




# Online transaction processing (OLTP)

02/12/2018 • 6 minutes to read • Contributors     

## In this article

[Transactional data](#)

[Typical traits of transactional data](#)

[When to use this solution](#)

[Challenges](#)

[OLTP in Azure](#)

[Key selection criteria](#)

[Capability matrix](#)

The management of transactional data using computer systems is referred to as Online Transaction Processing (OLTP). OLTP systems record business interactions as they occur in the day-to-day operation of the organization, and support querying of this data to make inferences.

## Transactional data

Transactional data is information that tracks the interactions related to an organization's activities. These interactions are typically business transactions, such as payments received from customers, payments made to suppliers, products moving through inventory, orders taken, or services delivered. Transactional events, which represent the transactions themselves, typically contain a time dimension, some numerical values, and references to other data.

Transactions typically need to be *atomic* and *consistent*. Atomicity means that an entire transaction always succeeds or fails as one unit of work, and is never left in a half-completed state. If a transaction cannot be completed, the database system must roll back any steps that were already done as part of that transaction. In a traditional RDBMS, this rollback happens automatically if a transaction cannot be completed. Consistency means that transactions always leave the data in a valid state. (These are very informal descriptions of atomicity and consistency. There are more formal definitions of these properties, such as [ACID](#).)

Transactional databases can support strong consistency for transactions using various locking strategies, such as pessimistic locking, to ensure that all data is strongly consistent within the context of the enterprise, for all users and processes.

The most common deployment architecture that uses transactional data is the data store tier in a 3-tier architecture. A 3-tier architecture typically consists of a presentation tier, business logic tier, and data store tier. A related deployment architecture is the [N-tier](#) architecture, which may have multiple middle-tiers handling business logic.

## Typical traits of transactional data

Transactional data tends to have the following traits:

Requirement	Description
Normalization	Highly normalized
Schema	Schema on write, strongly enforced
Consistency	Strong consistency, ACID guarantees

Requirement	Description
Integrity	High integrity
Uses transactions	Yes
Locking strategy	Pessimistic or optimistic
Updateable	Yes
Appendable	Yes
Workload	Heavy writes, moderate reads
Indexing	Primary and secondary indexes
Datum size	Small to medium sized
Model	Relational
Data shape	Tabular
Query flexibility	Highly flexible
Scale	Small (MBs) to Large (a few TBs)

## When to use this solution

Choose OLTP when you need to efficiently process and store business transactions and immediately make them available to client applications in a consistent way. Use this architecture when any tangible delay in processing would have a negative impact on the day-to-day operations of the business.

OLTP systems are designed to efficiently process and store transactions, as well as query transactional data. The goal of efficiently processing and storing individual transactions by an OLTP system is partly accomplished by data normalization — that is, breaking the data up into smaller chunks that are less redundant. This supports efficiency because it enables the OLTP system to process large numbers of transactions independently, and avoids extra processing needed to maintain data integrity in the presence of redundant data.

## Challenges

Implementing and using an OLTP system can create a few challenges:

- OLTP systems are not always good for handling aggregates over large amounts of data, although there are exceptions, such as a well-planned SQL Server-based solution. Analytics against the data, that rely on aggregate calculations over millions of individual transactions, are very resource intensive for an OLTP system. They can be slow to execute and can cause a slow-down by blocking other transactions in the database.
- When conducting analytics and reporting on data that is highly normalized, the queries tend to be complex, because most queries need to de-normalize the data by using joins. Also, naming conventions for database objects in OLTP systems tend to be terse and succinct. The increased normalization coupled with terse naming conventions makes OLTP systems difficult for business users to query, without the help of a DBA or data developer.
- Storing the history of transactions indefinitely and storing too much data in any one table can lead to slow query performance, depending on the number of transactions stored. The common solution is to maintain a relevant

window of time (such as the current fiscal year) in the OLTP system and offload historical data to other systems, such as a data mart or [data warehouse](#).

## OLTP in Azure

Applications such as websites hosted in [App Service Web Apps](#), REST APIs running in App Service, or mobile or desktop applications communicate with the OLTP system, typically via a REST API intermediary.

In practice, most workloads are not purely OLTP. There tends to be an analytical component as well. In addition, there is an increasing demand for real-time reporting, such as running reports against the operational system. This is also referred to as HTAP (Hybrid Transactional and Analytical Processing). For more information, see [Online Analytical Processing \(OLAP\)](#).

In Azure, all of the following data stores will meet the core requirements for OLTP and the management of transaction data:

- [Azure SQL Database](#)
- [SQL Server in an Azure virtual machine](#)
- [Azure Database for MySQL](#)
- [Azure Database for PostgreSQL](#)

## Key selection criteria

To narrow the choices, start by answering these questions:

- Do you want a managed service rather than managing your own servers?
- Does your solution have specific dependencies for Microsoft SQL Server, MySQL or PostgreSQL compatibility? Your application may limit the data stores you can choose based on the drivers it supports for communicating with the data store, or the assumptions it makes about which database is used.
- Are your write throughput requirements particularly high? If yes, choose an option that provides in-memory tables.
- Is your solution multitenant? If so, consider options that support capacity pools, where multiple database instances draw from an elastic pool of resources, instead of fixed resources per database. This can help you better distribute capacity across all database instances, and can make your solution more cost effective.
- Does your data need to be readable with low latency in multiple regions? If yes, choose an option that supports readable secondary replicas.
- Does your database need to be highly available across geo-graphic regions? If yes, choose an option that supports geographic replication. Also consider the options that support automatic failover from the primary replica to a secondary replica.
- Does your database have specific security needs? If yes, examine the options that provide capabilities like row level security, data masking, and transparent data encryption.

## Capability matrix

The following tables summarize the key differences in capabilities.

### General capabilities

--	--

Capability	Azure SQL Database	SQL Server in an Azure virtual machine	Azure Database for MySQL	Azure Database for PostgreSQL
Is Managed Service	Yes	No	Yes	Yes
Runs on Platform	N/A	Windows, Linux, Docker	N/A	N/A
Programmability <sup>1</sup>	T-SQL, .NET, R	T-SQL, .NET, R, Python	T-SQL, .NET, R, Python	SQL

[1] Not including client driver support, which allows many programming languages to connect to and use the OLTP data store.

### Scalability capabilities

Capability	Azure SQL Database	SQL Server in an Azure virtual machine	Azure Database for MySQL	Azure Database for PostgreSQL
Maximum database instance size	4 TB	256 TB	1 TB	1 TB
Supports capacity pools	Yes	Yes	No	No
Supports clusters scale out	No	Yes	No	No
Dynamic scalability (scale up)	Yes	No	Yes	Yes

### Analytic workload capabilities

Capability	Azure SQL Database	SQL Server in an Azure virtual machine	Azure Database for MySQL	Azure Database for PostgreSQL
Temporal tables	Yes	Yes	No	No
In-memory (memory-optimized) tables	Yes	Yes	No	No
Columnstore support	Yes	Yes	No	No
Adaptive query processing	Yes	Yes	No	No

### Availability capabilities

Capability	Azure SQL Database	SQL Server in an Azure virtual machine	Azure Database for MySQL	Azure Database for PostgreSQL
Readable secondaries	Yes	Yes	No	No

Capability	Azure SQL Database	SQL Server in an Azure virtual machine	Azure Database for MySQL	Azure Database for PostgreSQL
Geographic replication	Yes	Yes	No	No
Automatic failover to secondary	Yes	No	No	No
Point-in-time restore	Yes	Yes	Yes	Yes

## Security capabilities

Capability	Azure SQL Database	SQL Server in an Azure virtual machine	Azure Database for MySQL	Azure Database for PostgreSQL
Row level security	Yes	Yes	Yes	Yes
Data masking	Yes	Yes	No	No
Transparent data encryption	Yes	Yes	Yes	Yes
Restrict access to specific IP addresses	Yes	Yes	Yes	Yes
Restrict access to allow VNet access only	Yes	Yes	No	No
Azure Active Directory authentication	Yes	Yes	No	No
Active Directory authentication	No	Yes	No	No
Multi-factor authentication	Yes	Yes	No	No
Supports <a href="#">Always Encrypted</a>	Yes	Yes	Yes	No
Private IP	No	Yes	Yes	No