DIMENSIÓN TIEMPO (Dimensión Fecha)

INTRODUCCIÓN

La **dimensión fecha** (o **dimensión tiempo**) es una de las más importantes en un modelo dimensional, ya que permite analizar los datos a lo largo del tiempo, un aspecto crucial en prácticamente cualquier tipo de análisis empresarial o de datos.

¿Qué es la dimensión fecha?

La **dimensión fecha** es una tabla en un modelo dimensional que contiene información detallada sobre fechas, organizada en varios niveles de granularidad, como día, mes, trimestre y año. Esta dimensión se usa para conectar hechos temporales (ventas, transacciones, pagos, etc.) con intervalos de tiempo, facilitando el análisis temporal.

¿Por qué es tan importante?

1. **Análisis temporal común**:
   La mayoría de las preguntas empresariales tienen un componente temporal:
   - ¿Cuáles fueron las ventas este mes?
   - ¿Cómo se comparan las ventas de este trimestre con el anterior?
   - ¿Cuántos usuarios activos tuvimos el año pasado?
2. **Rendimiento**:
   Una tabla de dimensión fecha preconstruida permite consultas rápidas sin necesidad de cálculos complicados sobre columnas de fechas.
3. **Flexibilidad analítica**:
   Facilita el agrupamiento y filtrado por diferentes granularidades temporales, como días, semanas, meses, trimestres o años.
4. **Estandarización**:
   Define reglas consistentes para trabajar con fechas (por ejemplo, semanas fiscales, fines de semana, días festivos, etc.).

¿Cómo diseñarla?

La dimensión fecha se debe diseñar como una tabla denormalizada (ideal para un modelo dimensional) que incluye todos los atributos relevantes sobre las fechas.

Atributos comunes en una dimensión fecha:

Columna	Descripción
`date_key`	Clave primaria, usualmente en formato entero (`YYYYMMDD`) para conectar con la tabla de hechos.
`full_date`	Fecha completa (`YYYY-MM-DD`).
`year`	Año (`2025`, `2026`, etc.).
`quarter`	Trimestre (`Q1`, `Q2`, etc.).
`month`	Mes (nombre: "January", número: `01`).
`week_of_year`	Número de la semana en el año (`1` a `52`).
`day_of_week`	Nombre del día de la semana ("Monday").
`is_weekend`	Indica si el día es fin de semana (`1` para sábado y domingo).

Columna	Descripción
`is_holiday`	Indica si el día es un festivo (basado en un calendario específico).
`fiscal_year`	Año fiscal.
`fiscal_quarter`	Trimestre fiscal.
`day_of_month`	Día del mes (`1` a `31`).
`is_last_day_of_month`	Indica si es el último día del mes.

Ejemplo de registros en la dimensión fecha:

date_key	full_date	year	month	quarter	week_of_year	day_of_week	is_weekend	is_holiday
20250101	2025-01-01	2025	January	Q1	1	Wednesday	0	1
20250102	2025-01-02	2025	January	Q1	1	Thursday	0	0

Pasos para construir una dimensión fecha

1. Generar las fechas

- Es necesario generar una tabla con todas las fechas que cubran el rango de análisis (por ejemplo, desde 2010 hasta 2030).
- En SQL:

```
CREATE TABLE DimTime (
    date_key INT PRIMARY KEY,
    full_date DATE,
    year INT,
    month VARCHAR(15),
    month_number INT,
    quarter VARCHAR(5),
    week_of_year INT,
    day_of_week VARCHAR(15),
    is_weekend BOOLEAN
);

INSERT INTO DimTime (date_key, full_date, year, month, month_number, quarter,
week_of_year, day_of_week, is_weekend)
SELECT
    DATE_FORMAT(cal.date, '%Y%m%d') AS date_key,
    cal.date AS full_date,
    YEAR(cal.date) AS year,
    MONTHNAME(cal.date) AS month,
    MONTH(cal.date) AS month_number,
    CONCAT('Q', QUARTER(cal.date)) AS quarter,
    WEEKOFYEAR(cal.date) AS week_of_year,
    DAYNAME(cal.date) AS day_of_week,
    CASE WHEN DAYOFWEEK(cal.date) IN (1, 7) THEN 1 ELSE 0 END AS is_weekend
FROM (
    SELECT ADDDATE('2000-01-01', INTERVAL @row_number:=@row_number+1 DAY) AS
date
```

```
    FROM (SELECT 1 UNION SELECT 2) t1,
         (SELECT 1 UNION SELECT 2) t2,
         (SELECT 1 UNION SELECT 2) t3,
         (SELECT 1 UNION SELECT 2) t4,
         (SELECT @row_number:=-1) t5
    LIMIT 36500
) cal;
```

2. Agregar información adicional

- **Festivos**:
  - Puedes incluir un archivo externo con los días festivos y relacionarlos con `date_key` para marcar `is_holiday`.
  - Ejemplo en Python:

```python
import pandas as pd

# Generar calendario
dates = pd.date_range(start="2000-01-01", end="2030-12-31")
dim_date = pd.DataFrame({'full_date': dates})
dim_date['is_holiday'] = dim_date['full_date'].apply(
    lambda x: 1 if x in ['2025-01-01', '2025-12-25'] else 0
)
```

- **Año fiscal**:
  - Define reglas para calcular años fiscales si son diferentes del año calendario (por ejemplo, del 1 de julio al 30 de junio).

3. Optimizar para consultas

- Crea índices en columnas clave como `date_key` y `year` para acelerar las consultas.

Consultas comunes con la dimensión fecha

1. **Ventas por mes**:

```sql
SELECT d.year, d.month, SUM(f.sales_amount) AS total_sales
FROM FactSales f
JOIN DimTime d ON f.date_key = d.date_key
GROUP BY d.year, d.month
ORDER BY d.year, d.month;
```

2. **Ingresos en fines de semana**:

```sql
SELECT SUM(f.sales_amount) AS weekend_sales
FROM FactSales f
JOIN DimTime d ON f.date_key = d.date_key
WHERE d.is_weekend = 1;
```

Script en Python para generar una Dimensión Fecha

```python
import pandas as pd
from datetime import datetime, timedelta

# Parámetros: rango de fechas
start_date = "2000-01-01"  # Fecha de inicio
end_date = "2030-12-31"    # Fecha de fin

# Generar rango de fechas
dates = pd.date_range(start=start_date, end=end_date)

# Crear DataFrame base
dim_date = pd.DataFrame({
    'full_date': dates,                          # Fecha completa
    'date_key': dates.strftime('%Y%m%d').astype(int),  # Clave de fecha
(YYYYMMDD)
    'year': dates.year,                          # Año
    'quarter': dates.quarter,                    # Trimestre (1-4)
    'month': dates.month,                        # Número del mes (1-12)
    'month_name': dates.strftime('%B'),          # Nombre del mes (January,
February, ...)
    'week_of_year': dates.isocalendar().week,    # Semana del año (1-52)
    'day_of_week': dates.dayofweek + 1,          # Día de la semana (1=Monday,
7=Sunday)
    'day_name': dates.strftime('%A'),            # Nombre del día (Monday,
Tuesday, ...)
    'day_of_month': dates.day,                   # Día del mes (1-31)
    'is_weekend': dates.weekday.isin([5, 6]).astype(int),  # Fin de semana (0=No,
1=Sí)
    'is_holiday': 0                              # Inicialmente 0, para marcar
festivos
})

# Agregar columnas adicionales
dim_date['quarter_name'] = 'Q' + dim_date['quarter'].astype(str)  # Nombre del
trimestre
dim_date['is_last_day_of_month'] = (dim_date['full_date'] +
pd.offsets.MonthEnd(0) == dim_date['full_date']).astype(int)

# Agregar columna para el año fiscal (Ejemplo: año fiscal empieza en julio)
dim_date['fiscal_year'] = dim_date['year']
dim_date.loc[dim_date['month'] >= 7, 'fiscal_year'] += 1  # Año fiscal avanza en
julio

# Opcional: marcar días festivos (puedes personalizar este calendario)
# Ejemplo simple: Año Nuevo y Navidad
holidays = ['2000-01-01', '2000-12-25', '2025-01-01', '2025-12-25']  # Ajustar
según necesidades
dim_date['is_holiday'] =
dim_date['full_date'].isin(pd.to_datetime(holidays)).astype(int)

# Vista previa de la tabla generada
print(dim_date.head())
```

```
# Exportar a CSV (opcional)
dim_date.to_csv('dim_date.csv', index=False)
```

Explicación del script:

1. **Definir el rango de fechas**:
   Utiliza `pd.date_range` para crear un rango de fechas desde `start_date` hasta `end_date`.
2. **Crear atributos clave**:
   A partir del rango de fechas, se generan columnas como `year`, `month`, `quarter`, `day_of_week`, etc. Estas se derivan directamente de las funciones de pandas o del formato de las fechas.
3. **Días festivos personalizados**:
   Se define una lista de fechas festivas (`holidays`) y se marca la columna `is_holiday` como `1` para estas fechas.
4. **Año fiscal**:
   El ejemplo asume que el año fiscal comienza en julio. Si el mes es mayor o igual a 7, el año fiscal avanza.
5. **Último día del mes**:
   Utiliza `pd.offsets.MonthEnd` para identificar si una fecha es el último día del mes.
6. **Exportar a CSV**:
   Si deseas usar esta tabla en otro entorno (por ejemplo, en una base de datos), puedes exportarla como un archivo CSV.

Resultado esperado (primeros registros):

full_date	date_key	year	quarter	month	month_name	week_of_year	day_of_week	day_name	is_weekend	is_holiday	fiscal_year
2000-01-01	20000101	2000	1	1	January	52	6	Saturday	1	1	2000
2000-01-02	20000102	2000	1	1	January	1	7	Sunday	1	0	2000
2000-01-03	20000103	2000	1	1	January	1	1	Monday	0	0	2000

Personalización adicional

- **Festivos locales**: Puedes cargar un archivo CSV con los festivos de tu país o región para marcarlos automáticamente.
- **Agregados adicionales**: Si tu organización usa semanas fiscales específicas (por ejemplo, semana 53), puedes incluir esa lógica.

Generación de la DIMENSIÓN FECHA

La generación de la **dimensión fecha** puede formar parte del **ETL** o ser un proceso independiente, dependiendo del diseño de tu sistema de datos. Vamos a analizar en qué momento podría entrar en el flujo y cuál es su propósito:

¿Es parte del ETL?

Sí, puede considerarse parte del proceso ETL en los siguientes casos:

1. **Cuando la dimensión fecha se genera o actualiza como parte de la carga del Data Warehouse**:
    - En un **ETL bien estructurado**, la **dimensión fecha** se crea o actualiza como una de las primeras tareas. Esto se debe a que otras tablas, como las tablas de hechos, necesitarán referencias a esta dimensión para conectar los eventos temporales con los análisis temporales.
    - Por ejemplo:
        - Durante el proceso ETL, mientras se carga la tabla de hechos (`FactSales`), cada fila debe enlazarse a la clave de fecha (`date_key`) de la dimensión fecha.
        - Esto asegura que los datos en las tablas de hechos estén correctamente conectados con la granularidad temporal.

    Tareas típicas en ETL relacionadas con la dimensión fecha:
    - Generar la tabla de dimensión fecha (como el ejemplo en Python).
    - Insertar nuevos registros en la dimensión fecha si el rango de fechas del Data Warehouse se expande.
    - Validar la consistencia entre la dimensión fecha y los datos cargados.
2. **Cuando se automatiza como una tarea inicial del ETL**:
    - Si estás usando una herramienta como **Apache Airflow** o un **script programado**, puedes incluir la generación o actualización de la dimensión fecha como una tarea automatizada. Esto asegura que la tabla esté lista antes de cargar datos en las tablas de hechos.

¿Es posterior al ETL y parte del análisis?

En algunos casos, la generación de la **dimensión fecha** se realiza **antes del ETL o de manera independiente**, como una tarea preparatoria o de infraestructura. En este contexto, se usa durante el análisis, pero no como parte del proceso directo de extracción, transformación y carga.

1. **Razón para ser independiente**:
    - La dimensión fecha es estática (no cambia frecuentemente) y puede generarse una sola vez para cubrir un rango de años amplio. Por ejemplo:
        - Si generas una dimensión fecha de 20 años (2000-2030), es probable que no necesites actualizarla frecuentemente.
        - Esto permite tratarla como un recurso estático que simplemente se referencia durante el análisis.
2. **Uso posterior para análisis**:
    - Si ya tienes la dimensión fecha lista, puedes usarla en herramientas como ***Tableau***, ***Power BI*** o ***SQL*** para realizar consultas analíticas directamente.
    - Ejemplo:
        - Analizar ventas por trimestre, comparar periodos fiscales o identificar patrones de temporada, sin necesidad de regenerar la dimensión cada vez.

¿Cuándo debería ser parte del ETL y cuándo no?

Caso	¿Dentro del ETL?	¿Independiente?
Dimensión fecha generada periódicamente (actualización continua).	Sí	No. Actualización frecuente en el ETL.
Dimensión fecha generada una sola vez para un rango amplio.	No	Sí. Se genera una vez y se usa de forma estática.
Necesidad de sincronizarla con datos de otras tablas.	Sí	No. Para asegurar que cubra las fechas necesarias.
Uso exclusivo para análisis visual posterior.	No	Sí. Si no forma parte del flujo de datos principal.

Flujo en un ETL típico (incluyendo dimensión fecha)

1. **Extract**:
   - Extraes los datos de las fuentes originales (bases transaccionales, archivos, etc.).
   - Verificas que las fechas en los datos extraídos están dentro del rango cubierto por la dimensión fecha.
2. **Transform**:
   - Generas la **dimensión fecha** o la actualizas si es necesario.
   - Limpias y estandarizas las fechas en los datos de entrada (por ejemplo, convertir fechas mal formateadas).
   - Creas las relaciones entre los datos transformados y la dimensión fecha usando claves como `date_key`.
3. **Load**:
   - Cargas la **dimensión fecha** y luego las tablas de hechos.
   - En las tablas de hechos, incluyes referencias a la clave `date_key` de la dimensión fecha.

   Ejemplo:
   - Mientras cargas una tabla de hechos de ventas (`FactSales`):

```sql
INSERT INTO FactSales (sales_id, date_key, product_key, sales_amount)
SELECT
    s.sales_id,
    d.date_key,
    p.product_key,
    s.sales_amount
FROM staging_sales s
JOIN DimTime d ON s.sale_date = d.full_date
JOIN DimProduct p ON s.product_id = p.product_id;
```

Conclusión

- **Si lo integras en el ETL**, la dimensión fecha asegura consistencia y sincronización entre tus datos de hechos y los periodos de análisis.
- **Si lo mantienes independiente**, puedes tratarla como una tabla estática para referencias posteriores en análisis y visualizaciones.

Vamos a construir un ejemplo práctico de cómo integrar la **dimensión fecha** en un proceso ETL completo. Usaremos un flujo que incluye:

1. **Extracción (Extract)**: Datos de una tabla de ventas transaccional.
2. **Transformación (Transform)**: Limpieza y mapeo de fechas con la dimensión fecha.
3. **Carga (Load)**: Cargar datos transformados en una tabla de hechos (`FactSales`) referenciando la dimensión fecha (`DimTime`).

Supuestos y Configuración

1. **Tabla de origen (`staging_sales`)**:
    - Contiene datos transaccionales de ventas.
    - Es un staging layer, temporalmente utilizada para ETL.

    ```sql
    CREATE TABLE staging_sales (
        sales_id INT,
        sale_date DATE,
        product_id INT,
        customer_id INT,
        sales_amount DECIMAL(10, 2)
    );
    ```

2. **Dimensión Fecha (`DimTime`)**:
    - Contiene todas las fechas y sus atributos (ya generada con Python o SQL como se explicó antes).

```sql
CREATE TABLE DimTime (
    date_key INT PRIMARY KEY,
    full_date DATE,
    year INT,
    month INT,
    month_name VARCHAR(20),
    day_of_week INT,
    day_name VARCHAR(20),
    is_weekend BOOLEAN,
    is_holiday BOOLEAN
);
```

3. **Tabla de hechos (`FactSales`)**:
    - Recibirá datos procesados, referenciando claves de `DimTime`.

```sql
CREATE TABLE FactSales (
    sales_id INT PRIMARY KEY,
    date_key INT,
    product_key INT,
    customer_key INT,
    sales_amount DECIMAL(10, 2),
    FOREIGN KEY (date_key) REFERENCES DimTime(date_key)
);
```

ETL Completo

1. Extracción (Extract)

Extraemos datos de la tabla de origen (`staging_sales`), asegurándonos de capturar todas las fechas en el rango temporal de análisis.

Ejemplo en SQL:

```sql
SELECT
    sales_id,
    sale_date,
    product_id,
    customer_id,
    sales_amount
FROM staging_sales;
```

---

2. Transformación (Transform)

a. Validar y estandarizar las fechas

- Comprobar que las fechas en `sale_date` son válidas y no están fuera del rango de la dimensión fecha.

```sql
SELECT MIN(sale_date), MAX(sale_date)
FROM staging_sales;
```

- Si hay fechas fuera del rango, actualiza la dimensión fecha para incluirlas.

b. Enlazar con la dimensión fecha (`DimTime`)

- Transformamos las fechas de las transacciones (`sale_date`) para obtener su `date_key` correspondiente en la dimensión fecha.

```
SELECT
    s.sales_id,
    d.date_key,          -- Mapeo a DimTime
    s.product_id,
    s.customer_id,
    s.sales_amount
FROM staging_sales s
JOIN DimTime d ON s.sale_date = d.full_date;
```

c. Identificar problemas comunes

- Fechas nulas o no válidas:

  Reemplaza valores nulos con un `date_key` especial para "fechas desconocidas" (`99999999`):

  ```
  UPDATE staging_sales
  SET sale_date = '1900-01-01'
  WHERE sale_date IS NULL;
  ```

---

3. Carga (Load)

a. Cargar en la tabla de hechos

Insertamos los datos transformados en la tabla `FactSales`.

```
INSERT INTO FactSales (sales_id, date_key, product_key, customer_key,
sales_amount)
SELECT
    s.sales_id,
    d.date_key,
    s.product_id,
    s.customer_id,
    s.sales_amount
FROM staging_sales s
JOIN DimTime d ON s.sale_date = d.full_date;
```

b. Validación después de la carga

Verificamos que los datos en `FactSales` estén correctamente cargados:

```
SELECT
    COUNT(*) AS total_rows,
    SUM(sales_amount) AS total_sales
FROM FactSales;
```

Automatización del ETL

Para un sistema de producción, puedes usar una herramienta ETL o escribir un script en Python que realice este proceso de forma automática. Aquí hay un ejemplo usando **Python** con SQLAlchemy:

```
import pandas as pd
from sqlalchemy import create_engine

# Configuración de la base de datos
engine = create_engine("mysql+pymysql://user:password@localhost/database")

# Extracción
query_extract = "SELECT * FROM staging_sales"
staging_sales = pd.read_sql(query_extract, engine)

# Transformación
dim_time = pd.read_sql("SELECT * FROM DimTime", engine)

# Mapeo de fechas
staging_sales['date_key'] = staging_sales['sale_date'].map(
    dim_time.set_index('full_date')['date_key']
)

# Carga
fact_sales = staging_sales[['sales_id', 'date_key', 'product_id', 'customer_id',
'sales_amount']]
fact_sales.to_sql('FactSales', engine, if_exists='append', index=False)
```

Resultado Final

- **DimTime** tendrá todas las fechas necesarias para el análisis:

  ```
  SELECT * FROM DimTime LIMIT 10;
  ```

- **FactSales** estará completamente cargada, enlazando los datos transaccionales con la dimensión fecha para permitir análisis temporales:

  ```
  SELECT year, month_name, SUM(sales_amount) AS total_sales
  FROM FactSales f
  JOIN DimTime d ON f.date_key = d.date_key
  GROUP BY year, month_name
  ORDER BY year, month;
  ```

Ampliaciones posibles

1. **Integración con más dimensiones**:
   - Puedes incluir dimensiones adicionales como productos, clientes o tiendas para enriquecer el análisis.
2. **Optimización**:

- Usa índices en columnas clave como `date_key` en `FactSales` para mejorar el rendimiento.

3. **Automatización avanzada**:
   - Implementa con Apache Airflow para programar tareas ETL regularmente.

Una **base de staging** (también conocida como área de staging o staging layer) es una parte del entorno de datos donde se almacenan temporalmente los datos extraídos de las fuentes originales antes de transformarlos y cargarlos en el Data Warehouse. Es una etapa intermedia del proceso ETL (**Extract, Transform, Load**) y desempeña un papel fundamental en el flujo de datos.

Características clave de una base de staging:

1. **Temporalidad**:
   - Los datos en esta base suelen ser temporales y se purgan o reemplazan periódicamente, dependiendo de las necesidades del proceso ETL.
2. **Estructura similar a la fuente**:
   - Normalmente, los datos en staging tienen una estructura similar a las fuentes originales, aunque pueden incluir algunas modificaciones para facilitar las transformaciones posteriores.
3. **No diseñada para consultas analíticas**:
   - El objetivo no es realizar análisis en esta base, sino preparar los datos para su limpieza y transformación.
4. **Aislamiento**:
   - Sirve como un entorno intermedio donde los datos están aislados de las fuentes y del Data Warehouse, asegurando que las transformaciones no afecten las fuentes originales.

Ventajas de usar una base de staging:

1. **Desacopla las fuentes de datos del Data Warehouse**:
   - Los datos extraídos se almacenan en staging, reduciendo la carga directa sobre las bases de datos operacionales (fuentes originales).
2. **Facilita la transformación**:
   - Las transformaciones complejas se realizan en staging, dejando el Data Warehouse para el análisis y las consultas optimizadas.
3. **Proporciona consistencia**:
   - Permite consolidar datos de múltiples fuentes en un solo lugar antes de procesarlos.
4. **Auditoría y depuración**:
   - Si algo falla en las transformaciones, los datos en la base de staging pueden ser revisados o reutilizados para repetir el proceso.

¿Cómo se implementa una base de staging?

1. Diseño de la base de staging

- **Tablas brutas**:
  - Se crean tablas con una estructura similar a las fuentes.
  - Ejemplo: Si la fuente tiene una tabla `customer`, la base de staging también tendrá una tabla `stg_customer`.
- **Tablas intermedias**:
  - Opcionalmente, puedes incluir tablas intermedias para aplicar limpiezas parciales (como eliminación de duplicados).

2. Proceso ETL típico con staging

1. **Extracción**:
   - Los datos se copian de las fuentes originales a la base de staging:

```
CREATE TABLE stg_customer AS
SELECT * FROM sakila.customer;
```

2. **Transformación**:
   - Limpieza y enriquecimiento de los datos dentro de la base de staging:
     - Eliminación de duplicados:

```
DELETE FROM stg_customer
WHERE id NOT IN (
    SELECT MIN(id)
    FROM stg_customer
    GROUP BY email
);
```

     - Enriquecimiento (agregar columnas calculadas):

```
ALTER TABLE stg_customer ADD COLUMN full_name VARCHAR(255);
UPDATE stg_customer SET full_name = CONCAT(first_name, ' ',
last_name);
```

3. **Carga**:
   - Una vez que los datos están listos, se mueven al Data Warehouse:

```
INSERT INTO DimCustomer (customer_key, full_name, email, active)
SELECT customer_id, full_name, email, active FROM stg_customer;
```

3. Herramientas para staging

- Puedes usar bases de datos relacionales como MySQL, PostgreSQL o incluso sistemas distribuidos como Amazon Redshift para implementar staging.
- Herramientas ETL (por ejemplo, Talend, Apache NiFi, o Airflow) pueden automatizar la gestión de la base de staging.

Ejemplo en el contexto de Sakila:

Imagina que extraes datos de las tablas transaccionales de Sakila (`rental`, `payment`, `customer`) para analizarlos en un Data Warehouse. Los datos primero pasan por la base de staging:

1. **Tabla de staging `stg_rental`**:
   - Copias todos los registros de la tabla

     ```
     rental
     ```

     de Sakila:

     ```
     CREATE TABLE stg_rental AS
     SELECT * FROM sakila.rental;
     ```

2. **Transformación en staging**:
   - Limpias datos nulos y eliminas columnas no necesarias:

     ```
     ALTER TABLE stg_rental DROP COLUMN last_update;
     UPDATE stg_rental SET return_date = '1900-01-01' WHERE return_date IS
     NULL;
     ```

3. **Carga en el Data Warehouse**:
   - Movemos los datos transformados a la tabla

     ```
     FactRental
     ```

     :

     ```
     INSERT INTO FactRental (rental_id, date_key, customer_key, film_key,
     store_key, staff_key)
     SELECT
         rental_id,
         DATE_FORMAT(rental_date, '%Y%m%d') AS date_key,
         customer_id,
         inventory_id,
         store_id,
         staff_id
     FROM stg_rental;
     ```

Cuándo NO usar una base de staging

Si el volumen de datos es bajo y las transformaciones son mínimas, puede que no necesites una base de staging. En esos casos, los datos pueden transformarse directamente durante la carga al Data Warehouse.

¿Quieres ejemplos más específicos o herramientas para implementar staging? 😊