POLI
ESCOLA SUPERIOR
TECNOLOGIA
GESTÃO
TÉCNICO
GUARDA

Relatório de Base de Dados II

Data Warehouse

Curso:	Lic. Engenharia Informática
Unidade curricular:	Base de Dados II
Ano letivo:	2022/2023
Nº e nome de aluno:	Paulo Proença - 1704890
Docente:	José Carlos Coelho Martins Fonseca
Data:	5 de abril de 2023

Conteúdo

INTRODUÇÃO

Neste relatório pretende-se documentar os passos efetuados na elaboração de uma Data Warehouse a partir de um modelo de uma base de dados relacional.
Para o projeto foi fornecida a instância de uma base de dados relacional cujo tema é vendas.
Será através desta instância que será efetuado o "Reverse Engineering" sobre o qual irei trabalhar.
Uma vez que não se terá acesso ao utilizador ao qual a base de dados pertence, o OLTP_2023, o acesso à base de dados será feito através de um outro utilizador, o OLTP_Query que detém privilégios de consulta dessa mesma base de dados. Será depois criado um "Database Link" na conta do meu utilizador através do qual e usando o OLTP_Query poderei aceder à base de dados do utilizador OLTP_2023. Na conta do utilizador bdii_ei_1704890_EXTRA serão criadas duas tabelas extra que irão esta conectadas à base de dados do OLTP_2023 através de uma chave externa, sendo as mesmas relacionadas entre si, sendo necessário para tal criar outro "Database Link". Serão depois criados "SYNONYMS" das tabelas da base dados do OLTP_2023 para uma mais fácil utilização.

REVERSE ENGINEERING

Como já foi referido anteriormente será feito o "*Reverse Engineering*" da base de dados do utilizador OLTP_2023, e apresento em seguida o resultado dessa ação bem como o código da criação dos "*Database Link*" utilizados para aceder a essa mesma base de dados.

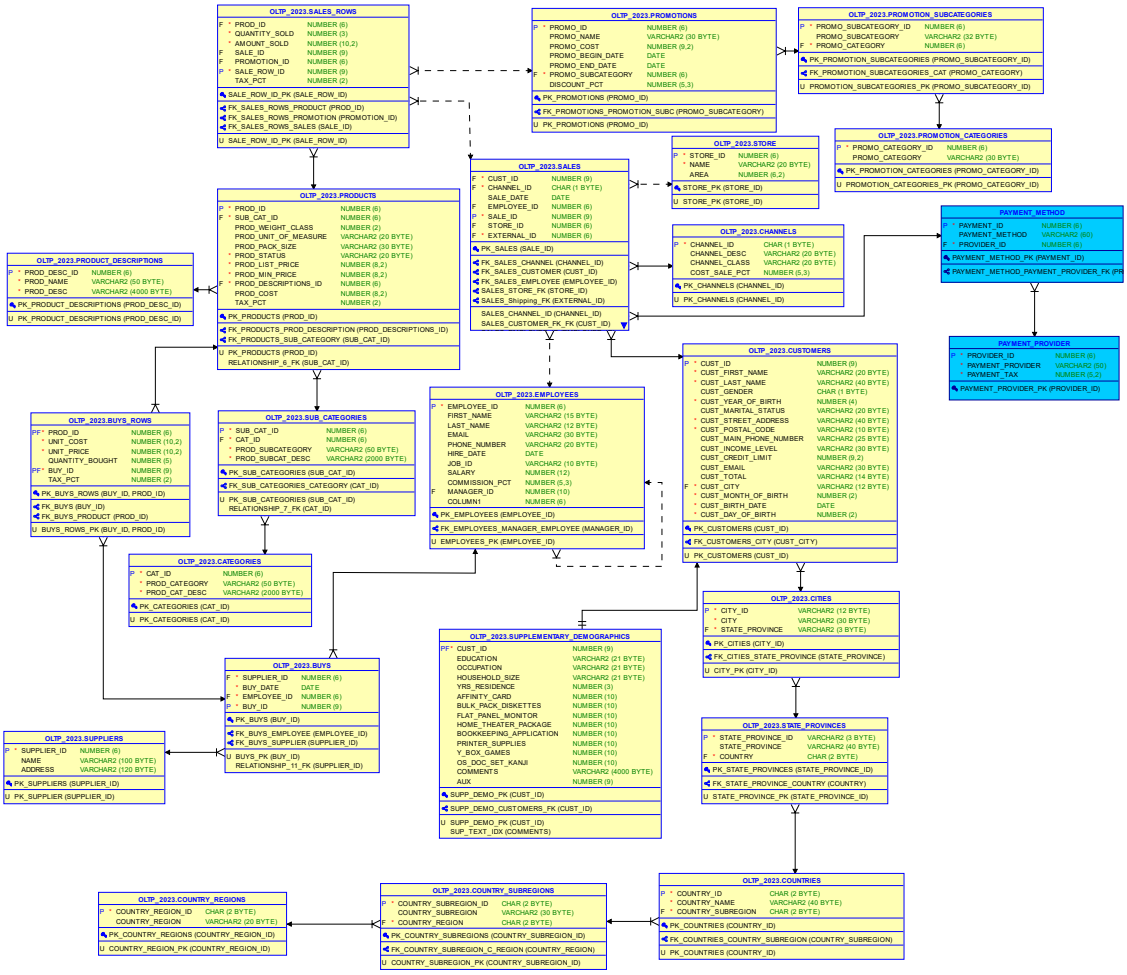

Figura 1 - Reverse Engineering com tabelas extra

```
CREATE DATABASE LINK oltp_2023 CONNECT TO oltp_query IDENTIFIED BY
oltp_query USING 'oltp';
```

```
CREATE DATABASE LINK sales_link CONNECT TO oltp_query IDENTIFIED BY
oltp_query USING 'oltp'.
```

CRIAÇÃO DE SYNONYMS

De seguida temos o script usado para a criação de "*synonyms*" para as contas bdii_ei_1704890 e bdii_ei_1704890_EXTRA:

bdii_ei_1704890:

```
SET HEADING OFF;
SET PAGESIZE 0;
SELECT 'CREATE SYNONYM ' || table_name || ' FOR oltp_2023.' || table_name
|| '@oltp_2023;' FROM all_tables@oltp_2023 WHERE owner = 'OLTP_2023';
CREATE SYNONYM BUYS FOR oltp_2023.BUYS@oltp_2023;
CREATE SYNONYM BUYS_ROWS FOR oltp_2023.BUYS_ROWS@oltp_2023;
CREATE SYNONYM CATEGORIES FOR oltp_2023.CATEGORIES@oltp_2023;
CREATE SYNONYM CHANNELS FOR oltp_2023.CHANNELS@oltp_2023;
CREATE SYNONYM CITIES FOR oltp_2023.CITIES@oltp_2023;
CREATE SYNONYM COUNTRIES FOR oltp_2023.COUNTRIES@oltp_2023;
CREATE SYNONYM COUNTRY_REGIONS FOR oltp_2023.COUNTRY_REGIONS@oltp_2023;
CREATE SYNONYM COUNTRY_SUBREGIONS FOR
oltp_2023.COUNTRY_SUBREGIONS@oltp_2023;
CREATE SYNONYM CUSTOMERS FOR oltp_2023.CUSTOMERS@oltp_2023;
CREATE SYNONYM STORE FOR oltp_2023.STORE@oltp_2023;
CREATE SYNONYM EMPLOYEES FOR oltp_2023.EMPLOYEES@oltp_2023;
CREATE SYNONYM PRODUCTS FOR oltp_2023.PRODUCTS@oltp_2023;
CREATE SYNONYM PRODUCT_DESCRIPTIONS FOR
oltp_2023.PRODUCT_DESCRIPTIONS@oltp_2023;
CREATE SYNONYM PROMOTIONS FOR oltp_2023.PROMOTIONS@oltp_2023;
CREATE SYNONYM PROMOTION_CATEGORIES FOR
oltp_2023.PROMOTION_CATEGORIES@oltp_2023;
CREATE SYNONYM PROMOTION_SUBCATEGORIES FOR
oltp_2023.PROMOTION_SUBCATEGORIES@oltp_2023;
CREATE SYNONYM SALES FOR oltp_2023.SALES@oltp_2023;
CREATE SYNONYM SALES_ROWS FOR oltp_2023.SALES_ROWS@oltp_2023;
CREATE SYNONYM STATE_PROVINCES FOR oltp_2023.STATE_PROVINCES@oltp_2023;
CREATE SYNONYM SUB_CATEGORIES FOR oltp_2023.SUB_CATEGORIES@oltp_2023;
CREATE SYNONYM SUPPLEMENTARY_DEMOGRAPHICS FOR
oltp_2023.SUPPLEMENTARY_DEMOGRAPHICS@oltp_2023;
CREATE SYNONYM SUPPLIERS FOR oltp_2023.SUPPLIERS@oltp_2023;
```

bdii_ei_1704890_EXTRA:

```
CREATE SYNONYM SALES FOR oltp_2023.SALES@sales_link;
```

INSERÇÃO DE REGISTOS NAS TABELAS EXTRA

Foram feitos alguns procedimentos para a inserção de dados nas tabelas extra, sendo a sua construção feita em pequenos passos que serão demonstrados a seguir:

1. **Criação do procedimento para inserção de dados na tabela Look_Up:**

```
CREATE SEQUENCE lookup_id START WITH 1;
CREATE OR REPLACE PROCEDURE insere_tabela_provider
IS
    v_date provider.contract_date%TYPE;
BEGIN
    FOR i IN 1..10
    LOOP
        v_date                                                    :=
to_date(trunc(dbms_random.value(to_char(to_date('15/01/1986','dd/mm/yy
yy'),'j'),
        to_char(to_date('23/05/2022','dd/mm/yyyy'),'j'))),'j');
        INSERT    INTO    provider    (provider_id,    payment_provider,
payment_tax,contract_date)
        VALUES
(lookup_id.nextVal,dbms_random.string('a',dbms_random.value(10,20)),
dbms_random.value(0,0.17),v_date);
    END LOOP;
   COMMIT;
END;
```

2. **Criação do procedimento para inserção de dados na tabela Principal:**

```
CREATE OR REPLACE PROCEDURE INSERT_p_method_principal IS
    v_max NUMBER(9);
    v_min NUMBER(9);
    max_ext NUMBER(9);
    min_ext NUMBER(9);
    v_random NUMBER(9);
    v_f_key NUMBER(9);
BEGIN
    SELECT MIN(external_id),MAX(external_id)
    INTO v_min, v_max
    FROM sales;
    FOR i IN v_min..(v_max) + 10
    LOOP
        SELECT count(*)
        INTO max_ext
        FROM provider;
        v_random := TRUNC(dbms_random.value(1, max_ext));
        SELECT provider_id
        INTO v_f_key
        FROM provider
        OFFSET v_random ROWS
        FETCH FIRST ROW ONLY;
        INSERT              INTO              payment_method(payment_id,
payment_method,card_type,paymemt_provider_id)
        VALUES
(i,dbms_random.string('a',20),dbms_random.string('a',20),v_f_key);
    END LOOP;
    COMMIT;
END;
```

3. **Criação do procedimento para apagar todos os registos das duas tabelas e executar os dois procedimentos anteriores:**

```
CREATE OR REPLACE PROCEDURE INSERT_tables_payment_methods
IS
BEGIN
    DELETE payment_method;
    DELETE provider;
    insere_tabela_provider;
    INSERT_p_method_principal;
END;
```

DATA WAREHOUSE

Um Data Warehouse é um sistema de armazenamento dados coleados a partir de várias fontes e que tem como objectivo ajudar os utilizadores numa tomada de decisão mais informada e completa, sendo utilizado para o efeito um sistema de integração de dados denominado ETL(Extract , Transform and Load), assim como uma modelagem dimensional, [1] sendo no fim implementada uma camada de visualização em que neste caso será implementada com recurso a uma ferramenta da Microsoft, o Power BI, sobre o qual me debruçarei mais adiante.

Para o nosso caso foi feito um estudo da Base de Dados Relacional e com base nela foram escolhidas tabelas para formarem 4 dimensões da Data Warehouse bem como a Granularidade da mesma. As escolhas foram as seguintes:

Modelo de Negócio:

- Venda de Vestuário e calçado

Granularidade:

- Métodos de Pagamento X Loja X Canais de Distribuição X Dia

Dimensões:

- Data
- Método de Pagamento
- Loja
- Canais de Distribuição

Seguiu-se depois a modelação dimensional, cujo resultado final passo a mostrar:

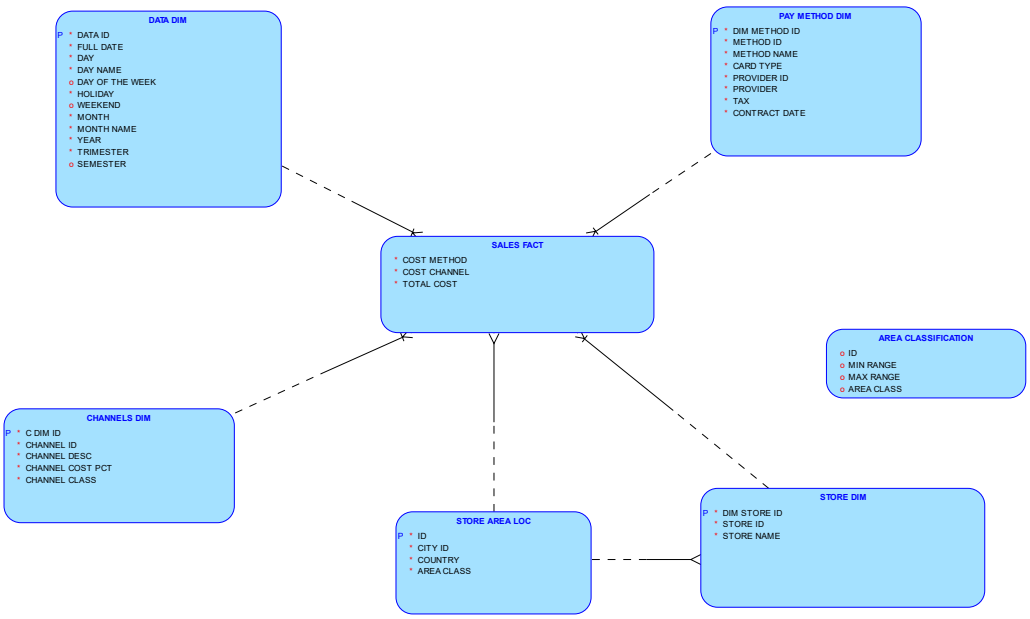

Figura 2 - Diagrama em Estrela da Data Warehouse

CÁLCULO DO ESPAÇO

Granularidade:

- Métodos de Pagamento X Loja X Canais de Distribuição X Dia

Métodos de Pagamento: 822
Lojas: 31
Canais de Distribuição: 5
Tempo: 3 anos

Tamanho médio dos registos: 8 atributos x 4 bytes = 32

Nº de registos de factos: 3 x 365 x 822 x 31 x 5 = 139 513 950

[2]**Tamanho aproximado da DW:** 32 x 139 513 950 = 4 464 446 400 ≈ 4,46 Gbs

INSERÇÃO DE REGISTOS NA DATA WAREHOUSE

De modo a facilitar a inserção de registos na data Warehouse, forma criados vários procedimentos e funções que permitiram que essas inserções fossem feitas de forma faseada de modo a permitir um melhor controlo sobre o que estava a acontecer, limitando assim também a probabilidade de acontecerem erros.

Procedimento para inserir registos na Channels_dim

```
--- procedimento para inserir registos na dimensao channels
CREATE OR REPLACE PROCEDURE etl_dim_channels_dim IS

    CURSOR c_dim IS
    SELECT
        channel_id,
        channel_desc,
        channel_class,
        cost_sale_pct
    FROM
        channels;
    v_count NUMBER(9) := 0;
BEGIN
    FOR v_rec_dim IN c_dim LOOP
        -- select que permite verificar se existe o id do registo que
estamos a tentar inserir
        SELECT
            COUNT(channel_id)
        INTO v_count
        FROM
            channels_dim
        WHERE
            channel_id = v_rec_dim.channel_id;
        -- condição que evita inserção de registos duplicados
        IF v_count = 0 THEN
            INSERT INTO channels_dim (
                c_dim_id,
                channel_id,
                channel_desc,
                channel_cost_pct,
                channel_class
            ) VALUES (
                seq_channels.NEXTVAL,
                v_rec_dim.channel_id,
                v_rec_dim.channel_desc,
                v_rec_dim.cost_sale_pct,
                v_rec_dim.channel_class
            );
        END IF;
    END LOOP;
END;
```

Procedimento para inserir registos na Data_dim

```
CREATE OR REPLACE PROCEDURE etl_dim_date IS
```

```
    v_end_day    VARCHAR2(2);
    v_data_id    data_dim.data_id%TYPE;
    v_full_date  VARCHAR2(30);
    v_semester   VARCHAR2(1);
    v_rows       NUMBER;
    v_max_date   DATE;
    v_min_date   DATE;
    v_m_holiday  VARCHAR2(1);
    v_is_weekend VARCHAR2(1);
BEGIN
    -- select para carregar a menor data e a maior data da tabela de
vendas
    SELECT
        MIN(sale_date),
        MAX(sale_date)
    INTO
        v_min_date,
        v_max_date
    FROM
        sales;
    -- loop para os anos
    FOR v_year IN to_char(v_min_date, 'yyyy')..to_char(v_max_date,
'yyyy') + 5 LOOP
        -- loop para os meses
        FOR v_month IN 1..12 LOOP
            --select para saber qual o ultimo dia do mes
            SELECT
                to_char(last_day(TO_DATE(v_month|| '/' || v_year,
'mm/yyyy')),'dd')
            INTO v_end_day
            FROM
                dual;
            --loop para os dias
            FOR v_day IN 1..v_end_day LOOP
                -- funçoes para criar id, para converter para data
completa,para calcular feriados e verificar se é fim de semana
                v_full_date := trim(to_char(v_day, '00'))
                               || trim(to_char(v_month, '00'))
                               || v_year;
                v_data_id := TO_NUMBER ( v_year
                                         || TRIM(to_char(v_month,
'00'))|| TRIM(to_char(v_day, '00')) );
                v_m_holiday := calc_holidays(TO_DATE(v_full_date,
'dd/mm/yyyy'), v_year, v_month, v_day);
                v_is_weekend :=
is_weekend(TO_NUMBER(to_char(TO_DATE(to_char(v_day, '00')

|| to_char(v_month, '00')

|| v_year,'dd/mm/yyyy'),'d')));
                -- select que verifica se ja existe o registo que
estamos a tentar inserir
                SELECT
                    COUNT(*)
                INTO v_rows
                FROM
                    data_dim
                WHERE
                    data_id = v_data_id;
```

```
                --verifica se nao existe esse registo e efetua o
insert
                IF v_rows = 0 THEN
                    INSERT INTO data_dim (
                        data_id,
                        full_date,
                        "DAY",
                        day_name,
                        day_of_the_week,
                        holiday,
                        weekend,
                        "MONTH",
                        month_name,
                        "YEAR",
                        trimester,
                        semester
                    ) VALUES (v_data_id,
                        TO_DATE(v_full_date, 'dd/mm/yyyy'),
                        v_day,
                        to_char(TO_DATE(to_char(v_day, '00')||
to_char(v_month, '00')|| v_year,'dd/mm/yyyy'),'DY'),
                        TO_NUMBER(to_char(TO_DATE(to_char(v_day,
'00')|| to_char(v_month, '00')|| v_year,'dd/mm/yyyy'),'d')),
                        v_m_holiday,
                        v_is_weekend,
                        v_month,
                        to_char(TO_DATE(to_char(v_day, '00')||
to_char(v_month, '00')|| v_year,'dd/mm/yyyy'),'MON'),
                        v_year,
                        calc_trimester(v_month),
                        calc_semester(v_month)
                    );
                    COMMIT;
                END IF;
            END LOOP;
        END LOOP;
    END LOOP;

END;
```

Neste procedimento foram criadas algumas funções para calcular alguns valores a inserir:

```
-- função que calcula o dia em que calha a pascoa
[3]create or replace function calc_pascoa (p_year number) return date
is
     MES number;
     DIA number;
     A   number;
     B   number;
     C   number;
     D   number;
     E   number;
     F   number;
     G   number;
     H   number;
     I   number;
     K   number;
     L   number;
     M   number;
     ANO NUMBER;
```

```
     DATA_PASCOA DATE;
BEGIN
    ANO:= p_year; /*ESTE PARAMETRO E O ANO PARA O QUAL SE DESEJA
CALCULAR A DATA DE PASCOA*/
    A := MOD(ANO, 19);
    B := TRUNC(ANO/100);
    C := MOD(ANO, 100);
    D := TRUNC(B/4);
    E := MOD(B, 4);
    F := TRUNC((B+8)/25);
    G := TRUNC((B-F+1)/3);
    H := MOD((19*A+B-D-G+15), 30);
    I := TRUNC(C/4);
    K := MOD(C, 4);
    L := MOD((32+2*E+2*I-H-K),7);
    M := TRUNC((A+11*H+22*L)/451);
    MES := TRUNC((H+L-7*M+114)/31);
    DIA := MOD((H+L-7*M+114), 31) + 1;
    DATA_PASCOA :=
TO_DATE(LPAD(DIA,2,'0')||'/'||LPAD(MES,2,'0')||'/'||ANO,
'DD/MM/YYYY');
    return data_pascoa;
 END;




-- Função que permite calcular os feriados
CREATE OR REPLACE FUNCTION calc_holidays (
    p_date  DATE,
    p_year  NUMBER,
    p_month NUMBER,
    p_day   VARCHAR2
) RETURN VARCHAR IS

    TYPE t_feriado_array IS
        TABLE OF VARCHAR2(10) INDEX BY BINARY_INTEGER;
    v_feriados t_feriado_array;
    v_month    VARCHAR(12);
    v_day      VARCHAR(12);
    is_holiday VARCHAR2(1) := 'N';
    easter     DATE;
    check_bool BOOLEAN := FALSE;
BEGIN
    v_month := trim(to_char(p_month, '00'));
    v_day := trim(to_char(TO_NUMBER(p_day), '00'));
    -- Array para guardar os feriados fixos
    v_feriados(1) := '01/01';
    v_feriados(2) := '25/04';
    v_feriados(3) := '01/05';
    v_feriados(4) := '10/06';
    v_feriados(5) := '15/08';
    v_feriados(6) := '05/10';
    v_feriados(7) := '01/11';
    v_feriados(8) := '01/12';
    v_feriados(9) := '08/12';
    v_feriados(10) := '25/12';
    easter := calc_pascoa(p_year); --- calcula o dia de Pascoa nesse
ano
     -- verifica se o dia é um feriado fixo
    FOR i IN 1..v_feriados.last LOOP
```

```
        IF v_feriados(i) = ( v_day|| '/' || v_month ) THEN
            is_holiday := 'Y';
            check_bool := TRUE;
        END IF;
    END LOOP;
    -- verifica se o dia é feriado móvel usando como base o dia da
Pascoa previamente calculado
    -- e verificando tambem se o dia já nao foi considerado feriado
fixo
    IF check_bool = false THEN
        CASE
            WHEN p_date = easter - 47 THEN
                is_holiday := 'Y';
            WHEN p_date = easter - 2 THEN
                is_holiday := 'Y';
            WHEN p_date = easter THEN
                is_holiday := 'Y';
            WHEN p_date = easter + 60 THEN
                is_holiday := 'Y';
            WHEN to_char(p_date, 'dd/mm') = '25/04' THEN
                is_holiday := 'Y';
            ELSE
                is_holiday := 'N';
        END CASE;
    END IF;
    RETURN is_holiday;
END;
-- função que verifica se é fim de semana
CREATE OR REPLACE FUNCTION is_weekend (
    p_day_num NUMBER
) RETURN VARCHAR2 IS
    v_is_weekend VARCHAR2(1);
BEGIN
    -- verifica se o numero do dia é 1 (Domingo) ou 7(Sábado)
    IF p_day_num = 1 OR p_day_num = 7 THEN
        v_is_weekend := 'Y';
    ELSE
        v_is_weekend := 'N';
    END IF;
    RETURN v_is_weekend;
END;

-- Função que calcula o Trimestre
CREATE OR REPLACE FUNCTION calc_trimester (
    p_month NUMBER
) RETURN NUMBER IS
    v_trimester NUMBER(1);
BEGIN
    IF p_month BETWEEN 1 AND 3 THEN
        v_trimester := 1;
    ELSIF p_month BETWEEN 4 AND 6 THEN
        v_trimester := 2;
    ELSIF p_month BETWEEN 7 AND 9 THEN
        v_trimester := 3;
    ELSE
        v_trimester := 4;
    END IF;
    RETURN v_trimester;
END;

---- função que permite calcular o semestre
```

```
CREATE OR REPLACE FUNCTION calc_semester (
    p_month NUMBER
) RETURN NUMBER IS
    v_semester NUMBER(1);
BEGIN
    IF p_month BETWEEN 1 AND 6 THEN
        v_semester := 1;
    ELSE
        v_semester := 2;
    END IF;
    RETURN v_semester;
END;
```

Procedimento para inserir registos na Payment_Method_dim

```
-- procedimento que insere registos na dimensao Payment_method_dim
-- desnormalizando as tabelas extra e lookup
CREATE OR REPLACE PROCEDURE etl_dim_pay_method_dim IS
BEGIN
    INSERT INTO pay_method_dim (
        dim_method_id,
        method_id,
        method_name,
        card_type,
        provider_id,
        provider,
        tax,
        contract_date
    )
        SELECT
            ( seq_pay_method.NEXTVAL ),
            payment_id,
            payment_method,
            card_type,
            provider_id,
            payment_provider,
            payment_tax,
            contract_date
        FROM
                pay_method
            INNER JOIN provider ON pay_method.payment_provider_id =
provider.provider_id
        WHERE
            payment_id NOT IN (
                SELECT
                    method_id
                FROM
```

```
                    pay_method_dim
            );
    COMMIT;
END;
```

Procedimento para inserir registos na dimensão Store_dim e na mini-dimensão Store_area_loc

```
--procedimento que permite efetuar a insersao simultanea na dimensao
na mini-dimensao
CREATE OR REPLACE PROCEDURE etl_dim_store_dim IS
    --inserir os registos num cursor
    CURSOR c_store IS
    SELECT
        store_id AS store_id,
        name,
        store_city,
        area
    FROM
        store;
    v_count     NUMBER(9);
    v_mini_id   store_area_loc.id%TYPE;
    nullhandler BOOLEAN := TRUE;
BEGIN
    FOR v_rec_store IN c_store LOOP
                --- verifica a existencia de um determinado registo
antes de inserir
        IF check_existence(v_rec_store.store_city, v_rec_store.area) =
false THEN
                    -- condição para adicionar um registo com o id = 0
de modo a tratar os nulls existentes na tabela OLTP
            IF nullhandler = true THEN
                INSERT INTO store_area_loc (
                    id,
                    city_id,
                    country,
                    area_class
                ) VALUES (
                    0,
                    'None',
```

```
                    'NA',
                    'Undefined'
                );
                INSERT INTO store_dim (
                    dim_store_id,
                    store_id,
                    store_name,
                    store_area_loc_id
                ) VALUES (
                    0,
                    0,
                    'Undefined',
                    0
                );
                nullhandler := FALSE;
            END IF;
                    --- utilizar um insert baseado em select para
adicionar registos a mini dimensao
            INSERT INTO store_area_loc (
                id,
                city_id,
                country,
                area_class
            )
                SELECT
                    seq_area_loc.NEXTVAL,
                    v_rec_store.store_city,
                    countries.country_id,
                    calc_range(v_rec_store.area)
                FROM
                         cities
                    INNER JOIN state_provinces ON
cities.state_province = state_provinces.state_province_id
                    INNER JOIN countries ON state_provinces.country =
countries.country_id
                WHERE
                    ( v_rec_store.store_city,
calc_range(v_rec_store.area) ) NOT IN (
                        SELECT
                            city_id, area_class
                        FROM
                            store_area_loc
                    )
                    AND ( v_rec_store.store_city = cities.city_id );
        END IF;
                -- select que devolve o numero de registos existentes
de um determinado id de forma a evitar duplicações
        SELECT
            COUNT(dim_store_id)
        INTO v_count
        FROM
            store_dim
        WHERE
            v_rec_store.store_id = store_dim.store_id;
                -- condição para evitar valores duplicados na dimensao
store
        IF v_count = 0 THEN
            INSERT INTO store_dim (
                dim_store_id,
                store_id,
                store_name,
```

```
                store_area_loc_id
            ) VALUES (
                seq_store.NEXTVAL,
                v_rec_store.store_id,
                v_rec_store.name,
                return_mini_id(v_rec_store.store_city,
v_rec_store.area)
            );
        END IF;
    END LOOP;
    COMMIT;
END;
```

Foram utilizadas as seguintes funções no procedimento:

```
--funçao que verifica a existencia de registos na na mini dimensao
CREATE OR REPLACE FUNCTION check_existence (
    p_city VARCHAR2,
    p_area NUMBER
) RETURN BOOLEAN IS
    v_exists BOOLEAN := FALSE;
    v_count  NUMBER(9);
BEGIN
    SELECT
        COUNT(return_mini_id(p_city, p_area))
    INTO v_count
    FROM
        dual;
    IF v_count = 1 THEN
        v_exists := TRUE;
    END IF;
    RETURN v_exists;
END;


-- função que permite associar o valor da area a uma gama de valores
-- com base numa tabela auxiliar
CREATE OR REPLACE FUNCTION calc_range (
    p_area NUMBER
) RETURN VARCHAR2 IS
    v_class area_classification.area_class%TYPE;
BEGIN
    SELECT
        area_class
    INTO v_class
    FROM
        area_classification
    WHERE
        p_area BETWEEN min_range AND max_range;
    RETURN v_class;
END;

-- função que retorna o valor do id da mini dimensão
CREATE OR REPLACE FUNCTION return_mini_id (
    p_city VARCHAR2,
    p_area NUMBER
) RETURN NUMBER IS
    v_mini_id    NUMBER(15);
    v_area_class VARCHAR2(40);
BEGIN
    v_area_class := calc_range(p_area); -- verifica qual a
classificação da area para
                                        -- que possa efetuar a
comparação na mini dimensao
    -- verifica a existencia do registo e devolve o id do mesmo
    SELECT
        store_area_loc.id
    INTO v_mini_id
    FROM
        store_area_loc
    WHERE
            city_id = p_city
        AND area_class = v_area_class;
    RETURN v_mini_id;
END;
```

Procedimento para inserir registos na tabela de factos

```
--- procedimento para inserir registos na tabela de factos
CREATE OR REPLACE PROCEDURE etl_fact_sales_fact IS
-- cursor baseado num select que vai efetuar os calculos dos factos
agrupando-os
-- pela granularidae adequada
    CURSOR c_cursor IS
    SELECT
        trunc(s.sale_date) AS sale_date,
        nvl(s.store_id, 0) AS store_id,
        nvl(s.external_id, 0) AS external_id,
        s.channel_id,
        SUM(pv.payment_tax) AS cost_method,
        ( SUM(sr.amount_sold) * cn.cost_sale_pct ) AS cost_channel,
        ( SUM(pv.payment_tax) + ( SUM(sr.amount_sold) *
cn.cost_sale_pct ) ) AS total_cost
    FROM
             sales s
        JOIN sales_rows sr ON s.sale_id = sr.sale_id
        JOIN channels   cn ON s.channel_id = cn.channel_id
        JOIN pay_method pm ON nvl(s.external_id, 0) = pm.payment_id
        JOIN provider   pv ON pm.payment_provider_id = pv.provider_id
    GROUP BY
        trunc(s.sale_date),
        nvl(s.store_id, 0),
        nvl(s.external_id, 0),
        s.channel_id,
        cost_sale_pct;
    -- record que vai retornar o id da store_dim e da mini dimensao em
simultaneo
    v_my_rec my_record_type := my_record_type(0, 0);
    v_count  NUMBER(9) := 1;
BEGIN
    FOR v_rec_fact IN c_cursor LOOP
        -- funçao que permite retornar os ids da dim store e a da sua
mini dimensão
        -- fazendo uso do record criado para o efeito
        v_my_rec := getstoredimid(v_rec_fact.store_id);
        -- select que vai servir para verificar a exitencia do registo
antes de o inserir de forma a evitar duplicados
        SELECT
            COUNT(*)
        INTO v_count
        FROM
            sales_fact
        WHERE
        -- foram usadas funçoes para retornar o id das dims
                data_dim_data_id = getdatedimid(v_rec_fact.sale_date)
            AND channels_dim_c_dim_id =
getchanneldimid(v_rec_fact.channel_id)
            AND pay_method_dim_dim_method_id =
getpaymethoddimid(v_rec_fact.external_id)
            AND store_dim_dim_store_id = v_my_rec.v_id_dim;
        -- condiçao que verifica a existencia de duplicados
        IF v_count = 0 THEN
            INSERT INTO sales_fact (
                data_dim_data_id,
                channels_dim_c_dim_id,
```

```
                pay_method_dim_dim_method_id,
                store_dim_dim_store_id,
                store_area_loc_id,
                cost_method,
                cost_channel,
                total_cost
            ) VALUES (
                getdatedimid(v_rec_fact.sale_date),
                getchanneldimid(v_rec_fact.channel_id),
                getpaymethoddimid(v_rec_fact.external_id),
                v_my_rec.v_id_dim,
                v_my_rec.v_id_mini,
                v_rec_fact.cost_method,
                v_rec_fact.cost_channel,
                v_rec_fact.total_cost
            );
            --- condição que vai efetuar commit do registos inseridos
as cada 10000 rows
            --- de forma a prevenir eventuais perdas de registos caso
exista algum erro ou problema na BD
            IF MOD(c_cursor%rowcount, 10000) = 0 THEN
                COMMIT;
            END IF;
        END IF;
    END LOOP;
    COMMIT;
END;
```

Record type criado para a devolução simultânea de dois ids

```
-- record type criado especificamente para que possam
-- ser devolvidos simultaneamente os ids da dimensao e da mini
dimensao
CREATE OR REPLACE TYPE my_record_type IS OBJECT (
        v_id_dim  NUMBER(9),
        v_id_mini NUMBER(9)
);
```

Funções utilizadas no procedimento que efectua inserção de registos na tabela de factos:

```
-- funçao que ira retornar os o id da dim e a mini dim em simultaneo
utilizando para isso
-- um user record type criado para o efeito
CREATE OR REPLACE FUNCTION getstoredimid (
    p_id store.store_id%TYPE
) RETURN my_record_type IS
    v_rec my_record_type := my_record_type(0, 0); -- inicialização do
record
BEGIN
    SELECT
        dim_store_id,
        store_area_loc_id
    INTO
        v_rec.v_id_dim,
        v_rec.v_id_mini -- insere os ids nos campos correpondentes no
record
    FROM
        store_dim
    WHERE
        store_id = nvl(p_id, 0);
    RETURN v_rec;
END;


-- funçao que devolve o id da dimensao data
CREATE OR REPLACE FUNCTION getdatedimid (
    p_date sales.sale_date%TYPE
) RETURN NUMBER IS
    v_id   NUMERIC(9);
    v_date NUMERIC(9);
BEGIN
 -- concatena a data para obter um formato igual ao do id da dimensao
data para efetuar a comparação
    SELECT
        to_char(p_date, 'yyyy') || to_char(p_date, 'mm') ||
to_char(p_date, 'dd')
    INTO v_date
    FROM
        dual;
    SELECT
        data_id
    INTO v_id
    FROM
        data_dim
    WHERE
        data_id = v_date;
    RETURN v_id;
END;
```

```
-- função que devolve o id da dimensao Channels
CREATE OR REPLACE FUNCTION getchanneldimid (
    p_id channels.channel_id%TYPE
) RETURN VARCHAR2 IS
    v_id channels_dim.c_dim_id%TYPE;
BEGIN
    SELECT
        c_dim_id
    INTO v_id
    FROM
        channels_dim
    WHERE
        channel_id = p_id;
    RETURN v_id;
END;


-- funçao que retorna o id da dimensao baseada nas tabelas extra e
lookup
CREATE OR REPLACE FUNCTION getpaymethoddimid (
    p_id pay_method.payment_id%TYPE
) RETURN NUMBER IS
    v_id pay_method_dim.method_id%TYPE;
BEGIN
    SELECT
        dim_method_id
    INTO v_id
    FROM
        pay_method_dim
    WHERE
        method_id = p_id;
    RETURN v_id;
END;
```

Procedimento principal

Por fim temos um procedimento principal que executa todos os outros na ordem adequada de forma a termos uma melhor organização de código.

```
      CREATE OR REPLACE PROCEDURE etl_principal IS
BEGIN
    DELETE sales_fact;
    DELETE pay_method_dim;
    DELETE channels_dim;
    DELETE store_dim;
    DELETE store_area_loc;
    DELETE data_dim;

    etl_dim_pay_method_dim;
    etl_dim_channels_dim;
    etl_dim_store_dim;
    etl_dim_date;
    etl_fact_sales_fact
END;
```

```
PL/SQL procedure successfully completed.

Elapsed: 00:02:52.291
```

Figura 3 - Tempo de execução do procedimento final

POWER BI

A ferramenta a utilizar neste projeto para a análise visual de dados é o Power BI. Esta ferramenta de analise de negócio tem valiosos recursos de visualização que permitem ao utilizador uma maior facilidade na análise de dados do seu negócio. O Power BI permite importar dados de várias fontes e transformá-los, sendo depois criadas Views interativas. [4]

Model View

Aqui temos o Model View que traduz o Star Scheme utilizado na modelação da Data Warehouse, no qual foram alterados os nomes das tabelas para uma melhor interpretação sobre o que de facto elas designam.

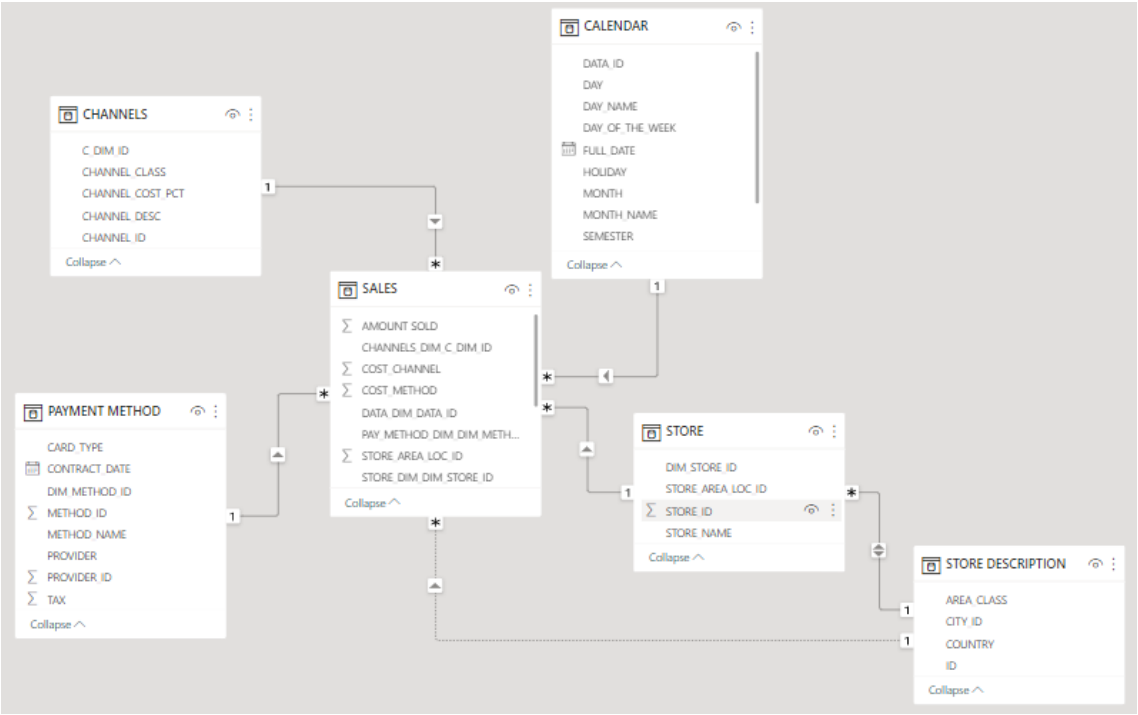

Figura 4 - Power BI model view

Foi necessário corrigir alguns pontos, nomeadamente o tipo de relação que estava erradamente como 1:1 em vez m:1 entre as tabelas Store e Store Description, bem como definir qual o caminho a seguir até a tabela Sales, de forma a evitar um relacionamento circular, sendo que neste caso colocou-se activa a relação da tabela Store.

Power Query

Como já foi referido anteriormente, o Power BI tem recursos e ferramentas bastante poderosos sendo um deles o Power Query, já utilizado no Excel. Foi recorrendo a esta ferramenta que foi criada uma coluna nova na tabela Sales que através do valor do campo Total Cost e do campo Channel_Cost_Pct da tabela Channels permite calcular o valor Amount Sold.

```
= Table.AddColumn(SALES_FACT1, "AMOUNT SOLD", each [TOTAL_COST]/[BDII_EI_1704890.CHANNELS_DIM][CHANNEL_COST_PCT])
```

	A_DIM	BDII_EI_1704890.PAY_METHOD_...	BDII_EI_1704890.STORE_AREA_L...	BDII_EI_1704890.STORE_DIM	AMOUNT SOLD
1		Value	Value	Value	1222,324942
2		Value	Value	Value	8702,999587
3		Value	Value	Value	63,49999698
4		Value	Value	Value	6390,999696
5		Value	Value	Value	2460,999883
6		Value	Value	Value	835,4999603
7		Value	Value	Value	1080,499949
8		Value	Value	Value	1127,249946
9		Value	Value	Value	1634,999922
10		Value	Value	Value	5089,099758

Figura 5 - Tabela Amount Sold criada usando o Power Query

Q&A

Foram criadas 5 páginas nas quais vão ser criadas Views para ajudar o utilizador a analisar os seus dados de negócio. A Primeira dessas Views é a Q&A. O Power BI conta com uma funcionalidade denominada Q&A que permite criar rapidamente uma View efetuando questões usando linguagem natural. Aqui está o exemplo no qual foram efetuadas duas questões sendo que os resultados obtidos servem para ter uma vista comparativa da média do valor de vendas com as médias dos custos.

Figura 6 - Página Q&A

Overview

A segunda página do relatório mostra uma vista geral de algumas medidas consideradas relevantes para a análise por parte do utilizador. Além das views gráficas e tabelas foram colocados cards com os totais e um slider com a data que iram estar presentes nas restantes páginas.

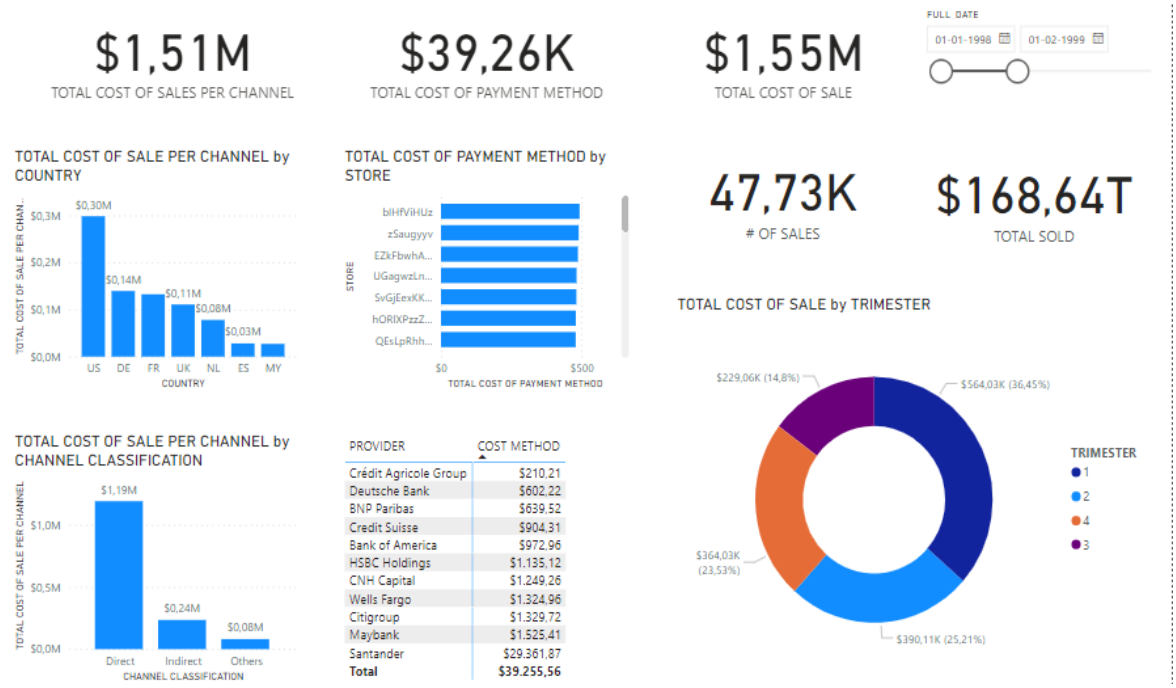

Figura 7 - Página Overview

Foram também adicionados filtros associados não apenas a esta pagina mas também alguns que serão possíveis aplicar a todas as outras.

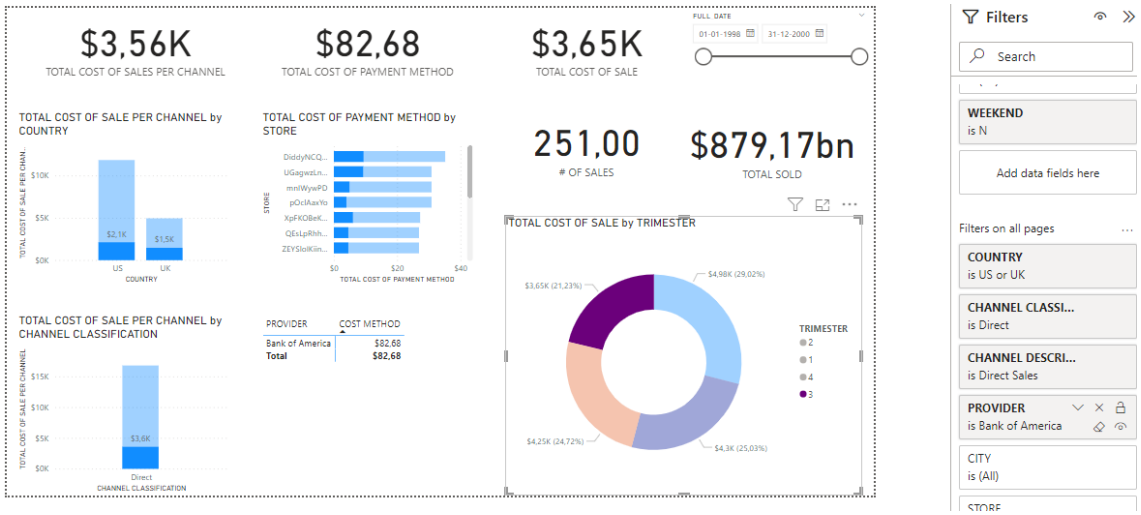

Figura 8 - Janela Overview com filtros

Channels, Store e Payment Method

Além das janelas anteriormente referidas foram criadas uma para cada Dimensão da Data Warehouse, exceptuando a Dimensão referente à data. Começamos então pela Channels.

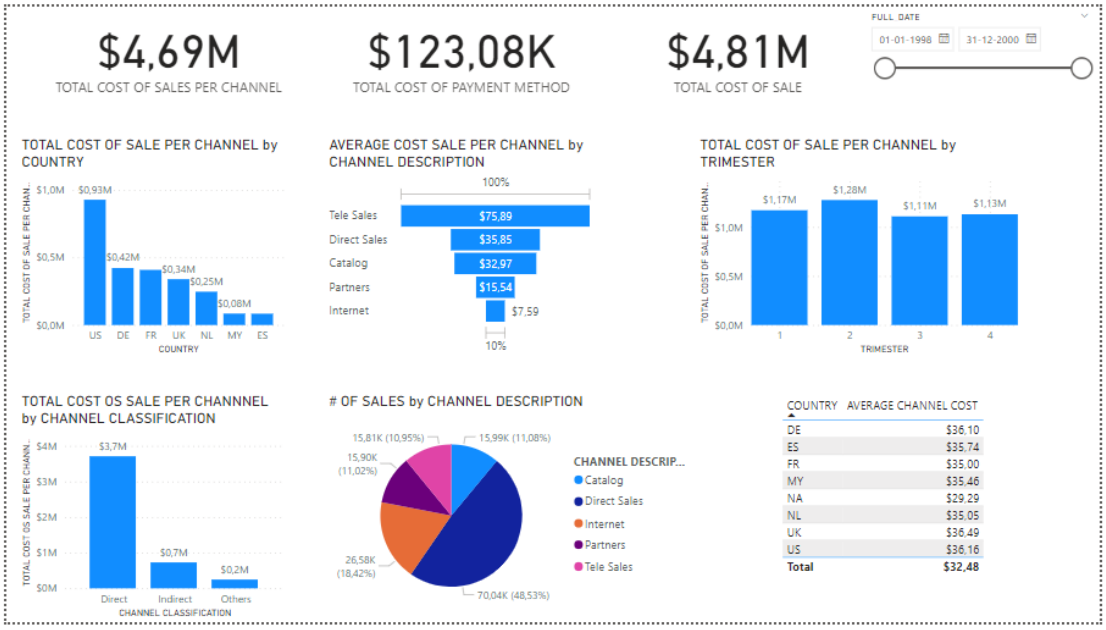

Figura 9 - Página Channels

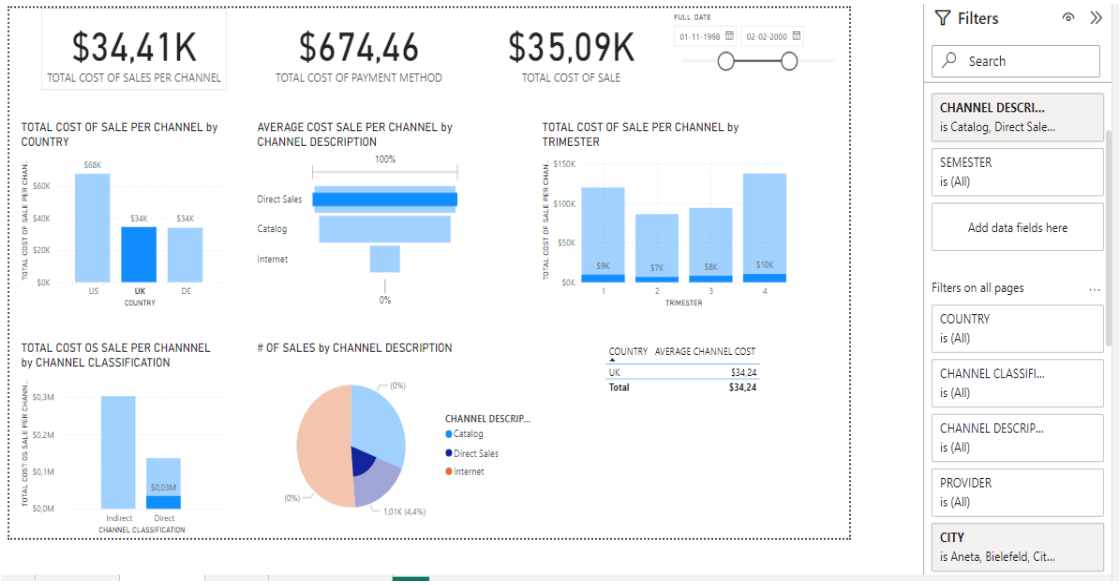

Figura 10 - Página Channels

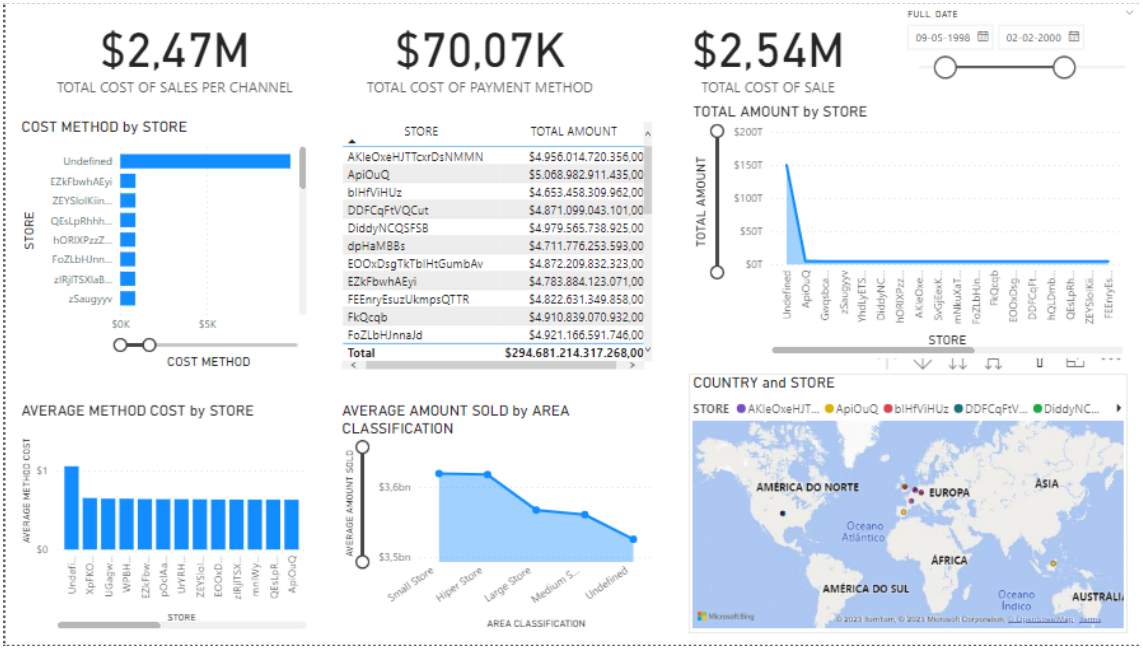

Figura 11 – Página Store

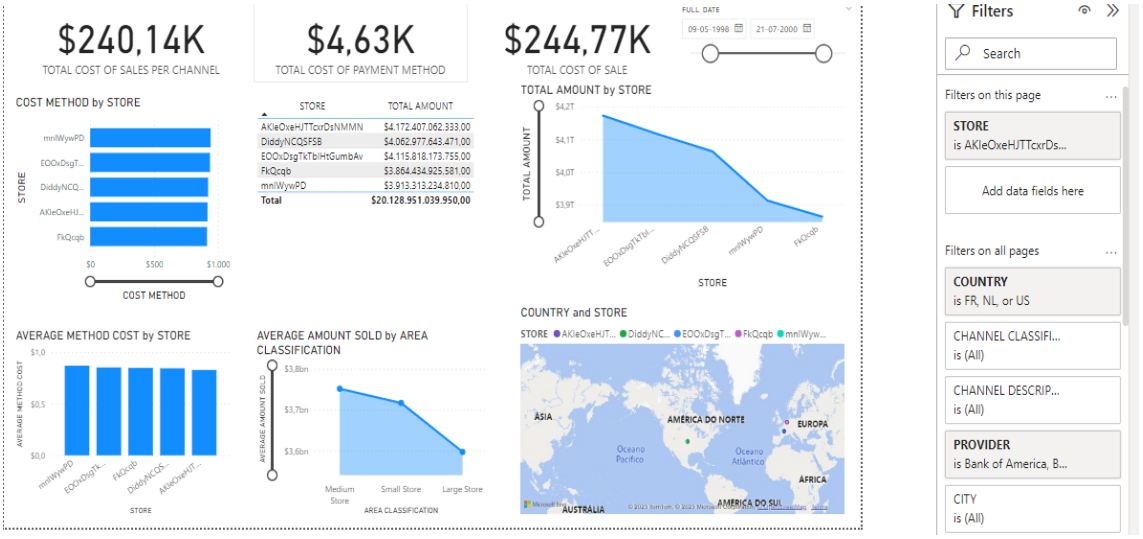

Figura 12 - Página store com filtros

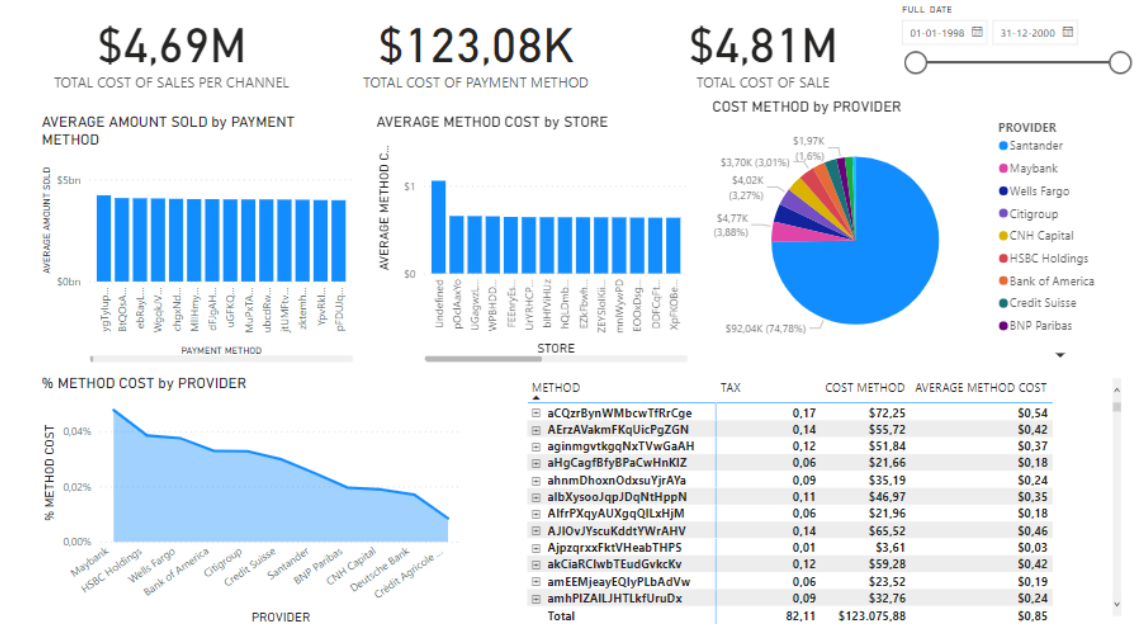

Figura 13 - Página Payment Method

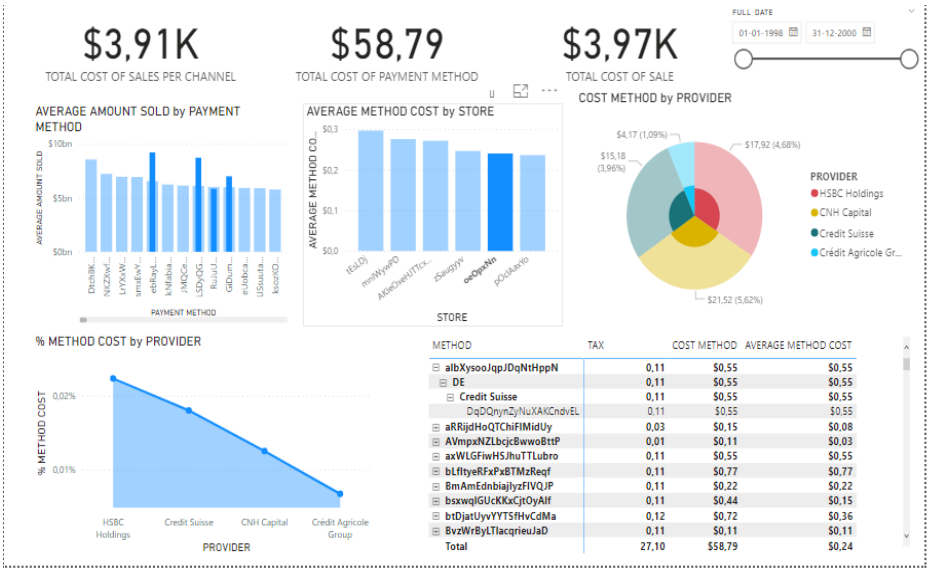

Figura 14 - Página Payment Method com filtros

CONCLUSÃO

Posso concluir que a criação de uma Data Warehouse é algo que requer bastante trabalho e minuciosidade, e que está em constante mutação ao longo projeto. Mas depois de concluído e aliado à ferramenta Power BI é uma ferramenta de enorme importância na gestão de qualquer negócio.
O projeto em si foi bastante desafiante devido a vários factores, e ajudou a perceber o impacto que todas as nossas acções podem ter no sistema.
De referir que os valores obtidos na tabela de factos deveriam ser maiores, mas devido a um erro que não me foi possível localizar o procedimento apenas insere 144320 registos aos invés dos 1329703 que deveria conter.

BIBLIOGRAFIA

[1] P. Potineni, "Data Warehousing Guide," [Online]. Available: https://docs.oracle.com/en/database/oracle/oracle-database/19/dwhsg/database-data-warehousing-guide.pdf.

[2] (. José Fonseca, "Apontamentos do Docente".

[3] Wikipédia, "Cálculo da Páscoa," [Online]. Available: https://pt.wikipedia.org/wiki/C%C3%A1lculo_da_P%C3%A1scoa.

[4] B. Costa, "2023-05-04 BI e PowerBI por Bruno Costa da CounTLines".

ANEXOS

Codigo DDL para a criação das tabelas extra

```
CREATE TABLE provider (
    provider_id      NUMBER(6) NOT NULL,
    payment_provider VARCHAR2(50) NOT NULL,
    payment_tax      NUMBER(5, 2) NOT NULL,
    contract_date    DATE NOT NULL
);
ALTER TABLE provider ADD CONSTRAINT provider_pk PRIMARY KEY ( provider_id
);

CREATE TABLE payment_method (
    payment_id          NUMBER(6) NOT NULL,
    payment_method      VARCHAR2(60) NOT NULL,
    card_type           VARCHAR2(60),
    payment_provider_id NUMBER(6) NOT NULL
);
ALTER TABLE payment_method ADD CONSTRAINT payment_method_pk PRIMARY KEY
( payment_id );
ALTER TABLE payment_method
    ADD CONSTRAINT payment_provider_fk FOREIGN
```

Código DDL para a criação das tabelas da "*Data Warehouse*"

```
DROP TABLE channels_dim CASCADE CONSTRAINTS;
DROP TABLE data_dim CASCADE CONSTRAINTS;
DROP TABLE pay_method_dim CASCADE CONSTRAINTS;
DROP TABLE sales_fact CASCADE CONSTRAINTS;
DROP TABLE store_area_loc CASCADE CONSTRAINTS;
DROP TABLE store_dim     CASCADE CONSTRAINTS;

-- predefined type, no DDL - MDSYS.SDO_GEOMETRY

-- predefined type, no DDL - XMLTYPE

CREATE TABLE channels_dim (
    c_dim_id         NUMBER(6) NOT NULL,
    channel_id       VARCHAR2(1) NOT NULL,
    channel_desc     VARCHAR2(60) NOT NULL,
    channel_cost_pct NUMBER(5, 3) NOT NULL,
    channel_class    VARCHAR2(60) NOT NULL
);

ALTER TABLE channels_dim ADD CONSTRAINT channels_dim_pk PRIMARY KEY (
c_dim_id );
```

```
CREATE TABLE data_dim (
    data_id         NUMBER(6) NOT NULL,
    full_date       DATE NOT NULL,
    day             NUMBER(2) NOT NULL,
    day_name        VARCHAR2(20) NOT NULL,
    day_of_the_week NUMBER(1),
    holiday         NUMBER NOT NULL,
    weekend         NUMBER,
    month           NUMBER(2) NOT NULL,
    month_name      VARCHAR2(20) NOT NULL,
    year            NUMBER(4) NOT NULL,
    trimester       NUMBER(1) NOT NULL,
    semester        NUMBER(1)
);

ALTER TABLE data_dim ADD CONSTRAINT data_dim_pk PRIMARY KEY ( data_id
);

CREATE TABLE pay_method_dim (
    dim_method_id NUMBER(6) NOT NULL,
    method_id     NUMBER(6) NOT NULL,
    method_name   VARCHAR2(60) NOT NULL,
    card_type     VARCHAR2(40) NOT NULL,
    provider_id   NUMBER(6) NOT NULL,
    provider      VARCHAR2(50) NOT NULL,
    tax           NUMBER(5, 2) NOT NULL,
    contract_date DATE NOT NULL
);

ALTER TABLE pay_method_dim ADD CONSTRAINT pay_method_dim_pk PRIMARY KEY
( dim_method_id );

CREATE TABLE sales_fact (
    data_dim_data_id             NUMBER(6) NOT NULL,
    channels_dim_c_dim_id        NUMBER(6) NOT NULL,
    pay_method_dim_dim_method_id NUMBER(6) NOT NULL,
    store_dim_dim_store_id       NUMBER(6) NOT NULL,
    store_area_loc_id            NUMBER(6) NOT NULL,
    cost_method                  NUMBER(9, 2) NOT NULL,
    used_method                  NUMBER(9, 0) NOT NULL,
    total_cost                   NUMBER(9, 2) NOT NULL
);

ALTER TABLE sales_fact
    ADD CONSTRAINT sales_fact_pk PRIMARY KEY ( data_dim_data_id,
                                               channels_dim_c_dim_id,

pay_method_dim_dim_method_id,
                                               store_dim_dim_store_id
);

CREATE TABLE store_area_loc (
    id         NUMBER(6) NOT NULL,
    city_id    VARCHAR2(40) NOT NULL,
    country    VARCHAR2(2) NOT NULL,
    area_class VARCHAR2(20) NOT NULL
);

ALTER TABLE store_area_loc ADD CONSTRAINT store_area_loc_pk PRIMARY KEY
( id );
```

```
CREATE TABLE store_dim (
    dim_store_id      NUMBER(6) NOT NULL,
    store_id          NUMBER(6) NOT NULL,
    store_name        VARCHAR2(60) NOT NULL,
    store_area_loc_id NUMBER(6) NOT NULL
);

ALTER TABLE store_dim ADD CONSTRAINT store_dim_pk PRIMARY KEY (
dim_store_id );

ALTER TABLE sales_fact
    ADD CONSTRAINT sales_fact_channels_dim_fk FOREIGN KEY (
channels_dim_c_dim_id )
        REFERENCES channels_dim ( c_dim_id );

ALTER TABLE sales_fact
    ADD CONSTRAINT sales_fact_data_dim_fk FOREIGN KEY ( data_dim_data_id
)
        REFERENCES data_dim ( data_id );

ALTER TABLE sales_fact
    ADD CONSTRAINT sales_fact_pay_method_dim_fk FOREIGN KEY (
pay_method_dim_dim_method_id )
        REFERENCES pay_method_dim ( dim_method_id );

ALTER TABLE sales_fact
    ADD CONSTRAINT sales_fact_store_area_loc_fk FOREIGN KEY (
store_area_loc_id )
        REFERENCES store_area_loc ( id );

ALTER TABLE sales_fact
    ADD CONSTRAINT sales_fact_store_dim_fk FOREIGN KEY (
store_dim_dim_store_id )
        REFERENCES store_dim ( dim_store_id );

ALTER TABLE store_dim
    ADD CONSTRAINT store_dim_store_area_loc_fk FOREIGN KEY (
store_area_loc_id )
        REFERENCES store_area_loc ( id );
```

Código de criação de "*Sequences* "para as dimensões

```
CREATE SEQUENCE seq_channels;
CREATE SEQUENCE seq_store;
CREATE SEQUENCE seq_pay_method;
CREATE SEQUENCE seq_area_loc;
```