DEPARTMENT OF COMPUTER SCIENCE

UNIVERSITY OF KAISERSLAUTERN

Master Thesis

Incremental Recomputation of Pig Latin's Nest and Unnest Operators

Rui Oscar Fernandes Miranda

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF KAISERSLAUTERN

Master Thesis

Incremental Recomputation of Pig Latin's Nest and Unnest Operators

Author:	Rui Oscar Fernandes Miranda
1. Supervisor:	Prof. Dr.-Ing. Stefan Deßloch
2. Supervisor:	M. Sc. Yong Hu
Date:	September 28, 2012

Ich versichere hiermit, dass ich die vorliegende Masterarbeit mit dem Thema 'Incremental Recomputation of Pig Latin's Nest and Unnest Operators' selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Die Stellen, die anderen Werken dem Wortlaut oder dem Sinn nach entnommen wurden, habe ich durch die Angabe der Quelle, auch der benutzten Sekundärliteratur, als Entlehnung kenntlich gemacht.

Kaiserslautern, den September 28, 2012 Rui Oscar Fernandes Miranda

Abstract

This master's thesis addresses the maintenance of pre-computed structures, which store a frequent or expensive query, for the nested bag data type in the high level work-flow language Pig Latin. This thesis defines a model suitable to accommodate incremental expressions over nested bags on Pig Latin. Afterwards, the partitioned normal form for sets is extended with further restrictions, in order to accommodate the nested bag model, allow the Pig Latin nest and unnest operators revert each other, and create a suitable environment to the incremental computations. Subsequently, the extended operators – extended union and extended difference – are defined for the nested bag data model with the partitioned normal form for bags (PNF Bag) restriction, and semantics for the extended operators are given. Finally, incremental data propagation expressions are proposed for the nest and unnest operators on the data model proposed with the PNF Bag restriction, and the proof of correctness is given.

Resumo

Esta tese de mestrado aborda a problemática da manutenção de estruturas pré-cumputas, que – regra geral – armazenam uma pesquisa computacionalmente dispendiosa ou frequente, para a linguagem de fluxo de dados de alto nível Pig Latin. Esta tese começa por definir um modelo de dados para acomodar o tipo de dados *nested bag* do Pig Latin. Posteriormente a *partitioned normal form* para *sets* é transformada para poder: acomodar o tipo de dados *nested bag*, permitir que os operadores do Pig Latin nest e unnest possam reverter-se mutuamente, e para criar um ambiente propicio à utilização de expressões incrementais. De seguida os operadores extendidos – *extended union* e *extended difference* – são definidos para o modelo *nested bag* com a restrição *partitioned normal form* para *bags* (PNF Bag), sendo a semântica dos operadores também dada nesta tese. Por fim, são propostas expressões incrementais de propagação da informação para os operadores *nest* e *unnest* no modelo proposto com a restrição PNF Bag.

Resumo Alargado

Esta tese de mestrado aborda a problemática da manutenção de estruturas pré-computadas para o tipo de dados *nested bag* na linguagem de fluxo de dados de alto nível Pig Latin. As estruturas pré-computadas (por exemplo *views* nos sistemas de bases de dados relacionais) tradicionalmente são utilizadas para garantir um menor tempo de resposta dos sistemas as pesquisas dos utilizadores, estas estruturas contem a resposta – total ou parcial – a uma pesquisa frequentemente utilizada ou dispendiosa de computar, sendo garantindo desta forma um menor tempo de resposta do sistema. No entanto, quando os dados que geraram estas estruturas se alteram, estas deixam de conseguir responder consistentemente e com exactidão às questões colocadas, razão pela qual necessitam de ser actualizas. O processo de actualização destas estruturas pré-computadas é denominado por manutenção. Esta manutenção pode ser feita repetindo a pesquisa sobre todos os dados (computação total), ou pode ser feita utilizando a estrutura – que contem uma resposta antiga – e as alterações que ocorreram nos dados (computação incremental). A manutenção de estruturas pré-computadas de forma incremental, geralmente, é executada num período de tempo inferior, utilizando menos recursos do sistema quando comparada com uma computação total. Estas duas vantagens, tempo de resposta e recursos despendidos, tornam a abordagem incremental interessante.

A linguagem Pig Latin destina-se a ser utilizada no processo de pesquisa sobre gigantescos volumes de dados, tendo como principais vantagens a performance e a facilidade de programação. A facilidade de programação refere-se à codificação dos programas ser feita numa sequencia de passos, à semelhança das linguagens procedimentais, e ao mesmo tempo estes passos apresentarem uma sintaxe de uma linguagem de pesquisa (*e.g.* SQL). A performance da Linguem Pig Latin é obtida pela utilização da *framework* de computação de alta performance MapReduce. Um exemplo de uma implementação da linguagem Pig Latin é a *framework* Pig, que opera sobre a *framework* Hadoop (uma *framework* de MapReduce). Os programas em Pig Latin são compilados em primitivas MapReduce, sende executas sobre uma *framework* de MapReduce, sendo desta forma obtida alta performance, sem a necessidade de o utilizador programar programas – para tarefas especificas – a um nível mais baixo, isto é, é utilizada uma linguagem genérica para executar pesquisas difíceis de codificar em MapReduce. A principal alternativa às linguagens declarativas de alto nível é o uso de bases de dados paralelas, no entanto as bases de dados paralelas apresentam elevados custos, podendo os mesmos tornar-se proibitivos à medida que o volume de dados aumenta .

Esta tese começa por apresentar a semântica dos operadores *nest* e *unnest*, já presentes na linguagem Pig Latin destinados à manipulação do tipo de dados *nested bag*. Após a introdução destes operadores é proposto um modelo de dados, o *nested bag* data model, compatível com a linguagem Pig Latin. A principal restrição é a *partitioned normal form* para *bags* (PNF Bag), que é a extensão da *partitioned normal form* para *sets*, uma das contribuições desta tese, que aproxima a estrutura de dados *bag* da estrutura de dados *set*. O modelo de dados e as restrições são tanto para; acomodar o tipo de dados *nested bag*, tornar os operadores *nest* e *unnest* revertíveis mutuamente, bem como tornar possível a utilização de expressões incrementais. De seguida são propostos os operadores extendidos, *extended union* e *extended difference*, que podem suscitante descritos como a adição de *bags* e subtracção de *bags*, sendo também apresenta a semântica dos mesmos. Os operadores extendidos, não

presentes no Pig Latin, são necessários para a computação incremental de estruturas. Adicionalmente é demonstrado em que situações os operadores (*nest*, *unnest*, *extended union* e *extended difference*) podem ser revertidos pelo seu inverso. Os operadores *nest* e *unnest* apenas podem reverter-se mutuamente sem limitações, desde que o tipo de dados das relações sobre os quais são aplicados, se encontrem em conformidade com o modelo proposto com a restrição PNF Bag. No que concerne aos operadores estendidos a irreversibilidade não é possível, podendo sucintamente ser justiçada pela semântica do operador *extended difference* que não permite a existência de elementos com número de ocorrências negativo no modelo. Como consequência desta irreversibilidade as operações de manutenção incremental devem reflectir a mesma ordem que ocorreram nos dados aos quais a estrutura pré-computada diz respeito. Por fim são propostas expressões de manutenção incremental das estruturas pré-computadas geradas pelos operadores *nest* e *unnest*. A opção por uma abordagem equacional em detrimento duma abordagem algorítmica prende-se com o facto de a prova de exactidão das expressões ser mais fácil quando comparada com a abordagem algorítmica. Outra das vantagens da abordagem equacional é a possibilidade de encadeamento de equações de diferentes operadores, o que alem de permitir que as expressões geradas possam ser analisadas por um optimizador de *queries*, e por fim, permitir a adição de novas expressões relativas a novos operadores. Esta última vantagem é de todo impossível na abordagem algorítmica. Não menos importante é a prova, formal que as expressões de manutenção incremental das estruturas pré-computas estão correctas. Sendo desta forma garantido que o resultado obtido por uma computação incremental é o mesmo de uma computação completa para as estruturas que requerem manutenção.

Contents

List of Tables

List of Examples

1 Introduction

This Chapter gives an introduction to the problem in hands on this thesis. Section 1.1 gives an overview of the technology related to this thesis problem. Section 1.2 presents the motivation to write this thesis. Section 1.3 presents the related work, and finally, Section 1.4 presents the structure this thesis.

1.1 Technology Overview

This section presents an overview over the technology related to the emerging problem. Subsection 1.1.1 gives an overview over the MapReduce framework, Subsection 1.1.2 gives an overview over the high level query language Pig Latin and its implementation. Finally, Subsection 1.1.3 gives the overview of the incremental computations to maintain pre-computed data structures ($e.g.$ materialized views on relational databases) and the relational database community approach to the problem.

1.1.1 MapReduce

MapReduce (Dean and Ghemawat, 2008) is a programing model to process large data sets over a cluster of hundreds or thousands of machines. Processing large amounts of data on a cluster arises concerns on how to parallelize custom computation, how to distribute the data and how to handle machine failures. Google introduced MapReduce to handle the problem of creating purpose specific programs to analyze large enormous amounts of data coming from crawled documents, search query logs, etc. and to produce derivate data such as graphs representing documents structure or the queries frequency (Dean and Ghemawat, 2008).

Mapreduce was designed not only to allow the programmers to escape from the complexity of the parallelization, distribution and balance over a cluster, but also to create high-performance, reliable and scalable clusters over non reliable low-end hardware (Dean and Ghemawat, 2008).

On a MapReduce implementation – $e.g.$ Hadoop (The Apache Software Foundation, 2012a (accessed September 13, 2012) - the programmers have to code a Map and a $Reduce$ functions and provide them to the framework:

- The Map function takes as input a pair of <key initial, value initial > and after perform a transformation will generate a list of intermediate pair <key intermediate ,value intermediate > and pass it to the MapReduce framework.
- The $Reduce$ function takes as input the intermediate key and a list intermediate values, and will merge, aggregate, filter or perform other transformations to produce the desired value or list of values.

Table 1.1: Inputs and outputs of Map and Reduce functions

Function	Input		Output
Map	<key A, value A >	→	list(<key B, value B >)
Reduce	<key B, list (value B) >	→	list(value C)

Table 1.1 summarizes the inputs and outputs of Map and $Reduce$ functions.

Under this paradigm the programmer only needs to focus on code Map and $Reduce$ functions, and can leave herself from other concerns such as parallelization, message passing, load balance, decide how workers – and of what kind – are required to perform each task, synchronization, etc.

MapReduce uses a master-slave architecture, where the master node assigns Map or $Reduce$ tasks to the slaves workers. Slave nodes can either run Map or $Reduce$ tasks. A MapReduce job has five main steps:

- Data split: the input data is split to be feed to workers running Map tasks.
- Map: slave workers execute the Map function. Transformations are done to the input and some intermediate <key, value> pairs are is written to local data storage and the master node is notified of such locations.
- Sort: intermediate <key ,value > pairs generated by Map workers are sorted by keys.
- Shuffle: read the relevant intermediate <key ,value > pairs, sharing a common key. A shuffle component is usually located within the worker running $Reduce$ tasks.
- Reduce: slave workers execute the $Reduce$ function. Operations such as aggregation, summarization, filter or further transformation are done. Finally, the result is written into storage.

After the description of the principal execution steps, it becomes obvious why $Reducer$ tasks are to be executed after all the Map tasks finish, otherwise the $Reducer$ tasks would not be able to gather all the intermediate <key,value> pairs – produced by the Map tasks and – required for merge operations.

Consider the example where a MapReduce job is created to analyze a music database containing the information music artist's performances, this is the information on Table 1.2. The MapReduce job has to for the given input identify of the popularity of music genre by country. The information of country where performance occurred and music genre of the artist is not – directly – present on the initial data; hence it will have to be introduced during the Map phase. Finally, during the $Reduce$ phase the intermediate data will be aggregated. The tasks assigned to workers look like this:

- Map: look up where performance location is on the world (identify the country), and look up what the genre of music the artist plays. The Map task uses as output keys the combination music genre and country.

Table 1.2: Music artist's performances

Artist Name	Performance Location
Madonna	Bologna
Green Day	Tokyo
Green Day	Tokyo
Madonna	London
Madonna	Cologne
Michael Jackson	London
Green Day	Osaka

- *Reduce*: count the occurrences for a given combination of genre/country, and output the genre, country and the result of this count function.

Figure 1.1 presents the overview of the execution of this MapReduce job.

As stated before, the MapReduce hardware infrastructure is built over over unreliable low-end hardware; hence fault tolerance is provided. In an occasion when a worker machine fails, the master node can assign the task running on the faulty node to other (idle) node. If the storage[1] of intermediate results fail, the results can be computed by another node and stored in the new node local storage. The final results storage is not likely to fail because they are stored on the global file system. The master node fault tolerance is guaranteed by tracking its state in checkpoints, and in the event of a master node failure a worker machine can be promoted to master and resume the MapReduce job from the last checkpoint.

1.1.2 Pig Latin

MapReduce have a rigid structure of the two phase functions – Map and Reduce – and the a single input data source, this limitation of the architecture might require the programmers to pre-process the input or run multiple MapReduce jobs. Another drawback, to the users, is the requirement to write the custom code used by the Map and Reduce functions; The effort and complexity required to write this code is less than to write a purpose specific full solution. However, the user still has to code functions to perform simple actions such as projection or selection (Olston, Reed, Srivastava, Kumar, and Tomkins, 2008). Pig Latin (Olston, Reed, Srivastava, Kumar, and Tomkins, 2008) was proposed to be used as a high level data flow query language and as a procedural programing language, this is, the language as designed to facilitate the query of the data – having query syntax similar to SQL language . However, the query operations on the low-level are done on MapReduce making possible the manipulation of huge amounts of data with high performance. Pig (The Apache Software Foundation, 2012b (accessed September 13, 2012) framework is an implementation of the Pig Latin language, which runs on top of Hadoop .

A Pig Latin program is declared as a sequence of high level operators, which facilitate human readability, and consequently the effort necessary to produce a program is lower

[1] intermediate results are stored on the local storage of the node assigned to execute the Map task

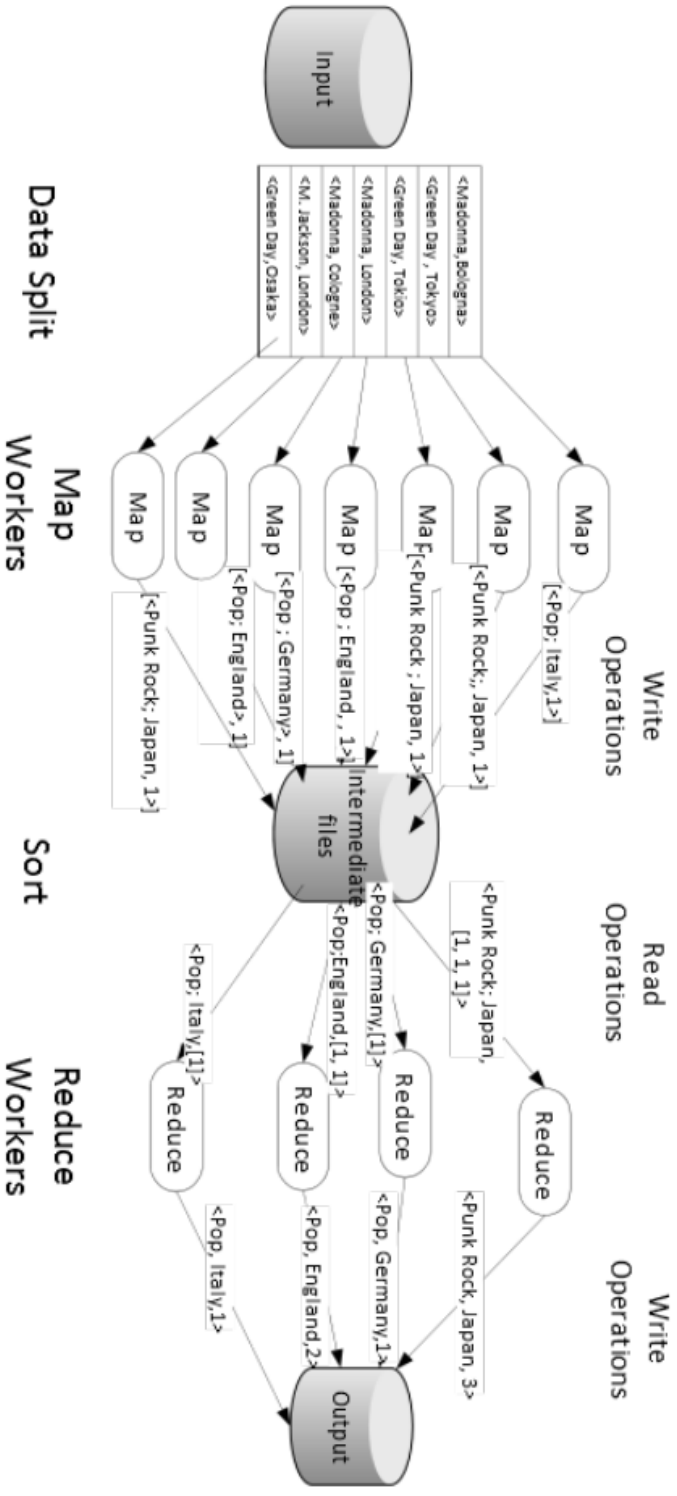

Figure 1.1: MapReduce execution overview

Algorithm 1.1 Pig Latin example code

```
rawData= LOAD 'Table 1.2' using PigStorage(';') AS (name: chararray, city:charray);
cities = GROUP rawData BY city;
concertsPerCity = FOREACH cities GENERATE group, COUNT(rawData.citiy);
STORE concertsPerCity INTO 'output' ;
```

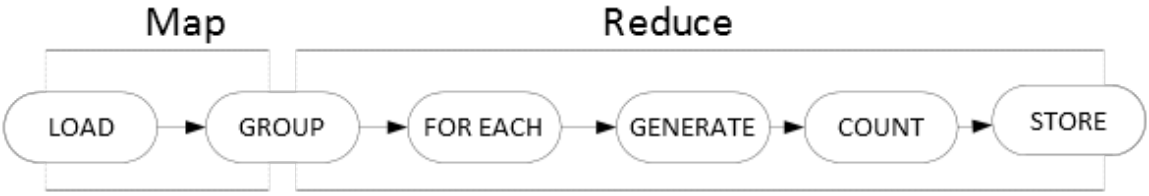

Figure 1.2: PigLatin compilation overview on MapReduce

compared to effort necessary to produce an equivalent program on MapReduce. Comparing Pig Latin with high-performance parallel databases the advantages are obvious: the price of the solution is lower when deployed on large cluster; to users PigLatin *step by step* approach is is easier to create code or scripts than SQL (Olston, Reed, Srivastava, Kumar, and Tomkins, 2008). PigLatin additionally allows the programmers to create their own custom functions – UDF (User Defined Functions)– on the Java programing language, to be used together with the PigLatin constructs extending the query's flexibility and expressiveness. Pig Latin is therefore a powerful language to perform ad-hoc queries over large datasets.

Consider the data present on Table 1.2 and a Pig Latin program, presented on Algorithm 1, to count the number of concerts music artist's gave by city. Algorithm 1 statements are explained bellow:

- The data is loaded and placed on the $rawData$ variable having the attributes *city* and *name* as schema.
- The data is aggregated by the attribute *city* and placed on the variable *cities*.
- The FOREACH .... GENERATE statement iterates over all tuples of the variable *cities* and generates a new variable containing for each *city* the number of occurrences on the variable *cities*.
- Finally, the query is saved into the permanent storage ($e.g.$ the file system).

Pig converts its sequence *step by step* instructions to a lower level Hadoop code, and the program is executed over the Hadoop framework. Figure 1.2 presents the compilation overview of the example program over the MapReduce framework.

Consider the case of a data analyst hugely proficient in the use of spreadsheets, it would be easier to query large amount of data using a high level query language, such as Pig Latin, and export the results to be further used with her spreadsheet software than perform the

same query task coding Map and Reduce functions. This is, coding Map and Reduce functions, in a language such as Java, is less natural than write *step by step* Pig Latin statements. It can be argued that MapReduce give more freedom to the user, it's completely true. However, Pig allows the user to write its own user defined functions (UDF) on the Java programming language to be used together Pig Latin operators. It can also be argued that to perform the queries in as a high-performance parallelized SQL database would be a better solution, this is not entirely true: Pig Latin has the infrastructure price advantage, as Pig Latin runs on top of the MapReduce framework; and the use of SQL not only does not feel very natural to the users, but also write complex SQL query's not an easy task.

1.1.3 Incremental Computations

As the time moves the data also moves, this is, the data is not static it keeps changing. For example, on the Internet based approach the new user generated content is generated every day: search queries, products added or removed from shopping wish lists, millions of sales done every day, new click streams generated, new pictures uploaded to social network sites, comments and 'likes' added to those pictures, etc.

The queries over the data are often repeated over the time. However, the data keeps changing, which explains why the same query is repeated over and over different versions of the data; it is of no use always make the same question to always obtain the same answer, but because of data changes the answer to the same question also changes. This arises the opportunity to merge the valid answer – given by old data – with a new answer – given by the new data –, this is, use an incremental approach to avoid a query over the full data in order to obtain a faster and accurate answer. As long the result obtained is similar to the result obtained by a query over the full data, the solution which delivers a response faster and uses less system resources is preferable. The relational database community had done similar work in the past, with for example, the view maintenance problem (Blakeley, Larson, and Tompa, 1986; Gupta and Mumick, 1995). A view can be defined as the answer or a partial answer to a query, if the view is physically stored – *e.g.* to answer a frequent or an expensive query – it is said to be a materialized view. The use of materialized views, by the relational database community, demonstrated to be a valid approach to reduce the response time of the system. When data on source database tables change, the pre-computed data present on materialized views is no longer consistent; therefore the views require maintenance to be able to accurately answer the incoming queries. Usually only a part of the source data changes; hence these changes can be propagated to the materialized to the view, this is, a incremental update computation. Using the incremental update computation is usually less expensive than a full re-computation; therefore use the the incremental approach to maintain materialized views is desirable (Blakeley, Larson, and Tompa, 1986; Gupta and Mumick, 1995).

The materialized views maintenance can be done using two approaches an algorithmic or equational. Equations, incremental expressions, hold several advantages when compared with maintenance algorithms. Not only the proof of correctness is easy, but most important, the equations can be chained with each other. This is, if new operators are included in the language, new equations have to be derivated for the new operators and the old equations – for the old operators – remain valid; in opposition, on the algorithmic approach a new algorithm have to be purposed. Additionally, the equations can be feed

to a query optimizer which might find a more efficient and equivalent expression (Griffin and Libkin, 1995).

In this thesis the term *pre-computed data structure* or *pre-computed structure* refers to a logical relation which is the answer to a query or a partial query. This answer is physically stored somewhere in file system, and this thesis makes no considerations regarding how the storage should be implemented. Furthermore, every time, an incremental update computation is performed, the state of the *pre-computed structure* is updated to always answer accurately to the last query made.

1.2 Motivation

Cloud computing (Armbrust, Fox, Griffith, Joseph, Katz, Konwinski, Lee, Patterson, Rabkin, Stoica, et al., 2010; Fox, Griffith, et al., 2009) is emerging as solution to provide supercomputer grade services and at same time provide the service at a reasonable price. Cloud computing can be seen as a usage-based pricing, this is a *pay as you go* solution where customers purchase hours of service which can be used non-uniformly in time (Armbrust, Fox, Griffith, Joseph, Katz, Konwinski, Lee, Patterson, Rabkin, Stoica, et al., 2010). (Grossman, 2009). MapReduce is one of these cloud computing services, being offered, for example, by Amazon Elastic Compute Cloud[2]. MapReduce delivers super-computer grade performance, which is desirable feature when it comes to analyze very large amounts of data. Another desirable feature on the data analysis process is the ability the users tell the system what is to be done; and here MapReduce fails, MapReduce requires the user to code the Map and Reduce functions; therefore, making it only usable by people familiar with programing languages. It is, indeed, true that a MapReduce programs complexity is lower than a purpose specific 'old style' distributed program, however, that is not enough. For this reason, the use of a high level data flow language, such as Pig Latin, on top of MapReduce makes this super-computer grade service available to a wide new audience. The use of a data flow programing have several arguments on its favor: makes the programs more intuitive, easy to code, easy to maintain and easy to debug.

The work on thesis focus on Pig Latin because, for the data analysts is easier to use Pig Latin on their daily work than MapReduce (Olston, Reed, Srivastava, Kumar, and Tomkins, 2008). On their jobs data analysts query the data, export the results of the query, and then import these results to be used on spreadsheets or dataminning software. Because Pig, a Pig Latin implementation, runs of top a MapReduce framework, high-performance is also delivered; hence users can create programs and scripts at a higher level and achieve the super-computer grade performance.

Data analysis queries are repeated over the time. However, the base data only changes in the amount of a small portion. Hence, instead of the make the query over the full data, it can be queried only the new data and merge the answer with the old answer previously made. As long the results of the incremental approach are identical to a full computation, does not take longer, and are done in a transparent way (this is, the user does everything

[2] Amazon Elastic MapReduce http://aws.amazon.com/elasticmapreduce/

as she always did, without notice any change), the user has nothing to argue against such solution or even notice any change had occurred. In addition, in most cases, the incremental computation of the queries have two benefits: the speedup of the computation; and a lower resource usage. This is a *win-win situation* for the user whom have to wait less for her results, and the companies which have to pay less for the computation power required.

This thesis focus on the maintenance of incremental computations on Pig Latin, and the equational approach is used. The equational approach was preferred over the algorithmic approach, in the way this thesis only covers the nested data type related operators of Pig Latin; hence, the incremental equations proposed can be used together with other incremental expressions valid for the nested bag data type.

1.3 Related work

Gupta and Mumick (1995) addressed the view maintenance problem, and proposed to the view maintenance problem to be classified under a taxonomy of four dimensions: information dimension, modification dimension, language dimension and instance dimension. Over this space problem they present algorithms developed under the taxonomy proposed.

Albert (1991) work focused on the algebraic properties of the bag data types and how algebraic transformations can be used to query optimization on the bag data type environment. Albert (1991) furthermore demonstrated some of the set algebraic properties fail if applied to bags, however – also demonstrated – that if a bag is result of a boolean selection and the extend operators are applied, query optimization algebraic transformations are valid. A later work from Libkin and Wong (1993) pointed that *'bag language is inadequate in expressive power as it stands'* (Libkin and Wong, 1993) and proposed further primitives to improve bag environment expressive power.
Dayal, Goodman, and Katz (1982) extended the union, intersection and difference to provide control over duplicate elimination. A latter work from Roth, Korth, and Silberschatz (1988) extended the relational algebra and extendeded the basic algebra operators for nested set relations, however the set data model does not allow duplicates. Furthermore, Albert (1991) pointed out that the extended operators on bags preserved the boolean algebra structure. Roth, Korth, and Silberschatz (1988) extended the relational nested set algebra to use the partitioned normal form (PNF) and gave the proof that under PNF on his extended algebra the unnest operator could be reverted by the nest operator. Hulin (1990) have a similar work on restructuring set nested relations on PNF.

Qian and Wiederhold (1991) proposed an algorithm to derive incremental expressions from incremental relations on sets. An improvement to this algorithm was latter proposed by Griffin, Libkin, and Trickey (1997) where the concept of minimality was introduced. Minimality, in the context of incremental expressions, refers to update operations which do not require unnecessary tuples computations – this is, all the tuples on the *to be deleted* set are present on the relation *to be maintained* (*e.g.* a view) and all tuples on the *to be inserted* set neither are present on the relation *to be maintained* or on the *to be deleted* set.

Later, Griffin and Libkin (1995) presented an algorithm to derive incremental expressions from incremental relations to maintain views with duplicates. However, Griffin and Libkin (1995) incremental expressions do not take in consideration the nest and unnest operators. Liu, Vincent, and Mohania (1999) derivated incremental expressions to be used with the nest and unnest operators in nested set relations and PNF nested set relations. Following this work Liu, Vincent, et al. (2003) derivatived incremental expressions for the expansion, projection, selection, intesection, join operators for nested relational model on PNF.

1.4 Thesis structure

This thesis is structured as the follows: on this Chapter it was given an overview of the technologies related to this thesis subject, and it was presented the related work and motivation for this master's work. Chapter 2 presents the nested bag data type model and the restrictions applied to the model. Chapter 3 presents the operators used to manipulate the nested bag data type which are already present on Pig Latin, and the extended operators required for the incremental computations. Chapter 4 introduces the reader to the need of process the incremental computations in the same order as they happened on the base relations and to which – and under what circumstances – operators can be reverted. Additionally, and most important, the data propagation rules are presented and the proof of correctness is given. Finally, on Chapter 5 the contributions of this masters work are summarized and directions to further work are given.

2 Data Model Definition

A bag, or 'multiset', is a container of elements, but unlike a set, the bag might contain duplicates (Albert, 1991). On Pig Latin a bag is a collection of data where the tuples might be characters, numbers, booleans – atomic attributes – or another bags – non atomic attributes. When a bag is contained inside a tuple of another bag, this last bag is a nested bag, this is, a bag with relational attribute on its schema.

This chapter is structured as the follows: Section 2.1 defines the nested bag data model to be used on this thesis, Section 2.2 gives the definition of the partitioned normal form (PNF), and finally, on Section 2.3 the partitioned normal form for bags (PNF Bag) is defined.

The use of the nested bag model and the PNF Bag, defined on this chapter, are the two main assumptions used on the remainder of this thesis.

2.1 Nested Bag Model

A nested relational schema R is a collection of rules and attributes having the form $R = (A_1, ..., A_n, R_1^*, ..., R_q^*)$, where the number of atomic attributes, A_i, is non-negative $(n \geq 0)$, the number of relational attributes, R_j^*, is non-negative $(q \geq 0)$, and the schema contains at least one attribute $(q + n \geq 1)$. Each relational attribute has it own sub-schema, this means the relational attributes are also sub-relations.

Example 2.1 (The music database schema)
*Consider the example where a music database contains the name of a music artist (*Artist Name*), the music genre (*Genre*) of this artist – to not generate a very complex example, each artist has only one music genre –, the artist's tour information (*Tour*), and the published albums (*Records*). The schema for this example contains the following rules:*

$$Music\ DB = (Artist\ Name, Genre, Tour^*, Records^*)$$
$$Tour = (Country, Performance^*)$$
$$Performance = (City, Date)$$
$$Records = (Album, Year)$$

On this example Music DB, Tour, Performance *and* Records *are relational attributes, and all the other fields* – Artist Name, Genre, Country, City, Date, Album *and* Year – *are atomic attributes.*

The relational attributes are also sub-schemes, for example Tour *is a relational attribute on the schema* Music DB, *and it has it owns scheme that is:*

$$Tour = (Country, Performance)$$

Table 2.1: A nested relation over the Music DB schema

Music DB						
Artist N.	Genre	Tour			Records	
		Country	Performance		Year	Album
			City	Date		
Green Day	Punk Rock	Japan	Tokyo	16 Aug 12	Kerplunk	1992
			Tokyo	18 Aug 12	Dookie	1994
			Osaka	19 Aug 12	Insomniac	1995
		England	London	23 Aug 12	Nimrod	1997
		Germany	M. Gladbach	29 Aug 12	Warning	2000
			Berlin	30 Aug 12	American Idiot	2004
			Berlin	30 Aug 12	American Idiot	2004
			Konstanz	01 Sep 12	¡Uno!	2012
Madonna	Pop	Portugal	Coimbra	24 Jun 12	Like a Prayer	1989
		Germany	Cologne	10 Jul 12	MDNA	2012
		Norway	Oslo	15 Aug 12	Music	2000
M. Jackson	Pop	England	London	28 Aug 09	Thriller	1982
			London	13 Jul 09		
			London	06 Mar 10		
			London	06 Mar 10		

A database instance r over the schema R ($R = \{A_1, ..., A_n, R_1^*, ..., R_q^*\}, n \geq 0, q \geq 0, n + q \geq 1$), is a finite set if of elements over R. A tuple t of a relation r, is composed by atomic attributes ($a_i \in \{A_1, ..., A_n\}$) and/or instances of the relational attributes ($r_i^* \in \{R_1^*, ..., R_q^*\}$), this is:

$$R = schema(r) = \{A_1, ..., A_n, R_1^*, ..., R_q^*\}, n \geq 0, q \geq 0, n + q > 0$$
$$t \in r : t = (a_1, ..., a_n, r_1^*, ..., r_q^*), a_i \in \{A_1, ..., A_n\}, r_i^* \in \{R_1^*, ..., R_n^*\}$$

Example 2.2 (The music database instances)

On the Table 2.1 is presented an database instance of the Music DB *schema introduced on Example 2.1.*
A tuple from this relation can be for example:

$$tuple = \left(M.Jackson, Pop, \left\{ England, \left\{ \begin{array}{l} (London, 28Aug09) \\ (London, 13Jul09) \\ (London, 06March10) \end{array} \right\} \right\}, \{(\text{Thriller}, 1982)\} \right)$$

The nesting level refers to the maximum number of sub-schemes a relation contains. The nesting level should be evaluated using the above rules:

- Atomic attributes are evaluated as 0.

- Relational attributes are evaluated as 1 incremented to their own level, this means the nesting level definition should be applied recursively.
- The highest value obtained on the above rules is the nesting level.

Example 2.3 (The music database nesting level)

For the Music DB *schema the level of the schema can be evaluated as 2, which is comes from the value of the highest level sub-relation,* $Tour$*, plus 1:*

$$level = \max\{\ Atomic\ attributes\ level, Tour\ level, Records\ level\} \Leftrightarrow$$
$$level = \max\{0, 1+1, 0+1\} = \max\{0, 2, 1\} = 2$$

$$Tour\ level = \max\{0, Performance\ level\} = \max\{0, 0+1\} = \max\{0, 1\} = 1$$
$$Performance\ level = \max\{City, Date\} = \max\{0, 0\} = 0$$
$$Records\ level = \max\{Album, Year\} = \max\{0, 0\} = 0$$

Let f be a function function that counts the occurrence of each an element, tuple, in a relation r. The outcome of the count function must always generate a non negative number, this is:

$$t \in r, \mathbb{N}^0 = \mathbb{N} \cup \{0\}$$
$$f_r\ (t) \to \mathbb{N}^0$$

Example 2.4 (The count function)
Let r *be a relation over the schema :*

$$schema(r) = (Artist\_Name, Tour^*)$$
$$Tour = (Country, CityA^*)$$
$$CityA = (CityB)$$

The relation r *is presented on Table 2.2 and above are illustrated the evaluations of some tuples the using the count function over the relation* r*.*

$f_r\ (Green\_Day, (Japan, \{(Tokyo)\})) = 2$
$f_r\ (Green\_Day, (Japan, \{(Osaka)\})) = 1$
$f_r\ (Madonna, (Japan, \{(Osaka)\})) = 0$, *this tuple does not belong to the relation*
$$f_r\left(M.Jackson, \left(England, \left\{\begin{array}{c}(London)\\(London)\\(London)\end{array}\right\}\right)\right) = 1$$
$$f_r\left(M.Jackson, \left(England, \left\{\begin{array}{c}(London)\\(London)\end{array}\right\}\right)\right) = 0,\ \textit{this tuple does not belong to the relation}$$
$f_r\ (Green\_Day) = 0$, *this tuple schema does not match with* r *schema*
$f_r\ (M.Jackson, (England, \{(London)\})) = 0$, *this tuple does not belong to the relation*

Table 2.2: Relation to be used on the count example

Artist Name	Tour	
	Country	CityA
		CityB
Green Day	Japan	Tokyo
Green Day	Japan	Tokyo
Green Day	Japan	Osaka
Green Day	England	London
Green Day	Germany	M. Gladbach
		Berlin
		Konstanz
Madonna	Portugal	Coimbra
Madonna	Germany	Cologne
Madona	Norway	Oslo
M. Jackson	England	London
		London
		London

Now let t *be a tuple from* r *such that:*

$$t = \left(M.\ Jackson, \left(England, \left\{ \begin{array}{l} (London) \\ (London) \\ (London) \end{array} \right\} \right) \right)$$

And let t′ *be a tuple* $\in t\,[Tour^*]$*, this is:*

$$t' = \left(England, \left\{ \begin{array}{l} (London) \\ (London) \\ (London) \end{array} \right\} \right)$$

Underneath is presented the evaluation of the count function for the relational attribute (sub-relation or sub-bag) CityA*:*

$f_{t[CityA^*]}\,(London) = 3$
$f_{t[CityA^*]}\,(Paris) = 0$*, this tuple does not belong to the relation*
$f_{t[CityA^*]}\left(\left\{ \begin{array}{l} London \\ London \\ London \end{array} \right\} \right) = 0$*, the count function takes a tuple as argument, and a bag was provided*

The nested bag data type can be defined as:

- *r* is a relation over a nested relation schema *R* ($R = \left\{ A_1, ..., A_n, R_1^*, ..., R_q^* \right\}, n \geq 0, q \geq 0, n + q \geq 1$), where A_i is a atomic attribute and R_j^* is a relational attribute.

- if a tuple belongs to the relation, the result of the count function is always positive, otherwise will be zero:
  1. $\forall t \in r, f_r(t) > 0$
  2. $\forall t \notin r, f_r(t) = 0$

2.2 Partition Normal Form (PNF)

The partitioned normal form (PNF) (Roth et al., 1988) is required for on the set data model the unnest operations can be reverted by a nest operation.

The partitioned normal form (PNF) relies on functional dependency; hence, after an introduction to the nest and unnest operators, the functional dependency definition will be given, and afterwards, the PNF definition will be given.

Nest and Unnest operations

The nest and unnest operators are already present on Pig Latin, they aim at the manipulation of the nested bag data type. The nest operator takes a relation and generates a new relation by aggregating the tuples sharing the same atomic nesting keys into a new tuple containing the nest keys as atomic attributes and the remainder attributes, of the initial tuples, inside a *new* relational attribute. The unnest operator have the reverse purpose of the nest operator, this is, for a given relation the relational attributes are to be flattened. In other words, the unnest operator takes a nested relation and generates a new relation, where, for each tuple of the initial relation, the relational attribute to be unnested is converted into multiple tuples and the remainder attributes of the initial tuple are remain the same. Example 2.5 introduces the use of nest and unnest over of two relations.

Example 2.5 (Nest and unnest)

Let r *be the relation present on Table 2.4 having the flat schema:*

$$schema(r) = (Artist\ Name, Country, City)$$

The relation r *nested by the atomic attribute Artist Name as key generates the relation* s *present on Table 2.3 .*
As it is possible to state on Table 2.3, the tuples were aggregated by Artist Name*, and two tuples were generated having on the atomic attribute the values* Green Day *and* Madonna*. The remainder attributes were 'placed' on the new relational attribute* Performance*, and the dependency they had with the* Artist Name *was preserved.*

Now let s*, Table 2.3, have the schema:*

$$schema(s) = (ArtistName, Performance^*)$$
$$Performance = (Country, City)$$

Unnesting the relation s *by the relational attribute* Performance *generates the relation* r*. As it is possible to state on Table 2.4, the relation was 'flattened'.*

Functional dependency

On the nested set model exists functional dependency between atomic attributes, and relation attributes *iff*, for all tuples of a relation *r* over a schema *R* ($R = (A_1, ..., A_n, R_1^*, ..., R_q^*), n \geq 0, q \geq 0, q+n \geq 1$), there are no two (or more) tuples, containing the same atomic attributes ($t[A_1, ..., A_n]$) and containing a different versions of the relational attribute, this is:

Table 2.3: Relation nested (Example 2.5)

Music DB		
Artist Name	Performance	
	Country	City
Green Day	Japan	Tokyo
	Japan	Osaka
	England	London
Madonna	Paris	France
	England	London

.

Table 2.4: Relation unnested (Example 2.5)

Music DB		
Artist Name	Country	City
Green Day	Japan	Tokyo
Green Day	Japan	Osaka
Green Day	England	London
Madonna	Paris	France
Madonna	England	London

$$\forall t \in r, if \exists t' : t\left[A_1, ..., A_n\right] = t'\left[A_1, ..., A_n\right] \Rightarrow t\left[A_1, ..., A_n, R_1^*, ..., R_q^*\right] = t'\left[A_1, ..., A_n, R_1^*, ..., R_q^*\right]$$

Example 2.6 (The music database functional dependency)

Consider the Music DB *instance presented on Table 2.5, the functional dependency is not fulfilled for this nested bag. There exists two tuples, whose atomic attributes for the* Artist Name *and* Genre *– M. Jackson and* Pop *respectively – which can refer to different instances of the relational attribute* Records*; hence on this instance the functional dependency is not preserved, this is:*

$$t = \left(M.Jackson, Pop, \left\{ England, \left\{ \begin{array}{l} (London, 28Aug09) \\ (London, 14Jul09) \\ (London, 06Mar10) \end{array} \right\} \right\}, \{(Thriller, 1982)\} \right)$$

$$t' = \left(M.Jackson, Pop, \left\{ England, \left\{ \begin{array}{l} (London, 28Aug09) \\ (London, 14Jul09) \\ (London, 06Mar10) \end{array} \right\} \right\}, \left\{ \begin{array}{l} (Music\&Me, 1973) \\ (Bad, 1987) \\ (Dangerous, 1991) \end{array} \right\} \right)$$

$$t\left[A_{Artist\ Name}, A_{Genre}\right] \nrightarrow t\left[R^*\right] \Leftrightarrow$$

$$t\left[A_{Artist\ Name}, A_{Genre}\right] = t'\left[A_{ArtisttextName}, A_{Genre}\right] \wedge t\left[R^*_{Records}\right] \neq t'\left[R^*_{Records}\right]$$

Table 2.5: Violation of the functional dependency

Music DB						
Artist N.	Genre	Tour			Records	
		Country	Performance		Year	Album
			City	Date		
M. Jackson	Pop	England	London	14 Jul 09	Thriller	1982
			London	14 Jul 09		
			London	06 Mar 10		
M. Jackson	Pop	England	London	28 Aug 09	Music & Me	1973
			London	14 Jul 09	Bad	1987
			London	06 Mar 10	Dangerous	1991

Partitioned normal form (PNF)

A relation r, over a schema R ($R = (A_1, ..., A_n, R_1^*, ..., R_q^*), n > 0, q > 0, q + n \geq 1$) on the partitioned normal form (PNF) (Roth, Korth, and Silberschatz, 1988), *iff*:

1. There is a functional dependency between the atomic attributes and the relational attributes:
$$\forall t \in r, if \exists t' : t\left[A_1, ..., A_n\right] = t'\left[A_1, ..., A_n\right] \Rightarrow$$
$$t\left[A_1, ..., A_n, R_1^*, ..., R_q^*\right] = t'\left[A_1, ..., A_n, R_1^*, ..., R_q^*\right]$$
2. For every tuple, all relational attributes – subrelations – are also in PNF:
$$\forall t \in r \wedge \forall r' \in \left\{t\left[R_1^*\right], ..., t\left[R_q^*\right]\right\}, \text{r' is also on PNF}$$
3. Tuples containing only atomic attributes are on PNF:
$$\forall t \in r, if \exists t' : t\left[A_1, ..., A_n\right] = t'\left[A_1, ..., A_n\right] \Rightarrow$$
$$t\left[A_1, ..., A_n\right] = t'\left[A_1, ..., A_n\right]$$

Example 2.7 (The music database PNF)

The relation example presented on Table 2.5 is not on PNF because there is no functional dependency – explained on Example 2.6 –, conversely the example presented on Table 2.6 is on PNF.

2.3 Bag Partitioned Normal Form (PNF Bag)

The partitioned normal form (PNF) is intent to be used on the set environment. However, the bag environment has different characteristics of the set environment, namely the bag allows duplicates. The partitioned normal form for bags (PNF Bag) is proposed as an extension of the PNF with further restrictions. With such restriction the unnest and nest operators can revert each other on the nested bag model with the PNF Bag restriction.

A relation r, over a schema R ($R = (A_1, ..., A_n, R_1^*, ..., R_q^*), n \geq 0, q \geq 0, q + n \geq 1$) on PNF Bag, iff:

Table 2.6: Relation on the partitioned normal form

Music DB						
Artist N.	Genre	Tour			Records	
		Country	Performance		Year	Album
			City	Date		
M. Jackson	Pop	England	London	28 Aug 09	Music & Me	1973
			London	14 Jul 09	Bad	1987
			London	06 Mar 10	Dangerous	1991
M. Jackson	Pop	England	London	28 Aug 09	Music & Me	1973
			London	14 Jul 09	Bad	1987
			London	06 Mar 10	Dangerous	1991

- The relation fulfills PNF.
- The outcome of the count function must be evaluated as zero or one for tuples having relational attributes, this is, the tuple might or might not belong to the relation and no duplicates are allowed for tuples having relational attributes: $if(q > 0) \rightarrow \forall t \in r, f(t) = 1$
- The outcome of the count function must be evaluated as non-negative for tuples having only atomic attributes, this is, duplicates are allowed for tuples without relational attributes: $if\ (q = 0) \rightarrow \forall t \in r, f(t) > 0$
- For every tuple, all it's relational attributes – subrelations – are also on PNF bag:

$$\forall t \in r, \forall r' \left\{ t\left[R_1^*\right], ..., t\left[R_q^*\right] \right\} \text{ , } r' \text{ is on PNF Bag}$$

Example 2.8 (PNF Bag example)
Consider two relations r *and* s *having the same schema that is:*

$$schema(r) = schema(s) = (Artist\ Name, Performance^*)$$
$$Performance = (Country, City)$$

Let r *be defined as :* r = $\{(Madonna, \{(England, London)\}), (Madonna, \{(England, London)\})\}$
r is on PNF, however is not on PNF bag. The count evaluation of the tuple $(Madonna, \{(England, London)\})$ *is higher than one; hence the relation r is not on PNF bag.*

$$f_r\ (Madonna, \{(England, London)\}) = 2$$

In opposition, let s *be defined as:* s = $\{(Madonna, \{(England, London), (England, London)\})\}$.
On s*, the count function evaluates all tuples – belonging to the relation – containing relational attributes as one, conversely tuples having exclusively atomic attributes can be evaluated with values higher than one.*

For example consider t *be the tuple from* s *having the atomic attribute* Madonna*, now let* t' *be a tuple extracted from the relational attribute* Performance *from* t *having as atomic attributes the values* England *and* London*. The count function evaluates this tuple as two, and because this*

tuple has only atomic attributes, the result of the evaluation of count functions is allowed to be evaluated a value higher than one. This is, s *fulfills PNF Bag:*

$$t \in s \wedge t = (Madonna, \{(England, London), (England, London)\})$$
$$t' \in t\,[Performance^*] \wedge t' = (England, London)$$
$$f_s\,(t) = 1$$
$$f_{t[Performance^*]}\,(t') = 2$$

3 Data Operation Models

This chapter presents and gives the formal definition of the Pig Latin operators that manipulate nested relations, this is nest and unnest operators. Additionally, the extended operators, required to perform incremental computations, are defined and formal definition is provided. All the operators definitions are given over the nested bag data model with the partitioned normal form for bags (PNF Bag) restriction.

This chapter is structured as the follows, section 3.1 presents the Pig Latin operators, and section 3.2 presents the extended operators.

3.1 Pig Latin Operators

This section describe, and provide a formal definition of, nest and unnest operators implemented and present on Pig Latin. Table 3.1 gives an overview of the nest and unnest operators.

Subsection 3.1.1 presents the nest operator and finally, Subsection 3.1.2 presents the unnest operator.

3.1.1 Nest Operator

The nest operator generates a new relation, from another already existent, by grouping attributes into a new relational attribute sharing common nesting keys – which must be atomic attributes. On Pig Latin the nest operator, represented as n, is implemented by the GROUP operator.

Let r be a relation having the schema: $schema\,(r) = \left\{A_1, .., A_n, R_1^*, ..., R_q^*\right\}, n > 0, q \geq 0$. The nesting of a finite number of atomic attributes ω belonging to r, this is, $\omega = \{A_1, ..., A_m\} \subseteq \{A_1, .., A_n\}$ and $m \leq n$, can be formally represented as:

(3.1)

Table 3.1: Pig Latin operators overview

Operator	Symbol	Pig Latin
Nest	$n_\omega\,(r)$	GROUP
Unnest	$\mu_\omega\,(r)$	FLATTEN

Table 3.2: Relation r to be nested (Example 3.1)

Music DB			
Artist N.	Country	City	Date
Green Day	Japan	Tokyo	16 Aug 12
Green Day	Japan	Tokyo	18 Aug 12
Green Day	Japan	Osaka	19 Aug 12
Green Day	England	London	23 Aug 12
Green Day	Germany	M. Gladbach	29 Aug 12
Green Day	Germany	Berlin	30 Aug 12
Green Day	Germany	Konstanz	01 Sep 12
Madonna	Portugal	Coimbra	24 Jun 12
Madonna	Germany	Cologne	10 Jul 12
Madonna	Norway	Oslo	15 Aug 12
M. Jackson	England	London	28 Aug 09
M. Jackson	England	London	13 Jul 09
M. Jackson	England	London	06 Mar 10

$$n_\omega(r) = \left\{ \begin{array}{l} t | \exists u \in r, (t[A_1, ...A_m] = r[A_1, ..., A_m]) \wedge \\ R_1^* = \left\{ \begin{array}{l} t' | t'[A_{m+1}, ..., A_n] = u[A_{m+1}, ..., A_n] \wedge \\ t'[R_1^*, ..., R_q^*] = u[R_1^*, ..., R_q^*] \end{array} \right\} \end{array} \right\} =$$

$$= \left\{ \begin{array}{l} t | \exists u \in r, (t[\omega] = r[\omega]) \wedge \\ R_1^* = \left\{ \begin{array}{l} t' | t'[\{A_1, .., A_n\} - \omega] = u[\{A_1, .., A_n\} - \omega] \wedge \\ t'[R_1^*, ..., R_q^*] = u[R_1^*, ..., R_q^*] \end{array} \right\} \end{array} \right\}$$

- If r is on PNF Bag, then $n_\omega(r)$ is also on PNF Bag.

Example 3.1 (Nesting example)

Let r *be the relation defined on Table 3.2, having the flat schema:*

$$schema(r) = (Artist\ Name, Country, City, Date)$$

A new relation v *generated from nesting the* r *by* Artist Name, Country, *this is:* $n_{Artist\ Name, Country}(r)$, *is presented on Table 3.3.*
The relation v, *can be further nested, if* v *is nested by* Artist Name, *this is:* $n_{Artist}(n_{ArtistName, Country}(r))$, *this relation is presented on Table 3.4.*

It is obvious that Table 3.4 have better redeability and takes less space to store than Table 3.2.

Notes regarding the nest definition

This definition differs from the real Pig implementation, in the way the nest operator places all the attributes (including the nesting keys) on the relational attribute of the generation relation, this is:

Table 3.3: s= $\mu_{Tour}(r) = \mu_{Tour}(Table3.4)$

Music DB			
Artist N.	Country	Performance	
		City	Date
Green Day	Japan	Tokyo	16 Aug 12
		Tokyo	18 Aug 12
		Osaka	19 Aug 12
Green Day	England	London	23 Aug 12
Green Day	Germany	M. Gladbach	29 Aug 12
		Berlin	30 Aug 12
		Konstanz	01 Sep 12
Madonna	Portugal	Coimbra	24 Jun 12
Madonna	Germany	Cologne	10 Jul 12
Madonna	Norway	Oslo	15 Aug 12
M. Jackson	England	London	28 Aug 09
		London	13 Jul 09
		London	06 Mar 10

$$n'_{\omega}(r) = \left\{ \begin{array}{l} t|\exists u \in r\,(t[\omega] = r[\omega]) \wedge \\ R^*_1 = \left\{t'|t'\left[A_1,..,A_n,R^*_1,...,R^*_q\right] = u\left[A_1,..,A_n,R^*_1,...,R^*_q\right]\right\} \end{array} \right\}$$

The placement of redundant atomic values on the relational attribute is unnecessary; therefore on the nest operator definition they are omitted.

3.1.2 Unnest Operator

The unnest operator generates a new relation, by flattening a relational attribute of a existing relation. The unnest operator, represented as μ, on Pig Latin is implemented by the FLATTEN function.

Let r be a relation having the schema: $schema(r) = \left\{A_1,..,A_n,R^*_1,...,R^*_q\right\}, n \geq 0, q > 0$, the unnest of an relational attribute ω belonging to the relation relational attributes, this is $\omega \in \left\{R^*_1,...,R^*_q\right\}$ and $q \geq 1$, can be formally represented as:

(3.2)

$$\begin{array}{l} \mu_{\omega}(r) = \\ \{t|\exists u \in r, t[A_1,...,A_n] = u[A_1,...,A_n]\} \text{ , if } r \text{ is flat} \\ \left\{ \begin{array}{l} t|\exists u \in r, t[A_1,...,A_n] = u[A_1,...,A_n], \\ \\ \left(t\left[\left\{R^*_1,...,R^*_q\right\} - \{\omega\}\right] = u\left[\left\{R^*_1,...,R^*_q\right\} - \{\omega\}\right]\right) \wedge \\ \left(t\left[A_{i1},...,A_{in},R^*_{j1},...,R^*_{jq}\right] \in u[\omega]\right) \end{array} \right\} \text{ , if } r \text{ is nested} \end{array}$$

Table 3.4: Relation *r* to be unnested (Example 3.2)

Music DB			
Artist N.	Tour		
	Country	Performance	
		City	Date
Green Day	Japan	Tokyo	16 Aug 12
		Tokyo	18 Aug 12
		Osaka	19 Aug 12
	England	London	23 Aug 12
	Germany	M. Gladbach	29 Aug 12
		Berlin	30 Aug 12
		Konstanz	01 Sep 12
Madonna	Portugal	Coimbra	24 Jun 12
	Germany	Cologne	10 Jul 12
	Norway	Oslo	15 Aug 12
M. Jackson	England	London	28 Aug 09
		London	13 Jul 09
		London	06 Mar 10

- If r is on PNF Bag, then $\mu_{\omega}(r)$ is also on PNF Bag.

Example 3.2 (Unnest example) *Let* r *be the relation present on Table 3.4, having the schema:*

$$schema(r) = (ArtistName, Tour^*)$$
$$Tour = (Country, Performance^*)$$
$$Performance = (City, Date)$$

A new relation v *generated from unnesting the relational attribute* Tour*, this is,* $\mu_{Tour}(r)$*. Relation* v *is presented on Table 3.3 and has the following schema:*

$$schema(v) = (ArtistName, Country, Performance^*)$$
$$Performance = (City, Date)$$

v *can be further unnested, this time by the* Performance *relational attribute generating the flat relation* s. s *is presented on Table 3.2 and has the following schema:*

$$schema(s) = (ArtistName, Country, City, Date)$$

3.2 Extended Operators For Nested Bags

This section presents, and give formal definition of, the extended operators for bags. These operators are not present on Pig Latin however, they are required to perform incremental computations. The extended operators for nested bags are different from the extended operators for nested sets, present on (Roth, Korth, and Silberschatz, 1988), in the way sets and bags have different semantics. Therefore, on this section the extended operators are

Table 3.5: Extended operators overview

Operator	Symbol
Extended Union	$r \oplus s$
Extended Difference	$r \ominus s$

redefined for the nested bag model with the partition normal form for bags (PNF Bag) restriction.

Table 3.5 presents an overview of the extended operators, presented in this this chapter as the follows: Subsection 3.2.1 presents the extended union, and Subsection 3.2.2 presents the extended difference.

3.2.1 Extended Union

The extended union is an operator which takes two relations over the same schema and generates a new relation under the same schema. The extended union uses, all, the relation atomic attributes as *union keys* and, recursively, performs the extended union of the relational attributes. If the two relations being *united* under the extended union operator are on PNF Bag, the new relation will also be on PNF Bag. On a more formal way, the extended union can be described as:

Let r and s be two nested bag relations sharing the same schema, this is, $schema(r) = schema(s) = \left\{A_1, ..., A_n, R_1^*, ..., R_q^*\right\}, n \geq 0, q \geq 0, n+q > 0$, where A_i are atomic attributes and R_j are relational attributes.

The extended union, represented by the $\oplus$ operator, can be formally defined as:

(3.3)

$$
\begin{array}{l}
r \oplus s = \\
\left\{t \middle| \left(t \in r \vee t \in s\right) \wedge f_{r \oplus s}\left(t\right) = f_r\left(t \in r\right) + f_s\left(t \in s\right)\right\} \text{ , if } r \text{ and } s \text{ are flat} \\
\left\{
\begin{array}{l}
t \middle| \left(\exists u \in r, \exists v \in s\right) \\
\left(\begin{array}{l}
\left(t[A_1, ..., A_n] = u[A_1, ..., A_n] = v[A_1, ..., A_n]\right) \wedge \\
\left(t\left[R_1^*\right] = u[R_1^*] \oplus v[R_1^*], ..., t\left[R_q^*\right]\right)
\end{array}\right) \vee \\
\left(\begin{array}{l}
\left(t[A_1, ..., A_n] = u[A_1, ..., A_n] \neq v[A_1, ..., A_n]\right) \wedge \\
\left(t\left[R_1^*\right] = u[R_1^*], ..., t\left[R_q^*\right] = u[R_q^*]\right)
\end{array}\right) \vee \\
\left(\begin{array}{l}
\left(t[A_1, ..., A_n] = v[A_1, ..., A_n] \neq u[A_1, ..., A_n]\right) \wedge \\
\left(t\left[R_1^*\right] = v[R_1^*], ..., t\left[R_q^*\right] = v[R_q^*]\right)
\end{array}\right)
\end{array}
\right\} \text{ , if } r \text{ and } s \text{ are nested relations}
\end{array}
$$

- If r and s are on PNF Bag, then $r \oplus s$ is also on PNF Bag.

The above extended union definition, can be described as:

- A tuple, belonging to the relation on the left side of the operator, with no match match on the second relation, will belong to the new relation.

Table 3.6: A nested relation over the Music DB schema

Music DB						
Artist N.	Genre	Tour			Records	
		Country	Performance		Year	Album
			City	Date		
Green Day	Punk Rock	Japan	Tokyo	16 Aug 12	Kerplunk	1992
			Tokyo	18 Aug 12	Dookie	1994
			Osaka	19 Aug 12	Insomniac	1995
		England	London	23 Aug 12	Nimrod	1997
		Germany	M. Gladbach	29 Aug 12	Warning	2000
			Berlin	30 Aug 12	American Idiot	2004
			Konstanz	01 Sep 12	¡Uno!	2012
Madonna	Pop	Portugal	Coimbra	24 Jun 12	Like a Prayer	1989
		Germany	Cologne	10 Jul 12	MDNA	2012
		Norway	Oslo	15 Aug 12	Music	2000
M. Jackson	Pop	England	London	28 Aug 09	Thriller	1982
			London	13 Jul 09		
			London	06 Mar 10		

- A tuple, belonging to the relation on the right side of the operator, with no match match on the left side of the operator, will belong to the new relation.
- A tuple belonging to the relation on left side of the operator, with no relational attributes and with a match on the second relation, the generated tuple it will be subject to the outcome of the addition of the count function for both relations.
- A tuple belonging to the relation on the left side of the operator, with relational attributes and with a match on the second relation, the generated tuple it will be subject to the outcome of the extended union of the relational attributes of both relations (applied recursively).
- The generate relation is on PNF Bag, if the both base relations are also on PNF Bag.

Example 3.3 (Extended union)

Let r *be the relation defined on Table 3.6, and the* s *the relation defined on Table 3.7. Both* r *and* s *share the same the* Music DB *schema, which is:*

$$Music\ DB = (Artist\ Name, Genre, Tour^*, Records^*)$$
$$Tour = (Country, Performance^*)$$
$$Performance = (City, Date)$$
$$Records = (Album, Year)$$

Let v *be the resultant relation from the extended union of* r *and* s*, this is:* r $\oplus$ s = v. *Table 3.8 presents the relation* v.

Table 3.7: s relation over the Music DB schema containing tuples to be inserted

Music DB						
Artist N.	Genre	Tour			Records	
		Country	Performance		Year	Album
			City	Date		
Green Day	Punk Rock	England	Reading	25 Aug 12	39/Smooth	1990
		USA	Las Vegas	21 Sep 12		
The Cranberries	Rock	Germany	Berlin	08 Oct 12	Roses	2012
M. Jackson	Pop	England	London	09 Jan 10	$\emptyset$	
			London	12 Jan 10		

3.2.2 Extended Difference

The extended difference is an operator takes two relations over the same schema and generates a new relation under the same schema. The new relation is generated from the tuples present on the relation on the left side of the extended operator, *deducted* from the tuples present on the relation on the right side of the extended operator operator. The extended difference uses the atomic attributes – as keys to match tuples – and, recursively, performs the extended difference of the relational attributes. The extended difference, represented by the $\ominus$ operator, can be formally defined as:

Let r and s be two nested bag relations sharing the same schema, which is $schema(r) = schema(s) = \left\{A_1, ..., A_n, R_1^*, ..., R_q^*\right\}, n \geq 0, q \geq 0, n+q > 0$, where A_i are atomic attributes and R_j are relational attributes.

(3.4)

$$
\begin{array}{l}
r \ominus s = \\
\left\{t|\left(t \in r \vee t \in s\right) \wedge f_{r \ominus s}\left(t\right) = \max\left\{0, f_r\left(t \in r\right) - f_s\left(t \in s\right)\right\}\right\} \text{ , if } r \text{ and } s \text{ are flat} \\
\left\{
\begin{array}{l}
t|\left(\exists u \in r, \exists v \in s\right) \\
\left(
\begin{array}{l}
\left(t[A_1, ..., A_n] = u[A_1, ..., A_n] = v[A_1, ..., A_n]\right) \wedge \\
\left(t\left[R_1^*\right] = u[R_1^*] \ominus v[R_1^*], ..., t\left[R_q^*\right] = u[R_q^*] \ominus v[R_q^*]\right) \wedge \\
\left(t\left[R^*\right] \neq \emptyset\right)
\end{array}
\right) \vee \\
\left(
\begin{array}{l}
\left(t[A_1, ..., A_n] = u[A_1, ..., A_n] \neq v[A_1, ..., A_n]\right) \wedge \\
\left(t\left[R_1^*\right] = u[R_1^*], ..., t\left[R_q^*\right] = u[R_q^*]\right)
\end{array}
\right)
\end{array}
\right\} \text{ , if } r \text{ and } s \text{ are nested relations}
\end{array}
$$

- If r is on PNF Bag, then $r \ominus s$ is also on PNF Bag.

The above extended difference definition, can be described as:

- A tuple, belonging to the relation on the left side of the operator, with no match match on the second relation, will belong to the new relation.

Table 3.8: Extended union from the relations present on Table 3.6 and Table 3.7

Music DB						
Artist N.	Genre	Tour			Records	
		Country	Performance		Year	Album
			City	Date		
Green Day	Punk Rock	Japan	Tokyo	16 Aug 12	Kerplunk	1992
			Tokyo	18 Aug 12	Dookie	1994
			Osaka	19 Aug 12	Insomniac	1995
		USA	Las Vegas	21 Sep 12	39/Smooth	1990
		England	Reading	25 Aug 12	¡Uno!	2012
			London	23 Aug 12	Nimrod	1997
		Germany	M. Gladbach	29 Aug 12	Warning	2000
			Berlin	30 Aug 12	American Idiot	2004
			Konstanz	01 Sep 12		
Madonna	Pop	Portugal	Coimbra	24 Jun 12	Like a Prayer	1989
		Germany	Cologne	10 Jul 12	MDNA	2012
		Norway	Oslo	15 Aug 12	Music	2000
The Cranberries	Rock	Germany	Berlin	08 Oct 12	Roses	2012
M. Jackson	Pop	England	London	28 Aug 09	Thriller	1982
			London	13 Jul 09		
			London	06 Mar 10		
			London	09 Jan 10		
			London	12 Jan 10		

- A tuple belonging to the relation on the left side of the operator, with no relational attributes and with a match on the second relation, the generated tuple it will be subject to the outcome of the difference of the count function for both relations.
- A tuple belonging to the relation on the left side of the operator, with relational attributes and with a match on the second relation, the generated tuple it will be subject to the outcome of the extended difference of the relational attributes (applied recursively).
- A relational attribute, $i.e$ a sub-relation, with no tuples is empty.
- A generated tuple, containing relational attributes, whose all relational attributes are empty is discarded – this is, the tuple will not be present on the result relation.
- The generate relation is on PNF Bag, if the relation on the left side of the extended difference operator is also on PNF Bag.

Example 3.4 (Extended difference) *Let* r *be the relation defined on Table 3.6, and the* s *the relation defined on Table 3.9. Both* r *and* s *share the same the* Music DB *schema, which is:*

Table 3.9: Relation over the Music DB schema, containing tuples to be 'deleted'

Music DB						
Artist N.	Genre	Tour			Records	
		Country	Performance		Year	Album
			City	Date		
Green Day	Punk Rock	England	London	23 Aug 12	Dookie	1994
		France	Paris	26 Aug 12	American Idiot	2004
The Cranberries	Rock	Germany	Berlin	08 Oct 12	Roses	2012
M. Jackson	Pop	England	London	06 Mar 10	∅	
			London	13 Jul 09		
			London	28 Aug 09		

Table 3.10: Extended difference from the relations present on Table 3.6 $\ominus$ Table 3.9

Music DB						
Artist N.	Genre	Tour			Records	
		Country	Performance		Year	Album
			City	Date		
Green Day	Punk Rock	Japan	Tokyo	16 Aug 12	Kerplunk	1992
			Tokyo	18 Aug 12	Nimrod	1997
			Osaka	19 Aug 12	Insomniac	1995
		Germany	M. Gladbach	29 Aug 12	Warning	2000
			Berlin	30 Aug 12	¡Uno!	2012
			Konstanz	01 Sep 12		
Madonna	Pop	Portugal	Coimbra	24 Jun 12	Like a Prayer	1989
		Germany	Cologne	10 Jul 12	MDNA	2012
		Norway	Oslo	15 Aug 12	Music	2000
M. Jackson	Pop	∅			Thriller	1982

$$Music\ DB = (Artist\ Name, Genre, Tour^*, Records^*)$$
$$Tour = (Country, Performance^*)$$
$$Performance = (City, Date)$$
$$Records = (Album, Year)$$

Let v *be the resultant relation from the extended difference of* r *and* s*, this is:* r $\ominus$ s = v. *Table 3.10 presents the relation* v.

4 Incremental Expressions

An incremental expression, is a rule that given a *pre-computed structure* (*e.g.* a relational database materialized view) and the changes of the base relations from which that *pre-computed structure* was generated, can produce a new version of the *pre-computed structure*. This version of the *pre-computed structure* have equivalent content to a full re-computation. However, it is desirable that the incremental expression is *cheaper* or faster to compute than a full re-computation.

This chapter is structured as the follows: Section 4.1 discusses the importance of the incremental operations be performed in the same order they occurred in the initial relations. Section 4.2 discusses what operations can can be reverted and what conditions must be fulfilled. Finally, on Section 4.3 the incremental data propagation rules are given for relations on the nested bag model on PNF Bag.

4.1 Execution Order

Let t_1 and t_2 be two different data modifications, and let r be a relation over the schema R. Let r_{t_1} be the state of relation r on t_1 and r_{t_2} be the state of relation r on t_2. Between t_1 and t_2, some tuples from r were deleted Δr and some tuples were inserted ∇r, this is:

$$r_{t_2} = r_{t_1} \oplus \Delta r \ominus \nabla r \tag{4.1}$$

The Equation 4.1 holds if no other data modifications occurred between t_1 and t_2, this means $\Delta r = \emptyset$ or $\nabla r = \emptyset$. If other data modifications occurred between t_1 and t_2 the Equation 4.1 might not hold as the Example: 4.1 illustrates.

Example 4.1 (Equation 4.1 does not hold multiple transactions)
Let t_1 and t_2 be two different transactions, and let the $MusicDB$ schema:

$$MusicDB = (Artist\ Name, Performance^*)$$
$$Performance = (Country, City)$$

Let the relation r, *presented on Table 4.1, be an relation having the schema* Music DB, *on t_1. Between t_1 and t_2 some tuples were deleted ∇r, this relation is presented on Table 4.2, and some tuples were inserted Δr, this relation is presented on Table 4.3. It becomes obvious that:*

$$r_{t_1} \oplus \Delta r \ominus \nabla r \neq r_{t_1} \ominus \nabla r \oplus \Delta r \Leftrightarrow$$
$$\Leftrightarrow Table4.5 \neq Table4.4$$

On Table 4.5 is presented the state of r_{t_2} – this is, the state of r *on t_2 – if the insertions are computed first than the deletions, conversely on Table 4.4 is presented another version of r_{t_2} state if the deletions are processed first than the insertions. Table 4.5 have a tuple which is not present on*

Table 4.1: State of r on t_1 (Example 4.1)

Music DB		
Artist Name	Performance	
	Country	City
Green Day	Japan	Tokyo
	Japan	Osaka
	England	London
	Germany	Berlin

Table 4.2: ∇r, tuples to be deleted (Example 4.1)

Music DB		
Artist Name	Performance	
	Country	City
Green Day	Japan	Tokyo
	Japan	Tokyo
	England	London
	Germany	Berlin
	Italy	Bologna

Table 4.4; therefore this example illustrates that the order the insertions and deletions are performed have influence the final state of the relation.

4.2 Reverse Operators

This section focus on present what operators can be reverted and under what circumstances. If base bag nested relations fulfill PNF Bag restriction, nest and unnest can always be reverted by an unnest or nest operation, respectively. In contrast an extended difference operation, cannot always be reverted by an extended difference operation.

Subsection 4.2.1 presents this claim for extended bag operators, and Subsection 4.2.2 presents this claim for nest and unnest operators.

Table 4.3: Δr, tuples to be inserted (Example 4.1)

Music DB		
Artist Name	Performance	
	Country	City
Green Day	Japan	Tokyo
	England	London
	France	Paris
	USA	Las Vegas
	Italy	Bologna

Table 4.4: $r'_{t_2} = r_{t_1} \ominus \nabla r \oplus \Delta r$ (Example 4.1)

Music DB		
Artist Name	Performance	
	Country	City
Green Day	Japan	Tokyo
	Japan	Osaka
	England	London
	France	Paris
	USA	Las Vegas

Table 4.5: $r''_{t_2} = r_{t_1} \oplus \Delta r \ominus \nabla r$ (Example 4.1)

Music DB		
Artist Name	Performance	
	Country	City
Green Day	Japan	Osaka
	France	Paris
	England	London
	USA	Las Vegas

4.2.1 Extended Union and Extended Difference

An extended difference operation cannot always be reverted by an extended union operation using the same bag on the right side of the operator, this is, let r and s be two bags sharing the same schema, then the following equation is not always true: $r \ominus s \oplus s = r$. This behavior is easily explained by the extended operator definition (Section 3.2); for the flat bag case the extended union perform the addition of the of two results of the count functions, and the extended difference performs the subtraction of the two results of count functions. However, the extended difference adds an additional condition, the result of the subtraction of the count functions is to be equalized to zero if its initially evaluated as a negative number. The Example 4.2 illustrates this claim.

Example 4.2 (The extended difference cannot always be reverted)

Let r *and* s *be two bags having a flat schema, this is:*

$$schema(r) = schema(s) = (Artist\ Name, City)$$
$$r = \{(GreenDay, Tokyo)\}$$
$$s = \{(Madonna, Paris)\}$$

Performing the extended difference from r *and* s *generates* v*, this is:*

$$v = r \ominus s =$$
$$= \{(GreenDay, Tokyo)\} \ominus \{(Madonna, Paris)\} =$$
$$= \{(GreenDay, Tokyo)\}$$

Reverting the operation using the extended union to perform v $\oplus$ s *generates* r*':*

$$
\begin{aligned}
&r' = v \oplus s = \\
&= \{(GreenDay, Tokyo)\} \oplus \{(Madonna, Paris)\} = \\
&= \{(GreenDay, Tokyo), (Madonna, Paris)\}
\end{aligned}
$$

The values from r′ *and* r *are not the same; hence this example demonstrated an extended difference operation cannot always be reverted by the extended union.*

$$
\begin{aligned}
&r' \neq r \Leftrightarrow \\
&\Leftrightarrow \{(GreenDay, Tokyo), (Madonna, Paris)\} \neq \{(GreenDay, Tokyo)\}
\end{aligned}
$$

4.2.2 Nesting and Unnesting

A nest operation can always be reverted by an unnest operation, conversely an unnest operation cannot always be reverted by a nest operation. If the relation, being unnested, is not on PNF Bag there is no guarantee that the nesting of its nested version generates the initial relation. The proof to this claim can be found on Roth, Korth, and Silberschatz (1988, p. 398-399). In order to for an unnest operation can be reverted by a nest operation the nested relation must be on both PNF and PNF Bag form. The PNF Bag restriction was added to the model to enforce relational database alike semantics to the model – and consequently make the model similiar to the one Roth, Korth, and Silberschatz (1988) used –, this is there should not be possible to find duplicate tuples unless the tuples are on the inner most lever. Using other words, only the tuples containing only atomic attributes can be evaluated by the count function with a value higher than one. If a relation fulfills PNF Bag conditions, a nest operation can be successfully revert an unnest operation.

Let r be a relation over a nested schema R ($R = \{A_1, ..., A_n, R_1, ..., R_q\}, n \geq 0, q > 0$), and ω a relational attribute of R ($\omega \in \{R_1, ..., R_q\}$), then the above equations are true:

$$
\begin{aligned}
&r = \mu_\omega (n_\omega (r)) \\
&r \stackrel{?}{=} n_\omega (\mu_\omega (r)) \text{ , if } r \text{ is not on PNF Bag} \\
&r = n_\omega (\mu_\omega (r)) \text{ , if } r \text{ is on PNF Bag}
\end{aligned}
$$

Example 4.3 illustrates how on a relation not on PNF Bag form cannot be reverted to its initial state after an unnest operation.

Example 4.3 (Nest operator is not always the inverse of the unnest operator)

Let r *be the relation presented on Table 4.6 – this relation is on PNF, but it is not on PNF Bag – having the schema:*

$$
\begin{aligned}
&schema(r) = (Artist\ Name, Tour^*) \\
&Tour = (Country, Performance^*) \\
&Performance = (City, Date)
\end{aligned}
$$

Table 4.6: Base relation to be unnested, this relation is not on PNF Bag (Example 4.3)

Music DB			
Artist N.	Tour		
	Country	Performance	
		City	Date
Green Day	Japan	Tokyo	16 Aug 12
	Japan	Tokyo	18 Aug 12
	England	London	23 Aug 12
	Germany	M. Gladbach	29 Aug 12
	Germany	Berlin	30 Aug 12
	Germany	Konstanz	01 Sep 12

Table 4.7: $r'' = \mu_{Performance}\left(\mu_{Tour}\left(r\right)\right)$ = $\mu_{Performance}\left(\mu_{Tour}\left(Table4.6\right)\right)$

Music DB			
Artist N.	Country	Performance	
		City	Date
Green Day	Japan	Tokyo	16 Aug 12
Green Day	Japan	Tokyo	18 Aug 12
Green Day	England	London	23 Aug 12
Green Day	Germany	M. Gladbach	29 Aug 12
Green Day	Germany	Berlin	30 Aug 12
Green Day	Germany	Konstanz	01 Sep 12

Let r' *be the relation generated by unnesting the* Tour *relational attribute from* r: r' = $\mu_{Tour}\left(r\right)$. *Let* r'', *presented on Table 4.7, be the relation generated by unnesting* r' *by the* Performance *relational attribute from* r'':

$$r'' = \mu_{Performance}\left(r'\right) \Leftrightarrow r'' = \mu_{Performance}\left(\mu_{Tour}\left(r\right)\right)$$

Now let s *be the relation generated by nesting* r'' *by the atomic attributes* Artist Name *and* Country:s = $n_{Artist\ Name,Country}\left(r''\right)$.
And let s', *the relation presented on Table 4.8, is generated by nesting* s' *by the atomic attribute* Artist Name:

$$s' = n_{Artist\_Name}\left(s'\right) \Leftrightarrow s' = n_{Artist\_Name}\left(n_{Artist\_Name,Country}\left(r''\right)\right) \Leftrightarrow$$
$$\Leftrightarrow s'' = n_{Artist\_Name}\left(n_{Artist\_Name,Country}\left(\mu_{Performance}\left(r'\right)\right)\right) \Leftrightarrow$$
$$\Leftrightarrow s'' = n_{Artist\_Name}\left(n_{Artist\_Name,Country}\left(\mu_{Performance}\left(\mu_{Tour}\left(r\right)\right)\right)\right)$$

The relation present on Table 4.7 is different from the relation on Table 4.8, this happens because the relation unnested is not on PNF Bag; This example illustrates how an unnest operation cannot always be reverted by a nest operation; hence the relation being unnested should be in PNF Bag in order to restored to its initial state by an nest operation.

Table 4.8: $n_{Artist\ Name}(n_{Artist\ Name,Performance}(Table4.7))$

Music DB			
Artist N.	Tour		
	Country	Performance	
		City	Date
Green Day	Japan	Tokyo	16 Aug 12
		Tokyo	18 Aug 12
	England	London	23 Aug 12
	Germany	M. Gladbach	29 Aug 12
		Berlin	30 Aug 12
		Konstanz	01 Sep 12

4.3 Incremental Data Propagation Rules

An incremental data propagation rule is a rule to be applied repeatedly on a complex structure generated by a query in order to incrementally compute the changes to the structure. These propagation rules use the *pre-computed data structure* (*e.g.* a relational database materialized view) and the changes made to the initial relations. As stated on Section 4.1 the order the operations are executed does matter; therefore the change data propagation rules are to be applied in the same order as the changes occurred on the initial relations.

The term *incremental update* is used to refer to the changes occurred on the initial relation and that will be propagated to the structure storing a query made over the initial relation. Consequently, the initial relation r and the incremental update s have the same schema, this is: $schema(\mathrm{r}) = schema(\mathrm{s})$.

For the nesting operator, the relations must be on PNF Bag and have at least one atomic attribute, $schema(r) = \{A_1, ..., A_n, R_1^*, ..., R_q^*\}, n > 0, q \geq 0$ and $\omega \subseteq \{A_1, ..., A_n\}$. The incremental data propagation rules for the nest operator are presented underneath:

$$n_\omega (r \oplus s) = n_\omega (r) \oplus n_\omega (s) \tag{4.2}$$

$$n_\omega (r \ominus s) = n_\omega (r) \ominus n_\omega (s) \tag{4.3}$$

Regarding the unnesting propagation rules the relations must be on PNF Bag and have at least one relational attribute, $schema(r) = \{A_1, ..., A_n, R_1^*, ..., R_q^*\}, n \geq 0, q > 0$ and $\omega \in \{R_1^*, ..., R_q^*\}$. The incremental data propagation rules for the unnnest operator are presented underneath:

$$\mu_\omega (r \oplus s) = \mu_\omega (r) \oplus \mu_\omega (s) \tag{4.4}$$

$$\mu_\omega (r \ominus s) = \mu_\omega (r) \ominus \mu_\omega (s),$$
$$(q = 1) \wedge (\forall R^* \text{ on the schema have one relational attribute, or all the attributes are atomic}) \tag{4.5}$$

4.3.1 Nest the Extened Union of Two Relations

Let r be the base relation, and s be an incremental addition update; hence is obvious that r and s share the same schema – that is: $schema(r) = schema(r) = \left(A_1, ..., A_m, ..., A_n, R_1^*, ..., R_q^*\right)$, $n \geq 1, q \geq 0$. And let ω be the atomic attributes for which the relation r was nested, this is: $\omega = (A_1, ..., A_m), m \leq n, \omega \subseteq (A_1, ..., A_m, ..., A_n), m \geq 1$. Under these circumstances the proof that equation 4.2 is correct will be given, equation 4.2 is:

$$n_\omega (r \oplus s) = n_\omega (r) \oplus n_\omega (s)$$

- r or s are empty relations
  - r is an empty relation, $r = \emptyset$.

    On this case by the extended union definition (Section 3.2.1) on the right hand side (RHS) the nesting of the extended union of r with s would be the same as the nesting of s, this is:

    $$n_\omega (r \oplus s) = n_\omega (\emptyset \oplus s) = n_\omega (s)$$

    On the left hand side (LHS) the nesting of an empty relation will also generate and empty relation; therefore is the same as the extended union of an empty bag with a nested relation, this is:

    $$n_\omega (r) \oplus n_\omega (s) = n_\omega (\emptyset) \oplus n_\omega (s) = \emptyset \oplus n_\omega (s) = n_\omega (s)$$

    When r is an empty relation, the equation holds.
  - s is an empty relation ($s = \emptyset$).

    Same case as when r is an empty relation.

  The equation holds when r or s are empty.

- The nesting of the extended union of two bags, r and s, is contained on the extended union of the nested version of these bags, this is:

  $$n_\omega (r \oplus s) \subseteq n_\omega (r) \oplus n_\omega (s)$$

  - On the LHS for every tuple α_{lhs} generated by nesting the extended union two bags, r and s, this tuple is contained on the extended union of the nested version of these two bags, that is:

    $$\forall \alpha_{lhs} \in n_\omega (r \oplus s) \Rightarrow \alpha_{lhs} \in n_\omega (r) \oplus n_\omega (s)$$

    For this tuple, α_{lhs}, it does exist another tuple β, such that α_{lhs} can be generated by nesting β. There is a r' and a s', contained on r and s respectively, and are not empty, such that their extended union generates β, this is:

    $$\beta = r' \oplus s', r' \subseteq r, s' \subseteq s, r \neq \emptyset, s \neq \emptyset$$

After this initial constraints, β can be written as:

$$\beta = \left\{ \begin{array}{l} t | \exists u \in r', \exists v \in s', \\ \left(\begin{array}{l} t\left[\mathrm{A}_1, ..., \mathrm{A}_m\right] = u\left[\mathrm{A}_1, ..., \mathrm{A}_m\right] = v\left[\mathrm{A}_1, ..., \mathrm{A}_m\right] \wedge \\ t\left[A_{m+1}, ..., A_n, R_1^*, ..., R_q^*\right] = u\left[\mathrm{A}_{m+1}, ..., \mathrm{A}_n, R_1^*, ..., R_q^*\right] \wedge \\ u\left[\mathrm{A}_{m+1}, ..., \mathrm{A}_n\right] \neq v\left[\mathrm{A}_{m+1}, ..., \mathrm{A}_n\right] \end{array} \right) \vee \\ \left(\begin{array}{l} t\left[\mathrm{A}_1, ..., \mathrm{A}_m\right] = u\left[\mathrm{A}_1, ..., \mathrm{A}_m\right] = v\left[\mathrm{A}_1, ..., \mathrm{A}_m\right] \wedge \\ t\left[A_{m+1}, ..., A_n, R_1^*, ..., R_q^*\right] = v\left[\mathrm{A}_{m+1}, ..., \mathrm{A}_n, R_1^*, ..., R_q^*\right] \wedge \\ u\left[\mathrm{A}_{m+1}, ..., \mathrm{A}_n\right] \neq v\left[\mathrm{A}_{m+1}, ..., \mathrm{A}_n\right] \end{array} \right) \vee \\ \left(\begin{array}{l} t\left[\mathrm{A}_1, ..., \mathrm{A}_m, \mathrm{A}_{m+1}, ..., \mathrm{A}_n\right] = u\left[\mathrm{A}_1, ..., \mathrm{A}_m, \mathrm{A}_{m+1}, ..., \mathrm{A}_n\right] = \\ v\left[\mathrm{A}_1, ..., \mathrm{A}_m, \mathrm{A}_{m+1}, ..., \mathrm{A}_n\right] \wedge \\ t\left[R_1^*\right] = u\left[R_1^*\right] \oplus v\left[R_1^*\right], ..., t\left[R_q^*\right] = u\left[R_q^*\right] \oplus v\left[R_q^*\right] \end{array} \right) \end{array} \right\}$$

Nesting β generates α_{lhs}:

$$\alpha_{lhs} \in n_\omega\left(\beta\right) = \left\{ \begin{array}{l} t' | \exists b \in \beta, t'\left[A_1, ..., A_m\right] = b\left[A_1, ..., A_m\right], t'\left[R_1^*\right] = \\ \left[\left\{ \begin{array}{l} \{t | \exists u \in r', \exists v \in s', \\ \left(\begin{array}{l} \left\{ \begin{array}{l} t\left[A_{m+1}, ..., A_n, R_1^*, ..., R_q^*\right] = \\ u\left[\mathrm{A}_{m+1}, ..., \mathrm{A}_n, R_1^*, ..., R_q^*\right] \wedge \\ u\left[\mathrm{A}_{m+1}, ..., \mathrm{A}_n\right] \neq v\left[\mathrm{A}_{m+1}, ..., \mathrm{A}_n\right] \end{array} \right\} \cup \\ \left\{ \begin{array}{l} t\left[A_{m+1}, ..., A_n, R_1^*, ..., R_q^*\right] = \\ v\left[\mathrm{A}_{m+1}, ..., \mathrm{A}_n, R_1^*, ..., R_q^*\right] \wedge \\ u\left[\mathrm{A}_{m+1}, ..., \mathrm{A}_n\right] \neq v\left[\mathrm{A}_{m+1}, ..., \mathrm{A}_n\right] \end{array} \right\} \cup \\ \left\{ \begin{array}{l} t\left[\mathrm{A}_{m+1}, ..., \mathrm{A}_n\right] = u\left[\mathrm{A}_{m+1}, ..., \mathrm{A}_n\right] = \\ v\left[\mathrm{A}_{m+1}, ..., \mathrm{A}_n\right] \wedge \\ t\left[R_1^*\right] = u\left[R_1^*\right] \oplus v\left[R_1^*\right], ..., \\ t\left[R_q^*\right] = u\left[R_q^*\right] \oplus v\left[R_q^*\right] \end{array} \right\} \end{array} \right) \end{array} \right] \right] \end{array} \right\}$$

– On the RHS for every tuple α_{rhs} generated by the extended union of two nested bags, r and s, this tuple is contained on the extended union of the nested version of these two bags, this is:

$$\forall \alpha_{rhs} \in n_\omega\left(r\right) \oplus n_\omega\left(s\right) \Rightarrow \alpha_{rhs} \in n_\omega\left(r \oplus s\right)$$

Additionally there is a r' and a s', contained on r and s respectively, and are not empty, such that after being nested their extended union generates α_{rhs}, that is:

$$\begin{array}{l} r' \subseteq r, s' \subseteq s, r \neq \emptyset, s \neq \emptyset \\ n_\omega\left(r' \oplus s'\right) \Rightarrow \alpha_{rhs} \end{array}$$

α_{rhs} can be expressed as:

$$\alpha_{rhs} \in \left\{ \begin{array}{l} t'|u \in r', (t'\left[A_1, ..., A_m\right] = u\left[A_1, ..., A_m\right]) \\ t'\left[R_1^*\right] = \left\{t|t\left[A_{m+1}, ..., A_n, R_1^*, ..., R_q^*\right] = r\left[A_{m+1}, ..., A_n, R_1^*, ..., R_q^*\right]\right\} \end{array} \right\} \oplus \left\{ \begin{array}{l} t'|v \in s', (t'\left[A_1, ..., A_m\right] = v\left[A_1, ..., A_m\right]) \\ t'\left[R_1^*\right] = \left\{t|t\left[A_{m+1}, ..., A_n, R_1^*, ..., R_q^*\right] = v\left[A_{m+1}, ..., A_n, R_1^*, ..., R_q^*\right]\right\} \end{array} \right\} \Leftrightarrow$$

$$\Leftrightarrow \alpha_{rhs} \in \left\{ \begin{array}{l} t'|t' = t'\left[A_1, ..., A_m\right] = u\left[A_1, ..., A_m\right] \\ = v\left[A_1, ..., A_m\right], t'\left[R_1^*\right] = \\ \left[\begin{array}{l} \{\exists u \in r', \exists v \in s', \\ \left(\begin{array}{l} \left\{ \begin{array}{l} t\left[A_{m+1}, ..., A_n, R_1^*, ..., R_q^*\right] = \\ u\left[\mathrm{A}_{m+1}, ..., \mathrm{A}_n, R_1^*, ..., R_q^*\right] \wedge \\ u\left[\mathrm{A}_{m+1}, ..., \mathrm{A}_n\right] \neq v\left[\mathrm{A}_{m+1}, ..., \mathrm{A}_n\right] \end{array} \right\} \cup \\ \left\{ \begin{array}{l} t\left[A_{m+1}, ..., A_n, R_1^*, ..., R_q^*\right] = \\ v\left[\mathrm{A}_{m+1}, ..., \mathrm{A}_n, R_1^*, ..., R_q^*\right] \wedge \\ u\left[\mathrm{A}_{m+1}, ..., \mathrm{A}_n\right] \neq v\left[\mathrm{A}_{m+1}, ..., \mathrm{A}_n\right] \end{array} \right\} \cup \\ \left\{ \begin{array}{l} t\left[\mathrm{A}_{m+1}, ..., \mathrm{A}_n\right] = u\left[\mathrm{A}_{m+1}, ..., \mathrm{A}_n\right] = \\ v\left[\mathrm{A}_{m+1}, ..., \mathrm{A}_n\right] \wedge \\ t\left[R_1^*\right] = u\left[R_1^*\right] \oplus v\left[R_1^*\right], ..., \\ t\left[R_q^*\right] = u\left[R_q^*\right] \oplus v\left[R_q^*\right] \end{array} \right\} \end{array} \right) \end{array} \right] \end{array} \right\}$$

$b\left[A_1, ..., A_m\right]$ can be re-written as $u\left[A_1, ..., A_m\right] = v\left[A_1, ..., A_m\right]$, making obvious that $\alpha_{lhs} \subseteq \alpha_{rhs}$; hence it can be concluded that $n_\omega\left(r \oplus s\right) \subseteq n_\omega\left(r\right) \oplus n_\omega\left(s\right)$.

- The proof of $n_\omega\left(r\right) \oplus n_\omega\left(s\right) \subseteq n_\omega\left(r \oplus s\right)$ is similar to the proof of $n_\omega\left(r \oplus s\right) \subseteq n_\omega\left(r\right) \oplus n_\omega\left(s\right)$, therefore is omitted.

The proof that $n_\omega\left(r \oplus s\right) = n_\omega\left(r\right) \oplus n_\omega\left(s\right)$ is concluded.

Example 4.4 (Nest the extended union of two relations)

Let r *and* s *be two flat relations sharing the same schema:*
$schema(r) = schema(s) = (Artist\ Name, Genre, Country, City, Date)$.
Relation r *is presented on Table 4.10 and relation* s *is presented on Table 4.9, the extended union of* r *and* s *is presented on Table 4.11. Nesting* r $\oplus$ s *by artist generates the relation present on Table 4.12.*
Let r′ *be the relation generated by nesting* r *by the atomic attribute* Artist Name*, present on Table 4.14, and on the same way* s′ *generated by nesting* s *by the atomic attribute* Artist Name*, present on Table 4.13. The extended union of* r′ *and* s′ *is presented on Table 4.15.*
The relation present on 4.12 is the same as the one present on Table 4.15, this is:

$$n_{ArtistName}\left(r \oplus s\right) = n_{ArtistName}\left(r\right) \oplus n_{ArtistName}\left(s\right) \Leftrightarrow$$
$$Table\ 4.12 = Table\ 4.15$$

Table 4.9: Flat relation s (Example 4.4)

Artist Name	Genre	Country	City	Date
Green Day	Punk Rock	Japan	Tokyo	18 Aug 12
Green Day	Punk Rock	Japan	Osaka	19 Aug 12
Madonna	Pop	Germany	Cologne	10 Jul 12
M. Jackson	Pop	England	London	13 Jul 09

Table 4.10: Flat relation r (Example 4.4)

Artist Name	Genre	Country	City	Date
Green Day	Punk Rock	Japan	Tokyo	16 Aug 12
Green Day	Punk Rock	England	London	23 Aug 12
Green Day	Punk Rock	Germany	M. Gladbach	29 Aug 12
Green Day	Punk Rock	Germany	Berlin	30 Aug 12
Green Day	Punk Rock	Germany	Konstanz	01 Sep 12
Madonna	Pop	Portugal	Coimbra	24 Jun 12
Madonna	Pop	Norway	Oslo	15 Aug 12
M. Jackson	Pop	England	London	28 Aug 09
M. Jackson	Pop	England	London	06 Mar 10

Table 4.11: $r \oplus s$ = Table 4.10 $\oplus$ Table 4.9

Artist Name	Genre	Country	City	Date
Green Day	Punk Rock	Japan	Tokyo	16 Aug 12
Green Day	Punk Rock	Japan	Tokyo	18 Aug 12
Green Day	Punk Rock	Japan	Osaka	19 Aug 12
Green Day	Punk Rock	England	London	23 Aug 12
Green Day	Punk Rock	Germany	M. Gladbach	29 Aug 12
Green Day	Punk Rock	Germany	Berlin	30 Aug 12
Green Day	Punk Rock	Germany	Konstanz	01 Sep 12
Madonna	Pop	Portugal	Coimbra	24 Jun 12
Madonna	Pop	Germany	Cologne	10 Jul 12
Madonna	Pop	Norway	Oslo	15 Aug 12
M. Jackson	Pop	England	London	28 Aug 09
M. Jackson	Pop	England	London	13 Jul 09
M. Jackson	Pop	England	London	06 Mar 10

Table 4.12: $n_{Artist\ Name}\,(r \oplus s) = n_{Artist\ Name}\,(Table4.10 \oplus Table4.9)$

Artist Name	Relational Attribute			
	Genre	Country	City	Date
Green Day	Punk Rock	Japan	Tokyo	16 Aug 12
	Punk Rock	Japan	Tokyo	18 Aug 12
	Punk Rock	Japan	Osaka	19 Aug 12
	Punk Rock	England	London	23 Aug 12
	Punk Rock	Germany	M. Gladbach	29 Aug 12
	Punk Rock	Germany	Berlin	30 Aug 12
	Punk Rock	Germany	Konstanz	01 Sep 12
Madonna	Pop	Portugal	Coimbra	24 Jun 12
	Pop	Germany	Cologne	10 Jul 12
	Pop	Norway	Oslo	15 Aug 12
M. Jackson	Pop	England	London	28 Aug 09
	Pop	England	London	13 Jul 09
	Pop	England	London	06 Mar 10

Table 4.13: $s' = n_{Artist\ Name}\,(s) = n_{Artist\ Name}\,(Table4.9)$

Artist Name	Relational Attribute			
	Genre	Country	City	Date
Green Day	Punk Rock	Japan	Tokyo	18 Aug 12
	Punk Rock	Japan	Osaka	19 Aug 12
Madonna	Pop	Germany	Cologne	10 Jul 12
M. Jackson	Pop	England	London	13 Jul 09

Table 4.14: $r' = n_{Artist\ Name}\,(r) = n_{Artist\ Name}\,(Table4.10)$

Artist Name	Relational Attribute			
	Genre	Country	City	Date
Green Day	Punk Rock	Japan	Tokyo	16 Aug 12
	Punk Rock	England	London	23 Aug 12
	Punk Rock	Germany	M. Gladbach	29 Aug 12
	Punk Rock	Germany	Berlin	30 Aug 12
	Punk Rock	Germany	Konstanz	01 Sep 12
Madonna	Pop	Portugal	Coimbra	24 Jun 12
	Pop	Norway	Oslo	15 Aug 12
M. Jackson	Pop	England	London	28 Aug 09
	Pop	England	London	06 Mar 10

Table 4.15: $r' \oplus s' = n_{Artist\ Name}(r) \oplus n_{Artist\ Name}(s)$

Artist Name	Relational Attribute			
	Genre	Country	City	Date
Green Day	Punk Rock	Japan	Tokyo	16 Aug 12
	Punk Rock	Japan	Tokyo	18 Aug 12
	Punk Rock	Japan	Osaka	19 Aug 12
	Punk Rock	England	London	23 Aug 12
	Punk Rock	Germany	M. Gladbach	29 Aug 12
	Punk Rock	Germany	Berlin	30 Aug 12
	Punk Rock	Germany	Konstanz	01 Sep 12
Madonna	Pop	Portugal	Coimbra	24 Jun 12
	Pop	Germany	Cologne	10 Jul 12
	Pop	Norway	Oslo	15 Aug 12
M. Jackson	Pop	England	London	28 Aug 09
	Pop	England	London	13 Jul 09
	Pop	England	London	06 Mar 10

4.3.2 Nest the Extened Difference of Two Relations

Let r be the bag relation on the left side of the extended difference operator, and s be the bag on the right side of the difference operator; hence, r and s must share the same schema – this is $schema(r) = schema(r) = \left(A_1, ..., A_m, ..., A_n, R_1^*, ..., R_q^*\right)$, $n \geq 1, q \geq 0$. And let ω be the atomic attributes for which the relations are to be nested, this is: $\omega = (A_1, ..., A_m), m \leq n, \omega \subseteq \left(A_1, ..., A_m, ..., A_n, R_1^*, ..., R_q^*\right), m \geq 1$. Under these circumstances the proof that equation 4.3 is correct will be given, equation 4.3 is:

$$n_\omega (r \ominus s) = n_\omega (r) \ominus n_\omega (s)$$

- r or s are empty relations
  - s is an empty relation, $s = \emptyset$.

    On this case by the extended difference definition (Subsection 3.2.2) on the right hand side (RHS) the nesting of the extended difference of r with s is the same as only nesting r, this is:

    $$n_\omega (r \ominus s) = n_\omega (r \ominus \emptyset) = n_\omega (r)$$

    On the left hand side (LHS) the nesting of an empty relation also generates an empty relation; therefore it is the same as the extended difference of a nested relation with an empty bag, this is:

    $$n_\omega (r) \ominus n_\omega (s) = n_\omega (r) \ominus n_\omega (\emptyset) = n_\omega (r) \ominus \emptyset = n_\omega (r)$$

    When s is an empty relation, the equation holds.

– r is an empty relation ($r = \emptyset$).

This case is very similar to when s is an empty bag. Applying the extended difference definition (Section 3.2.2) on the RHS the nesting of the extended difference of r with s would be the same as the nesting of an empty relation, this is:

$$n_\omega (r \ominus s) = n_\omega (\emptyset \ominus s) = n_\omega (\emptyset) = \emptyset$$

On the LHS the nesting of an empty relation will also generate and empty relation; therefore is the same as the extended difference of an empty bag with a nested relation, this is:

$$n_\omega (r) \ominus n_\omega (s) = n_\omega (\emptyset) \ominus n_\omega (s) = \emptyset$$

When s is an empty relation, the equation holds.

The equation holds when r or s are empty.

- The relation generated by nesting of the extended difference of two bags, r and s, is contained on the extended union of the nested version of these bags, that is:

$$n_\omega (r \ominus s) \subseteq n_\omega (r) \ominus n_\omega (s)$$

– On the LHS for every tuple α_{lhs} generated by nesting the extended difference two bags, r and s, this tuple is contained on the extended difference of the nested version of these two bags, this is:

$$\forall \alpha_{lhs} \in n_\omega (r \ominus s) \Rightarrow \alpha_{lhs} \in n_\omega (r) \ominus n_\omega (s)$$

For this tuple, α_{lhs}, it does exist another tuple β, such that α_{lhs} can be generated by nesting β. There is an r' and an s', contained on r and s respectively, which are not empty, such that their extended difference generates β, that is:

$$\beta = r' \ominus s', r' \subseteq r, s' \subseteq s, r \neq \emptyset, s \neq \emptyset$$

After these initial constraints β can be expressed as:

$$\beta = \left\{ t | \exists u \in r', \exists v \in s', \begin{array}{l} \left(\begin{array}{l} t\left[\mathrm{A}_1, ..., \mathrm{A}_m\right] = u\left[\mathrm{A}_1, ..., \mathrm{A}_m\right] \wedge \\ t\left[A_{m+1}, ..., A_n, R_1^*, ..., R_q^*\right] = u\left[\mathrm{A}_{m+1}, ..., \mathrm{A}_n, R_1^*, ..., R_q^*\right] \wedge \\ \left(\; u\left[\mathrm{A}_1, ..., \mathrm{A}_m\right] \neq v\left[\mathrm{A}_1, ..., \mathrm{A}_m\right] \; \right) \end{array} \right) \vee \\ \left(\begin{array}{l} t\left[\mathrm{A}_1, ..., \mathrm{A}_m, \mathrm{A}_{m+1}, ..., \mathrm{A}_n\right] = u\left[\mathrm{A}_1, ..., \mathrm{A}_m, \mathrm{A}_{m+1}, ..., \mathrm{A}_n\right] = \\ v\left[\mathrm{A}_1, ..., \mathrm{A}_m, \mathrm{A}_{m+1}, ..., \mathrm{A}_n\right] \wedge \\ t\left[R_1^*\right] = u\left[R_1^*\right] \ominus v\left[R_1^*\right], ..., t\left[R_q^*\right] = u\left[R_q^*\right] \ominus v\left[R_q^*\right] \wedge \\ t\left[R^*\right] \neq \emptyset \end{array} \right) \end{array} \right\}$$

Nesting β generates α_{lhs}:

$$\alpha_{lhs} \in n_\omega(\beta) =$$
$$\left\{ t'|\exists b \in \beta, t' = [A_1, ..., A_m] = b[A_1, ..., A_m], t'[R_1^*] = \left[\{t|\exists u \in r', \exists v \in s', \left(\left\{ \begin{array}{l} t\left[A_{m+1}, ..., A_n, R_1^*, ..., R_q^*\right] = u\left[\mathrm{A}_{m+1}, ..., \mathrm{A}_n, R_1^*, ..., R_q^*\right] \wedge \\ u[\mathrm{A}_1, ..., \mathrm{A}_m] \neq v[\mathrm{A}_1, ..., \mathrm{A}_m] \end{array} \right\} \cup \left\{ \begin{array}{l} t[\mathrm{A}_1, ..., \mathrm{A}_m, \mathrm{A}_{m+1}, ..., \mathrm{A}_n] = u[\mathrm{A}_1, ..., \mathrm{A}_m, \mathrm{A}_{m+1}, ..., \mathrm{A}_n] = \\ v[\mathrm{A}_1, ..., \mathrm{A}_m, \mathrm{A}_{m+1}, ..., \mathrm{A}_n] \wedge \\ t[R_1^*] = u[R_1^*] \ominus v[R_1^*], ..., t\left[R_q^*\right] = u\left[R_q^*\right] \ominus v\left[R_q^*\right] \wedge \\ t[R^*] \neq \emptyset \end{array} \right) \right] \right\}$$

– On the RHS for every tuple α_{rhs} generated by the extended difference of two nested bags, r and s, this tuple is contained on the extended difference of the nested version of these two bags, this is:

$$\forall \alpha_{rhs} \in n_\omega(r) \ominus n_\omega(s) \Rightarrow \alpha_{rhs} \in n_\omega(r \ominus s)$$

Additionally there is a r' and a s', contained on r and s respectively, and are not empty, such that after being nested their extended union generates α_{rhs}, that is:

$$r' \subseteq r, s' \subseteq s, r \neq \emptyset, s \neq \emptyset$$
$$n_\omega(r' \ominus s') \Rightarrow \alpha_{rhs}$$

α_{rhs} can be expressed as:

$$\alpha_{rhs} \in \left\{ \begin{array}{l} t'|u \in r', (t'[A_1, ..., A_m] = u[A_1, ..., A_m]) \\ t'[R_1^*] = \left\{ \begin{array}{l} t|t\left[A_{m+1}, ..., A_n, R_1^*, ..., R_q^*\right] = \\ r\left[A_{m+1}, ..., A_n, R_1^*, ..., R_q^*\right] \end{array} \right\} \end{array} \right\} \ominus$$
$$\left\{ \left\{ \begin{array}{l} t'|v \in s', (t'[A_1, ..., A_m] = v[A_1, ..., A_m]) \\ t'[R_1^*] = \left\{ \begin{array}{l} t|t = \left[A_{m+1}, ..., A_n, R_1^*, ..., R_q^*\right] \\ v\left[A_{m+1}, ..., A_n, R_1^*, ..., R_q^*\right] \end{array} \right\} \end{array} \right\} \right\} \Leftrightarrow$$
$$\alpha_{rhs} \in$$
$$\left\{ t'|t' = t'[A_1, ..., A_m] = u[A_1, ..., A_m], t'[R_1^*] = \left[\{\exists u \in r', \exists v \in s', \left(\left\{ \begin{array}{l} t\left[A_{m+1}, ..., A_n, R_1^*, ..., R_q^*\right] = u\left[\mathrm{A}_{m+1}, ..., \mathrm{A}_n, R_1^*, ..., R_q^*\right] \wedge \\ u[\mathrm{A}_1, ..., \mathrm{A}_m] \neq v[\mathrm{A}_1, ..., \mathrm{A}_m] \end{array} \right\} \cup \left\{ \begin{array}{l} t[\mathrm{A}_1, ..., \mathrm{A}_m, \mathrm{A}_{m+1}, ..., \mathrm{A}_n] = u[\mathrm{A}_1, ..., \mathrm{A}_m, \mathrm{A}_{m+1}, ..., \mathrm{A}_n] = \\ v[\mathrm{A}_1, ..., \mathrm{A}_m, \mathrm{A}_{m+1}, ..., \mathrm{A}_n] \wedge \\ t[R_1^*] = u[R_1^*] \ominus v[R_1^*], ..., t\left[R_q^*\right] = u\left[R_q^*\right] \ominus v\left[R_q^*\right] \wedge \\ (t[R^*] \neq \emptyset) \end{array} \right) \right] \right\}$$

$b[A_1, ..., A_m]$ can be re-written as $u[A_1, ..., A_m]]$, making obvious that $\alpha_{lhs} \subseteq \alpha_{rhs}$; hence it can be concluded that $n_\omega(r \ominus s) \subseteq n_\omega(r) \ominus n_\omega(s)$.

Table 4.16: Relation r (Example 4.5)

Artist Name	Country	City	Date
Green Day	Japan	Tokyo	16 Aug 12
Green Day	Japan	Tokyo	18 Aug 12
Green Day	Japan	Osaka	19 Aug 12
Green Day	USA	Las Vegas	21 Sep 12
Green Day	England	Reading	25 Aug 12
Green Day	England	London	23 Aug 12
Green Day	Germany	M. Gladbach	29 Aug 12
Green Day	Germany	Berlin	30 Aug 12
Green Day	Germany	Konstanz	01 Sep 12
Madonna	Portugal	Coimbra	24 Jun 12
Madonna	Germany	Cologne	10 Jul 12
Madonna	Norway	Oslo	15 Aug 12
The Cranberries	Germany	Berlin	08 Oct 12
M. Jackson	England	London	28 Aug 09
M. Jackson	England	London	06 Mar 10
M. Jackson	England	London	09 Jan 10
M. Jackson	England	London	12 Jan 10

- The proof of $n_\omega(r) \ominus n_\omega(s) \subseteq n_\omega(r \ominus s)$ is similar to the proof of $n_\omega(r \ominus s) \subseteq n_\omega(r) \ominus n_\omega(s)$, therefore omitted.

The proof that $n_\omega(r \ominus s) = n_\omega(r) \ominus n_\omega(s)$ is concluded.

Example 4.5 (The extended difference nesting example)

Let r *and* s *be two flat relations sharing the same schema:*
($schema(r) = schema(s) = (Artist\ Name, Country, City, Date)$*).*
Relation r *is presented on Table 4.16 and relation* s *is presented on Table 4.17 , the extended difference of* r $\ominus$ s *is presented on table 4.18, the relation generated by nesting* r $\ominus$ s *by the atomic attribute* Country *is presented on Table 4.19.*

r' *is generated by nesting* r *by the atomic attribute* Country *is presented in table 4.20, and on the same way* s' *is generated by nesting* s *by the atomic attribute* Country *is presented on Table 4.21. The extended difference of* r' $\ominus$ s' *is presented on Table 4.22.*
The relation present on 4.19 is the same as the one present on Table 4.22.

$$n_{Country}(r \ominus s) = n_{Country}(r) \ominus n_{Country}(s) \Leftrightarrow$$
$$Table\ 4.19 = Table\ 4.22$$

Table 4.17: Relation s (Example 4.5)

Artist Name	Country	City	Date
Green Day	Germany	M. Gladbach	29 Aug 12
Green Day	Germany	Berlin	30 Aug 12
Green Day	Germany	Konstanz	01 Sep 12
Madonna	Norway	Oslo	15 Aug 12
M. Jackson	England	London	09 Jan 10
M. Jackson	England	London	12 Jan 10

Table 4.18: $r \ominus s$ = Table 4.16 $\ominus$ Table 4.17

Artist Name	Country	City	Date
Green Day	Japan	Tokyo	16 Aug 12
Green Day	Japan	Tokyo	18 Aug 12
Green Day	Japan	Osaka	19 Aug 12
Green Day	USA	Las Vegas	21 Sep 12
Green Day	England	Reading	25 Aug 12
Green Day	England	London	23 Aug 12
Madonna	Portugal	Coimbra	24 Jun 12
Madonna	Germany	Cologne	10 Jul 12
The Cranberries	Germany	Berlin	08 Oct 12
M. Jackson	England	London	28 Aug 09
M. Jackson	England	London	06 Mar 10

4.3.3 Unnest the Extended Union of Two Relations

Let r and s to be two nested bag relations on PNF Bag to be *united* under the extended union and then unnested; hence, r and s must share the same schema, this is: $schema(r) = schema(s) = \left(A_1, ..., A_n, R_1^*, ..., R_q^*\right)$, $n \geq 0, q > 0$. And let ω be the relational attribute for which the addition of the relation is to be unnested, this is: $\omega \in \left\{R_1^*, ..., R_q^*\right\}$. Under these circumstances the proof that equation 4.4 is correct will be given, equation 4.4 is:

$$\mu_\omega \left(r \oplus s\right) = \mu_\omega \left(r\right) \oplus \mu_\omega \left(s\right)$$

- r or s are empty relations
  - r is an empty relation, $r = \emptyset$.

    On this case by the extended union definition (Subsection 3.2.1), on the right hand side (RHS) the unnesting of the extended union of r with s would be the same as unnesting s, this is:

$$\mu_\omega \left(r \oplus s\right) = \mu_\omega \left(\emptyset \oplus s\right) = \mu_\omega \left(s\right)$$

Table 4.19: $n_{Country}\,(r \ominus s) = n_{Country}\,(Table4.16 \ominus Table4.17)$

Artist Name	Country	City	Date
Green Day	Japan	Tokyo	16 Aug 12
Green Day		Tokyo	18 Aug 12
Green Day		Osaka	19 Aug 12
Green Day	USA	Las Vegas	21 Sep 12
Green Day	England	Reading	25 Aug 12
Green Day		London	23 Aug 12
M. Jackson		London	28 Aug 09
M. Jackson		London	06 Mar 10
Madonna	Portugal	Coimbra	24 Jun 12
Madonna	Germany	Cologne	10 Jul 12
The Cranberries		Berlin	08 Oct 12

On the left hand side (LHS) the unnesting of an empty relation will also generate and empty relation; therefore is the same as the extended union of an empty bag with a unnested relation, this is:

$$\mu_\omega\,(r) \oplus \mu_\omega\,(s) = n_\omega\,(\emptyset) \oplus \mu_\omega\,(s) = \emptyset \oplus \mu_\omega\,(s) = \mu_\omega\,(s)$$

When r is an empty relation, the equation holds.

– s is an empty relation ($s = \emptyset$).

Same case as when r is an empty relation.

The equation holds if r or s are empty.

- The relation generated by unnesting of the extended union of two bags, r and s, is contained on the extended union of the unnested version of these bags, this is:

$$\mu_\omega\,(r \oplus s) \subseteq \mu_\omega\,(r) \oplus \mu_\omega\,(s)$$

– On the LHS for every tuple α_{lhs} generated by unnesting the extended union two bags, r and s, this tuple is contained on the extended union of the unnested version of these two bags, this is:

$$\forall \alpha_{lhs} \in \mu_\omega\,(r \oplus s) \Rightarrow \alpha_{lhs} \in \mu_\omega\,(r) \oplus \mu_\omega\,(s)$$

For this tuple, α_{lhs}, it does exist another tuple β, such that α_{lhs} can be generated by unnesting β, that is: there is an r' and s', contained on r and s respectively, and are not empty, such that their extended union generates β, that is:

$$\beta = r' \oplus s', r' \subseteq r, s' \subseteq s, r \neq \emptyset, s \neq \emptyset$$

β can be expressed as:

Table 4.20: $n_{Country}(r) = n_{Country}(Table4.16)$

Artist Name	Country	City	Date
Green Day	Japan	Tokyo	16 Aug 12
Green Day		Tokyo	18 Aug 12
Green Day		Osaka	19 Aug 12
Green Day	USA	Las Vegas	21 Sep 12
Green Day	England	Reading	25 Aug 12
Green Day		London	23 Aug 12
M. Jackson		London	28 Aug 09
M. Jackson		London	06 Mar 10
M. Jackson		London	09 Jan 10
M. Jackson		London	12 Jan 10
Green Day	Germany	M. Gladbach	29 Aug 12
Green Day		Berlin	30 Aug 12
Green Day		Konstanz	01 Sep 12
Madonna		Cologne	10 Jul 12
The Cranberries		Berlin	08 Oct 12
Madonna	Portugal	Coimbra	24 Jun 12
Madonna	Norway	Oslo	15 Aug 12

$$\beta = \left\{ \begin{array}{l} t | \exists u \in r', \exists v \in s', \\ \left(\begin{array}{l} (t\,[\mathrm{A}_1, ..., \mathrm{A}_n] = u\,[\mathrm{A}_1, ..., \mathrm{A}_n] = v\,[\mathrm{A}_1, ..., \mathrm{A}_n]) \wedge \\ \left(t\left[R_1^*, ..., R_q^*\right] = u\left[R_1^*, ..., R_q^*\right] \oplus v\left[R_1^*, ..., R_q^*\right]\right) \end{array} \right) \vee \\ \left(\begin{array}{l} (t\,[\mathrm{A}_1, ..., \mathrm{A}_n] = u\,[\mathrm{A}_1, ..., \mathrm{A}_n]) \wedge \\ (u\,[\mathrm{A}_1, ..., \mathrm{A}_n] \neq v\,[\mathrm{A}_1, ..., \mathrm{A}_n]) \wedge \\ t\left[R_1^*, ..., R_q^*\right] = u\left[R_1^*, ..., R_q^*\right] \end{array} \right) \vee \\ \left(\begin{array}{l} (t\,[\mathrm{A}_1, ..., \mathrm{A}_n] = v\,[\mathrm{A}_1, ..., \mathrm{A}_n]) \wedge \\ (u\,[\mathrm{A}_1, ..., \mathrm{A}_n] \neq v\,[\mathrm{A}_1, ..., \mathrm{A}_n]) \wedge \\ t\left[R_1^*, ..., R_q^*\right] = v\left[R_1^*, ..., R_q^*\right] \end{array} \right) \end{array} \right\}$$

Unnesting β generates α_{lhs}:

Table 4.21: $n_{Country}(s) = n_{Country}(Table4.17)$

Artist Name	Country	City	Date
Green Day	Germany	M. Gladbach	29 Aug 12
Green Day		Berlin	30 Aug 12
Green Day		Konstanz	01 Sep 12
Madonna	Norway	Oslo	15 Aug 12
M. Jackson	England	London	09 Jan 10
M. Jackson		London	12 Jan 10

Table 4.22: $n_{Country}(r) \ominus n_{Country}(s)$ = Table 4.20 $\ominus$ Table 4.21

Artist Name	Country	City	Date
Green Day	Japan	Tokyo	16 Aug 12
Green Day		Tokyo	18 Aug 12
Green Day		Osaka	19 Aug 12
Green Day	USA	Las Vegas	21 Sep 12
Green Day	England	Reading	25 Aug 12
Green Day		London	23 Aug 12
M. Jackson		London	28 Aug 09
M. Jackson		London	06 Mar 10
Madonna	Germany	Cologne	10 Jul 12
The Cranberries		Berlin	08 Oct 12
Madonna	Portugal	Coimbra	24 Jun 12

$$\alpha_{lhs} \in \mu_\omega \beta = \left\{ t | \exists u \in r', \exists v \in s', \left(\begin{array}{l} (t[\mathrm{A}_1, ..., \mathrm{A}_n] = u[\mathrm{A}_1, ..., \mathrm{A}_n] = v[\mathrm{A}_1, ..., \mathrm{A}_n]) \wedge \\ \left(\begin{array}{l} t\left[\left\{R_1^*, ..., R_q^*\right\} - \{\omega\}\right] = \\ u\left[\left\{R_1^*, ..., R_q^*\right\} - \{\omega\}\right] \oplus v\left[\left\{R_1^*, ..., R_q^*\right\} - \{\omega\}\right] \end{array} \right) \wedge \\ \left(t\left[A_{i1}, ..., A_{in}, R_{j1}^*, ..., R_{jq}^*\right] \in (u[\omega] \oplus v[\omega])\right) \end{array} \right) \vee \left(\begin{array}{l} (t[\mathrm{A}_1, ..., \mathrm{A}_n] = u[\mathrm{A}_1, ..., \mathrm{A}_n] \neq v[\mathrm{A}_1, ..., \mathrm{A}_n]) \wedge \\ \left(t\left[\left\{R_1^*, ..., R_q^*\right\} - \{\omega\}\right] = u\left[\left\{R_1^*, ..., R_q^*\right\} - \{\omega\}\right]\right) \wedge \\ \left(t\left[A_{i1}, ..., A_{in}, R_{j1}^*, ..., R_{jq}^*\right] \in u[\omega]\right) \end{array} \right) \vee \left(\begin{array}{l} (t[\mathrm{A}_1, ..., \mathrm{A}_n] = v[\mathrm{A}_1, ..., \mathrm{A}_n] \neq u[\mathrm{A}_1, ..., \mathrm{A}_n]) \wedge \\ \left(t\left[\left\{R_1^*, ..., R_q^*\right\} - \{\omega\}\right] = v\left[\left\{R_1^*, ..., R_q^*\right\} - \{\omega\}\right]\right) \wedge \\ \left(t\left[A_{i1}, ..., A_{in}, R_{j1}^*, ..., R_{jq}^*\right] \in v[\omega]\right) \end{array} \right) \right\}$$

– On the RHS for every tuple α_{rhs} generated by the extended union of two unnested bags, r and s, this tuple is contained on the extended union of the unnested version of these two bags, this is:

$$\forall \alpha_{rhs} \in \mu_\omega(r) \oplus \mu_\omega(s) \Rightarrow \alpha_{rhs} \in \mu_\omega(r \oplus s)$$

Additionally there is a r' and a s', contained on r and s respectively, which are not empty, such that after being unnested their extended union generates α_{rhs}, that is:

$$r' \subseteq r, s' \subseteq s, r \neq \emptyset, s \neq \emptyset$$
$$\mu_\omega\left(r' \oplus s'\right) \Rightarrow \alpha_{rhs}$$

α_{rhs} can be expressed as:

$$\mu_\omega\left(r'\right) \oplus \mu_\omega\left(s'\right) \Rightarrow \alpha_{rhs} \Leftrightarrow$$
$$\alpha_{rhs} \in \left\{ \begin{array}{l} t | \exists u \in r', \\ \left(\begin{array}{l} t\left[\left\{A_1, .., A_n, R_1^*, ..., R_q^*\right\} - \{\omega\}\right] = \\ u\left[\left\{A_1, .., A_n, R_1^*, ..., R_q^*\right\} - \{\omega\}\right] \end{array} \right) \wedge \\ t\left[A_{i1}, ..., A_{in}, R_{j1}^*, ..., R_{jq}^*\right] \in u\left[\omega\right] \end{array} \right\} \oplus$$
$$\left\{ \begin{array}{l} t | \exists v \in s', \\ \left(\begin{array}{l} t\left[\left\{A_1, .., A_n, R_1^*, ..., R_q^*\right\} - \{\omega\}\right] = \\ v\left[\left\{A_1, .., A_n, R_1^*, ..., R_q^*\right\} - \{\omega\}\right] \end{array} \right) \wedge \\ t\left[A_{i1}, ..., A_{in}, R_{j1}^*, ..., R_{jq}^*\right] \in v\left[\omega\right] \end{array} \right\} \Leftrightarrow$$
$$\alpha_{rhs} \in$$
$$\left\{ \begin{array}{l} t | \exists v \in s', \exists u \in r', \\ \left(\begin{array}{l} \left(t\left[\mathrm{A}_1, ..., \mathrm{A}_n\right] = u\left[\mathrm{A}_1, ..., \mathrm{A}_n\right] = v\left[\mathrm{A}_1, ..., \mathrm{A}_n\right]\right) \wedge \\ \left(\begin{array}{l} t\left[\left\{A_1, .., A_n, R_1^*, ..., R_q^*\right\} - \{\omega\}\right] = \\ u\left[\left\{A_1, .., A_n, R_1^*, ..., R_q^*\right\} - \{\omega\}\right] \oplus \\ v\left[\left\{A_1, .., A_n, R_1^*, ..., R_q^*\right\} - \{\omega\}\right] \end{array} \right) \wedge \\ \left(t\left[A_{i1}, ..., A_{in}, R_{j1}^*, ..., R_{jq}^*\right] \in u\left[R_\omega^*\right] \oplus v\left[\omega\right]\right) \end{array} \right) \vee \\ \left(\begin{array}{l} \left(t\left[\mathrm{A}_1, ..., \mathrm{A}_n\right] = u\left[\mathrm{A}_1, ..., \mathrm{A}_n\right] \neq v\left[\mathrm{A}_1, ..., \mathrm{A}_n\right]\right) \wedge \\ \left(\begin{array}{l} t\left[\left\{A_1, .., A_n, R_1^*, ..., R_q^*\right\} - \{\omega\}\right] = \\ u\left[\left\{A_1, .., A_n, R_1^*, ..., R_q^*\right\} - \{\omega\}\right] \end{array} \right) \wedge \\ t\left[A_{i1}, ..., A_{in}, R_{j1}^*, ..., R_{jq}^*\right] \in u\left[\omega\right] \end{array} \right) \vee \\ \left(\begin{array}{l} \left(t\left[\mathrm{A}_1, ..., \mathrm{A}_n\right] = v\left[\mathrm{A}_1, ..., \mathrm{A}_n\right] \neq u\left[\mathrm{A}_1, ..., \mathrm{A}_n\right]\right) \wedge \\ \left(\begin{array}{l} t\left[\left\{A_1, .., A_n, R_1^*, ..., R_q^*\right\} - \{\omega\}\right] = \\ v\left[\left\{A_1, .., A_n, R_1^*, ..., R_q^*\right\} - \{\omega\}\right] \end{array} \right) \wedge \\ t\left[A_{i1}, ..., A_{in}, R_{j1}^*, ..., R_{jq}^*\right] \in v\left[\omega\right] \end{array} \right) \end{array} \right\}$$

α_{rhs} can be re-written as:

$$
\begin{array}{l}
\alpha_{rhs} \in \\
\left\{ \begin{array}{l}
t | \exists v \in s', \exists u \in r', \\
\left(\begin{array}{l}
(t\left[\mathrm{A}_1, ..., \mathrm{A}_n\right] = u\left[\mathrm{A}_1, ..., \mathrm{A}_n\right] = v\left[\mathrm{A}_1, ..., \mathrm{A}_n\right]) \wedge \\
\left(\begin{array}{l}
t\left[\left\{R_1^*, ..., R_q^*\right\} - \{\omega\}\right] = \\
u\left[\left\{R_1^*, ..., R_q^*\right\} - \{\omega\}\right] \oplus v\left[\left\{R_1^*, ..., R_q^*\right\} - \{\omega\}\right]
\end{array} \right) \wedge \\
\left(t\left[A_{i1}, ..., A_{in}, R_{j1}^*, ..., R_{jq}^*\right] \in u\left[R_\omega^*\right] \oplus v\left[\omega\right]\right)
\end{array} \right) \vee \\
\left(\begin{array}{l}
(t\left[\mathrm{A}_1, ..., \mathrm{A}_n\right] = u\left[\mathrm{A}_1, ..., \mathrm{A}_n\right] \neq v\left[\mathrm{A}_1, ..., \mathrm{A}_n\right]) \wedge \\
\left(\begin{array}{l}
t\left[\left\{A_1, .., A_n, R_1^*, ..., R_q^*\right\} - \{\omega\}\right] = \\
u\left[\left\{A_1, .., A_n, R_1^*, ..., R_q^*\right\} - \{\omega\}\right]
\end{array} \right) \wedge \\
t\left[A_{i1}, ..., A_{in}, R_{j1}^*, ..., R_{jq}^*\right] \in u\left[\omega\right]
\end{array} \right) \vee \\
\left(\begin{array}{l}
(t\left[\mathrm{A}_1, ..., \mathrm{A}_n\right] = v\left[\mathrm{A}_1, ..., \mathrm{A}_n\right] \neq u\left[\mathrm{A}_1, ..., \mathrm{A}_n\right]) \wedge \\
\left(\begin{array}{l}
t\left[\left\{A_1, .., A_n, R_1^*, ..., R_q^*\right\} - \{\omega\}\right] \\
= v\left[\left\{A_1, .., A_n, R_1^*, ..., R_q^*\right\} - \{\omega\}\right]
\end{array} \right) \wedge \\
t\left[A_{i1}, ..., A_{in}, R_{j1}^*, ..., R_{jq}^*\right] \in v\left[\omega\right]
\end{array} \right)
\end{array} \right\}
\end{array}
$$

The equations for α generated on the RHS and on the LHS are equivalent, making obvious that $\alpha_{lhs} \subseteq \alpha_{rhs}$; hence it can be concluded that $\mu_\omega\left(r \oplus s\right) \subseteq \mu_\omega\left(r\right) \oplus \mu_\omega\left(s\right)$.

- The proof of $\mu_\omega\left(r\right) \oplus \mu_\omega\left(s\right) \subseteq \mu_\omega\left(r \oplus s\right)$ is similar to the proof of $\mu_\omega\left(r \oplus s\right) \subseteq \mu_\omega\left(r\right) \oplus \mu_\omega\left(s\right)$, therefore omitted.

The proof that $\mu_\omega\left(r \oplus s\right) = \mu_\omega\left(r\right) \oplus \mu_\omega\left(s\right)$ is concluded.

Example 4.6 (Unnest the extended union of two relations)

Let r *and* s *be two nested relations sharing the same schema:*

$$
\begin{array}{l}
schema(r) = schema(s) = (Artist\_Name, Performance^*) \\
Performance = (Country, City, Date)
\end{array}
$$

Relation r *is presented on Table 4.23 and relation* s *is presented on Table 4.24, the extended union of* r *and* s *is presented on Table 4.25. Unnesting* r $\oplus$ s *by the relational attribute* Performance *generates the relation present on Table 4.26.*

Let r', *presented on Table 4.27, be the relation generated by unnesting* r *over the relational attribute* Performance, *and let* s', *presented on Table 4.28, be the relation generated by unnesting* s *over the relational attribute* Performance. *The extended union of* s' *with* r' *is presented on Table 4.29.*

Apart from the order of the tuples – there is not ordering guarantee on Pig Latin – the tuples from the relation on Table 4.29 are the same as the tuples from the relation on Table Table 4.26.

$$
\begin{array}{l}
\mu_{Performance}\left(r \oplus s\right) = \mu_{Performance}\left(r\right) \oplus \mu_{Performance}\left(s\right) \Leftrightarrow \\
Table\ 4.26 = Table\ 4.29
\end{array}
$$

Table 4.23: Nested relation r (Example 4.6)

Artist Name	Performance		
	Country	City	Date
Green Day	Japan	Tokyo	16 Aug 12
	Japan	Tokyo	18 Aug 12
	Japan	Osaka	19 Aug 12
	England	London	23 Aug 12
	Germany	M. Gladbach	29 Aug 12
	Germany	Berlin	30 Aug 12
	Germany	Konstanz	01 Sep 12
Madonna	Portugal	Coimbra	24 Jun 12
	Germany	Cologne	10 Jul 12
	Norway	Oslo	15 Aug 12
M. Jackson	England	London	28 Aug 09
	England	London	13 Jul 09
	England	London	06 Mar 10

Table 4.24: Nested relation s (Example 4.6)

Artist Name	Performance		
	Country	City	Date
Green Day	France	Paris	26 Aug 12
	USA	Los Angeles	06 Sep 12
Radiohead	Germany	Berlin	29 Sep 12
	England	Manchester	06 Oct 12
	France	Strasbourg	16 Oct 12
Madonna	Austria	Vienna	29 Jul 12
	Switzerland	Zurich	18 Aug 12

4.3.4 Unnest the Extended Difference of Two Relations

Let r be the base relation, and s be a bag containing deleted tuples; hence, is obvious that, r and s share the same schema, this is: $schema(r) = schema(s) = \left(A_1, ..., A_m, ..., A_n, R_1^*, ..., R_q^*\right)$, $n \geq 1, q > 0$. And let ω be the relational attribute for which the relation r was unnested, this is: $\omega \in \left(R_1^*, ..., R_q^*\right)$. Under these circumstances equation 4.5 is correct, if both the following constraints is verified:

- The schema only allows one relational attribute.
- All the relational attributes, this is, included the nested ones allow only at most one relational attribute.

Equation 4.5 is:

Table 4.25: $r \oplus s$ = Table 4.23 $\oplus$ Table 4.24

Artist Name	Performance		
	Country	City	Date
Green Day	France	Paris	26 Aug 12
	USA	Los Angeles	06 Sep 12
	Japan	Tokyo	16 Aug 12
	Japan	Tokyo	18 Aug 12
	Japan	Osaka	19 Aug 12
	England	London	23 Aug 12
	Germany	M. Gladbach	29 Aug 12
	Germany	Berlin	30 Aug 12
	Germany	Konstanz	01 Sep 12
Madonna	Portugal	Coimbra	24 Jun 12
	Germany	Cologne	10 Jul 12
	Norway	Oslo	15 Aug 12
	Austria	Vienna	29 Jul 12
	Switzerland	Zurich	18 Aug 12
Radiohead	Germany	Berlin	29 Sep 12
	England	Manchester	06 Oct 12
	France	Strasbourg	16 Oct 12
M. Jackson	England	London	28 Aug 09
	England	London	13 Jul 09
	England	London	06 Mar 10

$\mu_\omega (r \ominus s) = \mu_\omega (r) \ominus \mu_\omega (s)$,
$(q = 1) \wedge (\forall R^*$on the schema have one relational attribute, or all the attributes are atomic)

The Example 4.8 illustrates a failure when a relation have more than one relational attribute and Example 4.7 illustrates a failure when a relation have more than one relational attribute and the relational attribute being unnested is a subset of the same relational attribute on the other bag.

Example 4.7 (The unnesting the extended difference constraint: subset)

Let r *and* s *be two nested bag relations sharing the same schema, which is:*

$$schema(r) = schema(s) = (Artist, Tour^*, Records^*)$$
$$Tour = (Performance^*)$$
$$Performance = (City, Date)$$
$$Records = (Album, Year)$$

$r = \{\text{KaiserChiefs}, \{(Portugal, \{(Gaia, 19\ Jul\ 12)\})\}, \{(Employment, 2005)\}\}$
$s = \{\text{KaiserChiefs}, \{(Portugal, \{(Gaia, 19\ Jul\ 12)\})\}, \{(Off\ with\ Their\ Heads, 2008)\}\}$

Table 4.26: $\mu_{Performance}\,(r \oplus s)$ =$\mu_{Performance}\,(Table\ 4.23 \oplus Table\ 4.24)$

Artist Name	Country	City	Date
Green Day	France	Paris	26 Aug 12
Green Day	USA	Los Angeles	06 Sep 12
Green Day	Japan	Tokyo	16 Aug 12
Green Day	Japan	Tokyo	18 Aug 12
Green Day	Japan	Osaka	19 Aug 12
Green Day	England	London	23 Aug 12
Green Day	Germany	M. Gladbach	29 Aug 12
Green Day	Germany	Berlin	30 Aug 12
Green Day	Germany	Konstanz	01 Sep 12
Madonna	Portugal	Coimbra	24 Jun 12
Madonna	Germany	Cologne	10 Jul 12
Madonna	Norway	Oslo	15 Aug 12
Madonna	Austria	Vienna	29 Jul 12
Madonna	Switzerland	Zurich	18 Aug 12
Radiohead	Germany	Berlin	29 Sep 12
Radiohead	England	Manchester	06 Oct 12
Radiohead	France	Strasbourg	16 Oct 12
M. Jackson	England	London	28 Aug 09
M. Jackson	England	London	13 Jul 09
M. Jackson	England	London	06 Mar 10

Unnesting r *and* s *over the relational attribute* Tour *generates* r′ *and* s′ *such that:*

$$\mu_{Tour}\,(r) = r' = \{\text{KaiserChiefs}, Portugal, \{(Gaia, 19Jul12)\}, \{(Employment, 2005)\}\}$$
$$\mu_{Tour}\,(s) = s' = \{\text{KaiserChiefs}, Portugal, \{(Gaia, 19Jul12)\}, \{(OffwithTheirHeads, 2008)\}\}$$

Performing the extended difference of r′ ⊖ s′*, generates* v*, which is:*

$$\mu_{Tour}\,(r) \ominus \mu_{Tour}\,(s) = v = \{\text{KaiserChiefs}, Portugal, \{\emptyset\}, \{(Employment, 2005)\}\}$$

In opposition r ⊖ s *generates:*

$$r \ominus s = \{\text{KaiserChiefs}, \{(null, \{\emptyset\})\}, \{(Employment, 2005)\}\}$$

Unnesting r ⊖ s *over the relational attribute* Tour *generates:*

$$\mu_{Tour}\,(r \ominus s) = \{\text{KaiserChiefs}, null, \{\emptyset\}, \{(Employment, 2005)\}\}$$

This example illustrated that when the relations have more than one relational attribute, and on the relation on the left side of the extended difference operator the relational attribute being unnested is contained on contained on the relational attribute from the relation on the right side of the extended difference operator, the equation 4.5 cannot be applied : $\mu_{Tour}\,(r \ominus s) \neq \mu_{Tour}\,(r) \ominus \mu_{Tour}\,(s)$

Table 4.27: $\mu_{Performance}(r)$ =$\mu_{Performance}(Table\ 4.23)$

Artist Name	Country	City	Date
Green Day	Japan	Tokyo	16 Aug 12
Green Day	Japan	Tokyo	18 Aug 12
Green Day	Japan	Osaka	19 Aug 12
Green Day	England	London	23 Aug 12
Green Day	Germany	M. Gladbach	29 Aug 12
Green Day	Germany	Berlin	30 Aug 12
Green Day	Germany	Konstanz	01 Sep 12
Madonna	Portugal	Coimbra	24 Jun 12
Madonna	Germany	Cologne	10 Jul 12
Madonna	Norway	Oslo	15 Aug 12
M. Jackson	England	London	28 Aug 09
M. Jackson	England	London	13 Jul 09
M. Jackson	England	London	06 Mar 10

Table 4.28: $\mu_{Performance}(s)$ =$\mu_{Performance}(Table\ 4.24)$

Artist Name	Country	City	Date
Green Day	France	Paris	26 Aug 12
Green Day	USA	Los Angeles	06 Sep 12
Radiohead	Germany	Berlin	29 Sep 12
Radiohead	England	Manchester	06 Oct 12
Radiohead	France	Strasbourg	16 Oct 12
Madonna	Austria	Vienna	29 Jul 12
Madonna	Switzerland	Zurich	18 Aug 12

Example 4.8 (The unnesting the extended difference constraint: multiple relational attributes)

Let r *and* s *be two nested bag relations sharing the same schema, that is:*

$$schema(r) = schema(s) = (Artist, Tour^*, Records^*)$$
$$Tour = (Performance^*)$$
$$Performance = (City, Date)$$
$$Records = (Album, Year)$$

$$r = \{\text{KaiserChiefs}, \{(Portugal, \{(Gaia, 19\ Jul\ 12)\})\}, \{(Employment, 2005)\}\}$$
$$s = \{\text{KaiserChiefs}, \{(Portugal, \{(Gaia, 19\ Jul\ 12)\})\}, \{(Off\ with\ Their\ Heads, 2008)\}\}$$

Unnesting r *and* s *over the relational attribute* Records *generates* r′ *and* s′*, which are:*

$$\mu_{Records}(r) = r' = \{\text{KaiserChiefs}, \{(Portugal, \{(Gaia, 19\ Jul\ 12)\})\}, Employment, 2005\}$$
$$\mu_{Records}(s) = s' = \{\text{KaiserChiefs}, \{(Portugal, \{(Gaia, 19\ Jul\ 12)\})\}, Off\ with\ Their\ Heads, 2008\}$$

Table 4.29: $\mu_{Performance}(r) \oplus \mu_{Performance}(s) = \mu_{Performanc}(T4.23) \oplus \mu_{Performanc}(T4.24)$

Artist Name	Country	City	Date
Green Day	Japan	Tokyo	16 Aug 12
Green Day	Japan	Tokyo	18 Aug 12
Green Day	Japan	Osaka	19 Aug 12
Green Day	England	London	23 Aug 12
Green Day	Germany	M. Gladbach	29 Aug 12
Green Day	Germany	Berlin	30 Aug 12
Green Day	Germany	Konstanz	01 Sep 12
Madonna	Portugal	Coimbra	24 Jun 12
Madonna	Germany	Cologne	10 Jul 12
Madonna	Norway	Oslo	15 Aug 12
M. Jackson	England	London	28 Aug 09
M. Jackson	England	London	13 Jul 09
M. Jackson	England	London	06 Mar 10
Green Day	France	Paris	26 Aug 12
Green Day	USA	Los Angeles	06 Sep 12
Radiohead	Germany	Berlin	29 Sep 12
Radiohead	England	Manchester	06 Oct 12
Radiohead	France	Strasbourg	16 Oct 12
Madonna	Austria	Vienna	29 Jul 12
Madonna	Switzerland	Zurich	18 Aug 12

Performing the extended difference of r′ ⊖ s′*, generates* r′ *– since none of the tuples of* r′ *and* s′ *do not share the same atomic attributes –, this is:* r′ ⊖ s′ = r′.

In opposition, the extended difference of r ⊖ s *generates:*

$$r \ominus s = \{\text{KaiserChiefs}, \{\emptyset\}, \{(Employment, 2005)\}\}$$

Unnesting $r \ominus s$ *over the relational attribute* $Records$ *generates:*

$$\mu_{Records}(r \ominus s) = \{\text{KaiserChiefs}, \{\emptyset\}, Employment, 2005\}$$

This example demonstrated that when the relations have more than one relational attribute: $\mu_{records}(r \ominus s) \neq \mu_{records}(r) \ominus \mu_{records}(s)$

When the constraints are fulfilled the Equation 4.5 holds, as the above proof confirms:

- r or s are empty relations
  - r is an empty relation, $r = \emptyset$. On this case by the extended difference definition (Subsection 3.2.2) on the left hand side (LHS) the nesting of the extended difference of r with s would be the same as the empty set, this is:

$$\mu_\omega\left(r \ominus s\right) = \mu_\omega\left(\emptyset \ominus s\right) = \mu_\omega\left(\emptyset\right) = \emptyset$$

On the right hand side (RHS) the unnesting of an empty relation will also generate and empty relation; therefore the outcome is the same as the extended difference of an empty bag with a unnested relation, this is:

$$\mu_\omega\left(r\right) \ominus \mu_\omega\left(s\right) = \mu_\omega\left(\emptyset\right) \ominus \mu_\omega\left(s\right) = \emptyset \ominus \mu_\omega\left(s\right) = \emptyset$$

When r is a empty relation, the equation holds.

- s is an empty relation, that is $s = \emptyset$. On this case by the extended difference definition (Subsection 3.2.2) on the LHS the unnesting of the extended difference of r with a empty set would be the same as the unnesting of r, this is:

$$\mu_\omega\left(r \ominus s\right) = \mu_\omega\left(r \ominus \emptyset\right) = \mu_\omega\left(r\right) \ominus \emptyset = \mu_\omega\left(r\right)$$

On the RHS, unnesting a empty relation generates an empty relation; therefore is the same as the extended difference of a unnested relation with an empty bag, this is:

$$\mu_\omega\left(r\right) \ominus \mu_\omega\left(s\right) = \mu_\omega\left(r\right) \ominus \mu_\omega\left(\emptyset\right) = \mu_\omega\left(r\right) \ominus \emptyset = \mu_\omega\left(r\right)$$

When s is an empty relation, the equation holds.

The equation holds if r or s are empty.

- The unnesting of the extended difference of two bags, r and s, is contained on the extended difference of the unnested version of these bags, that is:

$$\mu_\omega\left(r \ominus s\right) \subseteq \mu_\omega\left(r\right) \ominus \mu_\omega\left(s\right)$$

  - On the LHS for every tuple α_{lhs} generated by unnesting the extended difference of two bags, r and s, this tuple is contained on the extended difference of the unnested version of these two bags, this is:

$$\forall \alpha_{lhs} \in \mu_\omega\left(r \ominus s\right) \Rightarrow \alpha_{lhs} \in \mu_\omega\left(r\right) \ominus \mu_\omega\left(s\right)$$

For this tuple, α_{lhs}, it does exist another tuple β, such that α_{lhs} can be generated by unnesting β, that is: There is an r' and s', contained on r and s respectively, and are not empty, such that their extended difference generates β, that is:

$$\beta = r' \ominus s', r' \subseteq r, s' \subseteq s, r \neq \emptyset, s \neq \emptyset$$

β can be expressed as:

$$\beta = \left\{ \begin{array}{l} t | \exists u \in r', \exists v \in s', \\ \\ \left(\begin{array}{l} (t\left[\mathrm{A}_1, ..., \mathrm{A}_m\right] = u\left[\mathrm{A}_1, ..., \mathrm{A}_m\right] = v\left[\mathrm{A}_1, ..., \mathrm{A}_m\right]) \wedge \\ (t\left[\omega\right] = u\left[\omega\right] \ominus v\left[\omega\right]) \wedge \\ t\left[\omega\right] \neq \emptyset \end{array} \right) \vee \\ \left(\begin{array}{l} (t\left[\mathrm{A}_1, ..., \mathrm{A}_m\right] = u\left[\mathrm{A}_1, ..., \mathrm{A}_m\right]) \wedge \\ (u\left[\mathrm{A}_1, ..., \mathrm{A}_m\right] \neq v\left[\mathrm{A}_1, ..., \mathrm{A}_m\right]) \wedge \\ t\left[\omega\right] = u\left[\omega\right] \end{array} \right) \end{array} \right\}$$

Unnesting β generates α_{lhs}:

$$\alpha_{lhs} \in \mu_\omega\left(\beta\right) = \left\{ \begin{array}{l} t | \exists u \in r', \exists v \in s', \\ \\ \left(\begin{array}{l} (t\left[\mathrm{A}_1, ..., \mathrm{A}_n\right] = u\left[\mathrm{A}_1, ..., \mathrm{A}_n\right] = v\left[\mathrm{A}_1, ..., \mathrm{A}_n\right]) \wedge \\ \left(t\left[A_{i1}, ..., A_{in}, R^*_{j1}\right] \in u\left[\omega\right] \ominus v\left[\omega\right]\right) \wedge \\ t\left[R^*_1\right] \neq \emptyset \end{array} \right) \vee \\ \left(\begin{array}{l} (t\left[\mathrm{A}_1, ..., \mathrm{A}_n\right] = u\left[\mathrm{A}_1, ..., \mathrm{A}_n\right] \neq v\left[\mathrm{A}_1, ..., \mathrm{A}_n\right]) \wedge \\ \left(t\left[A_{i1}, ..., A_{in}, R^*_{j1}\right] \in u\left[\omega\right]\right) \end{array} \right) \end{array} \right\}$$

– On the RHS for every tuple α_{rhs} generated by the extended difference of two unnested bags, r and s, this tuple is contained on the extended difference of the unnested version of these two bags, this is:

$$\forall \alpha_{rhs} \in \mu_\omega\left(r\right) \ominus \mu_\omega\left(s\right) \Rightarrow \alpha_{rhs} \in \mu_\omega\left(r \ominus s\right)$$

Additionally there is a r' and a s', contained on r and s respectively, which are not empty, such that after being unnested their extended union generates α_{rhs}, that is:

$$\begin{array}{l} r' \subseteq r, s' \subseteq s, r \neq \emptyset, s \neq \emptyset \\ \mu_\omega\left(r' \ominus s'\right) \Rightarrow \alpha_{rhs} \end{array}$$

α_{rhs} can be expressed as:

$$\mu_\omega(r') \ominus \mu_\omega(s') \Rightarrow \alpha_{rhs} \Leftrightarrow$$
$$\alpha_{rhs} \in \left\{ \begin{array}{l} t|u \in r', \\ (t[A_1,..,A_n] = u[A_1,..,A_n]) \wedge \\ t\left[A_{i1},...,A_{in},R^*_{j1}\right] \in u[\omega] \end{array} \right\} \ominus$$
$$\left\{ \begin{array}{l} t|v \in s', \\ (t[A_1,..,A_n] = v[A_1,..,A_n]) \wedge \\ t\left[A_{i1},...,A_{in},R^*_{j1}\right] \in v[\omega] \end{array} \right\} \Leftrightarrow$$
$$\alpha_{rhs} \in \left\{ \begin{array}{l} t|v \in s', u \in r', \\ \\ \left(\begin{array}{l} (t[\mathrm{A}_1,...,\mathrm{A}_n] = u[\mathrm{A}_1,...,\mathrm{A}_n] = v[\mathrm{A}_1,...,\mathrm{A}_n]) \wedge \\ \left(t\left[A_{i1},...,A_{in},R^*_{j1}\right] \in u[\omega] \ominus v[\omega]\right) \wedge \\ t[R^*_1] \neq \emptyset \end{array} \right) \vee \\ \left(\begin{array}{l} (t[\mathrm{A}_1,...,\mathrm{A}_n] = u[\mathrm{A}_1,...,\mathrm{A}_n] \neq v[\mathrm{A}_1,...,\mathrm{A}_n]) \wedge \\ t\left[A_{i1},...,A_{in},R^*_{j1}\right] \in u[\omega] \end{array} \right) \vee \end{array} \right\}$$

The equations for α generated on the RHS and on the LHS are equivalent, making obvious that $\alpha_{lhs} \subseteq \alpha_{rhs}$; hence it can be concluded that $\mu_\omega(r \ominus s) \subseteq \mu_\omega(r) \ominus \mu_\omega(s)$.

- The proof of $\mu_\omega(r) \ominus \mu_\omega(s) \subseteq \mu_\omega(r \ominus s)$ is similar to the proof of $\mu_\omega(r \ominus s) \subseteq \mu_\omega(r) \ominus \mu_\omega(s)$, therefore is omitted.

The proof that $\mu_\omega(r \ominus s) = \mu_\omega(r) \ominus \mu_\omega(s)$ is concluded.

Example 4.9 (Unnest the extended difference of two relations)

Let r *and* s *be two nested relations sharing the same schema:*

$$schema(r) = schema(s) = (Artist\ Name, Records^*)$$
$$Records = (Album, Year)$$

Relation r *is presented on Table 4.30 and relation* s *is presented on Table 4.31, the extended difference of* r $\ominus$ s *is presented on Table 4.32. Unnesting* r $\ominus$ s *by the relational attribute* Records *generates the relation present on Table 4.33.*

Let r', *presented on Table 4.34 be the relation generated by unnesting* r *over the relational attribute* Records, *and let* s', *presented on Table 4.35, generated by unnesting* s *over the relational attribute* Records. *The extended union of* s' *with* r' *is presented on Table 4.36.*

Apart from the order of the tuples – Pig does not guarantee ordering of the tuples on a relation being nested or unnested – the tuples from the relation on Table 4.33 are the same as as the tuples from the relation on Table Table 4.36.

$$\mu_{Records}(r \oplus s) = \mu_{Records}(r) \oplus \mu_{Records}(s) \Leftrightarrow$$
$$Table\ 4.33 = Table\ 4.36$$

Table 4.30: r nested relation (Example 4.9)

Artist Name	Records	
	Album	Year
Green Day	Kerplunk	1992
	Dookie	1994
	Insomniac	1995
	Nimrod	1997
	Warning	2000
	American Idiot	2004
	¡Uno!	2012
Madonna	Like a Prayer	1989
	MDNA	2012
	Music	2000
M. Jackson	Thriller	1982

Table 4.31: s nested relation (Example 4.9)

Artist Name	Records	
	Album	Year
Green Day	Insomniac	1995
	Nimrod	1997
	Warning	2000
Madonna	MDNA	2012
	Music	2000
M. Jackson	Thriller	1982

Table 4.32: $r \ominus s$ = Table 4.30 $\ominus$ Table 4.31

Artist Name	Records	
	Album	Year
Green Day	Kerplunk	1992
	Dookie	1994
	American Idiot	2004
	¡Uno!	2012
Madonna	Like a Prayer	1989

Table 4.33: $\mu_{Records}\,(r \ominus s)$ = $\mu_{Records}\,(Table4.30 \ominus Table4.31)$

Artist Name	Album	Year
Green Day	Kerplunk	1992
Green Day	Dookie	1994
Green Day	American Idiot	2004
Green Day	¡Uno!	2012
Madonna	Like a Prayer	1989

Table 4.34: $\mu_{Records}\,(r) = \mu_{Records}\,(Table4.30)$

Artist Name	Album	Year
Green Day	Kerplunk	1992
Green Day	Dookie	1994
Green Day	Insomniac	1995
Green Day	Nimrod	1997
Green Day	Warning	2000
Green Day	American Idiot	2004
Green Day	¡Uno!	2012
Madonna	Like a Prayer	1989
Madonna	MDNA	2012
Madonna	Music	2000
M. Jackson	Thriller	1982

Table 4.35: $\mu_{Records}\,(s) = \mu_{Records}\,(Table4.31)$

Artist Name	Album	Year
Green Day	Insomniac	1995
Green Day	Nimrod	1997
Green Day	Warning	2000
Madonna	MDNA	2012
Madonna	Music	2000
M. Jackson	Thriller	1982

Table 4.36: $\mu_{Records}\,(r) \ominus \mu_{Records}\,(s) = \mu_{Records}\,(Table4.30) \ominus \mu_{Records}\,(Table4.31)$

Artist Name	Album	Year
Green Day	Kerplunk	1992
Green Day	Dookie	1994
Green Day	American Idiot	2004
Green Day	¡Uno!	2012
Madonna	Like a Prayer	1989

5 Conclusion

This thesis proposes incremental expressions to maintain *pre-computed data structures* generated from queries containing Pig Latin's nest or unnest operators. The incremental expressions rely on the use of the extended operators for nested bags, purposed and defined on this thesis, and the use of the nested bag model with the partitioned normal form for bags restriction, also proposed and defined on this thesis. This contributions and how they relate to other authors work are explored in detail on Section 5.1. Finally, Section 5.2 presents directions to further work.

5.1 Contribution

This thesis proposes a nested bag data type model, that when used with the partitioned normal form for bags (PNF Bag), suitable to use incremental data propagation expressions. The PNF Bag extends the partitioned normal form (PNF), allowing only duplicates on the inner most levels of a relation. The use of PNF Bag makes the semantics of the bag more alike with a set; therefore it suggests that other authors contributions for models that do not allow duplicates might be possible on a model that allows duplicates. By further extended the PNF restrictions with PNF Bag the nest operator is the reverse of the unnest operator (and vice versa) on bag nested bag relation that respect PNF Bag. The addition of PNF bag to the nested bag model suggests that the conditions presented by Roth, Korth, and Silberschatz (1988) for nested relations on sets are fulfilled.

It proposes and gives the formal definition of the extended nested bag operators under the nested bag model with PNF Bag restriction. The extended operators presented on this thesis are different from the nested set extended operators from Roth, Korth, and Silberschatz (1988) in the way the nested bag data model allows duplicates. Given the nature of semantics the extended difference operator for nested bags, it was demonstrated the order the update operations are performed – using the extended operators – does matter; Consequently incremental updates should be performed by the same order the changes occurred on the relations that originate the *pre-computed structures*.

It gives incremental data propagation expressions for nest and unnest operators, and the proof of correctness. This incremental expressions – with the exception of the nest of the extended union – differ from the ones from Liu, Vincent, and Mohania (1999) in the way they clear and plainer; which is explained by the different semantics of the restrictions introduced by PNF Bag over the nested bag data model, and by the Pig Latin restrictions of nest operator – Pig Latin only allow atomic attributes to be nested, in opposition (Liu, Vincent, and Mohania, 1999) nest operator allow the nesting of both atomic and relational attributes.

Although the incremental expression for the unnest of the extended difference have limitations, the incremental expression can only be used under a very limited number of restrictions, the reminder incremental expressions contribute to the solution of the incremental *pre-computed data structures* maintenance problem on the bag nested data type with the PNF Bag restriction.

5.2 Further Work

Two main issues arrive when performing the maintenance of *pre-computed data structures*: identify the changes to the base relations and apply the changes to the *pre-computed structures*. This thesis focused on propagate the changes to the *pre-computed structures* assumed that changes made to base relations are delivered by the right order; hence, capture the changes to the base relations by the same order they occurred is a challenge that should be addresses to guarantee the consistency of the *pre-computed data structures*.

The use of minimal incremental updates, in a similar way as presented by Griffin, Libkin, and Trickey (1997), would contribute to solve the right order execution of the updates problem. However, bags and sets have different semantics; hence the definition of the semantics of minimal update for nested bag relations must be provided.

Another challenge is to decide when a full re-computation should be performed in preference of an incremental re-computation of the *pre-computed data structures*. Usually the incremental computation is computationally cheaper than a full computation. However, this should not be taken for granted and the system should be able to evaluate and decide on this matter. The multidimensional space of the *pre-computed data structures* maintenance problem is complex (Gupta and Mumick, 1995) therefore, evaluation heuristic algorithms should be proposed.

Finally, the Pig Latin incremental expressions have to be implemented within the MapReduce framework. The incremental expressions are constructed using the extended operators for nested bags. The extended operators are defined in a recursive way; hence to use a recursive approach, several MapReduce rounds are required to perform the computation. The use of an alternative architecture such as Map-Reduce-Merge (Yang, Dasdan, Hsiao, and Parker, 2007) seems to be more appropriate, in the way the update usually is very small compared to the full amount of data; therefore most of the tuples are not to be changed and can be transferred a latter happening computation round; in a simmiliar way *endgame* problem reported on Afrati, Borkar, Carey, Polyzotis, and Ullman (2011).

Bibliography

F.N. Afrati, V. Borkar, M. Carey, N. Polyzotis, and J.D. Ullman. Map-reduce extensions and recursive queries. In *Proceedings of the 14th International Conference on Extending Database Technology*, pages 1–8. ACM, 2011.

J. Albert. Algebraic properties of bag data types. In *Proceedings of the 17th International Conference on Very Large Data Bases*, pages 211–219. Citeseer, 1991.

M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, et al. A view of cloud computing. *Communications of the ACM*, 53 (4):50–58, 2010.

J.A. Blakeley, P.A. Larson, and F.W. Tompa. Efficiently updating materialized views. In *ACM SIGMOD Record*, volume 15, pages 61–71. ACM, 1986.

U. Dayal, N. Goodman, and R.H. Katz. An extended relational algebra with control over duplicate elimination. In *Proceedings of the 1st ACM SIGACT-SIGMOD symposium on Principles of database systems*, pages 117–123. ACM, 1982.

J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

A. Fox, R. Griffith, et al. Above the clouds: A berkeley view of cloud computing. *Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS*, 28, 2009.

T. Griffin and L. Libkin. Incremental maintenance of views with duplicates. *ACM SIGMOD Record*, 24(2):328–339, 1995.

T. Griffin, L. Libkin, and H. Trickey. An improved algorithm for the incremental recomputation of active relational expressions. *Knowledge and Data Engineering, IEEE Transactions on*, 9(3):508–511, 1997.

R.L. Grossman. The case for cloud computing. *IT professional*, 11(2):23–27, 2009.

A. Gupta and I.S. Mumick. Maintenance of materialized views: Problems, techniques, and applications. *Data Engineering Bulletin*, 18(2):3–18, 1995.

G. Hulin. On restructuring nested relations in partitioned normal form. In *16th International Conference on Very Large Data Bases*, pages 626–637, 1990.

L. Libkin and L. Wong. Some properties of query languages for bags. In *Proceedings of 4th International Workshop on Database Programming Languages, New York*, pages 97–114, 1993.

J. Liu, M.W. Vincent, and M.K. Mohania. Incremental evaluation of nest and unnest operators in nested relations. In *Proc. of 1999 CODAS Conf*. Citeseer, 1999.

J. Liu, M.W. Vincent, et al. Derivation of incremental equations for pnf nested relations. *Acta Cybern.*, 16(1):93–131, 2003.

C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins. Pig latin: a not-so-foreign language for data processing. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1099–1110. ACM, 2008.

X. Qian and G. Wiederhold. Incremental recomputation of active relational expressions. *Knowledge and Data Engineering, IEEE Transactions on*, 3(3):337–341, 1991.

M.A. Roth, H.F. Korth, and A. Silberschatz. Extended algebra and calculus for nested relational databases. *ACM Transactions on Database Systems (TODS)*, 13(4):389–417, 1988.

Apache The Apache Software Foundation. Apache hadoop. http://hadoop.apache.org/, 2012a (accessed September 13, 2012). URL http://hadoop.apache.org/.

Apache The Apache Software Foundation. Apache pig. http://pig.apache.org/, 2012b (accessed September 13, 2012). URL http://pig.apache.org/.

H. Yang, A. Dasdan, R.L. Hsiao, and D.S. Parker. Map-reduce-merge: simplified relational data processing on large clusters. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 1029–1040. ACM, 2007.