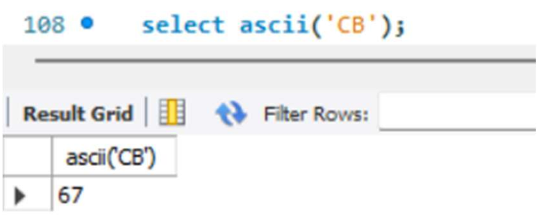
Name: Krushnakumar Patle

Email: krishnapatle128@gmail.com

Batch: Data Engineering Batch-1

1. String functions

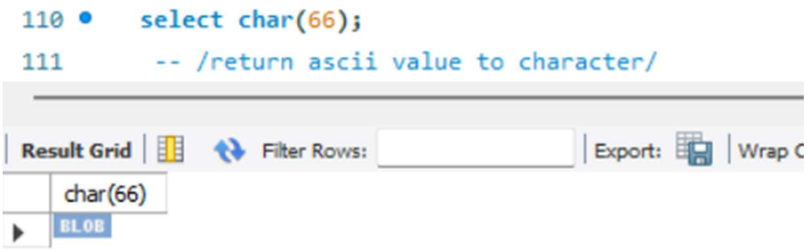
Calculate ascii value in sql



```
select ascii('CB');
```

```
ascii('CB')
67
```

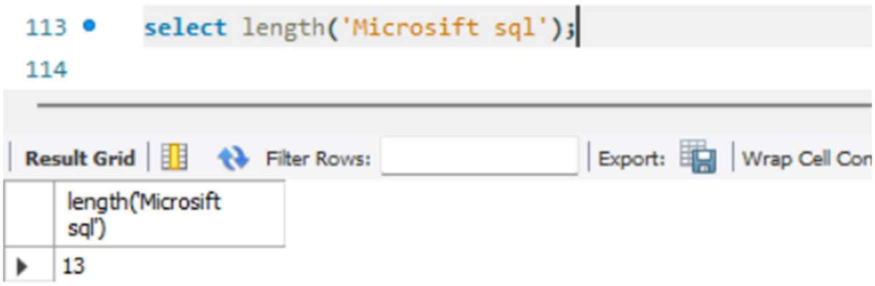
Calculate ascii of char



```
select char(66);
 -- /return ascii value to character/
```

```
char(66)
BLOB
```

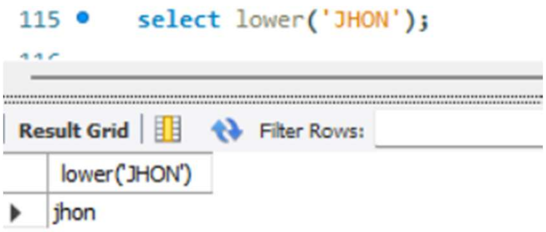
Calculate length in sql



```
select length('Micrrosift sql');
```

```
length('Microsift
sql')
13
```

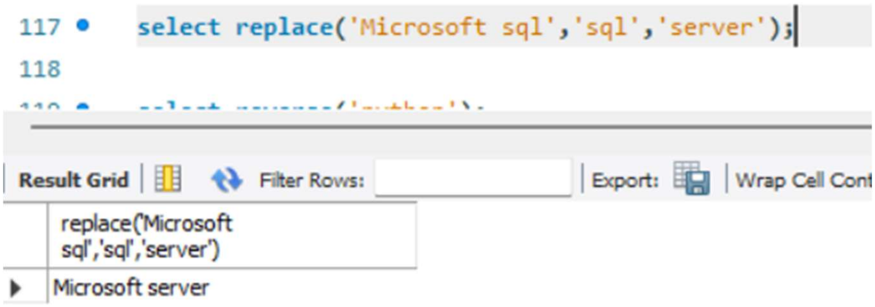
Use lower function in sql



```
select lower('JHON');
```

```
lower('JHON')
jhon
```

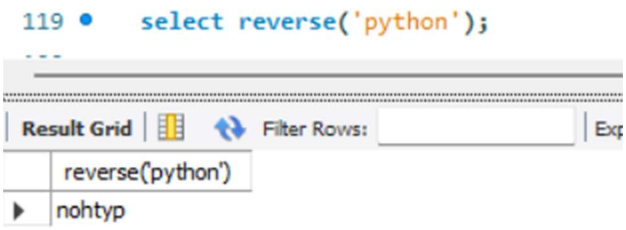
Use replace function in sql



```
select replace('Microsoft sql','sql','server');
```

```
replace('Microsoft
sql','sql','server')
Microsoft server
```

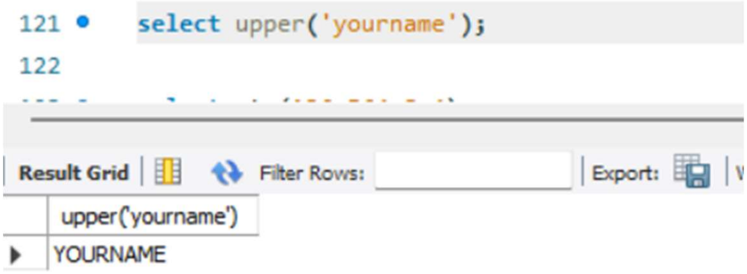
Use reverse function in sql



```
select reverse('python');
```

```
reverse('python')
nohtyp
```

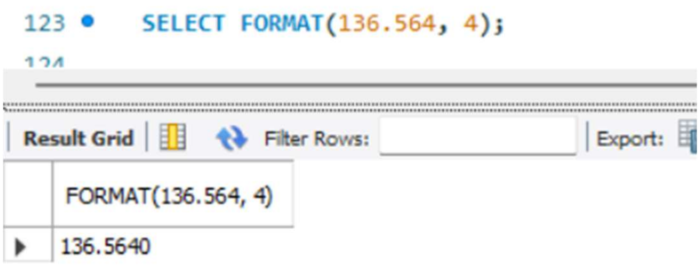
Use upper function in sql



```
select upper('yourname');
```

```
upper('yourname')
YOURNAME
```

Use format function in sql

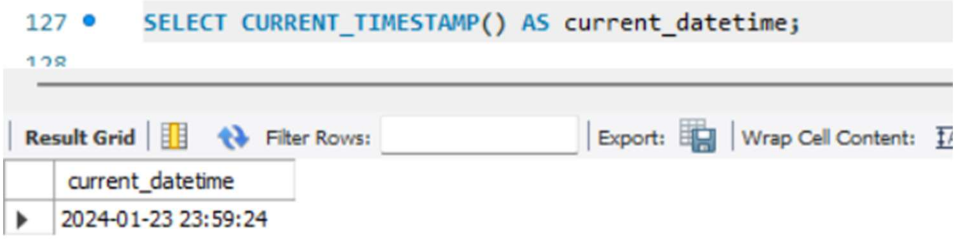


```
SELECT FORMAT(136.564, 4);
```

```
FORMAT(136.564, 4)
136.5640
```

2. Date Functions

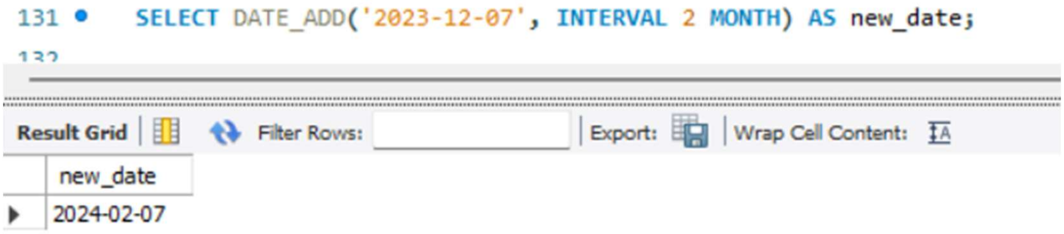
Calculate current datetime in sql



```
SELECT CURRENT_TIMESTAMP() AS current_datetime;
```

```
current_datetime
2024-01-23 23:59:24
```

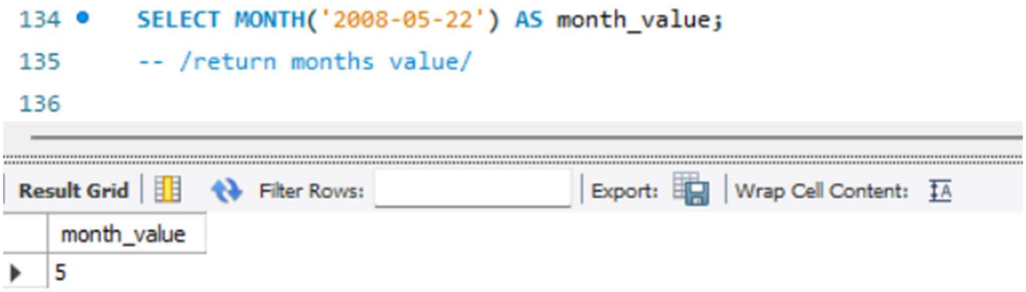
Use DATE_ADD function in sql



```
SELECT DATE_ADD('2023-12-07', INTERVAL 2 MONTH) AS new_date;
```

```
new_date
2024-02-07
```

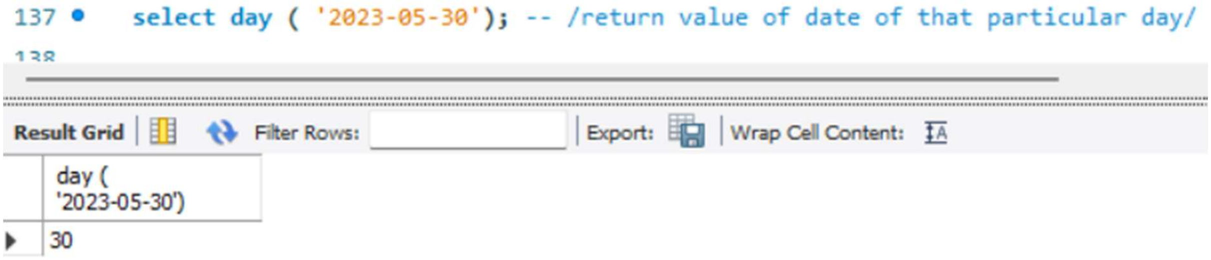
Calculate month from date



```
SELECT MONTH('2008-05-22') AS month_value;
-- /return months value/
```

```
month_value
5
```

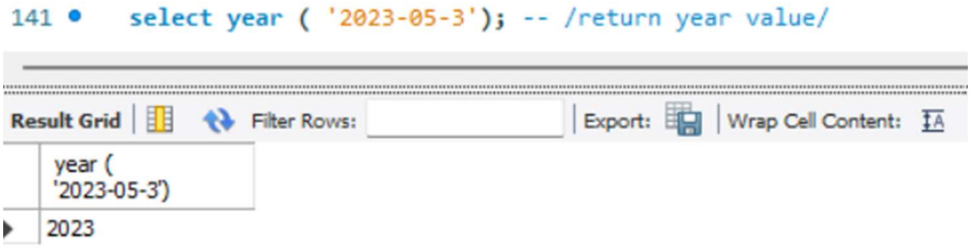
Calculate day



```
select day ( '2023-05-30'); -- /return value of date of that particular day/
```

```
day (
'2023-05-30')
30
```

Calculate year

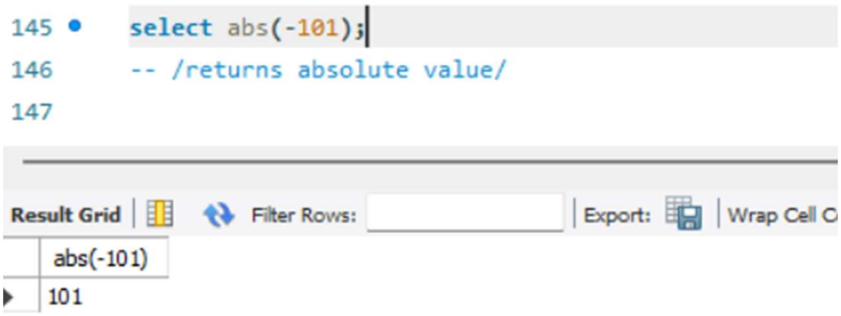


```
select year ( '2023-05-3'); -- /return year value/
```

```
year (
'2023-05-3')
2023
```

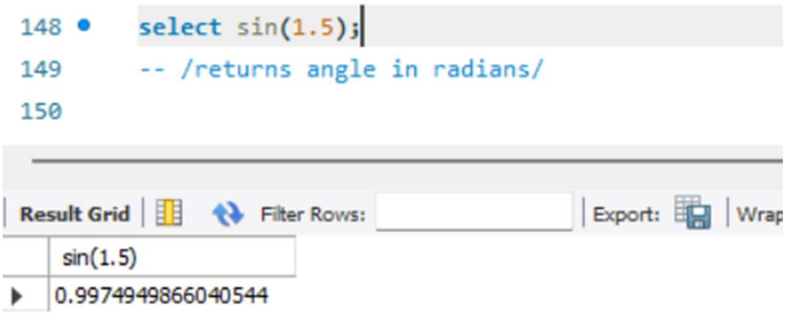
3. Mathematical Functions

Day 5 Assessment



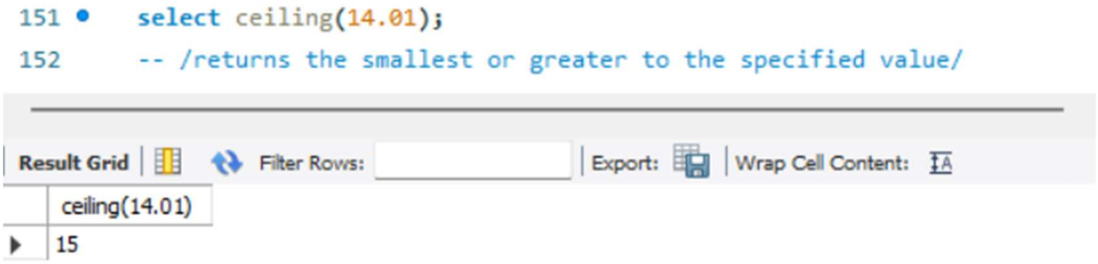
```sql
select abs(-101);
-- /returns absolute value/
```

```
abs(-101)
101
```



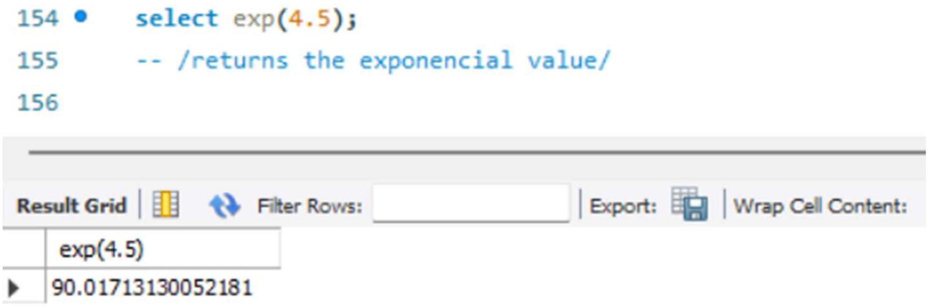
```sql
select sin(1.5);
-- /returns angle in radians/
```

```
sin(1.5)
0.9974949866040544
```



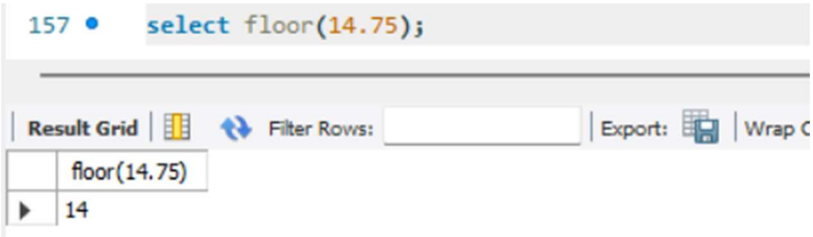
```sql
select ceiling(14.01);
-- /returns the smallest or greater to the specified value/
```

```
ceiling(14.01)
15
```



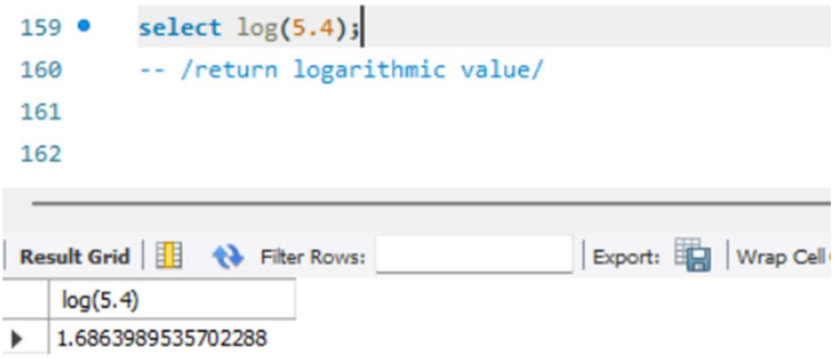
```sql
select exp(4.5);
-- /returns the exponencial value/
```

```
exp(4.5)
90.01713130052181
```



```sql
select floor(14.75);
```

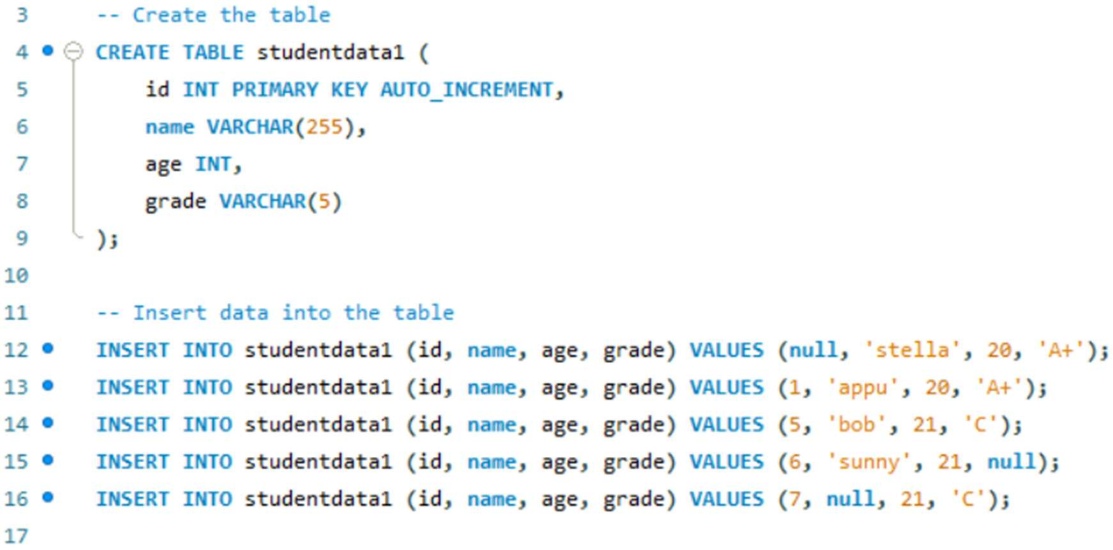
```
floor(14.75)
14
```



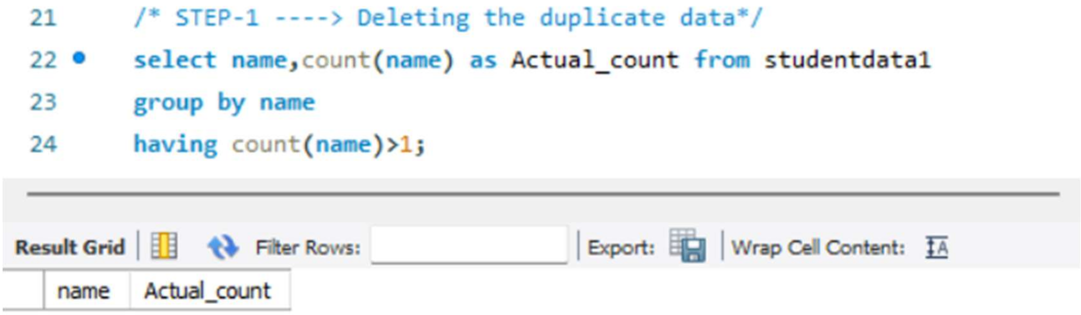
```sql
select log(5.4);
-- /return logarithmic value/
```

```
log(5.4)
1.6863989535702288
```

4. Data cleaning and transformation

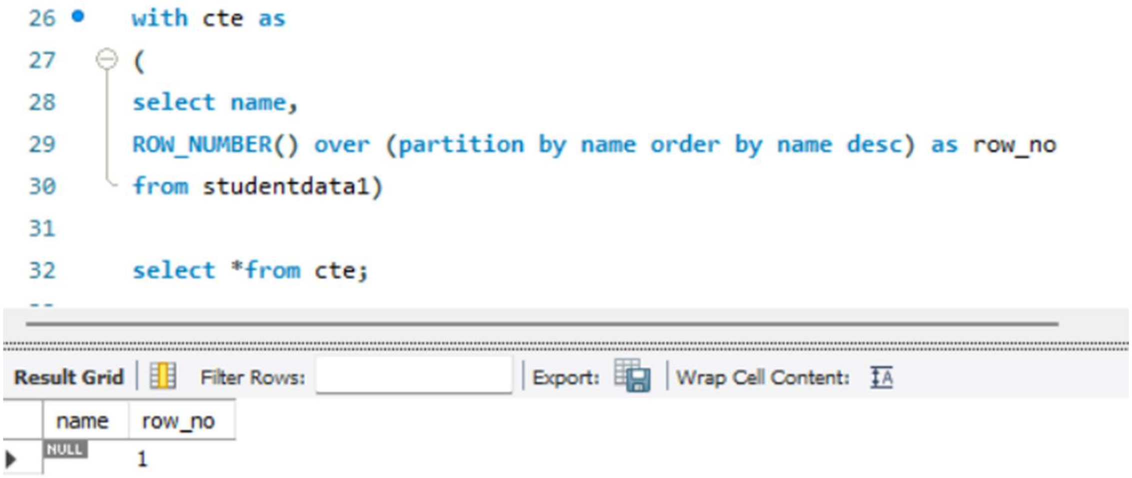


```sql
-- Create the table
CREATE TABLE studentdata1 (
    id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(255),
    age INT,
    grade VARCHAR(5)
);

-- Insert data into the table
INSERT INTO studentdata1 (id, name, age, grade) VALUES (null, 'stella', 20, 'A+');
INSERT INTO studentdata1 (id, name, age, grade) VALUES (1, 'appu', 20, 'A+');
INSERT INTO studentdata1 (id, name, age, grade) VALUES (5, 'bob', 21, 'C');
INSERT INTO studentdata1 (id, name, age, grade) VALUES (6, 'sunny', 21, null);
INSERT INTO studentdata1 (id, name, age, grade) VALUES (7, null, 21, 'C');
```



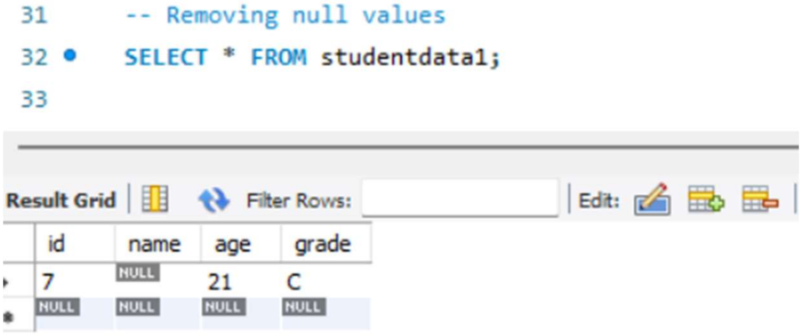
```sql
/* STEP-1 ----> Deleting the duplicate data*/
select name,count(name) as Actual_count from studentdata1
group by name
having count(name)>1;
```

```
name   Actual_count
```



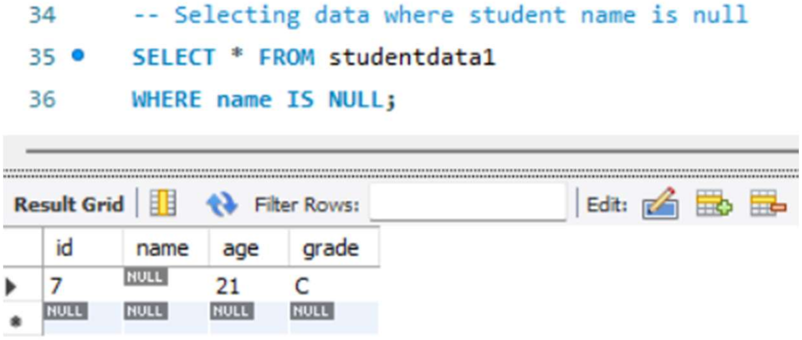
```
with cte as
(
select name,
ROW_NUMBER() over (partition by name order by name desc) as row_no
from studentdata1)

select *from cte;
```

```
  name   row_no
▶ NULL   1
```



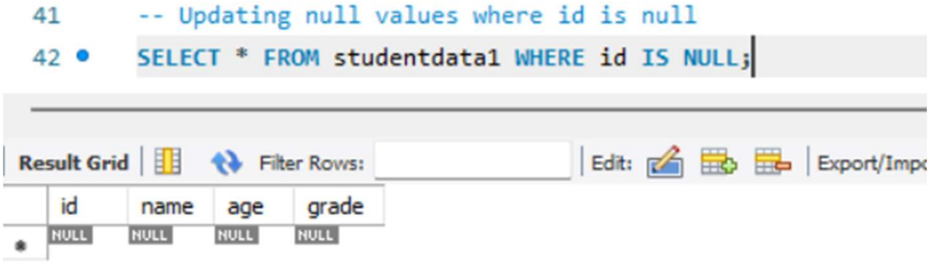
```
-- Removing null values
SELECT * FROM studentdata1;
```

```
  id     name   age    grade
▶ 7      NULL   21     C
* NULL   NULL   NULL   NULL
```



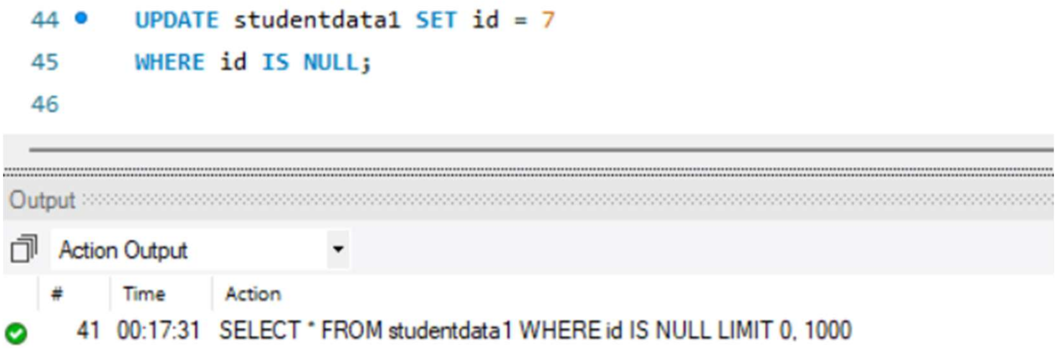
```
-- Selecting data where student name is null
SELECT * FROM studentdata1
WHERE name IS NULL;
```

```
  id     name   age    grade
▶ 7      NULL   21     C
* NULL   NULL   NULL   NULL
```



```
-- Updating null values where id is null
SELECT * FROM studentdata1 WHERE id IS NULL;
```

```
  id     name   age    grade
* NULL   NULL   NULL   NULL
```

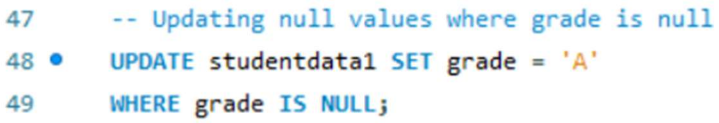


```
UPDATE studentdata1 SET id = 7
WHERE id IS NULL;

```

Output

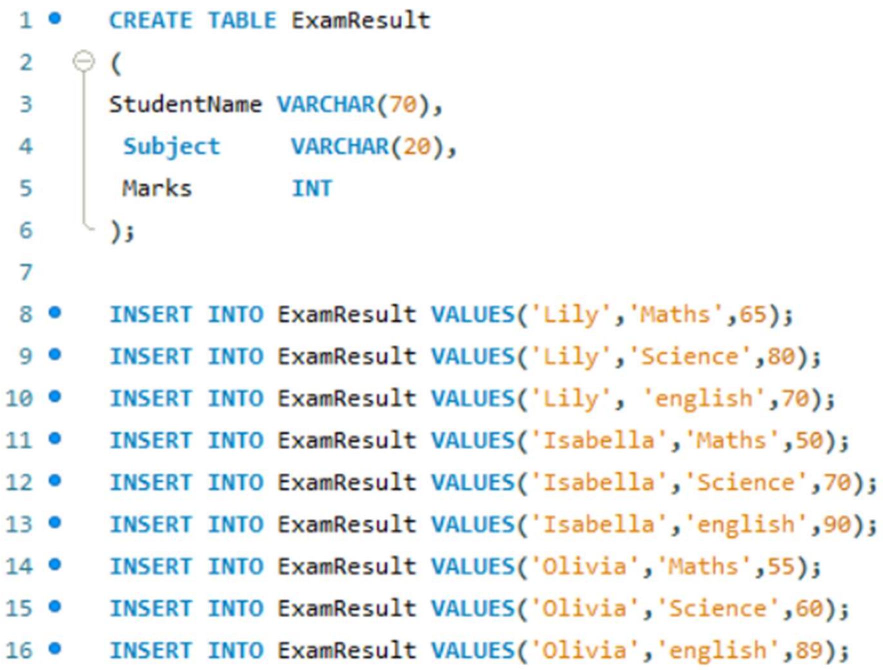
Action Output

```
#    Time      Action
41   00:17:31  SELECT * FROM studentdata1 WHERE id IS NULL LIMIT 0, 1000
```



```
-- Updating null values where grade is null
UPDATE studentdata1 SET grade = 'A'
WHERE grade IS NULL;
```

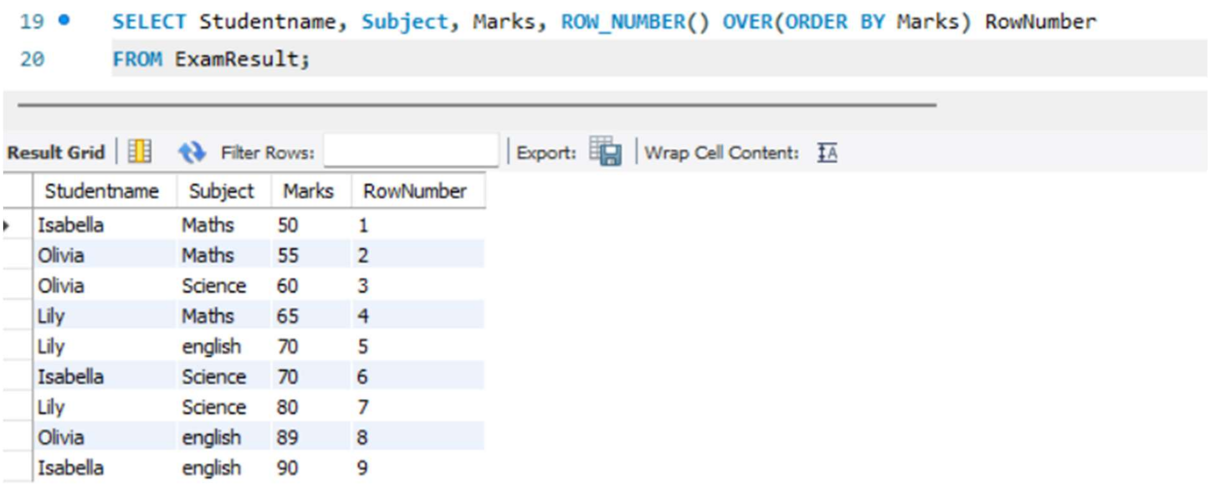
5. Ranking in SQL



```
CREATE TABLE ExamResult
(
StudentName VARCHAR(70),
 Subject     VARCHAR(20),
 Marks       INT
);

INSERT INTO ExamResult VALUES('Lily','Maths',65);
INSERT INTO ExamResult VALUES('Lily','Science',80);
INSERT INTO ExamResult VALUES('Lily', 'english',70);
INSERT INTO ExamResult VALUES('Isabella','Maths',50);
INSERT INTO ExamResult VALUES('Isabella','Science',70);
INSERT INTO ExamResult VALUES('Isabella','english',90);
INSERT INTO ExamResult VALUES('Olivia','Maths',55);
INSERT INTO ExamResult VALUES('Olivia','Science',60);
INSERT INTO ExamResult VALUES('Olivia','english',89);
```

Query 1 - ROW_NUMBER:

- Assigns a unique row number to each row based on the ascending order of the **Marks** column.
- Rows with lower marks will have lower **RowNumber**, and vice versa.

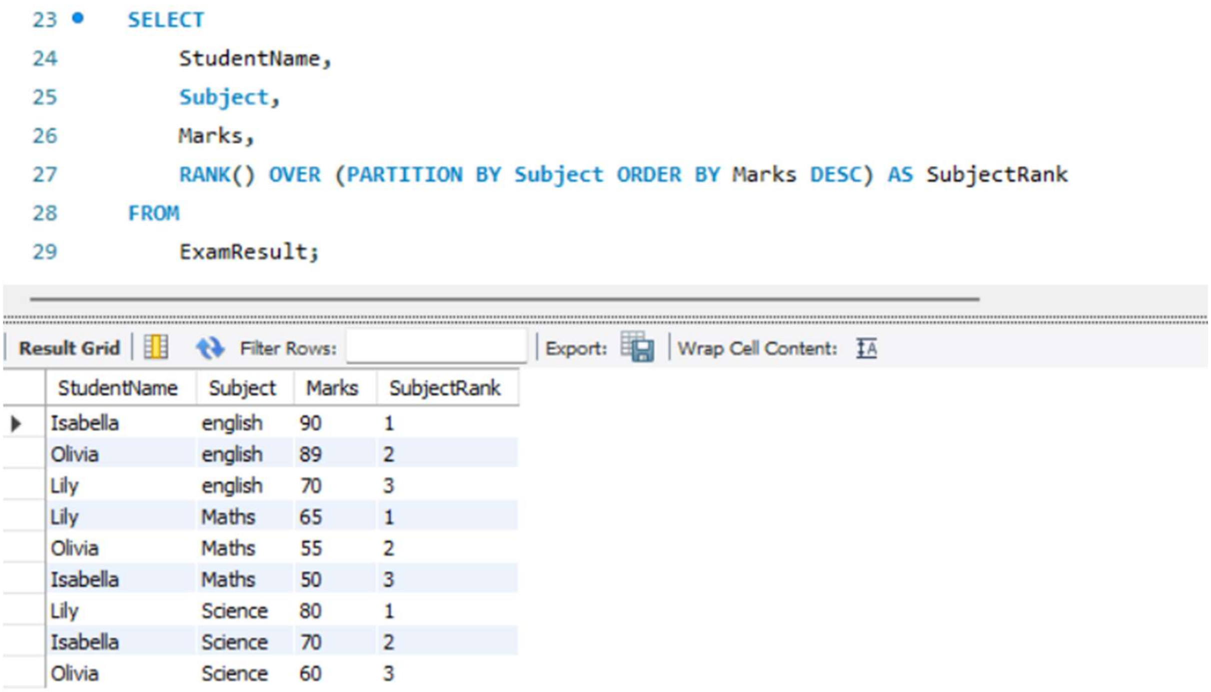


```sql
SELECT Studentname, Subject, Marks, ROW_NUMBER() OVER(ORDER BY Marks) RowNumber
FROM ExamResult;
```

Studentname	Subject	Marks	RowNumber
Isabella	Maths	50	1
Olivia	Maths	55	2
Olivia	Science	60	3
Lily	Maths	65	4
Lily	english	70	5
Isabella	Science	70	6
Lily	Science	80	7
Olivia	english	89	8
Isabella	english	90	9

Query 2 - RANK:

- Assigns a rank to each row within each **Subject** based on the descending order of the **Marks** column.
- The **PARTITION BY** clause ensures that ranking is done separately for each **Subject**.



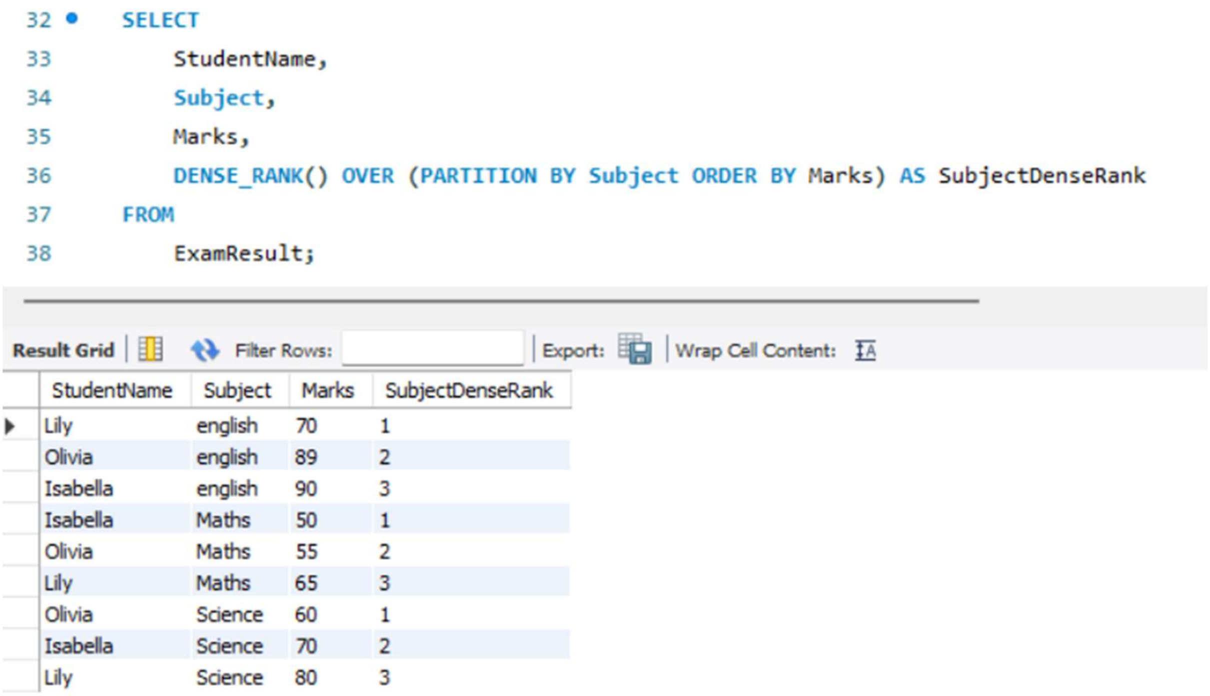
```sql
SELECT
    StudentName,
    Subject,
    Marks,
    RANK() OVER (PARTITION BY Subject ORDER BY Marks DESC) AS SubjectRank
FROM
    ExamResult;
```

StudentName	Subject	Marks	SubjectRank
Isabella	english	90	1
Olivia	english	89	2
Lily	english	70	3
Lily	Maths	65	1
Olivia	Maths	55	2
Isabella	Maths	50	3
Lily	Science	80	1
Isabella	Science	70	2
Olivia	Science	60	3

Query 3 - DENSE_RANK:

- Similar to **RANK**, assigns a rank within each **Subject** based on the ascending order of the **Marks** column.

- However, unlike **RANK**, it does not leave gaps in the ranking when there are tied values.

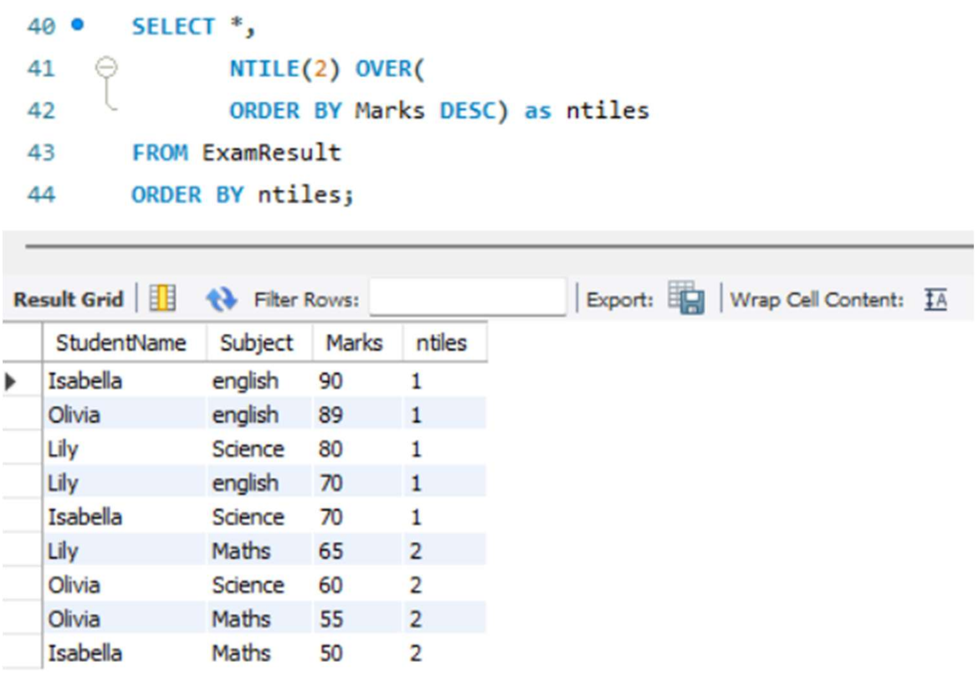


```
SELECT
    StudentName,
    Subject,
    Marks,
    DENSE_RANK() OVER (PARTITION BY Subject ORDER BY Marks) AS SubjectDenseRank
FROM
    ExamResult;
```

StudentName	Subject	Marks	SubjectDenseRank
Lily	english	70	1
Olivia	english	89	2
Isabella	english	90	3
Isabella	Maths	50	1
Olivia	Maths	55	2
Lily	Maths	65	3
Olivia	Science	60	1
Isabella	Science	70	2
Lily	Science	80	3

Query 4 - NTILE:

- **Meaning:** Divides the result set into equal-sized buckets (tiles) based on the descending order of the **Marks** column.
- In this case, it divides the data into two tiles (**NTILE(2)**), assigning each row to a tile based on its **Marks**.

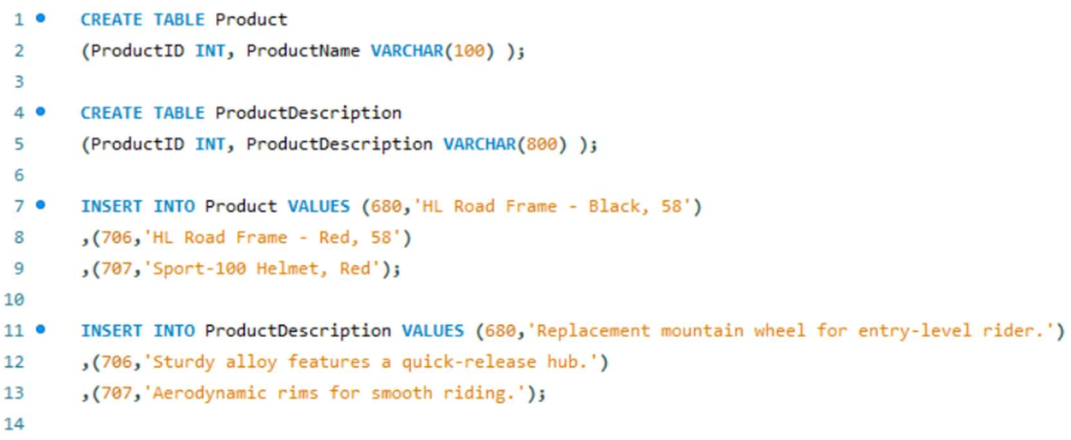


```
SELECT *,
       NTILE(2) OVER(
       ORDER BY Marks DESC) as ntiles
FROM ExamResult
ORDER BY ntiles;
```

StudentName	Subject	Marks	ntiles
Isabella	english	90	1
Olivia	english	89	1
Lily	Science	80	1
Lily	english	70	1
Isabella	Science	70	1
Lily	Maths	65	2
Olivia	Science	60	2
Olivia	Maths	55	2
Isabella	Maths	50	2

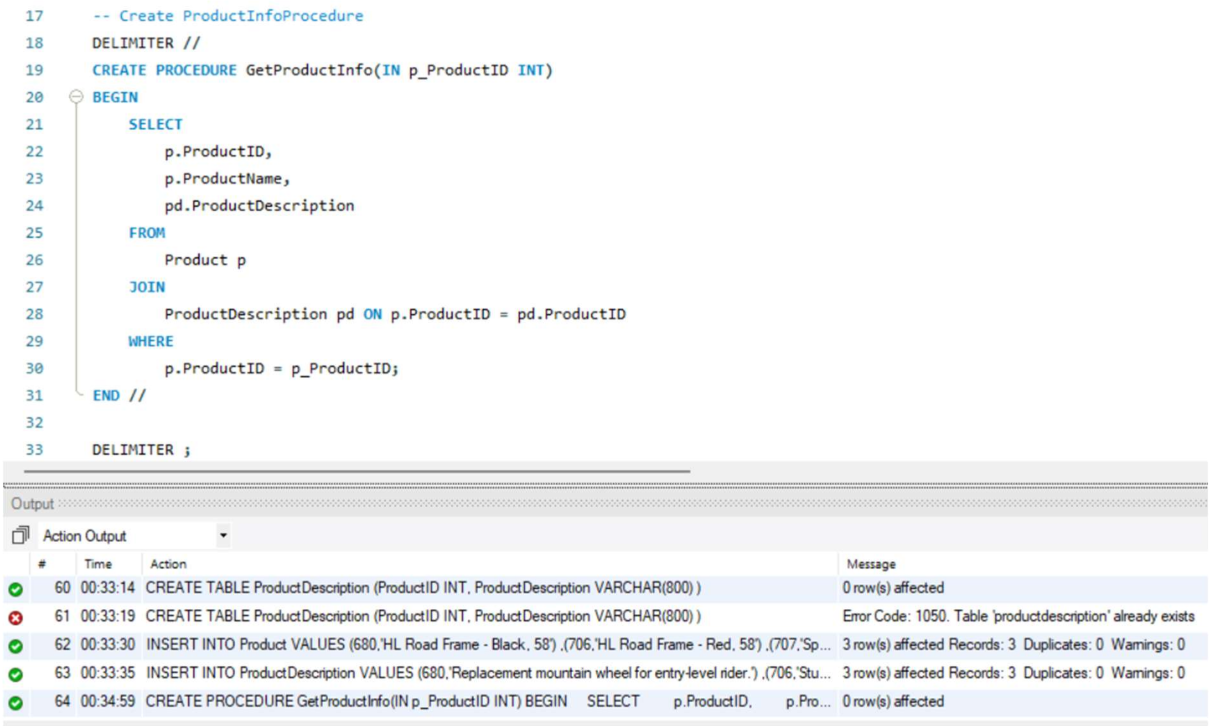
6. Stored procedure

Table Creation and Data Insertion:



```
CREATE TABLE Product
(ProductID INT, ProductName VARCHAR(100) );

CREATE TABLE ProductDescription
(ProductID INT, ProductDescription VARCHAR(800) );

INSERT INTO Product VALUES (680,'HL Road Frame - Black, 58')
,(706,'HL Road Frame - Red, 58')
,(707,'Sport-100 Helmet, Red');

INSERT INTO ProductDescription VALUES (680,'Replacement mountain wheel for entry-level rider.')
,(706,'Sturdy alloy features a quick-release hub.')
,(707,'Aerodynamic rims for smooth riding.');

```

Stored Procedure Creation:



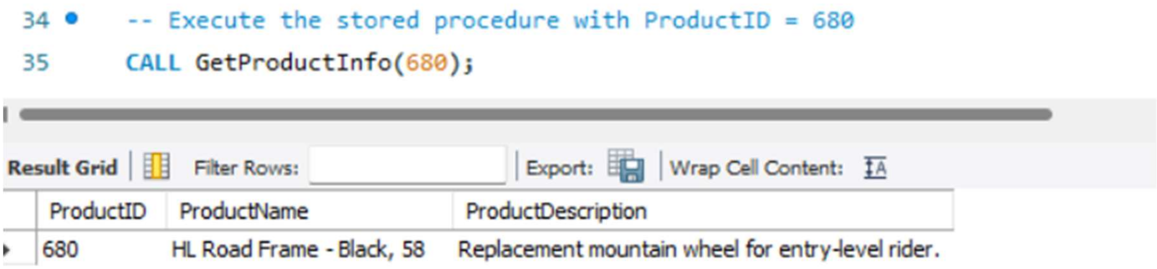
```sql
-- Create ProductInfoProcedure
DELIMITER //
CREATE PROCEDURE GetProductInfo(IN p_ProductID INT)
BEGIN
    SELECT
        p.ProductID,
        p.ProductName,
        pd.ProductDescription
    FROM
        Product p
    JOIN
        ProductDescription pd ON p.ProductID = pd.ProductID
    WHERE
        p.ProductID = p_ProductID;
END //

DELIMITER ;
```

Output

Action Output

	#	Time	Action	Message
✓	60	00:33:14	CREATE TABLE ProductDescription (ProductID INT, ProductDescription VARCHAR(800))	0 row(s) affected
✗	61	00:33:19	CREATE TABLE ProductDescription (ProductID INT, ProductDescription VARCHAR(800))	Error Code: 1050. Table 'productdescription' already exists
✓	62	00:33:30	INSERT INTO Product VALUES (680,'HL Road Frame - Black, 58') ,(706,'HL Road Frame - Red, 58') ,(707,'Sp...	3 row(s) affected Records: 3 Duplicates: 0 Warnings: 0
✓	63	00:33:35	INSERT INTO ProductDescription VALUES (680,'Replacement mountain wheel for entry-level rider.') ,(706,'Stu...	3 row(s) affected Records: 3 Duplicates: 0 Warnings: 0
✓	64	00:34:59	CREATE PROCEDURE GetProductInfo(IN p_ProductID INT) BEGIN SELECT p.ProductID, p.Pro...	0 row(s) affected

Execute the Stored Procedure:



```sql
-- Execute the stored procedure with ProductID = 680
CALL GetProductInfo(680);
```

ProductID	ProductName	ProductDescription
680	HL Road Frame - Black, 58	Replacement mountain wheel for entry-level rider.