Task

I intend to outline the design process for an ETL structure containing columns and tables that can be used to answer the following questions:

1. Daily/Weekly/Monthly Dynamic of Revenue and Orders per Country, Region. Only Revenue from completed orders should be counted.
2. What were the top 10 companies by number of orders in the last week?
3. How many companies signed up?
4. How many monthly active users do we have (at least with 1 completed order)

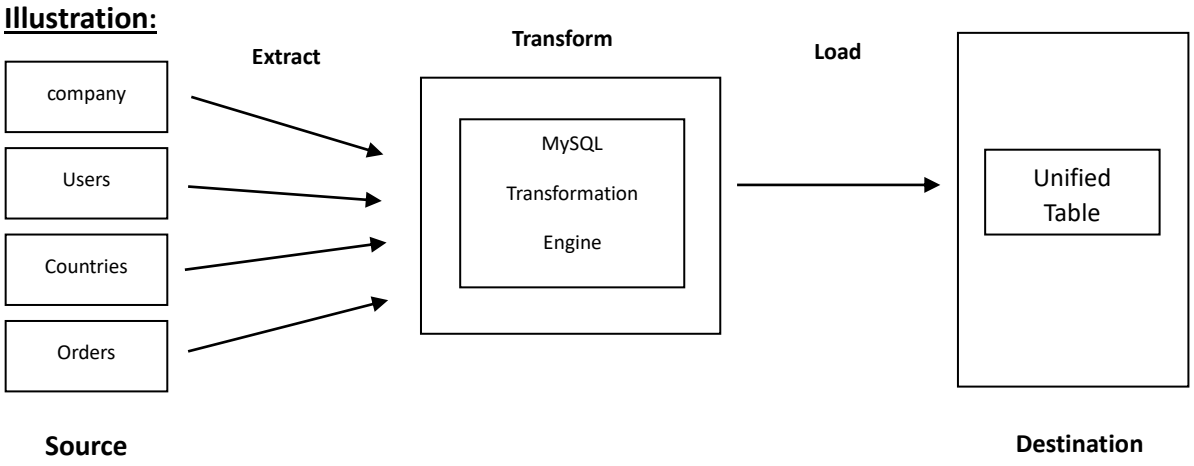
Solution:

Choice Tables: The choice tables are **Company**, **Users**, **Countries**, and **Orders** tables respectively.

Assumptions:

1. There's data completeness and consistency across the various tables which implies consistency in formats like **date formats, text formats,** and **numbers**.
2. The join keys such as **user_id, admin_id** is unique and valid ensuring consistency as the primary and foreign keys.
3. In the context of our analytics engineering process, the selection of tables for the ETL pipeline is directly guided by the critical questions that the unified table aims to address.
4. Given that this process supports continuous reporting, it's crucial to design it for regular updates. Hence, Incremental updates, rather than full reloads would optimize efficiency and minimize resource consumption.
5. The business closes the financial books each day by **11:59 pm** to permit table refreshes; Hence, I have permissions to create and modify stored procedures as well as setup alerts.
6. The choice of ETL programming is only MySQL and the MySQL Event Scheduler is enabled.
7. The query performance of the Datawarehouse is optimal.
8. All six tables reside within the same schema of the Datawarehouse. This schema consistency simplifies data management and maintenance.

Illustration:



ETL Code:

Step 1: Extract data from the source tables: **Company**, **Users**, **Countries**, and **Orders**.

Step 2: Transform the data from the source tables by joining the relevant tables based on the relationship between them using **admin_id**, **user_id,** and **country_code.**

Step 3: Load the transformed data into a table called **UNIFIED_TABLE,** which contains all the key data needed to answer the questions.

Create the unified table by joining the relevant tables on the same schema.

```
CREATE TABLE unified_table AS
SELECT
    c.admin_id,
    c.country_code,
    c.name AS company_name,
    c.created_at AS company_created_at,
    u.user_id,
    u.full_name AS user_name,
    u.created_at AS user_created_at,
    co.country_name,
    co.region,
    o.order_id,
    o.created_at AS order_created_at,
    o.revenue AS order_revenue

FROM company c
JOIN users u ON c.admin_id = u.admin_id
JOIN countries co ON c.country_code = co.country_code
JOIN orders o ON u.user_id = o.user_id;
```

This **unified_table** contains all the key data we need to answer the questions, including company information, user information, country information, and order details.

Answers to the questions:

1. Daily/Weekly/Monthly Dynamic of Revenue and Orders per Country, Region. Only Revenue from completed orders should be counted.

```
-- Revenue and orders per country per day

SELECT
    DATE(order_created_at) AS order_date,
    country_name,
    region,
    SUM(order_revenue) AS total_revenue,
    COUNT(order_id) AS total_orders
FROM unified_table
GROUP BY order_date, country_name, region
ORDER BY order_date, total_revenue DESC;
```

```sql
-- Revenue and orders per country per week

SELECT
    DATE_TRUNC('week', order_created_at) AS order_week,
    country_name,
    region,
    SUM(order_revenue) AS total_revenue,
    COUNT(order_id) AS total_orders
FROM unified_table
GROUP BY order_week, country_name, region
ORDER BY order_week, total_revenue DESC;


-- Revenue and orders per country per month

SELECT
    DATE_TRUNC('month', order_created_at) AS order_month,
    country_name,
    region,
    SUM(order_revenue) AS total_revenue,
    COUNT(order_id) AS total_orders
FROM unified_table
GROUP BY order_month, country_name, region

ORDER BY order_month, total_revenue DESC;
```

2. What were the top 10 companies by number of orders in the last week?

```sql
SELECT
    c.name AS company_name,
    COUNT(order_id) AS total_orders

FROM unified_table
WHERE order_created_at >= DATE_SUB(CURDATE(), INTERVAL 1 WEEK)
GROUP BY c.name
ORDER BY total_orders DESC

LIMIT 10;
```

3. How many companies signed up?

```sql
SELECT COUNT(DISTINCT admin_id) AS total_companies

FROM unified_table;
```

4. How many monthly active users do we have (at least with 1 completed order)

```sql
SELECT COUNT(DISTINCT user_id) AS monthly_active_users
FROM unified_table

WHERE order_created_at >= DATE_SUB(CURDATE(), INTERVAL 1 MONTH);
```

Incremental Load:

As earlier mentioned, it's assumed that the table would be used for continuous reporting thereby necessitating the need for implementing incremental load. The pros of using incremental load are:

a) The performance and refresh process is fast since only new data would be loaded.
b) This process would ensure there's latest data without loosing historical information.
c) As the volume of orders keeps growing, the incremental load approach will continually be efficient as it only processes new data

Step 1: A stored procedure called **load_incremental_unified_table** is created which creates a new temproary table called **new_unified_table** that holds new data to be added to the **unified_table**. The temporary table picks data from the same source as **unified_table** but only data where the order date is higher than the maximum order date of the **unified_table.** This data is then inserted into the unified_table.

```
-- Create a stored procedure for the incremental load

DELIMITER $$
CREATE PROCEDURE load_incremental_unified_table()
BEGIN

    -- Create a temporary table to hold the new data

    CREATE TEMPORARY TABLE new_unified_data AS
    SELECT
        c.admin_id,
        c.country_code,
        c.name AS company_name,
        c.created_at AS company_created_at,
        u.user_id,
        u.full_name AS user_name,
        u.created_at AS user_created_at,
        co.country_name,
        co.region,
        o.order_id,
        o.created_at AS order_created_at,
        o.revenue AS order_revenue
    FROM company c
    JOIN users u ON c.admin_id = u.admin_id
    JOIN countries co ON c.country_code = co.country_code
    JOIN orders o ON u.user_id = o.user_id

    WHERE o.created_at >= (SELECT MAX(order_created_at) FROM
    UNIFIED_TABLE);

    -- Insert the new data into the UNIFIED_TABLE
        INSERT INTO unified_table
        SELECT * FROM new_unified_data;

        -- Drop the temporary table
        DROP TABLE new_unified_data;
    END $$
    DELIMITER ;
```

Step 2: An event called **daily_load_incremental_unified_table** is created and scheduled to run the **load_incremental_unified_table** stored procedure everyday at 11:59pm starting April 8th, 2024.

```
-- Create a scheduled event to run the
load_incremental_unified_table() procedure daily at 11:00 PM

CREATE EVENT daily_load_incremental_unified_table
ON SCHEDULE EVERY 1 DAY
STARTS '2024-04-08 23:59:00'

DO CALL load_incremental_unified_table();
```