Physical Data Model – UK Housing Transactions

Platform: PostgreSQL
Modelling Approach: Kimball Star Schema
Transformation Framework: dbt
Design Focus: Analytics performance, data quality, and production-readiness

---

1. Purpose and Scope

This document defines the **physical data model** for the UK housing transactions dataset. It translates the logical data model into concrete PostgreSQL DDL, aligned with modern analytics engineering and dbt best practices.

The physical model is designed to support:

- Scalable CSV ingestion
- Incremental transformations
- High-performance BI dashboards
- Downstream AI and RAG-based analytics

---

2. Database and Schema Strategy

To ensure clear separation of concerns, the database is organized into multiple schemas.

2.1 Schema Design

Schema	Purpose
raw	Unmodified ingested source data
staging	Cleaned, standardized, lightly transformed data
analytics	Star schema (facts and dimensions)

2.2 dbt Mapping

- raw → Ingested via Airbyte or bulk COPY
- staging → dbt models prefixed with stg_
- analytics → dbt models prefixed with dim_ and fact_

This structure enforces data lineage, testability, and maintainability.

---

3. Raw Layer (PostgreSQL)

3.1 Raw Source Table: raw.price_paid_2025

Design Principles:

- Schema mirrors the CSV source exactly
- Minimal constraints to avoid ingestion failures
- Text-heavy column types for robustness

```
CREATE TABLE raw.price_paid_2025 (
  transaction_id TEXT,
  price INTEGER,
  transaction_date DATE,
  postcode TEXT,
  property_type CHAR(1),
  old_new CHAR(1),
  duration CHAR(1),
  street TEXT,
  city TEXT,
  district TEXT,
  county TEXT
);
```

Notes:

- No primary keys or foreign keys
- Source uniqueness is trusted initially
- Annual tables may exist (e.g., price_paid_2024, price_paid_2026)

---

4. Staging Layer (dbt Models)

The staging layer is responsible for **data normalization and standardization**.

4.1 Staging Table: staging.stg_price_paid

```
CREATE TABLE staging.stg_price_paid AS
SELECT
  transaction_id,
  price::NUMERIC(12,2) AS price,
  transaction_date,
  UPPER(postcode) AS postcode,
  property_type,
  old_new,
  duration,
  INITCAP(district) AS district,
  INITCAP(county) AS county
FROM raw.price_paid_2025
WHERE transaction_id IS NOT NULL;
```

Responsibilities of the Staging Layer:

- Type casting and normalization
- Standardized casing
- Removal of invalid or incomplete records
- No business logic or aggregations

---

5. Analytics Layer: Dimension Tables

5.1 Date Dimension: analytics.dim_date

```
CREATE TABLE analytics.dim_date (
   date_key INTEGER PRIMARY KEY,
   date DATE NOT NULL UNIQUE,
   year INTEGER NOT NULL,
   quarter INTEGER NOT NULL,
   month INTEGER NOT NULL,
   month_name TEXT NOT NULL,
   day_of_month INTEGER NOT NULL,
   day_of_week INTEGER NOT NULL,
   is_weekend BOOLEAN NOT NULL
);
```

Design Notes:

- Pre-populated date spine
- Shared across all fact tables
- Primary key index sufficient for performance

5.2 Property Type Dimension: analytics.dim_property_type

```
CREATE TABLE analytics.dim_property_type (
   property_type_id SERIAL PRIMARY KEY,
   property_type_code CHAR(1) UNIQUE NOT NULL,
   property_type_desc TEXT NOT NULL
);
```

Seed Data:

Code	Description
F	Flat
D	Detached
S	Semi-Detached
T	Terraced
O	Other

5.3 Region Dimension: analytics.dim_region

```
CREATE TABLE analytics.dim_region (
   region_id SERIAL PRIMARY KEY,
   region_name TEXT NOT NULL,
   county TEXT,
   district TEXT,
   UNIQUE (region_name, county, district)
);
```

Design Notes:

- Region derived using ONS reference data
- Supports region-centric UK housing analysis

Indexes:

```
CREATE INDEX idx_dim_region_name
```

```
ON analytics.dim_region(region_name);
```

---

6. Analytics Layer: Fact Table

6.1 Fact Table: analytics.fact_property_sales

```
CREATE TABLE analytics.fact_property_sales (
  transaction_id TEXT PRIMARY KEY,
  date_key INTEGER NOT NULL,
  region_id INTEGER NOT NULL,
  property_type_id INTEGER NOT NULL,
  price NUMERIC(12,2) NOT NULL,
  old_new_flag CHAR(1),
  duration CHAR(1),

  CONSTRAINT fk_date
    FOREIGN KEY (date_key)
    REFERENCES analytics.dim_date(date_key),

  CONSTRAINT fk_region
    FOREIGN KEY (region_id)
    REFERENCES analytics.dim_region(region_id),

  CONSTRAINT fk_property_type
    FOREIGN KEY (property_type_id)
    REFERENCES analytics.dim_property_type(property_type_id)
);
```

Grain:

One row per completed property transaction

---

7. Indexing Strategy

Indexes are applied based on **query access patterns**.

```
CREATE INDEX idx_fact_sales_date
ON analytics.fact_property_sales(date_key);

CREATE INDEX idx_fact_sales_region
ON analytics.fact_property_sales(region_id);

CREATE INDEX idx_fact_sales_property_type
ON analytics.fact_property_sales(property_type_id);
```

Supported Query Patterns:

- Time-series analysis
- Regional breakdowns
- Property-type filtering

---

8. Incremental Loading and Performance

8.1 Incremental Key

transaction_date

Used in dbt to:

- Load only new or changed records
- Support daily or near-real-time refreshes

8.2 Partitioning (Optional Enhancement)

```
PARTITION BY RANGE (date_key)
```

Example strategy:

- Yearly partitions (e.g., 2024, 2025)

This is optional but recommended for very large datasets.

---

9. Data Quality and Constraints

Data quality enforcement is handled primarily in **dbt**, not at the database layer.

Rule	Enforcement Layer
transaction_id not null	dbt test
price > 0	dbt test
Valid property type	dbt test
Foreign key integrity	dbt test

PostgreSQL constraints are kept minimal to preserve ingestion stability.

---

10. Physical Model Summary

This physical data model follows analytics engineering best practices:

- Clear separation of raw, staging, and analytics layers
- Star schema optimized for BI workloads
- Surrogate keys and foreign key relationships
- Query-driven indexing strategy
- Incremental, testable transformations using dbt

This design is suitable for **production-grade analytics platforms**, executive dashboards, and AI-driven insight generation.