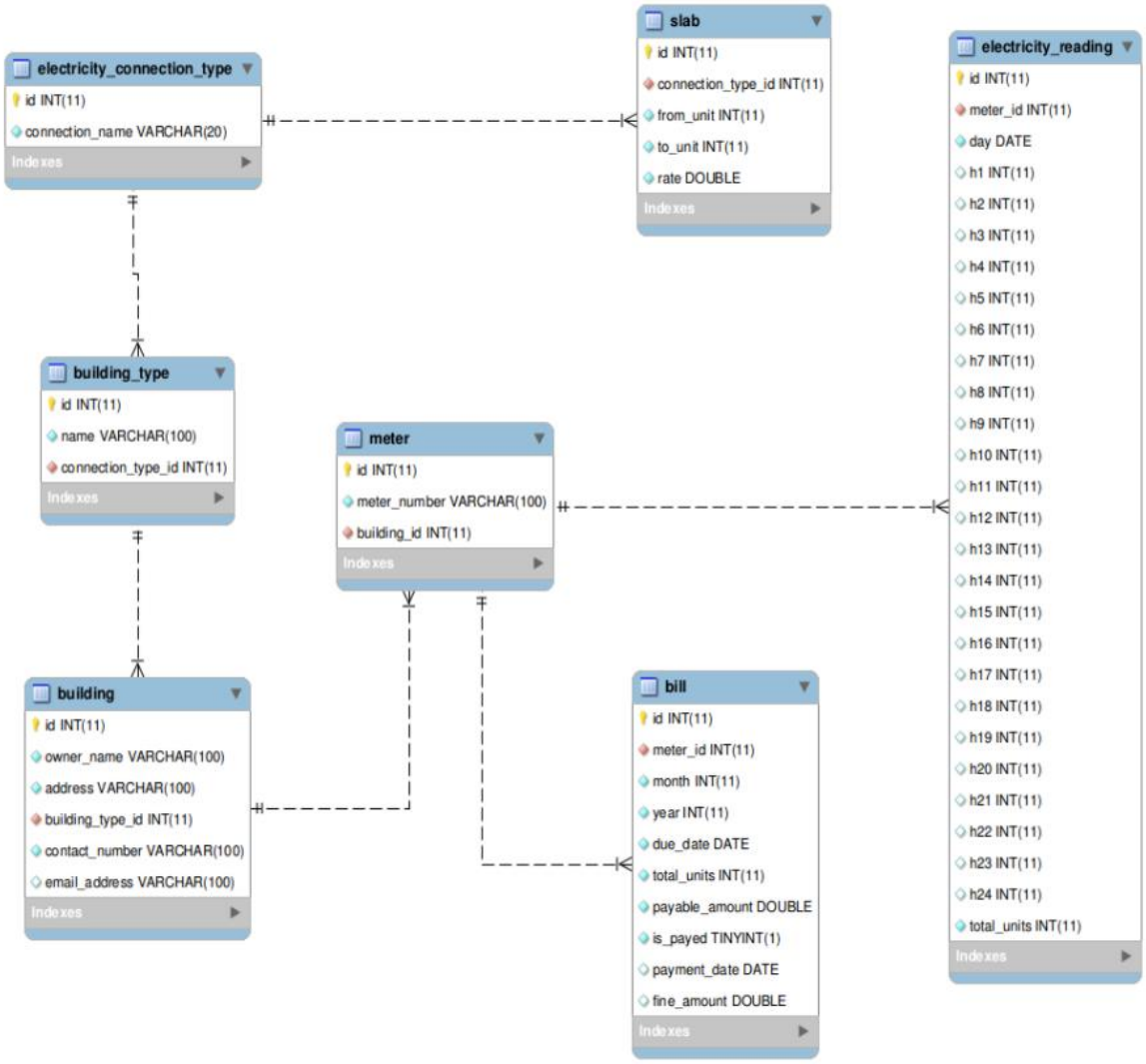


Create a view named **'owner_details'** to display the **owner name** and **contact number** of the buildings having length of the owner name greater than **15**.

CREATE VIEW [owner_details] AS

SELECT owner_name, contact_number

FROM building

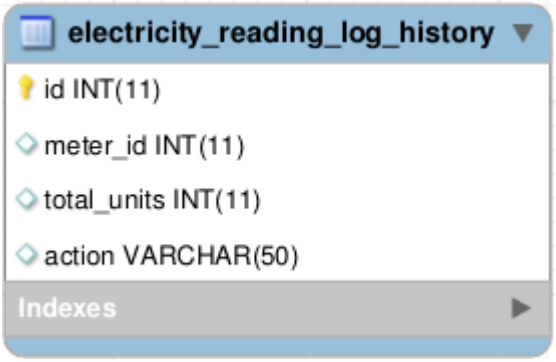
WHERE owner_name > 15;

Create a trigger named '**trigger_electricity_reading_delete**' that is triggered whenever a record in the electricity_reading table is deleted. This trigger will insert the meter_id, total_units and action into the table '**electricity_reading_log_history**' after the deletion of electricity reading details. The action name in the affected log table electricity_reading_log_history is '**After_Delete_Electricity_Reading**'.

Hints:
Trigger name : trigger_electricity_reading_delete
Table name : electricity_reading_log_history
Field names : meter_id, total_units,action

Action : 'After_Delete_Electricity_Reading'.

The table structure of electricity_reading_log_history is as follows:




```
CREATE TRIGGER trigger_electricity_reading_delete

ON electricity_reading

After DELETE

As

BEGIN

INSERT INTO electricity_reading_log_history (meter_id,total_units,action)

SELECT meter_id,total_units,'After_Delete_Electricity_Reading'

FROM DELETED

END
```

Create a procedure named **'insertConnection'** which has **connection_name** as an input parameter with varchar(100) as its datatype. This procedure will take the count of the existing table records(electricity_connection_type) and add 1 with that to generate the new electricity_connection_type id.The newly generated id along with the connection_name should be inserted into the electricity_connection_type table.

Hints:
Procedure name : insertConnection
Parameters : connection_name(varchar(100))

```
CREATE PROCEDURE insertConnection

@connection_name VARCHAR(100)

AS

BEGIN

DECLARE @cnt INT

SELECT @cnt = COUNT (*) FROM electricity_connection_type

INSERT INTO electricity_connection_type VALUES(@cnt+1,@connection_name)

END
```

Create a procedure named **'getBillLevel'** which takes 1 input parameter namely, **bill_id** int and 1 output parameter namely, **level** varchar(50). This procedure should determine the level of the bill as either PLATINUM or GOLD based on the total units consumed for the month. This procedure should set the level as **GOLD** if the total units for the bill is less than 10000 units and the level is **PLATINUM** if the total units is greater than or equal to 10000 units.

Hints:
Procedure name : getBillLevel
Parameters : bill_id(int),level(varchar(50))

```
CREATE PROCEDURE getBillLevel

@bill_id INT,

@level VARCHAR(50) OUT

AS

BEGIN

DECLARE @total_units INT

SELECT @total_units = total_units FROM bill

where id = @bill_id

IF @total_units < 10000

BEGIN

  SET @level = 'GOLD'

END

ELSE

BEGIN

  SET @level = 'PLATINUM'

END

END
```

Create a function named '**showPayedOrNot**' which takes meterNumber as its input parameter and should return the String "Payed" or "Not Payed" from the bill corresponding to the given input.

Hints:
Function name:showPayedOrNot
Input parameter: meterNumber (varchar(255))

Design Rules:
1) If the **meterNumber** passed as input matches with the meter_number in the table and if '**is_payed**' value is '**1**' then it should return the string '**Payed**' .

2) If the **meterNumber** passed as input matches with the meter_number in the table and if '**is_payed**' value is '**0**' then it should return the string '**Not Payed**'.

```
CREATE FUNCTION showPayedOrNot(@meterNumber VARCHAR(255))

RETURNS VARCHAR(255)

AS

BEGIN

DECLARE @pay TINYINT

DECLARE @notpay VARCHAR(20)

SELECT @pay = is_payed from bill where meter_id = (

SELECT id from meter where meter_number = @meterNumber)

IF @pay = 1

BEGIN

SET @notpay = 'Payed'

END

ELSE

BEGIN

SET @notpay = 'Not Payed'

END

RETURN @notpay

END
```