Information Lifecycle Management in Evolving Healthcare Databases

Nitin Chandra Badam[+]
IIT Guwahati
Assam, India
chandra.nitin@iitg.ernet.in

Aradhna Kumari
Sarada Research Labs
Bangalore, India
aradhna.kumari@saradaresearchlabs.org

Jagannathan Srinivasan
Oracle Corporation
Nashua, NH, USA
jagannathan.srinivasan@oracle.com

Abstract

Information Lifecycle Management is a must for ever growing healthcare databases. In practice, this problem is further compounded due to the evolving nature of these databases, where the schema itself evolves over time. In this paper, we describe PRLM (Patient Record Lifecycle Management), a tool developed to manage lifecycle of patient visit records of Ramakrishna Mission Sevashrama Hospitals at Vrindaban, and Kankhal. The key features of the PRLM tool are: i) a customizable scheme for information lifecycle management of healthcare databases ii) tool works on evolving databases, and iii) incurs minimal downtime by leveraging underlying RDBMS utilities for data movement.

1. Introduction

Information Lifecycle Management (ILM) is an active area of research covering processes and technologies that help manage information to meet the needs of an enterprise and to comply with legal and regulatory mandates regarding data accessibility and retention (such as Sarbanes-Oxley Act [1], Health Insurance Portability and Accountability Act (HIPAA) [2]). Both hardware ILM solutions (IBM [3], and Symantec [5]) as well as software ILM solutions (Oracle ILM Assistant [6], and SAP ILM [7]) have been developed.

In this paper, we consider the problem of lifecycle management of patient visit records. Specifically, we have deployed a database-centric web application, Patient Services Accounting System (PSAS) [4], at Ramakrishna Sevashrama Hospitals (RKMS) at Kankhal, Uttarakhand [16] and Vrindaban, Uttar Pradesh [17] to manage both OPD and IPD patient visit records. The complete visit information including patient registration, service registrations, discharge, billing, payment, and clinical information (such as lab test results, operations, etc.) is maintained. The visit data resides on a server accessible by clients over the hospital campus LAN.

To reduce the total cost of ownership, we use the free Oracle Database 10gR2 Express Edition (Oracle XE [8]) and at clients only a web browser is needed. However, Oracle 10gR2 XE limits the size of the database to 4GB and utilizes a single CPU of the machine for execution. As the number of patient visit records grows, we face two problems:

- The performance of complex reports degrades, and
- Eventually we will reach the database size limit imposed by Oracle XE.

To circumvent these problems, we have developed a Patient Record Lifecycle Management (PRLM) tool, which can be used to control the number of patient visit records that are retained on server by pushing data beyond a chosen cut-off point to an archive database. We provide read-only access to archive data, which is also held in an Oracle database residing on a separate server on the same LAN (Figure 1). In addition, for very old data, we provide the option of altogether purging the patient visit records.

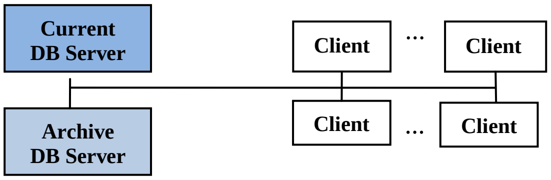


Figure 1: Deployment at RKMS Hospitals

Thus, the tool primarily supports purge and archive operations. In addition, the tool allows supports a merge archives operation to merge the successive archives generated by the PRLM tool thereby allowing a single archive server to host the archive database.

Challenging aspects in development of the tool are:

- Underlying database is continuously evolving so that

The 19th International Conference on Management of Data (COMAD), 19th-21st Dec, 2013 at Ahmedabad, India.

[+] This work was done as part of a summer internship at Sarada Research Labs, Bangalore.

subsequent PRLM operations, which usually occur after six to nine months, should work on the evolved database. The evolution typically includes addition of new tables owing to automation of new departments, or modification to existing tables such as addition of columns or changing of column data types.

- The PRLM tool should work on hospital databases managed by the PSAS software at two different hospitals, which has similar yet different data and growth characteristics.
- For the archive operation, we need a quiet point so the hospital database is brought down. Thus, the archive operation needs to be performed with minimal time to reduce the downtime.
- The total cost of ownership must be kept low as the hospitals managed are charitable hospitals.

Taking these aspects into account we have developed the PRLM tool whose key characteristics are: i) supports a customizable scheme for information lifecycle management of healthcare databases ii) works on evolving databases, and iii) incurs minimal downtime by leveraging RDBMS utilities for data movement.

The PRLM tool was developed as a web application using Oracle Application Express (APEX) [9], a rapid application development tool, using Oracle XE Database.

We have successfully applied the PRLM tool to perform archive and merge archive operations on two hospital databases of RKMS Hospitals at Vrindaban and Kankhal. Also, to illustrate its working we present an experimental study by applying PRLM tool on real hospital data that shows the effectiveness of archiving algorithm as well as benefits of archiving/purging.

The key contributions of this work are:

- A simple and customizable schema-aware archiving and purging scheme for patient visit records,
- A scheme for allowing merging of archives generated by successive PRLM operations,
- Support for archive, purge, and merge archive on evolving hospital databases, and
- An experimental evaluation using real hospital data.

The rest of the paper is organized as follows. Section 2 presents the key concepts. Section 3 covers the design and implementation. Section 4 gives a tour of dashboard-style PRLM tool. Section 5 reports the experimental study. Section 6 covers the related work and Section 7 concludes the paper and outlines the future work.

2. Key Concepts

This section discusses the key concepts and presents the basic scheme for PRLM operations.

2.1 The Basic Model

Each record of a table represents a unit of information that has a value when it comes into existence, which typically decreases with time. Child records (connected by referential constraint), typically inherit the value from the parent (or ancestor) record. In some cases, child records may have intrinsic value independent of the parent, in which case the maximum of the two has to be considered as the value of that information.

The PRLM strategy partitions the records containing information into current and archive based on a value threshold. Archive data is stored on archive server as a read-only database where as current data is retained in the server as a read-write database (Figure 1).

2.2 Unit of Information

In our case, all information pertaining to a single patient visit (either OPD or IPD) forms a unit of information, whose life cycle needs to be managed (Figure 2). A visit entry of a patient can be viewed as birth of a visit record. From visit_start_date to visit_end_date, the record will be in active state. After visit_end_date, record keeps on aging. Finally after sometime, the record is no longer needed and hence it can be purged (death of the record).

Birth
Active
Aging
Death

Figure 2: Patient Visit Record Life Cycle

2.3 PRLM Operations

Figure 3 gives an overview of the PRLM operations.

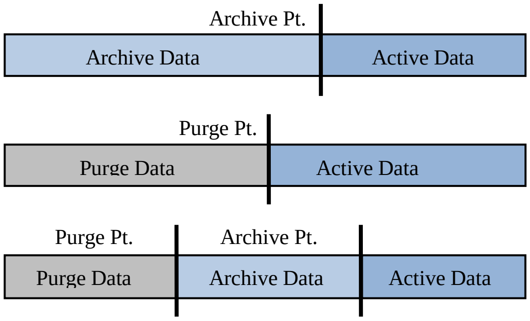


Figure 3: PRLM operations and types of data

Specifically there are four key operations:

1. *Archive only*: the visit data is partitioned into two parts, active data and archive data. The data after archive point is the current (active) data and the data before archive point is archived.
2. *Purge only*: the visit data gets partitioned into two parts active and purge data. The data after purge point is current (active) data and the data before purge point is purged (deleted).
3. *Purge and archive:* When the tool does both archive and purge simultaneously, the visit data gets partitioned into three parts, active data, archive data and purge data. The data after archive point is active data, between archive and purge point is archived, and the data before purge point is purged (Figure 3).

4. *Size Estimator:* Prior to performing the operation, for a given cut-off point, the tool estimates the expected reduction in size. This is useful in choosing the PRLM retention, archive, and purge policies.

Additionally, the PRLM tool supports *Merge Archives* operation, to merge archives generated from two successive PRLM archive operations (Figure 4). Thus, archive data can be hosted in a single archive database.

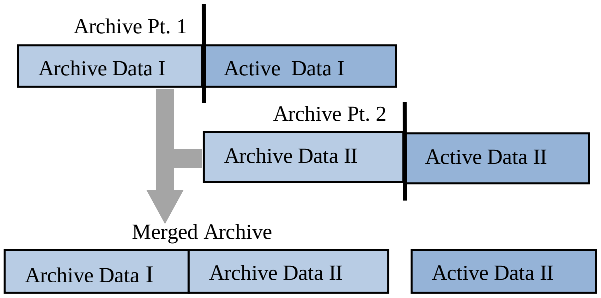


Figure 4: Merging of Successive Archives

2.4 Visit atomicity

It means that the entire visit should belong either to archive partition or to current partition. This works fine for visit, which entirely falls before (v1) or after archive point (v4). However, for a visit which partially falls in both regions (v2, v3), we use the following criterion for deciding where it belongs. For visit start date before archive point, but the visit is active (that is, patient is still taking treatment: v3), the visit record cannot be archived, since archived records go into read-only state. But a patient with visit entry date before archive point and is currently discharged (v2) can be archived since his record is frozen and needs no more modification.

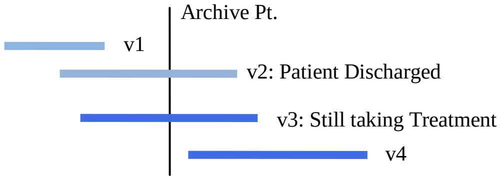


Figure 5: Partitioning of Patient Visits

For purge point, the visit record partitioning follows the same semantics as for archive point above, if the purge is being done on current data. However, if purge operation is done on archive data, which is by definition devoid of active visits (such as v3, v4), we simply partition by visit_start_date. However, in this case, since the purged data would be lost forever, a conservative strategy of partitoning by visit_end_date could also be considered.

3. Design

This section discusses the basic scheme for the purge, archive, and archive merge operations.

3.1 Table Categorization for PRLM

The tables in the database have been divided into three categories to facilitate PRLM operations:

1. *Archive truly*: The tables contain only active data and the archive data is moved to archive DB. Tables with historical data and which show significant growth in their data are classified as archive-truly e.g. VISITS table of PSAS database, which holds visit information. If a table is classified as archive-truly, then its child tables should either be archived or truncated, else the foreign key of the child table may refer to a non-existing primary key of the parent table. Since we do not want to lose any data, we archive the child tables rather than truncating them and thus they are classified as archive truly tables.

2. *Archive and retain completely*: The tables, for which the complete data is retained in current DB and a copy of that is also maintained in archive DB. Tables, whose data is always active is classified as archive and retained completely, for example. DOCTORS, and PATIENTS tables of PSAS database. Tables in this category are typically the metadata tables and they should be present in both current and archive DB completely and be kept up-to-date. Deleting any record from such tables can create orphan records in the database. Thus instead of delete, we mark the status of that particular record as inactive.

3. *Pseudo-Archive and retain completely*: The tables, which although belonging to archive truly category are treated as archive and retain completely to reduce the amount of individual table data movement. It is assumed that these are not fast growing tables. Tables in this category do not have referential constraints explicitly asserted, which gives us the flexibility of having them present without parent records as well.

This classification among the tables is used by the PRLM tool to complete the purge or archive operation with minimum amount of individual table data movement. These tables are archived or purged, based on the cut-off date specified and the existing parent-child relationships among these tables (see Figure 6).

From PRLM point of view the task is to partition archive truly tables only. Thus, data is selectively removed from archive truly tables, that is, each visit record along with all descendent records will be co-located with it to preserve the atomicity of the visit information. The rest of the tables are fully present at both current and archive DB.

The structure of the archive-truly tables set might evolve with time i.e. new tables might be added or new columns can be added to some existing table. Hence whenever an archive DB is present, the database upgrade scripts consisting of DDL changes are applied to both the current DB as well as the archive DB to keep their structures up to date.

3.2 PRLM Archive and Purge Operations

The purge operation is implemented using *Purge* primitive with mode *purge_before*, whereas the archive operation is implemented as a logical combination of *two Purge* operations one with mode *purge_before* and other with mode *purge_after*. We first describe Purge operation and next cover Archive operation.

Algorithm 1: Purge(Before/After) primitive

Input : cut-off date, database schema name, mode
Output : reduced PSAS database and datapump file containing the reduced data of the archive-truly tables

```
 purge_primitive (cutoff_date, schema_name, mode)
 -- mode:  purge_before / purge_after
1. create_ST (schema_name)
   -- creates  ST (Staging tables) for archive-truly tables
2. copy2ST (cutoff_date, schema_name, mode)
   -- determine and copy the filtered records to ST
     if(mode = = 'purge_before') then
        Get root (visit) records after cutoff_date;
     else
        Get root (visit) records before cutoff_date;
     end if;
     current_level ← visit_record;
     for i in 1..num_levels loop
        --num_levels: levels in the visit record hierarchy
        Copy current level records to ST;
        Determine the next level descendant archive-truly
         tables based on the 'parent present in ST';
        increment current_level;
     end loop;
3. disable_C_TRIG (schema_name)
   -- disable constraints and triggers among all tables
4. truncate_T (schema_name)
   -- truncates all the archive-truly tables
5. copy_ST2T (schema_name)
   -- copy all the data from ST → archive-truly tables
6. enable_C_TRIG (schema name);
   -- enable all the constraints and triggers among all tables
7. drop_ST
   -- drop all the ST tables
8. create_datapump_dump (schema_name);
   -- create the datapump export file of the latest
     archive-truly tables
```

Purge Operation: For *Purge* primitive invoked with mode *purge_before*, in the step 2 of the algorithm, the copying of data into the ST is done by traversing the tree (Figure 6) in a top-down fashion. After the root (visit) records are determined, they are first copied into the ST tables. Then we go to the next level (descendants) to determine the records to be copied by using the parent records present in the corresponding ST tables. This determination of the records to be copied depends only on the immediate parent (from the tree) present in the ST i.e. it is implemented by using those parent records from the ST tables which were populated in the previous level.

After the completion of the purge-before primitive operation, database present in the server is the current reduced database (with the data before cutoff purged).

Note that although data may be simply distinguished table-wise independently, we preserve visit atomicity, that is, all the child records of a particular parent record should remain together or be archived together if the referential constraint is explicitly asserted. In the copy2ST procedure we start with the `VISIT` table where the records are selected based on the visit_end_date in case of IPD visits and pseudo_visit_end _date for the OPD visits. The need for pseudo_visit_end_date arises for OPD records as the visit_end_date is not definite for an OPD visit. By definition, OPD visit is expected to end same day, but in practice, patient may visit next day to complete a test or make payment and collect test results. Thus, the last activity is used to determine the pseudo_visit_end_date, which is important to prevent sending an active OPD visit to archive.

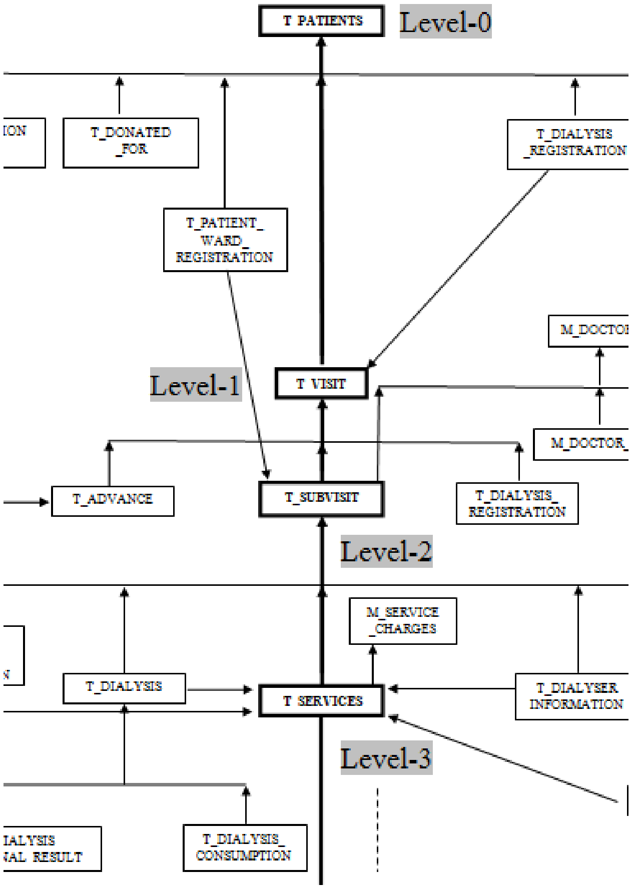


Figure 6: (Part of) Visit tree involving archive-truly tables

Only if the visit_end_date of a record is after the cut-off, that visit record is inserted into the corresponding ST. Once all the visit records are filtered, the remaining tables are dealt recursively based on the parent-child relations (as discussed in the previous paragraph) among the archive-truly tables, thereby maintaining the visit atomicity. Remaining tables (pseudo and regular Archive and retain completely) are left untouched and hence are retained completely by this scheme. Thus, we achieve the archive with individual table data movement of archive truly tables only. Furthermore, the table movement is performed by use of table level truncate and bulk-insert operation, which make it very efficient.

Archive Operation: The DB activity is stopped and export dump (copy of original DB) is taken. First on a copy of the original DB, the purge primitive is invoked with mode *purge_berfore* as described to generate the current reduced DB. Next, on another copy of the original DB, the purge primitive is called with the mode *purge_after* to generate the archive DB. All the steps followed for purging are also used for archiving. Instead of moving the data after the cut-off (from the archive-truly tables) to ST tables, for archive operation, we move the data before the cut-off point to the ST tables starting from root visit records, which is recursively followed as before. All the subsequent steps are identical to purge operation. Since the same idea of 'purge' operation is used for archiving, archive process is also known as 'purge-after' operation (that is, the current data is purged, leaving behind the data that needs to be archived). After the completion of purge-after operation (archive), the resulting database becomes the latest archive database which will eventually be merged with the previous archive data (if any) as described in Section 3.3.

3.3 PRLM Merge Archives Operation

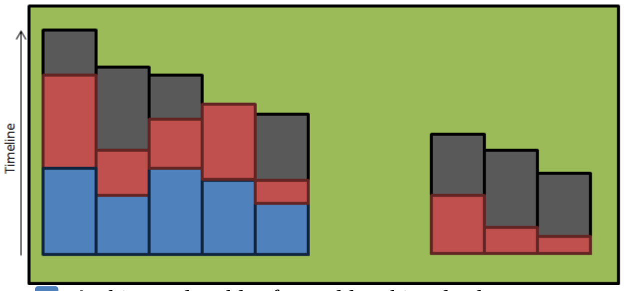
Each time the data is archived, the latest archive shall be merged with the previous archive which resides on a single server. Algorithm 2 shows the basic steps implemented in the archive merge process.

```
Algorithm 2: Merging of archives
Input: schema name of PSAS database, list of archive-truly
tables, all the required dump-files
Output: complete archive database (merged)

1. import (schema_name, original(latest) DB dump-file)
2. import_replace (schema_name, old_archive datapump,
                                   arch_truly_tables)
    -- inherits archive-truly tables data from the old archive
     for i in 1..arch_truly_tables.COUNT loop
        impdp(arch_truly_tables[i],TAB_ACTION=replace);
     end loop;
3. disable_C_TRIG (schema_name)
    -- disable the constraints and triggers among all tables
4. import_append (schema_name, new_archive datapump,
                               arch_truly_tables)
    for i in 1..arch_truly_tables.COUNT loop
       impdp (arch_truly_tables[i],TAB_ACTION=append);
    end loop;
    -- new set of archive data appended
5. Enable_C_TRIG (schema_name)
   -- enable constraints and triggers among all tables
```

Firstly, the original DB, i.e., the latest current database is imported into the server ensuring that all the non-archive-truly tables have the up-to-date data in the archive server. Then all the archive-truly tables are replaced with the data from the previous archive datapump. In step 3 of the algorithm, all the constraints and triggers among the archive-truly tables are disabled temporarily and this prevents 'foreign key constraint violated' error during the import_append subroutine. Once the latest archive data is appended the constraints and triggers are enabled again. After the completion of archive-merge operation, database present in the server is the complete archive database.

Figure 7 shows how the merging process works for different tables in the database. All the tables other than the archive-truly tables are simply copied from the current running database and hence will be having the latest data of those tables. The tables represented in black border are obtained when the current-running database is imported into the server. These tables are first replaced by the those from the old-archive database (shown in blue). Note that the table data represented in grey denote the removed data. After which the appending of the latest archive data (shown in red) is done to all the corresponding archive-truly tables.



🟦 Archive-truly tables from old archive database
🟥 Archive-truly tables from newly cut archive database
⬛ Archive-truly tables from current-running database
🟩 Remaining tables other than archive-truly

Figure 7: The effect of Merge Archive Operation

The case pertaining to the evolved database is potrayed in the right three archive-truly tables of Figure 7. For such case, we won't have any data from the old-archive database as at that time the new tables were not yet introduced. Thus, it contains only those records which were archived from the current-running database.

The Figure 8 shows the different types of referential constraints among the tables.

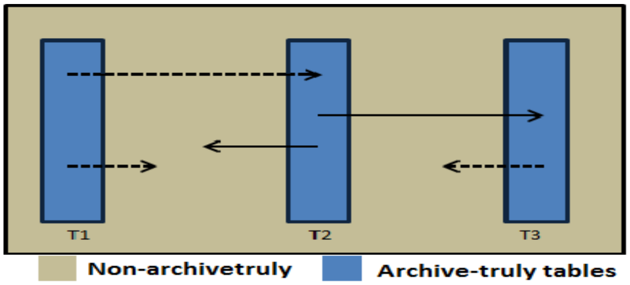


Figure 8: The relationships between various types of tables

The arrow with a solid line denotes an explicitly asserted referential constraint from child to parent table and this requires the child to be co-located with its parent. When the reference is between two archive-truly tables (for example, T2 references T3), if the parent record is moved from the database, the child record(s) also move with it. However, when a archive-truly table (T2) references a non-archive truly table, the constraint is implicitly met as the parent records of non-archive truly table are present in both archive and current DB as they are retained and archived completely.

The arrow with dotted line denotes a referential constraint that is not asserted for tables. When the reference to another archive-truly table is not asserted (for example, T1 to T2), the parent table can be placed separately from child records. This is used when the information contained in child has intrinsic worth (independent of parent). For example, `T_BLOOD_BAG` table is child of `T_SUBVISIT`. Although a blood bag record comes into existence as part of subvisit but it should be kept in active database till it is issued. This case is handled by not explicitly asserting the constraint and hence it is archived independently based on its activity. Lastly, there could be referential constraints among non-archive truly tables which are automatically satisifed as those tables are co-located in both current and archive DB.

3.4 PRLM Design Challenges

PRLM is developed to function properly in event of data-entry errors, which could occur due to high volume (over 800 patients per day registered during 5 working hours) of OPD patients. For example, sometimes a new OPD visit record is not created for a patient even though he revisits the hospital after a substantial gap. Such cases are dealt in the following way. By default, OPD records do not have an actual end date for the visit i.e. a patient might revisit the hospital multiple times for the same problem under a single visit and may complete the payment for the diagnosis later, hence that particular visit record is assigned a pseudo_visit_end_date based on the last activity. The pseudo_visit_end_date is used to decide the placement of that visit record.

3.5 Handling the Database Schema Evolution

The successive PRLM operations typically occur after a gap of six to nine months and during this time the schema can evolve. Table I summarizes the types of evolution and the need for PRLM tool modification. The archive DB is kept up to date as and when the schema evolves by running the corresponding database upgrade scripts so PRLM tool operates on the same database structure in both the archive and current DB.

For the case, when a column in an existing table is added or modified it works without any changes. Normally the usage of `'SELECT *'` usage is avoided as it might lead to selection of unwanted data (when new columns are added). But in the case of PRLM scripts, it is used specifically so as to handle the evolving nature of a database table.

Regarding adding new archive-truly tables, they are typically added for recording clinical information and they come under the level 3, i.e., under the `T_SERVICES` as a child table (in Figure 6). Since the visit descendent tree is modified, the traversal during the movement of data detects new tables in the leaf position, and the records of newly added tables also get archived. Similarly, if multiple new archive-truly tables comprising of a parent-child clinical relationships are added, they become an additional branch in existing visit descendent tree, and thus records of those tables will also get archived. Note that addition of other (non-archive-truly tables) does not require changes to the PRLM tool.

Table I Impact of DB Schema Evolution

Type of Schema Evolution	**Occurs**	**PRLM Tool modification needed?**
Addition/Modification of Column	Yes	No
Deletion of Column	No	N/A
Adding Archive-Truly Table	Yes	Yes
Adding non-Archive-Truly Table	Yes	No
Deletion of Tables	No	N/A
Change in Table Classification	Yes	Yes

The categorization of the tables can also change, for example a pseudo archive and retain completely table may be classified as archive-truly in the next application of the PRLM. In such cases the table is added to archive-truly list, its content is truncated, and then the merging of archives process is applied.

3.6 Correctness of the PRLM Operations

The data is partitioned in a way that the visit atomicity of the records is preserved; this is ensured by the fact that all the records pertaining to a particular visit are connected by referential constraints. Thus, as long as the tables are properly classified into the categories mentioned in section 3.1 and the parent-child relationships among the tables are properly asserted, the mechanism should work fine. Scope of an error is only when the data is wrongly entered or incomplete, and some of those cases are also tackled (as discussed in section 3.4). In addition, as a precaution, after each PRLM operation a validation check is done module-wise which normally takes up to an hour and then only the database is made live.

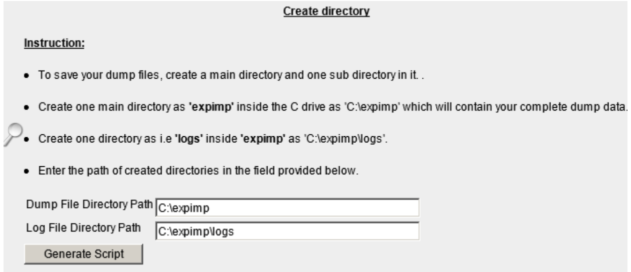
3.7 Discussion

We reduce only the top growing tables as limiting the database size is the primary goal. This ensures faster processing of the movement of records. Once the data is generated, the use of Oracle's Data Pump utility to apply the change on production DB to create current reduced DB ensures that the downtime is kept to minimal.

The tool works for hospital databases of two different sites, which differ in data and growth characteristics. In general, the scheme can be applied to other domains also as it doesn't rely on the specific domain knowledge of the tables but only their connectivity in term of referential constraints and the classification of the tables into archive-truly, archive and retain completely, and pseudo-archive and retain completely.

4. A Tour of PRLM Tool and Usage

This section describes the features and functioning of PRLM **P**atient **R**ecord **L**ifecycle **M**anagement tool.



Create directory

Instruction:

- To save your dump files, create a main directory and one sub directory in it. .
- Create one main directory as **'expimp'** inside the C drive as 'C:\expimp' which will contain your complete dump data.
- Create one directory as i.e **'logs'** inside **'expimp'** as 'C:\expimp\logs'.
- Enter the path of created directories in the field provided below.

Dump File Directory Path	C:\expimp
Log File Directory Path	C:\expimp\logs

Figure 9: PRLM Setup

4.1 PRLM Setup

PRLM Setup is used to specify directory where archived data is to be saved and grant all the required privileges. This is done by executing the script generated in the '**PRLM Setup'** page as shown in Figure 9.

4.2 DB Characteristics and Size Reduction Estimates

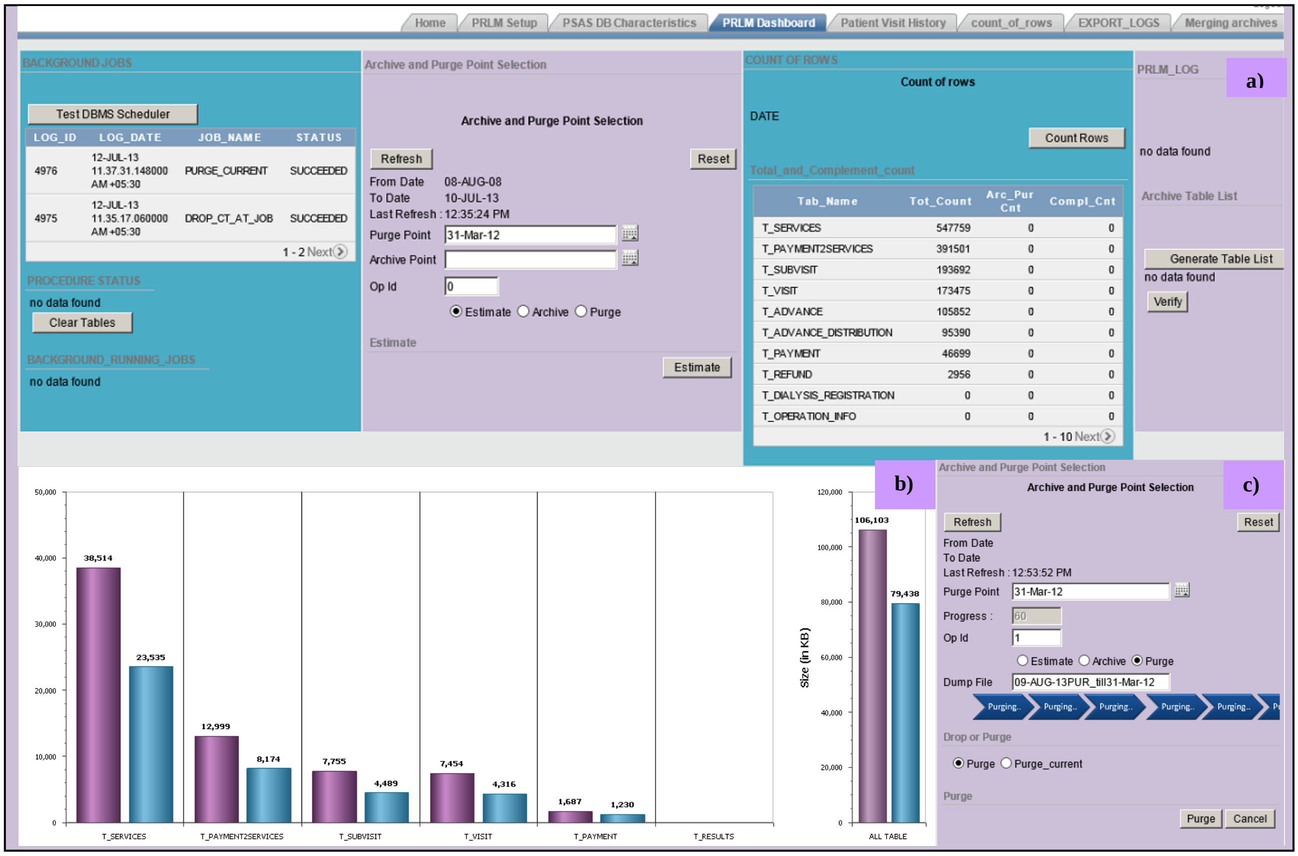
The database characteristics such as total size of database, size of all large tables ordered by size, latest and the earliest service activity date, visit_start_date, and visit_end_date, subvisit_start_date, and subvisit_end_date are shown.

In order to know the percentage of size reduction before actual archive or purge or archive and purge, we have developed the method of estimation of size reduction of database. Here in this page, application will show the estimated size of rapidly growing tables before and after the process.

We are estimating on the basis of two criteria 1) If only archive or only purge point will be selected the estimation will be according to archive or purge point. 2) If both are selected simultaneously then estimation will be done according to archive point because the data left out in the database after the process will be essentially data after the archive point.

4.3 PRLM Dashboard

This page contains the main processes i.e. 'Archive', 'Purge' (Figure 10-a). User can estimate the size reduction or actually carry out the operation after selecting archive or purge points, for example Figure 10-b shows estimation analysis after selecting a purge point and Figure 10-c shows purge operation which is in progress after selecting a purge point (Figure 10-c).



LOG_ID	LOG_DATE	JOB_NAME	STATUS
4976	12-JUL-13 11.37.31.148000 AM +05:30	PURGE_CURRENT	SUCCEEDED
4975	12-JUL-13 11.35.17.060000 AM +05:30	DROP_CT_AT_JOB	SUCCEEDED

1 - 2 Next

PROCEDURE STATUS

no data found

BACKGROUND_RUNNING_JOBS

no data found

Archive and Purge Point Selection

Archive and Purge Point Selection

From Date 08-AUG-08
To Date 10-JUL-13
Last Refresh : 12:35:24 PM
Purge Point 31-Mar-12
Archive Point
Op Id 0
● Estimate ○ Archive ○ Purge

Estimate

COUNT OF ROWS

Count of rows

DATE

Total_and_Complement_count

Tab_Name	Tot_Count	Arc_Pur Cnt	Compl_Cnt
T_SERVICES	547759	0	0
T_PAYMENT2SERVICES	391501	0	0
T_SUBVISIT	193692	0	0
T_VISIT	173475	0	0
T_ADVANCE	105852	0	0
T_ADVANCE_DISTRIBUTION	95390	0	0
T_PAYMENT	46699	0	0
T_REFUND	2956	0	0
T_DIALYSIS_REGISTRATION	0	0	0
T_OPERATION_INFO	0	0	0

1 - 10 Next

PRLM_LOG

no data found

Archive Table List

no data found

Archive and Purge Point Selection

Archive and Purge Point Selection

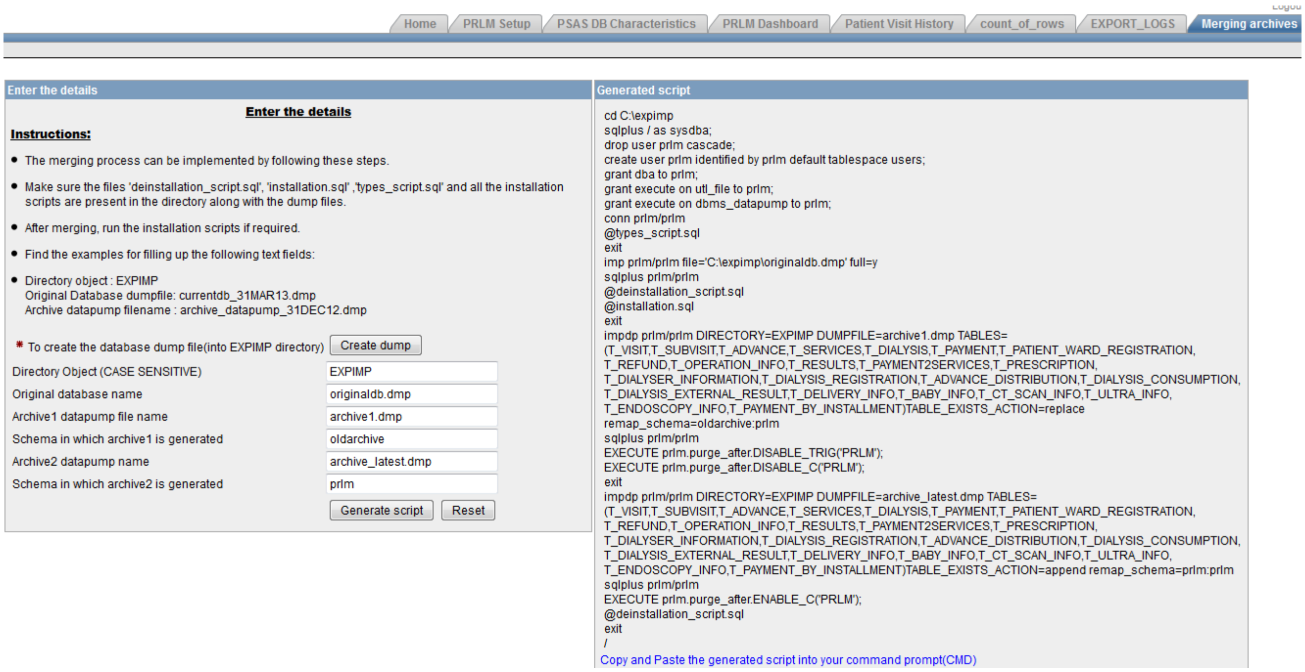
From Date
To Date
Last Refresh : 12:53:52 PM
Purge Point 31-Mar-12
Progress : 60
Op Id 1
○ Estimate ○ Archive ● Purge
Dump File 09-AUG-13PUR_till31-Mar-12

Drop or Purge

● Purge ○ Purge_current

Purge

Figure 10: a) PRLM Dashboard, b) Purge Estimated Size Reduction, and c) Purge Operation in progress



Enter the details

Enter the details

Instructions:

- The merging process can be implemented by following these steps.
- Make sure the files 'deinstallation_script.sql', 'installation.sql' ,'types_script.sql' and all the installation scripts are present in the directory along with the dump files.
- After merging, run the installation scripts if required.
- Find the examples for filling up the following text fields:
- Directory object : EXPIMP
  Original Database dumpfile: currentdb_31MAR13.dmp
  Archive datapump filename : archive_datapump_31DEC12.dmp

* To create the database dump file(into EXPIMP directory) [Create dump]

Directory Object (CASE SENSITIVE)	EXPIMP
Original database name	originaldb.dmp
Archive1 datapump file name	archive1.dmp
Schema in which archive1 is generated	oldarchive
Archive2 datapump name	archive_latest.dmp
Schema in which archive2 is generated	prlm

[Generate script] [Reset]

Generated script

```
cd C:\expimp
sqlplus / as sysdba;
drop user prlm cascade;
create user prlm identified by prlm default tablespace users;
grant dba to prlm;
grant execute on utl_file to prlm;
grant execute on dbms_datapump to prlm;
conn prlm/prlm
@types_script.sql
exit
imp prlm/prlm file='C:\expimp\originaldb.dmp' full=y
sqlplus prlm/prlm
@deinstallation_script.sql
@installation.sql
exit
impdp prlm/prlm DIRECTORY=EXPIMP DUMPFILE=archive1.dmp TABLES=
(T_VISIT,T_SUBVISIT,T_ADVANCE,T_SERVICES,T_DIALYSIS,T_PAYMENT,T_PATIENT_WARD_REGISTRATION,
T_REFUND,T_OPERATION_INFO,T_RESULTS,T_PAYMENT2SERVICES,T_PRESCRIPTION,
T_DIALYSER_INFORMATION,T_DIALYSIS_REGISTRATION,T_ADVANCE_DISTRIBUTION,T_DIALYSIS_CONSUMPTION,
T_DIALYSIS_EXTERNAL_RESULT,T_DELIVERY_INFO,T_BABY_INFO,T_CT_SCAN_INFO,T_ULTRA_INFO,
T_ENDOSCOPY_INFO,T_PAYMENT_BY_INSTALLMENT)TABLE_EXISTS_ACTION=replace
remap_schema=oldarchive:prlm
sqlplus prlm/prlm
EXECUTE prlm.purge_after.DISABLE_TRIG('PRLM');
EXECUTE prlm.purge_after.DISABLE_C('PRLM');
exit
impdp prlm/prlm DIRECTORY=EXPIMP DUMPFILE=archive_latest.dmp TABLES=
(T_VISIT,T_SUBVISIT,T_ADVANCE,T_SERVICES,T_DIALYSIS,T_PAYMENT,T_PATIENT_WARD_REGISTRATION,
T_REFUND,T_OPERATION_INFO,T_RESULTS,T_PAYMENT2SERVICES,T_PRESCRIPTION,
T_DIALYSER_INFORMATION,T_DIALYSIS_REGISTRATION,T_ADVANCE_DISTRIBUTION,T_DIALYSIS_CONSUMPTION,
T_DIALYSIS_EXTERNAL_RESULT,T_DELIVERY_INFO,T_BABY_INFO,T_CT_SCAN_INFO,T_ULTRA_INFO,
T_ENDOSCOPY_INFO,T_PAYMENT_BY_INSTALLMENT)TABLE_EXISTS_ACTION=append remap_schema=prlm:prlm
sqlplus prlm/prlm
EXECUTE prlm.purge_after.ENABLE_C('PRLM');
@deinstallation_script.sql
exit
/
```

Copy and Paste the generated script into your command prompt(CMD)

Figure 11: PRLM Merge Archives Operation

After the archive operation is done the actual reduction can be seen as a separate report. Similarly Figure 11 shows the scripts generated for PRLM Merge Archive operation, which takes as input previously generated two successive archive database dumps.

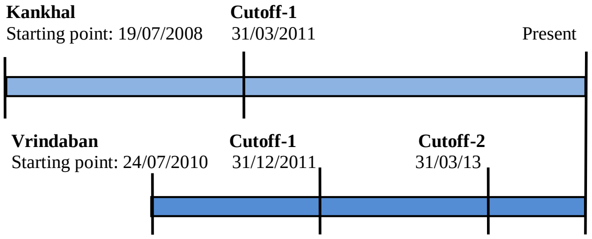


Figure 12: PRLM Usage on Production Databases

4.4 PRLM Usage at Vrindaban & Kankhal Hospitals

Electronic medical records have been maintained at Kankhal and Vrindaban hospitals since July 19, 2008 and July 24, 2010 respectively using our PSAS application. At these sites, the PRLM has been applied twice for Vrindaban and once for Kankhal hospital to create the reduced and the archive DBs. Figure 12 shows the cut-off dates chosen whereas Figure 13 shows the original, reduced, and archive DB sizes. The reduced size was 53%, 54%, and 66% of the original DB for the three cases respectively. The archive DB sizes for Vrindaban-1 and Kankhal-1 are the first archive size, whereas for Vrindaban-2 the archive DB was obtained by merging it with the previous archive (performed on 05/07/2013). Note that each hospital chose cut-off dates independently. Since the three PRLM operations were performed at three different times (Vrindaban on 22/08/2012, and 12/06/2013, and Kankhal on 26/01/2013) the PRLM tool had to operate on an evolved database each time.

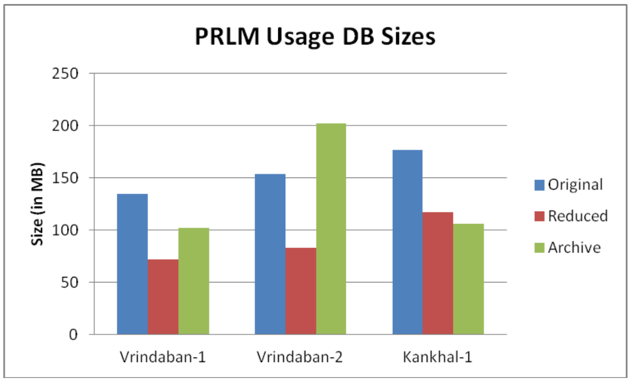


Figure 13: PRLM Usage DB Sizes

The data and growth characteristics were quite different at the two database sites (see details in Section 5.6). The PRLM operations at site completed within 15 minutes. However, for each PRLM usage, an additional hour was taken to do verification on the reduced final databases prior to making it available for use.

Apart from using the PRLM tool for production databases, we have also used it internally for creating reduced versions of the databases (typically containing one month of data) for testing and development purposes. A side benefit of using the PRLM tool is that it forces us to validate the consistency of visit data almost every six months.

5. PRLM Experiments

This section deals with the experimental study that evaluates the PRLM tool. All the experiments were performed on a system with Intel Core i5-2410M CPU @ 2.30 GHz, 4GB RAM, 750 GB disk space, Windows 7 OS, using Oracle Database 10gR2 Express Edition [8], and Oracle Application Express (APEX) [9] generated database applications. The key DB parameters are: `sga_max_size=768 MB, database buffer size=504 MB` and `pga_aggregate_target=256 MB`.

For experiments on Vrindaban database, we have used cutoff-1 as 31/12/2012 and cutoff-2 as 31/03/2013. Similarly for the Kankhal database, 31/12/2012 has been used as the cut-off date. For experiments in Section 5.1, 5.3, 5.4, and 5.5, only Vrindaban database is used. However, for experiments in Section 5.2 and 5.6 that involving comparison between the two hospital sites, both Vrindaban and Kankhal database are used.

5.1 PRLM Operation Time Observations

The PRLM operation was repeated 5 times and the time taken for each operation has been recorded. The Table II shows the time taken for the operations (purge, archive and merge) on the 2 databases original and reduced-1, five readings for each operation have been taken. The mean value and the standard deviation (σ) for the time taken by each operation have been calculated.

Table II PRLM Operation Time

	Time (in sec)					
Cutoff-31/12/12; ORIGINAL DB (size- 86.7 MB)						
Purge	**1**	**2**	**3**	**4**	**5**	**AVG**
	50.8	53.1	52.8	49.9	52.1	**51.74** σ=1.36
Archive1	71.4	67.8	70.4	76.4	74.2	**72.04** σ=3.34
Cutoff-31/03/13 REDUCED1DB (size- 42.3 MB)						
Purge2	28.1	27.4	28.7	28.3	28.7	**28.24** σ =0.54
Archive2	25.4	25.1	24.9	25.9	25.8	**25.42** σ =0.43
Merging Archive1& Archive2	43.8	40.1	36.9	40.5	37.2	**39.7** σ =2.82

ON ORIGINAL DATABASE : 24/07/2010 – 12/06/2013 (size-86.7 MB)
ON CURRENT REDUCED-1: 31/12/2012 – 12/06/2013 (size-42.3 MB)

Note that merging of archives is different from archive and purge operations, so merge time cannot be compared with these operations.

5.2 Analysing PRLM time taken –purge and archive operations

Figure 14 shows the bar charts for time taken for the two operations. It is clear that as the size of the database on which PRLM is performed decreases, the PRLM time for purging or archiving decreases. The majority of the time in either of the operations is consumed by copying of data from archive-truly tables (`'INSERT AS SELECT'` operations). Thus, if the size of resulting archive truly tables is high, the amount of data to be inserted increases and hence the time consumed by the insert operation rises. Similarly, the select operation takes some time to acquire the filtered data which depends on the initial size of archive truly tables. Other operations like `'TRUNCATE tables'` or `'Enable/Disable Triggers and Constraints'` do not contribute much to the running time of the tool for purge/archive.

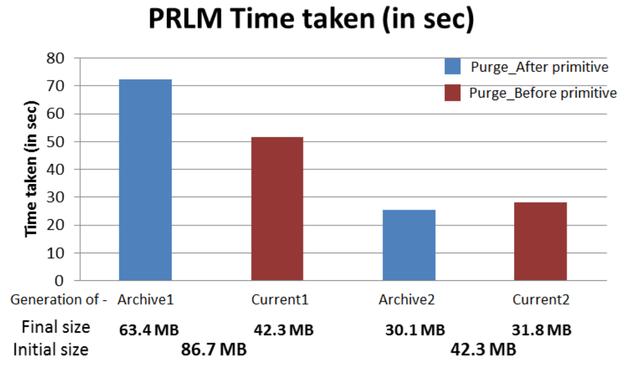


Figure 14: PRLM Purge/Archive Time

Note that both purge and archive operations show similar characteristics as is seen in Figure 15. In this Figure, we plot the time taken (in sec) with respect to result DB size (in MB) and as expected the slope is similar for purge as well as archive operations. Also, the times for databases are from two different sites (Vrindaban and Kankhal), the line graph between final result DB size and PRLM time taken are almost parallel to each other. In spite of different growth characteristics among the archive-truly tables of the two databases (see Section 5.6), this similarity is due to INSERT (as append operation) time, which primarily depends on size of the data being inserted into staging table and back.

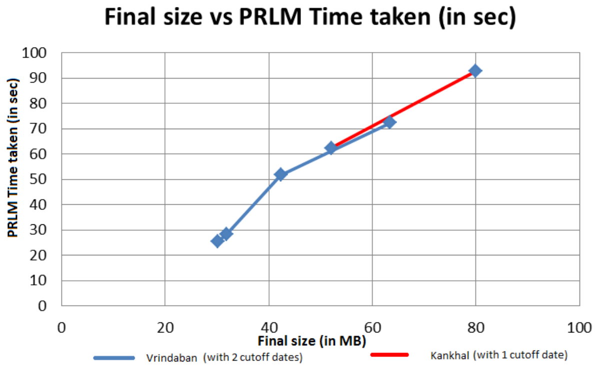
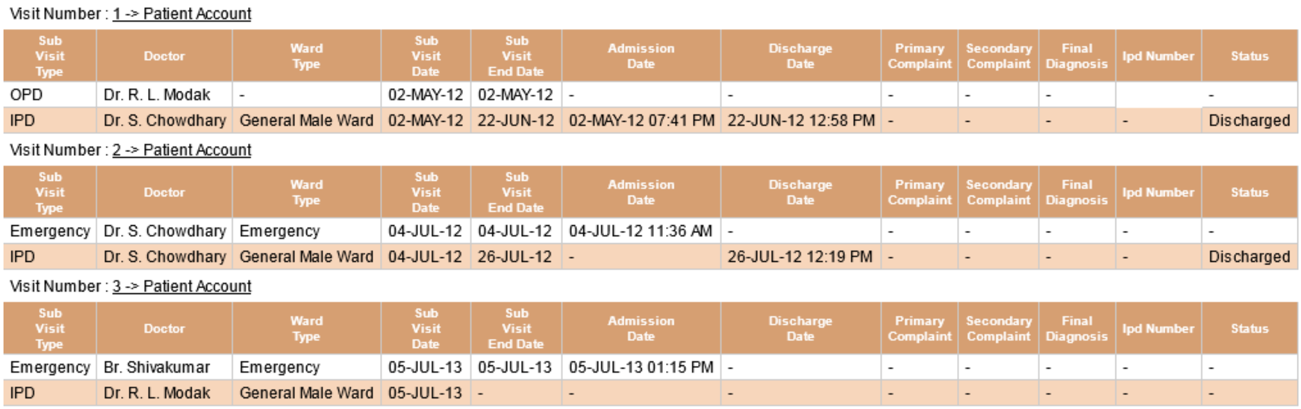


Figure 15: PRLM Purge/Archive Time vs. Result DB Size



Visit Number : 1 -> Patient Account

Sub Visit Type	Doctor	Ward Type	Sub Visit Date	Sub Visit End Date	Admission Date	Discharge Date	Primary Complaint	Secondary Complaint	Final Diagnosis	Ipd Number	Status
OPD	Dr. R. L. Modak	-	02-MAY-12	02-MAY-12	-	-	-	-	-		-
IPD	Dr. S. Chowdhary	General Male Ward	02-MAY-12	22-JUN-12	02-MAY-12 07:41 PM	22-JUN-12 12:58 PM	-	-	-	-	Discharged

Visit Number : 2 -> Patient Account

Sub Visit Type	Doctor	Ward Type	Sub Visit Date	Sub Visit End Date	Admission Date	Discharge Date	Primary Complaint	Secondary Complaint	Final Diagnosis	Ipd Number	Status
Emergency	Dr. S. Chowdhary	Emergency	04-JUL-12	04-JUL-12	04-JUL-12 11:36 AM	-	-	-	-	-	-
IPD	Dr. S. Chowdhary	General Male Ward	04-JUL-12	26-JUL-12	-	26-JUL-12 12:19 PM	-	-	-	-	Discharged

Visit Number : 3 -> Patient Account

Sub Visit Type	Doctor	Ward Type	Sub Visit Date	Sub Visit End Date	Admission Date	Discharge Date	Primary Complaint	Secondary Complaint	Final Diagnosis	Ipd Number	Status
Emergency	Br. Shivakumar	Emergency	05-JUL-13	05-JUL-13	05-JUL-13 01:15 PM	-	-	-	-	-	-
IPD	Dr. R. L. Modak	General Male Ward	05-JUL-13	-	-	-	-	-	-	-	-

Figure 16: A patient visit history consisting of three visits

5.3 Comparing different stages of the PSAS database while applying PRLM

The archive-truly tables are generally the fast growing tables, and using the PRLM tool the data in these tables shall be classified into current (active) or archive based on the age of the visit and parent/child relations among the records in these tables as discussed in Section 3.1.

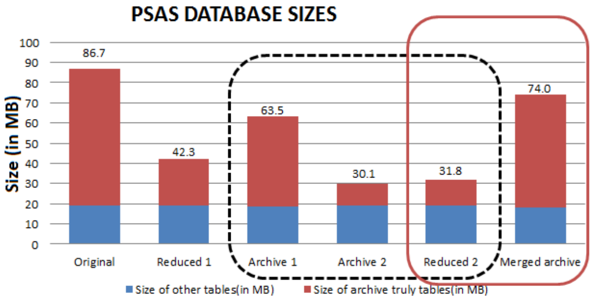


Figure 17: PRLM Purge/Archive Databases

Figure 17 shows all the 6 database sizes for Vrindaban as stacked bar charts with archive-truly tables shaded in red and the remaining tables shaded in blue. Initially only the original database was hosted and then when the performance degraded, the database was transformed into reduced1 and archive1.

Next, the tool was applied again with the cut-off date i.e. 31/03/13, which resulted in archive2 and reduced 2 DBs. Thus, three servers were maintained to host archive1, archive2 and reduced 2 DBs, which forms the previous configuration (shown in dotted box above). Once the merge archive method was developed, the two archives were merged and the configuration reduced2 and merged archive, was released (shown in the red box).

The size on non-archive truly tables is dominated by the 'T_PATIENTS', which constitutes about half of the non-archive truly tables (approx. 10 MB). This we have to maintain in the archive and retain completely category as old patients can potentially visit any time (even after a long gap).

5.4 Analysing the VISIT history processing time of the PSAS database

As the size of the database increases, the performance of complex reports that access information about the patient visit degrades. To study this we considered PSAS visit history report (Figure 16), which shows the complete visit history, including visit dates, wards the patient stayed, and discharge dates. Note that the individual visit details are available in the Patient Account hyperlink. In Figure 16, the last visit is still active as the patient is admitted but not yet discharged. The query used in fetching the visit history records in the PSAS was used to measure the time to process the data of a patient in different databases.

A patient with id '10-010296' was selected such that the service records of this patient are present in the either side of both the cut-off dates. Table III shows the number of service registration records retrieved in each database for this particular patient.

Table III Visit History Processing Time

Database	Service Count	Visit History Processing Time
Original	28	0.161 s
Reduced 1	16	0.053 s
Archive1	12	0.109 s
Archive2	9	0.025 s
Reduced 2	7	0.029 s
Merged Archive	21	0.135 s

The time taken vs. DB size was plotted in a line graph shown in Figure 18. Note that the average times used were based on the warm run. Here the red dots indicate the original and the reduced databases and the black ones indicates both the archives and also the merged archive. In this experiment, the number of service records correlated with the size of the database. We got an almost linear graph indicating that the visit history processing time is directly proportional to the size of the database.

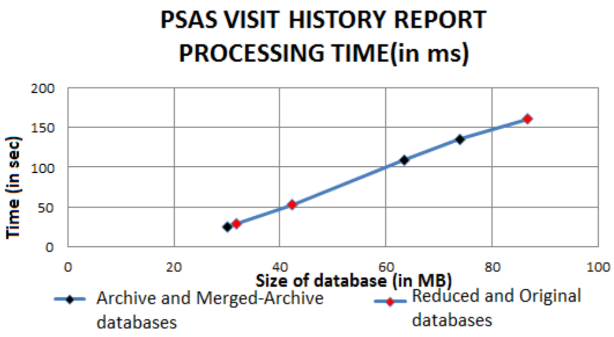


Figure 18: Visit History Time vs. DB Size

As expected, the performance degrades as database size increases, which can be restored by performing a PRLM operation. The same behavior was exhibhited by other complex reports used in PSAS such as daily collection report, individual patient account reports, etc.

5.5 Degree of replication

The non-archive truly tables are fully retained in current and reduced databases during a PRLM operation. Hence the combined size of the archive and the reduced or current databases will be greater than the original database (Figure 19).

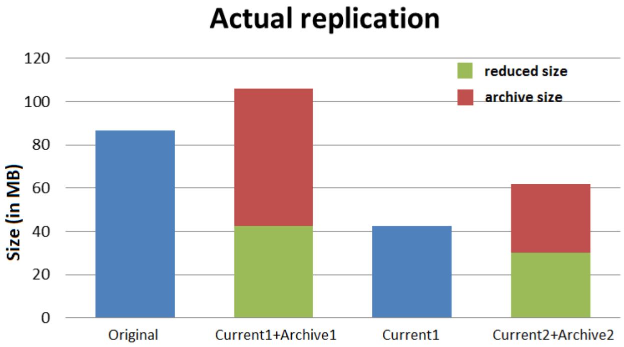


Figure 19: The actual replication of data

The actual replication is about 20 MB (i.e. approx. 25% degree of replication with respect to the original database). But essentially only 10 MB data is replicated (i.e. approx. 12% degree of replication of the original database), if we exclude `T_PATIENTS` and `T_GUARDIAN_DETAILS` tables which are needed in both archive and current databases (Figure 20). Thus, our strategy of performing PRLM on fast growing archive-truly tables keeps the scheme simple and quick and at the same time the degree of replication is acceptable.

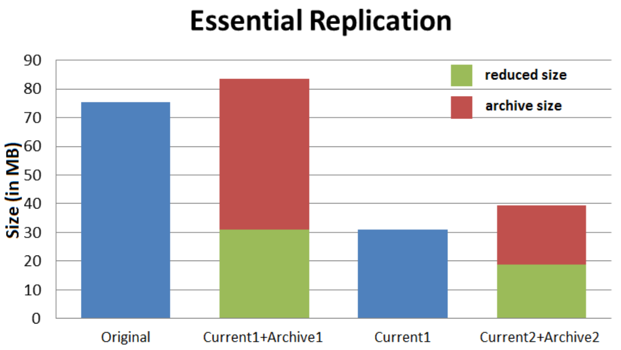
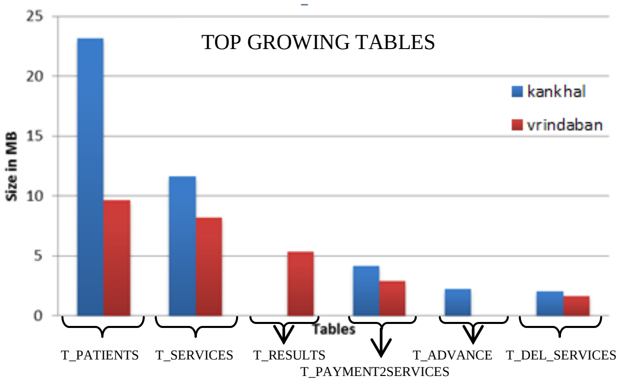


Figure 20: The essential replication of data

5.6 Table growth characteristics at different sites

In this experiment, we compare the top 5 growing tables of the reduced databases from Kankhal and Vrindaban.



KANKHAL CURRENT-1 : 31/12/2012 – 10/07/2013 (size-52.0 MB)
VRINDABAN CURRENT-1: 31/12/2012 – 12/06/2013 (size-42.3 MB)

Figure 21: Table Growth Characteristics

Figure 21 shows that the growth characteristics of the same tables at different databases can differ due to hospital site specific usage. For example, `T_RESULTS` table, which is the 3rd largest tables in the Vrindaban database is not even present among the top 5 tables of the Kankhal database. Similarly, the `T_PAYMENT2SERVICES` table which is 4th largest in Vrindaban database, is the 3rd largest is the Kankhal database.

The PRLM tool is versatile in that it works with multiple hospital sites data, each have different data and growth characteristics. Furthermore, the basic scheme of PRLM, although designed for healthcare domain, can be adapted to work on other application domains as well. The PRLM scheme primarily relies on understanding the implicit and explicit referential constraints among tables, and categorizing tables into archive-truly and rest as discussed in Section 3.1.

6. Related Work

Information Lifecycle Management (ILM) has been an active area of research and development. The basic idea of ILM have been formalized and presented using definitions of *information*, its *value*, *information class*, and *storage class* [12]. Our scheme uses the same ideas, but in addition, introduces the notion of a child record's *information value* being inherited from its parent record. This extension is needed as a single unit of information is typically stored in multiple tables in a normalized schema. A framework clarifying the relationship between *value-at-risk* and *total cost of ownership* is presented to enhance ILM [15].

Both storage oriented ILM solutions [3, 5, 11] as well as software [6,7] have been developed. Storage-solutions have evolved from traditional hierarchical (multi-tier) storage solutions to a more comprehensive policy based information management [11]. The software solutions offered by database vendors [6,7] are typically domain agnostic in that they provide generic solution applicable to any data requiring data retention, archiving, and purging. Specifically, they provide starter packages to simplify the task of developing the complete ILM solution.

Our work falls in the category of ILM software targeted for healthcare domain. For healthcare domain, companies offer generic solutions involving tiered network storage both for medical records and medical images (such as from EMC^2 [14]). We have built a customizable schema-aware solution for patient record lifecycle management that primarily relies on parent-child relationships and classification of tables. Also, rather than integrating with a tiered storage, we take the approach of *sharding* [10], the data across multiple servers. This is somewhat similar to the approach of network-aware placement of online social community data on shards, which is shown to improve system performance [13].

7. Conclusions and Future Work

The paper presented a Patient Record Lifecycle Management (PRLM) tool developed to supplement an already deployed Patient Services Accounting System (PSAS) in hospitals with support for purging, archiving, and data retention. It provides a schema-aware approach to ILM, by taking into account the referential constraints present between tables. The tool works even as the database schema evolves between successive PRLM operations.

The tool has been successfully applied at Ramakrishna Mission Sevashrama Hospitals, at Kankhal, and Vrindaban to manage their patient records. The experimental study conducted with the real data obtained from them further illustrates the benefits and tradeoffs involved.

In future, we plan to extend the tool to handle archiving of medical images as well, by using a hybrid approach involving sharding and tiered storage.

Acknowledgments: We thank Swamijis of Ramakrishna Mission Sevashrama Hospital, Kankhal and Vrindaban for providing the data for experimentation. We thank the members of Sarada Research Labs, Dehradun, and Bangalore for their help in verification while applying PRLM on production databases. We thank Saravana J (past SRL member), and past summer interns from IIT Guwahati (Swathi Karri, Snehlata, Kiranmayi Kasarapu, and Komal Agarwal) for their contribution to the development of the PRLM tool.

References

[1] Sarbanes-Oxley Act 2002, http://www.soxlaw.com/index.htm

[2] HIPAA, www.hhs.gov › OCR Home › Health Information Privacy.

[3] ILM Library: Information Lifecycle Management Best Practices Guide, http://www.redbooks.ibm.com/abstracts/sg247251.html

[4] Patient Services Accounting System (PSAS), Sarada Research Labs, http://www.saradaresearchlabs.org

[5] Symantec Global Services Datasheet, http://eval.symantec.com/mktginfo/enterprise/fact_sheets/ent-datasheet_symantec_storage_services_02-2008.en-us.pdf

[6] Implementing ILM using Oracle Database 11g, http://www.oracle.com/technetwork/database/focus-areas/storage/ilm-on-oracle11g-1-129053.pdf

[7] Information Lifecycle Management (SAP), http://www.sdn.sap.com/irj/sdn/ilm

[8] Oracle Database Express Edition 11g Release 2, www.oracle.com/technetwork/database/express-edition

[9] Oracle Application Express, www.oracle.com/technetwork/developer-tools/apex

[10] R. Cattell: Scalable SQL and NoSQL data stores. SIGMOD Record 39(4): 12-27 (2010).

[11] M. Beigi, M. V. Devarakonda, R. Jain, M. Kaplan, D. Pease, J. Rubas, U. Sharma, A. Verma: Policy-Based Information Lifecycle Management in a Large-Scale File System. POLICY 2005: 139-148.

[12] L. A.Turczyk, O. Heckmann, R. Berbner, R.Steinmetz, R. (2006). A formal approach to Information Lifecycle Management. Proceedings of 17th Annual IRMA Int'l Conf., Washington DC.

[13] Q. Duong, S. Goel, J. M. Hofman, S. Vassilvitskii: Sharding social networks. WSDM 2013: 223-232.

[14] R. A. Katz, Applying Information Lifecycle Management Strategies Enables Healthcare Providers to Accelerate Clinical Workflow, http://www.emc.com/collateral/hardware/white-papers/h1435-ilm-health.pdf

[15] P. P. Tallon, R. Scannell: Information life cycle management. Commun. ACM 50(11): 65-69 (2007)

[16] Ramakrishna Mission Sevashrama (A Charitable Hospital founded in 1901), Kankhal, Haridwar, http://www.rkmkankhal.org

[17] Ramakrishna Mission Sevashrama (A Charitable Hospital founded in 1907), Vrindaban, http://www.rkmsvrind.org/hospital.html