Choosing a Database Architecture: An Essential Guide for Data Warehousing Professionals

Saibal Samaddar is a principal consultant at PwC, India.
saibal.samaddar@in.pwc.com

Saibal Samaddar

Abstract

With the ever-increasing data volumes and increasing demand from business users to access data faster, data warehouse architects and designers are facing tremendous challenges to build robust systems that can sustain the volume of data over time and provide faster access to reliable data. Big data analytics using Hadoop, NoSQL databases with several incarnations (such as key-value stores, document stores, and graph database systems) are still in a nascent stage of evaluation in most telecoms. Significant efforts are being made to squeeze out the best of what relational database management systems have to offer.

Introduction

The growth in voice services has slowed and OTT (over the top) players are providing stiff competition. Non-voice services (such as bill pay and data services) are helping telecommunication companies to grow in this time of uncertainty. The data volume growth is a function of the increase in a telecom's subscriber base and the subscribers' increased use of varied services. Data volume growth varies from country to country; in developing markets such as southeast Asia and the Middle East, the year-over-year data growth is around 25 percent.

This article discusses two key database management system (DBMS) designs—the *disk-based, column-oriented DBMS* and the *in-memory, column-oriented DBMS*—that might well provide an answer to a CTO's demand for sustaining data growth while providing faster access to data. A generic telecom case study will be used to discuss the two key concepts. Before delving

into the details of each key concept for data warehousing design, let us first define several telecom terms that we will use in this article.

- *Base transceiver station* (BTS) is a piece of equipment that facilitates wireless communication between user equipment (such as mobile phones) and a network (such as GSM or CDMA)
- *Incoming call*, sometimes known as mobile terminating call (MTC), is the call received by a subscriber
- *Outgoing call*, sometimes known as mobile originating call (MOC), is the call made by a subscriber

The Traditional, Row-Oriented DBMS

Relational DBMS (RDBMS) data is structured in database tables, fields, and records. Each RDBMS table consists of rows, each of which consists of one or more fields.

Region	BTS	Date	Time	Subscriber Account #	Duration of Incoming Calls (Min)	Duration of Outgoing Calls (Min)
West	BANDRA1	12-Jun-13	9:01	983158777	10	0
West	BANDRA1	12-Jun-13	9:11	903158778	0	5
West	BANDRA1	12-Jun-13	9:23	980158779	12	0
West	BANDRA1	12-Jun-13	10:21	983158777	2	0
West	BANDRA1	12-Jun-13	10:29	903158778	3	0
West	BANDRA2	12-Jun-13	10:46	980158779	0	5
West	BANDRA1	12-Jun-13	10:48	903158778	5	0

This data must be serialized so it can be stored in hardware.

Depending on the database, the database engine generates a unique ID for each row (commonly referred to as the *rowid*). This data is usually stored in one of several blocks depending on its volume. The initial block information and the pointers to other blocks are stored in the database engine.

The rowid holds the information the database needs to access a row. For Oracle, the rowid is an 18-digit hex number composed of the following subsections:

Data Object No.	Data File No.	Data Block No.	Row No.
000000	AAF	BBBBBB	001

In a row-oriented DBMS, data is serialized in rows. The stored data will look like this:

OOOOOOAAFBBBBBB001:
West,BANDRA1,12-Jun-13,9:01,983158777,10,0;
OOOOOOAAFBBBBBB002: West, BANDRA1,
12-Jun-13,9:11,903158778,0,5;

OOOOOOAAFBBBBBB001 and
OOOOOOAAFBBBBBB002 are the rowids corresponding to row 1 and row 2 in the table.

Imagine that our data warehouse uses this architecture and we need to build a report to determine the total incoming and outgoing calls made on 12-Jun-13 between 9:00 and 11:59 a.m., per BTS. This type of information is often required to judge the utilization of a particular BTS.

To find all the records, the DBMS must examine the entire data set looking for matching records. To improve performance, the query could have been indexed on time and partitioned on date. Partitioning allows a table to be subdivided into smaller pieces, each having its own name and optionally its own storage area.

The indexed, serialized data will be stored in the following format.

OOOOOOAAFBBBBBB001: 9:01;
OOOOOOAAFBBBBBB002: 9:11;

OOOOOOAAFBBBBBB001 and
OOOOOOAAFBBBBBB002 are the rowids corresponding to row 1 and row 2 in the table.

This database approach is best suited for OLTP workloads because all data will be serialized at once.

There are two key disadvantages:

- Running ad hoc queries (a common requirement for self-service BI) will slow down performance because not all the columns are indexed.

- If the same data appears in multiple rows, there is data redundancy, which increases data volumes and distributes the data to additional data blocks. According to Moore's law, the computing power doubles every two to three years while costs fall. CPU processing, memory, and disk storage are all subject to some variation of this law, but the latency in seeking data from hard disks will create a bottleneck and system performance will suffer.

The Disk-Based, Column-Oriented DBMS

A column-oriented DBMS stores data tables as sections of columns of data rather than as rows. This has advantages for data warehouses, customer relationship management (CRM) systems, and other systems that support ad hoc inquiries where aggregates are computed over large numbers of similar data items.

In column-oriented storage, the same data will be stored as:

West: OOOOOOAAFBBBBBB001, OOOOOOAAFBBBBBB002; Bandra1: OOOOOOAAFBBBBBB001, OOOOOOAAFBBBBBB002; 12-Jun-13: OOOOOOAAFBBBBBB001, OOOOOOAAFBBBBBB002; 9:01: OOOOOOAAFBBBBBB001; 9:11: OOOOOOAAFBBBBBB002;

983158777: OOOOOOAAFBBBBBB001; 903158778: OOOOOOAAFBBBBBB002; 10: OOOOOOAAFBBBBBB001; 0: OOOOOOAAFBBBBBB002; 0: OOOOOOAAFBBBBBB001; 5: OOOOOOAAFBBBBBB002;

Note that OOOOOOAAFBBBBBB001 and OOOOOOAAFBBBBBB002 are the rowids corresponding to row 1 and row 2 in the table.

The storage pattern resembles the structure of an index in a row-based system; however, in a row-oriented system the primary key is the rowid. In a column-oriented DBMS, data *is* the primary key. Thus, this architecture eliminates the possibility of data duplication. This behavior is extremely important when writing sparse data in the database because it improves performance and reduces the overall storage requirement (which, in turn, reduces I/O to fetch data from blocks).

In this example, data will be retrieved much faster because the fetch is based on the data values it will retrieve (the rowids) when it calculates the total duration of incoming and outgoing calls.

This DBMS architecture has three key advantages:

- The architecture is best suited to OLAP workloads where sparse data will be encountered, thus reducing the physical storage needed

- Query performance is improved because all the columns are already indexed by values

- Because column data has a uniform type, columnar compression is much greater than for row-oriented storage

There is at least one disadvantage to the design, however: It is not a good choice if the full record is to be retrieved (as for OLTP transactional queries).

Among the better-known column-oriented databases are Sybase IQ, Google's BigTable, Teradata Columnar, and Vertica. Teradata claims to decrease (by 90 percent) the amount of data needed for the query. Oracle Exadata provides a hybrid columnar compression to achieve significant storage savings.

The In-Memory, Column-Oriented DBMS

Column-oriented databases store data more efficiently and with greater compression. They store huge amounts of data in the same physical space, which reduces the memory needed to perform a query. The design also increases processing speed. This has given rise to the in-memory database (IMDB) design.

With IMDB, all information is initially loaded into memory, eliminating the need for optimizing databases

(such as creating indexes, aggregation, and designing cubes and star schemas). Most in-memory tools use compression algorithms such as run-length encoding, dictionary encoding, or bit-vector encoding, that reduce the size of in-memory data needed in RAM. Users query the data loaded into the system's memory, eliminating the latency to seek data from disk.

This speedier data retrieval is the design's biggest advantage. However, as the number of users and data volumes grow, the amount of RAM needed to hold the data also grows, increasing hardware costs.

Some well-known in-memory databases are Oracle TimesTen, IBM solidDB, and SAP HANA DB.

Conclusion

As enterprises seek reliable data faster while optimizing their hardware spending, column-oriented databases provide the optimal solution. Column-oriented databases can yield a storage savings of about 90 percent. If the size of a row-oriented database is 100 TB, the equivalent column-oriented database will require only 10 TB. With databases doubling in size every two years, at the end of six years that database will grow to 80 TB. Thus, without procuring any additional hardware, the enterprise can handle the growth in data for six years.

Moreover, thanks to a column-oriented database's internal storage format, OLAP operations perform faster than they would in a row-oriented database. Though IMDB can provide data faster, the additional hardware cost it requires often outweighs its benefit. An Oracle TimesTen database uses 2 TB of RAM; a column-oriented database such as Sybase IQ, Teradata Columnar, or Oracle's Exadata Hybrid Columnar Compression should be ideally used for data warehousing requirements to sustain the ever-growing data volume and faster time retrieval. If your IT budget is not a constraint, then your enterprise should consider moving to an in-memory, column-oriented database to achieve even faster performance. ■