**Day 10**

**Date: - 19-08-2024**

# Advanced SQL Concepts

## Advanced data retrieval:

## Functions:

SQL Functions are built-in programs that are used to perform different operations on the database.

A function takes parameters, performs actions, and returns the outcome. It should be noted that functions always return either a single value or a table.

There are two types of functions in SQL:

- Aggregate Functions
- Scalar Functions

### Aggregate Functions:

Aggregate functions operate on a collection of values and return a single value.

SQL Aggregate Functions operate on a data group and return a singular output. They are mostly used with the GROUP BY clause to summarize data.

**System Aggregate Function**

| Aggregate Function | Description | Syntax |
|---|---|---|
| **AVG()** | Calculates the average value | SELECT AVG(column_name) FROM table_name; |
| **COUNT()** | Counts the number of rows | SELECT COUNT(column_name) FROM table_name |
| **FIRST()** | Returns the first value in an ordered set of values | SELECT FIRST(column_name) FROM table_name; |

| Aggregate Function | Description | Syntax |
|---|---|---|
| **LAST()** | Returns the last value in an ordered set of values | SELECT LAST(column_name) FROM table_name; |
| **MAX()** | Retrieves the maximum value from a column | SELECT MAX(column_name) FROM table_name; |
| **MIN()** | Retrieves the minimum value from a column | SELECT MIN(column_name) FROM table_name; |
| **SUM()** | Calculates the total sum of values in a numeric column | SELECT SUM(column_name) FROM table_name; |

## Scalar functions:

**SQL Scalar Functions** are built-in functions that operate on a single value and return a single value.

Scalar functions in SQL helps in efficient data manipulation and simplification of complex calculations in SQL queries.

**Scalar Function**

**Description**

Abs (-10.67)→ This returns an absolute number of the given number, which means 10.67.

Rand (10) → This will generate a random number of 10 characters.

round(17.56719,3)→ This will round off the given number to 3 places of decimal meaning 17.567

upper('dotnet')→ This will return the upper case of the given string meaning 'DOTNET'

lower('DOTNET')→ This will return the lowercase of the given string means 'dotnet'

ltrim(' dotnet')→ This will remove the spaces from the left-hand side of the 'dotnet' string.

convert(int, 15.56)→ This will convert the given float value to integer means 15.

## Test-1:

**Create a database function and create a table called products and create a function and call it by using function name and see the results.**

```
create database Functions;
GO
create table Products(
ProductID int primary key not null,
ProductName varchar(50),
Price decimal,
Quantity int
);
insert into products(ProductID, ProductName, Price, Quantity)
Values(1,'chai',30,5),
(2,'Biscuit',50,200),
(3,'Rust',10,140),
(4,'Sugar',24.50,30),
(5,'Coffee',78.89,10);

Select * from products;
CREATE FUNCTION CalculateTotal
(@Price decimal(10,2), @Quantity int)
returns decimal(10,2)
AS
Begin
    Return @Price * @Quantity
END

Select ProductName, Quantity, Price,
dbo.CalculateTotal(price, Quantity)
AS Total
From Products;
```

| | ProductName | Quantity | Price | Total |
|---|---|---|---|---|
| 1 | chai | 5 | 30 | 150.00 |
| 2 | Biscuit | 200 | 50 | 10000.00 |
| 3 | Rust | 140 | 10 | 1400.00 |
| 4 | Sugar | 30 | 25 | 750.00 |
| 5 | Coffee | 10 | 79 | 790.00 |

## Test-2:

**Create a table called Marks and perform some Aggregate functions(sum(), Min(), Max(), Avg()).**

| | Student_id | name | branch | marks |
|---|---|---|---|---|
| 1 | 1001 | NITISH | EEE | 82 |
| 2 | 1002 | RISHABH | EEE | 91 |
| 3 | 1003 | ANIKANT | EEE | 89 |
| 4 | 1004 | RUPESH | EEE | 78 |
| 5 | 1005 | SHUBHAM | CSE | 77 |
| 6 | 1006 | VEDHAL | CSE | 43 |
| 7 | 1007 | DEEPAK | ECE | 95 |
| 8 | 1008 | ARPAN | ECE | 87 |
| 9 | 1009 | VINAY | ECE | 76 |
| 10 | 1010 | ROHITH | MECH | 67 |
| 11 | 1011 | PRASHANTH | MECH | 69 |
| 12 | 1012 | AMITH | MECH | 89 |
| 13 | 1013 | SUNNY | MECH | 78 |

**Fig-1:Marks table**

| | Student_id | name | branch | marks | Average Marks | Minimum Marks | Maxmimum Marks | SUM OF MARKS |
|---|---|---|---|---|---|---|---|---|
| 1 | 1005 | SHUBHAM | CSE | 77 | 60 | 43 | 95 | 120 |
| 2 | 1006 | VEDHAL | CSE | 43 | 60 | 43 | 95 | 120 |
| 3 | 1007 | DEEPAK | ECE | 95 | 86 | 76 | 95 | 258 |
| 4 | 1008 | ARPAN | ECE | 87 | 86 | 76 | 95 | 258 |
| 5 | 1009 | VINAY | ECE | 76 | 86 | 76 | 95 | 258 |
| 6 | 1001 | NITISH | EEE | 82 | 85 | 78 | 95 | 340 |
| 7 | 1002 | RISHABH | EEE | 91 | 85 | 78 | 95 | 340 |
| 8 | 1003 | ANIKANT | EEE | 89 | 85 | 78 | 95 | 340 |
| 9 | 1004 | RUPESH | EEE | 78 | 85 | 78 | 95 | 340 |
| 10 | 1010 | ROHITH | MECH | 67 | 75 | 67 | 95 | 303 |
| 11 | 1011 | PRASHA... | MECH | 69 | 75 | 67 | 95 | 303 |
| 12 | 1012 | AMITH | MECH | 89 | 75 | 67 | 95 | 303 |
| 13 | 1013 | SUNNY | MECH | 78 | 75 | 67 | 95 | 303 |

**Fig: Output**

```sql
create table Marks(
Student_id int primary key not null,
name varchar (50),
branch varchar(100),
marks int);

insert into Marks(Student_id, name, branch, marks)
values(1001, 'NITISH', 'EEE',82),
(1002, 'RISHABH','EEE',91),
(1003, 'ANIKANT', 'EEE', 89),
(1004, 'RUPESH', 'EEE', 78),
(1005, 'SHUBHAM', 'CSE', 77),
(1006, 'VEDHAL','CSE',43),
(1007, 'DEEPAK','ECE',95),
(1008,'ARPAN','ECE',87),
(1009,'VINAY','ECE',76),
(1010, 'ROHITH','MECH',67),
(1011,'PRASHANTH','MECH',69),
(1012,'AMITH','MECH',89),
(1013,'SUNNY','MECH',78);

SELECT * FROM MARKS;
```

```sql
SELECT * FROM MARKS;

SELECT branch, AVG(marks) AS "AVERAGE OF MARKS BRANCH WISE"
FROM MARKS
GROUP BY BRANCH;

SELECT *, MIN(MARKS) OVER(PARTITION BY BRANCH) AS "MIN OF MARKS"
FROM MARKS;

select *, avg (marks) over(partition by branch) as "Average Marks",
min(marks) over (partition by branch) as "Minimum Marks",
max(marks) over() AS "Maxmimum Marks",
SUM(MARKS) OVER (PARTITION BY BRANCH) AS "SUM OF MARKS"
from marks ORDER BY MARKS;

 ------print the details who are getting more than avg marks
SELECT name ,AVG(marks) over (partition by branch ) as "branch_avg"
from marks;

SELECT *
FROM (SELECT * ,AVG(marks) over (partition by branch ) as "branch_avg"
        from marks) t
where t.marks>t.branch_avg;
```

| | Student_id | name | branch | marks | branch_avg |
|---|---|---|---|---|---|
| 1 | 1005 | SHUBHAM | CSE | 77 | 60 |
| 2 | 1007 | DEEPAK | ECE | 95 | 86 |
| 3 | 1008 | ARPAN | ECE | 87 | 86 |
| 4 | 1002 | RISHABH | EEE | 91 | 85 |
| 5 | 1003 | ANIKANT | EEE | 89 | 85 |
| 6 | 1012 | AMITH | MECH | 89 | 75 |
| 7 | 1013 | SUNNY | MECH | 78 | 75 |

```
select *, DENSE_RANK() OVER(PARTITION BY branch order by marks desc) AS "BRANCH_WISE_RANKING"
FROM MARKS;
```

| | Student_id | name | branch | marks | BRANCH_WISE_RANKING |
|---|---|---|---|---|---|
| 1 | 1005 | SHUBHAM | CSE | 77 | 1 |
| 2 | 1006 | VEDHAL | CSE | 43 | 2 |
| 3 | 1007 | DEEPAK | ECE | 95 | 1 |
| 4 | 1008 | ARPAN | ECE | 87 | 2 |
| 5 | 1009 | VINAY | ECE | 76 | 3 |
| 6 | 1002 | RISHABH | EEE | 91 | 1 |
| 7 | 1003 | ANIKANT | EEE | 89 | 2 |
| 8 | 1014 | NITHA | EEE | 89 | 2 |
| 9 | 1001 | NITISH | EEE | 82 | 3 |
| 10 | 1004 | RUPESH | EEE | 78 | 4 |
| 11 | 1012 | AMITH | MECH | 89 | 1 |
| 12 | 1013 | SUNNY | MECH | 78 | 2 |
| 13 | 1011 | PRASHANTH | MECH | 69 | 3 |
| 14 | 1010 | ROHITH | MECH | 67 | 4 |

```
SELECT *,
LAST_VALUE(MARKS) OVER (ORDER BY MARKS DESC) as " marks from last in desc order"
FROM MARKS;
```

| | Student_id | name | branch | marks | marks from last in desc order |
|---|---|---|---|---|---|
| 1 | 1007 | DEEPAK | ECE | 95 | 95 |
| 2 | 1002 | RISHABH | EEE | 91 | 91 |
| 3 | 1003 | ANIKANT | EEE | 89 | 89 |
| 4 | 1012 | AMITH | MECH | 89 | 89 |
| 5 | 1014 | NITHA | EEE | 89 | 89 |
| 6 | 1008 | ARPAN | ECE | 87 | 87 |
| 7 | 1001 | NITISH | EEE | 82 | 82 |
| 8 | 1013 | SUNNY | MECH | 78 | 78 |
| 9 | 1004 | RUPESH | EEE | 78 | 78 |
| 10 | 1005 | SHUBHAM | CSE | 77 | 77 |
| 11 | 1009 | VINAY | ECE | 76 | 76 |
| 12 | 1011 | PRASHANTH | MECH | 69 | 69 |
| 13 | 1010 | ROHITH | MECH | 67 | 67 |
| 14 | 1006 | VEDHAL | CSE | 43 | 43 |

## Assignment 1:

**Draft a brief report on the use of transaction logs for data recovery and create a hypothetical scenario where a transaction log is instrumental in data recovery after an unexpected shutdown.**

## Report on the Use of Transaction Logs for Data Recovery

### Introduction

- Transaction logs are essential tools in database management systems (DBMS) for ensuring data integrity and facilitating recovery in case of unexpected events, such as system crashes or hardware failures.
- These logs provide a detailed record of all database transactions, including changes made to the data and the sequence of operations.
- This report explores the role of transaction logs in data recovery and presents a hypothetical scenario demonstrating their effectiveness.

### Operations supported by the transaction log

The transaction log supports the following operations:

- Individual transaction recovery.
- Recovery of all incomplete transactions when SQL Server is started.
- Rolling a restored database, file, filegroup, or page forward to the point of failure.
- Supporting transactional replication.
- Supporting high availability and disaster recovery solutions: Always On availability groups, database mirroring, and log shipping.

### Function of Transaction Logs

Transaction logs, also known as redo logs or write-ahead logs, serve several critical functions:

1. **Data Integrity**: Transaction logs capture every modification made to the database. Each transaction is recorded with details such as the time of modification, the type of operation, and the data affected. This ensures that the database can be restored to a consistent state if needed.
2. **Recovery**: In the event of a system failure, transaction logs can be used to recover lost or corrupted data. The recovery process involves applying the logged transactions to bring the database back to the point of failure or to a specific recovery point.
3. **Auditing**: Transaction logs provide a trail of changes that can be used for auditing purposes, helping administrators track who made changes and when.
4. **Consistency**: By replaying the transactions recorded in the log, a DBMS can ensure that all operations are completed correctly, maintaining data consistency.

**Recovery using Log records**

After a system crash has occurred, the system consults the log to determine which transactions need to be redone and which need to be undone.

1. Transaction Ti needs to be undone if the log contains the record <Ti start> but does not contain either the record <Ti commit> or the record <Ti abort>.

2. Transaction Ti needs to be redone if log contains record <Ti start> and either the record <Ti commit> or the record <Ti abort>.

**Hypothetical Scenario**

**Background:** Sunny uses a simple database to manage daily sales transactions. The database records each sale, including the items sold and the total amount. To simplify operations, the database is designed to automatically back up the data every night, and transaction logs are used to capture changes throughout the day.

**Incident:** During a busy lunch hour, Jane accidentally deletes the entire sales record for the last two hours while trying to fix a minor issue with the database. She realizes the mistake immediately but the deleted records are crucial for the day's financial reconciliation.

## Assignment 2:

**Explain the ACID properties of a transaction in your own words. Write SQL statements to simulate a transaction that includes locking and demonstrate different isolation levels to show concurrency control.**

**ACID Properties of a Transaction**

Transactions in database systems are designed to be reliable and consistent, and they adhere to four key properties collectively known as ACID: Atomicity, Consistency, Isolation, and Durability. Here's a simplified explanation of each property:

1. **Atomicity:** A transaction is an indivisible unit of work. This means that all operations within a transaction must complete successfully as a single unit. If any operation fails, the entire transaction is rolled back, and the database is left unchanged. In other words, a transaction either fully completes or does not occur at all.

2. **Consistency:** A transaction must transition the database from one consistent state to another. It ensures that any transaction will leave the database in a valid state, adhering to all defined rules and constraints, both before and after the transaction is executed.

3. **Isolation:** Transactions are isolated from each other. This means that the operations of one transaction are not visible to other transactions until the transaction is completed.

This prevents transactions from interfering with each other, ensuring that they do not produce unintended results.

4. **Durability:** Once a transaction has been committed, its effects are permanent and survive any subsequent system failures. The changes made by the transaction are recorded in non-volatile memory and are not lost, even if the system crashes.

```
create table inventory (
ItemID int primary key,
ItemName varchar(50),
stockQuantity int
);

Insert into inventory(ItemID,ItemName,stockQuantity)
values(1,'widget',100);

Insert into inventory(ItemID,ItemName,stockQuantity)
values(2,'Gadget',200);

begin transaction;
select * from inventory with(ROWLOCK, UPDLOCK) where ItemId = 1;

update inventory set stockQuantity = stockQuantity - 10 where ItemId = 1;
commit TRANSACTION;

begin transaction;
select * from inventory with(ROWLOCK, UPDLOCK) where ItemId = 1;

update inventory set stockQuantity = stockQuantity + 20 where ItemId = 1;
commit TRANSACTION;
```

```
set transaction Isolation level read uncommitted;
begin transaction;
select * from inventory
where ItemId =1;
commit transaction;

set transaction Isolation level read committed;
begin transaction;
select * from inventory
where ItemId =1;
commit transaction;

set transaction Isolation level repeatable read ;
begin transaction;
select * from inventory
where ItemId =1;
commit transaction;

set transaction Isolation level serializable;
begin transaction;
select * from inventory
where ItemId =1;
commit transaction;
```

## Assignment 3:

**Begin a transaction, perform a series of INSERTs into 'orders', setting a SAVEPOINT after each, rollback to the second SAVEPOINT, and COMMIT the overall transaction.**

```
create table Orders(
OrderId int identity(1,1) primary key,
OrderDate date not null,
CustomerId int  not null,
TotalAmmount float not null,
);

BEGIN TRANSACTION;
INSERT INTO ORDERS (OrderId, OrderDate, CustomerId, TotalAmmount)
VALUES(1, '2024-07-15', 104, 140.35);
 SAVE TRANSACTION SAVEPOINT1;

INSERT INTO ORDERS (OrderId, OrderDate, CustomerId, TotalAmmount)
VALUES(2, '2024-08-15', 105, 200.65);
SAVE TRANSACTION SAVEPOINT2;

INSERT INTO ORDERS (OrderId, OrderDate, CustomerId, TotalAmmount)
VALUES(3, '2024-05-25', 105, 257.90);
SAVE TRANSACTION SAVEPOINT3;

ROLLBACK TRANSACTION SAVEPOINT3;

SELECT * FROM ORDERS;
COMMIT TRANSACTION;
```

| | OrderId | OrderDate | CustomerId | TotalAmmount |
|---|---|---|---|---|
| 1 | 1 | 2024-07-15 | 101 | 140.35 |
| 2 | 1 | 2024-07-15 | 104 | 140.35 |
| 3 | 2 | 2024-08-15 | 102 | 200.65 |
| 4 | 2 | 2024-08-15 | 105 | 200.65 |
| 5 | 3 | 2024-05-25 | 103 | 257.9 |
| 6 | 3 | 2024-05-25 | 105 | 257.9 |

Fig: Output