Lab 9 - PART 3 SOLUTIONS

1- BATCH, SCRIPT, AND TRANSACTIONS

Batch	Script	Transaction
A batch is a group of SQL statements that are compiled and executed together by the SQL server. The server processes the statements sequentially. If one statement encounters an error that does not affect the others, the batch will still execute successfully. **Explanation**: If an error occurs in a particular statement, the execution of subsequent statements is unaffected, and the batch is processed. The specific statement that caused the error won't be executed.	A script is a file or collection of SQL statements and batches. It is used to automate tasks and can contain multiple batches, separated by the "GO" statement. **Explanation**: A script is used when you need to perform a series of tasks that may not be related to each other. These tasks are grouped into batches and statements, with each batch separated by the "GO" command.	A transaction is a sequence of one or more SQL statements that are executed as a single unit of work. Transactions ensure data integrity and consistency by adhering to the **ACID** properties (Atomicity, Consistency, Isolation, Durability). **Explanation**: A transaction ensures that all SQL statements within it are treated as a single operation. If any part of the transaction fails, none of the changes are committed, and the database remains unchanged. For example, if a transfer of money fails during a banking operation, the transaction ensures that no money is deducted from the sender or added to the recipient's account unless the entire operation succeeds. **ACID Properties**: • **Atomicity**: Ensures that all operations in a transaction are completed successfully or none at all. • **Consistency**: Guarantees that a transaction will bring the database from one valid state to another. • **Isolation**: Ensures that the operations of one transaction are isolated from others, preventing interference. • **Durability**: Once a transaction is committed, the changes are permanent, even in the case of a system crash.

Batch Example:

```
SELECT * FROM Employee
UPDATE Employees
SET Salary = Salary * 1.1
WHERE Department = 'Sales'
SELECT *
FROM Employees
WHERE Department = 'Sales'
```

Script Example:

```
-- Script Example
-- Batch 1
SELECT * FROM Customers


-- Batch 2
UPDATE Orders SET Status =
'Shipped' WHERE OrderDate <
'2025-01-01'


-- Batch 3
SELECT * FROM Orders WHERE
Status = 'Shipped'
```

Transaction Example

```
BEGIN TRANSACTION

UPDATE Accounts SET Balance = Balance - 500 WHERE AccountID = 1
UPDATE Accounts SET Balance = Balance + 500 WHERE AccountID = 2

-- Check if any error occurs, and commit or roll back the
transaction
IF @@ERROR <> 0
BEGIN
    ROLLBACK TRANSACTION
    PRINT 'Transaction failed, rolled back'
END
ELSE
BEGIN
    COMMIT TRANSACTION
    PRINT 'Transaction successful, committed'
END
```

2- Triggers and Stored Procedure:

Triggers	Stored Procedure
A *trigger* is a special type of stored procedure that automatically runs when a specific event (such as INSERT, UPDATE, or DELETE) occurs on a specific table or view. **Key Points:** • Triggers are *event-driven* – they execute in response to changes in data. • You **cannot call a trigger manually** – it is fired automatically. • Mainly used for **auditing** and **enforcing business rules**. • Can access **inserted** and **deleted** pseudo-tables to track data changes.	A *stored procedure* is a precompiled group of one or more SQL statements that you can execute manually to perform a specific task. **Key Points:** • You **call it manually** using EXEC or CALL. • Useful for **modularizing** logic and **reusing** code. • Helps improve **performance** and **security** (can hide logic from users). • Stored in the database's **query tree** – optimized and ready to execute.

Examples on Triggers:

```
CREATE TRIGGER trg_AuditSalaryChange
ON Employees
FOR UPDATE
AS
BEGIN
    INSERT INTO SalaryAudit(EmployeeID, OldSalary,
NewSalary, ChangeDate)
    SELECT
        d.EmployeeID,
        d.Salary,
        i.Salary,
        GETDATE()
    FROM deleted d
    JOIN inserted i ON d.EmployeeID = i.EmployeeID
    WHERE d.Salary <> i.Salary
END
```

Stored Procedure Example:

```
CREATE PROCEDURE GetEmployeeSalary
    @EmployeeID INT
AS
BEGIN
    SELECT Salary FROM Employees WHERE EmployeeID = @EmployeeID
END

-- To call it:
EXEC GetEmployeeSalary @EmployeeID = 101
```

3- Stored Procedure and Functions

Stored Procedure	Functions
A precompiled collection of one or more SQL statements that can be executed as a single unit and it's deigned to perform a specific tasks such as querying data, modifying data	A reusable SQL code that performs a specific task and returns a single value or a table primarily functions are used for a computations, data transformations and returning specific values
Can accept input and output parameters	Can accept parameters
Can perform actions that affect the database state, such as inserting, updating, or deleting rows **Can modify database state**	Functions are deterministic and cannot modify the database state. They cannot perform actions like inserting, updating, or deleting rows **Cannot modify database state**
	Types of Functions Scalar, Inline, Multivalued

4- drop, truncate and delete statement:

Drop	Truncate	Delete
Delete the entire table from the hard disk	delete data from the table keeps the structure of the table	delete data from the table keeps the structure of the table
Doesn't use where condition	delete data unconditionally (doesn't have WHERE clause) cannot be rolled back because it's DDL statement	can use WHERE clause, can be rolled back because it's DML statement, keeps the physical memory assigned to the data until a roll back or commit is issued
DDL command	DDL command	DML command

5- select and select into statement:

SELECT	SELECT INTO
retrieve data from existing tables or view (db objects)	Create a new table and insert query results into it

```
SELECT *
FROM student -----> /*retrieve all data from student table*/
```

```
SELECT INTO


SELECT id , name INTO new_table
FROM student
WHERE name LIKE 'a%'-----> /*create a new_table and insert into it id,
name columns from student table where the name begins with an 'a' letter
*/
```

6- local and global variables:

Local Variable	Global Variable
Can be declared	can't be declared
Can assign values to it	can't assign value to it
User defined variable	Built in variable
used in carrying a value inside it	used in Display only

7- convert and cast statements:

Cast	Convert
Used for straightforward data type conversions and is SQL standard, making it portable across different database systems. It is simpler but lacks advanced formatting options	Provides additional formatting capabilities, particularly for date and time conversions, and is specific to SQL Server. It offers more control over the output format but is less portable

```
-- Example 1: Converting an integer to a string
SELECT CAST(150 AS VARCHAR(10)) AS StringValue;

-- Example 2: Converting a string to a decimal
SELECT CAST('123.45' AS DECIMAL(5,2)) AS DecimalValue;

-- Example 3: Converting a datetime to a date
SELECT CAST(GETDATE() AS DATE) AS DateOnly;
```

```
-- Example 1: Converting an integer to a string
SELECT CONVERT(VARCHAR(10), 150) AS StringValue;

-- Example 2: Converting a string to a decimal
SELECT CONVERT(DECIMAL(5,2), '123.45') AS DecimalValue;

-- Example 3: Converting a string to a date with a specific format
SELECT CONVERT(DATE, '10/06/2024', 103) AS DateValue; -- 103 is the style
code for 'dd/mm/yyyy'

-- Example 4: Converting a datetime to a string with a specific format
SELECT CONVERT(VARCHAR(20), GETDATE(), 100) AS FormattedDateTime; -- 100
is the style code for 'mon dd yyyy hh:miAM'
```

8- DDL, DML, DCL, DQL and TCL:

DDL	DML	DCL	DQL	TCL
refers to Data definition language its commands are responsible for creating the structure of the DB	refers to Data Manipulation Language its commands are responsible for manipulating database	refers to Data Control language its commands are responsible for system controlling and giving privileges to users	Refers to Data Query Language and are responsible for retrieving the data	Refers to Transaction Control language used to manage transactions within the database

DDL	DML	DCL	DQL	TCL
CREATE ,ALTER DROP, AND TRUNCATE) commands.	INSERT, UPDATE, DELETE)	(GRANT, REVOKE) commands	SELECT statement	BEGIN TRANSACTION, COMMIT,ROLLBACK

9- For xml raw and for xml auto

XML RAW	XML Auto
Transforms each row in the result set into an XML element	Returns query results in a simple, nested XML tree. Each table in the FROM clause for which at least one column is listed in the SELECT clause is represented as an XML element. The columns listed in the SELECT clause are mapped to the appropriate element attributes.

10- Table valued and multi statement function:

Inline Function	Multi Statement Function
Return table Body has Select statement	Return a new table as a result of insert statement Body can have Select + variables and IF ,WHILE statements

Inline Function Example

```
create function highage()
returns table
as
return
(
select st_fname,st_age from student where st_age>=20
)

select * from dbo.highage()
```

Multivalued Function Example

```
create function student_names(@format nvarchar(50))
returns @t table
        (
         student_id int primary key,
         student_name nvarchar(50)
        )
as
begin
    if @format='fullname'
        insert into @t
        select st_id,st_fname+' '+st_lname
        from student
    else
    if @format='firstname'
        insert  into @t
        select st_id,st_fname
        from student
return
end

select * from student_names('fullname')
```

11- Varchar(50) and varchar(max):

Varchar(50)	Varchar(Max)
it allows the length of the variable to be 50 characters max	it determines the length of the string that applied on a column based on the maximum length of value that column has

12- Datetime, datetime2(7) and datetimeoffset(7)

Datetime	Datetime2(7)	datetimeoffset(7)
stores date and time data with a fixed fractional precision.	an extension of `datetime` with a larger date range and higher precision for the time component.	includes all the functionality of `datetime2` but with an additional time zone offset component.
Date Range: January 1, 1753, to December 31, 9999.	**Date Range**: January 1, 0001, to December 31, 9999.	**Date Range**: January 1, 0001, to December 31, 9999

Datetime	Datetime2(7)	datetimeoffset(7)
Storage Size: 8 bytes.	**Storage Size**: Varies between 6 and 8 bytes depending on the precision.	**Storage Size**: 10 bytes.
Precision: 3 or above (.000)	**Precision**: Up to 7 decimal places	**Precision**: Up to 7 decimal places for seconds.

13- Default instance and named instance

Default instance	Named Instance
The SQL server name by default takes the computer name	Another SQL server name with a specific name and you can have multiple named instance on the same machine

14- SQL and windows Authentication

SQL Authentication	Windows Authentication
Managed within SQL Server	Managed by Windows/Active Directory
Requires explicit management within SQL Server	Uses Windows policies and enforcement
Requires separate security measures	Uses Windows security features

15- Clustered and non-clustered index

Clustered index	Non-Clustered Index
Save the data based on its primary key to retrieve it faster	Save the data based on a column that we retrieve frequently
Defines the order of data rows	Does not define data order
Automatically created if primary key constraint is defined	Separate from primary key constraint

Clustered index	Non-Clustered Index
Only one per table	Multiple per table

16- Group by rollup and group by cube

Group by rollup	Group by cube
generates subtotals for the specified columns in the `GROUP BY` clause, from right to left, with the last column being the grand total.	generates subtotals for all possible combinations of the specified columns in the `GROUP BY` clause, including no grouping (i.e., the grand total
It generates all possible grouping sets in a hierarchical order	It generates a result set that represents a multi-dimensional cube

Example on GROUP BY ROLLUP

```
SELECT region, product, SUM(amount) AS total
FROM sales
GROUP BY ROLLUP (region, product)
```

region	product	total
East	A	300
East	B	350
North	A	100
North	B	150
South	A	200
South	B	250
West	A	400
West	B	450
East	NULL	650
North	NULL	250
South	NULL	450
West	NULL	850
NULL	NULL	2200

Example on GROUP BY CUBE

```
SELECT region, product, SUM(amount) AS total
FROM sales
GROUP BY CUBE (region, product);
```

region	product	total
East	A	300
East	B	350
East	NULL	650
North	A	100
North	B	150
North	NULL	250
South	A	200
South	B	250
South	NULL	450
West	A	400
West	B	450
West	NULL	850
NULL	NULL	2200

17- Sequence object and identity

Sequence object	identity
database object that generates a sequence of numeric values according to the specified properties.	An identity column is a column in a table that automatically generates unique values when a new row is inserted. The identity property is tied directly to the table.

18- Inline function and view

Inline function	view
return table as a result of select statement	A virtual table that specify user view of a data
can have parameters	Can't have parameters
can't include `INSERT`, `UPDATE`, `DELETE` directly	Has no DML queries inside its body

19- Table variable and temporary table:

Table variable	Temp Table
a variable that can store a table value	local table that's created within the session

Example on Table variable

```
DECLARE @t Table
( id int,
name varchar(50)
)
INSERT INTO @t Values(1,"ahmed"),(2,"mohamed")
```

Example on local Table

```
CREATE TABLE #TEMP
( id int,
name varchar(50)
)
INSERT INTO #TEMP Values(1,"ahmed"),(2,"mohamed")
```

20- Row_number() and dense_Rank() function

ROW_NUMBER()	RANK()
order the data based on a specific column	Ordering the data based on a specific column but considering the row number with it

Example on ROW_NUMBER()

```
SELECT*,
ROW_NUMBER()OVER(ORDER BY esal DESC)AS RN
FROM employee
```

eid	ename	esal	did	RN	DR
15	ahmed	10000	10	1	1
14	ali	10000	10	2	1
12	eman	9000	10	3	2
1	nada	9000	10	4	2
2	reem	9000	10	5	2
3	khalid	8000	10	6	3
7	mohamed	7000	20	7	4
8	sayed	7000	20	8	4
6	hassan	6000	20	9	5
5	omar	6000	20	10	5
9	sally	5000	30	11	6
10	shimaa	4000	30	12	7
11	hana	4000	30	13	7
12	lama	3000	30	14	8

Example on RANK ()

```
SELECT
    student_name,
    score,
    RANK() OVER (ORDER BY score DESC) AS rank
FROM
    students;
```

student_name	score	rank
Alice	90	1
Charlie	90	1
Bob	85	3
Emma	85	3
David	75	5