WebSphere. software

IBM

Concepts

IBM WebSphere Process Server

Query Table Builder
SupportPac PA71

Version 1.0
December, 2008

Table of contents

Abstract

This document describes Business Process Choreographer Query Tables, which is new functionality in WebSphere Process Server, version 6.2. Query tables are used in the context of the human tasks and business processes of Business Process Choreographer.

A common usage pattern of business process and human task applications is as follows: a list of human tasks or business processes is presented to a user. The user picks and works on a business process or a human task, and finally returns to the refreshed list, for example My To Dos. In order to display the list of human tasks or business processes, a query is run against the Business Process Choreographer database.

Query tables allow defining a customized view on the data that is contained in the Business Process Choreographer database; inclusion of external business data is also supported. Query tables are defined by the designer of the client application according to the needs of the client. That is, a query table is defined in such a way that it contains all the columns that are displayed when the business user later retrieves the respective task list or business process list. Query tables allow optimizing for query performance without changing a query table's definition. They are highly configurable, and are developed visually using the Query Table Builder tool.

The target audience of this document is administrators, architects, and solution specialists that create and manage business process based or human task based applications on WebSphere Process Server.

1. Overview

Query tables support task and process list queries on data that is contained in the Business Process Choreographer database schema. This includes human task data and business process data, both managed by Business Process Choreographer, as well as external business data. Query tables provide an abstraction on the data of Business Process Choreographer that can be used by client applications. By that, client applications become independent of the actual implementation of the query table. Query table definitions are deployed on Business Process Choreographer containers and are accessible using the query table API.

Using query tables has an impact on the way applications are developed and deployed. The following steps describe the roles involved in designing and developing a Business Process Choreographer application that uses query tables:

Table 1: Query table development steps

Step	Who	Description
1. Analysis **(Refer to sections 2, 6)**	**Business Analyst,** **Client Developer**	**Analyze which query tables are needed in the particular client application. Questions to be answered are:** • **How many task or process lists are provided to the user? Are there task or process lists that can share the same query table?** • **What kind of authorization is used? Instance based authorization, role based authorization, or none?** • **Are there other query tables already defined in the system that can be re-used?**
2. Query Table Development **(Refer to sections 2, 03, 5, 6)**	**Client Developer,** **Business Analyst**	**Develop the query tables that are used in the client application. Try to specify the definition of the query tables such that the best performance is achieved with query table queries.**
3. Query Table Deployment **(Refer to section 4)**	**Administrator**	**Query tables must be deployed to the runtime before they can be used. This step is done using a new wsadmin command which uses the command line tools in the WebSphere Process Server scripting environment.**

4. Query Table Queries **(Refer to sections 52, 5, 6)**	**Client Developer**	**To include queries against query tables is the last step of query table development. The client developer must know the name of the query table along with its attributes.**

This document is structured as followed:

- **Section 2 explains the elements of query table definitions: attributes, filters, selection criteria, and authorization options.**
- **Section 3 contains a short notice about the Query Table Builder.**
- **Section 4 provides an overview of query table deployment.**
- **Section 5 introduces the query table API.**
- **Section 6 explains factors which influence the performance of query tables; the focus is on query response time.**
- **Section 7 provides a comparison between the query table API and the standard query API.**

Note: Currently, query tables are not supported if the Business Process Choreographer database is on Informix 10.

2. Model

There are three different kinds of query tables: predefined, supplemental, and composite query tables:

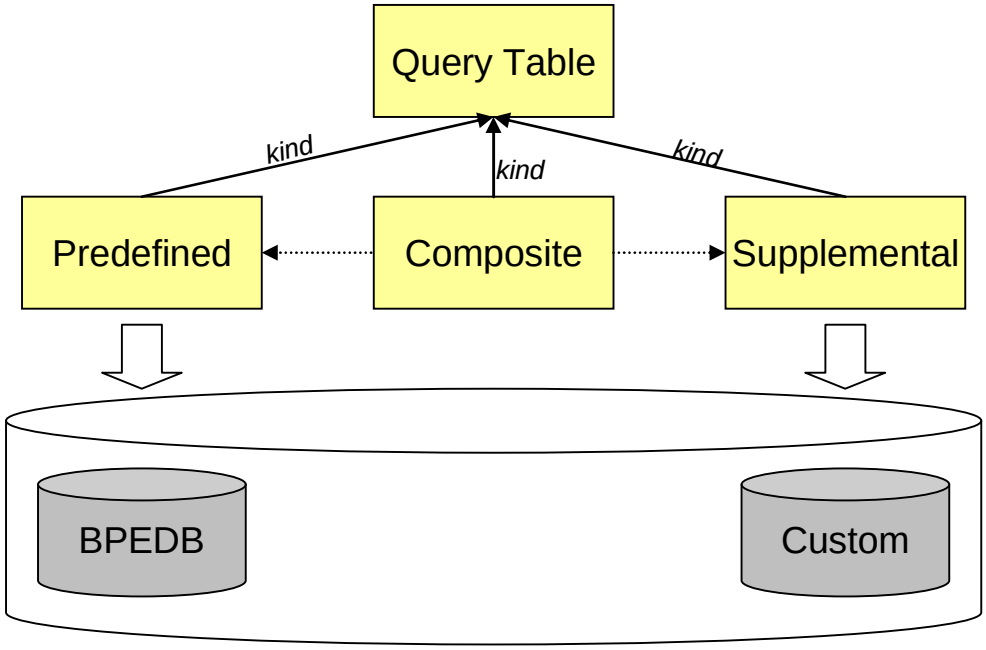


Figure 1: Query table kinds

While predefined and supplemental query tables directly point to tables or views in the database, composite query tables compose parts of this data and make it available in a single query table.

Predefined, supplemental, and composite query tables are represented using similar models in the query table runtime and can be queried using the query table API.

Composite query tables are designed by client developers. They are suggested for use in production scenarios in favor of the standard Business Process Choreographer query APIs, because they provide an abstraction over the actual implementation of the query and thus enable query optimizations. They also allow for a fine-grained configuration of authorization and filters. Furthermore, composite query tables allow changes at runtime without redeployment of the client that accesses the query table.

Predefined and supplemental query tables relate to database tables or database views in the Business Process Choreographer database. Supplemental query tables are query tables that contain supplemental business data that is maintained by customer applications. Currently, composite query tables do not have a specific representation of data in the database; they access the database contents of the related predefined and supplemental query tables.

2.1. Common properties of query tables

Query tables have the following properties:

Table 2: Properties of query tables

Property	Description
Name	The query table name must be unique within a Business Process Choreographer installation. When the query is run, this query table name is used to identify the query table that is queried.
Attributes	Attributes of query tables define the pieces of information that are available for queries. For predefined query tables, these are the columns specified by the predefined database views. Attributes of query tables are defined with a name and a type. The name of an attribute is defined with uppercase letters. The type of an attribute is one of the following: • boolean: A boolean value • decimal: A floating point number • ID: An object ID, such as TKIID of query table TASK • number: An integer, short, or long • string: A string • timestamp: A timestamp
Authorization	Each query table defines whether instance based authorization is used when queries are run on it. If instance based authorization is not used, the query is run without checks against the existence of work items of the related objects in the query table. If instance based authorization is used, only objects with a work item for the user who performs the query are returned. However, with using the AdminAuthorization-Options this check can be reduced to a check of the existence of a work item of any user. The user must be in the J2EE role BPESystemAdministrator for those queries.

2.2. Predefined query tables

Predefined query tables in Business Process Choreographer are the query table representation of the corresponding predefined Business Process Choreographer database views, such as TASK or PROCESS_INSTANCE. They provide access to the data in the Business Process Choreographer database. Accessing predefined query tables using the query table API offers more options for configuration than accessing the predefined database views using the standard query API. The query table API is described in section 5.

Although the predefined query tables can be queried directly using the query table API, the suggested use of query tables is to develop a composite query table that contains all the information to be retrieved when the query is run, not just the information from a single table.

2.2.1. Predefined query tables with instance data

Query tables in the following table:

- **Can be used as the primary query of a composite query table. See chapter 2.4.1 for details.**
- **Use instance based authorization if queried directly. This is accomplished with a join (SQL-) with the view that stores authorization information, that is, the predefined WORK_ITEM view or query table.**
- **Contain instance data, for example instance data of task instances or process instances.**

Table 3: Predefined query tables containing instance data

Query table name	Description
ACTIVITY	Information about activities of a process instance.
ACTIVITY_ATTRIBUTE	
ACTIVITY_SERVICE	
ESCALATION	Information about escalations belonging to human tasks.
ESCALATION_CPROP	
ESCALATION_DESC	
PROCESS_ATTRIBUTE	Information about process instances.
PROCESS_INSTANCE	
QUERY_PROPERTY	
TASK	Information about human tasks.
TASK_CPROP	
TASK_DESC	

The information contained in the WORK_ITEM query table also contains instance data, but this is not available as the primary query table or an attached query table. This reflects the fact that in Business Process Choreographer work items represent authorization of a certain kind, whether reader authority, administrator authority, potential owner authority, and so on, on a given object such as a human task or business process. This is different to systems such as WMQWF. For more details, refer to the Business Process Choreographer Programming Model white paper [BPCProgModel]. Work item information is available implicitly when querying query tables that use instance based authorization. That is, attributes of the WORK_ITEM query table can be used when querying a query table with instance based authorization, although not explicitly specified by the query table.

2.2.2. Predefined query tables with template data

Query tables in the following table:

- **Can be used as the primary query table of a composite query table.**
- **Require administrator authorization if queried directly.**
- **Contain template data, for example the template data of task templates or process templates.**

Table 4: Predefined query tables containing template data

Query table name	Description
ESC_TEMPL	Information about escalation templates.
ESC_TEMPL_CPROP	
ESC_TEMPL_DESC	
PROCESS_TEMPLATE	Information about process templates.
PROCESS_TEMPL_ATTR	
TASK_TEMPL	Information about task templates.
TASK_TEMPL_CPROP	
TASK_TEMPL_DESC	

2.3. Supplemental query tables

Supplemental query tables in Business Process Choreographer expose data to the query table API that is not managed by Business Process Choreographer. Supplemental query table are often used by customers to make additional business data available. With supplemental query tables, this external data can be used together with data from the predefined query tables of Business Process Choreographer when retrieving business process instance information or human task information.

Instance based authorization with work items is not supported for supplemental query tables. All authenticated users can access the contents of supplemental query tables.

The purpose of supplemental query tables is to provide information in a composite query table in addition to information that is contained in a predefined query table. Supplemental query tables should not be used in order to simplify programming when accessing database tables or views from client applications without being correlated to Business Process Choreographer data.

The following properties are defined on a supplemental query table:

Table 5: Properties on supplemental query tables

Property	Description
name	The name of the supplemental query table. It must follow the syntax of *prefix.name*. Only uppercase letters may be used. The total length is restricted to 28 characters. The prefix must be different from the reserved word 'IBM'.

database name	**The name of the related table or view in the database. Only uppercase letters may be used.**
database schema	**The schema of the related table or view in the database. Only uppercase letters may be used. The database schema should be different to the database schema of the Business Process Choreographer database. Nevertheless, the table or view must be accessible with the same JDBC data source that is used for accessing the Business Process Choreographer database.**
attributes	**Attributes on supplemental query tables must match the related name of the columns in the related database table or view. Only uppercase letters may be used. See section 2.7 for rules that apply for the type of an attribute and the type in the database. Section 2.7.1 contains an example with sample attributes referencing database columns of a database table of which the create table statement is provided.**
join	**Joins must be defined on supplemental query tables if they are attached in composite query tables. A join defines which attributes are used to correlate information in the supplemental query table with the information in the primary query table. When a join is defined, the source attribute and the target attribute must be of the same type.**

2.4. Composite query tables

Composite query tables are composed of predefined query tables and supplemental query tables. They combine data from existing tables or views. Typically, a composite query table is used to retrieve the information that is shown on a process instance list or a task list, such as "My To Dos". Composite query tables allow filters and authorization options for optimized data access when the query is run.

A query table is uniquely identified using its name, which is defined as *prefix.name*. The maximum length of the *prefix.name* is 28 characters, and the prefix must be different from the reserved prefix 'IBM'.

2.4.1.Content

Figure 2 provides an overview of the content of composite query tables:

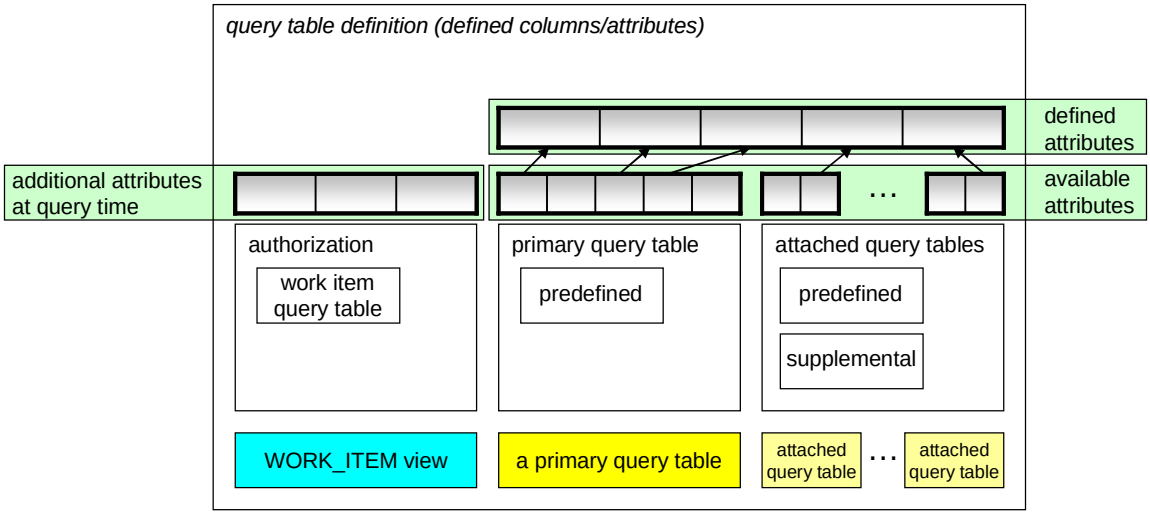


Figure 2: Query table contents overview

All composite query tables are defined with one *primary query table* and zero or more *attached query tables*. Information contained in the primary and attached query tables provide the set of information, or available attributes, which can be defined to be part of the query table. If the primary query table contains instance data, such as TASK or PROCESS_INSTANCE, and instance based authorization is used, work item information can be retrieved when running the query. This is in addition to the attributes that are part of the query table definition (see section 02.2 for a list of available predefined query tables).

The primary query table must be one of the predefined query tables as listed in section 0. The primary query table of a composite query table is used for:

- **Authorizing the contents of a query table using work items which are contained in the WORK_ITEM query table, if instance based authorization is used.**
- **Determining the list of objects returned as rows of a table when querying the composite query table. Information which is contained in the attached query tables or in the WORK_ITEM query table is available in the composite query table only if it is related to a row that is contained in the primary query table.**

Typically, the primary query table is chosen based on the purpose of the composite query table. If the composite query table describes a task list, the query table TASK is the primary query table. If the composite query table describes a process list, the query table PROCESS_INSTANCE is the primary query table. Lists of activities are retrieved using a primary query table, ACTIVITY. Lists of human task escalations are retrieved using a primary query table, ESCALATION. Various other predefined query tables are available as a primary query table, but they are intended to be used for special purposes only.

Attached query tables are available in order to provide information in addition to the information that is provided by the primary query table. For example, if TASK is the primary query table, the description of the task, provided in query table TASK_DESC, can be added to the contents of the composite query table.

Work item information can be queried at runtime if the primary query table contains instance data and if the composite query table is configured to use instance based authorization.

2.4.2. The relationship between primary and attached query tables

The relationship between a piece of information, or row, in the primary query table and the information, or row, which is added with an attached query table is described as follows. At maximum, one single row of the attached query table must qualify for a corresponding row in the primary query table, which is referred to as one-to-one or one-to-zero relationship. If the one-to-one or one-to-zero relationship is violated, a runtime exception occurs when the query is run.

Primary query tables and attached query tables are correlated using a join attribute that is defined on the attached query table. This join attribute cannot be changed for predefined query tables, because it describes the relationship between the data in the various query tables of Business Process Choreographer. Frequently, this join attribute is sufficient to maintain the one-to-one or one-to-zero relationship. For example, CONTAINMENT_CTX_ID is used on the TASK query table to attach the related process instance information that is identified by the PIID attribute on query table PROCESS_INSTANCE. However, in cases where a one-to-many relationship exists, an additional criterion must be specified. This is called the selection criterion.

Selection criteria are used in the query table definition in order to choose one piece of information from the one-to-many relationship. In the sample contents of Table 6, this is "`LOCALE='en_US'`". Many descriptions identified using different locales exist for a single task. Selection criteria which are based on the Query Table Condition Language are specified during query table development using the Query Table Builder tool. See section 2.5 for details.

Example 1: The following figure provides a sample visualization of the selection criteria that is specified on attached query tables:

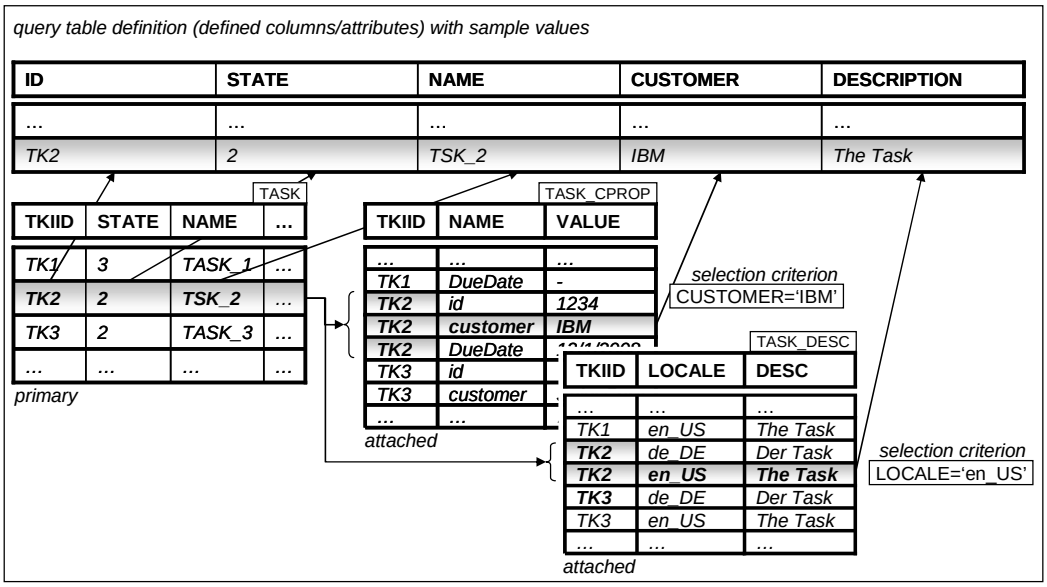


Figure 3: Sample composite query table with selection criteria

The query table shown in Figure 3 contains the attributes ID, STATE, NAME, CUSTOMER, and DESCRIPTION.

ID, STATE, and NAME are provided by the primary query table TASK.

CUSTOMER is a custom property on TASK. Custom properties are stored in the query table TASK_CPROP. For a particular task, a custom property is uniquely identified using its name. This is reflected in the selection criterion "CUSTOMER='IBM'".

DESCRIPTION is the task's description, stored in query table TASK_DESC. The task description for a particular task is uniquely identified through its locale. This is reflected in the selection criterion "LOCALE='en_US'".

Example 2: For example, if TASK is the primary query table and TASK_DESC is attached to it, a particular locale must be chosen, which is attribute LOCALE of query table TASK_DESC. The focus of this example is on the relationship between the primary and the attached query tables, using TASK as the primary query table and TASK_DESC as the attached query table. Table 6 shows sample contents of a composite query table with a valid selection criterion for the attached query table TASK_DESC. Table 7 shows hypothetical invalid contents if the selection criterion is set incorrectly, which means that the one-to-one or one-to-zero relationship is violated:

Table 6: Valid contents of a composite query table

Information from TASK (primary query table)	*Information from TASK_DESC (attached query table)*	
NAME	LOCALE	DESCRIPTION
task_one	en_US	This is a description.
task_two	en_US	This is a description.
...	...	...

Table 7: Invalid contents of a composite query table: TASK as primary and TASK_DESC as attached query table

Information from TASK (primary query table)	*Information from TASK_DESC (attached query table)*	
NAME	LOCALE	DESCRIPTION
task_one	en_US	This is a description.
task_one	de_DE	Das ist eine Beschreibung.
...	...	...

2.4.3.Filters

Filters are used to limit the number of objects, or rows, that are contained in a composite query table:

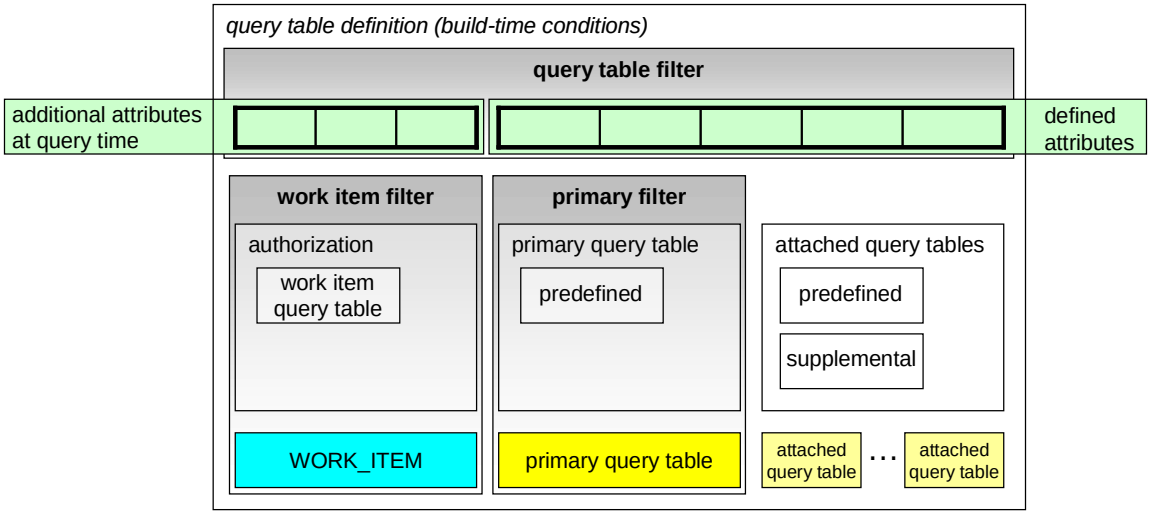


Figure 4: Filters in composite query tables

Filters in composite query tables can be defined during development at three different locations:

- **On the primary query table, this is the *primary query table filter.***
- **On the implicitly available WORK_ITEM query table which is responsible for authorization if the primary query table contains instance data. This filter is called the *authorization filter*. It is available only if the composite query table is configured to use instance based authorization.**
- **On the composite query table, this is the query table filter.**

Filters are defined using the query table condition language, as described in section 2.5. For example, a composite query table with the primary query table TASK can filter on tasks that are in state ready ("STATE=STATE_READY" as the primary query table filter).

2.4.4.Authorization

Authorization for accessing the contents of a composite query table with a primary query table *primaryQT* is similar to the authorization that is used to access the contents of *primaryQT*. The difference is that composite query tables can be configured to be more restrictive. Only predefined query tables are available as the primary query table.

Composite query tables with a primary query table that contains instance data: They can be configured to not use instance based authorization. In this case, all users have access to the contents of the composite query table.

If instance based authorization is configured for use, the data contained in the composite query table is checked for existing work items in query table WORK_ITEM. This check is made against the primary query table. Everybody work items, individual work items, group work items, and

inherited work items are used for this check, depending on the configuration of the composite query table.

Composite query tables with a primary query table that contains template data: The use of role based authorization of composite query tables with a primary query table that contains template data cannot be changed. Queries against those query tables can be run only by users that are in the J2EE role BPESystemAdministrator of Business Process Choreographer. The AdminAuthorizationOptions object must be used for those queries.

For more details on authorization, see section 2.6 and 5.2.3.

2.4.5. Attributes

Attributes of composite query tables are defined using a reference to attributes of the primary query table or the attached query tables, which can be predefined query tables or supplemental query tables. Both types and constants of referenced attributes are inherited from the attributes of the composite query table. For more information on attributes and its related types, see section 2.7.

2.5. Query table condition language

Filters and selection criteria (see section 2.4) are defined with the query table condition language. The query table condition language is used to specify a condition which evaluates to *true* or *false* at runtime. This language is similar to SQL where clauses.

2.5.1. General

An expression in the query table condition language is defined as follows:

Table 8: Query table condition language: expressions

<expression> ::= <attribute> <binary_op> <value> | <attribute> <unary_op> | <attribute> <list_op> <list> | (<expression>) | <expression> AND <expression> | < expression> OR <expression>
Rules: • **AND takes precedence over OR.** • **Brackets can be used to group expressions and must be balanced.**
Examples: • STATE = STATE_READY • NAME IS NOT NULL • STATE IN (2, 5, STATE_FINISHED) • ((PRIORITY=1) OR (WI.REASON=2)) AND (STATE=2)

The following binary operators are available:

Table 9: Query table condition language: binary operators

<binary_op> ::=	=	<
	>	<>
	<=	>=
	LIKE	NOT LIKE

Rules:

- **The left side operand of a binary operator must reference an attribute of a query table. Valid attributes depend on the location of the filter or selection criterion. See Table 13 for details.**
- **The right side operand of a binary operator must be a literal value, constant value, or parameter.**
- **The LIKE and NOT LIKE operators are only valid for attributes of attribute type STRING.**
- **The left side operand and the right side operand must be of compatible attribute types. See section 2.7.5 for details.**
- **User parameters are converted to the attribute type of the left side attribute.**

Examples:

- `STATE > 2`
- `NAME LIKE 'start%'`
- `STATE <> PARAM(theState)`

The following unary operators are available:

Table 10: Query table condition language: unary operators

<unary_op> ::=	IS NULL	IS NOT NULL

Rules:

- **The left side operand of a unary operator must reference an attribute of a query table. Valid attributes depend on the location of the filter or selection criterion. See Table 13 for details.**
- **All attributes can be checked for null values.**

Example:

- `DESCRIPTION IS NOT NULL`

The following list operators are available:

Table 11: Query table condition language: list operators

<list_op> ::=	IN	NOT IN

Rules:

- **The right side of a list operator must not be replaced by a user parameter.**
- **User parameters can be used within the list on the right side operand.**

Example:
• STATE IN (STATE_READY, STATE_RUNNING, PARAM(st), 1)

Lists are represented as follows:

Table 12: Query table condition language: lists

<list> ::= <value> [, <list>]
Rules: • **See Table 11.**
Examples: • (2, 5, 8) • (STATE_READY, STATE_CLAIMED)

For the <attribute> element, see section 2.5.2; for the <value> element, see section 2.5.3.

2.5.2. Attributes

Attributes in an expression refer to attributes of query tables. Depending on the location of the expression, different attributes are available. For the client developer, *query filters* passed into the query table API (see section 5.2.2) are the only location where expressions can be used. For developers of composite query tables, various other locations exist where expressions can be used. The following table describes the attributes that are available at the different locations:

Table 13: Query table condition language expressions - available attributes

Where	Expression	Available attributes
query table API	Query filter	• All attributes defined on the query table
composite query table	Query table filter	• All attributes defined on the query table • If instance based authorization is used: All attributes defined on the WORK_-ITEM query tables, prefixed with "WI." Examples: • STATE=STATE_READY, if the query table contains an attribute STATE and if a constant STATE_READY is defined on this attribute • STATE=STATE_READY AND WI.-REASON=REASON_POTENTIAL_OWNER, if the query table contains an attribute STATE and if the query table uses instance based authorization
	Primary query table filter	• All attributes defined on the primary query table Examples: • STATE=STATE_READY, if the query

		table contains an attribute STATE and if a constant STATE_READY is defined on this attribute
	Authorization filter	• All attributes defined on the predefined query table WORK_ITEM, prefixed with "WI." **Example:** • WI.REASON=REASON_POTENTIAL_OWNER
	Selection criterion	• All attributes defined on the related attached query table **Example:** • LOCALE='en_US', if the attached query table contains an attribute LOCALE, such as TASK_DESC

Figure 5 shows the various locations of query table condition language expressions, and includes examples:

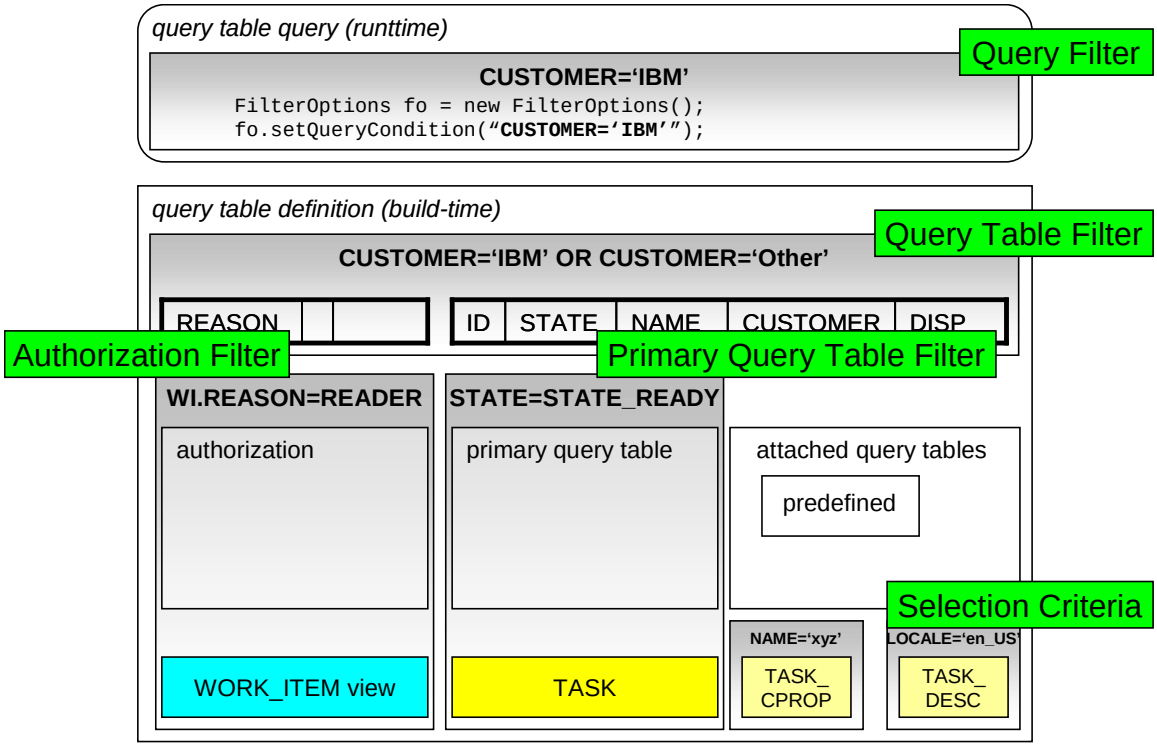


Figure 5: Query table condition language - expressions at various locations

2.5.3. Values

Within query table condition language expressions, values are one of the following options:

- **Constants: A constant value, which is defined on the attribute of a predefined query table. For example, STATE_READY is defined on attribute STATE of the query table TASK.**

- **Literals: Any hard coded value.**
- **Parameters: A parameter is replaced when the query is run with a specific value.**

Constants are available for some attributes of predefined query tables. For information on constants that are available on attributes of predefined query tables, see [predefinedViews]. Only constants that define integer values are exposed with query tables. Also, instead of constants, related literal values, or parameters can be used. Examples are:

- **STATE_READY on the attribute STATE of query table TASK can be used in a filter to check whether the task is in the ready state.**
- **REASON_POTENTIAL_OWNER on the attribute REASON of query table WORK_ITEM can be used in a filter in order to check whether the user who runs the query against a query table is a potential owner.**
- **Query filter "STATE=STATE_READY" is equal to "STATE=2", if the query is run on the query table TASK.**

Literals can also be used in expressions. A special syntax must be used for timestamps and for IDs. See section 2.7 for details. Examples are:

- **STATE=1**
- **NAME='theName'**
- **CREATED > TS ('2008-11-26 T12:00:00').**
- **TKTID=ID('_TKT:801a011e.9d57c52.ab886df6.1fcc0000')**

Parameters in expressions allow for a certain dynamicity of composite query tables. There are two types of parameters, user parameters and system parameters:

- **User parameters are specified using PARAM (*name*). This parameter must be provided when the query is run. It is passed as an instance of class com.ibm.bpe.api.Parameter into the query table API.**
- **System parameters are parameters that are provided by the query table runtime, without being specified when the query is run. There are two system parameters available, $USER and $LOCALE. $USER, which is a string, contains the value of the user who runs the query. $LOCALE, which is a string, contains the value of the locale that is used when the query is run. If this is not specified using the query table API, the server's locale is used. An example for the value of $LOCALE is 'en_US'.**

It might be useful to specify a parameter in the selection criteria of an attached query table which selects on a specific locale, such as, if in a composite query table the primary query table is TASK and an attached query table is TASK_DESC. Examples are:

- **STATE=PARAM(theState)**
- **LOCALE=$LOCALE**
- **OWNER=$USER**

2.6. Authorization

Two different types of authorization concepts are used when queries are run against query tables, instance based authorization and role based authorization.

Instance based authorization provided by Business Process Choreographer is based on work items. Each work item describes who (an individual, a group, or everybody) has which rights (reason) on what (object ID). This information is accessible using the WORK_ITEM query table if instance based authorization is used. The instance based authorization flag on query tables is described in detail in section 2.6.2; details on work items are described in section 2.6.1.

Role based authorization is based on J2EE roles. If role based authorization is used, callers of the query table API must be in the J2EE role BPESystemAdministrator. Role based authorization is currently only available for predefined query tables with template data or for composite query tables with a primary query table that contains template data.

2.6.1. Work items

The table describes the different types of work items that are considered if instance based authorization is used:

Table 14: Work item types

Work item type	Description
everybody	Everybody work items allow all users to access a particular object, such as a task or a process instance. In this case, the EVERYBODY attribute of the related work item is set to TRUE.
individual	Individual work items are work items that are created for particular users. The OWNER_ID attribute of the related work item is set to a particular user in this case. Multiple work items which differ in the attribute OWNER_ID can exist for one particular object (such as a task).
group	Group work items are work items that are created for users of a particular group. In this case, the GROUP_NAME attribute of the related work item is set to a particular group.
inherited	Inherited work items reflect the fact that readers and administrators of process instances are also allowed to access the human tasks which belong to these process instances, including escalations, and so on. Checks for an inherited work item in task queries is performed with complex SQL joins at runtime, which impacts on performance.

Work items are created by the Business Process Choreographer in different situations. For example, at task creation, work items are created

for the different roles, such as reader, potential owner, and so on, if related people assignment criteria were specified. For task lists or process lists, work items that are defined using staff assignment criteria are usually important. The following table describes the types of work items that are created, depending on the people assignment criterion that has been used. Inherited work items do not appear in the Table 15 because inherited work items reflect a relationship that is not explicitly modeled during process application development.

Table 15: Work items and staff verbs

Work item type	Related people assignment criterion
everybody	Everybody
individual	*all people assignment criteria except verbs Nobody[1], Everybody, and Group*
group	Group

2.6.2. Instance based authorization flag

The instance based authorization flag on query tables indicates whether or not objects are authorized using instance based authorization before returning objects in a query table. This is done by verifying if a suitable work item exists. Thus the instance based authorization flag, together with the query table API, influence which objects that are returned when a query is run on a query table. For more information of the query table API, see section 5.

For predefined query tables with instance data, instance based authorization is always used. Predefined query tables with template data use role based authorization, which requires the caller to be in the J2EE role BPESystemAdministrator. Objects in those query tables do not have related work items.

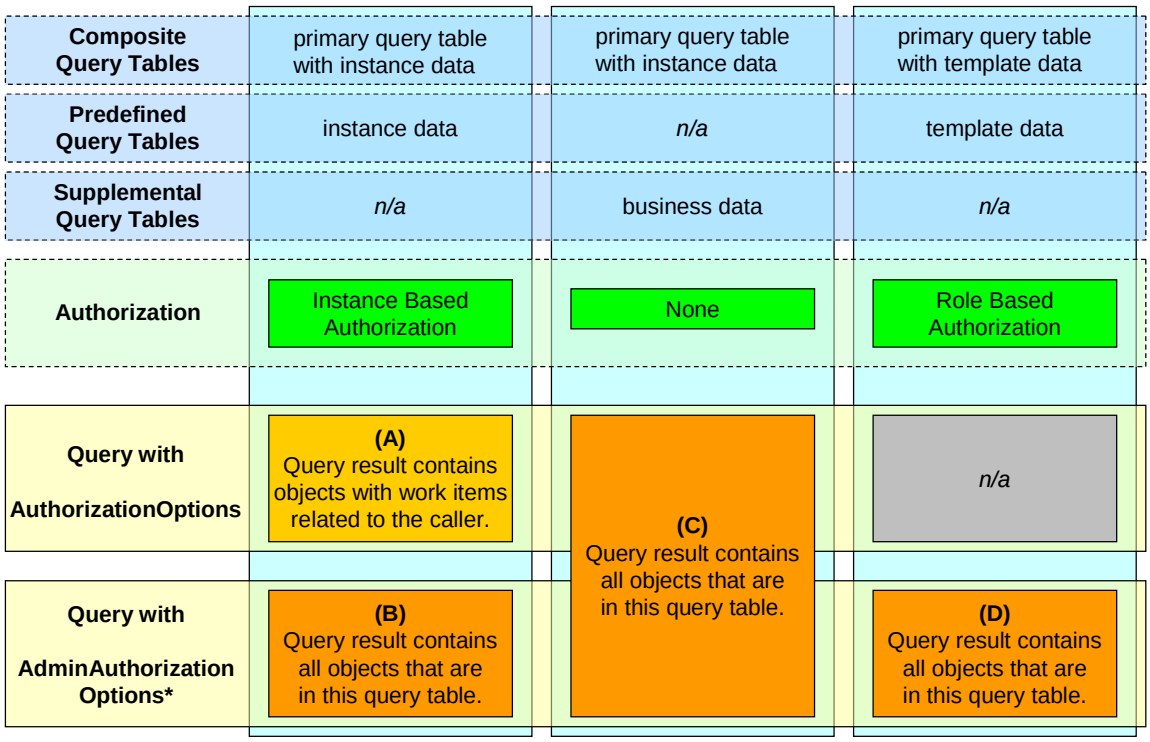
The instance based authorization flag on composite query tables can be set to false if a predefined query table with instance data is the primary query table. In such cases the security constraints implied by instance based authorization artifacts are overridden. That is, every authenticated user can use the respective query table to retrieve data, independent of whether or not they are authorized for the respective objects. Composite query tables with a primary query table that contains template data must not be set to use instance based authorization.

Also, supplemental query tables must not be set to use instance based authorization because supplemental query tables are not managed by Business Process Choreographer and therefore Business Process Choreographer has no authorization information for these table contents.

Figure 6 provides an overview of the available options for the instance based authorization flag, depending on the kind of query table. Also, it

[1] **The Nobody people assignment criterion results in a work item that is visible only by administrators.**

outlines the different behaviors together with the query table API and its authorization options:



*) If the onBehalfUser is set, (A) applies

Figure 6: Instance based authorization flag on query tables

Instance based authorization for objects in the query result using work items depend on the authorization parameter that is passed to the query table API and on the setting of the instance based authorization flag of the query table. For details on the authorization options of the query table API, see section 5.2.3.

(A) Queries on predefined or composite query tables using the AuthorizationOptions object return entities that correlate with a related work item for this particular user. This is also the case if the AdminAuthorizationOptions object is used and the onBehalfUser is set. Standard clients which present task or process lists to users would usually use this combination of query tables and query table API parameters.

(B) The full content of a query table consists of the entities that have a related work item, as configured with the instance based authorization of the query table. Instance based authorization considers four types of work items: everybody, individual, group, and inherited. The caller must be in the J2EE role BPESystemAdministrator. This combination of query tables and query table API parameters is intended for use in administrative scenarios where the full list of available tasks or processes must be shown or searched.

(C) Queries on query tables not using instance based authorization return the same result if AdminAuthorizationOptions or AuthorizationOptions is passed into the query table API. This is available for supplemental query tables and composite query tables with a primary query table with instance data. There is no check on work items, therefore all authenticated users see the full content. Clients that do not want to restrict object visibility by applying the instance based authorization constraints provided by Business Process Choreographer can use this combination of query tables and query table API parameters for task and process list queries. For other operations, such as claim and complete, users must have a related work items.

(D) Template data, as contained in predefined query tables with template data or related composite query tables, can be accessed only with role based authorization. This requires the caller to be in the J2EE role BPESystemAdministrator. The query table API can be used to access template information instead of the standard query API.

2.6.3. Authorization filter

On composite query tables, an authorization filter can be specified if instance based authorization is used. This filter restricts the work items which are used for authorization, based on certain attributes of work items. For example, the authorization filter "WI.REASON=REASON_ POTENTIAL_OWNER" on a composite query table with the primary query table TASK restricts the tasks that are returned when a person runs the query. The result only contains tasks that represent a to-do for that person, that is, the result is restricted to those tasks the person is authorized to claim. This filter can also be specified as the query table filter or as the query filter. Nevertheless, for query performance reasons, it is a best practice to specify those filters.

2.7. Attribute types

A subset of types that are available in the Java programming language and databases is used to define the type of an attribute of a query table. Attribute types are an abstraction of the concrete Java type or database type. Attribute types are needed when query tables are defined, when literal values are used in queries, and when values of a query result are accessed. For supplemental query tables, it is important that a valid database type to attribute type mapping is used. This section lists the various attribute types and their specific rules and mappings.

The following table describes the existing attribute types:

Table 16: Attribute type to database type mapping

Attribute type	Description
ID	The ID which is used to identify a human task (TKIID), a process instance (PIID), or other objects. For example, IDs are used to claim or complete a particular human task, which is identified with the specified TKIID.
STRING	Task descriptions or query properties can be represented as a string.
NUMBER	Numbers are used for attributes, such as the priority on a task.
TIMESTAMP	Timestamps describe a point in time, such as the time when a human task is created, or a process instance is finished.
DECIMAL	Decimals can be used as the type for query properties, for example when defining a query property with a variable of XSD type double.
BOOLEAN	Booleans can have one of two values, true or false. For example, human tasks provide an attribute, autoClaim, which identifies whether the task is claimed automatically if only a single user exists as the potential owner for this task.

2.7.1. Database type to attribute type mapping

The following table lists the attribute types and their mapping to database types:

Table 17: Attribute type to database type mapping

Database type	Attribute type
A binary type with 16 bytes. This is the type used for IDs such as TKIID on TASK of the Business Process Choreographer tables.	ID
A character based type. The length depends on the column in the database table that is referenced by the attribute of the query table.	STRING
An integer database type, such as integer, short, or long.	NUMBER
A timestamp database type.	TIMESTAMP
A decimal type such as float or double.	DECIMAL
A type that is convertible to a boolean value, such as a number. 1 is interpreted as *true* and all other numbers as *false*.	BOOLEAN

Example:

Consider a table in a DB2 environment, CUSTOM.ADDITIONAL_INFO, which should be represented in Business Process Choreographer as a supplemental query table. The related SQL statement for the creation of the database table would be[2]:

Table 18: Example of create table statement for DB2

```
CREATE TABLE CUSTOM.ADDITIONAL_INFO
(
  PIID          CHAR(16)     FOR BIT DATA,
  INFO          VARCHAR(220),
  COUNT         INTEGER
);
```

The following mapping of database column types to query table attribute types would be used for a supplemental query table for table CUSTOM.ADDITIONAL_INFO:

Table 19: Database types to attribute types mapping example

Database column and type	Query table attribute and type
PIID CHAR(16) FOR BIT DATA	**PIID (ID)**
INFO VARCHAR(220)	**INFO (STRING)**
COUNT INTEGER	**COUNT (NUMBER)**

Supplemental query tables typically refer to existing database tables and views, such that table or view creation is not necessary.

2.7.2. Attribute type to literal representation mapping

The following table lists the attribute types and their mapping to literal values. These can be used in expressions of the query table condition language, in filters of composite query tables, in selection criteria, and in filters that are passed to the query table API. Placeholders are marked *italic*. Note that the attribute types ID and TIMESTAMP use a special syntax, defined in Table 20, which is also used by the standard query API.

Table 20: Attribute type to literal values mapping

Attribute type	Syntax and usage as literal value in expressions
ID	**ID ('*string representation of an ID*')**
	When developing client applications, IDs are represented either as a string or as an instance of the interface com.ibm.bpe.api.OID. The string representation can be obtained from an instance of the interface com.ibm.bpe.api.OID using the toString() method, and

[2] The create table statement is provided for a better understanding of the example only. Supplemental query tables typically refer to existing database tables or views.

	must be enclosed in quotes.
STRING	*'the string'*
	The string must be enclosed in quotes.
NUMBER	*number*
	The number as text, and no quotation marks. Constants are defined for some number attributes on predefined query tables.
TIMESTAMP	TS ('*YYYY-MM-DD*T*hh*:*mm*:*ss*')
	The timestamp must be specified as defined above, where: • *YYYY* is the 4-digit year • *MM* is the 2-digit month of the year • *DD* is the 2-digit day of the month • *hh* is the 2-digit hour of the day (24-hour) • *mm* is the 2-digit minutes of the hour • *ss* is the 2-digit seconds of the minute The timestamp is interpreted as defined in the user's time zone.
DECIMAL	*number.fraction*
	The decimal number as text and no quotation marks; the *.fraction* part is optional.
BOOLEAN	true (ignore case), false (ignore case)
	The boolean value as text.

Examples:

- filterOptions.setQueryCondition("STATE=2");
- filterOptions.setQueryCondition("STATE=STATE_READY");
- a selection criterion on an attached query table TASK_DESC: "LOCALE='en_US'"
- filterOptions.setQueryCondition(
  "PTID=ID('_PT:8001011e.1dee8e51.247d6df6.29a60000')");

2.7.3. Attribute type to user parameter mapping

The following table lists the attribute types and their mapping to parameter values that can be used in expressions of the query table condition language, in filters of composite query tables, in selection criterions, and in filters passed to the query table API. Note that placeholders are marked *italic*.

Table 21: Attribute type to user parameter values mapping

Attribute type	Usage as parameter value in expressions
ID	PARAM (*name*) When developing client applications, IDs are represented either as a string or as an instance of the interface com.ibm.bpe.api.OID.

	As a parameter, both representations are valid. An array of bytes reflecting a valid OID can also be used (byte[16]).
STRING	**PARAM(*name*)** **The string representation of the object passed to the query table API at runtime. Note that the "toString()" method is used.**
NUMBER	**PARAM(*name*)** **A java.lang.Long, java.lang.Integer, java.lang.Short, or a java.lang.String representation (as defined in Table 20) of the number must be passed to the query table API.**
TIMESTAMP	**PARAM(*name*)** **The following representations are valid:** • **a java.lang.String representation of the timestamp as defined in Table 20** • **instances of com.ibm.bpe.api.UTCDate** • **instances of java.util.Calendar**
DECIMAL	**PARAM(*name*)** **A java.lang.Long, java.lang.Integer, java.lang.Short, java.lang.Double, java.lang.Float, or a java.lang.String representation (as defined in Table 20) of the decimal must be passed to the query table API.**
BOOLEAN	**PARAM(*name*)** **Valid values are:** • **a java.lang.String representation (as defined in Table 20) of the boolean** • **a java.lang.Short, java.lang.Integer, java.lang.Long with appropriate values 0 (meaning false) or 1 (meaning true)** • **a java.lang.Boolean object**

Example:

Table 22: Query example with parameters

```
…
// this example shows a query against a composite query table
// COMP.TASKS with a parameter "customer"

java.util.List params = new java.util.ArrayList();
list.add(new com.ibm.bpe.api.Parameter("customer", "IBM");
bfm.queryEntities("COMP.TASKS", null, null, params);
…
```

2.7.4. Attribute type to Java object type mapping

The following table lists the attribute types and their mapping to Java object types in query result sets.

Table 23: Attribute type to Java object type mapping

Attribute type	Related Java object type
ID	com.ibm.bpe.api.OID
STRING	java.lang.String
NUMBER	java.lang.Long
TIMESTAMP	java.util.Calendar
DECIMAL	java.lang.Double
BOOLEAN	java.lang.Boolean

Example:

Table 24: Query example with attribute type conversion

```
…
// the follwing example shows a query against a composite query table
// COMP.TA; attribute "STATE" is of attribute type NUMBER
…
// run the query
EntityResultSet rs = bfm.queryEntities("COMP.TA",null,null,params);

// get the entities and iterate over it
List entities = rs.getEntities();
for (int i = 0 ; i < entities.size(); i++) {

      // work on a particular entity
      Entity en = (Entity) entities.get(i);

      // note that the following code could be written
      // more generalized using the attribute info objects
      // contained in ei.getAttributeInfo()

      // get attribute STATE
      Long state = (Long) en.getAttributeValue("STATE");
      …
}
…
```

2.7.5. Attribute type compatibility

The following table lists the attribute types and their compatible attribute types which can be used in filters and selection criterions. Compatible attribute types are marked with 'X'.

Table 25: Attribute type compatibility

Attribute type	ID	STRING	NUMBER	TIMESTAMP	DECIMAL	BOOLEAN
ID	X					
STRING		X				
NUMBER			X		X	
TIMESTAMP				X		
DECIMAL			X		X	
BOOLEAN						X

3. Query Table Builder

The Query Table Builder is provided as an Eclipse plug-in and it supports the visual development of supplemental and composite query tables. Use the Query Table Builder to:

- **Develop composite and supplemental query tables**
- **Import and export Query Table definitions in XML format**

On WebSphere Process Server installations that are local to the Query Table Builder, the following functionality is provided:

- **Test query tables**
- **Deploy, update, and undeploy query tables**

The following figure shows the Query Table Builder tool:

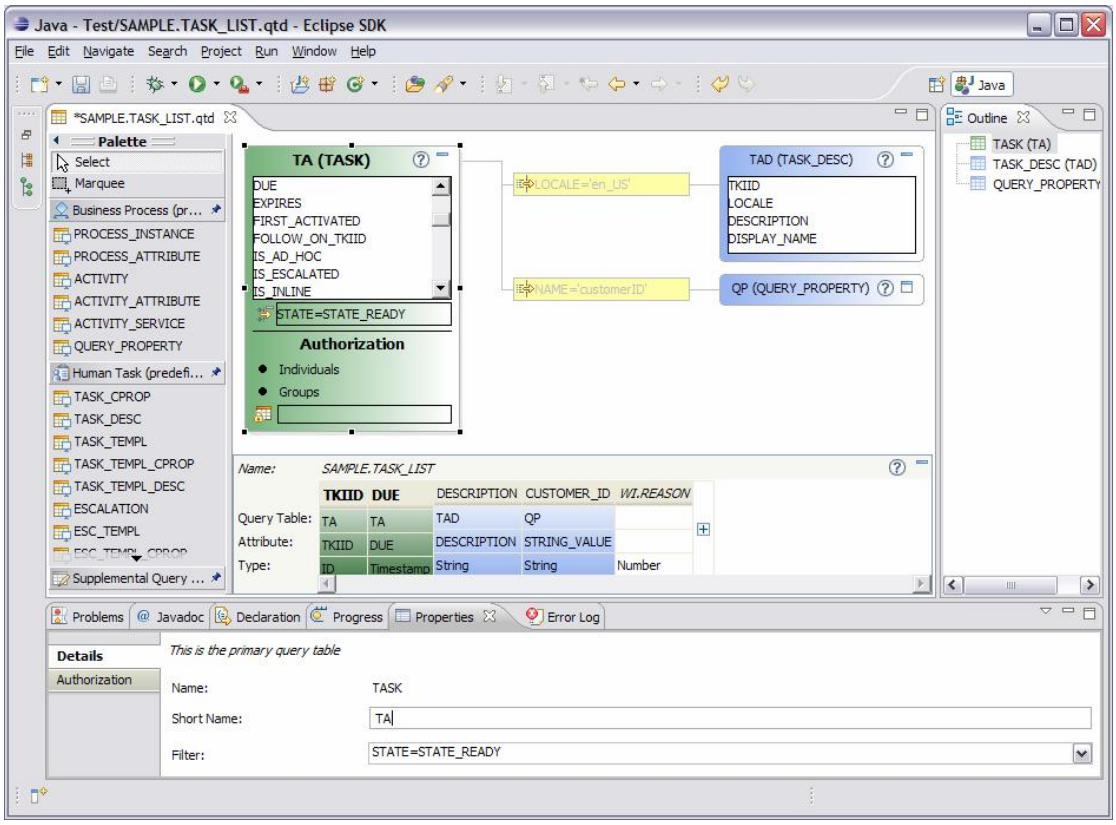


Figure 7: The Query Table Builder tool

See [QueryTableBuilder] for more details.

4. Administration

Query tables are administered using the wsadmin script, manageQueryTable.py. Unlike predefined query tables which are available out-of-the-box, composite and supplemental query tables must be deployed on WebSphere Process Server before being used with the query table API.

When query tables are deployed, the query table definition is stored in the Business Process Choreographer database. Additional database artifacts are not created in WebSphere Process Server, version 6.2. Any changes to composite and supplemental query tables, including deployment, update, and undeployment, are visible to the query table API without restarting the server.

For supplemental query tables, the user, or administrator, is responsible for providing the related database table or view.

For composite query tables, the information is composed of the existing database tables or views that relate to the predefined or supplemental query tables. Data is not duplicated in WebSphere Process Server, version 6.2.

Supplemental query tables which are referenced by deployed composite query tables must not be updated or undeployed.

For more details on query table administration, see [QTAdministration].

5. Queries

Queries are run on query tables using the query table API. With Business Process Choreographer, version 6.2, the query table API is available on the Business Flow Manager EJB only. The following methods are provided:

Table 26: Methods for queries run on query tables

Purpose	Methods
Query contents	• queryEntities (...) • queryRows (...) Both methods return contents of the query table. The method queryEntities returns content based on entities and queryRows returns content based on rows.
Query the number of objects	• queryEntityCount (...) • queryRowCount (...) Both methods return the number of objects in the query table, while the actual number can depend on whether the entity based or the row based approach is taken.

The two types of API methods, entity based methods and row based methods, can be used to retrieve content from query tables. Also, all four methods of the query table API take the same parameters as input.

5.1. Entity based and row based query table API

Composite query tables, which are suggested to be used for task and process list queries, are composed of one primary query table and zero or more attached query tables. A specific instance that is contained in any predefined query table only exists once within a Business Process Choreographer environment. Examples of instances are human tasks and business processes. Those instances are uniquely identified using an ID or a set of IDs[3], which are the TKIID for instances of human tasks and the PIID for process instances. Objects contained in composite query tables are uniquely identified by the unique ID of the objects contained in the primary query table. The attached query tables may not lead to duplicates – this is enforced by the one-to-one or one-to-zero relationship between the primary and attached query tables.

A client application programmer for user interfaces is typically interested in unique instances without duplicates, for example, a human task is to be displayed once only on the user interface. Unique instances are returned

[3] Business processes, human tasks, escalations, and activities are uniquely identified through one ID, respectively PIIDs, TKIIDs, EIIDs, and AIIDs. Additional information is often identified with a set of IDs or attributes. For example, a task description is identified using TKIID and LOCALE.

by the entity based query table API. Also, in object oriented programming, dealing with entities, as returned by the entity based query table API, can be more natural than dealing with rows, as returned by the row based query table API.

Row based queries may return duplicate rows of the primary query table, if instance based authorization is used:

- Information from the WORK_ITEM query table is retrieved with the query. For example, if WI.REASON is retrieved in addition to the attributes that are defined on the query table, multiple rows qualify because there can be multiple reasons why a user can access the entity, for example, a task or a process instance.
- Instance based authorization is used, and distinct has not been specified. Even though work item information is not retrieved, multiple rows may be returned if instance based authorization is used.

If the entity based query table API is used, the two cases above are taken into account. The consequences are as follows:

- Entity based queries are always executed with the SQL distinct operator.
- Entity based queries return a result which allows array values for work item related information.

The two different types of query result sets are described in section 5.3.

5.2. Query table API parameters

The following input parameters are passed into the methods of the query table API:

Table 27: Parameters of the query table API

Parameter	May be null (optional)	Type and brief description
query table name	no	java.lang.String
		The unique name of the query table.
filter options	yes	com.ibm.bpe.api.FilterOptions
		Options which can be used to configure the query. For example, a query threshold is set on this parameter.
authorization options	yes	com.ibm.bpe.api.AuthorizationOptions or com.ibm.bpe.api.AdminAuthorizationOptions
		Authorization can be further constrained, and administrator queries are configured with this parameter.
parameters	yes	a java.util.List of com.ibm.bpe.api.Parameter
		This parameter is used to pass user parameters which have been specified in a filter or selection criterion on a composite query table.

If composite query tables are used, it is expected that most of the optional parameters do not need to be specified by the client application developer. This is because composite query tables can be customized when the build is done, similar to the customization options that are available using the query table API when the query is run.

5.2.1. Query table name

The query table name is the name of the query table on which the query is run.

- For predefined query tables, this is the name of the predefined query table.
- For composite and supplemental query tables, this is the name of the respective query table that is specified while modeling the query table. The name of a composite or supplemental query table follows the *prefix.name* naming convention, and *prefix* may not be 'IBM'.

Both the query table name and prefix must be in uppercase letters.

5.2.2. Filter options

An instance of the class com.ibm.bpe.api.FilterOptions can be passed to the query table API. The filter options allow a configuration of the query using:

- a threshold and offset (skipCount)
- sort attributes (similar to the order by clause in an SQL query)
- an additional query filter
- the set of attributes returned, including work item information
- other

While the result set that can be obtained from a query table is specified by the definition of the respective query table, there are situations where specifying additional options when the query is run are required or advantageous, such as if system parameters are used. The following table describes the options that can be specified as filter options using the com.ibm.bpe.api.FilterOptions object:

Table 28: Query table API parameters: Filter options

Option	Type	Description
selected attributes	java.lang. String	• A comma separated list of attributes of the query table that must be returned in the result set. • If instance based authorization is used, work item information can be retrieved by specifying attributes of the WORK_ITEM query table, prefixed with 'WI.'. • If null is specified, all attributes of the query table are returned, without work item information.
query filter	java.lang. String	The query filter which is defined using the query table condition language. For details see section 2.5, section 0, and section 2.5.2.

sort attributes	**java.lang. String**	**A comma separated list of attributes of the query table, optionally followed by ASC or DESC, similar to the SQL order by clause: <sortAttributes> ::= *attribute* [ASC|DESC] [, <sortAttributes>][4]. If ASC or DESC is not specified, ASC is assumed. ASC means ascending, DESC means descending. Sorting occurs in the sequence of the sort attributes. This example sorts tasks in query table TASK in ascending order by state, and within the groups of the same STATE by NAME in ascending order: "STATE DESC, NAME ASC".**
threshold	**java.lang. Integer**	**Defines the maximum:** • **number of rows returned if queryRows is used.** • **number of entities returned if queryEntities is used. The actual number of available entities in the respective query table may exceed *threshold* even if the entity result set does not contain a number of *threshold* entities. This is due to technical reasons if work item information is selected.** • **count returned if queryRows or queryEntities is used.** **The default is null which means that no *threshold* is set.**
skip count	**java.lang. Integer**	**Defines the number of rows (row based queries) or the number of entities (entity based queries) that are skipped. As with the threshold parameter, *skipCount* may not be accurate for entity based queries.** **Skip count is used to allow paging over a large result set. The default is null which means that no *skipCount* is set.**
time zone	**java.util. TimeZone**	**The time zone that is used when converting timestamps. If not specified (null), the time zone on the server is used.**
locale	**java.util. Locale**	**The locale which can be specified when the query is run in order to override the system parameter $LOCALE.** **The default is null, which means that the locale on the server is used.**
distinct rows	**java.lang. Boolean**	**Used for row based queries only. If set to true, row based queries return distinct rows. This does not imply that unique "entities" are returned due to the possible multiplicity of work item information.**

[4] This is referred to as the query table sort language in some instances.

5.2.3. Authorization options

An instance of the class com.ibm.bpe.api.AuthorizationOptions can be passed to the query table API if the query is run on a predefined query table that contains instance data. It can also be passed if the query is run on a composite query table with a primary query table that contains instance data and instance based authorization is configured to be used. If the query is run on a predefined query table with template data or a composite query table with a primary query table that contains template data, an EngineNotAuthorizedException is thrown. In all other cases, the authorization options passed to the query table API are ignored.

Instances of the class com.ibm.bpe.api.AuthorizationOptions allow the specification of the type of work items used to identify eligible instances that are returned by the query. Refer to section 2.6 for details on types of work items.

Composite query tables can restrict the types of work items that are taken into account when identifying objects (or entities) that are contained in it. For example, if the authorization options that are passed to the query table API are configured to use *everybody* work items, this is only taken into account if everybody work items are defined for use on the definition of the composite query table. As a simple rule, a work item type that is not specified to be considered on the query table definition cannot be overwritten to be considered by the query table API, but a work item type that is specified to be considered on the query table definition can be overwritten not to be used. Also, the property "instance based authorization" on a composite or predefined query table cannot be overwritten by the query table API.

Depending on the kind of query table being queried, different authorization option defaults apply if the authorization object is not specified or if the related attributes (everybody, individuals, groups, or inherited) are set to null, which is the default.

Table 29: Query table API parameters: Authorization option defaults for instance based authorization

Work item types / Query table kinds	everybody	individual	group	inherited
predefined with instance data	TRUE	TRUE	TRUE	FALSE
predefined with template data	*n/a*	*n/a*	*n/a*	*n/a*
composite with a primary query table with instance data	TRUE	TRUE	TRUE	TRUE
composite with a primary query table with template data	*n/a*	*n/a*	*n/a*	*n/a*
supplemental	*n/a*	*n/a*	*n/a*	*n/a*

If *n/a* is specified in Table 29 it means that instance based authorization is not used and, therefore, any setting on the authorization object with respect to work items is ignored.

If TRUE is specified, the resulting query will only consider the specific work item type if the query table is defined to use this type of work item. This is true for all predefined query tables with instance data, but might not be true for a composite query table.

An instance of the class com.ibm.bpe.api.AdminAuthorizationOptions needs to be specified instead of an instance of the class com.ibm.bpe.api.AuthorizationOptions if:

1) **A query is run on a predefined query table with template data or on a composite query table that has a primary query table with template data.**
2) **A query is run on a query table with instance data or on a composite query table with a primary query table that contains instance data. It should return the content of that query table, regardless of restrictions due to authorization for a particular user. This behavior is equivalent to using the queryAll(...) method on the standard query API.**
3) **A query should be executed on behalf of another user.**

The following table describes how the various behaviors above are accomplished:

Table 30: Query table API parameters: AdminAuthorizationOptions

Situation	Description
onBehalfUser set to null	**1) If the query is run on a predefined query table with template data, or on a composite query table that has a primary query table with template data, all contents of that query table are returned.** **2) If the query is run on a predefined query table which uses instance based authorization, the particular objects contained in the query table are not checked for work items for a particular user. All objects that are contained in the query table are returned. Instance based authorization is used for all predefined query tables with instance data.**
onBehalfUser set to a particular user	**3) The query is run under the authority of the specified user, and the objects in the query table are checked against the work items for this user.**

For more details on authorization with query tables, see section 2.6.

5.2.4. Parameters

In query table definitions, parameters can be specified in filters on the primary query table, on the authorization, and on the query table, and in selection criteria on attached query tables.

The system parameters, $USER and $LOCALE, are replaced at runtime, and are not required to be passed into the query table API. The $LOCALE parameter can be overridden by setting the locale in the filter options. Refer to section 2.5.3 for details.

User parameters must be passed into the query table API when the query is run. This is accomplished by passing a list of instances of the class com.ibm.bpe.api.Parameter. The following properties must be specified on a parameter object:

Table 31: Query table API parameters: User parameters

Property	Description
Name	The name of the parameter as used in the query table definition. The name is case sensitive.
Value	The value of the parameter. See section 2.7 for valid object types.

5.3. Query results

The result of a queryEntityCount(...) or queryRowCount(...) query is a number. Results returned by the queryEntities(...) and the queryRows(...) method are more complex and described in sections 5.3.1 and 5.3.2.

5.3.1. EntityResultSet

An instance of the class com.ibm.bpe.api.EntityResultSet is returned by the method queryEntities(...). An entity result set has the following properties:

Table 32: Query table API entity result set: Entity result set properties

Property	Description
queryTableName	Name of the query table on which the query was run.
entityTypeName	• If the query was run on a composite query table, this is the name of the primary query table. • If the query was run on a predefined query table or on a supplemental query table, this is the name of the query table, that is, the same value as property *queryTableName*.
entityInfo	This property contains the meta information of the entities that are contained in the entity result set. A java.util.List list of com.ibm.bpe.api.AttributeInfo objects can be retrieved on this object. This list contains the attribute names and attribute types of the information contained in the entities of this result set.
entities	A java.util.List list of entity objects.

Instances of the class com.ibm.bpe.api.Entity contain the information that is retrieved from the query table query. An entity represents a uniquely identifiable object such as a task, a process instance, an activity, or an escalation. The following properties are available for entities:

Table 33: Query table API entity result set: Entity properties

Property	Description
entityInfo	**The EntityInfo object which is also contained in the entity result set (see Table 32).**
attributeValue (*attributeName*)	**The value of the specified attribute that has been retrieved for this entity. Type conversion is described in section 2.7.4. The type is contained in the related AttributeInfo object of this attribute.**
attributeValuesOfArray (*attributeName*)	**An array of values. Use this property if the attribute info property *array* is set to true which is the case only if the attribute refers to work item information.**

Example:

Table 34: Example of the entity based query table API

```
…
// the follwing example shows a query against
// predefined query table TASK, using the entity based API

…
// run the query
EntityResultSet rs = bfm.queryEntities("TASK", null, null, null);

// get the entitie's meta information
EntityInfo ei = rs.getEntityInfo();
List atts = ei.getAttributeInfo();

// get the entities and iterate over it
Iterator entitiesIter = rs.getEntities().iterator();
while (entitiesIter.hasNext()) {

      // work on a particular entity
      Entity en = (Entity) entitiesIter.next();

      for (int i = 0; i < atts.size(); i++) {
            AttributeInfo ai = (AttributeInfo) atts.get(i);
            Serializable value = en.getAttributeValue(ai.getName()) ;

            // process…
      }
}
…
```

5.3.2.RowResultSet

An instance of the class com.ibm.bpe.api.RowResultSet is returned by the method queryRows(…). This type of result set is similar to a JDBC result set. A row result set has the following properties:

Table 35: Query table API row result set: Row result set properties

Property / Method	Description
queryTableName	Name of the query table on which the query was run.
primaryQuery-TableName	• If the query was run on a composite query table, this is the name of the primary query table. • If the query was run on a predefined query table or on a supplemental query table, this is the name of the query table, that is, the same value as property *queryTableName*.
attributeInfo	This property contains a list of the com.ibm.bpe.api.AttributeInfo objects that describe the meta information for this result set. AttributeInfo objects contain the attribute names and attribute types of the information.
attributeValue	The value of the specified attribute that was retrieved for this row. Type conversion is described in section 2.7.4. The type is contained in the related AttributeInfo object of this attribute.
next, first, last, previous	The row result set is navigated using these methods. Compare its usage to iterators, enumerations, or JDBC result sets.

Example:

Table 36: Example of the row based query table API

```
…
// the follwing example shows a query against
// predefined query table TASK, using the entity based API
…
// run the query
RowResultSet rs = bfm.queryRows("TASK", null, null, null);

// get the entitie's meta information
List atts = rs.getAttributeInfo();

// get the entities and iterate over it
while (rs.next()) {
      // work on a particular row
      for (int i = 0; i < atts.size(); i++) {
            AttributeInfo ai = (AttributeInfo) atts.get(i);
            Serializable value = rs.getAttributeValue(ai.getName()) ;

            // process…
      }
}
…
```

6. Performance

Query tables introduce a clean programming model for developing client applications that retrieve lists of human tasks and business processes. Query tables have a positive effect on query performance. Sections 6.1 and 6.2 describe options for query tables, as well as query table API parameters that impact on query performance. Section 6.3 describes other factors that impact on performance.

6.1. Composite query table definition

The following table provides information about the query performance impact of options that are defined on composite query tables. It also provides information other topics related to composite query table definitions. The impact given in column *Performance Impact* is an average performance impact, actual impact observations may vary.

Table 37: Query performance impact of composite query table options

Object or topic	Performance impact	Description
query table filter	negative	Filters on query tables are the filters with the highest negative impact on query performance. These filters typically cannot use any defined indexes in the database.
primary query table filter	positive	A filter on the primary query table provides high performance filtering at a very early stage of the query result set calculation. It is suggested to restrict the contents of the query table using a primary query table filter.
authorization filter	positive	A filter on authorization can improve the performance of the query, such as how the primary query table filter improves it. If possible, an authorization filter should be applied. For example, everybody work items are often not used by applications and therefore can be excluded.
selection criteria	none	Some primary query table to attached query table relationships require the definition of a selection criterion in order to meet the one-to-one or one-to-zero relationship. A selection criterion typically has low performance impact because it is evaluated for a small numbers of rows only.

parameters	**none**	**Currently, using parameters in query tables has no negative performance impact. Nevertheless, parameters should be used only if needed.**
instance based authorization	**negative**	**If instance based authorization is used, each object in the query table must be checked against the existence of a work item. Work items are represented as entries in the WORK_ITEM query table. This check has a performance impact; most applications use instance based authorization, however.**
authorization: • **everybody** • **individuals** • **groups** • **inherited**	**negative**	**Each type of work item that is specified for use in the query table has a performance impact. Applications with high volume queries should only use individual and group work items, or only one of those. Inherited work items are usually not required, in particular when defining task lists that return human tasks representing to-dos. They should be used only when it is clear that they are needed, for example, to return lists of tasks that belong to a business process where a person might have read access based on the authorization for the enclosing business process.**
contained attributes	**none at this point in time**	**Currently, the number of attributes contained in a query table has no impact on performance. Nevertheless, only those attributes that are needed should be part of a query table.**

6.2. Query table API

The following table provides information about the query performance impact of options that are specified on the query table API. Note that the impact given in column *Performance Impact* is an average performance impact, actual impact observations may vary.

Table 38: Query performance impact of query table API options

Option	Performance impact	Description
selected attributes	negative (less is better)	The number of attributes that are selected when a query is run on a query table impacts on the number that need to be processed both by the database and by the Business Process Choreographer query table runtime. Also, for composite query tables, information from attached query tables need be retrieved only if those are either specified by the selected attributes or referenced by the query table filter or by the query filter.
query filter	negative	If specified, the query filter currently has the same performance impact as the query table filter (see Table 37). However, it is a good practice if filters are specified on query tables rather than passed into the query table API.
sort attributes	negative	The sorting of query result sets is an expensive operation, and database optimizations are restricted if sorting is used. If not needed, sorting should be avoided. Most applications require sorting, however.
threshold	positive	The specification of a threshold can greatly improve the performance of queries. It is a best practice to always specify a threshold.
skip count	negative	Skipping a particular number of objects in the query result set is expensive and should be done only if required, for example when paging over a query result.
time zone	none	The time zone setting has no performance impact.
locale	none	The locale setting has no performance impact.
distinct rows	negative	Using distinct in queries has some performance impact but might be necessary in order to retrieve non-duplicate rows. This option impacts only on row based queries and is ignored otherwise.
count queries	positive	If only the total number of entities or the number of rows for a particular query is needed, that is, the contents are not needed for all entries of the query table, the method queryEntityCount or queryRowCount should be used. The Business Process Choreographer runtime can apply optimizations that are valid only for count queries.

6.3. Other

In addition to the options and topics that are described in section 6.1 and 6.2, other factors that influence performance are described in Table 39:

Table 39: Query table performance: Miscellaneous topics

Topic	Description
Number of query tables on the system	The number of query tables which are deployed on a Business Process Choreographer container does not influence the performance of query table queries. Also, currently, it does not influence the navigation of business process instances, nor does it have impact on claim or complete operations on human tasks. Due to maintainability, keep the number of query tables at a reasonable level. Typically, one query table represents one task list or process list which is displayed on the user interface.
Database tuning	Although optimized SQL is used to access the contents of a query table, database tuning best practices need still to be implemented on a Business Process Choreographer database: • Database memory should be set to a maximum, taking into account other processes that are running on the database server, as well as hardware constraints. • Statistics on the database must be up-to-date, and should be updated on a regular basis. Typically, those procedures are already implemented in large topologies. For example, collect database statistics for the optimizer once per week in order to reflect changes of the data in the database. • Database systems provide tools to re-organize (or defragment) the data containers. The physical layout of the data in a database can also influence query performance and access paths of queries. • Optimal indexes are the key for good query performance. Business Process Choreographer comes with predefined indexes which are optimized for both process navigation and query performance of typical scenarios. In customized environments, additional indexes may be necessary in order to support high volume task or process list queries. Use tools provided by the database in order to support the queries which are run on a query table.

7. The query table API and the standard query API

This section compares the main differences between the standard query API, which is available with WebSphere Process Server, version 6 and the query table API which is available with WebSphere Process Server, version 6.2. Technical terms related to the standard query API are in *italic*. Technical terms used in the standard query API may also be used in the query table API, such as the term work items.

Table 40: The query table API and the standard query API

Difference	Description
EJB API methods and signatures	A number of EJB methods on the Business Flow Manager and/or Human Task Manager EJB interface are available in order to query data that is contained in the Business Process Choreographer database: • standard query API (incomplete list) o query(...) o queryAll(...) o queryProcessTemplates() o ... • query table API (complete list) o queryEntities(...) o queryEntityCount(...) o queryRows(...) o queryRowCount(...) Currently, query tables can be accessed only using the query table API which is available on the Business Flow Manager EJB interface. All other methods use the predefined database views which are not query tables. The query table counterparts are called predefined database views.
protocols	The standard query API is available on additional interfaces, such as the Web services interface and the REST interface of Business Process Choreographer. The query table API is currently only available on the EJB interface.
query table name	The query table API allows queries to be run on one specific query table, which is identified by the query table name. The standard query API provides similar functionality using the select clause.
select clause and selected attributes	The select clause in the standard query API allows the specification of the attributes or columns that the query should return. Those attributes are specified fully qualified using *view.name*, for example, TASK.STATE.

	The selected attributes that are specified in the filter options of the query table API also specify the attributes that the query should return. In contrast to the standard query API, the selected attributes of the query table API are not specified fully qualified because the query is run on one query table of which attributes are uniquely identifiable by the names defined for them.
where clause **and filters**	**The where clause which is passed to the standard query API defines a filter which is applied to the query. The same is true for the query filter (property queryCondition on the query table API) on the query table API.**
where clause **and selection criteria**	**In the standard query API, a concept of selection criteria does not exist. Nevertheless, it can be compared to the part of the where clause that defines, for example, the name or locale of QUERY_PROPERTY, or TASK_CPROP, or TASK_DESC that must be added to the result. For example, a where clause of "QUERY_PROPERTY.NAME='xyz'" would relate to a selection criterion of "NAME='xyz'".**
work items **and authorization**	**In order to provide a personalized view on the data in the Business Process Choreographer database, work items are used. Work items are accessible using the WORK_ITEM view or the WORK_ITEM query table, respectively.** **If the standard query API is used, all four types of work items are considered: everybody, individual, groups, and inherited work items, if applicable. The configuration of the authorization is achieved by a customized where clause, such as "WORK_ITEM.EVERYBODY=0", for the exclusion of everybody work items.** **If the query table API is used, you can customize the use of work items on the query table definition when the query table is developed and on the query table API, using the AuthorizationOptions or AdminAuthorizationOptions object.**
parameters	**The standard query API does not have a concept of parameters that can be set on the query API.** **A query table definition allows the use of parameters that must be set when the query is executed.** • **Composite query tables allow parameters in filters and selection criteria.** • **Predefined query tables do not contain parameters.** • **Supplemental query tables do not allow the specification of parameters.**
stored queries **and query**	**Within the standard query API, stored queries can be used to predefine a set of options that is passed to the query API under a particular name, which is the name**

tables	**of the stored query. Stored queries are then used to run a query that is based on those options. Stored queries allow the use of parameters, which are passed to the standard query API at runtime.** **Query tables are highly customizable and also can be defined to use parameters.** **The difference between a stored query and a query table is that stored queries are defined for one particular query, while a query table is defined for a particular set of queries. For example, the query table definition does not allow the specification of an order by clause because this information is typically available only when the query is run.**
materialized views	**Business Process Choreographer materialized views are available with WebSphere Process Server, version 6.0.2.1 for DB2 and version 6.0.2.2 for Oracle. Materialized views provide a query performance improvement using database technologies. Materialized views are currently only available for the standard query API.**
custom tables	**Custom tables have been introduced with WebSphere Process Server, version 6.0.2.4. Custom tables are used to include data external to the Business Process Choreographer database schema into queries using the standard query API.** **In WebSphere Process Server, version 6.2, the query table API offers the same functionality with supplemental query tables as the replacement for custom tables on the standard query API.**
***queryAll* and Admin Authorization Options**	**The standard query API offers a queryAll(...) method which can be used by users that are in the J2EE role BPESystemAdministrator. On the standard query API, if the queryAll(...) method is used instead of the query(...) method, the query result contains all objects without being restricted by work items for a particular user, group, or everybody.** **In the query table API, the queryAll functionality is provided by the AdminAuthorizationOptions object, which can be passed to the query table API instead of the AuthorizationOptions object.**

Related References

[BPCSamples]
Business Process Choreographer Samples
http://publib.boulder.ibm.com/bpcsamp/index.html

[InfoCenter]
WebSphere Process Server, version 6.2, information center
http://publib.boulder.ibm.com/infocenter/dmndhelp/v6r2mx/index.jsp

[predefinedViews]
WebSphere Process Server, version 6.2, information center – Database Views for Business Process Choreographer
http://publib.boulder.ibm.com/infocenter/dmndhelp/v6r2mx/topic/com.ibm.websphere.bpc.620.doc/doc/bpc/r6bpc_dbviews.html

[QTAdministration]
WebSphere Process Server, version 6.2, information center – Administering query tables
http://publib.boulder.ibm.com/infocenter/dmndhelp/v6r2mx/topic/com.ibm.websphere.bpc.620.doc/doc/bpc/t4querytables_admin.htm

[QueryTableBuilder]
The Query Table Builder tool is available as an Eclipse plugin and can be downloaded on the WebSphere® Business Process Management SupportPacs site. Look for PA71 WebSphere Process Server - Query Table Builder.
http://www-01.ibm.com/support/docview.wss?rs=2308&uid=swg27009734

[BPCProgModel]
WebSphere Process Server V6.1 Business Process Choreographer Programming Model
http://www-01.ibm.com/support/docview.wss?uid=swg27012602

Tables and Figures

Tables

Figures