# Techniques used to transform data, part 2

So far, you've learned that data transformations enable you to make changes to your data so it is usable for analysis and visualization. You also learned about useful transformation techniques in SQL for handling aggregation, deduplication, derivation, and filtering. In this reading, you'll learn more techniques for transforming data.

## Data transformation types

You won't always receive data in a ready-to-use format, so you may have to transform it depending on your purpose. Many of the previous methods you learned for SQL can be applied to the techniques you will learn about in this reading. These techniques can be used on large datasets:

- Data integration with unions
- Data joining with joins
- Data splitting
- Formatting data with concatenation

### Data integration

Data integration is the combination of rows from two or more tables to create a single dataset. Integrating data is useful for data spread across multiple tables, or databases that need a unified view.

For example, an analyst needs to work with sales data for two different years. The sales data for one year is in one table, and the second year is in another. The analyst needs a combined view to perform analysis to show how sales have changed over time.

The analyst writes a **SELECT** statement that takes the product name and sales amount from the first year's table, and writes a **UNION** statement to join the product name and sales amount from the second year's table.

Unset

```
SELECT product_name, sales_amount FROM sales_2021

UNION

SELECT product_name, sales_amount FROM sales_2022;
```

If the sales_2021 and sales_2022 tables contain these rows of data:

Unset

```
| product_name | sales_amount |
|--------------|--------------|
| Product A    |     1500     |
| Product B    |     1000     |
| Product C    |     500      |
| Product D    |     800      |
```

Unset

```
| product_name | sales_amount |
|--------------|--------------|
| Product A    |     1800     |
| Product B    |     1100     |
| Product C    |     700      |
| Product E    |     1200     |
```

Then the query will return this result:

Unset

```
| product_name | sales_amount |
|--------------|--------------|
| Product A    |     1500     |
| Product B    |     1000     |
| Product C    |     500      |
| Product D    |     800      |
| Product A    |     1800     |
| Product B    |     1100     |
| Product C    |     700      |
| Product E    |     1200     |
```

The SQL **UNION** operator integrates data from multiple tables by combining the results of one or more **SELECT** statements. Each **SELECT** statement in a **UNION** must have the same columns in the same order, and the columns must have the same datatypes in each underlying table. The **UNION** operator selects only distinct values by default, meaning if there's a row with the same product_name and sales_amount in both tables, it will appear only once in the result.

## Data joining

Joins in SQL combine rows from two or more tables based on the related columns between them. If you have several tables that contain information you need, use a **JOIN** statement to combine the data into one table. There needs to be a related column between the tables you're joining. The columns don't need to have the same name, but they do need to contain the same data and data type.

For example, an analyst wants to determine if a customer is more likely to purchase products if they're on the company email list. Customer information, like email address and order numbers, are in one table, but descriptions and categorizations of products bought are in another table. Both columns contain a customer_id column.

The analyst writes a **SELECT** statement to take the order id and order date from the orders table, and joins the customer name from the customers table on the customer_id column.

Unset

```
SELECT orders.orders_id, customers.customer_name,
orders.order_date FROM orders JOIN customers ON orders.customer_id
= customers.customer_id;
```

If the orders and customers table contain these rows of data:

Unset

```
| orders_id | customer_id | order_date |
|-----------|-------------|------------|
| 1         | 101         | 2023-01-01 |
| 2         | 102         | 2023-02-01 |
| 3         | 103         | 2023-03-01 |
| 4         | 101         | 2023-04-01 |
```

Unset

```
| customer_id | customer_name |
|-------------|---------------|
| 101         | John Doe      |
| 102         | Jane Doe      |
| 103         | Jim Bean      |
```

Then the query will return this result:

Unset

```
| orders_id | customer_name | order_date |
|-----------|---------------|------------|
| 1         | John Doe      | 2023-01-01 |
| 2         | Jane Doe      | 2023-02-01 |
| 3         | Jim Bean      | 2023-03-01 |
| 4         | John Doe      | 2023-04-01 |
```

## Data splitting

Data splitting is when you divide data within a column to create two or more columns. Sometimes, data arrives in a combined format, but needs to be stored separately for better analysis or clarity. Data splitting is a useful technique for extracting important information from a column. Analysts often use this approach for extracting product codes from descriptions.

If an analyst wants to extract data from product_code SKU092023, the first field would return a '09' and the second would return '2023' , which represents month and year of sale.

Unset

```
SELECT SUBSTRING(product_code, 4, 2) as product_month,
SUBSTRING(product_code, 6, 4) as product_year from product_table
```

If the product_table contains these rows of data:

Unset

```
| product_id | product_code | product_name |
|------------|--------------|--------------|
| 1          | SKU092023    | Product A    |
| 2          | SKU122022    | Product B    |
| 3          | SKU082021    | Product C    |
| 4          | SKU072020    | Product D    |
| 5          | SKU042019    | Product E    |
```

Then the query will return the this result:

Unset

```
| product_month | product_year |
|---------------|--------------|
| 09            | 2023         |
| 12            | 2022         |
| 08            | 2021         |
| 07            | 2020         |
| 04            | 2019         |
```

## Formatting data

Formatting data involves changing the presentation of data, like modifying text cases or merging columns. Formatting data creates uniformity and contributes to better reporting because the data is standardized. Imagine a dataset that contains data in lower case, upper case, and a mix of both. If you're using this data to create a dashboard, these inconsistencies will feed into the visualization and create confusion for your audience.

Consider a table called donor_table containing information on donor name and contribution amount to a charity event. If you wanted to create a bar graph associated with each donor_name, the name case would be inconsistent and hard to read:

Unset

```
| donor_name    | contribution_amt_USD |
|---------------|----------------------|
| JOHN DOE      | 1000                 |
| jane doe      | 500                  |
| MiKe Black    | 750                  |
| SARAH WHITE   | 1200                 |
| daniel GREEN  | 600                  |
| AMY o’connell | 300                  |
| RACHEL Brown  | 450                  |
| aLan smith    | 900                  |
```

The **CONCAT** statement is useful for merging columns with data better suited to be combined. For example, if you have columns with first name, last name, and birthday, you can combine all three columns to create a unique ID. This will make it easier to identify duplicates later, but it will also reduce the number of columns you’ll see in the final view.

Unset

```
SELECT CONCAT(first_name, ‘’, last_name, ‘’, birthdate) AS
unique_id)) FROM employees;
```

If the employees table contains these rows of data:

Unset

```
| id | first_name | last_name      | product_name |
|----|------------|--------------|--------------|
| 1  | John       | Doe            | 1990-01-01   |
| 2  | Jane       | Smith          | 1985-06-15   |
| 3  | Alan       | Johnson        | 1978-12-12   |
| 4  | Mary       | Lee            | 1992-04-03   |
| 5  | Jack       | White          | 1982-09-10   |
```

Then the query will return this result:

Unset

```
| unique_id           |
|---------------------|
| JohnDoe19900101     |
| JaneSmith19850615   |
| AlanJohnson19781212 |
| MaryLee19920403     |
| JackWhite19820910   |
```

## Key takeaways

Techniques like unions and joins, and splitting and formatting data play a vital role in preparing and transforming data for analysis.

These techniques ensure that data is in a usable format for analysis by helping to reduce unnecessary information, so that analysts can provide meaningful insights and impactful visualizations.

## Resources for more information

The following resource further explores how you can collect data effectively:

- Solutions marketer and MBA Firoj Alam's perspective on the importance of proper formatting, and how it impacts data collection: Importance of Proper Formatting in Data Collection