Relation table Guid:

List of APIs for field search.

In dividual Search API:

If user select any option from system fields or archival type fields we are adding result data to table.

In this table we are displaying data in rows for each row we have search and delete button.

In above screen shot if you observed there are total 5 columns in each row.

* First column represents a ArchivalTypes nothing but a parent column.
* Second column represents a Selected field value.
* Third column represents a operator value.
* Fourth column represents values selected or entered by the user.
* Fifth column represents action buttons like search and Delete.

**Parent Column (1st Column-ArchivalTupe):**

There are two cases on this parent column data fill.

Case-1:

* If user selects any option from the direct SystemFields then we need to provide a select option to user in dropdown form with multiselect option.
* Which means there may be a chance of getting data in array format why because user can select more than one values.
* So we are representing this with **parentSelected:[]** key. If this is field with options then it is selected by system fields.

Case-2:

* If user selected any option form Archival Type selected systemfields or any accordion options then we need to add directly to the parent column.
* Which means there is only on object. It should be in id and label format.
* So we are representing this as **parent: {id: string; label: string; };**

**Field Column (2nd Column-Selected Field):**

This is direct field we are going to display the selected field value in this column. We represent it as field: {id: string; label: string; };

**Operator Column (3rd Column- relation Operations)**

In this we are going to display operation options user should select one option among them. So it represents as operator: { id: string; label: string; };

**Value Column (4th Column- Values)**

Based on the relation between field and operator selections in values column we are going to display controls. Here also we have two cases.

Case-1:

Generally, we are going to display single value in in this value column. If it is single we are displaying directly as value=’ ’.

Case-2

As I said based on relation we are displaying controls in value column. If we select multiple control option in operator field. We will display two controls. So we should save two values. In this scenario we are separating the values with ‘-‘. In case of dual values it should be value:’Test1’ – ‘Test2’.

Actions Column (5th Column- Search and Delete)

To perform search functionality, we need API call.

After adding data if user clicks on search button which is present on row. We are going to send a all the selected fields in array form. The format looks as “xyz.ts” file.

From Frontend we are going to pass the xyz.ts type of data as FormData.

**Row Based Search Functionality API**

This feature enables users to construct search conditions dynamically using a row-based table structure. Each row represents a single search condition, which is composed of the following five components:

1. Parent Column (Archival Type)
2. Field Column (Selected Field)
3. Operator Column (Relation Operations)
4. Value Column (Input/Selection)
5. Action Column (Search/Delete)

The search is executed per row and allows for both system field-based and archival type-based selections.

**1. Parent Column (Archival Type)**

This column represents the **source field group** (either System Fields or Archival Types). There are two primary cases depending on how the field is selected.

**Case 1: Selection from System Fields**

* If the user selects a field from the **System Fields**, a **multi-select dropdown** is provided.
* This implies **multiple values** can be selected for a single condition.
* These selections are stored in an **array format**.

parentSelected: [

{ id: "1", label: "Windows-EN" },

{ id: "2", label: "Mac-EN" }

]

isParentArray: true

Example UI: A dropdown where the user selects multiple operating systems.

**Case 2: Selection from Archival Type or Nested Accordion**

* If the field comes from a selected **Archival Type field** or **accordion** (nested group), then the value is a **single object**.
* This object contains id and label.

parent: { id: "100", label: "Employee Records" }

isParentArray: false

Example UI: User selects a nested field under "Employee Records" → "Personal Info".

**2. Field Column (Selected Field)**

* This column displays the field selected by the user.
* It is always a **single object**.

field: { id: "Age", label: "Age" }

Example: If user selects “Age” field, this object holds its metadata.

**3. Operator Column (Relation Operations)**

* The operator defines the logical relationship to apply for the field value (e.g., Equals, Between, Empty).
* It is also represented as a **single object**.

operator: { id: "between", label: "Between" }

Common Operators: equals, contains, empty, between, lessThan, greaterThan, etc.

**4. Value Column (Input/Selection)**

This column holds the value(s) input by the user. There are two cases:

**Case 1: Single Value**

* Used for simple operators like equals, contains, empty, etc.

value: "25"

Example: For field “Age” and operator “equals”, the user enters "25".

**Case 2: Dual Values**

* Used for operators like between.
* Two controls are shown, and values are stored in a **hyphen-separated string**.

value: "20 - 30"

Example: “Age” field with “between” operator, user enters "20" and "30".

**5. Action Column (Search/Delete)**

* Contains two buttons:
  + **Search**: Triggers API call using the entire row data.
  + **Delete**: Removes the current row.

**Data Format (Frontend to Backend)**

Each row in the table is serialized into the following format and sent to the API as FormData:

**Sample JSON (xyz.ts):**

{

rowid: "781QGWJ",

parent: { id: "", label: "" },

parentSelected: [

{ id: "1", label: "Windows-EN" }

],

field: { id: "Age", label: "Age" },

operator: { id: "empty", label: "empty" },

value: null,

isParentArray: true,

parentTouched: true,

operatorTouched: true,

valueTouched: true

}

**API Integration Notes**

* On clicking **Search**, each row’s object is sent to the backend via API.
* API expects **FormData** structure with key-value pairs from the row.
* Each row is treated as a separate request unless batch search is implemented.

**Validation & UI Feedback**

* **parentTouched**, **operatorTouched**, **valueTouched**: Useful flags to handle user interactions and validations.
* Empty fields should not allow submission.
* Value control rendering should depend on operator selected.