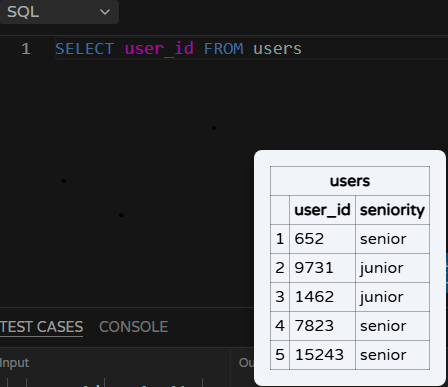
**Introduction**

**SQL** (Structured Query Language) is a standard language for managing and manipulating relational databases. It allows users to store, retrieve, and analyze data efficiently, making it essential for businesses and organizations worldwide. Example below:



**What is a Database?**

Databases are like large buckets that store data in an organized manner. A few examples of when we would like to create a database:

* A database for a university to save data about students, courses, and lecturers.
* A database for a car agency to track sales, car storage, and employees.
* And many more

Inside a database there are tables, and each table has a name, column names, and rows. For example, this is a **workers** table below:

Workers

|  | **firstName** | **lastName** | **age** | **exp\_years** | **gender** |
| --- | --- | --- | --- | --- | --- |
| 1 | Ghully | Thuas | 29 | 2.3 | Female |
| 2 | Bostal | Shkolky | 32 | 0.2 | Male |
| 3 | Qaostu | Malop | 21 | 4 | Female |

The workers table has 5 columns and 3 rows. We don't need any special tool to know that we have 3 workers, and it's easy to calculate the average age of all of them (29 + 32 + 21) / 3. But what happens when we have a thousand or even a million rows?

That's where databases and the SQL language come in. Databases store all of the tables, and SQL extracts the data.

**Challenge**

To extract the whole table from the database, we need to specify which columns to **SELECT** and **FROM** which table to extract.

To do this we'll write:

SELECT column1, column2, ... FROM table\_name

Look at the input on the workers table. For this challenge extract the whole table from the database.

Solution: SELECT firstName, lastName, age, exp\_years, gender FROM Workers

**Database concepts**

In databases, rows are called **records**, and columns are called **fields**.

Tables have a fixed number of fields (columns) but can contain many records (rows). Each field has a unique name, usually in lowercase and singular form. Tables typically include an id field, which serves as a unique identifier for each record, helping to distinguish between similar entries.

In SQL, we can use the asterisk \* symbol as a shortcut to select **all** columns from a table. Instead of listing each column name, simply write:

SELECT \* FROM table\_name

This query fetches every column in the specified table.

**Challenge**

Write an SQL query to retrieve all data from the objects table.

|  |  |  |  |
| --- | --- | --- | --- |
| **objects** | | | |
|  | **id** | **pieces** | **shape** |
| 1 | 251 | 3 | rectangle |
| 2 | 35 | 1 | circle |
| 3 | 39 | 23 | octagon |
| 4 | 21 | 5 | line |
| 5 | 1 | 5 | line |

Solution: SELECT \* FROM objects

**Unique values**

Let's assume we have the following table:

**sales**

|  | **country** | **city** | **amount** |
| --- | --- | --- | --- |
| 1 | Poland | Warsaw | 13 |
| 2 | Germany | Berlin | 24 |
| 3 | Poland | Katowice | 56 |

And we would like to know all of the countries where the product was sold.

If we use the normal query we know: SELECT country from sales it will return Poland, Germany, Poland. This is not what we are looking for because Poland is repeated twice.

To solve it we can use the DISTINCT keyword:

SELECT DISTINCT country FROM sales

**Challenge**

Fetch all of the unique **coins** that were used on the sales table below.

|  |  |  |
| --- | --- | --- |
| **sales** | | |
|  | **coin** | **amount** |
| 1 | AGK | 1.6 |
| 2 | GBL | 7.2 |
| 3 | KLQ | 3.3 |
| 4 | AGK | 1.9 |
| 5 | BPO | 6.3 |
| 6 | THL | 7.9 |

Solution: SELECT DISTINCT coin FROM sales

**Conditional statements part 1**

Sometimes we would like to fetch records that meet a certain condition.

For example

* fetch all of the records that have the family name "Aothly"
* fetch all of the records that the amount is bigger than 5
* fetch all of the records with the country "Mexico"

To add conditions we can use the **WHERE** keyword

For example here is a **sales** table:

|  |  |
| --- | --- |
| **coin** | **amount** |
| AGK | 13 |
| GOL | 21 |
| KLA | 15 |
| AGK | 18 |

To fetch all of the records with the coin "AGK" we will write:

SELECT \* FROM sales  
WHERE coin = "AGK"

To fetch all of the records with amount **smaller or equal** to 20 we will write:

SELECT \* FROM sales  
WHERE amount <= 20

**Challenge**

Fetch all of the event\_ids with less than 14 people.

|  |  |  |
| --- | --- | --- |
| **events** | | |
|  | **event\_id** | **people** |
| 1 | 1 | 9 |
| 2 | 6 | 23 |
| 3 | 9 | 5 |
| 4 | 13 | 7 |
| 5 | 2 | 28 |
| 6 | 4 | 11 |
| 7 | 99 | 22 |
| 8 | 83 | 7 |

Solution: SELECT event\_id FROM events

WHERE people < 14

**Conditional statements part 2**

Creating a query with only one condition is not sufficient. Sometimes we would like to check something more complicated. For that SQL (and many other programming languages) have the AND, OR, and NOT keywords to increase our ability to fetch the right result we need.

The AND and OR keywords are used like this:

SELECT col1, col2   
FROM table1  
WHERE condition1 AND condition2 OR condition3 ...

We can stack as many conditions as we want together.

| **people** | | |
| --- | --- | --- |
| **name** | **age** | **gender** |
| Joas | 13 | male |
| Holwa | 17 | male |
| Nohlas | 24 | female |
| Polar | 23 | male |
| Loopa | 18 | female |

The AND keyword means that **both** conditions must be true; if either of them is not, then the condition will not be met. For example, if we will write

SELECT \*   
FROM people  
WHERE gender = "female" AND age < 20

It means that we are looking for all records that the gender is "female" and the age is less than 20.

| **name** | **age** | **gender** |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Loopa | 18 | female |  |  |  |

This will be the result:

**Challenge:** Fetch all of the people who are between the ages of 20 and 28 (including 20 and 28).

|  |  |  |  |
| --- | --- | --- | --- |
| **people** | | | |
|  | **name** | **age** | **status** |
| 1 | Charles | 28 | employed |
| 2 | Fatima | 38 | unemployed |
| 3 | Eric | 11 | unemployed |
| 4 | Diya | 44 | employed |
| 5 | Hanna | 22 | employed |
| 6 | Ali | 20 | unemployed |

Solution: SELECT \* FROM people

WHERE age >= 20 AND age <= 28

**Conditional statements part 3**

The OR keyword means that we want one of the conditions will be true.

For example, if we take the same example from above and change the AND keyword to OR

SELECT \*   
FROM people  
WHERE gender = "female" OR age < 20

| **people** | | |
| --- | --- | --- |
| **name** | **age** | **gender** |
| Joas | 13 | male |
| Holwa | 17 | male |
| Nohlas | 24 | female |
| Loopa | 18 | female |

It means that we are looking for all records that either the gender is female or the age is less than 20. This will be the result:

The NOT keywords mean that we don't want the condition to be met.

For example, if we write:

SELECT \*   
FROM people  
WHERE NOT gender = "male"

| **name** | **age** | **gender** |
| --- | --- | --- |
| Nohlas | 24 | female |
| Loopa | 18 | female |

This will be the result:

It is important to use parenthesis when combining different conditions because:

WHERE age > 20 AND age < 30 OR gender = 'female'

WHERE age > 20 AND (age < 30 OR gender = 'female')

**N: B:** These are not the same thing and conditions are also different.

The first query will return all people aged 21-29 (regardless of gender) AND all females (regardless of age). The second query will return all people over 20 who are either under 30 OR female.

**Challenge**

Fetch all of the people who are either unemployed or between the ages of 20 and 28 (including 20 and 28) but not age 22.

|  |  |  |  |
| --- | --- | --- | --- |
| **people** | | | |
|  | **name** | **age** | **status** |
| 1 | Charles | 28 | employed |
| 2 | Fatima | 38 | unemployed |
| 3 | Eric | 11 | unemployed |
| 4 | Diya | 44 | employed |
| 5 | Hanna | 22 | employed |
| 6 | Ali | 20 | unemployed |
| 7 | Gabriel | 37 | employed |
| 8 | Beatriz | 17 | employed |
| 9 | Troy | 29 | unemployed |
| 10 | Angelica | 32 | employed |

Solution: SELECT \* FROM