# Improving Clinical Data Warehouse Performance via a Windowing Data Structure Architecture

Joshua Nealon, Wenny Rahayu and Eric Pardede
*School of Computer Science and*
*Computer Engineering*
*La Trobe University*
*Bundoora, Victoria 3086*
*Email: {J.Nealon,W.Rahayu,E.Pardede}@latrobe.edu.au*

## Abstract

*Data warehousing and on-line analytical processing (OLAP) are essential elements of decision support and commonly require large periods of time to process query results. Given that, it is crucial to tailor individual performance solutions based on the structure and merits of the data within the data warehouse to obtain the most optimal configuration. Therefore, this paper presents an analysis of the defining features of a clinical data warehouse, where the focus of the analysis is centered on the opportunities and threats for optimization purposes, based on the types of queries that are likely to arise. From the findings, the paper introduces a Windowing Data Structure Architecture (WDSA) to increase the performance of OLAP queries on a clinical data warehouse, which manages a collection of popular windows. The cost of processing a query using a simple instance of the WDSA was compared against existing query processing techniques such as nested join and hash join, to which the technique greatly outperformed both in almost all cases.*

## 1. Introduction

On-line Analytical Processing (OLAP) technology has grown in popularity dramatically since the first OLAP product (Express) appeared on the market in the year 1970 [1]. This technology coupled with data warehousing technology is a vital tool in the industry to analyse the performance of a companies business processes, which in turn has increased the amount of data stored by businesses [2], as larger data sets allows for a greater level of analysis. Given that, a greater level of analysis comes with a performance trade-off due to the fact that the larger the data set is, the longer it will take to seek through the data for a desired result. This is commonly not acceptable for a Decision Support System (DSS) as most decisions will require a relatively quick response.

This trend is also obvious in the health informatics sector, even though the usage of a data warehouse for analysis purposes in a clinical environment is relatively young and not yet standardized [3]. To reduce the response time of queries a number of optimization techniques have been introduced via research and commercial systems.

The introduction of moving window operations and more complex aggregates, current physical optimization techniques alone can no longer be seen as satisfactory. This is due to the fact that work from previously researched techniques do not adapt to reflect current data warehouse access trends. In contrary, this paper considers utilizing windowing [4] [5] as a means to derive an effective OLAP optimization technique by exploiting it's logical level application to adapt to the users current interests.

## 2. Related Work

Improving OLAP query performance has been a heavily researched area within the database community and a large variety of techniques have been introduced each with their own strengths and weaknesses.

The first form of optimization, *Physical Tuning*, has been categorised into *data warehouse tuning* [6] [7] and *query related physical tuning* [8] [9] [10]. Both categories have similar disadvantages due to physical nature that they share. The main disadvantages of physical tuning techniques are the issues of large initialization and storage space costs. These issues

978-0-7695-3701-6/09 $25.00 © 2009 IEEE
DOI 10.1109/ICCSA.2009.23

typically arise when implementing physical optimization techniques on large data warehouses due to the fact that physical optimization techniques are usually implemented to represent particular aspects of the data warehouse schema. Therefore, due to the large nature of data warehouses, physical optimization techniques need to represent a much larger problem space. Some of the techniques proposed address this issue by treating the weaknesses of physical tuning as a trade-off for performance and have created algorithms to determine an effective physical implementation whilst reducing the spatial repercussions [7]. This creates a different trade-off of initialization time versus an effective physical implementation due to the cost of computing effective implementations of the optimization techniques.

The second form of optimization, *Query Tuning* contains research that identifies a limitation introduced via the use of physical optimizations. Physically tuning a data warehouse allows for the speed up of OLAP query performance but if the user is unaware of the physical tuning implemented than there is no means for OLAP queries to speed up unless there is some form of query rewriting available [11] [12] [13] [14]. Given that, query tuning is a very effective optimization technique, because the extra time spent deriving more efficient methods of processing the query based on the underlying physical tuning is negligible in comparison to the response time improvement. Therefore the main issue involved in implementing query rewriting is the ability to derive an efficient usage of physical optimizations applied to the data warehouse. This issue has been addressed by a number of research papers that propose a large number of algorithms to determine a more effective methods of processing OLAP queries and is therefore a reasonably established area of research.

One area of research where a limited amount of work has been contributed is in the area of *combining previously researched techniques to tune OLAP queries*. In [15], the authors combined the techniques of partitioning, indexing and materialized views. The research provided an interesting set of results, as there are a number of cases where the best results were gained from only using two out of the three optimization techniques. It was also interesting to note that every time that when this phenomenon occurred, the optimal two techniques to combine were indexing and partitioning. A large number of optimization combinations have not been considered.

From the research conducted it was discovered that an optimization technique, which operates at a logical level with a physical presence, could greatly benefit data warehouse performance, as the technique would provide it's own form of optimization whilst still allowing you to use other physical optimization techniques concurrently. Another area of research that is relatively undeveloped involves the development of clinical data warehouses. Therefore, it is of interest to take into account some of the irregularities of a clinical data warehouse [16].

## 3. Motivation

The health informatics sector, like many others, is experiencing a larger fluctuation of incoming data, due to the greater number of requirements that the data is used for. Currently in Australia, Victoria, the Department of Human Services maintains several collections of health information such as, (i) the Victorian Admitted Episode Data Set (VAED), (ii) the Victorian Emergency Minimum Data Set (VEMD), (iii) the Elective Surgery Information System, and (iv) the Victorian Perinatal Data Collection [17]. The increase in available data has introduced the need to warehouse the data sets to make the data available for deeper analysis.

It should be immediately obvious that the data warehouses formed at almost any level of granularity for a clinical data warehouse will consume a large volume of space, which will cause query response times to greatly increase. Therefore, a clinical data warehouse will benefit greatly to any form of optimization technique that reduces the search space of queries such as partitioning, query rewriting and materialized views.

Secondly, given that clinical data warehouses like most data warehouses are developed to extract unknown information or information that may not be easily established by a user. Therefore an intelligent optimization technique, should take into account the expected data warehouse access patterns. It would be expected that access patterns when performing a deep analysis, such as data mining, would focus heavily on a select number of dimensions. This would continue until either the analysis being performed finishes executing or the user decides that they have had enough. Given that, it can be seen that the general access patterns for a deep analysis would involve a large number of accesses on select number of data sets, where the data sets of interest will change gradually based on a change in the requirements. Based on this pattern, the ability to track popular data sets would lend itself as a useful optimization mechanism.

For the research of this project, it is important for us to consider the design peculiarities of a clinical data

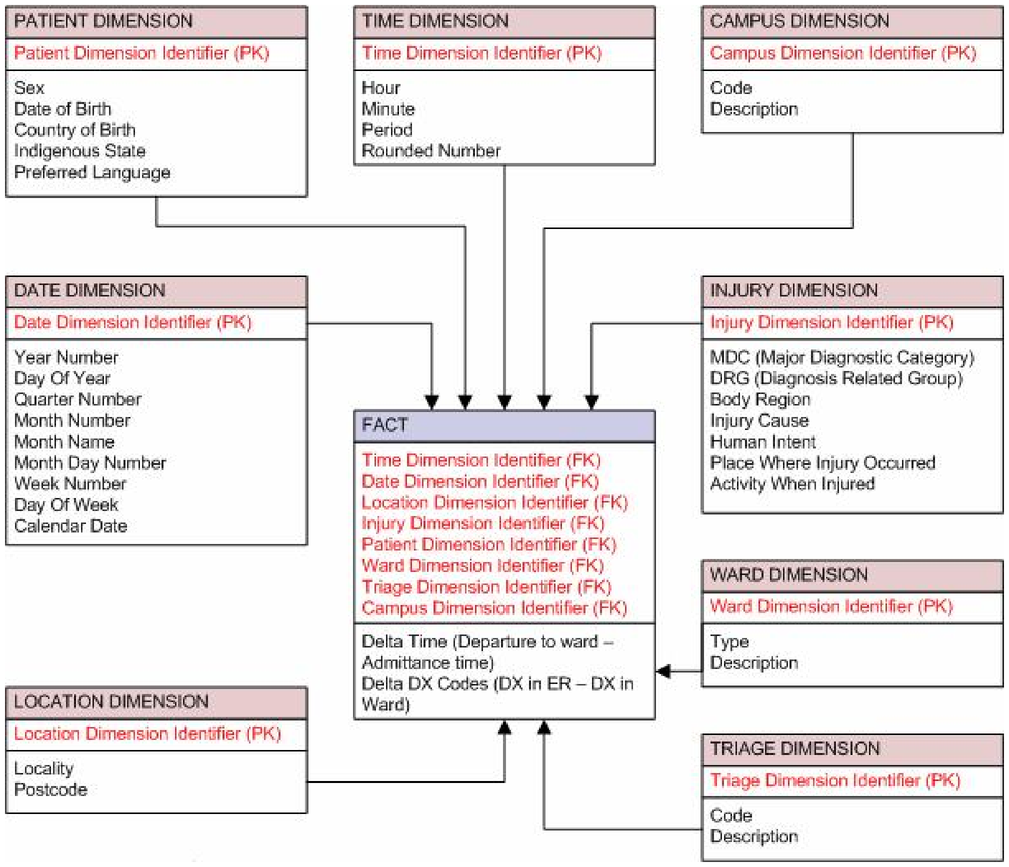

PATIENT DIMENSION
Patient Dimension Identifier (PK)
Sex
Date of Birth
Country of Birth
Indigenous State
Preferred Language
TIME DIMENSION
Time Dimension Identifier (PK)
Hour
Minute
Period
Rounded Number
CAMPUS DIMENSION
Campus Dimension Identifier (PK)
Code
Description
DATE DIMENSION
Date Dimension Identifier (PK)
Year Number
Day Of Year
Quarter Number
Month Number
Month Name
Month Day Number
Week Number
Day Of Week
Calendar Date
FACT
Time Dimension Identifier (FK)
Date Dimension Identifier (FK)
Location Dimension Identifier (FK)
Injury Dimension Identifier (FK)
Patient Dimension Identifier (FK)
Ward Dimension Identifier (FK)
Triage Dimension Identifier (FK)
Campus Dimension Identifier (FK)
Delta Time (Departure to ward – Admittance time)
Delta DX Codes (DX in ER – DX in Ward)
INJURY DIMENSION
Injury Dimension Identifier (PK)
MDC (Major Diagnostic Category)
DRG (Diagnosis Related Group)
Body Region
Injury Cause
Human Intent
Place Where Injury Occurred
Activity When Injured
WARD DIMENSION
Ward Dimension Identifier (PK)
Type
Description
LOCATION DIMENSION
Location Dimension Identifier (PK)
Locality
Postcode
TRIAGE DIMENSION
Triage Dimension Identifier (PK)
Code
Description

Figure 1. Clinical Data Warehouse Schema

warehouse. One of the most interesting design peculiarities is the process of de-identifying individual patient's information to protect the privacy of the patient. De-identifying a patients information includes either removing anything that uniquely identifies the patient or categorizing information of interest and reducing the categories to a set of codes. This inadvertently reduces the types of selections available when producing ad-hoc OLAP queries to range selections, which means that clinical data warehouses will benefit from secondary indexes due to the larger cardinalities from the categorization process. Generally, most commercial data warehouses will share the same characteristics, as it is extremely common for a data warehouse to contain sensitive information. Having said that, the clinical data warehouse schema, in Figure 1, provides a simple case study that depicts each of the data warehouse design issues that were mentioned.

There are a number of optimization techniques that individually address most of the current performance issues. Having said that, all of the techniques that are currently available do not produce a solution that is tailored to the current operations being performed on a data warehouse without introducing a large maintenance requirement.

Windowing is a technique commonly used to logically represent an active window of data. The characteristics of windowing, explored in Section 4, show a large level of promise as an optimization technique with only minor tweaks to the traditional windowing definitions, that introduce the ability to manage and store a number of windows in an intuitive and practical manner.

## 4. Windowing

The windowing technique implemented in a number of different DBMS [4] [5] provides a novel research area to consider exploring. The windowing techniques that have been recognised by researchers in the area [18] have entirely focused on the static declaration of windows. This is clearly due to the fact that these techniques were satisfactory for previous needs. Static windowing has proved to be a powerful technique in the field of networking, and data stream aggregation.

Having said that, windowing as it stands cannot be directly applied as an optimization technique, as there are a number of OLAP environment peculiarities that are not accounted for:

- 'Popular' data is a more meaningful term than 'Active' data for an optimization technique.
- OLAP queries can often have a large number of selections, which currently windowing would not support
- Statically created windows are not derivable using relational algebra.

These three important concepts show that current windowing functionality needs to be adapted and extended for it to be used for an optimization technique. The first point provides a context for how a window of information could be utilized within an optimization technique. A definition of both terms: Active Window and, Popular Window is provided below.

**Definition 1:** ***Active Window.*** A contiguous subset of data that is in focus of current operations.

**Definition 2:** ***Popular Window.*** A subset of data that does not have to be contiguous that is frequently accessed.

The second point mentioned that OLAP queries can often have a large number of selection criteria, for example `<attribute> = '<attribute-value>'`, which is not currently supported by windowing. This is important for a number of different reasons. One of the main reasons is because each of the selection clauses found in a query defines the result set from the query. Secondly, the result set of a query defines what data is popular within the data warehouse. Therefore, this indirectly states that the selection clauses of a query defines the popularity of the data found within a data warehouse. Based on these reasons, it will be useful to have functionality to define popular windows of information by defining popular selection clauses.

**Definition 3:** ***Conditional Window.*** A popular window of information that can be defined by a selection clause.

```
SELECT T.Hour, D.Day_Of_Week,
  COUNT(1) "Patients Admitted",
  Cum_Dist() OVER (PARTITION BY D.Day_Of_Week
                   ORDER BY SUM(COUNT(1)))
                   "Patient Admittance Rank"
FROM Fact F, Time_Dim T, Date_Dim D, Patient_Dim P
WHERE F.Time_Dim_Id = T.Time_Dim_Id AND
  F.Date_Dim_Id = D.Date_Dim_Id AND
  F.Patient_Dim_Id = P.Patient_Dim_Id AND
  (P.Indigenous_State = '2' OR
    P.Indigenous_State = '7')
GROUP BY D.Day_Of_Week, T.Hour;
```

Listing 1. OLAP Query Example

Conceptually, popular windows can be determined before queries have been executed. Figure 2(a) depicts the popular windows that would be derived if a query with the conditions `Indigenous_State='2'` and `Indigenous_State='7'` are present in the where clause. The conditions within an SQL where statement directly indicate the information that users are exploring will indicate which windows of data are popular.

The inclusion of conditional windowing solves the issues that were earlier stated, but not without introducing further complications. Statically defining a window will cause complications as statically defined windows can not be mapped to a conditional statement. A simple solution would be reducing the window to a collection of derivable windows, but then it is quite likely that some of the windows will not be complete(Partial).

**Definition 4:** ***Complete Window.*** A window of information that does not have to be contiguous and fully covers a subset of data that is defined by a selection clause.

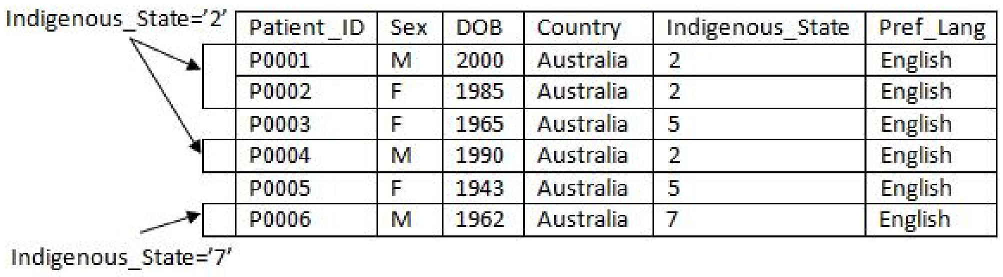
**Definition 5:** ***Partial Window.*** A window of information that does not have to be contiguous and does not fully cover a subset of data that is defined by a selection clause.

The example in Figure 2(b) shows a case where this has occurred, because a window has been created on just the first two records in the table. This information is not completely useless, in fact it is quite the opposite this data could lead to showing an extremely popular section of data as these two records may be queried thousands of times. Therefore, if it is known that the popular section of data is entirely contained within the popular window defined by the condition `Indigenous_State = '2'`, then this data set could be used to produce the same results instead of the entire tables records.

## 5. Windowing Data Structure Architecture

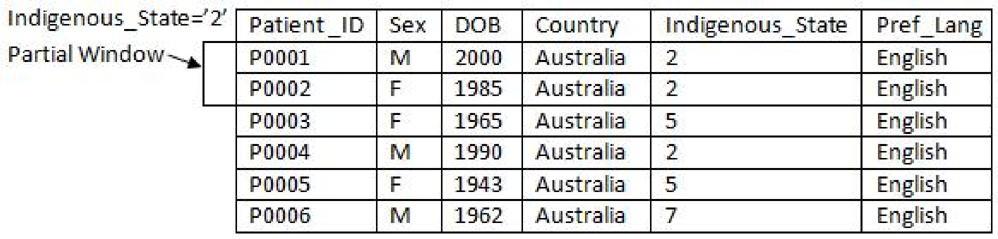
The windowing data structure architecture (WDSA), in Figure 3 is a direct translation of a common relational DBMS environment, as the hierarchy relationships that exist between tables, columns and popular windows of information are the same as what you would expect in a DBMS environment.

One advantage of representing the solution space via a WDSA, is that any search through the WDSA will inadvertently be more informed than if a generic data structure was used, due to the fact that to traverse through the WDSA the search method has to be aware of what type of object it is searching through. The following describes each of the object collections/constructions:


Indigenous_State='2'
Indigenous_State='7'

| Patient_ID | Sex | DOB | Country | Indigenous_State | Pref_Lang |
|---|---|---|---|---|---|
| P0001 | M | 2000 | Australia | 2 | English |
| P0002 | F | 1985 | Australia | 2 | English |
| P0003 | F | 1965 | Australia | 5 | English |
| P0004 | M | 1990 | Australia | 2 | English |
| P0005 | F | 1943 | Australia | 5 | English |
| P0006 | M | 1962 | Australia | 7 | English |

(a) Complete Window


Indigenous_State='2'
Partial Window

| Patient_ID | Sex | DOB | Country | Indigenous_State | Pref_Lang |
|---|---|---|---|---|---|
| P0001 | M | 2000 | Australia | 2 | English |
| P0002 | F | 1985 | Australia | 2 | English |
| P0003 | F | 1965 | Australia | 5 | English |
| P0004 | M | 1990 | Australia | 2 | English |
| P0005 | F | 1943 | Australia | 5 | English |
| P0006 | M | 1962 | Australia | 7 | English |

(b) Partial Window

Figure 2. Conditional Windowing Example

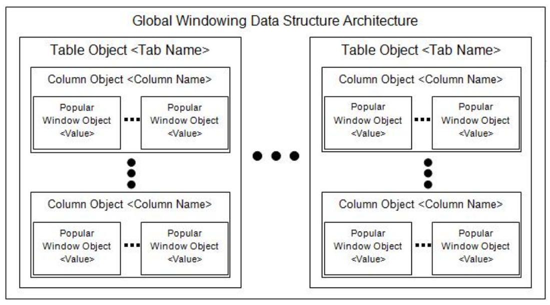

Global Windowing Data Structure Architecture
Table Object <Tab Name>
Column Object <Column Name>
Popular Window Object <Value>
Table Object <Tab Name>
Column Object <Column Name>
Popular Window Object <Value>

Figure 3. Windowing Data Structure Architecture

**Table Objects** represent a normal table environment and identifies the key aspects and characteristics involved.

**Column Objects** are used to depict a column environment. Given that, the column object is relatively simple.

**Popular Window Objects** are the finest grain object of the WDSA. The popular window object represents a popular window of information for a table column. The attributes of a popular window node define the condition that the window is based on along with the data set that it corresponds to. This allows for faster seeking, because as soon as a window is matched no further seeking needs to be performed to find information from that particular data set.

Each of the collections introduced by the WDSA can ideally be represented by any data structure that contains the full collection of objects, but for the sake of the contribution made in this paper a simple data structure configuration is considered to provide a clear performance evaluation, also the act of defining a data structure will restrict this technique to optimizing a particular aspect of an OLAP environment. Therefore, the architecture defined in Figure 3 identifies the granularities and node contents of each collection of objects, which allows the user to utilize different data structures.

## 6. Windowing Data Structure Architecture Operations

The operation that is likely to be performed most frequently using the WDSA is searching for popular windows of information, as this is the basis for which the architecture was designed. This operation is defined in Algorithm 1.

**Algorithm 1:** ***Search WDSA***

```
Require: $windowConditionObject$: winCond,
    $tableDataStructure$: GlobalWS
 1: Let $T$ be used to check if the $tableObject$ exists
 2: Let $C$ be used to check if the $columnObject$ exists
 3: $T \leftarrow searchTables(GlobalWS, winCond.Table)$
 4: if T $\neq$ null then
 5:     $C \leftarrow searchColumns(T, winCond.Column)$
 6: end if
 7: if C $\neq$ null then
 8:     return $searchPopWindows(C, winCond.Value)$
 9: end if
10: return null
```

| Object | Attribute | Description |
|---|---|---|
| *Table* | Table Name | Name of the table that the object represents. |
| | Primary Keys | List of primary key column names so that they are identifiable from other columns. |
| | Columns | Collection of column objects that the table consists of. |
| | Children Tables | Collection of table objects that have a foreign key relationship with this table. |
| | Parent Tables | Collection of table objects that this table has a foreign key relationship with. |
| *Column* | Column Name | Name of the column that the object represents. |
| | Popular Windows | Collection of popular window objects that this column consists of. |
| *Popular Window* | Attribute Value | Conditional value that the window is based on. |
| | Data Set | List of records that this popular window contains. |
| | Hit Counter | Integer that monitors the number of times that this window has been referenced. |
| | Last Access | Date that monitors the windows last access. |

Table 1. WDSA Object Attribute Descriptions

Algorithm 1 performs three simple searches, one for each data structure granularity, to retrieve a single popular window of information. It should be noted that the search processes through each data structure granularity is subject to the data structure that is implemented.

So far a number of concepts and processes have been introduced that describe what is a WDSA as well as its functionality, up until now the context of the concepts and processes have not been considered. Therefore, before diverging into a method of processing OLAP queries, the processes have been placed into context below based on the order that they are executed to provide a general sequence of events.

The general sequence of events when implementing a WDSA can be broken down into the following steps:

1) Receive ad-hoc OLAP queries
2) Analyse the query received to extract the explicitly state windows
3) Store the explicitly stated windows in a WDSA
4) Process the query received using the WDSA
    a) Derive the individual table sets
    b) Join the table sets as specified by the OLAP query
5) Apply aggregates and projections to finalise results

Step four is broken down in this manner, as both of the steps in the sub sequence can be performed using the WDSA predominantly without having to communicate with the data warehouse. Step five is then performed on the result sets obtained from step four by the DBMS. The process of deriving individual table result sets, Step 4a, is represented by Algorithm 2.

The algorithm is split into two major tasks, which can be observed from the two highest level 'for each' statements on lines 2 and 13. The first task recognised by Algorithm 2 is the process of grouping the window conditions based on the table that it applies to. Once the window conditions have been group each windowConditionSet is reduced to a singular table result set.

**Algorithm 2:** ***Create table results using WDSA***

```
Require: $windowConditionObjectSet$: winConds,
    $tableDataStructure$: GlobalWS
 1: Let $tableSBSet$ be a $windowConditionTableSet$s that is
    used to group the window conditions by table
 2: for all elements wc $\in$ $windowConditionObject$ winConds
    do
 3:     if wc.TableName $\in$ tableSBSet.TableName then
 4:         $tableSBSet.winConds \leftarrow$
            $tableSBSet.winConds \cup wc$
 5:     else
 6:         Let $tableSB$ be a $windowConditionObjectSet$
 7:         $tableSB.TableName = wc.TableName$
 8:         $tableSB.winConds = wc$
 9:         $tableSBSet \leftarrow tableSBSet \cup tableSB$
10:     end if
11: end for
12: Let $tableRS$ be a $popularWindowSet$
13: for all elements sb $\in$ windowConditionSet tableSBSet do
14:     Let $tRes$ be a $popularWindowObject$
15:     for all elements wc $\in$ windowCondtionObject sb do
16:         if $tRes = null$ then
17:             $tRes \leftarrow searchWS(wc, GlobalWS)$
18:         else
19:             $tRes \leftarrow tRes\forall^{1} searchGWS(wc, GlobalWS)$
20:         end if
21:     end for
22:     $tRS \leftarrow tableRS \cup tRes$
23: end for
```

The second step in processing OLAP queries using a WDSA is joining tables based on the query that was received. Algorithm 3 introduces a methodology

1. symbol denotes that the operator applied between conditions that correspond to the popular windows within the query should also be applied here

of performing this task on a single join. Therefore, any number of joins can be processed by executing the algorithm as many times as needed.

**Algorithm 3: *Join two table result sets using the WDSA***

```
Require: $joinConditionObject$: jc,
    $tableDataStructure$: GlobalWS,
    $popularWindowSet$:tableRS
 1: Let $t1F$ be a $Boolean$
 2: Let $t2F$ be a $Boolean$
 3: Let $t1$ be a $tableResult$
 4: Let $t2$ be a $tableResult$
 5: $t1F := false$
 6: $t2F := false$
 7: for all elements r $\in$ tableResult tableRS AND !(t1F AND
    t2F) do
 8:     if $jc.Table1 = r.TableName$ then
 9:         $t1 \leftarrow r$
10:         $t1F := true$
11:     else if $jc.Table2 = r.TableName$ then
12:         $t2 \leftarrow r$
13:         $t2F := true$
14:     end if
15: end for
16: if !t1F AND !t2F then
17:     Perform a normal join in the database
18: else
19:     if !t1F then
20:         $t1 :=$
            $popularWindowObject(jc.Table1, jc.PK, all)$
21:     else if !t2F then
22:         $t2 :=$
            $popularWindowObject(jc.Table2, jc.FK, all)$
23:     end if
24:     Let $tRes$ be a $popularWindowObject$
25:     for all elements pk $\in$ String t1.dataSet do
26:         Let $w$ be a $popularWindowObject$
27:         $w :=$
            $popularWindowObject(jc.Table2, jc.FK, pk.value)$
28:         $tRes \leftarrow tRes \cup searchGWS(GlobalWS, window)$
29:     end for
30:     $tRes.dataSet \leftarrow table2.dataSet \cap tRes.dataSet$
31:     $results \leftarrow fetchSQLResults(tRes)$
32:     $results \leftarrow$
        $results \cup fetchSQLResults(results \bowtie_{FK=PK} t1)$
33: end if
```

The join process is again broken up into two major operations. The first task delineated by Algorithm 3 selects the two table result sets that are going to be pajoined from the set of table result sets produced by Algorithm 3, based on the tables identified by the joinConditionObject. The first task is represented by the instructions found between line 2 and line 16.

Line 17 and onwards, continues to utilise both the table result sets selected, to reduce both table result sets to represent the final table result sets that would be used after a join has occurred. To complete the task the algorithm first determines whether the table result sets are available, on lines 19 to 23. If both table result sets are available then a temporary result set is used to determine the final table result sets that would be obtained by joining one table to the other.

The temporary result set is created by utilizing the result set of table one, as this is the table that is joining via the use of one of its primary keys. This means that each of the primary keys from table one's result set can be used to directly search through table two's foreign key column objects for the corresponding implicit popular window. Given that, to construct the complete temporary result window all of the popular windows found are union together. The temporary result set can then be intersected with table two's result set to derive the final result set for table two, as long as no further joins are to be performed on table two.

When deriving the final result set for table one communication finally needs to made with the data warehouse, as the result set for table two is retrieved so that a list of foreign keys can be produced. The list of foreign keys can then be intersected with the list of primary keys to provide the final result set for table one, again as long as no further joins are to be performed on table one. As mentioned previously, Algorithm 3 can be executed for each join that is stated in the query that was received. Therefore, once the final result set is obtained for each table any projections or aggregations defined by the received query can be applied.

## 7. Evaluation

The evaluation provided will focus on the gain in performance obtained from utilizing a WDSA when compared to the traditional nested loop join and the frequently implemented hash join processing techniques.

In order to define a collection of cost models that depict the behaviour of the WDSA technique and the expenses involved in utilizing the technique, Table 2 defines a set of notation, along with relevant descriptions that act as the foundation for each of the cost models introduced. The notation has been divided into four subcategories for ease of understanding; system and data parameters, WDSA parameters, query parameters and time unit costs. For the purpose of our evaluation in this section, the basic parameters and cost models for the traditional nested loop and hash join processing have been adopted from [19].

Some constraints that have been placed on the parameters, from Table 2, include those of the selectivity and sparsity ratios found in the query parameters category, as they are restricted to real values ranging from zero to one. One constraint that is not evident from the list of parameters is due to the fact that a number of the parameters are directly related. Data sparsity ratio is one of these notations that is directly

| Parameter | Description |
|---|---|
| *System and Data Parameters* | |
| Mips | MIPS of the processor |
| R and S | Size of tables R and S |
| $\|R\|$ and $\|S\|$ | Number of records in tables R and S |
| P | Page Size |
| H | Maximum hash table entries |
| *WDSA Parameters* | |
| T | Size of table objects table |
| C | Size of the column objects table |
| PW | Size of the popular window objects table |
| $\|T\|$ | Number of records in table T |
| $\|C\|$ | Number of records in table C |
| $\|PW\|$ | Number of records in table PW |
| *Query Parameters* | |
| $\sigma_{TC}$ | Join selectivity between tables T and C |
| $\sigma_{TCP}$ | Join selectivity between tables C and PW |
| $\sigma_{wR}$ and $\sigma_{wS}$ | Join selectivity of table result set on tables R and S |
| $\sigma_j$ | Join selectivity ratio between tables R and S |
| $\sigma_{jw}$ | Join selectivity ratio between R and S's result windows |
| $N_R$ and $N_S$ | Number of windows that correspond to tables R and S |
| $\theta_R$ and $\theta_S$ | Data sparsity ratio for tables R and S |
| $\|PWK_n\|$ | Number of records in popular window n on table K |
| *Time Unit Cost* | |
| IO | Effective time to read a page from disk |
| $t_r$ | Time to read a record |
| $t_c$ | Time to compare two records |
| $t_w$ | Time to write a record |
| $t_u$ | Time to union a record |
| $t_h$ | Time to compute a hash value |
| $t_j$ | Time to compare a record with a hash table entry |

Table 2. Parameter Descriptions

influenced by a number of other parameters which is explored in more detail later.

## 7.1. WDSA Search Cost Model

Scanning the WDSA plays a vital role when processing queries as it is a frequently performed operation. Therefore, it is equally as important to deduce a cost model that represents the search performance behaviour. This paper considers a WDSA instance that uses a binary search to seek through each of the WDSAs object collections.

The first phase of searching a WDSA involves loading the relevant data to be scanned. This phase of query execution is captured by the scanning and selection cost models defined in Table 3. Therefore, the cost models defined in Table 3 are a simple addition of binary search costs via each of the object collections.

After the data has been loaded into memory, it is then determined which records being scanned meet the requirements to become part of the result set. The comparison cost encompasses this phenomenon, since comparisons are performed on all records that are selected the same formula can be used with a different operation cost.

Searching the WDSA will always result in a singular popular window information being selected for this WDSA instance. This means that there will simply be three objects written to memory for every search that is performed on the WDSA. Once every popular window of data has been obtained their data sets are union together to create a singular result set.

## 7.2. WDSA Processing Costs

The cost models derived in the previous section provides a sound platform to establish a new collection of cost models that represent the behaviour of processing a join between two tables whilst utilising a WDSA. Given that, the formulas provided in Table 4 depict the expenses involved in processing a join between two tables. For simplicity, the cost formulas have been split into two categories; join processing and WDSA search costs.

The cost models provided for joining two table result windows together, in Table 4, implements a nested join solution, as any join process can be used because it is still just joining two data sets. Therefore, using a

| **Cost Phase** | **Binary Search Costs** |
|---|---|
| **Scanning (WDSAFetch)** | $\text{IO}\left[\frac{\log_2(T)}{P}+\frac{\log_2(C\sigma_{TC})}{P}+\frac{\log_2(PW\sigma_{TCP})}{P}\right]$ |
| **Selection (WDSASelect)** | $(t_r+t_w)\ [\log_2(\lvert T\rvert)+\log_2(\lvert C\rvert\sigma_{TC})+\log_2(\lvert PW\rvert\sigma_{TCP})]$ |
| **Comparison (WDSAComparison)** | $t_c\ [\log_2(\lvert T\rvert)+\log_2(\lvert C\rvert\sigma_{TC})+\log_2(\lvert PW\rvert\sigma_{TCP})]$ |
| **Writing (WDSAWrite)** | $3\times t_w$ |
| **Union (WDSAU(X))** | $t_u\left[\sum_{n=1}^{N_K}\lvert PWK_n\rvert\right]$ |

Table 3. Costs Of Binary Search WDSA

| **Cost Category/Phase** | **Result Window Join Costs With Join Result Preprocessing** |
|---|---|
| *Join Processing Cost* | |
| **Scanning** | $\text{IO}\left[\frac{\sigma_{wR}^{1-\theta_R}\sigma_{jw}^{1-\theta_R}(S+R)}{P}+\frac{\sigma_{wS}^{1-\theta_S}\sigma_{jw}^{1-\theta_S}S}{P}\right]$ |
| **Selection** | $(t_r+t_w)\,[\sigma_{wR}\sigma_{jw}\,(\lvert\text{S}\rvert+\lvert\text{R}\rvert)+\sigma_{wS}\sigma_{jw}\lvert\text{S}\rvert]$ |
| **Reading** | $t_r\,[\sigma_{wR}\sigma_{jw}\,(\lvert\text{S}\rvert+\lvert\text{R}\rvert)+\sigma_{wS}\sigma_{jw}\lvert\text{S}\rvert]$ |
| **Comparison** | $t_c\,[\sigma_{wR}\sigma_{jw}\lvert\text{R}\rvert\times\sigma_{wS}\lvert\text{S}\rvert]$ |
| **Union** | $t_u\,[\sigma_{wR}\sigma_{jw}\lvert\text{R}\rvert+\sigma_{wS}\lvert\text{S}\rvert]$ |
| **Writing** | $t_w\,[\sigma_{jw}\sigma_{wR}\lvert\text{R}\rvert\times\sigma_{wS}\lvert\text{S}\rvert]$ |
| **IOWriting** | $\text{IO}\left[\frac{\sigma_{jw}\sigma_{wR}R\times\sigma_{wS}S}{P}\right]$ |
| *WDSA Search Cost* | |
| **Scanning** | $(\text{N}_R+\text{N}_S+\sigma_{wR}\lvert\text{R}\rvert)$WDSAFetch |
| **Selection** | $(\text{N}_R+\text{N}_S+\sigma_{wR}\lvert\text{R}\rvert)$WDSASelect |
| **Comparison** | $(\text{N}_R+\text{N}_S+\sigma_{wR}\lvert\text{R}\rvert)$WDSAComparison |
| **Writing** | $(\text{N}_R+\text{N}_S+\sigma_{wR}\lvert\text{R}\rvert)$WDSAWrite |
| **Union** | WDSAU(R) + WDSAU(S) |

Table 4. WDSA Join Processing Cost Models

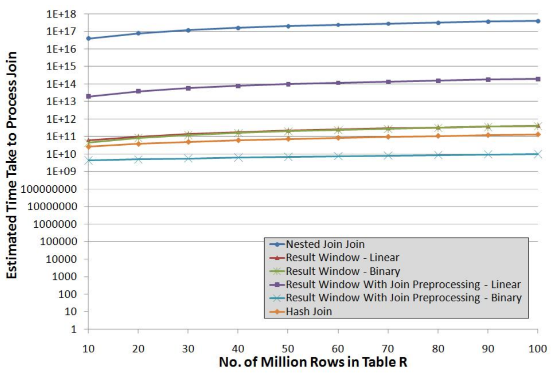
nested join produces the most traditional and simplistic approach.

An interesting concept that was necessary to capture is that the table result sets produced by the WDSA is not likely to be contiguous sets of results, due to the fact that a table result set is a subset of the original table. This means that for the first phase of joining two table result sets together, cannot be taken for granted that the minimum amount of pages can be retrieved. Therefore, the concept of a data sparsity ratio($\theta_X$) was introduced for both of the scanning cost models. The aim of this notation is to accurately represent the best and worst cases for data sparsity, which when coupled with the constraint below is provided.

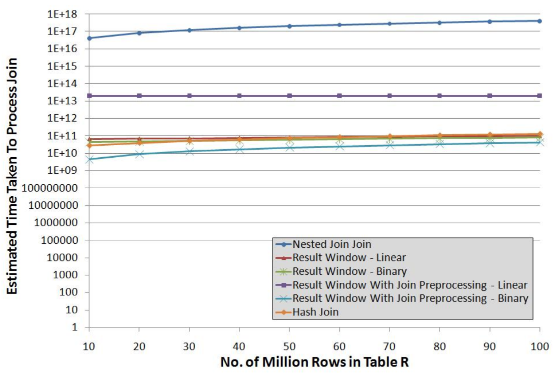
The following constraint defined restricts data sparsity to a real value between 0 and 1, with a restriction placed on the maximum data sparsity to compensate for when a window of information is relatively small and contains less records than there are pages that exist for the table. This provides a more realistic mapping of the maximum sparsity value, because it allows for the relational dependency that the sparsity ratio has with the number of pages in a table, and the number of records in a popular window on that table.

$$\theta_X = \left| \begin{array}{ll} ContiguousData & 0 \\ & \\ SparseData & \\ \frac{|X|\times\sigma_{wX}}{\frac{X}{P}} < 1 & \frac{|X|\times\sigma_{wX}}{\frac{X}{P}} \\ \frac{|X|\times\sigma_{wX}}{\frac{X}{P}} >= 1 & 1 \end{array} \right. \qquad (1)$$

Algorithm 3 introduced the concept of using table result sets to process the effect of joins on the parent table before sending a data request to the data



(a) Fixed Window Selectivity



(b) Fixed Window Size

Figure 4. Total Processing Cost vs. Number of Rows in Table R

warehouse. The cost of effectively performing this operation has been formulated down into a number of costs in table 4. The total cost can be calculated by simply summing each of the formulas. Table 4 has categorised the costs into 'Join Processing Costs' and 'WDSA Search Costs'. Given that, the 'Join Processing Costs' reflect the costs of the join operation between the two table result sets given that they already exist. The 'WDSA Search Costs' indicate the total WDSA search costs involved when creating the two table result sets and when preprocessing the join between the two.

## 7.3. Results

Based on the cost models introduced in the previous section a sensitivity analysis has been completed. The analysis performed, considered parameters of interest and varies the parameters values to determine the expected output based on the new configuration. The results from the sensitivity analysis will supply observable proof that the WDSA provides a large potential for an increase in performance when compared with two previously existing join processing techniques; nested join and a hash join.

In order to produce a viewable distinction between each of the techniques performance capabilities, the graphs in this section have utilized a logarithmic scale. Which means that they do not provide a direct indication of the estimated performance gain between each of the techniques, although it does provide a strong indication of the orders of magnitude difference between each of the techniques, as well as the order of expected performance.

The results found in Figure 4, provide an indication of the relative time taken to perform a join for each processing technique. It can be seen that the binary result window join process with join result preprocessing (RWJPJRP) easily provides the best performance by orders of magnitude from the original nested join technique. This is due to the innovative method of storing data that the WDSA provides, as it provides an intuitive environment that delivers the possibility to perform more informed operations in a clinical data warehouse, by exploiting data warehouses large datasets and large relational cardinalities, which are both extremely common within a clinical data warehouse due to de-idenitification procedures.

## 8. Conclusion

This paper defined a new processing architecture known as the WDSA, which was created to exploit the characteristics of the extended version of the existing windowing technique. The effectiveness of implementing a WDSA in a clinical warehouse environment was depicted via an extensive analysis of the potential performance benefits using cost models. From this, it was observed that the WDSA severely reduced the search space without incurring much processing overhead in the worst case scenario, which means that the WDSA provided a significant improvement in most cases.

Having said that, there were still a number of areas that have been identified where further research could prove to further establish this techniques potential to increase OLAP query performance. Firstly, the WDSA could easily be implemented in conjunction with a number of other physical optimization techniques due to the fact that the popular windows operate at a logical level. Another interesting area of research would be determining efficient and practical data structure combinations, for the table, column and popular window collections. Lastly, research can continue into WDSA implementation, as it may be worth considering the effects of using a WDSA as the data warehouse so that the whole data warehouse is represented inside a WDSA.

## References

[1] N. Pendse, ”What is OLAP? An analysis of what the often misused OLAP term is supposed to mean,” The OLAP Report. Available: http://www.olapreport.com/fasmi.htm. Accessed April 21, 2008.

[2] Sun Microsystems, ”Worldś largest data warehouse: sun$^{TM}$ data warehouse reference architecture - scalability over 1 PB,” Sun Microsystems. Available: http://www.sun.com/servers/sparcenterprise/data_warehouse_solution_brief.pdf. Accessed July 25, 2008.

[3] *Health Informatics – Deployment of a clinical data warehouse*, ISO/PDTS Technical Specification 29585, 2008.

[4] IBM Corp., ”DB2 Database for Linux, Unix, and Windows,” IBM Corp. Available: http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp. Accessed April 21, 2008.

[5] P. Lane, et al., *Oracle 10g data warehousing guide, Release 1,* Redwood City: Oracle Corp., 2007.

[6] D. J. Abadi, A. Marcus, S. Madden and K. J. Hollenbach, ”Scalable semantic web data management using vertical partitioning,” in *Proceedings of International Conference on Very Large Databases (VLDB)*, 2007, pp 411–412.

[7] L. Bellatreche, K. Boukhalfa and H. I. Abdalla, ”SAGA: A combination of genetic and simulated annealing algorithms for physical data warehouse design,” in *Proceedings of British National Conference on Databases (BNCOD)*, 2006, pp 212–219.

[8] T. S. Jung, M. S. Ahn and W. S. Cho, ”An efficient OLAP query processing technique using measure attribute indexes,” in *Proceedings of International Conference on Web Information Systems Engineering (WISE)*, 2004, pp 218–228.

[9] R. Elmasri, S. B. Navathe, *Fundamentals of database systems (5th Edition).* Boston: Addison-Wesley Longman Publishing Co., Inc., 2006.

[10] P. E. O'Neil, ”Model 204 Architecture and Performance,” in *Proceedings of International Workshop on High Performance Transaction Systems*, 1987, pp 40–59.

[11] C. Park, M. Kim, Y. Lee, ”Rewriting OLAP queries using materialized views and dimension hierarchies in data warehouses,” in *Proceedings of International Conference on Data Engineering (ICDE)*, 2001, pp 515–523.

[12] D. Theodoratos and A. Tsois, ”Heuristic Optimization of OLAP Queries in Multidimensionally Hierarchically Clustered Databases,” in *Proceedings of International Workshop of. Data warehousing and OLAP*, 2001, pp 48–55.

[13] F. Akal, K. Böhm and H. Schek, ”OLAP query evaluation in a database cluster: A performance study on intra-query parallelism,” in *Proceedings of East-European Conference Advances in Databases and Information Systems (ADBIS)*, 2002, pp 181–184.

[14] A. A. B. Lima, M. Mattoso and P. Valduriez, ”OLAP query processing in a database cluster,” in *Proceedings of European Conference on Parallel Processing (Euro-Par)*, 2004, pp 355–362.

[15] L. Bellatreche, M. Schneider, H. Lorinquer and M. K. Mohania, ”Bringing together partitioning, materialized views and indexes to optimize performance of relational data warehouses,” in *Proceedings of International Conference on Data Warehousing and Knowledge Discovery (DAWAK)*, 2004, pp 15–25.

[16] A. H. Landberg, H. Grain, J. W. Rahayu, and E. Pardede, ”Incorporating Privacy Support into Clinical Data Warehouse”, *electronic Journal of Health Informatics*, Accepted 22 November 2008.

[17] Victorian Government Health Information, ”Information about health data standards and systems (hdss) used in victoria's hospitals 2008,” Victorian Government Health. Available: http://www.health.vic.gov.au/hdss/index.htm. Accessed May 1, 2008.

[18] I. Botan, P. M. Fischer, D. Florescu, D. Kossmann, T. Kraska and R. Tamosevicius, ”Extending XQuery with window functions,” in *Proceedings of International Conference on Very Large Databases (VLDB)*, 2007, pp 75–86.

[19] D. Taniar, C. H. C. Leung, W. Rahayu and S. Goel, *High Performance Parallel Database Processing and Grid Databases, John Wiley & Sons, 2008.*