Procedures

1.

What is a Stored Procedure?

A stored procedure is a prepared SQL code that you can save, so the code can be reused over and over again.

So if you have an SQL query that you write over and over again, save it as a stored procedure, and then just call it to execute it.

You can also pass parameters to a stored procedure, so that the stored procedure can act based on the parameter value(s) that is passed.

2. Syntax for procedure:

Stored Procedure Syntax

3.

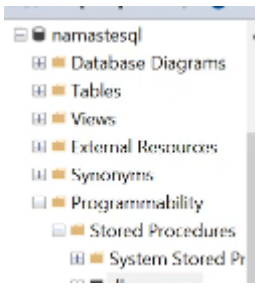
```
CREATE PROCEDURE procedure_name
AS
sql_statement
GO;
```

4. Sample procedure

5.

```
--procedure
create procedure spemp
as
select * from employee
```

6. After creating a procedure..we can check all available procedure in our Database->programmability->Stored Procedures..



Execute a Stored Procedure

7. To run our procedure we use
```
EXEC procedure_name;
```
8. To execute our sample procedure we use **exec spemp**
9. It looks little bit like a **views** but we can do more with procedures
10. Now lets look at stored procedure with parameters
11. As we already have spemp procedure..to add a parameter to it..we have to alter our procedure
12. We took a parameter @salary
```
alter procedure spemp (@salary int)
as
select * from employee where salary > @salary
```
13. Now using this parameter we can get rows where salary > **@salary(our parameter)** using exec spemp and giving our parameter a value
14.
```
alter procedure spemp (@salary int)
as
select * from employee where salary > @salary


exec spemp @salary=100
```
15. If we have two parameters for our procedure…then we have to two params while using exec
16.
```
alter procedure spemp (@salary int , @dept_id int)
as
select * from employee where salary > @salary and dept_id= @dept_id
;

exec spemp @salary=10000 , @dept_id = 100
```
17. We can also give our parameters like this
```
exec spemp 10000 , 100
```
...
18. Now if we use
```
exec spemp 100, 10000
```
..then we get nothing….because it finding rows with salary > 100 and dept_id = 10000…which we don't have in our table
19. It will take values as per the position in the exec command….
20. If we use parameter name then we can give in any order
```
exec spemp  @dept_id = 100 , @salary=10000
```

21. Lets suppose we have another table in our procedure…

22.
```
alter procedure spemp (@salary int , @dept_id int)
as
select * from employee where salary > @salary and dept_id= @dept_id
select * from dept where dep_id= @dept_id;
;

exec spemp  @dept_id = 100 , @salary=10000
```

	emp_id	emp_name	dept_id	salary	manager_id	emp_age	dob
1	2	Mohit	100	15000	5	48	1974-12-02

	dep_id	dep_name
1	100	Analytics

23. In the above query…we have 2 queries in our procedure with parameters of salary and dept_id…and next if we execute our procedure with our 2 params…then our queries will take this 2 params value..and matches with the params in our procedure's query..see pic and understand
24. Another example

25.
```
alter procedure spemp (@salary int , @dept_id int)
as
select * from employee where salary > @salary
select * from dept where dep_id= @dept_id;
;

exec spemp  @dept_id = 100 , @salary=10000
```

Results | Messages

	emp_id	emp_name	dept_id	salary	manager_id	emp_age	dob
1	2	Mohit	100	15000	5	48	1974-12-02
2	5	Mudit	200	12000	6	55	1967-12-02
3	6	Agam	200	12000	2	14	2008-12-02

	dep_id	dep_name
1	100	Analytics

26. Here we have retrieved rows with dept_id = 200 ..using procedures
27. But what if there are no rows with dept_id = 200…if there are no rows ..we have to send a msg
28. In the below query…we used count variable to count the number of rows that has matched with our dept_id…if the count is 0..then it means that…there are no rows that matches with our dept_id parameter…and we print a msg

29.
```
alter procedure spemp (@dept_id int)
as
declare @cnt int

select @cnt = count(1) from employee where dept_id=@dept_id

if @cnt=0
print 'there is no employee in this dept'

;

exec spemp  @dept_id = 900
```

30. We can also use else and print the count of rows ..that matches with our dept_id param

31.
```
alter procedure spemp (@dept_id int)
as
declare @cnt int

select @cnt = count(1) from employee where dept_id=@dept_id

if @cnt=0
print 'there is no employee in this dept'
else
print @cnt
;

exec spemp  @dept_id = 100
```

```
00 %
Messages
4
```

32.
```
alter procedure spemp (@dept_id int)
as
declare @cnt int

select @cnt = count(1) from employee where dept_id=@dept_id

if @cnt=0
print 'there is no employee in this dept'
else
print 'total employees ' + cast(@cnt as varchar(10))
;

exec spemp  @dept_id = 100
```

```
100 %
Messages
total employees 4

Completion time: 2022-12-20T07:33:00.9372243+05:30
```

33. Pivot and unpivot
34. To get sum of sales of category for year 2020 and 2021 ..we use
35.
```
select
category
,sum(case when datepart(year,order_date)=2020 then sales end) as sales_2020
,sum(case when datepart(year,order_date)=2021 then sales end) as sales_2021
from
orders
group by category;
```
36. We can get the same result using pivot
37. Here first we have to retrieve category,year part and sales from orders and name this result as t1
38. Now we have our table..with values of category , datepart and sales
39. Basically pivot convert rows into cols ..using an aggregate function
40.
```
select * from
(select category , datepart(year,order_date) as yod , sales
from orders) t1
pivot (
sum(sales) for yod in ([2020],[2021])
) as t2
```

100 %

Results | Messages

	category	2020	2021
1	Office Supplies	183939.982	246097.175
2	Furniture	198901.436	215387.2692
3	Technology	226364.18	271730.811

41. Here we have 2020 and 2021 in rows actually…using pivot we have calculates sum of sales in 2020 and 2021
42. Pivot example : https://chat.openai.com/share/bb68c283-c36e-49c3-a0ac-2d038a87fd15
43. From our orders table…pivot can be used to find sum of sales…in region , year etc
44. Another example of pivot ..findind sum of sales of categories in region east and west
45.
```
select * from
(select category , region , sales
from orders) t1
pivot (
sum(sales) for region in (West,East)
) as t2
```

100 %

Results | Messages

	category	West	East
1	Office Supplies	220853.249	205516.055
2	Furniture	252612.7435	238291.204
3	Technology	251991.832	264973.981

46. In the same query…suppose if we gave a region ..which is actually not present in rows …then it returns null…in that region column

```
select * from
(select category , region , sales
from orders) t1
pivot (
sum(sales) for region in (West,East,South,eastnorth)
) as t2
```

100 %

Results | Messages

	category	West	East	South	eastnorth
1	Office Supplies	220853.249	205516.055	125651.313	NULL
2	Furniture	252612.7435	208291.204	117298.684	NULL
3	Technology	251991.832	264973.981	148771.908	NULL

47.

48. We can also create table on the results

```
select * into sales_yearwise from
(select category , region , sales
from orders) t1
pivot (
sum(sales) for region in (West,East,South)
) as t2
```

49. here we have created a new table which is sales_yearwise

50. Here we have created a table which consists orders from east region …see pic and

```
select * into orders_east from orders where region='East'
```

100 %

Messages

```
(2848 rows affected)

Completion time: 2022-12-20T08:22:15.7785608+05:30
```

understand

```
select * from orders_east
```

100 %

Results | Messages

	row_id	order_id	order_date	ship_date	ship_mode	customer_id	customer_name	segment	country	city	state
1	1372	CA-2021-136672	2021-03-07 00:00:00.000	2021-03-12 00:00:00.000	Standard Class	MG-17890	Michael Granlund	Home Office	United States	Clinton	Maryland
2	8320	CA-2021-133620	2021-11-13 00:00:00.000	2021-11-18 00:00:00.000	Standard Class	EM-14065	Erin Mull	Consumer	United States	New York City	New York
3	8323	CA-2020-130778	2020-11-19 00:00:00.000	2020-11-25 00:00:00.000	Standard Class	ND-18370	Natalie DeChemey	Consumer	United States	Long Beach	New York
4	8324	CA-2020-130778	2020-11-19 00:00:00.000	2020-11-25 00:00:00.000	Standard Class	ND-18370	Natalie DeChemey	Consumer	United States	Long Beach	New York
5	8329	CA-2021-118577	2021-10-06 00:00:00.000	2021-10-11 00:00:00.000	Standard Class	XP-21865	Xylona Preis	Consumer	United States	Belleville	New Jerse

51.

52. Syntax will be different in most of the db's..

53. `create table orders_east as (select * from orders where region='East')` this is the most commonly used syntax

54. This type of creation table..is very useful for data backup and to experiment on queries

55. To create a duplicate of entire order table …we use

```
select * into orders_back from orders
```

56. And later if something goes wrong in the orders table while querying …then we can truncate our orders table using `truncate table orders` ..it will delete the data
57. And later we can insert data from orders_back table ..using

```
insert into  orders select * from orders_back
```

58. And truncate is faster than delete…fyi
59.