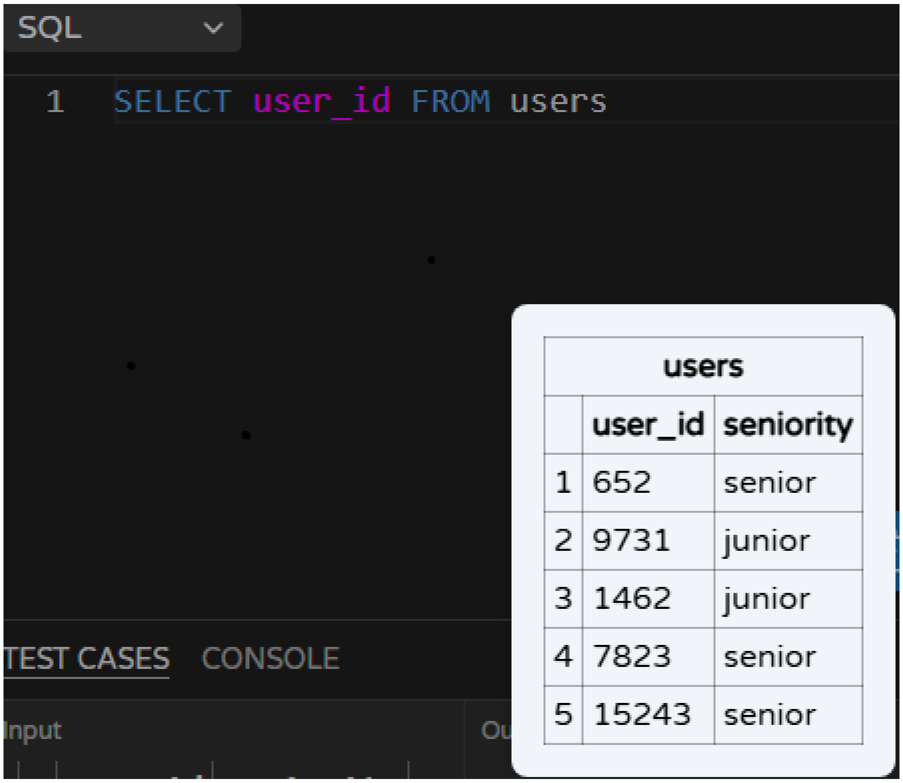
Introduction

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:



```sql
SELECT user_id FROM users
```

users		
	user_id	seniority
1	652	senior
2	9731	junior
3	1462	junior
4	7823	senior
5	15243	senior

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

people		
name	**age**	**gender**
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.

This will be the result:

name	age	gender
Loopa	18	female

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
```

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:

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

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"
```

This will be the result:

name	age	gender
Nohlas	24	female
Loopa	18	female

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')
```

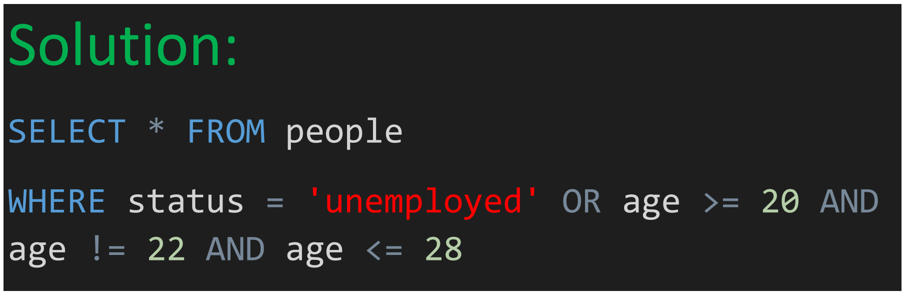
N: B: These are not the same thing and conditions are also different.

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 people

WHERE status = 'unemployed' OR age >= 20 AND
age != 22 AND age <= 28
```

Conditional statements part 4

Conditions are booleans. Boolean is a data type with two possible values: `TRUE` or `FALSE`.

For example

- `10 > 100` - FALSE
- `10 > 5` - TRUE
- `10 > 5 AND 100 < 5` - FALSE

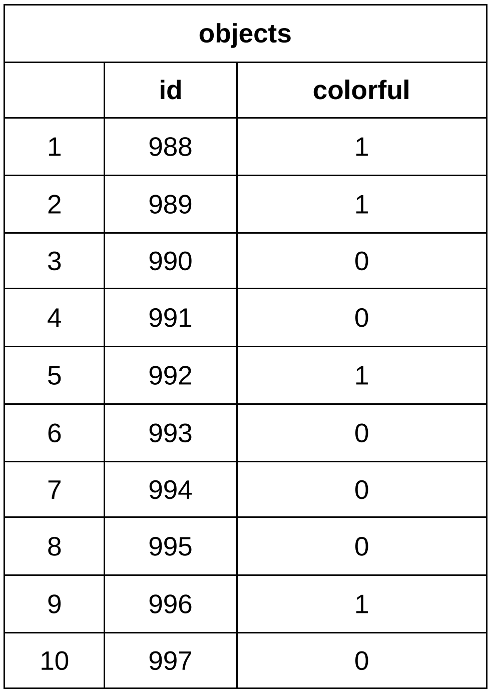
Boolean columns have only two values - either 1 or 0. `TRUE` indicates 1 and `FALSE` indicates 0

We can replace columns such as employed or unemployed to `1` or `0` to make it easier to filter data. To filter data using booleans we will use the `IS TRUE` or `IS NOT TRUE` keywords.

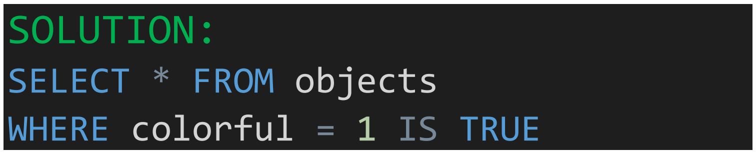
```
SELECT *
FROM table1

WHERE col1 IS NOT FALSE AND col2 IS TRUE
```

Challenge

Fetch all of the colorful objects. Instead of writing `colorful = 1` try to use the `TRUE` keyword.



objects		
	id	colorful
1	988	1
2	989	1
3	990	0
4	991	0
5	992	1
6	993	0
7	994	0
8	995	0
9	996	1
10	997	0



```
SOLUTION:
SELECT * FROM objects
WHERE colorful = 1 IS TRUE
```

Sort results

The results might be confusing. Showing them in an ascending or descending order can greatly improve our data analysis.

competition

runner_id	age	avg_speed
1	47	3.65
2	62	3.07
3	57	6.82
4	56	4.34
5	25	4.93
6	40	3.94
7	23	6.58
8	40	3.43

To sort the result we use the `ORDER BY` keyword and after that, we should specify by which field we are ordering

For example

```
SELECT *
FROM competition
WHERE age > 50
ORDER BY avg_speed
```

by default, it sorts by ascending order.

The result on the right side:

runner_id	age	avg_speed
2	62	3.07
4	56	4.34
3	57	6.82

To specify how to sort that data we can add `DESC` or `ASC` keywords after the name of the column.

```
ORDER BY avg_speed ASC
```

We can even specify multiple columns to sort. For example:

```
SELECT *
FROM competition
WHERE age < 50
ORDER BY age DESC, avg_speed DESC
```

It will first be sorted by `age` in descending order and if there are two equal values then it will sort those records by the `avg_speed` in descending order

The result from the competition table:

runner_id	age	avg_speed
1	47	3.65
8	40	3.43
6	40	3.94
5	25	4.93
7	23	6.58

Challenge

Return all of the **id**’s after ordering them by the weight in descending order.

feathers		
	id	**weight**
1	1	0.01
2	2	0.13
3	3	0.03
4	4	0.09
5	5	0.002
6	6	0.21
7	7	0.35
8	8	0.0045
9	9	0.062

```
SOLUTION:
SELECT id FROM feathers
ORDER BY weight DESC
```

Limit number of records

Let's assume we fetched a lot of data. Sometimes we only need the top 5 or the top 10 records.

To limit the number of records we can use the `LIMIT` keyword. For example:

```
SELECT *
FROM table1
LIMIT 10
```

This will return the top 10 records.

Challenge

Fetch the 5 coldest places from the temperature table

temperature		
	place_id	**avg_temp**
1	1	-21
2	2	-13
3	3	-9
4	4	23
5	5	-1
6	6	0
7	7	6
8	8	4
9	9	15
10	10	-12

```
SOLUTION:
SELECT * FROM temperature
ORDER BY avg_temp ASC
LIMIT 5
```

Null values

In the real world, we might have fields with no values. A field with no value is called null. We can manipulate our query using `IS NULL` or `IS NOT NULL` to fetch relevant data. For example:

```
SELECT *
FROM table1
WHERE col1 IS NULL
```

Will return all of the records where col1 has no value.

Challenge

Fetch all of the unique names without missing values.

people	
	name
1	
2	Roy
3	Dani
4	Esther
5	Bestie
6	
7	Mash
8	
9	
10	Olivier

```
SOLUTION:
SELECT * FROM people
WHERE name IS NOT NULL
```

Recap challenge #1

Challenge

As an owner of a vehicle factory, you have agreed to provide a salary raise for the four employees with the lowest salaries who are also married, as they are struggling to finance their families. Return only the IDs of the relevant employees. Sort the results by salary in ascending order.

employees			
	id	**salary**	**status**
1	1	2016	married
2	2	5903	single
3	3	7608	married
4	4	6448	single
5	5	9551	married
7	7	5753	single
8	8	7313	single
9	9	4219	single
10	10	3140	married
11	11	2702	married
12	12	3035	single
13	13	7590	single
14	14	3404	married
15	15	4551	married

```
SOLUTION:
SELECT id FROM employees
WHERE status = 'married'
ORDER BY salary ASC
limit 4
```

Challenge

You have a cyber-security firm that experienced an arbitrary attack, resulting in all of your systems shutting down. To solve this issue, you need to identify all of the events that appear suspicious. A suspicious event meets **one or more of the following criteria**:

1. Its size is significantly different from the average normal event size of 50MB (you'll need to analyze the data in the table to determine the thresholds for 'too small' and 'too big')
2. It was created before the year 2000
3. It has a missing name

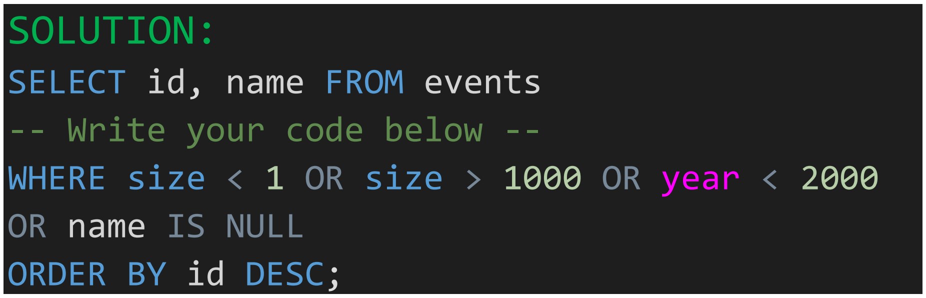
Your task:

1. Examine the provided table of events to determine what should be considered 'too small' or 'too big' based on the distribution of event sizes.
2. Identify all suspicious events based on the criteria mentioned above.
3. Return the event IDs and their names in descending order by their ID.

Note: The exact thresholds for 'too small' and 'too big' should be inferred from the data. Look for patterns or outliers in the event sizes to make this determination."

events

	id	name	size	year
1	153	foat	43	2009
2	154	antiMAL	70	1999
3	155	devdev	1009	2011
4	156		53	2005
5	157	hacker	0.02	2010
6	158	log15234	72	1051
7	159	plural	9999	2055
8	160	system	0.5	2001
9	161	system182	35	2009
10	162	system124	85	2013
11	163	virus	10021	0
12	164	svg	55	2023
13	165	system982	45	2023
14	166	photio	53	2016
15	167	favicon	49	2016
16	168	system	0.002	2049
17	169		50	1209
18	170	server host	49	2015
19	171	boot	9102	2000
20	172	angryBOT	0.001	9999



```
SOLUTION:
SELECT id, name FROM events
-- Write your code below --
WHERE size < 1 OR size > 1000 OR year < 2000
OR name IS NULL
ORDER BY id DESC;
```

The IN keyword

The following query is very long:

```
SELECT *
FROM table1
WHERE col1 = 'a' OR col1 = 'b' OR col1 = 'c' OR col1 = 'd' OR ...
```

We can simplify it by using the `IN` keyword like this:

```
SELECT *
FROM table1
WHERE col1 IN ('a', 'b', 'c', 'd', 'e', 'f')
```

Challenge

Return all the records from the following countries:

Oman, Nicaragua, Bhutan, Senegal, Belarus.

```
SOLUTION:
SELECT * FROM countries
WHERE country IN ('Oman',
'Nicaragua', 'Bhutan', 'Senegal',
'Belarus')
```

countries			
	location_x	**location_y**	**country**
1	53.39	27.53	Belarus
2	57.18	24.96	Latvia
3	-25.64	134.25	Australia
4	20.33	55.85	Oman
5	3.59	45.45	Somalia
6	12.85	-85.32	Nicaragua
7	27.26	90.5	Bhutan
8	36.65	139.61	Japan
9	14.52	-14.47	Senegal
10	39.39	-3.21	Spain

Challenge

Fetch the 5 coldest places from the temperature table.

```
SOLUTION:
SELECT * FROM  temperature
ORDER BY avg_temp ASC
LIMIT 5
```

temperature		
	place_id	**avg_temp**
1	1	-21
2	2	-13
3	3	-9
4	4	23
5	5	-1
6	6	0
7	7	6
8	8	4
9	9	15
10	10	-12

The BETWEEN keyword

As of now, we learned to use bigger `>` and smaller `<` to demand a range for a field. But there is another way.

Instead of writing `col1 >= 5 AND col1 <= 10` we can write: `col1 BETWEEN 5 AND 10`

Challenge

Fetch all of the records between 5 and 10.

```
SOLUTION:
SELECT * FROM data
WHERE value BETWEEN 5 AND 10
/* Instead of -- value >= 5 AND value <= 10 */
```

data	
	value
1	13
2	3
3	5
4	9
5	2
6	7
7	21
8	10

The LIKE keyword

The `LIKE` keyword is used to check the similarities of strings. For example, if we want to fetch all of the records that the name starts with the letter `a` then we will use the `LIKE` keyword.

Two main wildcards are used:

- `%` - means any number of characters
- `_` - means exactly one character

For example:

- `%a` means any string that ends with `a`
- `a%` means any string that starts with `a`
- `%a%` means any string that contains `a`
- `_a%` means that the letter `a` is the second character in the string
- `%a__` means that the string contains `a` in the 3rd from last place

To use it we will write:

```
SELECT col1, col2, ...
FROM table1
WHERE col1 LIKE '%a__'
/* Or examples, ‘%a’, ‘a%’, ‘%a%’, ‘_a%’ */
```

Challenge

Fetch all of the people that their name starts with `k` and ends with `a` and order the results by the names in descending.

people		
	id	**name**
1	13	Jhon
2	14	Kayle
3	15	Kyla
4	16	Somala
5	17	Katarina
6	18	Koa
7	19	Olerrte
8	20	Kassandra
9	21	Kirra
10	22	Koval

```
SOLUTION:
SELECT * FROM people
WHERE name LIKE 'k%a'
ORDER BY name DESC
```

Challenge

As an owner of a vehicle factory, you have agreed to provide a salary raise for the four employees with the lowest salaries who are also married, as they are struggling to finance their families. Return only the IDs of the relevant employees. Sort the results by salary in ascending order.

employees			
	id	**salary**	**status**
1	1	2016	married
2	2	5903	single
3	3	7608	married
4	4	6448	single
5	5	9551	married
6	6	6505	married
7	7	5753	single
8	8	7313	single
9	9	4219	single
10	10	3140	married
11	11	2702	married
12	12	3035	single
13	13	7590	single
14	14	3404	married
15	15	4551	married

```
SOLUTION:
SELECT id FROM employees
WHERE status = 'married'
ORDER BY salary ASC
LIMIT 4
```

Aliases

Column names are important to present data in a meaningful way. If you show a table with bad column names it will be hard for your audience to understand what are you talking about. To change column names you may use the `AS` keyword.

```
SELECT col1 AS firstColumn, col2 AS secondColumn, ...
FROM table1
```

Challenge

Fetch all of the kitchen items that the cutlery have less than 3 items. Change the cutlery column name to `silverware` .

kitchen_items		
	cutlery	**amount**
1	knife	3
2	spoon	13
3	fork	9
4	toothpick	49
5	straw	32
6	chopsticks	14
7	nutcracker	1
8	spatula	2
9	rolling pin	1
10	honey dipper	1

```
SOLUTION:
SELECT cutlery AS silverware,
amount AS amount FROM kitchen_items
WHERE amount < 3
```

Recap challenge #1

Challenge

Fetch all of the cellphone models that start with the letter `m` and the 3rd letter is `o`, the price range is between 1000 and 1500, and they support 5G.

Return only the cellphone model and replace the name to `id`

```
SOLUTION:
SELECT model AS id FROM cellphones
WHERE model LIKE 'm_o%'
    AND price BETWEEN 1000 AND 1500
    AND wifi_5g = 1;
```

cellphones			
	model	**price**	**wifi_5g**
1	mqopal	2590	1
2	mlop12	1293	1
3	maqw99	1490	0
4	qpola21	1092	1
5	hj52wdf	800	0
6	m1oa32	1392	1
7	12o09p	999	0
8	mtozavg	452	1
9	kflwp67	3098	0
10	nbgfert	1189	1

Challenge

Fetch the top 5 severe criminal **names** in **descending** order (by `severe_score`) that are not listed in the police report.

A severe criminal is someone who matches the following criteria:

- `report` is either empty, or the `report` contains one of the following letters: `g`, `b`, `G`, or `B`.
- `map` is one of the following places: `Caerleon`, `Dewsbury`, `Kirekwall`, `Findochty`. Name the column as `worst_criminals`.

```
SOLUTION:
SELECT name AS worst_criminals

FROM police_report

WHERE (report IS NULL OR report LIKE '%g%'
OR report LIKE '%b%' OR report LIKE '%G%'
OR report LIKE '%B%' ) AND map IN
('Caerleon', 'Dewsbury', 'Kirekwall',
'Findochty')

ORDER BY severe_score DESC

LIMIT 5;
```

police_report				
	name	**report**	**map**	**severe_score**
1	Domingos Holden	VYyPJw	Lockinge	2
2	Isabel Enid		Findochty	7
3	Tadgán Musa	FgDqud	Kara's Vale	10
4	Filip Baxter	D1rJqH	Kirekwall	3
5	Edorta Elias	pQ53RC	Dewsbury	9
6	Kəmal Davy	BsI86j	Kara's Vale	2
7	Isokrates Bituin		Findochty	5
8	Kassy Ramirus	Pkjv7J	Kirekwall	7
9	Doris Blessing	0kJXq1	Dalmerlington	6
10	Nedeljka Ganesh	EVe9ha	Kara's Vale	8
11	Karlene Timotheus	2HUBHz	Dewsbury	7
12	Emerens Raman	PAGdHl	Findochty	1
13	Mijo Ambrosios		Luton	5
14	Karlene Jairus	4Z63Q5	Dalmerlington	8
15	Khordad Peter		Findochty	2
16	Biagio Mai		Findochty	9
17	Liwen Sigiward	B51ETs	Mansfield	3
18	Svante Mona	DC0kGh	Caerleon	5
19	Kshitija Ladislav	cSA0hA	Caerleon	2
20	Ha-Eun Tatiana	YNYhHV	Kara's Vale	4

Parliamentary elections

In SQL we can also make calculations on columns and check the condition of the result

```
...
WHERE col1 + col2 > 0 AND ... OR ...
...
```

To check if the remainder by 2 is 0, use the `%` Sign:

```
...
WHERE sit % 2 = 0
...
```

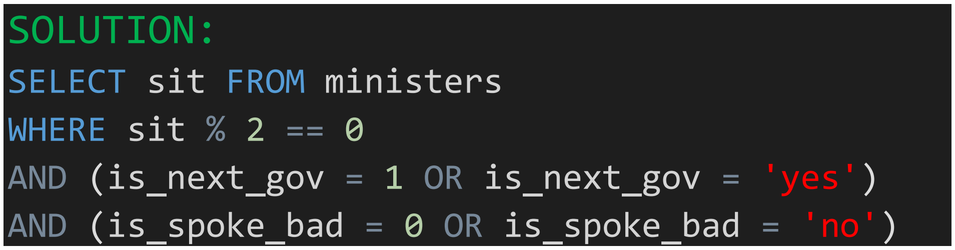
Challenge

Every **second** minister in the government is considered a "safe" minister. Your job is to return all of the "safe" ministers `sits` that want to continue serving in the next government while excluding any minister who spoke a bad word.

The result should contain only the relevant sits.

Before starting to solve the challenge take a look at the table data. It is not clean.

ministers			
	sit	is_next_gov	is_spoke_bad
1	1	0	0
2	2	0	1
3	3	yes	no
4	4	no	no
5	5	1	no
6	6	yes	no
7	7	1	no
8	8	0	yes
9	9	0	1
10	10	no	1
11	11	yes	yes
12	12	1	1
13	13	1	1
14	14	1	1
15	15	yes	0
16	16	0	0
17	17	0	0
18	18	0	no
19	19	1	yes
20	20	1	no
21	21	0	0
22	22	1	0
23	23	yes	1
24	24	yes	1
25	25	no	yes
26	26	1	no
27	27	0	yes
28	28	yes	no
29	29	no	0
30	30	yes	0



```
SOLUTION:
SELECT sit FROM ministers
WHERE sit % 2 == 0
AND (is_next_gov = 1 OR is_next_gov = 'yes')
AND (is_spoke_bad = 0 OR is_spoke_bad = 'no')
```

Bar beverage container

Challenge

Your bar stocks a vast selection of juices, some of which have expired. Identify and sort these juices based on how they should be processed, according to the following criteria:

1. Old Expired Juices: These are juices where the expiration year is more than 6 years before the current year. These should be recycled.
2. Almost Expired Juices: These are juices expiring this year or the next year. These can be sent for renewal.

Extract the IDs of juices that are either old expired or almost expired. Sort the results based on the urgency of their processing needs, with those needing immediate attention (greater difference between current year and expiration year) first.

Rename the ID column to `to_renew` in your output.

```
SOLUTION:
SELECT id AS to_renew FROM beverages
WHERE current_year - expiration_year > 6 OR
expiration_year >= current_year AND
expiration_year <= current_year + 1
ORDER BY
    CASE
        WHEN current_year - expiration_year
> 6 THEN current_year - expiration_year
        ELSE -(expiration_year -
current_year)
    END DESC;
```

beverages

	id	current_year	expiration_year
1	145	2013	2014
2	156	2001	2015
3	167	2009	2004
4	178	2005	2000
5	124	2013	2006
6	189	2002	2014
7	198	2007	2013
8	201	2004	2007
9	206	2000	2000
10	112	2011	2002
11	209	2008	2004
12	125	2015	2012
13	980	2008	2005
14	402	2010	2011
15	391	2008	2009
16	144	2015	2013
17	213	2014	2007
18	100	2001	2000
19	145	2002	2004
20	981	2011	2014
21	210	2002	2007
22	392	2010	2006
23	393	2007	2013
24	113	2010	2002
25	255	2001	2008