Name: **Elipian, Jethro Ryan C,** Year & Section: **3-BSCS-A**

Types of databases

1. Relational

A relational database is a collection of data items with pre-defined relationships between them. These items are organized as a set of tables with columns and rows. Tables are used to hold information about the objects to be represented in the database. Each column in a table holds a certain kind of data and a field stores the actual value of an attribute. The rows in the table represent a collection of related values of one object or entity. Each row in a table could be marked with a unique identifier called a primary key, and rows among multiple tables can be made related using foreign keys. This data can be accessed in many different ways without reorganizing the database tables themselves.es

SQL

SQL or Structured Query Language is the primary interface used to communicate with Relational Databases. SQL became a standard of the American National Standards Institute (ANSI) in 1986. The standard ANSI SQL is supported by all popular relational database engines, and some of these engines also have extension to ANSI SQL to support functionality which is specific to that engine. SQL is used to add, update or delete rows of data, retrieving subsets of data for transaction processing and analytics applications, and to manage all aspects of the database.

Data Integrity

Data integrity is the overall completeness, accuracy and consistency of data. Relational databases use a set of constraints to enforce data integrity in the database. These include primary Keys, Foreign Keys, ‘Not NULL’ constraint, ‘Unique’ constraint, ‘Default’ constraint and ‘Check’ constraints. These integrity constraints help enforce business rules on data in the tables to ensure the accuracy and reliability of the data. In addition to these, most relation databases also allow custom code to be embedded in triggers that execute based on an action on the database.

Transactions

A database transaction is one or more SQL statements that are executed as a sequence of operations that form a single logical unit of work. Transactions provide an "all-or-nothing" proposition, meaning that the entire transaction must complete as a single unit and be written to the database or none of the individual components of the transaction should go through. In the relation database terminology, a transaction results in a COMMIT or a ROLLBACK. Each transaction is treated in a coherent and reliable way independent of other transactions.

ACID Compliance

All database transactions must be ACID compliant or be Atomic, Consistent, Isolated and Durable to ensure data integrity.

Atomicity requires that either transaction as a whole be successfully executed or if a part of the transaction fails, then the entire transaction be invalidated. Consistency mandates the data written to the database as part of the transaction must adhere to all defined rules, and restrictions including constraints, cascades, and triggers. Isolation is critical to achieving concurrency control and makes sure each transaction is independent unto itself. Durability requires that all of the changes made to the database be permanent once a transaction is successfully completed.

2. Analytic (OLAP)

OLAP (online analytical processing) is a computing method that enables users to easily and selectively extract and query data in order to analyze it from different points of view. OLAP business intelligence queries often aid in trends analysis, financial reporting, sales forecasting, budgeting and other planning purposes.

For example, a user can request that data be analyzed to display a spreadsheet showing all of a company's beach ball products sold in Florida in the month of July, compare revenue figures with those for the same products in September and then see a comparison of other product sales in Florida in the same time period.

How OLAP systems work

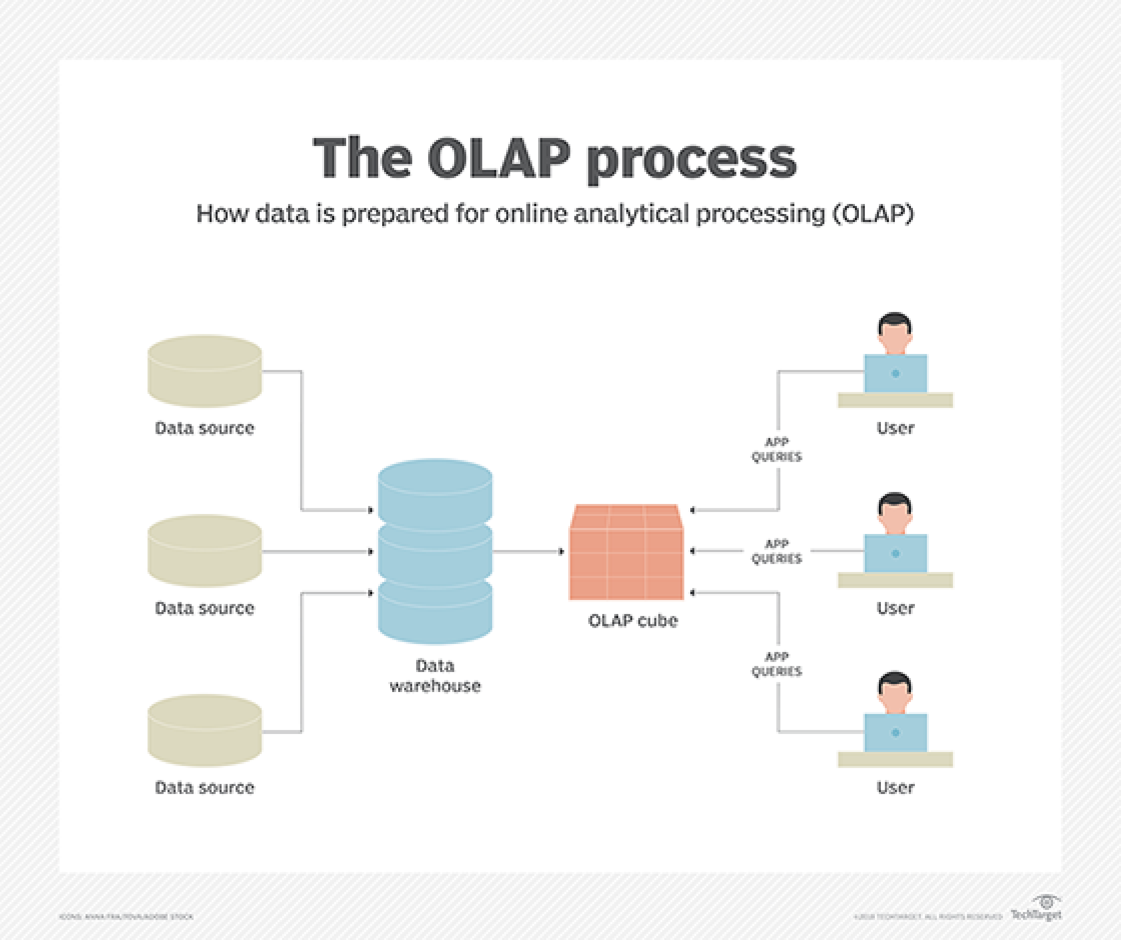
To facilitate this kind of analysis, data is collected from multiple data sources and stored in data warehouses then cleansed and organized into data cubes. Each OLAP cube contains data categorized by dimensions (such as customers, geographic sales region and time period) derived by dimensional tables in the data warehouses. Dimensions are then populated by members (such as customer names, countries and months) that are organized hierarchically. OLAP cubes are often pre-summarized across dimensions to drastically improve query time over relational databases.

Analysts can then perform five types of OLAP analytical operations against these multidimensional databases:

- **Roll-up**. Also known as consolidation, or drill-up, this operation summarizes the data along the dimension.
- **Drill-down**. This allows analysts to navigate deeper among the dimensions of data, for example drilling down from "time period" to "years" and "months" to chart sales growth for a product.

- **Slice.** This enables an analyst to take one level of information for display, such as "sales in 2017."
- **Dice**. This allows an analyst to select data from multiple dimensions to analyze, such as "sales of blue beach balls in Iowa in 2017."
- **Pivot.** Analysts can gain a new view of data by rotating the data axes of the cube.

OLAP software then locates the intersection of dimensions, such as all products sold in the Eastern region above a certain price during a certain time period, and displays them. The result is the "measure"; each OLAP cube has at least one to perhaps hundreds of measures, which are derived from information stored in fact tables in the data warehouse.


The OLAP process
How data is prepared for online analytical processing (OLAP)
Data source
Data source
Data source
Data warehouse
OLAP cube
APP QUERIES
APP QUERIES
APP QUERIES
User
User
User
ICONS: ANNA FRAJTOVA/ADOBE STOCK
©2018 TECHTARGET. ALL RIGHTS RESERVED
TechTarget

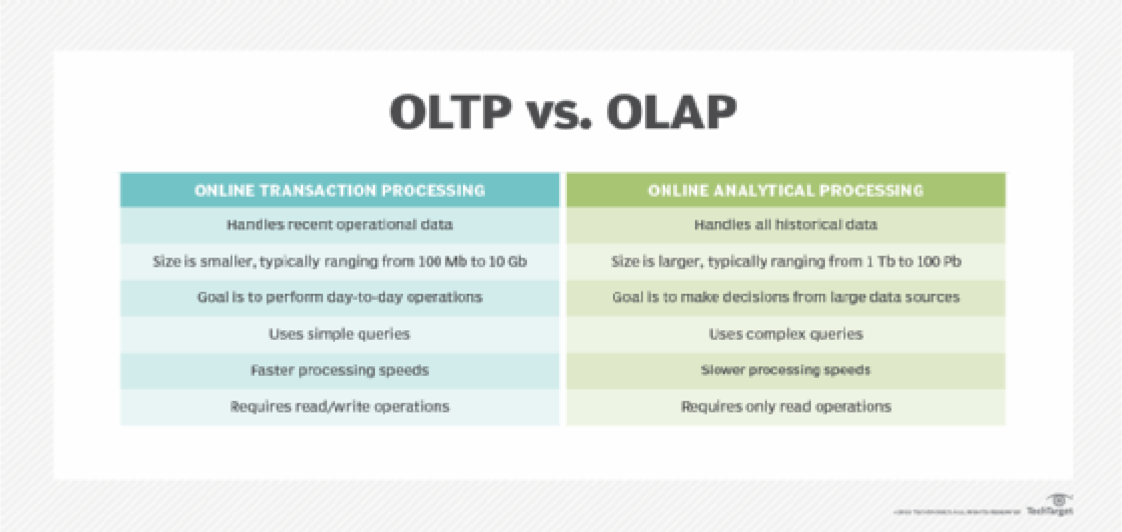
TECHTARGET

OLAP begins with data accumulated from multiple sources and stored in a data warehouse. The data is then cleansed and stored in OLAP cubes, which users run queries against.

Types of OLAP systems

OLAP (online analytical processing) systems typically fall into one of three types:

- **Multidimensional OLAP (MOLAP)** is OLAP that indexes directly into a multidimensional database.
- **Relational OLAP (ROLAP)** is OLAP that performs dynamic multidimensional analysis of data stored in a relational database.
- **Hybrid OLAP (HOLAP)** is a combination of ROLAP and MOLAP. HOLAP was developed to combine the greater data capacity of ROLAP with the superior processing capability of MOLAP.



OLTP vs. OLAP

ONLINE TRANSACTION PROCESSING	ONLINE ANALYTICAL PROCESSING
Handles recent operational data	Handles all historical data
Size is smaller, typically ranging from 100 Mb to 10 Gb	Size is larger, typically ranging from 1 Tb to 100 Pb
Goal is to perform day-to-day operations	Goal is to make decisions from large data sources
Uses simple queries	Uses complex queries
Faster processing speeds	Slower processing speeds
Requires read/write operations	Requires only read operations

TechTarget

This table shows the differences between online analytical processing and online transaction processing.

Uses of OLAP

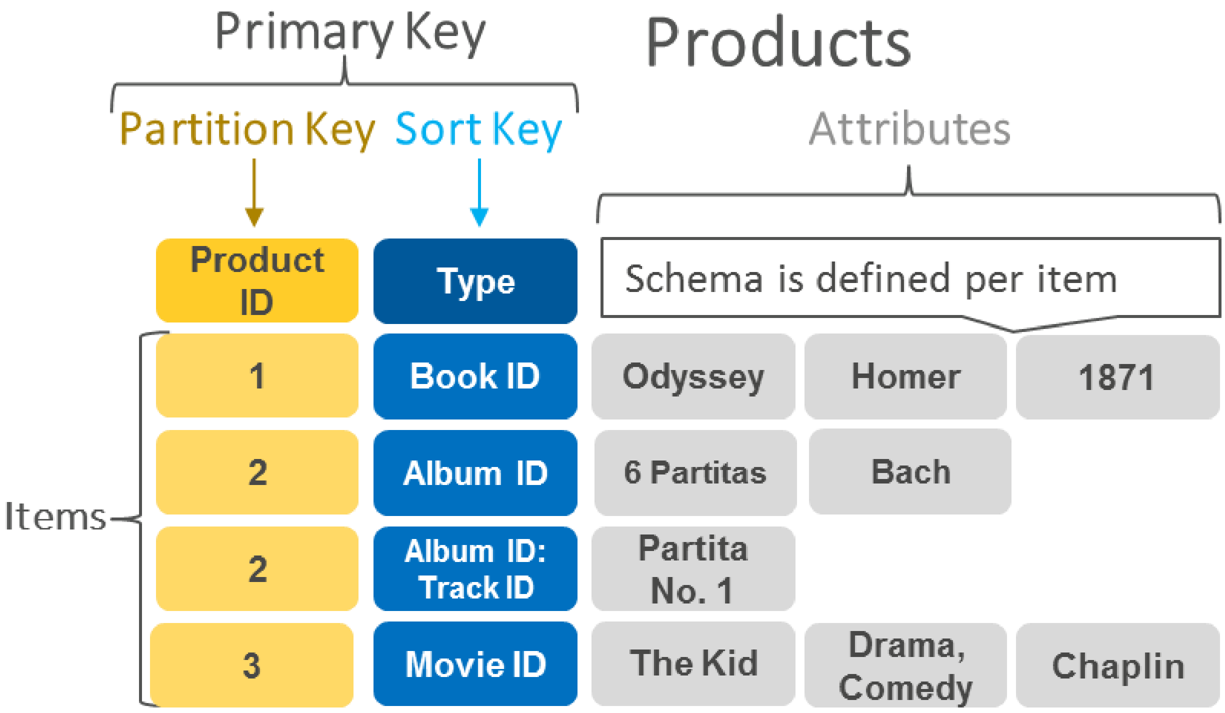
OLAP can be used for data mining or the discovery of previously undiscerned relationships between data items. An OLAP database does not need to be as large as a data warehouse, since not all transactional data is needed for trend analysis. Using Open Database Connectivity (ODBC), data can be imported from existing relational databases to create a multidimensional database for OLAP.

OLAP products include IBM Cognos, Oracle OLAP and Oracle Essbase. OLAP features are also included in tools such as Microsoft Excel and Microsoft SQL Server's Analysis Services). OLAP products are typically designed for multiple-user environments, with the cost of the software based on the number of users.

3. Key-Value

A key-value database is a type of nonrelational database that uses a simple key-value method to store data. A key-value database stores data as a collection of key-value pairs in which a key serve as a unique identifier. Both keys and values can be anything, ranging from simple objects to complex compound objects. Key-value databases are highly partitionable and allow horizontal scaling at scales that other types of databases cannot achieve. For example, Amazon DynamoDB allocates additional partitions to a table if an existing partition fills to capacity and more storage space is required.

The following diagram shows an example of data stored as key-value pairs in DynamoDB.



Product ID	Type			
1	Book ID	Odyssey	Homer	1871
2	Album ID	6 Partitas	Bach	
2	Album ID: Track ID	Partita No. 1		
3	Movie ID	The Kid	Drama, Comedy	Chaplin

Use cases

- **Session store**

A session-oriented application such as a web application starts a session when a user logs in and is active until the user logs out or the session times out. During this period, the application stores all session-related data either in the main memory or in a database. Session data may include user profile information, messages, personalized data and themes, recommendations, targeted promotions, and discounts. Each user session has a unique identifier. Session data is never queried by anything other than a primary key, so a fast key-value store is a better fit for session data. In general, key-value databases may provide smaller per-page overhead than relational databases.

- **Shopping cart**

During the holiday shopping season, an e-commerce website may receive billions of orders in seconds. Key-value databases can handle the scaling of large amounts of data and extremely high volumes of state changes while servicing millions of simultaneous users through distributed processing and storage. Key-value databases also have built-in redundancy, which can handle the loss of storage nodes.

Popular key-value databases

Amazon DynamoDB is a nonrelational database that delivers reliable performance at any scale. It's a fully managed, multi-region, multi-master database that provides consistent single-digit millisecond latency, and offers built-in security, backup and restore, and in-memory caching. In DynamoDB, an Item is composed of a primary or composite key and a flexible number of attributes. There is no explicit limitation on the number of attributes associated with an individual item, but the aggregate size of an item, including all the attribute names and attribute values, cannot exceed 400 KB. A table is a collection of data items, just as a table in a relational database is a collection of rows. Each table can have an infinite number of data items.

You can be up and running with DynamoDB in 10 minutes with this step-by-step tutorial. Learn more about DynamoDB and get started today.

4. Column-Family

NoSQL column family database is another aggregate oriented database. In NoSQL column family database we have a single key which is also known as row key and within that, we can store multiple column families where each column family is a combination of columns that fit together.

Column family as a whole is effectively your aggregate. We use row key and column family name to address a column family.

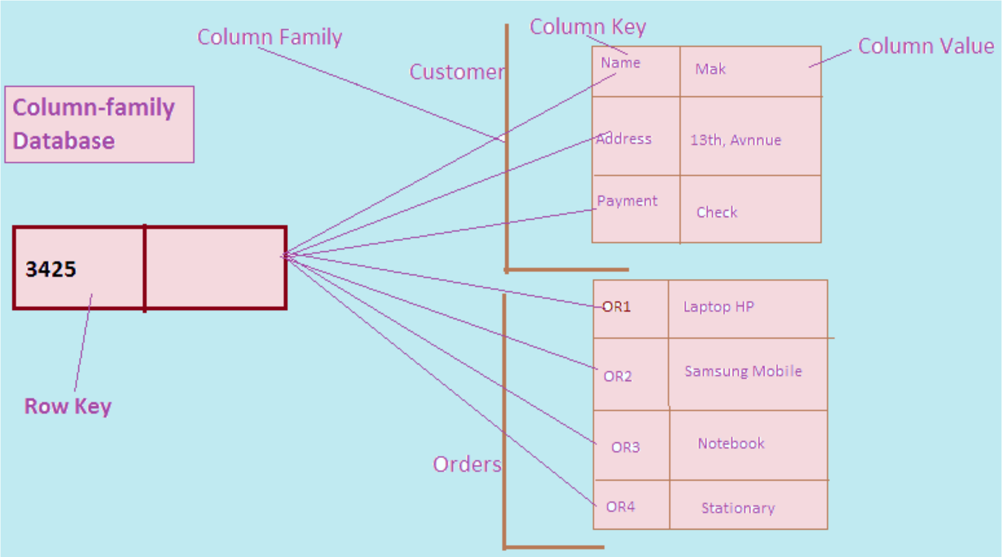
It is, however, one of the most complicated aggregate databases but the gain we have in terms of retrieval time of aggregate rows. When we are taking these aggregates into the memory, instead of spreading across a lot of individual records we store the whole thing in one database in one go.

The database is designed in such a way that it clearly knows what the aggregate boundaries are. This is very useful when we run this database on the cluster.

As we know that aggregate binds the data together, hence different aggregates are spread across different nodes in the cluster.

Therefore, if somebody wants to retrieve the data, say about a particular order, then you need to go to one node in the cluster instead of shooting on all other nodes to pick up different rows and aggregate it.

Among the most popular column family NoSQL databases are Apache HBase and Cassandra.



Column Family Database

Application of Column family NoSQL Database

Let us understand the key application of column family NoSQL database in real world scenarios.

Big Table (Column Family Database) to store sparse data

We know that NULL values in the relational database typically consume 2 bytes of space. This is a significant amount of wasted space when there are a number of NULL values in the database. Let us suppose we have a "Contact Application" that stores username and contact details for every type of the network such as Home-Phone, Cell-Phone, Zynga etc. If let us say for few of the user only Cell-Phone detail is available then there will be hundreds of bytes wasted per record. Below is the sample contact table in RDBMS which clearly depicts the waste of space per record.

ContactID	Home-Phone	Cell-Phone	Email1	Email2	Facebook	Twitter
1X2B	NULL	9867	x@abc	NULL	NULL	NULL
2X2B	1234	NULL	NULL	NULL	NULL	#bigtable
3X3Y	3456	9845	NULL	y@wqa	a@fb.com	#hadoop

Contact Table in RDBMS

The storage issue can be fixed by using the BigTable which manages sparse data very well instead of RDBMS. The BigTable will store only the columns that have values for each record instance. If we indicate only Home-Phone, Cell-Phone and Email1 details that need to be stored for ContactID ‘1X2B’ then it will store only these three column values and rest will be ignored i.e. null will not be considered and hence no wastage of space.

RowKey	Column Values	
1234	**ph:cell=9867**	**email:1=x@abc**
3678	**social:twitter=#bigtable**	**ph:home=1234**
5987	**email:2=y@wqa**	**social:facebook=a@fb.com**
Contact Table in Big Table Storage		

Analysing Log File using BigTable

Log analysis is a common use case for any Big Data project. All data generated through log files by your IT infrastructure often are referred to as data exhaust. The vast information about the logs is stored in big tables. It is then analyzed nearly in real time in order to track the most updated information. The reason why log files are stored in the BigTable is that they have flexible columns with varying structures.

HostName	**IP Address**	**Event DT**	**Type**	**Duration**	**Description**
server1	**10.219.12.3**	**4-Feb-15 1:04:21 PM**	**dwn.exe**	**15**	**Desktop Manager**
server2	**10.112.3.45**	**4-May-15 1:04:21 PM**	**jboss**	**45**	
Typical Log File					

RowKey	**Column Values**					
12AE23	**host:name=server1**	**ip:address =10.219.12.3**	**event:DT=4-Feb-15 1:04:21 PM**	**Type= dwn.exe**	**Dur=15**	**Desc= Desktop Manager**
Log File stored in BigTable						

5. Graph

A graph database is a type of NoSQL database that is based on graph theory. Graph databases are ideal for storing data that has complex many to many relationships. In this article, we will study the very basics of graph databases with the help of a simple example.

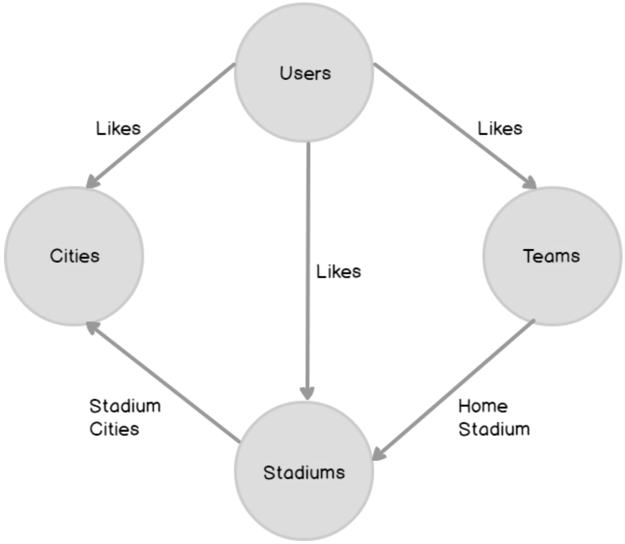
Characteristics of a Graph Database

A graph data consists of nodes and edges. Nodes are also sometimes called vertices. The nodes represent an entity such as a Person, City, Employee, and Customer etc. Edges are used to map the relationship between entities. Graph databases are well suited to things like supply chain management systems. Graph databases have also proven efficient to develop recommender systems where the relationships like “Person who bought item X, also bought item Y,” have to be mapped.

A Simple Example of a Graph Database

Social media platforms are one of the best examples of how graph databases work. Consider a scenario where a person likes a particular football team. A user can also like one or more football stadiums. Alternatively, one football stadium can be liked by multiple users. Users can also like football stadiums and cities. A football team has a home stadium. A stadium can be located in a particular city and one city can have multiple stadiums. A graph database is ideally suited to storing this type of information. Users, teams, stadiums and cities can be implemented as entities or nodes. On the other hand, likes, home stadium and stadium cities can be implemented as relationships or edges.

The following figure represents such a graph database:



Implementing a Graph Database

As discussed earlier, a graph database primarily consists of nodes and edges. Let’s first implement the nodes in our graph database. We will be using SQL Server Management Studio to run our scripts.

Let’s create our database:

```
USE master;
GO
DROP DATABASE IF EXISTS PLGraph;
GO
CREATE DATABASE PLGraph;
GO
```

In the script above, we created a database named “PLGraph”. The database will store information about users who like premier league teams. A few premier league teams will also be stored in the database along with their home stadiums and cities in which the stadiums are located.

Implementing Nodes

We identified 4 nodes in our database. Let’s implement them one by one. We will start with the Users node.

Users Node

Execute the following script to implement the Users node.

```
USE PLGraph;
GO
DROP TABLE IF EXISTS Users;
GO
CREATE TABLE Users (
  UserID INT IDENTITY PRIMARY KEY,
  UserName NVARCHAR(100) NOT NULL,
  ) AS NODE;
```

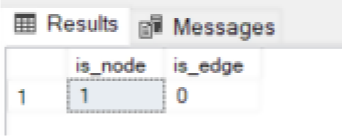
You can see that creating a node in a graph database is very similar to creating a table. The syntax is pretty similar. However, to create a node, all you have to do is specify “AS NODE” at the end of table definition as shown in the above script.

You can see that the node has one primary key UserID and one column UserName. It is important to mention that a node must contain a primary key.

To see if a table is a node or an edge, you can use the following script:

```
SELECT is_node, is_edge FROM sys.tables
  WHERE name = 'Users';
```

The output looks like this:



	is_node	is_edge
1	1	0

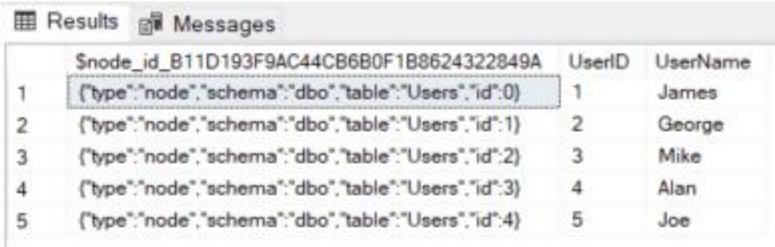
In the output, you can see a 1 in the “is_node” column and 0 in the “is_edge” column, which means that this table represents a node.

Let’s now enter some records in the Users node.

```
VALUES
  ('James'),
  ('George'),
  ('Mike'),
  ('Alan'),
  ('Joe')
```

Now, execute the following script to see the records in the Users node:

```
SELECT * FROM Users
```



	$node_id_B11D193F9AC44CB6B0F1B8624322849A	UserID	UserName
1	{"type":"node","schema":"dbo","table":"Users","id":0}	1	James
2	{"type":"node","schema":"dbo","table":"Users","id":1}	2	George
3	{"type":"node","schema":"dbo","table":"Users","id":2}	3	Mike
4	{"type":"node","schema":"dbo","table":"Users","id":3}	4	Alan
5	{"type":"node","schema":"dbo","table":"Users","id":4}	5	Joe

You can see that there are three columns in the Users node. Two of the columns i.e. UserID and UserName are user-defined columns. If you look at the first column it contains JSON data that contains type, scheme and id for each record in the node. By default, the id for the records in the node begins with 0.

Conclusion

Graph databases are changing the way complex many to many operations are implemented. In this article, we briefly reviewed how to create nodes and edges in graph databases. We also saw how to implement relationships between different nodes and how to perform insert, read and delete operations on the edges. To continue your learning on Graph Databases in SQL Server, you can direct to the following articles:

- An introduction to a SQL Server 2017 graph database
- How to implement a graph database in SQL Server 2017

6. Document

A document database is a type of nonrelational database that is designed to store and query data as JSON-like documents. Document databases make it easier for developers to store and query data in a database by using the same document-model format they use in their application code. The flexible, semistructured, and hierarchical nature of documents and document databases allows them to evolve with applications' needs. The document model works well with use cases such as catalogs, user profiles, and content management systems where each document is unique and evolves over time. Document databases enable flexible indexing, powerful ad hoc queries, and analytics over collections of documents.

In the following example, a JSON-like document describes a book.

```
[
    {
        "year" : 2013,
        "title" : "Turn It Down, Or Else!",
        "info" : {
            "directors" : [ "Alice Smith", "Bob Jones"],
            "release_date" : "2013-01-18T00:00:00Z",
            "rating" : 6.2,
            "genres" : ["Comedy", "Drama"],
            "image_url" : "http://ia.media-
imdb.com/images/N/O9ERWAU7FS797AJ7LU8HN09AMUP908RLlo5JF90EWR7LJKQ7@@._V1_SX400_.jp
g",
            "plot" : "A rock band plays their music at high volumes, annoying the
neighbors.",
            "actors" : ["David Matthewman", "Jonathan G. Neff"]
        }
    },
    {
        "year": 2015,
        "title": "The Big New Movie",
        "info": {
            "plot": "Nothing happens at all.",
            "rating": 0
        }
    }
]
```

Use cases

Content management

A document database is a great choice for content management applications such as blogs and video platforms. With a document database, each entity that the application tracks can be stored as a single document. The document database is more intuitive for a developer to update an application as the requirements evolve. In addition, if the data model needs to change, only the affected documents need to be updated. No schema update is required and no database downtime is necessary to make the changes.

Catalogs

Document databases are efficient and effective for storing catalog information. For example, in an e-commerce application, different products usually have different numbers of attributes. Managing thousands of attributes in relational databases is inefficient, and the reading performance is affected. Using a document database, each product’s attributes can be described in a single document for easy management and faster reading speed. Changing the attributes of one product won’t affect others.

Document databases on AWS

Amazon DocumentDB (with MongoDB compatibility) is a fast, scalable, highly available, and fully managed document database service that supports MongoDB workloads. Developers can use the same MongoDB application code, drivers, and tools to run, manage, and scale workloads on Amazon DocumentDB and enjoy improved performance, scalability, and availability without having to worry about managing the underlying infrastructure.