Study Guide for Exam DP-420: Designing and Implementing Native Applications Using Microsoft Azure Cosmos DB

Purpose of this document

This study guide should help you understand what to expect on the exam and includes a summary of the topics the exam might cover and links to additional resources. The information and materials in this document should help you focus your studies as you prepare for the exam.

Useful links	Description
Review the skills measured as of February 6, 2023	This list represents the skills measured AFTER the date provided. Study this list if you plan to take the exam AFTER that date.
Review the skills measured prior to February 6, 2023	Study this list of skills if you take your exam PRIOR to the date provided.
Change log	You can go directly to the change log if you want to see the changes that will be made on the date provided.
How to earn the certification	Some certifications only require passing one exam, while others require passing multiple exams.
Certification renewal	Microsoft associate, expert, and specialty certifications expire annually. You can renew by passing a **free** online assessment on Microsoft Learn.
Your Microsoft Learn profile	Connecting your certification profile to Learn allows you to schedule and renew exams and share and print certificates.
Passing score	A score of 700 or greater is required to pass.
Exam sandbox	You can explore the exam environment by visiting our exam sandbox.

Useful links	Description
Request accommodations	If you use assistive devices, require extra time, or need modification to any part of the exam experience, you can request an accommodation.
Take a practice test	Are you ready to take the exam or do you need to study a bit more?

Updates to the exam

Our exams are updated periodically to reflect skills that are required to perform a role. We have included two versions of the Skills Measured objectives depending on when you are taking the exam.

We always update the English language version of the exam first. Some exams are localized into other languages, and those are updated approximately eight weeks after the English version is updated. Other available languages are listed in the **Schedule Exam** section of the **Exam Details** webpage. If the exam isn't available in your preferred language, you can request an additional 30 minutes to complete the exam.

Note

The bullets that follow each of the skills measured are intended to illustrate how we are assessing that skill. Related topics may be covered in the exam.

Note

Most questions cover features that are general availability (GA). The exam may contain questions on Preview features if those features are commonly used.

Skills measured as of February 6, 2023

Audience profile

A candidate for this exam should have subject matter expertise designing, implementing, and monitoring cloud-native applications that store and manage data.

Responsibilities for this role include designing and implementing data models and data distribution, loading data into an Azure Cosmos DB database, and optimizing and maintaining the solution. These professionals integrate the solution with other Azure services. They also design, implement, and monitor solutions that consider security, availability, resilience, and performance requirements.

A candidate for this exam must have solid knowledge and experience developing apps for Azure and working with Azure Cosmos DB database technologies. They should be proficient at developing applications that use the API for Azure Cosmos DB for NoSQL. They should be able to write efficient SQL queries for the API and be able to create appropriate index policies. They should have experience creating server-side objects with JavaScript. Additionally, they should be familiar with provisioning and

managing resources in Azure. They should be able to interpret JSON, read C# or Java code, and use PowerShell.

- Design and implement data models (35–40%)
- Design and implement data distribution (5–10%)
- Integrate an Azure Cosmos DB solution (5–10%)
- Optimize an Azure Cosmos DB solution (15–20%)
- Maintain an Azure Cosmos DB solution (25–30%)

Design and implement data models (35–40%)

Design and implement a non-relational data model for Azure Cosmos DB for NoSQL

- Develop a design by storing multiple entity types in the same container
- Develop a design by storing multiple related entities in the same document
- Develop a model that denormalizes data across documents
- Develop a design by referencing between documents
- Identify primary and unique keys
- Identify data and associated access patterns
- Specify a default TTL on a container for a transactional store

Design a data partitioning strategy for Azure Cosmos DB for NoSQL

- Choose a partitioning strategy based on a specific workload
- Choose a partition key
- Plan for transactions when choosing a partition key
- Evaluate the cost of using a cross-partition query
- Calculate and evaluate data distribution based on partition key selection
- Calculate and evaluate throughput distribution based on partition key selection
- Construct and implement a synthetic partition key
- Design partitioning for workloads that require multiple partition keys

Plan and implement sizing and scaling for a database created with Azure Cosmos DB

- Evaluate the throughput and data storage requirements for a specific workload
- Choose between serverless and provisioned models
- Choose when to use database-level provisioned throughput
- Design for granular scale units and resource governance
- Evaluate the cost of the global distribution of data
- Configure throughput for Azure Cosmos DB by using the Azure portal

Implement client connectivity options in the Azure Cosmos DB SDK

- Choose a connectivity mode (gateway versus direct)
- Implement a connectivity mode
- Create a connection to a database
- Enable offline development by using the Azure Cosmos DB emulator
- Handle connection errors
- Implement a singleton for the client
- Specify a region for global distribution
- Configure client-side threading and parallelism options
- Enable SDK logging

Implement data access by using the SQL language for Azure Cosmos DB for NoSQL

- Implement queries that use arrays, nested objects, aggregation, and ordering
- Implement a correlated subquery
- Implement queries that use array and type-checking functions
- Implement queries that use mathematical, string, and date functions
- Implement queries based on variable data

Implement data access by using Azure Cosmos DB for NoSQL SDKs

- Choose when to use a point operation versus a query operation
- Implement a point operation that creates, updates, and deletes documents
- Implement an update by using a patch operation
- Manage multi-document transactions using SDK Transactional Batch
- Perform a multi-document load using Bulk Support in the SDK
- Implement optimistic concurrency control using ETags
- Override default consistency by using query request options
- Implement session consistency by using session tokens
- Implement a query operation that includes pagination
- Implement a query operation by using a continuation token
- Handle transient errors and 429s
- Specify TTL for a document
- Retrieve and use query metrics

Implement server-side programming in Azure Cosmos DB for NoSQL by using JavaScript

- Write, deploy, and call a stored procedure
- Design stored procedures to work with multiple documents transactionally
- Implement and call triggers

- Implement a user-defined function

Design and implement data distribution (5–10%)

Design and implement a replication strategy for Azure Cosmos DB

- Choose when to distribute data
- Define automatic failover policies for regional failure for Azure Cosmos DB for NoSQL
- Perform manual failovers to move single master write regions
- Choose a consistency model
- Identify use cases for different consistency models
- Evaluate the impact of consistency model choices on availability and associated RU cost
- Evaluate the impact of consistency model choices on performance and latency
- Specify application connections to replicated data

Design and implement multi-region write

- Choose when to use multi-region write
- Implement multi-region write
- Implement a custom conflict resolution policy for Azure Cosmos DB for NoSQL

Integrate an Azure Cosmos DB solution (5–10%)

Enable Azure Cosmos DB analytical workloads

- Enable Azure Synapse Link
- Choose between Azure Synapse Link and Spark Connector
- Enable the analytical store on a container
- Enable a connection to an analytical store and query from Azure Synapse Spark or Azure Synapse SQL
- Perform a query against the transactional store from Spark
- Write data back to the transactional store from Spark

Implement solutions across services

- Integrate events with other applications by using Azure Functions and Azure Event Hubs
- Denormalize data by using Change Feed and Azure Functions
- Enforce referential integrity by using Change Feed and Azure Functions
- Aggregate data by using Change Feed and Azure Functions, including reporting
- Archive data by using Change Feed and Azure Functions
- Implement Azure Cognitive Search for an Azure Cosmos DB solution

Optimize an Azure Cosmos DB solution (15–20%)

Optimize query performance when using the API for Azure Cosmos DB for NoSQL

- Adjust indexes on the database
- Calculate the cost of the query
- Retrieve request unit cost of a point operation or query
- Implement Azure Cosmos DB integrated cache

Design and implement change feeds for Azure Cosmos DB for NoSQL

- Develop an Azure Functions trigger to process a change feed
- Consume a change feed from within an application by using the SDK
- Manage the number of change feed instances by using the change feed estimator
- Implement denormalization by using a change feed
- Implement referential enforcement by using a change feed
- Implement aggregation persistence by using a change feed
- Implement data archiving by using a change feed

Define and implement an indexing strategy for Azure Cosmos DB for NoSQL

- Choose when to use a read-heavy versus write-heavy index strategy
- Choose an appropriate index type
- Configure a custom indexing policy by using the Azure portal
- Implement a composite index
- Optimize index performance

Maintain an Azure Cosmos DB solution (25–30%)

Monitor and troubleshoot an Azure Cosmos DB solution

- Evaluate response status code and failure metrics
- Monitor metrics for normalized throughput usage by using Azure Monitor
- Monitor server-side latency metrics by using Azure Monitor
- Monitor data replication in relation to latency and availability
- Configure Azure Monitor alerts for Azure Cosmos DB
- Implement and query Azure Cosmos DB logs
- Monitor throughput across partitions
- Monitor distribution of data across partitions
- Monitor security by using logging and auditing

Implement backup and restore for an Azure Cosmos DB solution

- Choose between periodic and continuous backup
- Configure periodic backup

- Configure continuous backup and recovery
- Locate a recovery point for a point-in-time recovery
- Recover a database or container from a recovery point

Implement security for an Azure Cosmos DB solution

- Choose between service-managed and customer-managed encryption keys
- Configure network-level access control for Azure Cosmos DB
- Configure data encryption for Azure Cosmos DB
- Manage control plane access to Azure Cosmos DB by using Azure role-based access control (RBAC)
- Manage data plane access to Azure Cosmos DB by using keys
- Manage data plane access to Azure Cosmos DB by using Microsoft Azure Active Directory (Azure AD)
- Configure Cross-Origin Resource Sharing (CORS) settings
- Manage account keys by using Azure Key Vault
- Implement customer-managed keys for encryption
- Implement Always Encrypted

Implement data movement for an Azure Cosmos DB solution

- Choose a data movement strategy
- Move data by using client SDK bulk operations
- Move data by using Azure Data Factory and Azure Synapse pipelines
- Move data by using a Kafka connector
- Move data by using Azure Stream Analytics
- Move data by using the Azure Cosmos DB Spark Connector

Implement a DevOps process for an Azure Cosmos DB solution

- Choose when to use declarative versus imperative operations
- Provision and manage Azure Cosmos DB resources by using Azure Resource Manager templates (ARM templates)
- Migrate between standard and autoscale throughput by using PowerShell or Azure CLI
- Initiate a regional failover by using PowerShell or Azure CLI
- Maintain index policies in production by using ARM templates

Study resources

We recommend that you train and get hands-on experience before you take the exam. We offer self-study options and classroom training as well as links to documentation, community sites, and videos.

Study resources	Links to learning and documentation
Get trained	Choose from self-paced learning paths and modules or take an instructor-led course
Find documentation	Azure Cosmos DB documentation Azure documentation
Ask a question	Microsoft Q&A | Microsoft Docs
Get community support	Analytics on Azure - Microsoft Tech Community Azure Data Factory - Microsoft Tech Community Azure - Microsoft Tech Community
Follow Microsoft Learn	Microsoft Learn - Microsoft Tech Community
Find a video	Exam Readiness Zone Data Exposed Browse other Microsoft Learn shows

Change log

Key to understanding the table: The topic groups (also known as functional groups) are in bold typeface followed by the objectives within each group. The table is a comparison between the two versions of the exam skills measured and the third column describes the extent of the changes.

Skill area prior to February 6, 2023	Skill area as of February 6, 2023	Change
Audience profile	Audience profile	No change
Design and implement data models	**Design and implement data models**	No change
Design and implement a non-relational data model for Cosmos DB Core API	Design and implement a non-relational data model for Azure Cosmos DB for NoSQL	Minor
Design a data partitioning strategy for Cosmos DB Core API	Design a data partitioning strategy for Azure Cosmos DB for NoSQL	Minor
Plan and implement sizing and scaling for a Cosmos DB	Plan and implement sizing and scaling for an Azure Cosmos DB	Minor
Implement client connectivity options in the Cosmos DB SDK	Implement client connectivity options in the Azure Cosmos DB SDK	Minor

Skill area prior to February 6, 2023	Skill area as of February 6, 2023	Change
Implement data access by using the Cosmos DB SQL language	Implement data access by using the Azure Cosmos DB SQL language	Minor
Implement data access by using SQL API SDKs	Implement data access by using SQL API SDKs	No change
Implement server-side programming in Cosmos DB Core API by using JavaScript	Implement server-side programming in Azure Cosmos DB for NoSQL by using JavaScript	Minor
Design and implement data distribution	**Design and implement data distribution**	No change
Design and implement a replication strategy for Cosmos DB	Design and implement a replication strategy for Azure Cosmos DB	Minor
Design and implement multi-region write	Design and implement multi-region write	Minor
Integrate a Cosmos DB solution	**Integrate an Azure Cosmos DB solution**	Minor
Enable Cosmos DB analytical workloads	Enable Azure Cosmos DB analytical workloads	Minor
Implement solutions across services	Implement solutions across services	No change
Optimize a Cosmos DB solution	**Optimize an Azure Cosmos DB solution**	Minor
Optimize query performance in Cosmos DB Core API	Optimize query performance in Azure Cosmos DB for NoSQL	Minor
Design and implement change feeds for a Cosmos DB Core API	Design and implement change feeds for a Azure Cosmos DB for NoSQL	Minor
Define and implement an indexing strategy for a Cosmos DB Core API	Define and implement an indexing strategy for a Azure Cosmos DB for NoSQL	Minor
Maintain a Cosmos DB solution	**Maintain an Azure Cosmos DB solution**	Minor
Monitor and troubleshoot a Cosmos DB solution	Monitor and troubleshoot an Azure Cosmos DB solution	Minor
Implement backup and restore for a Cosmos DB solution	Implement backup and restore for an Azure Cosmos DB solution	Minor

Skill area prior to February 6, 2023	Skill area as of February 6, 2023	Change
Implement security for a Cosmos DB solution	Implement security for an Azure Cosmos DB solution	Minor
Implement data movement for a Cosmos DB solution	Implement data movement for an Azure Cosmos DB solution	Minor
Implement a DevOps process for a Cosmos DB solution	Implement a DevOps process for an Azure Cosmos DB solution	Minor

Skills measured prior to February 6, 2023

- Design and implement data models (35–40%)
- Design and implement data distribution (5–10%)
- Integrate an Azure Cosmos DB solution (5–10%)
- Optimize an Azure Cosmos DB solution (15–20%)
- Maintain an Azure Cosmos DB solution (25–30%)

Design and implement data models (35–40%)

Design and implement a non-relational data model for Azure Cosmos DB for NoSQL

- Develop a design by storing multiple entity types in the same container
- Develop a design by storing multiple related entities in the same document
- Develop a model that denormalizes data across documents
- Develop a design by referencing between documents
- Identify primary and unique keys
- Identify data and associated access patterns
- Specify a default TTL on a container for a transactional store

Design a data partitioning strategy for Azure Cosmos DB for NoSQL

- Choose a partitioning strategy based on a specific workload
- Choose a partition key
- Plan for transactions when choosing a partition key
- Evaluate the cost of using a cross-partition query
- Calculate and evaluate data distribution based on partition key selection
- Calculate and evaluate throughput distribution based on partition key selection
- Construct and implement a synthetic partition key
- Design partitioning for workloads that require multiple partition keys

Plan and implement sizing and scaling for a database created with Azure Cosmos DB

- Evaluate the throughput and data storage requirements for a specific workload
- Choose between serverless and provisioned models
- Choose when to use database-level provisioned throughput
- Design for granular scale units and resource governance
- Evaluate the cost of the global distribution of data
- Configure throughput for Azure Cosmos DB by using the Azure portal

Implement client connectivity options in the Azure Cosmos DB SDK

- Choose a connectivity mode (gateway versus direct)
- Implement a connectivity mode
- Create a connection to a database
- Enable offline development by using the Azure Cosmos DB emulator
- Handle connection errors
- Implement a singleton for the client
- Specify a region for global distribution
- Configure client-side threading and parallelism options
- Enable SDK logging

Implement data access by using the SQL language for Azure Cosmos DB for NoSQL

- Implement queries that use arrays, nested objects, aggregation, and ordering
- Implement a correlated subquery
- Implement queries that use array and type-checking functions
- Implement queries that use mathematical, string, and date functions
- Implement queries based on variable data

Implement data access by using Azure Cosmos DB for NoSQL SDKs

- Choose when to use a point operation versus a query operation
- Implement a point operation that creates, updates, and deletes documents
- Implement an update by using a patch operation
- Manage multi-document transactions using SDK Transactional Batch
- Perform a multi-document load using Bulk Support in the SDK
- Implement optimistic concurrency control using ETags
- Override default consistency by using query request options
- Implement session consistency by using session tokens
- Implement a query operation that includes pagination
- Implement a query operation by using a continuation token

- Handle transient errors and 429s
- Specify TTL for a document
- Retrieve and use query metrics

Implement server-side programming in Azure Cosmos DB for NoSQL by using JavaScript

- Write, deploy, and call a stored procedure
- Design stored procedures to work with multiple items transactionally
- Implement and call triggers
- Implement a user-defined function

Design and implement data distribution (5–10%)

Design and implement a replication strategy for Azure Cosmos DB

- Choose when to distribute data
- Define automatic failover policies for regional failure for Azure Cosmos DB Core API
- Perform manual failovers to move single master write regions
- Choose a consistency model
- Identify use cases for different consistency models
- Evaluate the impact of consistency model choices on availability and associated RU cost
- Evaluate the impact of consistency model choices on performance and latency
- Specify application connections to replicated data

Design and implement multi-region write

- Choose when to use multi-region write
- Implement multi-region write
- Implement a custom conflict resolution policy for Azure Cosmos DB Core API

Integrate an Azure Cosmos DB solution (5–10%)

Enable Azure Cosmos DB analytical workloads

- Enable Azure Synapse Link
- Choose between Azure Synapse Link and Spark Connector
- Enable the analytical store on a container
- Enable a connection to an analytical store and query from Azure Synapse Spark or Azure Synapse SQL
- Perform a query against the transactional store from Spark
- Write data back to the transactional store from Spark

Implement solutions across services

- Integrate events with other applications by using Azure Functions and Azure Event Hubs
- Denormalize data by using Change Feed and Azure Functions

- Enforce referential integrity by using Change Feed and Azure Functions
- Aggregate data by using Change Feed and Azure Functions, including reporting
- Archive data by using Change Feed and Azure Functions
- Implement Azure Cognitive Search for an Azure Cosmos DB solution

Optimize an Azure Cosmos DB solution (15–20%)

Optimize query performance when using the API for Azure Cosmos DB for NoSQL

- Adjust indexes on the database
- Calculate the cost of the query
- Retrieve request unit cost of a point operation or query
- Implement Azure Cosmos DB integrated cache

Design and implement change feeds for Azure Cosmos DB for NoSQL

- Develop an Azure Functions trigger to process a change feed
- Consume a change feed from within an application by using the SDK
- Manage the number of change feed instances by using the change feed estimator
- Implement denormalization by using a change feed
- Implement referential enforcement by using a change feed
- Implement aggregation persistence by using a change feed
- Implement data archiving by using a change feed

Define and implement an indexing strategy for Azure Cosmos DB for NoSQL

- Choose when to use a read-heavy versus write-heavy index strategy
- Choose an appropriate index type
- Configure a custom indexing policy by using the Azure portal
- Implement a composite index
- Optimize index performance

Maintain an Azure Cosmos DB solution (25–30%)

Monitor and troubleshoot an Azure Cosmos DB solution

- Evaluate response status code and failure metrics
- Monitor metrics for normalized throughput usage by using Azure Monitor
- Monitor server-side latency metrics by using Azure Monitor
- Monitor data replication in relation to latency and availability
- Configure Azure Monitor alerts for Azure Cosmos DB
- Implement and query Azure Cosmos DB logs
- Monitor throughput across partitions
- Monitor distribution of data across partitions

- Monitor security by using logging and auditing

Implement backup and restore for an Azure Cosmos DB solution

- Choose between periodic and continuous backup
- Configure periodic backup
- Configure continuous backup and recovery
- Locate a recovery point for a point-in-time recovery
- Recover a database or container from a recovery point

Implement security for an Azure Cosmos DB solution

- Choose between service-managed and customer-managed encryption keys
- Configure network-level access control for Azure Cosmos DB
- Configure data encryption for Azure Cosmos DB
- Manage control plane access to Azure Cosmos DB by using Azure role-based access control (RBAC)
- Manage data plane access to Azure Cosmos DB by using keys
- Manage data plane access to Azure Cosmos DB by using Microsoft Azure Active Directory (Azure AD), part of Microsoft Entra
- Configure Cross-Origin Resource Sharing (CORS) settings
- Manage account keys by using Azure Key Vault
- Implement customer-managed keys for encryption
- Implement Always Encrypted

Implement data movement for an Azure Cosmos DB solution

- Choose a data movement strategy
- Move data by using client SDK bulk operations
- Move data by using Azure Data Factory and Azure Synapse pipelines
- Move data by using a Kafka connector
- Move data by using Azure Stream Analytics
- Move data by using the Azure Cosmos DB Spark Connector

Implement a DevOps process for an Azure Cosmos DB solution

- Choose when to use declarative versus imperative operations
- Provision and manage Azure Cosmos DB resources by using Azure Resource Manager templates (ARM templates)
- Migrate between standard and autoscale throughput by using PowerShell or Azure CLI
- Initiate a regional failover by using PowerShell or Azure CLI
- Maintain index policies in production by using ARM templates