University of Stavanger

**Faculty of Science and Technology**

# MASTER'S THESIS

| | |
|---|---|
| **Study program/Specialization:**<br><br>Master Degree Program in Computer Science. | Spring semester, 2012<br><br>Open / Restricted access |
| **Writer:**<br>Sanjeev Khatiwada | ……………………………………………<br>(Writer's signature) |
| **Faculty supervisor:** Professor Chunming Rong, (UiS)<br><br>**External supervisor(s):** Dr. Demissie Bediye Aredo, (Bouvet ASA) | |
| **Titel of thesis:**<br><br>Architectural Issues in Real-time Business Intelligence | |
| **Credits (ECTS):** 30 | |
| **Key words:**<br><br>Business Intelligence (BI), Real-time BI, Architecture | **Pages:** 89<br><br>+ **enclosure:** CD<br><br>**Stavanger**, 15/06/2012 |

# Architectural Issues in Real-time Business Intelligence

Sanjeev Khatiwada
Department of Electrical and Computer Engineering
University of Stavanger
E-mail: s.khatiwada@stud.uis.no

Thesis submitted in partial fulfillment of the
Requirements for MASTER DEGREE
In Computer Science

June 14, 2012

# Contents

# List of Figures

# List of Tables

# Acronyms

| | |
|---|---|
| **API** | Application Programming Interface |
| **BI** | Business Intelligence |
| **BIT** | Business Intelligence Tools |
| **BPM** | Business Performance Management |
| **CEP** | Complex Event Processing |
| **CQL** | Continuous Query Language |
| **DB** | Database |
| **DBA** | Data mart bus architecture |
| **DIM** | Dimensional modeling |
| **DM** | Data Mining |
| **DS** | Data Sources |
| **DSMS** | Data Stream Management System |
| **DW** | Data Warehouse |
| **EAI** | Enterprise Application Integration |
| **EPL** | Event Processing Language |
| **ETL** | Extract Transform and Load |
| **FED** | Federated architecture |
| **GFS** | Google File System |
| **HAS** | Hub and Spoke Architecture |
| **HDFS** | Hadoops Distributed File System |
| **IDM** | Independent data marts |

| | |
|---|---|
| **JKM** | Journalizing Knowledge Module |
| **KPI** | Key Performance Indicator |
| **LINQ** | .Net Language Integrated Query |
| **MBCFRTR** | Memory Based Component for Real-time Reporting |
| **OCEP** | Oracle Complex Event Processing |
| **ODI** | Oracle Data Integrator |
| **ODS** | Operational Data Store |
| **OLAP** | OnLine Analytical Processing |
| **OS** | Operational Sources |
| **POJO** | Plain Old Java Object |
| **PQL** | Physical Query Plan |
| **RDBMS** | Relational Database Management System |
| **Redis** | Remote Dictionary Server |
| **RTBI** | Real-time Business Intelligence |
| **RTI** | Right Time Integrator |
| **RT** | Real Time |
| **SSDT** | SQL Server Data Tools |

## Abstract

Business organizations are always in need of fast and intelligent decision support system. Today's Business environment is dynamic so data every minute are valuable. Real time Business intelligence (RTBI) is intelligence in business system which can make decision with minimum data latency from the time it is created to the time it is presented. The integration layer in Business Intelligence(BI) that extracts, transforms and loads(ETL) data in to data warehouse(DW) is the main component that adds data latency in BI. A new architecture to support RTBI along with the existing functionality is suggested in this thesis work. A prototype is made and implemented. Through the implemented prototype, tests are done to measure the performance of the architecture. A memory based processing component which is implemented in the architecture has better performance on report generation for critical real time data.

## Acknowledgements

I would like to express my sincere gratitude to my supervisors Professor Chunming Rong and Dr. Demissie Bediye Aredo for their continuous support and motivation that has encouraged me to propose and complete with this thesis work. I would also like to thank to Dr. Son Thanh Nguyen and Dr. Tomasz Wiktor Wlodarczyk for their valuable suggestions and for organizing regular weekly meetings. I would also like to acknowledge Joar Ulvers$\phi$y, who works in Bouvet, for demonstrating how DW and ETL are managed in the existing architecture that is used in BI projects he is working on.

Finally, I would like to thank all my friends and family for their support during the thesis work.

# Chapter 1

# Introduction

Business Intelligence (BI) is the intelligence in business aided by software programs which help in decision support system. The main purpose of this added intelligence is for fast analysis and action. "It is a business management term used to describe applications and technologies which are used to gather, provide access to and analyze data and information about the organization, to help in making better business decisions" [1]. BI has mainly components for gathering data from data source, extracting information from data and converting them to knowledge as shown in the Fig. 1.1 [2].

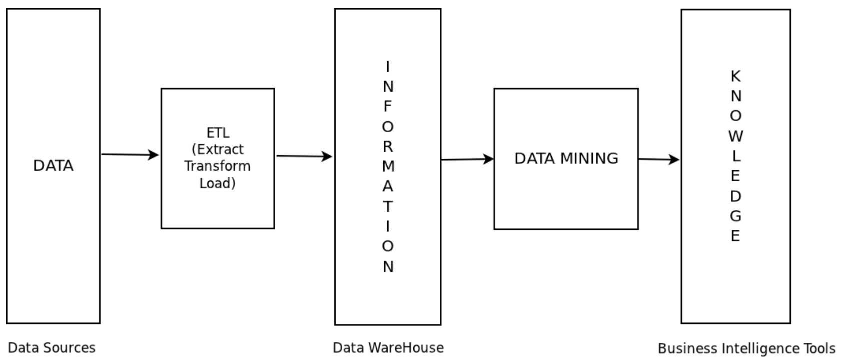


Figure 1.1: Components of BI

The first stage would be to prepare information from the raw data and then through data mining required knowledge is obtained. Raw data are extracted and transformed to the desired format through ETL functions and stored as information in DW. Knowledge can be in the form of reports, charts and graphs that help business analysts in making decision.

## 1.1 Motivation and Problem Description

BI should cope with the need of enterprises to deal with the dynamic and continuously changing business environments. In the real-world settings, business conditions and environments are in constant state of flux: sales patterns change from place-to-place and from time-to-time; currency valuations shift and affect profit margins; suppliers change delivery schedules and their prices; and customers become more aware about business conditions and therefore become more demanding. Hence there is a need for a BI system that is adaptive to accommodate the dynamics of the business environments.

Extracting, transforming and loading information from the ever increasing magnitude of source data also adds to the complexity of BI architecture and tools. A BI needs to access data from a variety of sources, transforms data into information and knowledge using sophisticated analytical and statistical tools, and provide a graphical interface to present the results in a user friendly way. A complete BI system integrates the following categories of technology: data warehousing to archive historical data; analytical/data mining tools; and tools for generating reports and dashboards. A next generation BI should enable users, for instance, to decide dynamically the source of data from which information is extracted based on their needs and other factors such as spatial and temporal parameters. Traditional BIs are built for archiving and retrieving static historical data and are not adequate to handle the dynamics of contemporary enterprises. There are many business sectors that require business analysis reports in real-time, possibly, with near zero latency.

Advances in the information technologies make real-time BI seemingly achievable: Internet has revolutionized information sharing; large quantity of data is available; almost any company's data sources can be made accessible over an intranet; capturing all sorts of data and storing them is effective and cheap; and a process can provide information whenever it is required by the management; RTBIs have the ability to derive key performance measures that relate to the situation at the current point in time and not just some historic situation. The future of BI lies in the development of systems that can autonomously and continuously improve decision-making process within the changing business environment in real time.

The thesis work is done in association with Bouvet ASA. Bouvet is a leading Norwegian provider of consultancy and system development services within information technology sector. Business analysis is one of the major areas of its focus in overall market strategic plan. In order to provide these consultancy services in business analysis It is working towards improvement of the business intelligence technology and supporting tools. Bouvet provide BI solutions in three platforms: Oracle, Microsoft and SAP and focuses on this research that will contribute to the improvement of the BI approaches and come up with suitable architecture which they can implement for its ongoing BI systems to support RTBI. The work done in this thesis project will have a significant contribution to the improvement of the services in BI and the area of business analysis.

The following were major results that are expected from this thesis work:

1. To propose a BI architecture that addresses the limitations of contemporary real-time BI systems, and performance issues posed by accessing to real-time enterprise data.

2. To develop a prototype of the proposed architecture in order to evaluate its performance.

3. When information is extracted from several distributed and heterogeneous data sources, some kind of data integration is necessary. Proposal of how such an integration should be performed, and discussion of the impact of data pre-processing on the performance of the proposed architecture.

## 1.2 Background

### 1.2.1 Components of Business Intelligence

The main components of BI are shown in Fig. 1.2 [3] and described below:

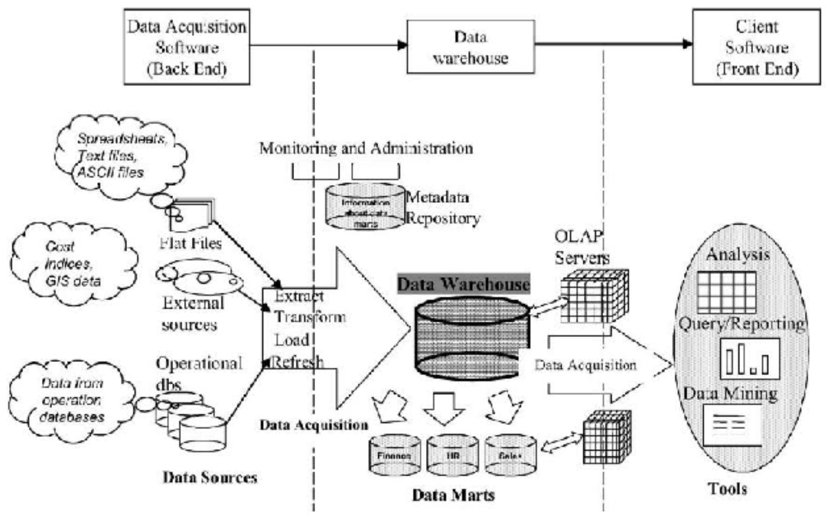


Figure 1.2: Components of BI

- *Data Sources (DS)* are systems that provide data. Generally, the DS are operational data stores, which can Relational databases, spreadsheets and query tools. The operational DS are kept on DW in regular intervals: monthly, daily, hourly, etc.

- *ETL* refers to three separate functions. The *Extract* function extracts desired subset of data from DS. The *Transform* function is used to transform acquired data into a desired state, using rules or lookup tables, or creating combinations with other data. Finally, the *Load* function is used to write the resulting data to a target database. The most time consuming of the ETL process is the transform function, especially when the source databases are heterogeneous and distributed/decentralized. Inconsistent codes, handling of incomplete data and changing codes to meaningful terms are all part of the transform process[4]. Popular solutions for ETL include: Informatica Power Center, IBM websphere

Data Stage, Oracle Data Integrator, Ab Initio and Microsoft Integration Services (a component of SQL server)[4].

- *Data Warehouse (DW)* are extracted from various sources and transformed into a single consistent type and loaded for analysis.
- *Data Mining (DM)* is the analysis step of knowledge discovery by discovering new patterns from large data sets and involves statistical and artificial intelligence methods.
- *Business Intelligence Tools (BIT)* are software for presenting the business intelligence as reports from data warehouses.

### 1.2.2 Latency problem in BI

Latency is the time taken from the event that is executed to the action taken in response to that event. It is the major issue in BI architecture's today. Let me explain about latency and its types in the following section.

BI is basically used to analyse the performance of business. At times, information is so critical that there is often frustrations in business users stating 'information arrives just too late to be really useful'. In many cases real insight is more important than large reports. Too much information can lack real insight and there can be dificiency of time to make sense of it at all.

Let us consider the scenario in value-time curve to illustrate the relationship between time and business value through Fig. 1.3[5] and Fig. 1.4 [6] [7]

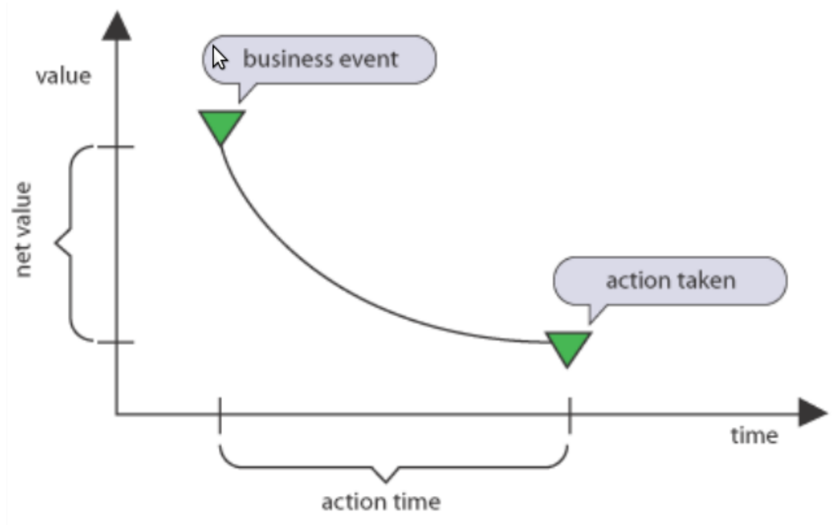


Figure 1.3: The Value-Time Curve

When any business event occurs, action is taken to respond to that event. For instance, when a customer asks for an information on a specific product, company provides information in response to his query within certain interval of time. The assumptions on this decay curve Fig 1.4 is that the longer the delay or latency to provide information, the less value accrues to the company.

In this example,

- Event is customers query for an information.
- Action is company provides information.
- Action time is the duration between the event and the action. In this case the duration between customer's query and company's response.
- And the net value is the business value lost or gained over this duration.

In the context of DW, there are three latencies associated with this action time as shown in Fig. 1.4 [6].

1. Data latency: It is the time required to capture data from the time it is created to the time it reached to DW for analysis.

2. Analysis latency: It is the time required to analyze and disseminate the results to the appropriate persons.
3. Decision latency: It is the time required for a person to understand the situation and take decision based on the analysis.

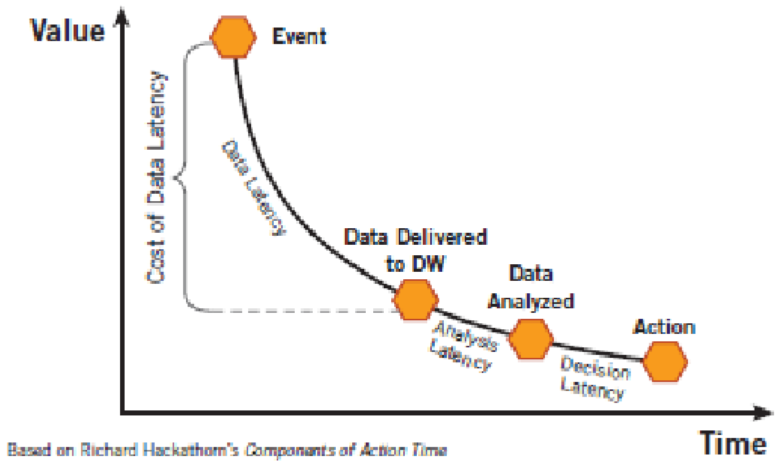


Figure 1.4: Latency Problem in BI

If we closely analyze the graph, the vertical portion that takes much of the value is data latency so the most critical part in BI is data latency. So, architecture of real time reporting should address minimization of the data latency.

### 1.2.3 Data Latency

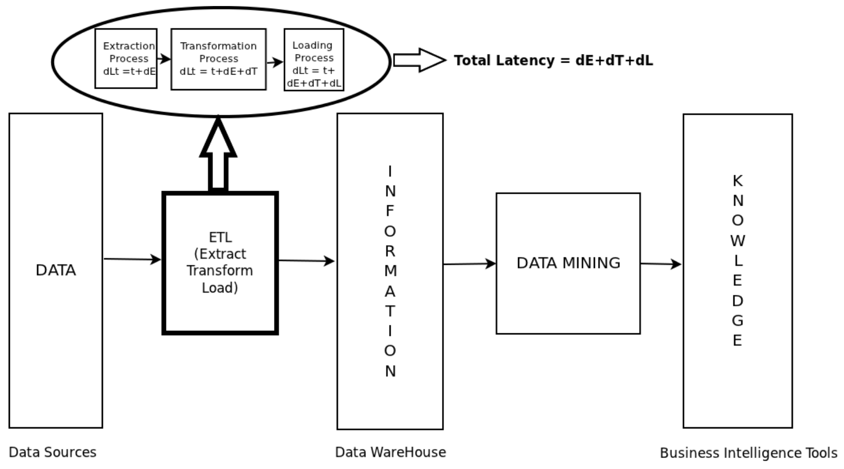

Extraction Process dLt =t+dE
Transformation Process dLt = t+dE+dT
Loading Process dLt = t+ dE+dT+dL
Total Latency = dE+dT+dL
DATA
ETL (Extract Transform Load)
INFORMATION
DATA MINING
KNOWLEDGE
Data Sources
Data WareHouse
Business Intelligence Tools

Figure 1.5: Data Latency Problem in BI

The latency in the report generated from DWs are caused due to the latency time that has incurred when transforming data from OS to DW. As shown in the Fig. 1.5, there are different processes in ETL block which adds latencies when data reach the DW.

If we express this mathematically, dE as the latency caused by extraction function, dT as the latency caused by transformation function and dL as the latency caused by loading function than the total latency caused through ETL process is dE+dT+dL.

Now there can be question that if there is so much latency caused by ETL, why we need the ETL and why is it so important in BI. Can't there be some sort of integration system and remove this layer.

#### Why we need ETL if it is causing latency?

The "main functionalities of ETL layers" can be summarized in the following prominent tasks [8]:

1. *Identification of relevant information at the source side.*

2. *Extraction of the information.*

3. *Customization and integration of the information into a common formats.*

4. *Cleaning of the resulting data sets based on database and business rules.*

5. *Propagation of data into data marts and/or data warehouse.*

Generally data is scattered in different sources like Relational database management system (RDBMS), Flat files, mainframes, CSV and different heterogeneous sources. Before they are loaded to DW for reporting, they need to be cleansed and made available in common formats as well as to make it efficient for report generation through DM. To facilitate this, we need ETL. ETL flow is described in Fig. 1.6 from [9]

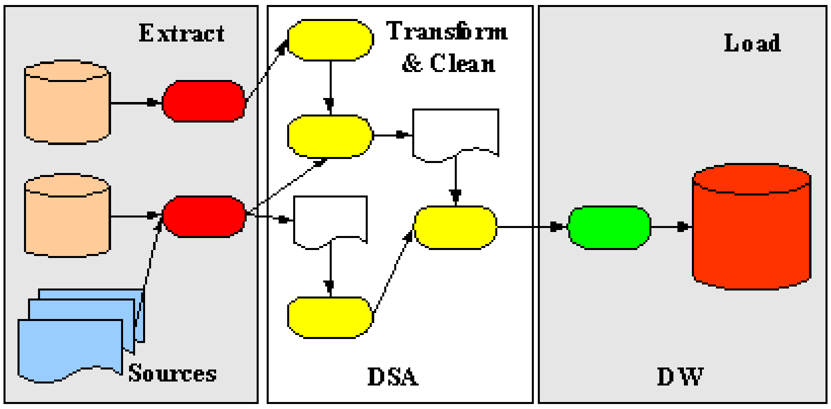


Figure 1.6: ETL Processes

### Why we need OS and DW

Those who are new to BI may want to skip DW and deploy BI tools directly against the operational system, which seems to be faster approach. But, the need for DW is felt because of its functionalities that are important and different from the functionalities of OS [4].

1. OS and DW have their own purposes where OS are used for processing the transactions in real time and DW are used just for reporting historical data. Hence the level of details that they contain also vary.

2. OS has real time data but DW has information extracted in periodic intervals and has only read operation for analysis of data, whereas OS can process the *Read, Write, Delete* and *Update* operations depending on the requirements.

3. Since the table structure of OS and DW are different, their response times also differ. For the inputs, OS is faster whereas queries are faster in DW since it is tuned for fast queries. OS are generally in normal form, whereas only parts of DW can be in normal form. But business users' queries are normally de-normalized and stored in snowflake schema and contain less tables than OS.

### 1.2.4 Analysis latency

Even though Analysis latency is less critical as shown in Fig 1.4 [6], it is equally important in some cases in connection with real time reporting. When data is huge, it lacks real insights and at this time we need analysis. Analysis of data in BI is the process of determining how comfortably and efficiently data can be mined from the data storage. Analysis latency becomes more important when the business problems are complex and there is a need for analysis of the right action on it.

#### Complex business problems and need for analysis

There is always a large contrast between having the right knowledge and making the right decision. Knowledge doesn't guarantee the right decision. Modern business firms are always in need of decision support systems for dynamic and ever-changing environments to make far-reaching decisions. Managers always need to deal with predicting what is going to happen in the future and what can be the best decision right now based on past knowledge. On the other hand, business sectors like stock exchange require immediate response on the changes that occurred in real data. Only up-to-a-minute data can be of great importance in this case [2].

### 1.2.5 RTBI

BIT always analyze information from DW, where information is kept and updated in regular intervals. Now there comes a question: What if we need data in real time? Real time data means ongoing data. Ongoing data are obtained from operational data sources, whereas DW is a collection of data extracted from different operational systems and transformed and optimized for data consistency and analysis.

RTBI is an approach in which up-to-a-minute data is analyzed, either directly from OS or feeding business transactions into a real time DW and BI system. RTBI analyzes real time data. As defined in [10], "*Real time*" mean:

1. The requirement that a process to obtain near to zero latency.
2. The requirement that a process to have access to information whenever it is required.
3. The requirement that a process provides information whenever it is required by management.
4. The ability to derive key performance measures that relate to the situation at the current point in time and not just to some historical situation.

In modern competitive businesses, up-to-a-minute analysis of data is the price paid to realize the opportunities before the competitors do that, which on the long run generate more business with higher sales and profit. The importance of up-to-a-minute data analysis can be viewed on fraud detection in credit card usage, stock exchange market, call center to provide best offer or action as per sales and stock, analyze web page usage as per click, page views, link views etc. And there is always a need for analysis based on this real time data.

For RTBI, the most important component is DW, ETL layer and some in the analysis phase through mining algorithm. In the literature review, I will discuss different existing architectures of DW, existing solutions to address

ETL layers and reduce the time frame, some existing solutions to address RTBI and some work on the analysis reviews.

### 1.2.6 Thesis Outline

The following chapter constitute the thesis:

- **Chapter1**: introduces RTBI and the cause of data latency. The scope and motivation of the thesis is summarized.
- **Chapter2**: gives the background knowledge on DW, study of related work on RTBI and study of components of BI in connection to RTBI.
- **Chapter3**: describes the technologies used in implementing suggested architecture for RTBI, analyses the important components from the solutions available for RTBI, describes the suggested architecture, its flow, data model, algorithms, and implementation.
- **Chapter4**: gives result and analysis on the performance of the suggested architecture and comparison with the existing BI.
- **Chapter5**: summarizes the major contributions and conclusions of this work, and suggests the problems for further research.

# Chapter 2

# Literature Reviews

The preliminary literature review of the thesis was done on "Computer Science Project" in autumn semester and some of the content are taken from that report.

To address the problem of RTBI, since all reports are based on DWs, study of the process of building DW and different architectures of DW are done to find out the architectural issues of DW. For this, few prominent DW and the architecture of the popular industrial BI system were also studied. Furthermore, NoSQL databases that supports RTBI are studied.

## 2.1 Data Warehouse

Over periods, many companies consider DW as a basic foundation of Decision Support System (DSS) and are critical enablers for BI. Despite the recognition of DW importance, relatively few studies have been conducted to assess data warehousing practices in general and critical success factors in particular [11].

### 2.1.1 Process of building Data Warehouse

There are many considerations in DW design that is different from OS database design as the purpose of DW is different from OS database. Data must be organized in DW as it is used for rapid access to information for analysis and reporting. The correctness of data from DS are very important before it goes to DW since DW are used for decision making and incorrect data can lead to wrong conclusions. For example, duplicate or missing information will produce incorrect or misleading statistics. So, the most important process during conversion of data from DS to DW is data cleaning. Data are cleaned, consolidated, aggregated and accumulated in multidimensional data structures to support direct querying and multidimensional analysis [4].

There is no defined methods of building complete and consistent DW and it depends on institution to institution which approach they wants to take [12]. Dimensional modeling(DIM) is the most common approach used in the design of DW database to organize the data for efficiency of queries that are intended to analyze and summarize large volumes of data. This data warehouse making process causes data latency. Based on the work in [13][14][12][15] [16], the following is the illustration of DIM fact schema.

DIM uses star or snowflake design, which would be easy to understand, supports simplified business queries and provides superior query performance by minimizing table join [16]. The principal characteristic of DIM is a set of detailed business facts, surrounded by multiple dimensions that describe those facts. These are described in DIM Fact Schema. A "fact schema" is defined in [12] as,

**Definition:** A fact scheme is a six-tuple

$$f = (M, A, N, R, O, S)$$

where:

1. M is a set of *measures*; each measure $m_i \epsilon M$ is defined by a numerical or Boolean expression which involves values acquired from the operational information system.

2. A is a set of *dimension attributes.* Each dimension attribute $a_i \epsilon A$ is characterized by a discrete domain of values, Dom(ai).

3. N is a set of *non-dimension attributes.*

4. R is a set of ordered couples, each having the form (ai,aj) where $ai\epsilon A \cup a0$ and $a_j\epsilon A \cup N(a_i \neq aj)$, such that the graph $qt(f) = (AUN \cup \{a_0\}, R)$ is a quasi-tree with root a0. a0 is a dummy attribute playing the role of the fact on which the scheme is centred. The couple (ai,aj) models a-to-one relationship between attributes ai and aj. We call *dimension pattern* the set $Dim(f) = \{a_i\epsilon A|\exists(a_0, a_i)\epsilon R\}$; each element in Dim(f) is called a *dimension*. When we need to emphasize that a dimension attribute ai is a dimension, we will denote it as $d_i$. We call *hierarchy* on dimension $d_i\epsilon Dim(f)$ the quasi-tree sub $(d_i)$.

5. $O \subset R$ is a set of *optional* relationships.

6. S is a set of *aggregation statements*, each consisting of a triple $(mj, di,$ $\Omega)$ where $m_j\epsilon M, d_i\epsilon Dim(f)$ and $\Omega$is an *aggregation operator*. Statement $(m_j, d_i,$ $\Omega)\epsilon S$ declares that measure $m_j$ can be aggregated along dimension di by means of $\Omega$. If no aggregation statement exists for a given pair $(m_j, d_i)$, then $m_j$ cannot be aggregated at all along $d_i$.

Graphically, Fact Schema with dimensions and measures can be represented as in Fig. 2.1 from [12]

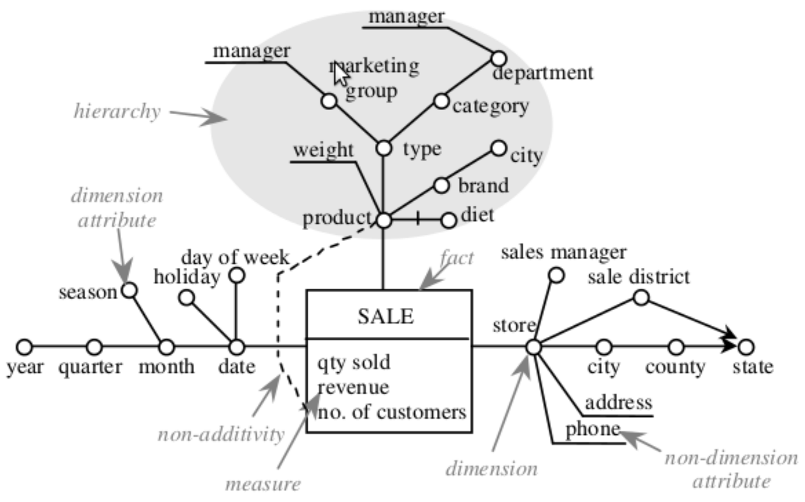


Figure 2.1: The SALE fact schema

As shown in example in Fig 2.1, Sale is the *fact* which is represented by a box with one or more numeric or continuous valued *measures* (quantity sold, revenue and no of customer in this case). Dimension attributes are represented by circles and each connected directly with fact are called *dimensions*. Dimension pattern of the sale schema is {date,product,store}. *Hierarchy* are the sub-trees rooted in dimensions as shown in the Fig. 2.1. There is a -to-one relationships between the arc connecting two attributes (for example, there is many-to-one relationship between city and country). Non-dimensional attributes are represented by lines instead of circles (for instance, address in Fig. 2.1 ), which contains additional information about an attribute of the hierarchy. The arc marked by a dash express *optional relationships* between pairs of attributes (for instance, attribute diet takes a value only for the food products).

If dimension tables can be joined directly to the fact table, this type of schema is called *Star Schema*. Whereas, if one or more dimension tables do not join directly to the fact table but join through other dimensions than this type of schema is called *snowflake schema*. Fig. 2.2 and Fig. 2.3 from [16] illustrates these two schema:

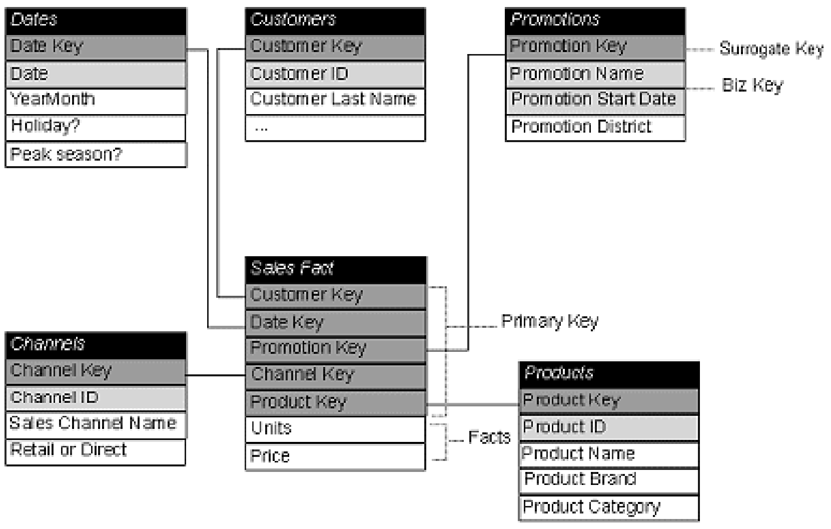


Figure 2.2: Star Schema

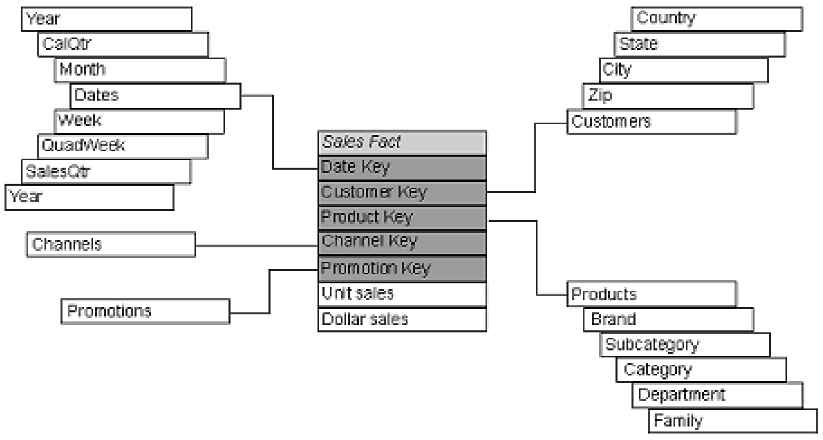


Figure 2.3: Snowflake Schema

## 2.2 Data Warehouse Architectures

There is still a considerable discussion and disagreement over which architecture of DW to use. The most common are Hub and spoke architecture (i.e. Centralized data warehouses with dependent marts), advocated by Bill Inmon, commonly referred to as the father of data warehousing and Data Marts Bus Architecture with linked dimensional data marts (i.e. bus architecture), advocated by Ralmp Kimball. In [17], Watson and Ariyachandra refer to four different data warehouse reference architectures which identify alternative ways in which data can be extracted, transformed, loaded, and stored in a data warehouse. They also studied some factors that affect selection of data warehouses and used metrics to compare and determine success of various architectures.

A multi-phased research done on various DW architecture in [17], which was the best for me to understand the different DW architecture present and also the factors that effect the architecture gave me the idea on what effects the architecture of DW and what is best for which scenario. The four architectures that were identified in [17] are:

1. *Independent data marts (IDM)* are independent of other data stores and are good for their own units. It doesn't give "a single version of

truth" since they are not integrated with other data marts and may have inconsistent data definitions. Independent data marts are described in Fig. 2.4 from [17]. Problems of Independent data marts:

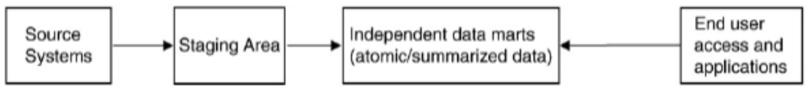

Source Systems
Staging Area
Independent data marts (atomic/summarized data)
End user access and applications

Figure 2.4: Independent Data Marts

- Since each data marts are independent, there are lots of redundant data.
- Since data marts are built independently by separate teams, there is a great deal of rework and analysis required.
- Independent data marts directly read operational system files and/or tables which limits decision support systems ability to scale.
- High level authorities and managers always want a combined report from all department irrespective of individual department. Since IDM is not integrated it cannot provide this function.

2. *Data mart bus architecture with linked dimensional data marts (DBA):* In this architecture, one mart is linked with another where the first contains common elements used by data marts such as conformed dimensions and measures that will be used with other marts. A logical integration of other marts is done to the first as they are developed based on the conformed dimensions of first mart. The demerits of this architecture is that it requires conformed dimensions across the system, in scalability and doesn't support RTBI. It is organized in a star schema to provide dimensional view of data, atomic and summarized data's are maintained. DBA is described in Fig. 2.5 [17].

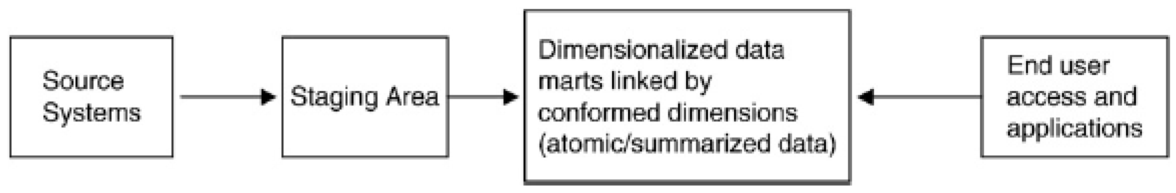

Source Systems
Staging Area
Dimensionalized data marts linked by conformed dimensions (atomic/summarized data)
End user access and applications

Figure 2.5: Data Mart Bus Architecture

3. *Hub and Spoke Architecture (HAS) :* In this Architecture, atomic level data is maintained in data warehouse in third normal form. Dependent data marts, whose source is data warehouse itself are developed for the departmental functional area or for special purposes like data mining as per the requirement. These data marts may be normalized, de-normalized or summarized/atomic. HAS is described on Fig. 2.6 from [17] and Fig. 2.7 from [18].

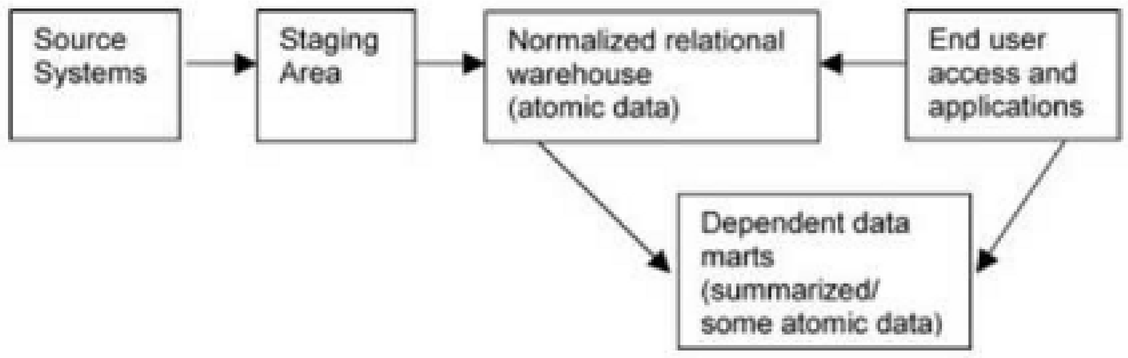

Source Systems
Staging Area
Normalized relational warehouse (atomic data)
End user access and applications
Dependent data marts (summarized/ some atomic data)

Figure 2.6: Hub and Spoke Architecture

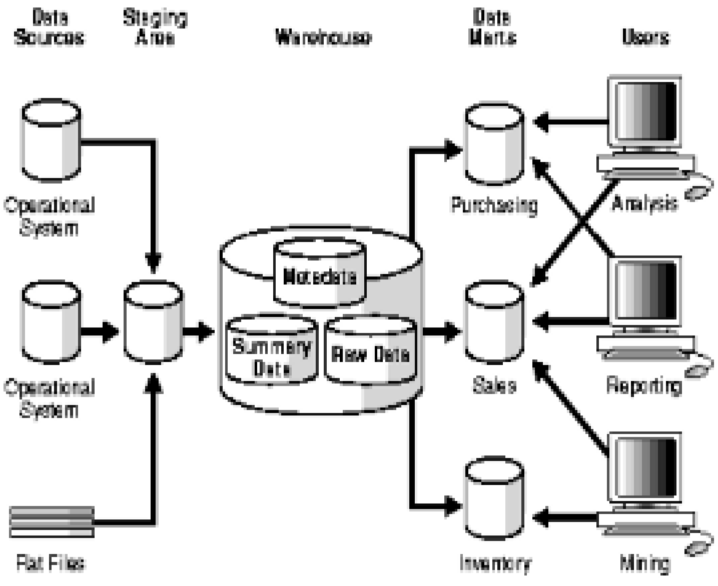


Figure 2.7: Hub and Spoke architecture with staging area, data warehouse and data marts

4. *Federated architecture (FED):* In this type of architecture all existing decision support structures(like operational systems, data marts and data warehouses) are in place and no modifications are done on preexisting environment. Data are accessed through these sources as per the requirement and are integrated either logically or physically using shared keys, global meta data, distributed queries or other methods. This architecture is also used in Oracle BI EE, SAP/BW. It requires conformed dimensions across systems, updates must be coordinated(between OLTP, DW, caches etc.), has sophisticated SQL generation and execution engine and requires high availability and problem diagnosis are hampered by multiple systems [19]. FED is described in Fig. 2.8 from [17].

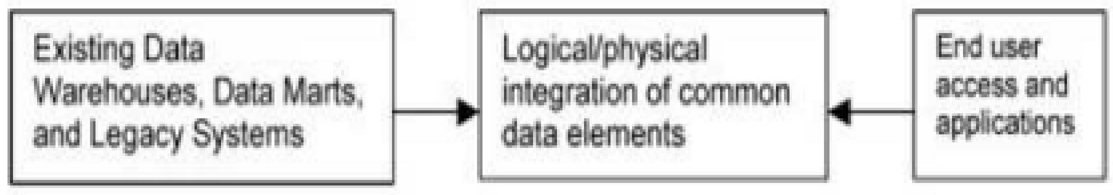

Existing Data Warehouses, Data Marts, and Legacy Systems
Logical/physical integration of common data elements
End user access and applications

Figure 2.8: Federated DW

There is also one more architecture, which Watson discussed in [20]:

5. *Centralized Architecture:* It has no dependent data marts like hub and spoke architecture. It contains only one centralized data warehouse which contains summarized/atomic data and logical dimensional views. Here since DW is centralized, department may lose control of their data and difficult to support high volume, low latency ETL along with complex ad-hoc decision support [19]. In some cases this is also defined as Enterprise data marts. Centralized Architecture is described in Fig. 2.9 and Fig. 2.10 from [17] [18].

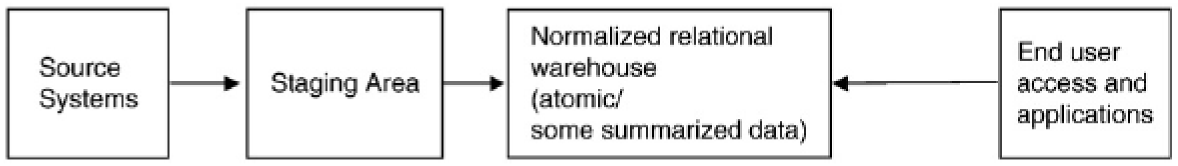

Source Systems
Staging Area
Normalized relational warehouse (atomic/ some summarized data)
End user access and applications

Figure 2.9: Centralized Data Warehouse Architecture

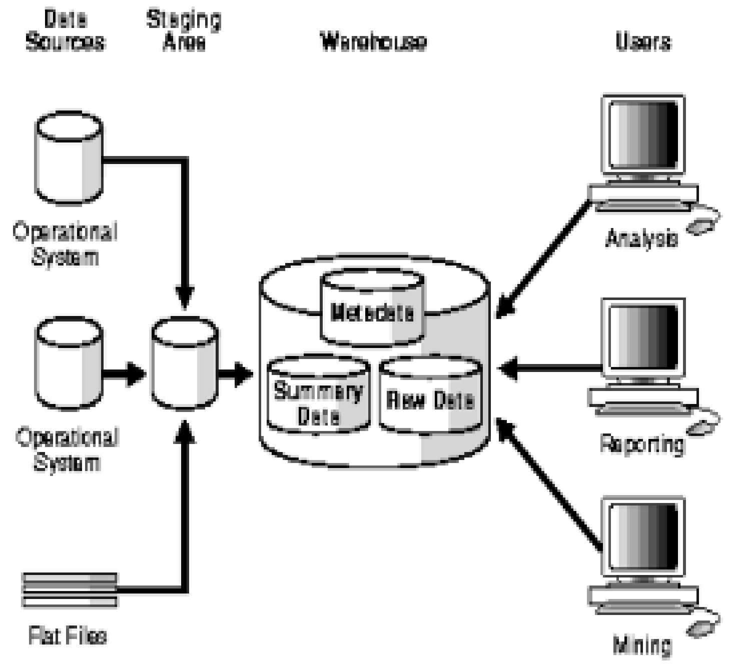


Figure 2.10: Centralized Architecture with staging area and data warehouse

With the advancement of the technology and the need for the business firms to have more robust DW architecture to support RTBI, different architectures come in light. Based on the five architecture we discussed, different alternative approaches to build the architecture were done.

## 2.3 Architecture of BI that supports Real Time

There are different architectures that are built to address RTBI. I categorized these solutions into following groups. Study of these work is done to find out the important components required for RTBI architectures.

### 2.3.1 Business Performance Management

The work in [21] presents a different approach to BI, called Business Performance Management(BPM). It includes DW but also requires a reactive component capable of monitoring the time-critical operational processes to allow tactical and operational decision-makers to tune their actions according to the company strategy. Business scenarios are always changing and the new requirement of the managers is to ensure that all processes are effective by continuously measuring their performance through Key Performance Indicators (KPI) and score cards. The paper describes the approach for metric driven management as shown in Fig. 2.11 below from [21].

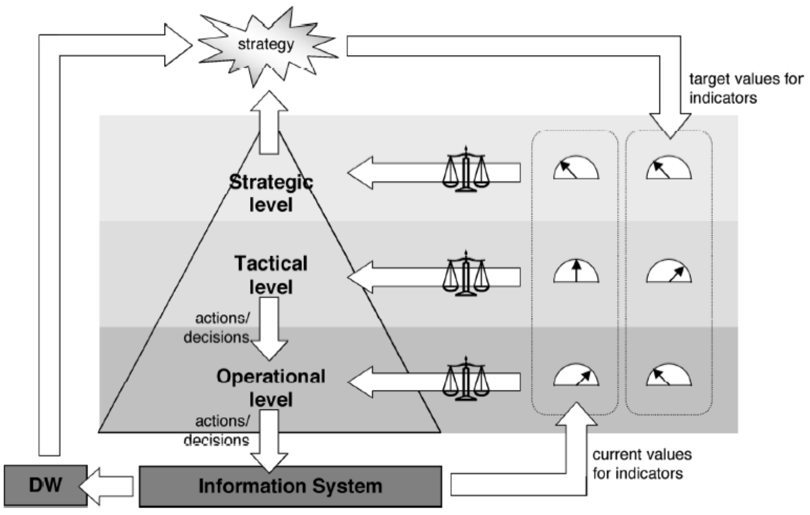


Figure 2.11: The closed-loop in the BPM approach

The organization structure with Operational level at the bottom, Tactical level in the middle and Strategic level on the top. All core activities are carried out on operational level and their decision power is limited in accordance with the main strategy. For example, optimizing specific production activities. Tactical level can have multiple divisions and they control the operational level, they controls set of functions and decision are taken in accordance with these functions. And the strategic level has the responsibility of making global strategy of the enterprise. BPM system supports decision making in right time but not in real time and is targeted not only to strategic but also to tactical and operational level users. If some piece of information is cannot be delivered on right time, then it is useless in the BPM context. The delivery time is faster in BPM since the operational level always need fast decision supports. The technology implementing BPM is called Business Activity Monitoring (BAM). BPM has BAM along with DW, which is illustrated as in the Figure 2.12 from [21].

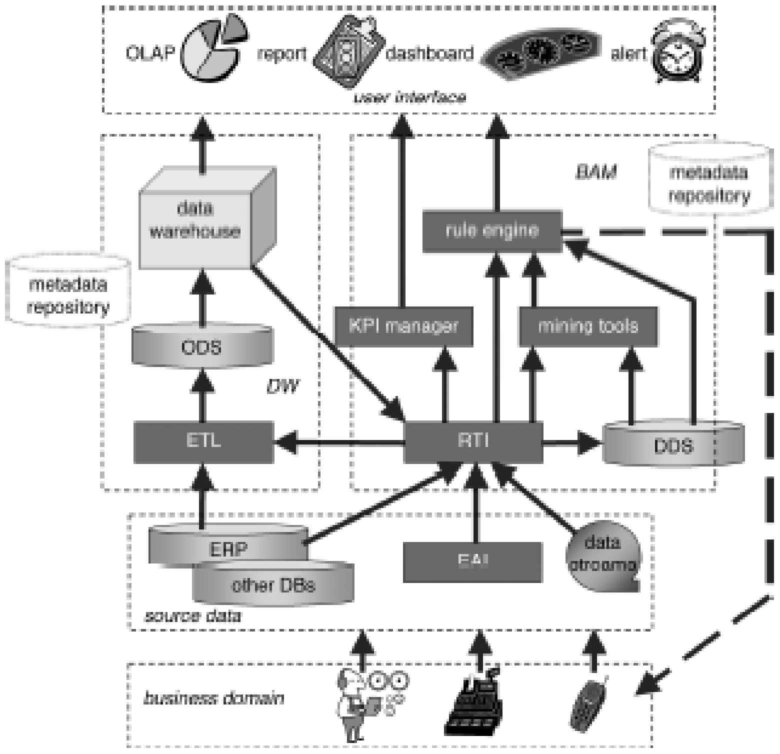


Figure 2.12: BPM Architecture

The "main components" introduced by BPM are [21]:

1. A Right time integrator (RTI) integrates data from operational databases, the DW, Enterprise Application Integration (EAI) systems and from real-time data streams at right-time.
2. A KPI manager computes all the indicators necessary at different levels to feed dashboards and reports.
3. A set of mining tools capable of extracting relevant patterns out of the data streams.

4. A rule engine continuously monitors the events filtered by the RTI or detected by the mining tools to deliver timely alerts to the users.

The strategic management analyzes medium and long term trends through OLAP tools and checks the effectiveness of strategy pursued in the short period through KPIs and dashboards whereas tactical and operational decision makers use other KPIs and dashboards to tune their actions to the company strategy. BAM emphasize in reducing the data latency by providing the tool capable of right-time filtering/cleaning/transforming/integrating the relevant data coming from OLTP/OLAP databases as well as from data streams [21]. This demand of right time data analysis made classical ETL and Operational Data Store (ODS) approaches unfeasible, as it raises problems in terms of data quality and integration, so we are in need of on the fly techniques. This on the fly integration has been investigated in some research prototypes [21], but still there are challenges:

- Most of the cleaning techniques used so far are in the presence of materialized integration level (purge/merge problem and duplicate deletion). Now, the expectation is to modify some of these techniques and re-implemented on proper data structures in main memory, in absence of this integration level.
- There are still many challenges on manipulating data streams and so complex queries are restricted to the offline and real queries are used only in simple filters.

The paper Real Time Business Intelligence for the Adaptive Enterprise describes business intelligence in three layers and the requirement for these layers to support RTBI. These layers are illustrated in Figure 2.13 from [10]

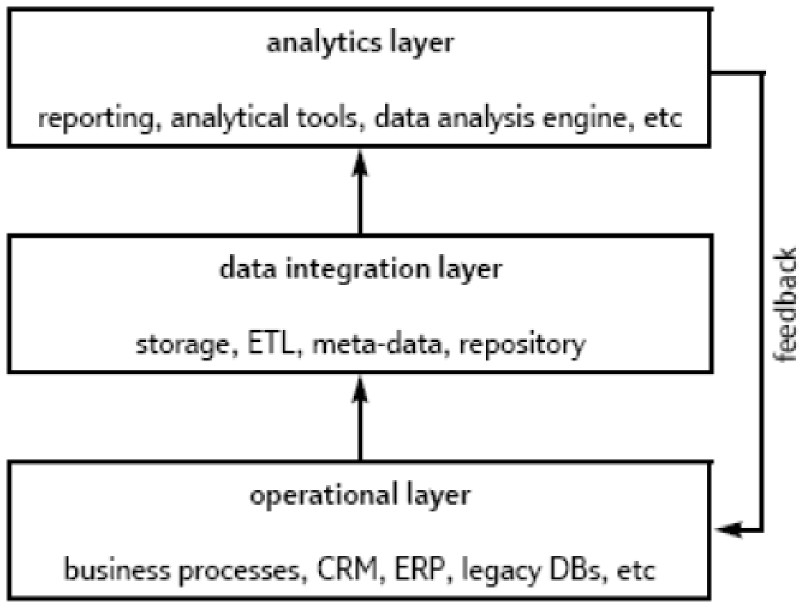

analytics layer
reporting, analytical tools, data analysis engine, etc
data integration layer
storage, ETL, meta-data, repository
operational layer
business processes, CRM, ERP, legacy DBs, etc
feedback

Figure 2.13: Layers on Business Intelligence

1. Analytic layer: BI requires expert analysts in between BI software that operates on the data and information used by the management, which prevents RTBI as it represents a time lag that can't be overcome. RTBI will require a high degree of automation. It must be able to select appropriate analysis methods and apply them automatically.

2. Data Integration layer: Traditional BI system has this layer for integrating different operational DS which causes data latency. However, for the RTBI to succeed, there should either be continuous real time data feed from operational DS to the data warehouses or has access to OS through some integration layer. There are many technology challenges such as platform, syntax, semantics and data quality metrics as well as many products and initiatives [10]. XML seems to have de facto syntax standard, J2EE and .NET as main platforms and the W3C Semantic Web initiative proposes to use ontology to address the data semantic problem.

3. Operational Layer: These layer must provide two functions for a com-

plete RTBI approach: BAM and real-time process tuning and change. The challenge for RTBI is to facilitate automated mapping of existing business processes within an organization and re-engineering, and monitor people and system for process conformance [10].

## 2.3.2 Complex Event Processing

Event processing is a form of computing that performs operations on events.

Complex event processing(CEP) is the technology that deals with processing of continuous arriving events with the goal of identifying meaningful patterns of the complex events [22]. It supports on the fly, real time processing of huge event streams. If we get an event, we do not immediately want to dispatch that to receiver but possibly want to filter these events, enrich them, combine them and run certain processing so that the receiver gets some useful output out of the system. So we have sender of events, receiver of events and in the middle event processing to do something useful to that data. CEP supports SQL language and pattern detection is one of the notable function of the event processing [22].

Following are the solutions based on complex event processing:

#### Concepts of Streaming SQL

Traditional Integration and business intelligence solutions can not address real time business models as queries are done on historical data, whereas in Streaming SQL queries and transforms data on the wire without any prior staging in a database. It is similar to database queries on how it analyze the data but differ by operating continuously on data as they arrive and by updating results in real time. It performs the same function as the ETL tool but differ as ETL process is a sequence of steps involved as a batch job. SQL Stream described in [23] is shown in Fig. 2.14.

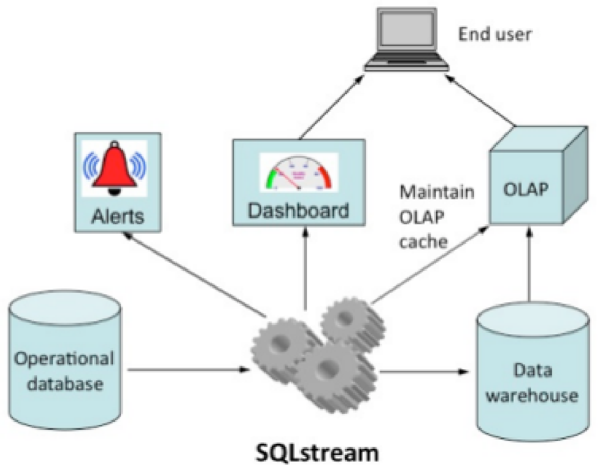


Figure 2.14: Concept of SQLStream

The diagram shows the architecture of a real-time business intelligence system. In addition to performing continuous ETL, the streaming query system populates a dashboard of business metrics, generates alerts if metrics fall outside acceptable bounds, and pro-actively maintains the cache of an OLAP server that is based upon the data warehouse [23]. Here are the "simple example", involving in which the SQL query delivers all orders from New York that are shipped within the time window of their service-level agreement (in this case, one hour) taken from [23].

```
SELECT STREAM *
FROM orders OVER sla
JOIN shipments OVER sla
ON orders.id = shipments.orderid
WHERE city = 'New York'
WINDOW sla AS
   (RANGE INTERVAL '1' HOUR PRECEDING)
```

Several research works have been done on SQL Streaming. Microsoft and Oracle build a module before ETL layer and it works based on SQL Streaming. The work in [24] build a general purpose prototype for data stream management system (DSMS), also called STREAM. While building a general purpose, DSMS poses many interesting challenges [24]:

1. Standard relation semantics cannot be applied to complex continuous stream of queries over data so they developed their own semantics and language for continuous queries over streams and relations called Continuous Query Language (CQL).

2. A physical query plan (PQL) composed of operators, queues and synopses are made from declarative queries, PQL are flexible enough to support optimizations and fine-grained scheduling decisions.

3. For high performance in DSMS, there should be possibilities of sharing state and computation within and across query plans. Furthermore, constrains in stream of data can be inferred and may reduce resource usage. Some of the techniques used to improve performance are by eliminating data redundancy, selectively discarding data that will not be used, and scheduling operators to most efficiently reduce intermediate state [24].

4. There can be change in data, system characteristics and query load may fluctuate over lifetime of a single continuous query so an adaptive approach where continuous monitoring and re optimization of subsystem is developed.

5. There can be more load incoming than the system‘s capacity which hinders exact result of the active queries. This problem is addressed through load-shedding by introducing approximations with some compromise in accuracy.

6. A graphical interface was developed to monitor and manipulate query plans as they run.

### Real Time Business Intelligence Support by Commercial BI

Microsoft's and Oracle's solutions for real time BI are based on complex Event processing where Microsoft introduced a technology called StreamInsight and oracle introduced a technology called CDC.

*Microsoft support for Real-Time BI[25]:* Real time BI are event based irrespective of traditional BI which are batched processing. Event processing was missing things before SQL Server 2008 R2. Now, SQL Server 2008 R2 includes several technologies such as PowerPivot and StreamInsight that facilitate the implementation of real-time BI solutions. In the presentation paper published by Microsoft for sql server R2, Microsoft StreamInsight is defined as a platform that we use to develop and deploy complex event processing (CEP) applications. Its high-throughput stream processing architecture and the Microsoft .NET Framework-based development platform enable you to quickly implement robust and highly efficient event processing applications [25]. In a presentation session provided by Microsoft, Real-time Business Intelligence with Microsoft SQL Server 2008 R2, illustrates a series of real-time BI scenarios powered by the use of Microsoft StreamInsight, Powerpivot and Microsoft Sharepoint Server 2010. Queries in StreamInsight are written in .NET Language-Integrated Query (LINQ) and we use windows concept if we need to know what is happening over a period of time.

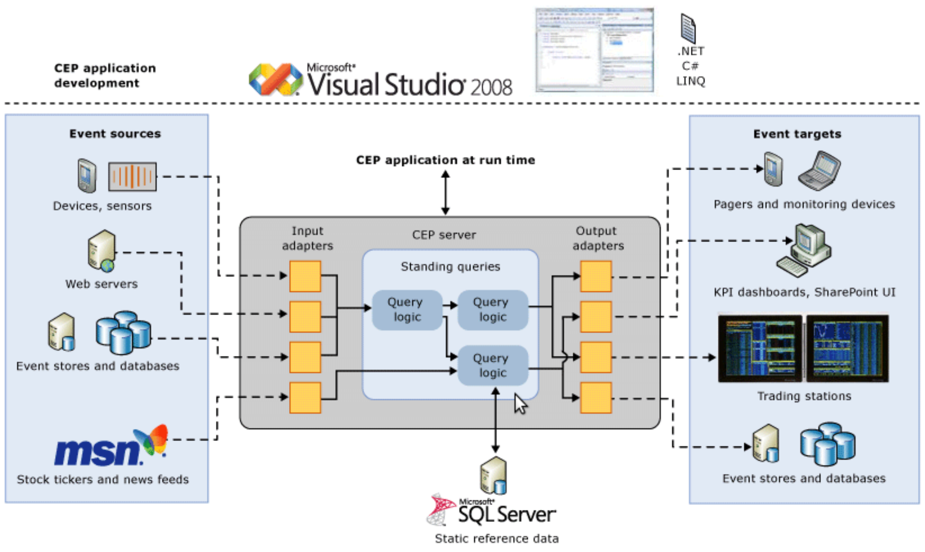

CEP application development
Microsoft Visual Studio 2008
.NET
C#
LINQ
Event sources
Devices, sensors
Web servers
Event stores and databases
msn
Stock tickers and news feeds
CEP application at run time
Input adapters
CEP server
Output adapters
Standing queries
Query logic
Query logic
Query logic
Microsoft SQL Server
Static reference data
Event targets
Pagers and monitoring devices
KPI dashboards, SharePoint UI
Trading stations
Event stores and databases

Figure 2.15: Stream Insight

*Oracle's solution to Real-Time BI through Oracle Complex Event Processing (OCEP)* OCEP [7] is Java based light weight application server specially designed to support event-driven application. As described in Fig. 2.3.2 from [7], the incoming data goes to the adapter module and is converted into an internal event representation. This event representation can be app defined java object or java-map. These event object created by the adapter goes to Stream Components, which are registered to "listen" to the adapter. The next component in the dataflow is an instance of CEP engine called a processor which hosts a set of queries written in EPL(Event processing language). The output of these configured EPL queries is send to POJO, which can perform additional processing and triggers action or send the output data to external system.

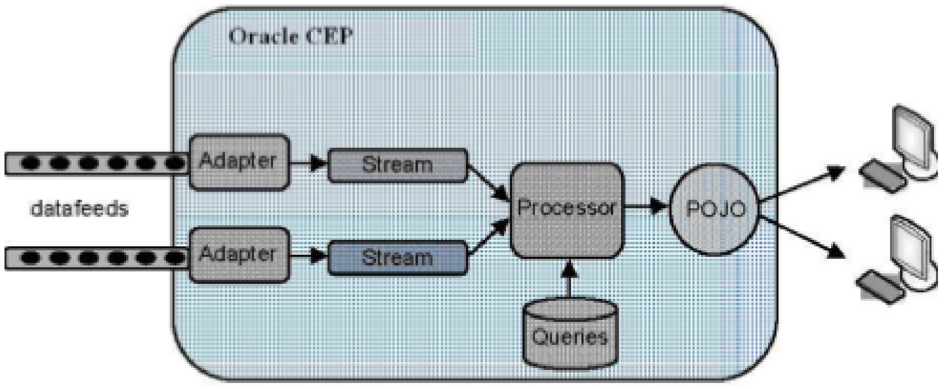


Figure 2.16: Oracle Complex Event Processing

*Oracle's Solutions for Real-Time BI Reporting [26]:* Oracle provides ODI to integrate data from various heterogeneous DS and perform integration in real time. Real time integration is possible through ODI's Changed data capture (CDC) feature. CDC as a concept is abstracted into a journalizing framework with Journalizing Knowledge Module (JKM) and Journalizing infrastructure at its core. Different methods for tracking changes using CDC are:

1. *Database Triggers:* When there is some table change, defined procedures are executed inside the source database. Procedures are defined by JKMs based on database triggers. Disadvantages of this method are limited scalability and performance of triggers produced. It is described in Fig. 2.17 from [26]:

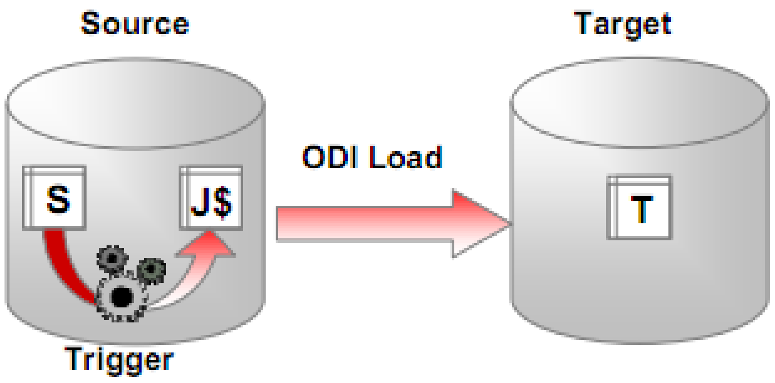


Figure 2.17: Triggers-based CDC

2. *Database log-facilities:* Log entries are processed and stored in separate table through Stream interface. These log-based JKMs have better scalability than trigger based. Some databases also provide API to process table change programmatically. It is described in Fig. 2.18 from [26]:

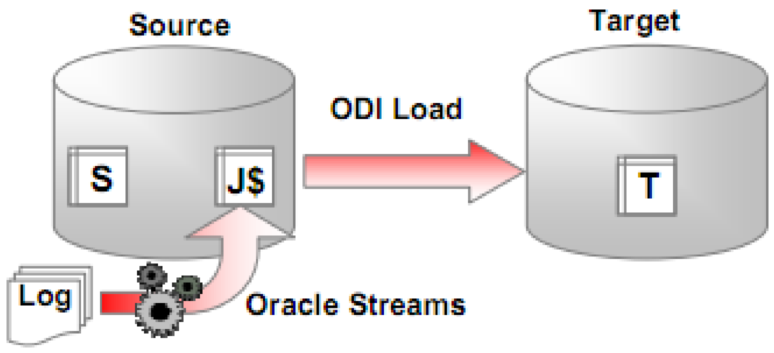


Figure 2.18: Streams based CDC

3. *Non-invasive CDC through oracle GoldenGate:* This architecture enables real time reporting in staging area and also loads and transforms data into analytical data warehouse. CDC mechanism provided by oracle goldengate can process changes in source by processing log files. It then stores captured changes in trial files independent of database

which are later transformed to staging area. JKM uses meta data managed by ODI to generate configuration file and process detected changes in staging area. These changes are then loaded to target data warehouse using ODI's declarative transformation mapping. It is described in Fig. 2.19 [26]:

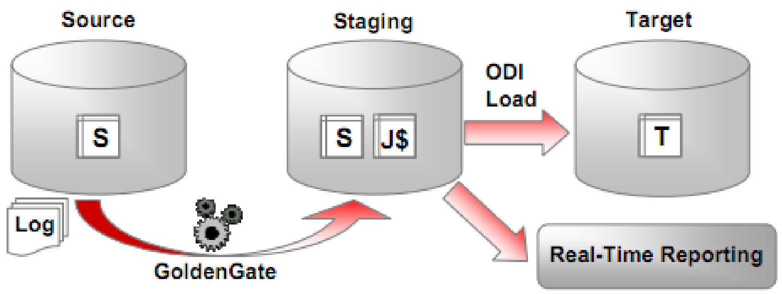


Figure 2.19: Golden Gate based CDC

The ODI Journalizing framework uses the publish-and-subscribe model. The framework has the following three step that are described in [26]:

- An identified subscriber, subscribes to changes that might occur in a data store. There can be multiple subscribers to these changes.
- Changes in the data stores are captured by the CDC framework and publish them for the subscriber.
- The tracked changes can be processed by the subscriber's at any time. The events are then consumed and no longer available for this subscriber.

Data Changes can be processed by ODI in two different ways. Pull mode, processes regularly in batch and Push mode, which processes in real time as the change occurs. The ODI journalizing framework using a publish-and-subscribe architecture is shown in Figure 2.20 from [26]:

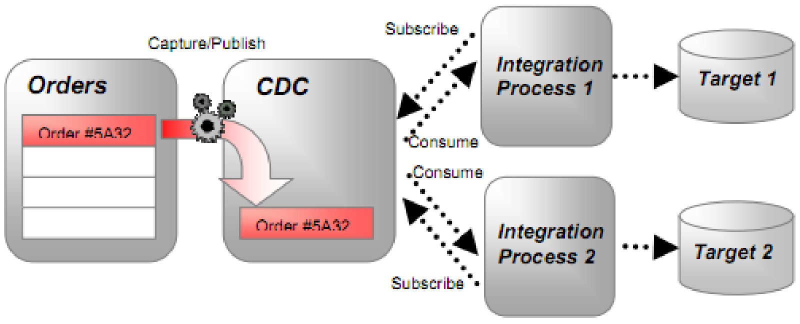


Figure 2.20: The ODI Journalizing Framework uses publish-and-subscribe architecture

### 2.3.3 NoSQL

NoSQL also called not only SQL is non-relational, semi structured data models where the query languages are MapReduce [27] unlike SQL in traditional SQL and can be used in Social data, data processing(Hadoop), search(lucene), caching, data warehousing and logging. NoSQL is defined in [28] as Next Generation Databases mostly addressing some of the points: being non-relational, distributed, open-source, horizontally scalable, schema-free, easy replication support, simple API, eventually consistent/BASE (not ACID), a huge data amount, and more.

*Amazon Dynamo DB* Amazon Dynamo is basically a simple key-value storage. Key-value stores have a simple data model in common, it is a map/dictionary allowing clients to put and request values per key. Dynamo provides only two operations to client applications: -get(key), returning a list of objects and a context. -put (key,context,object), with no return value.

Incremental hashing is maintained through consistent hashing which dynamically partition data across the storage hosts present in the system at given time. Dynamo also uses the concept of virtual hosts to overcome the problem of unbalanced distribution of data and load. Since, component failure is the standard mode of operation in dynamo, availability and durability is maintained through replication of data among nodes. Each data item is replicated N-times (where N can be configured per-instance of Dynamo). The storage mode which is in charge of storing a tuple with key $k^2$, is also responsible for replicating updated version of tuple with key k to its N-1 successors [29] [30] [31]. The replication mechanism is illustrated in Fig. 2.21 from [29] [31]

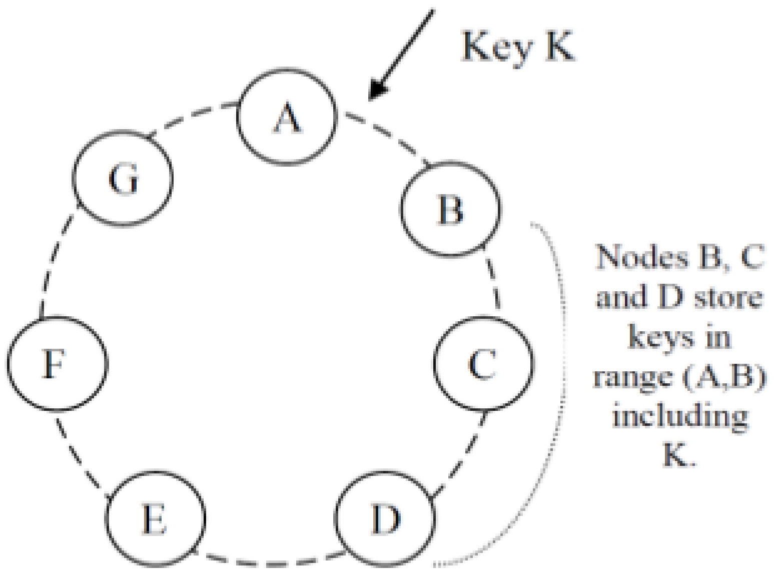


Figure 2.21: Consistent Hashing With Replication

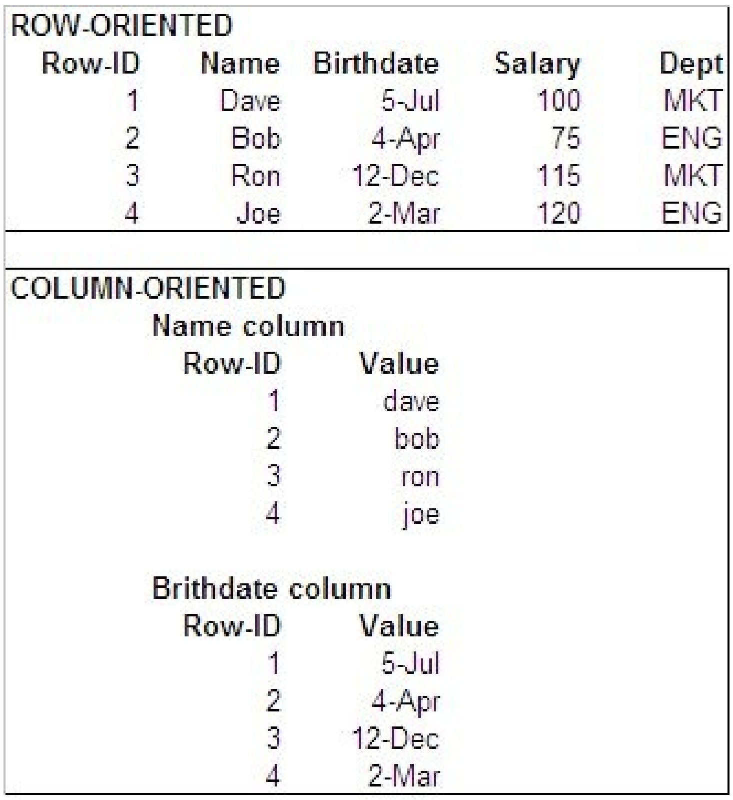
The need of low latency big data infrastructure were addressed by Google with BigTable [32] and at Amazon with Dynamo [31]. Hbase [33] is known as the open source solution of BigTable. At its core, HBase/BigTable is a map like associative array in PHP or dictionary in Python. Map is a type of datatype with collection of keys and a collection of values where each key is associated with one value.

for example:

```
Key=>value
{'1'=>"apple",
'2'=>'orange" }
```

Hbase and BigTable are built upon distributed file system so that the underlying file storage can be spread out among an array of independent machines. Hbase sits atop either Hadoop's Distributed File System (HDFS) or Amazon's simple storage service (S3), while BigTable makes use of the Google File System (GFS).

BigTable and Hbase both were meant to address big data solutions So, they are fundamentally distributed. Data are stored in large number of commodity hardware by partitioning and replications. Partitioning means each data is partitioned by its keys in different servers and replication means the same data element is replicated multiple times at different servers [34]. They are columnar database which means that each column is stored in disk unlike RDBMS where each row is stored continuously in disk. Fig 2.22 from [35] illustrates the difference between row based and column based:



ROW-ORIENTED

| Row-ID | Name | Birthdate | Salary | Dept |
|---|---|---|---|---|
| 1 | Dave | 5-Jul | 100 | MKT |
| 2 | Bob | 4-Apr | 75 | ENG |
| 3 | Ron | 12-Dec | 115 | MKT |
| 4 | Joe | 2-Mar | 120 | ENG |

COLUMN-ORIENTED

Name column

| Row-ID | Value |
|---|---|
| 1 | dave |
| 2 | bob |
| 3 | ron |
| 4 | joe |

Brithdate column

| Row-ID | Value |
|---|---|
| 1 | 5-Jul |
| 2 | 4-Apr |
| 3 | 12-Dec |
| 4 | 2-Mar |

Figure 2.22: Row-Oriented Vs Column-Oriented DBMS

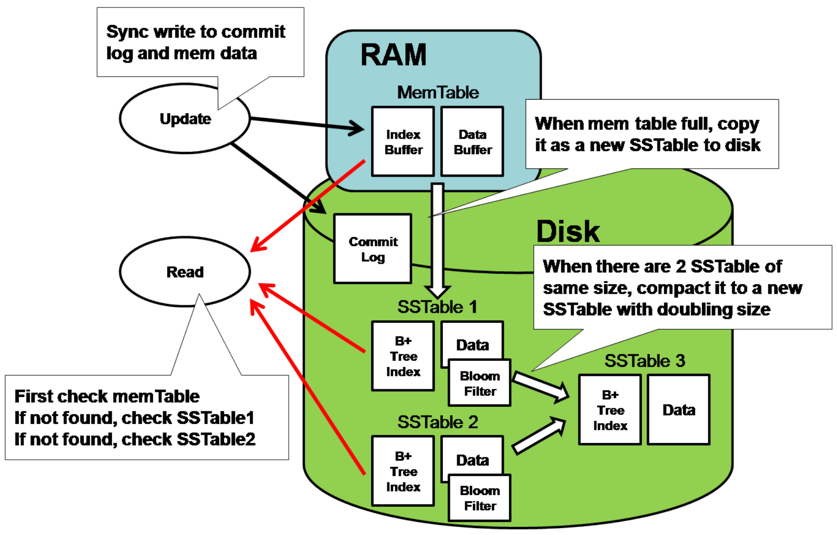

Sync write to commit log and mem data
Update
RAM
MemTable
Index Buffer
Data Buffer
When mem table full, copy it as a new SSTable to disk
Commit Log
Disk
Read
When there are 2 SSTable of same size, compact it to a new SSTable with doubling size
SSTable 1
B+ Tree Index
Data
Bloom Filter
SSTable 3
B+ Tree Index
Data
SSTable 2
B+ Tree Index
Data
Bloom Filter
First check memTable
If not found, check SSTable1
If not found, check SSTable2

Figure 2.23: Big Table Architecture

As shown in the Fig 2.23 from [36] , write operation is sequential in BigTable. Whenever write operation is done, it first append transaction entry to the log file, followed by writing the data in an in-memory memtable. All the latest entry will be stored in memtable until it is full. When memtable reach to threshold, then data will be copied to disk as an SSTable (stored by the string key).Over the period, there will be multiple SSTable which could be inefficient for the read operation, so the system periodically merge two SSTable.

In case of machine crash and all in-memory state is lost, recovery is done by replaying the updates in the log file to memtable. For read operation, the system will fist look at memtable by its rowkey to see if it contains the data. If not, it will check the SSTable in disk. SSTable has a companion called Bloom filter such that is can rapidly detect the absence of the row-key, which speed up the detection in SSTable [36].

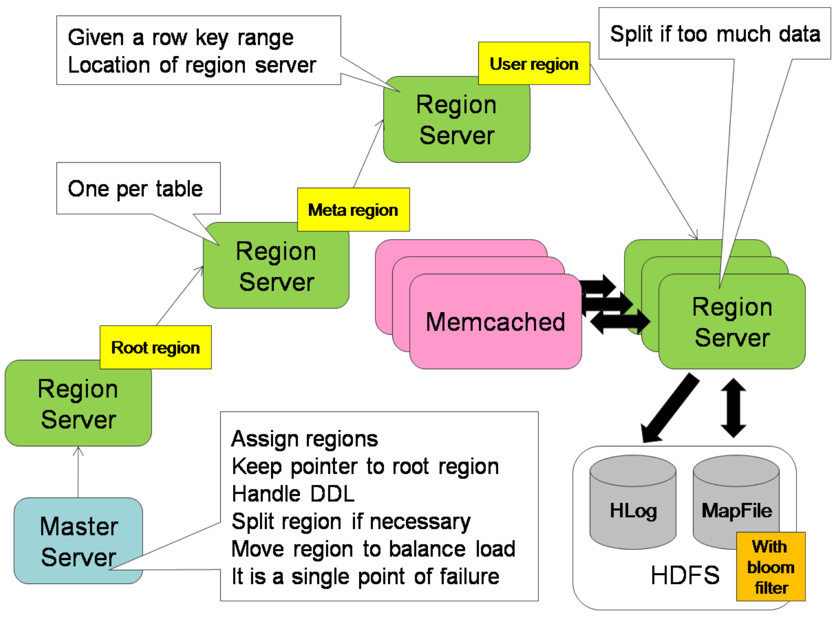


Figure 2.24: Hbase Architecture

As data storage engine is maintained in HDFS, data replication, data consistency and resiliency are all handled by HDFS. This is an advantage for Hbase but in the other hand it also has to rely on constrained of HDFS like it is not optimized for random read access and there is always an extra network latency between the DB server to the file server(which is the data node of hadoop).

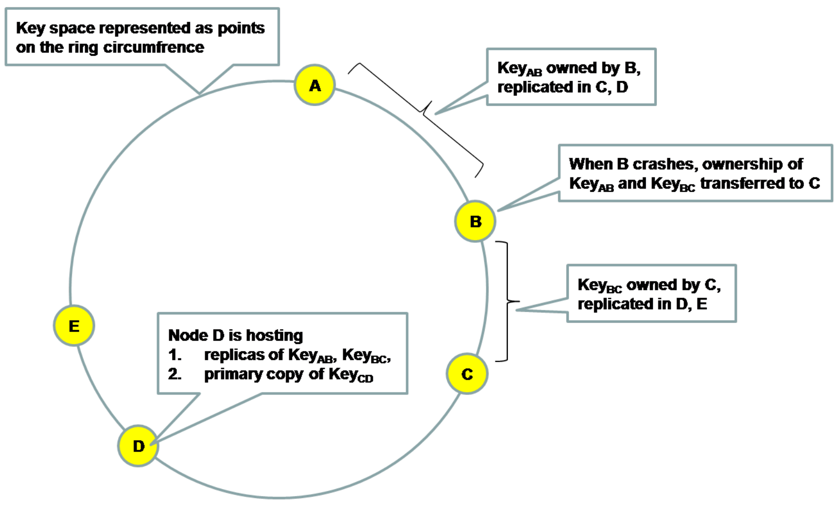

Key space represented as points on the ring circumfrence
A
Key$_{AB}$ owned by B, replicated in C, D
When B crashes, ownership of Key$_{AB}$ and Key$_{BC}$ transferred to C
B
Key$_{BC}$ owned by C, replicated in D, E
C
Node D is hosting
1. replicas of Key$_{AB}$, Key$_{BC}$,
2. primary copy of Key$_{CD}$
D
E

Figure 2.25: Cassandra Replication Architecture

*Cassandra [37]:* BigTable and Amazon Dynamo successfully meet the goal of scalability and reliability but they were not released publicly. Facebook on the other hand set out to build a technology that provide best of both: the powerful data model of BigTable with simplicity and peer-based replication and fault-tolerance of Dyanamo. This technology was made open-source by Facebook in 2008 and named Cassandra [38]. It is in use at Digg, Facebook, Twitter, Reddit, Rackspace, CloudKich, Cisco and more companies.

1. *DataStax's Brisk [39]:* The work in [39] describes it as enhanced open-source Hadoop and Hive distribution which provides integrated Hadoop MapReduce, Hive and job tracking and task tracker functionalities, while providing an HDFS (Hadoop File System)-compatible storage layer powered by Casandra (i.e CassandraFS). The result is a simple stack that eliminates the complexity of single-point-of-failure of the HDFS layer where Cassandra provides the single layer in which every node is pair of the other and automatically knows the position of the cluster. On startup, all Brisk nodes automatically start a Hadoop task tracker and one node is elected as the job tracker, if this node fails the job tracker is automatically restarted on different node. Traditionally, users were forced to move data in systems through complex ETL layer

which adds data latency but with brisk, both take place in distributed system but users have flexibility to assign resources so the analytical work is not slowed down and add data latency. They simply define one or more groups, and configure the role of each group one or more of Cassandra, Hadoop or HDFS (i.e HDFS without job/task tracker). DataStax's Brisk Architecture is discussed in [39] and shown Fig. 2.26.

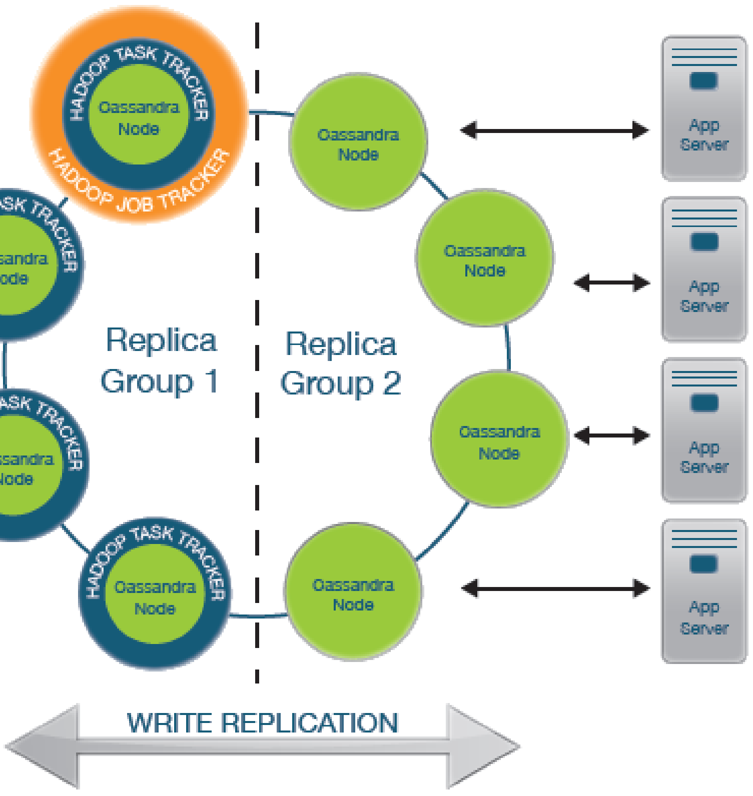


Figure 2.26: DataStax's Brisk Architecture

2. *Rainbird [40]:* is a layer top of the distributed counters patch, Cassandra-1072 to address low latency data requirement in twitter. It relies on zookeeper, Cassandra, scribe, thrift and is written in Scala. The real time data are kept in the aggregation buffers for one minute and they also intelligently flush to Cassandra. Query serves all data read once it is written. The architecture of RainBird is illustrated in Fig. 2.27

from [40].

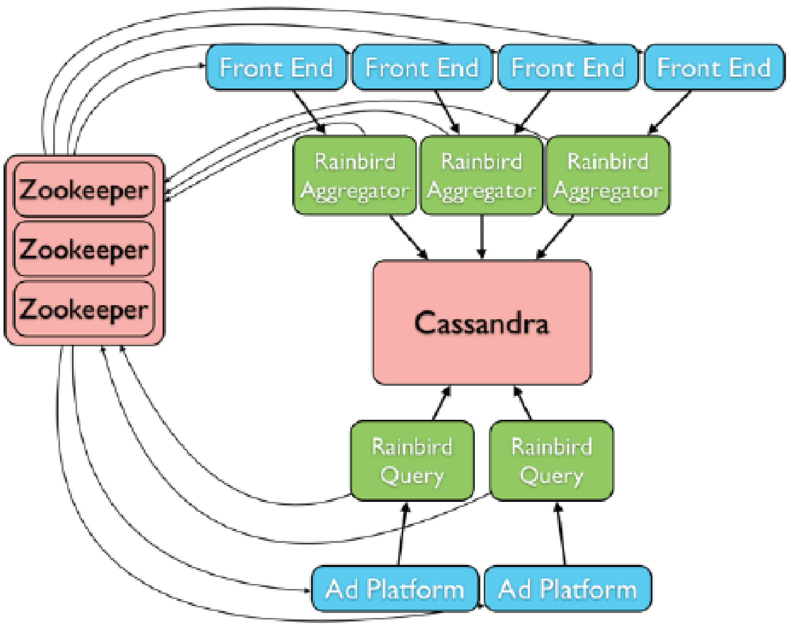


Figure 2.27: RainBird Architecture

The basic "Data Structure of RainBird" is specified as follows [40]:

```
struct Event
{
1:  i32 timestamp,
2:  string category,
3:  list<string> key,
4:  i64 value,
5:  optional set<Property> properties,
6:  optional map<Property, i64>
              propertiesWithCounts
}
```

### Memory based Computing

There are different commercial memory based computing appliances developed by industries giants in BI to address RTBI. Some of them are:

*SAP HANA [41]* is a commercial product from SAP with an in-memory computing appliance. It combines SAP database software with their own hardware designed for memory storage. It has parallel processing data stores which combines row-based, column-based and object-based storage techniques to support real-time analytics and transactional processing's [41].

*Oracle Exalytics [42]* is Oracle's in-memory analytic appliance was introduced at Oracle OpenWorld in October 2011.It is commercial product from oracle and is designed to run on Sun-only hardware with mash-up of various oracle technologies.

### Memory Databases

The most popular memory based databases are Memcached [43] and Redis [44]. Memcached is open source in-memory key-value store for small chunks of arbitrary data (Strings, objects) [43]. Redis on the other hand is also open source in-memory databases with slight additional functionalities than memcached like disk-backing, replication and use of virtual memory. Redis also supports datastructures like Hashes, Lists, Sets, Sorting etc in addition to normal Key-value pair.

# Chapter 3

# Analysis, Design and Implementation

## 3.1 Technologies overview

A program module is made to verify and test the architecture suggested in Chapter 3 to measure the performance. To complete the program module following technologies were used during implementations and testing.

1. Redis (REmote DIctionary Server) [44] is open source advanced key-value store written in ANSI C. Redis claim outstanding performance when it works with an in-memory dataset. But as per the use case, data can be persisted either by dumping the dataset to disk or by appending each command to a log. It is also referred as data structure server as keys can contain strings, hashes, lists, sets and sorted sets [44]. Redis was first developed as key-value store by Salvatore Sanfilippo in early 2009, to improve the performance of his own LLOOGG, an analytics product. Redis is simple to learn, lightweight text-based TCP protocol [45] [46]. Replication in Redis is maintained as Master-slave configurations. Scalability in Redis can be maintained through the use of disk virtual memory.

   Theoretically, On-the-fly reporting of data is considered the fastest way for real time reporting than any other disk storage as in this case it

| | Redis | Memcache |
|---|---|---|
| Strings | Yes | Yes |
| Hashes | Yes | No |
| Lists | Yes | No |
| Sets | Yes | No |
| Disk-backed | Yes | No |
| Replication | Yes | No |

Table 3.1: Redis Vs Memcache

saves a lot of extra time spend in fetching data from disk and writing data to disk. To implement this the best choice would be RAM based data storage. For this I chose Redis as it is open source, simple to learn and supports many data structures like Hashs, Sets etc which makes computation easy. Unlike normal key-value pair, Redis has many data structures and as I have implemented mostly Hashes and SET in my program, it is the best choice for my implementation. Hashes and Set is implemented as it makes computation more efficient than normal key-value pairs. As for instance, table below shows the comparison between Redis and popular in-memory database called memcache.

What makes Redis unique choice is described in table 1

2. Jedis[47] is an open source Redis java client developed first by Jonathan Leibiusky. Jedis was the unique choice for my implementation as it supports all the features in Redis through its API, is compatible with latest version of Redis and actively developed.

3. AdventureWorks Sample Database

   AdventureWorks database is Microsoft sample databases shipped with SQL server for education purpose. It has data which is like real scenarios and are avaliable for different components of BI. For instance: OS database, DW database etc. Following database samples were used in the implementation to test the performance of ETL tool and to make relative comparision with on-the-fly reporting managed in architecture described in chapter 3.

(a) AdventureWorks OLTP database samples.

(b) AdventureWorks DW sample database.

AdventureWork refresh is the sample ETL layer used to convert from OS to DW sample. This was the best choice for me as all databases for OS,DW and as well ETL sample script is available in single pack. Thus time taken for all the conversions can be done through already available example sample. This is just tested to make an idea on how the conversion process in ETL is going and how long it takes.

AdventureWorks for MySQL database was also used for the purpose of testing in architecture described in chapter 3.

Microsoft integration Services (a component of SQL server) is the ETL tools for microsoft which can be run on SQL server data tools(SSDT). SSDT is the business intelligence tools of microsoft where ETL can be made and run in Microsoft C$\sharp$ programming language. This was the best choice for ETL testing as I am using Microsoft sample database and SSDT is the one supporting it with readymade sample code to test. Furthermore, it is the one suggested by Bouvet technical team for the test of the architecture.

## 3.2 Architecture Analysis for RTBI

With different architectures and solutions discussed on Chapter 2, architecture which can support RTBI is of our concern.

Data that reach to DW has latencies. Along with these latencies, time is spend on querying and displaying data from DW to BIT and dashboards. Most of the BI architecture used in Bouvet are RDBMS based so the focus of this thesis is on improvement of RDBMS based BI architecture to support real time analysis and reporting. It is also excepted to be suitably integrated in the existing system which has ongoing DS,ETL and DW running on it. Since, the improvement is expected to suitably integrated and not effect the ongoing functionality that RDBMS based BI has, it is felt that real time analysis and reporting should be dealt with separately. For this the cases where real time intelligence are important is first identified and is processed separately.

The implementation of new architecture also need to be plug and play so it is easily integrated with the existing one. In short we need some component to deal with real time data in BI architecture.

### 3.2.1 Identification of Important Components for RTBI

Now my first task was to identify the important components used by different solutions to address RTBI.

1. In a Complex Business System, there are lot of DS available, and they can be heterogeneous. The complexity of the heterogeneous are addressed in different DW architecture by Data sampling module as discussed on 2.2. This is one of the component we are searching for our module.

2. To reduce the data latency for real time data analysis, can we build on the fly techniques irrespective of offline data query from DW? There are memory based databases evolving with noSQL vibes. Can we implement any one of them for RTBI. These memory based databases are noted.

3. Based on the study of CDC and oracles reporting services for RTBI, CDC technology that oracle has implemented for capturing the new data changes in DS and passing these data in the component for real time processing is noted. All data are not required for real time processing. There should be some mechanism to select some changed data that are of our interest. For example in banking system, if credit card data are only required for real time processing than data associated with its change should only be captured. This filters huge amount of data passing through the component and also helps in the real insight focusing on the data we are interested on. Furthermore, if the component for Real time intelligence can run continuously as the streaming

system, it can really reduce the amount of time it incurs in reporting of real time data. Whereas, CDC technology runs all its stored queries on the stream of data to check either they need it or not. If we can keep all the real time changes of data in memory and process it in the memory itself then we may increase the performance of the system.

4. If we closely analyze cloud based databases like Google's Bigtable and Hbase, both of them have temporary storage of data, memtable in Google and memcache in Hbase. These are used as a cache until it is full and then the data are transformed to ssTable in GFS in Google and HDFS in Hbase. There is no temporary storage of data from OS before it goes to DW in RDBMS. If a temporary storage and in-memory approach can be taken as in these approach, this may help in fast reporting. So, this component is noted for the new architecture.

5. There is need of on-the-fly reporting than from databases. When data are saved to databases, it is saved in hard drive. Operations are all done in RAM memory, so if real time data can be saved in-memory in RAM and when required report can be dispatched from memory itself, it will obviously be fast enough as it removes the overhead it has to go when saving to hard disk from RAM memory and reading from hard disk to RAM memory. This is noted for the new architecture.

6. The structure of the DW schema as described in 2.1, has some fact based storage of the tables for quick reporting. Can we build similar rule based storage in new architecture for better reporting as described in chapter 2 section 2.1. For example, if sales in certain area(for instance Stavanger) is intended to get than it will directly get from the memory without need of any computations.

7. As per the solution of SAP Hana, which is commercial and includes its own hardware to support memory based operation, what about if commodity hardware can support this features and can be used in any heterogeneous sources. This is noted.

Based on the problem description on 1.1 and the discussion on Section. 3.2, to address the problem of real time, modification in the architecture of traditional BI is done and came up with the following overview.

1. A plug and play Memory based component for Real time reporting (MBCFRTR) is made to address the problem of RTBI on existing BI architecture.

2. Data once stored in DW is not real time. It is considered as static data so for real time reporting, on the fly processing and storage is necessary. I added a module to store data in the RAM memory itself, for those data that are more critical for real time reporting. ETL has best performance if it is executed in batch and run in periodic manner. For the time until ETL is executed per period, most critical data are stored in RAM memory.

3. For the data to be stored in-memory, it should be converted to data models that is supported by RAM processing. Heterogeneity of the data sources are addressed by conversion of all the source data to single data model the architecture supports. It is the data kept in memory in Hashes and key-value pair. Details of the data model is described in 3.3.3.

4. A programming API is made for CDC in real time at the time when event is executed, as the real time data that should be captured is the most important.

## 3.3 Architecture Overview

The basic flow diagram for the implementation of MBCFRTR is described in Fig. 3.1. Critical data that are needed for the real time processing are processed separately through MBCFRTR component and passed to the presentation layer as shown in Fig. 3.1. Detail description of how the processing and reporting is done in MBCFRTR is described in the following sections.

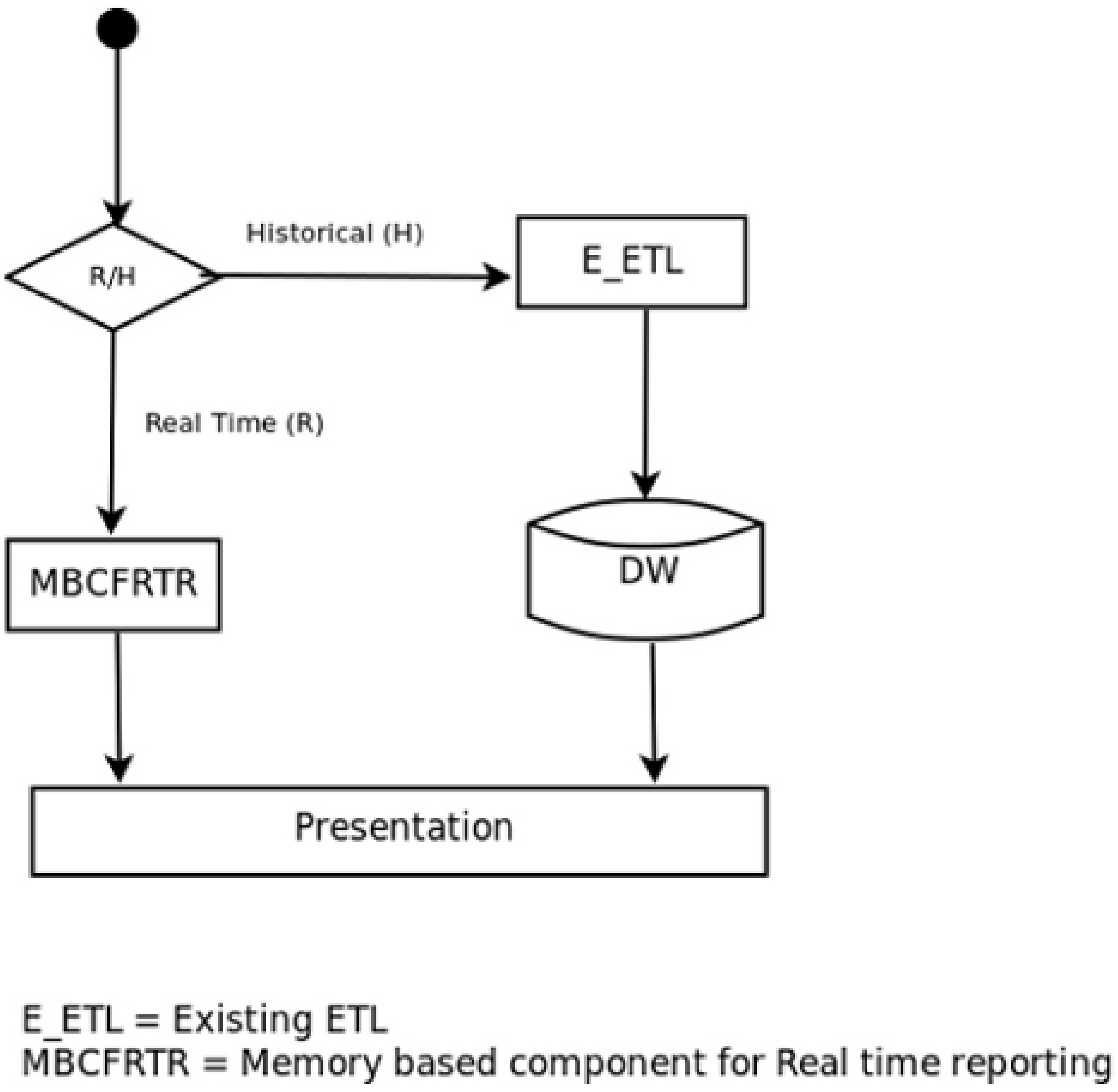


Figure 3.1: Flow Diagram for Real time BI

## 3.3.1 Architecture diagrams and its descriptions

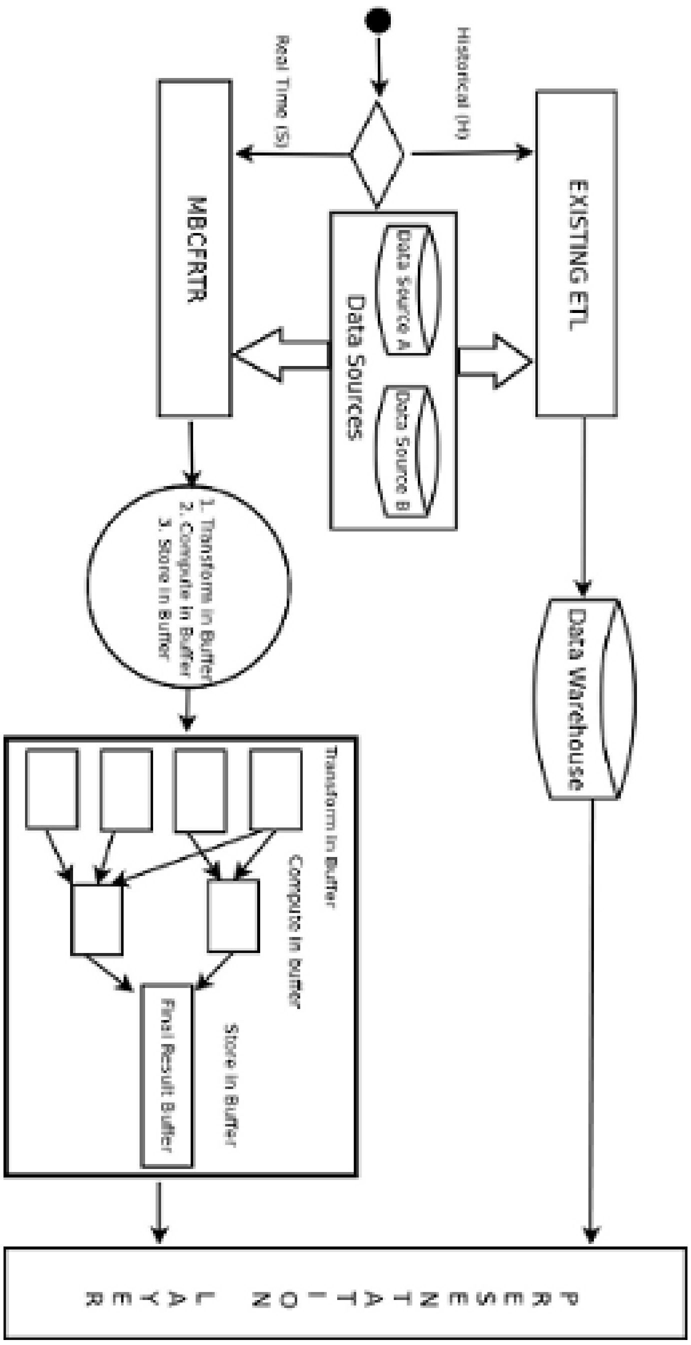


Figure 3.2: Architectural diagram for RTBI

The basic steps of operation for MBCFRTR is "everything is done in RAM memory" as described in the Figure 3.3.1:

1. Transform in memory.
2. Compute in memory.
3. Store in memory.

When any event is executed, data are initially transformed into memory through the functions created. These data while inserting in the memory, are stored in various buffer, based on the rules made for this particular type of event. Finally, when report is required, these are taken from these buffers. If some kind of processing is required before reporting, this is done in RAM itself and transformed into screen.

While inserting data in RAM memory, the expire time is also set through the function *setExpireTime*. The expire time can be set as per the table. The functionality of this expire time is to make the buffer free after certain interval of time. This is because MBCFRTR is meant to address the real time data that can be lost while the ETL process is executing and/or it is waiting for its period to execute. MBCFRTR also intended to make the system online while ETL and data transformation is in progress. But after the ETL has already transformed data to DW, the value of data is no more as Real time as it is already available in DW so deleting of this data would be more efficient. This is done by freeing the buffer by checking the expire time. Once expire time is set, it will automatically handle the deleting of the data and making the buffer memory free.

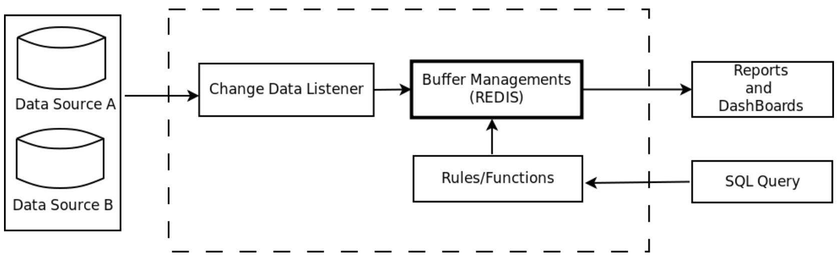


Figure 3.3: Detail View of Real time BI

The detail view of MBCFRTR is described as shown in Fig. 3.3. Data are captured from the data sources or in the program itself through some API. For the testing of this architecture, data are captured from the data sources and transformed into memory based data models, so am describing it based on changes captured through data sources. These data are when transformed into memory, they are stored in RAM through open source data structure server called REDIS. Redis is described in detail in Section 1, Section 3.1. Data model used to store data in memory is described in 3.3.3.

When real time reporting is required, they are just fetched from the memory buffer maintaning it and displayed to screen. For this SQL query is not supported by MBCFRTR, since data models are different in both. So, different functions are made just to replace the SQL statements to memory based functions to test the report generated. Some of them are:

```
selectAll(String tableName)
selectWithJoin(String tableA,String tableB)
```

### 3.3.2 Layers of MBCFRTR

The layer for the buffer management used in the implementation are described in Fig. 3.4. Data are stored in Redis data structure server. Data in Redis are accessed to the programming module through Jedis, which is Java API to access Redis Server. A programming module is developed which can be used to communicate between RDBMS data storage and Redis data storage. These are made in java to test the functionality of the MBCFRTR.

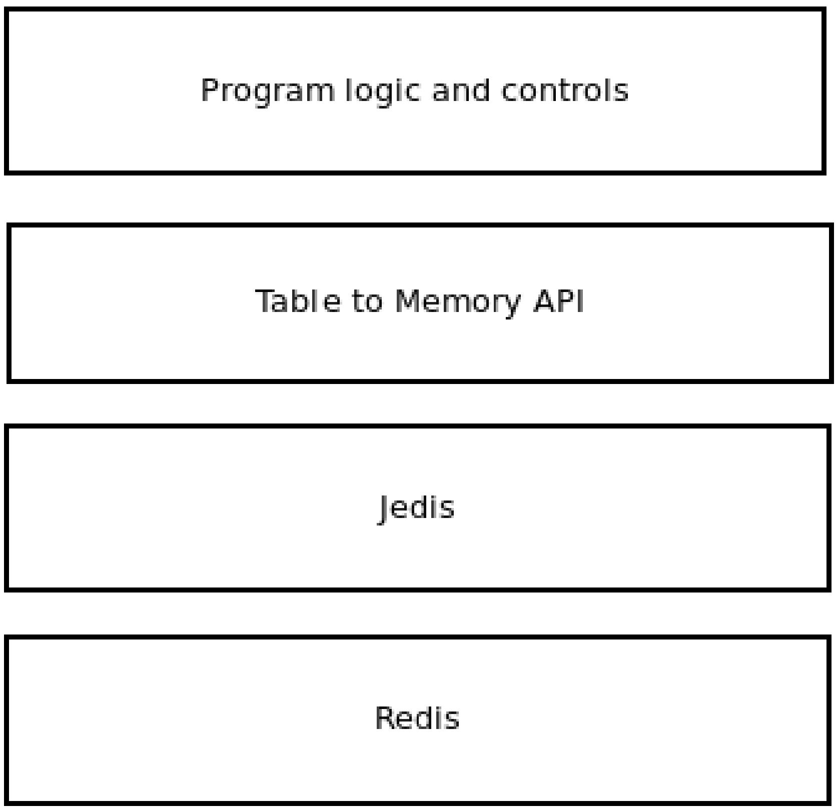


Figure 3.4: Layers on Real time BI

### 3.3.3 Data Model

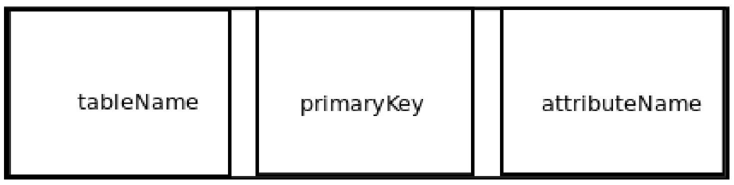


Figure 3.5: Data Models used for MBCFRTR

Data are stored in RAM memory in data structures like key-value pair, Hashes, Linked list etc. Therefore the first phase of the algorithm is to get the critical data and store them in data structures.RAM memory is selected for storage as it is considered the fastest in operating and our requirement is to maintain near zero latency. Ram storage best fits with data modeled in data structures. Therefore data structures like Strings, Hashes and Sets are used to store data in the implementation.

Data are uniquely stored as a key-value pair. Key is the one that will refer to the data and is unique. Modeling of tabular data to data strore in RAM is done through unique key creation. The model of the key that is implemented as a data model is as shown in Figure 3.5

Each row data is uniquely mapped as a key. This is done to maintain the sequence of tabular structure. Let us take an example to illustrate key model and how data is stored. For this let me take a table Product as shown in Figure 3.6.

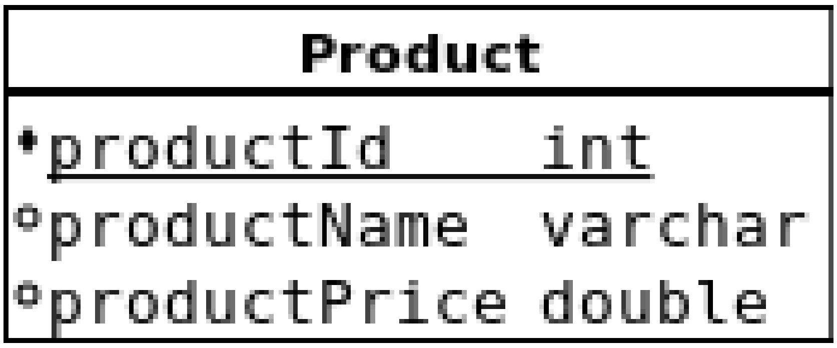
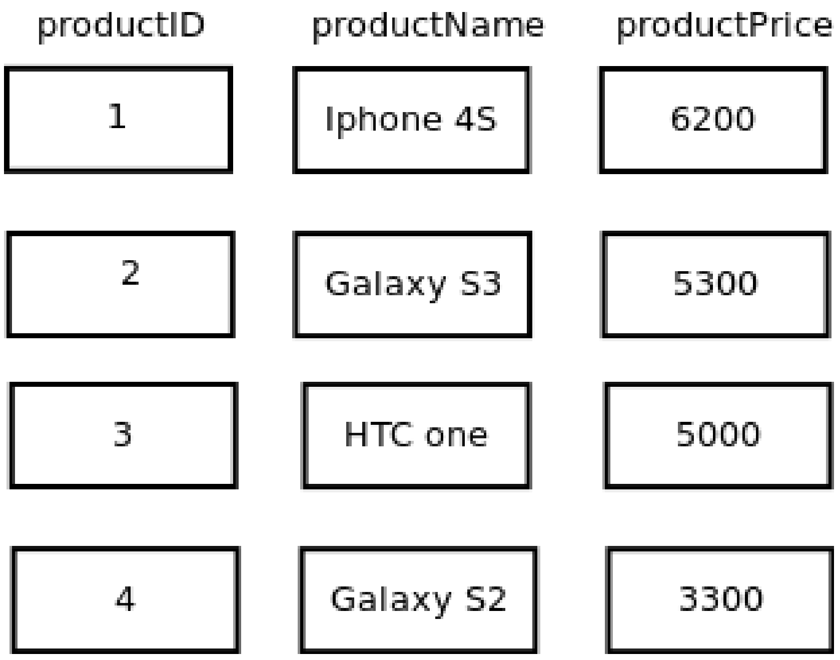


Figure 3.6: Example table: Product

Here Product table have productId,productName and productPrice attributes. This can have thousands of rows. The implementation of data model is done such a way that each data has its unique key as well as it can represent its relation to which row it lies. This maintains the uniqueness of each row. For instance, let us consider a data in product table as shown in the Figure 3.7



| productID | productName | productPrice |
|---|---|---|
| 1 | Iphone 4S | 6200 |
| 2 | Galaxy S3 | 5300 |
| 3 | HTC one | 5000 |
| 4 | Galaxy S2 | 3300 |

Figure 3.7: Example table : Product in table Form

The memory representation of the Product table is done such a way that each single data is represented as tableName:{primaryKey}:{attributeName}. Product table represented in Figure 3.6 with data in it as shown in Figure 3.7 is represented in memory as shown in Figure 3.8.

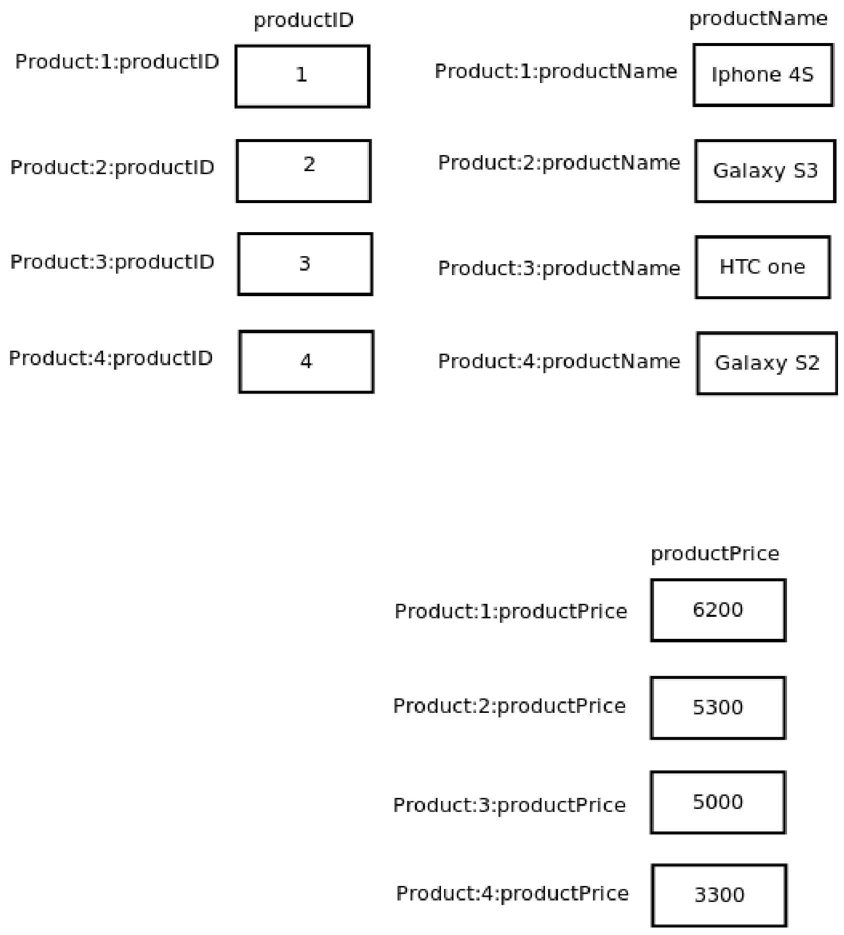


Figure 3.8: Product Table in in-memory data model

## 3.4 Additional Features

As Data model is maintained in Redis server, Redis takes care of fault tolerance in the architecture and some level of scalability which are important in times of fail of any nodes and bigger data. This solution is not intended for big data but it can maintain some level of scalability when the rush of data is too much. This is the feature of Redis which this architecture has advantage automatically.

1. Fault tolerant is maintained through replication mechanism of Redis. The implementation is very simple, all we need to do is change the

configuration file with slave of. eg. *s*laveof 192.168.0.1 6380, where 192.168.0.1 is the ip address of master node and 6380 is the port number.

2. Even though big data is not its target, memory of the system can be increased with the addition of virtual memory. This can be done by allowing Redis to use the disk space of the system as memory. eg. *v*m-enabled yes in redis-conf file.

## 3.5 Algorithms

The First step of the algorithm is to store data in noSQL format

A function tableToMemory is made to store data into Hashs like an object.In this case each row is stored in Hash table. And a function toMemory is used to store the data like a key-value pair in Hash table.

1. Step♯1. Data is retrieved from given table to NoSQL format.
2. Step♯2. Primary key of the table is stored in SET with data model like *all:tanleName*.
3. Step♯3. Key is generated per row and data are stored in Hashs like an object. It is like all the attributes are stored in fields of Hashs. *tableName:[primaryKey]:{attribute1,attribute2,attribute3}* so here key is tableName:[primarykey] in Redis hash, field is attribute1,attribute2... and value is the value it holds.
4. Step♯4. Expire time for the generated table is set, if and only if it is already set previously for this table. With this, if data is not important after certain interval of time, it is deleted automatically.
5. Step♯4. Rules based on the tableNames are checked. If there are rules defined and satisfies with the current value, they are kept in separate SET, with primary key. For instance, Rules with product sold in district Rogaland is kept in *equalsto:district:in:rogaland*.

The main purpose of storing data in rule based is for fast retrieval at the time of reporting. For instance, very important data can be set in rules when conditions are satisfied. This can be then retrieved directly when data with that condition is satisfied. Like in case of we need product sold with price equal to 765, rules are made and data are stored in this rule buffer.This can than be retrieved directly from memory and display on screen.

Note that another approach is also taken to store the data in normal key-value pair in Hashs itself. For this new function toMemory is made and all data are stored in single buffer. In this case, key is tableName, field is [primaryKey]:attributeName, and value is the value it holds. Comparison on both is done on Chapter 4.

The insertion of data from SQL to NoSQL format is also described in flow diagram shown in Figure 3.9

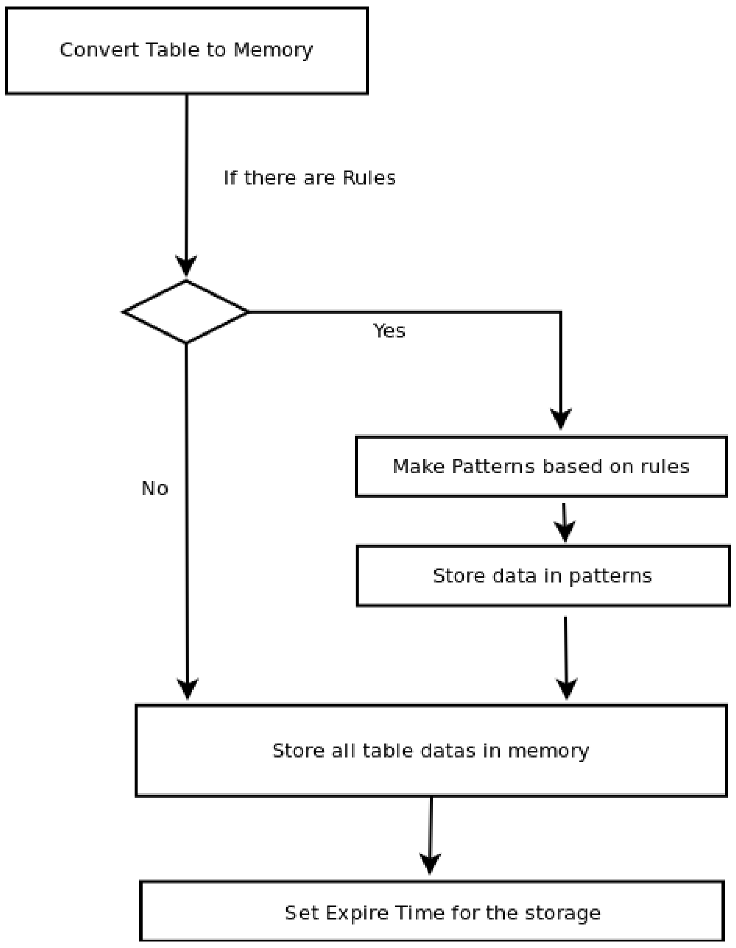

Convert Table to Memory
If there are Rules
Yes
No
Make Patterns based on rules
Store data in patterns
Store all table datas in memory
Set Expire Time for the storage

Figure 3.9: Insert table data into memory on MBCFRTR

### Data Acquisition Process

Data acquisition is done through custom functions created. It can be done directly when event is executed in programming module or captured from DS. To use directly from the programming module, simple functions can be used. Example code snippet looks like:

```
RedisDataBase rdb = RedisDataBase.getInstance();
String[] shipmodeAttribute = {"productID","productName",
"productQuantity","price"};
String[] shipmodeValues = {"1","HP","23","2999"};
```

```
rdb.insertIntoRedis("Shipmode",shipmodeAttribute,shipmodeValues);
```

To use from the data sources, data can be captured through CDC like mechanism in oracle. That is by using triggers and/or log updates or transforming from table to memory based using custom functions. Simple example snippet looks like:

```
MySqlToRedis mrd = new MySqlToRedis("root","root","adventureworks");
mrd.tableToMemory("product");
OR
mrd.toMemory("product");
```

Here complexity to deal heterogeneous data source is addressed by converting all of the data into single format (i.e. in memory). I chose memory approach rather than on other sources like file based, single database or xml, rdf and ontology based because it is the fastest way since all the data that we need is in memory and instantly available for processing. This approach is best for real time processing rather than file or database based as it reduces the time taken in reading and writing from file/database to memory.

### Rule based Storage

Previously, certain rules are made for the storage of the data per table. And while inserting data to the memory, if it satisfies, they are stored in separate pattern. Patterns are set as per the rules defined. Simple rules are defined initially targeting to a table. For example, when product price goes beyond level1, keep in the rule based storage.

### Join of tables in memory.

Complex joins are not efficient when there is huge amount of data so to speed up the join operation, possible patterns were matched and stored in SETS during the time of insertion of data when any events were detected in the system. This is done during the process of capturing the change events in the system. These processing speeds are maintained with less overload by creating some kind of pattern before and checking on those patters with the identification of the conditions. These identified patterns are kept in Redis's SETS so there is easy operations like union, intersection and other mathematical deviation of sets supported by Redis Server.

**Steps of implementation.**

1. Data converted from relational database format to key-value pair format. Datas tructures used are Hashes, Sets and Strings. This is implemented in two approaches. One with all the data in one Hash buffer and the another with single row of Tabular data to single hash buffer. Comparison on the results and analysis is done on Chapter 4.

2. Assign rules based on which memory buffer will store data for faster retrivel of data while reporting. This is to reduce data processing time when the retrieval of data is demanded.

3. Testing done on reports generated directly from the data stored in data store.

4. Testing done through some operations on data store and then the reporting.

# Chapter 4

# Results and Analysis

As described in 1.2.3,latency in BI is mainly caused due to dE+dT+dL function that is associated with the ETL layer. ETL layer has critical functionalities as described in 1.2.3. For these functionalities to execute, the basic operations of databases that are used will be Select, Insert statements and select joins. Furthermore, for the reporting basic operations that need to be fast enough is the select and insert statements so experiments were done on insert and select statements to check the performance of the MBCFRTR.

## 4.1 Experiment Scenario

Tests are done to check the performance of the purposed architecture. The purpose of the architecture is to reduce the latency time when the data are presented in the dashboards.

The data used to verify the tests are:

1. AdventureWorks OLTP

2. AdventureWorks DW

Tests are done based on MySQL, SQL, SSDT and MBCFRTR approach I have implemented.

Since, the window of the time that is intended for the data to populate is very small, so it is assumed that there will be no big data. This approach is

| Component | Technical Specification |
|---|---|
| Hardware platform | 4 Quad-Core Intel(R) Core(TM) i5-2400 CPU @ 3.10GHz |
| RAM Memory | 16GB |
| Operating System | Ubuntu |
| JVM | jdk1.4 |
| Redis | 2.4.14 Stable version |

Table 4.1: System Configuration for testing the architecture

| Component | Technical Specification |
|---|---|
| Hardware platform | 4 Quad-Core Intel(R) Core(TM) i5-2400 CPU @ 3.10GHz |
| RAM Memory | 16GB |
| Operating System | Windows 7 |
| Visual Studio | Visual Studio 10 |
| Business Intelligence tools | SQL Server Data Tools |
| SQL Server | SQL Server 2008 R2 |

Table 4.2: System Configuration for testing the ETL

to address the problem of data latency rather than to address the processing of the big data. Tests are done for up to 100,000 of records (each row with 4 column). The data base schema used is of adventureWorks sample database.

Every test were ran three times, and the higher figure obtained is used. Same machine was used as Redis client and Server to avoid network adding latency in the result. Also, Same machine was used as OS, DW and ETL to test the SQL integration system.

To test the functionality of the architecture with Redis Server, configuration is used as shown in table 4.1:

To test the functionality of existing ETL tool from microsoft with adventureWorks database, configurations is used as shown in table 4.1.

## 4.2 Benchmark Results

Tests are done on basic database operations on purposed architecture.

### 4.2.1 Data Insertion

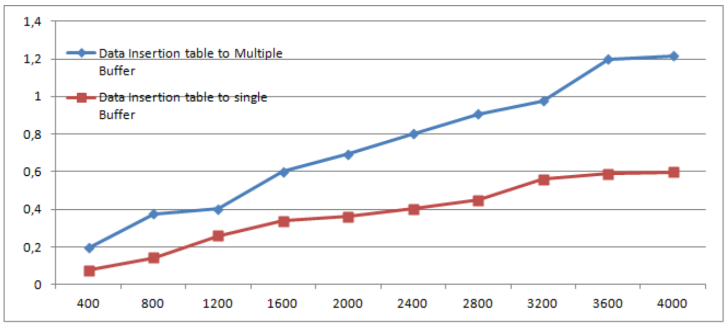


Figure 4.1: Data insertion in memory on MBCFRTR : for 400 to 4000 keys

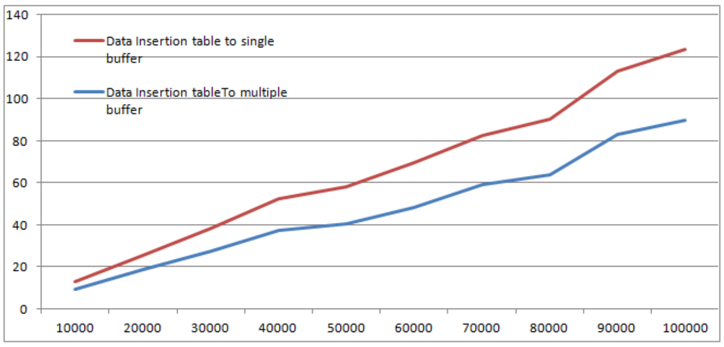


Figure 4.2: Data Insertion in memory on MBCFRTR : for 1000 to 100000 keys

When data is inserted into MDCFRTR, they can be done in two ways, either to store data on the fly when any event creates it in a programming module or transfer data from OS to MBCFRTR. For my test, I transfer it from OS and the time is recorded for the conversion as per the algorithm described in 3.5. Data is transferred in two phases, one with a small range of data and another with bigger data, up to 100,000 keys. Figure 4.1 is trend for the small range of data, and Figure 4.2 is for big range of data. I compared both approaches when whole data are kept in a single memory buffer or each row data in OS kept in unique buffer. Buffer is maintained with Hashs data structure of Redis.

The X-axis of the graph contains the number of key Redis can hold as per the time interval in Y-axis.The time interval represented is in Seconds. As we can see from the graph above, storing data in single buffer works fine for the small range of data but with the increase of data the performance of using multiple buffer work efficiently. As described in the graph, the maximum key the architecture can insert in one second is 3200 if single buffer is created for single row. Storing data in single Hash is effective if the data range is low but as the data range increases maintenence of multiple buffer looks effective.

Even though, data can be inserted with minimum time when data is inserted in single buffer, data manipulation and processing is much easier when it is in many hashes. This is inserting data as a object so in the implementation I used one row to one hash buffer mechanism while reporting.

## 4.2.2 Data Select

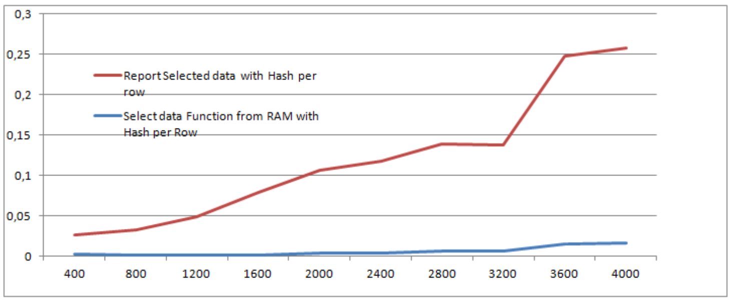


Figure 4.3: Select All data from Memory buffer in MBCFRTR - Small Range

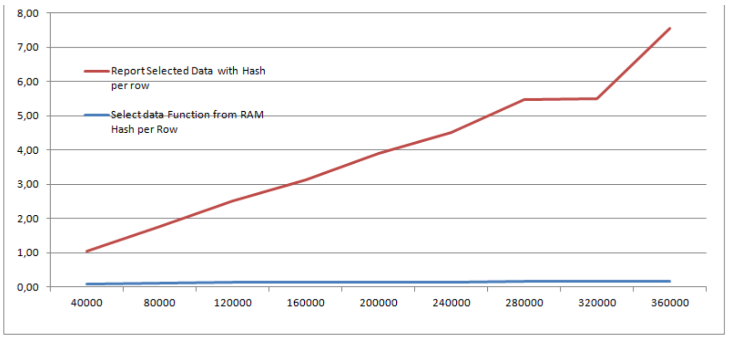


Figure 4.4: Select All data from Memory buffer in MBCFRTR - Big Range

As shown in the graphical figure 4.3 and 4.4, the select time for the data from memory is negligible as it is already in the memory and the time that is attached is due to the API code running while displaying data in screen. Time takes to display data in screen so time depends on how huge data it contains.

### 4.2.3 Random Select

Random Selection from the Ram database ranged from 0.16 to 0.26 ms and it does not show any dependency on the amount of data, buffer contains.

### 4.2.4 Join Operation

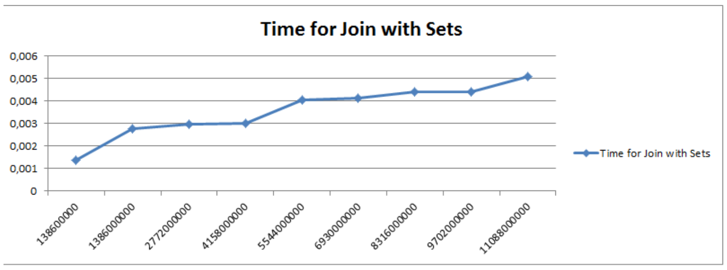


Figure 4.5: Join operation with Sets on MBCFRTR

Join operation is always complex as there are some kind of comparisons. For the suggested architecture, I used the feature of Redis data structure SET so the comparison became more easier and faster. The graph represented in Figure 4.5 is the time taken for retrieving data through joining two tables with the number of key represented in X-axis and time taken to do so in Y-axis(in seconds). It is quite fast with 110880000000 key comparisons in about 0.005 second. This join operation and also on processing other analysis has direct effect on how data are stored in memory while data insertion. For the operation of join, I stored the foreign key and primary key in different sets and did the comparison based on its default functionality of Redis.

But this may be changed if there are more complex operations and joins need to be calculated with complex computation.

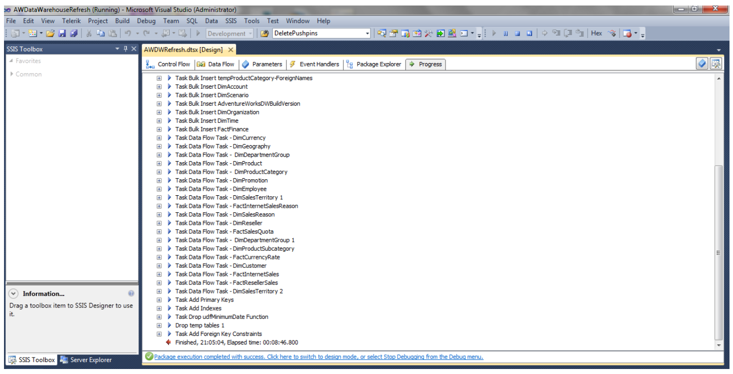
## 4.3 Analysis on Results

To compare the results, sample adventureWorks ETL function of SQL server is executed and time is recorded for it to transform to DW. Generally, ETL is executed periodically so there is latency added on the wait for it to execute. Additionally, it requires time to execute query from DW and transforming data from DW to RAM memory to display it in BIT.

Conversion of data from OS to DW is done in SQL Server Integration Services of Microsoft to keep the record of the time taken to transform the data. It is done in sample data base of microsoft called adventureworks. These data are small, which may not be the same as real time scenario, where data through ETL is large. The execution of sample ETL is just to test how much time it takes in the conversion and the complexity of algorithms it use in transforming whole table. If this wait is to be done for the real time reporting than the data may lose its value. The purpose of this testing is just to explain what advantage, the new added component can do. In my test scenario it took 8.46 min to transform whole table.

The table schema of OS and DW are attached in the enclosed CD along with the Integration codes The size of the table recorded is 192 MB.

The purpose of the additional MBCFRTR layer is not to replace the existing ETL layer, it is to address the data that loose its value while it has to wait for the ETL layer to finish and/or wait for the ETL period to execute. This is the way how ETL works and it works well for bigger data. During these period and on periodic wait, data are generated and stored from OS or with the event creation. These are kept in memory with MBCFRTR. Now for a big data up to maximum 100,000 records, how much time it takes to transform from event created to report in screen is of our concern, These are compared on the following diagram. This is one of the scenario, which illustrates that this memory storage will give robust performance when implemented. As explained in Figure. 4.7, now the memory based storage of data for real time can have expire time for not less than 8.46 minute. After 8.46 minute OS data are reached to DW and hence the value of data in MBCFRTR loses its value.



```
Task Bulk Insert tempProductCategory-ForeignNames
Task Bulk Insert DimAccount
Task Bulk Insert DimScenario
Task Bulk Insert AdventureWorksDWBuildVersion
Task Bulk Insert DimOrganization
Task Bulk Insert DimTime
Task Bulk Insert FactFinance
Task Data Flow Task - DimCurrency
Task Data Flow Task - DimGeography
Task Data Flow Task -  DimDepartmentGroup
Task Data Flow Task - DimProduct
Task Data Flow Task -  DimProductCategory
Task Data Flow Task - DimPromotion
Task Data Flow Task - DimEmployee
Task Data Flow Task - DimSalesTerritory 1
Task Data Flow Task - FactInternetSalesReason
Task Data Flow Task - DimSalesReason
Task Data Flow Task - DimReseller
Task Data Flow Task - FactSalesQuota
Task Data Flow Task -  DimDepartmentGroup 1
Task Data Flow Task - DimProductSubcategory
Task Data Flow Task - FactCurrencyRate
Task Data Flow Task - DimCustomer
Task Data Flow Task - FactInternetSales
Task Data Flow Task - FactResellerSales
Task Data Flow Task - DimSalesTerritory 2
Task Add Primary Keys
Task Add Indexes
Task Drop udfMinimumDate Function
Drop temp tables 1
Task Add Foreign Key Constraints
Finished, 21:05:04, Elapsed time: 00:08:46.800
Package execution completed with success. Click here to switch to design mode, or select Stop Debugging from the Debug menu.
```

AWDataWarehouseRefresh (Running) - Microsoft Visual Studio (Administrator)
File Edit View Telerik Project Build Debug Team SQL Data SSIS Tools Test Window Help
Development
DeletePushpins
SSIS Toolbox
AWDWRefresh.dtsx [Design]
Control Flow
Data Flow
Parameters
Event Handlers
Package Explorer
Progress
Information...
Drag a toolbox item to SSIS Designer to use it.
SSIS Toolbox
Server Explorer

Figure 4.6: Conversion time in Microsoft SQL Server Integration Services

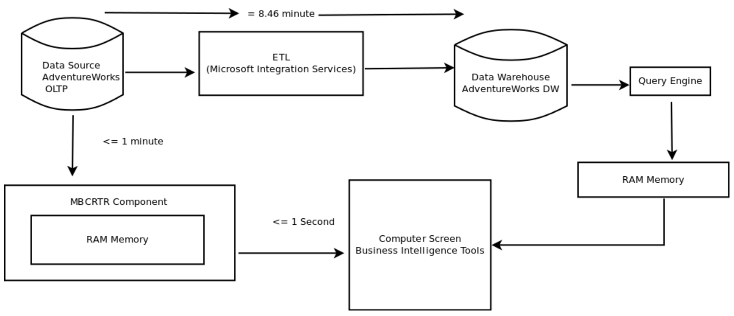

= 8.46 minute
Data Source AdventureWorks OLTP
ETL (Microsoft Integration Services)
Data Warehouse AdventureWorks DW
Query Engine
<= 1 minute
RAM Memory
MBCRTR Component
RAM Memory
<= 1 Second
Computer Screen Business Intelligence Tools

Figure 4.7: Comparison when data flow from existing ETL and from MBCFRTR

## 4.4 Differences between MBCFRTR with the existing solutions.

### 4.4.1 MBCFRTR Vs Complex Event Processing

In CEP, there are a collection of query events that are executed in a stream of data to check if the condition has satisfied for the required report. Afterwards, report is broadcasted to the reporting tools. On the other hand, in MBCFRTR, whenever any condition is satisfied during the time of execution of an event they are automatically stored in memory, if required computation is done in memory and the system reports the query only when it is asked to MBCFRTR for reports.

### 4.4.2 MBCFRTR Vs Amazons key-value

Amazons dynamo has very simple data model with key-value pairs and it can be used as a replacement of whole database. Therefore, data are saved in the disk in regular intervals. Whereas in MBCFRTR, there are more complex data structures in addition to simple key-value pair. In the implemented architecture work, data are not saved to the disk as the purpose is to use memory based storage for keeping records for the time interval up to ETL layer finishes its functionality. MBCFRTR is not a replacement of the whole database system but is an additional component of BI. Furthermore, amazon key-value pair is used for parallel processing and cloud computing so key-value pair is made to make the computation more simple. Whereas MBCFRTR is not implemented for cloud computing.

### 4.4.3 MBCFRTR Vs Cloud Computing

Cloud computing is not suitable for all problems. As described in Hbase website [33],

"If there are hundreds of millions or billions of rows, then HBase is a good candidate whereas if there are a few thousand/million rows, then using a traditional RDBMS might be a better choice due to the fact that all of your data might wind up on a single node (or two) and the rest of the cluster may be sitting idle."

This is the reason I didnt used cloud computing in my implementation since my target is for real time reporting. This system is targetted for certain amount of data and the time frame of running the system is almost continuous. Hence for continuous running of the application, there is no probability that the data will be too big upto millions to transfer. Furthermore, after certain interval of time when the ETL has finished its performance and data are already send to DW, the component deletes the data. So, cloud computing approach was not suitable for the implementation as the target of the system was for continuous and almost real time and also the data in memory buffer are in intervals flushed out. Therefore, there will never be very big data to compute in the component.

# Chapter 5

# Conclusion and Future Work

## 5.1 Conclusion

Real time BI is addressed through MBCFRTR and a plug and play architecture is suggested which supports the ongoing ETL functionalities of BI as well as an additional System for real time data. Implementation is done through open source in-memory data structure server to manage the buffer for the storage of RT data. This implemented architecture is tested for performance and compared with the time it saves if not going through the DW. Reporting in this approach is done on the fly. This architecture is not a replacement of the ongoing architecture as a whole but is an improvement on the present architecture for Real time intelligence.

As defined in the problem description 1.1, new architecture is made to address the RTBI, prototype is developed and implemented to check the performance of the proposed architecture. The heterogeneity of the data sources which cause complexity in processing is addressed by converting all the data in the form of memory based data structures. Data are then processed from memory and stored in the memory for fast display in the computer screen through dashboards or reports. This addresses the data latency in the system. Furthermore, the performance of the architecture is made more robust through some kind of pattern creation which address analysis latency in BI.

## 5.2 Future Work

1. Automatic conversion from SQL query to functions can make the SQL administrator easy and feel like reports are displayed from the SQL.
2. Although simple replication is supported, Redis cluster is still on progress and we have not implement our approach in Redis cluster so that it can deal for big data as well through cloud computing.
3. Administrating the system so that it can deal with right data to right people.

## 5.3 Limitations

1. The time frame to capture the changed data should be very small as it is intended for real time. So, it is assumed that windows size will be small which captures less data for the operation.
2. It is not intended for batch processing and is not a replacement of ETL.
3. This is a module to address RTBI. It is made in plug and play approach and work together with existing ETL as described in Figure 3.1.
4. MBCFRTR is not for loading, processing and analyzing huge volumes of data, commonly referred to as big data.
5. Does not support traditional SQL queries and functions are written to get the report. So, those who are used to with sql query may feel difference while generating reports.
6. Various database administrator works and access control mechanism which is easy to maintain in RDBMS are not easy to maintain in MBCFRTR component.

# Bibliography

[1] L. Wu, G. Barash, and C. Bartolini, “A service-oriented architecture for business intelligence,” in *Proceedings of the IEEE International Conference on Service-Oriented Computing and Applications*, SOCA ’07, (Washington, DC, USA), pp. 279–285, IEEE Computer Society, 2007. 1

[2] Z. Michalewicz and M. Michalewicz, *Adaptive business intelligence / Zbigniew Michalewicz ... [et al.]*. Springer, Berlin :, 2007. 1, 10

[3] I. Ahmad, S. Azhar, and P. Lukauskis, “Development of a decision support system using data warehousing to assist builders/developers in site selection,” *Automation in Construction*, vol. 13, no. 4, pp. 525 – 542, 2004. 4

[4] Cindi Howson, “Techno babble: Components of a business intelligence architecture.” http://www.b-eye-network.com/view/7105. Components of Business Intelligence Architectures, Mar. 2008. 4, 5, 9, 14

[5] Richard Hackathorn, “The bi watch:real-time to real-value.” DM Review, January. 2004. 5

[6] Z. Panian, “Just-in-time business intelligence and real-time decisioning,” in *Proceedings of the 9th WSEAS international conference on Applied informatics and communications*, AIC’09, (Stevens Point, Wisconsin, USA), pp. 106–111, World Scientific and Engineering Academy and Society (WSEAS), 2009. 5, 6, 10

[7] O. Corporation, “Real-time data integration for data warehousing and operational business intelligence.” Oracle White paper, August. 2010. 5, 31

[8] A. Simitsis, P. Vassiliadis, and T. Sellis, “Optimizing etl processes in data warehouses,” *Data Engineering, International Conference on*, vol. 0, pp. 564–575, 2005. 8

[9] P. V. et al., “Data provenance in etl scenarios.” University of Ioannina. 9

[10] B. Azvine, Z. Cui, D. D. Nauck, and B. Majeed, “Real time business intelligence for the adaptive enterprise,” *E-Commerce Technology, IEEE International Conference on, and Enterprise Computing, E-Commerce, and E-Services, IEEE International Conference on*, vol. 0, p. 29, 2006. 11, 25, 26, 27

[11] M. I. Hwang and H. Xul, “The effect of implementation factors on data warehousing success : An exploratory study,” *Journal of Information, Information Technology, and Organizations*, vol. 2, 2007. 13

[12] M. Golfarelli and S. Rizzi, “Designing the data warehouse: Key steps and crucial issues,” *Journal of Computer Science and Information Management*, vol. 2, 1999. 14, 15

[13] I.-Y. Song and K. LeVan-Shultz, “Data warehouse design for e-commerce environments,” in *Proceedings of the Workshops on Evolution and Change in Data Management, Reverse Engineering in Information Systems, and the World Wide Web and Conceptual Modeling*, ER ’99, (London, UK, UK), pp. 374–387, Springer-Verlag, 1999. 14

[14] M. Golfarelli, D. Maio, and S. Rizzi, “Conceptual design of data warehouses from e/r schemes,” pp. 334–343, 1998. 14

[15] M. Golfarelli and S. Rizzi, “A methodological framework for data warehouse design,” in *Proceedings of the 1st ACM international workshop*

*on Data warehousing and OLAP*, DOLAP '98, (New York, NY, USA), pp. 3–9, ACM, 1998. 14

[16] Microsoft, "Data Warehouse Design Considerations ." http://msdn.microsoft.com/en-us/library/aa902672(v=sql.80).aspx, May. 2012. 14, 16

[17] T. Ariyachandra and H. Watson, "Key organizational factors in data warehouse architecture selection," *Decision Support Systems*, vol. 49, pp. 200–212, May 2010. 17, 18, 19, 20, 21

[18] V. e. a. Lane, Pau;Schupmannl, *Oracle9i Data Warehousing Guide, Release 2 (9.2).* Oracle Corporation, http://docs.oracle.com/cd/B10500_01/server.920/a96520/concept.htm, 2002. 19, 21

[19] D. A. Schneider, "Practical considerations for real-time business intelligence," in *Proceedings of the 1st international conference on Business intelligence for the real-time enterprises*, BIRTE'06, (Berlin, Heidelberg), pp. 1–3, Springer-Verlag, 2007. 20, 21

[20] H. Watson and T. Ariyachandra, "Data warehouse architectures: Factors in the selection decision and the success of the architectures," tech. rep., Terry College of Business, University of Georgia, July 2005. 21

[21] M. Golfarelli, S. Rizzi, and I. Cella, "Beyond data warehousing: what's next in business intelligence?," in *Proceedings of the 7th ACM international workshop on Data warehousing and OLAP*, DOLAP '04, (New York, NY, USA), pp. 1–6, ACM, 2004. 22, 23, 24, 25

[22] N. Stojanovic, L. Stojanovic, D. Anicic, J. Ma, S. Sen, and R. Stühmer, "Semantic complex event reasoning—beyond complex event processing," in *Foundations for the Web of Information and Services* (D. Fensel, ed.), pp. 253–279, Springer Berlin Heidelberg, 2011. 27

[23] Team, The SQLstream, "Concepts in streaming sql, enabling real-time business intelligence and data integration." www.SQLstream.com, 2009. 27, 28

[24] A. e. a. Arasu, "Stream: The stanford data stream management system.," IEEE Data Engineering Bulletin, 26(1), 2003. 29

[25] Rodriguez, Jesus, "Real-time business intelligence with microsoft sql server 2008 r2." http://channel9.msdn.com/Events/TechEd/NorthAmerica/2010/BIE403, June 2010. 30

[26] O. Corporation, "Best practice for real-time data warehousing." Oracle Corporation, World Head Quaters 500 Oracle Parkway, 2010. 32, 33, 34

[27] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Commun. ACM*, vol. 51, pp. 107–113, Jan. 2008. 35

[28] S. Edlich, "Nosql databases." http://www.nosql-database.org, 2011. 35

[29] C. S. (cs134@hdm stuttgart.de), "Nosql databases." 36

[30] amazon.com, "Amazon dynamodb." http://aws.amazon.com/dynamodb/, 2012. 36

[31] G. e. a. DeCandia, "Dynamo: Amazon's highly available key-value store.," pp. 205–220, ACM, 2007. 36

[32] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: a distributed storage system for structured data," in *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation - Volume 7*, OSDI '06, (Berkeley, CA, USA), pp. 15–15, USENIX Association, 2006. 36

[33] "Hbase." http://hbase.apache.org/, 2012. 36, 70

[34] R. Ho, “Pragmatic programming techniques: Bigtable model with cassandra and hbase.” http://horicky.blogspot.com/2010/10/bigtable-model-with-cassandra-and-hbase.html, October. 2010. 37

[35] D. Kellogg, “What’s a column-oriented dbms?.” http://kellblog.com/2007/03/31/whats-a-column-oriented-dbms/, March. 2007. 37

[36] R. Ho, “Bigtable model with cassandra and hbase | javalobby.” http://java.dzone.com/news/bigtable-model-cassandra-and, December. 2010. 38

[37] “The apache cassandra project.” http://cassandra.apache.org/, 2012. 40

[38] P. Lakshman, A.;Malik, “Cassandra: a decentralized structured storage system,” vol. 44, pp. 35–40, ACM SIGOPS Operating Systems Review, 2010. 40

[39] DataStax, “Evolving hadoop into a low-latency data infrastructure.” DataStax, www.datastax.com, 2011. 40, 41

[40] K. Weil, “Rainbird: Real-time analytics @twitter.” http://assets.en.oreilly.com/1/event/55/Realtime Analytics at Twitter Presentation.pdf, 2011. 41, 42

[41] M. Bernard, “Sap high-performance analytic appliance 1.0 (sap hana),” February. 2011. 43

[42] M. G. e. a. Vasu Murthy, “Oracle white paper – oracle exalytics in-memory machine: A brief introduction,” October. 2011. 43

[43] M. Bernard, “Memcached.” http://memcached.org/, 2011. 43

[44] Redis, “Redis.” http://redis.io, April. 2012. 43, 44

[45] T. Macedo and F. Oliveira., *Redis Cookbook*. O’REILLY, 2011. 44

[46] K. Seguin, *The Little Redis Book*. 44

[47] Jonathan Leibiusky, "Jedis." https://github.com/xetorthio/jedis, April. 2012. 45