2. Develop for Azure Storage -> 2.1 Develop solutions that use Azure Cosmos DB -> 2.1.2 Set the appropriate consistency level for operations

1. What are the consistency levels supported by Azure Cosmos DB?
2. What is the default consistency level and why is it recommended?
3. How do you configure consistency level at the account level?
4. How do you override the consistency level per request?
5. What is session consistency and when should it be used?
6. What are the trade-offs between strong and eventual consistency?
7. How does consistency affect performance and availability?
8. How do you check the current consistency level of an account?
9. Which operations are affected by the chosen consistency level?
10. What are best practices for setting consistency levels in real-world applications?

---

1. What are the consistency levels supported by Azure Cosmos DB?

- **Strong**
- **BoundedStaleness**
- **Session** *(default)*
- **ConsistentPrefix**
- **Eventual**

---

2. What is the default consistency level and why is it recommended?

- **Session** is default.
- Guarantees *read-your-own-writes* within a session.
- Balanced choice for consistency and performance.

---

3. How do you configure consistency level at the account level?

- Set during Cosmos DB account creation or via Azure Portal:
  - *Settings → Default consistency*
- Or using SDK:

```
CosmosClientOptions.ConsistencyLevel = ConsistencyLevel.Session;
```

---

4. How do you override the consistency level per request?

```
var requestOptions = new QueryRequestOptions
{
   ConsistencyLevel = ConsistencyLevel.Eventual
};
```

- Applies only to the specific request.
- Must be equal or weaker than the account-level setting.

---

5. What is session consistency and when should it be used?

- Guarantees *read-your-own-writes* for a session token.
- Ideal for user-specific data scenarios (e.g., profile updates, shopping carts).

6. What are the trade-offs between strong and eventual consistency?

- **Strong**: Highest data accuracy, lowest availability across regions.
- **Eventual**: Best performance and availability, but stale reads are possible.

---

7. How does consistency affect performance and availability?

- Weaker levels (Eventual, ConsistentPrefix) offer lower latency and higher throughput.
- Stronger levels (Strong, BoundedStaleness) increase latency and reduce write availability in multi-region setups.

---

8. How do you check the current consistency level of an account?

- Use Azure Portal → *Settings* → *Default consistency*
- Or SDK:
  ```
  var consistency = cosmosClient.ClientOptions.ConsistencyLevel;
  ```

---

9. Which operations are affected by the chosen consistency level?

- **Read operations**: The chosen level impacts how up-to-date the reads are.
- **Write operations** are always consistent.

---

10. What are best practices for setting consistency levels in real-world applications?

- Use **Session** for most app scenarios (low latency + strong enough).
- Use **Strong** only when global read consistency is critical.
- Use **Eventual** or **ConsistentPrefix** for high-throughput, read-heavy apps where data freshness is not critical.