

## SET OPERATIONS:

I used UNION

UNION operator

In SQL the UNION clause combines the results of two SQL queries into a single table of all matching rows. The two queries must result in the same number of columns and compatible data types in order to unite. Any duplicate records are automatically removed unless UNION ALL is used.

**SELECT AURORA\_BOREALIS from bobross.elements**

**UNION**

**SELECT BEACH from bobross.elements;**

The screenshot shows the MySQL Workbench interface. The left sidebar displays the 'Schemas' tree with 'bobross' expanded, showing tables like 'elements'. The main editor window contains the following SQL query:

```
1 SELECT * FROM bobross.elements;
2
3
4 SELECT AURORA_BOREALIS from bobross.elements
5 UNION
6 SELECT BEACH from bobross.elements
```

Below the query editor, the 'Result Grid' shows the output of the query. The table has columns: EPISODE, TITLE, APPLE\_FRAME, AURORA\_BOREALIS, BARN, BEACH, BOAT, BRIDGE, BUILDING, BUSHES. The data rows are:

EPISODE	TITLE	APPLE_FRAME	AURORA_BOREALIS	BARN	BEACH	BOAT	BRIDGE	BUILDING	BUSHES
S01E01	"A WALK IN THE WOODS"	0	0	0	0	0	0	0	1
S01E02	"MT. MCKINLEY"	0	0	0	0	0	0	0	0
S01E03	"EBONY SUNSET"	0	0	0	0	0	0	0	0
S01E04	"WINTER MIST"	0	0	0	0	0	0	0	1

The bottom status bar shows the query completed successfully at 19:37:10, with a message 'ANALYZE TABLE 'bobross'.elements-by-episode' OK' and 'Apply changes to elements' changes applied.

## SUBQUERIES:

Subqueries have may be called other names as well such as an Inner query. A Nested query is a query within another SQL query and embedded within the WHERE clause. Which is what I believe I used. A subquery is used to return data that will be used in the main query as a condition to further restrict the data to be retrieved. Subqueries can be used with the SELECT, INSERT, UPDATE, and DELETE statements along with the operators like =, <, >, >=, <=, IN, BETWEEN,

**SELECT EPISODE,**

**TITLE**

**FROM bobross.elements**

**WHERE WAVES = (**

**SELECT MAX(WAVES)**

**FROM bobross.elements**

**);**

The screenshot shows the MySQL Workbench interface. The SQL editor contains the following query:

```
5 UNION
6 SELECT BEACH from bobross.elements;
7
8
9 SELECT EPISODE,
10 TITLE
11 FROM bobross.elements
12 WHERE WAVES = (
13 SELECT MAX(WAVES)
14 FROM bobross.elements
15 );
16
```

The Results window displays the output of the query, showing a table with two columns: EPISODE and TITLE. The results are as follows:

EPISODE	TITLE
S02E03	"EBONY SEA"
S02E09	"BLACK & WHITE SEASCAPE"
S03E02	"BLUE MOON"
S03E08	"NIGHT LIGHT"
S04E05	"EVENING SEASCAPE"

The Output window shows the execution of the query, indicating that 34 rows were returned.

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

## ORDER OF OPERATIONS:

The SQL order of execution defines the order in which the clauses of a query are evaluated. Six orders of operations are SELECT, FROM, WHERE, GROUP, HAVING and ORDER BY. I have demonstrated my example below.

**SELECT TITLE**

**FROM bobross.elements**

**WHERE EPISODE <> 'FRAMED'**

**GROUP BY EPISODE**

**HAVING AVG(FRAMED)**

The screenshot shows the MySQL Workbench interface. The 'Navigator' pane on the left displays the 'bobross' database schema, including tables like 'cactus\_and\_clouds', 'elements', and 'columns'. The 'SQL Editor' pane in the center contains the following query:

```
18
19 SELECT TITLE
20 FROM bobross.elements
21 WHERE EPISODE <> 'FRAMED'
22 GROUP BY EPISODE
23 HAVING AVG(FRAMED);
24
```

The 'Result Grid' pane at the bottom shows the results of the query, which are the titles of the episodes that are not 'FRAMED'. The results are listed in a table with one column, 'TITLE', and 15 rows of data:

TITLE
"WINTER SAWSCAPE"
"LAKEIDE CABIN"
"MAJESTIC PINE"
"WINTER OVAL"
"MOUNTAIN OVAL"
"TRIPLE VIEW"
"OVAL BARN"
"SUNSET OVAL"
"MOUNTAIN IN AN OVAL"
"MEADOW BROOK"
"OVAL ESSENCE"
"MOUNTAIN MOONLIGHT"
"MOUNTAIN RIVER"
"RIVERSIDE ESCAPE OVAL"
"MISTY FOREST OVAL"
"HIDDEN WINTER MOON"

The 'SQL Additions' pane on the right displays a message: 'Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.'

The status bar at the bottom indicates 'Query Completed' and shows the system clock as 9:59 PM on 10/24/2020.

## CREATING and ALTERING TABLES:

You can define the columns and assign field attributes when you create the table. After a table has been set up, it can be modified using the ALTER table command. To create a table, use the following command with the new table's name, column names, and data type for each column. To drop a table simply use DROP TABLE.

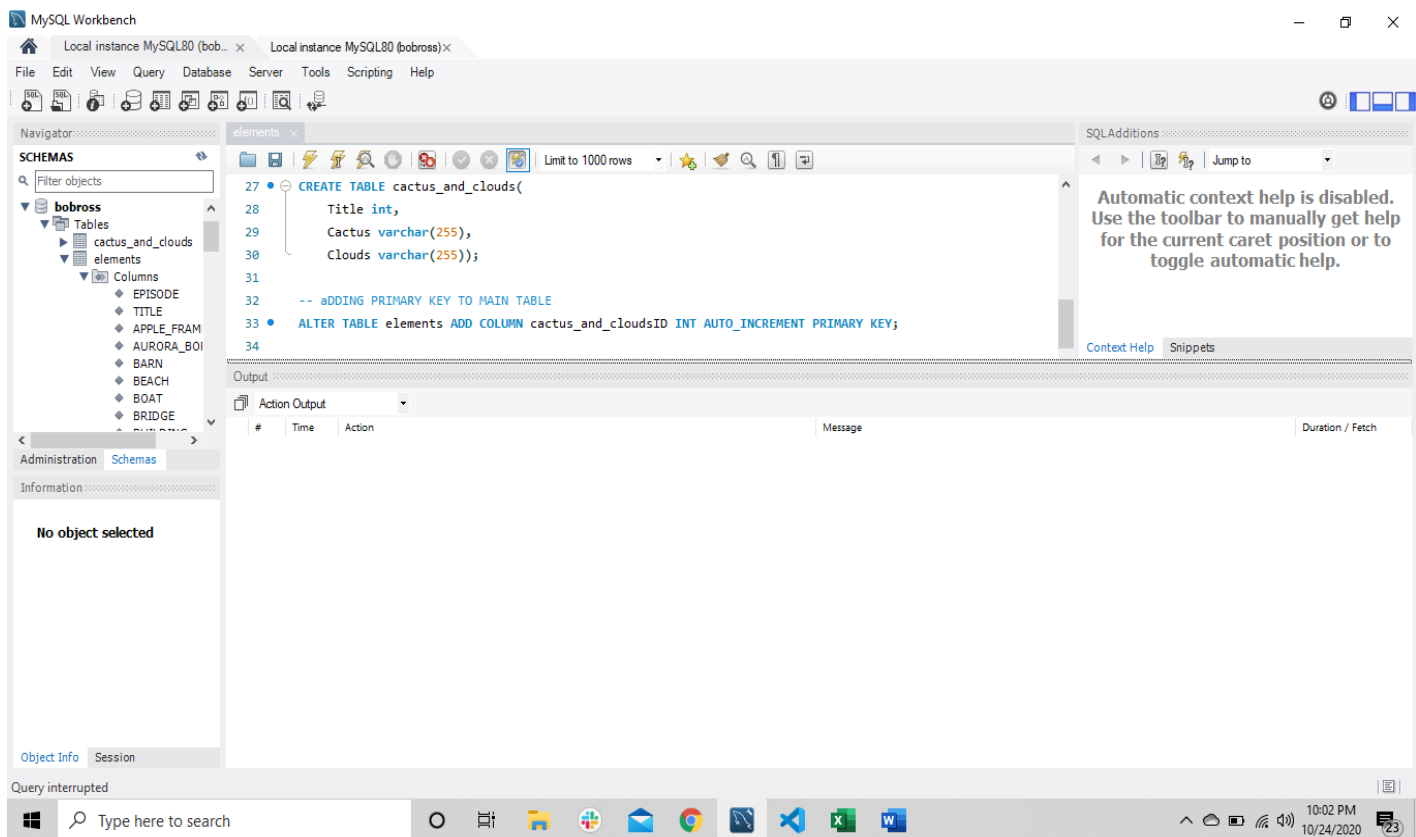
**CREATE TABLE cactus\_and\_clouds(**

**Title int,**

**Cactus varchar(255),**

**Clouds varchar(255));**

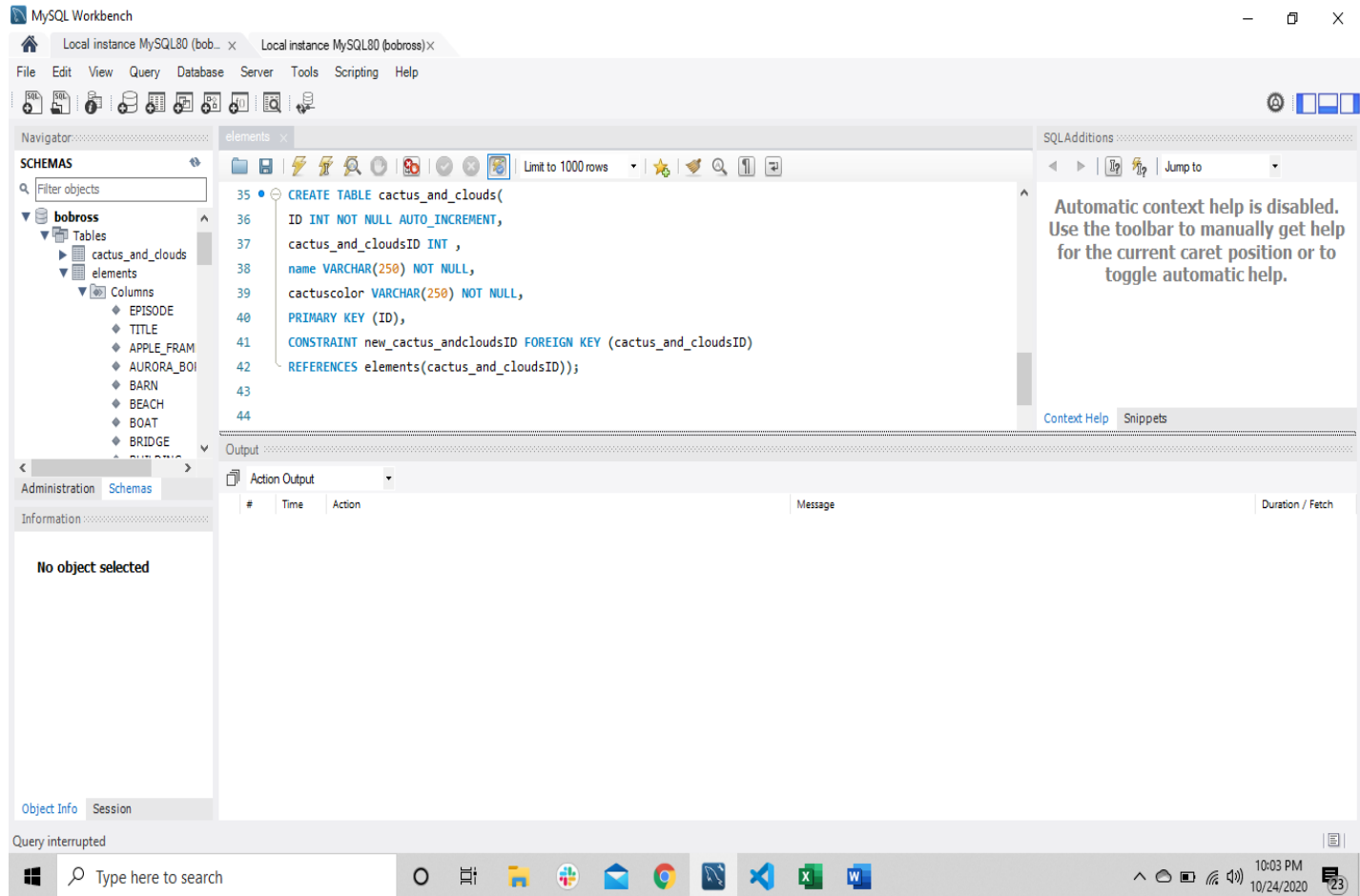
**ALTER TABLE elements ADD COLUMN cactus\_and\_cloudsID INT AUTO\_INCREMENT PRIMARY KEY;**



## **ASSOCIATIONS:**

An association are tables that are used for many-to-many relationships between two objects. They consist of at least two foreign keys, each of which references one of the two objects. My example is below.

```
CREATE TABLE cactus_and_clouds(  
ID INT NOT NULL AUTO_INCREMENT,  
cactus_and_cloudsID INT ,  
name VARCHAR(250) NOT NULL,  
cactuscolor VARCHAR(250) NOT NULL,  
PRIMARY KEY (ID),  
CONSTRAINT new_cactus_andcloudsID FOREIGN KEY (cactus_and_cloudsID)  
REFERENCES elements(cactus_and_cloudsID));  
  
INSERT INTO cactus_and_clouds(cactus_and_cloudsID, cactuscolor, name)  
values (5,'blue','prickly'),(10, 'green', 'soft'),(15, 'white','fluffy');
```



## JOIN:

A JOIN creates a set that can be saved as a table or used as it is. A JOIN is a means for combining columns from one or more tables by using values common to each.

```
SELECT distinct a.title, a.episode, b.name, b.cactuscolor, b.cactus_and_cloudsID
```

```
FROM elements a
```

```
JOIN cactus_and_clouds b
```

```
ON b.cactus_and_cloudsID = b.cactus_and_cloudsID
```

```
ORDER by a.title;
```

The screenshot shows the MySQL Workbench interface. The SQL editor contains the following query:

```
49 • SELECT distinct a.title, a.episode, b.name, b.cactuscolor, b.cactus_and_cloudsID
50 FROM elements a
51 JOIN cactus_and_clouds b
52 ON b.cactus_and_cloudsID = b.cactus_and_cloudsID
53 ORDER by a.title;
54
```

The query is executed, and the results are displayed in the Result Grid. The results are as follows:

title	episode	name	cactuscolor	cactus_and_cloudsID
"A MILD WINTER'S DAY"	S17E06	prickly	blue	5
"A MILD WINTER'S DAY"	S17E06	soft	green	10
"A MILD WINTER'S DAY"	S17E06	fluffy	white	15
"A PRETTY AUTUMN DAY"	S24E05	fluffy	white	15
"A PRETTY AUTUMN DAY"	S24E05	soft	green	10
"A PRETTY AUTUMN DAY"	S24E05	prickly	blue	5
"A WALK IN THE WOODS"	S01E01	prickly	blue	5
"A WALK IN THE WOODS"	S01E01	soft	green	10
"A WALK IN THE WOODS"	S01E01	fluffy	white	15
"ABSOLUTELY AUTUMN"	S18E02	soft	green	10
"ABSOLUTELY AUTUMN"	S18E02	fluffy	white	15
"ABSOLUTELY AUTUMN"	S18E02	prickly	blue	5
"AFTER THE RAIN"	S19E10	fluffy	white	15
"AFTER THE RAIN"	S19E10	prickly	blue	5
"AFTER THE RAIN"	S19E10	soft	green	10

The interface also shows a sidebar with a tree view of the database schema, including tables like 'cactus\_and\_clouds' and 'elements'. The bottom status bar indicates 'Query Completed'.