Module/Week6 – Assignment 4

Topic:  **Introduction to Cloud Computing; Cloud Data Platform**

*Student’s Full Name****: LIBIN NAITHELLOOR GEORGE***

*Course Title:* *: Data Warehousing and Analytics in the Cloud*

*Term name and year: Example, Fall 2023*

*Submission Week: Week 6- Assignment 4*

*Instructor’s Name:* *Dr.Nayem Rahman*

*Date of Submission:* *17 August 2024*

***PURPOSE:***

This assignment introduces you to the Microsoft Azure portal, related databases, tools, technologies and also how to create Azure Resource Group, Azure SQL databases using the Azure portal, how to create Azure Blob storage, and how to connect to the database and query data using the Query editor (preview) in the Azure portal. This also shows you how to delete resource groups when you are finished with all these activities. You will perform activities to delete the resource group you created, which will also delete the server and single database within it.

**Q1. Create your Azure free account**

Go to the address https://azure.microsoft.com/en-us/free/students/ and create a free Azure account using your university email address. Note that you do NOT need a credit card to create the account. If you find yourself entering your credit card information know that you are doing it wrong.

|  |
| --- |
|  |
|  |

**Q2. Create a container in Azure Storage, and to upload and download block blobs in that container.**

Follow the instructions here to create a container in Azure Storage, and to upload and download block blobs in that container. https://learn.microsoft.com/en-us/azure/storage/blobs/storage-quickstart-blobs-portal Assumption: You already have an account using your university email address. Go to the portal and login.

| Create Resource Group |
| --- |
|  |
| Create Storage Account |
|  |
|  |
|  |
| Creating Containers |
|  |
|  |
| Uploading Blob Data |
|  |
| The file is uploaded in a folder called SampleDemoFolder |
| Downloading a Blob Resource |
|  |
|  |
| Deleting Blob files |
|  |
|  |

**Q3. Create a SQL database on Azure**

| Create SQL Database Server |
| --- |
|  |
| Create SQL Database |
|  |
|  |
|  |
|  |
|  |
|  |

**Q4. Summarize SQL database creation in Azure**

Write in one or two paragraphs what you have done and learned by creating a single database - Azure SQL Database – in Azure Portal.

*We need to create a new database server before creating the SQL database just like how we need a system to act as an SQL database server. We can choose the Service tier as General Purpose and the Compute tier will be Serverless (Most budget-friendly, serverless compute) as it is the least costly. Then we can create the database using this server by giving a DB name. We can choose a production or development Workload environment. This Workload environment option helps to preset some default configuration settings. We can select Locally redundant storage (LRS), Zone-redundant storage (ZRS), and Geo-redundant storage (GRS) as Storage redundancies for backups. We can choose the most basic LRS for this assignment. ZRS and GRS have safer storage redundancies to use in production to get High Availability. We should use Sample existing data to run the query given in the next question. Also, make sure we have chosen a public endpoint in connectivity to access the DB server globally. Some firewall rules can be chosen accordingly to increase security and access.*

*I have learned to create a Database Server as well as creating SQL database. I have seen different authentication methods, storage redundancies, connectivity options, etc. which can be configured while creating an SQL database.*

**Q5. Query the SQL database in Azure**

Query the database you created in Azure. You can use the Query editor (preview) in the Azure portal to connect to the database and query data. Run the following query in the query editor of the database on the Azure portal and take a screenshot of the result of the query. SELECT TOP 10 pc.Name as CategoryName, p.name as ProductName FROM SalesLT.ProductCategory pc JOIN SalesLT.Product p ON pc.productcategoryid = p.productcategoryid;

|  |
| --- |

**Q6. Describe the SQL database "SalesLT" on Azure.**

Analyze data in the tables in the SalesLT database schema using the Azure Query editor. Identify the primary key (PK) columns, and different data types, and observe the relationships among these tables. Write at least one paragraph.

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

*The* ***SalesLT*** *database provide in the sample contains* ***Address, Customer, Product, ProductCategory, ProductModel, ProductDescription, SalesOrderDetail, SalesOrderHeader, CustomerAddress*** *and* ***ProductModelProductDescription*** *tables. Also note that unique identifier* ***rowguid*** *and* ***ModifedDate****(datetime) is present for all the tables.*

***Address*** *table stores address details like* ***AddressLine1, AddressLine2, City, StateProvinance, CountryRegion*** *and* ***PostalCode****. The unique identifier* ***rowguid*** *and* ***ModifedDate****(datetime) is also present just like any other tables. Note that only* ***AddressLine2*** *is allowed to be null. We can also see a new datatype called Name which is a User defined Type. We can see other types as similar to MySQL.* ***AddressID*** *is the Primary Key.*

***Customer*** *table stores customer details like* ***NameStyle, Title, FirstName, MiddleName, LastName, Suffix, Companyname, SalesPerson, EmailAddress, Phone, PasswordHash, PasswordSalt.*** *The unique identifier* ***rowguid*** *and* ***ModifedDate****(datetime) is also present just like any other tables.* ***MiddleName****,* ***Suffix****,* ***Title****,* ***Phone****,* ***EmailAddress****,* ***CompanyName****,* ***SalesPerson*** *are allowed to be null. We can also see a new datatype called Name is used in* ***FirstName*** *and* ***LastName****. We can see a new User defined datatype* ***NameStyle*** *defined for* ***NameStyle*** *column. We can see other types as similar to MySQL.* ***CustomerID*** *is the Primary Key.* ***PasswordHash*** *and* ***PasswordSalt*** *stores password in encrypted format.*

***CustomerAddress*** *Tables Connect* ***Address*** *Table and* ***Customer*** *Table. We can see both* ***CustomerID*** *and* ***AddressID*** *becomes the primary key here. We can also see an extra column showing the* ***AddressType.***

***Product*** *table stores product details like* ***Name, ProductNumber, Color, StandardCost, ListPrice, Size, Weight, ProductCategoryID, ProductMoelID, SellStartDate, SellEndDate, DiscontinuedDate, ThumbNailPhoto, ThumbnailPhotoFileName****. Columns* ***Color, Size, Weight, ProductCategoryID, ProductModelID, SellEndDate, DiscontinuedDate, ThumbNailPhoto, ThumbNailPhotoFileName*** *are allowed to be null. We can also see a new datatype called money which is a User defined Type for* ***ListPrice*** *column. We can see other types as similar to MySQL.* ***ProductID*** *is the Primary Key. Note that the foreign keys* ***ProductCategoryID, ProductModelID*** *can be null. This indicates that the relationship between these tables (both* ***ProductCategory*** *and* ***ProductModel****) to* ***Product*** *becomes zero or one relationship where product may not have these* ***ProductCategory*** *or* ***ProductModel****.*

***ProductCategory*** *table stores details like* ***ParentProductCategoryID, Name****. Columns* ***ParentProductCategoryID*** *is allowed to be null.* ***ProductCategoryID*** *is the Primary Key. Note that the foreign keys* ***ParentProductCategoryID*** *can be null.* ***ProductCategoryID*** *is a foreign key which is foreign on the same table as it is defined. We can also see this field is nullable and therefore it represents a one or zero relationship with itself. If we look at the data, we see top level category has no Parent while children can have sub child category.*

***ProductDescription*** *table stores details like* ***Description****.* ***ProductDescriptionID*** *is the Primary Key. Looking in details shows that the description can be of different languages.*

***ProductModel*** *table stores details like* ***Name, CatelogDescription****. Columns* ***CatelogDescription*** *is allowed to be null.* ***ProductModelID*** *is the Primary Key.*

***ProductModelProducteDescription*** *table stores relation between* ***ProductModel*** *and* ***ProductDescription****.* ***ProductDescriptionID, ProductModelID*** *and* ***Culture*** *together forms a composite Primary Key. Note that Culture is showing the Language Culture and datatype is nchar.* ***ProductDescriptionID, ProductModelID*** *are foreign keys that connects to table* ***ProductDescription*** *and* ***ProductModel.***

***SalesOrderDetails*** *table stores details like* ***SalesOrderID, OrderQty, ProductID, UnitPrice, UnitPriceDiscount, LineTotal.*** *We can also see a datatype called money is used in* ***UnitPrice*** *and* ***UnitPriceDiscount****. We can see other types as like MySQL. Here* ***ProductID*** *and* ***SalesOrderID*** *are foreign keys corresponding to* ***Product*** *and* ***SalesOrderHeader*** *tables showing one to one relationship.* ***SalesOrderID*** *and* ***SalesOrderDetailID*** *forms the composite primary key.*

***SalesOrderHeader*** *table stores details like* ***RevisionNumber, OrderDate, DueDate, ShipDate, Status, OnlineOrderFlag, SalesOrderNumber, PurchaseOrderNumber, AccountNumber, CustomerID, ShipToAddressID, BillToAddressID, ShipMethod, CreditCardApprovalCode, SubTotal, TaxAmt, Freight, TotalDue, Comment.*** *We can also see a datatype called money is used in* ***Freight, SubTotal, TaxAmt*** *and* ***TotalDue****.* ***AccountNumber****,* ***OrderNumber*** *and* ***Flag*** *also has user-defined datatype corresponding to it. We can see other types as like MySQL.* ***SalesOrderID*** *forms the primary Key in* ***SalesOrderHeader*** *table.* ***CustomerID, BillToAddressID*** *and* ***ShipToAddressID*** *are foreign keys corresponding to* ***Customer*** *and* ***Address*** *tables showing one to one relationship.*

| **Foreign Keys** |
| --- |
| SELECT      fk.name AS ForeignKeyName,      tp.name AS ParentTable,      ref.name AS ReferencedTable,      COL\_NAME(fkc.parent\_object\_id, fkc.parent\_column\_id) AS ParentColumn,      COL\_NAME(fkc.referenced\_object\_id, fkc.referenced\_column\_id) AS ReferencedColumn  FROM      sys.foreign\_keys AS fk  INNER JOIN      sys.foreign\_key\_columns AS fkc      ON fk.object\_id = fkc.constraint\_object\_id  INNER JOIN      sys.tables AS tp      ON fkc.parent\_object\_id = tp.object\_id  INNER JOIN      sys.tables AS ref      ON fkc.referenced\_object\_id = ref.object\_id  ORDER BY      fk.name, tp.name, ref.name; |
|  |
|  |

**Q7. Clean up resources in Azure**

When you're finished using these resources, delete the resource group you created, which will also delete the blob storage, server and single database within it. Show the screenshot that you deleted the resource group.

|  |
| --- |
|  |

**Q8. Summarize the article on "Understanding Data Store Models" . Link -->** [**https://learn.microsoft.com/en-us/azure/architecture/guide/technology-choices/data-store-overview**](https://learn.microsoft.com/en-us/azure/architecture/guide/technology-choices/data-store-overview)

It teaches about different storage models.

1. **Relational Database Management Systems (RDBMS):** Organize data in two-dimensional tables and use ACID principles for transactions. They support relationships, transactions, and indexing to optimize performance. Examples include MySQL and PostgreSQL.

2**. Key/Value Stores**: Store data as key-value pairs with simple atomic operations. They are optimized for quick lookups and are used for caching and session management. Examples include Redis.

3. **Document Databases**: Store data as documents with named fields and support semi-structured data. Documents are retrieved by unique keys, and indexing is used for efficient queries. Examples include MongoDB.

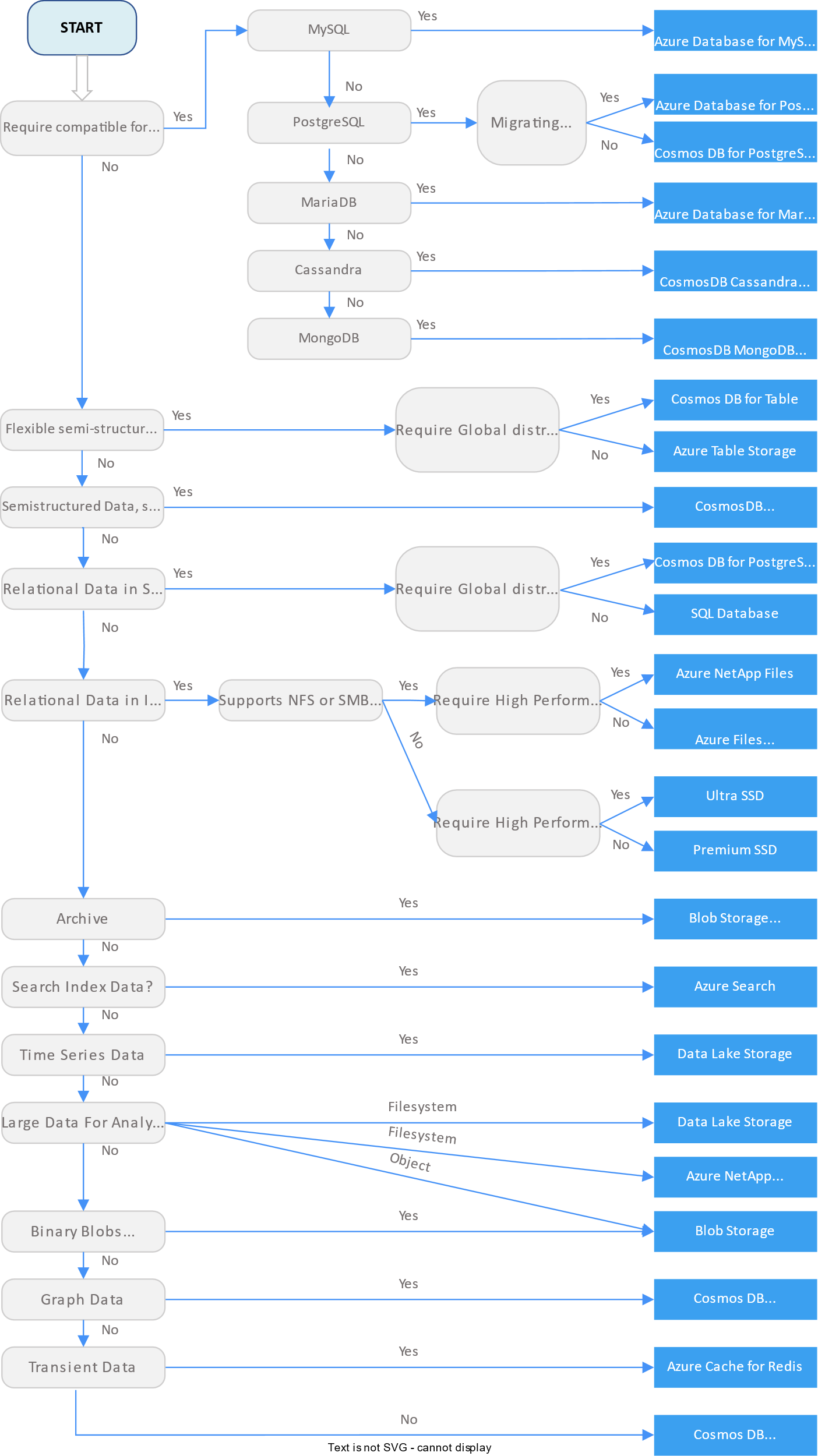
4. **Graph Databases**: Represent data as nodes and edges, modeling complex relationships. They excel in scenarios with dynamic and intricate relationships. Examples include Neo4j.

5. **Data Analytics**: Handle large-scale data processing and analysis, typically in a distributed manner. They use formats like CSV and Parquet and often involve data warehouses with denormalized schemas.

6. **Column-Family Databases:** Organize data into rows and columns but group columns into families. They offer high throughput and are used for applications needing rapid write access. Examples include Apache Cassandra.

7. **Search Engine Databases**: Index large volumes of data for fast and complex searches. They handle semi-structured or unstructured data and are used for site search and indexing. Examples include Elasticsearch.

8. **Time Series Databases**: Manage data organized by time, with a focus on high write volumes and sequential data processing. They are used for IoT and telemetry data. Examples include InfluxDB.



9. **Object Storage**: Optimized for storing large binary objects such as files and videos, identified by unique keys. It is used for storing vast amounts of unstructured data. Examples include Amazon S3.

10. **Shared Files**: Allow files to be accessed over a network with mechanisms for security and concurrent access. Suitable for basic read/write operations across distributed systems.

**Q9. Summarize the article on "Understanding the differences between NoSQL and relational databases". Link -->** [**https://learn.microsoft.com/en-us/azure/cosmos-db/relational-nosql**](https://learn.microsoft.com/en-us/azure/cosmos-db/relational-nosql)

NoSQL databases were designed to simplify horizontal scaling by adjusting consistency to minimize the trade-offs to concurrency, latency, and availability. NoSQL databases offered configurable levels of consistency so that data can scale across many nodes and offer speed or availability that better mapped to the needs of some applications. With a distributed database that is also a NoSQL database, high transactional workloads suddenly became easier to build and manage.

Relational Databases engines apply locks and latches to enforce strict ACID semantics. While this approach promises high consistency, it makes heavy trade-offs with respect to concurrency, latency, and availability. This restriction is fundamentally an architectural restriction and will force any team with a high transactional workload to find workarounds like manually distributing, or sharding, data across many different databases or database nodes.

**Q10. Summarize the article on "Understanding Azure Cosmos DB". Link -->** [**https://learn.microsoft.com/en-us/azure/cosmos-db/introduction**](https://learn.microsoft.com/en-us/azure/cosmos-db/introduction)

Azure Cosmos DB is a versatile, fully managed database service that simplifies application development by supporting a variety of data models including NoSQL, relational, vector, key-value, graph, and table. It caters to modern applications such as AI, digital commerce, IoT, and booking management with features like geo-replication, distributed caching, and vector indexing.

**Key Features:**

* **Performance**: Single-digit millisecond response times, automatic scalability, and guaranteed speed at any scale.
* **Management**: Fully managed with automatic updates, patching, and serverless options for cost-effective scaling.
* **Free Options**: Includes a lifetime free tier, a 90-day throughput offer for Azure AI customers, and a 30-day trial.
* **Flexibility**: Supports various schemas, offers open-source APIs, and integrates with popular programming languages and tools.
* **Reliability**: Provides 99.999% availability, enterprise-grade security, and business continuity with multi-region data distribution and automatic replication.

Best for Flexible schema applications, latency-sensitive workloads, highly elastic and high throughput workloads, and mission-critical applications.

Not Ideal for Analytical workloads requiring OLAP, highly relational apps, and complex analytics scenarios. For these, alternatives like Microsoft Fabric or Azure SQL might be better suited.

It can also be integrated with Azure Synapse Analytics using Azure Synapse Link.

**Q11. Summarize the article on "Azure Cosmos DB API for MongoDB". Link -->** [**https://learn.microsoft.com/en-us/azure/cosmos-db/mongodb/mongodb-introduction**](https://learn.microsoft.com/en-us/azure/cosmos-db/mongodb/mongodb-introduction)

Azure Cosmos DB for MongoDB provides an SLA that covers the full stack: the database and the underlying infrastructure. Cosmos DB for MongoDB implements the wire protocol for MongoDB. This implementation allows transparent compatibility with MongoDB client SDKs, drivers, and tools. Azure Cosmos DB doesn't host the MongoDB database engine. Any MongoDB client driver compatible with the API version you're using should be able to connect, with no special configuration. Azure Cosmos DB for MongoDB provides 2 Architectures - vCore architecture and Request Unit (RU) architecture.

VCore Architecture offers a familiar vCore architecture for MongoDB users, instantaneous scaling, and seamless native integration with Azure services. It has features like Native Vector Search, Instantaneous scalability, Flat pricing with Low total cost of ownership, Elevate querying with Text Indexes, Scale with no shard key required, Free 35 day Backups with point in time restore (PITR).

Request Unit (RU) architecture provides a fully managed MongoDB-compatible service with flexible scaling using Request Units (RUs), designed for cloud-native applications. It offers notable features like Instantaneous scalability, Automatic and transparent sharding, Five 9's of availability, Active-active database, Cost efficient, granular, unlimited scalability, Real time analytics (HTAP) at any scale, Serverless deployments.

**Q12. Summarize the article on "Nodes and tables in Azure Database for PostgreSQL – Hyperscale". Link -->** [**https://learn.microsoft.com/en-us/azure/postgresql/hyperscale/concepts-nodes**](https://learn.microsoft.com/en-us/azure/postgresql/hyperscale/concepts-nodes)

Azure Cosmos DB for PostgreSQL employs a distributed architecture to scale and manage data efficiently across multiple nodes. The architecture consists of Nodes - Coordinator Node which manages query routing and results aggregation and Worker Nodes which stores and process data.

It has a number of table types.

**Distributed Tables**: Horizontally partitioned across worker nodes. Rows are distributed based on a chosen column (distribution column). Schema changes are applied cluster-wide. The cluster administrator must designate this column when distributing a table. Making the right choice is important for performance and functionality.

**Reference Tables**: Contained in a single shard replicated across all workers. Useful for small, frequently accessed data like status lists.

**Local Tables**: Stored only on the coordinator node, suitable for small, non-joined tables like user authentication data.

**Local Managed Tables**: Created automatically or manually to handle foreign key references to reference tables. Accessible from any node.

**Schema Tables**: Introduced in Citus 12.0, these are tables in schemas that are automatically distributed without a shard key, shown as schema tables in the citus\_tables view.

**Q13. Summarize the article on "Overview - Azure Database for PostgreSQL - Flexible Server". Link -->** [**https://learn.microsoft.com/en-us/azure/postgresql/flexible-server/overview**](https://learn.microsoft.com/en-us/azure/postgresql/flexible-server/overview)

Azure Database for PostgreSQL flexible server is a fully managed database service designed to provide more granular control and flexibility over database management functions and configuration settings. It supports high availability within and across availability zones with synchronous replication for data durability. Users can customize maintenance schedules and benefit from automatic backups stored in zone redundant storage. It offers three compute tiers: Burstable, General Purpose, and Memory Optimized, catering to different workload needs. The service supports on-demand server management to reduce costs and employs FIPS 140-2 validated encryption for data security. It includes PgBouncer for efficient connection pooling, ensuring easy migration from existing PostgreSQL setups.

**Q14. Summarize the article on "Azure Database for PostgreSQL - Single Server". Link-->** [**https://learn.microsoft.com/en-us/azure/postgresql/single-server/concepts-servers**](https://learn.microsoft.com/en-us/azure/postgresql/single-server/concepts-servers)

An Azure Database for PostgreSQL - Single Server acts as a central administrative point for multiple databases within an Azure subscription. It provides a namespace, connection endpoint, and management scope for its databases. It supports various PostgreSQL versions and extensions, and includes security features like authentication, authorization,TCP/IP and firewall controls. When a server is deleted, its contained databases are also removed. Configurable parameters in this service are a subset of those in a local Postgres instance. As Azure Database for PostgreSQL - Single Server is being retired, users are advised to upgrade to Azure Database for PostgreSQL - Flexible Server.The pricing is structured per-server, based on the configuration of pricing tier, vCores, and storage (GB).

**Q15. Summarize the article on "What is Azure Database for PostgreSQL?". Link -->** [**https://learn.microsoft.com/en-us/azure/postgresql/single-server/overview**](https://learn.microsoft.com/en-us/azure/postgresql/single-server/overview)

Azure Database for Postgresql has two modes. Flexible server and single server. Flexible server is recommended to use. Azure Database for PostgreSQL flexible server is a fully managed database service designed to provide more granular control and flexibility over database management functions and configuration settings. Azure Database for Postgresql single server is the legacy version which acts as a single administrative point for multiple databases. It is not as configurable as a flexible server. Single server is designed for minimal customization.

**Q16. Summarize the article on "Azure Database for MySQL - Flexible Server". Link -->** [**https://learn.microsoft.com/en-us/azure/mysql/flexible-server/overview**](https://learn.microsoft.com/en-us/azure/mysql/flexible-server/overview)

Azure Database for MySQL Flexible Server is a fully managed relational database service based on MySQL Community Edition, providing high control and flexibility over database management. Here’s a summary of its key features and benefits:

**Key Features:**

1. **High Availability:**
   1. **Zone Redundant HA**: Ensures infrastructure redundancy across multiple availability zones.4
   2. **Same-Zone HA**: Provides high availability within a single zone with lower latency.
2. **Automated Management**:
   1. **Patching and Maintenance**: Automated updates for hardware, OS, and database engine with customizable maintenance windows.
   2. **Backups**: Automatic backups with retention options between 1 and 35 days, encrypted with AES 256-bit encryption.
3. **Performance and Scalability**
   1. **Compute Tiers**: **Burstable, General Purpose, and Business Critical** tiers for various performance needs. The storage scaling is online and supports storage autogrowth. Azure Database for MySQL flexible server enables you to provision additional IOPS up to 80 K IOPs above the complimentary IOPS limit independent of storage. Using this feature, you can increase or decrease the number of IOPS provisioned based on your workload requirements at any time. Dynamic scalability enables your database to respond to rapidly changing resource requirements transparently.
4. **Read Replicas**:
   1. Support for up to 10 read replicas for scaling read workloads with asynchronous replication.
5. **Cost Optimization**
   1. **Stop/Start Server**: Option to stop and start servers to save costs during non-usage periods.
   2. **Reserved Instances**: Up to 63% cost savings with reserved instances for predictable workloads.
6. **Security and Compliance**:
   1. **Data Encryption**: AES 256-bit encryption for data at rest and TLS 1.2 for data in transit.
   2. **Network Isolation**: Options for private access through Azure Virtual Network or public access with allowed IPs.
7. **Monitoring and Alerts**:
   1. Built-in performance monitoring, alerting, and integration with Azure Monitor workbooks for visualizing performance metrics.
8. **Migration**
   1. **Tools and Methods**: Support for offline migrations using Azure Data Migration Service and mydumper/myloader, and online migrations for minimal downtime.
9. **Data Synchronization**: **Data-in Replication** Allows synchronization of data from external MySQL servers into Azure Database for MySQL Flexible Server. This can include on-premises servers, Azure Database for MySQL Single Server, or databases hosted by other cloud providers.
   * 1. **Hybrid Data Synchronization**: Facilitates synchronization between on-premises MySQL databases and Azure Database for MySQL Flexible Server.
     2. **Multicloud Synchronization**: Enables data replication between MySQL databases across different cloud providers.
     3. **Minimal Downtime Migration**: Supports minimal downtime migration from external MySQL databases to Azure Database for MySQL Flexible Server using position-based binary log replication.

**Best Suited For**

* Applications requiring a community MySQL version with high availability, security, and scalability.
* Development, production workloads, and enterprises needing minimal database management overhead and optimized performance

Azure Database for MySQL Flexible Server simplifies database management, enabling focus on app development while providing enterprise-grade security, performance, and cost efficiency.

**Q17. Summarize the article on "Azure Database for MySQL Single Server". Link -->** [**https://learn.microsoft.com/en-us/azure/mysql/single-server/single-server-overview**](https://learn.microsoft.com/en-us/azure/mysql/single-server/single-server-overview)

Azure Database for MySQL Single Server is a fully managed database service designed for minimal configuration, offering automatic management of patching, backups, and high availability with a 99.99% uptime guarantee. It supports MySQL versions 5.6 (retired), 5.7, and 8.0 and is available in many Azure regions.

**Key Features**:

* **High Availability**: Utilizes a proprietary compute container and Azure storage with three synchronous copies to ensure data durability and maintain availability during failovers, with typical recovery times of 60-120 seconds.
* **Automated Patching**: Handles hardware, OS, and MySQL engine updates automatically, with minor upgrades included. Users are notified of planned maintenance 72 hours in advance.
* **Automatic Backups**: Creates and stores encrypted backups, configurable from 7 to 35 days, with encryption using AES 256-bit.
* **Performance and Scalability**: Available in Basic, General Purpose, and Memory Optimized tiers. Supports online storage scaling and dynamic resource adjustments.
* **Security and Compliance**: Uses FIPS 140-2 validated encryption for data at-rest and TLS for data in-motion. Supports private access via private link and integrates with Microsoft Defender for threat protection. Compliance with FedRAMP, HIPAA, and PCI DSS.
* **Monitoring and Alerting**: Provides built-in performance monitoring, alerting, slow query logs, and Query Store for troubleshooting and optimization.
* **Migration**: Compatible with community MySQL versions and supports migration via dump and restore, Azure Database Migration Service, or data-in replication for minimal downtime.

For new developments or migrations, the Flexible Server deployment option is recommended over Single Server.