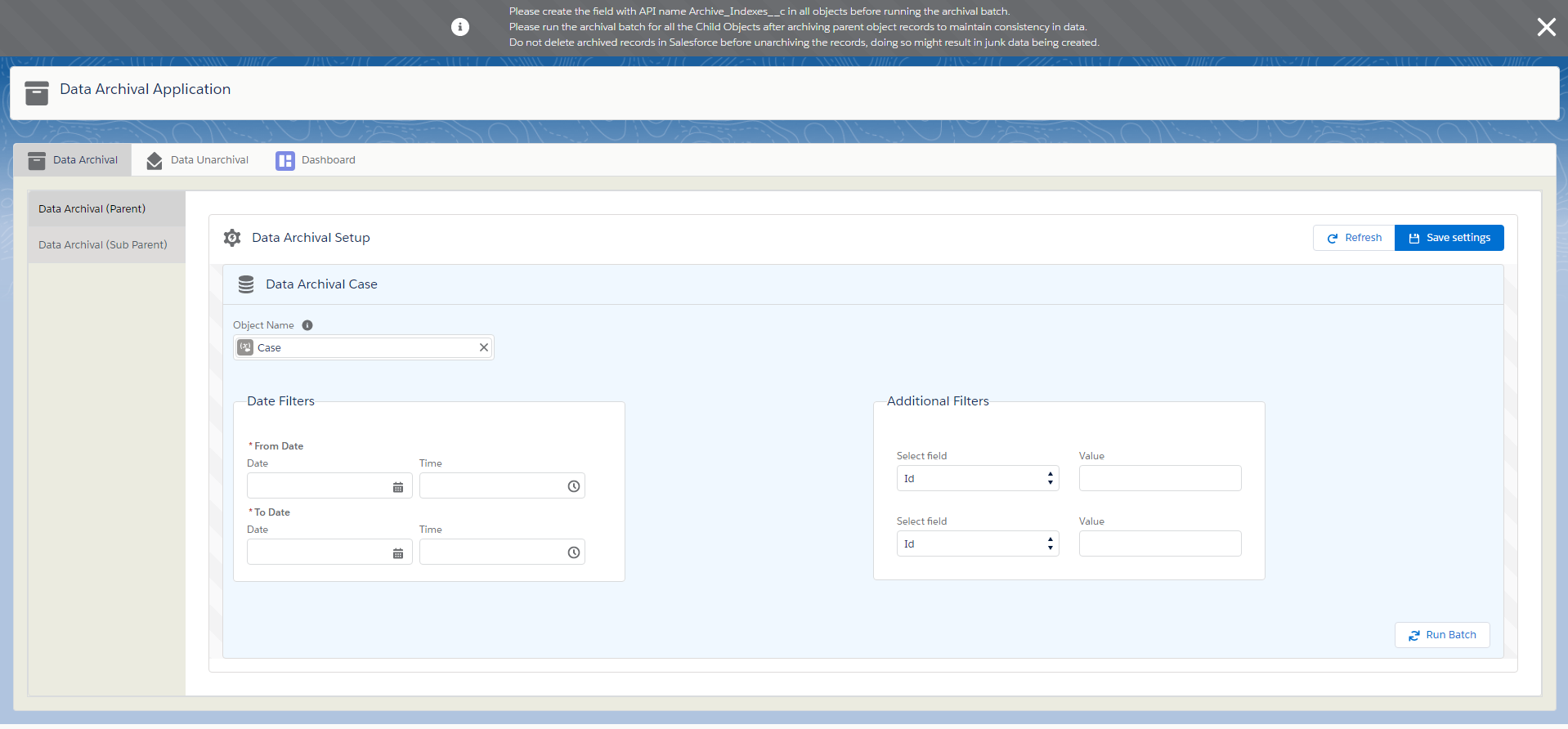
Data Archival Application

This application allows users to archive records from any custom or standard objects and allows for unarchiving the records. It also provides a way to visualize the amount of data archived segregated by the object Name using a dashboard.

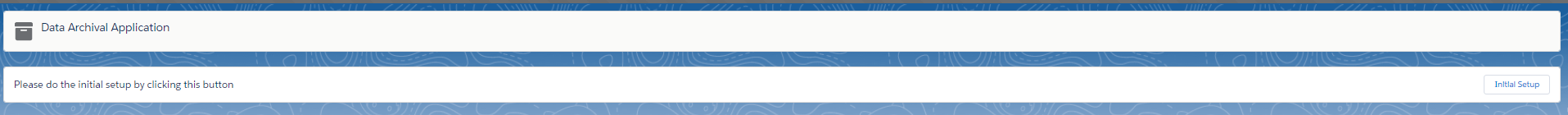
Archiving is done by selecting a parent object and running a batch by providing appropriate filters. Once the batch is submitted for parent, we can run the batch for the child object records and the 2nd level of child objects.

Upto 2 levels of child object records can be traversed and archived.



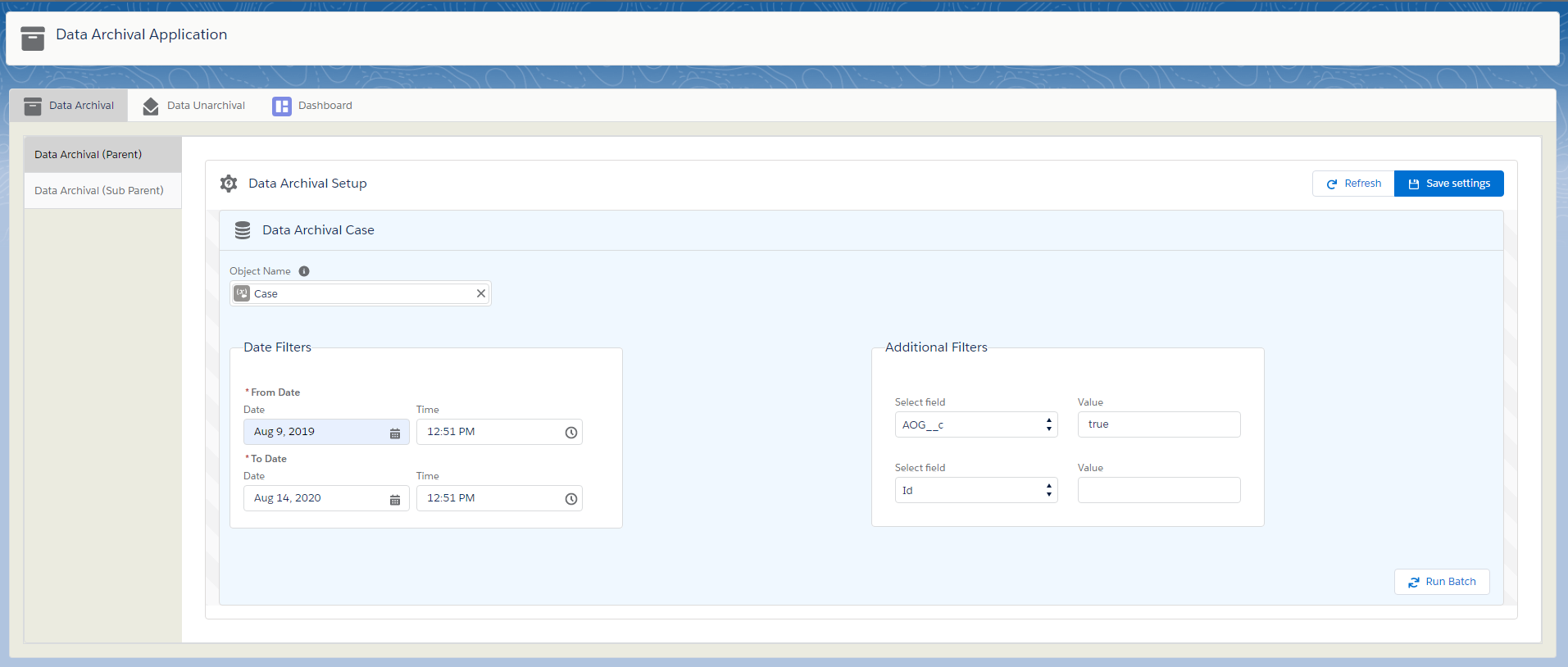
**Initial Setup:**

During the Initial start of application, we will see a button Initial setup. Once clicked it will do the necessary setup for archival and dashboard and then we can start using the application.

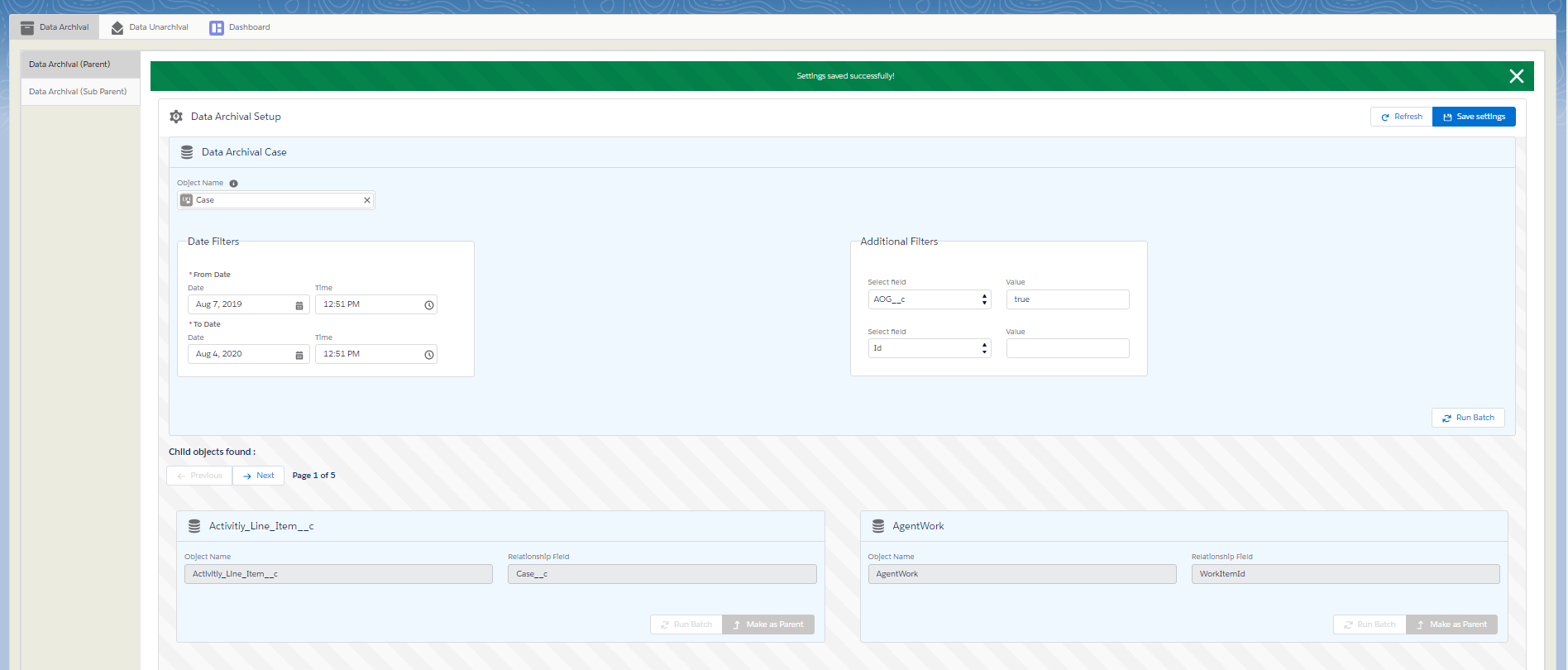


**Tab 1 - Data Archival Tab:**

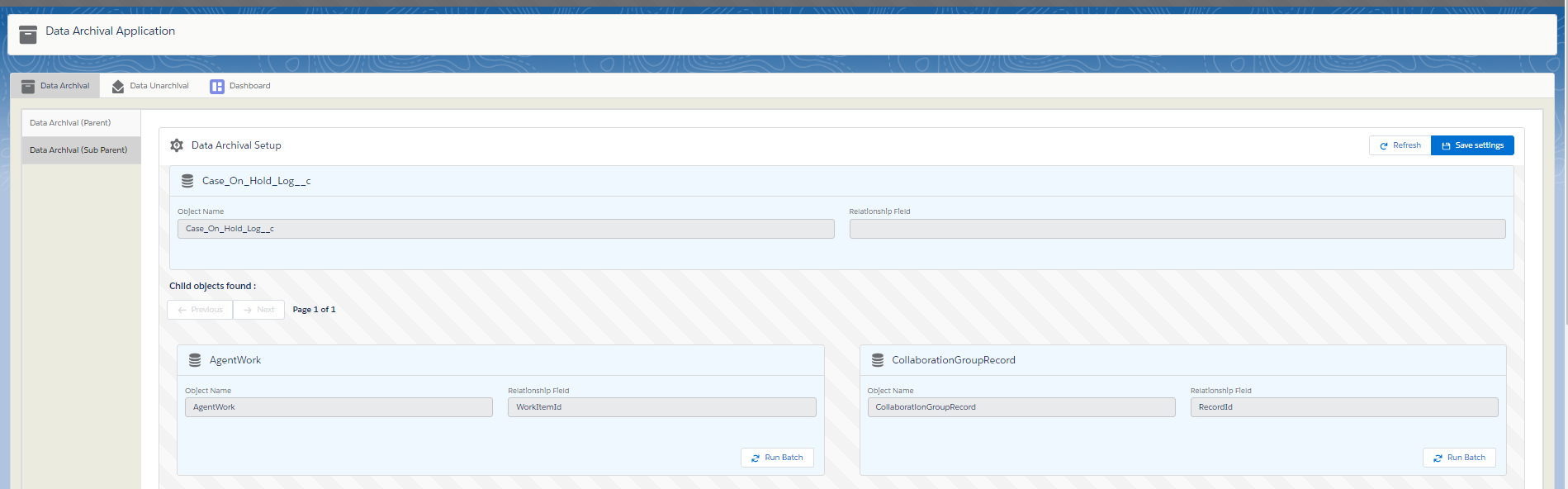
Select an object and provide date filters and if required 2 additional field filters and save settings.



Once the settings are saved the child objects of the selected parent will be shown below the setup section. The user can run the batch for the parent object first and then the access to running child object batches are enabled once it is completed. Refresh button can be used to refresh the statuses of each object batch. After archival the archived records will have blank values and any text area field values will be added to the record as an attachment.



Once the batch is run for parent object, we can run the batch for child objects similarly and make the child object as the new parent to run the archival batch for 2nd level child objects.



**Components used:**

**Lightning Components:**

dataArchivalApplication – Standalone application for data archival components.

ArchivalHeader – Used to display the header for the application with the theme and icon.

dataArchivalSectionsComponent- displays the tabs for data archival, unarchival and dashboard

dataArchivalComponent – holds the vertical tabs for Parent and sub parent. Also contains the DataArchivalRowComponent for displaying object details along with refresh and save settings button.

DataArchivalRowComponent – displays the object name, date filter and 2 additional field filters to be applied if the object is the first selected parent object. For all the child objects and the 2nd level child objects it will show the object name and the relationship field api name to the parent.

refreshParent – event is triggered when we make a new child object as the Sub parent (to get its child objects). This refreshes the archivalComponent and changes the tab to the Sub Parent.

searchComponent – used for searching objects

searchResults- displays the matched results as user types object names

clearValueEvent – triggered when the object selection is cleared from the field

oSelectedValueEvent – triggered when an object is selected

refreshParent – triggered when one of the parent objects is made as the new child and refreshes the current tab to get the updated settings.

setupComponent – This is a component that will be displayed during the initial start of the application. This will create a record in the dataArchive\_\_b big object with the header record of indexvalue=’0’

dateTimeStampIndex=’HeaderIndex’

recordjson1\_\_c=a json string of all the object names, record count where the record count of all objects is initialized to 0.

Ex: {“Account”:0, ”Contact”:0,..}

**Objects:**

Archival\_Settings\_\_c custom setting: This is used to track the index number for EmailArchive\_\_b big object where email message records are archived, to maintain uniqueness of big object indexes. Whenever an EmailArchive\_\_b big object record is inserted, we use this to get the current big object index to be used, update the setting record with the new index and save it.

|  |  |
| --- | --- |
| **Field Name** | **Purpose** |
| Name | Should be equal to ‘currentIndex’ |
| Record\_id\_\_c | Stores the record Id of the same record to update without querying |
| indexNumber\_\_c | Stores the next index number to be used for big object insertion. |

Data\_Archival\_Settings\_\_c custom setting: This is used to track the index number for DataArchive\_\_b big object where all sobject records except EmailMessage are archived, to maintain uniqueness of big object indexes. Whenever a DataArchive\_\_b big object record is inserted, we use this to get the current big object index to be used, update the setting record with the new index and save it.

|  |  |
| --- | --- |
| **Field Name** | **Purpose** |
| Name | Should be equal to ‘currentIndex’ |
| Record\_id\_\_c | Stores the record Id of the same record to update without querying |
| indexNumber\_\_c | Stores the next index number to be used for big object insertion. |

Data\_Archival\_Object\_Settings\_\_c custom setting: This will have records for each parent and child object and the status of their batch class execution.

|  |  |
| --- | --- |
| **Field Name** | **Purpose** |
| Name | To maintain uniqueness this value will be the current date time stamp |
| Archival\_From\_Date\_\_c | Created date of records from which we need to archive. |
| Archival\_To\_Date\_\_c | Created date of records to which we need to archive. |
| FilterField\_1\_\_c | 1st filter Field name |
| FilterField\_2\_\_c | 2nd filter Field name |
| FilterValue\_1\_\_c | 1st filter Field Value |
| FilterValue\_2\_\_c | 2nd filter field value |
| Hierarchy\_level\_\_c | Level of hierarchy ex: first selected object will have hierarchy level=1  The child objects of this object will have a hierarchy level=2  The grandchild objects of this first selected object will have a hierarchy level=3 |
| Object\_Name\_\_c | Name of the object |
| Object\_type\_\_c | Parent / child |
| Parent\_Setting\_Name\_\_c | Setting name of this object’s parent (similar to lookup relationship but instead of Id we use name for faster retrieval) |
| Status\_\_c | Archival batch job status |
| Submitted\_By\_\_c | Id of the user who submitted |
| Relationship\_field\_\_c | This is the field name of the child object which has the relationship to the parent (lookup field or master detail field name) |

Case object: There are 2 fields created for holding the index values of the archived records in big objects. It is used for unarchival.

|  |  |
| --- | --- |
| **Field Name** | **Purpose** |
| Archive\_Indexes\_\_c | Stores the index value of the big object dataArchive\_\_b where the data is stored for this record. **This field should be created on every object that needs archiving.** |
| Email \_archive\_indexes\_\_c | Stores the index value of the big object emailArchive\_\_b where the email records are stored. |

DataArchive\_\_b big object: Archives the records of any sobject in one of the 12 long text area fields in JSON format.

**1 record of big object = 12 records of sobjects**

|  |  |  |
| --- | --- | --- |
| **Field Name** | **index** | **Purpose** |
| indexNumber\_\_c | Yes | The primary index field where the values are in increments of 1. The new index number value for inserting new records are got from data\_Archival\_settings\_\_c custom setting. |
| Datetimestampindex\_\_c | Yes | The secondary index field which holds the current date time stamp of record creation. |
| RecordJson0\_\_c |  | Holds a single record of any sobject in JSON format |
| RecordJson1\_\_c |  | Holds a single record of any sobject in JSON format |
| RecordJson2\_\_c |  | Holds a single record of any sobject in JSON format |
| RecordJson3\_\_c |  | Holds a single record of any sobject in JSON format |
| RecordJson4\_\_c |  | Holds a single record of any sobject in JSON format |
| RecordJson5\_\_c |  | Holds a single record of any sobject in JSON format |
| RecordJson6\_\_c |  | Holds a single record of any sobject in JSON format |
| RecordJson7\_\_c |  | Holds a single record of any sobject in JSON format |
| RecordJson8\_\_c |  | Holds a single record of any sobject in JSON format |
| RecordJson9\_\_c |  | Holds a single record of any sobject in JSON format |
| RecordJson10\_\_c |  | Holds a single record of any sobject in JSON format |
| RecordJson11\_\_c |  | Holds a single record of any sobject in JSON format |

**HeaderRecord**:

**HeaderRecord**:

There will be one record which will be the first record in the big object with index ‘0’ and datetimestamIndex\_\_c=’HeaderIndex’ where additional information from big object records are stored.

The recordJson0\_\_c will hold the unarchived indexes from the big object records.

1. Ex: there is a record at recordJson3\_\_c at the indexNumber\_\_c 34 in the big object. Now we unarchive that record which will put the value of that record to the string **“null”**. When this happens, the unarchived index will be tracked at the position recordJson0\_\_c at the headerRecord in the below format:

**columnNumber where the record resides ‘ – ‘ index number where the record resides ;**

1. so for the above example the recordJson0\_\_c at the header record will be **3-34;**
2. if a new record is removed at recordJson5\_\_c at index number 145, the new unarchived index will be appended to the old one in the recordJson at header like **3-34;5-145;**
3. for next archivals the records are stored at these unarchived positions first.

The recordJson1\_\_c will hold the count of records for each object stored in the big object in the form of a JSON map.

Ex : {

“Account” : 40000,

“Opportunity” : 50000…… etc

}

EmailArchive\_\_b big object: Archives the records of **emailMessage** object in one of the 12 long text area fields in JSON format.

**1 record of big object = 12 records of emailmessage**

|  |  |  |
| --- | --- | --- |
| **Field Name** | **index** | **Purpose** |
| indexNumber\_\_c | Yes | The primary index field where the values are in increments of 1. The new index number value for inserting new records are got from Archival\_settings\_\_c custom setting. |
| Datetimestampindex\_\_c | Yes | The secondary index field which holds the current date time stamp of record creation. |
| RecordJson0\_\_c |  | Holds a single record of emailmessage record in JSON format |
| RecordJson1\_\_c |  | Holds a single record of emailmessage record in JSON format |
| RecordJson2\_\_c |  | Holds a single record of emailmessage record in JSON format |
| RecordJson3\_\_c |  | Holds a single record of emailmessage record in JSON format |
| RecordJson4\_\_c |  | Holds a single record of emailmessage record in JSON format |
| RecordJson5\_\_c |  | Holds a single record of emailmessage record in JSON format |
| RecordJson6\_\_c |  | Holds a single record of emailmessage record in JSON format |
| RecordJson7\_\_c |  | Holds a single record of emailmessage record in JSON format |
| RecordJson8\_\_c |  | Holds a single record of emailmessage record in JSON format |
| RecordJson9\_\_c |  | Holds a single record of emailmessage record in JSON format |
| RecordJson10\_\_c |  | Holds a single record of emailmessage record in JSON format |
| RecordJson11\_\_c |  | Holds a single record of emailmessage record in JSON format |

**HeaderRecord**:

There will be one record which will be the first record in the big object with index ‘0’ and datetimestamIndex\_\_c=’HeaderIndex’ where additional information from big object records are stored.

**Classes:**

bigObjectQueryService –

Used to access the big object records in the test classes. big object can’t be queried from test classes and it will return an empty list if we call a method that is querying a big object.

To avoid this we have created a class that has a private Map<string,List<sobject>> and a method called getQuery.

The getQuery method is called with the query in the controller.

Ex: **bigObjectQueryService.getQuery('SELECT id, MasterLabel,DeveloperName**

**FROM dataArchive\_\_b** **‘);**

This method will actually query the records when not in Test Context and returns a list of sobjects.

When we call from test class it will return the value contained in the map. So, in the test class we need to do the below:

**bigObjectQueryService.bigObjectQueryMap.put('SELECT id, MasterLabel,DeveloperName**

**FROM dataArchive\_\_b‘, bigobjectList );**

Where bigobjectListis the list of dataArchive\_\_b record instances that we want to be returned in the controller when the query executes.

dataArchivalController –

Main controller for the Data Archival tab.

Methods:

getObjectFields – used to get all the editable fields from the selected sobject that would serve as out first parent object.

getSettings – gets all the custom setting records created by the current user and based on the hierarchy level. For the first level of parent object, the method returns the parent object and the immediate child objects that are displayed in the ‘Child objects found’ section.

deleteSettings – deletes all settings created by the current user.

saveSettings – saves the custom setting records for an object and inserts a list of custom setting records for its children based on the hierarchy level of parent.

Ex: if the hierarchy level of parent object custom setting record is 1 then the child object custom setting records will be 2.

runBatch- It accepts 2 parameters. One is object type, and another is object name.

If object type is parent, then we run the dataArchivalBatch with a batch size in **multiples of 12 and 10 (strictly to be used as multiples).**

If object type is child or if the hierarchy is at level 2 or 3, then we run the childArchivalBatch.

If the object type is ‘EmailMesage’ then irrespective of it being a parent or child, the emailArchivalBatch is run with a batch size in **multiples of 12.**

getBatchStatus – used to get the running batch status for archival.

changeParent – It is used for ‘make as parent’ functionality. Here we delete the custom setting records for any object at hierarchy level 3 or (hierarchy level 2 and object type=parent). Then we create new custom setting records for the object passed to the method and for the child objects of the object passed.

emailArchivaLbatch –

Archival batch will be called from the controller when the selected sobject is emailmessage. The batch will get all the email messages having no attachments for the selected date filters in the application and then runs a query to get all the email messages with a batch size that is a multiple of 12.

The execute method runs on these records in batches of 12 and creates 1 record of big object emailArchive\_\_b for 12 records of emailmessage. Each record is saved in one of the 12 **recordJSON** fields of the big object in JSON format. The index for this big object record is got from one record in Archival\_settings\_\_c custom setting record.

**1 big object record -12 email message records in JSON format**

Once the batches are completed, we call the same batch class again from the finish method until there are no email message records without attachments are left for the given date filter.

The value of the indexNumber where the records are archived in the big objects are to be added in **email\_archive\_indexes\_\_c** field of the parent Case record of the emailmessage that is being archived. For this the field has to be created in case object.

childArchivalBatch –

We run this job on the child object settings that were created when we select the parent object that needs to be archived. If the setting hierarchy is 2 or 3, we run this batch by passing the object name to it.

We create a query on the parent object of the object passed to the batch. This parent object will be the one which was first selected for archival and has the hierarchy level =1.

Once we get all the records of the parent by providing the date and filed filters that were submitted by user, we run the dataArchivalBatch by passing that child object name that was passed to childArchivalBatch initially.

1. Pass the child object name to which we need to archive.
2. Get the parent of this child object that was initially selected.
3. Get the record Ids of the parent object that were already archived.
4. Pass the record Ids to dataArchivalBatch so it can run the batch on the child objects filtered by the parent record Ids.

dataArchivalBatch –

This is the main batch class that runs the archival. It nullifies all fields of the archived record and stores those values in the big object record field**. The long text and rich text area fields of the sobject record are created as an attachment to the record and not included in the archived record in big object.**

When we run this batch for an object that is at hierarchy level 1 (user selected object), It creates a query using the constructQuery method on the object and runs the batch on it. **The batch strictly runs with a size that is a multiple of 12 and 10 (ideally 600)**

12 because, since we need to save 12 sobject records in each record of big object.

10 because, batch runs in a chunk of 100 records which means if we run the batch size of 12, 100%12 which is 6 batches run with the batch size of 12 and the 7th batch runs with a batch size of 4 since the pki chunking is handled by salesforce system.

So ideally, we use 600 here.

Logic –

1. First, we get the header record of big object dataarchive\_\_b and get recordJson0\_\_c and recordJson1\_\_c.
2. We then get all the unarchived record indexes from recordJson0\_\_c which is in the format of

**columnNumber where the record resides ‘ – ‘ index number where the record resides ;**

we get the index numbers from this and query the big object records at those indexes and then fill the empty recordJson fields at those indexes with the new sobject records that need to be archived.

1. We then get the record from the latest index number that will be tracked in Data\_archive\_setting\_\_c records, query the big object record at that index and fill the empty fields of recordJson in that record.
2. Then we finally process the other sobject records and save them in the rest of the big object fields 12 at a time.
3. Once all big object record instances are created, we send it to insertBigObjectQueueable which commits the big object records in a different transaction.
4. The value of the indexNumber where the records are archived in the big objects are to be added in **archive\_indexes\_\_c** field of the record which is being archived. For this the field has to be created in all the objects that will be archived.
5. We also update the **recordJson1\_\_c** from the header record to update the record count for the current sobject type of records being archived.

insertBigObjectQueueable –

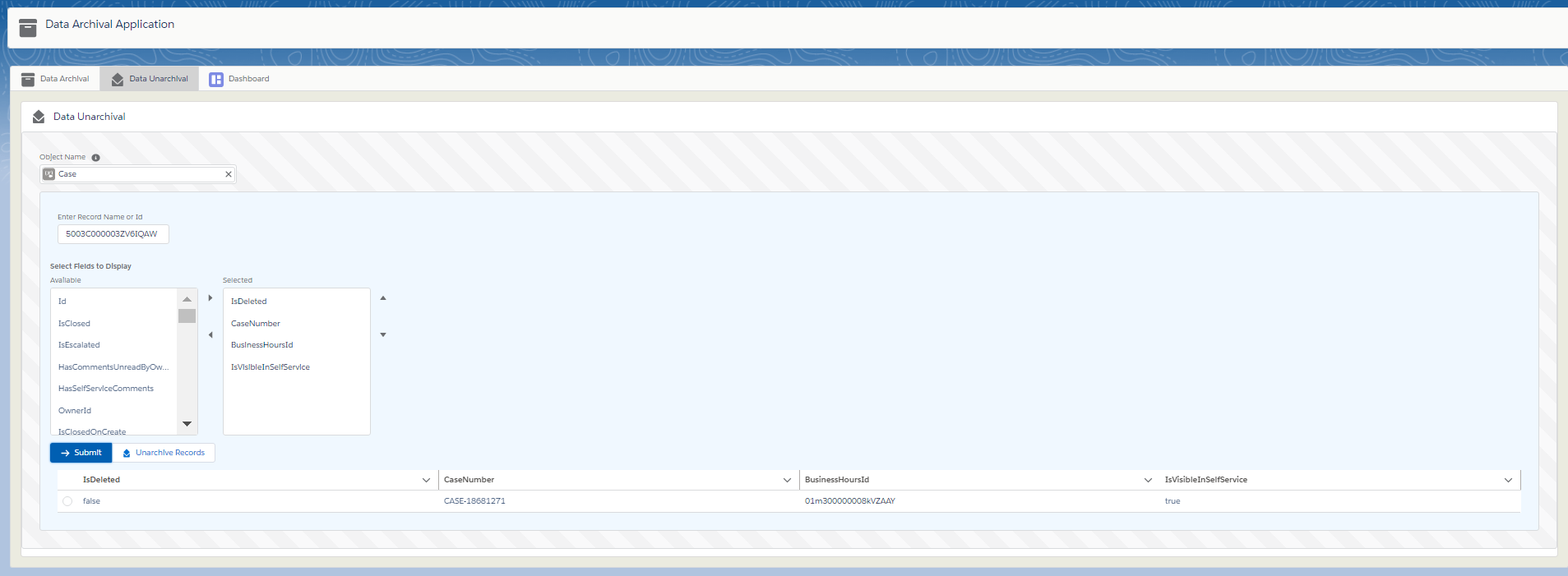
creates the big object records in a different transaction.

searchComponentController –

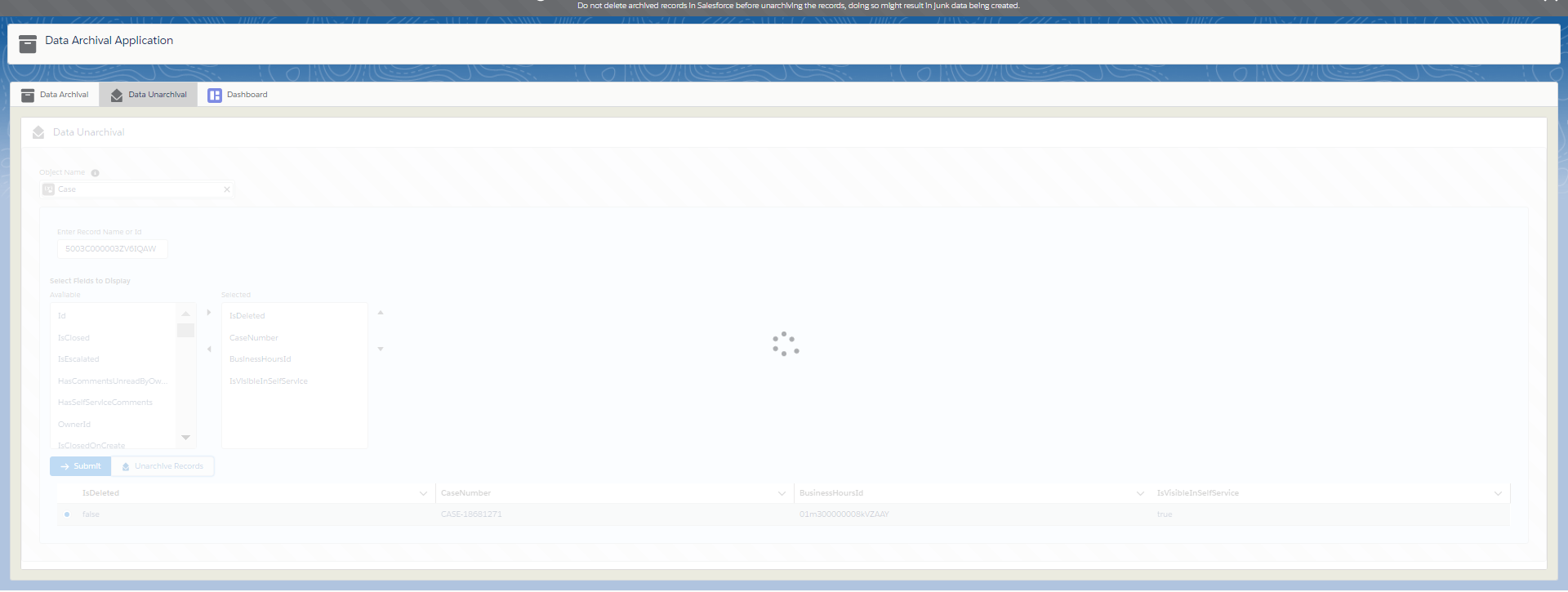
Used for searching the sobject record to be selected for archival by searchComponent.

**Tab 2 – Data Unarchival**

This tab allows users to select the object by API name and select the fields that need to be displayed in the table.



Once submitted the record is displayed in the table. Users can select the record and click on unarchive Record if they want to retrieve the record data into the actual record.



**Components used:**

**Lightning components:**

DataUnarchivalComponent – we use this component that is hosted in the Data Unarchival tab to select the object, search for the record using Id or name, select the fields and display the record that is archived in the big object dataArchive\_\_b.

It can be selected and unarchived which will restore the field values from the archived big object to the sobject record.

**Classes:**

DataUnarchivalLight –

Main class for data unarchival component.

Methods:

getUnarchivedRecords – This will get the sobject record with null values using the Id passed by user and gets the index value of the big object where the record is archived. The index value is obtained from the sobject record’s ‘archive\_indexes\_\_c’ field.

Using this index, we query the big object record and get all the 12 sobject records from the big object record fields (recordJson). We then find the record with the record id entered by the user among these 12 archived sobject records and return it.

unarchiveRecordList – This will accept the objectName and the unarchived Record from which we will get the **archive\_index\_\_c** value and query the big object record for this index.

1. Then we make the recordJson field where the record was found as ‘null’ and save the big object record.
2. Then we use the unarchived Index and update it in the below format in recordJson0\_\_c field of Header record.

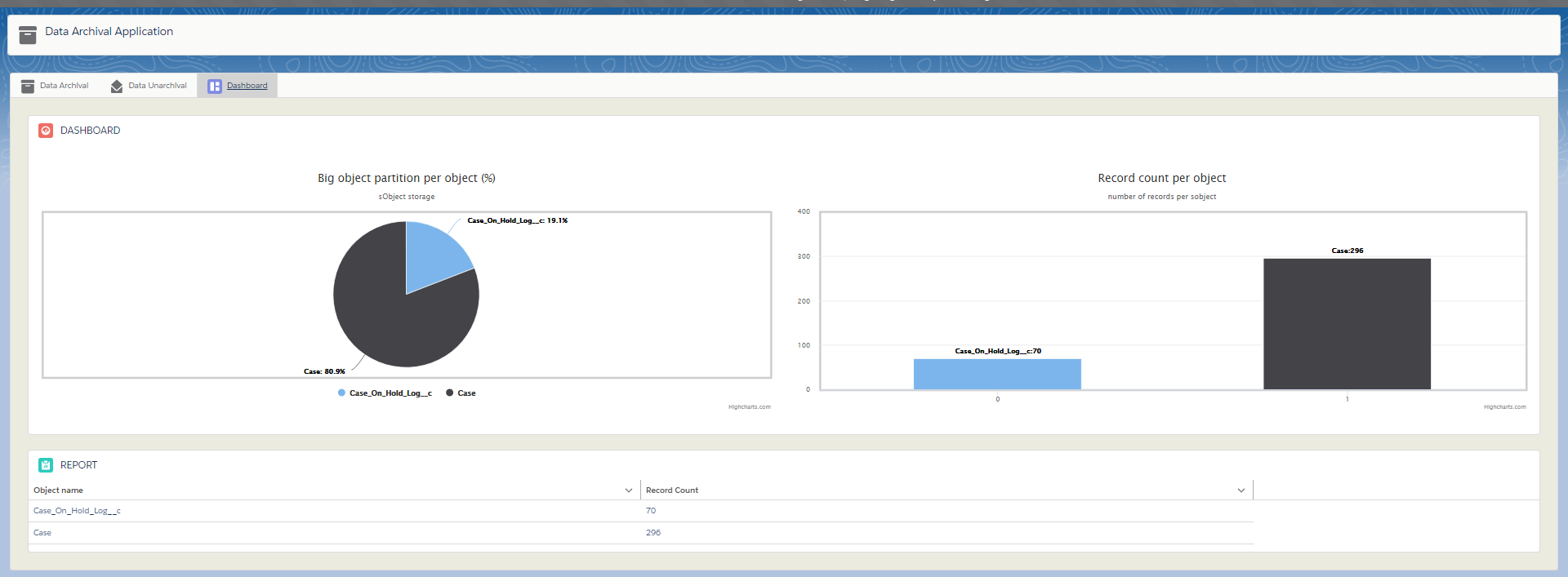
**columnNumber where the record resides ‘ – ‘ index number where the record resides ;**

1. Then we update the recordJson1\_\_c of the header record with the new count of records for the sobject type of the record which was unarchived.

**Tab 3 – Archival Dashboard**

This is a dashboard that shows a pie chart which shows the percentage distribution of archived records of each object in the archived dataset.

There is a bar graph and a report which shows the appropriate count of archived records of each object.

****

**Components used:**

**Lightning components:**

archivalDashboardComponent – uses the highcharts.js to display the pie chart and bar chart to show the record distribution in big object for each sobject.

**Classes:**

ArchivalHelperController –

This is the main controller for dashboard component and for creating setup for archival application.

Methods:

getSetupRequired – returns true if the big object header record is present with indexnumber=0 and headerIndex=’headersIndex’

createSetupForArchival- creates the big object header record with:

indexvalue=’0’

dateTimeStampIndex=’HeaderIndex’

recordjson1\_\_c=a json string of all the object names, record count where the record count of all objects is initialized to 0.

Ex: {“Account”:0, ”Contact”:0,..}

reportData – wrapper class to send the data to dashboard to feed the highcharts.js component to display data in the charts. Attributes are ‘name’ and ‘y’.

getReportingData – returns the data from recordJson1\_\_c from the big object header record and in the form of list<reportdata> wrapper list.

**Post Installation steps:**

1. Create a record in archival\_settings\_\_c and data\_archival\_settings\_\_c with

Name=’currentIndex’

Indexvalue\_\_c=’1’

Record\_id\_\_c= the record id for the same record after saving.

1. A field with api name ‘**Archive\_indexes\_\_c’** must be created in the object which we will archive.
2. A field with api name ‘**Email\_Archive\_indexes\_\_c’** must be created in the case object where we store the email archived indexes.
3. A header record with indexvalue\_\_c=’0’ and datetimestampindex\_\_c=’HeaderIndex’ must be created in both dataArchive\_\_b and emailArchive\_\_b big objects.
4. For dataarchive\_\_b header record:

recordjson1\_\_c=a json string of all the object names, record count where the record count of all objects is initialized to 0.

Ex: {“Account”:0, ”Contact”:0,..}