

SUMMARY: DATE MANIPULATION

SESSION OVERVIEW:

By the end of this session, the students will be able to:

- Understand the DATE data type and its representation in SQL databases.
- Understand the difference between various DATE functions and their use cases.
- Perform DATE functions to retrieve, clean, create additional charts, and manipulate date values in SQL queries.

KEY TOPICS AND EXAMPLES:

Understand the DATE data type and its representation in SQL databases:

In MySQL, the DATE data type is used to store date values without time components. It represents a specific calendar date, consisting of a year, month, and day.

Here's an overview of the DATE data type and its representation in MySQL:

- **Format and Range:** The DATE data type stores dates in the format YYYY-MM-DD. The valid range for the DATE data type in MySQL is from '1000-01-01' to '9999-12-31'.
- **Internal Representation:** Internally, MySQL represents the DATE data type as a packed binary value, where the lower 5 bits represent the day of the month (0-31), the next 4 bits represent the month (1-12), and the remaining bits represent the year. The internal representation is optimized for efficient storage and retrieval of date values.
- **Storage Size:** The DATE data type in MySQL requires 3 bytes of storage.
- **Time Components:** The DATE data type does not store time components like hours, minutes, and seconds. It only represents the date portion. If you need to store both date and time components, you should use the DATETIME or TIMESTAMP data types instead.

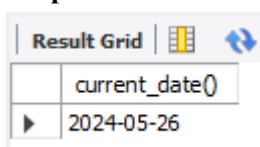
Understanding the difference between various DATE functions and their use cases:

1. CURRENT_DATE function:

In MySQL, the CURRENT_DATE returns the current date in 'YYYY-MM-DD' format or YYYYMMDD format depending on whether numeric or string is used in the function.

```
select current_date()
```

Output:



Result Grid
current_date()
2024-05-26

The output of the above query will return you today's date. The date in the screenshot might vary with

your output as the above query was run on 26 May 2024.

Example 1:

```
INSERT INTO orders (order_date, order_amount)
VALUES (CURRENT_DATE, 300.00);
```

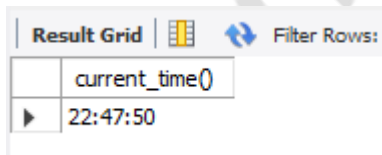
This is the use case of CURRENT_DATE function. Suppose I have received an order today, I can simply just run this query using the current date function to enter order_date as today's date which reduces the manual entry of the date of the order.

2. CURRENT_TIME function:

In MySQL the CURRENT_TIME() returns the current time in 'HH: MM: SS' format or HHMMSS.uuuuuu format depending on whether numeric or string is used in the function.

```
select current_time();
```

Output:



current_time()
22:47:50

Similar to the CURRENT_DATE function, we also have the CURRENT_TIME function which helps us retrieve the present time.

Example 1:

```
INSERT INTO events (event_name, event_time) VALUES ('Start Break',
CURRENT_TIME);
```

Here we are setting the time of the event as the current time. The use case is similar to the CURRENT_DATE function which allows us to enter the current time without manually entering it.

3. CURRENT-TIMESTAMP or NOW() function:

In MySQL, the CURRENT_TIMESTAMP returns the current date and time in the 'YYYY-MM-DD HH: MM: SS' format or YYYYMMDDHHMMSS.uuuuuu format depending on whether numeric or string is used in the function.

```
select current_timestamp();
```

```
select now();
```

Outputs:

Result Grid		Filter Rows:
	current_timestamp()	
▶	2024-05-26 22:49:20	

Result Grid		Filter Rows:
	now()	
▶	2024-05-26 22:49:44	

CURRENT_TIMESTAMP

NOW

In the above outputs, we can observe that CURRENT_TIMESTAMP and NOW functions return the same output and can be interchangeably used. These functions are very important from an industry perspective as current_timestamp or now functions help us to record the date and time in a single place.

Extracting date parts in SQL allows you to retrieve specific components (such as the year, month, day, hour, minute, second) from a date or timestamp. This is useful for various operations like grouping data by a particular time period, filtering records based on date parts, or performing date arithmetic.

1. **YEAR()**: Extracts the year part from a date or datetime value.

Example 1:

```
SELECT YEAR('2024-05-26') AS year_part;
```

Alternative 1:

```
SELECT EXTRACT(YEAR FROM CURRENT_DATE) AS current_day;
```

Alternative 2:

```
SELECT DATE_TRUNC('year', CURRENT_DATE) AS truncated_date;
```

Note: The above alternative is not widely used in the industry.

Output:

Result Grid		Filter Rows:
	year_part	
▶	2024	

Example 2:

```
select year (current_date()) as current_year;
```

Output:

Result Grid	
	year_part
	2024

The query helps us to get the year of the current date which implies the current year. For both the above queries, the output will be the same as the image attached above.

2. MONTH(): Extracts the month part from a date or datetime value.

Example 1:

```
SELECT MONTH('2024-03-26') AS month_part;
```

Output:

Result Grid	
	month_part
	3

Example 2:

```
select month(current_date()) as current_month;
```

Alternative:

```
SELECT EXTRACT(MONTH FROM CURRENT_DATE) AS current_day;
```

Alternative 2:

```
SELECT DATE_TRUNC('month', CURRENT_DATE) AS truncated_date;
```

Note: The above alternative is not widely used in the industry.

Output:

Result Grid	
	current_month
	5

The query helps us to get the month of the current date which implies the current month. For both the above queries, the output will be the same as the image attached above.

3. DAY(): Extracts the day part from a date or datetime value.

Example 1:

```
select day(current_date()) as current_month;
```

Alternative:

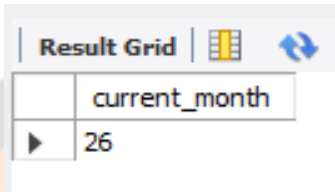
```
SELECT EXTRACT(DAY FROM CURRENT_DATE) AS current_day;
```

Alternative 2:

```
SELECT DATE_TRUNC('day', CURRENT_DATE) AS truncated_date;
```

Note: The above alternative is not widely used in the industry.

Output:



	current_month
▶	26

For both the above queries, the output will be the same as the image attached above.

Example:


Let's take this small example to understand how the above functions work. We are creating a separate hypothetical table named events and have put some of the entries into it. This example will help you understand how these functions work in terms of using other functions.

```
CREATE TABLE events (  
    event_id INT AUTO_INCREMENT PRIMARY KEY,  
    event_name VARCHAR(255),  
    event_date DATE  
);  
  
INSERT INTO events (event_name, event_date) VALUES  
( 'Event A', '2024-01-15'),  
( 'Event B', '2024-02-20'),  
( 'Event C', '2024-05-26'),  
( 'Event D', '2024-12-31');
```

1. Extracting year, month, and day:

```
SELECT  
    event_name,  
    YEAR(event_date) AS year_part,  
    MONTH(event_date) AS month_part,  
    DAY(event_date) AS day_part  
FROM  
    events;
```


Output:

Result Grid  Filter Rows: <input type="text"/> Export:				
	event_name	year_part	month_part	day_part
▶	Event A	2024	1	15
	Event B	2024	2	20
	Event C	2024	5	26
	Event D	2024	12	31

2. Grouping Events by Year:

```
SELECT YEAR(event_date) AS year_part,
       COUNT(*) AS event_count
FROM events
GROUP BY year_part;
```

Output:

Result Grid  Filter Rows:		
	year_part	event_count
▶	2024	4

This example will help you understand that extracting year, month, or day from the given date is possible using the aggregation function or any other functions that we have discussed in our previous sessions.

Date arithmetic in MySQL involves performing operations on dates and timestamps to calculate durations, add or subtract time intervals, and manipulate date values. MySQL provides several functions and operators for these operations.


1. **Adding and Subtracting Intervals:** This helps us add or subtract intervals (such as days, months, and years) to/from a date or timestamp using the `DATE_ADD()` and `DATE_SUB()` functions.

Note: With the help of examples we will be learning the syntax of the date arithmetic functions which will also help you to understand how these functions work in a single place.

Example 1:

```
-- Adding 1 day to a date
SELECT DATE_ADD('2024-05-26', INTERVAL 1 DAY) AS next_day;
```

Output:

Result Grid  Filter Rows:	
	next_day
▶	2024-05-27

Example 2:

```
-- Adding 10 days to a date
SELECT DATE_ADD('2024-05-26', INTERVAL 10 DAY) AS next_day;
```

Output:

Result Grid		Filter Rows:
	next_day	
▶	2024-06-05	

We can observe that it is adding a day to the specified date in the query.

Example 2:

```
-- Subtracting 1 month from a date
SELECT DATE_SUB('2024-05-26', INTERVAL 1 MONTH) AS previous_month;
```

Output:

Result Grid		Filter Rows:
	previous_month	
▶	2024-04-26	

We can observe that the query has helped us reduce one month from the specified date in the query.

Example 3:

```
-- Adding 1 year to a timestamp
SELECT DATE_ADD('2024-05-26 14:30:00', INTERVAL 1 YEAR) AS next_year;
```

Output:

Result Grid		Filter Rows:
	next_year	
▶	2025-05-26 14:30:00	

We can observe that it is adding a year to the specified date in the query.

Example 4:

```
SELECT DATE_ADD(DATE_ADD('2024-05-26', INTERVAL 1 MONTH), INTERVAL 10
DAY) AS combined_interval;
```

Output:

Result Grid		Filter Rows:
	combined_interval	
▶	2024-07-06	

2. **Date Difference:** In MySQL, you can calculate the difference between two dates or timestamps using the **DATEDIFF() function**. This function returns the difference in days between two dates.

Note: Here we will be using a hypothetical example to understand how Date difference works. This example will help you build a fundamental understanding of this important concept. The implementation of all related functions in the dataset will be in the next half of the session.

Example:

Let's create this small hypothetical table and insert some entries to understand the base of the concept.



```
CREATE TABLE orders (
  order_id INT AUTO_INCREMENT PRIMARY KEY,
  order_date DATE,
  delivery_date DATE
);
INSERT INTO orders (order_date, delivery_date) VALUES
('2024-05-01', '2024-05-05'),
('2024-05-10', '2024-05-15'),
('2024-05-20', '2024-05-25'),
('2024-06-01', '2024-06-03');
```

Now we'll use the DATEDIFF() function to calculate the difference in days between delivery_date and order_date for each order.

```
SELECT order_id, order_date, delivery_date,
       DATEDIFF(delivery_date, order_date) AS delivery_days
FROM orders;
```

Output:

Result Grid

Filter Rows:

Export:

	order_id	order_date	delivery_date	delivery_days
▶	1	2024-05-01	2024-05-05	4
	2	2024-05-10	2024-05-15	5
	3	2024-05-20	2024-05-25	5
	4	2024-06-01	2024-06-03	2

The above query calculates the difference between the days of delivery_date and order_date to find the number of days that the company took to deliver its orders to the customers.

How to handle Negative differences while using the DATEDIFF function:

Here are certain ways that can be used to handle negative differences in the number of days. The demonstration is on the above example that we have taken to understand the DATEDIFF function.

1. **Using ABS() Function:** You can use the ABS() function to get the absolute value of the date difference, which removes the sign and ensures that all differences are positive.


```
SELECT order_id, order_date, delivery_date,  
       ABS(DATEDIFF(delivery_date, order_date)) AS absolute_delivery_days  
FROM orders;
```

The output for the above query will be the same as the one that we have derived in the previous query. This is because in the previous query, we didn't have any negative value in the delivery_days. But this query will help you get rid of the negative values and will always return absolute values.

2. **Using Conditional Logic:** You can use the CASE statement to handle negative differences differently, for example, by setting them to zero or some other value.

```
SELECT order_id, order_date, delivery_date,  
       CASE  
         WHEN DATEDIFF(delivery_date, order_date) < 0 THEN 0  
         ELSE DATEDIFF(delivery_date, order_date)  
       END AS adjusted_delivery_days  
FROM orders;
```

The output for the above query will also be the same as the one that we have derived in the previous query. This query will help us ignore or bring the negative entries to our attention if the entries were mistaken. Here, the negative entries will give us the value of 0.

Date formatting in MySQL involves converting date or datetime values into a specific string format. The DATE_FORMAT() function is used to format date values according to a specified format string.

Common Format Specifiers:

Here are some common format specifiers you can use with DATE_FORMAT():

- **%Y:** Year, numeric, four digits
- **%y:** Year, numeric, two digits
- **%m:** Month, numeric (00-12)
- **%M:** Month name (January-December)
- **%d:** Day of the month, numeric (00-31)
- **%H:** Hour (00-23)
- **%i:** Minutes, numeric (00-59)
- **%s:** Seconds, numeric (00-59)
- **%b:** Abbreviated month name (Jan-Dec)
- **%W:** Weekday name (Sunday-Saturday)
- **%w:** Day of the week (0=Sunday, 6=Saturday)
- **%a:** Abbreviated weekday name (Sun-Sat)

Example 1: Formatting Date to Month Day, Year

```
SELECT DATE_FORMAT('2024-05-26', '%M %d, %Y') AS formatted_date;
```

Output:

Result Grid	Filter Rows:
formatted_date	
May 26, 2024	

Example 2: Formatting DateTime to Day of the Week, Month Day, Year Hour:Minute: Second

```
SELECT DATE_FORMAT('2024-05-26 14:30:00', '%W, %M %d, %Y %H:%i:%s') AS formatted_datetime;
```

Output:

Result Grid	Filter Rows:
formatted_datetime	
Sunday, May 26, 2024 14:30:00	

Example 3: Extracting month name

Note: In the previous topic we have seen how to extract the month number which might get a little confusing while deriving insights related to months. Here DATE_FORMAT functions play an important role in extracting the month name which gives a better understanding of the months.

```
SELECT DATE_FORMAT('2024-05-26', '%M') AS month_name;
```

Output:

Result Grid	Filter Rows:
month_name	
May	

Example 4:

Here we are taking the same table that we have created in the previous section named orders.

```
SELECT order_id, order_date, delivery_date,
       DATE_FORMAT(order_date, '%b %d, %Y') AS formatted_order_date,
       DATE_FORMAT(delivery_date, '%b %d, %Y') AS formatted_delivery_date
FROM orders;
```

Output:

Result Grid

Filter Rows:

Export:

Wrap Cell Content:

	order_id	order_date	delivery_date	formatted_order_date	formatted_delivery_date
▶	1	2024-05-01	2024-05-05	May 01, 2024	May 05, 2024
	2	2024-05-10	2024-05-15	May 10, 2024	May 15, 2024
	3	2024-05-20	2024-05-25	May 20, 2024	May 25, 2024
	4	2024-06-01	2024-06-03	Jun 01, 2024	Jun 03, 2024

Converting strings to dates in MySQL is often necessary when dealing with data imported from external sources where date values are stored as strings. MySQL provides several functions to handle this conversion, including STR_TO_DATE() and CAST().

***NOTE:** This section will help you understand the difference between the concepts of the DATE_FORMAT() function and the STR_TO_DATE() function.*

1. **STR_TO_DATE:** The STR_TO_DATE() function converts a string into a date using a specified format.

Example 1: Month Name Conversion

Convert a string with a month name to a date.

```
SELECT STR_TO_DATE('26 May 2024', '%d %M %Y') AS converted_date;
```

Output:

Result Grid	
	converted_date
▶	2024-05-26

***NOTE:** The instructor will perform the above functions in the Calendar dataset for clear understanding.*

How are date_format() and str_to_date() different from each other?

The DATE_FORMAT() and STR_TO_DATE() functions in MySQL serve different purposes and are used in different contexts.

DATE_FORMAT() Function	STR_TO_DATE() Function
The DATE_FORMAT() function is used to format a date or datetime value into a string according to a specified format.	The STR_TO_DATE() function is used to convert a string into a date or datetime value using a specified format.
Converts a date or datetime value to a string based on a format.	Converts a string to a date or datetime value based on a format.
Input: A date or datetime value. Output: A formatted string.	Input: A string. Output: A date or datetime value.
Used when you need to display a date or datetime value in a specific format for reporting or presentation.	Used when you need to convert strings (often from user input or text files) into date or datetime values for further date-related operations.

Example: Combined query of DATE_FORMAT and STR_TO_DATE

This example will help you better understand and notice the difference between both functions. Here we will be creating another hypothetical table where the date column will be in string format.

```
CREATE TABLE events_string (
  event_id INT AUTO_INCREMENT PRIMARY KEY,
  event_date_string VARCHAR(50)
);
INSERT INTO events_string (event_date_string) VALUES
('26 May 2024 14:30:00'),
('15 June 2024 10:00:00'),
('04 July 2024 18:45:00');
```

Once you have created this table, run the below-mentioned query.

```
SELECT event_id, event_date_string,
       STR_TO_DATE(event_date_string, '%d %M %Y %H:%i:%s') AS
converted_date,
       DATE_FORMAT(STR_TO_DATE(event_date_string, '%d %M %Y %H:%i:%s'),
'%W, %M %d, %Y %H:%i:%s') AS formatted_date
FROM events_string;
```

Output:

Result Grid Filter Rows: Export: Wrap Cell Content:				
	event_id	event_date_string	converted_date	formatted_date
▶	1	26 May 2024 14:30:00	2024-05-26 14:30:00	Sunday, May 26, 2024 14:30:00
	2	15 June 2024 10:00:00	2024-06-15 10:00:00	Saturday, June 15, 2024 10:00:00
	3	04 July 2024 18:45:00	2024-07-04 18:45:00	Thursday, July 04, 2024 18:45:00

From the output, you can figure out how both the functions are different from each other. Make use of the above-mentioned abbreviation list to understand it in a better way.

Date cleaning:

One of the very important steps when working with date columns is cleaning the column before we step onto any analysis. Moreover, when we import data in MySQL Workbench it does not allow us to set the date column data types as date, rather it gets saved as text data type which makes it difficult while performing analysis. Here we will be using the calendar dataset that we have provided you in the beginning of the module. First, we will learn how to clean the date columns correctly.

Steps to the data type when the format of the column is uniform:

1. In the calendar dataset, first of all, we have to change the name of the “date” column as it might create a lot of confusion when performing the cleaning because the data type for the date column is also called a date. Thus it is suggested not to name the columns as the data type names.

```
alter table calendar
rename column date to calendar_date;
```

2. In the second step, we will update the calendar _date to our desired format using STR_TO_DATE. You can refer to the STR_TO_DATE portion to get a better understanding of how to keep the format uniform.

```
UPDATE calendar
SET calendar_date = STR_TO_DATE(calendar_date, '%d-%m-%Y');
```

3. Now we have made the format of the column uniform thus we can alter the data type of the calendar_date from text to date using the alter table statement.

```
alter table calendar
modify calendar_date date;
```

Now we can perform any analysis in the calendar_date column of the calendar table.

Steps to the data type when the format of the column is not uniform:

In the sales tables, we can observe that we have two columns named orderdate and stockdate where the format is not uniform which might create problem when working with these columns. Here is how we can make the format of these columns uniform using certain functions.

1. Here we will update the orderdate column with a uniform format.

```
UPDATE sales_2015
SET OrderDate = STR_TO_DATE(OrderDate, '%d-%m-%Y')
WHERE OrderDate LIKE '__-__-__' AND STR_TO_DATE(OrderDate, '%d-%m-%Y')
IS NOT NULL;
```

NOTE:

- Choosing which rows to change: It looks at all the rows in the sales_2015 table. It selects only those rows where the orderDate looks like it's written as mm-dd-yyyy (for example, 12/31/2023).
- Changing the date format: For each selected row, it takes the date written as mm-dd-yyyy. It changes this date into a standard date format recognized by MySQL.
- Ignoring already correct dates: If the date is already in a standard format or if it's not possible to convert it, the query skips that row.

2. The second step is to alter the column data type using an alter statement.

```
alter table sales_2015
modify OrderDate date;
```

Now the data type of the orderdate column has been changed and can be used in queries where we have to analyze the order date of the sales in the year 2015.

Similarly, we can perform the same operation in the stock date column in the sales_2015 table to get a uniform format. Here is the query:

```
UPDATE sales_2015
SET StockDate = STR_TO_DATE(StockDate, '%d-%m-%Y')
WHERE StockDate LIKE '__-__-__' AND STR_TO_DATE(StockDate, '%d-%m-%Y')
IS NOT NULL;
```

The next step is to convert the data type of the stock date column to date.

```
alter table sales_2015
modify stockdate date;
```

After going through the tables we figured out that the returns table has a column named returndate which has some non-uniformity in the date column. Here is how we can fix this:

```
UPDATE returns
SET returndate = CASE
    WHEN returndate LIKE '%-%-%' THEN STR_TO_DATE(returndate,
'%Y-%m-%d')
    ELSE STR_TO_DATE(returndate, '%m/%d/%Y')
END;
```

Once the format is appropriate, we will convert the alter statement into a date data type.

```
ALTER TABLE returns
MODIFY COLUMN returndate DATE;
```

MISCELLANEOUS CONCEPTS:

1. LAST_DAY():

The LAST_DAY() function in MySQL is used to return the last day of the month for a given date.

Example 1:

```
SELECT LAST_DAY('2024-05-26') AS last_day_of_month;
```

Output:

Result Grid		Filter Rows:
	last_day_of_month	
▶	2024-05-31	

Example 2:

```
SELECT LAST_DAY(DATE_ADD('2024-05-26', INTERVAL 1 MONTH)) AS
last_day_of_next_month;
```

Output:

Result Grid	Filter Rows:
last_day_of_next_month	
2024-06-30	

Example 3:

```
SELECT '2024-02-15' AS input_date, LAST_DAY('2024-02-15') AS
last_day_of_month
UNION ALL
SELECT '2024-06-10', LAST_DAY('2024-06-10')
UNION ALL
SELECT '2024-11-01', LAST_DAY('2024-11-01');
```

Output:

Result Grid	Filter Rows:
input_date	last_day_of_month
2024-02-15	2024-02-29
2024-06-10	2024-06-30
2024-11-01	2024-11-30

2. CONVERT_TZ():

The CONVERT_TZ() function in MySQL is used to convert a datetime value from one timezone to another. It takes three arguments: the datetime value to be converted, the current timezone of the datetime value, and the timezone to convert to.

```
SELECT CONVERT_TZ('2024-05-26 10:00:00', 'UTC', 'America/New_York') AS
converted_time;
```

3. UNIX TIME:

Unix time, also known as Epoch time or POSIX time, is a system for tracking time that is widely used in computing.

```
SELECT FROM_UNIXTIME(1622520000) AS date_from_timestamp;
```

Result Grid	Filter Rows:
date_from_timestamp	
2021-06-01 09:30:00	

```
SELECT UNIX_TIMESTAMP('2024-05-26 10:00:00') AS timestamp_from_date;
```

Result Grid		Filter Rows:
	timestamp_from_date	
▶	1716697800	

Questions on Date Manipulation:

These questions will help you to understand how date functions play an important role in terms of analysis. The instructor will explain all the queries mentioned below in the class in detail.

Example 1:

```
SELECT pc.categoryname, SUM(r.returnquantity) AS total_return_quantity
FROM returns r
JOIN products p ON r.productkey = p.productkey
JOIN product_subcategories ps ON p.productsubcategorykey =
ps.productsubcategorykey
JOIN product_categories pc ON ps.productcategorykey =
pc.productcategorykey
WHERE YEAR(r.returndate) = 2017
GROUP BY pc.categoryname;
```

Output:

Result Grid		Filter Rows:
	categoryname	total_return_quantity
▶	Accessories	639
	Bikes	171
	Clothing	162

Example 2:

```
SELECT YEAR(r.returndate) AS year, AVG(r.returnquantity) AS
avg_return_quantity_per_customer
FROM returns r
GROUP BY YEAR(r.returndate);
```

Output:

Result Grid		Filter Rows:
	year	avg_return_quantity_per_customer
▶	2015	1.0118
	2016	1.0079
	2017	1.0125

Example 3:


```
SELECT YEAR(s.orderdate) AS year, t.region, ROUND(SUM(s.orderquantity *
p.productprice), 2) AS total_sales_revenue
FROM (
    SELECT * FROM sales_2015
    UNION ALL
    SELECT * FROM sales_2016
    UNION ALL
    SELECT * FROM sales_2017
) AS s
JOIN products p ON s.productkey = p.ProductKey
JOIN territories t ON s.territorykey = t.salesterritorykey
GROUP BY YEAR(s.orderdate), t.region
order by total_sales_revenue desc;
```

Output:

Result Grid			
Filter Rows:			
	year	region	total_sales_revenue
▶	2016	Australia	2887686.68
	2017	Australia	2408104.87
	2015	Australia	2120664.65
	2017	Southwest	1911216.91
	2016	Southwest	1647370.18
	2015	Southwest	1264207.61
	2016	United Kingdom	1216441.26
	2017	Northwest	1208890.84
	2017	United Kingdom	1119087.76
	2016	Northwest	1051683.04
	2017	Germany	1015557.33

Example 4:

```
SELECT p.productname,
    AVG(DATEDIFF(s.orderdate, s.stockdate)) AS average_days_to_sell
FROM sales_2017 s
JOIN products p ON s.productkey = p.ProductKey
GROUP BY p.productname
ORDER BY average_days_to_sell DESC;
```

Output:

Result Grid			Filter Rows:	
	productname	average_days_to_sell		
▶	Touring-2000 Blue, 54	4824.4490		
	Touring-3000 Yellow, 44	4823.9730		
	Mountain-500 Black, 52	4823.5000		
	Mountain-500 Black, 42	4823.3704		
	Touring-3000 Blue, 58	4822.8125		
	Road-250 Black, 48	4822.0156		
	Touring-3000 Blue, 62	4821.0238		
	Mountain-400-W Silver, 46	4820.3571		
	Mountain-500 Silver, 44	4820.3333		
	Mountain-500 Silver, 44	4820.3333		

Example 5: Analyze Seasonal Trends for the year 2015

```

SELECT YEAR(s.orderdate) AS year,
       MONTH(s.orderdate) AS month,
       pc.categoryname,
       SUM(s.orderquantity) AS total_sales_quantity
FROM sales_2015 s
     JOIN products p ON s.productkey = p.ProductKey
     JOIN product_subcategories ps ON p.productssubcategorykey =
ps.productssubcategorykey
     JOIN product_categories pc ON ps.productcategorykey =
pc.productcategorykey
GROUP BY YEAR(s.orderdate), MONTH(s.orderdate), pc.categoryname
ORDER BY YEAR(s.orderdate), MONTH(s.orderdate);

```

Output:

Result Grid			Filter Rows:		Export:
	year	month	categoryname	total_sales_quantity	
▶	2015	1	Bikes	198	
	2015	2	Bikes	164	
	2015	3	Bikes	219	
	2015	4	Bikes	209	
	2015	5	Bikes	194	
	2015	6	Bikes	219	
	2015	7	Bikes	239	
	2015	8	Bikes	269	
	2015	9	Bikes	186	
	2015	10	Bikes	217	
	2015	11	Bikes	218	
	2015	12	Bikes	298	