Research Paper Engineering

# Filestream : A New Object Storage Technique in Databases

**Mr. BHAVESH LUKKA** | Research Scholar, Singhania University.

**Dr. DEEPAK KUMAR** | Research Supervisor, Singhania University.

**ABSTRACT**

The present paper is dedicated to a suggestion of a new method of storing the objects into databases. This suggests a new way so that storing large objects into database does not slow down the performance of the databases at the same time, the database size also doesn't grow drastically, at the same time, the query processing becomes easier and more user friendly, that the user doesn't need to make a system call to read the desired object from file. The object is virtually stored within database only, whereas, physically it is stored somewhere separately into the file system, and only a pointer to that is actually stored into the database.

**KEYWORDS** | FileStream, Object Storage Technique, Database Performance, Easy Query Processing

## INTRODUCTION

Data is the heart of every task. If we have data, we can process it and can get the desired results. The more data one has, the more data processing capability one has. Database is a collection of data, typically describing the activities of one or more related organizations. It is a computer application software that interact with users, other applications and the database its' self to access and manage the data as per the requirement. The ease with which information can be obtained from a database often determines its value to a user [1]. The size of data is increasing day-by-day. With the increase of size comes various challenges pertaining to data warehousing, including, storage technique, retrieval efficiency and data relevance. One can see the search engine Google. It has a very huge data warehouse, containing information on almost all topics. The database of Google also search the string looking to its relevance, so that, the top results may be useful for its users. A single search there produces lacs and crores of results, but we never usually go beyond few hundreds, as we get the desired information within the first hundred results. This way, it is a big challenge, not only to store data systematically, but also to make its retrieval faster and relevant.

In databases, lots of data-types are available, which are comprehensive and support almost all possible types of storage requirement. The data sizes are varying from bit to large objects, from smallest of the numbers to large number, from fixed size to varying size etc. All the possible data types have already been covered.

This paper is not about to introduce a new type of data to be stored into a database. It is only to suggest a new type, which supports only the existing type, but handles it in a better and efficient way. It has been purposefully given a new name, so that, databases may maintain their backward compatibility to deal with older style data types.

FileStream, as the name suggests, is to store files into database. It can be any file, whether text or binary, whether a word document, a PDF file, an image file, an image of CD/DVD, a database backup file. Any type of file can be stored here. Unlike an Operating System, the file here is not associated with any application. The association of file with its corresponding application is the task of the Operating System, it is running on. This data type is proposed to store the bit-stream of file, in the byte-order of the parent Operating System.

## BACKGROUND

All the database management systems present today have their performance-vise ups and downs [11] and they all support a comprehensive list of data types, varying from primitive data types to user defined data types, smallest possible data types to largest object data types, text data types to binary data types as listed under :

| My SQL | MS SQL | PgSQL | DB2 | Oracle | SQLite |
|---|---|---|---|---|---|
| Integer (int) | Bigint | Smallint (int2) | Smallint | Char | Int |
| SmallInt | Bit | Bigint (int8) | Integer (int) | Varchar | Integer |
| Decimal (Dec, Fixed) | Decimal | Bigserial (serial8) | Bigint | varchar2 | Tinyint |
| Numeric | Int | Bit | Decimal (numeric) | Nchar | Smallint |
| Float | Money | Varbit | Decfloat | Nvarchar2 | Mediumint |
| Real | Numeric | Boolean (bool) | Real | Lob | Bigint |
| Double | Smallint | Box | Double | Long | Unsigned big int |
| Bit | Smallmoney | Bytea | Character | Long raw | Int2 |
| TinyInt | Tinyint | Varchar | Varchar | Raw | Int8 |
| MediumInt | Float | Character (char) | Clob | Number | Character |
| BigInt | Real | Cidr | Graphic | Float | Varchar |
| Date | Date | Circle | Vargraphic | Binary_float | Varying character |
| DateTime | Datetime | Date | Dbclob | Binary_double | Nchar |
| TimeStamp | Datetime2 | Double precision (float8) | Binary | Date | Native character |
| Time | Datetimeoffset | Inet | Varbinary | Timestamp | Nvarchar |
| Year(2 / 4) | Smalldatetime | Integer (int, int4) | Blob | Timestamp with time zone | Text |
| Char | Time | Internal | Date | Timestamp with local time zone | Clob |
| Varchar | Char | Line | Time | Interval year to month | Blob |
| Binary | Varchar | Lseg | Timestamp | Interval day to second | Real |
| Varbinary | Text | Macaddr | Xml | Blob | Double |
| Blob | Nchar | Money | | Clob | Double precision |
| Text | Nvarchar | Numeric (decimal) | | Nclob | Float |

| My SQL | MS SQL | PgSQL | DB2 | Oracle | SQLite |
|---|---|---|---|---|---|
| ENUM | Ntext | Path | | Bfile | Numeric |
| Set | Binary | Point | | RowID | Decimal |
| | Varbi-nary | Polygon | | URowId | Boolean |
| | Image | Real (float4) | | XMLType | Date |
| | Cursor | Serial (serial4) | | UriType | Datetime |
| | Times-tamp | Text | | Character | |
| | Hierar-chyid | Time | | Numeric (decimal, dec) | |
| | Uniquei-dentifier | Timetz | | Integer | |
| | Sql_vari-ant | Times-tamp | | Int | |
| | Xml | Times-tamptz | | Smallint | |
| | Table | Tsquery | | Double | |
| | | tsvector | | Real | |
| | | Txid_ snapshot | | OrdAudio | |
| | | Uuid | | OrdImage | |
| | | xml | | OrdVideo | |
| | | | | OrdDoc | |
| | | | | OrdDicom | |

**Table 1: List Of Data Types Available In Major Database Management Systems. [4][5][6][7][8][9].**

The above list covers all the available and possible data types. This paper is not about adding any new type of data, but it suggests only the change in the way of storing large objects. Presently the database management systems supports large objects in the data type, shown in bold in the above list.

**HYPOTHESIS**

The main problem with existing object data types, as highlighted hereinabove, is that they grow the database size, thereby, reducing the performance. They are not always required data types. Most of the times, other columns only work fine with the queries. These data types are used only as and when they are really required, whereas, other columns are required comparatively more frequently.

An alternate approach to this is to store the object into a file and saved into the file system of the parent operating system and thereafter, the path of the file is to be stored into the database. For example, if student table is to be created with photograph of each student, these photographs may be stored into the table itself, or may be stored in the form of a JPG or PNG file within the file system and its path to be given in the table.

Both the approaches have their own pros and cons. The first approach is better in terms of security and integrity, but hits badly in terms of performance and grows database size drastically. The second one, while has almost no effect on the database size, is not secured and sometimes, loses the data integrity, as the file may have been moved or removed, without changing the path stored in the database.

This paper presents a new way of storing large objects. For the brevity of supporting old databases, the existing data type and the way it stores data should be continued. In that case, a new data type may be added, which will store large objects into database, in an enhanced way.

**DESIGN**

In the present scenario, if objects are stored within the database, its size grows drastically thereby, hitting very badly its performance. These objects are not actually required in every query we make to the databases, but their effect is always seen in the form of slow execution of every query. To overcome this, people started storing the object(s) in the form of a file within the file system of the parent operating system and storing its reference, i.e. its full absolute path, to the database for use. Whenever the user needs this object, he refers database to fetch its reference and then requests the OS file system to retrieve the file for further use. Both the approaches have their own pros and cons. The first provides secured store of objects but at the same time, it slows down the performance. The second one is less secured, the file being accessible outside the database system, but the only overhead to the database is its reference, which is simply a small string and therefore, its impact on the performance is almost null.

In the present paper, it is proposed to include the better part of both the approaches. In a nutshell, it can be depicted as storing the reference in the form of a filename only will not suffice. The reference should contain all the information required to fetch the file from the file system. This may be dependent on the parent file system, but usually in production environment, the database system(s) once deployed, work for years continuously. Frequent movement of data is never preferred in production environment. Following diagram depicts the information a database should contain for every file, to enable it to access the file directly.

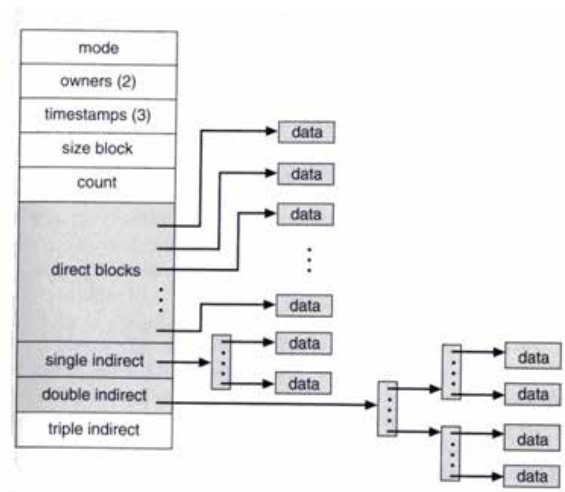


**Figure 1 : Object In Databases To Enable To Access File Directly**

Operating Systems store the first column that contains various metadata about the file, called i-node. This i-node helps operating systems access the file, wherever it is stored. The size of i-node is less than a block. A single read/write on the disk can access the entire metadata of the file. Therefore, for very smaller files (usually less than 60 bytes), some file systems support inlining. Inlining refers to a technology where the file contents are stored within the i-node only, instead of direct / single-indirect / double-indirect / triple-indirect block(s) information. In some Operating Systems this feature can be enabled at the time of file system creation.

Therefore, inclusion of i-node within the database column will help the database management system to access the file directly, without the intervention of the parent operating system. This will not only save the time of the user, who, after reading information from database, sends a system call through operating system to read the file, but also, will save our database from the loss of performance, which may have been occurred if the object is directly stored in the database, as in the traditional way.

**CONCLUSIONS**

The approach presented in this paper will help users of the databases to store large objects and files with ease in their databases itself, without having any bad impact on the performance of the databases. Almost all the available database

management systems support way of storing large objects and/or files into the database itself, but they store it within data file. This approach eventually slows down the performance of the system, affecting the performance of query execution very badly. The object-storage technique presented in this paper actually stores the object in the form of a file within the file systems and keeps its pointer-block (i-node) within the database, thereby, occupying less space, usually fixed size with every row.

## REFERENCES

[1]Ramakrishnan R. and J. Gehrke, "Database Management System", McGraw-Hill International Edition Publication, 2000. | [2]Baron Schwartz, Peter Zaitsev and Vadim Tkachenko, "High Performance MySQL", ISBN : 978-1-449-31428-6, O'reilly Publication, 2012. | [3]Paul Nilsen, Uttam Parui, "Microsoft SQL Server Bible", ISBN : 978-11-180-7987-4, John Wiley & Sons. Publications, 2011. | [4] Oracle Documentation Available On : http://docs.oracle.com/en/database/database.html [Accessed : February 2015]. | [5]DB2 Documentation Available On : | http://www-01.ibm.com/support/knowledgecenter/ [Accessed : February 2015]. | [6]PostgreSQL Documentation Available On : http://www.postgresql.org/docs/ [Accessed : February 2015]. | [7]MySQL Documentation Available On : http://dev.mysql.com/doc/refman/5.7/en/ [Accessed : February 2015]. | [8] SQLite Documentation Available On : https://www.sqlite.org/docs.html [Accessed : February 2015]. | [9]MS SQL Server Documentation Available On : https://msdn.microsoft.com/en-us/library/dd206988.aspx [Accessed : February 2015]. | [10] Joseph M. Hellerstein, Michael Stonebraker and James Hamilton, "Architecture Of Database A System", Foundations and Trends in Databases Vol. 1, No. 2 (2007) Pages : 141–259. | [11] Bhavesh Lukka and Dr. Deepak Kumar, "Performance Analysis Of Database Management Systems With Different Front-ends", Research Matrix International Multidisciplinary Journal Of Applied Research ISSN : 2321 – 7073 [Volume : 01, Issue : 09, April 2014]