, em que:
  - t tem os mesmos valores que t_r em r.
  - t tem os mesmos valores que t_s em s
- Exemplo:

```
R = (A, B, C, D)
S = (E, B, D)
Esquema resultado: (A, B, C, D, E)
```

em que as colunas comuns B e D foram usadas para seleccionar os tuplos a incluir no resultado.

Junções — Exemplos II

-
```
SELECT *
  FROM Emprestimo
NATURAL INNER JOIN ClienteEmp
```

numero	balcaoNome	quantia	nome
L-170	Central	3000	João
L-230	Pólo I	4000	Paulo

- De forma equivalente podiamos escrever

```
SELECT *
  FROM Emprestimo
INNER JOIN ClienteEmp
   USING (numero)
```

neste caso explicita-se o atributo sobre o qual recaí a junção.

A junção natural (NATURAL [LEFT] JOIN) de duas tabelas é semanticamente equivalente a uma junção interna (INNER JOIN ou LEFT JOIN) com uma claúsula USING que considera todas as colunas comuns a ambas as tabelas.

Junção Externa (ou exterior)

- Uma extensão da operação de junção que evita a perda de informação.
- Calcula a junção e depois adiciona ao resultado os tuplos de uma relação que não estão relacionados com a outra relação na junção.
- Utiliza valores nulos :
  - NULL significa que o valor é desconhecido ou que não existe.
  - Simplificadamente, todas as comparações com `NULL` são falsas por definição.

Junções — Exemplos III

- ```
  SELECT *
    FROM Emprestimo
  NATURAL RIGHT OUTER JOIN ClienteEmp
  ```

numero	balcaoNome	quantia	nome
L-170	Central	3000	João
L-230	Pólo I	4000	Paulo
L-155	NULL	NULL	Raquel

- ```
  SELECT *
    FROM Emprestimo
  FULL OUTER JOIN  ClienteEmp
    USING (numero)
  ```

numero	balcaoNome	quantia	nome
L-170	Central	3000	João
L-230	Pólo I	4000	Paulo
L-260	Pólo II	1700	NULL
L-155	NULL	NULL	Raquel

Sub-consultas Embebidas

A linguagem SQL permite a inclusão de sub-consultas na cláusula `FROM` de uma consulta.

Por exemplo: achar a média dos balanços dos balcões cuja média dos balanços é maior do que 12000€.

```
SELECT balcaoNome, mediaBalanco
  FROM (SELECT balcaoNome, avg(balanco)
          FROM Conta
      GROUP BY balcaoNome) AS mediaBalcao (balcaoNome,mediaBalanco)
 WHERE mediaBalanco > 12000
```

Deixa de ser necessário a cláusula `HAVING` dado que se calcula a sub-consulta (vista temporária) e os atributos da mesma podem ser usados na consulta principal.

Relações Derivadas

A cláusula `WITH` permite a definição de uma sub-consulta (vista temporária) cujo alcance é somente o da consulta aonde a mesma ocorre.

Exemplo: achar todas as contas cujo balanço seja igual ao valor máximo achado.

```
    WITH BalancoMax(valor)
      AS SELECT max(balanco)
           FROM Conta
  SELECT numero
    FROM Conta, BalancoMax
   WHERE Conta.balanco = BalancoMax.valor
```

A cláusula `WITH` é o correspondente à instrução de atribuição na álgebra relacional. No entanto é de notar que a mesma não pode ser usada sem estar associada a uma consulta.

Relações Derivadas

Outro exemplo: Achar todos os balcões aonde o valor total dos depósitos é maior do que a média dos valores dos depósitos em todos os balcões.

```
WITH TotalBalcao(balcaoNome, valor)
  AS SELECT balcaoNome, sum(balanco)
       FROM Conta
    GROUP BY balcaoNome
WITH MediaTotalBalcao(valor)
  AS SELECT avg(valor)
       FROM TotalBalcao
SELECT balcaoNome
  FROM TotalBalcao, MediaTotalBalcao
 WHERE TotalBalcao.valor > MediaTotalBalcao.valor
```

Notar que após a definição da relação derivada a mesma pode ser usada de imediato.

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 devera ver apenas a relação

```
(SELECT nome, numero
   FROM ClienteEmp, Emprestimo
  WHERE ClienteEmp.numero = Emprestimo.numero)
```

- Uma **vista** providencia um mecanismo de sonegação de informação.
- Qualquer relação que não pertença ao modelo conceptual mas que se torne visível ao utilizador como uma «relação virtual» é designada por **vista**.

Definição de Vistas

- Uma vista é definida por intermédio da instrução **CREATE VIEW**

```
CREATE VIEW NomeVista AS <consulta>
```

  Em que ⟨consulta⟩ é uma consulta SQL qualquer. O nome da vista é `NomeVista`.

- Após a definição de uma vista, o seu nome pode ser utilizado para se referir à relação virtual gerada pela vista.
- Uma definição de uma vista não é o mesmo que criar uma 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.

Exemplos de Vistas

- Considere-se a vista (com o nome `todosOsClientes`) contento os nomes das agências e seus clientes.

```
CREATE VIEW TodosOsClientes
         AS (SELECT balcaoNome, nome
               FROM ClienteEmp, Conta
              WHERE ClienteEmp.numero = Conta.numero)
             UNION
            (SELECT balcaoNome, nome
               FROM ClienteEmp, Emprestimo
              WHERE ClienteEmp.numero = Emprestimo.numero)
```

- Pode-se assim encontrar todos os clientes da agência de `Penacova` através de:

```
SELECT nome
  FROM TodosOsClientes
 WHERE balcaoNome = 'Penacova'
```

Vistas definidas a partir de outras vistas

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

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:

```
Repetir
  Encontrar uma vista v_i em e_1
  Substituir a vista v_i pela expressão que a define
Até que não ocorram mais vistas em e_1
```


Modificação da base de dados - Inserção

A inserção de tuplos numa tabela é feita em SQL com a instrução `INSERT`.

```
INSERT INTO <tabela> [(<lista_de_atributos>)]
     VALUES (<lista_de_expressões>)
```

No caso de não se especificar a lista_de_atributos subentende-se que são todos os atributos.
Exemplos:

- Adicionar um novo tuplo a Produtos

```
INSERT INTO Produtos
     VALUES (8,'F4U Corsair',403.57,5)
```

ou equivalentemente

```
INSERT INTO Produtos (nome,preco,quantidade)
     VALUES ('F4U Corsair',403.57,5)
```

Para todos os atributos da tabela não especificados são usados os valores por omissão (no caso do atributo `codProduto` é auto-incremental).

Modificação da base de dados - Inserção (cont.)

É também possível usar uma sintaxe alternativa.

```
INSERT INTO <tabela>
  SET <atributo1> = <expressão1>,
      <atributo2> = <expressão2>,
      ...
      <atributoN> = <expressãoN>
```

Exemplo:

```
INSERT INTO Produtos
       SET nome='P-47D Thunderbolt',
           preco=403.57,
           quantidade=3
```

Modificação da base de dados - Inserção (cont.)

Finalmente, é possível copiar valores entre tabelas através da instrução `INSERT ... SELECT`
Exemplo: dar como bónus a todos os clientes com empréstimos na agência Central de Coimbra da CGD, uma conta poupança de 2000€. O número do empréstimo servirá de número de conta poupança.

```
INSERT INTO Conta
     SELECT numero, 2000, balcaoNome
       FROM Emprestimo
      WHERE balcaoNome = 'Coimbra-central'
```

A instrução `SELECT ... FROM ...WHERE` é avaliada previamente à inserção de tuplos na relação (caso contrário consultas como `INSERT INTO tabela1 SELECT * FROM tabela1` causariam problemas)

Modificação da base de dados - Actualização

A actualização de tuplos duma tabela é feita em SQL com a instrução `UPDATE`

```
UPDATE <tabela>
   SET <atributo1> = <expressão1>,
       ...
       <atributoN> = <expressãoN>
 [ WHERE <condição> ]
```

Exemplos:

- Actualizar a quantidade em armazém de um produto após a recepção de uma encomenda:

  ```
  UPDATE Produtos
     SET quantidade = quantidade+2
   WHERE codProduto=6;
  ```

- Actualizar os preços de acordo com uma taxa de inflação de 2%

  ```
  UPDATE Produtos
     SET preco=preco*1.02;
  ```


Modificação da base de Dados - Remoção

A remoção de tuplos de uma tabela é feita em SQL com a instrução `DELETE`.

```
DELETE
  FROM <tabela>
[ WHERE <condição> ]
```

Por exemplo: retirar da tabela Produtos todos os produtos dos quais não haja nenhum exemplar em armazém

```
DELETE
  FROM Produtos
 WHERE quantidade=0
```

Nota: esta instrução vai falhar (na actual implementação da base de dados) à conta de uma restrição de integridade (FornecedorDe).

Mudanças Através de Vistas

É possível usar uma **Vista** para actualizar uma tabela, utilizando as instruções `UPDATE`, `DELETE`, ou `INSERT`.

Para que seja possível actualizar uma tabela com uma **Vista** é necessário que haja uma relação injectiva (um-para-um) entre as linhas da **Vista** e as linhas da tabela que ela referência.

Além das restrições eventualmente impostas pelas permissões de acesso às tabelas subjacentes existem outras restrições que podem impedir a actualização de valores através de **Vistas**.

Mudanças Através de Vistas (cont.)

Para poder actualizar as tabelas subjacentes a **Vista** não pode conter:

- funções de agregação;
- `DISTINCT`; `GROUP BY`; `HAVING`; `UNION`;
- uma sub-consulta na lista de selecção;
- alguns tipos de junções;
- uma **Vista** não actualizável na cláusula `FROM`;
- múltiplas referências a uma dada coluna na tabela base.

Quanto à inserção de valores verifica-se ainda que:

- não pode haver duplicação nos nomes das colunas;
- a **Vista** tem de conter todas as colunas de preenchimento obrigatório e em que não esteja definido um valor por omissão;
- as colunas da **Vista** não pode ser uma coluna derivada de outras colunas.

Operação de União em SQL

Relações R_1 e R_2:

$R_1 =$

col1	col2
1	a
2	b
3	c
4	d

$R_2 =$

col1	col2
1	a
3	c
5	e

```
SELECT * from R1
UNION
SELECT * from R2
```

$R_1 \bigcup R_2 =$

col1	col2
1	a
2	b
3	c
4	d
5	e

Operação de Diferença em SQL

O MySQL não suporta directamente a operação de diferença.

$R_1 - R_2 =$

col1	col2
2	b
4	d

- ```
  SELECT * FROM R1
   WHERE (col1,col2)
          NOT IN (SELECT *
                    FROM R2);
  ```
- ```
  SELECT * FROM R1
   WHERE NOT EXISTS (SELECT * FROM R2
                       WHERE R1.col1=R2.col1
                         AND R1.col2=R2.col2);
  ```
- O modo mais eficiente:

```
SELECT DISTINCT *
  FROM R1
LEFT OUTER JOIN R2 USING (col1,col2)
 WHERE R2.col1 IS NULL
```


Operação de Intersecção em SQL

O MySQL não suporta directamente a operação de intersecção.

$R_1 \cap R_2 =$

col1	col2
1	a
3	c

- ```
  SELECT * FROM R1
   WHERE (col1,col2)
          IN (SELECT *
                FROM R2);
  ```
- ```
  SELECT * FROM R1
   WHERE EXISTS (SELECT * FROM R2
                   WHERE R1.col1=R2.col1
                     AND R1.col2=R2.col2);
  ```
- O modo mais eficiente:

```
SELECT DISTINCT *
  FROM R1
INNER JOIN R2 USING (col1,col2)
```

Operação de Divisão em SQL

- Propriedade
  - Seja $q = r \div s$
  - Então q é a maior relação satisfazendo $q \times s \subseteq r$.
- Definição em termos de operações básicas da álgebra rel. Sejam $r(R)$ e $s(S)$ relações, com $S \subset R$

$$r \div s = \Pi_{R-S}(r) - \Pi_{R-S}((\Pi_{R-S}(r) \times s) - \Pi_{R-S,S}(r))$$

Porquê?
  - $\Pi_{R-S}(r) \times s$ dá os elementos de r com todos os valores de S.
  - $\Pi_{R-S,S}(r)$ construi uma versão de r com os atributos da expressão anterior.
  - $\Pi_{R-S}(\Pi_{R-S}(r) \times s) - \Pi_{R-S,S}(r))$ dá os tuplos t em $\Pi_{R-S}(r)$ tal que para algum tuplo $u \in s$, $tu \notin r$.

Operação de Divisão em SQL

Adequada para consultas que incluam a frase ”para todo”.

$$\begin{aligned} r \div s &= \Pi_{R-S}(r) - \Pi_{R-S}((\Pi_{R-S}(r) \times s) - \Pi_{R-S,S}(r)) \\ &= \Pi_{(A,B,C)}(r) - \Pi_{(A,B,C)}((\Pi_{(A,B,C)}(r) \times s) - \Pi_{(A,B,C,D,E)}(r)) \end{aligned}$$

$$R = (A, B, C, D, E) \quad S = (D, E) \quad R - S = (A, B, C)$$

- $\Pi_{(A,B,C)}(r) \times s$ dá os elementos de r com todos os valores de (D, E).
- $\Pi_{(A,B,C,D,E)}(r)$ construi uma versão de r com os atributos da expressão anterior. Neste caso: $\Pi_{(A,B,C,D,E)}(r) = r$.
- $\Pi_{(A,B,C)}((\Pi_{(A,B,C)}(r) \times s) - \Pi_{(A,B,C,D,E)}(r))$ dá os tuplos t em $\Pi_{(A,B,C)}(r)$ tal que para algum tuplo $u \in s$, $tu \notin r$.

MySQL — Exemplos de Consultas

Uma aproximação às consultas do tipo ÷ (do manual do MySQL)

- Que tipos de lojas estão presentes em uma, ou mais, cidades?

```
SELECT DISTINCT tipos_lojas FROM Lojas
WHERE EXISTS
  (SELECT * FROM cidades_lojas
    WHERE cidades_lojas.tipo_loja = lojas.tipos_lojas)
  AS Aux;
```

- Que tipos de lojas não estão presentes em nenhuma cidade?

```
SELECT DISTINCT tipo_loja FROM lojas
WHERE NOT EXISTS
  (SELECT * FROM cidades_lojas
    WHERE cidades_lojas.tipo_lojas = lojas.tipo_lojas)
  AS Aux;
```


MySQL — Exemplos de Consultas

- Que tipos de lojas estão presentes em todas as cidades?

```
SELECT DISTINCT store_type FROM stores s1
 WHERE NOT EXISTS
  (SELECT * FROM cities
    WHERE NOT EXISTS
    (SELECT * FROM cities_stores
      WHERE cities_stores.city = cities.city
       AND cities_stores.store_type = stores.store_type)
    AS Aux1)
  AS Aux2;
```

Este último exame é uma consulta “NOT EXISTS” duplamente aninhada. Isto é, contém uma cláusula “NOT EXISTS” dentro de uma outra cláusula do mesmo tipo.
Formalmente, responde à questão “não existe uma cidade com uma loja que não está em Lojas?”. É mais fácil de ver que uma consulta “NOT EXISTS” duplamente aninhada responde à questão “é x verdade para todo o y?”

SQL Embutido

- 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 (por exemplo, fechos transitivos); usando SQL juntamente com linguagens gerais é possível suprir esta deficiência;
  - nem tudo nas aplicações de bases de dados é declarativo (por exemplo, acções de afixar resultados, interfaces, etc). Essa parte pode ser programado em linguagens genéricas.
- O standard SQL define uma série de "encaixes", para várias linguagens de programação (e.g. Pascal, PHP, C, C++, Cobol, etc). À linguagem na qual se incluem comandos SQL chama-se linguagem hospedeira. Às estruturas SQL permitidas na linguagem hospedeira chama-se SQL embutido.

SQL Embutido

- 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 hospedeira recebe os resultados e manipula-os.
- O código tem que ser pré-processado. A parte SQL é transformada em código da linguagem hospedeira, mais chamadas a "run-time" do servidor.
- A forma particular como é feita a ligação entre a linguagem hospedeira e a linguagem SQL varia de linguagem para linguagem.

MySQL & PHP

Este exemplo simples mostra como, na linguagem PHP, fazer a ligação, executar uma consulta, mostrar as linhas do resultado e desligar do banco de dados MySQL.

Exemplo 1. Exemplo de visão geral da extensão MySQL

```
<?php
// Ligação e escolha da base de dados
$ligacao = mysql_connect('mysql_host','mysql_user','mysql_password')
    or die('Não foi possível ligar: '.mysql_error());

echo 'Ligação bem sucedida';

mysql_select_db('my_database') or die('Não foi possível selecionar
a base de dados');
?>
```

A ligação entre o PHP e o MySQL é feita através de funções PHP próprias que estão definidas no módulo (extensão) php-mysql:

MySQL & PHP

- `mysql_connect` — Abre ou reutiliza uma conexão com um servidor MySQL.

  `resource mysql_connect([string server[, string username[, string password[, bool new_link[, int client_flags]]]]])`

  Retorna um identificador de ligação MySQL em caso de sucesso, ou "FALSE" em caso de insucesso.
- `mysql_select_db` — Selecciona uma base de dados MySQL.

  `bool mysql_select_db (string database_name[, resource link_identifier] )`

  Devolve "TRUE" em caso de sucesso ou "FALSE" em caso de insucesso.

```
<?php
$ligacao = mysql_connect('localhost','mysql_user','mysql_password');
if (!$ligacao) {
     die('Não foi possível fazer a ligação: '.mysql_error());
}

echo 'Ligação bem sucedida';
mysql_close($ligacao);
?>
```

MySQL & PHP

```
// Executando a consulta SQL
$consulta = 'SELECT *
               FROM my_table';
$resultado = mysql_query($consulta) or die('A consulta falhou!:'.mysql_error());

// Exibindo os resultados em HTML
echo "<table>";
while ($line = mysql_fetch_array($resultado, MYSQL_ASSOC)) {
     echo "<tr>";
     foreach ($line as $col_value) {
           echo "<td> $col_value</td>";
     }
    echo "</tr>";
}
echo "</table>";

// Libertar o conjunto de resultados
mysql_free_result($resultado);

// Fechar a ligação
mysql_close($ligacao);
?>
```


MySQL & PHP

- `mysql_query` — Realiza uma consulta MySQL

```
resource mysql_query(string query[,resource link_identifier])
```

`mysql_query()` envia uma consulta para a base de dados activa no servidor da ligação presente `link_identifier`.
Nota: A string da query **não** deve terminar com ponto e virgula(;).

Para os comandos SELECT, SHOW, EXPLAIN ou DESCRIBE o valor de retorno é identificador de recurso ou "FALSE" se a consulta não foi executada correctamente. Para outros tipos de comandos SQL, o valor de retorno é "TRUE" em caso de sucesso e "FALSE" em erro.
O identificador de recurso pode depois ser usado em funções que lidam com os resultados das consultas (tabelas).

Com `mysql_num_rows()` e com `mysql_affected_rows()` podem-se obter quantas linhas foram devolvidas, ou quantas linhas foram afectadas, respectivamente.

`mysql_free_result()` liberta (explicitamente) os recursos.

MySQL & PHP

- `mysql_fetch_array` — Obtém uma linha como uma matriz associativa, uma matriz numérica, ou ambas.

```
array mysql_fetch_array (resource resultado [, int result_type] )
```

Retorna uma matriz que corresponde a linha obtida **e move o ponteiro interno dos dados adiante**.

Parâmetros: resultado (o resultado obtido pela consulta)
O tipo de vector ("array") que é obtido é uma constante e pode ter os seguintes valores: `MYSQL_ASSOC`, `MYSQL_NUM`, e o valor por omissão de `MYSQL_BOTH`.

Valores de retorno: uma vector que corresponde à linha obtida, ou FALSE se não houver mais linhas. O tipo do vector devolvido depende de como `result_type` esta definido.

Usando MYSQL_BOTH (por omissão), você terá um vector com ambos os índices, numérico e associativo. Usando `MYSQL_ASSOC`, você tem apenas os índices associativos, usando `MYSQL_NUM`, você tem apenas os índices numéricos.

ODBC

A norma ODBC (Open DataBase Connectivity) define um modo de um programa, concebido e compilado por um dado programador, comunicar com um servidor de bases de dados.
A norma define um interface entre programas em C e os SGBD (API, Application Program Interface) capaz de:

- estabelecer a ligação a uma base de dados;
- enviar consultas (de todos os tipos);
- obter os resultados.

Qualquer tipo de aplicação pode usar este tipo de interface para ter acesso a um SGBD que suporte a norma ODBC.
Each database system supporting ODBC provides a library that must be linked with the client program. When the client program makes an ODBC API call, the code in the library communicates with the server to carry out the requested action, and fetch results.