Assignment 2 (AI-611 ADMEP-EL1) RollNo_25201317

Q1) You are hired as a data engineer for ShopSmart, a national retail chain that operates 100+ stores and an online e-commerce platform. ShopSmart wants to build a central analytics warehouse to analyze sales performance, customer behavior, and inventory trends across multiple channels.

1 Identify Fact and Dimension Tables

Fact Table

Fact Table	Measures / Metrics
`Sales_Fact`	sales_amount, quantity_sold, discount_applied, profit
`Inventory_Fact`	stock_on_hand, reorder_level, units_sold
`Promotion_Fact`	promotion_discount, promotion_revenue, product_count

Dimension Tables

Dimension Table	Attributes
`Customer_Dim`	customer_id, first_name, last_name, gender, birth_date, loyalty_level, city, state, region
`Product_Dim`	product_id, product_name, category, subcategory, brand, supplier_id, price
`Store_Dim`	store_id, store_name, city, state, region, store_manager
`Time_Dim`	date_key, date, day, week, month, quarter, year, holiday_flag
`Promotion_Dim`	promotion_id, promotion_name, start_date, end_date, discount_percent

---

2️⃣ Star Schema Design

```
          Customer_Dim
                |
Product_Dim — Sales_Fact — Time_Dim
                |
             Store_Dim
```

- Fact table: `Sales_Fact`
- Dimensions: `Customer_Dim`, `Product_Dim`, `Store_Dim`, `Time_Dim`

Advantages:

- Simple for BI tools
- Faster query execution

3️⃣ Snowflake Schema

```
              Customer_Dim
                    |
               Sales_Fact
              /    |     \
     Product_Dim  Store_Dim  Time_Dim
           |
     Category_Dim
```

Advantages: Reduces redundancy
Disadvantage: Slower joins

4️⃣ Slowly Changing Dimensions (SCD)

- `Customer_Dim`: Loyalty level changes → **Type 2 SCD**
- `Product_Dim`: Category changes → **Type 2 SCD**

5 Example Queries

(a) Total Sales per Region per Month

```
SELECT t.year, t.month, s.region,
       SUM(f.sales_amount) AS total_sales
FROM Sales_Fact f
JOIN Time_Dim t ON f.time_key = t.date_key
JOIN Store_Dim s ON f.store_key = s.store_id
GROUP BY t.year, t.month, s.region
ORDER BY t.year, t.month, s.region;
```

(b) Top 5 Products by Revenue

```
SELECT p.product_name,
       SUM(f.sales_amount) AS revenue
FROM Sales_Fact f
JOIN Product_Dim p ON f.product_key = p.product_id
GROUP BY p.product_name
ORDER BY revenue DESC
LIMIT 5;
```

(c) Customer Retention Analysis

```
SELECT c.loyalty_level,
       COUNT(DISTINCT f.customer_key) AS active_customers
FROM Sales_Fact f
JOIN Customer_Dim c ON f.customer_key = c.customer_id
WHERE f.time_key BETWEEN '2025-01-01' AND '2025-12-31'
GROUP BY c.loyalty_level;
```

---

6 Business Impact

- **Sales Optimization:** Insights into best-selling products per region
- **Customer Retention:** Loyalty-based segmentation
- **Inventory Management:** Minimize stockouts

Q2) You are a data engineer for QuickEats, an online

food delivery platform operating in multiple cities. QuickEats collects and processes data from multiple sources. Currently, the system struggles with scalability, real-time processing, and analytics performance. Suggest a suitable model.

1️⃣ Identify Fact and Dimension Tables

Fact Tables

Fact Table	Measures / Metrics
`Orders_Fact`	order_count, order_value, delivery_fee, tip_amount, preparation_time_sec, delivery_time_sec, order_status_code
`Delivery_Fact`	trip_duration_sec, distance_km, driver_idle_time_sec, delivery_cost
`Payments_Fact`	payment_amount, payment_fee, commission_amount

Dimension Tables

Dimension Table	Key Attributes
`Customer_Dim`	customer_id, first_name, last_name, signup_date, city, state, city_zone, loyalty_tier, avg_order_value
`Restaurant_Dim`	restaurant_id, name, cuisine_type, rating, price_level, city, partner_since, activation_status
`MenuItem_Dim`	item_id, restaurant_id, item_name, category, price, is_veg, prep_time_est
`Driver_Dim`	driver_id, name, vehicle_type, city, rating, active_since
`Time_Dim`	date_key, date, hour, day_of_week, week, month, quarter, year, is_holiday
`Location_Dim`	location_id, city, zone, lat, lon, avg_traffic_index
`Promotion_Dim`	promo_id, promo_type, discount_pct, start_date, end_date

2️⃣ Star Schema Design (recommended for analytics / BI)

```
             Customer_Dim
                    |
MenuItem_Dim - Orders_Fact - Time_Dim
                    |
              Restaurant_Dim
                    |
              Driver_Dim
```

- Central fact: `Orders_Fact` referencing `Customer_Dim`, `Restaurant_Dim`, `MenuItem_Dim`, `Driver_Dim`, `Time_Dim`, `Location_Dim`, `Promotion_Dim`.
- Denormalized dims for fast aggregation (dashboards, KPIs).

Why star: Simple, low-join latency for large-volume analytics queries (top restaurants, hourly demand, etc.).

3️⃣ Snowflake Schema Option

- Normalize `Restaurant_Dim` into `Restaurant_Dim` → `Cuisine_Dim`, `Chain_Dim` (if multi-branch)
- Normalize `Location_Dim` into `City_Dim` → `Zone_Dim`

Pros: less redundancy, smaller storage for very large dimension cardinality.
Cons: more joins → slower interactive queries; more complex ETL.

4️⃣ Slowly Changing Dimensions (SCD) Policies

- `Customer_Dim`: address/loyalty_tier → **Type 2** (preserve historical churn/retention analysis).
- `Restaurant_Dim`: rating, price_level → **Type 2** for rating changes relevant to historical quality analysis.
- `MenuItem_Dim`: price, category → **Type 2** if price changes must be tracked historically; otherwise **Type 1** for minor corrections.

- `Driver_Dim`: vehicle_type → **Type 1** (overwrite) unless you need historical driver-vehicle analytics then Type 2.

5️⃣ Example Queries

a) Hourly Orders and Average Delivery Time (last 24 hours)

```
SELECT t.date, t.hour,
       COUNT(DISTINCT f.order_id) AS orders,
       AVG(f.delivery_time_sec) / 60.0 AS avg_delivery_min
FROM Orders_Fact f
JOIN Time_Dim t ON f.time_key = t.date_key
WHERE f.event_time >= CURRENT_TIMESTAMP - INTERVAL '24 HOURS'
GROUP  BY  t.date,  t.hour
ORDER BY t.date DESC, t.hour;
```

b) Top 10 Restaurants by GMV (gross merchandise value) this week

```
SELECT r.restaurant_id, r.name, SUM(f.order_value) AS gmv
FROM Orders_Fact f
JOIN Restaurant_Dim r ON f.restaurant_key = r.restaurant_id
JOIN Time_Dim t ON f.time_key = t.date_key
WHERE t.week = DATE_PART('week', CURRENT_DATE)
GROUP BY r.restaurant_id, r.name
ORDER BY gmv DESC
LIMIT 10;
```

c) Drivers with High Average Delay (>X mins)

```
SELECT d.driver_id, d.name, AVG(f.delivery_time_sec -
f.estimated_delivery_sec)/60.0 AS avg_delay_min
FROM Delivery_Fact f
JOIN Driver_Dim d ON f.driver_key = d.driver_id
GROUP BY d.driver_id, d.name
HAVING AVG(f.delivery_time_sec - f.estimated_delivery_sec) > 10*60;
```

d) Promo lift: orders with promo vs without (last 30 days)

```
SELECT p.promo_id, p.promo_type,
       COUNT(f.order_id) AS orders_with_promo,
       SUM(f.order_value) AS gmv_with_promo
FROM Orders_Fact f
LEFT JOIN Promotion_Dim p ON f.promo_key = p.promo_id
WHERE f.order_date BETWEEN CURRENT_DATE - INTERVAL '30 days' AND
CURRENT_DATE
GROUP BY p.promo_id, p.promo_type
ORDER BY orders_with_promo DESC;
```

---

6 Technology & Processing Pattern (Suggested architecture)

- **Ingestion:** Kafka for order/driver events, CDC for partner/restaurant updates.
- **Stream Processing (real-time):** Flink or Spark Structured Streaming (ETL, real-time ETAs, surge detection).
- **Serving / OLAP:** Delta Lake on S3 or BigQuery / Snowflake for analytical queries.
- **Feature Store / ML:** Feast or Hopsworks for features (ETL + streaming features used by ETA and recommender models).
- **Monitoring:** Prometheus + Grafana for pipeline metrics; DataDog for SLA alerts.

---

7 Justification & Business Impact

- **Why this schema & stack:** Food delivery requires low-latency decisions (ETAs, surge pricing) + accurate historical analytics (restaurant performance). Star schema + streaming-first ingestion yields fast dashboards and real-time features for ML.
- **Business outcomes:** Improved customer experience (accurate ETAs), reduced delivery costs (better routing/surge pricing), higher partner retention (transparent metrics), and

uplift in GMV through targeted promotions.

Q3) You are a data engineer for StreamFlix, a global video streaming platform (like Netflix). StreamFlix collects millions of events per day. The company wants to build a high-performance analytics warehouse to support:

- **Real-time viewer engagement analytics**
- **Top trending videos per region**
- **AI models for recommendation engines**

1 Identify Fact and Dimension Tables

Fact Tables

Fact Table	Measures / Metrics
`Playback_Fact`	play_count, watch_seconds, start_time, end_time, play_success_flag, pause_count
`Engagement_Fact`	like_count, share_count, comment_count, ad_clicks, skip_ad_flag
`Recommendation_Fact`	rec_shown_count, rec_click_count, rec_conversions
`Buffering_Fact`	buffer_events, total_buffer_duration_sec

Dimension Tables

Dimension Table	Key Attributes
`User_Dim`	user_id, signup_date, country, region, city, age_bucket, subscription_tier, device_type
`Content_Dim`	content_id, title, genre, language, duration_sec, release_date, is_original
`Time_Dim`	date_key, date, hour, day_of_week, week, month, quarter, year

`Device_Dim`	device_id, device_type, os, os_version, app_version
`Region_Dim`	region_id, country, continent, timezone
`Session_Dim`	session_id, user_id, session_start, session_end, session_device
`Ad_Dim`	ad_id, advertiser, ad_length_sec, ad_type

2️⃣ Star Schema Design (for analytical queries)

```
      User_Dim        Content_Dim
          \                 /
           \              /
            Playback_Fact - Time_Dim
             /       \
     Device_Dim    Region_Dim
```

- `Playback_Fact` is central, joined to `User_Dim`, `Content_Dim`, `Time_Dim`, `Device_Dim`, `Region_Dim`.
- Fast aggregations for trending content, CTRs, engagement metrics.

3️⃣ Snowflake Schema Option

- Normalize `Content_Dim` → `Genre_Dim`, `ProductionHouse_Dim`, `Language_Dim`.
- Normalize `User_Dim` → `Demographic_Dim` (age_bucket, gender) and `Subscription_Dim`.

Tradeoff: saves storage but increases join complexity; acceptable if dim cardinality is huge.

4️⃣ Slowly Changing Dimensions (SCD) Policies

- `User_Dim`: subscription_tier, region → **Type 2** (knowing user tier at time of watch is critical for retention / revenue analysis).
- `Content_Dim`: title metadata rarely changes → **Type 1** for minor metadata fixes, **Type 2**

if content reclassification affects historical reporting (e.g., category/genre changes).

- `Device_Dim`: app_version → **Type 1** overwrite (we usually keep version history in logs rather than SCD).

5️⃣ Example Queries

a) Real-time viewer count per region (last 5 minutes)

```
SELECT r.region, COUNT(DISTINCT f.user_id) AS active_viewers
FROM Playback_Fact f
JOIN Time_Dim t ON f.time_key = t.date_key
JOIN Region_Dim r ON f.region_key = r.region_id
WHERE f.event_time >= CURRENT_TIMESTAMP - INTERVAL '5 MINUTES'
GROUP BY r.region;
```

b) Top trending videos per region (last hour)

```
SELECT c.content_id, c.title, r.region, COUNT(*) AS views
FROM Playback_Fact f
JOIN Content_Dim c ON f.content_key = c.content_id
JOIN Region_Dim r ON f.region_key = r.region_id
WHERE f.event_time >= CURRENT_TIMESTAMP - INTERVAL '1 HOUR'
GROUP BY c.content_id, c.title, r.region
ORDER BY r.region, views DESC
LIMIT 10;
```

c) Feature store extraction for recommender (user recent watch features)

```
-- pseudo-SQL / feature pipeline step
SELECT user_id,
       COUNT(DISTINCT content_id) FILTER (WHERE event_time >= NOW() -
INTERVAL '7 DAYS') AS unique_views_7d,
       SUM(watch_seconds) FILTER (WHERE genre='Drama' AND event_time
>= NOW() - INTERVAL '30 DAYS') AS drama_watch_30d,
       AVG(session_length_sec) AS avg_session_sec
FROM Playback_Fact
GROUP BY user_id;
```

d) Buffering rate by device / app version (SLA monitoring)

```
SELECT d.device_type, p.app_version,
       SUM(p.buffer_events) AS total_buffer_events,
       SUM(p.total_buffer_duration_sec)/SUM(p.watch_seconds) AS
buffer_ratio
FROM Buffering_Fact p
JOIN Device_Dim d ON p.device_key = d.device_id
GROUP BY d.device_type, p.app_version
ORDER BY buffer_ratio DESC;
```

6️⃣ Architecture & Processing Pattern (Streaming-first / Kappa)

Recommended stack:

- **Event ingestion:** Kafka for high-throughput events (play, pause, seek, buffer)
- **Stream processing:** Flink / Spark Streaming for real-time metrics, aggregations, and feature extraction
- **Storage:** Delta Lake / BigQuery / Snowflake for served analytics; clickstream raw events in S3/Parquet
- **Feature Store:** Redis/Feast for low-latency features to recommenders
- **ML infra:** Kubeflow or AWS Sagemaker for model training; online model serving via TF-Serving or Triton
- **Query engine:** Presto/Trino for ad-hoc analytics; materialized views for trending lists
- **Realtime dashboarding:** Superset / Looker + materialized Kafka→OLAP pipelines

---

7️⃣ Support for ML Recommendation Engines

- **Feature engineering:** streaming + batch features (recent watch history, genre affinity, device preferences).
- **Candidate generation:** use approximate nearest neighbors (FAISS) on content embeddings (from metadata + collaborative signals).
- **Ranking model:** deep learning model (e.g., DSSM, two-tower) using streaming features to produce top-N for each session.
- **A/B experimentation:** use event-sourced logs to measure CTR, watch-through-rate, revenue lift.

---

8 Justification & Business Impact

- **Why streaming-first:** user engagement is real-time — recommendations and trending must adapt within minutes.
- **Business outcomes:**
    - **Better retention:** personalized recommendations increase watch-time and reduce churn.
    - **Content investment decisions:** accurate trending signals guide licensing and original content production.
    - **Ad monetization:** higher engagement and better ad targeting increase ad revenue.
    - **Operational SLAs:** detecting device/version buffering patterns reduces churn caused by technical issues.