

PostgreSQL Extract, Transform Load project

Created a data pipeline that transforms input sales and product datasets into a star schema ready for analysis and querying.

Project Includes the following

- Raw data loading
- Staging, transforming and validating data
- Warehouse, loading data into star schema data model.
- ETL scripts, fully automated so can easily convert new raw data into ready to query tables
- Joins

Tools used:

- PostgreSQL Database
- pgAdmin 4
- Vscode (with many extensions)
- SQL
- Python
- Tableau

Stage 1, create tables and schemas

Schema creation

```
CREATE TABLE warehouse.dim_customer (
    customer_id SERIAL PRIMARY KEY,
    name TEXT,
    city TEXT,
    zip INT
);

CREATE TABLE warehouse.dim_product (
    product_id SERIAL PRIMARY KEY,
    name TEXT, -- unique here would have been approach
    price NUMERIC,
    category TEXT
);

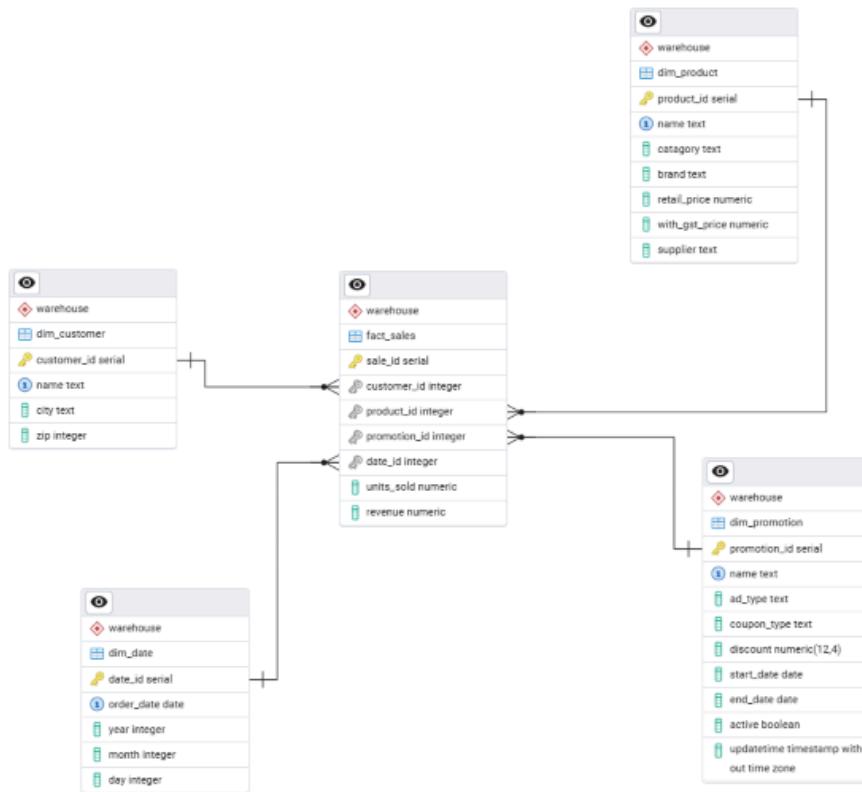
CREATE TABLE warehouse.dim_date (
    date_id SERIAL PRIMARY KEY,
    order_date Date,
    year int,
    month int,
    day INT
);

CREATE TABLE warehouse.dim_promotion (
    promotion_id SERIAL PRIMARY KEY,
    name TEXT,
    ad_type TEXT,
    coupon_type TEXT,
    discount DECIMAL(12,4),
    start_date DATE,
    end_date DATE,
    Active BOOLEAN,
    UpdateTime TIMESTAMP
);

CREATE TABLE warehouse.fact_sales (
    sale_id SERIAL PRIMARY KEY,
    customer_id INT REFERENCES warehouse.dim_customer(customer_id),
    product_id INT REFERENCES warehouse.dim_product(product_id),
    promotion_id INT REFERENCES warehouse.dim_promotion(promotion_id),
    date_id INT REFERENCES warehouse.dim_date(date_id),
    units_sold NUMERIC,
    revenue NUMERIC
);
```

File saved successfully.

Entity Relationship Diagram below after creating tables.



Stage 2, load raw data into tables

Query	Query History	Query	Query History
<pre> 1 2 v CREATE OR REPLACE FUNCTION etl.load_raw_sales() 3 RETURNS void 4 LANGUAGE 'sql' 5 COST 100 6 VOLATILE PARALLEL UNSAFE 7 AS \$BODY\$</pre>		<pre> 1 2 v CREATE OR REPLACE FUNCTION etl.load_raw_products() 3 RETURNS void 4 LANGUAGE 'sql' 5 COST 100 6 VOLATILE PARALLEL UNSAFE 7 AS \$BODY\$</pre>	
<pre> 8 9 DROP TABLE IF EXISTS raw.sales_raw;</pre>		<pre> 8 9 DROP TABLE IF EXISTS raw.products_raw;</pre>	
<pre> 10 11 v CREATE TABLE raw.sales_raw (12 sale_id INT, 13 customer_name TEXT, 14 city TEXT, 15 product_name TEXT, 16 promotion_name TEXT, 17 units_sold INT, 18 price NUMERIC, 19 order_date DATE 20);</pre>		<pre> 10 11 v CREATE TABLE raw.products_raw (12 product_Id INT, 13 product_name TEXT, 14 catagory TEXT, 15 brand TEXT, 16 retail_price NUMERIC, 17 supplier TEXT 18);</pre>	
<pre> 21 22 v COPY raw.sales_raw 23 FROM 'C:\projects\data_warehouse_project\data\sales.csv' 24 DELIMITER ',' 25 CSV HEADER; 26 27 \$BODY\$;</pre>		<pre> 20 21 v COPY raw.products_raw 22 FROM 'C:\projects\data_warehouse_project\data\products.csv' 23 DELIMITER ',' 24 CSV HEADER; 25 26 v ALTER FUNCTION etl.load_raw_products() 27 OWNER TO postgres;</pre>	
<pre> 28 29 v ALTER FUNCTION etl.load_raw_sales() 30 OWNER TO postgres;</pre>			

Stage 2, Very basic data cleaning and transformation

Script for transforming the raw data in clean staging tables

```
1
2 ✓ CREATE OR REPLACE FUNCTION etl.transform_products()
3     RETURNS void
4     LANGUAGE 'sql'
5     COST 100
6     VOLATILE PARALLEL UNSAFE
7 AS $BODY$  
8
9
10
11    DROP TABLE IF EXISTS staging.products_cleaned;
12 ✓ CREATE TABLE staging.products_cleaned AS
13     SELECT DISTINCT ON (product_id)
14         product_id,
15         INITCAP(TRIM(product_name)) AS product_name,
16         INITCAP(TRIM(catagory)) AS catagory,
17         INITCAP(TRIM(brand)) AS brand,
18         retail_price,
19         retail_price * 1.15 AS with_gst_price,
20         INITCAP(supplier) AS supplier
21     FROM raw.products_raw
22     WHERE retail_price > 0;
23     AND product_name != '' AND product_name IS NOT NULL;  
24
25
26 $BODY$;  
27 ✓ ALTER FUNCTION etl.transform_products()
28     OWNER TO postgres;  
29
```

Stage 3, Script to load warehouse (loading 5 tables in star schema)

Script of loading the promotion diminution table and fact table below.

```

Query  Query History
40 JOIN staging.products_cleaned p ON s.product_name = p.product_name
41 ON CONFLICT (name) DO NOTHING;
42
43 ✓ INSERT INTO warehouse.dim_promotion (name,ad_type,coupon_type,
44     discount,start_date, end_date, active, updatetime)
45 SELECT DISTINCT -- default values
46     promotion_name,
47     'Unknown' AS ad_type,
48     'None' AS coupon_type,
49     0 AS discount,
50     CURRENT_DATE AS start_date,
51     CURRENT_DATE + INTERVAL '3 years' AS end_date,
52     TRUE AS active,
53     CURRENT_TIMESTAMP AS updatetime
54 FROM staging.sales_cleaned
55 WHERE promotion_name IS NOT NULL
56 ON CONFLICT (name) DO NOTHING;
57
58 ✓ INSERT INTO warehouse.fact_sales (customer_id, product_id,
59     promotion_id, date_id, units_sold, revenue)
60 SELECT
61     c.customer_id,
62     pr.product_id,
63     pro.promotion_id,
64     d.date_id,
65     s.units_sold,
66     s.total_revenue AS revenue
67
68 FROM staging.sales_cleaned s
69 JOIN warehouse.dim_customer c ON s.customer_name = c.name
70 JOIN warehouse.dim_product pr ON s.product_name = pr.name
71 --left join as many sales have may have no promotion.
72 LEFT JOIN warehouse.dim_promotion pro ON s.promotion_name = pro.name
73 JOIN warehouse.dim_date d ON s.order_date = d.order_date;
74
75 $BODY$;
76 ✓ ALTER FUNCTION etl.load_warehouse()
77     OWNER TO postgres;
78

```

Stage 3, Create automated pipeline procedure to automate etl process

Created a script etl() that calls functions and different scripts including, load_raw_sales(), transform_sales() and load_warehouse().

In the image below I reset the raw and staging tables before running fully automated etl with 100,000 rows of data. This effectively works in 17s loading the fact table with clean.

The screenshot shows a database interface with a query editor and a results viewer.

Query History:

```
1 SELECT etl.delete_sales_tables();
2 SELECT etl.clear_warehouse();
3 CALL etl.etl();
4
5 v SELECT * FROM warehouse.fact_sales
6 ORDER BY revenue DESC
7 LIMIT 10;
```

Data Output:

Showing rows: 1 to 10 |

	sale_id [PK] integer	customer_id integer	product_id integer	promotion_id integer	date_id integer	units_sold numeric	revenue numeric
1	50649	16651	3	4	344	20	19986.60
2	19809	27270	6	3	157	20	19986.20
3	29273	67018	7	5	52	20	19985.60
4	10126	10906	7	2	329	20	19982.20
5	86414	80045	9	2	294	20	19981.00
6	46940	37700	3	2	194	20	19977.00
7	66128	38933	2	4	351	20	19970.0
8	67094	46798	7	4	140	20	19959.60
9	1500	21525	9	2	111	20	19959.20
10	37149	51175	10	4	118	20	19952.0

From Bradley Erskine