Dynamic Queries Interface

This document explains use of dynamic query interface to define and use dynamic queries.
Table of Contents

About Dynamic Queries Interface ............................................. 3
1. Adding Dynamic Query for Different Purposes ............................................. 4
   1.1 Dynamic Field Purpose .................................................. 4
   1.2 DatabaseMenu Purpose ............................................... 9
   1.3 NumericRangeList Purpose ........................................... 13
   1.4 DatabaseRangeList Purpose .......................................... 19
   1.5 DynamicMenu Purpose ............................................... 26
   1.6 Lookup Purpose .................................................... 32
2. Renaming Dynamic Query .................................................. 38
3. Removing Dynamic Query .................................................. 39
4. Restoring Dynamic Query .................................................. 42
5. Understanding Test Query .................................................. 43
About Dynamic Queries Interface

The dynamic queries are named *parameterized* queries and are accessed in many application objects such as in rules, variables, UDFs, CBRs, etc. by their IDs. Some of their parameters can be set at run time.

Data can be retrieved at run-time from database based on some criteria etc. without writing explicit code. Dynamic query object holds database query (named query) which is used to retrieve/populate the data at run time. These objects save lot of explicit coding inside Rulebase as well as in other interfaces such as retrieving contents dynamically from databases. Also, queries can be written as general-purpose queries applicable to many applications. The system holds all dynamic query objects as an application collection object. Parameters can be passed through sessions or through URLs.

Queries can also be predefined to bring datasets for lists, dynamic lists, collaborative filtering, content filtering, clustering etc.

The following figure shows, the default dynamic query interface.

![Figure 1: Dynamic Query Interface](image)

### Table 1: Fields of Dynamic Query Interface

<table>
<thead>
<tr>
<th>Fields</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dyn Query ID</td>
<td>Shows the ID of selected dynamic query.</td>
</tr>
<tr>
<td>New ID</td>
<td>New dynamic query can be added with given ID or rename the existing one.</td>
</tr>
<tr>
<td>Purpose</td>
<td>It specifies the different purposes for which dynamic queries can be created.</td>
</tr>
<tr>
<td>Is reload database menu when update/refresh?</td>
<td>It completely reloads database menu (even if database menu created using this DQ is set as <em>Is use as static menu?</em>)</td>
</tr>
<tr>
<td>Selected Database</td>
<td>It specifies the database ID from which dynamic query will fetch results at run time.</td>
</tr>
<tr>
<td>Field Name/s</td>
<td>It contains the field names to be selected from the table. The list, names and order of fields must follow the syntax based on purpose. Otherwise, results may not be correct.</td>
</tr>
<tr>
<td>-------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Skip distinct clause?</td>
<td>It allows adding duplicate entries in the result.</td>
</tr>
<tr>
<td>View Name</td>
<td>Name of table/view of query.</td>
</tr>
<tr>
<td>Filter</td>
<td>Where condition in the where clause of SQL. Filter can include variable names whose values would be populated at run time. Variables must be embedded inside square brackets.</td>
</tr>
<tr>
<td>Is ignore filter during query test?</td>
<td>It ignores the filter or where clause at the time of query execution.</td>
</tr>
<tr>
<td>Variable</td>
<td>It contains the list of variables of selected variable group.</td>
</tr>
<tr>
<td>Is use code of this variable?</td>
<td>It is applicable, when variable value is to be populated in filter, code is populated instead of value.</td>
</tr>
<tr>
<td>Insert Fields</td>
<td>When dynamic query is to be used to insert/update data into DB at run-time. Example purpose: SimpleDBSimilarities, additional fields with data to be inserted/updated can be specified here. It is given in FieldName=Value pair separated by comma (,).</td>
</tr>
<tr>
<td>Mandatory Variables</td>
<td>Since WKAS allows parameterized queries where parameter values are populated at run-time through session variables or through filters (URLs). Parameters passed through URLs are subject to SQL attacks, mandatory variables safeguard retrieval of data from database. Such parameters (variables) must be provided values in session to retrieve any data from DB.</td>
</tr>
<tr>
<td>Group By Field/s</td>
<td>It contains list of comma-separated columns or expressions to be used as group by clause.</td>
</tr>
<tr>
<td>Having Clause</td>
<td>Filter conditions for group of rows or aggregates. This clause is useful when Group by clause is used. For example, query that returns records of employees of Computer Science department whose salary is greater than 10K.</td>
</tr>
<tr>
<td>Sort By Field/s</td>
<td>Name of the field by which the values would be retrieved from database.</td>
</tr>
<tr>
<td>Description</td>
<td>It specifies the description of dynamic query.</td>
</tr>
</tbody>
</table>

1. Adding Dynamic Query for Different Purposes
Dynamic queries save lot of explicit coding inside rule-base. They facilitate retrieval of data at run-time by just using their IDs and filters.

Dynamic query can be added for various purposes like DynamicField, DatabaseMenu, DynamicMenu, NumericRangeList, Lookup, Clustering, and SimpleDBSimilarities etc.

1.1 Dynamic Field Purpose
Dynamic query can be created for DynamicField purpose when a single value is to be retrieved at run-time and assigned to a variable. It uses the filter to filter the results from database. Variable values in the filter are populated dynamically at run-time by the inference engine. System retrieves only first row value (or first values of all rows as LIST or TREN) even SQL query returns multiple rows.
For example, create dynamic query for getting *CombineBalance* from bank *customers* account for credit/debit transaction.

1. To add query, first select variable group (*Account*) from the list and specify the name in *New ID* field. Select *DynamicField* purpose from *Purpose* dropdown menu. To add new query with purpose, click *Add New*.

![Figure 2: Adding Dynamic Query](image)

2. After clicking *Add New*, a popup is shown, click *OK* to add query.

![Figure 3: Adding Dynamic Query](image)

3. After clicking *OK*, dynamic query (*DQBalance*) is added into selected variable group.
4. Select database ID (WKADemo_BankApps) from Selected Database dropdown menu from which data is fetched. To select CombineBalance field from Account table, click Build Query and the page is redirected to Query Builder interface.

5. After clicking Build Query, query builder interface is opened. Select the required field (CombinedBalance) from the table (account) which needs to be retrieved and click Insert Field to insert the field into Direct queries/Field list. Click OK.
6. After clicking OK, the page is redirected to Dynamic Queries interface. Selected field name (CombinedBalance) gets added in Field Name/s field and table name gets added in View Name field. Even criteria can also be added using Query Builder.

![Figure 7: Selected Field & Table is Added](image)

7. To add filter, select the Variable Group (Transaction) and Variable (AccountID) from Variable dropdown menu and click the button to move the variable and variable group in Filter field.

![Figure 8: Adding Filter](image)

8. To update query, click Update/Refresh.
9. Dynamic query (DQBalance) with DynamicField purpose must be linked to variable (Balance). To link query, create a variable Balance of Type: Input, Data Type: Number (To create a variable refer Global Variables Interface document).

Go to Main Menu -> Domain Vocabulary -> Global Variables and click Input Options and When Added from the list of options and select dynamic query (Account.DQBalance) from Database Field dropdown menu (query with purpose DynamicField are added in Database Field list). To update variable, click Update.

Figure 9: Updating Dynamic Query

10. To use the variable Balance in debit/credit rules which is already created, go to Rules/Expert Systems interface. The following screen shows the debit and credit rule.

Figure 10: Updating the Variable

10. To use the variable Balance in debit/credit rules which is already created, go to Rules/Expert Systems interface. The following screen shows the debit and credit rule.
1.2 Database Menu Purpose

It creates the menu when the application object is initialized by executing the query and retrieving the data from database. This would be applicable when menu options are to be taken from database rather than explicitly creating.

Once menu is created (data is fetched from DB), it is treated as static menu and options remain unless they are refreshed explicitly. Sequence of fields (other than option all fields are optional) Option, Code, AlternateValues, Inference, Description, ImageFile, ExternalID. The database menu is retrieved only once whenever application starts or explicitly when clicked on Update/Refresh button. The change in database is not reflected into application.
For example, create database menu for employee’s Department.

1. To add query, select variable group (EmpInfo) from the list and specify name (DepartmentMenu) in New ID field. Select DatabaseMenu purpose from Purpose dropdown menu and click Add New.

2. After clicking Add New, a popup is shown, click OK.

3. After clicking OK, dynamic query with purpose DatabaseMenu is added.
4. Select database ID (DemoApp) from Selected Database dropdown menu from which data is fetched. To select Department field from Employee table, click Build Query and the page is redirected to Query Builder interface.

5. Select Department field from Employee table and click Insert Field to insert the field into Direct queries/Field list. Click OK to go back to Query Interface.

6. Selected field (Department) and table name (employee) is added in Field Name and View Name respectively. To update query, click Update/Refresh. After updating the query, a message OK:6 is shown which indicates that total six departments are added in menu.
7. To check the menu is added, go to Menus & Lists interface and select variable group (EmpInfo) in which menu (DepartmentMenu) is created.

8. To view values added in menu, go back to Dynamic Queries interface. Select DepartmentMenu query from the list and click Test Query which executes the query report with given parameters.
9. To add duplicate queries in result set, check the option *Skip distinct clause?* To execute query, click *Test Query* and the result set has duplicate entries for *Department* field as shown in screen shot below.

**Figure 21: Executing Query**

### 1.3 NumericRangeList Purpose

The dynamic query with purpose *NumericRangeList*, converts numeric values into qualitative ones or numeric itself.

The fields *FromValue, ToValue, Value* are the mandatory fields for dynamic query with *NumericRangeList* purpose.

For example, convert the student marks into grade.

1. Create a table (*MarksRange*) with *FromValue, ToValue and Value* fields as shown in screen shot below *(Refer DB Schema Manager Interface Document)*.
2. Enter data into table using *Data Entry* interface ([Refer Data Entry Interface Document](#)). The field values should be in ascending order.

![Figure 22: Creating Table](image)

3. To add query, select variable group (*Student*) and specify query name in *New ID* field. Select *NumericRangeList* purpose from *Purpose* dropdown menu and click *Add New*.

![Figure 23: MarksRange Table](image)
4. After clicking Add New, a popup is displayed, click OK.

5. After clicking OK, dynamic query (MarksRange) with purpose NumericRangeList is added.

6. Select database ID (DemoApp) from Selected Database dropdown menu from which data is fetched. To select FromValue, ToValue and Value fields from MarksRange table, click Build Query and the page is redirected to Query Builder interface.
7. Select *FromValue*, *ToValue* and *Value* fields from *MarksRange* table and click *Insert Selected Fields* to insert the fields into *Direct Queries/Field List*. Click *OK* to go back to the *Query Interface*.

8. Selected fields and table name is added in *Field Name* and *View Name* respectively. To update query, click *Update/ Refresh*.
9. To create numeric range, go to Main Menu -> Domain Vocabulary -> Scores & Ranges (Refer Scores & Ranges Interface Document). Select the range (MarksRangeList) from the list to import the range of values from dynamic query, select dynamic query (Student.MarksRange) from From Dynamic Query dropdown menu. To update the dynamic query, click Update/Refresh

![Image of the Dynamic Query interface showing how to select and update dynamic queries.]

Figure 30: Creating RangeList

10. After creating range list, link the range list created to Marks field of Student table. When the data is retrieved in field, it uses this range and transforms the field value into other value (For Example: If marks entered (30) in Student table are within the range then grade (F) corresponding to the marks is shown, refer figure 23). To link range list to the value which is to be changed, select Marks variable from the list and select the range from Transform Range ID dropdown menu. To update variable DB field linking, click Update.

![Image of the Variable DB Mapping interface showing how to link range list to the field.]

Figure 31: Linking Range List to the Field
11. To view field transformation, run the application. To run the application, go to Apps interface and click Student Application. Select View DB operation from dropdown menu and click Next.

Figure 32: Selecting DB Operation

12. Enter Student ID and click Next.

Figure 33: Entering Student ID

13. After clicking Next, form corresponding to Student ID is displayed. The value of Marks field is transformed into another value as explained above in step 10.

Figure 34: Field value (marks) transformed into other value (grade)
1.4 DatabaseRangeList Purpose
Dynamic query with purpose DatabaseRangeList converts the variable values into field values when data is updated to database.

The fields FromValue, ToValue are the mandatory fields for dynamic query with DatabaseRangeList purpose.

For example, select gender as Female/ Male but in database, gender field is stored as F or M.

1. To add dynamic query, first create a table (DatabaseRange) with FromValue, ToValue fields (Refer DB Schema Manager Interface Document).

![Figure 35: Creating Table](image)

2. Enter data into table using Data Entry interface (Refer Data Entry Interface Document). The field values should be in ascending order. The following screen shot shows the data entered in table.

![Figure 36: DatabaseRange Table](image)
3. To add query, first select variable group (Student) and specify query name (DBRange) in New ID field. Select DatabaseRangeList purpose from Purpose dropdown menu and click Add New.

![Figure 37: Adding Dynamic Query with DatabaseRangeList Purpose](image)

4. After clicking Add New, a popup is displayed, click OK.

![Figure 38: Adding Dynamic Query](image)

5. After clicking OK, dynamic query with purpose DatabaseRangeList is added.

![Figure 39: Adding Dynamic Query](image)

6. Select database ID (DemoApp) from Selected Database dropdown menu from which data is fetched. To select FromValue, ToValue fields from DatabaseRange table, click Build Query, and the page is redirected to Query Builder interface.
7. Select *FromValue, ToValue* fields from *databaserange* table and click **Insert Selected Fields** to insert the fields into **Direct Queries/Field list**. Click **OK** to go back to **Query Interface**.

8. Selected fields and table name is added in **Field Name** and **View Name** field respectively. To update the query, click **Update/Refresh**.
9. To create database range, go to Main Menu -> Domain Vocabulary -> Scores & Ranges (Refer Scores & Ranges Interface Document). Select the range (DBRangeList) from the list. To import the range of values from dynamic query, select dynamic query (Student.DBRange) from From Dynamic Query dropdown menu. To update the dynamic query, click Update/Refresh.

10. After creating range list, link the range list created to Gender field of Student table. When the data is retrieved in field, it uses this range and transforms the field value into other value (For Example: If gender entered (Female) in Student table then in database the gender is stored as F, refer figure 36). To link range list to the value which is to be changed, select Gender variable from the list and select the range from Transform DB Range ID dropdown menu. To update variable DB field linking, click Update.
11. To view the changes, run the application. To run the application, go to Apps interface and click Student Application. To add the record, select Add operation from Select DB operation dropdown menu and click Next.

![Figure 44: Linking Range List to the Field](image)

12. Enter Student ID which is to be added and click Next.

![Figure 45: Selecting DB Operation](image)

13. Enter the required student details in form and click Next.

![Figure 46: Entering Student ID](image)

![Figure 47: Entering Student Details](image)
14. The following message (*DB Operation Result:OK:1*) screen shows that one record is inserted into database.

![Figure 48: DB Operation Status](image)

15. To view newly added record, go to *Query Builder* interface, select the required table (*Student*) and click *Execute Query* (Refer *Query Builder interface document*).

![Figure 49: Viewing Student Table](image)

16. Similarly, to add another record with *Student ID* (112) follow the same steps as explained above. Select *Add* option from *DB Operation* dropdown menu and click *Next*.
17. To view newly added record, go to *Query Builder* interface, select the required table (*Student*) and click *Execute Query* (Refer *Query Builder Interface Document*).
1.5 DynamicMenu Purpose

If values are changing in database frequently, then it is advised to use *Dynamic Menu* instead of *Database Menu*. It is applicable when menu options are to be fetched/determined at run-time from database.

The filter (condition) can be used to filter the results (field values) from the database. In case of options from existing menu, the code value is picked up from the filter. Variable values in the filter are populated dynamically at run-time by the inference engine. Sequence of fields (other than option all fields are optional): *option, code, image file/URL, description, inference*.

Let us create dynamic menu for *Name* field of *Student* table.

1. To add dynamic query, select variable group (*Student*) from the list and specify the query name (*StudNameMenu*) in *New ID* field. Select *DynamicMenu* purpose from *Purpose* dropdown menu and click *Add New*.
2. After clicking *Add New*, a popup is displayed, click *OK*.

![Figure 56: Adding Dynamic Query](image)

3. After clicking *OK*, dynamic query (*StudNameMenu*) with purpose *DynamicMenu* is added.

![Figure 57: Dynamic Query Added](image)

4. Select database ID (*DemoApp*) from *Selected Database* dropdown menu from which data is fetched. To select *Name* field from *Student* table, click *Build Query*, and the page is redirected to *Query Builder* interface.
5. Select *Name* field from *student* table and click *Insert Selected Fields* to insert the fields into *Direct Queries/Field list*. Click *OK* to go back to *Query Interface*.

Figure 58: Selecting Database

![Selecting Database](image)

6. Selected field and table name is added in *Field Name* and *View Name* field respectively. Add *StudentID* as filter variable which is used as *Menu Code* and *Name* field is used as *Menu Option*. To add filter, select variable group (*Student*) and variable (*StudentID*) from Variable dropdown menu. To move the variable group and variable in *Filter* field, click the button. To update the query, click *Update/Refresh*.

Figure 59: Selecting Field

![Selecting Field](image)

7. To link dynamic query with the variable (*Name*), go to *Global Variables* interface and click *Input Options and When Added*. Select *Name* variable from variable group.

Figure 60: Adding Filter and Updating Dynamic Query

![Adding Filter and Updating Dynamic Query](image)
and select dynamic menu from *Dynamic Menu* dropdown menu which is to be linked. To update the variable details, click *Update*.

**Figure 61: Linking Dynamic Menu**

7. If the variable (*Name*) is not asked in rules, then that variable first needs to be asked in rules in *Rules/Expert Systems* interface as shown in screen shot below. To update the rule created, click *Update* and the message with update screen is displayed.

**Figure 62: Asking Variable in Rule**

8. To view the changes, run the application. To run the application, go to *Apps* interface and click *Student Application*. Select *Add* option from *DB Operation* dropdown menu.
9. Enter the desired StudentID and click Next.

10. Fill the required information in form displayed and click Next.

11. After clicking Next, the following screen shows that one record is added and click Back button.
12. After clicking *Back* button, to view the details added, select *View* operation from *DB Operation* dropdown menu.

9. Enter *StudentID* and click *Next*.

10. After clicking *Next*, the option/value (*Name*) corresponding to *StudentID* is displayed.
1.6 Lookup Purpose

Dynamic query is created with Lookup purpose when view is used to load data in lookup table. Let is create lookup table for Student table.

1. To create dynamic query, first select variable group (Student) from the list and specify dynamic query name (LTStudentData) in New ID field. Select Lookup purpose from Purpose dropdown menu and click Add New.

2. After clicking Add New, a popup is displayed, click OK.

3. After clicking OK, dynamic query with Lookup purpose is added in the list.
4. Select database ID (DemoApp) from Selected Database dropdown menu from which data is fetched. To select the fields (StudentID, Name, Age etc.) from Student table, click Build Query, and the page is redirected to Query Builder interface.

5. Select the required fields from student table and click Insert Selected Fields to insert the fields into Direct Queries/Field list. Click OK to go back to Query Interface.
6. Selected field and table name is added in Field Name and View Name field respectively. To update the query, click Update/Refresh.

7. To retrieve the data corresponding to key variable (StudentID), go to Variable DB Mapping interface, set Field Content Type for key variable StudentID to Lookup. To update variable DB field linking, click Update.
8. To fetch the data from lookup table using dynamic query, first create a lookup table (LTStudent) from Lookup Table interface (Refer Lookup Table Interface Document). Select ID (DemoApp) from Database ID dropdown menu, select dynamic query from Dynamic Query dropdown menu from where the data is to be fetched and key field name (StudentID) is populated in Key Field Name dropdown menu. To add data in lookup table, select Key Variable name (StudentID). To create lookup table (key and columns) from field names specified in dynamic query (Fields must be mapped to variables before this operation) click Create entries using dynamic qry.

Note: Lookup table can be loaded using dynamic query and all fields in the query should be mapped to column variables. Query must retrieve all field values
including the keys for each row.

9. After clicking *Create entries using dynamic qry*, a popup is displayed, click *OK*.

![Figure 78: Adding Data in Lookup Table](image)

10. After clicking *OK*, entries are added in lookup table.

![Figure 79: Data Added in Lookup Table](image)

11. To load the lookup table at the start of WKA studio, check *Is load table at start?* and click *Update*. 
12. To load values in lookup table, click **Load Values**.

13. To view records in lookup table, click **Show Values** and corresponding values loaded in selected lookup table is displayed.
2. Renaming Dynamic Query

This interface allows renaming the dynamic query. For example, rename \textit{StudNameMenu} query as \textit{NameMenu}.

1. To rename query, go to \textit{Dynamic Queries} interface and select the query (\textit{StudNameMenu}) from \textit{Variable Groups (Student)} which is to be renamed and specify new query name (\textit{NameMenu}) in \textit{New ID} field.

2. To rename the query, click \textit{Rename} and selected query is renamed with given \textit{New ID} (\textit{NameMenu}).
3. Removing Dynamic Query

This interface allows removing dynamic query from variable group. For example, remove `LTStudentData` query.

1. To remove the required query, select `LTStudentData` query from `Variable Groups (Student)` and click `Remove`.

2. After clicking `Remove`, a popup is displayed, click `OK`. 

Figure 84: Renaming Dynamic Query

Figure 85: Removing Dynamic Query
3. After clicking OK, selected query is removed from its variable group.

4. In interface below, if for example: $DQStudentID$ query needs to be removed then removing dynamic query is not allowed since the query is linked to the variable $StudentIDFromDB$ or used in rule. After clicking Remove, it gives the following error as shown in screen shot below.
5. To remove such queries, first remove the link with the variable from *Global Variables* interface and click *Input Options and When Added*. Select variable *StudentIDFromDB*, set *Database Field* to *Not Set* and click *Update*.

![Figure 89: Removing Query Linked With Variable](image1)

6. After removing the link with the variable, go back to *Dynamic Queries* interface to remove dynamic query, select the query and click *Remove*.

![Figure 90: Removing Dynamic Query](image2)

7. After clicking *Remove*, dynamic query is removed and the following message is displayed.
4. Restoring Dynamic Query

This interface allows restoring the dynamic query removed from variable group. For example, restore LTStudentData and DQStudentID queries.

1. To restore LTStudentData query, specify query name in New ID field which should be same as removed query name and click Restore.

2. After clicking Restore, a popup is displayed, click OK.

3. After clicking OK, dynamic query (LTStudentData) is restored in its variable group.
4. Similarly, to restore DQStudentID query, follow the same steps as explained above for removing LTStudentData query.

5. Understanding Test Query

Test query allows executing the query with given parameters. For example: To execute a query for female employee names and their departments:

1. Write a query in Field Name/s, View Name, Filter field to get the female employee names and their departments and click Test Query which executes the given query with given parameters.
2. After clicking Test Query, only female employee’s details are retrieved as shown in screen shot below.

3. To retrieve all records from table and execute a simple select statement, check the option Is ignore filter during query test? which ignores where clause or filter part of query. To execute the query, click Test Query.
4. After clicking *Test query*, it ignores the filter query and retrieves all records from table as shown in screen shot below.

![Query Executed](image)

**Figure 99: Query Executed**

**New features added to dynamic query interface:**

1. **Insert Field:**

The *Insert Field* option is used when dynamic query is to be used to insert/update data into database directly from forms using dynamic database menus.

For example, in patient registration form, there is menu *LocalAddress* which contains the list of hotel names nearby clinic. Suppose patient stays at any other hotel which is not included in the list then this option allows to add this new hotel in database.

   1. Create dynamic query with purpose *dynamic menu* and add field *HotelName* in *Insert Field* option.

![Adding Dynamic Query](image)

**Figure 100: Adding Dynamic Query**
2. Link this dynamic query to the variable through *Global variable* interface. To update, click *Update*.

![Variable Linking](image1.png)

**Figure 101: Linking to variable**

3. To recreate form, click *Create* button.

![Form Recreating](image2.png)

**Figure 102: Recreating Form**

4. Now run the patient application. Select Add Patient option, following form will be displayed.
5. Add new name which not included in list.

6. Now click, **Add**, to add this new name in database.
2. Mandatory Variables:
This feature is useful for preventing SQL injection attack especially when data is retrieved using filters in forms. The fields names corresponding to these mandatory variables will be appended to the where clause of query. The value of these variables must be in session or memory. Variable used in this option must be mapped to fields. If variable is not mapped to any field, then it will show error as follows.

For e.g. add PatientInfo.Status variable in mandatory variables list. Since the variable is not mapped (read-only mapping) to any of the field, it shows the error while adding variable.

Figure 106: Adding Variable

This feature becomes useful at the time of displaying personal details like profile photos etc.

For example, patient profile report should display the profile photo only when the value of patient id available in session.

1. To add variable in mandatory variable list, select variable from variable list and click.
2. To update query, click Update.

3. Now run the Patient Application and select Patient Profile option.
4. Select patient name from list to view the profile details. Click Next to proceed further.

![Figure 101: Selecting option](image)

Figure 101: Selecting option

5. Since the value of patient id is not available in session, patient profile photo be cannot be viewed.

![Figure 110: Selecting patient name](image)

Figure 110: Selecting patient name

6. Now add PatientID in a dynamic query field list. To update query, click Update.

![Figure 1021: Hides the patient profile photo](image)

Figure 1021: Hides the patient profile photo

7. Now run the Patient Application and select Patient Profile option. Select patient name from list.

![Figure 112: Adding field into field list](image)

Figure 112: Adding field into field list
8. As the patient ID is loaded in session, the patient profile photo becomes displayed.

---

### Figure 113: Selecting patient name

---

### Figure 114: Displays patient profile photo

---

3. **OnInput Event**

When validation scripts are generated for textboxes, `onchange` event used. When this option is used `oninput` event is used instead of `onChange`. The `oninput` event occurs when an element gets user input (key). `oninput` event occurs immediately after the value of an element has changed, while `onchange` occurs when the element loses focus, after the content has been changed.

For e.g. add `oninput` event for variable `PatientInfo.PatientPaidAmount`, which displays alert message while entering text value in control.

1. Go to *Global Variable* interface, select variable from variable list and click option *HTML Web Page Options*. To add `oninput` event click *Is oninput event?* option.
2. To update variable, click **Update**.

3. To recreate form, click **Create**.
4. Now run the *Admin Application* and select *Add Admin Info* option and click *Next* to proceed further.

5. Enter patient id.
6. Enter text value instead of numeric value in Paid Amount (INR) field. It will immediately validate the value and shows the message.

**Figure 120: Executing oninput event**