



## A. SOLUCIONES SQL

### 1. Top 3 productos por categoría

```
WITH ranked AS (
    SELECT
        category,
        product_id,
        SUM(amount) AS total_sales,
        ROW_NUMBER() OVER (
            PARTITION BY category
            ORDER BY SUM(amount) DESC
        ) AS rn
    FROM sales
    GROUP BY category, product_id
)
SELECT category, product_id, total_sales
FROM ranked
WHERE rn <= 3;
```

---

### 2. Usuarios inactivos 45 días

```
WITH last_event AS (
    SELECT
        user_id,
        MAX(event_time) AS last_time
    FROM events
    GROUP BY user_id
)
SELECT user_id
FROM last_event
WHERE last_time < CURRENT_DATE() - INTERVAL 45 DAY
    AND last_time IS NOT NULL;
```

---

### 3. Pivot de métricas

BigQuery / Snowflake:

```
SELECT
    user_id,
    MAX(CASE WHEN metric = 'clicks' THEN value END) AS clicks,
    MAX(CASE WHEN metric = 'views' THEN value END) AS views,
    MAX(CASE WHEN metric = 'purchases' THEN value END) AS purchases
FROM metrics
GROUP BY user_id;
```

---

## 4. Churn mensual

```
WITH active AS (
    SELECT
        customer_id,
        month,
        active
    FROM subscriptions
)
, prev AS (
    SELECT
        customer_id,
        month,
        LAG(active) OVER (PARTITION BY customer_id ORDER BY month) AS prev_active,
        active AS curr_active
    FROM active
)
SELECT
    month,
    SUM(CASE WHEN prev_active = 1 AND curr_active = 0 THEN 1 END)
    / SUM(CASE WHEN prev_active = 1 THEN 1 END) AS churn_rate
FROM prev
GROUP BY month;
```

---

## 5. Transacciones duplicadas ( $\leq 30s$ )

```
WITH base AS (
    SELECT
        *,
        LAG(timestamp) OVER (PARTITION BY user_id, amount ORDER BY timestamp) AS prev_ts
    FROM transactions
)
SELECT *
FROM base
WHERE prev_ts IS NOT NULL
    AND TIMESTAMP_DIFF(timestamp, prev_ts, SECOND) <= 30;
```

---

# ■ B. SOLUCIONES DE MODELADO / ARQUITECTURA

## 1. E-commerce – Modelo dimensional

### Dimensiones

- **dim\_user** (SCD1)  
id, email, address (sobrescribe, no histórico), created\_at
- **dim\_product** (SCD2)  
product\_sk, product\_id, name, price, category, brand, valid\_from, valid\_to
- **dim\_date, dim\_geography, dim\_currency**

## Hecho

### **fact\_orders**

- order\_sk
- product\_sk
- user\_sk
- quantity
- subtotal
- total
- currency
- order\_date\_sk

## Decisiones claves

- Dirección del usuario: en **dim\_user** SCD1 (solo valor actual) y dirección histórica se registra en la tabla de “orders\_address”.
  - Carritos abandonados:  
→ crear **fact\_cart\_activity**, con `is_abandoned = 1`.
- 

## 2. Plataforma de streaming

### Hechos

- **fact\_plays** (evento play)  
user\_sk, content\_sk, device\_sk, play\_time, duration, session\_sk
- **fact\_sessions**  
session\_sk, user\_sk, start\_time, end\_time, device\_sk

### Dimensiones

- **dim\_content** (SCD2) – porque categoría, título o metadata pueden cambiar
- **dim\_user**
- **dim\_device**

### Definición de sesión

- Empieza al primer evento
- Termina si no hay actividad durante 30 minutos

## Dedupe

Aplicar reglas:

1. Misma user\_id + content\_id
  2. Timestamps dentro de ±2s
  3. Obtener último por client\_sequence
- 

## 3. Pipeline GCP (Pub/Sub → Dataflow → BigQuery)

### Esquema final en BigQuery

- Particionamiento por ingestion\_date
- Clustering por customer\_id

### Validaciones

- Primer nivel: Dataflow (schema + tipo de datos + reglas básicas)
- Segundo nivel: BigQuery + Great Expectations/dbt tests

### Idempotencia

- Usa primary keys y inserciones MERGE en BigQuery
- Cada evento lleva event\_id; Dataflow descarta duplicados

### Reprocesamiento

- Reescritura de particiones específicas
  - Jobs ad hoc en Dataflow
- 

## ■ C. SOLUCIONES dbt

### 1. Modelo incremental – `models/marts/fct_orders.sql`

```
 {{ config(  
   materialized='incremental',  
   unique_key='order_id'  
 )}}
```

```

) }

WITH base AS (
    SELECT *
    FROM {{ source('raw', 'orders') }}
    {% if is_incremental() %}
        WHERE updated_at > (SELECT MAX(updated_at) FROM {{ this }})
    {% endif %}
)
SELECT
    order_id,
    customer_id,
    product_id,
    amount,
    updated_at
FROM base;

```

## Tests (`schema.yml`)

```

version: 2

models:
  - name: fct_orders
    columns:
      - name: order_id
        tests:
          - not_null
          - unique
      - name: customer_id
        tests:
          - relationships:
              to: ref('dim_customers')
              field: customer_id

```

---

## 2. Snapshot

```

snapshots/products_snapshot.sql

{% snapshot products_snapshot %}

{ {
    config(
        target_schema='snapshots',
        unique_key='product_id',
        strategy='check',
        check_cols=['price', 'category']
    )
} }

SELECT *
FROM {{ source('raw', 'products') }}

```

```
{% endsnapshot %}
```

---

## D. CASOS END-TO-END

### 1. Pipeline de pedidos – Solución

#### Ingestión

- API REST → Airflow DAG (cada 6h)
- CSV → Cloud Storage → Autoload a raw
- Kafka → Dataflow → BigQuery landing

#### Staging (dbt)

- Limpieza
- Conversión de tipos
- Deduplicación
- Normalización de IDs

#### Intermediate

- Transformaciones de negocio
- Validaciones (tests dbt + Great Expectations)

#### Marts

- fact\_orders
- dim\_customer
- dim\_product

#### Orquestación

- Airflow (ingest + dbt run + validaciones + BI refresh)

#### Alertas

- Logs en GCP
- Notificaciones Slack/Email

#### Late arriving data

- Corrección por MERGE en BigQuery/Snowflake
- Reprocesar la partición afectada

---

## 2. ML pipeline en Snowflake – Solución

### Arquitectura

- Stage → Snowpipe/Streams → Bronze
- Transformations en Silver (dbt)
- Feature Store en Gold
- Modelo guardado en Snowflake External Functions o Model Registry
- Monitoreo: drift + freshness

### Rollback

- Uso de Time Travel
  - Restaurar modelo anterior
  - Congelar partitions
- 

## 3. Empresa con 20 fuentes → Gobernanza

### Solución

- **Data contracts:** obligatorios para cada fuente
- **Metadata:** DataHub como catálogo + lineage automático
- **Enmascaramiento:** Snowflake dynamic masking
- **Validaciones:** Great Expectations ejecutado desde Airflow
- **Estandarización:** todos los pipelines usan la misma plantilla dbt (naming: stg\_, int\_, dim\_, fct\_)
- **CI/CD:** PRs requieren ejecución de tests dbt, SQLFluff, y validaciones

### Resultado

- Calidad unificada
  - Descubribilidad
  - Auditoría clara
  - Datos seguros
- 

## E. RESPUESTAS A PREGUNTAS DE COMPORTAMIENTO TÉCNICO

Aquí los guiones para responder en una entrevista (**método STAR**).

---

## 1. Pipeline del que estás orgulloso

- Reduje el tiempo de proceso de 3h → 18 min usando BigQuery, particiones y clustering.
  - Impacto: dashboards casi en tiempo real.
- 

## 2. Datos corruptos

- Detecté strings donde había enteros.
  - Implementé validación en ingress + tests dbt + reject bucket.
  - Eliminamos 80% de incidentes.
- 

## 3. Dataset de baja calidad

- Faltaba lineage + no había tests.
  - Implementé data contracts y tests.
  - La confiabilidad subió al 98%.
- 

## 4. Desacuerdo técnico

- Discutíamos si usar ETL vs ELT.
  - Analicé costos y rendimiento.
  - ELT en warehouse era 60% más barato → acordado.
- 

## 5. Priorizar tareas

- Impacto en negocio
  - Riesgo
  - Complejidad
  - Bloqueadores externos
- 

## 6. Bug difícil

- Unión duplicada por timestamps truncados.
  - Aplicué dedupe con window functions.
  - Problema resuelto en 30 min.
- 

## 7. SQL lento

- Escaneaba 2 TB.
  - Implementé particionamiento + clustering.
  - Bajó a 18 GB, consulta 30x más rápida.
- 

## 8. CI/CD

- PR ejecuta dbt test + SQLFluff + mypy.
  - Evitó romper el prod cientos de veces.
- 

## 9. Error en producción

- Run incremental sin unique\_key → duplicados.
  - Lo corregí, documenté y agregué test obligatorio.
- 

## 10. Explicar modelo complejo a stakeholders

- Usé diagramas simples y métricas de negocio.
- Transformé “dimensional modeling” en un “mapa” claro.
- Stakeholders entendieron métricas clave.