TÉCNICO LISBOA

Bases de Dados

T25 - OLAP Parte III

Prof. Daniel Faria

Sumário

- Recapitulação
- OLAP: Exemplos
- Engenharia de Dados

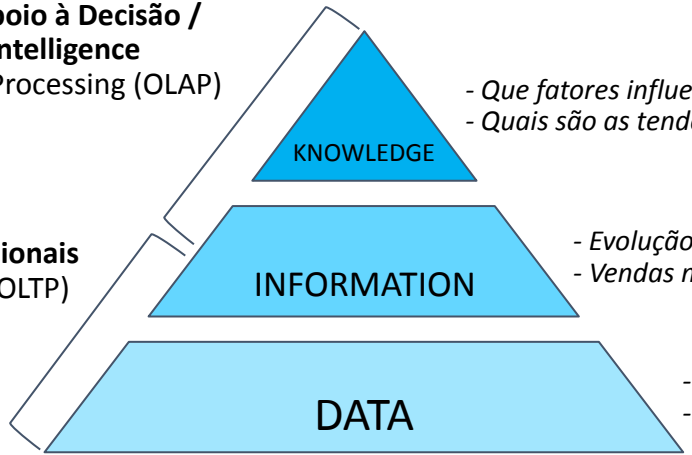
TÉCNICO LISBOA

Recapitulação

Sistemas de Informação

Sistemas de Apoio à Decisão / Business Intelligence
Online Analytic Processing (OLAP)

Sistemas de Informação Operacionais
Online Transaction Processing (OLTP)



- Que fatores influenciam as vendas?
- Quais são as tendências das vendas?

- Evolução das vendas por cidade e mês
- Vendas mensais por produto em Lisboa

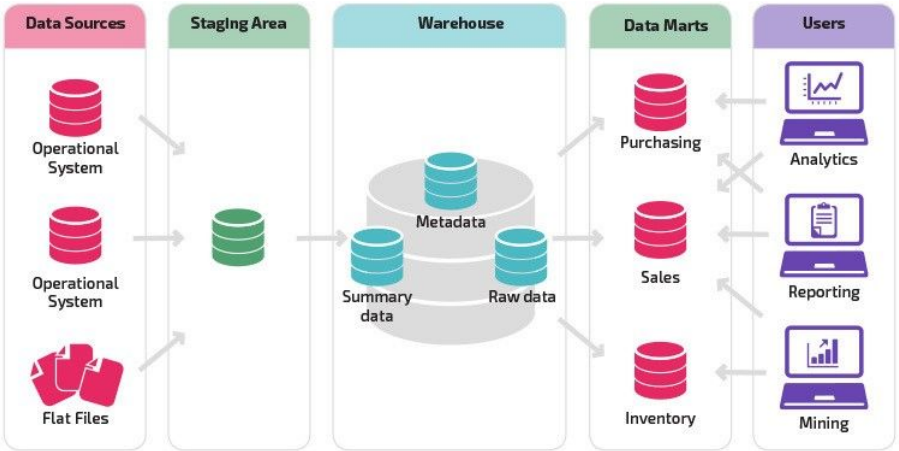
- Listar o preço de cada produto
- Registar vendas

OLTP vs. OLAP

- **Online Transaction Processing (OLTP)**
  - Dados dinâmicos
    - Operações de escrita frequentes
  - Transações de escrita ou leitura rápidas e simples
- **Online Analytic Processing (OLAP)**
  - Dados quase estáticos
    - Atualizações periódicas (e.g. mensais, anuais) em bulk
  - Transações complexas mas só de leitura

Data Warehouse

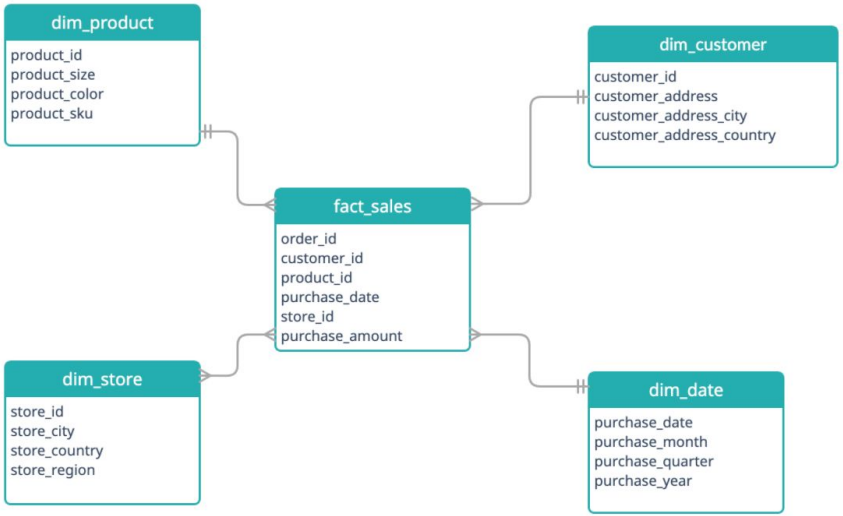
- Repositório central de grande volume de dados consolidados, históricos e agregados, complementados com sumários
- Permite simplificar queries complexas para análise de dados
- Base para reporte e análise de dados e componente chave de business intelligence



Esquema em Estrela

Dois tipos de tabelas:

- **Tabela(s) de factos**
  - Grande dimensão
  - Frequentemente normalizada(s)
  - Objeto primário de análise de dados
- **Tabelas de dimensões**
  - Relativamente pequenas
  - Geralmente não normalizadas
  - Contém informação adicional sobre os elementos (ou dimensões) da tabela de factos



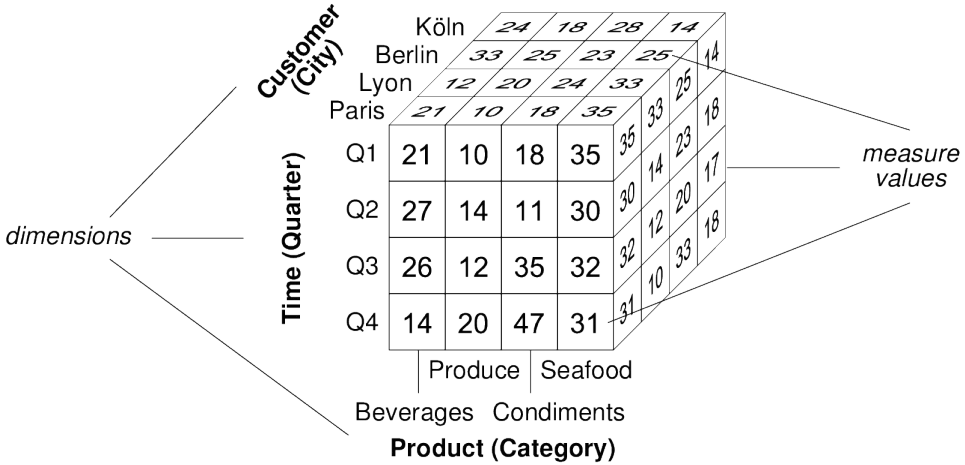
TÉCNICO LISBOA

OLAP

- **Processo interactivo** de criar, gerir, analisar, e reportar sobre dados
- Análise de grandes quantidades de dados em **tempo real** (i.e., com latência negligível)
- Os dados são compreendidos e manipulados como se estivessem guardados num **array multi-dimensional** (um hipercubo)
    - Mas podemos implementar modelo em estrela em SQL, e existe suporte para várias operações OLAP

OLAP

- Modelo de dados: hipercubo (pode ser implementado com esquema em estrela)

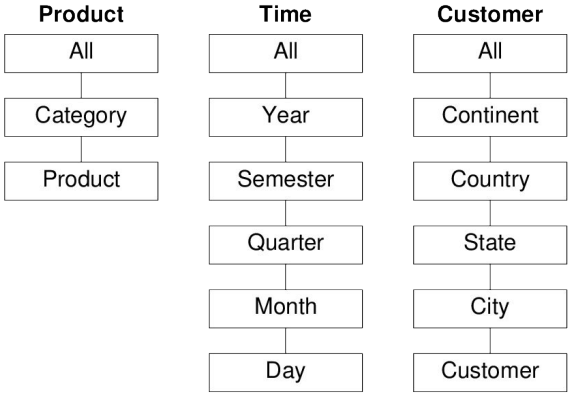


A. Vaisman, E. Zimányi, *"Data Warehouse Systems: Design and Implementation"*, Springer, 2014

TÉCNICO LISBOA

OLAP

- Dimensões tipicamente podem ser organizadas hierarquicamente
  - Podemos ter cubos com diferentes granularidades nas dimensões
  - Podemos alterar a granularidade dos cubos:
    - **Rollup:** diminuir a granularidade
    - **Drill-down:** aumentar a granularidade



OLAP

- **Pivoting / Cross-Tabulating:**
  - Transformação de duas dimensões do array multidimensional, em que uma dimensão passa a coluna (corresponde a uma face do cubo)
  - Tipicamente com agregação seletiva em múltiplas dimensões
- **Slicing & Dicing:**
  - Restrições por igualdade e intervalo numa ou mais dimensões

Cross-Tab / Pivot em SQL

crosstab(*text source_sql*, text *category_sql*)

```
SELECT * FROM CROSSTAB('SELECT …', 'SELECT …')
    AS tablename(colname1 TYPE, colname2 TYPE, …);
```

- *category_sql* é uma statement SQL que produz a lista de categorias
  - Tem de retornar apenas uma coluna e pelo menos uma linha (categoria)
  - Não pode incluir duplicados
- *source_sql* pode conter uma ou mais colunas “extra”
  - A coluna *row_name* tem de ser a primeir
  - As colunas *category* e *value* têm de ser as duas últimas (nesta ordem)
  - Quaisquer outras colunas no meio são consideradas “extra”
  - Têm de ser listadas no AS pela mesma ordem

https://www.postgresql.org/docs/current/tablefunc.html

Cross-Tab / Pivot em SQL

```
 id |  name   | species | job |     dob
----+---------+---------+-----+------------
  1 | Daniel  | pig     |     |
  2 | Daniel  | pig     |     |
  3 | Daniel  | goat    |     |
  4 | Daniel  | goat    |     |
  5 | Daniel  | cow     |     |
  6 | Daniel  | sheep   |     |
  7 | Rita    | goat    |     |
  8 | Rita    | cow     |     |
  9 | Rita    | cow     |     |
 10 | Rita    | sheep   |     |
 11 | Rita    | sheep   |     |
 12 | Rita    | chicken |     |
```

```
SELECT * FROM CROSSTAB(
    'SELECT name, species, COUNT(*) FROM animalD
        GROUP BY name, species ORDER BY name, species',
    'SELECT DISTINCT species FROM animalD ORDER BY
    species')
AS names (name VARCHAR(80), chicken BIGINT, cow BIGINT,
goat BIGINT, pig BIGINT, sheep BIGINT);
```

```
  name   | chicken | cow | goat | pig | sheep
---------+---------+-----+------+-----+-------
 Daniel  |         |   1 |    2 |   2 |     1
 Rita    |       1 |   2 |    1 |     |     2
```

https://www.postgresql.org/docs/current/tablefunc.html

Agregações OLAP em SQL

- Agregações complexas permitindo queries OLAP
- Podem ser incluídas na cláusula GROUP BY:
  - GROUPING SETS: grupos explicitamente definidos
  - ROLLUP: combinações hierárquicos das colunas listadas (da direita para a esquerda)
  - CUBE: todas as combinações das colunas listadas
- Parte do standard SQL desde 1999

https://www.postgresql.org/docs/current/queries-table-expressions.html#QUERIES-GROUPING-SETS

Agregações OLAP

- **GROUPING SETS:**

```
SELECT name, species, COUNT(*) FROM animalD GROUP BY
GROUPING SETS ((name), (species), (name,species));

   name   | species | count
----------+---------+-------
 Rita     | cow     |   2
 Rita     | goat    |   1
 Rita     | sheep   |   2
 Daniel   | goat    |   2
 Daniel   | pig     |   2
 Daniel   | sheep   |   1
 Daniel   | cow     |   1
 Rita     | chicken |   1
 Rita     |         |   6
 Daniel   |         |   6
          | goat    |   3
          | cow     |   3
          | pig     |   2
          | chicken |   1
          | sheep   |   3
```

TÉCNICO LISBOA

Agregações OLAP

- **ROLLUP:**

```
ROLLUP ( e1, e2, e3 )
```

```
GROUPING SETS (
    ( e1, e2, e3 ),
    ( e1, e2 ),
    ( e1 ),
    ( )
)
```

ROLLUP não é simétrico!

```
SELECT name, species, COUNT(*) FROM animalD GROUP BY
ROLLUP (name, species);

    name   | species | count
-----------+---------+-------
           |         |    12
 Rita      | cow     |     2
 Rita      | goat    |     1
 Rita      | sheep   |     2
 Daniel    | goat    |     2
 Daniel    | pig     |     2
 Daniel    | sheep   |     1
 Daniel    | cow     |     1
 Rita      | chicken |     1
 Rita      |         |     6
 Daniel    |         |     6
```

Agregações OLAP

- **CUBE:**

```
CUBE ( e1, e2, e3 )
```

```
GROUPING SETS (
    ( e1, e2, e3 ),
    ( e1, e2     ),
    ( e1,     e3 ),
    ( e1         ),
    (     e2, e3 ),
    (     e2     ),
    (         e3 ),
    (            )
)
```

```
SELECT name, species, COUNT(*) FROM animalD GROUP BY CUBE
(name,species);

   name   | species | count
----------+---------+-------
          |         |    12
 Rita     | cow     |     2
 Rita     | goat    |     1
 Rita     | sheep   |     2
 Daniel   | goat    |     2
 Daniel   | pig     |     2
 Daniel   | sheep   |     1
 Daniel   | cow     |     1
 Rita     | chicken |     1
 Rita     |         |     6
 Daniel   |         |     6
          | goat    |     3
          | cow     |     3
          | pig     |     2
          | chicken |     1
          | sheep   |     3
```

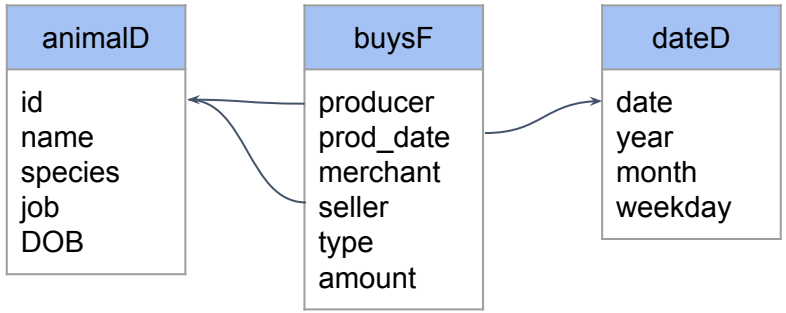
TÉCNICO LISBOA

TÉCNICO LISBOA

OLAP: Exemplos

BD Porcos Esquema em Estrela

```
CREATE TABLE animalD(
    id INTEGER PRIMARY KEY,
    name VARCHAR(80),
    species VARCHAR(80),
    job VARCHAR(80),
    DOB DATE);

CREATE TABLE dateD(
    date DATE PRIMARY KEY,
    year NUMERIC(4,0),
    month NUMERIC(2,0),
    weekday NUMERIC(1,0));

CREATE TABLE buysF(
    producer INTEGER REFERENCES animalD(id),
    prod_date DATE REFERENCES dateD(date),
    merchant NUMERIC(8),
    seller INTEGER REFERENCES animalD(id),
    type CHAR(4),
    amount  FLOAT);
```



TÉCNICO LISBOA

Exemplos

1) Produzir uma tabela pivot (cross-tab) com a quantidade total produzida por mês do ano passado (linhas) e por espécie (colunas)

```
SELECT * FROM CROSSTAB(
   'SELECT month, species, SUM(amount) FROM buysF
        JOIN animalD ON (producer = id) JOIN dateD ON (prod_date = date)
        WHERE EXTRACT(YEAR FROM CURRENT_DATE) - year = 1
        GROUP BY month, species ORDER BY name, species',
   'SELECT DISTINCT species FROM animalD WHERE species != ''pig''
        ORDER BY species')
AS names (month INT, chicken FLOAT, cow FLOAT, goat FLOAT, sheep FLOAT);
```

Notar que é preciso double-quotes em ‘‘pig’’ porque o SELECT statement já está dentro de single-quotes

Exemplos

2) Analisar a produção diária média no ano passado com agrupamentos por dia da semana e mês, por espécie

```
SELECT month, weekday, species, AVG(amount) FROM buysF
    JOIN animalD ON (producer = id) JOIN dateD ON (prod_date = date)
    WHERE EXTRACT(YEAR FROM CURRENT_DATE) - year = 1
    GROUP BY GROUPING SETS ((month, species), (weekday, species))
    ORDER BY month, weekday, species;
```

Exemplos

3) Produzir uma tabela pivot com o número total de compras no ano passado por cada mercador (linhas) por espécie (colunas), incluindo o total de espécies

```
SELECT * FROM CROSSTAB(
    'SELECT merchant, COALESCE(species,''total'') AS sp, COUNT(*)
        FROM buysF JOIN animalD ON (producer = id) JOIN dateD ON (prod_date = date)
        WHERE EXTRACT(YEAR FROM CURRENT_DATE) - year = 1
        GROUP BY ROLLUP(merchant, sp) ORDER BY merchant, sp',
   'SELECT DISTINCT species FROM animalD WHERE species != ''pig''
        UNION SELECT ''total'' ORDER BY species')
AS names (SSN NUMERIC(8), chicken BIGINT, cow BIGINT, goat BIGINT, sheep BIGINT, total
BIGINT);
```

Podemos juntar o agrupamento por espécie com o agrupamento global com uma UNION ou com ROLLUP, mas no segundo caso é necessário COALESCE para substituir os NULLs

Exemplos

4) Analisar o número médio diário de vendas feitas por cada porco com rollup no tipo de produto (tipo > espécie) e no tempo (ano > mês)

```
WITH temp AS
     (SELECT s.id, year, month, date, type, species, COUNT(*) AS counts
          FROM buysF JOIN animalD p ON (producer = p.id) JOIN animalD s
          ON (seller = s.id) RIGHT JOIN dateD ON (prod_date = date)
          GROUP BY s.id, month, date, type, species)
SELECT id, year, month, type, species, AVG(counts) FROM temp
    GROUP BY id, ROLLUP(year,month), ROLLUP(type,species)
    ORDER BY id, year, month, type, species;
```

Queremos agregar um agregado (a média da contagem diária) pelo que precisamos de um SELECT encadeado (ou um WITH) que tem de ter todos os atributos que queremos no SELECT final, mais o atributo sobre o qual vamos fazer a agregação externa (a data).

O RIGHT JOIN com dateD assegura que temos todas as datas (mesmo aquelas em que não houve produção).

TÉCNICO LISBOA

Engenharia de Dados

Chaves Substitutas

- Uma **chave semântica** (ou natural, ou de negócio) é uma chave baseada nos atributos do objeto representado pela tabela
  - É uma maneira simples e intuitiva de comparar objetos
- Uma **chave substituta** (ou técnica, ou sintética) é uma chave cujos valores geralmente não têm qualquer relação com os atributos do objeto
  - Tipicamente é **numérica auto-incremental**, criada quando os dados são inseridos na tabela, e imutável

Chaves Substitutas

- O uso de chaves substitutas facilita os joins de tabelas, particularmente em casos de tabelas com chaves compostas
    - Podíamos pensar que aumenta o volume de dados da BD mas frequentemente redu-lo, e reduz sempre o processamento de joins
- É **prática comum em BDs operacionais**, mas é particularmente **crítica em BDs de esquema em estrela**, para facilitar os joins entre factos e dimensões
    - Chaves naturais das tabelas de dimensões são frequentemente compostas por múltiplos atributos

Chaves Substitutas: Exemplo

```
CREATE TABLE animal(
    id SERIAL PRIMARY KEY,
    name VARCHAR(80),
    DOB DATE,
    UNIQUE(name,DOB));

CREATE TABLE nonpig(
    id INTEGER PRIMARY KEY REFERENCES animal,
    species VARCHAR(80) NOT NULL);

CREATE TABLE produce(
    id SERIAL PRIMARY KEY,
    nonpig_id INTEGER REFERENCES nonpig(id),
    production_date DATE,
    amount  FLOAT NOT NULL,
    type CHAR(4) NOT NULL)
    UNIQUE(nonpig_id,production_date));
```

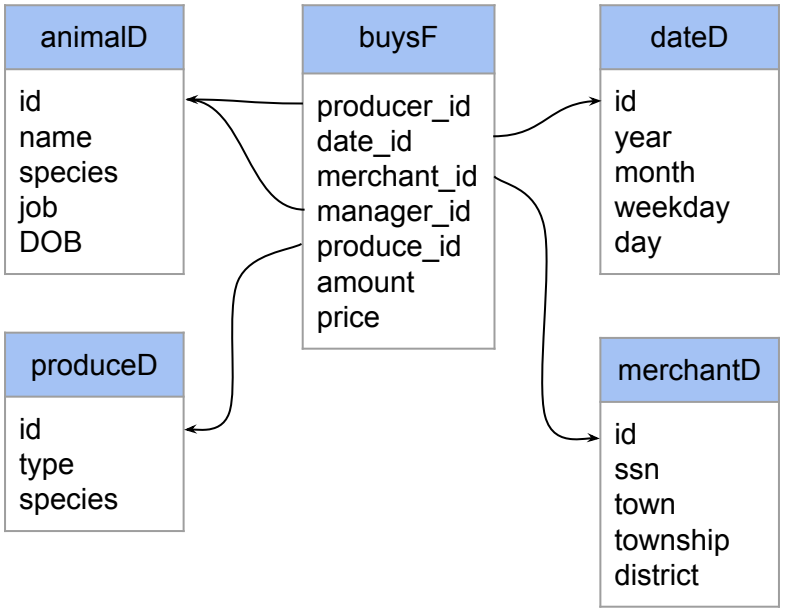
```
CREATE TABLE pig(
    id INTEGER PRIMARY KEY REFERENCES animal,
    job VARCHAR(80) NOT NULL);

CREATE TABLE merchant(
    id SERIAL PRIMARY KEY,
    ssn VARCHAR,
    address VARCHAR,
    UNIQUE(ssn));

CREATE TABLE buys(
    id SERIAL PRIMARY KEY
    merchant_id INTEGER REFERENCES merchant(id),
    produce_id INTEGER REFERENCES produce(id),
    seller_id INTEGER REFERENCES seller(id),
    price NUMERIC(20,4),
    UNIQUE(SSN,produce_id));
```

Em PostgreSQL: chave substituta: SERIAL ou BIGSERIAL; referência à chave: INT ou BIGINT

Restrições de Integridade e Índices

- Chaves estrangeiras, checks e outras restrições de integridade são pouco importantes numa BD OLAP que deriva de BDs operacionais normalizadas
  - Dados são inseridos em batch e a integridade é assegurada pelas BDs originais
- Índices são mais úteis numa BD OLAP do que numa BD operacional, dado que vai ser só de leitura, e suportar agregações complexas
  - Devem no entanto ser criados apenas após a inserção dos dados (ou removidos e re-adicionados aquando de atualização dos dados)

Esquema em Estrela em SQL BD Porcos



Esquema em Estrela em SQL BD Porcos

```
CREATE TABLE animalD(
    id SERIAL PRIMARY KEY,
    name VARCHAR,
    species VARCHAR,
    job VARCHAR,
    DOB DATE);

CREATE TABLE dateD(
    id SERIAL PRIMARY KEY,
    year INTEGER,
    month INTEGER,
    weekday INTEGER,
    day INTEGER);

CREATE TABLE produceD(
    id SERIAL PRIMARY KEY,
    type VARCHAR,
    species  VARCHAR);
```

```
CREATE TABLE merchantD(
    id SERIAL PRIMARY KEY,
    ssn VARCHAR,
    zip_code VARCHAR,
    town VARCHAR,
    township VARCHAR,
    district VARCHAR);

CREATE TABLE buysF(
    producer_id INTEGER REFERENCES animal(id),
    date_id INTEGER REFERENCES dateD(id),
    merchant_id INTEGER REFERENCES merchant(id),
    seller_id INTEGER REFERENCES animal(id),
    produce_id INTEGER REFERENCES produce(id),
    amount  FLOAT,
    price  NUMERIC(20,4));
```

TÉCNICO LISBOA

Popular Tabelas de Dimensões

- Tabelas de dimensões são tipicamente pequenas
- São populadas com valores distintos da dimensão correspondente:
  - Importando da BD operacional (dimensões de negócio)
  - Gerando uma série de dados (dimensões temporais)
  - Importando a partir de ficheiros externos (e.g. dimensões espaciais)
- Tipicamente incluem uma chave substituta que é criada automaticamente quando são inseridos dados

Popular Tabelas de Dimensões: Negócio

```
INSERT INTO animalD (id, name, DOB, species, job)
     SELECT id, name, DOB, COALESCE(species,'pig'), job
          FROM animal JOIN pig USING (id)
          JOIN nonpig USING (id);

INSERT INTO produceD (type, species)
     SELECT DISTINCT type, species
          FROM produce p JOIN nonpig n ON (p.nonpig_id = n.id);
```

- É frequentemente preciso lidar com NULLs ao juntar dados de tabelas (ou BDs) distintas: COALESCE ou fazer UPDATE após o INSERT
- É frequentemente necessário fazer UNION (ou FULL JOIN) de tabelas distintas (em particular quando dados vêm de BDs distintas)

Popular Tabelas de Dimensões: Tempo

```
INSERT INTO dateD (year, month, weekday, day)
     SELECT EXTRACT(YEAR FROM dd), EXTRACT(MONTH FROM dd),
          EXTRACT(DOW FROM dd), EXTRACT(DAY FROM dd)
          FROM GENERATE_SERIES(
          (SELECT MIN(date) FROM produce),
          (SELECT MAX(date) FROM produce),
          '1 day'::INTERVAL) dd;
```

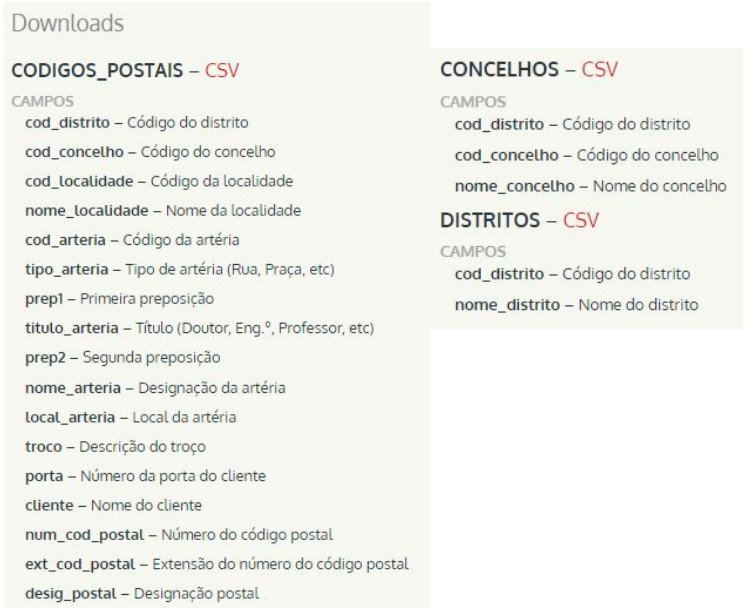
- População gerando série de datas que abarcam todos os dados e depois desconstruindo as datas nas partes desejadas
- Para tempo a abordagem é a mesma, mas definindo intervalos temporais (e.g. ‘1 minute’, ‘1 second’);

Popular Tabelas de Dimensões: Espaço

0. Ponto de partida:
    - Temos moradas terminando em código postal e localidade de Portugal
    - Não temos informação interna sobre concelho ou distrito
1. Procurar fonte externa de informação
2. Pré-processar informação da fonte externa (se necessário)
3. Carregar informação em tabelas de *staging*
4. Processar a informação para popular as tabelas finais

Popular Tabelas de Dimensões: Espaço

1. Procurar fonte externa de informação:
   - http://centraldedados.pt/codigos_postais/



Downloads

CODIGOS_POSTAIS – CSV

CAMPOS

- **cod_distrito** – Código do distrito
- **cod_concelho** – Código do concelho
- **cod_localidade** – Código da localidade
- **nome_localidade** – Nome da localidade
- **cod_arteria** – Código da artéria
- **tipo_arteria** – Tipo de artéria (Rua, Praça, etc)
- **prep1** – Primeira preposição
- **titulo_arteria** – Título (Doutor, Eng.º, Professor, etc)
- **prep2** – Segunda preposição
- **nome_arteria** – Designação da artéria
- **local_arteria** – Local da artéria
- **troco** – Descrição do troço
- **porta** – Número da porta do cliente
- **cliente** – Nome do cliente
- **num_cod_postal** – Número do código postal
- **ext_cod_postal** – Extensão do número do código postal
- **desig_postal** – Designação postal

CONCELHOS – CSV

CAMPOS

- **cod_distrito** – Código do distrito
- **cod_concelho** – Código do concelho
- **nome_concelho** – Nome do concelho

DISTRITOS – CSV

CAMPOS

- **cod_distrito** – Código do distrito
- **nome_distrito** – Nome do distrito

TÉCNICO LISBOA

Popular Tabelas de Dimensões: Espaço

2. Pré-processar informação da fonte externa (se necessário)
    - Não é necessário, pois o PostgreSQL consegue carregar ficheiros CSV
3. Carregar informação em tabelas de *staging*
    - Criar tabelas com o esquema dos ficheiros de dados: códigos postais, concelhos distritos
    - Usar comando COPY para importar os ficheiros de dados para as tabelas correspondentes

Popular Tabelas de Dimensões: Espaço

4. Processar a informação para popular as tabelas finais

```
INSERT INTO merchantD (id, ssn, zip_code)
     SELECT id, ssn, SUBSTRING(address FROM '[0-9][0-9][0-9][0-9]-[0-9][0-9][0-9]')
          FROM merchant;

UPDATE merchantD SET town = (SELECT nome_localidade FROM codigos_postais
     WHERE num_cod_postal = SUBSTRING(zip_code FOR 4)
     AND ext_cod_postal = SUBSTRING(zip_code FROM 6),
SET township = (SELECT nome_concelho FROM codigos_postais JOIN concelhos USING (cod_concelho)
     WHERE num_cod_postal = SUBSTRING(zip_code FOR 4)
     AND ext_cod_postal = SUBSTRING(zip_code FROM 6),
SET district = (SELECT nome_distrito FROM codigos_postais JOIN distritos USING (cod_distrito)
     WHERE num_cod_postal = SUBSTRING(zip_code FOR 4)
     AND ext_cod_postal = SUBSTRING(zip_code FROM 6);
```

Popular Tabelas de Factos

- Tabela(s) de factos são tipicamente grandes (muito mais linhas do que as tabelas de dimensões)
- São quase sempre populadas com dados importados de BDs operacionais
  - Possivelmente unindo dados de várias tabelas
  - Possivelmente “desnormalizando” (i.e. fundindo) tabelas normalizadas
    - E.g. tabela buyF engloba a junção da tabela produce com a tabela buys
- É necessário traduzir chaves semânticas para chaves substitutas para todas as dimensões

Popular Tabelas de Factos

```
CREATE TABLE temp AS
     SELECT * FROM produce p JOIN buys b ON (p.id = b.produce_id);

ALTER TABLE temp ADD date_id INTEGER;

UPDATE temp SET date_id = (SELECT id FROM dateD
     WHERE year = EXTRACT(YEAR FROM production_date)
     AND month = EXTRACT (MONTH FROM production_date)
     AND day = EXTRACT (DAY FROM production_date);

ALTER TABLE temp ADD produce_id INTEGER;

UPDATE temp t SET produce_id = (SELECT id FROM produceD p
     WHERE p.type = t.type AND p.species = t.species);

INSERT INTO buysF (producer_id, date_id, merchant_id, seller_id, produce_id, amount, price)
     SELECT nonpig_id, date_id, merchant_id, seller_id, produce_id, amount, price
     FROM temp;
```

TÉCNICO LISBOA