Data Query Language

Data Query Language (**DQL**) allows us to **communicate** with **databases**, facilitating precise extraction and manipulation of data.

The **SELECT** keyword is used to **retrieve data from a database,** essentially forming the foundation of any data extraction query.

The **WHERE** keyword is used to **retrieve data conditionally**, enabling more precise and targeted data extraction.

Querying data with SQL

Here we have a table of data. Some data may be more relevant to us than others. For example, we may want to:

- Get data where the `Price` is **more than $0.22**.
- Get all data for the month of **February**.
- **Sort** data based on `Quality`.
- **Group** data together using `Market_ID`.
- **Change dates** to days of the week.
- **Change** the price into another currency.

DQL is a set of SQL commands that allows us to filter, organise, and retrieve **specific data** based on various **criteria**. This is known as **querying** a database.

| Date	Quality	Market_ID	Price ($)
2006-01-15	High	11	0.2235
2006-01-15	Low	56	0.2145
2006-01-15	Good	24	0.2055
2006-01-15	Excellent	77	0.1796
2006-02-15	Low	21	0.2299
2006-02-15	Good	61	0.2299
2006-02-15	Fair	34	0.2499
2006-02-15	Fair	28	0.1839
2006-03-15	High	67	0.236
2006-03-15	Good	82	0.235
2006-03-15	Excellent	24	0.205
2006-03-15	High	28	0.185
2006-04-15	Low	56	0.24
2006-04-15	Excellent	82	0.238
2006-04-15	Good	75	0.2321
2006-04-15	Good	75	0.2321
2006-05-15	Fair	67	0.2798

Querying with SQL

SELECT retrieves data from a database. We can use tools like filters and functions with **SELECT** to manipulate, clean, and organise data.

Manipulating or **changing** data adds value to data by transforming data or extracting more information from existing data.

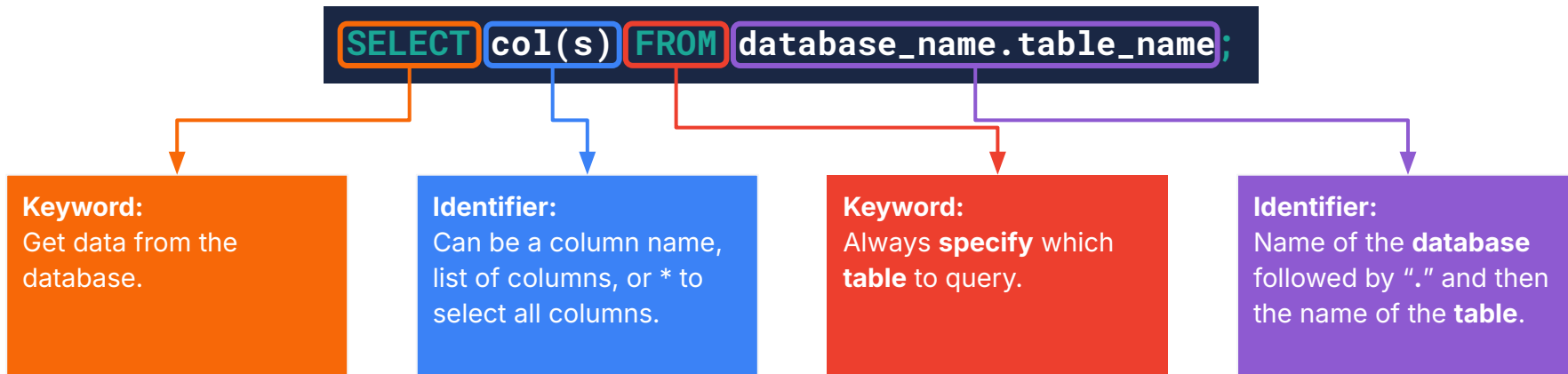**Cleaning** data improves the quality of our data.

Organising data makes it simpler to work with and understand.

Together, these tools help us to **understand data**.
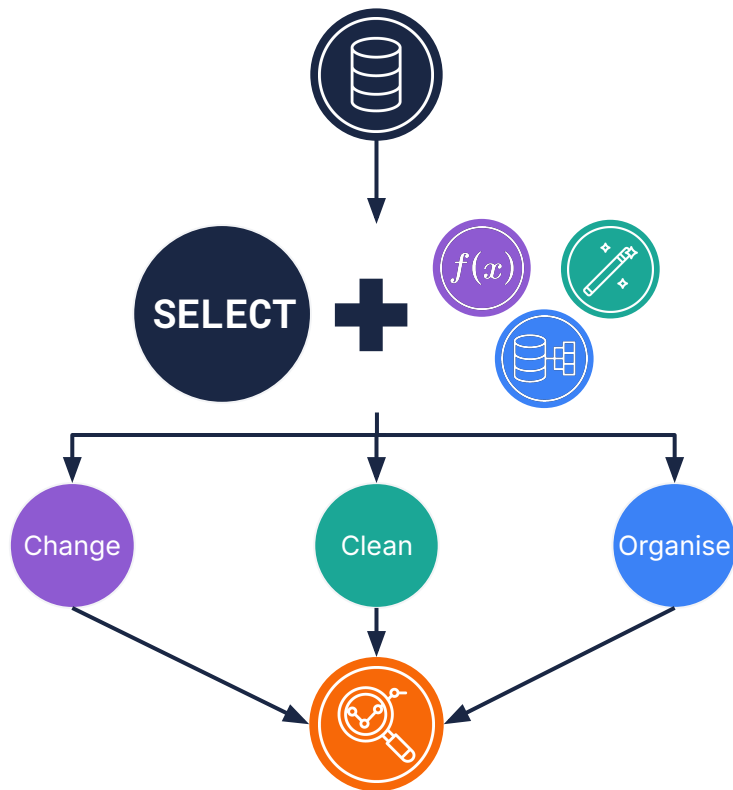
How to query data: SELECT

The **SELECT** statement retrieves a column, list of columns, or all columns from a specific table in a specific database and outputs a **results set**.

```
SELECT col(s) FROM database_name.table_name;
```

Keyword:
Get data from the database.

Identifier:
Can be a column name, list of columns, or * to select all columns.

Keyword:
Always **specify** which **table** to query.

Identifier:
Name of the **database** followed by "**.**" and then the name of the **table**.

When we use **SELECT**, we're not creating a new table in our database, but rather generating a **temporary**, read-only **results set** that contains the data we've requested.

How to query data: FROM

FROM specifies **which table** to query from since databases often have more than one table.

Syntax: `SELECT col(s) FROM Database.Table_3;` will retrieve data only from Table_3:

Database (db)

Table_1		
col1	**col2**	**col3**
x	34	s
y	73	m
z	22	l
w	12	m

Table_2		
col1	**col2**	**col3**
z	68	s
x	1	l
w	7	l
y	56	m

Table_3		
col1	**col2**	**col3**
x	7	s
x	42	s
z	9	s
x	7	s

Note: These are bad naming examples, but we use them here to show how queries work. Always try to use descriptive titles.

SELECT one column

SELECT can be used to retrieve data from a **single** column.

Syntax: `SELECT col FROM db.table;`

Table_1

col1	col2	col3
x	34	s
y	73	m
z	22	l
w	12	m

```
SELECT
    col1
FROM
    db.Table_1;
```

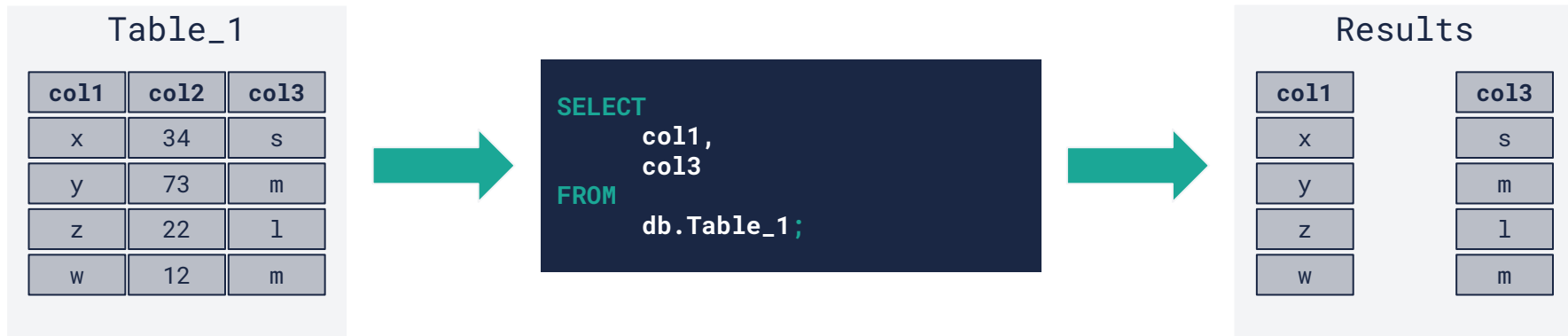### Results

col1
x
y
z
w

In our **results set**, we have retrieved data from a **single column**, col1.

SELECT multiple columns

SELECT can retrieve data from a comma-separated **list** of **columns**.

Syntax: `SELECT col1, col2, … FROM db.table;`

Table_1

col1	col2	col3
x	34	s
y	73	m
z	22	l
w	12	m

```
SELECT
      col1,
      col3
FROM
      db.Table_1;
```

Results

col1		col3
x		s
y		m
z		l
w		m

In this example, a **list** of **columns** (col1, col3) is specified. They can be in any order and are separated by commas. Our result has data from col1 and col3.

SELECT all columns

SELECT * retrieves data from **all** the columns available in a table.

Syntax: `SELECT * FROM db.table;`

Table_1

col1	col2	col3
x	34	s
y	73	m
z	22	l
w	12	m

```
SELECT
        *
FROM
        db.Table_1;
```

Results

col1	col2	col3
x	34	s
y	73	m
z	22	l
w	12	m

Using the asterisk (*) with **SELECT** acts as a **wildcard** and represents **all columns**. Using **SELECT** * with Table_1, we retrieve **all columns** from Table_1 in our results set.

LIMIT

LIMIT restricts the number of rows that are returned in the results set.

Syntax: `SELECT * FROM db.table LIMIT number of rows;`

Table_1

col1	col2	col3
x	34	s
y	73	m
z	22	l
w	12	m

```
SELECT
    *
FROM
    db.Table_1
LIMIT 2;
```

Results

col1	col2	col3
x	34	s
y	73	m

Tables may have thousands or millions of records, so limiting the number of records allows you to **quickly view** your **results**.

SELECT DISTINCT

When working with databases, we often encounter duplicate data. Sometimes, this repetition can be useful or necessary. However, there are instances where we want to eliminate duplicates to gain a clearer picture of our data. This is where the **SELECT DISTINCT** keyword is used.

SELECT DISTINCT eliminates rows where data are duplicated. Like **SELECT**, we can check if data in specific column(s) or all columns are duplicated, and remove them.

SELECT DISTINCT doesn't change the original table, it only affects the query's **results**.

A **common use** of **SELECT DISTINCT** is to remove all duplicates from a table, using * or from a specific set of columns.

Syntax: `SELECT DISTINCT col(s) FROM db.table;`

Using DISTINCT

DISTINCT is used with **SELECT** to only return **unique** (**non-duplicated**) values in a specified column.

Table_2

col1	col2	col3
z	68	s
x	1	l
w	7	l
y	56	m

```
SELECT DISTINCT
      col3
FROM
      db.Table_2;
```

Results

col3
s
l
m

Since 'l' appears in col3 twice, only the unique values, 's', 'm', and 'l' in col3 are listed, with the additional 'l' removed.
Note: Only the specified column is returned.

DISTINCT on multiple columns

When used with multiple columns, the **DISTINCT** keyword returns **unique combinations** of values across **all** the listed columns. If the same **combination** of values appears in more than one row, only one **distinct** row is returned.

Table_3

col1	col2	col3	
x	7	s	1
x	42	s	2
z	9	s	3
x	7	s	4

```
SELECT DISTINCT
        col1,
        col3
FROM
        db.Table_3;
```

Results

col1		col3	
x		s	1
z		s	2

col1		col3	
x		s	1
x		s	2
z		s	3
x		s	4

Step 1: SELECT only col1 and col3. **DISTINCT** checks these two columns for pairs of unique combinations. **x** and **s** are unique, **z** and **s** are unique.

Step 2: Only one **distinct** copy of each unique combination is included. **(x** and **s)** and **(z** and **s)** are both unique combinations.

DISTINCT *

When used with *, **DISTINCT** removes all rows that are exact duplicates.

Table_3

col1	col2	col3	
x	7	s	1
x	42	s	2
z	9	s	3
x	7	s	4

```
SELECT DISTINCT
        *
FROM
    db.Table_3;
```

Results

col1	col2	col3	
x	7	s	1
x	42	s	2
z	9	s	3

- Rows 1 and 4 have identical values in col1, col2, and col3, so they are duplicates.
- While rows 1 and 2 and 4 all have **x** in col1 and **s** in col3, they have different values in col2, so the combination of **x**, **42**, and **s** in row 2 is different to **x**, **7**, and **s**.

DISTINCT

While **DISTINCT** is a powerful tool for managing duplicate data, it can **slow down** your query if used on **large** tables.

SQL has to compare every row with every other row to identify duplicates, which can be **time-consuming**.

So use **DISTINCT** with care, when the need to remove duplicates from our results set outweighs the potential **performance cost**.
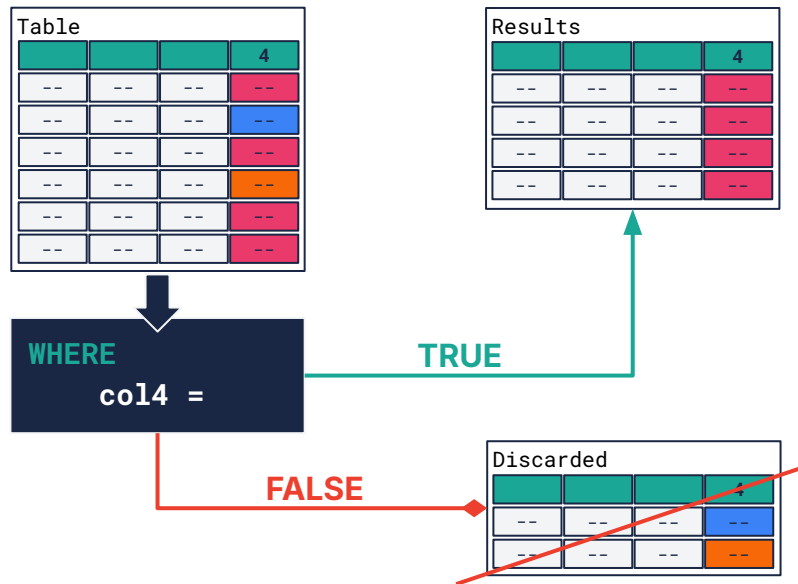
WHERE

> The **WHERE** clause in SQL is another powerful tool for filtering our data where a specific condition is **TRUE**. **WHERE** filters the results to include only the rows where the value in a column equals `<value>`.

In this example, only the rows where `col4 =` ▮ are in our results, while rows with `col4 =` ▮ and ▮ are discarded.

Essentially, **WHERE** acts as a gatekeeper, deciding which rows from the original table meet the condition and should be included in the query result.

For example, in a medical clinic, we may want to only see the data **where** patients are **currently admitted**, or we may want the data **where** patients are **children**.

WHERE and conditions

WHERE checks if a **condition** is **TRUE** for a row, and only returns the rows that meet that condition. A condition consists of three parts:

Syntax: `SELECT col(s) FROM db.table WHERE col_to_filter = value;`

Column to filter on

The name of the **column** that the **condition** should apply to.

A condition can only be applied to **one column** at a time. We can **combine** conditions by using **logic operators** like **AND**, **OR**, and **NOT**.

Operator

This is **how** we want to **compare** the values in the column to a specific value.

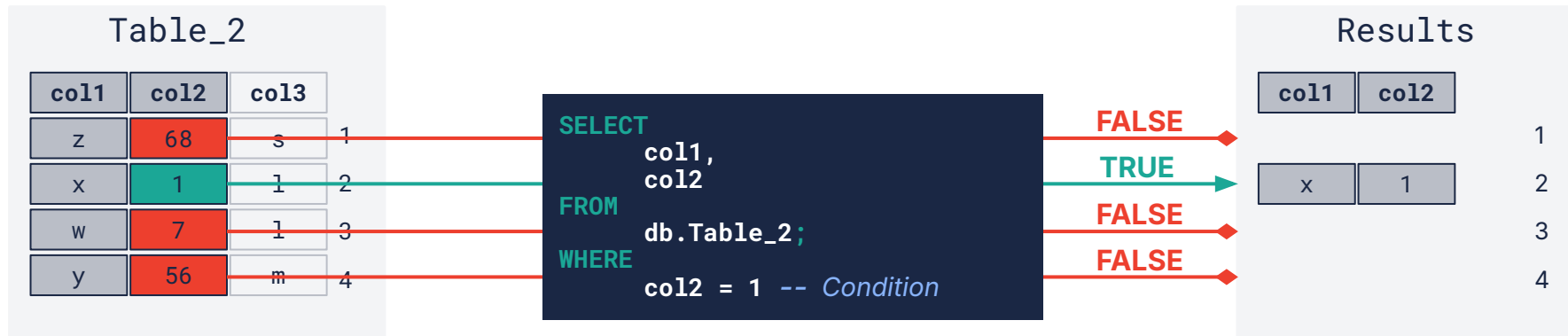SQL offers many other comparison operators (**<**, **>**, **<=**, **>=**, **!=**).

Value

The specific **value** we want to **compare** the column values to.

<value> can be a number, text, another column, or even another query. Each of these options has a specific syntax.

Using WHERE

WHERE only includes rows where the given condition is **TRUE**.

Table_2

col1	col2	col3
z	68	s	1
x	1	l	2
w	7	l	3
y	56	m	4

```
SELECT
        col1,
        col2
FROM
        db.Table_2;
WHERE

        col2 = 1 -- Condition
```

FALSE
TRUE
FALSE
FALSE

Results

col1	col2
		1
x	1	2
		3
		4

Only col1 and col2 are selected in our results set.
The only row where the condition is **TRUE** is row 2, so it is **included**.