Available online at www.sciencedirect.com

ScienceDirect

Procedia Engineering 192 (2017) 516 – 521

Procedia Engineering

www.elsevier.com/locate/procedia

TRANSCOM 2017: International scientific conference on sustainable, modern and safe transport

Temporal index location in multiple tablespace type definitions

Michal Kvet[a]*, Karol Matiaško[a]

[a]*University of Zilina, Faculty of Management Science and Informatics, Univerzitna 8215/1, 010 26 Slovakia*

Abstract

Temporal database has been defined using object level granularity in the recent past. Our develoved system uses fine-grained granularity - column or column group. In this paper, we propose developed approach, but mostly highlight the performance with regards tablespace type for data location. Tablespace as intermediate layer between database instance and data files themselves is mainly characterized by the location, size and structure. Block size database parameter is fixed and cannot be changed after database creation at all. In this paper, we use multiple size and structure of the block in the tablespace and compare the performance based on temporal access queries. Such concepts and solutions can be used in any information system dealing with time limited validity objects as well as sensor data processing.

Keywords: temporal database; column level temporal approach; tablespace; index; balancing

1. Introduction

Complex information systems need techniques for managing, evaluating, transforming, storing and retrieving data. The key part is just database. Nowadays, we can perceive multiple database approach streams, however, to provide transaction integrity, conventional database is still used. The paradigm is based on accessing actual data, thus, if there is requirement to change stored values, *Update* statement is executed consequencing overwriting non-actual data. Even in the introduction of database systems themselves, there have been significant pushes for storing also historical data, to be able to reconstruct object states in the past. Moreover, also managing future valid data can

* Corresponding author. Tel.: +421-41-513-4024.
*E-mail address:*michal.kvet@fri.uniza.sk

doi:10.1016/j.proeng.2017.06.089

be important and useful. If the operator knows about the environment properties to be changed in the future, naturally, they can be stored in the database sooner with reflection of the timepoint defining validity. In that case, automatic state changes and reflections must be provided. Most of the ideas and developed systems formed the temporal concept resulting in temporal standard. Unfortunately, it has not been approved resulting in total revocation of the temporal approaches in 2001 [1] [7]. The only way used was based on extension of the primary key. Thus, primary key of the table can be divided into two groups – object identifier itself and temporal definition, which is mostly modelled by the validity. However, generalization of such approach can hold multiple temporal characteristicts, like validity, transaction validity, reliability or locality. It is called object level temporal approach [6].

2. Column level temporal approach architecture

Our proposed solution is based on management temporal data using attribute level granularity. Thanks to that, there are no redundancy, if some object state values (attribute values) are not changed during the *Update* statement. The point is that the *Update* statement does not need to influence all attribute values. Column level temporal approach is based on managing direct attribute itself regardless the number of attribute values changed. The main advantage is just sensorial data processing reflecting various granularity and frequency of changes.

Defined column level temporal approach consists of three layers. The first layer deals with actual valid data, thus existing applications based on conventional approach can work without necessity to rebuild and reconstruct application code. Second layer is the core of the system consisting of temporal management. If the attribute value is updated, particular information is stored in temporal tables. This layer manages also planned states – which validity will start in the future. Our approach manages such updates automatically using calendar and *Job plan module.* Non-actual values (historical and future planned) are stored in the third layer. Complete architecture and management is described in [8] [10].

Later, temporal layer has been extended and splitted into two separate layers. Temporal characteristics themselves are managed in the original layer. However, particular attribute definitions and structures can evolve over the time, attributes can be transformed from conventional to temporal or vice versa, monitoring of some attributes can be terminated and new can be added. Moreover, visibility of stored states can be defined for specific users, groups or based on privilege rules. It is determined by the privacy and audit strategies [11].

3. Tablespace and Oracle block definition

Data in the database are physically stored in data files located on disc storage. However, database manager and user processes cannot access disc space storage, mainly due to security reasons to declare consistency with data dictionary and performance views. Tablespace is the interface between database instance and physical database. Oracle defines tablespace also as repository for schema object data, including the data dictionary, which is located in the SYS schema. All databases must have at least SYSTEM and SYSAUX tablespace [2] [13] [14]. Standard data objects are usually located in user defined tablespaces, which is advantageous from the performance and access perspective, it reduces the bottleneck from accessing data dictionary and user data using the same access path – SYSTEM tablespace - illustrated also by the approach changes in the past – dictionary management of the tablespace has been replaced by the local management [12] [15]. Another aspect is also structure and management.

Thanks to the separation of physical storage from the logical sphere operated by tablespaces, developers do not need to deal with physical representation, storage structures, data copying and accessing methods. They have even only limited information about the finding, where, the row is physically stored. The relational database theory paradigm, except dealing with only actual valid data, states that programmer can address only logical structures and should let the database manage the mapping mechanisms [12]. Significant part of the storage unit is just database and operating system block. Tablespace usually contains multiple datafiles and manages multiple database objects (segments) and their data extents. In the past, old database management systems mapped each object to separate data files with no possibility for paralelism. It is no longer true, now. Database block can be covered by multiple operating systems block, or there can be 1:1 mapping principle. It is not useful to define it another way – size of the database block would be smaller than operating system block, due to performance [16]. Limitation of the definition