SQL

Assessment - 2025

Instruction:

- Use this document as answer sheet.
- Copy and paste your SQL query and sample output table screenshot below each question.
- Save the document with your name in given format. (SQL_2025_Your_Name)
- Email it to all 5 SQL trainers. (Kundan, Lejoy, Karthikeyan, Rajasundar, Smeer)
- Time – 2.5 hour
- No internet

Questions:

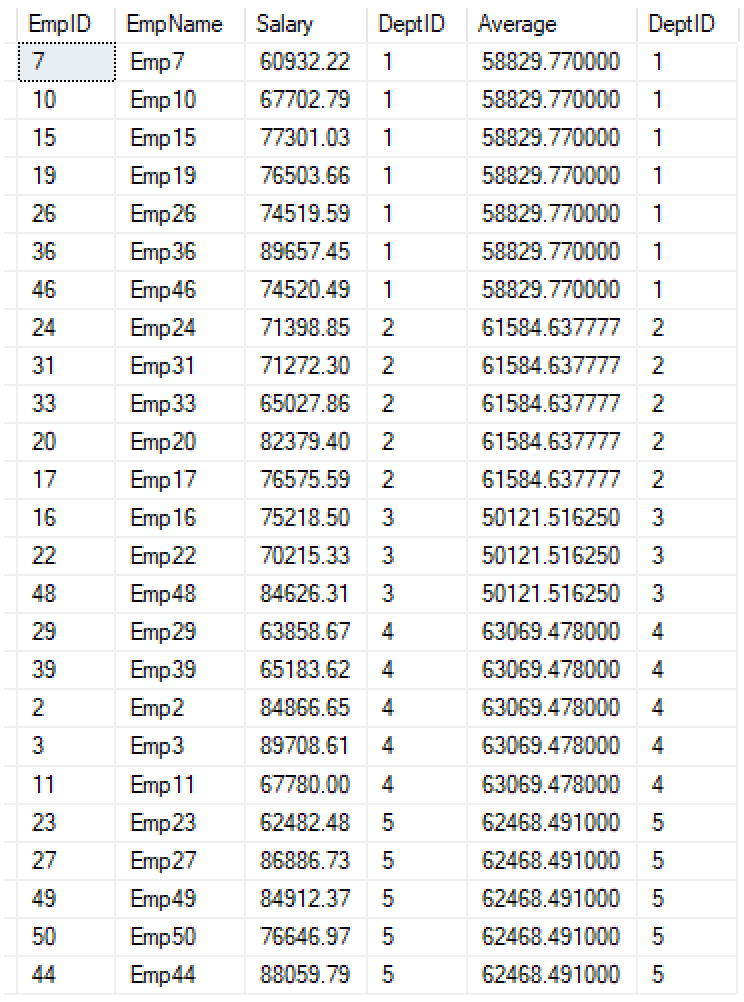
1. Provide details of the top five employees from each department, but only if their earnings exceed the department's average.

   Dataset: execute below statements to create input table

```
CREATE TABLE Departments (
    DeptID INT PRIMARY KEY IDENTITY(1,1),
    DeptName VARCHAR(50)
);

CREATE TABLE Employees (
    EmpID INT PRIMARY KEY IDENTITY(1,1),
    EmpName VARCHAR(50),
    Salary DECIMAL(10,2),
    DeptID INT FOREIGN KEY REFERENCES Departments(DeptID)
);

INSERT INTO Departments (DeptName)
VALUES
('HR'), ('Finance'), ('IT'), ('Marketing'), ('Sales');

INSERT INTO Employees (EmpName, Salary, DeptID)
SELECT
    'Emp' + CAST(ROW_NUMBER() OVER (ORDER BY (SELECT NULL)) AS VARCHAR),
    ROUND(RAND(CHECKSUM(NEWID())) * (90000-30000) + 30000, 2),
    (ABS(CHECKSUM(NEWID())) % 5) + 1
FROM master.dbo.spt_values
WHERE type = 'P' AND number BETWEEN 1 AND 50;
```

```
SELECT * FROM (
        SELECT * FROM employees
        ) e
        join
        (SELECT
        avg(salary) AS Average,
        DeptID FROM employees
        GROUP BY deptid
        ) a
        ON
        e.deptid=a.deptid
        WHERE
        salary>Average);
```

OUTPUT:



EmpID	EmpName	Salary	DeptID	Average	DeptID
7	Emp7	60932.22	1	58829.770000	1
10	Emp10	67702.79	1	58829.770000	1
15	Emp15	77301.03	1	58829.770000	1
19	Emp19	76503.66	1	58829.770000	1
26	Emp26	74519.59	1	58829.770000	1
36	Emp36	89657.45	1	58829.770000	1
46	Emp46	74520.49	1	58829.770000	1
24	Emp24	71398.85	2	61584.637777	2
31	Emp31	71272.30	2	61584.637777	2
33	Emp33	65027.86	2	61584.637777	2
20	Emp20	82379.40	2	61584.637777	2
17	Emp17	76575.59	2	61584.637777	2
16	Emp16	75218.50	3	50121.516250	3
22	Emp22	70215.33	3	50121.516250	3
48	Emp48	84626.31	3	50121.516250	3
29	Emp29	63858.67	4	63069.478000	4
39	Emp39	65183.62	4	63069.478000	4
2	Emp2	84866.65	4	63069.478000	4
3	Emp3	89708.61	4	63069.478000	4
11	Emp11	67780.00	4	63069.478000	4
23	Emp23	62482.48	5	62468.491000	5
27	Emp27	86886.73	5	62468.491000	5
49	Emp49	84912.37	5	62468.491000	5
50	Emp50	76646.97	5	62468.491000	5
44	Emp44	88059.79	5	62468.491000	5

2. **Stored Procedure**

   Scenario:

   A retail company needs to generate a monthly sales report. They want to aggregate sales data, apply business rules, and store the results in a reporting table.

   Task: Design a stored procedure to:

   1.**Calculate Total Sales and Revenue:** Sum quantities and revenue (Quantity * PricePerUnit) for completed orders only.

```
CREATE PROCEDURE TotalSales_Revenue

AS

BEGIN

          SELECT (quantity*priceperunit) AS total_sales FROM retail_sales_orders WHERE
status='Completed';

END

EXEC TotalSales_Revenue;
```

   2.**Apply Regional Tax Rates:** Calculate the total tax based on the store’s region.

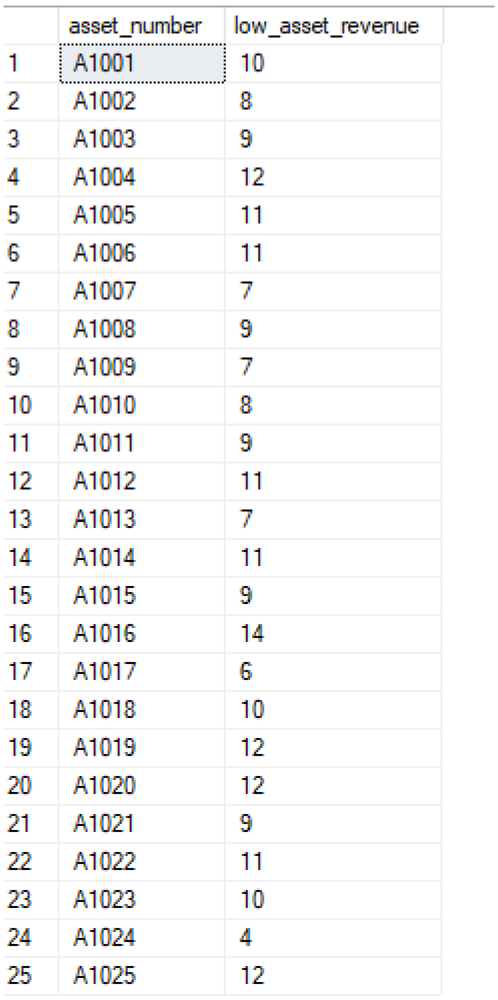
   3.**Insert the Results:** Save the aggregated data into the SalesReport table, with a timestamped report month.

3. The client is facing an issue where they generated an invoice in one month and received payment in a different month from a customer in a different region. Due to currency rate changes, they experience either a gain or a loss when closing the invoice. Calculate the gain or loss amount for each invoice and create a flag to indicate whether the invoice amount reflects a gain or a loss. (Use CTE's and functions if needed to solve the problem) (Dataset : Exchange rate .csv and Invoice dataset 1.csv)

4. For each **asset_number**, **calculate the total number of times** the asset's revenue **decreased** compared to the previous order date.

```
SELECT

 d.asset_number,

 count(*) as low_asset_revenue

 FROM

 (SELECT

    asset_number,

          client_id,

          equipment_type_id,

          daily_revenue,
                category_id,

                description_id,

                order_date,

                is_active,

                region_id,

                lag(daily_revenue) OVER (PARTITION BY asset_number ORDER BY order_date
ASC) AS lag_value FROM fact_asset_revenue) d

                WHERE

                d.lag_value is not null and d.daily_revenue<d.lag_value group by d.asset_number;
```

OUTPUT:



	asset_number	low_asset_revenue
1	A1001	10
2	A1002	8
3	A1003	9
4	A1004	12
5	A1005	11
6	A1006	11
7	A1007	7
8	A1008	9
9	A1009	7
10	A1010	8
11	A1011	9
12	A1012	11
13	A1013	7
14	A1014	11
15	A1015	9
16	A1016	14
17	A1017	6
18	A1018	10
19	A1019	12
20	A1020	12
21	A1021	9
22	A1022	11
23	A1023	10
24	A1024	4
25	A1025	12

5.

a. Detecting the First Time an Asset's Revenue Exceeded ₹3,000

```
SELECT
```

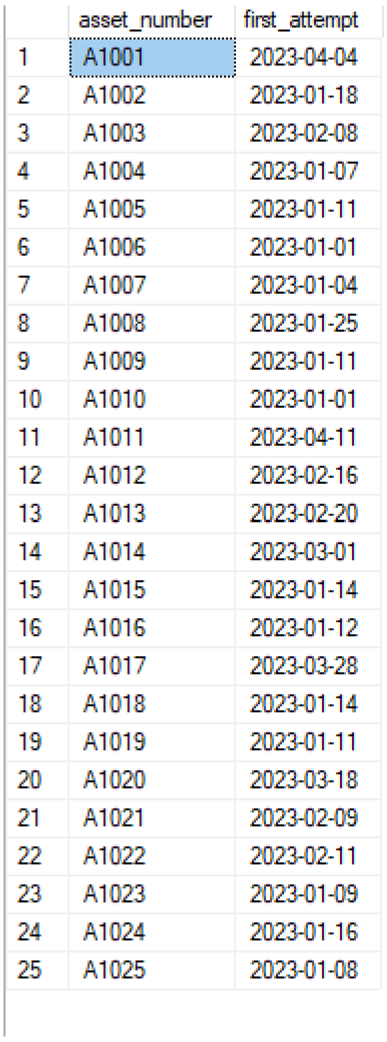
asset_number,

MIN(order_date) AS first_attempt

FROM fact_asset_revenue

WHERE daily_revenue>3000

GROUP BY asset_number;



	asset_number	first_attempt
1	A1001	2023-04-04
2	A1002	2023-01-18
3	A1003	2023-02-08
4	A1004	2023-01-07
5	A1005	2023-01-11
6	A1006	2023-01-01
7	A1007	2023-01-04
8	A1008	2023-01-25
9	A1009	2023-01-11
10	A1010	2023-01-01
11	A1011	2023-04-11
12	A1012	2023-02-16
13	A1013	2023-02-20
14	A1014	2023-03-01
15	A1015	2023-01-14
16	A1016	2023-01-12
17	A1017	2023-03-28
18	A1018	2023-01-14
19	A1019	2023-01-11
20	A1020	2023-03-18
21	A1021	2023-02-09
22	A1022	2023-02-11
23	A1023	2023-01-09
24	A1024	2023-01-16
25	A1025	2023-01-08

b. For each client and category, find the first and last order dates along with the total revenue within this period.

SELECT

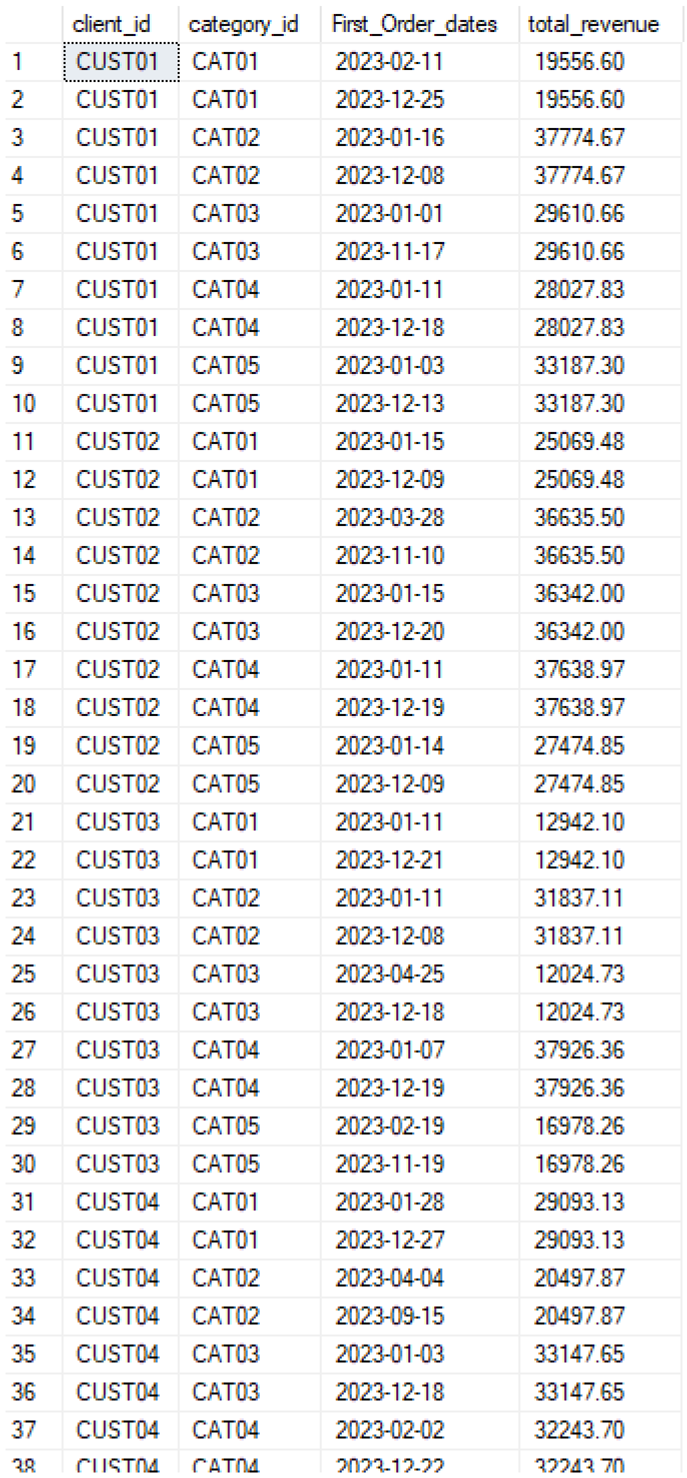
client_id,

category_id,

MIN(order_date) AS First_Order_dates,

```
      SUM(daily_revenue) AS total_revenue

      FROM fact_asset_revenue

      GROUP BY client_id,category_id

UNION

SELECT

    client_id,

      category_id,

      MAX(order_date) AS Last_Order_dates,

      SUM(daily_revenue) AS total_revenue

      FROM fact_asset_revenue

     GROUP BY client_id,category_id;
```

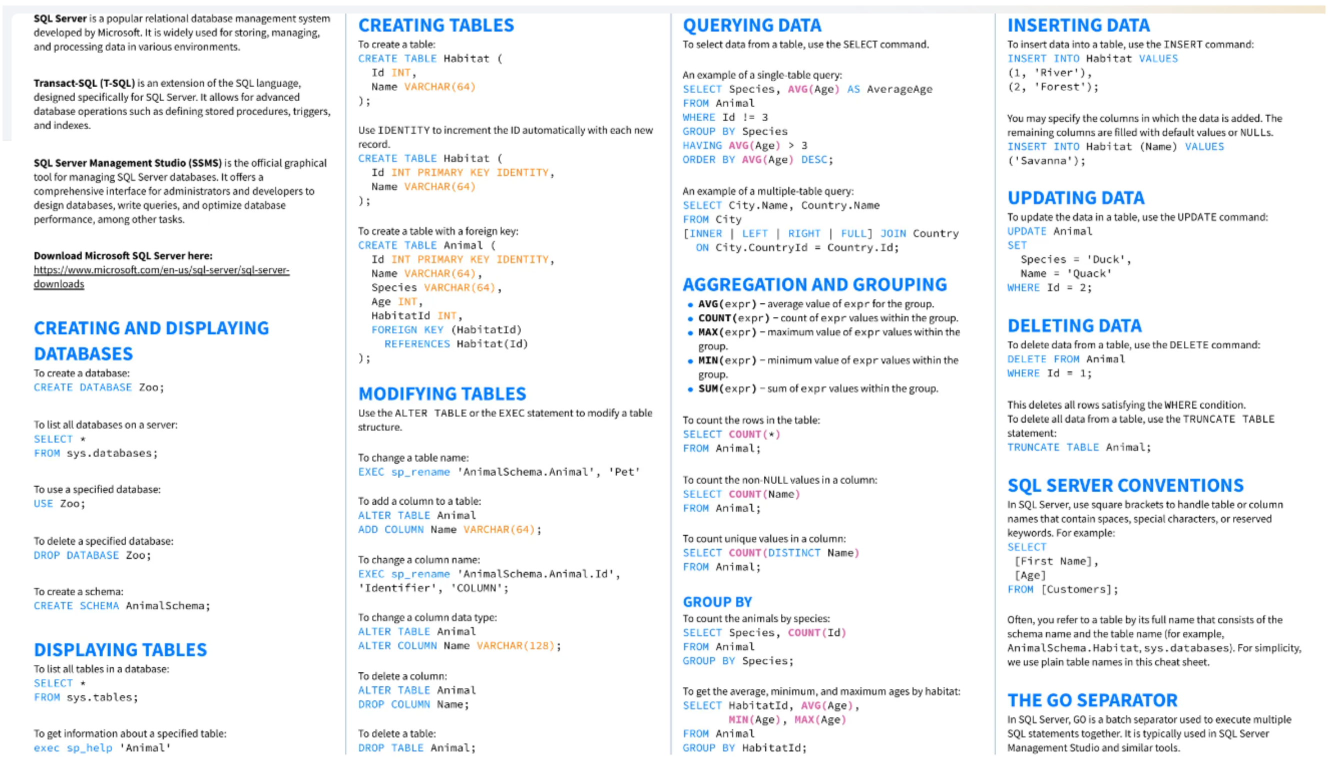
OUTPUT:



	client_id	category_id	First_Order_dates	total_revenue
1	CUST01	CAT01	2023-02-11	19556.60
2	CUST01	CAT01	2023-12-25	19556.60
3	CUST01	CAT02	2023-01-16	37774.67
4	CUST01	CAT02	2023-12-08	37774.67
5	CUST01	CAT03	2023-01-01	29610.66
6	CUST01	CAT03	2023-11-17	29610.66
7	CUST01	CAT04	2023-01-11	28027.83
8	CUST01	CAT04	2023-12-18	28027.83
9	CUST01	CAT05	2023-01-03	33187.30
10	CUST01	CAT05	2023-12-13	33187.30
11	CUST02	CAT01	2023-01-15	25069.48
12	CUST02	CAT01	2023-12-09	25069.48
13	CUST02	CAT02	2023-03-28	36635.50
14	CUST02	CAT02	2023-11-10	36635.50
15	CUST02	CAT03	2023-01-15	36342.00
16	CUST02	CAT03	2023-12-20	36342.00
17	CUST02	CAT04	2023-01-11	37638.97
18	CUST02	CAT04	2023-12-19	37638.97
19	CUST02	CAT05	2023-01-14	27474.85
20	CUST02	CAT05	2023-12-09	27474.85
21	CUST03	CAT01	2023-01-11	12942.10
22	CUST03	CAT01	2023-12-21	12942.10
23	CUST03	CAT02	2023-01-11	31837.11
24	CUST03	CAT02	2023-12-08	31837.11
25	CUST03	CAT03	2023-04-25	12024.73
26	CUST03	CAT03	2023-12-18	12024.73
27	CUST03	CAT04	2023-01-07	37926.36
28	CUST03	CAT04	2023-12-19	37926.36
29	CUST03	CAT05	2023-02-19	16978.26
30	CUST03	CAT05	2023-11-19	16978.26
31	CUST04	CAT01	2023-01-28	29093.13
32	CUST04	CAT01	2023-12-27	29093.13
33	CUST04	CAT02	2023-04-04	20497.87
34	CUST04	CAT02	2023-09-15	20497.87
35	CUST04	CAT03	2023-01-03	33147.65
36	CUST04	CAT03	2023-12-18	33147.65
37	CUST04	CAT04	2023-02-02	32243.70
38	CUST04	CAT04	2023-12-22	32243.70

Note : For Question 4 and 5 Use the Asset Dataset provided for Handson

Cheat sheet:



SQL Server is a popular relational database management system developed by Microsoft. It is widely used for storing, managing, and processing data in various environments.

Transact-SQL (T-SQL) is an extension of the SQL language, designed specifically for SQL Server. It allows for advanced database operations such as defining stored procedures, triggers, and indexes.

SQL Server Management Studio (SSMS) is the official graphical tool for managing SQL Server databases. It offers a comprehensive interface for administrators and developers to design databases, write queries, and optimize database performance, among other tasks.

Download Microsoft SQL Server here:
https://www.microsoft.com/en-us/sql-server/sql-server-downloads

CREATING AND DISPLAYING DATABASES

To create a database:
```
CREATE DATABASE Zoo;
```

To list all databases on a server:
```
SELECT *
FROM sys.databases;
```

To use a specified database:
```
USE Zoo;
```

To delete a specified database:
```
DROP DATABASE Zoo;
```

To create a schema:
```
CREATE SCHEMA AnimalSchema;
```

DISPLAYING TABLES

To list all tables in a database:
```
SELECT *
FROM sys.tables;
```

To get information about a specified table:
```
exec sp_help 'Animal'
```

CREATING TABLES

To create a table:
```
CREATE TABLE Habitat (
  Id INT,
  Name VARCHAR(64)
);
```

Use IDENTITY to increment the ID automatically with each new record.
```
CREATE TABLE Habitat (
  Id INT PRIMARY KEY IDENTITY,
  Name VARCHAR(64)
);
```

To create a table with a foreign key:
```
CREATE TABLE Animal (
  Id INT PRIMARY KEY IDENTITY,
  Name VARCHAR(64),
  Species VARCHAR(64),
  Age INT,
  HabitatId INT,
  FOREIGN KEY (HabitatId)
    REFERENCES Habitat(Id)
);
```

MODIFYING TABLES

Use the ALTER TABLE or the EXEC statement to modify a table structure.

To change a table name:
```
EXEC sp_rename 'AnimalSchema.Animal', 'Pet'
```

To add a column to a table:
```
ALTER TABLE Animal
ADD COLUMN Name VARCHAR(64);
```

To change a column name:
```
EXEC sp_rename 'AnimalSchema.Animal.Id',
'Identifier', 'COLUMN';
```

To change a column data type:
```
ALTER TABLE Animal
ALTER COLUMN Name VARCHAR(128);
```

To delete a column:
```
ALTER TABLE Animal
DROP COLUMN Name;
```

To delete a table:
```
DROP TABLE Animal;
```

QUERYING DATA

To select data from a table, use the SELECT command.

An example of a single-table query:
```
SELECT Species, AVG(Age) AS AverageAge
FROM Animal
WHERE Id != 3
GROUP BY Species
HAVING AVG(Age) > 3
ORDER BY AVG(Age) DESC;
```

An example of a multiple-table query:
```
SELECT City.Name, Country.Name
FROM City
[INNER | LEFT | RIGHT | FULL] JOIN Country
  ON City.CountryId = Country.Id;
```

AGGREGATION AND GROUPING

- **AVG**(expr) – average value of expr for the group.
- **COUNT**(expr) – count of expr values within the group.
- **MAX**(expr) – maximum value of expr values within the group.
- **MIN**(expr) – minimum value of expr values within the group.
- **SUM**(expr) – sum of expr values within the group.

To count the rows in the table:
```
SELECT COUNT(*)
FROM Animal;
```

To count the non-NULL values in a column:
```
SELECT COUNT(Name)
FROM Animal;
```

To count unique values in a column:
```
SELECT COUNT(DISTINCT Name)
FROM Animal;
```

GROUP BY

To count the animals by species:
```
SELECT Species, COUNT(Id)
FROM Animal
GROUP BY Species;
```

To get the average, minimum, and maximum ages by habitat:
```
SELECT HabitatId, AVG(Age),
       MIN(Age), MAX(Age)
FROM Animal
GROUP BY HabitatId;
```

INSERTING DATA

To insert data into a table, use the INSERT command:
```
INSERT INTO Habitat VALUES
(1, 'River'),
(2, 'Forest');
```

You may specify the columns in which the data is added. The remaining columns are filled with default values or NULLs.
```
INSERT INTO Habitat (Name) VALUES
('Savanna');
```

UPDATING DATA

To update the data in a table, use the UPDATE command:
```
UPDATE Animal
SET
  Species = 'Duck',
  Name = 'Quack'
WHERE Id = 2;
```

DELETING DATA

To delete data from a table, use the DELETE command:
```
DELETE FROM Animal
WHERE Id = 1;
```

This deletes all rows satisfying the WHERE condition.
To delete all data from a table, use the TRUNCATE TABLE statement:
```
TRUNCATE TABLE Animal;
```

SQL SERVER CONVENTIONS

In SQL Server, use square brackets to handle table or column names that contain spaces, special characters, or reserved keywords. For example:
```
SELECT
 [First Name],
 [Age]
FROM [Customers];
```

Often, you refer to a table by its full name that consists of the schema name and the table name (for example, AnimalSchema.Habitat, sys.databases). For simplicity, we use plain table names in this cheat sheet.

THE GO SEPARATOR

In SQL Server, GO is a batch separator used to execute multiple SQL statements together. It is typically used in SQL Server Management Studio and similar tools.