Bases de Dados

MIEIC | BDAD 2017/2018 | José Devezas

Conteúdos de apoio

Os conteúdos aqui apresentados servem informalmente de suporte aos conteúdos lecionados nas aulas teóricas, tentando-se uma abordagem mais coloquial e menos formal (onde possível), de forma a fomentar a aprendizagem durante as aulas teórico-práticas (TP).

O que precisam de saber?

- Modelo Conceptual
- Esquema Relacional
- Modelo Relacional
  - Dependências funcionais
  - Formas normais
    - 1ª Forma Normal (1NF)
    - 2ª Forma Normal (2NF)
    - 3ª Forma Normal (3NF)
    - Forma Normal Boyce-Codd
  - Forma minimal
  - Fecho de um conjunto de atributos
  - Teste da caça
  - Chaves
    - Super-chave
    - Chave-candidata
    - Chave primária
    - Chave alternativa
- Linguagem de Definição de Dados (SQL)
- Álgebra Relacional
- Linguagem de Manipulação de Dados (SQL)
- NoSQL (MongoDB)

Projeto

Datas Importantes (2017/2018) 💡

- 25 Fevereiro => Definição de grupo e submissão do tema
- 11 Março => Entrega 1
- 15 Abril => Entrega 2
- 27 Maio => Entrega 3

Visão Geral

- Grupos de 3 alunos
- Devem propor tema ao docente das teórico-práticas via Moodle, enviando uma descrição detalhada (máx. 100 palavras), similar aos enunciados das aulas TP.
- Trabalho similar ao dos guiões, a entregar em 3 fases:
  - Entrega 1:
    - A. Definição do Modelo Conceptual
  - Entrega 2:
    - B. Definição do Esquema Relacional
    - C. Análise de Dependências Funcionais e Formas Normais
    - D. Criação da base de dados em SQLite
    - E. Adição de restrições à base de dados
    - F. Carregamento de dados
  - Entrega 3:
    - G. Interrogação da Base de dados
    - H. Adição de gatilhos à base de dados

Consulta Recomendada

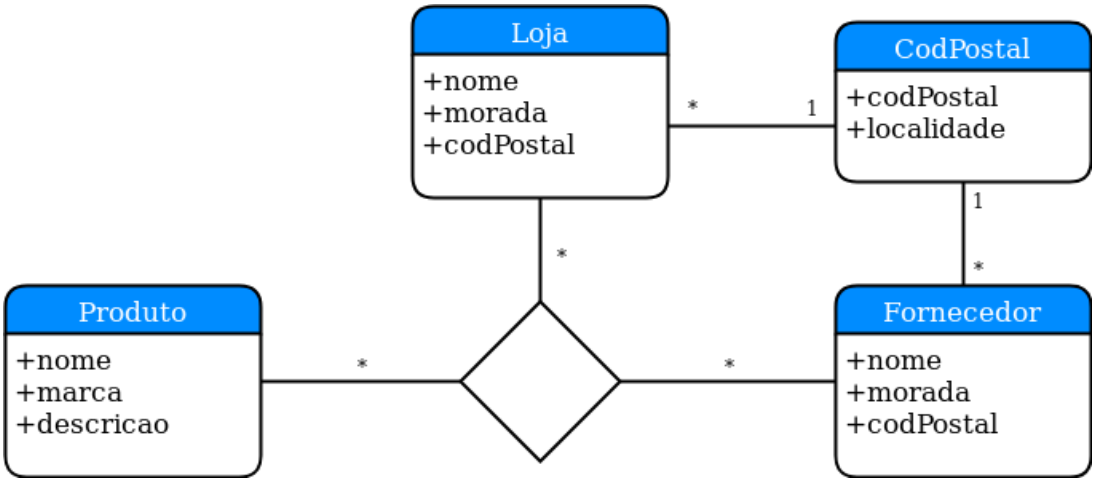
> *Hector Garcia-Molina, Jeffrey D. Ullman, Jennifer Widom:*
> *Database Systems - The Complete Book (2nd Edition). Pearson Education 2009, ISBN 978-0-13-187325-4, pp. I-XXVI, 1-1203*

Aula 1

Modelo Conceptual

Esta é uma representação mais próxima dos requisitos do negócio e mais desligada da implementação e das necessidades de desenho do esquema da base de dados.

Exemplo de um Modelo Conceptual simples para gestão de produtos e fornecedores de uma loja:



Esquema Relacional

Chaves primárias identificadas com sublinhado e chaves estrangeiras identificadas com seta para relação (i.e., tabela) respetiva:

```
CodPostal(codPostal, localidade)
Loja(idLoja, nome, morada, codPostal -> CodPostal)
Produto(idProduto, nome, marca, descricao)
Fornecedor(idFornecedor, nome, morada, codPostal -> CodPostal)
LojaProdutoFornecedor(idLoja -> Loja, idProduto -> Produto, id
Fornecedor -> Fornecedor)
```

Aula 2

Modelo Conceptual vs Modelo Relacional

Diferentes propósitos:

https://www.visual-paradigm.com/support/documents/vpuserguide/3563/3564/85378_conceptual,l.html

- **Modelo Conceptual:** mais próximo dos requisitos do negócio.
- **Modelo Relacional:** modelo físico, traduzido do modelo conceptual, de forma a representar diretamente a estrutura da base de dados.

Modelo Relacional

https://en.wikipedia.org/wiki/Relational_model

Esta é uma representação utilizada para o desenho do esquema da base de dados e é geralmente obtida com o suporte do Modelo Conceptual.

Dependências funcionais

A dependência funcional é funcional no sentido em que um determinado conjunto de atributos $\{A_1, A_2, A_3\}$ implica outro conjunto de atributos $\{B_1, B_2\}$, tal que determinada combinação de valores em $\{A_1, A_2, A_3\}$ apenas gera uma combinação de valores em $\{B_1, B_2\}$. Vários registos podem ser recuperados com base em $\{A_1, A_2, A_3\}$, mas os valores de B_1 e B_2 são sempre iguais, se a dependência funcional $A_1, A_2, A_3 \rightarrow B_1, B_2$ se verificar.

É como numa função matemática. Por exemplo:

- Se $f(x) = 2x$, então:
  - $f(2) = 4$
  - $f(-2) = 4$
- Mas $f(2)$ apenas tem um valor, que é 4.
- Não pode ter mais do que um valor!

Nós definimos as dependência funcionais com base nos requisitos apresentados, geralmente a partir de uma relação que representa uma tabela (ou várias, depois da normalização) numa base de dados, como o conjunto de todos os atributos que aí queremos armazenar. Por exemplo, vamos tentar definir a relação `Loja`:

```
Loja(nomeLoja, moradaLoja, codPostalLoja,
     nomeProduto, marcaProduto, descricaoProduto,
     nomeFornecedor, moradaFornecedor, codPostalFornecedor)
```

Ou, de forma mais sucinta:

```
Loja(NL, ML, CPL, NP, MP, DP, NF, MF, CPF)
```

Alguns exemplos de dependências funcionais (requisitos em `monospace`):

$D = \{$

`Podem existir várias lojas com o mesmo nome, mas apenas uma por código postal.`

$NL, CPL \rightarrow ML,$

`Um produto pode ser fabricado por várias marcas, mas cada produto tem a sua descrição por marca.`

$NP, MP \rightarrow DP,$

`Qualquer produto de determinada marca é sempre fornecido pelo mesmo fornecedor.`

$MP \rightarrow NF,$

`Mas o mesmo fornecedor pode distribuir diferentes produtos da mesma marca a partir de diferentes localizações.`

$NP, MP \rightarrow NF, CPF$

$\ldots$

$\}$

Formas normais
http://www.bkent.net/Doc/simple5.htm
http://www2.york.psu.edu/~lxn/IST_210/normal_form_definitions.html

Tal como o nome indica, as formas normais ajudam-nos a normalizar as nossas relações, no sentido de garantir propriedades como a atomicidade ou de evitar redundância, minimizando assim o custo de armazenamento e facilitando a atualização de valores.

1ª Forma Normal (1NF):
https://en.wikipedia.org/wiki/First_normal_form

- Foco na atomicidade (i.e., indivisibilidade)
- Regras
  - Eliminar grupos repetidos em tabelas individuais
    - Exemplos de grupos repetidos:
      - Coluna `Telefone`, com o valor `555-222-000, 555-333-111`;
      - Duas colunas `Telefone1` e `Telefone2`.
  - Criar uma tabela separada para cada conjunto de dados relacionados.
    - Exemplo: dados sobre uma `Loja`s e dados sobre `Fornecedor`es devem estar separados.

    - Identificar cada conjunto de dados relacionados com uma chave primária.
        - Qualquer chave-candidata pode ser chave-primária, mas apenas pode existir uma chave-primária, enquanto podem existir várias chaves-candidatas (UNIQUE em SQL).

2ª Forma Normal (2NF):
https://en.wikipedia.org/wiki/Second_normal_form

> *Second normal form is violated when a non-key field is a fact about a subset of a key.*
> -- *http://www.bkent.net/Doc/simple5.htm*

- Foco na relação entre atributos chave e atributos não-chave.
- Regras
    - Deve estar na 1ª Forma Normal.
    - Todos os atributos não-chave devem ser diretamente dependentes dos atributos chave no seu todo.
    - Ou seja, atributos não-chave não podem ser dependentes de partes das chaves.

3ª Forma Normal (3NF):
https://en.wikipedia.org/wiki/Third_normal_form
http://rdbms.opengrass.net/2_Database%20Design/2.2_Normalisation/2.2.6_3NF-Transitive%20Dependency.html

> *Third normal form is violated when a non-key field is a fact about another non-key field*
> -- *http://www.bkent.net/Doc/simple5.htm*

> *The key, the whole key, and nothing but the key, so help me Codd.* 🙏
> -- *http://www.itprotoday.com/microsoft-sql-server/so-help-me-codd*

- Foco na relação de transitividade entre atributos chave e atributos não-chave.
- Geralmente considera-se uma base de dados "normalizada" quanto está na 3ª Forma Normal.
- Regras
    - Deve estar na 2ª Forma Normal.
    - Não devem existir dependências transitivas entre atributos não-chave e atributos chave.

Forma Normal Boyce-Codd (BCNF; 3.5NF):
https://en.wikipedia.org/wiki/Boyce%E2%80%93Codd_normal_form

- 3ª Forma Normal *on steroids*.

- Regra
  - Todos os lados esquerdo das dependências funcionais (não triviais) devem ser super-chaves da relação.
    - Se estiver na 3NF e não houver chaves-candidatas com sobreposição, garantidamente está na BCNF.
    - Mas se houver sobreposição, tanto pode estar como não estar.

Forma minimal
(Minimal basis; The Synthesis Algorithm for 3NF Schemas)

- Regras
  - Todos os lados direitos das dependências funcionais (DFs) têm apenas um atributo.
  - Se alguma dependência funcional for removida, deixamos de ter uma forma minimal.
  - Se removermos algum dos atributos do lado esquerdo, deixamos de ter uma forma minimal.
- A forma minimal ajuda-nos a chegar à 3ª Forma Normal:
  - Os lados esquerdos das das DFs tornam-se chaves primárias das novas relações;
  - Os respetivos lados direitos por chave primária são incluídos nessa relação;
  - Deve existir pelo menos uma nova relação com uma super-chave da relação original (caso contrário a junção terá perdas).

Chaves

Super-chave:
https://en.wikipedia.org/wiki/Superkey
https://dba.stackexchange.com/a/71926

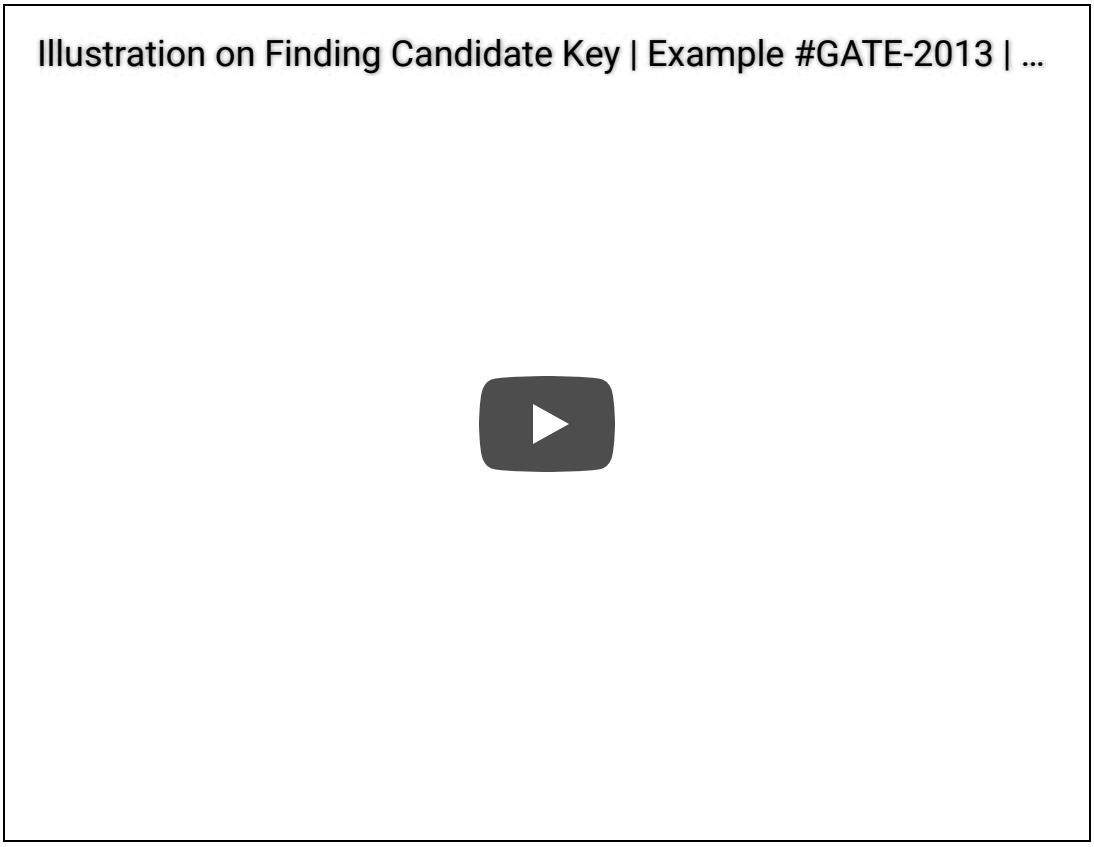
- "**Super**conjunto de uma **chave**", ou seja, chave que contém as chaves minimais.
- Identifica de forma única os registos, mas não é minimal!
- Por exemplo: se os registos numa relação podem ser unicamente identificados pelos atributos A_1, A_2 e A_3, então $\{A_1, A_2, A_3, A_7\}$ é uma super-chave dessa relação. Mas se retirarmos A_7 já passa a ser uma chave-candidata, porque é minimal.

Chave-candidata:
https://en.wikipedia.org/wiki/Candidate_key

- Uma chave-candidata (ou simplesmente chave, no livro recomendado) é uma super-chave minimal.
- Ou seja, não inclui atributos redundantes para identificar de forma única os registos.
- Para encontrar todas as chaves-candidatas:
  - Com base nas dependências funcionais, criar o conjunto de atributos que não aparecem no lado direito das DFs.

- Calcular o fecho desse conjunto e verificar se contém todos os atributos.
  - Se sim, então é uma chave-candidata.
  - Se não, estender o conjunto de atributos com combinações de outros atributos

Este vídeo ajuda:



https://youtu.be/9fuJUQJd-A8

Chave-primária:
https://en.wikipedia.org/wiki/Primary_key

- Corresponde à escolha de uma chave-candidata.
- Todas as outras chaves-candidatas passam a ser chaves-alternativas.

Chave-alternativa:
https://en.wikipedia.org/wiki/Primary_key#Alternate_key

- Qualquer chave-candidata que não foi escolhida para ser chave-primária.

Fecho de um conjunto de atributos
(Closure of a set of attributes)

- Fecho do conjunto X é representado por X^+.

- Para calcular o fecho de um conjunto de atributos $\{J, T, Y\}$, começamos por incluir os atributos originais e expandir os vários subconjuntos com base nas dependências funcionais.
  - Por exemplo, se tivermos as dependências $D = \{JT \rightarrow A, J \rightarrow D, T \rightarrow AY\}$,
  - Vamos expandir $\{J\}$, $\{T\}$, $\{Y\}$, $\{J, T\}$, $\{J, Y\}$, $\{T, Y\}$ e $\{J, T, Y\}$
  - Resultando em (passo-a-passo):
    - $\{J, D\}$
      - Expandir $\{J\}$ resulta em $\{J, D\}$ (inclui o próprio J e, com base em $J \rightarrow D$, adicionamos o D).
    - $\{J, D, T, A, Y\}$
      - Não existe dependência funcional a partir de $\{J, D\}$, por isso vamos expandir $\{T\}$, incluindo o próprio T e, com base em $T \rightarrow AY$, incluímos A e Y.
    - $\{J, T, Y\}^+ = \{J, D, T, A, Y\}$
      - Já não podemos fazer mais nenhuma expansão, por isso chegamos ao fecho $\{J, D, T, A, Y\}$ do conjunto $\{J, T, Y\}$.

Teste da caça

(Chase test for lossless join)

https://en.wikipedia.org/wiki/Chase_(algorithm)

- Algoritmo (tradução para humano):
  a. Criar um quadro com todos os atributos como colunas, em maiúsculas (p.e., A, B, C).
  b. Dada uma decomposição de uma relação R em várias relações $R_1, R_2, \ldots, R_n$, preencher uma linha do quadro por relação, colocando:
     i. Em minúsculas (p.e., a, b, c) todos os atributos que fazem parte da relação.
     ii. E em minúsculas com o índice da linha (p.e., a_1, b_1, c_1) todos os atributos que não fazem parte da relação.
  c. Numa passagem pelo quadro, resolver, linha a linha, as dependências funcionais, removendo o índice para os atributos que podem ser resolvidos através dos atributos sem índice.
  d. Se encontrarmos uma linha sem índices, então podemos garantir a junção sem perdas, caso contrário não.

Aulas 3 e 4

Linguagem de Definição de Dados (SQL)

SQL, de Structured Query Language, é uma linguagem que tanto permite definir as estruturas de dados como manipular os dados nelas armazenados. Vamos olhar para definição de dados em SQL, demonstrando com um workflow típico, para a criação de um esquema em SQLite:

Apagar tabelas, se existirem
Garante que não vamos ter erros no passo seguinte de criação das tabelas.

```
DROP TABLE IF EXISTS tabela;
```

Criar uma tabela
Definição do nome da tabela, conjunto de atributos e seus tipos e restrições.

```
CREATE TABLE tabela (
  attr1 TEXT NOT NULL,
  attr2 TEXT REFERENCES outra_tabela(attr_pk_o_tabela),
  attr3 NUMBER CHECK (attr3 > 0),
  email TEXT,
  CONSTRAINT attr1_pk PRIMARY KEY (attr1),
  CONSTRAINT ch_attr1_attr2 CHECK (attr1 <> attr2)
);
```

Como experimentar

Devem instalar o `sqlite3`. Podem utilizar a linha de comandos para carregar ficheiros `*.sql`, quer com o esquema da base de dados (DDL), quer com interrogações (DML).

Se quiserem algo mais visual, existem várias ferramentas integradas nos IDEs do costume (p.e., o Database Navigator nos produtos da JetBrains, como o IntelliJ ou o PyCharm). Podem ainda experimentar o DB Browser for SQLite, compatível com Windows, Mac e Linux (comando `sqlitebrowser`).

Aula 5

Álgebra Relacional

https://en.wikipedia.org/wiki/Relational_algebra
(Pensar nisto como uma versão matemática do SQL, para interrogação à BD)

O SQL tanto pode servir para a definição de dados (os esquemas que definem as tabelas, suas chaves e restrições, bem como vistas e gatilhos), como para a manipulação de dados (inserções, remoções, atualizações ou interrogações).

Antes de olharmos para o SQL como linguagem de manipulação de dados, vamos primeiro aprender sobre a notação matemática (álgebra relacional) que estabelece as bases teóricas para a interrogação das bases de dados (o nosso `SELECT` em matemática).

Vamos definir alguns operadores unários e binários aplicáveis a relações e que resultam em conjuntos. Se temos conjuntos, então podemos também usar os operadores de teoria de conjuntos. Por exemplo:

- União: $\bigcup$
- Interseção: $\bigcap$
- Produto cartesiano: $\times$
- Diferença: $-$ ou $\setminus$

Operadores unários
Projeção: $\Pi_{a_1,\ldots,a_n}(R)$
Filtragem de atributos $a_1, \ldots, a_n$ numa relação R (colunas da tabela).
Projeção => **P**i

Seleção: $\sigma_{\varphi}(R)$
Filtragem de tuplos com base na condição φ numa relação R (linhas da tabela; os registos).
Seleção => **S**igma

Renomeação: $\rho_{a/b}(R)$
Alteração do nome do atributo b para a, em todos os tuplos.
Renomeação => **R**ho

Operadores binários
Junção natural: $R \bowtie S$
Junta R e S com base em todos os atributos em comum.

θ-junção: $R \overset{\bowtie}{_{a\,\theta\,b}} S$
Filtra o produto cartesiano $R \times S$ pela condição $a\ \theta\ b$. Quando θ é o operador de igualdade ($=$), então temos uma equijunção.

Semijunção esquerda: $R \ltimes S$

Filtra todos os tuplos de R com base na junção com S, mas não inclui atributos de S no resultado.

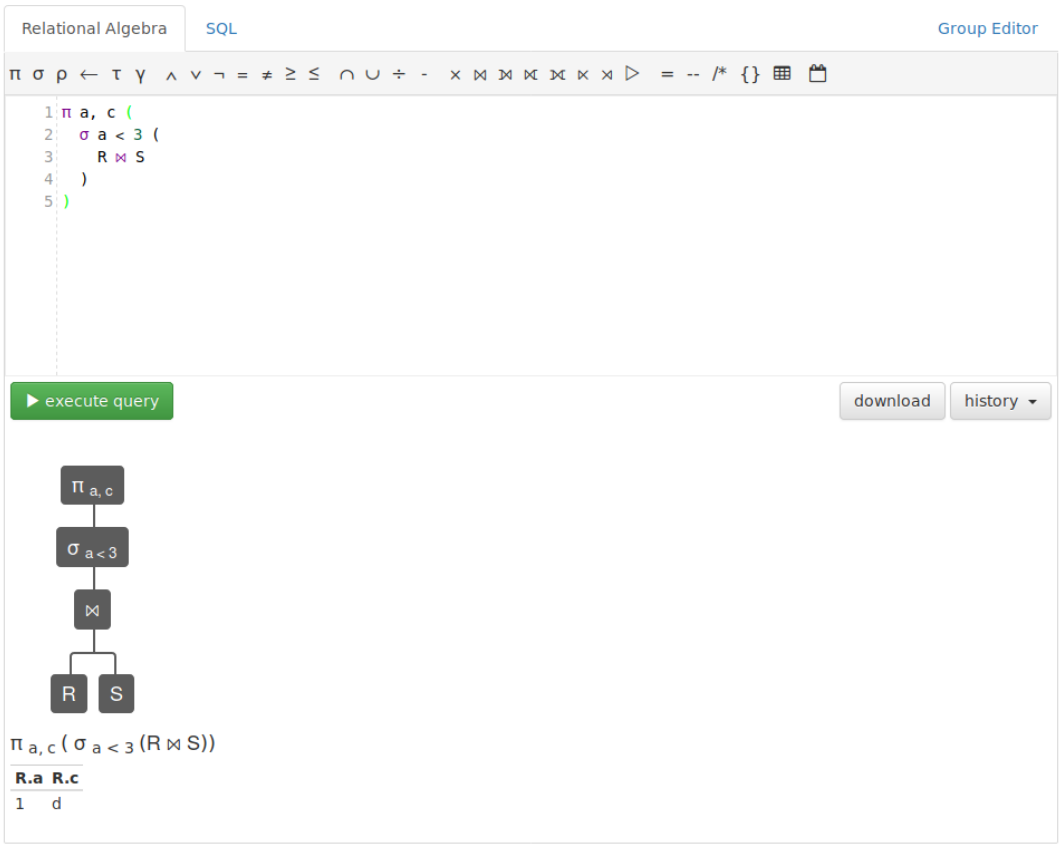
Semijunção direita: $R \rtimes S$
Filtra todos os tuplos de S com base na junção com R, mas não inclui atributos de R no resultado.

Anti junção: $R \triangleright S$
Filtra todos os tuplos de R com base na junção com S, incluindo apenas todos os tuplos que não fazem parte da junção.

Divisão: $R \div S$
Na divisão, fazemos a junção com S, mantendo os atributos únicos de R e removendo tuplos duplicados. Mais detalhes em: http://users.abo.fi/soini/divisionEnglish.pdf

---

Podem experimentar usar o RelaX, uma ferramenta online para aprenderem mais sobre álgebra relacional, testando as vossas interrogações e visualizando as equivalências em SQL. O RelaX foi criado pelo Johannes Kessler, da Universidade de Innsbruck, na Áustria.



Tables from and for the lecture Databases: Foundations, Data Models and System Concepts - University of Innsbruck chapter 3

Aula 6

Linguagem de Manipulação de Dados (SQL)

Inserir dados numa tabela

Geralmente feito a partir de um programa.

```
INSERT INTO tabela VALUES ('567', '23', 987, 'jld@fe.up.pt');
```

Notar que, durante esta inserção, deve existir um registo em `outra_tabela` com chave-primária `attr_pk_outra_tabela` com valor '23'.

Apagar registos

```
DELETE FROM tabela;
```

Ou, com uma condição:

```
DELETE FROM tabela WHERE attr1 = '567';
```

Atualizar registos

```
UPDATE tabela SET attr3 = 988 WHERE attr1 = '567';
```

Aula 7

Gatilhos (SQL / SQLite)

Exemplo de um gatilho em SQLite para validação de e-mails

Adaptado de http://www.sqlitetutorial.net/sqlite-trigger/

```
CREATE TRIGGER validar_email_tabela BEFORE INSERT ON tabela
BEGIN
  SELECT
    CASE
      WHEN NEW.email NOT LIKE '%_@__%.__%'
        THEN RAISE (ABORT, 'E-mail inválido!')
    END;
END;
```

Informação adicional relevante na documentação do SQLite:

- Criar gatilhos: https://www.sqlite.org/lang_createtrigger.html
  - Um gatilho permite definir uma ação que ocorre antes (`BEFORE`), depois (`AFTER`) ou em substituição (`INSTEAD OF`) de um `DELETE`, `INSERT` ou `UPDATE`.
- Criar vistas: https://www.sqlite.org/lang_createview.html
  - Uma vista permite guardar interrogações frequentes à base de dados que geralmente envolvem a junção de várias tabelas ou a aplicação de uma condição (ver Aula 6).
  - As vistas podem por vezes ser otimizadas (vistas materializadas), embora o SQLite não suporte esta funcionalidade.

Aula 8

NoSQL (MongoDB)

Existem várias bases de dados NoSQL:

- **Chave-valor** (p.e., Berkeley DB);

- **Grafos** (p.e., Neo4j);
- **Documentos** (p.e., MongoDB);
- **Multi-modelo** (p.e., OrientDB, que vê o NoSQL como "Not only SQL" e oferece armazenamento de documentos, grafos, dados relacionais, com esquema, parcialmente com esquema ou sem esquema).

MongoDB é uma das bases de dados NoSQL mais utilizada (e robusta). Assim, vamos utilizá-la como exemplo na aprendizagem do NoSQL.

Características do MongoDB:

- Permite armazenar documentos em JSON;
- Dados não normalizados:
  - Por exemplo, se tivermos documentos de texto com `titulo`, `conteudo` e `tags`, as tags são geralmente armazenadas no próprio documento como um array (p.e., `["databases", "nosql", "lecture"]`); se quisermos alterar globalmente o nome de uma tag na base de dados, teremos de o fazer individualmente para cada instância da tag nos vários documentos onde ocorre;
- Esta desnormalização, permite armazenar os documentos conforme estes são frequentemente acedidos, tornando as leituras mais eficientes.
- Em particular no MongoDB e para aplicações web, é útil que estes conteúdos já estejam em formato JSON, evitando assim o overhead do processo de desserialização e contribuindo novamente para um maior eficiência.

Selecionar documentos segundo uma condição e projetar o resultado:

```
db.colecao.find({ campo: "valor" }, { _id: 0, campo: 0 })
```

A base de dados ativa é sempre identificada por `db`, contendo várias coleções de documentos (p.e., `colecao` no exemplo). O primeiro parâmetro do `find()` recebe a nossa condição e o segundo parâmetro determina a projeção, permitindo decidir quais os campos a incluir (0 exclui e 1 inclui, com alguns *defaults* diferentes, mas intituivos, dependendo das combinações usadas).

Contar documentos segundo uma condição:

```
db.colecao.count({ campo: "valor" })
```

Atualizar valores num conjunto de documentos:

```
db.colecao.update({ campo: "valor" }, { $set: { outroCampo: 10
, aindaOutroCampo: true } })
```

Se não tivéssemos usado o operador `$set`, o segundo parâmetro iria substituir por completo o documento.

Remover documentos:

```
db.colecao.remove({ campo: "valor" })
```

Exemplo de uma interrogação (real) mais interessante:
Esta interrogação utiliza o `aggregate()` para aplicar uma série de operações à coleção `extra_round_feedback` ao estilo pipeline (i.e., é um array de instruções `$unwind`, `$group` e `$project`.

```
db.extra_round_feedback.aggregate([
  {
    /***
     * Desdobra doclist, devolvendo um documento
     * para cada elemento desse array.
     */
    $unwind: "$doclist"
  },
  {
    /***
     * Conta os clicados e não clicados, separadamente,
     * por run e equipa, para cada entrada de feedback.
     */
    $group: {
      _id: {
        runid: "$runid",
        _id: "$_id",
        team: "$doclist.team",
        clicked: "$doclist.clicked"
      },
      count: { $sum: 1 }
    }
  },
  /***
   * Transforma as contagens anteriores num array de objetos
   * "stats" com chave "participant" ou "site" e valor igual
   * às respetivas contagens.
```

```
   */
  {
    $group: {
      _id: { runid: "$_id.runid", _id: "$_id._id" },
      stats: {
        $push: {
          $cond: {
            if: {
              $and: [
                { $eq: [ "$_id.team", "participant" ] },
                { $eq: [ "$_id.clicked", true ] }
              ]
            },
            then: {
              "participant": "$count"
            },
            else: {
              $cond: {
                if: {
                  $and: [
                    { $eq: [ "$_id.team", "site" ] },
                    { $eq: [ "$_id.clicked", true ] }
                  ]
                },
                then: {
                  "site": "$count"
                },
                else: {
                  // Não pode acontecer, por isso foi ignorado.
                }
              }
            }
```

```
          }
        }
      }
    }
  },
  /***
   * Determina vitórias, empates e derrotas para a equipa
   * "participant", comparando o número de cliques do
   * "participant" e do "site".
   */
  {
    $project: {
      _id: 1,
      win: { $gt: [ "$stats.participant", "$stats.site" ] },
      tie: { $eq: [ "$stats.participant", "$stats.site" ] },
      loss: { $lt: [ "$stats.participant", "$stats.site" ] }
    }
  },
  /***
   * Agrega o número de vitórias, empates e derrotas por run.
   */
  {
    $group: {
      _id: "$_id.runid",
      wins: { $sum: { $cond: [ "$win", 1, 0 ] } },
      ties: { $sum: { $cond: [ "$tie", 1, 0 ] } },
      losses: { $sum: { $cond: [ "$loss", 1, 0 ] } }
    }
  },
  /***
   * Calcula o "outcome" com base no número de vitórias e
   * derrotas.
   */
```

```
  {
    $project: {
      _id: 0,
      runid: "$_id",
      wins: 1,
      ties: 1,
      losses: 1,
      outcome: {
        $cond: [
          {
            $and: [
              { $eq: [ "$wins", 0 ] },
              { $eq: [ "$losses", 0 ] }
            ]
          },
          0,
          {
            $divide: [
              "$wins",
              { $sum: [ "$wins", "$losses" ] }
            ]
          }
        ]
      }
    }
  }
])
```

Observações

Agradeço ao Professor Jorge Sousa Pinto que teve a ideia de utilizar o Dropbox Paper para organizar as aulas e me inspirou a fazer o mesmo.