Adaptive Invoice Processing  
- (using fuzzy pattern matching)

**A PROJECT REPORT**

***In partial fulfilment for the course***

***Of***

Adaptive Invoice Processing  
- (using fuzzy pattern matching)

**GROUP: TERAGIGS**

**TEAM MEMBERS:**

**Bhavik Vora – 001615489**

**Charishma Seenappa – 001684861**

**Deepti Pundlik – 001677326**

**Juilee Jadhav – 001628927**

**Prarthana Angadi - 001619217**



**NORTHEASTERN UNIVERSITY**

**BOSTON, MASSACHUSSETTS**

**DECEMBER 2015**

**ACKNOWLEGDEMENT**

We have gained a lot of knowledge during the completion of this project. However, it would not have been possible without the support and help of many individuals. We would like to express our gratitude to all of them.

We are highly grateful to Prof. Chaiyaporn Mutsalklisana for his constant support, guidance and supervision in completing the project.

We would also like to extend a special thanks to our Teaching Assistant Ashwinkumar Dinoriya for giving us his valuable time and attention.

We would like to thank our classmates and the members of Northeastern University for their kind co-operation and encouragement which helped us in completion of this project.

**INDEX**

|  |  |  |
| --- | --- | --- |
| **Sr. No** | **Title** | **Page no.** |
| 1 | Introduction | 5 |
| 2 | Limitations | 6 |
| 3 | EER Diagram | 7 |
| 4 | Pre-Processing | 8 |
| 5 | Views | 10 |
| 6 | Procedure | 11 |
| 7 | Indexes | 14 |
| 8 | Backups | 18 |
| 9 | Conclusion and Future Scope | 19 |
| 10 | References | 20 |

**INTRODUCTION**

With the steady increase in the financial data it’s essential that we have a precise, reliable & intelligent system which consolidates the data gathered from financial organizations. There exists various such standards where the data has been communicated across the systems such as ANSI X12, EDIFACT, SWIFT they all follow a specific file structure which comprises of the data appearing as per specifications. To process these files you would require to setup various B2B translators, setup maps to translate the data and other tools involving huge costs of setup and maintenance.

There arises a need for a tool which processes these files irrespective of the file format or the structure which would be an effective mode of communication and bearing a low cost of implementation.

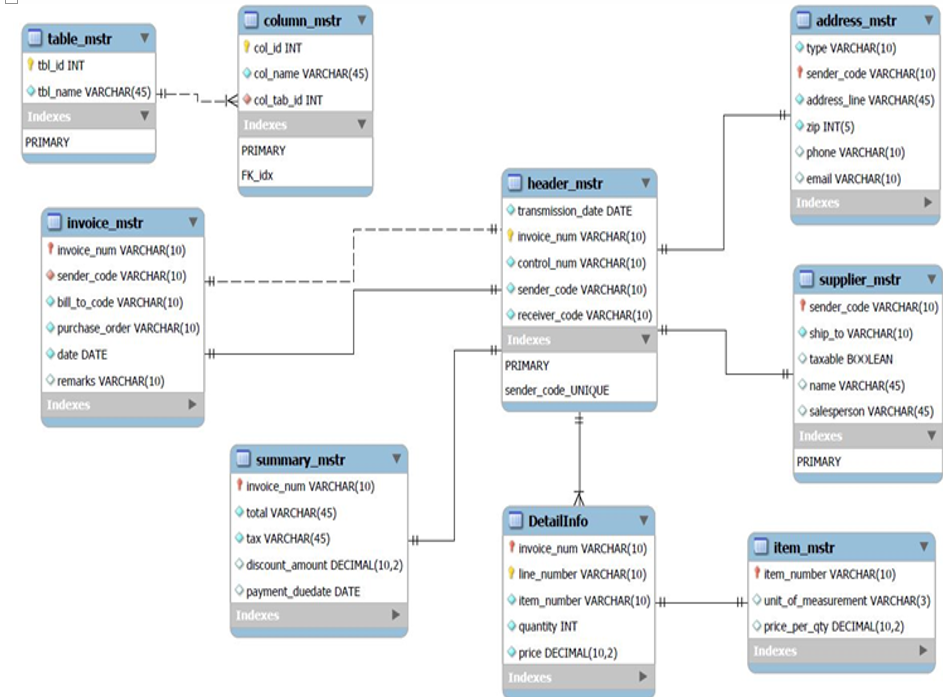
Our application looks towards this need and provides a solution for the same. It is a learning application which takes the input as any file format without a predefined specification and it would learn and understand the structure by storing the data structure every time a new structure or a change arrives creating a repository of the file formats with its structures. This will reduce the failure of these files during the processing and reduce the time delays. The files will be processed as an implementation of Pattern learning, Fuzzy logic and Keyword search.

We have considered a sample file format of an Invoice in an XML Format for the purpose of the implementation. This can be useful to all the firms in representing its financial invoices on the same basis as its foreign competitors, making comparisons easier. Implementation of this system will give the liberty to all the financial bodies to setup & follow their own standards.

**LIMITATIONS**

* The maximum depth of an XML file can only be 3
* The tags which are the primary keys of each table mentioned at level 2 should be the first sub tag.
* In order to achieve the pre-processing successfully no table should contain composite primary key. (As tags are read one by one and data is stored in one column at one time)
* Only one sender code can be associated with one invoice
* Invoice & header tag should be the first tags & sub tags respectively in the incoming file.
* Optional tags can be skipped but no new tags can introduced in the file at level 2. (System doesn’t support dynamic creation of tables)
* File structure should be in such a way that only one tag can be present on one line
* No partial transaction is taken into consideration
* For matching algorithm no tag can be less than 3 characters
* Files can only be in English

**EER DIAGRAM**

****

**PRE-PROCESSING**

Pre –processing of the invoice format file is done in JAVA using NetBeans IDE 8.0.2.

There are multiple tools present in the market for parsing and processing XML files and store the data. Some of them include Simple API for XML (SAX) parser or Document Object Model (DOM) parser or Java Architecture of XML Binding (JAXB). All these parsers require the structures of the XML to be static. Dynamic changes to the tag names or structure cannot be accommodated. The main purpose of our project was to accommodate dynamic structures of an XML file and any changes to the tag names should be accommodated. So usage of any one particular parser was not suiting the requirements.

We have used Node of SAX parser to retrieve the information about XML tag names, data and hierarchy level. This information is stored so that it can be used further. The tags present at level 2 form the tables in the database and their child nodes are the column names of those tables. As the names of the tags are not fixed i.e. they can change with every format, we need to first find out tag name matches which table name. To find that, a connection to the database is made using JBDC and stored procedure is called. This stored procedure will return the table name in which the data should be inserted. Table names for all the tags present at level 2 hierarchy are stored.

Next step is to perform pattern matching for the column names for each table. One column value is inserted at a time. So in order to maintain the key constraints the primary key is first inserted in the table. We have used a status flag in order to find out whether data needs to be inserted in a new row or is a part of the previous row. The flag is set to “new” every time a new row is encountered and is then set to “old”. If the status flag value is “new” then an insert query fired with the table name, column name and data. If the status flag value is “old” then an update query is fired with the same parameters to update the same row but different column values.

In this way the invoice document is parsed and stored in appropriate tables.

**VIEWS**

In mySql, a view refers to a virtual table based on the results of a mySQL statements.It comprises rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database. We can add mySql functions, WHERE, and JOIN statements to a view and present the data as if the data were coming from one single table.

mySql CREATE VIEW Syntax

*CREATE VIEW view\_name AS  
SELECT column\_name(s)  
FROM table\_name  
WHERE condition*

The application contains majorly one view named “invoiceview”



The view “invoiceview” creates a view for the user to extract data from all the tables in the database dbteragigs to get an overall view of the invoice entries into database.It has join statements for header\_mstr, supplier\_mstr, invoice\_mstr, detailinfo, summary\_mstr and address\_mstr.

**STORED PROCEDURES**

Stored Procedures lets us define an API for databases. Reusing this API then becomes easier in multiple applications and programming languages. This technique avoids duplicating database code, saving time and effort when you make updates due to schema changes, tune the performance of queries, or add new database operations for logging, security, and so on.

Create Procedure Syntax

*DELIMITER //*

*CREATE PROCEDURE procedureName(parameters)*

*BEGIN*

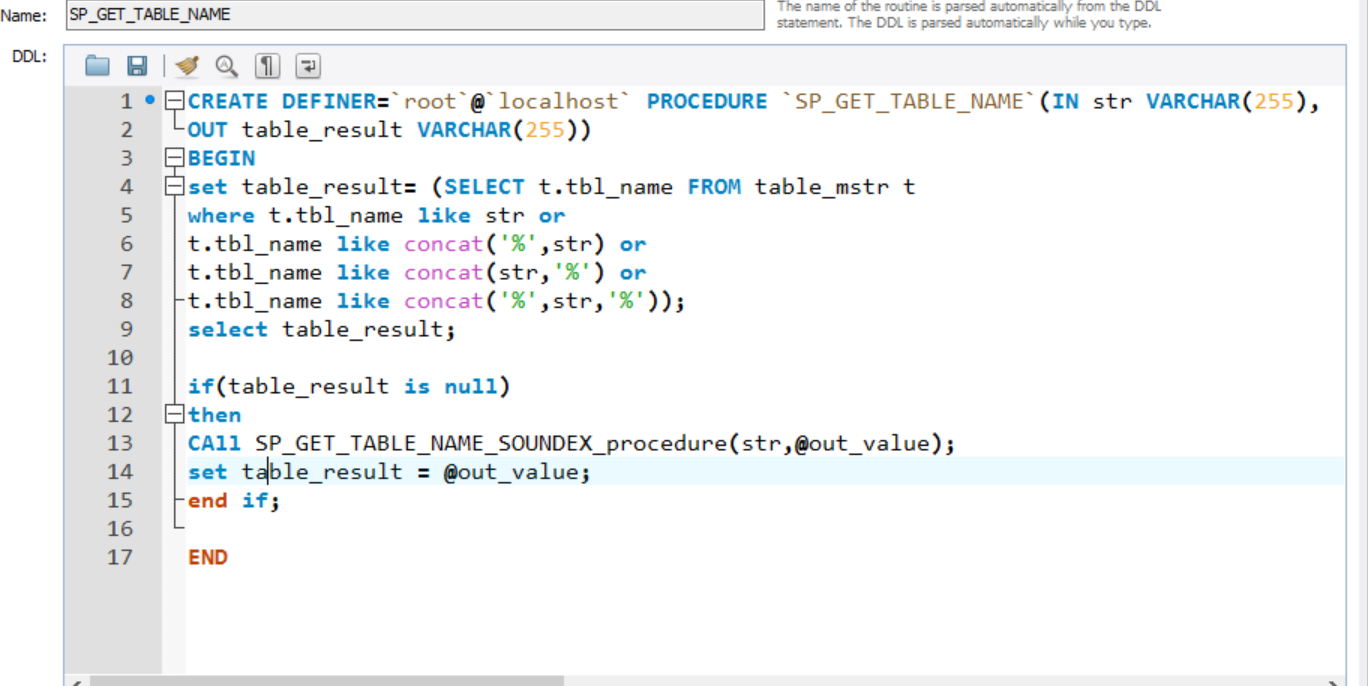
*//Queries*

*END //*

*DELIMITER ;*

The project includes 3 routines for fuzzy pattern matching required for extracting table names and column names and also creating new columns if they do not exist.

The following stored procedure is called through Java code for extracting table names through pattern matching-



The following steps -

SP\_GET\_TABLE\_NAME takes in the level-2 tag name as a varchar parameter from the file format and output parameter which results in the table name after pattern Matching.

Pattern Matching includes –

1) Keyword Matching

2) Phonetic/Sound Matching

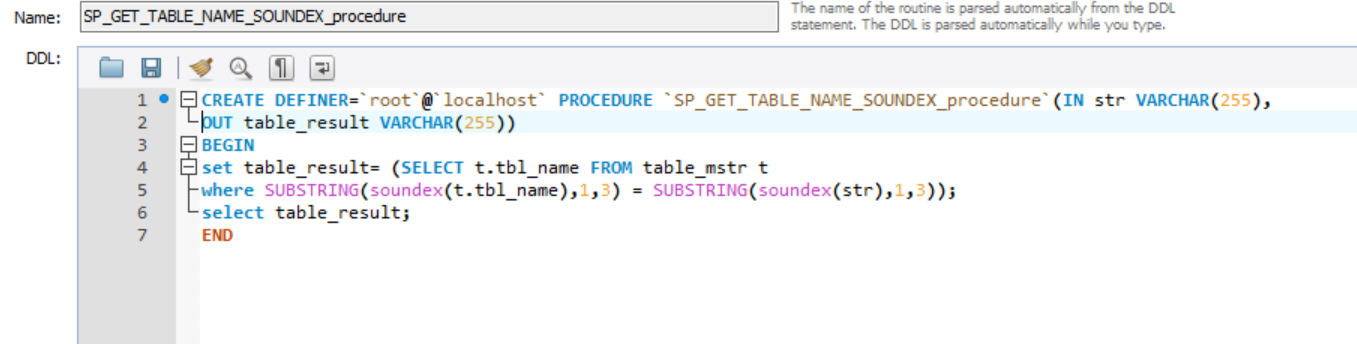
**1) Keyword Matching:-**

The MySQL LIKE condition allows wildcards to be used in the[WHERE clause](http://www.techonthenet.com/mysql/where.php) of a [SELECT](http://www.techonthenet.com/mysql/select.php), [INSERT](http://www.techonthenet.com/mysql/insert.php), [UPDATE](http://www.techonthenet.com/mysql/update.php), or [DELETE](http://www.techonthenet.com/mysql/delete.php) statement. This allows you to perform pattern matching.

For example, if string name that is, tag name is ‘header’ or ‘head’ ,this kind of pattern matching will extract the table ‘header\_mstr’.

**2) Phonetic/Sound matching:-**

Further in the stored procedure, if the compared string is null the following stored procedure is called **–**

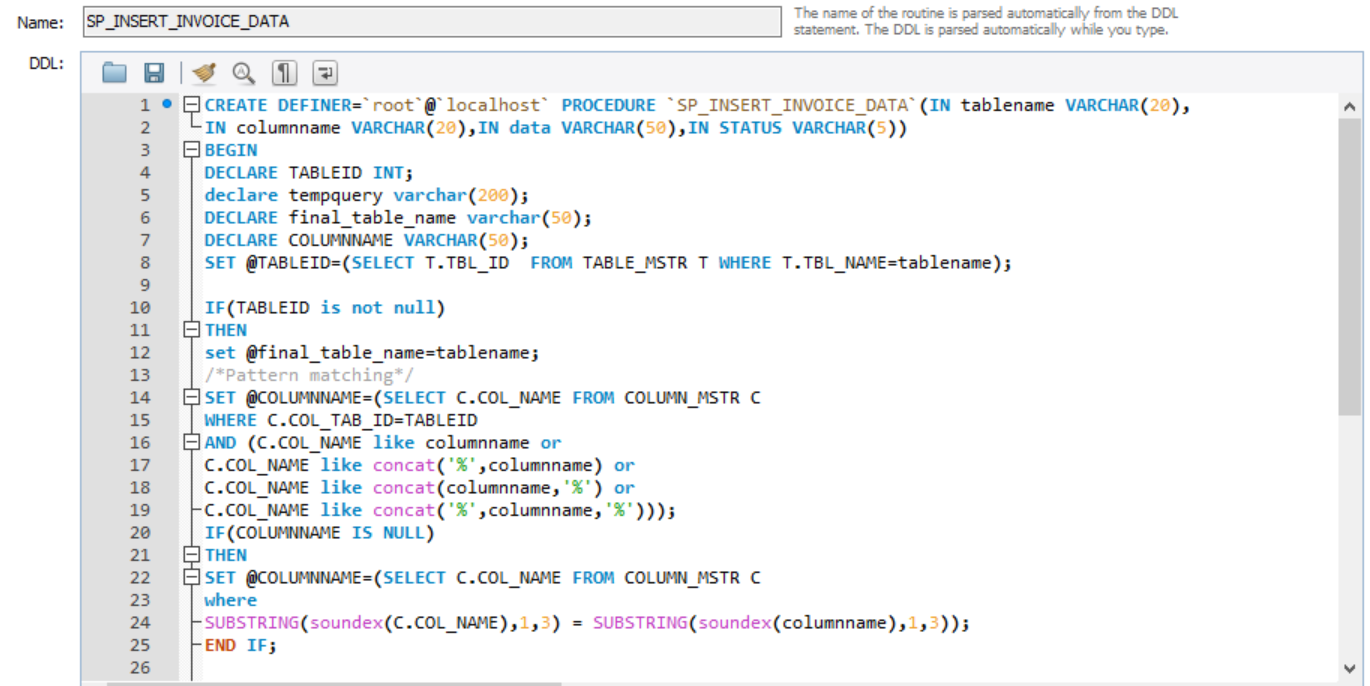


**Soundex** is a phonetic algorithm for indexing names by sound, as pronounced in English. The goal is for homophones to be encoded to the same representation so that they can be matched despite minor differences in spelling.

Soundex is a phonetic algorithm for indexing names by sound, as pronounced in English.  SOUNDEX codes from different strings can be compared to see how similar the strings sound when spoken. The first character of the code is the first character of the expression, converted to upper case. The second through fourth characters of the code are numbers that represent the letters in the expression. The letters A, E, I, O, U, H, W, and Y are ignored unless they are the first letter of the string.  All international alphabetic characters outside the A-Z range are treated as vowels.  Hence, two strings that sound almost the same should have identical soundex strings.  For instance, the words "Assistance" and "Assistants" both produce a soundex of “A223”.

In this stored procedure, substring of every sound is calculated and accordingly, returns sound matched string table name.

* The same logic code applies for column name extraction.

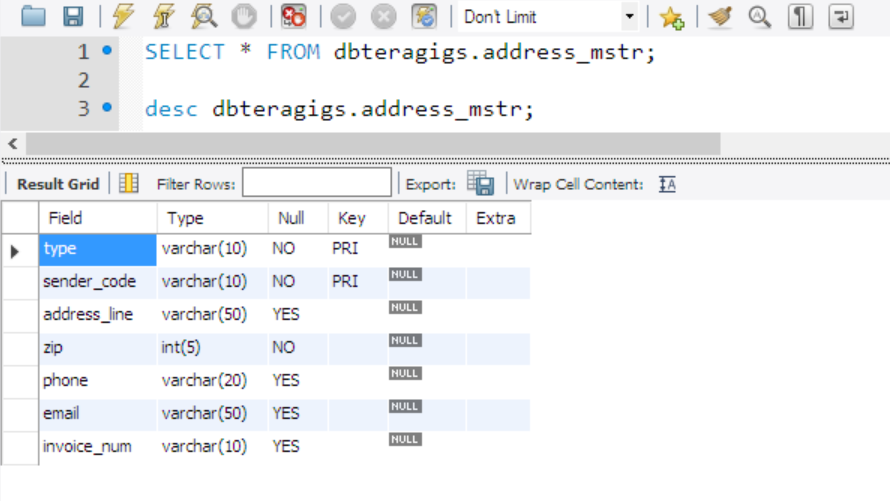


* Further, after pattern matching is done ,if the column name doesn’t exist in the table ,the table is altered to add a new column.

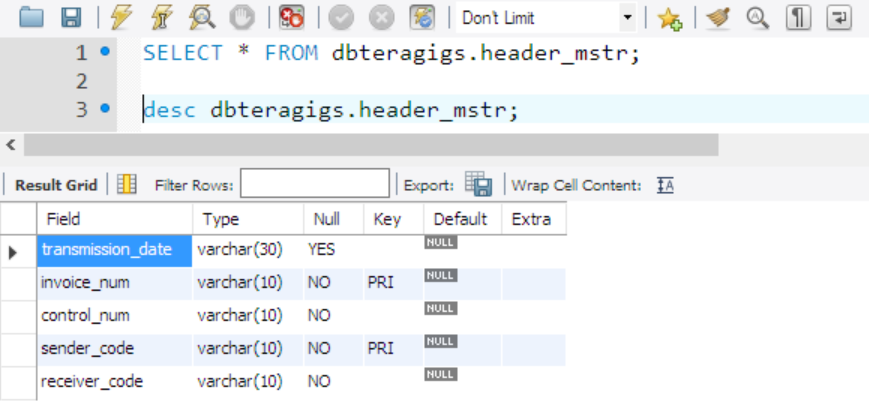
**INDEXES**

* Below shown are the Indexes for the respective tables in the Database ‘dbteragigs’

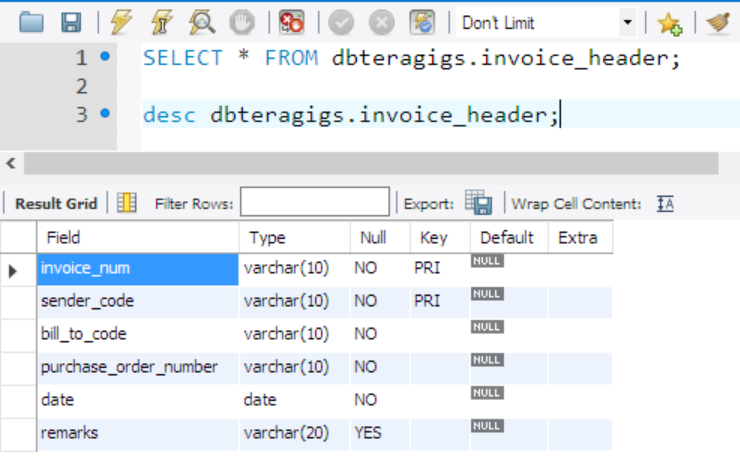
**Table : address\_mstr**



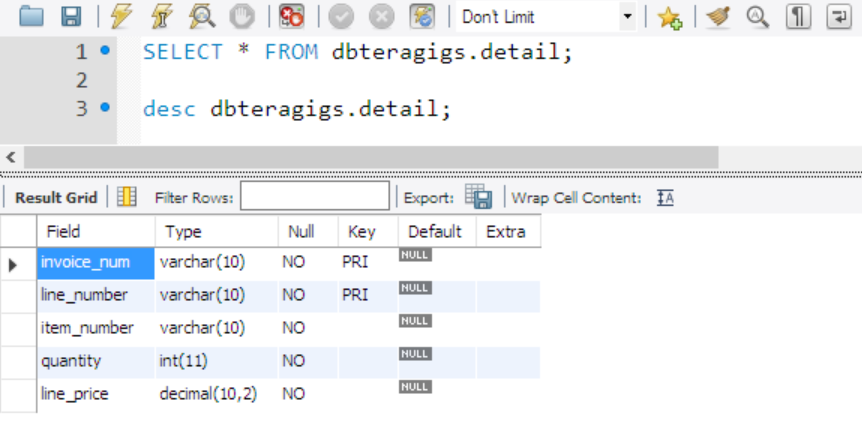
**Table : header\_mstr**



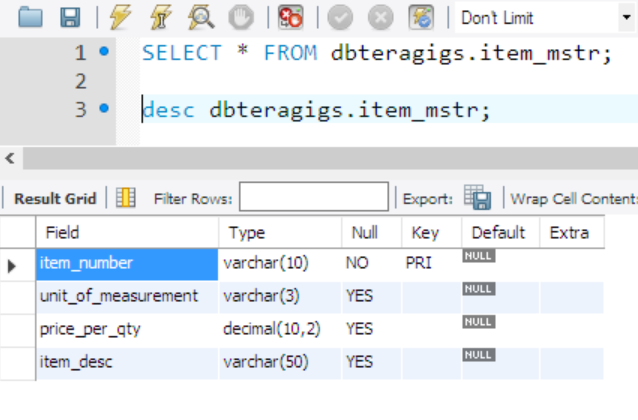
**Table : invoice\_header**



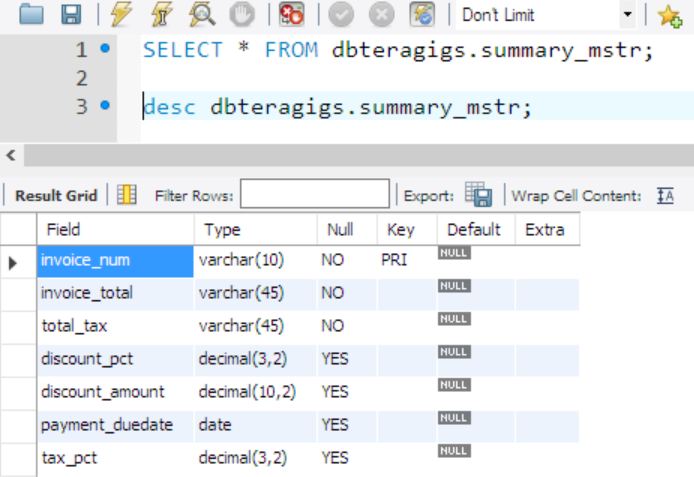
**Table : detail**



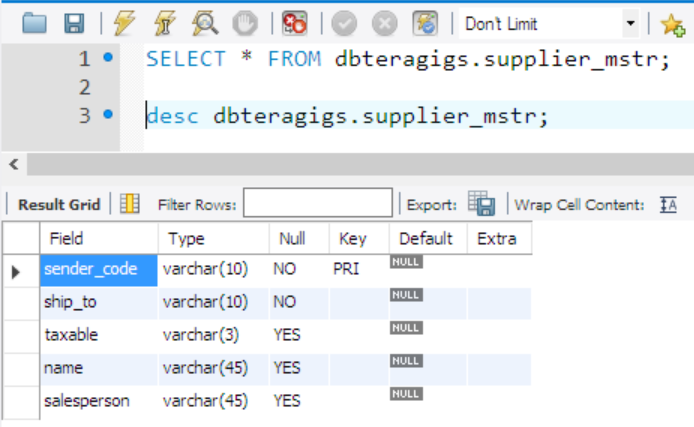
**Table : item\_mstr**



**Table : summary\_mstr**



**Table : header\_mstr**



**BACKUP PLAN**

* Considering the application involves processing of documents such as Invoices, Payments, Orders etc., critical information will be transmitted as well as processed and stored which will require timely backing up.
* Since the functionality of the application has transactional processing, it will require to be up and running for most part of the day thereby offering a very less down time which will provide a little time perform schedule full backup of the database.
* There will be two types of Backups involved for our application :

1. Incremental Backup
2. Full Backup

* The downtime will be available during the night and during this window an Incremental Backup will be scheduled for only the changes made beyond the last backup save points as the time availability will be less for a full backup
* As per the assumption that there will be 6 working days from Monday to Saturday and the time taken for a full backup will be around 8 hours to complete while the Incremental backup will take about 4 hours. Considering the time taken and time available, following will be the schedule for backup process :

**Backup Schedule :**

|  |  |  |
| --- | --- | --- |
| **Day of the Week** | **Backup** | **Schedule** |
| Monday | Incremental | 1:00 am to 5:00 am |
| Tuesday | Incremental | 1:00 am to 5:00 am |
| Wednesday | Incremental | 1:00 am to 5:00 am |
| Thursday | Incremental | 1:00 am to 5:00 am |
| Friday | Incremental | 1:00 am to 5:00 am |
| Saturday | Incremental | 1:00 am to 5:00 am |
| Sunday | Full Backup | 10:00 am to 6:00 pm |

**CONCLUSION AND FUTURE SCOPE**

Thus we have achieved the main objective of our project to build a database to accept invoices of different formats & structures. We have completely normalized all our tables in the database. We have performed all the operations on these normalized tables. We have successfully implemented concepts of Procedures, Views, Indexes and drafted a backup strategy for our system. We have also integrated Business Intelligence concepts.

There are still few functionalities which we can work on in the future:

* We would want to implement dynamic table creation when new tags are introduced in the file.
* Apart from invoice all different types of file structures like credit transfers, direct debits of different formats shall be accepted.

**REFERENCES**

1. <http://ap.finance.ucla.edu/UC_XML_Invoice>.
2. <http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3920815/>
3. <https://dev.mysql.com/>
4. <https://generateData.com/>
5. <https://www.mysqltutorial.org/>
6. <http://www.databasejournal.com/features/mysql/mysql-fuzzy-text-searching-using-the-soundex-function.html>

**Books**

My SQL Documentation

Modern Database Management

Presentation slides by Prof. Mutsalklisana