Visualization for the InsightNotes Database Annotation Management System

A Major Qualifying Project Report submitted to the faculty of
Worcester Polytechnic Institute
in partial fulfillment of the requirements for the
Degree of Bachelor of Science in Computer Science

by

Armir Bashllari
Tyler Menard

Date: 5-1-14

Approved:

____

1. Abstract

InsightNotes is an annotation management system designed for scientific databases. It addresses the increasing scale of annotations that occur within systems such as biological databases, where the number of annotations can be 10s or even 100s of times greater than the number of records that are stored. This immense quantity of annotations can make it difficult to decipher useful information from a record. The solution that InsightNotes provides is to create concise representations, or summaries, of the annotations in order to produce more meaningful data. However, there is currently no visualization for InsightNotes. Having a way to easily display the results of this system will help in creating a practical way to access information from a large database. What we provide is a visualization for the functionality of InsightNotes, making it easier to manipulate and retrieve the data along with the annotations and summaries.

Table of Contents:

List of Figures

2. Introduction

2.1 Introduction to InsightNotes

Modern relational database systems capture the details and observations of different users through annotations that are attached to the tuples of data. The credibility of a scientific database system may be assessed through the quality of its annotations. With the increasing scale of collaboration between scientists and the use of automated annotation tools, database systems eventually have too many annotations to get any useful meaning from them. A tuple of data may have as many as hundreds of annotations attached to it, making it difficult to process all of the information, so a method for

InsightNotes offers a solution through summarizing these annotations. It provides concise representations of the raw annotations in what are called annotation summaries. There are various methods for summarizing these annotations, one of which allows users to define their own summarization criteria, making InsightNotes extensible to a wide variety of applications.

2.2 Overview

Currently, the InsightNotes system is only implemented through a command-line interface making it difficult to organize and retain the data, as well as making it difficult to view a record, its annotations, and its summary simultaneously. In order to retrieve and display all this data in a useful way some new system must be created that allows the user to query the database and retrieve the results of the query, including the records returned and their annotations. The system must also be able to query the database to generate and retrieve any desired summary, as well as displaying this summary along with its associated records and annotations. Because it provided tools for connecting to, querying, and retrieving data from PostgreSQL databases, LibreOffice was chosen as the platform on which to build the desired interface.

2.3 Goals for the Project

Using LibreOffice:

- Add functionality to query a PostgreSQL database running InsightNotes and retrieve the appropriate records along with their annotations.
- Display the query results in a spreadsheet, with annotations attached appropriately.
- Allow the user to generate summaries
- Allow the user to retrieve the annotations from the summaries (support for the “zoom in” feature of InsightNotes).

2.4 Chapter Overview

The rest of the report will describe the InsightNotes system and our implementation of the visualization in detail. Chapter 3 covers the overview of InsightNotes, which goes over the different summarization techniques and the data structures necessary to implement the system. There are also optimization techniques and algorithms for the summarization techniques which are beyond the scope of this report and will not be covered. Chapter 4 will go over some background information on the tools we used to create our implementation. Chapter 5 will describe our visualization in detail. These sections will go over each piece of functionality, such as connecting to the database, inserting annotations, and generating annotations. Chapter 6 will contain the conclusion for our visualization, followed by a section for any documents related to the project.

3. InsightNotes Overview

InsightNotes addresses the fact that varying types of annotation summaries may be suitable for different applications. For example, a biological database on birds (ornithology) may want to have its annotations categorized into different groups such as Habitat, Behavior, and General Comments. InsightNotes allows for database administrators to apply this sort of customization. If the user then wanted to look at the raw annotations for each category, a “zoom in” feature is implemented to expand annotations for a given category.

3.1 Summary Types

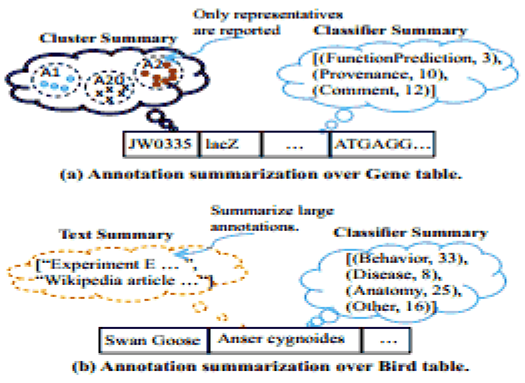

Only representatives are reported
Cluster Summary
A1
A20
A2
Classifier Summary
[(FunctionPrediction, 3), (Provenance, 10), (Comment, 12)]
JW0335
lacZ
...
ATGAGG...
(a) Annotation summarization over Gene table.
Summarize large annotations.
Text Summary
["Experiment E ...", "Wikipedia article ..."]
Classifier Summary
[(Behavior, 33), (Disease, 8), (Anatomy, 25), (Other, 16)]
Swan Goose
Anser cygnoides
...
(b) Annotation summarization over Bird table.

Figure 1. Examples of Annotation summaries

Figure 1 above shows the three types of data mining techniques for summarizing raw annotations that InsightNotes supports. The first is text summarization, which is used to create concise snippets of large text documents, as shown in figure 1(b). The second is clustering, which clumps similar content together into distinct groups, as shown in figure 1(a). The final technique is classification, shown on the right side of both (a) and (b), which is what allows administrators to define annotations based on their desired classifiers.

Each annotation summary has a type that consists of a pair of values {Name, TypeProperties}, where *Name* is a unique identifier that defines the summary type, such as "Snippet," "Cluster" or "Classifier," and *TypeProperties* is a set of properties to be used in the creation of the summaries. All summaries will have at least two Boolean type properties. The first of these is *AnnotationsInvariant*, which indicates whether the summarization of a newly added annotation over a tuple depends on that tuple's existing annotation. The second is *DataInvariant*, which indicates whether the summarization depends on the contents of a given tuple. There are also type-specific properties, for example, a summary of the type "Snippet" (text summary) may also have the *LargeObjThreshold* property, which specifies the threshold at which an annotation is considered large enough to be summarized into a text summary. As mentioned before, database administrators would have the option to organize summaries into custom categories. This is possible with summaries of type "Classifier," where *ClassLables* is included in the set of type properties.

3.2 Summary Instances

Each summary has a summary instance, which is a four-part structure {*InstanceID, TypeName, FunctionID, InstanceProperties*} that defines the exact algorithm used to implement the summary and its properties. *InstanceID* is a unique identifier for each instance, *TypeName* is the summary type, *FunctionID* refers to the name of the function/algorithm that implements the instance, and *InstanceProperties* is a set of values indicating instance-level properties for the summary. Once summary instances are defined in the database, they can be linked to the user's relations through a many-to-many relation. Each relation can have many instances linked to it in order to summarize the annotations attached to each tuple and to create summary objects.

3.3 Summary Objects

The algorithms that are specified by the Summary Instances create Summary objects, which are what summarize the raw annotations of a relation. These objects will be attached to relations and be maintained and updated by the system. Also, summary objects are what will be propagated to the end-user. A summary object is five-piece structure {*ObjID, InstanceID, TupleID, Rep[], Elements[][]*} where *ObjID* is an object's unique identifier, *InstanceID* refers to the object's corresponding summary instance, *TupleID* refers to the object's corresponding data tuple, *Rep[]* is an array of the representatives produced from the summarization algorithm, and *Elements[][]* is a two-dimensional array that stores the contributing raw annotations for each representative. Each summary type has a different representative structure that affects what the end-user would see. This is also pictured in figure 1.

3.4 Summary Propagation Strategies

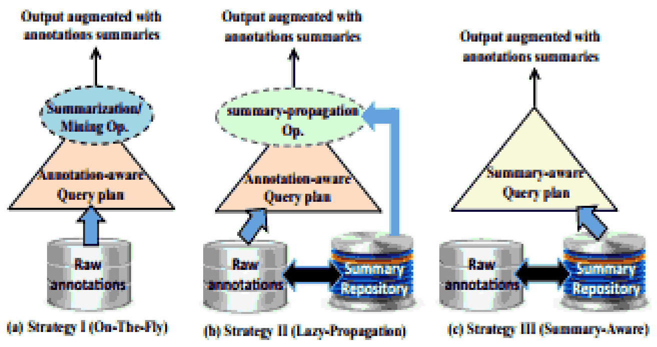
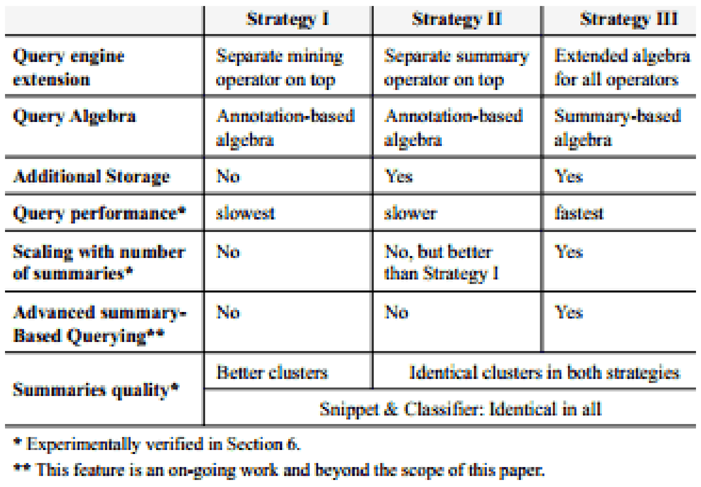

Output augmented with annotations summaries
Summarization/ Mining Op.
Annotation-aware Query plan
Raw annotations
(a) Strategy I (On-The-Fly)
Output augmented with annotations summaries
summary-propagation Op.
Annotation-aware Query plan
Raw annotations
Summary Repository
(b) Strategy II (Lazy-Propagation)
Output augmented with annotations summaries
Summary-aware Query plan
Raw annotations
Summary Repository
(c) Strategy III (Summary-Aware)

Figure 2. Summary Propagation Strategies

There are three different strategies for creating and propagating the annotation summaries, as shown in figure 2 above. The first is the *On-The-Fly* strategy in figure 2(a), which postpones the creation of summaries until query time. This means that the query process is left unchanged until the very end. The *Lazy-Propagation* strategy in figure 2(b) adds in the summaries at the last stage of query processing, where annotations have already been generated. Finally, the *Summary-Aware* strategy depicted in figure 2(c) integrates summaries at the early stages of query processing so that raw annotations do not need to be retrieved if they are not needed. Figure 3 below compares the details between strategies.

Figure 3. Strategy Comparisons



	Strategy I	Strategy II	Strategy III
Query engine extension	Separate mining operator on top	Separate summary operator on top	Extended algebra for all operators
Query Algebra	Annotation-based algebra	Annotation-based algebra	Summary-based algebra
Additional Storage	No	Yes	Yes
Query performance*	slowest	slower	fastest
Scaling with number of summaries*	No	No, but better than Strategy I	Yes
Advanced summary-Based Querying**	No	No	Yes
Summaries quality*	Better clusters	Identical clusters in both strategies	
	Snippet & Classifier: Identical in all		

* Experimentally verified in Section 6.

** This feature is an on-going work and beyond the scope of this paper.

4. Other Background Information

4.1 LibreOffice

We decided to develop the visualization of InsightNotes on linux in order to avoid any compatibility issues with prerequisite programs that needed to be installed. InsightNotes was developed on the Mac-OS. Due to our lack of Mac-OS experience we chose to work on linux. LibreOffice is a free and open source alternative to Microsoft Office that works well on linux and has the tools required to create the visualization. Specifically, we used Open Office Calc, which is a spreadsheet program similar to excel. LibreOffice is the default office suite for some linux distributions, however it does not necessarily come with LibreOffice Base. In order to connect to a database and issue commands to it through any of the LibreOffice programs, LibreOffice Base must be installed. This allows users to add code through the programming language LibreOffice Basic, which is similar to Microsoft Visual Basic.

4.2 PostgreSQL

PostgreSQL is an object-relational database management system. It is cross-platform and runs on many operating systems including Linux, Mac OS, and Windows. InsightNotes was developed using PostgreSQL version 9.1.2, so the first step for creating the visualization was getting this version of Postgres installed. The details of the installation process will be included at the end of the report in the documents/appendix section

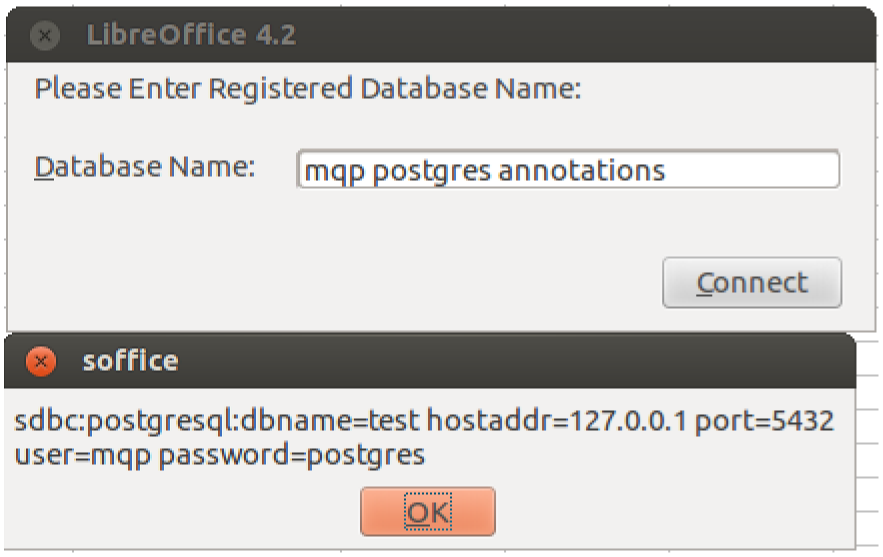
4.3 VirtualBox

Our visualization for InsightNotes was developed in a virtual machine running Ubuntu. The virtual machine software we used was Oracle VM Virtualbox. The virtual machine allowed us to easily transport the builds of our visualization onto different computers without having to reinstall all the prerequisite programs or having to worry about what operating system was being used.

5. Visualization

5.1 Connecting to the Database

Connecting to the InsightNotes PostgreSQL database was achieved using built in functionality in LibreOffice BASIC. Databases running on the system are visible through the DataSource Service and are given a registered name. In order to connect to an InsightNotes database, a dialog box is presented which asks the user to input the name of a database registered on their client.



```
mqp postgres annotations
```

```
sdbc:postgresql:dbname=test hostaddr=127.0.0.1 port=5432
user=mqp password=postgres
```

After the user has input a registered database name and pressed "Connect", this String is read by the connectDB Sub. This sub creates a DatabaseContext and a connDB UnoService to retrieve all available databases and handle a database connection. From the available databases, the proper URL is selected from the registered database name and a connection is attempted. If there is no password required for the database then the connection details are stored and Calc is now connected to the database and the database URL is printed in a message box so that the user can verify their selection. If a password is required, an interaction handler is created and the user must complete the login before the connection is established.

```
Sub connectDB
    Dim ConnectionDetails As Object
    Dim dbName As String

       DatabaseContext = createUnoService("com.sun.star.sdb.DatabaseContext")
       ConnectionDetails = createUnoService("com.sun.star.sdb.connDB")
       dbName = Dlg.getControl("dbName").Text

       DataSource = DatabaseContext.getByName(dbName)

       If Not DataSource.IsPasswordRequired Then
       Connection = DataSource.GetConnection("","")
       Else
       InteractionHandler =  createUnoService("com.sun.star.sdb.InteractionHandler")
       Connection = DataSource.ConnectWithCompletion(InteractionHandler)
       End If

       MsgBox DataSource.URL()
End Sub
```

5.2 Querying the Database

Database queries are performed by constructing a SQL SELECT statement from user input for the table to query and any WHERE clause conditions. The user cannot, however, select which columns to return and will have all columns of the requested table returned. Annotations are automatically fetched with the table records and inserted as comments into the spreadsheet cell corresponding to the tuple and column specified in the data_anno table.

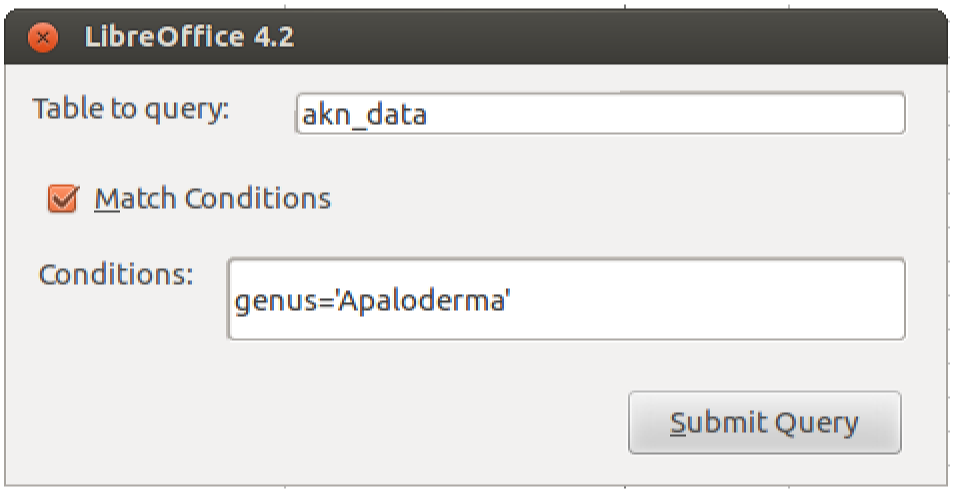

LibreOffice 4.2

Table to query: akn_data

[✓] Match Conditions

Conditions: genus='Apaloderma'

The dialog above will generate the SQL SELECT statement "SELECT * FROM akn_data WHERE genus='Apaloderma';".

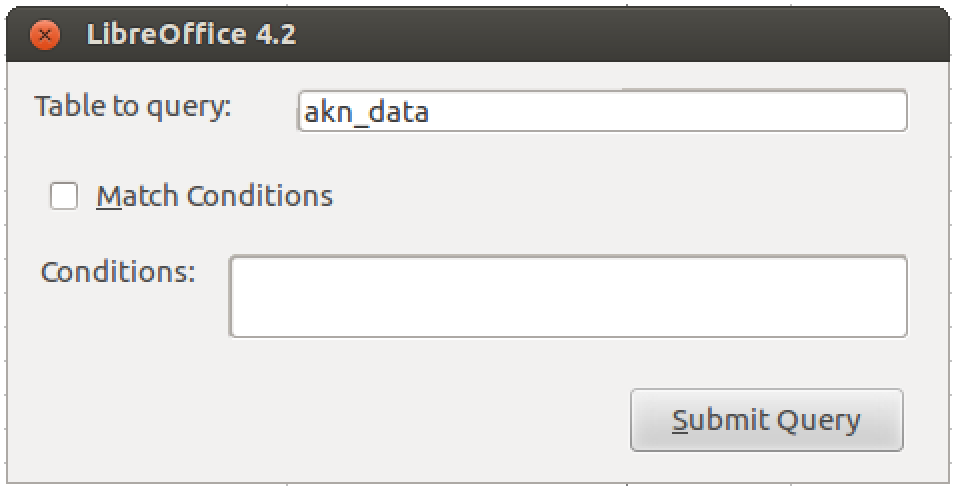
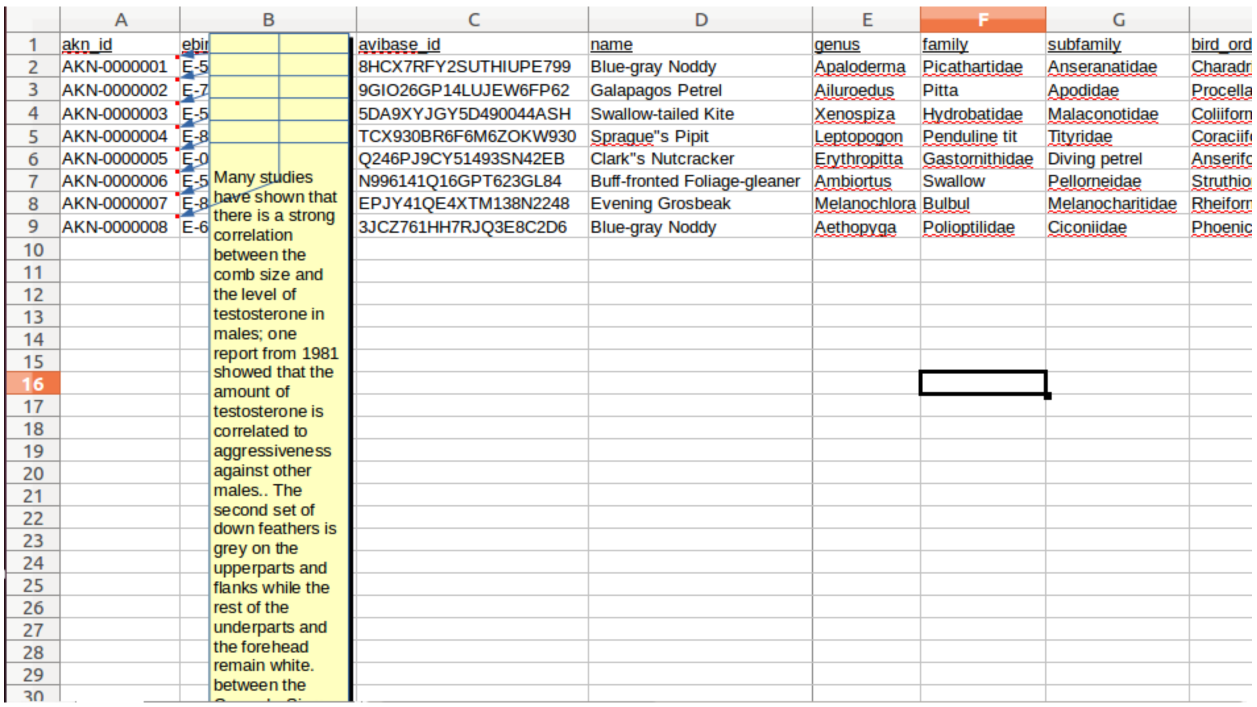

LibreOffice 4.2

Table to query: akn_data

[ ] Match Conditions

Conditions:

The dialog above will generate the SQL SELECT statement "SELECT * FROM akn_data;". From this query the spreadsheet will be filled out as shown in the image below with the column names listed across the top, tuples listed in the spreadsheet, and annotations in comment boxes attached to the appropriate cells.



	A	B	C	D	E	F	G	
1	akn_id	ebir	avibase_id	name	genus	family	subfamily	bird_ord
2	AKN-0000001	E-5	8HCX7RFY2SUTHIUPE799	Blue-gray Noddy	Apaloderma	Picathartidae	Anseranatidae	Charadr
3	AKN-0000002	E-7	9GIO26GP14LUJEW6FP62	Galapagos Petrel	Ailuroedus	Pitta	Apodidae	Procella
4	AKN-0000003	E-5	5DA9XYJGY5D490044ASH	Swallow-tailed Kite	Xenospiza	Hydrobatidae	Malaconotidae	Coliiforn
5	AKN-0000004	E-8	TCX930BR6F6M6ZOKW930	Sprague"s Pipit	Leptopogon	Penduline tit	Tityridae	Coraciif
6	AKN-0000005	E-0	Q246PJ9CY51493SN42EB	Clark"s Nutcracker	Erythropitta	Gastornithidae	Diving petrel	Anserifc
7	AKN-0000006	E-5	N996141Q16GPT623GL84	Buff-fronted Foliage-gleaner	Ambiortus	Swallow	Pellorneidae	Struthio
8	AKN-0000007	E-8	EPJY41QE4XTM138N2248	Evening Grosbeak	Melanochlora	Bulbul	Melanocharitidae	Rheiforn
9	AKN-0000008	E-6	3JCZ761HH7RJQ3E8C2D6	Blue-gray Noddy	Aethopyga	Polioptilidae	Ciconiidae	Phoenic

Many studies have shown that there is a strong correlation between the comb size and the level of testosterone in males; one report from 1981 showed that the amount of testosterone is correlated to aggressiveness against other males.. The second set of down feathers is grey on the upperparts and flanks while the rest of the underparts and the forehead remain white. between the

The comment boxes are able to be resized to better fit the data and hidden to reduce clutter.

The code below implements querying the database with for the given table and conditions, displaying the results in the spreadsheet, getting the annotations for the table returned, and determining the cells to which it should add the annotations.

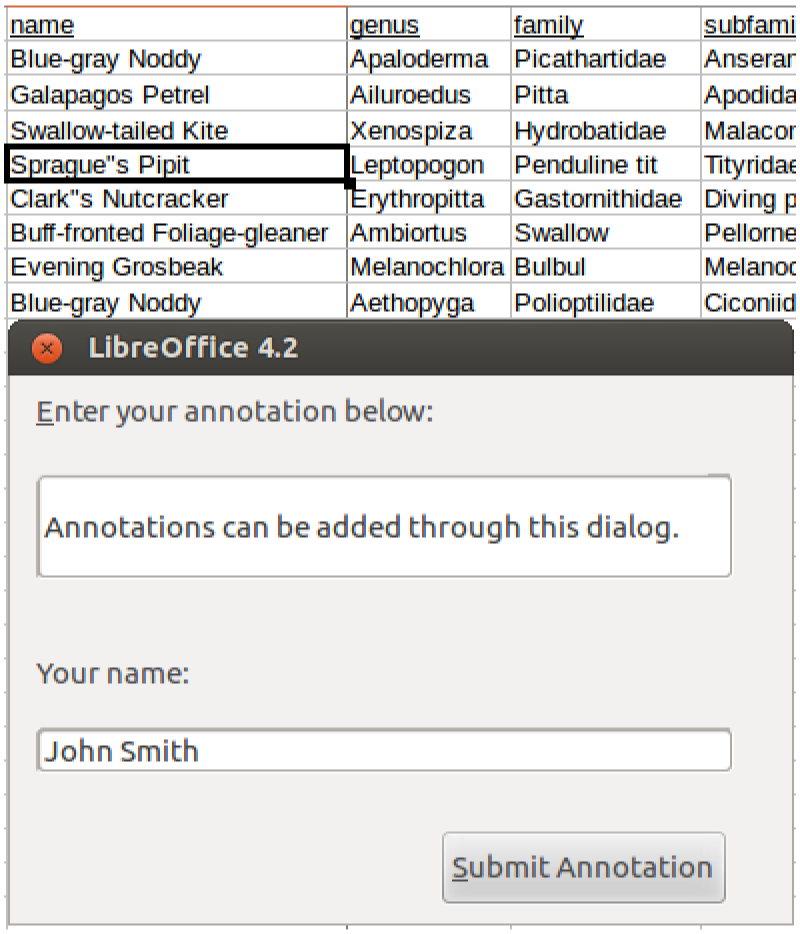
```
Sub queryDB
       Dim QueryDefinition As Object
       Dim queryCond, queryString As String
       Dim Statement as Object
       Dim Cell As Object
       Dim row
       Dim col
       Dim table As String

       row = 0
       Doc = ThisComponent
       Sheet = Doc.Sheets(0)

       clearSheet()

       DialogLibraries.LoadLibrary("Standard")

       QueryDefinition = createUnoService("com.sun.star.sdb.QueryDefinition")
       queryTable = Dlg.getControl("queryTable").Text
       If(Dlg.getControl("matchCond").State)  Then
        queryCond = Dlg.getControl("queryCond").Text
        queryString = "SELECT * FROM " & queryTable & " WHERE " & queryCond & ";"
       Else
        queryString = "SELECT * FROM " & queryTable & ";"
       EndIf

       MsgBox queryString
       Statement = Connection.createStatement()
       Statement = Connection.PrepareStatement(queryString)

       ResultSet = Statement.executeQuery()

       Statement = Connection.createStatement()
       Statement = Connection.PrepareStatement("SELECT * FROM " & queryTable & ";")

       wholeTableSet = Statement.executeQuery()

       If Not IsNull(ResultSet) Then
        While col < ResultSet.getMetaData().getColumnCount()
               Cell = Sheet.getCellByPosition(col, 0)
               Cell.CharUnderline = 1
               Cell.String = ResultSet.getMetaData().getColumnLabel(col + 1)
               col = col + 1
        Wend
        While ResultSet.next
               col = 0
               While col < ResultSet.getMetaData().getColumnCount()
                     Cell = Sheet.getCellByPosition(col, row + 1)
                     Cell.String = ResultSet.getString(col + 1)
                     getAnnotations(queryTable, col, getTupleID(ResultSet,
ResultSet.getRow()), Cell)
                     col = col + 1
               Wend
               row = row + 1
        Wend
       End If
       ResultSet.first()
End Sub

Sub getAnnotations (table As String, col As Integer, tuple As Integer, cell As Object)
    Dim Statement as Object
    Dim AnnoResultSet as Object

    If(tuple<>-1) Then
        Statement = Connection.createStatement()
       Statement = Connection.PrepareStatement("SELECT data_anno.tuple_id,
data_anno.tuple_column, data_anno.table_name, anno_table.value FROM anno_table INNER JOIN
data_anno ON anno_table.id=data_anno.id WHERE data_anno.tuple_id=" & tuple & " AND
data_anno.table_name='" & table & "';")
       AnnoResultSet = Statement.executeQuery()

       While(AnnoResultSet.next)
               If(tuple=AnnoResultSet.getInt(1)) Then
                      If(col=CInt(RIGHT(LEFT(AnnoResultSet.getString(2), 2), 1))) Then
                             Annotations = Sheet.getAnnotations()
                             CellAddr = cell.CellAddress
                             Annotations.insertNew(CellAddr, cell.annotation.String &
chr(13) & AnnoResultSet.getString(4))
                             cell.Annotation.isVisible = True
                      End If
               End If
       Wend
    EndIf
End Sub
```

The clearSheet sub called by queryDB is used to remove any excess data from the spreadsheet in preparation for displaying the results of a new query. In order to create this sub the LibreOffice Macro Recorder was used to record the process of selecting and of the cells on the spreadsheet, clearing all their contents, and selecting cell A1. The original code generated has been modified slightly and is given below.

```
Sub clearSheet()

    Dim Document   As Object
    Dim Dispatcher As Object

    Document   = ThisComponent.CurrentController.Frame
    Dispatcher = createUnoService("com.sun.star.frame.DispatchHelper")

    Dispatcher.executeDispatch(document, ".uno:SelectAll", "", 0, Array())

    Dispatcher.executeDispatch(document, ".uno:ClearContents", "", 0, Array())

    Dim args4(0) As New com.sun.star.beans.PropertyValue
    args4(0).Name = "ToPoint"
    args4(0).Value = "$A$1"

    Dispatcher.executeDispatch(document, ".uno:GoToCell", "", 0, args4())
End Sub
```

5.3 Adding New Annotations

Adding annotations through the Postgres CLI requires updating one tables with information about the annotation and another with information about which tuple and column it is associated with. Using this interface, annotations can now be added to tables which have been queried simply by selecting a cell and filling in the annotation text and the author name in the proper dialog, and submitting the annotation.



name	genus	family	subfami
Blue-gray Noddy	Apaloderma	Picathartidae	Anserar
Galapagos Petrel	Ailuroedus	Pitta	Apodida
Swallow-tailed Kite	Xenospiza	Hydrobatidae	Malacor
Sprague"s Pipit	Leptopogon	Penduline tit	Tityridae
Clark"s Nutcracker	Erythropitta	Gastornithidae	Diving p
Buff-fronted Foliage-gleaner	Ambiortus	Swallow	Pellorne
Evening Grosbeak	Melanochlora	Bulbul	Melanoc
Blue-gray Noddy	Aethopyga	Polioptilidae	Ciconiid

LibreOffice 4.2

Enter your annotation below:

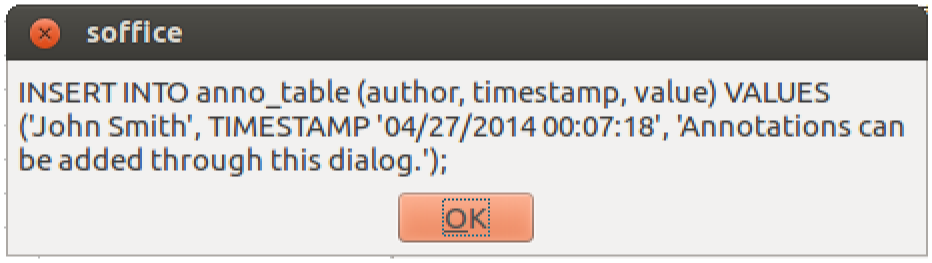
Annotations can be added through this dialog.

Your name:

John Smith

Submit Annotation

This information is used to first generate a SQL INSERT statement to fill in the appropriate data in the data_anno table.

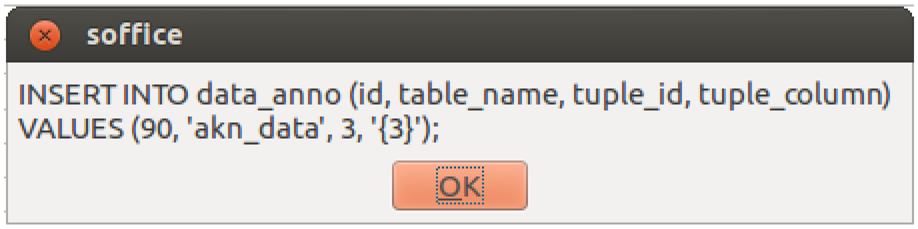


```
INSERT INTO anno_table (author, timestamp, value) VALUES
('John Smith', TIMESTAMP '04/27/2014 00:07:18', 'Annotations can
be added through this dialog.');
```

And then from the selected cell, another INSERT statement is generated to properly link the annotation with the appropriate tuple and column by matching the data in the row containing the selected cell with the same row in the previously selected table.

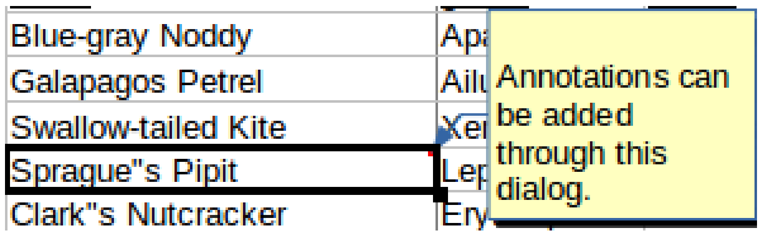
Because of the method of storing which cells annotations are attached to and the ResultSet data structure used in LibreOffice BASIC to return the results of SQL queries some limitations were introduced into the system. Because there is no way to get the index of a selected tuple, in reference to the entire table that it exists in, with the information supplied in a ResultSet, the tuple is matched by checking it against each row in the table, checking the each column until the whole row is matched, because it is not guaranteed that all columns of a table contain distinct values which could lead to false-positives when matching data.

After the tuple index is found a SQL INSERT statement is generated to store information attaching the annotation to the matched tuple and column.



```
INSERT INTO data_anno (id, table_name, tuple_id, tuple_column)
VALUES (90, 'akn_data', 3, '{3}');
```

After the two annotation tables are updated, the spreadsheet is updated with the new annotation using the GetAnnotations Sub at the end of Section 5.2.



Blue-gray Noddy	Apa
Galapagos Petrel	Ailu
Swallow-tailed Kite	Xei
Sprague"s Pipit	Lep
Clark"s Nutcracker	Ery

Annotations can be added through this dialog.

The code implementing the functionality in this section is given below.

```
Sub newAnnotation()
    Dim QueryDefinition As Object
    Dim AnnoText As String
    Dim AnnoAuthor As String
    Dim AnnoQuery As String
    Dim Statement As Object
    DialogLibraries.LoadLibrary("Standard")


       QueryDefinition = createUnoService("com.sun.star.sdb.QueryDefinition")
       AnnoText = Dlg.getControl("newAnno").Text
       AnnoAuthor = Dlg.getControl("authName").Text

       AnnoQuery = "INSERT INTO anno_table (author, timestamp, value) VALUES " & "('" &
AnnoAuthor & "', TIMESTAMP '" & Now & "', '" & AnnoText & "');"
       MsgBox AnnoQuery
       Statement = Connection.createStatement()
       Statement = Connection.PrepareStatement(AnnoQuery)
       Statement.executeUpdate()

       AnnoQuery = "SELECT MAX(id) AS highestId FROM anno_table"
       Statement = Connection.createStatement()
       Statement = Connection.PrepareStatement(AnnoQuery)
       MaxIDResultSet = Statement.executeQuery()
       MaxIDResultSet.next
       nextID = MaxIDResultSet.getInt(1)

    AnnoQuery = "INSERT INTO data_anno (id, table_name, tuple_id, tuple_column) VALUES "
& _
                      "(" & nextID & ", '" & queryTable & "', " & getTupleID(ResultSet,
ThisComponent.getCurrentSelection().CellAddress.Row) & _
                      ", '{" & ThisComponent.getCurrentSelection().CellAddress.Column &
"}');"
    MsgBox AnnoQuery
    Statement = Connection.createStatement()
       Statement = Connection.PrepareStatement(AnnoQuery)
       Statement.executeUpdate()

       ThisComponent.getCurrentSelection.annotation.String = ""
       GetAnnotations(queryTable, ThisComponent.getCurrentSelection().CellAddress.Column,
getTupleID(ResultSet, ThisComponent.getCurrentSelection().CellAddress.Row),
ThisComponent.getCurrentSelection)
```

```
End Sub

Function getTupleID(CheckResultSet As Object, row As Long)
    Dim match As Boolean
    match = false
    Dim i As Integer
    i = 0
    CheckResultSet.first
    wholeTableSet.first
    CheckResultSet.absolute(row)
    While wholeTableSet.next
        If(i=0) Then
               wholeTableSet.first
        EndIf
        If(CheckResultSet.getString(1)=wholeTableSet.getString(1)) Then
               col = 2
               match = true
               While(col < wholeTableSet.getMetaData().getColumnCount())
                      If(CheckResultSet.getString(col)=wholeTableSet.getString(col)) Then
                      Else
                             match = false
                      EndIf
                      col = col + 1
               Wend
               If(match) Then
                      getTupleID = i
               EndIf
        EndIf
        i = i+1
    Wend
    If Not match Then
        getTupleID = -1
    EndIf
End Function
```

Conclusion

Interfacing with a SQL database system through LibreOffice BASIC through built in functionality is relatively straight forward. However, for the purpose of displaying query results and adding annotations, extensive checking of several tables is required to create properly formatted SELECT and UPDATE statements. Constructing these statements allowed for the process of viewing data with its annotations to be greatly simplified, as well as simplifying the process of adding new annotations to the database. From the functionality implemented in this system thus far, the next step would be to add a visualization for annotation summaries.

Appendix

Installation of the postgres system from the custom InsightNotes source code was completed using the following procedure:

1) pre-requirement
JDK for running eclipseCDT
link: http://www.blogs.digitalworlds.net/softwarenotes/?p=41
EclipseCDT any version
for building the project and debugging the entire system

Packages(some package are optional) I only list the necessary ones.
libreadline5-dev, zlib1g-dev
sudo apt-get install package name

2) change directory to the root of extracted postgresql
1) configuration:
./configure - - prefix= /path/project - -enable-depend - - enable-cassert   - - enable-debug

2) import the source code into eclipse
file->import->
type: existing code as Makefile Project
remember: Language: C(checked) C++(not)
Toolchain for Indexer: Linux GCC
click finish
3) build the full project

4) create one target: install
5) right click the target "install".
Installation complete.
6) try to run dataBase
change directory to the root of extracted postgresql
export PATH=$path/project:$PATH
export PGDATA=DemoDir
initdb
7) then can run your dataBase as you like. Follow the documentation in website.

More detailed instructions for installing Postgres from source code can be found in the online PostgresSQL Documentation at http://www.postgresql.org/docs/9.1/static/install-procedure.html.