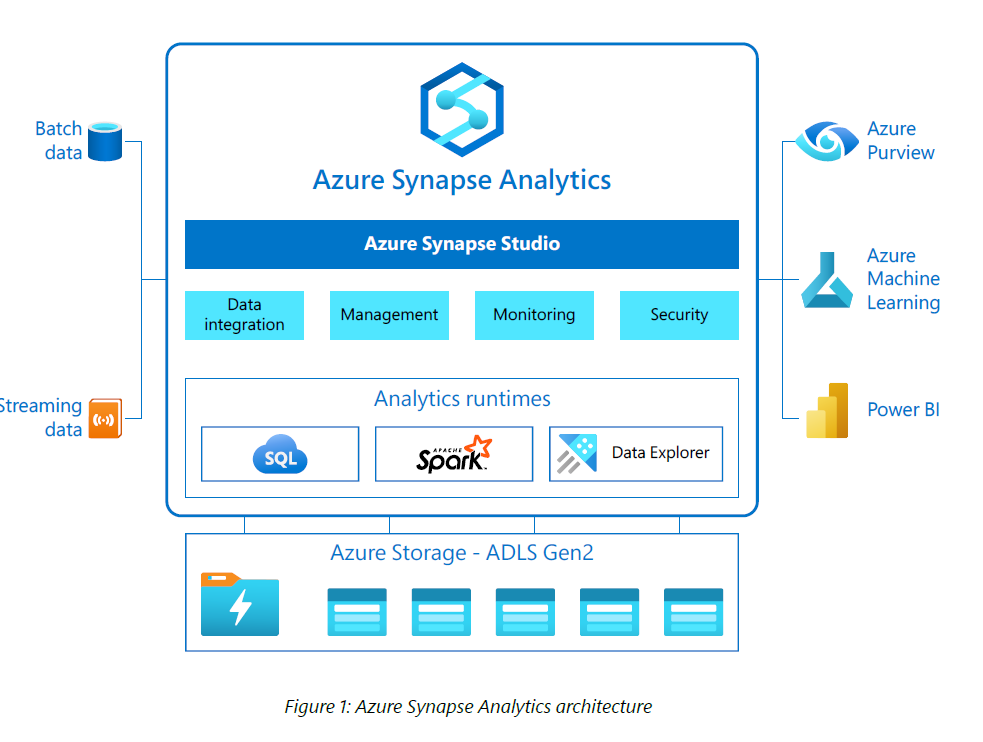
# [Azure Synapse Analytics](https://learn.microsoft.com/en-us/azure/synapse-analytics/)

Azure Synapse Analytics is a limitless and unified analytics service that brings together data integration, data exploration, enterprise data warehousing, and modern data platform analytics integrated with Azure Machine Learning and Microsoft Power BI. It helps get insights from batch and real‑time transactional data stored in operational databases such as Azure Cosmos DB with Azure Synapse Link.

Azure Synapse Analytics offers simplified ETL, code-free interactive data exploration, an optimized Apache Spark engine with in-engine predictive analytics, and intelligent workload management, which can optimize the functions of a data warehouse.



Azure SQL Data Warehouse has been renamed to Dedicated Pool

# Azure Synapse SQL POOL

Azure Synapse SQL Pool is distributed query engine.

Azure Synapse Analytics is Microsoft’s enterprise-grade analytics service, and **SQL Pools** are at the heart of it. Think of SQL Pools as the **compute engines** inside Synapse that let you query and process data with T-SQL.

There are **two types of SQL Pools**: **Dedicated** and **Serverless**.

**🌐 Azure Synapse SQL Pools Overview**

* A **SQL Pool** provides a SQL-based engine in Synapse Analytics for querying structured and semi-structured data.
* You can use it to query data in **Azure Data Lake**, **Synapse Tables**, or external sources.
* Choice of pool depends on whether you need **provisioned compute (always on)** or **on-demand pay-per-query**.

## ⚡ 1. Dedicated SQL Pool

✅ Also called **Provisioned pool** (previously Azure SQL Data Warehouse).

**🔹 What it is**

* A **massively parallel processing (MPP)** engine where you **provision compute resources (DWUs)** up front.
* Data is distributed across **60 distributions** behind the scenes, which allows parallel query execution.
* You create **tables inside the pool** and load data into them for analytics.

**🔹 Key Features**

* **Performance**: Optimized for large-scale queries and data warehouse workloads.
* **Persistent Storage**: Data is stored in Synapse’s distributed storage.
* **Scalability**: You can scale DWUs (Data Warehouse Units) up/down depending on workload.
* **Pause/Resume**: You can pause compute when not in use to save costs.

**🔹 Best Use Cases**

* Enterprise **data warehousing**.
* Regularly queried **structured datasets** (fact + dimension tables).
* BI dashboards (Power BI, Tableau) needing **consistent high performance**.

## 🌐 2. Serverless SQL Pool

✅ Also called **On-Demand pool**.

**🔹 What it is**

* A **pay-per-query** model where you don’t provision compute resources.
* Instead of storing data in Synapse tables, you can **query raw data directly in Azure Data Lake (CSV, Parquet, JSON, Delta Lake, etc.)** using T-SQL.
* Automatically scales and you only pay for the **data processed by queries**.

**🔹 Key Features**

* **No provisioning required** (always available).
* **Schema-on-read**: Define schema at query time.
* **Supports external tables and views** pointing to Data Lake files.
* **Great for exploration** and **ad-hoc queries** on big data.

**🔹 Best Use Cases**

* **Data exploration** without moving/transforming data.
* Quick **ad-hoc analysis** on raw files in Azure Data Lake.
* Lightweight BI scenarios where cost efficiency is critical.
* Staging/validation queries before moving data into a Dedicated Pool or Data Warehouse.

**📊 Quick Comparison: Dedicated vs Serverless**

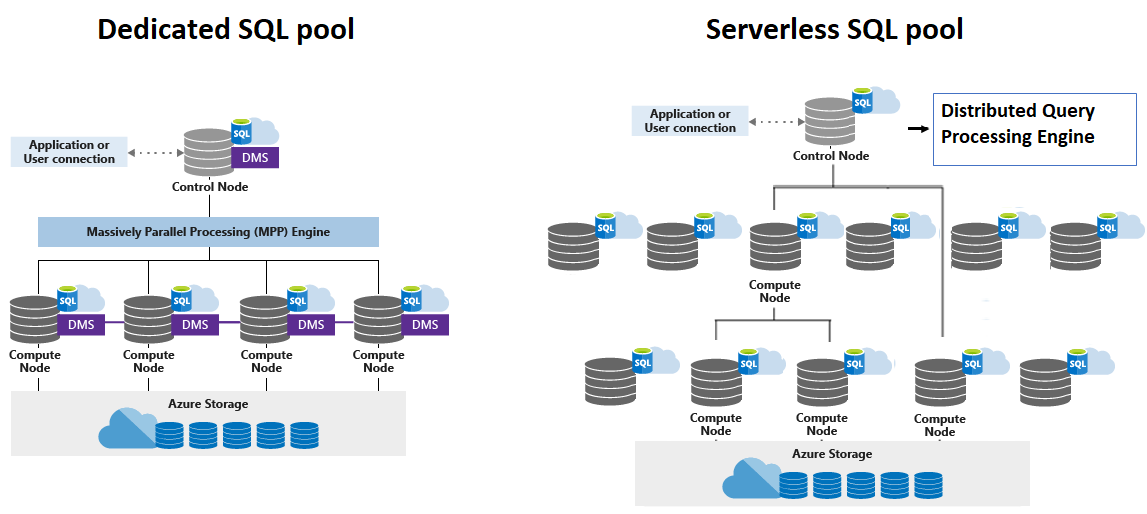
|  |  |  |
| --- | --- | --- |
| Feature | Dedicated SQL Pool 🚀 | Serverless SQL Pool 🌐 |
| Compute model | Provisioned (DWUs) | On-demand (pay per TB processed) |
| Storage | Stores data in Synapse tables | Queries raw files in Data Lake |
| Performance | High + predictable | Depends on query size |
| Cost model | Pay for provisioned DWUs (can pause) | Pay per query scanned data |
| Best for | Enterprise data warehouse, BI dashboards | Ad-hoc queries, data exploration, cost-efficient workloads |

👉 So in simple terms:

* **Dedicated =** Big warehouse, fixed resources, high performance.
* **Serverless =** Pay as you go, query raw data directly, no setup.

For dedicated SQL pool, the unit of scale is an abstraction of compute power that is known as a [data warehouse unit](https://learn.microsoft.com/en-us/azure/synapse-analytics/sql/resource-consumption-models).

For serverless SQL pool, being serverless, scaling is done automatically to accommodate query resource requirements. As topology changes over time by adding, removing nodes or failovers, it adapts to changes and makes sure your query has enough resources and finishes successfully. For example, the following image shows serverless SQL pool using four compute nodes to execute a query.



**Control node**

The Control node is the brain of the architecture. It's the front end that interacts with all applications and connections.

In Synapse SQL, the distributed query engine runs on the Control node to optimize and coordinate parallel queries. When you submit a T-SQL query to dedicated SQL pool, the Control node transforms it into queries that run against each distribution in parallel.

In serverless SQL pool, the DQP engine runs on Control node to optimize and coordinate distributed execution of user query by splitting it into smaller queries that will be executed on Compute nodes. It also assigns sets of files to be processed by each node.

**Compute nodes**

The Compute nodes provide the computational power.

In dedicated SQL pool, distributions map to Compute nodes for processing. As you pay for more compute resources, pool remaps the distributions to the available Compute nodes. The number of compute nodes ranges from 1 to 60, and is determined by the service level for the dedicated SQL pool. Each Compute node has a node ID that is visible in system views. You can see the Compute node ID by looking for the node\_id column in system views whose names begin with **sys.pdw\_nodes**.

For a list of these system views, see [Synapse SQL system views](https://learn.microsoft.com/en-us/sql/relational-databases/system-catalog-views/sql-data-warehouse-and-parallel-data-warehouse-catalog-views?view=azure-sqldw-latest&preserve-view=true).

In serverless SQL pool, each Compute node is assigned task and set of files to execute task on. Task is distributed query execution unit, which is actually part of query user submitted. Automatic scaling is in effect to make sure enough Compute nodes are utilized to execute user query.

## 🚀 Use Cases for Dedicated SQL Pool

(Provisioned, MPP, data warehouse style)

1. **Enterprise Data Warehouse for BI Dashboards**
   * A retail company consolidates data from ERP, CRM, and POS systems into Synapse tables.
   * Power BI dashboards need **fast, consistent performance** on fact/dimension tables with billions of rows.
   * Dedicated SQL Pool ensures predictable performance with pre-loaded, partitioned data.
2. **Financial Reporting & Compliance**
   * A bank stores historical transactions, risk calculations, and customer portfolios.
   * Compliance reports must run on-demand with **guaranteed speed** and **auditable storage**.
   * Dedicated SQL Pool is ideal because data is always available, secured, and query-optimized.
3. **Customer 360 Platform**
   * A telecom company integrates customer touchpoints (calls, web activity, app usage) into a single warehouse.
   * Analysts run **complex joins** across terabytes of structured data.
   * Dedicated SQL Pool allows schema enforcement, indexing, and fast query execution.
4. **Manufacturing Supply Chain Analytics**
   * Global supply chain data (orders, shipments, warehouse logs) stored in structured format.
   * Need for **daily scheduled jobs** that refresh dashboards and feed AI forecasting models.
   * Dedicated SQL Pool ensures reliable refresh windows and **predictable workload scaling**.

**🌐 Use Cases for Serverless SQL Pool**

(On-demand, schema-on-read, data exploration)

1. **Ad-hoc Data Exploration on Data Lake**
   * A media company stores raw event logs (JSON, Parquet) in ADLS Gen2.
   * Analysts need to quickly query subsets of data **without ETL**.
   * Serverless SQL Pool lets them query raw files directly, only paying per TB scanned.
2. **Data Validation & Staging Before Ingestion**
   * A consulting firm receives CSV drops from multiple clients in Data Lake.
   * They use Serverless to validate schema, detect missing values, and check row counts **before ingestion into a Dedicated Pool**.
3. **Cost-Efficient Reporting for Infrequent Queries**
   * An NGO analyzing climate datasets (TBs of sensor data in Data Lake).
   * They don’t run queries daily — only occasionally when research teams need insights.
   * Serverless is perfect because they only pay when they query.
4. **Lightweight Integration with External Tools**
   * A startup runs marketing analytics where campaign data lands as raw JSON in Data Lake.
   * They query it via Serverless and expose results directly to Power BI **without building a warehouse**.

**📊 Summary**

* **Dedicated SQL Pool** → Long-term storage + predictable high-performance queries on structured data (like an enterprise data warehouse).
* **Serverless SQL Pool** → Flexible, cost-efficient, quick exploration of raw/unstructured/semi-structured data in Data Lake.

## 📊 Dedicated vs Serverless SQL Pool

|  |  |  |
| --- | --- | --- |
| Feature | Dedicated SQL Pool 🚀 | Serverless SQL Pool 🌐 |
| Compute Model | Provisioned (DWUs = Data Warehouse Units). Always-on resources. | On-demand, auto-scaled. Compute is allocated only when queries run. |
| Cost Model | Pay for provisioned capacity (DWUs/hour), even if idle (can pause to save). | Pay per query, based on amount of data scanned (per TB). |
| Data Storage | Stores structured data inside Synapse tables (distributed MPP architecture). | Reads directly from **Azure Data Lake (CSV, Parquet, JSON, Delta, etc.)**. |
| Performance | Predictable, high-performance for large queries and enterprise workloads. | Varies based on query size & data volume. Best for light-to-medium workloads. |
| Best Fit For | Enterprise Data Warehousing, BI dashboards, complex joins, regulatory & compliance reporting. | Ad-hoc analysis, data exploration, staging/validation, cost-efficient infrequent reporting. |
| Data Model | Schema-on-write (data is loaded and optimized before querying). | Schema-on-read (schema applied dynamically at query time). |
| Scaling | Manual scaling of DWUs (can scale up/down as needed). | Auto-scales seamlessly based on query load. |
| Pause/Resume | Yes – you can pause to stop billing for compute. | Always available, but you only pay when queries are executed. |
| Integration | Tight integration with **Power BI** for fast, consistent dashboards. | Great for **ad-hoc Power BI queries** directly on Data Lake files. |
| Use Case Examples | Retail EDW, Banking Risk Analytics, Healthcare Clinical Warehouse, Supply Chain Optimization. | IoT/Clickstream log analysis, Data Lake exploration, ESG/Climate reporting, Quarterly NGO analysis. |

# What is SQL database vs Lake Database in Synapse?

In **Azure Synapse Analytics**, you can create two main types of databases: **SQL Database** and **Lake Database**.

**🔹 SQL Database (in Synapse)**

* A **relational database** hosted inside **Synapse SQL pool** (Dedicated or Serverless).
* Stores structured data in **tables** with schema (rows & columns).
* Supports **T-SQL queries** (similar to SQL Server).
* Designed for **analytical queries, joins, aggregations, reporting**.
* Works well when data is already cleaned and structured.

**Example use case:**

* Running Power BI dashboards on a sales transactions database.
* Querying processed, structured data for analytics and reporting.

**🔹 Lake Database (in Synapse)**

* A **logical database** built on top of a **data lake (ADLS Gen2)**.
* Data is stored in **Parquet/CSV/Delta format** in the lake, but exposed as **tables**.
* Uses **Spark engine** (and sometimes Serverless SQL pool) to query.
* Schema is **metadata-only** (schema-on-read).
* Useful for **big data, semi-structured/unstructured data (JSON, Parquet, Avro)**.

**Example use case:**

* Querying large IoT logs, clickstream data, or raw JSON files directly from the data lake.
* Building a data lakehouse where both structured + semi-structured data coexist.

**🔄 Comparison Table**

|  |  |  |
| --- | --- | --- |
| Feature | SQL Database (Synapse SQL) | Lake Database (Synapse Lake DB) |
| Storage | Inside Synapse SQL pool | Azure Data Lake (ADLS Gen2) |
| Engine | SQL Engine (Dedicated/Serverless) | Spark Engine / Schema-on-read |
| Data Types | Structured (rows/columns) | Structured + Semi-structured |
| Schema | Schema-on-write | Schema-on-read |
| Best For | BI dashboards, reporting, star-schema | Big Data analytics, raw/bronze layer queries |
| Performance | Optimized for SQL queries | Scales for large files & varied formats |
| Integration | Power BI, T-SQL queries | Spark ML, AI/ML, semi-structured pipelines |

👉 **In short:**

* Use **SQL Database** in Synapse when you want **structured, high-performance analytics** (like a data warehouse).
* Use **Lake Database** when you want to **query raw/semi-structured data directly from the lake** (like a data lakehouse).

**✅ 1. Is Lake Database in Synapse a Lakehouse?**

Yes — **Lake Database** in Synapse is Microsoft’s way of implementing a **Lakehouse architecture** on top of **Azure Data Lake Storage (ADLS Gen2)**.

* It provides a **structured SQL-like schema** layer on top of unstructured/semi-structured data in your **data lake**.
* This means you can treat data in ADLS like it’s in a relational database — with **tables, columns, primary keys, relationships** — but the data physically remains in the **data lake (Parquet, Delta, CSV, etc.)**.

So:  
👉 **Lake Database = Lakehouse implementation in Synapse.**

**✅ 2. What is a Lakehouse?**

A **Lakehouse** is a **hybrid architecture** that combines the best of a **Data Lake** and a **Data Warehouse**:

|  |  |  |  |
| --- | --- | --- | --- |
| Feature | Data Lake | Data Warehouse | Lakehouse (Hybrid) |
| Data Types | Raw, unstructured (logs, JSON, images, parquet, etc.) | Structured (tables, schema-based) | Supports both raw + structured |
| Storage Cost | Cheap (ADLS/S3, blob) | Expensive (SQL, Synapse DW, Snowflake) | Cheap (lake) with warehouse-like structure |
| Schema | Schema-on-read | Schema-on-write | Flexible (can do both) |
| Use Case | Store everything (data swamp risk) | BI, reporting, fast queries | Unified storage + analytics for AI, ML + BI |

**✅ 3. Why Lakehouse matters?**

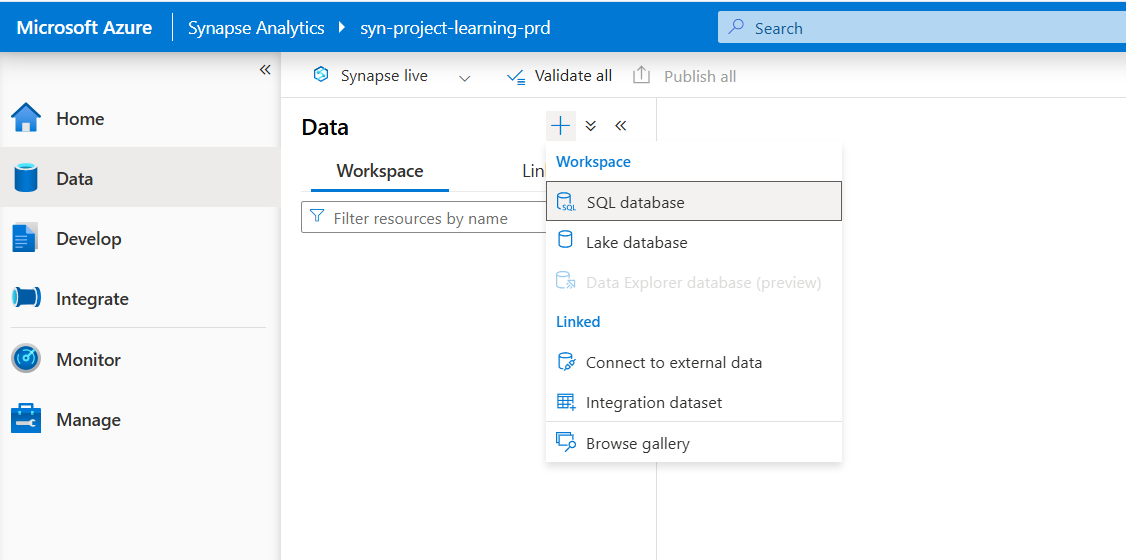
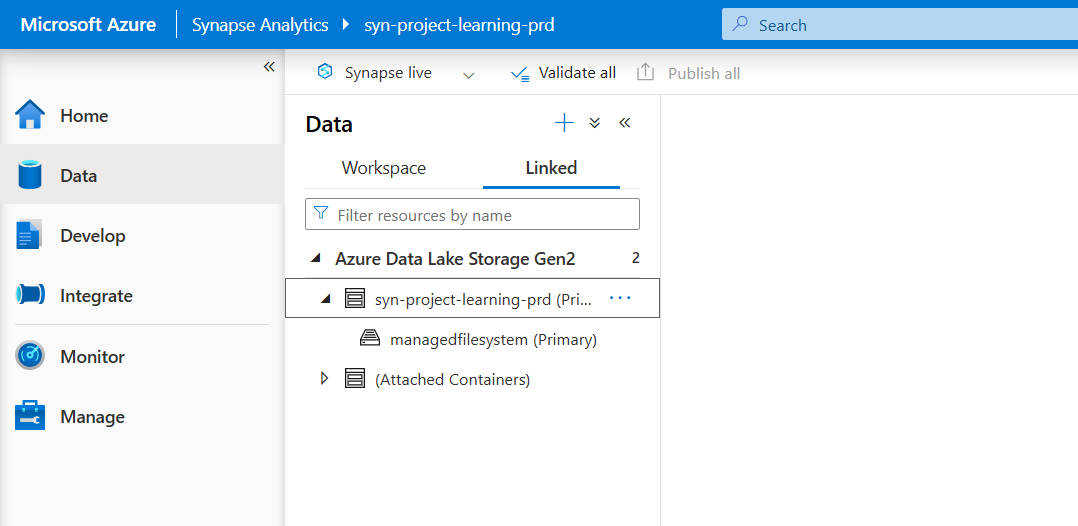
* **Single source of truth**: Instead of keeping one copy in Data Lake and another in Warehouse, you store it once in the lake and query it like a warehouse.
* **BI + AI/ML friendly**: Analysts can query via **SQL** and data scientists can run **PySpark/ML** on the same data.
* **Cost-efficient**: You don’t have to ETL data from a data lake to a warehouse all the time.

**🔑 In Synapse Context:**

* **SQL Database** → classic relational database in Synapse (like a traditional data warehouse table).
* **Lake Database** → lakehouse-style table definitions on top of files in **ADLS (CSV, Parquet, Delta Lake)**.

👉 Quick analogy:

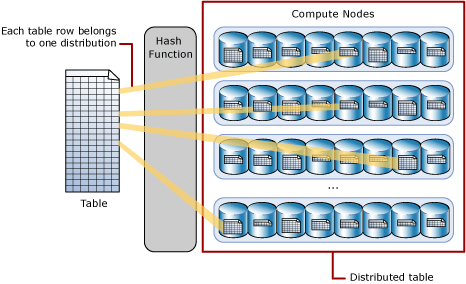
* **SQL Database** = A “restaurant menu” → well-structured, predefined dishes only.
* **Lake Database (Lakehouse)** = A “buffet + chef” → you can pick raw items (lake), but also enjoy ready-made dishes (warehouse).

# Distributed Table

## Hash distributed

A hash-distributed table distributes table rows across the Compute nodes by using a deterministic hash function to assign each row to one [distribution](https://learn.microsoft.com/en-us/azure/synapse-analytics/sql-data-warehouse/massively-parallel-processing-mpp-architecture#distributions).



Since identical values always hash to the same distribution, SQL Analytics has built-in knowledge of the row locations. In dedicated SQL pool this knowledge is used to minimize data movement during queries, which improves query performance.

Hash-distributed tables work well for large fact tables in a star schema. They can have very large numbers of rows and still achieve high performance. There are some design considerations that help you to get the performance the distributed system is designed to provide. Choosing a good distribution column or columns is one such consideration that is described in this article.

Consider using a hash-distributed table when:

* The table size on disk is more than 2 GB.
* The table has frequent insert, update, and delete operations.

**Hash Distribution**

* Data is distributed across compute nodes based on the hash value of a column (distribution column).
* Ensures even distribution of rows → prevents data skew.
* Best when queries frequently use joins or group by on the distribution column.
* Each row is assigned to one of the 60 distributions based on a hash function applied to a distribution column.
* Ensures that rows with the same value in the distribution column always go to the same distribution.
* Best for large fact tables where you frequently join on a particular column.
* Reduces data movement during queries.

✅ When to use:

* Large fact tables in a star schema.
* Tables where joins are common on a specific key (e.g., CustomerID, ProductID).

**Example**:  
A Sales fact table can be hash-distributed on CustomerID, so all rows of the same customer are in the same distribution.

## Round-robin distributed

A round-robin distributed table distributes table rows evenly across all distributions. The assignment of rows to distributions is random. Unlike hash-distributed tables, rows with equal values are not guaranteed to be assigned to the same distribution.

As a result, the system sometimes needs to invoke a data movement operation to better organize your data before it can resolve a query. This extra step can slow down your queries. For example, joining a round-robin table usually requires reshuffling the rows, which is a performance hit.

Consider using the round-robin distribution for your table in the following scenarios:

* When getting started as a simple starting point since it is the default
* If there is no obvious joining key
* If there is no good candidate column for hash distributing the table
* If the table does not share a common join key with other tables
* If the join is less significant than other joins in the query
* When the table is a temporary staging table
* Loading staging tables before transformations.
* Tables without frequent joins.

**Round Robin Distribution**

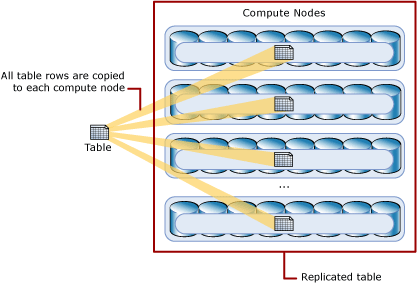
* Rows are randomly assigned across all compute nodes.
* No intelligent distribution → just spreads evenly but without regard to keys.
* Useful for staging or temporary data.

**Example**:  
A staging table where you load raw sales data before transformation.

## Replicated Tables

A replicated table provides the fastest query performance for small tables.

A table that is replicated caches a full copy of the table on each compute node. Consequently, replicating a table removes the need to transfer data among compute nodes before a join or aggregation. Replicated tables are best utilized with small tables. Extra storage is required and there is additional overhead that is incurred when writing data, which make large tables impractical.



**Replicated Distribution**

* A full copy of the table is stored on every compute node.
* Eliminates data movement during joins since each node already has a full copy.
* But consumes more storage → not scalable for large tables.

✅ **When to use:**

* Small dimension tables.
* Lookup/reference data used in joins.

## 📊 Tabular Comparison of Distribution Strategies

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Strategy | How it Works | Best For | Advantages | Disadvantages |
| Hash | Rows assigned using hash function on chosen column | Large fact tables, frequent joins on same column | Minimizes data movement, balances workload | Needs correct column selection, risk of skew |
| Round-Robin | Rows assigned evenly across distributions (random-like) | Staging tables, small-medium data | Easy to load, balanced distribution | Joins cause high data movement |
| Replicated | Full copy of table stored in every distribution | Small dimension tables | Zero data movement for joins, very fast | High storage cost, not scalable for big tables |

Create a Dedicated SQL POOL:

* 1. Go to Manage > SQL Pool > + New
  2. Provide name and Performance level
  3. Review + Create