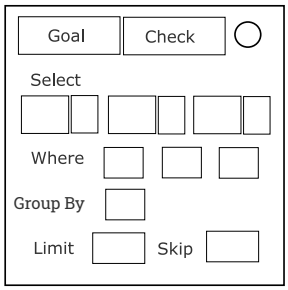
On the Subject of SQL - Cruel



How fun is it to ride through a large table of numbers to find similarities...

This module is a SQL code simulator. To disarm it, produce a SQL query that outputs a given result. The input data is in this document, use it to reproduce the given result. Use the documentation to build a SQL query that will reproduce the expected result from the input data.

The Goal button

Press the **Goal** button to see what results the module expects. The expectation is a table of numbers with anywhere from 1 to 3 rows and 3 columns. Below are a few examples:

2	9	6
4	2	1
7	1	4

2	9	6
4	2	1

2	9	6

The Check button

Upon pressing the **Check** button, the module compiles the SQL query and executes it against the input data. The result is compared to the goal table. If it matches, the module is disarmed; otherwise, a strike is registered.

Disarming the module

To disarm the module, find numbers from the goal table in the input data that could indicate what options were used. The rows are always used from top to bottom, but the columns can be reordered. Example:

Goal matrix

9	0	1
1	1	4
1	0	7

Values from the goal in column 3 seem to match column F!

This could mean we used column F to group values together!

Example input data

A	B	C	D	E	F
9	3	8	6	9	1
9	3	0	4	3	1
1	2	1	6	2	4
1	7	0	4	3	7

The input data

The input data used to disarm the module is shown below. Find the values from the goal table inside this table and craft a SQL query that generates the same output.

A	B	C	D	E	F	G
9	5	0	2	1	0	3
7	8	3	1	0	6	4
7	5	7	4	6	5	3
9	8	0	4	1	0	4
1	2	7	2	6	6	3
1	8	3	1	6	0	5
9	2	7	2	1	5	5
1	5	7	2	6	6	4
7	8	0	4	0	0	4
9	2	0	1	1	5	5

Remember:

SQL results are not necessarily in reading order. The **rows** will be processed from top to bottom, but the **columns** can be arbitrarily reordered in the SELECT clause in the query.

On the Subject of SQL

SQL is a query language that describes how to manipulate sources of data and generate a result. It comprises many different keywords, only a small subset of which is used by this module.

How SQL is processed

SQL is processed as follows:

1. Each row is compared against the **WHERE** clause (described below). If a row complies with the WHERE clause, it is kept for further processing.
2. Then, all rows are grouped together based on the **GROUP BY** clause (described below). You **must** provide 1 group by expression.
3. Each group's rows are then processed against the **SELECT** clause and aggregated as configured (described below). You **must** aggregate all but 1 expression, that one expression **has** to be the one you use in the **GROUP BY** clause.
4. Finally, the **LIMIT** clause (described below) can be used to discard a certain number of rows by skipping the first X rows from the result and then taking the next Y lines after that.

SELECT X, Y, Z

The **SELECT** clause chooses which columns to return (and in what order). The module offers up to 3 simple column references. Each column can be clicked to cycle between columns A to G and a "-" which disables that selection. An example of this follows:

A	B	C	D	E	F	G
9	3	0	2	0	4	6
7	8	6	1	2	9	8
2	0	5	4	7	3	1
4	7	1	5	3	0	9
6	2	7	9	8	6	5
1	9	3	7	6	5	4
8	5	4	3	1	8	2

SQL: SELECT D, A

Notice there is no 3rd expression. We used a "-".

D	A
2	9
1	7
4	2
5	4
9	6
7	1
3	8

WHERE X operation Y

WHERE introduces "filters". The rows are tested against the condition. A row is kept in the output only if it complies with the filter condition. In this simple SQL emulator, only comparisons of columns to fixed values are possible, such as: A < 2, C = 3, F >= 5. The possible operators are simple, familiar mathematical operators such as:

- "=": Equals
- "<>": Not equals
- "<": Less than
- "<=": Less than or equal to
- ">": Greather than
- ">=": Greather than or equal to

A basic example of a single filter follows:

A	B	C	D	E	F	G
9	3	0	2	0	4	6
7	8	6	1	2	9	8
2	0	5	4	7	3	1
4	7	1	5	3	0	9
6	2	7	9	8	6	5
1	9	3	7	6	5	4
8	5	4	3	1	8	2

SQL: WHERE C >= 5

A	B	C	D	E	F	G
7	8	6	1	2	9	8
2	0	5	4	7	3	1
6	2	7	9	8	6	5

> **Note:** Conditions can be applied to any column, not just the **SELECT**ed ones.

GROUP BY X

The **GROUP BY** clause chooses which column to use to create groups of rows. When you group by a certain column, all the values for that column are inspected and then, groups are made from those values. Then, you apply aggregators (See below) to the group of rows. For example:

A	B	C	D	E	F	G
9	5	0	2	1	0	3
7	8	3	1	0	6	4
7	5	7	4	6	5	3
9	8	0	4	1	0	4
1	2	7	2	6	6	3
1	8	3	1	6	0	5
9	2	7	2	1	5	5

SQL: GROUP BY C

A	B	D	E	F	G
C = 0					
9	5	2	1	0	3
9	8	4	1	0	4
C = 3					
7	8	1	0	6	4
1	8	1	6	0	5
C = 7					
7	5	4	6	5	3
1	2	2	6	6	3
9	2	2	1	5	5

SELECT Aggregators

This module allows usage of SELECT aggregators. Aggregators are group functions used to simplify groups of rows into single values. To use aggregators, you **must** use a **GROUP BY** expression. Here are a few examples of how aggregators affect groups of rows:

<table>
<tr><th>Example data</th><th>Condition</th><th>Result</th></tr>
<tr><td>
<table><tr><th>A</th><th>B</th></tr><tr><td colspan="2">C = 0</td></tr><tr><td>9</td><td>5</td></tr><tr><td>9</td><td>8</td></tr></table>
<table><tr><th>A</th><th>B</th></tr><tr><td colspan="2">C = 3</td></tr><tr><td>7</td><td>8</td></tr><tr><td>1</td><td>8</td></tr></table>
<table><tr><th>A</th><th>B</th></tr><tr><td colspan="2">C = 7</td></tr><tr><td>7</td><td>5</td></tr><tr><td>1</td><td>2</td></tr><tr><td>9</td><td>2</td></tr></table>
</td><td>SELECT MIN(A), MAX(B), C</td><td>
<table><tr><td>9</td><td>8</td><td>0</td></tr><tr><td>1</td><td>8</td><td>3</td></tr><tr><td>1</td><td>5</td><td>7</td></tr></table>
</td></tr>
<tr><td>
<table><tr><th>A</th><th>B</th></tr><tr><td colspan="2">C = 0</td></tr><tr><td>9</td><td>5</td></tr><tr><td>9</td><td>8</td></tr></table>
<table><tr><th>A</th><th>B</th></tr><tr><td colspan="2">C = 3</td></tr><tr><td>7</td><td>8</td></tr><tr><td>1</td><td>8</td></tr></table>
<table><tr><th>A</th><th>B</th></tr><tr><td colspan="2">C = 7</td></tr><tr><td>7</td><td>5</td></tr><tr><td>1</td><td>2</td></tr><tr><td>9</td><td>2</td></tr></table>
</td><td>SELECT COUNT(A), AVG(B), C</td><td>
<table><tr><td>2</td><td>6</td><td>0</td></tr><tr><td>2</td><td>8</td><td>3</td></tr><tr><td>3</td><td>3</td><td>7</td></tr></table>
</td></tr>
<tr><td>
<table><tr><th>A</th></tr><tr><td>C = 0</td></tr><tr><td>9</td></tr><tr><td>9</td></tr></table>
<table><tr><th>A</th></tr><tr><td>C = 3</td></tr><tr><td>7</td></tr><tr><td>1</td></tr></table>
<table><tr><th>A</th></tr><tr><td>C = 7</td></tr><tr><td>7</td></tr><tr><td>1</td></tr><tr><td>9</td></tr></table>
</td><td>SELECT SUM(A), C</td><td>
<table><tr><td>18</td><td>0</td></tr><tr><td>8</td><td>3</td></tr><tr><td>17</td><td>7</td></tr></table>
</td></tr>
</table>

Note: The AVG aggregator will be rounded down automatically because the resulting decimal values are casted to integers, therefore decimals are lost in the process. You will never see decimal numbers in this module.

LIMIT X, Y

LIMIT is followed by two values:

- First value: indicates the maximum number of rows to keep after applying conditions. If you select "All", no rows are discarded.
- Second value: indicates the number of rows to skip after applying conditions. If you select "None", no rows are discarded.

Note that the second value (the skip) is applied first by skipping that number of rows from the top of the results. Afterwards, the first value is applied to return a maximum of that many rows.

Examples:

Example data	Condition	Result
9 3 0 7 8 6 2 0 5 4 7 1	LIMIT 2, 0 (Limit of 2; skip none)	9 3 0 7 8 6
9 3 0 7 8 6 2 0 5 4 7 1	LIMIT 999, 1 (No limit; skip 1)	7 8 6 2 0 5 4 7 1
9 3 0 7 8 6 2 0 5 4 7 1	LIMIT 1, 2 (Limit of 1; skip 2)	2 0 5