**ACID in System Design - Detailed Explanation**

**ACID** (Atomicity, Consistency, Isolation, Durability) is a set of **properties that ensure reliable database transactions**. It is essential in relational databases (SQL) to **maintain data integrity** and prevent issues like data corruption or loss.

**🔹 What is ACID?**

| **Property** | **Explanation** | **Example** |
| --- | --- | --- |
| **A - Atomicity** | A transaction is **all or nothing**. If one part fails, the entire transaction is rolled back. | ✅ **Money Transfer**: If a user transfers $100 from Account A to Account B, both **debit and credit must succeed**; otherwise, the transaction is canceled. |
| **C - Consistency** | The database **remains in a valid state** before and after a transaction. No partial updates allowed. | ✅ **Bank Balance Check**: A user cannot withdraw money if the balance is insufficient. |
| **I - Isolation** | Concurrent transactions do not interfere with each other. | ✅ **Two users booking the same flight seat**: The database ensures only one booking succeeds. |
| **D - Durability** | Once a transaction is committed, it is permanently stored—even if the system crashes. | ✅ **Order Confirmation**: If an order is placed, it **remains confirmed** even if the server goes down. |

**🔹 ACID Explained with a Real-World Example: Banking System**

Imagine a banking system where **User A transfers $100 to User B**.

**1️⃣ Step 1: Begin Transaction**

* User A wants to transfer **$100** to User B.

**2️⃣ Step 2: Deduct Money from User A (Atomicity)**

* **SQL Query:**

sql

CopyEdit

UPDATE accounts SET balance = balance - 100 WHERE user\_id = 'A';

* If this **fails**, the entire transaction is rolled back.

**3️⃣ Step 3: Add Money to User B (Consistency)**

* **SQL Query:**

sql

CopyEdit

UPDATE accounts SET balance = balance + 100 WHERE user\_id = 'B';

* The **total money in the system must remain unchanged** (Consistency).

**4️⃣ Step 4: Ensure Isolation**

* If another user tries to withdraw money at the same time, it **waits until this transaction completes** (Isolation).

**5️⃣ Step 5: Commit Transaction (Durability)**

* **SQL Query:**

sql

CopyEdit

COMMIT;

* If the transaction **is committed, it is permanently stored**, even if the server crashes.

✅ **Final State:** Money is successfully transferred without any inconsistency.

**🔹 When to Use ACID Transactions?**

✅ **1. When Data Integrity is Critical**

* **Example**: Banking, financial transactions, healthcare records.

✅ **2. When You Need Strong Consistency**

* **Example**: E-commerce (inventory management), airline ticket booking.

✅ **3. When Data Loss is Not Acceptable**

* **Example**: Order processing, tax records, and financial auditing.

**🔹 When NOT to Use ACID?**

❌ **1. When Performance & Scalability Matter More**

* ACID transactions **can slow down large-scale distributed systems**.
* **Example**: High-speed messaging apps (WhatsApp, Twitter).

❌ **2. When You Need Eventual Consistency**

* **Example**: NoSQL databases like MongoDB, Cassandra, and DynamoDB **prioritize availability** over strict ACID compliance.

❌ **3. When Read Performance is More Important**

* In big data applications, ACID can be an **overhead** because data is mostly read rather than written.
* **Example**: Logging systems, analytics dashboards.

**🔹 SQL vs NoSQL in ACID Compliance**

| **Feature** | **SQL (Relational Databases)** | **NoSQL (Non-Relational Databases)** |
| --- | --- | --- |
| **ACID Compliance** | ✅ Fully supports ACID transactions | ❌ Limited ACID support (except some like MongoDB with transactions) |
| **Performance** | 🚀 Slower due to strict consistency | ⚡ Faster due to relaxed consistency |
| **Scalability** | ❌ Harder to scale horizontally | ✅ Easier to scale across multiple nodes |
| **Example Databases** | MySQL, PostgreSQL, Oracle, SQL Server | MongoDB, Cassandra, DynamoDB |

**🔹 Real-World Examples of ACID Usage**

1️⃣ **Banking Systems (MySQL, PostgreSQL, Oracle)** → Ensure money transfers **do not fail midway**.  
2️⃣ **E-commerce (SQL Server, PostgreSQL)** → Ensures **orders are processed correctly**.  
3️⃣ **Flight Booking (MySQL, Oracle)** → Prevents **duplicate seat reservations**.

**🔹 Final Takeaways**

✔ **ACID ensures data reliability, making it essential for critical applications.**  
✔ **SQL databases fully support ACID, while NoSQL databases trade off consistency for scalability.**  
✔ **Use ACID when strict consistency and integrity are required (e.g., finance, healthcare).**  
✔ **Avoid ACID when performance and scalability are more important (e.g., NoSQL-based systems).**