Author: Madhurima Rawat

Designing ETL Process for Data Warehousing

In this experiment, an ETL process was designed and implemented to migrate data from operational databases to a data warehouse.

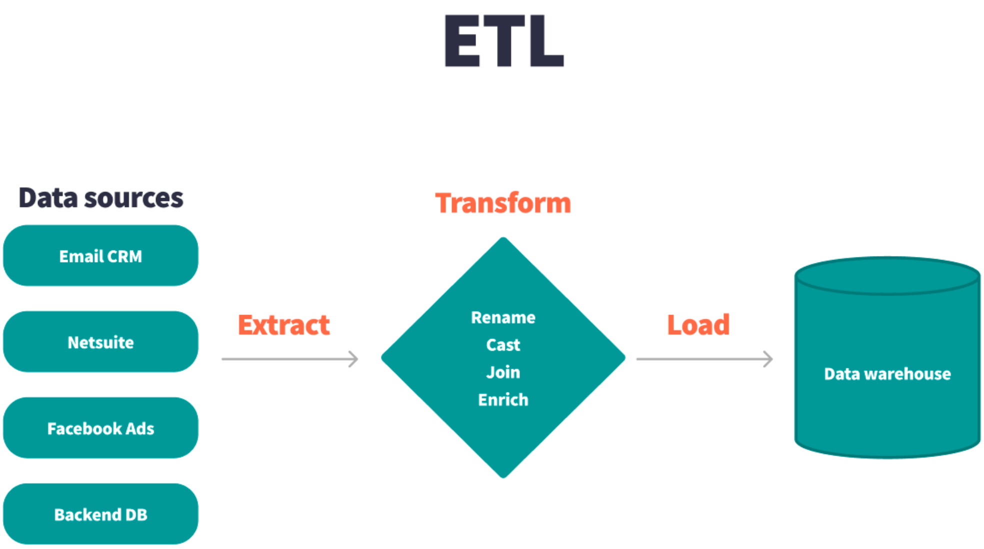
ETL (Extract, Transform, Load) Process in Data Warehousing

Overview of ETL

ETL (Extract, Transform, Load) is a fundamental process in data warehousing that ensures **structured, clean, and consistent** data for **business intelligence, analytics, and reporting**. It helps in integrating data from multiple sources into a centralized data warehouse.

ETL Process with Diagram and Explanation

```
+----------------+   +----------------+   +----------------+
| Extract        | -> | Transform      | -> | Load           |
| (Source Data)  |   | (Clean &       |   | (Data Warehouse|
|                |   | Format)        |   | Storage)       |
+----------------+   +----------------+   +----------------+
```



Step-by-Step Breakdown of ETL Process

1. Extract – Collecting Data from Multiple Sources

The first step involves retrieving data from different sources, such as:

- **Databases** (e.g., MySQL, PostgreSQL)
- **APIs** (e.g., Stripe API for payment transactions)
- **Flat Files** (e.g., CSV, JSON)
- **Streaming Data** (e.g., IoT sensor logs, real-time event streams)

Real-World Example:

A retail company extracts **daily sales records** from:

- **Point-of-sale (POS) system** (transactions stored in MySQL).
- **E-commerce website** (customer purchases from an API).
- **Supplier inventory reports** (CSV files).

2. Transform – Cleaning and Standardizing Data

Once data is extracted, it needs to be cleaned and reformatted to meet business requirements. This step includes:

- **Handling Missing Data** (filling missing values or removing incomplete records).
- **Data Type Conversion** (converting dates into `YYYY-MM-DD` format).
- **Deduplication** (removing duplicate customer records).
- **Aggregation** (calculating total sales per day).

Real-World Example:

- The extracted sales data has missing values in the **customer address column**, so a default value is added.
- Prices from different stores are stored in **different currencies**, so they are converted to a standard currency (USD).
- Duplicate transactions are removed to avoid double-counting revenue.

3. Load – Storing Data in a Data Warehouse

The transformed data is loaded into a **data warehouse** or an **analytical database** for querying and reporting.

- **Batch Loading**: Large amounts of processed data are periodically uploaded (e.g., every 24 hours).
- **Real-Time Loading**: Continuous streaming of transformed data for up-to-date analytics.

Real-World Example:

- The cleaned and formatted sales data is loaded into **Amazon Redshift** or **Google BigQuery**.
- A **business intelligence (BI) tool** like Power BI or Tableau queries the data for sales performance reports.
- The finance team uses the warehouse data for **quarterly revenue analysis**.

Summary of ETL with Real-World Application

ETL Stage	Description	Retail Sales Example
Extract	Collects raw data from multiple sources	Pulls customer purchases from MySQL, API, and CSV files
Transform	Cleans, formats, and prepares data for storage	Converts currency, removes duplicates, aggregates total sales
Load	Saves transformed data in a data warehouse	Loads cleaned data into Amazon Redshift for reporting

The **ETL process ensures high-quality, structured data** for accurate and efficient **business intelligence, forecasting, and decision-making**.

ETL Process: Extract, Transform, and Load in MySQL

This guide explains a **complete ETL process using MySQL**. We will start by **extracting unstructured data**, then **transforming it into a clean and structured format**, and finally **loading it into a data warehouse**.

Step 1: Extract - Creating an Unstructured Table

The first step in ETL is to extract data from different sources. In this case, we create a table `unstructured_operational_data`, which contains **inconsistent, redundant, and unnormalized** data.

SQL Query to Create the Table

```
CREATE TABLE unstructured_operational_data (
    id INT AUTO_INCREMENT PRIMARY KEY,
    customer_name VARCHAR(255),
    product_details TEXT,
    purchase_date VARCHAR(50),
    price VARCHAR(50),
    quantity VARCHAR(50),
    address TEXT
);
```

Explanation

- `id` : A unique identifier for each transaction.
- `customer_name` : Stores the customer's name.
- `product_details` : Contains product name along with its specifications.
- `purchase_date` : Stored as a **string** in various formats (e.g., "2024-02-01", "Feb 2, 2024").
- `price` : Stored as a **string**, needs conversion to numeric type.
- `quantity` : Stored as a **string**, needs conversion to an integer.
- `address` : Stores the customer's address.

Expected Output (Table Structure)

Column Name	Data Type
id	INT (Primary Key)
customer_name	VARCHAR(255)
product_details	TEXT
purchase_date	VARCHAR(50)
price	VARCHAR(50)
quantity	VARCHAR(50)
address	TEXT

Step 2: Insert Sample Data

We now insert **25 rows of unstructured and inconsistent data**, containing **inconsistent date formats, text-based price values, and unnormalized product names**.

SQL Query

```
INSERT INTO unstructured_operational_data
(customer_name, product_details, purchase_date, price, quantity, address)
```

```
VALUES
('John Doe', 'Laptop - Model A', '2024-02-01',
'1000', '1', 'New York, USA'),
('Jane Smith', 'Smartphone - Model X',
'2024-02-02', '700', '2', 'Los Angeles, USA'),
('Ali Khan', 'Tablet - Model B', '02-Feb-24',
'450', '1', 'Mumbai, India'),
('Emily Davis', 'Laptop - Model A', '2024/02/03',
'1000', '1', 'London, UK'),
('Carlos Ruiz', 'Headphones - Wireless',
'2024-02-04', '150', '3', 'Madrid, Spain'),
('Sophia Lee', 'Smartwatch - Series 6',
'2nd Feb 2024', '300', '1', 'Seoul, South Korea'),
('Michael Brown', 'Monitor - 27 inch',
'2024-02-05', '250', '2', 'Berlin, Germany'),
('Linda Green', 'Keyboard - Mechanical',
'05-02-2024', '80', '5', 'Sydney, Australia'),
('David White', 'Mouse - Gaming', '6/2/2024', '60',
'4', 'Toronto, Canada'),
('Olivia Black', 'Laptop - Model C', '2024-02-07',
'1200', '1', 'Paris, France'),
('Ethan Miller', 'Smartphone - Model Y',
'Feb 08 2024', '800', '2', 'Dubai, UAE'),
('Mia Wilson', 'Headphones - Wired',
'9-2-2024', '100', '2', 'Rome, Italy'),
('Lucas Adams', 'Tablet - Model D',
'10th Feb 2024', '500', '1', 'Singapore'),
('Ava Scott', 'Laptop - Model B', '11-02-24', '1100',
'1', 'Hong Kong'),
('Daniel Hall', 'Smartwatch - Series 7',
'12/02/2024', '350', '1', 'Bangkok, Thailand'),
('Emma Carter', 'Monitor - 24 inch',
'Feb 13, 2024', '200', '2', 'Amsterdam, Netherlands'),
('Noah Phillips', 'Keyboard - RGB',
'14-02-2024', '90', '3', 'Zurich, Switzerland'),
('Liam Robinson', 'Mouse - Wireless',
'2024-02-15', '75', '2', 'Tokyo, Japan'),
('Charlotte Martinez', 'Laptop - Model D', '16/02/24',
'1300', '1', 'Beijing, China'),
('James Anderson', 'Smartphone - Model Z', 'Feb 17 2024', '
900', '2', 'Mexico City, Mexico'),
('Isabella Thomas', 'Headphones - Bluetooth',
'18-02-2024', '120', '3', 'Cairo, Egypt'),
('Benjamin Harris', 'Tablet - Model E', '2024-02-19', '550',
'1', 'Johannesburg, South Africa'),
('Amelia Martin', 'Laptop - Model E',
'Feb 20 2024', '1400', '1', 'Kuala Lumpur, Malaysia'),
('William Thompson', 'Smartwatch - Series 8',
'21-02-2024', '400', '1', 'Hanoi, Vietnam'),
('Sophie Walker', 'Monitor - 32 inch', '22-02-2024',
'300', '2', 'Wellington, New Zealand');
```

Explanation

- Different **date formats** (e.g., "2024-02-01", "Feb 2, 2024", "02-Feb-24").
- **Price stored as text**, needs conversion.
- **Product details are unstructured**, containing model names.

Expected Output (Sample Data in Table)

id	customer_name	product_details	purchase_date	price	quantity	address
1	John Doe	Laptop - Model A	2024-02-01	1000	1	New York, USA
2	Jane Smith	Smartphone - Model X	2024-02-02	700	2	Los Angeles, USA
3	Ali Khan	Tablet - Model B	02-Feb-24	450	1	Mumbai, India

Step 3: Transform - Cleaning and Standardizing Data

Now, we transform the extracted data to ensure **consistency, correctness, and normalization**.

1. Extracting Product Name and Formatting Dates

```
CREATE TABLE transformed_data AS
SELECT
    id,
    customer_name,
    SUBSTRING_INDEX(product_details, ' - ', 1)
    AS product_name, -- Extracts product name before '-'
    purchase_date,
    CAST(price AS DECIMAL(10,2)) AS price,
    -- Converts price to decimal format
    CAST(quantity AS INT) AS quantity,
    -- Converts quantity to integer
    address
FROM unstructured_operational_data;
```

Explanation

- **Extracts the product name** from `product_details` (before the `-` separator).

- **Converts price from text to numeric** (`DECIMAL(10,2)` ensures proper decimal values).
- **Converts quantity to an integer** to allow numerical operations.

Expected Output (Before Transformation)

id	customer_name	product_name	purchase_date	price	quantity	address
1	John Doe	Laptop	2024-02-01	1000.00	1	New York, USA
2	Jane Smith	Smartphone	2024-02-02	700.00	2	Los Angeles, USA
3	Ali Khan	Tablet	02-Feb-24	450.00	1	Mumbai, India

2. Error: Incorrect Date Format Conversion

Issue

```
CREATE TABLE transformed_data AS
SELECT
    id,
    customer_name,
    SUBSTRING_INDEX(product_details, ' - ', 1) AS product_name,
    STR_TO_DATE(purchase_date, '%Y-%m-%d') AS formatted_date,
    CAST(price AS DECIMAL(10,2)) AS price,
    CAST(NULLIF(TRIM(quantity), '') AS SIGNED) AS quantity,
    address
FROM unstructured_operational_data;
```

Error:

```
ERROR 1411 (HY000): Incorrect datetime value:
'02-Feb-24' for function str_to_date
```

Reason:

- `STR_TO_DATE(purchase_date, '%Y-%m-%d')` assumes all dates are in `YYYY-MM-DD` format.
- Some dates are formatted as `02-Feb-24` , `2nd Feb 2024` , or `05-02-2024` , which do not match `%Y-%m-%d` .

3. Attempted Fix: Handling Multiple Date Formats

```
CREATE TABLE transformed_data AS
SELECT
    id,
    customer_name,
    SUBSTRING_INDEX(product_details, ' - ', 1) AS product_name,
    CASE
        WHEN purchase_date LIKE '%-%-%' AND CHAR_LENGTH(purchase_date) = 10
            THEN STR_TO_DATE(purchase_date, '%Y-%m-%d')
            -- Handles '2024-02-05'
        WHEN purchase_date LIKE '%/%/%'
            THEN STR_TO_DATE(purchase_date, '%Y/%m/%d')
            -- Handles '05/02/2024'
        WHEN purchase_date LIKE '%-%-%' AND CHAR_LENGTH(purchase_date) = 9
            THEN STR_TO_DATE(purchase_date, '%d-%b-%y')
            -- Handles '05-Feb-24'
        WHEN purchase_date LIKE '% % %'
            THEN STR_TO_DATE(purchase_date, '%d %M %Y')
            -- Handles '2 Feb 2024'
        ELSE NULL
    END AS formatted_date,
    CAST(price AS DECIMAL(10,2)) AS price,
    CAST(NULLIF(TRIM(quantity), '') AS SIGNED) AS quantity,
    address
FROM unstructured_operational_data;
```

Error:

```
ERROR 1411 (HY000): Incorrect datetime value:
'2nd Feb 2024' for function str_to_date
```

Reason:

- MySQL does not recognize ordinal suffixes (`st` , `nd` , `rd` , `th`) in dates like `"2nd Feb 2024"` .

4. Correction: Removing Ordinal Suffixes Before Conversion

```
UPDATE unstructured_operational_data
SET purchase_date = REPLACE(REPLACE(REPLACE(REPLACE
(purchase_date, 'st', ''),
'nd', ''), 'rd', ''), 'th', '');
```

Explanation:

- Removes suffixes (`st` , `nd` , `rd` , `th`) from dates to ensure compatibility with `STR_TO_DATE()` .

Expected Output (After Update)

id	purchase_date
1	2 Feb 2024
2	05-Feb-24
3	2024-02-01

5. Final Fix: Handling All Date Formats Using Regular Expressions

```
CREATE TABLE transformed_data AS
SELECT
    id,
    customer_name,
    SUBSTRING_INDEX(product_details, ' - ', 1) AS product_name,
    CASE
        WHEN purchase_date REGEXP
        '^[0-9]{4}-[0-9]{2}-[0-9]{2}$'
            THEN STR_TO_DATE(purchase_date, '%Y-%m-%d')
            -- Handles '2024-02-05'
        WHEN purchase_date REGEXP '^[0-9]{2}/[0-9]{2}/[0-9]{4}$'
            THEN STR_TO_DATE(purchase_date, '%d/%m/%Y')
            -- Handles '05/02/2024'
        WHEN purchase_date REGEXP '^[0-9]{2}-[A-Za-z]{3}-[0-9]{2}$'
            THEN STR_TO_DATE(purchase_date, '%d-%b-%y')
            -- Handles '05-Feb-24'
        WHEN purchase_date REGEXP '^[0-9]{2}-[0-9]{2}-[0-9]{4}$'
            THEN STR_TO_DATE(purchase_date, '%d-%m-%Y')
            -- Handles '05-02-2024'
        WHEN purchase_date REGEXP '^[0-9]{1,2} [A-Za-z]{3} [0-9]{4}$'
            THEN STR_TO_DATE(purchase_date, '%d %b %Y')
            -- Handles '5 Feb 2024'
        ELSE NULL
    END AS formatted_date,
    CAST(price AS DECIMAL(10,2)) AS price,
    CAST(NULLIF(TRIM(quantity), '') AS SIGNED) AS quantity,
    address
FROM unstructured_operational_data;
```

Expected Output (Final Transformed Data)

id	customer_name	product_name	formatted_date	price	quantity	address
1	John Doe	Laptop	2024-02-01	1000.00	1	New York, USA

id	customer_name	product_name	formatted_date	price	quantity	address
2	Jane Smith	Smartphone	2024-02-02	700.00	2	Los Angeles, USA
3	Ali Khan	Tablet	2024-02-03	450.00	1	Mumbai, India

Summary of Errors and Fixes

Issue	Error Message	Cause	Fix Applied
Incorrect date format	`ERROR 1411 (HY000)`	Date format mismatch	Used `CASE` with multiple formats
Dates with ordinal suffixes	`ERROR 1411 (HY000)`	Dates like `2nd Feb 2024`	Used `REPLACE()` to remove suffixes
Truncated incorrect date value	`ERROR 1292 (22007)`	MySQL could not parse date	Used `REGEXP` to match valid formats

This completes the **data transformation step**, ensuring **cleaned, standardized, and structured data** for loading into the data warehouse.

Step 4: Load - Storing Clean Data in a Warehouse Table

We now load the **transformed data** into the final **fact table** for business intelligence analysis.

SQL Query

```
CREATE TABLE fact_sales (
    sale_id INT AUTO_INCREMENT PRIMARY KEY,
    customer_name VARCHAR(255),
    product_name VARCHAR(255),
    sale_date DATE,
    total_price DECIMAL(10,2),
    address TEXT
);
```

Explanation

- Stores structured and cleaned sales data.
- Ensures numeric and date consistency.

Inserting Cleaned Data into Final Table

```
INSERT INTO fact_sales (customer_name, product_name,
sale_date, total_price, address)
SELECT customer_name, product_name,
STR_TO_DATE(purchase_date, '%Y-%m-%d'), (price * quantity), address
FROM transformed_data;
```

Expected Output

sale_id	customer_name	product_name	sale_date	total_price	address
1	John Doe	Laptop	2024-02-01	1000.00	New York, USA
2	Jane Smith	Smartphone	2024-02-02	1400.00	Los Angeles, USA
3	Ali Khan	Tablet	2024-02-03	450.00	Mumbai, India

Final Summary of ETL Process

ETL Stage	Process	Changes Made
Extract	Collected unstructured sales data	Data stored with mixed formats, inconsistencies
Transform	Cleaned and formatted data	Converted dates, standardized names, fixed numbers
Load	Stored structured data in a fact table	Data ready for analytics and reporting

This completes a **full ETL pipeline using MySQL**, making data ready for **business intelligence and reporting**.