- 12 minutes

Databases, as you can imagine, encompass a broad range of applications and requirements. To this end, we will discuss a few specific design patterns that database creators have to make when creating their database systems.

Schema enforcement

Database systems, in general, require some information regarding the types and structures of the data to be stored in the database. With relational database systems, the schema is formally defined in the following terms:

- Tuples (rows), which consist of attributes (columns), all of which are defined to be a certain type.
- A relation (table), which consists of multiple tuples of the same type.
- Constraints, which define a set of rules on tuples, attributes, and relations.

Schemas are not mere guidelines, but are enforced in relational systems. Schema-based databases have the following advantages:

- Schemas express the data model in a manner that allows for complex operations and queries to be constructed. For example, a join (where data is inferred from two or more distinct tables) is relatively easy to express in a relational database.
- Schemas can contain constraints that can be used to enforce program logic at the database level. For example, simple constraints can check if a person's age is a positive number, while a bank ledger implemented as a set of tables in a relational database can enforce the property (using a complex constraint) that the sum of all credits and debits in the system must equal zero. The system can automatically reject or return transactions that violate these constraints.
- Relational systems take advantage of the information provided in the schema to construct elaborate and efficient query plans, allowing the database to efficiently answer queries regarding the data stored.

However, the rigid enforcement of a schema is also a barrier to flexibility. As an application evolves, changing the schema of a table that is already populated with rows is difficult, if not impossible, depending on the table and the constraints provided.

On the other hand, there are systems that have either flexible schemas or no schema whatsoever. An example of a system with practically no schema is a generic **key-value store**, which basically acts as mapping between keys of a certain type and an associated value. Some key-value stores are very flexible; they allow for any arbitrary binary data to be stored under a particular key. Some key-value stores, such as AWS Dynamo DB, require that the type of the key (string, integer, etc.) be made explicit when a new table is created. Some systems allow for nested values to be present in data attributes, typically stored as JSON or XML, such as Azure Table storage.

Another variation is systems like GCP's Bigtable and Apache HBase, which have a semi-flexible schema. These systems require the number of columns to be declared when a table is created, but a column can have any number of nested sub-columns. These systems will be covered in detail in the next module.

Transaction vs. analytical processing

A key design choice that needs to be made in optimizing a database system is to optimize for the common case of the type of queries that it will receive. Two main patterns in database workloads have emerged.

Transactional workloads, also known as **Online Transactional Processing (OLTP)**, are workloads that are mainly composed of short online transactions. OLTP workloads mainly consist of insertions, updates, and/or deletions. The emphasis on OLTP systems is on fast query processing and maintaining data integrity in environments where there is simultaneous access. A good example of an OLTP workload is financial transactions. Queries will typically involve only a few tables and rows and will not require vast scans of the database.

On the other hand, certain systems aggregate and analyze data in order to gain insights and derive information from the data. Such systems are called **Online Analytic Processing (OLAP) systems**. OLAP systems generally involve a lower volume of transactions, but individual queries may be complex and involve aggregate computations that span multiple rows and multiple tables. Typical OLAP queries thus are significantly longer running than OLTP queries.

The following video provides an overview of OLTP vs. OLAP:

Scalability

Over time, when a database system's user base and data grow, the system may require some form of **scaling**, which would expand its capacity and/or performance in order to support more users or data, or both. Database scaling is complex for a variety of reasons, which we will cover in detail in the following video:

Vertical scaling

Vertical scaling (or scaling up) is the process of increasing a database's capacity or ability to handle load without adding more hosts. This can be done by hardware upgrades such as a faster CPU, increased RAM, or disks with more capacity and/or more I/O operations per second (IOPS). The scalability with vertical scaling is limited by the amount of CPU, RAM, and disk that can be configured on a single machine.

Horizontal scaling

Horizontal scaling (or scaling out) means the scaling of databases by adding more nodes to handle database queries. Different RDBMS products offer different ways of scaling out, including **replication** and database **sharding**. With replication, the same data is stored across multiple nodes. This enables the RDBMS to handle additional requests in parallel, thereby increasing the number of transactions per second. This approach works when the database is read-heavy. In sharding, a large table is partitioned across multiple nodes (typically on an index or key). The amount of data shared across nodes is limited in this case, and limited sharing is preferred because it reduces issues with consistency of replicated data.

Replication is the process of duplicating data over multiple servers in order to increase the number of available paths to individual pieces of data. Data can be replicated for performance, availability, and/or fault tolerance reasons. Replicated data allows for faster performance, particularly for reads, as replication allows for parallel access to multiple copies of the same data. Replication assists in availability and fault tolerance, as data is available in a backup location should a part of the database system fail.

Another method, which is orthogonal to replication, it **sharding**. Sharding is the process by which data is partitioned into sections (known as shards) and distributed over multiple database systems. In contrast to replication, each shard acts as the single source for the subset of data contained by it. Sharding is a technique that is specifically used to improve the performance of databases, and in particular, write performance.

In an ideal case, a sharded database will evenly spread out the data across all partitions. But this is difficult to achieve, as the data distribution across all partitions should be more or less balanced. A popular technique that is the mainstay of many current database systems is **consistent hashing**.

Consistent hashing is a special kind of hashing technique that allows for a hash table to be expanded in an efficient manner. If `K` is the total number of keys and `n` is the number of buckets in a given hash table, consistent hashing requires only `K/n` keys to be remapped on average every time a new bucket is added to the hash table.[1]

Some systems use a combination of replication and sharding to provide high performance while maintaining high availability and fault tolerance. However, in using replication, a major concern is consistency, which will be discussed next.

References

1. *Karger, David and Lehman, Eric and Leighton, Tom and Panigrahy, Rina and Levine, Matthew and Lewin, Daniel (1997). Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web Proceedings of the Twenty-ninth Annual ACM Symposium on Theory of Computing*

Check your knowledge

1.

Suppose you're designing a storage system for an application that extracts tables from webpages. The application plans to assign a unique identifier for each table and then store the table in the database in a raw XML-based format. What kind of system might be best suited for this type of application?

Strict schema-based database

Schemaless database (such as a key-value store)

2.

An airline reservation system uses a database to keep track of passengers, airlines, and flights. The system is used to book passengers on flights and retrieve the information regarding passengers that have an existing booking on a flight. What kind of database is most likely in use to power this system?

3.

A travel application analyzes flight trends by ingesting worldwide passenger data to provide tourist trends for travel agents and international tour operators. What kind of underlying database system is this application most likely to use?