Chapter 5
Managing the Data

The data stored in your PegaRULES database includes instances of rule, data, work, assignment, and all the other classes present in your Process Commander system. This chapter describes how data is stored, how you map the classes that represent your application’s data model to database tables, and how to ensure that the data can be retrieved by reports.

You use the information in this chapter during the data design and implementation phase of your development cycle. This chapter contains the following sections.

■ Overview
■ PegaRULES Database Schema and Schema Requirements
■ Mapping Classes to Tables
■ Data and Reporting
■ Declarative Indexing
■ Creating Reports
Overview

When Process Commander saves an object to the PegaRULES database, the destination of the object is determined by a definition that maps it and other instances of the class to which it belongs (the Applies To class) to a specific database table. The mapping definitions are called database table objects. The name of the object is the same as that of the class it is mapping. It specifies the name of the database and the name of the table where instances of the class are to be stored.

In most cases, when an object is saved, the entire object is saved to a blob (Binary Large Object) column named pzPVStream. The purpose of the blob is to store data that is structured in forms that do not readily fit into predefined database columns, such as properties that contain multidimensional data – pages, groups, and lists, for example. A subset of the object’s properties is also saved in separate columns in the table.

Database Name Objects

The database servers used by your Process Commander system are represented as database name objects (Data-Admin-DB-Name). A database name object provides connection information that Process Commander uses to communicate with the database the object represents.

Your system will always contain at least one database name object – the one for the PegaRULES database. If your developers create applications that integrate with external databases through SQL connect rules or activity methods that communicate directly with external tables, you create a database name instance that represents the external database as part of the implementation of that integration project.
Database Table Objects

Database table objects (Data-Admin-DB-Table) represent data mappings between Process Commander classes and database tables. The mapping specifies a database name object and points to a specific table in the database (represented by the database name object) where instances of a specific class and its descendants through pattern inheritance are stored.

You can map more than one class to the same database table and frequently you will do just that.

Class Groups

Class group objects (Data-Admin-DB-ClassGroup) represent two or more concrete work classes, instances of which should be stored in the same database table. When implementing the data model of your application, you specify that a concrete parent class that inherits from the Work- base class is a class group. When you save the class, Process Commander generates a class group data object for you. Then you specify that its children work classes – instances of which are called work objects – belong to that class group.

To complete the data mapping of a class group and its classes, you create one table mapping definition for the class group class. With that one table mapping, you have mapped all the children classes that belong to the class group to the same table so they do not need separate table mapping definitions of their own. Keeping related data together in this way improves the efficiency of database queries.

**Note:** Classes that belong to class groups cannot be mapped to separate tables.
Reporting on Data

By default, Process Commander stores most rule and data information for an object in a blob column. Because the blob is compressed, this data storage method is faster and more space-efficient than storing the value of each property in a separate column; this storage method also supports the complex property modes available to you in Process Commander.

However, when you are creating a report – with the Report Wizard, by manually creating a list view or summary view rule, or with a third-party reporting tool – you cannot use as selection criteria the properties whose values are stored only in the blob column. Reports can select records based only on property values of properties that are stored in separate columns, or “exposed.”

A single value property that is represented as a separate column in a database table is said to be exposed. Each of the standard PegaRULES database tables has a set of columns that represent standard properties present in all Process Commander instances – for example, pyLabel and pxCreateOperator. The properties represented by those columns are “exposed” and the values for those properties can be used as selection criteria for reports.

Declarative Indexing

Your DBAs can change the schema of the storage tables in the PegaRULES database by adding columns for any single-value properties you want to include in reports. However, in addition to single-value properties, Process Commander supports the following complex (or aggregate) property modes:

- Value List and Page List, which store arrays of values
- Value Group and Page Group, which store unordered group structures
- Pages that contain other properties, called embedded page properties

Complex and embedded properties cannot be stored in a single table column and therefore cannot be “exposed.” To report on complex properties, you use declarative indexing. Declarative indexing is implemented through additional database tables (index tables) and additional Process Commander classes (index classes).
**External Databases and Tables**

Your Process Commander applications can interact with external database tables through SQL statements stored in SQL connector rules and through the Obj- activity methods (for example, Obj-Open, Obj-Save).

To use either option (Obj- methods or SQL connectors), there must be a database name object that represents the external database. To create SQL connector rules, you can use the Create Connector Rules Accelerator. To create rules that support interactions with an external database table through the Obj- activity methods, you use the External Data Table Rules wizard.

**History Classes**

Instances of history classes identify the time, the date, and the user who updated a work object, rule, or data object. A history instance may also contain attachments, notes made by the user, or an internal copy of the object as it was before the update.

History records are used for audit trails and accountability. When using the rule management feature (rule checkin/checkout), the historical copy is used to undo changes or roll back a rule to its previous state.

All concrete classes that inherit from either the Work- or Data- base classes and is either a class group or is a class that does not belong to a class group has a matching history class to hold its history records. History classes are automatically generated when the source class is saved.

History classes are named as follows: <History-> + <class group or class name>. For example, if you saved a new concrete class named MyCo-HR-EmployeeData that inherits from Data- and is not a member of a class group, a class named History-MyCo-HR-EmployeeData is also created.
PegaRULES Database Schema and Schema Requirements

Before you begin mapping classes to tables, take some time to examine the PegaRULES database schema. In addition to using your vendor provided database management tools, you can see the schema of the PegaRULES database in the following ways:

- Use the PegaRULES database schema reference, available on your installation media. The schema reference is a set of HTML files that presents the state of the PegaRULES database as it is after installation, before any new tables or columns are added.

- Through the Database Schema wizard, a tool accessible from the Process Commander user interface that displays the current schema of any database represented by a Process Commander database name object. You must have the PegaRULES:SysAdmin4 role to obtain access to this tool. To access this tool, choose Tools > Database > Modify Database Schema from the menu.

The naming convention used for the PegaRULES database tables relies on the prefixes described in Figure 5-1.

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Identifies</th>
</tr>
</thead>
<tbody>
<tr>
<td>pr_</td>
<td>Rules-related activities, properties, requestors, and other features that belong to or support the PegaRULES RuleSet</td>
</tr>
<tr>
<td>trpr_</td>
<td>Database triggers</td>
</tr>
<tr>
<td>sppr_</td>
<td>Stored Process Commander procedures</td>
</tr>
<tr>
<td>pc_</td>
<td>Rules-related flows, work objects, and assignments, things that belong to or support the Pega-ProCom RuleSet</td>
</tr>
<tr>
<td>pcv_</td>
<td>Views that speed queries and reporting</td>
</tr>
<tr>
<td>sppc_</td>
<td>Stored Process Commander procedures</td>
</tr>
</tbody>
</table>

Figure 5-1. Prefixes Used for PegaRULES Database Tables and Views
Creating New Tables in the PegaRULES Database

Your development team may determine that you or the DBAs need to add new tables to the PegaRULES database in order to implement your data storage strategy. In addition to the columns your team may decide to add, there is a required set of columns for any new table you add to the PegaRULES database.

The columns required for a new table depend on what you plan to store in it. The easiest way to ensure that you include the required columns is to use an existing PegaRULES tables as a template. Figure 5-2 presents the PegaRULES tables that store objects of specific kinds. Use this table to locate the PegaRULES table you should use as a template for the new table.

<table>
<thead>
<tr>
<th>Object Type</th>
<th>Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>Work</td>
<td>pc_work</td>
</tr>
<tr>
<td>Data</td>
<td>pr_data</td>
</tr>
<tr>
<td>Declarative index</td>
<td>pr_index, but do not include the pzPVStream column</td>
</tr>
<tr>
<td>Work history</td>
<td>pc_history_work</td>
</tr>
<tr>
<td>Data history</td>
<td>pr_history_data</td>
</tr>
</tbody>
</table>

*Figure 5-2. Tables to Use as Templates for New Tables*

The column names and data types of the new tables must exactly match those of the PegaRULES table. To determine the appropriate PegaRULES table to use as the template for a table that is to store an object type not include in Figure 5-2, examine the table mappings in Process Commander. Choose SysAdmin > Database Table and examine the list of mappings.

Note that the standard Process Commander work and assignment reports rely on views. If your DBAs add work tables to the PegaRULES database, remind them to update the pcv4_assignment_summary, pcv4_assignmentwb_summary, and pcv4_work_history to reference the new table.
Mapping Classes to Tables

When you and the developers create the classes that represent the data model of your applications, you need to consider where objects of those types should be stored.

Start by examining the existing mappings. From the Rules by Type explorer tree, choose SysAdmin > Database Table. A list of the existing table mappings appears. For example:

- Workbasket assignments are stored in the pr_assign_workbasket table.
- Worklist assignments are stored in the pr_assign_worklist table.
- The Work-base class is mapped to the pc_work table, a table with columns that represent all of the work object properties that need to be exposed for flow processing to function correctly. Your work object classes should be mapped to either the pc_work table or another table that has the same columns.
- The Data-base class is mapped to the pr_data table. Your data classes should be mapped to either the pr_data table or another table that has the same columns.

For a report of all the existing table mappings, complete the following steps:

1. From the menu, choose Tools > System Management Application.
2. In the System Management application, choose Advanced > Reports > PegaRULES Database Information Reports.
3. Click Database Table Report.
4. Click the Download link.

How Process Commander Finds the Table When Saving an Object

When saving an object, Process Commander uses pattern matching to find a table mapping for an object of the same type. Process Commander examines the list of table mappings (database table objects), looking for one that matches the class. If there is no table mapping for the class, Process Commander follows the pattern inheritance tree of the class to find a table mapping for the parent of that class, and
then a table corresponding to the parent of that class, and so on. If it does not find a table mapping by the time it reaches the top of that class’s pattern hierarchy, Process Commander stores the object in the pr_other table. It does not progress to the direct inheritance tree to find a match.

The pr_other table does not have columns for all of the work object properties that must be exposed for work objects to successfully participate in flows. Therefore, if you do not map a work class to an appropriate database table, your work objects exhibit unpredictable behavior. You map work object classes to database tables by mapping their class group, not the individual work classes.

Creating Database Name Objects

Your applications can interact with an external database either through SQL statements stored in SQL connector rules or through the Obj- activity methods. When you use activity methods to interact with an external database, you must map classes to the external tables. Before you can create database table objects that map classes to tables, you must create a database name object that represents the external database and that holds the information Process Commander needs to connect to it. (Note that you also need to create a database name object to support SQL connector rules.)

Before you begin, complete the following tasks:

- In the external database, create or identify a database user account for Process Commander access.
- Make the appropriate JDBC driver available to the application server running Process Commander. Then create a JDBC data source for the database in the application server and write down the JNDI name of the data source. If you need help with this task, consult the documentation for your application server software.

To create a database name object, complete the following steps:

1. In the Rules by Type explorer tree, choose SysAdmin > Database. A list of databases appears.
2. Click New.

3. Name the database and click Create.

4. On the Database tab, click in the How to Connect field and choose Use JDBC Connection Pool from the list.

5. In the JDBC Datasource Settings field, enter the JNDI name of the data source (Figure 5-3).

![Database Name Form](image)

*Figure 5-3. Database Name Form*

6. On the History tab, enter descriptions in the Full Description and the Usage fields.

7. Save the database definition.

8. Click Test Connection to verify that the data source and database name object are configured correctly.
Creating Database Table Objects (Mapping Classes)

Before you can map classes to tables, the following must be true:

- The database the table belongs to must have a database name object (Data-Admin-DB-Name) that represents it.
- The table must be present in the schema of that database.
- The class you want to map must exist.

**Caution:** Changes to table mappings take effect immediately. The best practice is to design the entire data model application, including the table mappings for it, during the application design process. You then map classes as you create them, before anyone is actually using them.

If you remap an existing class that already has instances to a different table, you must move the existing instances of the class from the original table to the new table or the existing data will be orphaned. In such a case, make the change when no one is using the system and move the data before you allow users back on the system. If you remap an abstract class whose descendents inherit the mapping, you must also move all instances of its descendents.

Work with your DBAs to create new databases, move data, or change the database schema.

Complete the following steps:

1. In the Rules by Type explorer tree, choose SysAdmin > Database Table. A list of table mappings appears.
2. Click New.
3. Choose the class you want to map and click Create.
4. On the Database tab, choose the database that holds the table you want to map the class to.
5. Enter the name of the table in the Table Name field (Figure 5-4).
6. If necessary, enter a catalog and schema name that locates the table.

7. On the History tab, enter descriptions in the Full Description and the Usage fields.

8. Save the table mapping.

9. Click Test Connectivity to verify that Process Commander can connect to the table.

Creating External Data Table Rules

When your developers create applications that interact with external databases by using the Obj-activity methods, you or the developers must implement in Process Commander a data model that represents the data model of the external system — that is, you create external data table rules. To do this, you use the External Data Table Wizard.

The External Data Table wizard creates the following rules:

- One class rule that represents the external table. This class is known as an Obj-external class because the data it represents is not stored in the PegaRULES database.
Managing the Data — Mapping Classes to Tables

- One model rule for that class rule.
- One property rule for each column in the external table.
- One table mapping instance (database table object) that represents the table and maps the class rule to the table. Note that there can be only one Process Commander class mapped to the external table.

Developers can then use the class rule that represents the external table as they would any other data class in Process Commander. When an activity uses the Obj-Open and Obj-Save activity methods on instances of the class rule, those methods retrieve data from or save data to the external table.

Before you can use the External Data Table Wizard, you must create a database name object that represents the external database. See “Creating Database Name Objects” on page 5-9.

To run the External Data Table wizard, click the Integration button on the home page. From the Wizards section, choose External Data Table Rules. If you need help with any of the forms the wizard displays, click the Help button.

Creating and Mapping Class Groups

Typically, developers use the Application Accelerator to generate a framework or rules and data objects for their applications. The accelerator generates class groups and maps the class groups to the pc_work table automatically.

This section describes how to create and map a class group manually. If you do not map a class group to a table, the work objects from that class group do not get stored in the pc_work table. Instead they are stored in the pc_other table and cannot participate in the rest of the process that follows the creation form.

There are three parts to creating a class group manually:

1. Create a concrete class and specify that it is a class group. When you save it, Process Commander automatically creates a class group data object and a history class, too.
2. Map the class group data object that represents the concrete class and its descendents to the appropriate work table.

3. Map the history class to the appropriate history table.

**Create the Class Group Class**

Complete the following steps:

1. In the Rules by Type explorer, choose SysAdmin > Class. A list of classes appears.

2. Click New.

3. Enter the name of the concrete class and click Create.

4. Under Settings, choose Concrete as the type and choose the RuleSet version.

5. In the This Class field, choose Is a Class Group.

6. Fill out the Class Inheritance fields as appropriate. Typically, the directed inheritance parent class for a class group class is the Work- base class.

7. Click in the Key Name field and choose the property that serves as the primary key for all instances of the classes that are to be grouped by the class group (Figure 5-5). Typically, you create class groups to group work so the primary key would be the pyID property. (The value of the pyID property is generated by Process Commander when it creates instances and is unique for each instance.)
8. For help with the fields on the Restricted and Advanced tabs, click the Help button.

9. On the History tab, enter descriptions in the Full Description and the Usage fields.

10. Click Save. Process Commander saves the class and generates both a history class and a class group data object to represent the class (Figure 5-6). A warning message states that this class group is mapped to the pr_other table.
Map the Class Group

You map the class group class as you do any other class. Complete the steps in the procedure “Creating Database Table Objects (Mapping Classes)” on page 5-11. You map class groups to either the pc_work table of the PegaRULES database or a table that is based on the pc_work table.

Figure 5-7 shows the database table object for the example class group in Figure 5-5 and Figure 5-6.
Map the Class Group History Class

You map the generated history class as you do any other class. Complete the steps in the procedure “Creating Database Table Objects (Mapping Classes)” on page 5-11. You map history classes for class groups to either the pc_history_work table of the PegaRULES database or a table that is based on the pc_history_work table.

About the Children Classes

All classes that belong to the class group must share a common key definition – typically pyID – and lock definition, so Process Commander can save them as rows in the table the class group is mapped to.

For more information about locking strategies, see the Pega Developer Network (PDN) article PRKB-18033.
Data and Reporting

Reports are implemented with the list view and summary view rule types, instances of which the Report Wizard generates for you.

While your application developers are planning and implementing the data design for your applications, be sure to consider your reporting needs. You may determine that additional standard properties need to be exposed and/or that new properties created for the classes that represent the data model of your applications need to be exposed.

Work with your DBAs to make any schema changes that are necessary to support the reports you want to use to manage your applications. With appropriate, careful planning, your DBA can adjust the schema of the database table and create columns that represent the properties that you need for your reports.

When you determine that you need to expose properties for reporting, consider the following guidelines:

- For single value, non-embedded properties, you can directly change the schema of the existing pc_work tables or other storage tables by adding new columns for the properties. For information, see “Exposing Property” which is next in this section.

- For complex or embedded properties, you must use declarative indexing. For information, see “Declarative Indexing” on page 5-24.

To see the columns for any of the tables in the PegaRULES database, you can use either the Database Schema Wizard or the database schema reference, which is located on the installation media.
Exposing Properties

It is likely that you and your DBAs will use your own database tools to add columns to tables. However, in some cases – if your database does not rely on schema names, for example – you can use the Modify/View Database Schema wizard.

The wizard enables you to accomplish this task in the following ways:

- If you have the appropriate credentials – a database user account that grants you the necessary rights – you can use the wizard to create the column directly.
- If you do not have clearance to make schema changes on your own, use the wizard to create a SQL script that you can give to your DBA.

Before you begin, complete the following tasks:

- If you have the appropriate access rights, obtain a database user ID and password that lets you modify the schema. Otherwise, you can generate a SQL script for your DBA to use.
- For each property you want to expose, determine which table stores instances of the Applies To class.

**Important:** Verify that you are proceeding with this task at a time when there are no other users on the Process Commander system – do not make schema changes while anyone else is using the system.

To use the Schema Wizard, complete the following steps:

1. From the menu, choose Tools > Database > Modify a Database Schema. The Database Selection form appears.
2. Choose a database and click Next. The Table Selection form appears.
3. Choose a table from the list of tables in the database you selected. Then click Next. The List of Classes form displays a list of all the Process Commander classes that are mapped to the table you selected.
4. Find the class with the property for which you want to create a column (expose) and then click the number in the “Properties set to be visible” column. The Exposed and Unexposed Properties form appears, listing the properties that can be exposed and the properties that are already exposed.

5. Choose each property you want to create a column for. For properties with a data type of Text, Identifier, or Password, you can also modify the column width. (If you change these values, be sure to verify that the Max Length value on the property rule form matches the value you specify.)

6. Do one of the following:
   - If you have the appropriate access rights, choose the Generate Database Columns option and then enter the database user ID and password in the appropriate fields.
   - If you do not have the necessary access rights to make schema changes, choose the Generate SQL Code option.

7. Click Create Selected Columns. Depending on which option you selected, Process Commander either adds the columns to the database or generates a SQL script.

8. Click Close.

9. If you generated a script, submit it to your DBAs so they can run it for you.

**Resaving Records**

Whether you use your own database tools or the Process Commander Schema Wizard, when you expose a property as a database column, existing records do not automatically move data to that column. Property values for new work objects created after the change are stored in the column, but previously saved work objects still have values for the exposed property stored in the blob column only. Running a report that includes records saved before the property was exposed will show “null” as the value for that property for the old records.
If you want all records – existing and new – to use the new column, you must resave the existing data. To do so, you use the resave function of the Process Commander Database Utilities application after you have created the new column.

The URL of the Database Utilities application is:

```
http://servername:portnumber/prdbutil
```

For example, if the Process Commander URL is:

```
http://serverX:8080/prweb/PRServlet
```

The URL of the database utilities servlet is:

```
http://serverX:8080/prdbutil
```

Before you begin, open and save the appropriate database name or database table object so Process Commander notices the change to the schema. If you changed one or two tables, open the database table objects that map instances of classes to those tables and resave them. If you changed several tables in the same database, open the database name object that represents that database and resave it. That way, all the table mappings that refer to that database will be refreshed.

1. Navigate to the URL of the Database Utilities application.

2. In the navigation panel of the main window, under Utilities, choose Resave PegaRULES Process Commander Application Instances. The Resave Rule Base Instances window appears (Figure 5-8)
3. Enter the name of the class whose instances must be resaved in the **Include classes** text box. Insert hard returns between each class name so there is one name on a line. If you want to resave the instances for a class plus all its descendant classes, use the **Include classes with descendents** text box rather than the **Include classes** text box.

**Important:** Be careful when entering class names in the **Include classes with descendents** text box. You could make an entry that would cause the entire database or a very large potion of it to be resaved, resulting in excessive and unnecessary processing. Be sure to specify both classes to include and to exclude so you are resaving the minimum number of instances possible.

4. Use the **Exclude classes** and **Exclude classes with descendents** text boxes to specify classes in the inheritance tree whose instances should not be resaved.

5. If you are resaving instances at a time when people are using the Process Commander system, choose the **Lock instances** option. For best performance,
resave instances at a time when the system is not in use so you do not have to use the **Lock instances** option.

6. Click **Resave**.
Declarative Indexing

If the data design for your applications includes embedded page properties or other complex properties whose values you must include in reports or whose values you need to use for sorting and queries, configure declarative indexing for those properties to expose them.

Overview

Declarative indexing is implemented through the following items:

■ Additional database tables (index tables)
■ Process Commander classes that inherit from the Index-base class (index classes)
■ Declarative index rules

The rules map the individual elements of a complex property from the source object class to the index class. You use a database table mapping to map the index class to the appropriate index table.

Where the Data Goes

When a work object that has a declarative index rule defined for it is saved, the data for the mapped property is stored in two places: the blob column in the database table the source class is mapped to, and the index table the index class is mapped to. Each instance of the embedded page property upon which an index is defined is one entry in that index table, with the appropriate values saved in each property column.

When a change is made to an instance of a class for which a declarative index rule is defined, the current data for the source properties defined in the rule is updated in both the appropriate index table and the source table.

You write your reports or code your activity queries against the index class rather than the source work object class. Because the index class holds only the values of interest – not the whole blob – reports and queries on that data are generated faster and more efficient than if they are run against the source table.
**Note:** There are both duplication of storage costs and system resource costs associated with declarative indexing. The property values of indexed properties are stored twice: once in the source table and once in the index table, which means you need more space in your database when you are using declarative indexing. Additionally, when instances are saved, the system takes time to determine whether indexes need to be updated and it takes more time to save the index records along with the source record.

**Configuration Steps**

Designing declarative indexes is a task typically performed by a developer, but you may be asked to implement the design and associated database setup. Following is an overview of the implementation steps:

1. Plan the indexing strategy: determine which properties for which work classes need to be indexed.

2. Create database tables to map the Process Commander index classes to. The tables should contain the same columns that are in the pr_index table without the pzPVStream column.

3. Create the Process Commander index classes and create properties to use for mapping the components of the complex property from the work class.

4. Map the index classes to the index tables with table mapping rules.

5. Map the properties from the source work object classes to the index classes with declarative indexing rules.

**Standard Declarative Indexing Rules**

Process Commander has several standard declarative indexing rules that support flow processing. These indexing rules map the values from the standard Work-classes to standard Index- classes, and those index classes are mapped to standard index tables.
You can examine the standard rules to see how they are set up, but do not map your (custom) index classes to the standard index tables. Instead, be sure to have your DBA create index tables specifically for your index classes.

Planning the Structure of a Declarative Index

There does not have to be a one-to-one relationship between index classes and the index tables in the database. As with work and data classes, several index classes can be stored in one table, as long as the data is compatible. Carefully design the grouping of index-classes to index database tables to maximize efficiency. For example, if you have defined several declarative indexes that all reference properties associated with client information, the entries could all be stored in the same database table. However, an index that references properties associated with product information would more logically be stored in a separate index table.

Declarative index rules follow the same inheritance patterns as the rule resolution process. When determining whether to track property information, the system looks up all instances of rules that apply, including those that apply due to inheritance and then uses the most appropriate one. Circumstance and date/time ranges are not used for rule resolution processing for declarative indexes. Therefore, depending on the profiles of the users involved, rule resolution may cause different users to use different versions of an index, even though they are working on the same class instance.

Creating Index Database Tables

By default, the Index-class is mapped to the pr_index database table. Your index classes should be mapped to their own database tables; do not map them to the pr_index table. Instead, be sure to have your DBAs create index tables specifically for your index classes.

The process of making database tables depends on your database software but in general consists of these steps:
1. Copy the pr_index table to make your table. Do not include the pzPVStream column.

2. Ensure that required columns are created.

3. Create additional columns for the work object properties you want to index.

**Creating and Mapping the Index- Class**

Next, create the index class and its properties and then map the class to the index table you created for it. Process Commander provides a top-level base class called Index-; your index classes must inherit from this class using direct inheritance.

**Create the Index Class**

Complete the following steps:

1. From the Rules by Type explorer tree, choose SysAdmin > Class.

2. Click New and name the new class.

3. Define the class with the following characteristics:
   - Concrete
   - Does not belong to a class group
   - The Index- base class or a direct descendant is its parent through direct inheritance

4. In the Keys section, specify the following properties as keys to the entries saved for this class, in the following order:
   - pxInsIndexedKey
   - pxIndexCount
   - pxIndexPurpose

Figure 5-9 shows an example.
5. Save the index class.

**Create the Index Properties**

Your index class already inherits the properties from the Index-base class. Now add the properties that represent the complex property from the work class for which you are building the index.

On the Rules by Type explorer tree, choose Property > Property to create the properties for your index class.
Map the Class to a Table

Map the index class to the index table you created in the PegaRULES database. Complete the steps in the procedure “Creating Database Table Objects (Mapping Classes)” on page 5-11.

Creating Declarative Index Rules

Before you begin, identify the following components:

- The index class.
- The source work class whose complex property you are creating the index rule for.
- The complex property you are mapping and the context of that property – for example, the page class of an embedded page property.

To create a declarative indexing rule, complete the following steps:

1. In the Rules by Type explorer tree, choose SysAdmin > Declare Index. A list of rules appears.

2. Click New.

3. In the New form, choose the class that the index applies to, enter the purpose (name) of the rule, and specify the RuleSet name and version. Click Create.

**Note:** You cannot create a declarative indexing rule for a class that represents an external table.

4. In the Index Class Data section of the Indexes tab, specify the source property (and its class) and the target index class. For an example, examine the standard declarative index rule Work-.PartyURI (Figure 5-10).
This rule maps work party information from classes that inherit from the Work-base class to the Index-WorkPartyURI class. Fill out the fields in this section as described in Figure 5-11:

**Figure 5-10. Standard Declare Index Rule Named Work-PartyURI**
<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source Page Context</td>
<td>Enter the complete property reference for the property whose component parts are to be mapped. For example, in the PartyURI rule, it is .pyWorkParty().</td>
</tr>
<tr>
<td>Source Page Context Class</td>
<td>Specify the page class of the property specified in the Source Page Context field. For example, the page class of the pyWorkParty() property is Data-Party.</td>
</tr>
<tr>
<td>Index Class To Write</td>
<td>Specify the target index class. For example, the index class for the PartyURI rule is Index-WorkPartyUri.</td>
</tr>
</tbody>
</table>

*Figure 5-11. Index Class Data fields*

5. In the property mapping section of the Indexes tab, map the components of interest from the source property to the appropriate index class property. Note the following guidelines:
   - The source property and the index class property must be of the same data type.
   - The index class property must evaluate to a scalar property.
   - Use the keyword “parent” followed by the period (.) character to refer to the page in which the class identified in the Source Page Context Class is embedded. For example, in the WorkPartyURI rule, the keyword “parent” means the pyWorkPage page. Note that you cannot nest the parents. That is a reference such as parent.parent.MyProperty is not valid.
   - Use the keyword “top” followed by a dot to refer to the top-level page in which the class identified in the Source Page Context Class is embedded.

6. Complete the History tab with a full description of the declarative index and its use.

7. Save the rule.
Declarative index processing begins as soon as any properties in the source class properties identified in the Declare Index rule change in value.

**Caution:** After you have created an index class and a declarative indexing rule that maps values to those properties from a source class, index records are created automatically. Do not alter the value of the properties in your index classes through any other processing.

If you add declarative indexing for a class that already has data and you want all records – existing and new – to use the new column, you must resave the existing data. Use the Resave function of the Process Commander database utilities servlet. For information, see “Resaving Records” on page 5-20.
Creating Reports

Use the Report wizard to define and customize interactive reports for yourself or others who share an access group.

The Report wizard presents a series of forms that prompt for each element needed to define a report. When you finish the forms, Process Commander creates a list view rule or summary view rule. If you have the appropriate access role and RuleSets, you can later update the created summary view or list view rule directly.

To see the standard reports, click the Monitor Activity button in the main window of the Process Commander user interface (Figure 5-12).

Reports are listed under the following categories:

- Rule Reports
- Monitor Assignments
- Monitor Processes
To start the Report wizard (Figure 5-13), click the New button in a section toolbar on the Monitor Activity page.

![Report Wizard](image-url)

*Figure 5-13. Report Wizard*

Then follow the instructions displayed on each form to construct your report.
To create reports without using the wizard, select the rule types from the Rules by Type explorer. Choose Reports > List View or Reports > Summary View. Then click New.

For help with creating list view or summary view rules without using the Report Wizard, see the Application Developer Help.