

# Database Creation

```
CREATE DATABASE IF NOT EXISTS task6_sales;
```

```
USE task6_sales;
```

```
CREATE TABLE online_sales (
```

```
    order_id INT,
```

```
    order_date DATE,
```

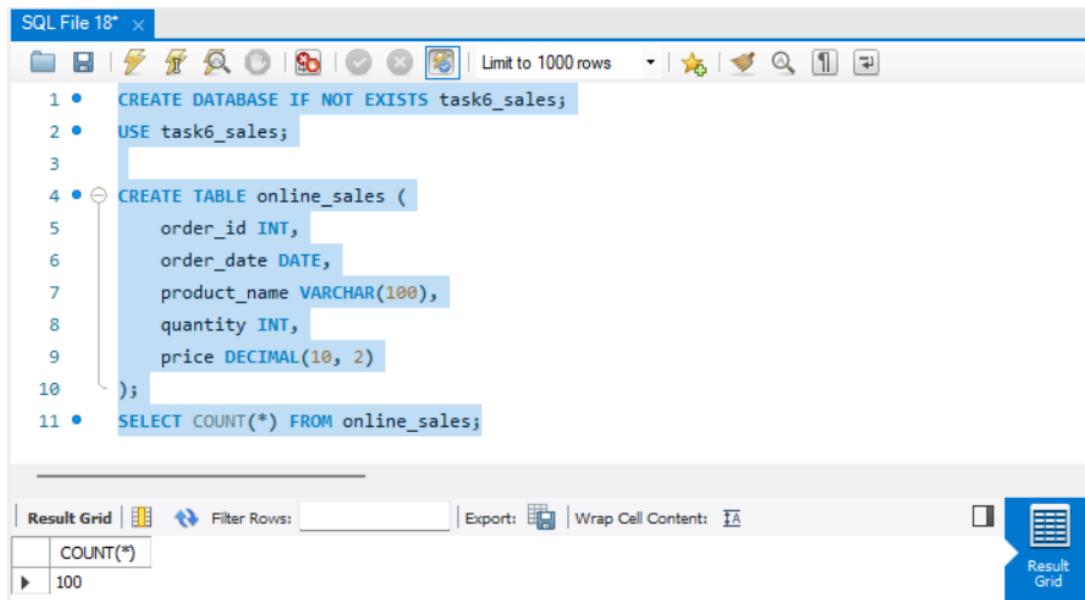
```
    product_name VARCHAR(100),
```

```
    quantity INT,
```

```
    price DECIMAL(10, 2)
```

```
);
```

```
SELECT COUNT(*) FROM online_sales;
```



The screenshot shows a MySQL Workbench interface with a SQL editor tab titled "SQL File 18\*". The code area contains the following SQL statements:

```
1 • CREATE DATABASE IF NOT EXISTS task6_sales;
2 • USE task6_sales;
3
4 • CREATE TABLE online_sales (
5     order_id INT,
6     order_date DATE,
7     product_name VARCHAR(100),
8     quantity INT,
9     price DECIMAL(10, 2)
10 );
11 • SELECT COUNT(*) FROM online_sales;
```

Below the code, there is a "Result Grid" table with one row of data:

COUNT(*)
100

A blue button labeled "Result Grid" is located to the right of the result grid.



## Query 1: Total Sales Amount

The screenshot shows a MySQL Workbench interface. At the top, there are two tabs: "SQL File 18\*" and "SQL File 20\*". Below the tabs is a toolbar with various icons for file operations, search, and navigation. A dropdown menu "Limit to 1000 rows" is open. The main area contains a SQL query:

```
1 •  SELECT
2      SUM(quantity * price) AS total_sales
3  FROM
4      online_sales;
5
```

Below the query is a horizontal line. Underneath it, the "Result Grid" tab is selected, showing the following data:

	total_sales
▶	8660500.00

There are also buttons for "Filter Rows", "Export", and "Wrap Cell Content".

```
SELECT
    SUM(quantity * price) AS total_sales
FROM
    online_sales;
```

📌 **Insight:** This query calculates the total revenue generated from all online sales. It multiplies quantity with price for each record and sums them up.

## ✓ Query 2: Total Orders

The screenshot shows a MySQL Workbench interface. The top bar has tabs for "SQL File 18\*", "SQL File 20\*", and "SQL File 21\*". The toolbar includes icons for file operations, search, and execution. A button for "Limit to 1000 rows" is visible. The main area contains the following SQL code:

```
1 •  SELECT
2      COUNT(DISTINCT order_id) AS total_orders
3  FROM
4      online_sales;
5
```

Below the code, there is a "Result Grid" tab selected, showing the following result:

total_orders
100

Other tabs in the results section include "Filter Rows:" and "Export:".

```
SELECT
    COUNT(DISTINCT order_id) AS total_orders
FROM
    online_sales;
```

📌 **Insight:** This gives the number of unique orders in the dataset. Helps understand the volume of transactions.

## ✓ Query 3: Total Quantity Sold

The screenshot shows a SQL editor interface with multiple tabs at the top labeled "SQL File 18\*", "SQL File 20\*", "SQL File 21\*", and "SQL File 22\*". The "SQL File 22\*" tab is active. Below the tabs is a toolbar with various icons for file operations like save, open, and search. A dropdown menu "Limit to 1000 rows" is visible. The main area contains a numbered SQL query:

```
1 •   SELECT
2       SUM(quantity) AS total_quantity_sold
3   FROM
4       online_sales;
5
```

Below the query is a "Result Grid" section with a single row of data:

	total_quantity_sold
▶	306

At the bottom of the grid are buttons for "Result Grid", "Filter Rows:", "Export:", and "Wrap Cell Content:".

```
SELECT
    SUM(quantity) AS total_quantity_sold
FROM
    online_sales;
```

💡 **Insight:** This query shows how many total items were sold. Useful for inventory or sales volume analysis.

## ✓ Query 4: Sales by Product

SQL File 18\* SQL File 20\* SQL File 21\* SQL File 22\* **SQL File 23\*** X

Folder | Refresh | Save | Run | Stop | Help | Open | Close | Limit to 1000 rows | Star | Print | Find | SQL

```
1
2 •   SELECT
3       product_name,
4           SUM(quantity * price) AS total_sales
5   FROM
6       online_sales
7   GROUP BY
8       product_name
9   ORDER BY
10      total_sales DESC;
```

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

	product_name	total_sales
▶	Smartwatch	1894500.00
	Headphones	1418000.00
	Tablet	1105500.00
	Mouse	1005500.00
	Monitor	951500.00
	Laptop	909000.00
	Keyboard	891000.00
	Smartphone	485500.00

Default 1 ▾

```
SELECT
    product_name,
    SUM(quantity * price) AS total_sales
FROM
    online_sales
GROUP BY
    product_name
ORDER BY
```

```
total_sales DESC;
```

📌 **Insight:** This shows which products generated the most revenue. Use it to identify top performers.

## ✓ Query 5: Most Sold Product (by Quantity)

The screenshot shows the MySQL Workbench interface with several tabs at the top: SQL File 18\*, SQL File 20\*, SQL File 21\*, SQL File 22\*, SQL File 23\*, and SQL File 24\*. Below the tabs is a toolbar with various icons. The main area contains the following SQL code:

```
1 •  SELECT
2      product_name,
3      SUM(quantity) AS total_quantity
4  FROM
5      online_sales
6  GROUP BY
7      product_name
8  ORDER BY
9      total_quantity DESC
10     LIMIT 1;
11
```

Below the code, there is a "Result Grid" section with the following data:

	product_name	total_quantity
▶	Smartwatch	58

```
SELECT
    product_name,
    SUM(quantity) AS total_quantity
FROM
    online_sales
GROUP BY
    product_name
```

```
ORDER BY
```

```
    total_quantity DESC
```

```
LIMIT 1;
```

📌 **Insight:** Finds the most sold product based on quantity. It might not be the most profitable but is in high demand.

## ✓ Query 6: Monthly Sales Trend

The screenshot shows a MySQL query editor interface. At the top, there are tabs for "SQL File 18\*", "SQL File 20\*", "SQL File 21\*", "SQL File 22\*", "SQL File 23\*", and "SQL File 24\*". Below the tabs is a toolbar with various icons for file operations, search, and database management. A dropdown menu labeled "Limit to 1000 rows" is open. The main area contains the SQL code for Query 6:

```
1 •   SELECT
2         DATE_FORMAT(order_date, '%Y-%m') AS month,
3             SUM(quantity * price) AS monthly_sales
4     FROM
5         online_sales
6     GROUP BY
7         month
8     ORDER BY
9         month;
10
```

Below the code, there is a "Result Grid" section with a table showing the monthly sales data:

	month	monthly_sales
▶	2023-01	2332500.00
	2023-02	1422500.00
	2023-03	2275000.00
	2023-04	2630500.00

```
SELECT
```

```
    DATE_FORMAT(order_date, '%Y-%m') AS month,
```

```
    SUM(quantity * price) AS monthly_sales
```

```
FROM  
    online_sales  
GROUP BY  
    month  
ORDER BY  
    month;
```

📌 **Insight:** Shows month-wise total sales, useful to identify sales trends over time.

## ✓ Query 7: Daily Orders

SQL File 20\* SQL File 21\* SQL File 22\* SQL File 23\* SQL File 24\* SQL File 25\*

Limit to 1000 rows

```
1 •  SELECT
2      order_date,
3      COUNT(DISTINCT order_id) AS orders_per_day
4  FROM
5      online_sales
6  GROUP BY
7      order_date
8  ORDER BY
9      order_date;
10
```

---

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

	order_date	orders_per_day
▶	2023-01-02	2
	2023-01-03	2
	2023-01-05	3
	2023-01-06	1
	2023-01-07	1
	2023-01-09	1
	2023-01-13	1

```
SELECT
    order_date,
    COUNT(DISTINCT order_id) AS orders_per_day
FROM
    online_sales
GROUP BY
    order_date
ORDER BY
```

```
order_date;
```

📌 **Insight:** This helps to analyze daily activity — can spot busy vs. slow days.

## Query 8: Top 3 Months by Revenue

The screenshot shows a MySQL query editor interface. At the top, there are tabs for "SQL File 21\*" through "SQL File 26\*". Below the tabs is a toolbar with various icons for file operations, search, and navigation. The main area contains the SQL code for Query 8, which retrieves the top 3 months by revenue from the "online\_sales" table. The code is as follows:

```
1 •  SELECT
2      DATE_FORMAT(order_date, '%Y-%m') AS month,
3      SUM(quantity * price) AS revenue
4  FROM
5      online_sales
6  GROUP BY
7      month
8  ORDER BY
9      revenue DESC
10     LIMIT 3;
11
```

Below the code, there is a "Result Grid" section. It includes buttons for "Result Grid" (selected), "Filter Rows:", "Export:" (with a file icon), "Wrap Cell Content:" (with a table icon), and "Fetch rows:". The result grid displays the following data:

	month	revenue
▶	2023-04	2630500.00
	2023-01	2332500.00
	2023-03	2275000.00

```
SELECT
```

```
    DATE_FORMAT(order_date, '%Y-%m') AS month,
```

```
    SUM(quantity * price) AS revenue
```

```
FROM
```

```
online_sales
```

GROUP BY

month

ORDER BY

revenue DESC

LIMIT 3;

💡 **Insight:** This shows the top 3 months where sales were highest. Very helpful for identifying peak business periods.

## Query 9: Using EXTRACT

The screenshot shows a SQL query editor interface with multiple tabs at the top labeled "SQL File 22\*", "SQL File 23\*", "SQL File 24\*", "SQL File 25\*", "SQL File 26\*", and "SQL File 27\*". Below the tabs is a toolbar with various icons for file operations, search, and filtering. The main area displays a numbered SQL query:

```
1 •  SELECT
2      EXTRACT(YEAR FROM order_date) AS year,
3      EXTRACT(MONTH FROM order_date) AS month,
4      SUM(quantity * price) AS revenue
5  FROM
6      online_sales
7  GROUP BY
8      year, month
9  ORDER BY
10     year, month;
11
```

Below the query, there is a "Result Grid" section with a table showing the results:

	year	month	revenue
▶	2023	1	2332500.00
	2023	2	1422500.00
	2023	3	2275000.00
	2023	4	2630500.00

```
SELECT  
    EXTRACT(YEAR FROM order_date) AS year,  
    EXTRACT(MONTH FROM order_date) AS month,  
    SUM(quantity * price) AS revenue  
FROM  
    online_sales  
GROUP BY  
    year, month  
ORDER BY  
    year, month;
```

 **Insight:** This query breaks sales down by year and month using SQL's EXTRACT() function.