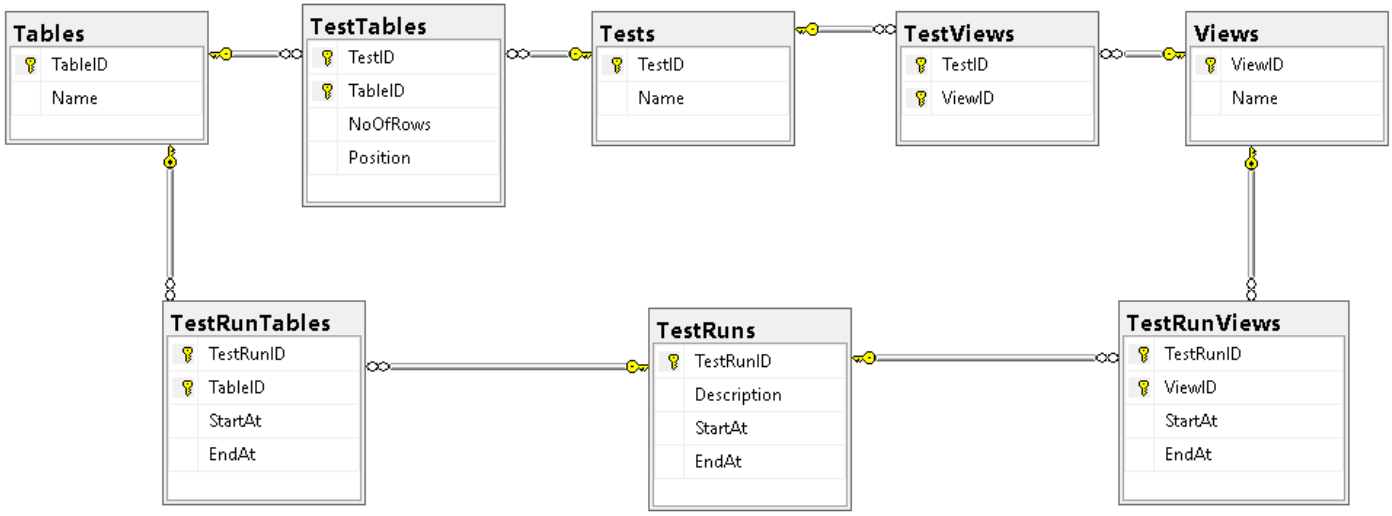
Laboratory 4 - Documentation

Please make a Back-up of your database before start this laboratory!

Structure that must be completed with dates (that will be generated from the script)



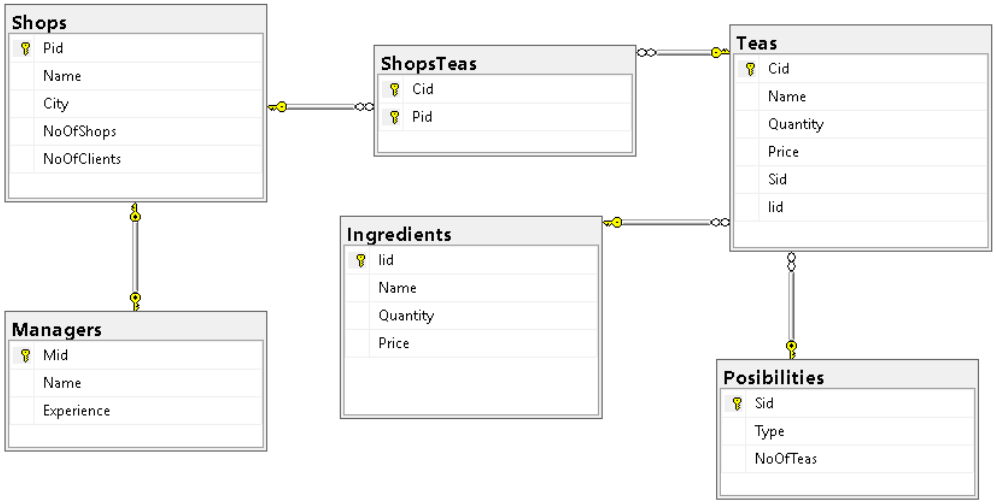
Tables/ views involved

3 tables	***3 views***
1 PK, no FK	1 table
1 PK + FK	2 tables
2 PK	2 tables + GROUP BY

Operations involved - for a lot of dates (can be decided with the help of a parameter or specified directly)

Tables	**Views**
- DELETE - INSERT	- SELECT * FROM View_name

Let's consider the following database



The structure will be completed as:

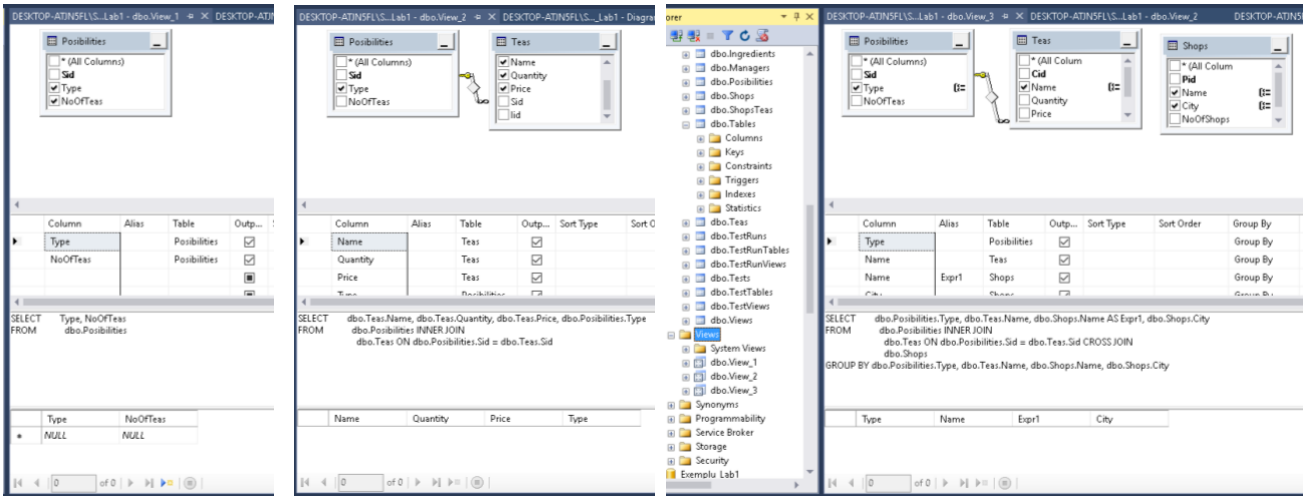
- Table ***Tables*** – with the names of the tables considered (one for each category)

	TableID	Name
	1	Posibilities
	2	Teas
	3	ShopsTeas
▶*	NULL	NULL

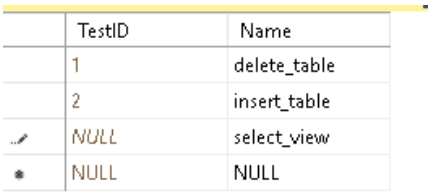
- Table ***Views*** – with the names of the views considered (not necessarily created on the tables considered in the table Tables, but preferable to see the time results)

	ViewID	Name
	1	View_1
	2	View_2
	3	View_3
▶*	NULL	NULL

```
CREATE VIEW View_1 AS
    SELECT Type, Name, Quantity
    FROM Posibilities p INNER JOIN Teas t ON p.Sid=t.Sid
GO
```

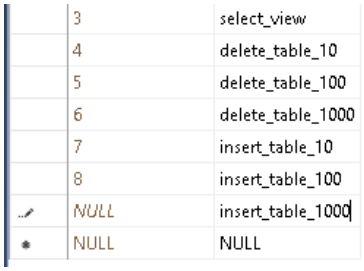


- Table ***Tests*** – will contain at least 3 tests (one referring the delete for the table(s), one referring the insert for the table(s), one referring the select for the view(s)) – can be the name of the procedures or just suggestive names given by you). Or, you can have for each operation (delete, insert, select), also, the number of rows involved in the operations (at insert only).



TestID	Name
1	delete_table
2	insert_table
NULL	select_view
NULL	NULL

or



3	select_view
4	delete_table_10
5	delete_table_100
6	delete_table_1000
7	insert_table_10
8	insert_table_100
NULL	insert_table_1000
NULL	NULL

or …

For example, delete_table_100 refers to the fact that the delete operation will be executed for 100 rows and insert_table_1000 refers to the fact that the insert operation will be executed for 1000 rows.

or

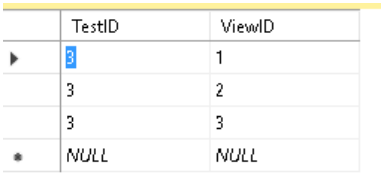
TestId	Name
1	Delete_Insert_Posibilities_View_1
2	Delete_Insert_Teas_View_2
3	Delete_Insert_ShopTeas_View3
4	…

or

TestId	Name
1	DI_Posibilities_View_1
2	DI_Posibilities_View_2
3	DI_Posibilities_View_3
4	…

or … anything else.. It is up to each of you ☺

- Table ***TestViews*** – will contain the combinations of the tests related only with the views (just the evaluate operation)

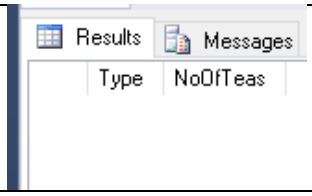


TestID	ViewID
3	1
3	2
3	3
NULL	NULL

where TestID=3 refer the test that has to be done for each view

To run a view you have to use the command SELECT * FROM View_name.

```
-- view
SELECT * FROM View_1
```



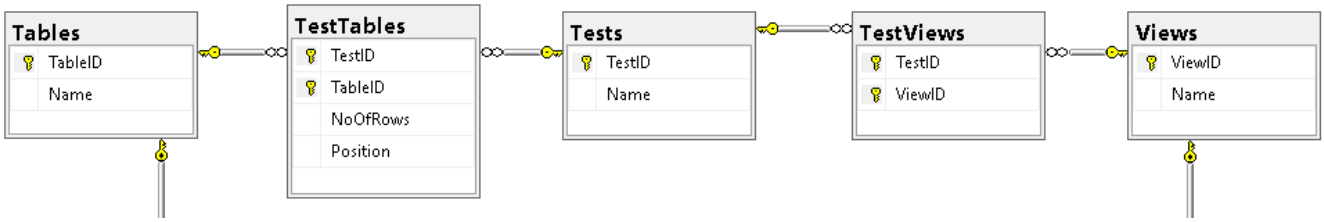
- Table ***TestTables*** – will contain the combinations of the tests related with the tables (only the operations delete and insert). The field **NoOfRows** will contain the number of rows that will be affected by that operation (delete or insert). The field **Position** will refer the order in which the tables will be consider (depending on the operation considered). (For example, we have to insert first in the table in which we have primary key, but not foreign key(s)…)

	TestID	TableID	NoOfRows	Position
	5	1	100	2
	9	2	1000	3
*	NULL	NULL	NULL	NULL

…

You must have the operations tested (delete an insert) for each table and select for the views.

So, first you have to complete the records in the tables from the first line ('manually', analyzing and deciding the tables, the tests, the views, their combinations in testtables, testviews). From there you will extract what you need (table name, NoOfRows, Position, …) and you will operate with them in stored procedures. In the other 3 tables (TestRuns, TestRunTables, TestRunViews) you will insert the values from the code that will be written in stored procedures at each execution and test ("automatically").



Pay attention to the order of the tables!

- When you insert records you must insert first in the tables where you have only primary keys (and not foreign keys), and only after in the tables in which are the foreign keys (that must have the same values as the primary key for the foreign keys). For example, the order in our case is Posibilities, Teas, Shops (we considered that this is the table with the primary key – even if this table is not included in the Tables, but we need these values to insert them in the next table involved), ShopsTeas. You must insert a lot of values (not one) – so that the time of insert operation be relevant for our test. You must insert in all 3 tables considered in Tables.

Example for the table in which we have only primary key (no foreign key(s))

```
USE Example_Lab1
GO

-- when insert on all the fields
Insert Into Posibilities VALUES (1, 'fruit', 3)
-- when insert only on specified fields
-- if the primary key is identity, you CANNO'T insert on that possition
Insert Into Posibilities(Type, NoOfTeas) VALUES ('green', 4)

-- the generalization will be considered here
```

```
DECLARE @NoOFRows int
DECLARE @n int
DECLARE @t VARCHAR(30)

SELECT TOP 1 @NoOfRows = NoOfRows FROM dbo.TestTables WHERE ...
SET @n=1 -- first we have no row inserted

WHILE @n<@NoOfRows
BEGIN
	-- we must set a unique value for the primary key
	-- we should insert a different value for the other fields
	SET @t = 'Posibility' + CONVERT (VARCHAR(5), @n)
	-- so, we will have Posibility1, Posibility2, ...
	INSERT INTO Posibilities (Sid, Type, NoOfTeas) VALUES (@n, @t, 2)
	SET @n=@n+1
END
```

Example for the table in which we have primary key and foreign key(s)

```
-- take the value of the foreign key from the table in which is primary key and insert it on
that column
DECLARE @fk int
DECLARE @fk1 int
SELECT TOP 1 @fk = Sid FROM Posibilities WHERE ...
-- or, SELECT @fk=MAX(Sid) FROM Posibilities where ...
-- or, @fk=(MIN(Sid) FROM Posibilities where ...
-- or, ...
SELECT TOP 1 @fk1 = Iid FROM Ingredients ORDER BY NEWID()...-- to be taken random
WHILE ...
	INSERT INTO Teas(Cid, Name, Quantity, Price, Sid, Iid) VALUES (@n, @t, 3, 2, @fk, @fk1)
...
```

Example for the table in which we have 2 primary keys

```
-- depends on the table considered
-- if are only primary keys - just insert unique values – for example, you can fix a value, and
modify the other at each step (e.g. (1,1), (1,2), (1,3), ... )
/* if the primary keys are also foreign keys, you can use cross join; OR take the values with
while from both of the tables in which are primary keys and combine them; OR work with 2 cursors
*/
-- for example
SELECT Pid, Cid
INTO ShopsTeas
FROM Shops CROSS JOIN Teas

/* pay attention to the NoOfRows - from TestTables - not all the time can be inserted
as many rows as are specified there
– in the intermediate table, with the primary keys that are also foreign keys – please avoid the
errors – by inserting as many as possible, or by stopping when the required number was fulfilled
*/
```

- When you delete rows (not one row, must be many rows - as at inserted), pay attention to the order in which you take the tables. The order is opposite to the insert order. For example, first you delete from where the foreign keys are and only after from the tables where are the primary keys that are also foreign keys. Also, you can use ON CASCADE.

```
-- delete from Posibilities implies the delete from more tables
DELETE FROM ShopsTeas
DELETE FROM Teas
DELETE FROM Posibilities
```

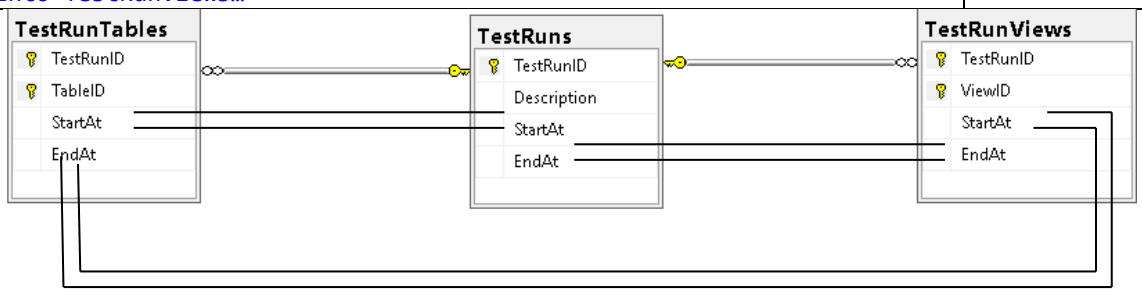
```
-- delete in which you want to keep the records you have will contain a WHERE clause
DELETE FROM table_name WHERE...
```

To get the Date, you can use GETDATE(). You must insert in tables TestRuns, TestRunTables and TestRunViews with code.

- Table ***TestRuns*** – will contain the tests involved. One test must have DELETE + INSERT operations on the table(s) AND SELECT operation on the view(s). The field **Description** will contain the description of the test, **StartAt** – the date when the test start (before the Delete from the table(s) + Insert), **EndAt** – the date when the test is done (after the Select of the view(s)).
- Table ***TestRunTables*** – will contain the tests involved for the tables (operations of delete and insert). One test must have DELETE + INSERT operations on the table(s). The field **StartAt** will contain the date when the test start (before the Delete from the table(s)) and **EndAt** – the date when the test is done (after the Insert for the table(s)).
- Table ***TestRunViews*** – will contain the tests involved for views (the select operation). One test must have SELECT operation on the view(s). The field **StartAt** will contain the date when the test start (before the Select for the view(s)) and **EndAt** – the date when the test is done (after the Select of the view(s)).

```
DECLARE @ds DATETIME()-- start time test
DECLARE @di DATETIME()-- intermediate time test
DECLARE @de DATETIME()-- end time test

SET @ds = GETDATE()
-- delete from table
-- insert into table
SET @di=GETDATE()

-- evaluate (select from) view
SET @de=GETDATE()

-- if you want to see the difference of these 2 times, you can use DATEDIFF
Print DATEDIFF(@de, @ds)

Insert into TestRuns …
-- extract the TestRunId and "combine" it with the Id of table involved and
also with the view involved in the corresponding tables

Insert into TestRunTables…
Insert into TestRunViews…
```

A TEST means:
- Delete from table
- Insert into table
- Evaluate a view



The way that you implement the code is up to you. Please, make it short and clear.