# Towards Comparative Analysis of Resumption Techniques in ETL

**M Muddasir[1], Raghuveer[2], and Dayanand[3]**

[1]Asst. Prof, Dept. of IS&E, Vidyavardhaka College of Engineering, Mysuru, India
[2]Prof and Head, Dept. of IS&E, National Institute of Engineering, Mysuru, India
[3]Technical Director, Mysuru, India

E-mail: mdmsir@gmail.com[1], raghunie@yahoo.com[2], dayanand_7@yahoo.com[3]

**Submitted: 27 October 2020, revised: 10 February 2021, accepted: 12 February 2021**

**Abstrak.** Gudang data dimuat dengan data dari sumber-sumber seperti basis data operasional. Kegagalan proses pemuatan atau kegagalan salah satu proses seperti ekstraksi atau transformasi menyebabkan kerugian finansial yang besar karena tidak tersedianya data untuk analisis. Dengan munculnya e-commerce dan banyak aplikasi *real time*, analisis data dalam waktu nyata menjadi sesuatu yang umum. Oleh karena itu, setiap kesalahan saat data sedang dimuat ke dalam data warehouse perlu ditangani dengan cara yang efisien dan optimal. Teknik untuk menangani kegagalan proses untuk mengisi data sama pentingnya dengan proses pemuatan yang sebenarnya. Penataan alternatif perlu dilakukan jika terjadi kegagalan sehingga proses masukan gudang data dilakukan tepat waktu. Penelitian ini membahas berbagai cara untuk mengetahui proses mana yang gagal dalam mengisi gudang data, namun tetap dapat dilanjutkan. Berbagai teknik memulai kembali (*resumption*) dibandingkan, dan teknik berbasis blok baru diusulkan untuk meningkatkan salah satu teknik yang sudah ada.

**Kata kunci:** gudang data; berbasis blok; *resumption*; kegagalan ETL.

**Abstract.** Data warehouses are loaded with data from sources such as operational data bases. Failure of loading process or failure of any of the processes such as extraction or transformation causes great financial losses because of the non-availability of data for analysis. With the advent of e-commerce and many real time applications, the analysis of data in real time becomes a norm. Hence, any misses when the data is being loaded into data warehouse need to be handled in an efficient and optimized way. The techniques to handle failure of process to populate the data are as much important as the actual loading process. An alternative arrangement needs to be made if failure happens so that processes of populating the data warehouse are done in time. This paper explores the various ways through which a failed process of populating the data warehouse could be resumed. Various resumption techniques are compared, and a novel block-based technique is proposed to improve one of the existing resumption techniques.

**Keywords:** data warehouse; block based; resumption; failed ETL.

## 1. Introduction

Extract Transform Load (ETL) is a process that helps integrate data from various disparate sources into a common data warehouse. This is performed using data integration tools, or because of the cost of integration tools, it sometimes is developed internally through the organizations development team [1]. Moving data from operational systems to the data warehouse was traditionally executed during off peak hour. Then, the advent of real time analysis has changed the ETL process. Now the ETL is

also moving towards real time process, with a minimal delay and it is no longer during off peak hours. This real time ETL[2] with a certain minimal delay is known as near real time ETL [3][4]. The process of near real time ETL is more challenging to complete within in a stipulated time [5][6]. While the process of ETL helps move the data from transactional system to an analytical system, there are several tasks which the ETL process needs to handle[7][8]. Each of the tasks has the potential to be optimized, such as identification of the correct data from the source that is relevant to the current cycle of ETL process. After identification process is the extraction step which is the most optimized way not to burden the operational process of the transactional database. The next step is to customize and integrate data into a common format. Before the final loading of the data, it is important to make sure that only useful and cleaned data is propagated and loaded into the data warehouse [9]**.**

Conceptually data warehouse design differs from transactional processing database system design. In a transactional processing design, the focus is to normalize and minimize redundancy, so entity relational model is used for conceptual modelling [10]. In a data warehouse the focus is for analysis, so redundancy is allowed. Moreover, the data is stored in the forms of dimensions and facts that are related to each other. Dimensions are the attributes that give the definition to the data, while facts record the measures of the data. Schemas are organized as star or snowflake design depending on the requirement to confine or elaborate the data.

What goes into the data warehouse without any hindrances shall be the part of decision making. At the end, everything boils down to decision making based on updated data. Hence, it can be said that business intelligence(BI) is the key beneficiary from the data warehouse setup [11][12]. BI helps in better and real time decision making by collection data into data warehouses for mining, analysis, reporting and visualizing, and others.

If any of the steps in the entire ETL functionality is not done completely or there is failure due to the unavailability of network, database management system, etc., there is a need for resumption of the entire process from the point of failure. The objective of this work is to compare various methods of performing the resumption of failed ETL process. The methods studied are repetition, redundant flow, and deployment of recovery point. Repetition is a simple technique to redo the entire ETL process. Redundant flow is to have parallel flow and to keep track of amount of data loaded. In case of failure, the parallel flow can take over from the point the ETL process stops. Deployment of recovery point is to store the intermediate results after the transformation step of ETL process in a storage area. In case of failure, one could reload the data from the intermediate storage without the need to repeat the transformation. After the comparative analysis identifies the best resumption technique, this study incorporates a novel block-based approach to further improve the resumption time for the deployment of recovery point based on resumption technique.

## 2. Theoretical Framework

This section studies the existing work on resumption techniques in case of failed ETL process. Identification of tuples that fail to load during a ETL process was proposed in [13]. Certain properties of input tuples were taken to know if a certain group of tuples failed or were successfully loaded. They propose an algorithm called as design-resume where the design phase identifies the failed tuples and resume phase loads those missing tuples. Design-resume algorithm does not burden the operational performance of the ETL process. To illustrate, when tuples from the source contribute to a set of tuples in the destination, the identification of contributing tuples helps filter them out from the resumption process. Design phase is taken as input to the graph G indicating the ETL process. In case of failed ETL scenario process, the tuples become in reverse topological order and construct a graph G' which would filter out all the processed tuples that have contributed to the warehouse. Resume phase acts as input to the graph G' and performs the normal resumption operation with only unprocessed tuples that remained to be loaded because of failure.

Improvement to the design-resume algorithm with multiple disparate sources was done by [14].The writers have identified a major drawback in design resume algorithm that is if the initial load completely fails without any rows being processed, then it is not possible to run the algorithm. In

addition, a save point mechanism to combine the data from various sources before transformation can be applied and the data can be loaded into the actual data warehouse.

Check point techniques to make sure the data is loaded until this point was combined with design-resume in [15]. Having a database management system technique of check point is an advantage during an ETL process because the data to the check point is processed and loaded successfully. Having check point puts a burden on the operational overhead of the ETL process. Hence, a combined solution of having a check point with additional design resume algorithm is a win-win situation where data loading is done with less over head during normal operation. In case of failure, the design and the resume can take up the resumption process.

The recovery of ETL jobs from an exception situation helps maintain the execution time. A multi agent system to do this job was developed in [16]. The idea is to have a log of the jobs being execution and divide the data into blocks with a check point. Whether a block of data was loaded successfully or not, check point is used to redo the loading of failed block. It is a multi-agent system that accomplishes the difficult task to keep track of failed loads to be fixed in an efficient manner.

Various scenarios of failed ETL process are taken into consideration to process the most suitable solution by considering the time window to complete the entire process [17]. Errors scenarios like fact table load, dimension table load, and fact extraction are considered. In the case of fact table load, it only affects the final warehouse table. In the case of dimension table load, failures stop the fact table load as well. The solution is to maintain a log of successful and unsuccessful loads of dimensions and facts. Moreover, the successful dimension fact table load is taken, or the other fact table load is skipped. Hence, the time window of redoing is saved.

When the data integration tools are not available and all the sources are being loaded to database management systems, Structured Query Language (SQL) supports various mechanisms to reload the failed processes. In [18] the author explained various ways through which the missing data could be loaded back into database system tables. SQL constructs the query to identify changes. Some of the constructions are NOT EXIST, MERGE and LEFT OUTER JOIN. The author compared each of the techniques for loading millions of rows that went missing and reported the best technique as insert using except clause (i.e., basically difference operator).

Pentaho is a data integration platform which helps integrate data from various sources like database management systems, file systems, etc. It can configure various transformations like filtering, joining, grouping and so on. The advantage of using data integration platforms is that any data source can be used in the ETL process. A solution for failed ETL loads in case of Pentaho data integration services is explained in the article [19]. The idea used in their solution is to have check points, control tables, and transactional transformations. Check point solutions adds a check point between two hops of the ETL flow. In case of failure, the hop after the check point is used as a starting point to restart the process. The disadvantage of check point is, in case of sub jobs, if the parent job fails, the entire suite of sub jobs needs to be repeated for a particular hop. In case of control table approach to a separate database, the table is maintained to keep track of the files that have been processed successfully. In case of failure, this table is referred to restart the process of ETL. Transactional transformation would roll back any changes made and start a fresh. This approach has the advantage to avoid duplicating processing.

Some other works on ETL in near real time known as stream ETL[20] [21] [22] explore the accuracy and the efficiency of the development of the data process in the fashion of streams. Another study is using grammar and machine learning to perform the analysis of incoming queries and predict the query processing time [23].

The above literature survey gave the writers motivation to explore resumption techniques further. In the process, the writers tried to identify possible solution for having resumption techniques that have fewer overheads. To consider, the percentage of failure examines the best strategy to perform resumption of a failed load. The main problem identified and tried to solve in this paper is to have recovery point for huge volumes of data. If the data volumes are very high, reaching a recovery point

costs additional storages and retrieval processing. However, the proposed approach of having block-based solution reduces the time for resumption of failed ETL load with recovery point as a technique.

## 3. Methodology

### *3.1. Resumption of failed ETL using repetition, recovery point and redundant flow*

The reliability of an ETL flow lies on how many failures can occur before a process is completed. In case of failure there could be three strategies to recover such as repetition, deployment of recovery points, and commissioning of multiple redundant flows [24]. Comparative analysis of techniques to perform resumption is discussed in this section. Three main strategies are considered, and experiments were repeated for each one with the same set of parameters. A brief introduction of the three strategies and possible challenges in them are discussed below. The implementation of these three strategies in order to get the most useful one and to improve is discussed in the next section.

#### *3.1.1. Repetition*

It is a simple strategy where the load is repeated to make sure the duplicate entries are avoided, and the integrity constraints are not violated. While doing repetition, the time-consuming part is to perform again all the transformations. This strategy could be used in simple transformations that are processing huge data so that the time to complete the transformation is less than the time to store the data redundantly. Simple transformations are those that do not involve joining data from multiple disparate sources and those that do not have aggregation operations like sum, min, max, and others. Huge data refers to data that could not be processed on a single machine and it requires a distributed cluster to perform reading operation[25][26].

#### *3.1.2. Recovery Point*

In this strategy after a complex transformation the data is loaded twice. One is in the recovery area and the other is in the actual data warehouse. While loading the data in the data warehouse, if there is a failure, then the data is recovered from the recovery area. This strategy avoids redoing the complex transformation, but it requires extra time in storing the data twice. This strategy could be used in complex transformation operation so that the time to execute the transformation is more than storing data redundantly. The complex transformation occurs in the data that joins operation from multiple disparate sources and has aggregate functions as mentioned in previous sub-section. Heuristics for adding recovery point was proposed in [24]. They identified three possible positions to place recovery points. The first is recovery point after every time-consuming transformation. The second is after an end of a phase of operation. The last is to have recovery point where it is feasible to have without additional overhead.

#### *3.1.3. Redundant Flow*

This strategy requires the load to have two or more flows. If a single flow fails, a failure over path could be taken. The extra efforts are required to keep track of the successful loaded parts from the unsuccessful load and to restart the failure from the point of previous load which fails. This strategy could be used in situation that required real time data with high reliability.

### *3.2. Methodology of Evaluating the Existing Techniques*

This comparative analysis experiments were conducted using the standard benchmark data set for ETL such as Transaction Processing Council (TPC). Under TPC various benchmarks are available to simulate ETL scenarios like TPC-DI, which DI stands for data integration, and TPC-DS, where DS is decision support. In this work TPC-DS is used because it contains merging content of two or more tables as the main transformation[27]. Besides that, any transformation with join is considered critical in an ETL process[28][29]. TPC-DS gives details of a model that represents retail sales. As shown in Figure 1, TPC-DS performs as several sources such as stores sales, web sales, and catalogue sales. In other words, every source performs ETL operations and populates several dimensions and facts.

In this experimentation, the writers have taken store sales as the fact table with dimensions of date, time, store, customer, promotion, and others. The data size was 1GB generated through DBGEN

program of TPC [27]. The initial data was already populated into the data warehouse tables. Incremental loads were populated using the data integration load and failure was simulated, so that experimentation could be performed. Store sales data warehouse is populated by merging data from dimension tables and fact tables. This helped simulate transformation which was very much required to access the efficiency of resumption techniques. As mentioned previously, repetition in resumption techniques requires redoing the transformation, whereas redundancy and recovery point-based resumption does not need to redo the transformations. Our results show that if transformation has to be repeated, then it takes more time (in this case more than 60%) to perform the resumption. However, the writers do not claim it is always the case because if the transformations are light weight (simple transformation as mentioned earlier) and the data volumes are huge [25], it is quite possible that other resumption techniques would take more time.

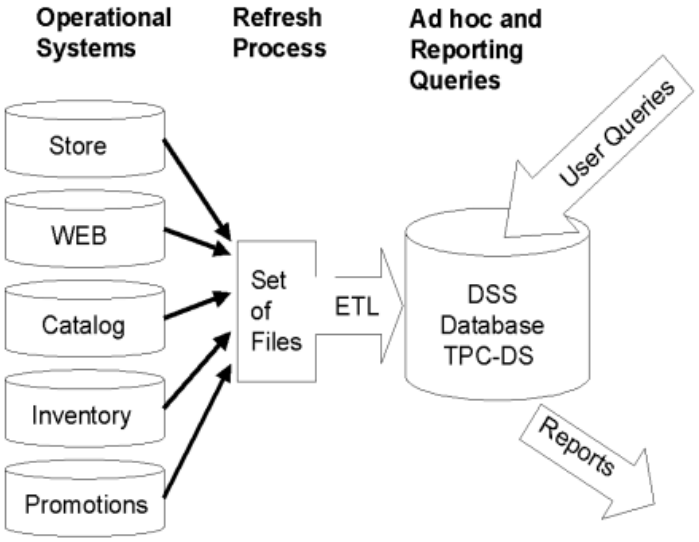


**Figure 1.** TPC-DS benchmark components.

Experimentation evaluation was done for loading 5lac, 10lac and 15lac rows (size of data in units of number of rows processed) with 10, 25, 50, 75 and 90 percentage failures in each case. Failure needs to be simulated to know how fast the resumption could be possible. In this regard, the writers simulated failure of ETL job after loading 10%, 25%, 50%, 75% and 90%. These percentages were used as a standard as mentioned din the work by authors of [13].The same percentage is used in this experiment as a reference to make sure all possible percentages of failures are covered. The hardware specification of the system used for the experiment was an Intel i5 2.5 GHz processor, with a memory on 4GB running on windows 10 operating system. The database management system used is PostgreSQL version 10.

*3.2.1. Results of evaluation of existing techniques*

Figure 2, Figure 3, and Figure 4 show the results of comparison of the three techniques. The graphs are plotted with percentage failure on the horizontal axis and time in milliseconds to perform the resumption on the vertical axis. Each of the failures starting from 10% to 90% was simulated for at least 5 times and the average time for resumption was recorded in the graphs shown. Taking the case of 5lac rows, 10% failure is the ETL process that stops after loading 50k row. So, this was simulated and all the three techniques (repetition, recovery point and redundant flow) were applied to record the time for resumption. Similarly, simulation was done for other percentage failures. Later the experiment was repeated by increasing the row size to 10lac and 15lac with the pattern of recording the resumption time remaining the same (5 times and taking average). As the result, a full proof system relays the resumption time for each of the three techniques.

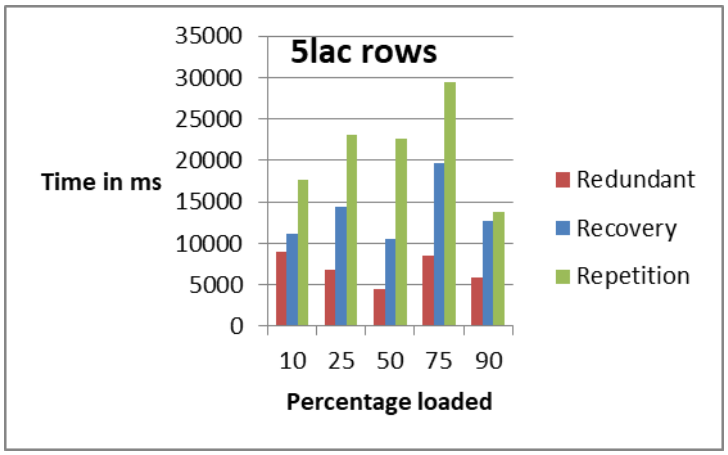


**Figure 2.** Comparison of 3 resumption techniques with data size of 5lac rows

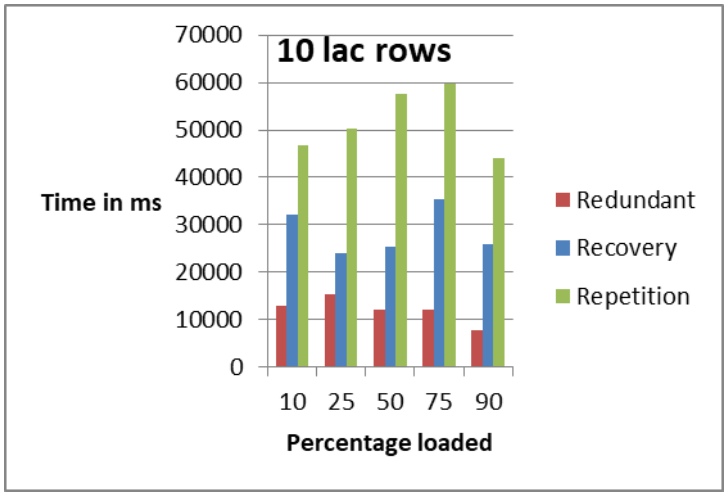


**Figure 3.** Comparison of 3 resumption techniques with data size of 10lac rows

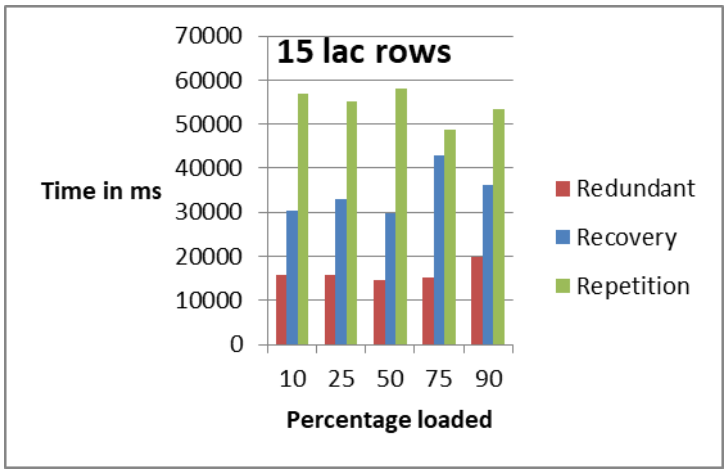


**Figure 4.** Comparison of 3 resumption techniques with data size of 15lac rows

*3.2.2. Analysis of the evaluation of existing technique*

The above experimentation results show the time taken for performing the resumption of failed ETL load using three techniques which are repetition, recovery point and redundant flow. Repetition techniques took more time to perform the resumption as shown in green color bar. Improvement to repetition technique was obtained by using recovery point techniques as shown in blue color bar. Finally, the redundant flow techniques took the least time (60% less) to perform the resumption as

shown in red color bar. However, doing redundant flow for every ETL setup was practically not feasible. This was an experimental setup that could easily achieve redundant flow. The implementation of redundant flow technique in real world is a challenging job because of the complexity involved in maintaining consistency amongst the various databases that hold a redundant copy. However, performing recovery point was a feasible solution because it is easier to maintain compared to redundant flow. The repetition was certainly the easiest, but it was time consuming. After the initial experiment, the writers wanted to improve the recovery techniques. This was because redundant flow technique was challenging, and repetition was time consuming. Hence, the writers took up a novel block-based approach to maintain the recovery points, as explained in the next section, and the approach reduced the resumption time. Recovery point-based technique was improved as shown in the results of the next and subsequent section of this work.

### *3.3. Methodology to create blocks of data*

Block based solution makes the available data into blocks. Data blocks are the primary unit to consider for the success or failure of the process. For example, it is possible that certain blocks are loaded successfully, and the other blocks are not. In prioritizing, the blocks are the basic unit so certain blocks can have a higher priority than certain other blocks. Replication is done on block level as well. During the replication operation, some blocks are replicated depending on the available time window.

The idea of making the data into blocks was taken from the blog in [30]**.** The creator of the blog shared the code to make the blocks of data and the writers used the same code. The enhancement of this research was the data combined from various dimensions tables and it made the data available to be loaded into store sales. Hence, the blocks-based database was extended to data joined with more than one table. Figure 5 and Figure 6 below show the screen shots of the data and how the same data looks with blocks in postgreSQL admin (PGADMIN). Things to notice in Figure 5 are the number of rows returned is 15lacs because of space constraint that could not show the exact rows. Consequently, only the message displayed by the query tool is shown. In Figure 6 a glimpse of the data has been shown, where blocksize1 column shows the size of each block and the next two columns shows the starting and ending value of row numbers. For 15lac rows with blocks size of 50k, there are 30 blocks created. Apart from this, the writers did the experiment with 5lac, 10lac rows that make the data of blocks 10k, 25k and 50k. The code snippet showed how the blocks of data were made with the available database table. The parameter value in this case was taken as 10000 which determines the block size. After the blocks were made, they were stored into the database as tables. The code in Figure 7 shows the creation of 5lac rows making block size of 10k. Similarly, by changing the parameter from 10k to 25k and 50k, other block size tables were created. The code was taken from [30] but the writers created a permanent table to store the output of blocks creation code.

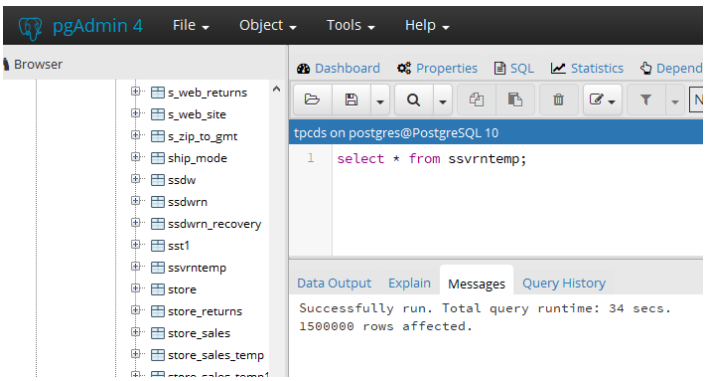


**Figure 5.** Screen shot of PGADMIN tool showing table with 15lac rows

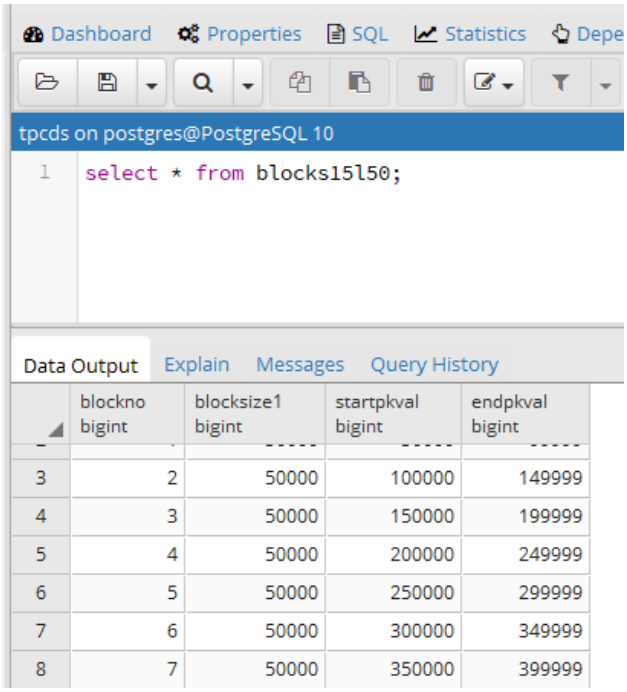


**Figure 6.** Screen shot of PGADMIN tool showing blocks of 50k for 15lac rows

```
CREATE TABLE  blocks5l10 as (
WITH myconstants (blocksize) as (
values (10000)
)
,
RowNumbers AS (
SELECT ROW_NUMBER() OVER (ORDER BY rownbr ASC) AS RowNo,
        rownbr
FROM ssvrntemp
),
Blocks AS (
SELECT   RowNo / @blocksize AS BlockNo
    ,COUNT(*) AS BlockSize1
                ,MIN(rownbr) AS StartPkVal
                ,MAX(rownbr) AS EndPkVal
                FROM RowNumbers,myconstants
GROUP BY RowNo / @blocksize
)

SELECT *
FROM Blocks
ORDER BY BlockNo);
```

**Figure 7.** Code snippet for creation of blocks

Making the smallest data unit like blocks has advantages to reduce the number of different operations, ease to track the amount of data loaded, ease to track available data at source, and replicate the repository and data warehouse. For example, if there are 1000 rows to be loaded and 500 rows are loaded successfully, the remaining 500 are supposed to be 400rows in the source and 100 in replication repository. Identification of failed rows at the source requires performing SQL DIFFERENCE or OUT JOIN operation among 500 rows successfully loaded at the data warehouse with the 1000 rows present at the source. Similarly, to know the 100 rows in the repository, the different operation or outer joining operation needs to be performed between the initial source data of

1000 rows and 100 rows data in the replication repository. If the same data is made into blocks and each block is the basic unit of execution, suppose 1000 rows are divided into blocks of 100 each then there are 10 blocks. So, with the preceding example 5 blocks are loaded successfully, 4 are at the source and 1 is in the replication repository. The task of identifying failed data that could be loaded is being now reduced to a smaller number of comparisons because of the block of data being made. To know how many rows have been successfully loaded, it only requires comparing the blocks available at source and destination. In case of 5 blocks presented at source, if 4 are loaded successfully and only 1 block is not loaded, then it is easy to identify that 1 block of data which failed to load. If this block is not made and there are 1000 rows at source and only 500 loaded successfully, then it becomes tedious to identify 500 rows at source which failed to load. Hence, it can be said that this block-based approach has reduced the time to identify data that failed to load by about 10 times.

### *3.4. Methodology of Block Based Recovery Point*

We repeated the recovery techniques with a novel block-based approach, where data are grouped into blocks and any missing data is to be found in one of the blocks. By keeping track of blocks of data for recovery, it is natural that number of comparisons is reduced. So, it can be said that the methodology recovery point technique has improved. The data is made into blocks using a range of primary key values. Each data set of 5lac rows is made into blocks of various size of about 10k 25k and 50k block size. That means 5lac rows would result in 50 blocks if made into blocks of size 10k each. Similarly, if made into blocks of size 25k each, 5lac rows would result in 20 blocks and 50k block size results in 10 blocks. As the size of block increases, the number of blocks for the data reduces.

## 4. Implementation and Results

Initially blocks of 10k were created for data of size 5lac. As shown in Figure 2, Figure 3, and Figure 4 75%, the loading takes the maximum time to recover in all the three data sizes of 5lac, 10lac and 15lac rows. Simulation of 75% failure was carried out and the resumption techniques using recovery method was deployed. Later the experiment was repeated with block size of 25k and 50k for same data size of 5lac rows with 75% failure only. To know the effectiveness of the method, the data size was increased to 10lac rows and 15lac rows keeping the block size and percentage failure common across all the experiments. Each experiment was repeated five times and the average time for resumption was recorded. As shown in Figure 8, Figure 9, and Figure 10, block-based approach reduced the resumption time (approximately by an average of 10%) as compared to normal recovery based method. Normal recovery time was taken from comparative section where all the three techniques were compared viz. recovery point, repetition, and redundant flow.

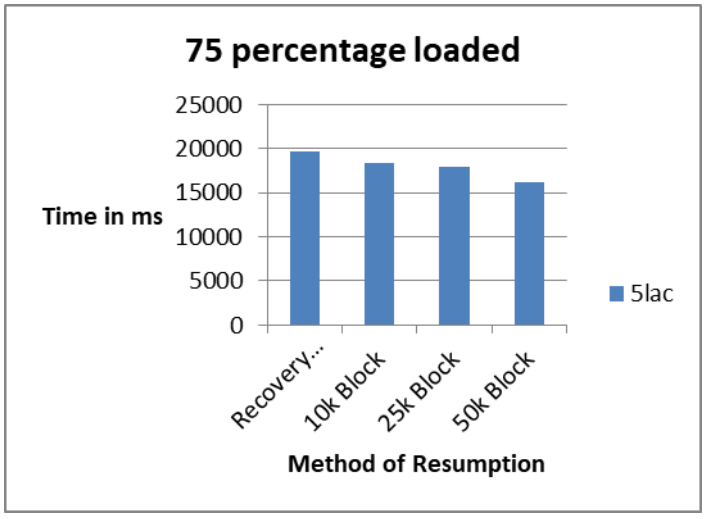


**Figure 8.** Blocked based recovery data size of 5lac rows

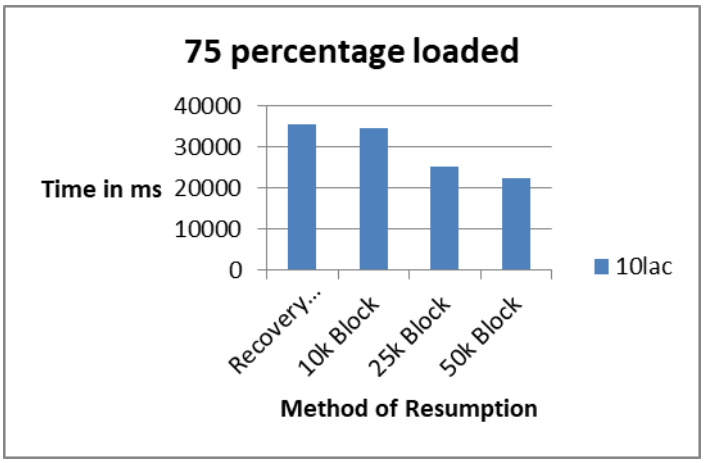


**Figure 9.** Blocked based recovery data size of 10lac rows

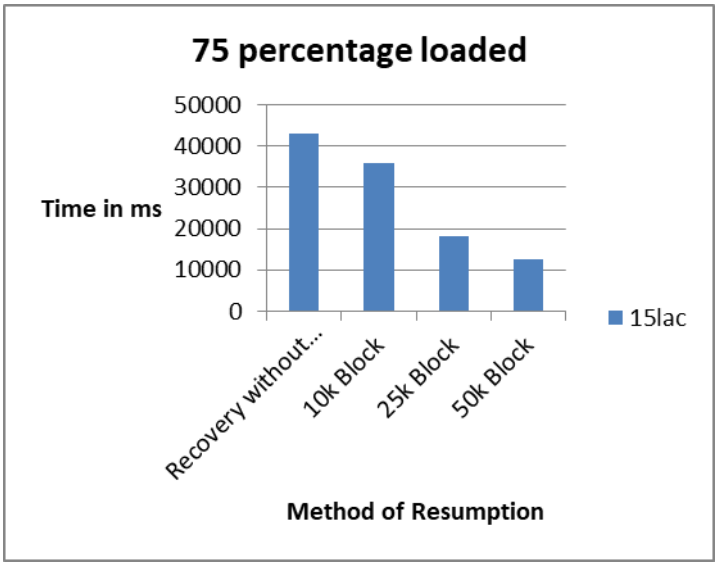


**Figure 10.** Blocked based recovery data size of 15lac rows

The results lead to the conclusion that as the number of blocks reduces, by increasing the row in each block the, resumption time also reduces. This is a natural scenario because when the number of blocks reduces, the time to compare and to note the failed and successful blocks also reduces. If the blocks are not made, then every data row needs to be checked for failed load or successful load.

## 5. Conclusion and Future Enhancements

Near real time ETL has many aspects that should be achieved before the data could be loaded into the data warehouse. One of the aspects such as the resumption of failed ETL process was studied in this paper. The writers have compared existing techniques for resumption for pointing out which is the best technique for the scenario and data set available as benchmark (TPC-DS) used in this work. The writers found that redundant flow was the best performing solution for resumption of failed ETL process as compared to repetition and deployment of recovery point. However, this result was specific to the case where transformation process took less time as compared to storing and retrieving data from the recovery point. Here it is assumed that the transformation process in case of ETL of store sales fact table of TPC-DS benchmark took less time and redundant flow outperformed recovery point of deployment technique. Citing this as the reason, this work has improved the recovery point of deployment-based resumption technique using a novel block-based approach. The result of the novel block-based approach was found to be encouraging (approximately 10% average reduction), which was tested on 75% failure case for data set of 5lac, 10lac and 15lac. This can be concluded that,

transformation step of ETL with light weight and huge recovery point of deployment-based approach would improve the time for resumption.

In the future the writers would like to study and compare the deployment of recovery point in a distributed environment where the data flow is partitioned. If the partition is unsuccessful, the study would discuss the strategies that can be used to get data from a particular portioned. Furthermore, how optimal a distributed recovery point for ever partition that should go with a redundant flow could be a possible future study for this work.

**References**

[1] P. Vassiliadis, A. Simitsis, and S. Skiadopoulos, "Conceptual modeling for ETL processes," *ACM Int. Work. Data Warehous. Ol.*, pp. 14–21, 2002, doi: 10.1145/583890.583893.

[2] N. Biswas, A. Sarkar, and K. C. Mondal, "Efficient incremental loading in ETL processing for real-time data integration," *Innov. Syst. Softw. Eng.*, vol. 16, no. 1, pp. 53–61, 2020, doi: 10.1007/s11334-019-00344-4.

[3] P. Vassiliadis and A. Simitsis, *Near Real Time ETL*, vol. 3. 2009.

[4] K. Kakish and T. a Kraft, "ETL Evolution for Real-Time Data Warehousing," *Proc. Conf. Inf. Syst. Appl. Res.*, pp. 1–12, 2012.

[5] S. Gorhe, "ETL in Near-Real Time Environment : Challenges and Opportunities," no. April, 2020.

[6] A. Sabtu *et al.*, "The challenges of Extract, Transform and Loading (ETL) system implementation for near real-time environment," *Int. Conf. Res. Innov. Inf. Syst. ICRIIS*, pp. 3–7, 2017, doi: 10.1109/ICRIIS.2017.8002467.

[7] and S. D. Weiping Qu, Vinanthi Basavaraj, Sahana Shankar, "Real-Time Snapshot Maintenance with Incremental ETL Pipelines in Data Warehouses," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 9263, pp. 28–39, 2015, doi: 10.1007/978-3-319-22729-0.

[8] W. Qu, "Incremental ETL Pipeline Scheduling for Near Real-Time Data Warehouses 1," no. Btw, pp. 299–308, 2017.

[9] A. Simitsis, P. Vassiliadis, and T. Sellis, "Optimizing ETL processes in data warehouses," *Proc. - Int. Conf. Data Eng.*, no. June 2014, pp. 564–575, 2005, doi: 10.1109/ICDE.2005.103.

[10] R. E. S. B. Navathe, *Database Systems*. 2016.

[11] U. Dayal, M. Castellanos, A. Simitsis, and K. Wilkinson, "Data integration flows for Business Intelligence," *Proc. 12th Int. Conf. Extending Database Technol. Adv. Database Technol. EDBT'09*, pp. 1–11, 2009, doi: 10.1145/1516360.1516362.

[12] M. Minhaj, "An Exploratory Study of Near-Real Time ETL Approaches for the Design of Agile Business Intelligence Infrastructure Mohamed Minhaj," *SDM Res. Cent. Manag. Stud.*, vol. V, pp. 23–44, 2016.

[13] W. J. Labio, J. L. Wiener, H. Garcia-Molina, and V. Gorelik, "Efficient resumption of interrupted warehouse loads," pp. 46–57, 2000, doi: 10.1145/342009.335379.

[14] M. Gorawski and P. Marks, "High efficiency of hybrid resumption in distributed data warehouses," *Proc. - Int. Work. Database Expert Syst. Appl. DEXA*, vol. 2006, pp. 323–327, 2005, doi: 10.1109/DEXA.2005.108.

[15] M. Gorawski and P. Marks, "Checkpoint-based resumption in data warehouses," *IFIP Int. Fed. Inf. Process.*, vol. 227, pp. 313–323, 2006, doi: 10.1007/978-0-387-39388-9_30.

[16] J. Huang and C. Guo, "An MAS-based and fault-tolerant distributed ETL workflow engine," *Proc. 2012 IEEE 16th Int. Conf. Comput. Support. Coop. Work Des. CSCWD 2012*, pp. 54–58, 2012, doi: 10.1109/CSCWD.2012.6221797.

[17] S. Tu and L. Zhu, "An optimized etl fault-tolerant algorithm in data warehouses," *2013 IEEE 3rd Int. Conf. Inf. Sci. Technol. ICIST 2013*, pp. 484–487, 2013, doi: 10.1109/ICIST.2013.6747594.

[18] D. Lozinski, "Fastest-way-to-insert-new-records-where-one-doesnt-already-exist," *The curious consultant*, 2015. .

[19] C. Morehouse, "Restratability in PDI," *Hitachi*, 2019. .

[20] M. Gorawski and A. Gorawska, "Research on the Stream ETL Process," *Commun. Comput.*

*Inf. Sci.*, vol. 424, no. April, pp. 61–71, 2014, doi: 10.1007/978-3-319-06932-6_7.

[21] G. V. Machado, Í. Cunha, A. C. M. Pereira, and L. B. Oliveira, "DOD-ETL: Distributed on-demand ETL for near real-time business intelligence," *J. Internet Serv. Appl.*, pp. 1–15, 2019, doi: https://doi.org/10.1186/s13174-019-0121-z.

[22] T. H. R. Munige, "NEAR REAL-TIME PROCESSING OF VOLUMINOUS, HIGH-VELOCITY DATA STREAMS FOR CONTINUOUS SENSING ENVIRONMENTS," Colorado State University, 2020.

[23] M. Gorawski, M. Gorawski, and S. Dyduch, "Use of grammars and machine learning in ETL systems that control load balancing process," *Proc. - 2013 IEEE Int. Conf. High Perform. Comput. Commun. HPCC 2013 2013 IEEE Int. Conf. Embed. Ubiquitous Comput. EUC 2013*, pp. 1709–1714, 2014, doi: 10.1109/HPCC.and.EUC.2013.243.

[24] A. Simitsis, K. Wilkinson, U. Dayal, and M. Castellanos, "Optimizing ETL Workflows for Fault-Tolerance," 2010.

[25] D. Eadline, *Hadoop 2 quick start*. 2016.

[26] A. Awadallah and D. Graham, "Hadoop and the Data Warehouse | When to Use Which," 2012.

[27] "Transaction Processing Council," 2020. [Online]. Available: http://www.tpc.org/tpc_documents_current_versions/pdf/tpc-ds_v2.13.0.pdf.

[28] S. H. A. El-Sappagh, A. M. A. Hendawi, and A. H. El Bastawissy, "A proposed model for data warehouse ETL processes," *J. King Saud Univ. - Comput. Inf. Sci.*, 2011, doi: 10.1016/j.jksuci.2011.05.005.

[29] Stitchdata, "ETL Transforms," *Talend*, 2019. .

[30] J. VANLIGHTLY, "Building-synkronizr-a-sql-server-data-synchronizer-tool-part-1," *RabbitMQ*, 2016. [Online]. Available: https://jack-vanlightly.com/blog/2016/11/12/building-synkronizr-a-sql-server-data-synchronizer-tool-part-1.