Azure SQL Database

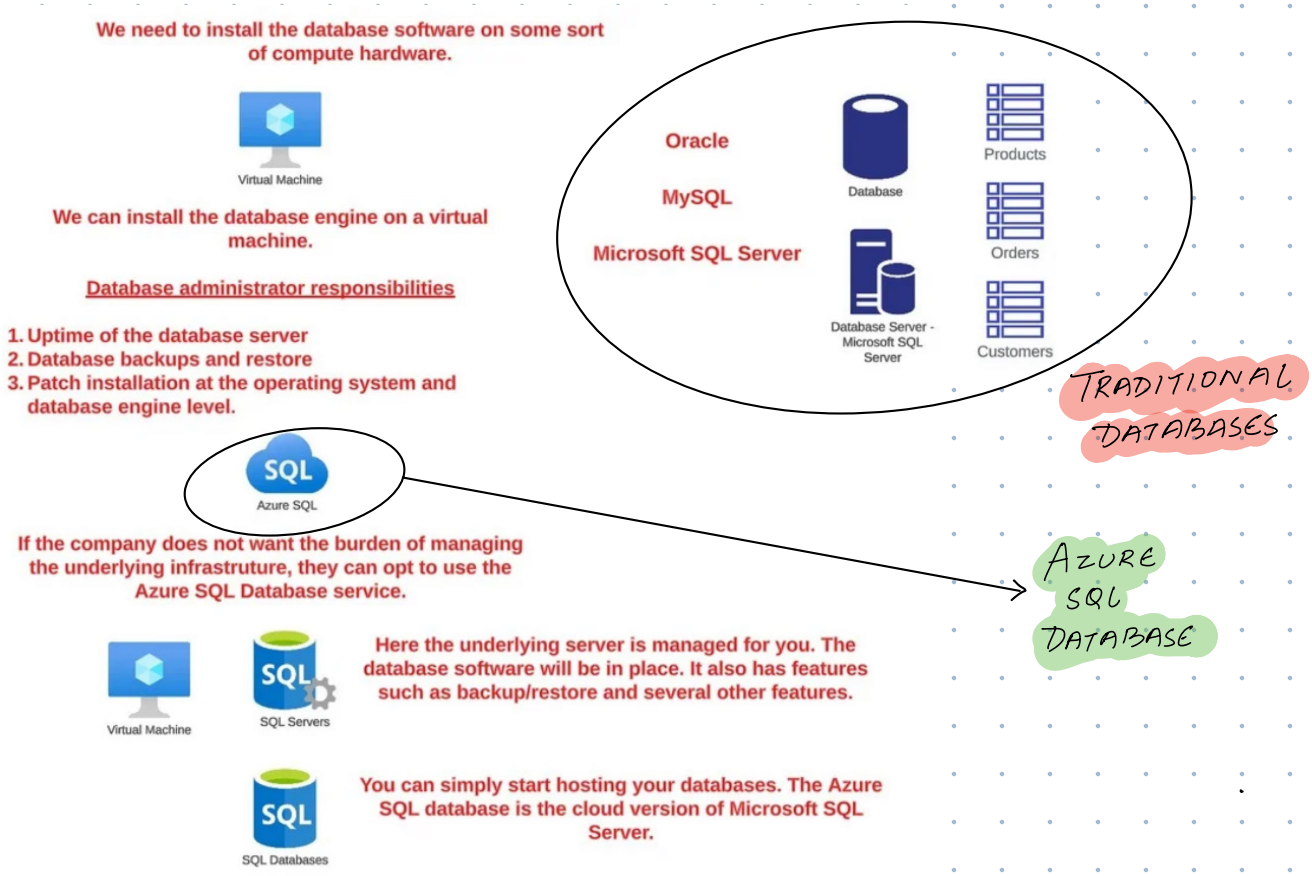
Azure SQL Database is a fully managed, cloud-based relational database service by Microsoft. It offers automatic scaling, high availability (99.99% uptime), built-in security, and AI-driven performance optimization, eliminating the need for manual database management. Ideal for web apps, BI solutions, and cloud-native applications.

Key Features:

- ✔ **Fully Managed** – Microsoft handles backups, updates, and scaling.
- ✔ **Built-in Security** – Features **encryption, threat detection, and access control**.
- ✔ **Scalability** – Supports **serverless** and **hyperscale** models for dynamic workloads.
- ✔ **High Availability** – Provides **99.99% uptime** with geo-replication options.
- ✔ **AI-powered Performance Optimization** – Automatically tunes indexes and queries.

In short it is online version of Microsoft SQL Server

MS SQL SERVER VS AZURE DATABASE



Create Azure SQL Database:

Create Resource → SQL Database

- Choose Database Name → mydatabase
- Choose Server Name → server012
- Create New

Authentication method

- ○ Use Microsoft Entra-only authentication
- ○ Use both SQL and Microsoft Entra authentication
- ◉ Use SQL authentication

Server admin login * : sqladmin ✓

Password * : •••••••• Toju ✓

Confirm password * : •••••••• ✓

Add current client IP address * : No | Yes

Data source

Start with a blank database, restore from a backup or select sample data to populate your new database.

Use existing data * : None | Backup | Sample

AdventureWorksLT will be created as the sample database.

☐	mydatabase (server012/mydatabase)	SQL database
☐	server012	SQL server

(Both are created)

Install Azure Data Studio

→ New Connection

Server name — Use this ↓

server012.database.windows.net

→ Make sure Server is Accessible

TRANSACT SQL is an extension of SQL developed by Microsoft, used to manipulate data in MSSQL

```sql
-- Select All
SELECT * FROM SalesLT.SalesOrderDetail

-- Select Specific Columns
SELECT OrderQty, UnitPrice from SalesLT.SalesOrderDetail

-- Use Alias
SELECT UnitPrice * UnitPrice as 'Total Sales' from SalesLT.SalesOrderDetail

-- Use Where Clause
SELECT UnitPrice FROM SalesLT.SalesOrderDetail
WHERE UnitPrice < 20

-- Order by Clause
SELECT OrderQty FROM SalesLT.SalesOrderDetail
ORDER BY UnitPrice DESC

-- Aggregate Function
SELECT DISTINCT COUNT(OrderQty) 'Unique Orders' FROM SalesLT.SalesOrderDetail

-- Group by and Having Clause
SELECT SalesOrderID, SUM(UnitPrice) AS TotalUnitPrice
FROM SalesLT.SalesOrderDetail
GROUP BY SalesOrderID
HAVING SUM(UnitPrice) > 1000;
```

CTE — Common Table Expression

is a temporary Result set that makes complex Queries easy to read and Reuse. More efficient than Sub Query.

Note; It is not create in DB or Temp DB, just the Temporary Result set.

```sql
WITH RankedSales AS (
    SELECT
        SalesID,
        Salesperson,
        Amount,
        RANK() OVER (PARTITION BY Salesperson ORDER BY Amount DESC) AS SalesRank
    FROM Sales
)
SELECT
    SalesID,
    Salesperson,
    Amount,
    SalesRank
FROM RankedSales
WHERE SalesRank = 1;
```

Window Functions

window functions perform calculations based on a window over existing rows without aggregating them into fewer rows a defined set of rows (the “window”) related to the current row without reducing the overall number of rows in the result set. unlike Group by.

Window Function → Apply function over existing Data based on (window)

```sql
SELECT
    SalesID,
    Salesperson,
    Amount,
    RANK() OVER (PARTITION BY Salesperson ORDER BY Amount DESC) AS SalesRank
FROM Sales;
```

Sample Table: Sales

SalesID	Salesperson	Amount
1	Alice	100
2	Bob	200
3	Alice	150
4	Bob	250
5	Alice	200

Expected Output

SalesID	Salesperson	Amount	SalesRank
5	Alice	200	1
3	Alice	150	2
1	Alice	100	3
4	Bob	250	1
2	Bob	200	2

↳ Here First partition be ↑ Order by Amount Desc ↑

Based on window, now we will finally apply Rank (without collapsing Data.)

LEAD & LAG

LEAD ↓ Next

LAG ↓ Previous

LEAD → Last Row Null
LAG → Fist Row Null

Example

ID	$	LEAD	LAG
1	100	200	Null
1	200	300	100
2	300	Null	200

Can be used to do previous or next order comparison

Create & Insert Data in Tables

Step 1: Create Tables with Constraints

sql Copy

```sql
-- Create the Salesperson table with a Primary Key constraint.
CREATE TABLE Salesperson (
    SalespersonID INT PRIMARY KEY,
    SalespersonName VARCHAR(50) NOT NULL
);

-- Create the Sales table with:
-- - Primary Key on SalesID
-- - Foreign Key referencing Salesperson(SalespersonID)
-- - A CHECK constraint ensuring Amount is non-negative
-- - A DEFAULT value for SaleDate using GETDATE()
CREATE TABLE Sales (
    SalesID INT PRIMARY KEY,
    SalespersonID INT NOT NULL,
    Amount DECIMAL(10,2) NOT NULL CHECK (Amount >= 0),
    SaleDate DATE NOT NULL DEFAULT GETDATE(),
    CONSTRAINT FK_Sales_Salesperson
        FOREIGN KEY (SalespersonID) REFERENCES Salesperson(SalespersonID)
);
```

Step 2: Insert Data into the Tables

sql Copy

```sql
-- Insert sample data into the Salesperson table.
INSERT INTO Salesperson (SalespersonID, SalespersonName)
VALUES
    (1, 'Alice'),
    (2, 'Bob');

-- Insert sample data into the Sales table.
INSERT INTO Sales (SalesID, SalespersonID, Amount, SaleDate)
VALUES
    (1, 1, 100.00, '2025-03-01'),
    (2, 2, 200.00, '2025-03-02'),
    (3, 1, 150.00, '2025-03-03');
```

DML - Data Manipulation Language

Insert → Insert into

Update → Update Set

Delete → Delete From

Note; Azure Synapse, we don't have concept of Foreign key Constraints