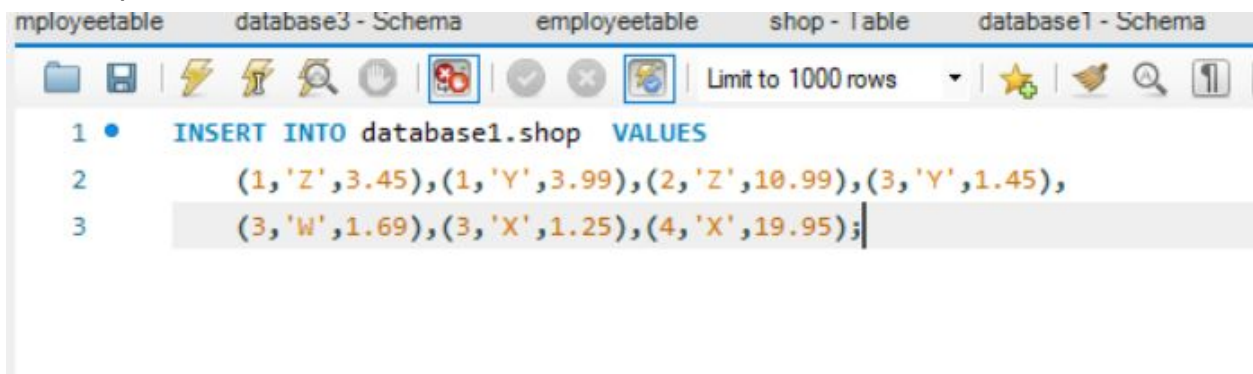


MYSQL2 -QUERIES

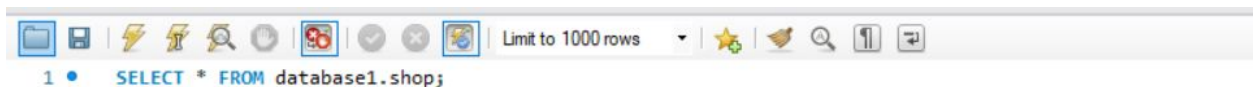
STEP 1: CREATE ANOTHER TABLE called shop in database1 AND INSERT VALUES into Shop table



The screenshot shows a SQL editor window with a toolbar at the top. The toolbar includes icons for file operations, execution, and a 'Limit to 1000 rows' dropdown. The SQL text in the editor is as follows:

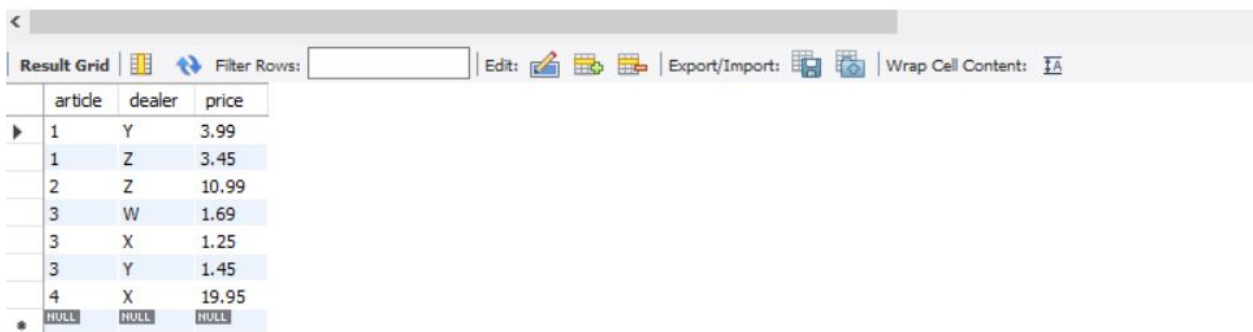
```
1 • INSERT INTO database1.shop VALUES
2     (1, 'Z', 3.45), (1, 'Y', 3.99), (2, 'Z', 10.99), (3, 'Y', 1.45),
3     (3, 'W', 1.69), (3, 'X', 1.25), (4, 'X', 19.95);
```

STEP 2: list the contents of Shop table



The screenshot shows a SQL editor window with a toolbar at the top. The toolbar includes icons for file operations, execution, and a 'Limit to 1000 rows' dropdown. The SQL text in the editor is as follows:

```
1 • SELECT * FROM database1.shop;
```



The screenshot shows a database client window with a toolbar at the top. The toolbar includes icons for file operations, execution, and a 'Limit to 1000 rows' dropdown. The result grid displays the following data:

	article	dealer	price
▶	1	Y	3.99
	1	Z	3.45
	2	Z	10.99
	3	W	1.69
	3	X	1.25
	3	Y	1.45
	4	X	19.95
*	NULL	NULL	NULL

STEP 3: SELECT MAX(ARTICLE) from the list

The screenshot shows a database management tool interface. The top toolbar includes icons for file operations, execution, and a 'Limit to 1000 rows' dropdown. The SQL editor contains the query: `1 • SELECT MAX(article) FROM database1.shop;`. Below the editor, the 'Result Grid' tab is active, displaying a single row with the value '4' under the column header 'MAX(article)'.

MAX(article)
4

STEP 4: Find the number, dealer and the price of the most expensive article in shop table


The screenshot shows the same database management tool interface. The SQL editor contains the query: `1 SELECT article, dealer, price
2 FROM shop
3 WHERE price=(SELECT MAX(price) FROM shop);`. The 'Result Grid' tab is active, displaying a single row with the values '4', 'X', and '19.95' under the column headers 'article', 'dealer', and 'price' respectively. A row of 'NULL' values is also visible below the main result.

article	dealer	price
4	X	19.95
NULL	NULL	NULL

STEP 5: Find the number, dealer and the price of the most expensive article using LEFT JOIN and LIMIT CLAUSE in shop table.



```
3 LEFT JOIN shop s2 ON s1.price < s2.price
4 WHERE s2.article IS NULL;
5
6 • SELECT article, dealer, price
7 FROM shop
8 ORDER BY price DESC
9 LIMIT 1;
```



Result Grid

	article	dealer	price
4		X	19.95
	NULL	NULL	NULL

STEP 6: Find the maximum of column per group in shop table.

The screenshot shows the MySQL Workbench interface. On the left, the 'SCHEMAS' pane shows a tree view with 'database1' expanded, containing 'Tables' and 'shop'. The main query editor displays the following SQL query:

```
1  
2 • SELECT article, MAX(price) AS price  
3 FROM shop  
4 GROUP BY article  
5 ORDER BY article;
```

Below the query editor, the 'Result Grid' shows the results of the query:

	article	price
▶ 1		3.99
2		10.99
3		1.69
4		19.95

STEP 7: For each article find the dealer or dealers with most expensive price using correlated query pn shop table.

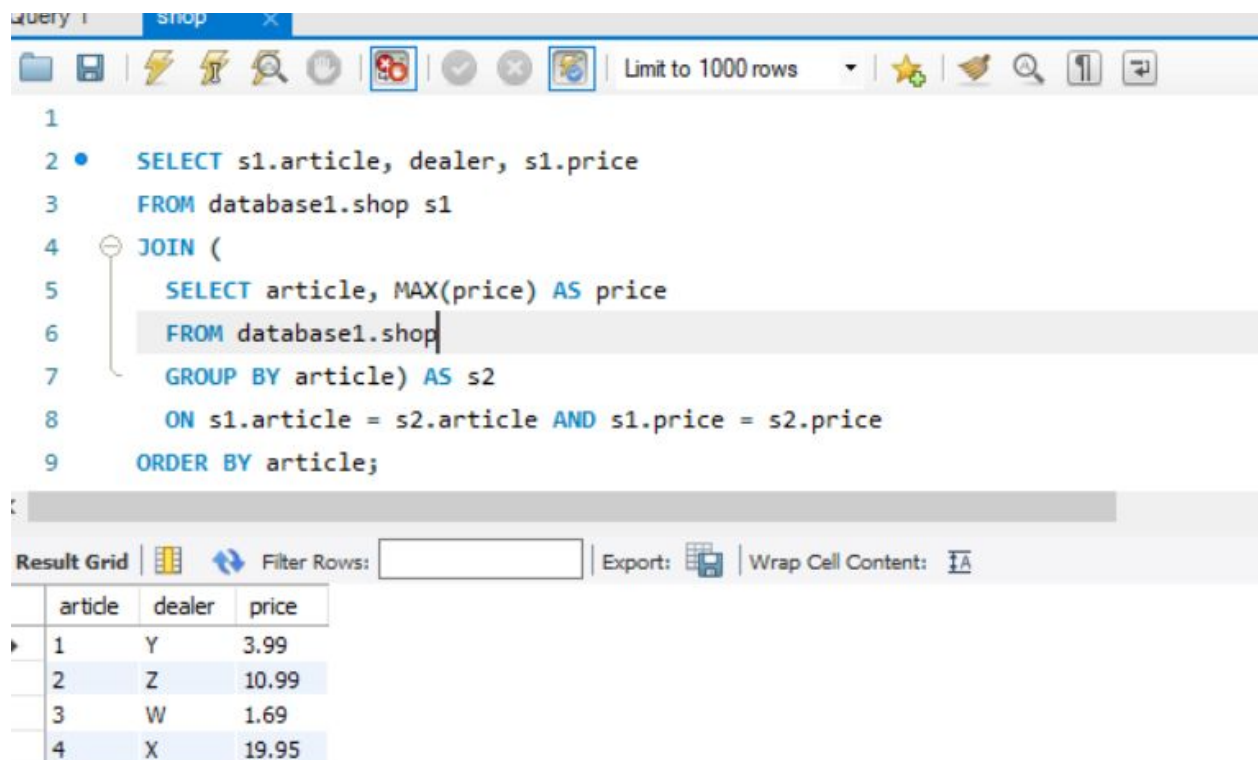
The screenshot shows the MySQL Workbench interface. The main query editor displays the following SQL query:

```
1 • SELECT article, dealer, price  
2 FROM shop s1  
3 WHERE price=(SELECT MAX(s2.price)  
4 FROM shop s2  
5 WHERE s1.article = s2.article)  
6 ORDER BY article;
```

Below the query editor, the 'Result Grid' shows the results of the query:

	article	dealer	price
▶ 1		Y	3.99
2		Z	10.99
3		W	1.69
4		X	19.95
*	NULL	NULL	NULL

STEP 8: For each article find the dealer or dealers with most expensive price using FROM clause using an un-correlated query. When there is no s2 price with a greater value LEFT JOIN works on the basis that s1 price is at its maximum.



The screenshot shows a database query editor window. The query is as follows:

```
1
2 • SELECT s1.article, dealer, s1.price
3 FROM database1.shop s1
4 JOIN (
5     SELECT article, MAX(price) AS price
6     FROM database1.shop
7     GROUP BY article) AS s2
8 ON s1.article = s2.article AND s1.price = s2.price
9 ORDER BY article;
```

Below the query editor, the 'Result Grid' is displayed, showing the results of the query. The grid has columns for 'article', 'dealer', and 'price'. The results are as follows:

	article	dealer	price
1	1	Y	3.99
2	2	Z	10.99
3	3	W	1.69
4	4	X	19.95

STEP 9: For each article find the dealer or dealers with most expensive price using LEFTJOIN clause using an un-correlated query in shop table.

```
1 • SELECT s1.article, s1.dealer, s1.price
2 FROM database1.shop s1
3 LEFT JOIN database1.shop s2 ON s1.article = s2.article AND s1.price < s2.price
4 WHERE s2.article IS NULL
5 ORDER BY s1.article;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: |

	article	dealer	price
1		Y	3.99
2		Z	10.99
3		W	1.69
4		X	19.95

STEP 10: Query database1.shop table with user-defined queries- find min and max price from shop table

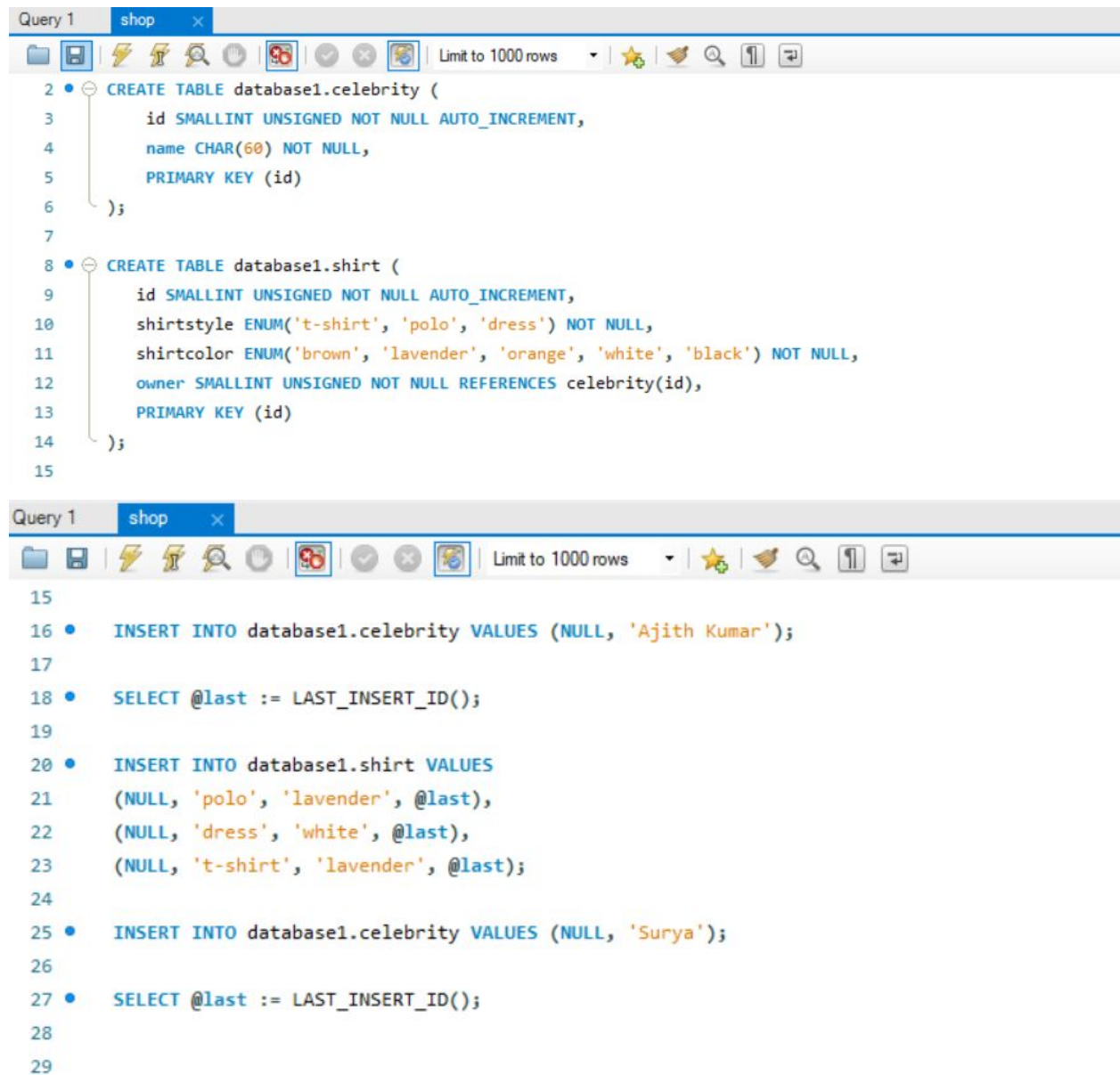
Query 1 shop x

```
1 • SELECT @min_price:=MIN(price),@max_price:=MAX(price) FROM database1.shop;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: |

	@min_price:=MIN(price)	@max_price:=MAX(price)
▶	1.25	19.95

STEP 11: create two tables celebrity and shirt and insert values



The screenshot displays a MySQL query editor window titled 'Query 1' with a tab labeled 'shop'. The editor contains two SQL queries. The first query creates two tables: 'celebrity' and 'shirt'. The 'celebrity' table has columns 'id' (SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT, PRIMARY KEY) and 'name' (CHAR(60) NOT NULL). The 'shirt' table has columns 'id' (SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT, PRIMARY KEY), 'shirtstyle' (ENUM('t-shirt', 'polo', 'dress') NOT NULL), 'shirtcolor' (ENUM('brown', 'lavender', 'orange', 'white', 'black') NOT NULL), and 'owner' (SMALLINT UNSIGNED NOT NULL REFERENCES celebrity(id)). The second query inserts data into these tables. It first inserts a record into 'celebrity' with name 'Ajith Kumar', then selects the last insert ID and uses it to insert three records into 'shirt' with styles 'polo', 'dress', and 't-shirt', all with color 'lavender'. Finally, it inserts a record into 'celebrity' with name 'Surya' and selects the last insert ID.

```
Query 1 shop x
2 • CREATE TABLE database1.celebrity (
3     id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
4     name CHAR(60) NOT NULL,
5     PRIMARY KEY (id)
6 );
7
8 • CREATE TABLE database1.shirt (
9     id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
10    shirtstyle ENUM('t-shirt', 'polo', 'dress') NOT NULL,
11    shirtcolor ENUM('brown', 'lavender', 'orange', 'white', 'black') NOT NULL,
12    owner SMALLINT UNSIGNED NOT NULL REFERENCES celebrity(id),
13    PRIMARY KEY (id)
14 );
15

Query 1 shop x
15
16 • INSERT INTO database1.celebrity VALUES (NULL, 'Ajith Kumar');
17
18 • SELECT @last := LAST_INSERT_ID();
19
20 • INSERT INTO database1.shirt VALUES
21 (NULL, 'polo', 'lavender', @last),
22 (NULL, 'dress', 'white', @last),
23 (NULL, 't-shirt', 'lavender', @last);
24
25 • INSERT INTO database1.celebrity VALUES (NULL, 'Surya');
26
27 • SELECT @last := LAST_INSERT_ID();
28
29
```



```

29
30 • INSERT INTO database1.shirt VALUES
31 (NULL, 'dress', 'orange', @last),
32 (NULL, 'polo', 'brown', @last),
33 (NULL, 'dress', 'lavender', @last),
34 (NULL, 't-shirt', 'white', @last);
35

```

STEP 12: Check whether the table has been updated with contents

Query 1 shop

Limit to 1000 rows

```

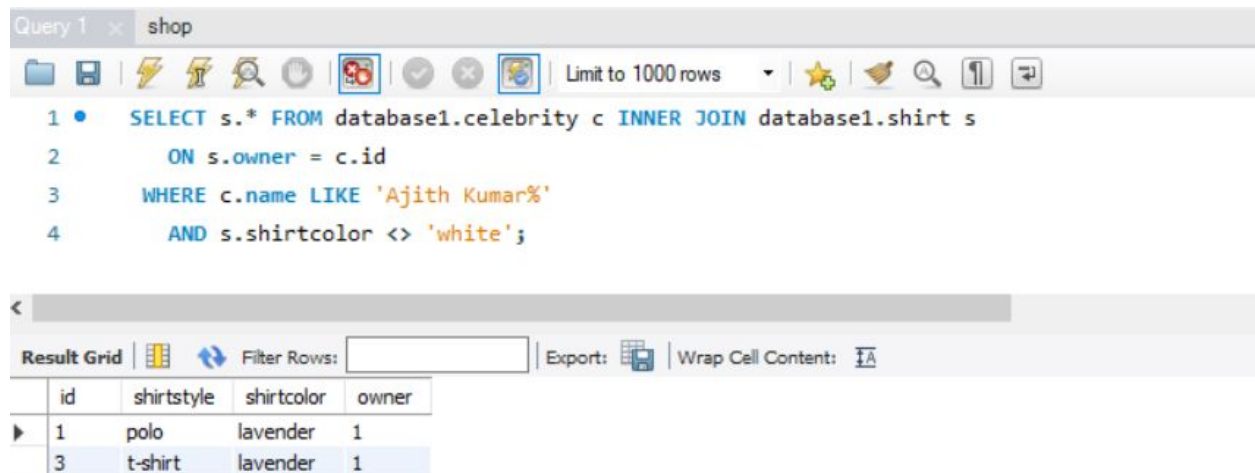
33 (NULL, 'dress', 'lavender', @last),
34 (NULL, 't-shirt', 'white', @last);
35
36 • SELECT * FROM database1.celebrity;
37 • SELECT * FROM database1.shirt;

```

Result Grid

	id	shirtstyle	shirtcolor	owner
▶	1	polo	lavender	1
	2	dress	white	1
	3	t-shirt	lavender	1
	4	dress	orange	2
	5	polo	brown	2
	6	dress	lavender	2
	7	t-shirt	white	2
*	NULL	NULL	NULL	NULL

STEP 13: Query using INNER JOIN and list shirts for celebrity Ajith other than color white..



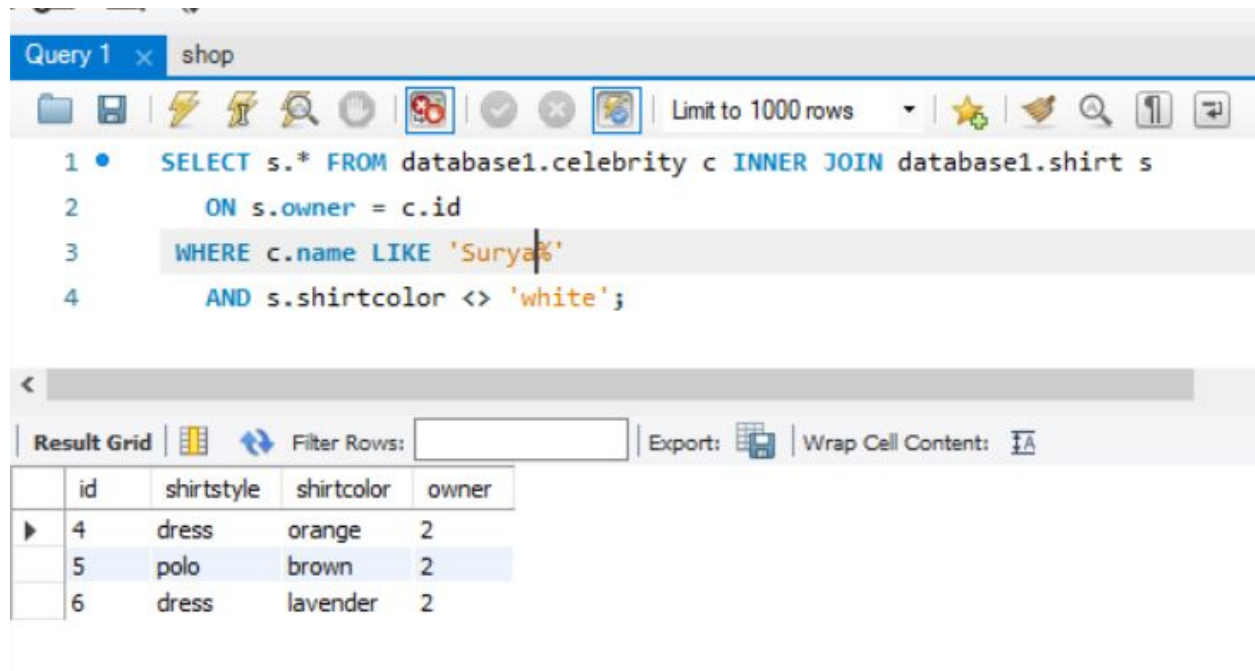
The screenshot shows a database query editor window titled "Query 1" with a tab labeled "shop". The query is as follows:

```
1 • SELECT s.* FROM database1.celebrity c INNER JOIN database1.shirt s
2     ON s.owner = c.id
3     WHERE c.name LIKE 'Ajith Kumar%'
4     AND s.shirtcolor <> 'white';
```

Below the query editor, the "Result Grid" is displayed with the following data:

	id	shirtstyle	shirtcolor	owner
▶	1	polo	lavender	1
	3	t-shirt	lavender	1

STEP 14: Query using INNER JOIN and list shirts for celebrity Surya other than color white..



The screenshot shows a database query editor window titled "Query 1" with a tab labeled "shop". The query is as follows:

```
1 • SELECT s.* FROM database1.celebrity c INNER JOIN database1.shirt s
2     ON s.owner = c.id
3     WHERE c.name LIKE 'Surya%'
4     AND s.shirtcolor <> 'white';
```

Below the query editor, the "Result Grid" is displayed with the following data:

	id	shirtstyle	shirtcolor	owner
▶	4	dress	orange	2
	5	polo	brown	2
	6	dress	lavender	2

STEP 15: Create Table visitsperday, Insert values and Query table using Bit group function to calculate the number of visits on a webpage eliminating duplicate entries.

```
Query 1 x shop
1 • CREATE TABLE database1.visitsperday (year YEAR, month INT UNSIGNED,
2     day INT UNSIGNED);
3 • INSERT INTO database1.visitsperday VALUES(2019,1,1),(2019,1,20),(2019,1,30),(2019,2,2),
4     (2019,2,23),(2019,2,23);
```

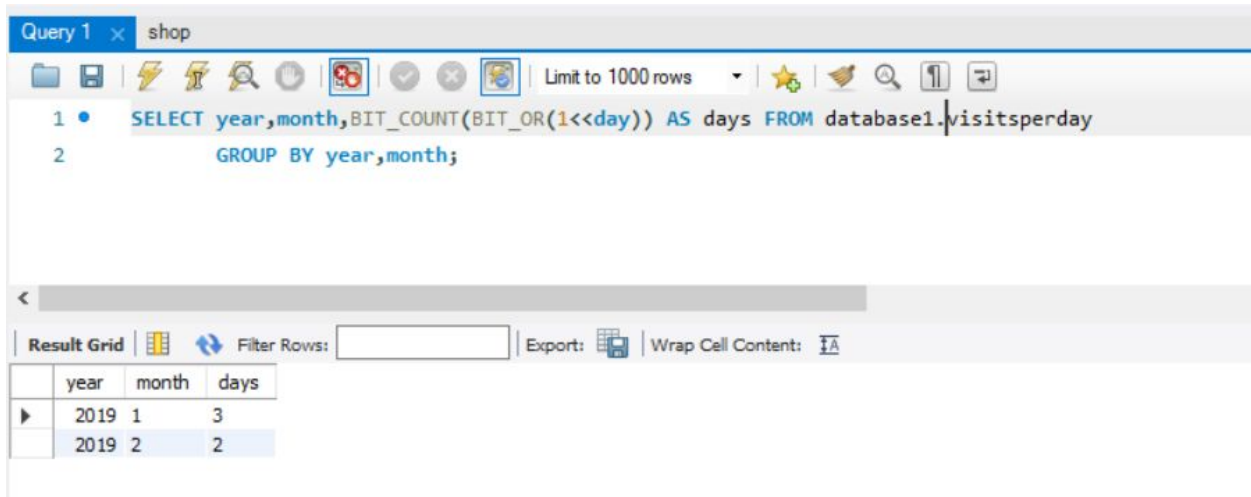
STEP 16: Query Table visitsperday

```
Query 1 x shop
1 • SELECT * FROM database1.visitsperday;
2
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: |

	year	month	day
▶	2019	1	1
	2019	1	20
	2019	1	30
	2019	2	2
	2019	2	23
	2019	2	23

STEP 17: Query table using Bit group function to calculate the number of visits on a webpage eliminating duplicate entries.



The screenshot shows a database query editor window titled "Query 1" with a tab labeled "shop". The SQL query is as follows:

```
1 • SELECT year,month,BIT_COUNT(BIT_OR(1<<day)) AS days FROM database1.visitsperday
2     GROUP BY year,month;
```

Below the query editor, the "Result Grid" is displayed, showing the results of the query. The grid has three columns: "year", "month", and "days". The results are as follows:

	year	month	days
▶	2019	1	3
	2019	2	2

STEP 18 : DROP celebrity table.

```
drop table database1.celebrity;
```

on Output

Time	Action	Message
02:18:58	SELECT @last := LAST_INSERT_ID() LIMIT 0, 1000	1 row(s) returned
02:18:58	SELECT * FROM database1.shirt LIMIT 0, 1000	24 row(s) returned
02:21:50	drop table database1.celebrity	0 row(s) affected

STEP 19 : DROP shirt table.



STEP 20 : VERIFY DROP table database1 shirt table Query again in order to verify if the tables were dropped.

