|  |  |  |  |
| --- | --- | --- | --- |
|  | 19/05/2017 |  | |
|  | |  | |
| Modelação de Data Warehousing  *Análise aos dados multidimensional dos serviços e praças de táxis no Porto*  *Tópicos Avançados em Bases de Dados – CC4007* | | | |
|  |  | |  |
|  |  | | **Fábio Teixeira** – 201305725  **Sara Pereira** – 201304112  **Vanessa Silva** – 201305731 |



# Objetivos

Este trabalho prático consiste na modelação de um *data warehouse* sobre o conjunto de dados guardados nas tabelas *taxi\_services* e *taxi\_stands* apresentadas nas aulas, bem como na análise destes dados multidimensionais.

Não existe uma definição rigorosa de *data warehouse*, mas pode ser considerada uma base de dados de suporte à decisão que é mantida separadamente da base operacional da organização e que suporta processamento de informação de forma a fornecer uma plataforma sólida para análise de dados históricos, consolidados. O processo de construir e usar *data warehouses* é denominado de *data warehousing*.

O *data warehouse* criado para este trabalho não inclui qualquer informação espacial (na forma de *geometry* ou varianates), para tal foi criada uma hierarquia conceptual sobre a dimensão espacial (disponível no site do [Instituto Geográfico do Exército](mailto:http://ftp.igeo.pt/produtos/cadastro/caop/versao5.htm?subject=CAOPV%205.0%20–%20julho%20de%202006)), com dois níveis, o concelho e a freguesia.

Por último, após a criação do *data warehouse* realizamos uma análise do mesmo. Esta análise é feita com o auxílio de consultas OLAP (*Online Analytical Processing*), e em particular consultas de histogramas, *cross-tabulation*, e a utilização do operador *CUBE BY*. OLAP refere-se à capacidade de manipular um grande volume de dados sob múltiplas perspetivas, as suas aplicações são normalmente usadas por analistas de mercado com o objetivo de suporte a decisões.

# Processo

Como referido anteriormente, partimos para a realização do trabalho com o suporte de duas tabelas já existentes: *taxi\_services* e *taxi\_stands*. A primeira representa um conjunto de dados relativos a serviços de táxis, nomeadamente, informação sobre o tempo em que inicia o serviço, tempo em que o termina, identificador do táxi e localização espacial (no tipo de dados *geometry*) do local onde inicial o serviço e outra do local onde termina; e a segunda representa a informação sobre as praças de táxis existentes, nomeadamente, o nome da praça e a sua localização espacial (no tipo de dados *geometry*).

O primeiro passo dado para a obtenção do *data warehouse* pedido foi criar um esquema Floco de Neve, com 5 novas tabelas, *dw\_taxi\_services* correspondente à tabela de factos e *dw\_tempo*, *dw\_taxi*, *dw\_local* e *dw\_stand* que correspondentes às tabelas dimensão.

Neste esquema a tabela de factos central é ligada ao conjunto das tabelas dimensão, (*dw\_tempo*, *dw\_taxi*, *dw\_local*), onde parte desta hierarquia dimensional, (*dw\_local*), é normalizada numa tabela dimensão mais pequena (*dw\_stand*), semelhantemente a um floco de neve.

A partir da informação contida nas tabelas base, conseguimos obter novos dados, como o tempo total de cada serviço (viagem) (*tempoTotal* presente em *dw\_taxi\_services*), resultado da simples diferença entre o tempo em que o serviço termina (*final\_ts*) e o tempo em que inicia (*initial\_ts*), valores obtidos da tabela *taxi\_services* através da correspondência do *taxi\_id*. Também obtivemos o tempo total de serviço de cada um dos táxis (*tempoTotal* presente em *dw\_taxi*), resultado da soma de todos os tempos de cada viagem, assim como o número de viagens que cada táxi realizou (*nViagens* presente em *dw\_taxi*) que resulta de um simples *count* das viagens realizadas por um determinado táxi, estas consultas são facilmente obtidas através da correspondência das chaves externas e primárias presentes na tabela de factos e na tabela dimensão, respetivamente.

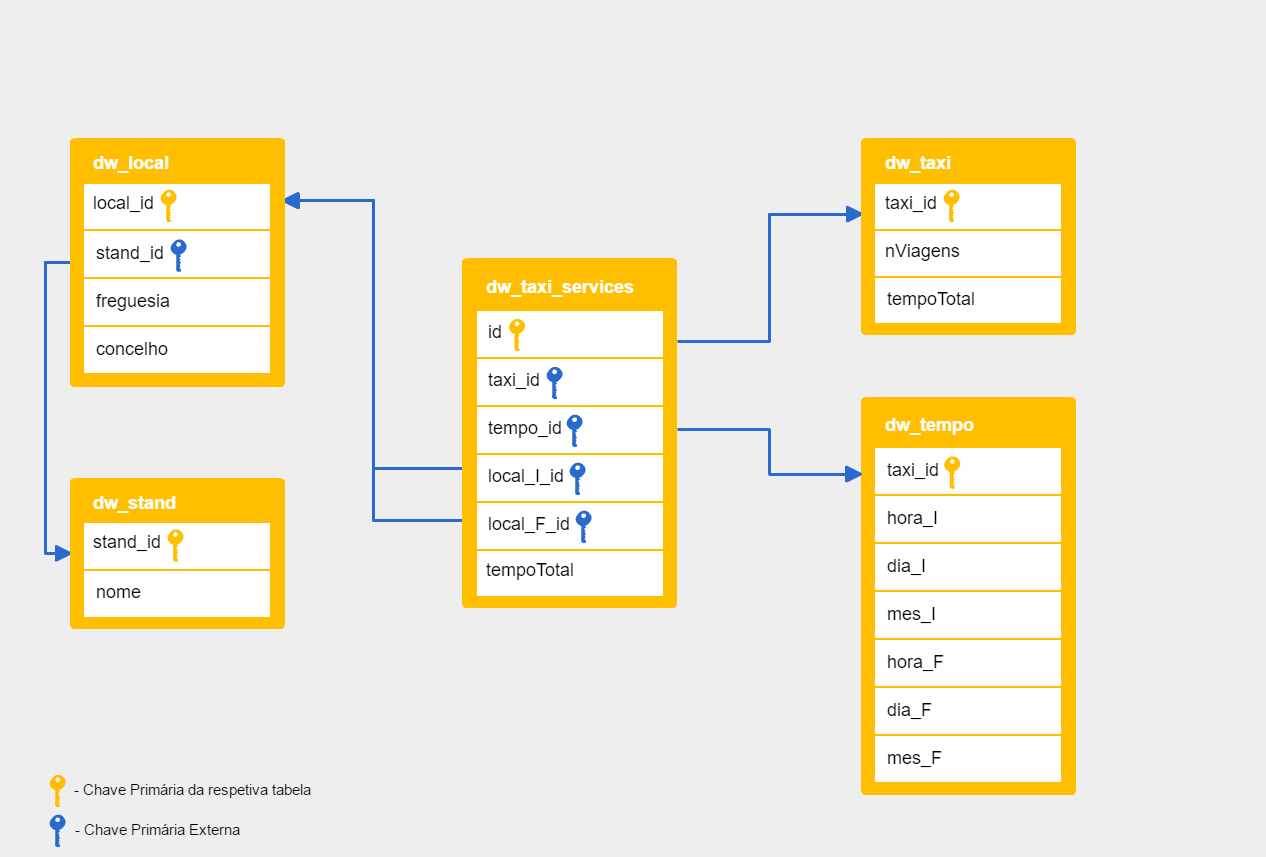
Trabalhando e manipulando os dados convenientemente, preenchemos finalmente as 5 tabelas com informação que nos conseguisse não só permitir detalhar e sumarizar o vasto conteúdo de mais e melhores formas, como também satisfazer todos os pedidos e intenções do projeto, nomeadamente não conter qualquer localização espacial no tipo de dados *geometry* ou variantes. No final obtivemos um modelo com o aspeto representado na figura 1.

Figura 1 – Data warehouse em esquema de floco de neve.

# Conteúdo

A tabela ***dw\_taxi\_services*** é a tabela central deste trabalho, (tabela de factos), pois é nela que está reunida toda a informação proveniente das outras tabelas associadas, (tabelas dimensão), a esta por chaves externas. Esta tabela contém informações sobre todos os serviços de táxis na zona do Porto representadas em 7 colunas: *id* do tipo *int*, *taxi\_id* do tipo *int*, que representa a identificação do táxi que realizou o serviço em causa, *tempo\_id* do tipo *int* que contém informação da duração do serviço, *local\_I\_id* do tipo *int*, identificador do ponto de partida, *local\_f\_ID* do tipo *int*, identificador do ponto de chegada, e *tempoTotal* do tipo *text* que indica a duração do serviço.

A tabela dimensão ***dw\_tempo***, como já mencionado, refere-se à duração de um serviço de táxi, esta informação é distribuída nas seguintes colunas que a compõe, *tempo\_id* do tipo *int*, chave primária que faz ligação à tabela *de factos*, *hora\_I* do tipo *text* que representa a hora de início do serviço, *dia\_I* do tipo *text* que representa o dia em que se iniciou o serviço, e o mês presente em *mes\_I* do tipo *text*, a *hora\_F* do tipo *text*, *dia\_F* do tipo *text* e *mes\_F* do tipo *text*, representam exatamente a mesma informação que as 3 colunas anteriores mas referente ao termino do serviço.

***dw\_taxi*,** tabela com a informação de cada táxi, possui apenas 3 colunas, *taxi\_id* do tipo *int*, chave primária que faz ligação à tabela *dw\_taxi\_services*, *nViagens* do tipo *int*, que como já referido representa o número total de viagens (serviços) que um dado táxi já realizou, e *tempoTotal* do tipo *int* que representa a duração de todos os serviços que o táxi realizou.

A tabela ***dw\_local*** para além de estar ligada à tabela *de factos* é também normalizada pela tabela de dimensão mais pequena *dw\_stand* através de chaves externas, esta tabela conta com 4 tipo de informações: chave primária (identificador) que se estende à tabela de factos presente na coluna *local\_id* do tipo *int*, referência à tabela *dw\_stand* na coluna *stand\_id* do tipo *int*, e informação sobre a freguesia e concelho desse local que estão presentes nas colunas *freguesia* e *conselho*, ambas do tipo *text*.

Por último, a tabela ***dw\_stand*** constituída por 2 colunas e ligada por uma chave externa à tabela *dw\_local*. A coluna *stand\_id* do tipo *int* é a chave primária e estende-se à tabela *dw\_local*, e a coluna *nome* do tipo *text* que indica o nome da praça de táxis em questão.

# Manipulação dos dados

Após termos definido bem as tabelas e respetivas dimensões começamos por associar/preencher os valores a/em cada coluna de acordo com o que foi mencionado acima.

dw\_stand:

Esta tabela foi praticamente uma exportação da tabela base *taxi\_stands*. Começamos por saber quantas linhas existiam, de seguida obtivemos os nomes das praças e os seus identificadores. Após estas consultas criamos um ciclo onde adicionamos à tabela essa informação.

**for i in range(0, nTuplos[0][0]):**

**cursor.execute("insert into dw\_stand (stand\_id, nome)**

**values (%s, %s)", (i+1, nomes[i][0]))**

dw\_local:

A tabela *dw\_local* foi uma das tabelas mais complexas para construir, uma vez que envolvia encontrar a freguesia e o conselho de cada uma das praças. Um problema inicial foi o tipo de dados em que estas informações estão, localização espacial da freguesia e do conselho em *geometry(MultiPolygon, 4326)* e localização espacial das praças em *geometry*.

O objetivo era verificar em que freguesias as praças estavam contidas, e após algumas investidas, encontramos a função ideal para resolver o nosso problema: **ST\_Within** que retorna **true** se um *geometry A* (*táxi\_*stands.*location*) estiver completamente dentro de uma *geometry B* (*cont\_freg\_v5.geom*).

Os resultados são obtidos iterativamente e inseridos após cada obtenção.

**cursor.execute("select freguesia from cont\_freg\_v5**

**where st\_within(%s, st\_astext(geom))**

**and distrito like 'PORTO'", (location[i][0],)**

dw\_tempo:

Aqui apenas tivemos de consultar o *initial\_ts* e o *final\_ts* da tabela *taxi\_services* e associar a cada tempo um identificador. Tivemos de realizar ainda uma conversão especial desses valores, pois estão no formato UNIX TIME na tabelas base, fizemos então consultas usando TIMESTAMP que nos permite converter esses valores para o formato pretendido, a *query* usada tem o seguinte aspeto:

**select TIMESTAMP 'epoch' + initial\_ts \* INTERVAL '1 second',**

**TIMESTAMP 'epoch' + final\_ts \* INTERVAL '1 second'**

**from taxi\_services**

O resultado obtido em determinada linha foi: “*2015-01-01 00:00:09 | 2015-01-01 00:08:01*”. Feito isto apenas “recolhemos” o mês, dia e hora referentes ao tempo de início e de fim do serviço, para isso acessamos a posição de cada campo na *string* obtida e associamos os valores aos respetivos campos (colunas).

dw\_taxi:

Para esta tabela começamos por ver quantos táxis diferentes existiam. E após a obtenção do número total de viagens realizadas por cada um dos táxis e a obtenção da duração total das mesmas, inserimos os valores nas respetivas colunas.

**#*nViagens:***

**select count(\*) from taxi\_services**

**where taxi\_id = %s", (str(taxi\_id[i][0]),)**

**#tempoTotal:**

**tempoTotal=0**

**for j in range(nViagens):**

**tempoTotal += tempoF[j][0] - tempoI[j][0]**

**tempoTotal = datetime.timedelta(seconds=tempoTotal)**

dw\_taxi\_services:

Nesta tabela a maior parte das colunas são identificadores externos, e para a preenchermos de forma concisa e correta tivemos de realizar algumas *querys* que referenciavam as tabelas envolvidas de forma a realizar o *matching* correto.

**#example for taxi\_id:**

**select taxi\_id from taxi\_services where st\_astext(initial\_point)**

**like %s and st\_astext(final\_point)**

**like %s", (pontoI[i][0], pontoF[i][0],)**

**#example for local\_I\_id:**

pontoI: **select st\_astext(initial\_point) from taxi\_services**

local\_id\_I: **select id, st\_distance(st\_astext(location), %s)**

**from taxi\_stands**

**order by st\_distance(st\_astext(location),**

**%s)", (pontoI[i][0], pontoI[i][0],)**