TABLE OF CONTENTS

AT A GLANCE

Organizations invest millions in their enterprise applications and supporting infrastructure. The stakes are high. Periodically reviewing and quantifying the value of the organization's application portfolio is an essential part of simplifying the IT infrastructure and controlling costs.

Every CIO and IT organization wants enterprise applications to deliver maximum business value. But many IT leaders today are faced with the problem that 80% of the budget is spent on "keeping the lights on" maintaining existing services and infrastructure. This leaves few resources available for innovation or addressing the never-ending queue of new business and user requests. Costs, performance and compliance (disaster recovery, client service agreements) continue to be a struggle for today's organizations.

Here you will find a DBArchive solution which addresses those struggles and helps you release significant resources that can be *redirected to better innovations*.

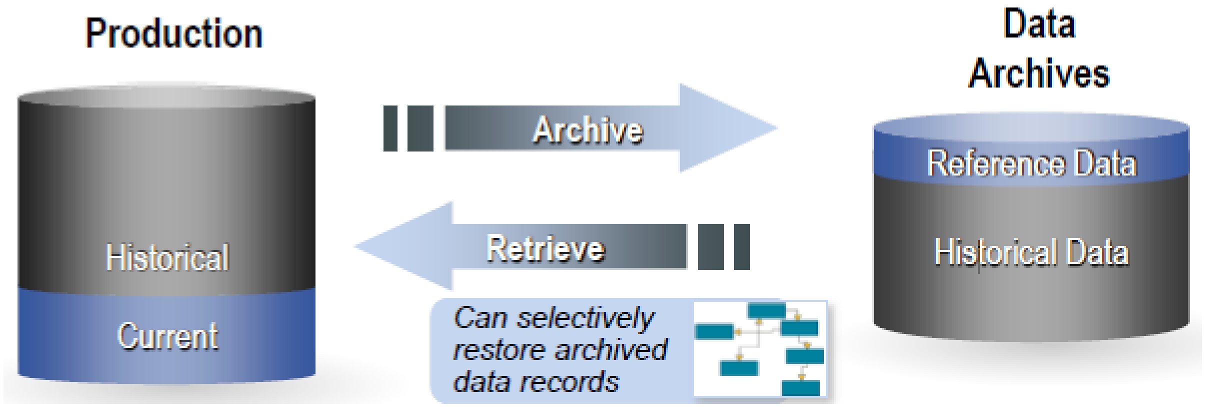
OVERVIEW

When it comes to data, more is not always better. Overgrown databases can impair the performance of your mission-critical services and custom applications. Periodic reviews of your application portfolio can help identify underperforming Systems where maintenance costs may outweigh business value. Therefore, natural question comes into play – how to move or archive complex historical data to separate environment thus improving production performance.

What is data archiving?

Data Archiving is an intelligent process for moving inactive or infrequently accessed data that still has value to new separate environment, while providing the ability to search and retrieve the archived data. This abstraction could be illustrated as in following Figure 1.

Figure 1 Basic Archiving Process



Archiving business-critical applications will help your company optimize applications portfolios to support your efforts of reducing hardware, software, storage, and maintenance costs, addressing risks and freeing valuable resources.

What is DBArchive solution in general? This is a database optimization solution built for MS Windows environment and currently supports all above (inclusive) v9.0 MS SQL versions.

DBArchive solution provides comprehensive database archiving capabilities that IT organizations can use to archive data and safely remove it from the production application by providing minimal inputs such as main archiving objects and filters. All the rest will be processed automatically.

DBArchive is automated, dynamic & intelligent solution with included capabilities:

- Discovers schema objects and relations;
- Merges schema from production to archive;
- Calculates valid archiving data sets of production data and iteratively move to archive;

  Note: Thereby streamlines the production database and reduces processing overhead;
- Process recursive or complex relationship cycles;
- Rollbacks data sets from archive to production if needed;
- Maintains referential integrity and consistency. On production – no change, on archive DB - full support;
- Fulfills final archiving process goal – create fully functional archive repository DB and iteratively maintain future data sets movements including ability review or test archived data in for example duplicated production application where only just connection string should be changed.

EXECUTIVE SUMMARY

Continued database growth, long-term data retention regulations and storage requirements are increasing operational costs. As a result, more CIOs are examining the potential benefits of implementing cost-effective strategies for managing data and information throughout its life cycle.

Because most of the historical data is inactive, not frequently used and are stored in high-performance application databases, more companies are realizing the value of database archiving as an essential component in an effective data management strategy. Organizations need an effective archiving strategy for managing application data throughout its life cycle.

This white paper explains how the DBArchive Solution provides comprehensive technology and capabilities that make archiving complex relational data both practical and desirable. As a result, archives can be easily managed and stored on the most cost-effective media, so the historical data remains accessible.

As we know, maintaining only current active data in production environments helps to improve:

- Service levels thus better user experience;
- Increase your solution availability and improve disaster recovery initiatives;
- Reclaims valuable capacity and lower costs.

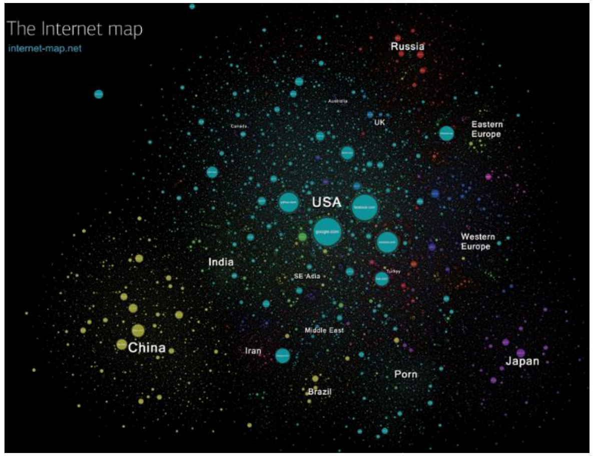
The DBArchive solution addresses a critical operational need for enterprises that rely on large complex relational databases.

DATA GROWTH. IMPACT & OPPORTUNITY.

Industry analysts and leading computing publications reports that the rapid accumulation of data worldwide continues at an unprecedented rate. This can be compared to the galaxy, in the sense it keeps expanding. We could touch this galaxy data in playful portal: The Internet Galaxy.

Figure 2 illustrates galaxy data that DBArchive solution is targeting as a potential to be optimized.

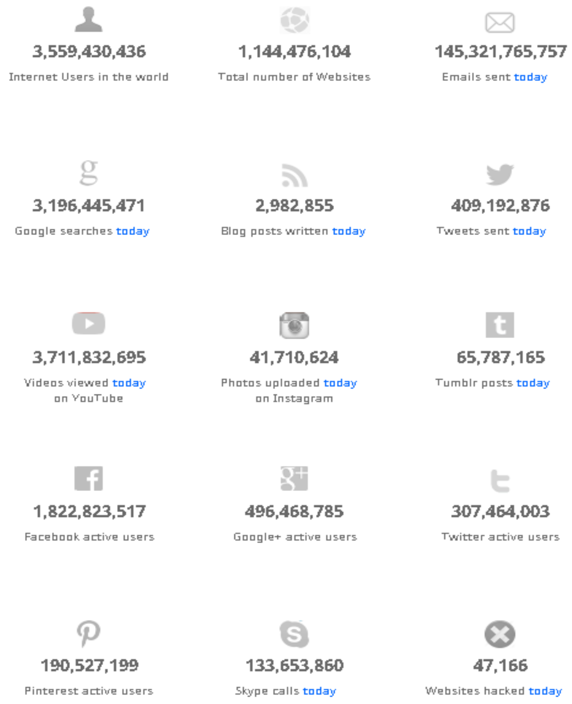
Figure 2 The Internet Galaxy



Nowadays there are over a billion websites and even more Internet users who consume and produce data.

Figure 3 illustrates Internet Live Stats.

Figure 3 Internet Live Stats

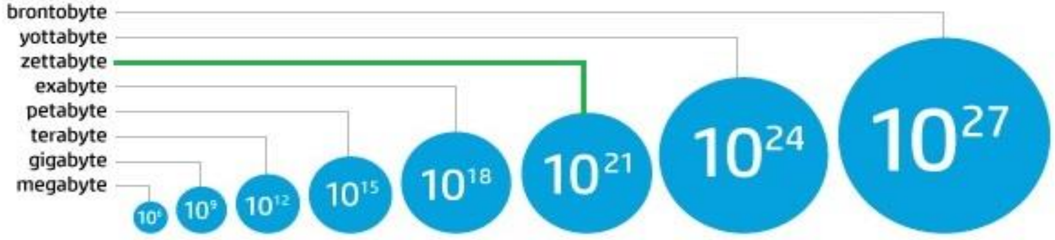


3,559,430,436 Internet Users in the world	1,144,476,104 Total number of Websites	145,321,765,757 Emails sent today
3,196,445,471 Google searches today	2,982,855 Blog posts written today	409,192,876 Tweets sent today
3,711,832,695 Videos viewed today on YouTube	41,710,624 Photos uploaded today on Instagram	65,787,165 Tumblr posts today
1,822,823,517 Facebook active users	496,468,785 Google+ active users	307,464,003 Twitter active users
190,527,199 Pinterest active users	133,653,860 Skype calls today	47,166 Websites hacked today

We could summarize current data situation as The Zettabyte Era Officially Begins. According to CISCO Visual Networking Index (VNI) estimate, the world's collective Internet traffic reached the Zettabyte threshold on September 9, 2016.

Figure 4 help us visualize the ratio of Zettabyte (ZB) compared to well-known Megabyte (MB).

Figure 4 Zettabyte Era ratio



Source: HP

According to Cisco Global Cloud Index, The Zettabyte Era-Trends and Analysis, Global Digitization: Impact of IoE our future will be:

- Global Data Center IP Traffic: Three-Fold Increase by 2020;
- Data Center Storage installed capacity will grow to 1.8 ZB: nearly a 5-fold growth by 2020.
- Cisco GCI estimates that 600 ZB will be generated by all people, machines, and things: four-fold increase by 2020;
- The number of devices connected to IP networks will be more than 3 times the global population by 2020;

Figure 5 illustrates data growth in near future.

Accumulating more data means degraded performance and slower response time. Availability is limited because routine database maintenance requires more processing time, often taking systems out of service.

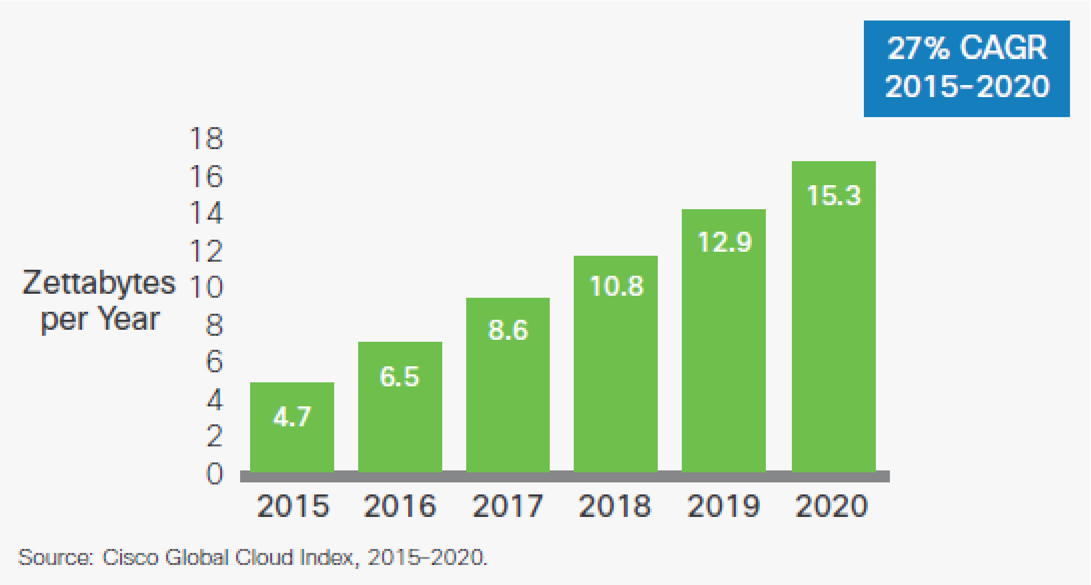
Forrester Research. Inc. estimates that, on average, structured data repositories for large application grow by 50 percent annually and **85 percent of data stored in databases is inactive.**

This is exactly what DBArchive solution is targeting - 85 percent of inactive data.

So, what happens when accelerating database growth is allowed to continue? For most companies, the solution to exponential database growth is to upgrade servers and acquire more storage capacity. Faster, more powerful processors can speed access to information, while the underlying databases continue to grow. However, this approach is rapidly losing its viability for many reasons:

1. **Poor performance and response time.** The larger the database, the more time it takes to load, unload, search, reorganize, index and optimize. Response time slows. Access to decision-making information becomes more difficult. Service levels decline.
2. **Limited system availability.** Managing more data means that maintenance tasks require more time to complete. Most companies struggle to complete their scheduled backups, index rebuilds etc. Without a way to safely remove older data from production databases, this problem will only worsen over time.
3. **Increasing costs.** Purchasing capacity upgrades might appear to be cost-effective as a "one-time" fix, but upgrades are needed on an ongoing basis to keep pace with database growth. As a result, IT organizations often spend millions of dollars in hardware and software license fees per year, just to expand server, storage and CPU capacity. Still, this short-term tactical approach is just a temporary solution for a much larger problem that is not going away—excessive data volume that continues to increase.
4. **Slow disaster recovery.** Many companies have made disaster recovery a high priority. In the event of a disaster, the key strategy is to get mission-critical systems operational as quickly as possible. With overloaded databases, all the data (including years and years of historical information) must be restored, just to get business-critical data back online. Restoring large volumes of rarely accessed data, in addition to the critical operational data, can slow the recovery process by hours or even days.

Figure 5 Data growth in near future by CISCO



Source: Cisco Global Cloud Index, 2015-2020.

FINDING THE IDEAL SOLUTION

Many IT organizations initiate projects to develop in-house solutions. Even with adequate time and resources, designing, developing, implementing and maintaining custom archiving solution for a particular application is an extremely labor-intensive and complex project.

Meeting the minimum requirements of archiving process is just the first step in enabling an enterprise database for archiving strategy. Additional challenges include enhancing and maintaining the custom archiving solution to support ongoing changes to the application and the database.

Here we could ask, what is the Ideal solution for the main problem: how to improve application global performance? And the answers could be – Horizontal Scaling, Elastic Caching or Partitioning, but let's review them one by one:

You Can't Easily Horizontally Scale A Database because:

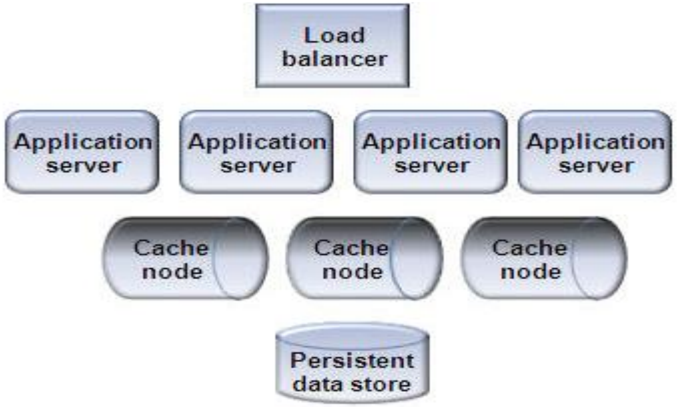
a. Difficulties due to joins;
b. Distributed system is prone to slow down operations which are bad tradeoff for modern apps that require fast response and have to deal with large heaps of input;
c. ACIDity. RDBMSes (Relational database management system) by design are ACID (Atomicity, Consistency, Isolation and Durability) which basically means - transactions and rows, table level locks. In distributed environments, it can be limiting factor if you have a lot of queries modifying a lot of data running at same time;
d. Relational databases are designed to run on a single server in order to maintain the integrity of the table mappings and avoid the problems of distributed computing. Otherwise customers must buy bigger, more complex, and more expensive proprietary hardware with more processing power, memory, and storage which is not cost-effective solution;
e. Upgrades are also a challenge, as the organization must go through a lengthy acquisition process, and then often take the system offline to actually make the change.

Alternative solutions like Elastic Caching or Partitioning have many drawbacks which are far away from simple, cost effective and ideal solution, for example:

1. Elastic Caching Platforms disadvantages:
   a. Caching platforms (Amazon ElastiCache, Alachisoft NCache, VMware GemFire, GigaSpaces XAP Elastic Application Platform, IBM WebSphere eXtreme Scale, Infinispan, ScaleOut StateServer, and Terracotta) comes with its own overhead, and not having a complete understanding of how much data are you working with and what are you trying to do with it, will lead you down the path of nowhere for example extreme risk management and off course huge investments in both money and human resources;
   b. Caching is fine for Web applications that run on one or two application servers, but it is insufficient if any or all the following conditions apply:
      i. The data is too big to fit in the application server memory space.
      ii. Cached data is updated and shared by users across multiple application servers.
      iii. User requests, and therefore user sessions, are not bound to a particular application server.
      iv. Failover is required without data loss.

   Figure 6 shows basic Elastic Caching architecture.

Figure 6 Elastic Caching architecture



2. Partitioning disadvantages:
   a. We cannot have different index models for different range. In general, we may need to have more number of indexes when the data is READ ONLY and less or different index model for the data which is READWRITE. That is not possible with partition table;
   b. You cannot rebuild a partitioned index with the ONLINE option set to ON, because the entire table will be locked during the rebuild;
   c. If your query does not join or filter on the partition key then there will be no improvement in performance over an unpartitioned table. In fact, a query that hits a partitioned table has the potential to be even slower than an unpartitioned table even if both tables have the same index defined. This is due to the fact that each partition in a partitioned table is actually its own b-tree which means that a partitioned index seek will need to do one seek per partition as opposed to one seek per table for an unpartitioned index seek;
   d. Queries may need to be updated to include the new partition key in the where clause or join to take advantage of the performance benefits of partitioning;
   e. Table partitioning is not a project that should be entered into lightly – it is a major structural change for the database and a major software change for database developers.

In contrast, why not to create universal archiving solution based on essential principle: client data in – intelligent solution out. DBArchive solution is the one.

WHAT IS DBARCHIVE SOLUTION?

- From the top abstraction level, DBArchive is a solution which can be labeled as a process that stands for improved global application performance by complying with basic archiving process (Figure 1).
- Going further, DBArchive is simple yet complex approach to discover and move inactive historical data from production to separate environment (Archive repository) and if needed vice versa.
- At the final stage, we can find that DBArchive solution is dynamic, iterative, continuous process that intelligently discovers, validates data, makes decisions and finally safely moves historical data sets from production to archive repository.

Without inactive historical data, global production environment performance increase and all remaining benefits (like user experience) improves too. These are the top priorities of DBArchive solution.

HOW DBARCHIVE SOLUTION WORKS?

As people say, one picture worth ten thousand words. So, let's present this solution with images. Figure 7 presents DBArchive solution architecture. In this architecture, we see two main components: server and client.

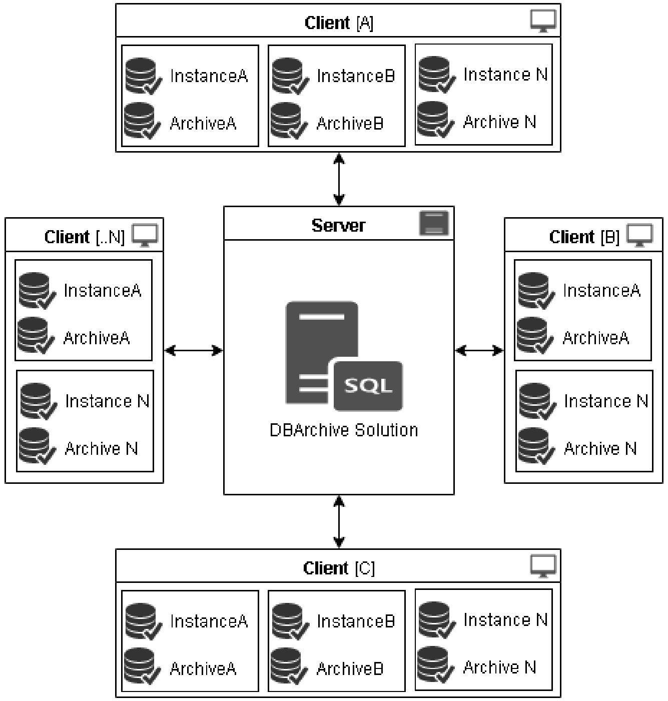
Server acts as a main engine for accepting Clients request, creates solutions and responds with them to requesting Client. Server response contains plain text DDL, DML statements targeted to specific client DB. Those statements can be downloaded from demo Configuration website (see typical scenario results).

Client acts as a worker (windows service, DbArchiveClient) who requests "jobs" from the server. Received jobs could be such as merge destination schema, move historical data, etc. These jobs are executed locally in Client environment/production.

This architecture is live because of archiving business process which could be laid down in these sequential steps:

1. Initial. Client provides initial data to Server configuration tables such as source database, archive repository names, physical database, backup paths, main objects names, types and their validity. When Server configuration is complete then Client receives prepared *.msi package which installs one Windows Service: DbArchive Client. DbArchive Client operates as a main connection point with the server;
2. Input. Client sends requests for new task to the server. Server responds with meta data request which is in fact only select statements of Dynamic Management Views. Client executes statements and responds back to server with meta data;
3. Analyze. Server performs analysis of received meta data and produces solution – select statements of referential key columns (foreign keys). This solution is interconnected with Initial step configuration data;
4. Input. In Client, select statements are executed against production DB and referential data (only key columns) are sent back to Server's local repository for auditing and further processing;

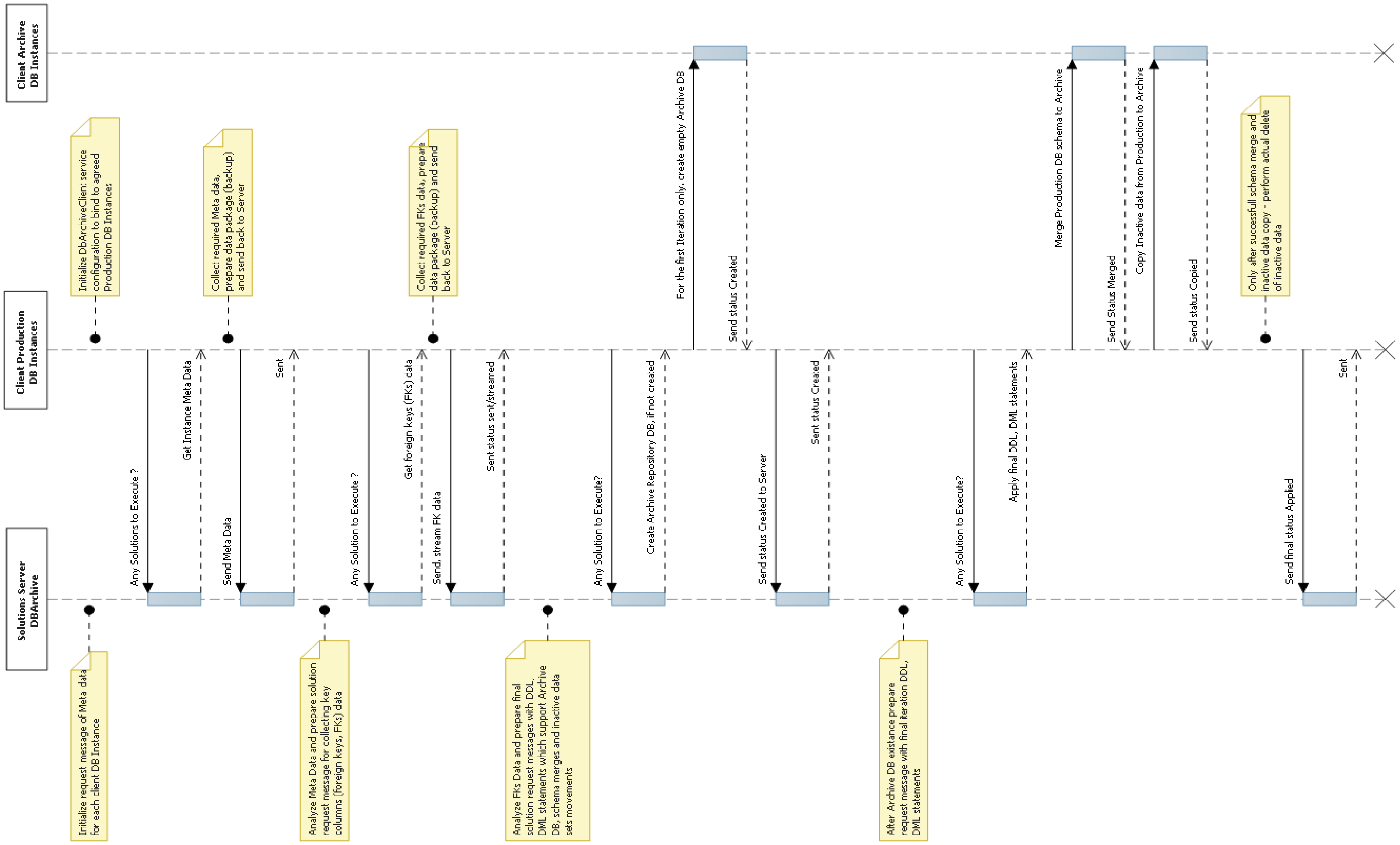
Figure 7 DBArchive architecture



5. Analyze. Server continues archiving process by discovering and analyzing referential data. When this process completes then final solution is produced. This final solution contains DDL, DML statements which describes archive schema merge and valid archiving data sets movements from production to archive repository;
6. Apply. Here, in the final stage, Client receives final solution from server and starts executing everything in new transaction. New transaction implementation is necessary because if for example integrity failure ocures all changes must be rollbacked. Execution result is responded back to server and persisted to DBArchive local repository;
7. Iterate. This workflow is repeated on periodic schedule;
8. Multi. Multi Production DB instances are supported and all this iterative workflow process is performed independently for each instance.

Figure 8 illustrates DBArchive process as a Sequence Diagram. Sequence Diagram represents details of archiving process in sequence (top->bottom, left->right) and timeline.

Figure 8 DBArchive process Sequence Diagram


Client Archive DB Instances
Client Production DB Instances
Solutions Server DBArchive
Initialize DbArchiveClient service configuration to bind to agreed Production DB Instances
Initialize request message of Meta data for each client DB Instance
Any Solutions to Execute ?
Get Instance Meta Data
Collect required Meta data, prepare data package (backup) and send back to Server
Send Meta Data
Sent
Analyze Meta Data and prepare solution request message for collecting key columns (foreign keys, FKs) data
Any Solution to Execute ?
Get foreign keys (FKs) data
Collect required FKs data, prepare data package (backup) and send back to Server
Send, stream FK data
Sent status sent/streamed
Analyze FKs Data and prepare final solution request messages with DDL, DML statements which support Archive DB, schema merges and inactive data sets movements
Any Solution to Execute?
Create Archive Repository DB, if not created
For the first Iteration only, create empty Archive DB
Send status Created
Send status Created to Server
Sent status Created
After Archive DB existance prepare request message with final iteration DDL, DML statements
Any Solution to Execute?
Apply final DDL, DML statements
Merge Production DB schema to Archive
Send Status Merged
Copy Inactive data from Production to Archive
Send status Copied
Only after successfull schema merge and inactive data copy - perform actual delete of inactive data
Send final status Applied
Sent

TYPICAL DATABASE ARCHIVING SCENARIO

STEP 1 – ANALYSIS OF CLIENT SOURCE/PRODUCTION DATABASE.

- According to RDBMS data is normalized and stored into so called main objects – tables;
- The table is a collection of related data entries where records are inter-connected with relations (foreign keys) to conform data integrity. All these foreign keys define the complexity of the database;
- Every database differs from the complexity view perspective, so to create Ideal/general solution is not an easy task. However, DBArchive stands up and addresses major complexity variations;
- To better understand complex relations, it's the best practice to visualize data and work with representing objects. In our case the most simple and self-describing model is Graph Data Model. Basically, this model contains just two different object types: nodes and links. Node represents Table and link represents Foreign key relation.

Figure 9 General/Demo Production DB model

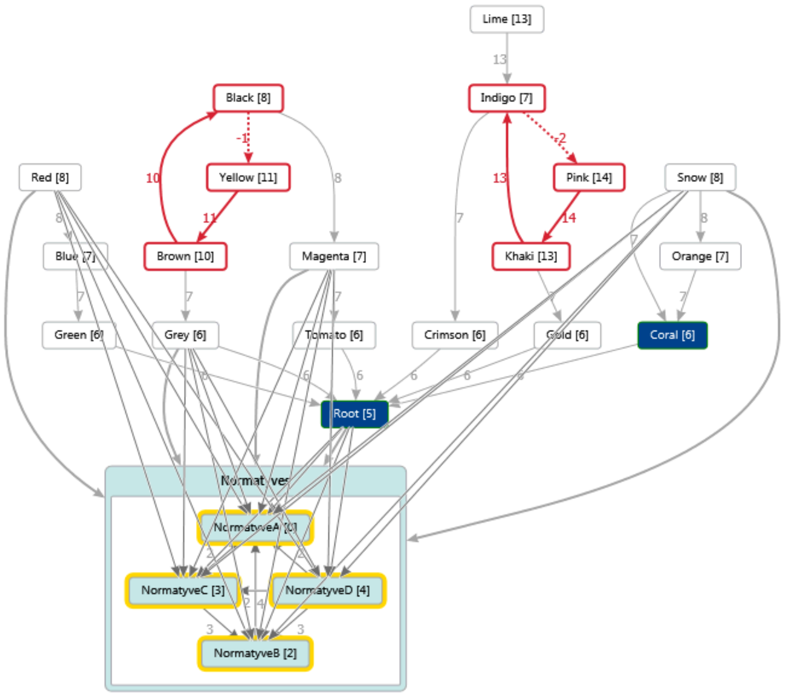
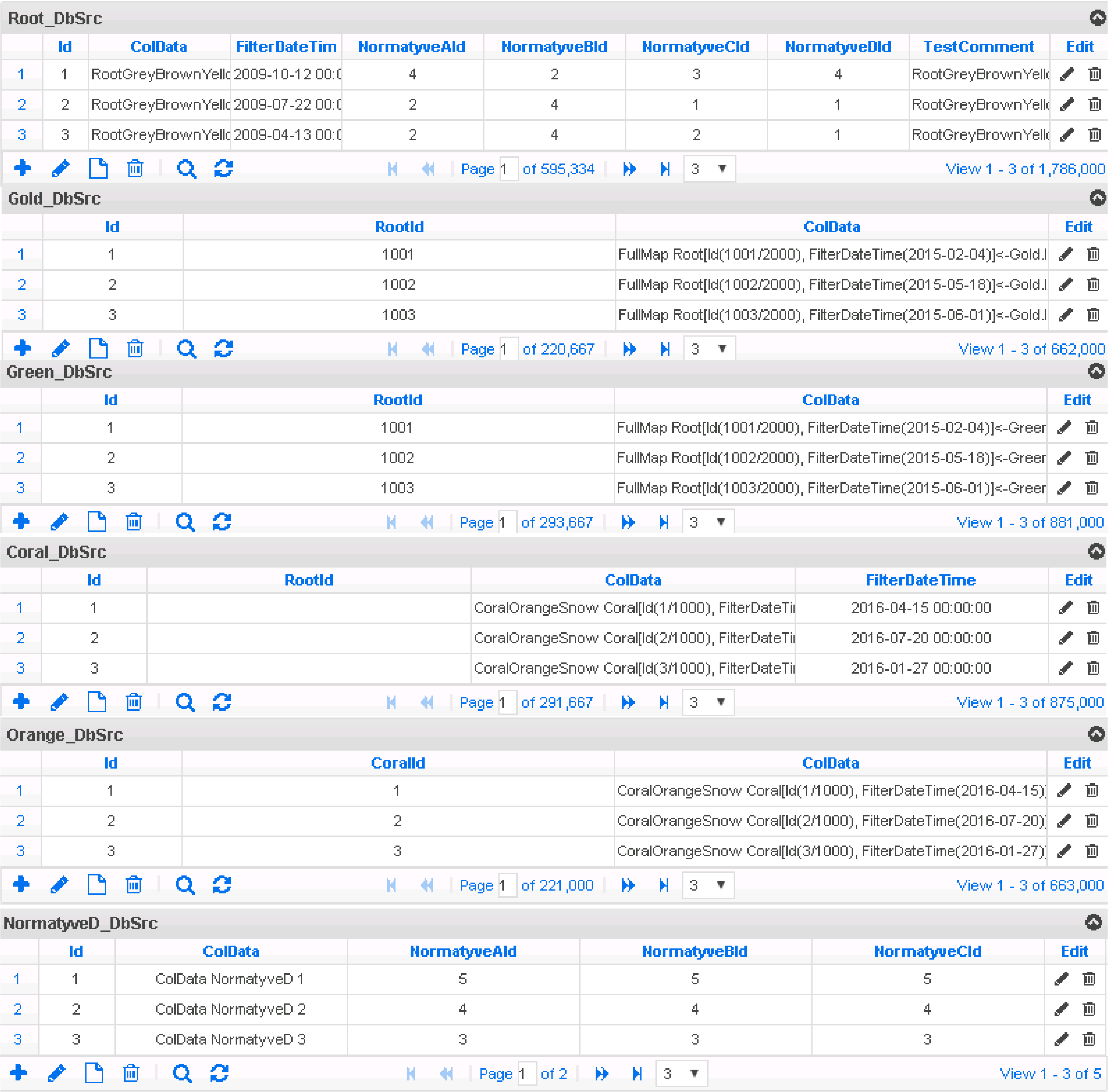


Figure 9 represents general (or demo) Production database model on which DBArchive is built. Ilustrated model contains these main objects:

1. Root nodes. These nodes ([Root], [Coral]) are starting points or from business perspective - main entities, for example Customers, WayBills, Accounts, etc…
    a. These entities should have columns which represents archiving filters like DateTime or TypeSets, for example: CreateDateTime, ValidToDateTime, IsExpired or anything else that forms data record groups and connected parent-child data sets.
2. Normatyve nodes. These nodes ([NormatyveA…D]) are common reference data for example Users, Stations, OrderTypes where they are referenced by almost all nodes.
    a. Normatyves are copied to destination or archive DB. Because they are common and we want archive Application to be fully functional for reviewing moved/archived data.
3. Links. Links represents relations between two nodes/tables.
    a. One table can have many links which are connected to Parent or Child nodes/tables;
    b. The same node could have self referencing links (small circle) and more interestingly we can have circular relations between many nodes in the connection path (big circles, marked in Red colour).

Figure 10 represents actual demo data of underlined database schema:

Figure 10 Actual demo data of General DB model



Root_DbSrc

	Id	ColData	FilterDateTim	NormatyveAId	NormatyveBId	NormatyveCId	NormatyveDId	TestComment	Edit
1	1	RootGreyBrownYellc	2009-10-12 00:(	4	2	3	4	RootGreyBrownYell(	✎ 🗑
2	2	RootGreyBrownYellc	2009-07-22 00:(	2	4	1	1	RootGreyBrownYell(	✎ 🗑
3	3	RootGreyBrownYellc	2009-04-13 00:(	2	4	2	1	RootGreyBrownYell(	✎ 🗑

Page 1 of 595,334 3 ▼ View 1 - 3 of 1,786,000

Gold_DbSrc

	Id	RootId	ColData	Edit
1	1	1001	FullMap Root[Id(1001/2000), FilterDateTime(2015-02-04)]<-Gold.I	✎ 🗑
2	2	1002	FullMap Root[Id(1002/2000), FilterDateTime(2015-05-18)]<-Gold.I	✎ 🗑
3	3	1003	FullMap Root[Id(1003/2000), FilterDateTime(2015-06-01)]<-Gold.I	✎ 🗑

Page 1 of 220,667 3 ▼ View 1 - 3 of 662,000

Green_DbSrc

	Id	RootId	ColData	Edit
1	1	1001	FullMap Root[Id(1001/2000), FilterDateTime(2015-02-04)]<-Greer	✎ 🗑
2	2	1002	FullMap Root[Id(1002/2000), FilterDateTime(2015-05-18)]<-Greer	✎ 🗑
3	3	1003	FullMap Root[Id(1003/2000), FilterDateTime(2015-06-01)]<-Greer	✎ 🗑

Page 1 of 293,667 3 ▼ View 1 - 3 of 881,000

Coral_DbSrc

	Id	RootId	ColData	FilterDateTime	Edit
1	1		CoralOrangeSnow Coral[Id(1/1000), FilterDateTi	2016-04-15 00:00:00	✎ 🗑
2	2		CoralOrangeSnow Coral[Id(2/1000), FilterDateTi	2016-07-20 00:00:00	✎ 🗑
3	3		CoralOrangeSnow Coral[Id(3/1000), FilterDateTi	2016-01-27 00:00:00	✎ 🗑

Page 1 of 291,667 3 ▼ View 1 - 3 of 875,000

Orange_DbSrc

	Id	CoralId	ColData	Edit
1	1	1	CoralOrangeSnow Coral[Id(1/1000), FilterDateTime(2016-04-15)]	✎ 🗑
2	2	2	CoralOrangeSnow Coral[Id(2/1000), FilterDateTime(2016-07-20)]	✎ 🗑
3	3	3	CoralOrangeSnow Coral[Id(3/1000), FilterDateTime(2016-01-27)]	✎ 🗑

Page 1 of 221,000 3 ▼ View 1 - 3 of 663,000

NormatyveD_DbSrc

	Id	ColData	NormatyveAId	NormatyveBId	NormatyveCId	Edit
1	1	ColData NormatyveD 1	5	5	5	✎ 🗑
2	2	ColData NormatyveD 2	4	4	4	✎ 🗑
3	3	ColData NormatyveD 3	3	3	3	✎ 🗑

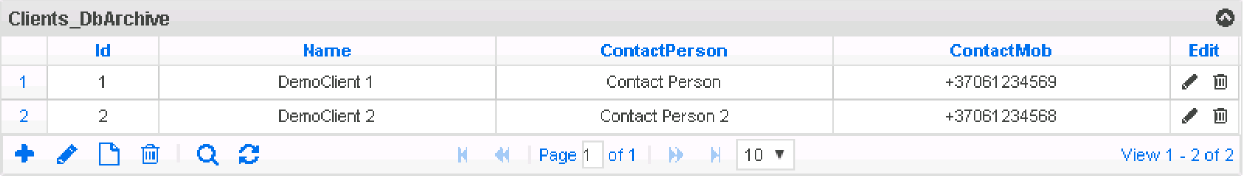
Page 1 of 2 3 ▼ View 1 - 3 of 5

STEP 2 – PROVIDE CLIENT CONFIGURATION DATA FOR DBARCHIVE SOLUTION.

Following Figures represents general/demo web based input grids where Clients provide initial configuration data.

1. Figure 11 represents Clients Contact Information:

Figure 11 Clients Contacts information



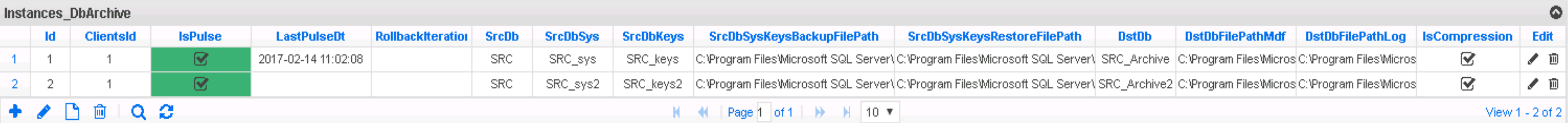
Clients_DbArchive

	Id	Name	ContactPerson	ContactMob	Edit
1	1	DemoClient 1	Contact Person	+37061234569	✎ 🗑
2	2	DemoClient 2	Contact Person 2	+37061234568	✎ 🗑

Page 1 of 1 | 10 ▼ | View 1 - 2 of 2

2. Figure 12 represents Source/Production database Instance properties:

Figure 12 Database Instance Properties



Instances_DbArchive

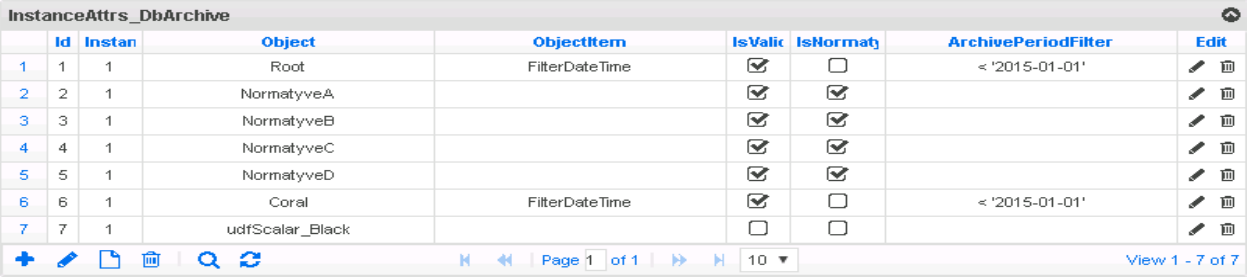
	Id	ClientsId	IsPulse	LastPulseDt	RollbackIteratio	SrcDb	SrcDbSys	SrcDbKeys	SrcDbSysKeysBackupFilePath	SrcDbSysKeysRestoreFilePath	DstDb	DstDbFilePathMdf	DstDbFilePathLog	IsCompression	Edit
1	1	1	☑	2017-02-14 11:02:08		SRC	SRC_sys	SRC_keys	C:\Program Files\Microsoft SQL Server\	C:\Program Files\Microsoft SQL Server\	SRC_Archive	C:\Program Files\Micros	C:\Program Files\Micros	☑	✎ 🗑
2	2	1	☑			SRC	SRC_sys2	SRC_keys2	C:\Program Files\Microsoft SQL Server\	C:\Program Files\Microsoft SQL Server\	SRC_Archive2	C:\Program Files\Micros	C:\Program Files\Micros	☑	✎ 🗑

Page 1 of 1 | 10 ▼ | View 1 - 2 of 2

   a. Notes: [SrcDb] – Production name, [DstDb] – Archive repository name, [*FilePath*] – source and destination physical paths;

3. Figure 13 represents Database instance objects attributes:

Figure 13 Production DB instance objects and attributes



InstanceAttrs_DbArchive

	Id	Instan	Object	ObjectItem	IsValic	IsNormaty	ArchivePeriodFilter	Edit
1	1	1	Root	FilterDateTime	☑	☐	< '2015-01-01'	✎ 🗑
2	2	1	NormatyveA		☑	☑		✎ 🗑
3	3	1	NormatyveB		☑	☑		✎ 🗑
4	4	1	NormatyveC		☑	☑		✎ 🗑
5	5	1	NormatyveD		☑	☑		✎ 🗑
6	6	1	Coral	FilterDateTime	☑	☐	< '2015-01-01'	✎ 🗑
7	7	1	udfScalar_Black		☐	☐		✎ 🗑

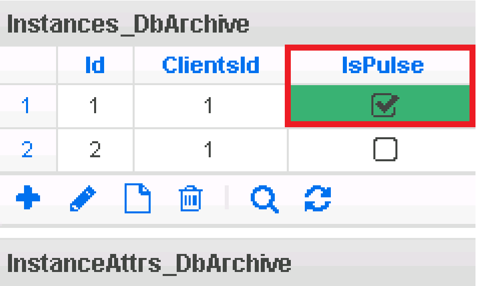
Page 1 of 1 | 10 ▼ | View 1 - 7 of 7

   a. Notes: Here we can provide Root objects and their archive filters, validity and IsNormatyve type attributes

STEP 3 – START ARCHIVING PROCESS BY CHECK-MARKING COLUMN [ISPULSE] OF [INSTANCES] GRID

In DBArchive solution initialization ends in final stage which is implemented as check box. If check box is checked – archiving process starts, else – no process is performed. Figure 14 shows final stage.

Figure 14 Final preparation stage, starting archiving process



Instances_DbArchive

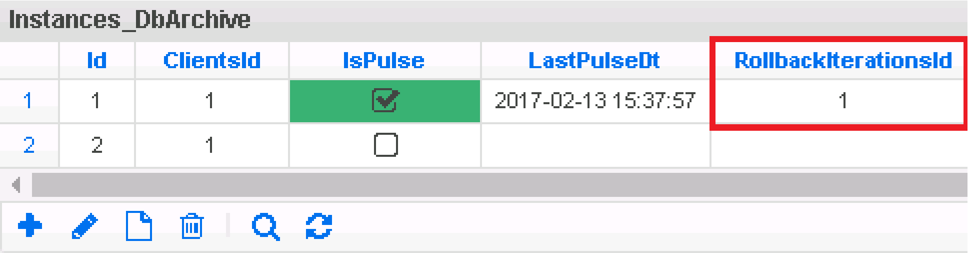
	Id	ClientsId	IsPulse
1	1	1	☑
2	2	1	□

InstanceAttrs_DbArchive

STEP 4 (OPTIONAL) – ROLLBACK ITERATION

DBArchive solution has a future: rollback entire previous iteration data sets. This feature could be executed by providing data in column [RollbackIterationsId] of [Instances_DbArchive] grid. Figure 15 shows rollback feature.

Figure 15 Iteration rollback feature



Instances_DbArchive

	Id	ClientsId	IsPulse	LastPulseDt	RollbackIterationsId
1	1	1	☑	2017-02-13 15:37:57	1
2	2	1	□		

RESULTS

Here you can find Summary and Detailed results. Summary results are dedicated for your quick insights and to keep your interests live. Note, all these results depend on workstation hardware specifics, so please find hardware reference in the end of this document.

Summary results:

1. PRODUCTION DATABASE PERFORMANCE INCREASED BY 95%

a. Before: 4 minutes, after: 11 seconds.

2. PRODUCTION DISK SPACE GAINED 85%

a. Before: 2666.38 MB, after: 391.75 MB;

b. Created new Archive DB with size of 2063.75 MB.

3. FIRST ARCHIVING ITERATION TOOK ONLY 15 MINUTES, NEXT ONE WILL TAKE EVEN LESS

a. Complex database relational analysis took 3 minutes;

b. Final solution batch was committed per 12 minutes.

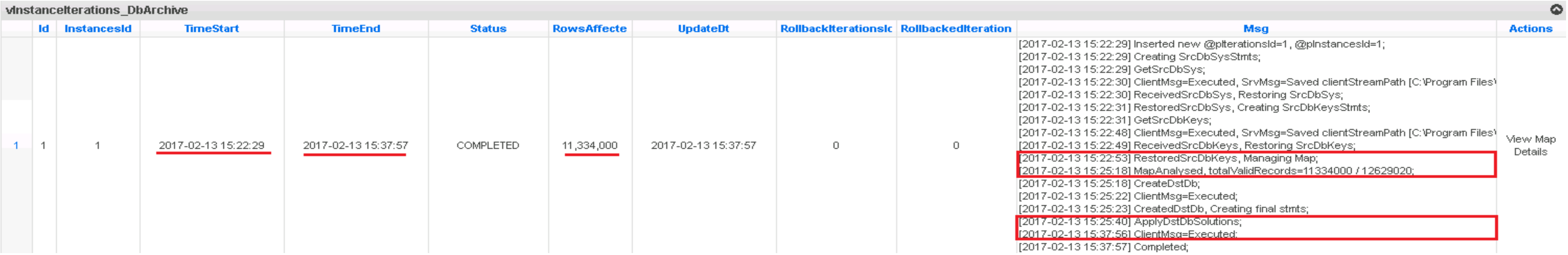
4. MOVED 11,334,000 RECORDS TO ARCHIVE REPOSITORY

a. Production database had 12,629,020 records, after archiving - 1,295,020 active records left.

Detailed results:

1. Grids [vInstanceIterations]. Archiving process is built on iterations. In following images, we could see what exactly was performed. Columns [TimeStart], [TimeEnd], [RowsAffected], [RollbackedIterationsId], [Msg] give us better understanding about what and when process was started, executed and finished.
   - In Figure 16 grid stands for the first iteration and data moving to Archive repository;

Figure 16 Iteration details of archiving data



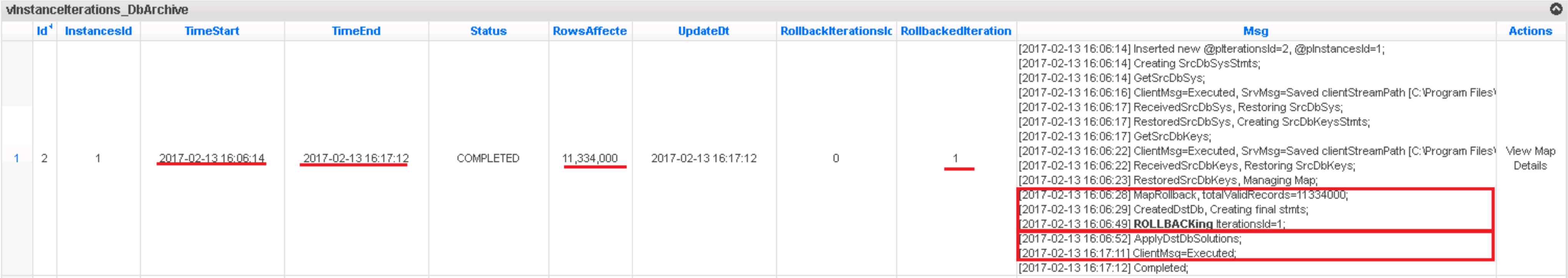
vInstanceIterations_DbArchive

	Id	InstancesId	TimeStart	TimeEnd	Status	RowsAffecte	UpdateDt	RollbackIterationsIc	RollbackedIteration	Msg	Actions

1	1	1	2017-02-13 15:22:29	2017-02-13 15:37:57	COMPLETED	11,334,000	2017-02-13 15:37:57	0	0	[2017-02-13 15:22:29] Inserted new @pIterationsId=1, @pInstancesId=1;
										[2017-02-13 15:22:29] Creating SrcDbSysStmts;
										[2017-02-13 15:22:29] GetSrcDbSys;
										[2017-02-13 15:22:30] ClientMsg=Executed, SrvMsg=Saved clientStreamPath [C:\Program Files\
										[2017-02-13 15:22:30] ReceivedSrcDbSys, Restoring SrcDbSys;
										[2017-02-13 15:22:31] RestoredSrcDbSys, Creating SrcDbKeysStmts;
										[2017-02-13 15:22:31] GetSrcDbKeys;
										[2017-02-13 15:22:48] ClientMsg=Executed, SrvMsg=Saved clientStreamPath [C:\Program Files\
										[2017-02-13 15:22:49] ReceivedSrcDbKeys, Restoring SrcDbKeys;
										[2017-02-13 15:22:53] RestoredSrcDbKeys, Managing Map;
										[2017-02-13 15:25:18] MapAnalysed, totalValidRecords=11334000 / 12629020;
										[2017-02-13 15:25:18] CreateDstDb;
										[2017-02-13 15:25:22] ClientMsg=Executed;
										[2017-02-13 15:25:23] CreatedDstDb, Creating final stmts;
										[2017-02-13 15:25:40] ApplyDstDbSolutions;
										[2017-02-13 15:37:56] ClientMsg=Executed;
										[2017-02-13 15:37:57] Completed;
											View Map Details

- In Figure 17 grid stands for second iteration and rollback process:

Figure 17 Rollback iteration details



vInstanceIterations_DbArchive

	Id	InstancesId	TimeStart	TimeEnd	Status	RowsAffecte	UpdateDt	RollbackIterationsIc	RollbackedIteration	Msg	Actions

1	2	1	2017-02-13 16:06:14	2017-02-13 16:17:12	COMPLETED	11,334,000	2017-02-13 16:17:12	0	1	[2017-02-13 16:06:14] Inserted new @pIterationsId=2, @pInstancesId=1;
										[2017-02-13 16:06:14] Creating SrcDbSysStmts;
										[2017-02-13 16:06:14] GetSrcDbSys;
										[2017-02-13 16:06:16] ClientMsg=Executed, SrvMsg=Saved clientStreamPath [C:\Program Files\
										[2017-02-13 16:06:17] ReceivedSrcDbSys, Restoring SrcDbSys;
										[2017-02-13 16:06:17] RestoredSrcDbSys, Creating SrcDbKeysStmts;
										[2017-02-13 16:06:17] GetSrcDbKeys;
										[2017-02-13 16:06:22] ClientMsg=Executed, SrvMsg=Saved clientStreamPath [C:\Program Files\
										[2017-02-13 16:06:22] ReceivedSrcDbKeys, Restoring SrcDbKeys;
										[2017-02-13 16:06:23] RestoredSrcDbKeys, Managing Map;
										[2017-02-13 16:06:28] MapRollback, totalValidRecords=11334000;
										[2017-02-13 16:06:29] CreatedDstDb, Creating final stmts;
										[2017-02-13 16:06:49] **ROLLBACKing** IterationsId=1;
										[2017-02-13 16:06:52] ApplyDstDbSolutions;
										[2017-02-13 16:17:11] ClientMsg=Executed;
										[2017-02-13 16:17:12] Completed;
											View Map Details

Key notes in these grids.

- Column [TimeStart] and [TimeEnd] represents overall Iteration lifetime: 15 minutes;
- [RowsAffacted] show total data set record count of iteration type (data move or rollback): 11,334,000;
- [RollbackedIterationsId] notes does this iteration is a rollback type;
- [Msg] gives us full picture of what, where all processing was performed and with what responses was executed:
  a. How much time took of Map analysis: 3 minutes;

b. How many data records are valid for further archiving process: 11,334,000 / 12,629,020;

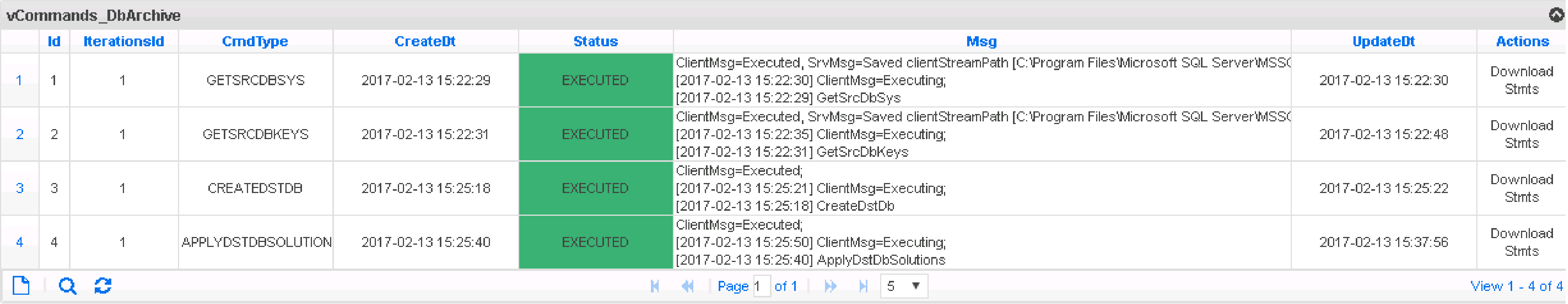
c. How much time took to apply final solution: 12 minutes;

- [Actions] lets us view archive analysis Map details;

2. Grids [vCommands].

   - These grid represent what commands were generated and executed. First represent first iteration with data move and second stands for rollbacked iteration commands;
   - Columns [CmdType], [CreateDt], [Status] and [Msg] provided full picture when and what solution was performed;
   - Column [Action] and action [Download Stmts] lets us download plain t-sql statements of what exactly was committed;

Figure 18 Archiving data iteration commands



vCommands_DbArchive

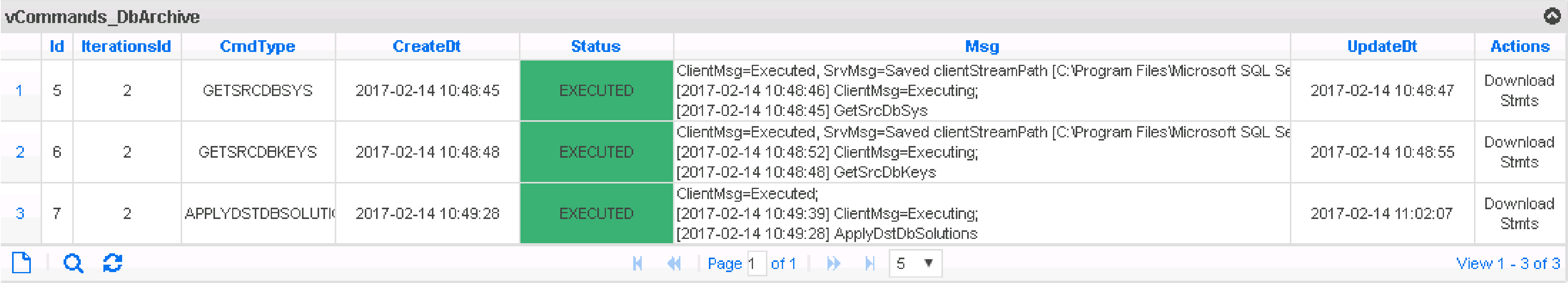
	Id	IterationsId	CmdType	CreateDt	Status	Msg	UpdateDt	Actions
1	1	1	GETSRCDBSYS	2017-02-13 15:22:29	EXECUTED	ClientMsg=Executed, SrvMsg=Saved clientStreamPath [C:\Program Files\Microsoft SQL Server\MSS([2017-02-13 15:22:30] ClientMsg=Executing; [2017-02-13 15:22:29] GetSrcDbSys	2017-02-13 15:22:30	Download Stmts
2	2	1	GETSRCDBKEYS	2017-02-13 15:22:31	EXECUTED	ClientMsg=Executed, SrvMsg=Saved clientStreamPath [C:\Program Files\Microsoft SQL Server\MSS([2017-02-13 15:22:35] ClientMsg=Executing; [2017-02-13 15:22:31] GetSrcDbKeys	2017-02-13 15:22:48	Download Stmts

3	3	1	CREATEDSTDB	2017-02-13 15:25:18	EXECUTED	ClientMsg=Executed;
						[2017-02-13 15:25:21] ClientMsg=Executing;
						[2017-02-13 15:25:18] CreateDstDb
							2017-02-13 15:25:22	Download Stmts

4	4	1	APPLYDSTDBSOLUTION	2017-02-13 15:25:40	EXECUTED	ClientMsg=Executed;
						[2017-02-13 15:25:50] ClientMsg=Executing;
						[2017-02-13 15:25:40] ApplyDstDbSolutions
							2017-02-13 15:37:56	Download Stmts

Page 1 of 1 | 5 | View 1 - 4 of 4

Figure 19 Rollback iteration commands



vCommands_DbArchive

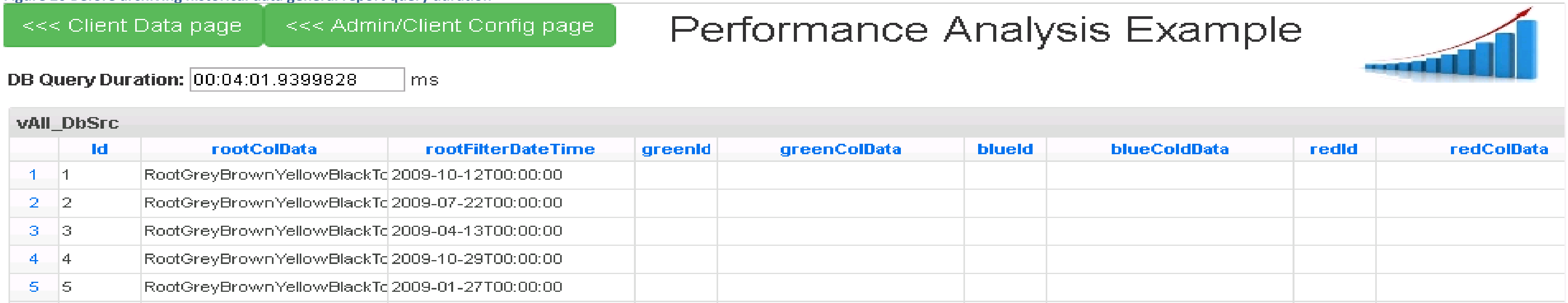
	Id	IterationsId	CmdType	CreateDt	Status	Msg	UpdateDt	Actions
1	5	2	GETSRCDBSYS	2017-02-14 10:48:45	EXECUTED	ClientMsg=Executed, SrvMsg=Saved clientStreamPath [C:\Program Files\Microsoft SQL Se [2017-02-14 10:48:46] ClientMsg=Executing; [2017-02-14 10:48:45] GetSrcDbSys	2017-02-14 10:48:47	Download Stmts
2	6	2	GETSRCDBKEYS	2017-02-14 10:48:48	EXECUTED	ClientMsg=Executed, SrvMsg=Saved clientStreamPath [C:\Program Files\Microsoft SQL Se [2017-02-14 10:48:52] ClientMsg=Executing; [2017-02-14 10:48:48] GetSrcDbKeys	2017-02-14 10:48:55	Download Stmts

3	7	2	APPLYDSTDBSOLUTI(	2017-02-14 10:49:28	EXECUTED	ClientMsg=Executed;
						[2017-02-14 10:49:39] ClientMsg=Executing;
						[2017-02-14 10:49:28] ApplyDstDbSolutions
							2017-02-14 11:02:07	Download Stmts

Page 1 of 1 | 5 | View 1 - 3 of 3

3. Grids [vAll] and performance analysis example. Created one big View which joins all tables of the demo schema. The most important element is [DB Query Duration] text box which shows duration of executed sql view:
   - Before all archiving activities grid [vAll] executed in 4 minutes;
   - After archiving process – only in 11 seconds;
   - Before and After figures:

Figure 20 Before archiving historical data general report query duration

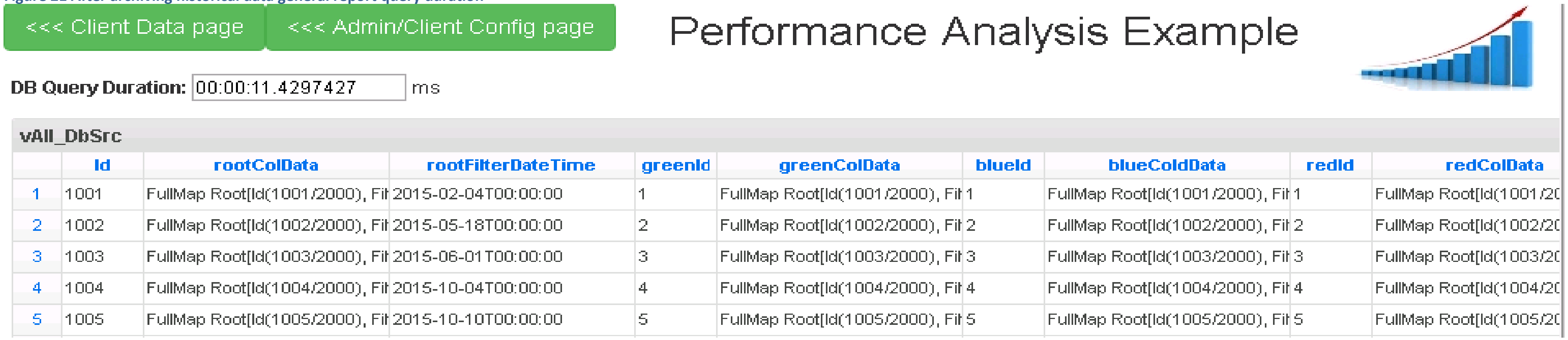


DB Query Duration: 00:04:01.9399828 ms

vAll_DbSrc

	Id	rootColData	rootFilterDateTime	greenId	greenColData	blueId	blueColdData	redId	redColData
1	1	RootGreyBrownYellowBlackTc	2009-10-12T00:00:00						
2	2	RootGreyBrownYellowBlackTc	2009-07-22T00:00:00						
3	3	RootGreyBrownYellowBlackTc	2009-04-13T00:00:00						
4	4	RootGreyBrownYellowBlackTc	2009-10-29T00:00:00						
5	5	RootGreyBrownYellowBlackTc	2009-01-27T00:00:00						

Figure 21 After archiving historical data general report query duration



DB Query Duration: 00:00:11.4297427 ms

vAll_DbSrc

	Id	rootColData	rootFilterDateTime	greenId	greenColData	blueId	blueColdData	redId	redColData
1	1001	FullMap Root[Id(1001/2000), Fil	2015-02-04T00:00:00	1	FullMap Root[Id(1001/2000), Fil	1	FullMap Root[Id(1001/2000), Fil	1	FullMap Root[Id(1001/2(
2	1002	FullMap Root[Id(1002/2000), Fil	2015-05-18T00:00:00	2	FullMap Root[Id(1002/2000), Fil	2	FullMap Root[Id(1002/2000), Fil	2	FullMap Root[Id(1002/2(
3	1003	FullMap Root[Id(1003/2000), Fil	2015-06-01T00:00:00	3	FullMap Root[Id(1003/2000), Fil	3	FullMap Root[Id(1003/2000), Fil	3	FullMap Root[Id(1003/2(
4	1004	FullMap Root[Id(1004/2000), Fil	2015-10-04T00:00:00	4	FullMap Root[Id(1004/2000), Fil	4	FullMap Root[Id(1004/2000), Fil	4	FullMap Root[Id(1004/2(
5	1005	FullMap Root[Id(1005/2000), Fil	2015-10-10T00:00:00	5	FullMap Root[Id(1005/2000), Fil	5	FullMap Root[Id(1005/2000), Fil	5	FullMap Root[Id(1005/2(

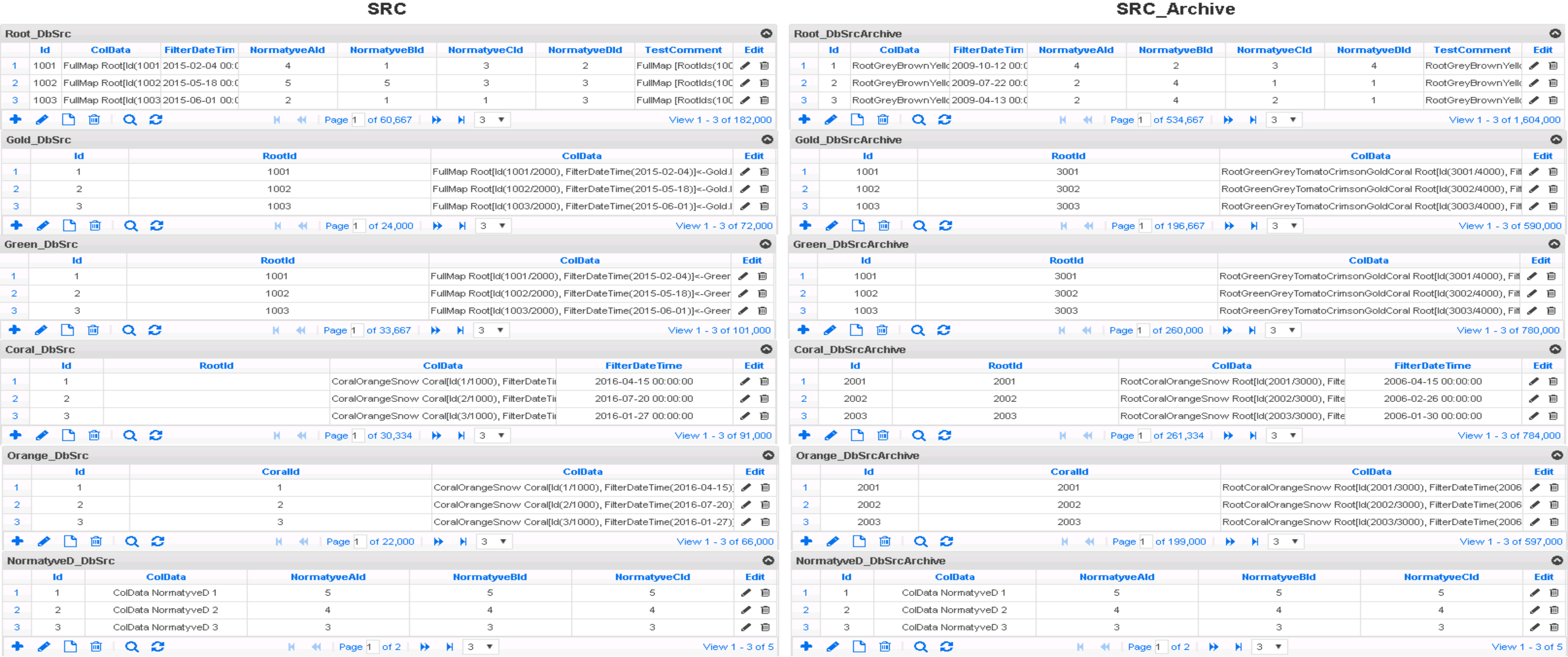
Before and after archiving we see results which leads as to 99% performance improvement. The reason why we have such outstanding results is that almost all data was moved to archive repository (11 of 12 million) and production left with 1 million records.

4. Production (SRC) and archive (SRC_Archive) application demo results (aligned main tables for better comparison):
   - Here we should pay attention to differences of tables total record counts before and after archiving process:

Table	Before	After	Archive DB
Root	1,786,000	182,000	1,604,000
Gold	662,000	72,000	590,000
Green	881,000	101,000	780,000
Coral	875,000	91,000	784,000
Orange	663,000	66,000	597,000

- Demo applications results in Figure 22:

Figure 22 After archiving historical data demo production (SRC) and Archive applications data



SRC

Root_DbSrc

	Id	ColData	FilterDateTim	NormatyveAId	NormatyveBId	NormatyveCId	NormatyveDId	TestComment	Edit
1	1001	FullMap Root[Id(1001	2015-02-04 00:(	4	1	3	2	FullMap [RootIds(10(	✎ 🗑
2	1002	FullMap Root[Id(1002	2015-05-18 00:(	5	5	3	3	FullMap [RootIds(10(	✎ 🗑
3	1003	FullMap Root[Id(1003	2015-06-01 00:(	2	1	1	3	FullMap [RootIds(10(	✎ 🗑

Page 1 of 60,667 3 ▼ View 1 - 3 of 182,000

Gold_DbSrc

	Id	RootId	ColData	Edit
1	1	1001	FullMap Root[Id(1001/2000), FilterDateTime(2015-02-04)]<-Gold.I	✎ 🗑
2	2	1002	FullMap Root[Id(1002/2000), FilterDateTime(2015-05-18)]<-Gold.I	✎ 🗑
3	3	1003	FullMap Root[Id(1003/2000), FilterDateTime(2015-06-01)]<-Gold.I	✎ 🗑

Page 1 of 24,000 3 ▼ View 1 - 3 of 72,000

Green_DbSrc

	Id	RootId	ColData	Edit
1	1	1001	FullMap Root[Id(1001/2000), FilterDateTime(2015-02-04)]<-Greer	✎ 🗑
2	2	1002	FullMap Root[Id(1002/2000), FilterDateTime(2015-05-18)]<-Greer	✎ 🗑
3	3	1003	FullMap Root[Id(1003/2000), FilterDateTime(2015-06-01)]<-Greer	✎ 🗑

Page 1 of 33,667 3 ▼ View 1 - 3 of 101,000

Coral_DbSrc

	Id	RootId	ColData	FilterDateTime	Edit
1	1		CoralOrangeSnow Coral[Id(1/1000), FilterDateTi	2016-04-15 00:00:00	✎ 🗑
2	2		CoralOrangeSnow Coral[Id(2/1000), FilterDateTi	2016-07-20 00:00:00	✎ 🗑
3	3		CoralOrangeSnow Coral[Id(3/1000), FilterDateTi	2016-01-27 00:00:00	✎ 🗑

Page 1 of 30,334 3 ▼ View 1 - 3 of 91,000

Orange_DbSrc

	Id	CoralId	ColData	Edit
1	1	1	CoralOrangeSnow Coral[Id(1/1000), FilterDateTime(2016-04-15)	✎ 🗑
2	2	2	CoralOrangeSnow Coral[Id(2/1000), FilterDateTime(2016-07-20)	✎ 🗑
3	3	3	CoralOrangeSnow Coral[Id(3/1000), FilterDateTime(2016-01-27)	✎ 🗑

Page 1 of 22,000 3 ▼ View 1 - 3 of 66,000

NormatyveD_DbSrc

	Id	ColData	NormatyveAId	NormatyveBId	NormatyveCId	Edit
1	1	ColData NormatyveD 1	5	5	5	✎ 🗑
2	2	ColData NormatyveD 2	4	4	4	✎ 🗑
3	3	ColData NormatyveD 3	3	3	3	✎ 🗑

Page 1 of 2 3 ▼ View 1 - 3 of 5

SRC_Archive

Root_DbSrcArchive

	Id	ColData	FilterDateTim	NormatyveAId	NormatyveBId	NormatyveCId	NormatyveDId	TestComment	Edit
1	1	RootGreyBrownYellc	2009-10-12 00:(	4	2	3	4	RootGreyBrownYellc	✎ 🗑
2	2	RootGreyBrownYellc	2009-07-22 00:(	2	4	1	1	RootGreyBrownYellc	✎ 🗑
3	3	RootGreyBrownYellc	2009-04-13 00:(	2	4	2	1	RootGreyBrownYellc	✎ 🗑

Page 1 of 534,667 3 ▼ View 1 - 3 of 1,604,000

Gold_DbSrcArchive

	Id	RootId	ColData	Edit
1	1001	3001	RootGreenGreyTomatoCrimsonGoldCoral Root[Id(3001/4000), Filt	✎ 🗑
2	1002	3002	RootGreenGreyTomatoCrimsonGoldCoral Root[Id(3002/4000), Filt	✎ 🗑
3	1003	3003	RootGreenGreyTomatoCrimsonGoldCoral Root[Id(3003/4000), Filt	✎ 🗑

Page 1 of 196,667 3 ▼ View 1 - 3 of 590,000

Green_DbSrcArchive

	Id	RootId	ColData	Edit
1	1001	3001	RootGreenGreyTomatoCrimsonGoldCoral Root[Id(3001/4000), Filt	✎ 🗑
2	1002	3002	RootGreenGreyTomatoCrimsonGoldCoral Root[Id(3002/4000), Filt	✎ 🗑
3	1003	3003	RootGreenGreyTomatoCrimsonGoldCoral Root[Id(3003/4000), Filt	✎ 🗑

Page 1 of 260,000 3 ▼ View 1 - 3 of 780,000

Coral_DbSrcArchive

	Id	RootId	ColData	FilterDateTime	Edit
1	2001	2001	RootCoralOrangeSnow Root[Id(2001/3000), Filte	2006-04-15 00:00:00	✎ 🗑
2	2002	2002	RootCoralOrangeSnow Root[Id(2002/3000), Filte	2006-02-26 00:00:00	✎ 🗑
3	2003	2003	RootCoralOrangeSnow Root[Id(2003/3000), Filte	2006-01-30 00:00:00	✎ 🗑

Page 1 of 261,334 3 ▼ View 1 - 3 of 784,000

Orange_DbSrcArchive

	Id	CoralId	ColData	Edit
1	2001	2001	RootCoralOrangeSnow Root[Id(2001/3000), FilterDateTime(2006	✎ 🗑
2	2002	2002	RootCoralOrangeSnow Root[Id(2002/3000), FilterDateTime(2006	✎ 🗑
3	2003	2003	RootCoralOrangeSnow Root[Id(2003/3000), FilterDateTime(2006	✎ 🗑

Page 1 of 199,000 3 ▼ View 1 - 3 of 597,000

NormatyveD_DbSrcArchive

	Id	ColData	NormatyveAId	NormatyveBId	NormatyveCId	Edit
1	1	ColData NormatyveD 1	5	5	5	✎ 🗑
2	2	ColData NormatyveD 2	4	4	4	✎ 🗑
3	3	ColData NormatyveD 3	3	3	3	✎ 🗑

Page 1 of 2 3 ▼ View 1 - 3 of 5

This typical scenario represents DBArchive solution capabilities in practice and key notes are:

1. None of integrity and consistency was violated;
2. No production runtime was stopped in order apply archiving process;
3. All data was successfully moved to Archive repository and vice versa.

DBARCHIVE DELIVERS BUSINESS VALUE

DBArchive is designed to increase company IT productivity. Organizations can also look forward to improving the quality of service, lowering the cost of ownership and supporting the governance of data, databases and data-driven applications.

In general, overloaded databases can be reduced by up to 50 percent or more during the initial archive iteration. Database and application performance improves thus valuable processing power is no longer required to support inactive data.

DBArchive offers a value propositions that can significantly lower Total Cost of Ownership (TCO) by optimizing company's investment in the relational databases that support mission-critical business applications:

- Improved global performance. With only relevant data held in production environment batch processing times improves and overall database availability improves too.
- Cost Reduction. Defer capacity upgrades and the associated expensive hardware and software license fees. Also, we can store archived database on the most convenient and cost-effective media;
- Faster Disaster Recovery, maintenance tasks. Production application left with only active data occupies less disk space therefore we have increased disaster recovery time by hours or days just by keeping only business-critical data. Also, we will notice better execution times in maintenance tasks (e.g. index rebuilds, statistics update, etc.) In case we should apply application upgrades which touches main schema objects (new indexes, full scan updates, etc.) we will win again by reduced down-time for application and better compliance of acceptable service levels (e.g. client agreements on maximum DB query execution timeout values and similar);
- Multiple Instance support. We can deploy a single, comprehensive archive solution to solve the problem of database growth across all applications in the enterprise;
- Archived data can be browsed directly and restored as needed;
- Human resources. By implementing regular, automated data archiving and purging, your IT team will spend less time on database maintenance and backups;
- Better user experience. By splitting out the historical data to a separate database, the main Production database is reduced in size, helping the overall application performance, response time improves and more clients' expectations are met.
- No, or very short down-time for production application. DBArchive solution could be iteratively configured to increase Archiving Filters to better suite mission-critical application overload. For example, we want to archive all but last two years of data. In this case, we could archive iteratively by smaller time frames (e.g. one month) until we reach two years' limit.
- Automatic and iterative archiving process continues to deliver long-term benefits.

SUMMARY

Data is the lifeblood of any organization. If data growth is not adequately managed, what was once a source of competitive advantage can quickly turn into a serious threat to the business.

Untamed data growth adds additional costs and most importantly - reduces productivity. As data grows and information flows more slowly through the organization, resources, both human and technological, become increasingly strained.

By reducing production data sets to more manageable levels, enterprise applications run faster and cost less to manage because existing infrastructure is utilized more efficiently. Data is archived based on business rules and retention policies to meet compliance objectives.

Access to all archived data is available through native application tools such as MS SQL services: Database Engine, Analysis, Reporting and Integration.

Results from DBArchive solution proves that moving inactive data can be done in general, simple and intelligent way.

HARDWARE SPECIFICATIONS

These hardware specifications represent machine on which all DBArchive solution was developed, tested and presented.

Motherboard	CPU Type	QuadCore Intel Core i5-4670, 3600 MHz (36 x 100)
	Motherboard Name	Intel Spring Cave DQ87PG (2 PCI, 1 PCI-E x1, 1 PCI-E x16, 4 DDR3 DIMM, Audio, Video, Gigabit LAN)
	Motherboard Chipset	Intel Lynx Point Q87, Intel Haswell
	System Memory	16328 MB (DDR3-1600 DDR3 SDRAM)
Storage	IDE Controller	Intel(R) 8 Series/C220 Chipset Family SATA AHCI Controller
	Disk Drive	Samsung SSD 850 EVO 250G SCSI Disk Device (232 GB)