Airtable Base and Table Limit Solution:

Auto Archive Script and Multi-Base Data Explorer Documentation

By Rich Duchin

The **Auto Archive** Script and **Multi-Base Data Explorer** were designed to work together in managing and exploring large datasets across multiple Airtable bases. Together, they address the limitations posed by Airtable's storage constraints and provide users with the ability to archive, track, and analyze data efficiently.

**Auto Archive Script**

The **Auto Archive Script** automates the archival of records from a master Airtable table into multiple archive bases based on a specified time period (monthly, quarterly, semi-yearly, or yearly). It helps keep the master table manageable by dynamically creating new archive bases as needed, moving records into these bases, and logging the entire process for auditing.

1. **Automated Archiving:** The script automatically transfers older records from the master table into new archive bases based on time intervals (e.g., 2023\_Q1).
2. **Efficient Record Management:** Records are filtered to ensure that only unarchived records are processed, and they are batched to improve performance with large datasets.
3. **Dynamic Base Creation:** The script dynamically creates new archive bases as required, tracks the bases in an Archive Base Tracking table, and ensures consistency between the master table and the new archive bases.
4. **Logging and Auditing:** Each archival operation is logged, providing insights into the number of records processed, errors encountered, and the overall success of the operation.

**Multi-Base Data Explorer**

The **Multi-Base Data Explorer** allows users to explore and analyze data spread across multiple archived bases. By fetching data from these bases, merging it into a unified dataset, and providing filtering and visualization tools, the explorer makes it easy to work with fragmented data sources.

1. **Merging Archive Bases**: The explorer pulls records from multiple archived bases and merges them into a single dataset. This allows users to view and analyze data as if it came from a single source, despite being spread across multiple bases.
2. **Data Caching with IndexedDB:** To improve performance, the merged dataset is cached in IndexedDB, reducing the need for repeated API calls and ensuring faster data retrieval. The cache is automatically refreshed based on a defined time limit (e.g., every hour).
3. **Advanced Filtering and Visualization:** Users can apply filters (e.g., by year, publication, status, date range) and visualize the merged dataset with a chart component. This provides a user-friendly way to explore large datasets.
4. **Table Pagination:** Records are displayed in paginated views, making it easier to navigate through large datasets while applying filters and interacting with the data.

**Collaborative Functionality**

Together, the Auto Archive Script and Multi-Base Data Explorer provide a comprehensive solution for managing large Airtable datasets. The Archive Script ensures that the master table remains manageable by archiving older records into smaller bases. Meanwhile, the Multi-Base Explorer gives users the tools to explore and analyze the archived data across these bases in a unified and efficient manner. The integration of both tools allows users to seamlessly manage large datasets while maintaining performance and ease of use.

**Detailed Explanation of the Archive Script**

The **archive script** is responsible for automatically archiving records from a master Airtable table into separate archived bases. These archives are organized based on a specified time period (monthly, quarterly, semi-yearly, or yearly) to ensure that older records are moved out of the main table, keeping it manageable and within Airtable’s storage limits. Here’s a detailed breakdown of its functionality:

**1. Script Configuration and Inputs**

At the top of the script, a configuration block is used to capture necessary inputs from the user. The input.config function creates a UI for the user to provide the following:

* **Master Table**: The table from which records will be archived.
* **API Key**: A personal access token or API key used to authenticate Airtable API requests.
* **Archive Frequency**: The period (monthly, quarterly, semi-yearly, or yearly) that determines how records are grouped and archived.
* **Workspace ID**: The ID of the Airtable workspace where new archive bases will be created.

This configuration enables the script to be flexible and adaptable based on user preferences.

**2. Tracking and Logging Tables**

Two special tables are used to keep track of archive activity:

* **Archive Base Tracking Table**: This table logs metadata about each created archive base. It stores the archive key (e.g., 2023\_Q1 for Q1 of 2023), the base ID, and other details.
* **Archive Process Log Table**: This table logs each archiving run, including the number of records processed and archived, errors encountered, and the time the operation took.

If these tables don’t exist in Airtable, the script creates them dynamically.

**3. Schema Extraction**

The script dynamically extracts the schema of the master table to recreate it in the new archive bases. This ensures that archived bases will have the same fields and structure as the master table, making it easier to query and maintain consistency.

**4. Record Processing and Filtering**

The script fetches all records from the master table and filters out the records that have already been archived. It uses the Archived field in the table to track which records have already been processed.

**5. Archive Base Creation and Management**

Depending on the archive frequency (e.g., monthly, quarterly), the script groups records by period (e.g., January 2023) and creates new bases for each period. The function createBaseIfNotExist ensures that a new base is created only if it doesn’t already exist in the Archive Tracking Table.

Once the base is created, the script logs its information (e.g., Base ID) in the Archive Tracking Table.

**6. Adding Records to Archive Bases**

The main logic of the script involves taking records from the master table and copying them into the corresponding archive base based on the archive key (e.g., 2023\_Q1). The addRecordToBase function handles this by making an API request to Airtable to create new records in the target base.

For better performance, this step is often done in **batches** of 10 records at a time, which significantly improves the speed when processing large datasets.

**7. Updating the Master Table and Marking as Archived**

After records are successfully added to the archive base, the script marks them as archived in the master table by setting the Archived field to true. This ensures that these records won’t be reprocessed in future archiving runs.

**8. Logging the Archiving Process**

After all records have been archived, the script logs important details about the process (e.g., number of records archived, execution time, and any errors) in the Archive Process Log table.

**9. Handling Large Data Sets and Batching**

Since Airtable limits API requests to 100 records at a time, the script is designed to handle large datasets by splitting them into smaller chunks. The MAX\_RECORDS\_PER\_BASE constant helps manage this, ensuring the system doesn’t hit Airtable’s API or storage limits.

**Summary of Archive Script Functionality**

The archive script automates the process of moving older records from a master Airtable table to separate archived bases. It is designed to:

* Dynamically create new archive bases as needed.
* Move records based on time periods (e.g., monthly or quarterly).
* Log the archiving process for future reference.
* Batch operations to improve performance with large datasets.
* Ensure that archived records are flagged in the master table to avoid duplication in future runs.

By efficiently managing data archival, this script helps keep the master table manageable while preserving historical records in archived bases.

Extension Overview

**Merging Multiple Archive Bases Together**

One of the key features of this tool is its ability to merge data from **multiple archive bases** into a single, unified view. The tool accesses these archives through an "Archive Base Tracking" table in Airtable and merges records from multiple bases so that users can view and analyze them as a single dataset. This merging is an essential function that allows users to work with fragmented data sources across different periods or categories.

Here’s how the merging of multiple archive bases works:

**1. Identification of Archive Bases:**

* The **Archive Base Tracking** table in Airtable stores metadata about different archived bases. These are essentially smaller subsets of the main dataset that have been archived based on some criteria like time periods (e.g., by year or quarter).
* Each archive base has an **Archive ID** and a **Primary Table ID**, which are key to fetching records from the Airtable API. The tool first retrieves all these identifiers and prepares to merge their contents.

**2. Fetching Records from Multiple Bases:**

* For each archive base, the tool fetches records via the Airtable API using the fetchRecordsFromBase function. This function loops through the tables within each base and retrieves records batch by batch (with a limit of 100 records per API call).
* The data from each base is added to an array called combinedRecords. Each fetched record includes both the base ID and table ID, which helps to maintain the provenance of the data, i.e., which base or table the record came from.

**3. Merging Records:**

* As records are fetched from each base, they are combined into a single dataset (combinedRecords). These records might come from different bases and tables, but once they are fetched, they are treated as a single dataset in memory.
* The code maintains a maximum record limit (MAX\_RECORDS\_PER\_BASE), which ensures that the combined data doesn’t exceed the allowed size for the tool to handle at once. This is crucial to avoid memory issues, especially when merging data from multiple large bases.

**4. Unified Data Display:**

* After all records have been fetched from the archive bases, they are combined and stored in the dataItems state. This unified dataset is then displayed in the user interface through filters, pagination, and charts.
* For users, it feels as though all records are coming from a single source, even though they are actually from multiple archive bases. This approach allows for seamless exploration of data, regardless of its original source.

**5. IndexedDB Caching for Merged Data:**

* Once the data is fetched from multiple archive bases, it is cached in **IndexedDB**. This means that the merged dataset is stored locally, reducing the need for frequent API calls and improving performance. On subsequent loads, the tool first checks if the merged dataset is available in IndexedDB and only makes new API requests if the cache is expired.

**6. User Interaction with Merged Data:**

* Users can interact with this merged data using filters, pagination, and charting. Since all records from different archive bases are combined into one dataset, the user doesn't have to manually switch between different bases or tables to view the full picture. They can apply filters (e.g., by year, status, publication) across the entire merged dataset, rather than filtering per base.

**Summary of Merging Multiple Archive Bases**

The tool automatically merges data from multiple archived bases (tracked in the "Archive Base Tracking" table). It fetches records from each base, merges them into a single dataset, and displays them in a unified view. This approach abstracts away the complexities of working with multiple sources, allowing users to interact with their data as if it came from a single place. This data is also cached in IndexedDB to improve performance and reduce the load on the Airtable API.

**Detailed Explanation of the MultiBase Data Explorer Extension**

The **MultiBase Data Explorer** is an Airtable extension that allows users to pull data from multiple archive bases, filter and search records, and visualize the data using a chart component. This extension addresses the need to manage and analyze data across multiple archived tables that share a common schema. Here’s a step-by-step explanation of how it works:

**1. Configuration Constants**

At the start of the script, we define some constants:

* **MAX\_RECORDS\_PER\_BASE**: Limits the number of records fetched from each base to 5,000 to prevent overwhelming Airtable’s API and ensure that data can be processed efficiently.
* **ITEMS\_PER\_PAGE**: Specifies how many records should be displayed per page in the table (10 in this case).
* **CACHE\_TIMEOUT\_MS**: Defines how long the data is cached (1 hour in milliseconds) before the cache is invalidated and refreshed.

These constants provide flexibility for controlling the behavior of the extension in terms of pagination and data caching.

**2. IndexedDB Utilities**

IndexedDB is used to store records locally on the client-side for caching purposes. This is particularly useful when pulling data from multiple bases, as it reduces the need to repeatedly make API calls, thus improving performance and reducing load times.

The following functions handle the interaction with IndexedDB:

* **openDB()**: Opens the IndexedDB database or creates it if it doesn’t exist.
* **addDataToDB()**: Adds records to the IndexedDB store.
* **getDataFromDB()**: Retrieves all records from IndexedDB.
* **clearDataFromDB()**: Clears the existing data from IndexedDB.

These utilities ensure that data can be cached locally and efficiently retrieved.

**3. State Variables and Airtable Base Setup**

In React, several state variables are defined to manage the UI and data fetching:

* **isModalVisible**: Tracks the visibility of the modal.
* **dataItems**: Stores the fetched records.
* **searchQuery, yearSelection, publicationSelection, statusSelection**: These track the user’s input for filtering the records.
* **isDataLoading, loadProgress**: Track the loading state and progress when fetching data.
* **currentPage**: Manages pagination in the table view.

The useBase and useRecords hooks are used to interact with Airtable. The script pulls data from the "Archive Base Tracking" table, which stores metadata about each archived base, such as its ID and table structure.

**4. Fetching Data from Airtable**

The fetchRecordsFromBase() function fetches records from a given Airtable base and table by making API calls. It uses pagination to retrieve data in batches (100 records per request) and continues fetching until all records are retrieved or the maximum limit of 5,000 records per base is reached.

**5. Data Caching and Loading**

The loadAllRecords() function is the core data-fetching mechanism. It checks if the data is already cached in IndexedDB. If the cache is still valid (within the 1-hour window), it loads data from the cache to avoid unnecessary API calls. If the cache is expired or the user forces a refresh, it fetches new data from the Airtable bases.

The function:

1. **Checks for cached data** in IndexedDB.
2. **Loads data from IndexedDB** if the cache is valid.
3. **Fetches data from Airtable** if no cache exists or if the cache is outdated.
4. **Stores fetched data in IndexedDB** for future use.

**6. Filtering and Pagination**

Once the data is loaded, the user can apply various filters (by year, publication, status, and date range) using the following functions:

* **toggleFilter()**: Allows the user to toggle filters on or off.
* **filteredData**: A useMemo hook is used to filter records based on the user’s selections (search query, year, publication, status, date range).
* **paginatedData**: This uses pagination to display records in manageable chunks.

**7. Data Visualization with ChartComponent**

The filtered data is visualized using the ChartComponent, which groups and counts records based on user selections (x-axis grouping, stacking, and bucketing by date). The component provides a graphical representation of the data, making it easier to analyze trends.

**8. Table View and Pagination**

The records are displayed in a table with sortable columns. The useTable hook from react-table is used to create the table view, allowing users to interact with the data. Pagination controls are also provided to navigate through the filtered records.

**Purpose of the Extension Functionality**

**The MultiBase Data Explorer** allows users to:

1. **Pull data from multiple archive bases**: The extension can pull data from different Airtable bases that follow the same schema. This is useful for exploring historical data stored in multiple archive tables.
2. **Apply filters and search through records**: Users can search and filter the data by various fields such as year, publication, status, and date range.
3. **Visualize the data with charts**: A chart component allows users to group, stack, and bucket data, providing a quick and intuitive way to understand the dataset.
4. **Cache data in IndexedDB**: To improve performance and reduce the number of API calls, the extension caches data locally using IndexedDB. This cache is refreshed periodically (every hour) to ensure that users are working with up-to-date data.